IJCSES

Volume 3, No: 4

April 2020

ISSN 2513-8359

# International Journal of Computer Science Education In Schools

Editors

Prof. Filiz Kalelioglu

Dr. Yasemin Allsop

www.ijcses.org

# International Journal of Computer Science Education in Schools

**April 2020, Vol 3, No 4**

**DOI:** https://doi.org/10.21585/ijcses.v3i4

## Table of Contents

# Exploring and Comparing Computational Thinking Skills in Students Who Take GCSE Computer Science and Those Who Do Not

**Lauren Gillott**[1]

**Andrew Joyce-Gibbons**[2]

**Elizabeth Hidson**[3]

[1]Mulberry Academy Shoreditch

[2]Durham University

[3]University of Sunderland

DOI: 10.21585/ijcses.v3i4.77

**Abstract**

This study compares computational thinking skills evidenced by two groups of students in two different secondary schools: one group per school was studying a qualification in Computer Science. The aim was to establish which elements of computational thinking were more prevalent in students studying Computer Science to a higher level. This in turn would evidence those elements likely to be present from their earlier computing education or through their complementary studies in Science or Mathematics, which all students also studied. Understanding this difference was important to identify any increased competence in computational thinking that was present in the Computer Science groups. Interviews involved a set of questions and a maze activity designed to elicit the sixteen students' computational thinking skills based on the Brennan and Resnick (2012) model of computational concepts, practices and perspectives. Analysis of students' responses showed surprisingly little difference between the computational thinking practices of the two groups in relation to abstraction, decomposition, evaluation, generalisation/reusing, logical reasoning and debugging/testing. The study concludes that general computational thinking skills can be developed either at a lower level of study or in cognate curriculum areas, leaving computer science as the rightful locus of computational thinking for automation.

**Keywords:** computational thinking; computer science; computing, automation, GCSE

## 1. Introduction: Defining Computational Thinking

There has been recurring international attention on the perceived lack of adequate preparation of future generations to participate fully in the changes which new technology is bringing to society (Grover and Pea, 2013; Webb et al, 2017). The Royal Society's 'Shut down or restart' report (2012) identified key areas of concern around curriculum and provision in computing education in England, calling for Computer Science to be introduced into the curriculum to increase creativity, rigour and challenge and redress the falling numbers and attrition rates of those studying advanced computing courses post-16.

Although the concept was first promoted by Papert (1980), it was Wing's (2006) call for computational thinking as a 'universally applicable attitude and skill set for all' (p.33) that was repurposed to underpin the Royal Society's position regarding Computer Science as a discipline. This influenced the 2014 National Curriculum computing programmes of study (Department for Education, 2013) and was also evident in the Computer Science General Certificate of Secondary Education (GCSE) subject content first published in 2015 (Ofqual, 2018a, Department for Education, 2015).

The term 'computational thinking' (CT) has come to be embodied in computing education discourse as useful for society (Wing, 2014) and as a transferrable skill set valued not only academically, but also by employers (Brown, Sentence, Crick & Humphreys, 2014). The extent to which CT can be considered a discrete set of skills, separate from mathematical or scientific reasoning is still a matter of some debate (Tedre & Denning, 2016, Weintrop et al., 2016). Many of the skills currently defined as 'computational' thinking, may be more properly considered among the higher order thinking skills (HOTS) articulated in mathematics by Pólya but tracing its heritage back to Plato (diSessa, 2018). Selby and Woollard's (2013) developing definition promoted this view by separating evidence of the practice of skills from the activity of thinking, but Denning (2017) argues that skill manifests tacit knowledge. CT as an activity that is often product-oriented is therefore not the same as whether or not a student can evidence the use of relevant skills. It is arguable that the nature and extent of computation for automation purposes resulting from applied computational thinking marks a conceptual dividing line in the field. Berry (2019) highlighted that thinking in relation to automation (and therefore demonstrable) is distinct from the thinking skills developed by a broader set of CT skills, a tension also alluded to by Cansu and Cansu (2019) when contrasting the different prevailing definitions of computational thinking.

Given the ongoing shortage of trained Computer Science teachers, the teaching or reinforcing of CT in other curriculum areas could help to relieve the pressure on limited resources as well as provide a safety net for the development of 'computational thinking without a machine' (Wing, 2014). Wider Science, Technology, Engineering and Mathematics (STEM) subjects provide a cognate space for the development of CT as these fields have also seen a growth in their computational counterparts and seek to develop a nuanced understanding of CT as it applies to their practices (Weintrop et al., 2016).

By exploring the proficiency of able learners who are taking the new GCSE in Computer Science (CS) compared to those who are not, this study questions whether those taking GCSE CS have a notable difference in general CT skills beyond the thinking for automation that might be expected from students opting to study for a qualification in Computer Science. The answer to this question may help to inform curriculum discussions and the allocation of scarce time and personnel resources, but more importantly, it contributes to our understanding of the development of computing as a school subject by anchoring systematic research in the teaching and learning that underpins it.

### *1.1 Operationalising Computational Thinking*

Wing originally characterised CT as a type of thinking that 'involves solving problems, designing systems and understanding human behaviour, by drawing on the concepts fundamental to computer science' (Wing, 2006, p.33). Computing education is concerned with 'the habits of mind developed from designing programs, software packages, and computations performed by machine.' (Denning, 2017, p. 33). Whereas the trend in the computing education literature leans often towards describing CT in general terms or as part of a wider set of twenty-first century skills (Livingston et al, 2015), for practical purposes CT still requires operationalisation. Brennan and Resnick's (2012) development of three domains of CT: computational concepts, computational practices and computational perspectives has influenced resources developed for schools by the British Computing Society such as the Barefoot suite (barefootcomputing.org).

In the Brennan and Resnick (2012) framework, computational concepts are key concepts that programmers engage with as they develop a computer program, such as sequencing, loops, parallelism, events, operators and data. CS students must learn how to select the most appropriate one for their program design. As they attempt to put these concepts into practice to meet their design goal, they will engage in a number of computational practices. These are the processes that programmers use when developing new software. CS students who are secure in these practices understand the 'how' of programming. They understand the appropriate use of strategies such as decomposition, debugging, logical reasoning, algorithmic thinking or abstraction to achieve their objective. Finally, computational perspectives are developed by CS students who are able to reflect on how their programming has the potential to alter the relationship they have with the wider world. When grounded in an understanding of concepts and an ability to apply practices, the computational perspectives developed by a student programmer gives them the ability to: i) create rather consume media; ii) use digital tools in innovative ways and iii) question the role of technology in daily life based on an appreciation of the possibilities and limitations afforded by technology. This model has provided a stimulus for the current study: CS alone is not CT, but it can provide evidence of CT.

The increased importance placed on CS in schools increases the need for studies that focus on the school phase, but extensive literature reviews have concluded that CT research in the school context is still in the relatively early stages (Lye & Koh, 2014; Sentance & Selby, 2015; Lockwood & Mooney, 2018). This point was also acknowledged by the Royal Society (2017) with their conclusions that additional research is needed into; i) teaching and learning CT, ii) tools and methods of assessment and, iii) a better understanding of the relationship between CT and CS as a curriculum subject (Crick, 2017; Kallia, 2017; Waite, 2017).

### *1.2 Assessing Computational Thinking*

The assessment of thinking skills of any kind presents a considerable challenge (Moseley, 2005; Burden, 2015, Bilbao et al. 2017). Assessment in the case of a school subject is vital in terms of being able to monitor and measure student progress, so much attention has been paid to developing and sharing teaching, learning and assessment materials, which can therefore provide proxies for CT. At one end of the spectrum, in an attempt to be able to assess at scale Korkmaz, Cakir, and Özden (2017) developed a Likert-scale survey to assess CT through 29 questions in five categories: creativity, algorithmic thinking, cooperativity, critical thinking and problem solving, but completely divorced from the practical elements as no practical skill is tested.
One commonly used method to assess CT is project analysis, examining projects previously created by students (Brennan & Resnick, 2012; Werner, Denner & Campe, 2015; Burke, 2012). Only the project, not the process to create the project is examined. Therefore, while project analysis can give some insight into students' CT skills, it does not give enough information about the process used to create the projects. Recent studies focused on assessment of CT in schools have tended to use practical tasks to uncover and quantify students' programming skills, which are then linked to CT skills. Zhong, Wang, Chen and Li (2015) used

practical tasks as well as students' written reflective reports, which were then graded, or coded, on a scale of 1-5 to assess the level of CT being shown. The focus on testing students' ability to use programming constructs was a similar theme in Román-González et al. (2017), where multiple-choice questions gave students the opportunity to solve problems and demonstrate CT. This type of testing is very closely related to programming and therefore only possible to use with students of similar levels of programming experience. However, it limits the scope of the assessment as it is partially dependent on the students' ability to respond in a suitably technical way.

Design scenarios, in which students are monitored when working with or creating a program (Brennan & Resnick, 2012; Lee, et al., 2011; Fields, et al., 2012; Webb, 2010; Fessakis, Gouli & Mavroudi, 2013; Zhong, et al., 2015; Lye & Koh, 2014) are a favoured method for measuring CT. This method is able to assess all three dimensions of CT, enhanced by the fact that students explain the process in real-time, but it can be very time-consuming.

Denning (2017) would support these approaches as evidence of the 'new' CT that recognises the significance of practical programming as evidence of CT. It can also be argued that this further separates CT from being considered as just another thinking skills framework and pushes the practical application towards the demonstrable outcomes of the CT process.

Román-González et al. (2017) categorised a range of assessment tools into five helpful categories, suggesting that using complementary tools can strengthen the quality of the assessment. Thinking about assessment of CT in terms of summative (such as tests), formative-iterative (using artefacts to develop CT skills), skill-transfer (through applying knowledge to problems), perceptions-attitudes scales and vocabulary assessment tools allows for a more nuanced understanding of what is possible in terms of assessment. This is further supported by Allsop (2019), whose longitudinal study triangulated a wealth of data gathered through conversations, interviews, journals, worksheets and completed games. It is clear that, on one hand, the ability to code is not enough to evidence CT, and on the other, that there are elements of coding ability that demonstrate computational practices that cannot be evidenced through more abstract approaches.

The current study was designed to access the participants' responses as evidence of their cognitive processes as well as giving them the opportunity to demonstrate some real-time computational practices. In assessment terms, this combined the aforementioned formative-iterative and skill-transfer approaches. This was important in developing the research questions.

### *1.3 Research Questions*

Taking the separation of CT into the three areas identified by Brennan and Resnick (2012), this study explores the ways in which CS students and non-CS students differ in their ability to apply computational concepts, practices and perspectives to scenario-based and practical computing problems. It seeks to answer three research questions:

RQ1: How do CS students and non-CS students differ in their ability to apply computational concepts to scenario-based and practical computing problems?

RQ2: How do CS students and non-CS students differ in their ability to apply computational practices to scenario-based and practical computing problems?

RQ3: How do CS students and non-CS students differ in their ability to apply computational perspectives to scenario-based and practical computing problems?

## 2. The study

### *2.1. Method: Data Collection From Interviews*

The study was a comparative ex post facto design, which compared participants from two secondary schools in England. The constraints of the participants' school timetables and limited free time led to the selection of artefact-based interviews as the best available data collection method (Webb, 2010; Lee, et al., 2011; Fields, et al., 2012; Fessakis, Gouli & Mavroudi, 2013; Kallia, 2017). An initial set of general CS-related questions exploring issues and scenarios was posed to each participant before they worked with an 'artefact' – in this case a pre-made Scratch game. The questions were related to the specific CT practices in Table 1. Artefact-based interviews can give insight into the learner's processes and objectives (Zhong et al., 2015), allowing students to be observed and engaged with while working on a program. By collecting real-time data as the participant worked on a CS problem, researchers were able to make note of the steps used to work through the artefact as well as discuss the process with the participant, exploring their reasoning for the choices they made. The research process is presented in Figure 1, below.

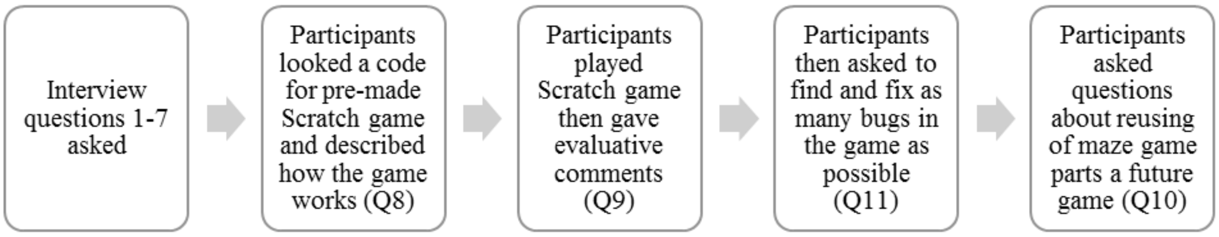

Interview questions 1-7 asked
Participants looked a code for pre-made Scratch game and described how the game works (Q8)
Participants played Scratch game then gave evaluative comments (Q9)
Participants then asked to find and fix as many bugs in the game as possible (Q11)
Participants asked questions about reusing of maze game parts a future game (Q10)

Figure 1. The Research Process

### *2.2. Participants*

The schools attended by the participants were of comparable size, locality and socio-economic circumstances. There were some differences in the approach to CS instruction in the schools. In terms of prior learning, in School A, pre-GCSE pupils learned to use Scratch, a popular block-based visual programming language (Noone & Mooney, 2018), but did not encounter Python, a text-based programming language, until they had started GCSE CS and begun to prepare for controlled assessment. In School B, in addition to visual programming, students were introduced to basic Python earlier, in Year 8 (age 12-13 years), prior to the beginning of GCSE CS. However, they continued to spend the early part of GCSE on the fundamentals of Python. Based on formative and summative assessments, both CS groups' programming skills were broadly comparable by the time of the interviews in Year 11.

Participants (n=16, aged 15-16) had self-selected to the extent that they had elected whether or not to study a GCSE qualification in CS up to age 16 following the completion of their period of mandatory computing education up to the age of 14. However, in both schools this option was only available to those who had demonstrated previous high attainment in Mathematics.

In each school there were 4 CS participants and 4 non-CS participants. Participants in CS and non-CS groups were predicted broadly similar grades in GCSE Mathematics. In both schools there was gender balance in the non-CS group (2 male, 2 female). However, there was imbalance in the CS groups (School A: 4 male; School B: 2 male, 2 female) because of the uptake of Computer Science.

Table 1. Interview Questions Related to Computational Concepts, Practices and Perspectives

| Question | Computational Thinking indicative references |
|---|---|
| 1. If asked to create a basic calculator that only did addition and subtraction, how would you go about doing this? | **Formulate Problems for a Computer** (CSTA & ISTE, 2011; Wing, 2006) |
| 2. Imagine you are a police detective and a murder has been committed in your area. You are given loads of information and are expected to find the murderer. How would you do this? | **Decomposition** (Riley & Hunt, 2014; Selby, Dorling & Woollard, 2014) |
| 3. I'm going to read a couple of statements, and you tell me if they are true, false, or if there is not enough information given: a. Joe is older than Tom and Matt is older than Joe. Is this statement true, false or not enough info: Tom is older than Matt. b. All the flowers in the garden are red. Some of the flowers in the same garden are roses. Is this statement true, false or not enough info: All roses are red. | **Logical Reasoning** (Riley & Hunt, 2014; CSTA & ISTE, 2011; Barr & Stephenson, 2011) |
| 4. Could you describe an algorithm to me that describes how you get to school in the morning, including if you cannot do your route one day. How would you represent this routine? | **Algorithmic thinking** (CSTA & ISTE, 2011; Barr & Stephenson, 2011;Werner, et al., 2012; Lee, et al., 2011; Selby, Dorling & Woollard, 2014) |
| 5. If you were to make a program to make a game of Chess, can you tell me specifically about some of the programming concepts that you would use and how you would use them? | **Computational Concepts** (CSTA & ISTE, 2011 |
| 6. How do you think the design of this game ties into your day-to-day life? | **Computational Perspectives** (Brennan & Resnick, 2012) |
| 7. Do you think technology like this has any impact on the world? | **Computational Perspectives** (Brennan & Resnick, 2012) |
| 8. After a quick glance through and before playing it, could you tell me what this program does? (Uses Scratch artefact). | **Abstraction** (CSTA & ISTE, 2011; Riley & Hunt, 2014; Lee, et al., 2011; Barr & Stephenson, 2011; Werner, et al., 2012; Selby, Dorling & Woollard, 2014) |
| 9. Could you tell me some positives and negatives of the design of this game? What would you change if you made it? (Uses Scratch artefact). | **Evaluation** (CSTA & ISTE, 2011; Selby, Dorling & Woollard, 2014) |
| 10. If asked to create a game on scratch where two users race each other through a course, how could you use some of the parts/ ideas of this Scratch program to do so? (Uses Scratch artefact). | **Generalisation/reusing** (Barr & Stephenson, 2011; Brennan & Resnick, 2012; Fields, et al., 2012; Webb, 2010) |
| 11. There are some bugs/errors in this program… can you find | **Debugging/testing** |

| | |
|---|---|
| them and fix them? (Uses Scratch artefact). | (Brennan & Resnick, 2012; CSTA & ISTE, 2011; Selby, Dorling & Woollard, 2014) |

### *2.3. The Data Collection Tools*

Table 1 presents the questions that were developed by the researcher to give the participants the opportunity to demonstrate a range of computational thinking practices. These questions were similar in nature to questions commonly used by Computer Science teachers to stimulate students' thinking in their lessons. Many such examples are available to teachers making use of shared resources such as the Computing At School website (Computing At School, 2020). In addition, questions were developed in response to assessable themes identified in the literature around CT.

Questions 8-11 used a Scratch maze game (Figure 1) designed by the researcher for the purposes of the study. This incorporated four key features of programming: sequence, selection, variables and events. Scratch was used to create the program because it was a common component in the pre-GCSE curriculum of both schools for Year 7 and Year 8 students (11-14 years old). All participants had previously worked with Scratch prior to the beginning their GCSEs.

Participants received one point for each star collected and three for completing the maze. If they touched the maze walls, they were sent back to the beginning of the progam. There were seven separate bugs in the Scratch script, which it was anticipated that the participants would have the ability to identify.

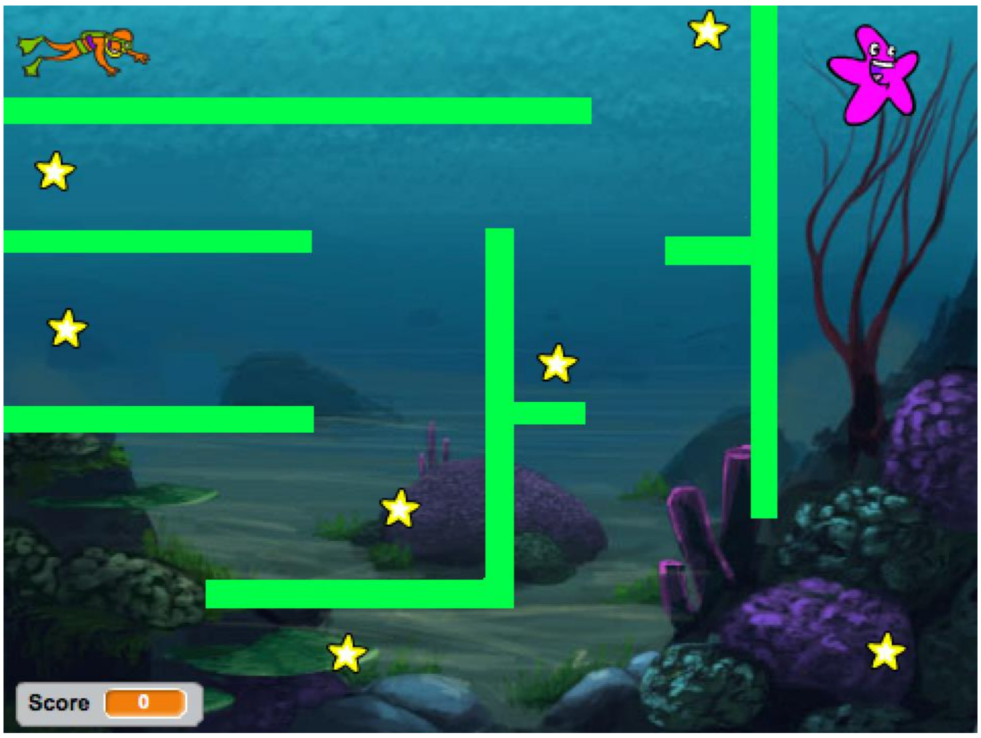


Figure 2. Screen Shot of the Scratch Maze Game Used In Questions 8-11

### *2.4. Data Analysis*

Data collection involved both audio recordings and field notes made by the researcher while conducting the artefact-based interviews. Field notes were used to highlight typical and atypical responses to the questions. Audio recordings were used to supplement these and as a basis for transcriptions of student utterances. For each question it was noted whether the participant's response provided evidence that they were able to engage with the CT *practices*, *concepts* or *perspectives*, as summarized in Table 1 below. A process of thematic analysis was used, colour-coding correct or incorrect answers, the use of appropriate key words and the amount of detail provided. Examples from the main themes are presented in the results section below.

## 3. Results

### *3.1 Summary of Responses*

The summary responses of all participants to all questions are presented in the tables below. Table 2 shows that overall, the mean number of correct responses given by CS students (79.7%) was higher than those given by non-CS students (62.5%). Table 3 shows the number of correct responses given by students to the first eight questions. The differences between CS and non-CS students are more pronounced in Q1 (formulation of problems) and Q2 (decomposition), with all CS students being able to give a correct answer to the question on formulation compared to only 37.5% of the non-CS students. In Q2, 87.5% of CS students gave a correct answer compared to 37.5% of non-CS students. CS students also gave more correct responses to Q3 (logic), Q5 (concepts) and Q8 (abstraction) than non-CS students. CS students were able to give responses as to how they would improve the Scratch artefact in Q9 (evaluation) in Table 4. However, both groups were able to supply reasonable ideas as to how to reuse some of the Scratch code in other contexts (Q10). Students from each group were also able to suggest some solutions to the bugs included in the Scratch program (Q11), with a mean average of three responses per student overall in each group.

Table 2. Mean Average of Correct Responses to Interview Questions

| Computer Science Students | Non-Computer Science Students |
|---|---|
| 79.7% | 62.5% |

Table 3. Summary of Correct Responses to Interview Questions Given by Participants

| Question Number and Focus | Computer Science Students | Non-Computer Science Students |
|---|---|---|
| 1. Formulation | 100% | 37.5% |
| 2. Decomposition | 87.5% | 37.5% |
| 3. Logic | 37.5% | 25.0% |
| 4. Algorithm | 87.5% | 100% |
| 5. Concepts | 87.5% | 50% |
| 6. Perspectives (specific) | 50% | 62.5% |
| 7. Perspectives (general) | 87.5% | 100% |
| 8. Abstraction | 100% | 87.5% |

Table 4. Types of Responses to Interview Questions on Evaluation and Generalizing or Reusing Code

| Question Number and Focus | Computer Science Students | Non-Computer Science Students |
|---|---|---|
| 9 i. Evaluation (Positive) | 75% | 100% |
| 9 ii. Evaluation (Negative) | 12.5% | 37.5% |

| | | |
|---|---|---|
| 9 iii. Evaluation (Changes) | 62.5% | 12.5% |
| 10. Generalize/Reuse | 75% | 75% |

The responses presented in these tables mask some differences in approach and general dispositions of some participants relating to the key CT areas: concepts, practices and perspectives. The next section will explore the verbal responses thematically to identify similarities and differences in the approach of the two groups of participants.

### *3.2 Computational Concepts*

The ability to elaborate and explain the purpose and function of the CT concepts they selected was the main difference between the responses of CS and non-CS participants. CS students used a diverse range of computational concepts in their responses to interview questions (particularly for Q5). They did so in a self-aware manner, able to explain why and how they were being used. For example, when discussing loops, selection and Booleans Participant 1 was able both name the concepts and to explain their use:

> *You would probably use a loop to loop from each of the different areas where the current piece could go. You would probably use IF and Else statements… to check if it already has a chess piece there or if it doesn't… You could use a switch statement also to check for the different pieces you've still got available and to see if they're dead or alive. You could probably use a Boolean to see if a chess piece is alive or dead. (CS Participant 1, Q5)*

GCSE CS participants were able to display a fluency and familiarity with CT concepts which made deploying them in problem solving reasoning seem logical and natural.

> *You would have to create like a basic counter, which would hold the values and then increment it or decrement it depending on if the value was adding or being taken away. (Participant CS 6, Q1)*

In the main, responses from the non-CS group were limited. Some could offer no answers for Q4 or Q5. When a prompt was given by the interviewer, for example, that a computational concept could be an IF statement or a loop, some non-CS participants were able to provide a basic answer focused exclusively on the 'IF' statement, most likely related to Key Stage 3 knowledge:

> *I'd probably use like IF: say the pawn moved onto where another pawn was, then it would be like a point to black, that sort of thing. (Non-CS Participant 17, Q5)*

To an extent this difference can be explained by the close alignment of some questions to tasks found on controlled assessments in CS. The CS students were more familiar with articulating this kind of reasoning.

### *3.3 Computational Practices*

#### 3.3.1 Abstraction

All CS and 7 non-CS students were able to offer reasonable responses to Q8, which focused on abstraction. When presented with the code 5 CS students spent time looking through the game compared to 3 non-CS students. Some CS and non-CS students chose to summarise as they read. Others were able to explore and then abstract key information from the code:

> *I think the diver starts off at the beginning of the maze and you have to go through and grab the stars, and this star indicates the finish. (CS Participant 4, Question 8)*

> *You sort of try to negotiate your way around the maze and collect the stars which will put you up points. But if you hit a green wall you are going down by minus 2. (Non-CS Participant 17, Question 8)*

### 3.3.2 Algorithmic Thinking

Both GCSE CS and non-GCSE CS students were able to explain their morning routine (Q4). However, CS students were better able to do so chronologically and with the use of conditionals (if… then…) to structure their responses.

> *If my alarm goes off at the right time, then I get up. If I actually get out of bed when I'm meant to, I go downstairs and have porridge, unless there's none left, and then I have toast. Then, if I have to walk my dog, then I do that, but if I don't then I just get ready for school. Then I just catch the bus, but if I miss that then I'll get a lift. (CS Participant 9, Question 4)*

There were also a number of CS students who discussed planning their responses to Question 1 with the use of an algorithmic flow chart, for example:

> *Before I like tried to do it, I would draw a flow diagram of what I had to do. (CS Participant 8, Question 1)*

Responses such as this demonstrated a strong grasp of two key concepts, sequencing and selection. This tendency to use a more algorithmic approach was not present in the non-GCSE CS group. Although they were able to describe their morning routine, the use of conditionals was not present in the same way:

> *Get in the car, get dropped off, then walk. And if I can't take my normal route I would... just get the bus if I can't get the car. (non-CS Participant 15, Question 4)*

Non- responses generally demonstrated some coherent chronological structure, but which still lacked algorithmic expression.

### 3.3.3 Debugging/Testing

Both GCSE CS and non-GCSE CS students took similar approaches to debugging. In order to find the bugs, the majority of participants chose to play the game rather than read the code. Those who read the code first could not find any bugs by doing so and then began to play.

Participants from both groups identified that there was a bug with one of the stars in the maze. However, none tested other stars to see if this result was inconsistent. All sixteen participants, regardless of group, took a linear approach to finding bugs by focusing on the goal of the maze. They only discovered those bugs that were in their path on the way to completion of the task. They did not, for instance, investigate whether all walls had the same functionality.

### 3.3.4 Decomposition

When approaching decomposition in Q2 no participants in either group talked in terms of taking a big problem and breaking it down into smaller problems. However, students from both groups explained how they would sort the data and make matches in the data to narrow it down.

*I would compare all of the information and see what parts of it are common to lots of sources and then use that information to find the murderer. (CS, Participant 9, Q2)*

*I'd probably make a table on… Excel or something and put in each suspect's names and what they've done, and I'd probably put evidence in and try to match up what suspect links in with the evidence and try to filter it that way. (non-CS, Participant, 17, Q2)*

### 3.3.5 Evaluation

When evaluating the game (Q9) the CS students were able to give a more detailed evaluation than the non-CS group. Although initially CS students were far more focused on the positives of the game than the negatives, with prompting from the interviewer they were able to give a more balanced view. Answers included reflections on the code that the game used. Some parts were criticised for being too complicated, others were praised for their simplicity.

*If you could like somehow like simplify all the blocks to make it look less complicated so you could spot errors if you had any. (CS, Participant 9, Q9)*

In contrast, non-CS participants only spoke about the experience of playing the game itself when discussing both positives and negatives. The non-CS students described the experience of playing the game as 'simple', without mention of the code. Most spoke about adding further levels to the game to add an increased level of difficulty:

*You could maybe have different levels of the game, for when you finish. (Participant 12, Question 9)*

### 3.3.6 Formulation of Aa Problem for a Computer

By including a stretch task (Q5), the researchers had hoped to explore the ability of the learners to formulate a problem in an appropriate way for a computer. Although challenging, this task was covered in the Key Stage 3 curriculum and so should not have been unfamiliar to any participants. Differences in approach and ideas indicate there were clear differences between CS and non-CS participants. The CS students discussed using various programming languages and operators throughout the interview whereas non-CS students did not. Many CS students discussed the actual operations that they would use. All CS answers were different but could be used as a valid approach to create a calculator:

*I would create a calculation function. I would probably use a Boolean to check if it's subtraction or addition, and then I would ask them to enter two different numbers, and then return the value once I've done the calculation. (Participant 1, Question 1)*

GCSE CS students also showed the ability to formulate a problem for a computer when answering Question 5. For example, one participant demonstrated understanding of how the chess game would need to be set up in a programme:

*I'd say if there's already a character thing on one of the spaces, make sure that you can only move the characters in the ways that those characters can be moved. (Participant 9, Question 5)*

In contrast, many non-CS students simply could not give an answer that formulated the problem in a meaningful way for a computer. The answers given were not accurate. Two participants attempted to come up with a solution but these lacked detail beyond using spreadsheet software (Participant 15). This task was covered in the Key Stage 3 curriculum and so should not have been unfamiliar to participants in either group.

#### 3.3.7 Generalization/Reusing

Overall, students from both groups were able to explain what they would reuse from the original game in a new game. The students from the CS group described how they would adapt the existing functionality to improve the game, for example, Participant 6 described how they would use the walls from the original game but would change the penalty incurred for hitting them:

> *Instead of having the point decrement, you could have it so it bounces you off. (CS, Participant 6, Question 10)*

In contrast, the non-GCSE CS students were able to describe how they would reuse elements of the original game but without adaptation:

> *You could use the maze walls as tracks. And you could use the finish as the finish line, so whoever gets there first wins. You could us a diver as the cars and stars as like extra points if you get them. (non-CS, Participant 15, Question 10)*

#### 3.3.8 Logical Reasoning

The three CS students who answered Q3a correctly also answered Q3b correctly as well. Although 4 non-CS students answered Q3a correctly, only 2 also got Q3b right. There were no obvious differences in the quality of logical reasoning displayed by either group in their approach to these tasks.

### *3.4 Computational Perspectives*

When invited to consider the effect of programming on daily life, there appeared to be a school effect among the GCSE CS students. Three of the four participants from this group in School A did not think there would be much effect of programming on daily life whereas all of the CS students from School B thought there would be. For example, when considering the personality of the programmer and the design of an automated chess player:

> *It could. Like if you're more of an attacking person, then it may go on to full attack, but if you're more of a defensive person then you could go to constantly defend. (Participant 7, Question 6)*

The non-CS responses from both schools indicated that this was not a topic they had considered in detail. Some non-CS students did not think there was any relationship between lifestyle and design (Q6). Others had ideas, but they had not experienced this for themselves:

> *Yeah, could do… like if you have a, well, stick to a sequence then it might be easier for you to like plan out how to do the algorithm. (Participant 12, Question 6).*

Regardless of their answers to Q6, almost all students in both groups appeared to think it obvious that programs and computers are having an important impact on the world. Students in the GCSE CS group were able to give more nuanced answers concerning to the wider impact of computers in the world compared to the non-GCSE CS group:

> *Yeah… I think in a positive and negative manner… they are very helpful and can do*

> *just easily, faster than people would. And I think they're quite time consuming [to make and maintain], for example. (Participant 4, GSCE CS group, Question 7)*
>
> *Yeah a lot… I don't know, it's just changed a lot over the years, like the development of technology is like has developed so much we're more reliant on them I guess. (Participant 13, non-GCSE CS group, Question 7)*

### 3.4.1 Terminology

Students in the GCSE CS group used computing terminology throughout their answers without prompting. They did so accurately, with the familiarity borne of exposure and practice. The language used by non-CS students suggested they saw computers as a 'black box', without any knowledge of its internal workings. They understood that computers were able to perform functions when given data inputs, but they did not understand how these then produced the outputs. In Q7 they repeatedly referred to programming as 'it' and to computers as 'them'. Participant 10's response to Q7 typifies the approach taken:

> *Yeah because, I don't know, there's a lot of Computer Science behind that no one is aware of, but it influences lots of technology and stuff online. (Participant 10, non-GCSE CS, Question 7)*

## *3.5 Summary*

There were evident differences in the answers given by GCSE CS and non-GCSE CS participants when they considered the computational concepts underpinning CT practices (see Table 5). Responses showed that in some areas there was surprisingly little difference between the CT practices of CS and non-CS students: Abstraction, Decomposition, Evaluation, Generalization/Reusing, Logical Reasoning and Debugging/testing. In other CT practices there were clear differences: Algorithmic thinking, Evaluation and Formulation of a problem for a computer. The data collected indicated that formulation of a problem for a computer was a particularly challenging task for students. There were also differences between the groups in their computational perspectives, seeing the impact of computers on daily life in very different terms. This was also reflected in their willingness to try to solve unfamiliar problems and the language they used to describe problems and solutions.

Table 5. Summary of Differences and Similarities in Responses by Participants in GCSE CS and Non-GCSE CS Groups

| **Area of Computational Thinking** | **Computer Science Students** | **Non-Computer Science Students** |
|---|---|---|
| **1. Computational Concepts** | Able to use many different concepts and correctly explain their use. | Many not able to answer; Some tried to explain 'if' statement. |
| **2. Computational Practices:** | | |
| **a) Abstraction** | Some found important details, and some talked about all parts. | Some found important details, and some talked about all parts. |
| **b) Algorithmic Thinking** | Explained routine in time order. | Explained routine, often out of time order. |
| **c) Debugging/ Testing** | Found some bugs, but not all. | Found some bugs, but not all. |
| **d) Decomposition** | Explained how to organise and sort data. | Explained how to organise and sort data. |

| | | |
|---|---|---|
| **e) Evaluation** | Main positive was that it was 'simple'. Some commented on repetition of code. | Main positive was that it was 'simple'. |
| **f) Formulate Problem for a Computer** | Gave detailed answers of potentially correct solutions. | Not able to explain how to create a calculator. |
| **g) Generalisation/ Reusing** | Explained what would be reused, with some criticality. | Explained what would be reused, with some criticality. |
| **h) Logical Reasoning** | Not consistent correct answers to Q3. Logical reasoning evident. | Not consistent correct answers to Q3. Logical reasoning evident. |
| **3. Computational Perspectives** | Thought that daily life affected programs. Gave specifics of the effect of programs on the world. | Thought programs affected the world, but not many details given. |

## 4. Discussion of Findings

The findings of the study in relation to the research questions are summarised below.

### 4.1 *Research Question 1: To what extent do CS students and non-CS students differ in their ability to apply computational concepts to scenario-based and practical computing problems?*

The detailed and accurate responses given by CS students suggest a strong knowledge of computational concepts. As a group, they were comfortable with the definition and usage of various concepts even when specifically asked. They tended to use computational concepts even when not specifically directed to do so. The non-CS students had more difficultly talking about computational concepts, with many unable to answer the questions.

### 4.2 *Research Question 2: To what extent do CS students and non-CS students differ in their ability to apply computational practices to scenario-based and practical computing problems?*

Algorithms and flowcharts created by the CS participants were better ordered than those of non-CS participants. They were able to create these as a means to structure thinking without prompting. Non-CS students designed algorithms that were less coherent, and they did not use algorithms to structure thinking without prompting.

There were a number of areas where there was little difference in the sophistication of approach between participants in either group: abstraction, debugging, generalisation, decomposition and logical reasoning. In particular, participants in both groups struggled to abstract from the practical to the general.

When evaluating programs, there was a difference in approach between participants in the two groups. The CS participants tended to approach evaluation from a programmer's perspective. They commented on the code and more closely evaluated how this was constructed. Non-CS participants tended to approach evaluation from a player's perspective, focusing on the end product rather than the underlying code.

### 4.3 *Research Question 3: To what extent do GCSE CS students and non-CS students differ in their ability to apply computational perspectives to scenario-based and practical computing problems?*

The CS participants demonstrated a richer understanding of how their lifestyle could affect their programming. They were also able to say specifically how this would happen. CS participants were also able to engage with discussion about how programs impact on the world around them. In their answers, many showed evidence of a deeper approach to CS: relying less on memory; able to demonstrate transferrable understanding of concepts; and able to use terminology with fluency (Ramsden, 2003). Non-CS participants were unable to give detailed examples of how changes in lifestyle would change the program. They were also less forthcoming about the role that computers play in daily life.

Overall, students studying for a GCSE qualification in CS demonstrated stronger CT skills than the non-CS students, as would be expected as a result of two additional academic years' worth of study. It is also fair to note that the majority of these strengths lay in the tasks directly related to programming. Given that the relevant literature in the field had highlighted a conceptual dividing line, Denning's (2017) classification of the traditional view of CT being cultivated through programming perhaps holds as true as the 'new' view of CT being seen as a conceptual framework that enables programming, assuming that programming skills are being taught. In this case, the study confirms that students who continued to study programming through GCSE CS had more strengths in this area. They showed they were stronger in the areas of computational concepts, algorithmic thinking, formulation of a problem for a computer, and computational perspectives.

To answer the research questions explored in this study, the data suggests that the CS students were better able to apply computational concepts and computational perspectives to new challenges than non-CS students. The opportunities to practice and apply their knowledge and skills ensured this. However, although the CS group performed better, the difference in ability between the two groups to apply computational practices was less pronounced. This is important because the wider STEM curriculum areas, recognising the importance of embracing CT in their cognate disciplines (Weintrop et al, 2016), are also making efforts towards this. This is the beginning of a working hypothesis suggesting that there are elements of CT that can be developed outside of programming, but that still have value in terms of overall CT education.

### *4.4 Willingness to Try*

There was also, in general, a greater willingness to try among CS participants, who, even when they did not know the answer, were willing to engage and offer a possible solution. Non-CS students frequently did not attempt answers to questions where they did not know an answer. For example, many non-CS students did not answer question 5 (focusing on computational concepts) despite being given support by the researcher. In addition, non-CS participants also appeared more likely to regard the computer as a 'black box'. They understood that computers had functionality and gave various outputs. However, they did not have an understanding of how this happened; indeed, there was a higher degree of apparent computer anxiety among these students that may have been related to their lower confidence in the use of computers (Doyle, *et al.*, 2005). The willingness to try may well indicate that higher degree of confidence with computers displayed by the CS group is the result of an existing pre-disposition that led them to opt for the course in the first place (Sam, *et al.*, 2005). This pre-disposition may then have been developed through additional experience using computers, leading to further improvements in their confidence (Compeau and Higgins, 1995).

### *4.5 Computational Thinking Skills and GCSE Computer Science*

The evidence presented in this study illustrates the differences in thinking between the two groups of participants, using Brennan and Resnick's (2012) model of the interface between computers and people (computational perspectives) and in their understanding of the underlying concepts that enable this relationship (computational concepts). These point to the development of the 'habits of mind' referred to by Denning (2017).

CS participants demonstrated greater fluency in the use of some computational practices in comparison to participants from the non-CS group (formulation of a problem, algorithmic thinking and evaluation). As such it would appear that the GCSE CS students are developing some but not all of the ‘practical skills’ described by Wing (2006). However, the similar behaviour by participants in both groups in some areas of computational practices suggests that there remain considerable overlaps with some other skill sets required for Mathematics and Science, which aligns with other studies that have identified CT practices in Science and Mathematics classrooms. (Tedre and Denning, 2016). On one hand the crossover between disciplines can be seen as a reflection on the evolving nature of research and study in Science and Mathematics where computing has become an ever more essential skill in recent years (Weintrop, *et al.*, 2016). On the other, it may support the idea that a large part of CT draws upon a broad and deep range of higher order thinking skills which underpin learning throughout the STEM curriculum (diSessa, 2018).

### *4.6 Limitations of the Method*

The number of participants in the sample was small due to the pressures of the curriculum at GCSE (students and teachers were intensely focused on the end of year exams) and also due to the relatively small number of GCSE CS students in each cohort. While the study tried to include elements of a design scenario structure (e.g. the summarising and debugging of the game) that related to the assessment tools categorised by Román-González et al. (2017), examination pressures meant that it was not possible to work with participants to develop a full design scenario study. Future research should focus on greater utilization of think-aloud interview protocols in combination with innovative digital data collection methods, which have proved fruitful in exploring teacher reasoning in this area (Hidson, 2018).

Although there is no firm evidence of this in the data collected, it is not possible to discount a teacher effect impacting on CS students from each of the two schools, particularly given shortages in this subject (Kemp et al., 2018). The necessarily limited range in teacher perspectives may have influenced attitudinal aspects of the CS curriculum such as computational perspectives. Future studies should seek where possible to draw participants from a wider base of schools to broaden the range of teacher inputs received across the range of participants. Interrogating the areas where there are fewer differences could also be fruitful, as this points to the area where other cognate areas may overlap and provide complementary CT development, leaving programming as the rightful place for thinking about automation.

In School A, the class was comprised entirely of male students. The gender balance was more equitable in School B. Whilst this reflected national trends in uptake of this subject at the time (Ofqual, 2018b), increasing the number of schools in future studies may yield a greater pool of students of both genders from which participants can be selected. This would allow exploration of gender differences in the adoption of CT concepts, practices and perspectives.

## 5. Conclusion

The interviews with CS and non-CS students indicate that there is some difference in areas of CT concepts, practices and perspectives (Brennan & Resnick, 2012). The introduction of a dedicated GCSE in Computer Science does have much to contribute to the development of a distinct disciplinary identity that can be articulated by the student. It allows and encourages the development of computational thinking for automation purposes that is not present in the general computational thinking skills displayed by the non-CS students.

A similar level of performance was shown by participants from both groups in some CT practices: namely, abstraction, debugging, generalisation, decomposition and logical reasoning. This may indicate the potential for these skills to be fostered successfully through other areas of the curriculum (Berry, 2019, Weintrop, 2017), or indeed to a sufficient extent in the Key Stage 3 computing curriculum. Further research should be conducted in this area to compare the types of computational thinking generated specifically in Mathematics or Design and Technology versus the evident computational thinking for automation that is present in the GCSE CS students' responses. Greater understanding of the potential for other disciplines to develop CT skills may alleviate pressure on under-staffed CS departments and enable the design of cross-curricular projects that meet the needs of CS and other STEM subjects. If computational thinking is to realise the benefits ascribed by Wing (2014), then logic suggests that additional study of how it is developed and transferred across cognate disciplines is needed.

The key finding from this study is that it is most likely the increased focus on programming in Key Stage 4 as part of GCSE CS that is responsible for the elements of computational thinking for automation that have hitherto been promoted as part of the universality of CT. The controversial point to be made is that this is something that can only be developed by continuing to learn programming. Rather than seeing this as a point of deficit, the concluding suggestion is that computational thinking for automation should be seen as the advanced development and application of CT specific to those whose interests and aptitudes lead them to opt to continue their study of programming. General computational thinking skills can be successfully developed at a lower level of study or in cognate areas, such as Science or Mathematics.

**Acknowledgements**

Scratch is a project of the Scratch Foundation, in collaboration with the Lifelong Kindergarten Group at the MIT Media Lab. It is available for free at https://scratch.mit.edu". The screenshot (Figure 1) from the Scratch application is licensed under the Creative Commons Attribution-ShareAlike license.

**References**

Allsop, Y., (2019). Assessing computational thinking process using a multiple evaluation approach. International journal of child-computer interaction, 19, pp.30-55.

Barr, V. & Stephenson, C. (2011). Bringing computational thinking to K-12. ACM Inroads, 2(1), 48-54. doi:10.1145/1929887.1929905

Berry, M., (2019). 'What I'm thinking about computational thinking'. *Hello World #8*. p. 97. Available online: https://helloworld.raspberrypi.org/issues/8/pdf. (Accessed 20/04/2020)

Bilbao, J., Bravo, E., García, O., Varela, C. & Rebollar, C. (2017), "Assessment of Computational Thinking Notions in Secondary School", Baltic Journal of Modern Computing, vol. 5, no. 4, pp. 391-397.

Brennan, K., & Resnick, M. (2012). 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 (pp. 1-25).

Brown, N. C., Sentance, S., Crick, T., & Humphreys, S. (2014). Restart: The resurgence of computer science in UK schools. ACM Transactions on Computing Education (TOCE), 14(2), 9.

Burden, R. (2015). 'Evidence for the efficacy of thinking skills approaches in affecting learning outcomes: the need for a broader perspective.' In Wegerif, L. and Kaufman J. (Eds.) The Routledge International Handbook of Research on Teaching Thinking. Abingdon: Routledge, pp. 291-304.

Burke, Q. (2012). The markings of a new pencil: Introducing programming-as-writing in the middle school classroom. *Journal of Media Literacy Education, 4(2),* 121-135.

Cansu, S. K., & Cansu, F. K. (2019). An Overview of Computational Thinking. International Journal of Computer Science Education in Schools, 3(1).

Compeau, D. R., & Higgins, C. A. (1995). Computer self-efficacy: Development of a measure and initial test. MIS quarterly, 189-211.

Computing at School (2020). "Resources". Available online at https://community.computingatschool.org.uk/resources/. Last accessed 24/04/2020.

Crick, T. (2017). Computing Education: An Overview of Research in the Field. London: Royal Society

CSTA and ISTE. (2011). *Computational Thinking in K–12 Education leadership toolkit.* http://csta.acm.org/Curriculum/sub/CurrFiles/471.11CTLeadershiptToolkit-SP-vF.pdf

Denning, P. J. (2017). Remaining trouble spots with computational thinking. Communications of the ACM, 60(6), 33-39.

Department for Education. (2013). Computing programmes of study: key stages 3 and 4 National curriculum in England. Retrieved from https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study

Department for Education (2015), Computer science: GCSE Subject Content, retrieved from: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/397550/GCSE_subject_content_for_computer_science.pdf

diSessa, A. (2018). Computational Literacy and "The Big Picture" Concerning Computers in Mathematics Education, Mathematical Thinking and Learning, 20:1, 3-31, DOI: 10.1080/10986065.2018.1403544

Doyle, E., Stamouli, I., & Huggard, M. (2005). Computer anxiety, self-efficacy, computer experience: An investigation throughout a computer science degree. In Frontiers in Education, 2005. FIE'05. Proceedings 35th Annual Conference (pp. S2H-3). IEEE.

Fessakis, G., Gouli, E., & Mavroudi, E. (2013). Problem solving by 5–6 years old kindergarten children in a computer programming environment: A case study. Computers & Education, 63, 87-97.

Fields, D. A., Searle, K. A., Kafai, Y. B., & Min, H. S. (2012). Debuggems to assess student learning in e-textiles. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education* (pp. 699-699). ACM.

Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. Educational Researcher, 42(1), 38-43.

Hidson, E, (2018). Challenges to Pedagogical Content Knowledge in lesson planning during curriculum transition: a multiple case study of teachers of ICT and Computing in England, Durham theses, Durham University. Available at Durham E-Theses Online: http://etheses.dur.ac.uk/12623/

Kallia, M. (2017). Assessment in Computer Science courses : A Literature Review. London: Royal Society.

Kemp, P.E.J., Berry, M.G. & Wong, B. (2018). The Roehampton Annual Computing Education Report: Data from 2017. London: University of Roehampton.

Korkmaz, Ö., Cakir, R. and Özden, M.Y., (2017). A validity and reliability study of the computational thinking scales (CTS). Computers in Human Behavior, 72, pp.558-569.

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J. & Werner, L. (2011). Computational thinking for youth in practice. ACM Inroads, 2(1), 32-37. doi:10.1145/1929887.1929902

Livingston, K., Hayward, L., Higgins, S., Wyse, D., (2015). Multiple influences on curriculum decisions in a supercomplex world. Curriculum Journal, 26(4), 515-517.

Lockwood, J., & Mooney, A. (2018). Computational Thinking in Secondary Education: Where Does It Fit? A Systematic Literary Review. International Journal of Computer Science Education in Schools, 2(1).

Lye, S. & Koh, J. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? Computers in Human Behavior, 41, 51-61. doi:10.1016/j.chb.2014.09.012

Moseley, D. (2005). Frameworks for thinking: A handbook for teaching and learning. Cambridge University Press.

Noone, M., & Mooney, A. (2018). Visual and textual programming languages: a systematic review of the literature. Journal of Computers in Education, 5(2), 149-174.

Ofqual. (2018a). GCSE Subject Level Conditions and Requirements for Computer Science. Department for Education. Retrieved from: https://www.gov.uk/government/publications/gcse-9-to-1-subject-level-conditions-and-requirements-for-computer-science

Ofqual.(2018b). Entries for GCSE, AS and A level Summer 2018 exam series. Retrieved from https://www.gov.uk/government/statistics/entries-for-gcse-as-and-a-level-summer-2018-exam-series

Papert, S. (1980). Mindstorms. Children, computers and powerful ideas. New York: Basic Books

Ramsden, P. (2003). Learning to Teach in Higher Education (2nd ed., pp. 43-). Oxon: RoutledgeFalmer.

Riley, D. & Hunt, K. (2014). Computational thinking for the modern problem solver (1st ed.). Boca Raton, Fla: CRC Press.

Román-González, M., Moreno-León, J., & Robles, G. (2017). Complementary tools for computational thinking assessment. In *Proceedings of International Conference on Computational Thinking Education (CTE 2017), S. C Kong, J Sheldon, and K. Y Li (Eds.). The Education University of Hong Kong* (pp. 154-159).

Sam, H. K., Othman, A. E. A., & Nordin, Z. S. (2005). Computer self-efficacy, computer anxiety, and attitudes toward the Internet: A study among undergraduates in Unimas. Educational Technology & Society, 8(4), 205-219.

Selby, C., Dorling, M., & Woollard, J. (2014). Evidence of assessing computational thinking. Author's original, 1-11.

Selby, C. & Woollard, J. 2013. Computational Thinking: The Developing Definition. Available: http://eprints.soton.ac.uk/356481/

Sentance, S., and Selby, C. (2015) A classification of research into computer science education in school from 2005-2014: Initial report, Retrieved from https://community.computingatschool.org.uk/resources/4119/single.

Tedre, M. and Denning, P. J. (2016) The Long Quest for Computational Thinking. In Proceedings of the 16th Koli Calling Conference on Computing Education Research, pages 120-129.

The Royal Society (2012). Shut down or restart? The way forward for computing in UK schools. The Royal Society, London.

Royal Society. (2017). After the reboot : computing education in UK schools. London: Royal Society. Retrieved from https://royalsociety.org/topics-policy/projects/computing-education/

Waite, J. (2017). Pedagogy in teaching Computer Science in schools: A Literature Review. London: Royal Society.

Webb, D. C. (2010). Troubleshooting assessment: an authentic problem solving activity for it education. Procedia-Social and Behavioral Sciences, 9, 903-907.

Webb, M., Davis, N., Bell, T., Katz, Y. J., Reynolds, N., Chambers, D. P., & Sysło, M. M. (2017). Computer science in K-12 school curricula of the 2lst century: Why, what and when? Education and Information Technologies, 22(2), 445-468.

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

Werner, L., Denner, J., & Campe, S. (2015). Children Programming Games: A Strategy for Measuring Computational Learning. *ACM Transactions On Computer Education, 14(4)*, 24:1-24:22. doi:10.1145/2677091

Wing, J. (2006). Computational thinking. Communications Of The ACM, 49(3), 33-35. http://dx.doi.org/10.1145/1118178.1118215

Wing, J., (2014). 'Computational thinking benefits society'. *Social Issues In Computing 40th Anniversary Blog*. January 10th, 2014. Available online at: http://socialissues.cs.toronto.edu/2014/01/computational-thinking/

Zhong, B., Wang, Q., Chen, J., & Li, Y. (2015). An Exploration of Three-Dimensional Integrated Assessment for Computational Thinking. Journal Of Educational Computing Research, 53(4), 562-590. http://dx.doi.org/10.1177/0735633115608444

# Factors Affecting Engineering Students' Achievement in Computer Programming

**Melih Derya Gurer**[1]

**Seyfullah Tokumacı**

[1]Bolu Abant Izzet Baysal University

DOI: 10.21585/ijcses.v3i4.74

**Abstract**

Literature indicated that attitude toward programming, programming self-efficacy, gender, and students' department was related to achievement in computer programming. However, there is a need for further studies investigating to what extent these factors explain programming achievement in a model. This study aimed to investigate the effects of programming self-efficacy, attitude towards programming, gender, and students' department on their perceived learning. The correlational study design was adopted for this study. The sample of the study was 742 students of an engineering faculty at a state university in Turkey. To collect data, Programming Self-Efficacy Scale, Computer Programming Attitude Scale, and Perceived Learning Scale were used. To analyze data, descriptive statistics such as mean, standard deviation, and Pearson Correlation tests were administered. In addition, to determine the factors affecting perceived learning, multiple regression analysis was employed. The results indicated that the engineering faculty students' attitudes towards programming, programming self-efficacy, and perceived learning were at a high level. In addition, significant correlations between perceived learning and predictive variables were found. Finally, it was concluded that gender, attitude towards programming, and programming self-efficacy significantly predicted perceived learning. The results of the study provide a deeper understanding of how students' learning was affected in programming courses.

**Keywords:** computer programming, perceived learning, attitude, self-efficacy, gender

## 1. Introduction

Computational thinking has been regarded as one of the crucial skills of next-generation students (International Society for Technology in Education [ISTE], 2016). Core components of computational thinking curated by ISTE (2016) are decomposition, gathering and analyzing data, abstraction, and algorithm design. Decomposition is the breaking down of a larger problem into smaller and manageable parts. Gathering and analyzing data refers to collecting, organizing, and representing data. Abstraction is determining what parts of the problem can be ignored, to decrease the computational complexity of a problem. Algorithm design is the process of designing a step-by-step process to achieve a task (ISTE, 2016). Such skill is not only crucial for students' professional careers, but also for the industry's economic competitiveness and the ability to find qualified employees (Gardiner, 2017). For this reason, as a fundamental tool of computational thinking, many studies have been carried out to introduce students to computer programming in all levels of education, from elementary school to graduate level.

Scholars have proposed that programming is one of the essential skills for many engineering schools (Hodge & Steele, 2002). To Zyda (2009), strong programming skills would be one of the essential criteria for the graduates of engineering to be employed by the industry. Programming education instills some of the concepts and abilities of computational thinking and provides a basis for computational thinking, which helps in following a mental path through comprehension and understanding of concepts (Kılıçarslan-Cansu & Cansu, 2019). Therefore, programming education is essential for shaping the perceptions and thinking strategies of engineering students.

Engineering faculty students face with the task of solving problems by using numerical approaches in their 1st and 2nd grades. Good programming skills will enable them to easily solve these problems (Naraghi & Bahman, 2001). Therefore, it is important for engineering and technology students to learn basic computer programming skills in the first years of university education. Almost all engineering programs contain basic information about programming as part of their curriculum. Introduction to programming languages is an essential and compulsory course for students in computer engineering, software engineering, information systems engineering, as well as in many engineering fields such as electrical engineering, industrial engineering, civil engineering, mechanical engineering.

With the emphasis on computer programming and the proliferation of programming education, the number of studies on programming education has increased. Researchers have studied the learning and teaching of programming (Askar & Davenport, 2009; Yılmaz, 2013), attitude toward programming (Anastasiadou & Karakos, 2011; Gurer, Cetin, & Top, 2019; Korkmaz & Altun, 2013), and perception of programming self-efficacy (Akçay & Çoklar, 2018; Özyurt & Özyurt, 2015). In addition, there are studies on the factors related to programming achievement (Askar & Davenport, 2009; Başer, 2013a; Clinkenbeard, 2017). Related literature has indicated that attitude toward programming, programming self-efficacy, gender, and department of students are related to programming achievement. However, the correlations between the variables are needed to be examined in a more comprehensive manner. Although a correlation between each variable and the programming achievement has been shown, there is a need to investigate to what extent each variable explains programming achievement. An investigation of factors is supposed to guide teachers in designing computer programming courses.

## *1.1 Literature Review*

As in the teaching of many disciplines and fields, achievement has been one of the topics that are emphasized and examined in programming studies. Literature showed that mostly test grades or final course grades were used to measure the level of learning. However, Ewell (1994) stated that grades might have little correlation with what students learned, and learning can also be measured effectively with self-assessment tools. Rovai and Barnum (2003) asserted that the use of grades to functionalize learning does not always give the best results. Learners can monitor their learning, and therefore perceived learning could be a valid measure of student learning (Metcalfe, 2009). Alavi, Marakas, and Yoo (2002) define perceived learning as the changes in the perceptions of learners about their knowledge and skill levels before and after the learning experience. According to Rovai, Wighting, Baker, and Grooms (2009), perceived learning has three components; cognitive, affective, and psychomotor learning. The cognitive domain is expressed as remembering or recognizing information, while the affective domain is expressed as the development of positive attitudes towards a specific content or subject. The psychomotor field is described as the development of skills related to manual tasks and physical movement. Considering components of perceived learning, it is expected that attitude toward computer programming and programming self-efficacy influence perceived learning.

Several researchers have aimed to examine the relationship between achievement and attitude in programming studies, and confounding results have been found. Aiken (2002) suggested that attitude is learned cognitive, affective, and behavioral tendencies to respond positively or negatively to specific objects, situations, institutions, concepts, or people. According to this definition, attitude consists of three dimensions: (1) cognitive dimension, consisting of beliefs about the object of attitude, (2) affective dimension, consisting of feelings about the object, and (3) behavioral dimension, composed of tendencies of action towards the object. As a result of their experimental study, Cetin and Andrews-Larson (2016) and Hongwarittorrn and Krairit (2010) stated that there was no explicit relationship between students' computer programming achievement and their attitudes towards computer programming (ATCP). Korkmaz (2016) confirmed this result in another study. On the other hand, after investigating 113 studies, Ma and Kishor (1997) offered a positive correlation between attitude and achievement. The positive relationship between attitude and achievement has been confirmed in computer programming related studies. In a recent study, the Gurer et al. (2019) investigated the relationship between perceived learning and ATCP and found that there was a positive correlation between two variables. Başer (2013a) conducted a study with

179 prospective teachers and stated that there was a significant relationship between students' ATCP and their success in programming. Additionally, Lee, Kim, and Lee (2017) conducted a study with 4221 primary school students and found that the ATCP was highly correlated with academic achievement in programming education. Hence, further investigation is needed to investigate the potential relation between students' learning and their attitudes towards computer programming.

Self-efficacy has been another psychological construct researched in computer programming studies. Perceived self-efficacy means one's beliefs about his or her ability to regulate and conduct the behavior to achieve specific goals (Bandura, 1997). According to Bandura, self-efficacy influences the way people think, motivate themselves, and behave. Schunk (1989) claimed that perceived self-efficacy is an important construct that directly influences students' learning and achievement-related behaviors. With this point of view, students with high self-efficacy tend to be more motivated, persistent, and perform better in a given task. On the other hand, students with low-level of self-efficacy perceive a given task threatening and unchallenging. Askar and Davenport (2009) stated that students' self-efficacy would lead to their future success. In a study related to computer programming, Yılmaz (2013) found that computer programming self-efficacy (CPSE) had a significant effect on programming achievement. Similarly, Wiedenbeck, LaBelle, and Kain (2004) stated that CPSE has a direct impact on students' overall achievement in programming. Clinkenbeard (2017) concluded that students' computer self-efficacy is an important determinant of their success in the introduction to computer programming. In a recent study, Cigdem (2017) indicated that self-efficacy was the strongest positive determinant of achievement in programming courses. Further studies need to consider the effect of self-efficacy in explaining students' computer programming achievement.

Gender, which may be one of the potential factors that affect students' programming achievement, should also be investigated. It has been argued that females are not adequately represented in computer-related work and computer science (Doube & Lang, 2012; Singh, Allen, Scheckler & Darlington, 2007) for some cultural and environmental reasons. Moreover, it was reported that males have higher attitudes towards computer programming than females (Başer, 2013b; Korkmaz & Altun, 2013; Özyurt & Ozyurt, 2015). Contrary to these results, some studies show that female students have higher programming success than male students. For example, Yılmaz (2013) concluded that female students' computer programming success was significantly higher than male students. Similarly, Pioro (2004) stated that female students had higher success in computer programming than male students. Lau and Yuen (2009) reported that in computer programming, secondary school female students perform slightly higher than male students. Despite such studies pointing to gender differences in information and communication technologies (ICT), gender differences in ICT use have generally been shown to decrease (Alsadoon, 2013; Top, Yukselturk & Cakir, 2011).

Students' discipline may be one of the factors which affect the achievement of computer programming. Each discipline puts a different emphasis on computer programming, and this emphasis is reflected in the curriculum of the program. The computer engineering programs have five or more compulsory and several selective programming courses in their curriculum. On the other hand, the curriculum of other engineering programs includes two or more compulsory programming courses. Students of these programs register to programming courses related to their interests and motivations. Studies are indicating a significant relationship between the students' discipline and the variables that are thought to be related to programming achievement. Ülkü, Doğan, Demir, and Yıldız (2017) reported that the electrical-electronics engineering department students' self-efficacy perception of programming is higher than that of the textile engineering department. Gezgin and Adnan (2016) found that there was a significant relationship between students' self-efficacy and the discipline of students. Altun and Mazman (2012) and Askar and Davenport (2009) state that computer engineering students have a higher perception of programming self-efficacy than students in other departments. Moreover, Korkmaz and Altun (2013) and Başer (2013b) found that computer engineering students had more positive attitudes towards programming than other department students. Although students' achievement in computer programming takes attention, the research on the factors affecting achievement in computer programming is still limited. This study aims to investigate the factors related to students' perceived learning on computer programming (PLCP), and to what extent gender, department, computer programming self-efficacy (CPSE), and attitude toward computer programming (ATCP) predict students' PLCP.

This study was guided with the following research questions:

1) What are the engineering students' PLCP, attitudes towards computer programming, and computer programming self-efficacy?
2) Is there a significant correlation between students' PLCP and the predictor variables (gender, department, computer programming self-efficacy, and attitude toward computer programming)?

3) What are the significant predictors of students' PLCP, and to what extent do the predictor variables explain PLCP?

## 2. Methodology

The correlational study design was implemented for this study. The relationship between the two or more variables, where no interventions are applied to the variables, are examined with correlational studies (Fraenkel, Wallen, & Hyun, 2015). As this study aims to examine to what extent the selected variables (gender, department, CPSE, and ATCP) accounts for engineering students' PLCP, a correlational study was considered to be appropriate for this study. The dependent variable of the study was engineering students' PLCP, and the independent variables were their gender, department, CPSE, and ATCP.

### *2.1 Participants*

The current study was conducted with 742 voluntary students of an engineering faculty in a state university located in the northeastern part of Turkey in the spring semester of 2018-2019 academic year. The participants were briefed about the purpose of the study and the privacy of the data with a statement on the first page of the questionnaire. They were also told that they had the right to withdraw from the study at any time. Table 1 indicates the demographics of the participant students.

Table 1. Demographic information about the participants

| Variables | Group | N | % |
|---|---|---|---|
| Gender | Female | 204 | 27.49 |
| | Male | 538 | 72.51 |
| Department | Computer Engineering | 54 | 7.28 |
| | Electrical & Electronics Engineering | 65 | 8.76 |
| | Industrial Engineering | 63 | 8.49 |
| | Civil Engineering | 69 | 9.30 |
| | Mechanical Engineering | 64 | 8.63 |
| | Mechatronics Engineering | 95 | 12.80 |
| | Metallurgy & Materials Engineering | 68 | 9.16 |
| | Automotive Engineering | 46 | 6.20 |
| | Rail Systems Engineering | 72 | 9.70 |
| | Medical Engineering | 63 | 8.49 |
| | Transportation Engineering | 42 | 5.66 |
| | Environmental Engineering | 41 | 5.53 |
| Grade level | Freshmen | 106 | 14.29 |
| | Sophomore | 162 | 21.83 |
| | Junior | 236 | 31.81 |
| | Senior | 238 | 32.08 |
| Total | | 742 | 100 |

The age of the students varied between 18 and 26, and the mean age was computed as 21.15. The number of male students (72.51%) was more than females (27.49%). While the number of juniors (31.81%) and seniors (32.08%) were nearly the same, they were more than freshman (14.29%) and sophomore (21.83%) students. In the faculty where this study was conducted, the students in the computer engineering department take six compulsory programming language courses, electrical-electronics and mechatronics students take three compulsory programming language courses, and students of other departments take one or two compulsory courses on programming language courses. However, the students of all departments take at least one course on programming languages in their first grade. As the data were collected at the end of the spring semester, it was considered that all the students who participated in this study had at least one computer programming language course. None of the students reported that they had not taken a programming language course. In addition, their success or failure in the programming language course(s) was not considered as a criterion for a student to be a participant of this study.

### *2.2 Data Collection*

Data of this study were collected with a paper-based survey at the end of the course year. After having required permissions from the faculty and the instructors, the surveys were administered to engineering faculty students in their classrooms. Only the volunteers completed the surveys. In this study, the surveys used to collect data were perceived learning on computer programming (PLCP) scale, computer programming self-efficacy (CPSE) scale, and attitudes toward computer programming (ATCP) scale.

PLCP was used to measure engineering students' perceived learning levels at computer programming lessons. The Perceived Learning survey was originally developed by Rovai et al. (2009) to reveal students' perceptions of their learning in any course. The initial form consisted of nine items in three constructs; cognitive, affective, and psychomotor learning. Then the survey was adopted by Top, Yukselturk, and Inan (2010) resulting in nine items and the Cronbach's alpha internal consistency coefficient of 0.81. The items were in a 5-point Likert type ranging from 1 = completely disagree to 5 = completely agree. In this study, the Cronbach's alpha coefficient was found to be 0.75, which was good (Field, 2009).

The CPSE was originally developed by Ramalingam and Wiedenbeck (1998) to study higher education students' self-efficacy beliefs on a computer programming language. The scale development study resulted in four factors; (1) independence and persistence, (2) complex programming tasks, (3) self-regulation, and (4) simple programming tasks. The reliability coefficient was computed as 0.98. The original survey was adopted by Altun and Mazman (2012). It resulted in nine items within two factors; the ability to perform simple programming tasks and the ability to perform complex programming tasks. They found the reliability of the scale as 0.93. In this study, the reliability coefficient was calculated as 0.88, which was good (Field, 2009).

To investigate higher education students' attitudes towards computer programming, Cetin and Ozden (2015) developed the ATCP. It included 18 items in a 5-point Likert type within three factors (affection, cognition, and behavior). The internal reliability coefficient of the original scale was determined to be 0.94. In this study, the Cronbach's alpha value of the scale was found to be 0.86, which was good (Field, 2009).

### *2.3 Data Analysis*

The negative items in the scales were reversed before the data analysis. Initially, descriptive statistics such as mean, standard deviation, skewness, and kurtosis were administered to analyze the data. The skewness and kurtosis tests were used to check the normality of data. To Field (2009), a normally distributed sample is satisfied if 95% of z-scores of skewness and kurtosis should lie between −1.96 and +1.96. The skewness and kurtosis values of each variable ranged between -0.57 and -0.06. Hence, it could be said that the data of each factor were normally distributed. Then, to investigate the correlation among the engineering students' gender, department, CPSE, ATCP, and PLCP, the Pearson Product-Moment Correlation test was administered. Finally, multiple regression analysis was run to examine to what extent the independent variables explain the students' PLCP. For multiple regression analysis, as the gender and department variables were nominal type variables, dummy coding was applied to the two variables. For gender, male was coded as "1", and female was coded as "0". In addition, for department variable, computer engineering was coded as "1", and the others were coded as "0". Multiple regression analysis was run based on this dummy coding.

## 3. Findings

### *3.1 Descriptive Findings*

The engineering students' PLCP, CPSE, and ATCP scores, including the sub-dimensions of the scales, are indicated in Table 2.

Table 2. Descriptive statistics for each scale

| **Variables** | **$\bar{X}$** | **S** |
|---|---|---|
| Attitude toward computer programming | 3.61 | 0.58 |
| Cognitive | 3.25 | 0.73 |
| Affective | 3.42 | 0.86 |
| Behavioral | 4.15 | 0.64 |
| Computer programming self-efficacy | 3.29 | 0.80 |

| | | |
|---|---|---|
| Self-efficacy of simple tasks | 3.93 | 0.95 |
| Self-efficacy of complex tasks | 2.65 | 0.87 |
| Perceived learning on computer programming | 3.15 | 0.55 |
| Cognitive | 2.97 | 0.70 |
| Affective | 3.28 | 0.84 |
| Psychomotor | 3.20 | 0.65 |

On the 5-point Likert type scale, it was found that students' ATCP were at a moderately high level. Although students' attitudes at affective ($\bar{X}$ = 3.42) and behavioral ($\bar{X}$ = 4.15) dimensions were found to be at a high level, they were at a moderate level in cognitive dimension ($\bar{X}$ = 3.25). Students' CPSE was found to be moderate ($\bar{X}$ = 3.29). While their self-efficacy for simple tasks was at a moderately high level ($\bar{X}$ = 3.93), their self-efficacy for complex tasks was found to be at a moderate level ($\bar{X}$ = 2.65). In addition, students' PLCP were found at a moderate level ($\bar{X}$ = 3.15). Likewise, their perceived learning at all sub-dimensions was found to be moderate.

### *3.2 Correlations among Variables*

Pearson correlation test was carried out to investigate the correlations between PLCP and the predictive variables. The correlation test was administered after the dummy coding of gender and department variables. However, as gender and department are categoric variables, the correlation between these variables was not computed. Table 3 shows the correlations between the variables.

Table 3. Pearson correlation coefficients among variables

| | **PLCP** | **Gender** | **Department** | **CPSE** | **ATCP** |
|---|---|---|---|---|---|
| 1. PLCP | 1 | -0.09* | 0.20** | 0.58** | 0.56** |
| 2. Gender | | 1 | --- | -0.02 | 0.00 |
| 3. Department | | | 1 | 0.25** | 0.15** |
| 4. CPSE | | | | 1 | 0.43** |
| 5. ATCP | | | | | 1 |

*. Significant at the 0.05 level.

**. Significant at the 0.01 level.

All of the predictive variables were significantly correlated with PLCP. A negative significant correlation between PLCP and gender (r = -0.09) means that, depending on the dummy coding, females' PLCP was higher than males' PLCP. In addition, due to the dummy coding of the department variable, the positive and significant correlation between PLCP and department means that computer engineering students' PLCP was higher than that of other departments. While the correlation between PLCP and department (r = 0.20) were found to be positive and low, PLCP was positively and moderately correlated with both CPSE (r = 0.58) and ATCP (r = 0.56). Furthermore, CPSE was found to be positively and moderately (r = 0.43) correlated with ATCP.

### *3.3 Regression Analysis*

As gender, department, CPSE, and ATCP were found to be significantly correlated with PLCP, they were entered the multiple regression analysis to test how well PLCP can be explained by them (Field, 2009). Firstly, possible multicollinearity, which is one of the assumptions of multiple regression, between the dependent and the independent variables was examined. Table 3 shows that the correlation coefficients were not higher than 0.80. Strong correlations between the predictor variables make it difficult to distinguish the unique estimates of regression coefficients (Cohen, Cohen, West, & Alken, 2003). Additionally, for the current model, the variance inflation factors (VIF) values are all below 10, and the tolerance statistics all well above 0.2. Hence multicollinearity between the predictors, which is the violation of one assumption of multiple regression analysis, was not worthy of concern for this study. Table 4 presents the results of the multiple regression analysis tests.

Table 4. Regression analysis results

| Variables | B | Std. Err. | Beta (β) | t | p | Zero-order r | Partial r |
|---|---|---|---|---|---|---|---|
| (Constant) | 1.054 | .120 | --- | 8.790 | .000 | --- | --- |
| Gender (Male) | -.099 | .034 | -.080 | -2.927 | .004* | -.086 | -.107 |
| Department (Computer Eng.) | .002 | .004 | .013 | .476 | .634 | -.090 | .018 |
| ATCP | .362 | .028 | .382 | 12.709 | .000* | .558 | .424 |
| CPSE | .287 | .021 | .413 | 13.632 | .000* | .577 | .449 |
| R = 0.676 R2 = 0.458 R2 adjusted = 0.455 F = 155.385 p = 0.00 | | | | | | | |

* Significant at 0.01 level

As a result of multiple linear regression analysis, gender, department, ATCP and CPSE variables together, showed a significant relationship with PLCP ($R = 0.676$, $R_2 = 0.458$) ($F = 155.385$, $p < 0.01$). In other words, it was found that this model was found to be significant and accounted for 45.8% of the variance ($R = .676$) in engineering students' PLCP. While ATCP ($t=12.709$, $p<0.01$) and CPSE ($t=13.632$, $p<0.01$) were positive significant predictors of the PLCP, being male student had negative significant impact on students' PLCP ($t=-2.927$, $p<0.01$). With this finding, it can be said that female students with high ATCP and CPSE scores were expected to have higher PLCP scores. According to the standardized regression coefficient beta (β) in the table, the relative importance of predictive variables on PLCP was as following; (1) computer programming self-efficacy ($\beta = 0.413$), (2) attitude towards computer programming ($\beta = 0.382$), (3) gender ($\beta = 0.080$), and (4) department ($\beta = 0.013$). Based on the multiple regression analysis, the regression equation for PLCP was;

$$PLCP=(0.362xATCP)+(0.287xCPSE)-(0.099xgender(male))+(0.002xdepartment(computer\ eng.))+1.054 \quad (1)$$

## 4. Discussion

This study investigated the factors affecting engineering students' perceived learning in computer programming. Related literature highlighted that gender, department, computer programming self-efficacy, and attitude toward computer programming were related to perceived learning. These variables were subjected to multiple regression analysis to predict PLCP. The analysis of data showed that these variables accounted for 45.8% of the variance in PLCP. While gender, attitude toward computer programming, and computer programming self-efficacy had a significant influence on PLCP, students' department was not determined to be a significant predictor of PLCP.

There was a positive and significant relationship between students' PLCP and ATCP in the current study. According to this result, it can be estimated that students with positive attitudes towards programming are likely to have higher PLCP, and students with low attitudes tend to have low PLCP. Furthermore, ATCP was found to be one of the significant predictors of PLCP. Researchers have been studying the relationship between achievement and attitude. Ma and Kishor (1997) analyzed 113 studies focused on the relationship between attitude and achievement. They noticed that the correlation between attitude and achievement was positive, but not significant. Contrary to this meta-analysis study result, Recber, Işıksal, and Koç (2018) found a significant relationship between attitude and achievement. Similar to the results of this study, studying with 168 higher education students, Akinola and Nosiru (2014) also found that students' attitudes had an impact on their programming success. The results of the studies conducted at the higher education level are similar to the results of the studies conducted at the primary education level. After surveying with primary school students, Lee et al. (2017) concluded that the students' attitudes towards programming were significantly related to their programming achievement. This result indicates that teachers of programming courses should consider students' attitudes towards programming to increase students' achievement in the course. In recent years, the number of programming languages and tools has increased, and they were shown to have an impact on attitude and achievement (Du, Wimmer & Rada, 2016). Prior programming experience has effects on students' attitudes and success in programming courses and attitudes towards programming. Tafliovich, Campbell, and Petersen (2013) suggested that the prior experience "affects students' expectations, work habits, attitude and confidence, and perceptions of self and peers" (p. 244). It was found that graphical programming languages as the first experience in programming has effects on students' performance (Chen, Haduong, Brennan, Sonnert, & Sadler, 2019). In addition, instructional strategies such as game-based learning environments have influenced students' attitudes toward programming (Goel & Kathuria, 2010). Hence, instructors can employ graphical programming languages, proper instructional strategies, select languages with a higher level of abstraction, and use software visualization tools to enhance students' attitudes towards programming.

Ramalingam and Wiedenbeck (1998) noted that self-efficacy is essential for personal motivation. Individuals with high self-efficacy can take on more challenging tasks and spend more effort in achieving these tasks. Similarly, students with high self-efficacy effort higher performance and sufficiency in activities and can achieve higher success in these activities (Sternberg & Williams, 2010). This study indicated that there was a positive and significant relationship between the perceived learning level of the students and their self-efficacy scores. Furthermore, CPSE was one of the significant predictors of PLCP and one of the factors affecting students' success in learning environments. According to this result, it can be said that students with higher CPSE are likely to have higher PLCP. There are studies supporting this result of the study in the literature. Gurer et al. (2019) indicated the positive correlation between PLCP and CPSE. Moreover, Cigdem (2017), Clinkenbeard (2017), Wiedenbeck et al. (2004) and Yılmaz (2013) concluded that students' computer self-efficacy is an important determinant of their success in introductory computer programming. The significant correlation between self-efficacy and learning in this study was also supported by the self-efficacy theory (Bandura, 1997). Students' practice, teachers' assistance, and students' value of computer programming influence students' efficacy of programming. Askar and Davenport (2009) stated that when faced with difficult tasks, learning achievement leads to an increase in learners' self-efficacy and success in future life. Therefore, teachers could lead students to more practicing and guide them. Additionally, they should adjust the difficulty of tasks based on the content of the course.

There are many studies in the literature showing that computer science is a male-dominated field, that women are not adequately represented in universities in computer science-related courses, and that only a small percentage of women choose computer science as a future career (Cheryan, Master & Meltzoff, 2015; Galpin, 2002). However, the gap between men and women is gradually decreasing in terms of access to education and technology (Ikolo & Okiy, 2012; Yılmaz, 2013). Therefore, it can be stated that female students could become as successful as male students when opportunities for education and technology are improved. According to the results of the study, gender was significantly correlated with PLCP and was a significant predictor of PLCP. In other words, in this study, female students think that they learn computer programming more than boys. This result is similar in some studies in the literature and contradicts with some others. Pillay and Jugoo (2005) stated that male students' computer programming achievement was higher than female students. Similarly, Pala and Mıhcı-Türker (2019) examined prospective teachers' views on programming and found that females found programming languages more difficult than male teachers. In some studies, it was stated that there is no significant relationship between computer programming success and gender (Byrne & Lyons 2001; McDowell, Werner, Bullock, & Fernald, 2003). Lau and Yuen (2009) stated that female students showed higher performance in computer programming than male students, but this difference was due to talent differences, and there was no statistically significant difference between female students and male students' computer programming performances. Contrary to these studies, Yılmaz (2013) concluded that the computer programming success of female students was significantly higher than that of male students. Likewise, Pioro (2004) stated that female students had higher success in computer programming than male students. It can be stated that these results presented in the literature differ according to the time of the study, the characteristics of the sample group and the cultural structure of the region where the research was conducted. Similarly, the results obtained in this study are thought to be due to the characteristics of the sample group. This conclusion emphasizes the importance of teachers' analysis of the target group. The analysis of the students in the classroom reveals the characteristics of learners, i.e., motivation, attitude, readiness. This supports teachers in designing teaching activities, instructional materials, learning environment, and selecting evaluation strategies. When a group of students with similar characteristics, i.e., gender, have lower performance and achievement, the teacher should put more effort into those students.

In this study, a positive and low-level correlation between the perceived learning levels of the students and their departments was found. However, department variable was not found to be a significant predictor of PLCP. This result implies that although the variation in the perceived learning of engineering students could be explained by being a computer engineering or other engineering department student, it is not a determinant of student's PLCP. The reason for this result may be related to the courses offered in the departments and the field of work they will work upon graduation. The number of courses on programming offered in the computer engineering department is higher than the other departments. The higher number of courses on programming and more in-depth content on computer science and programming may have led to higher PLCP of computer engineering students. In addition, students of the computer engineering department are potential computer scientists or will likely work in a profession related to information and communication technologies. It can be thought that computer engineering students consider programming as an important gain for their future professions and therefore have high motivation for computer programming. In this case, computer engineering students are expected to have a higher PLCP than other departments' students. Previous studies have indicated that computer engineering students' perceptions of programming self-efficacy (Altun & Mazman, 2012; Askar & Davenport, 2009), and their attitudes towards programming (Başer, 2013b; Korkmaz & Altun, 2013) are higher than those of other department students. Additionally, students of computer-related departments have higher attitudes towards programming than other departments' students (Gezgin & Adnan, 2016; Yılmaz, 2013). Therefore, these variables, which are positively related to perceived learning, may cause computer engineering students to have higher PLCP.

## 5. Conclusion and Future Works

The study contributes to the literature on engineering faculty students' learning on computer programming. Data collected from engineering students indicated that the PLCP of engineering students could be predicted using gender, attitude toward computer programming, and computer programming self-efficacy. The results of this study can inform the instructors of computer programming lessons. Teachers may use the findings of this study to understand better the role of different variables in students' learning of computer programming. As it consists of cognitive, affective, and psychomotor domains of learning, perceived learning is a strong indicator of learning outcomes. While designing and implementing the programming courses, to increase learning, gender, attitude toward computer programming, and computer programming self-efficacy can be handled together.

One of the limitations of this study is the population. 63.89% of the students were in their third and fourth grade. The freshman and sophomore students were not represented equally. Hence, in a further study, equally represented groups in terms of grade-level could be created while collecting data to make more concrete comparisons and conclusions. In the current study, to apply dummy coding on department variable, the departments of students were categorized as computer engineering and other departments. Therefore, the generalizability of the result about the relationship between PLCP and department is another limitation of this study, and this result should be considered carefully. In this study, variables used to predict programming success explain only 45.8% of programming achievement, while 54.2% cannot be explained. There are other variables that could affect students' perception of learning in programming such as measured learning (Gurer et al., 2019), satisfaction with the course (Lee et al., 2017), or first-experience with programming (Chen et al., 2019). A more comprehensive study can be done by adding variables such as measured learning, attitude towards the course, satisfaction, motivation. Multiple regression analysis was used to investigate the effect of different variables on programming achievement. In another study, structural equation modeling, which is an analysis that is running multiple regression models simultaneously, can be used to investigate the direct and indirect effects of different variables on the learning of programming.

## References

Akçay, A., & Çoklar, A. N. (2018) Bilişim teknolojileri ve yazılım dersi öğretmen adaylarının programlamaya ilişkin algılanan öz yeterliklerinin farklı değişkenler açısından incelenmesi. *Kastamonu Eğitim Dergisi, 26*(6), 2163-2176. doi:10.24106/kefdergi.2904

Akinola, O. S., & Nosiru, K. A. (2014). Factors influencing students' performance in computer programming: a fuzzy set operations approach. *International Journal of Advances in Engineering & Technology*, *7*(4), 1141–1149.

Alavi, M., Marakas, G. M., & Yoo, Y. (2002). A comparative study of distributed learning environments on learning outcomes. *Information Systems Research*, *13*(4), 404–415.

Alsadoon, E. A. (2013). *Factors influencing faculty to adopt web applications in their teaching.* Unpublished Ph.D. Thesis, Ohio University, OH.

Altun, A., & Mazman, S. G. (2012). Programlamaya ilişkin öz yeterlilik algısı ölçeğinin Türkçe formumun geçerlilik ve güvenirlik çalışması. *Eğitimde ve Psikolojide Ölçme ve Değerlendirme Dergisi*, *3*(2), 297–308.

Anastasiadou, S. D., & Karakos, A. S. (2011). The beliefs of electrical and computer engineering students' regarding computer programming. *The International Journal of Technology, Knowledge, and Society*, *7*(1), 37–52. DOI:10.18848/1832-3669/CGP/v07i01/56170

Askar, P., & Davenport, D. (2009). An investigation of factors related to self-efficacy for java programming among engineering students. *The Turkish Online Journal of Educational Technology*, *8*(1), 26–33.

Bandura, A. (1997). *Self-efficacy: The exercise of control*. New York: Freeman and Company.

Başer, M. (2013a). Attitude, gender, and achievement in computer programming. *Middle-East Journal of Scientific Research*, *14*(2), 248–255.

Başer, M. (2013b). Bilgisayar programlamaya karşı tutum ölçeği geliştirme çalışması. *The Journal of Academic Social Science Studies*, *6*(6), 199–215.

Byrne, P., & Lyons, G. (2001). The effect of student attributes on success in programming. *ACM SIGCSE Bulletin*, *33*(3), 49–52.

Cetin, I., & Andrews-Larson, C. (2016). Learning sorting algorithms through visualization construction. *Computer Science Education*, *26*(1), 27–43. https://doi.org/10.1080/08993408.2016.1160664

Cetin, I., & Ozden, M. Y. (2015). Development of computer programming attitude scale for university students. *Computer Applications in Engineering Education*, *23*(5), 667–672. https://doi.org/10.1002/cae.21639

Chen, C., Haduong, P., Brennan, K., Sonnert, G., & Sadler, P. (2019). The effects of first programming language on college students' computing attitude and achievement: a comparison of graphical and textual languages. *Computer Science Education, 29*(1), 23-48. https://doi.org/10.1080/08993408.2018.1547564

Cheryan, S., Master, A., & Meltzoff, A. N. (2015). Cultural stereotypes as gatekeepers: Increasing girls' interest in computer science and engineering by diversifying stereotypes. *Frontiers in psychology, 6*, 49. DOI:10.3389/fpsyg.2015.00049

Cigdem, H. (2017). How does self-regulation affect computer-programming achievement in a blended context? *Comtemporary Educational Technology, 6*(1), 19-37.

Clinkenbeard, D. A. (2017). *Factors that influence the success of male and female computer programming students in college*. Unpublished Doctoral Dissertation, Claremont Graduate University, California.

Cohen, J., Cohen, P., West, S. G., & Alken, L.S. (2003). *Applied multiple regression/correlation analysis for the behavioral sciences. 3rd Edition*. New York: Lawrence Erlbaum Associates.

Doube, W., & Lang, C. (2012). Gender and stereotypes in motivation to study computer programming for careers in multimedia. *Computer Science Education*, *22*(1), 63–78. https://doi.org/10.1080/08993408.2012.666038

Du, J., Wimmer, H., & Rada, R. (2016). "Hour of Code": Can it change students' attitudes toward programming? *Journal of Information Technology Education: Innovations in Practice, 15*, 52-73.

Ewell, P. (1994). *A Preliminary Study of the Feasibility and Utility for National Policy of Instructional" Good Practice" Indicators in Undergraduate Education*. Washington, DC: National Center for Education Statistics (ED).

Field, A. (2009). *Discovering Statistics with SPSS, 3rd Edition.* California: Sage Publications.

Fraenkel, J. R., Wallen, N. E., & Hyun, H. H. (2015). *How to design and evaluate research in education* (9th ed.). New York: McGraw-Hill Education.

Galpin, V. (2002). Women in computing around the world. *ACM SIGCSE Bulletin, 3*4(2), 94-100.

Gardiner, B. (2017). Adding coding to the curriculum. Retrieved 09.20.2018, from https://www.nytimes.com/2014/03/24/world/europe/adding-coding-to-the-curriculum.html

Gezgin, D. M., & Adnan, M. (2016). Makine mühendisliği ve ekonometri öğrencilerinin programlamaya ilişkin öz yeterlik algılarının incelenmesi. *Ahi Evran Üniversitesi Kırşehir Eğitim Fakültesi Dergisi (KEFAD)*, *17*(2), 509–525.

Goel, S., & Kathuria, V. (2010). A novel approach for collaborative pair programming. *Journal of Information Technology Education: Research, 9*(1), 183-196.

Gurer, M. D., Cetin, I., & Top, E. (2019). Factors affecting students' attitudes toward computer programming. *Informatics in Education*, *18*(2), 281-296.

Hodge, B.K., & Steele, W.G. (2002). A survey of computational paradigms in undergraduate mechanical engineering education. *Journal of Engineering Education*, *91*, 415–417. https://doi.org/10.1002/j.2168-9830.2002.tb00726.x

Hongwarittorrn, N., & Krairit, D. (2010). Effects of program visualization (Jeliot3) on students' performance and attitudes towards java programming. *The spring 8th International Conference on Computing,* *Communication and Control Technologies,* pp. 6–9.

Ikolo, V. E., & Okiy, R. B. (2012). Gender differences in computer literacy among clinical medical students in selected southern Nigerian Universities. *Library Philosophy & Practice (e-journal)*, *5*, 34–41.

International Society for Technology in Education (ISTE) (2016). *ISTE standards for students*. Retrieved at 08.28.2019 from www.iste.org/standards.

Kılıçarslan-Cansu, S., & Cansu, F. K. (2019). An overview of computational thinking. *International Journal of Computer Science Education in Schools, 3*(1), 17-30.

Korkmaz, O. (2016). The effects of Scratch-based game activities on students' attitudes, self-efficacy, and academic achievement. *International Journal of Modern Education and Computer Science*, *8*(1), 16–23. DOI: 10.5815/ijmecs.2016.01.03

Korkmaz, Ö., & Altun, H. (2013). Mühendislik ve BÖTE öğrencilerinin bilgisayar programlama öğrenmeye dönük tutumları. *International Journal of Social Science*, *6*(2), 1169–1185. DOI: http://dx.doi.org/10.9761/JASSS_690

Lau, W.W.F., & Yuen, A.H.K. (2009). Exploring the effects of gender and learning styles on computer programming performance: Implications for programming pedagogy. *British Journal of Educational Technology*, *40*(4), 696–712. https://doi.org/10.1111/j.1467-8535.2008.00847.x

Lee, S., Kim, J., & Lee, W. (2017). Analysis of factors affecting achievement in maker programming education in the age of wireless communication. *Wireless Personal Communications*, *93*(1), 187–209. DOI:10.1007/s11277-016-3450-2

Ma, X., & Kishor, N. (1997). Assessing the relationship between attitude toward mathematics and achievement in mathematics: A meta-analysis. *Journal for Research in Mathematics Education*, *28*(1), 26–47.

McDowell, C., Werner, L., Bullock, H. E. & Fernald, J. (2003). The impact of pair programming on student performance, perception, and persistence. *25th International Conference on Software Engineering, 2003. Proceedings*, *6*, 602–607.

Metcalfe, J. (2009). Metacognitive judgments and control of study. *Current Directions in Psychological Science*, *18*(3), 159–163.

Naraghi, M. H. N., & Bahman, L. (2001). An effective approach for teaching computerprogramming to freshman engineering students. *Proceedings of the 2001 American Society for Engineering Education Annual Conference & Exposition*.

Özyurt, Ö., & Özyurt, H. (2015). A study for determining computer programming students' attitudes towards programing and their programming self-efficacy. *Journal of Theory and Practice in Education*, *11*(1), 51–67.

Pala, F. K., & Mıhcı-Türker, P. (2019). Öğretmen adaylarının programlama eğitimine yönelik görüşleri. *Kuramsal Eğitimbilim Dergisi*, *12*(1), 116–134. https://doi.org/10.30831/akukeg.399921

Pillay, N., & Jugoo, V. R. (2005). An investigation into student characteristics affecting novice programming performance. *ACM SIGCSE Bulletin*, *37*(4), 107.

Pioro, B. T. (2004). Performance in an introductory computer programming course as a predictor of future success for engineering and computer science majors. *International Conference on Engineering Education, Gainesville, FL.*

Ramalingam, V., & Wiedenbeck, S. (1998). Development and validation of scores on a computer programming self-efficacy scale and group analyses of novice programmer self-efficacy. *Journal of Educational Computing Research*, *19*(4), 367–381.

Recber, S., Işıksal, M., & Koç, Y. (2018). Investigating self-efficacy, anxiety, attitudes and mathematics achievement regarding gender and school type. *Anales De Psicología/Annals of Psychology, 34(1)*, 41-51. https://doi.org/10.6018/analesps.34.1.229571

Rovai, A. P., & Barnum, K. T. (2003). On-line course effectiveness: an analysis of student interactions and perceptions of learning. *Journal of Distance Education*, *18*(1), 57–73.

Rovai, A. P., Wighting, M. J., Baker, J. D. & Grooms, L. D. (2009). Development of an instrument to measure perceived cognitive, affective, and psychomotor learning in traditional and virtual classroom higher education settings. *The Internet and Higher Education*, *12*(1), 7–13. DOI:10.1016/J.IHEDUC.2008.10.002

Schunk, D.H. (1989). Self-efficacy and achievement behaviors. *Educational Psychology Review, 1*(3), 173-208.

https://doi.org/10.1007/BF01320134

Singh, K., Allen, K. R., Scheckler, R., & Darlington, L. (2007). Women in computer-related majors: A critical synthesis of research and theory from 1994 to 2005. *Review of Educational Research*, *77*(4), 500–533. https://doi.org/10.3102/0034654307309919

Sternberg, R. J., & Williams, W. M. (2010). *Educational psychology* (2nd ed.). Upper Saddle River, N.J.: Merrill.

Tafliovich, A., Campbell, J., & Petersen, A. (2013). A student perspective on prior experience in CS1. *Proceeding of the 44th ACM technical symposium on Computer science education*, Denver, CO.

Top, E., Yukselturk, E., & Cakir. R. (2011). Gender and Web 2.0 technology awareness among ICT teachers. *British Journal of Educational Technology, 42*(5), E106-E109. https://doi.org/10.1111/j.1467-8535.2011.01208.x

Top, E., Yukselturk, E., & Inan, F. A. (2010). Reconsidering usage of blogging in preservice teacher education courses. *The Internet and Higher Education*, *13*(4), 214–217. https://doi.org/10.1016/j.iheduc.2010.05.003

Ülkü, E. E., Doğan, B., Demir, Ö. & Yıldız, K. (2017). An analysis of the self efficacy about computer programming of the electrical-electronics and textile engineering students in technology faculty. *Journal of Educational and Instructional Studies in The World, 7*(4), 79-84.

Wiedenbeck, S., Fix, V., & Scholtz, J. (1993). Characteristics of the mental representations of novice and expert programmers: An empirical study. *International Journal of Man-Machine Studies*, *39*(5), 793–812.

Wiedenbeck, S., LaBelle, D., & Kain, V. N. R. (2004). Factors affecting course outcomes in introductory programming. In *Proceedings of the Sixteenth Annual Workshop of the Psychology of Programming Interest Group (PPIG '04),* 97-110.

Yılmaz, F. (2013). *Meslekyüksek Okulu Öğrencilerinin Programlama Başarısını Etkileyen Faktörlerin İncelenmesi*. Unpublished master's thesis, Gazi University Institution of Educational Sciences, Ankara.

Zyda, M. (2009). Computer science in the conceptual age. *Commun. ACM*, *52*(12), 66–72. DOI: 10.1145/1610252.1610272

*International Journal of Computer Science Education in Schools, April 2020, Vol. 3, No. 4*
*ISSN 2513-8359*

# The Relationship between Executive Functions and Computational Thinking

**Judy Robertson**[1]
**Stuart Gray**[2]
**Toye Martin**[1]
**Josephine Booth**[1]

1 University of Edinburgh
2 University of Bristol

DOI: 10.21585/ijcses.v3i4.76

**Abstract**

We argue that understanding the cognitive foundations of computational thinking will assist educators to improve children's learning in computing. We explain the conceptual relationship between executive functions and aspects of computational thinking. We present initial empirical data from 23 eleven year old learners which investigates the correlation between assessments of programming and debugging in the visual language Scratch and scores from the BRIEF2 assessment of executive functions. The initial data shows moderate to large correlations between assessments of debugging and programming with the BRIEF2 teachers' rating of executive function as manifested in classroom behaviour. Case studies from the empirical data are used to qualitatively illustrate how executive functions relate to a game making task. We discuss the implications of these findings for educators, and present suggestions for future work.

**Keywords:** computational thinking, executive functions, primary school

## 1. Introduction

The recent Royal Society report on computing education in UK schools reviewed the landscape after major curricular reform in which computing lessons became a requirement for learners aged 5 and older in England and Wales (Royal Society, 2017). While it welcomes the changes (which it was partly responsible for instigating) and notes that there are pockets of excellence, it identifies that computing education is still "patchy and fragile" (Royal Society, 2017, p. 6). The report demonstrates the curriculum changes are not enough; they must be supported by high quality teacher education and computing education research. It proposes that a research agenda in the UK should focus on the questions: "What is the most effective, best-evidenced curriculum framework for computing? …Which specific instructional techniques and teaching strategies are most effective for raising attainment in computing?"(Royal Society, 2017, p. 95). These are also open questions within the international research community. In order to answer these questions, however, we need to further develop our understanding of the cognitive and psychological skills which underpin different aspects of computational thinking, and how these develop throughout childhood. This paper focuses on the potential link between computational thinking and underlying executive functions.

Much work has been done on defining computational thinking (also referred to as CT) and its component skills. In this paper, we use the Royal Society's clear and succinct definition of computational thinking: " the process of recognising aspects of computation in the world that surrounds us, and applying tools and techniques from Computer Science to understand and reason about both natural and artificial systems and processes" (The Royal

Society, 2012, p. 12). A review of computational thinking research identified the following core computational thinking elements: abstractions and pattern generalizations (including models and simulations); systematic processing of information; symbol systems and representations; algorithmic notions of flow of control; structured problem decomposition (modularizing); iterative, recursive, and parallel thinking; conditional logic; efficiency and performance constraints ; debugging and systematic error detection (Grover & Pea, 2013). In addition, "programming is not only a fundamental skill of CS and a key tool for supporting the cognitive tasks involved in CT but a demonstration of computational competencies as well." (Grover & Pea, 2013, p. 40)

There has been debate about the extent to which computational thinking can be distinguished from the other sorts of thinking which children learn at school. Although historically some doubt has been cast on the nature of the relationship between programming and problem solving (Palumbo, 1990), recent evidence synthesis reveals that learning to program can improve scores on other measures of problem solving. A meta-analysis of studies which explore the transfer of programming skills to general problem solving found an overall transfer effect of g =0.49, with a transfer to mathematical reasoning of g=0.57 (Scherer, Siddiq, & Sanches Viveros, 2018). It is also likely that programming and other computational thinking abilities are enabled by well-researched lower level psychological processes. Grover and Pea make the case that while there might be overlap between computational thinking and other STEM problem solving approaches, it was recognisably and crucially absent from previous curricula (Grover & Pea, 2013). Recent empirical work supports this; while computational thinking is predicted by other cognitive abilities, it appears to some extent to be an independent construct. In a study of 1251 Spanish school students, Román-González and colleagues investigated the relationship between computational thinking (as measured by their CTt instrument) and other cognitive abilities as measured by the Primary Mental Ability and RP30[1] Resolucion-de-Problemas problem solving standardised psychological tests (Román-González, Pérez-González, & Jiménez-Fernández, 2017). They found a high correlation (r=0.67) between general problem solving ability and computational thinking scores. In a regression model, spatial ability and logical reasoning as measured by RP30 problem solving tasks were significant predictors of the CTt scores, explaining 27% of the variance. The authors interpret the high proportion of unexplained variance to suggest a "certain independence of CT as a psychological construct, distinct from the traditional aptitudes" (Román-González et al., 2017, p. 9). They recommend that further research should relate computational thinking with other cognitive abilities including working memory and other executive functions (also referred to as EF).

This paper pursues this line of research by exploring the relationship between two important aspects of computational thinking (programming and debugging) and executive functions. It begins with an explanation of executive functions, the role they play in academic success, and the reasons why they are likely to be related to computational thinking. This is followed by an account of how executive functions (as measured by the Behavior Rating Inventory of Executive Functioning-2 (BRIEF2) instrument) may map to creative programming and debugging. Having made the case that specific EF skills are likely to underpin these aspects of computational thinking, the paper then reports on an empirical study to investigate this issue. Data on 23 eleven year-old learners' creative programming and debugging performance, and EF abilities was gathered. Results of a correlation analysis do indeed support the case that EFs are related to computational thinking. Case studies from the empirical data are used to qualitatively illustrate how executive functions relate to a game making task. The paper concludes with some recommendations for practitioners about how this might affect classroom decision making, and suggestions of future research work.

## 2. Literature Review

### *2.1 Executive Functions and the Link to Computational Thinking*

Executive functions (EF) is an umbrella term for higher order cognitive functions linked with the frontal lobes of the human brain (Aron, Robbins, & Poldrack, 2004) and include abilities such as inhibiting impulsive responses, the ability to hold and simultaneously manipulate information in mind (known as working memory), attention shifting (or cognitive flexibility), planning and risk taking (Diamond, 2013; Miyake et al., 2000). EF serve as general purpose control mechanisms that help modulate human cognition (Miyake et al., 2000), underpin self-control (Denckla, 1996; Pennington & Ozonoff, 1996) and are commonly implicated in problem-solving or goal-directed-behaviour (Luria, 1966). These functions mature at different rates through childhood and into adolescence (Dolan & Molen, 2006).

Executive functioning and educational attainment in the primary school age-range has also been linked with

[1] http://web.teaediciones.com/RP30--Resolucion-de-Problemas---Problem-Solving.aspx

metacognition (MC) (Bryce, Whitebread, & Szűcs, 2015), with some studies reporting a link between MC and problem solving in computer science learning specifically (Allsop, 2019; Parham, Gugerty, & Stevenson, 2010). Parham and colleagues' analysis of a think-aloud study with eleven college-level programmers indicates that the meta-cognitive strategies of checking/comparing code and stating/revisiting goals were commonly used mental processes. In a study of a class of 11 year learners, Allsop noted the use of metacognitive practices to control and regulate programming activities including planning, monitoring and evaluation (Allsop, 2019). Whilst related constructs in children's thought and action (Lyons & Zelazo, 2011), EF and metacognition have been based in different research and theoretical traditions in the psychological literature (Roebers, 2017a). In this study, for the sake of clarity, we have chosen to focus on EF.

We believe that the link between EF and CT is worth exploring for two reasons: 1) EF is a predictor of academic success in general, including in the development of mathematical skills and science learning (Cragg & Gilmore, 2014) so it is reasonable to assume that they are also required in CT; and 2) conceptual analysis of the processes involved in programming and debugging predict that cognitive regulation aspects of EF are required. However, further empirical evidence of the relationship is required; this paper makes an initial contribution by providing the results of an exploratory study in a primary school classroom.

Programming and debugging are not the only components of computational thinking; it is likely that EFs are also implicated in other aspects. We have chosen to start with these components because they are commonly taught in classrooms in the UK and other countries internationally, and there are extensive online teaching materials to support them. Other aspects of computational thinking should be the subject of future research.

### *2.2 EF as a Predictor of Academic Success*

Executive functions are implicated in a wide range of areas of academic learning and attainment. For example, in reading (Altemeier, Abbott, & Berninger, 2008), maths (Gilmore et al., 2013) and science (St Clair-Thompson & Gathercole, 2006). They are also predictive of school achievement more generally (Bull & Scerif, 2010; McLean & Hitch, 1999; St Clair-Thompson & Gathercole, 2006; Titz & Karbach, 2014) as well as university achievement (Knouse, Feldman, & Blevins, 2014), and job success (Daly, Delaney, Egan, & Baumeister, 2015). Some interpret these findings to suggest a domain-general relationship between EF and school attainment (Best, Miller, & Naglieri, 2011), a growing number of studies highlight a particularly important role for EF in relation to learning in STEM subjects (St Clair-Thompson & Gathercole, 2006; Van der Ven, Kroesbergen, Boom, & Leseman, 2012), although to our knowledge, no previous studies have considered the relationship between EF and CT.

Neuroscience and education research indicates that executive skills play a critical role in developing mathematical proficiency, particularly updating and manipulating working memory, inhibition and shifting (Cragg & Gilmore, 2014). Given Weintrop and colleagues' detailed argument exploring the reciprocal relationship between computational thinking and maths and science learning (Weintrop et al., 2016), and the meta-analysis results which indicate programming improves mathematical test scores (Scherer et al., 2018), there is good reason to investigate the relationship between computational thinking and executive skills. The executive skills which support the development of maths proficiency are also likely to play a role in developing computational thinking. Indeed, empirical evidence from design based research with middle school children confirms that the outcomes of a course in computational thinking were predicted by maths performance ( Grover, Pea, & Cooper, 2015).

### *2.3 Conceptual Analysis Of Efs Involved In Programming and Debugging*

#### 2.3.1 Why Efs Are Required For Creative Programming Tasks

The playful constructionist approach advocated by the Scratch creators, based on Papert's intellectual legacy, encourages creativity and self-directed exploration. Following the footsteps of Logo, Scratch was designed as a constructionist environment to support a spiral of creativity, in which learners "imagine what they want to do, create a project based on their ideas, play with their creations, share their ideas and creations with others, and reflect on their experiences—all of which leads them to imagine new ideas and new projects"(Resnick, 2007, p. 18) . Brennan and Resnick describe design in the creative programming context as "an adaptive process, one in which the plan might change in response to approaching the solution in small steps". They describe a process of "iterative cycles of imagining and building, developing a little bit, trying it out then developing further, based on their experiences and new ideas."(Brennan & Resnick, 2012, p. 7).

The merits of pure discovery learning (for example in Papert's constructionist work with Logo) have been questioned (Mayer, 2004). Previous researchers have argued that "learners often struggle with algorithmic concepts, especially if they are left to tinker in programming environments, or if they are not taught these concepts using appropriately supportive pedagogies." (Grover et al., 2015, p. 205). That is, unstructured programming tasks by themselves may not improve computational thinking. Grover et al. recommend minimally guided discovery learning for computational thinking courses for middle school learners (Grover et al., 2015), including approaches such as scaffolding, cognitive apprenticeships, code reading and tracing, and modelling the process of decomposition. They aim to enable children to build on computational concepts which they have been taught, in a way which fosters creativity and ownership.

The complexity of creative programming tasks can greatly challenge novices. The individual computational thinking skills can be difficult to acquire in themselves, but the higher order executive function skills place additional load on the learner, particularly in terms of planning and self-monitoring. The learner must decide what to make, how to make it and be able to monitor her own progress in reaching her goals. Depending on the stage of the learner, it may be the case that the requisite planning and monitoring executive functions are still developing. In addition, the task places a load on working memory because the programmer must hold in mind the end goal, and steps needed to achieve the goal. Inhibition is also required to avoid distractions from the goal.

Papert (Papert, 1991) and later Resnick and Brennan (Brennan & Resnick, 2012) favour a less top down approach to planning in which the programmer is: "guided by the work as it proceeds rather than staying with the pre-established plan" (Papert, 1991, p. 3). A learner working in this way would still task-monitor periodically to evaluate whether the current code produces a desirable output, and if not, decide what changes are needed and map out the sub-steps to get from the current state to the desired state. While open ended creative tasks can be fun when appropriately challenging, it may be overwhelming and frustrating for some learners unless they are adequately supported. For learners with developing EF skills, it may be difficult – and demoralising – to engage with such tasks. With appropriate support with planning and monitoring, however, working on motivating programming tasks may be one approach to EF skills development. It is possible that the requisite EF and programming skills can be further developed alongside each other in creative projects, if the learner has previously had support to develop both programming skills and planning and monitoring skills in other contexts.

### *2.4 Why EFs Are Required For Debugging Tasks*

Resnick and Brennan describe computational practices relating to solving and anticipating problems. An interviewee described her debugging activities as "identify the source of the problem, read through the scripts, experiment with scripts, try writing scripts again, find example scripts that work, tell or ask someone else, take a break…" (Brennan & Resnick, 2012, p. 7). Rich and colleagues present a helpful literature syntheis and learning trajectory of debugging for children (Rich, Andrew Binkowski, Strickland, & Franklin, 2019). They identify strategies documented in the literature for finding and fixing errors including: hypothesising and testing theories about the cause of a problem and deciding how to change a program when it does not produce the intended results. These behaviours are likely to rely on underpinning executive function capacities such as working memory and cognitive flexibility ("the ability to shift between response sets, learn from mistakes, devise alternative strategies, divide attention, and process multiple sources of information concurrently" (Anderson, 2002, p. 74)). Rich et al. observe that emotional regulation and the ability to preserve in the face of failure is a requirement for successful debugging; emotional regulation is an aspect of executive function which is assessed by the emotional control subscale within the BRIEF2 instrument which we used in this study. In addition, Rich and colleagues note that at the end of the learning trajectory, learners become aware that debugging techniques can be chosen strategically. The ability to evaluate and slect the best strategy for a task requires well developed executive function capacities (Roebers, 2017b).

In order to systematically detect errors in code, the learner must have the ability to understand the decription of the incorrect program behaviour and why it is different from the required behaviour, and be able to develop and follow a plan of detecting, fixing and testing which places high demands on working memory, as well as the ability to switch between tasks and switch back and forth between different representations. This last point is particularly salient in a visual language like Scratch where the user must look at the visual behaviour of a sprite on screen, compare it to a mental representation of what the ideal behavior would look like, and then switch to a different visual representation of code blocks in order to fix the problem.

The programmer must think of possible reasons why the program is not working, prioritise which is most likely to begin with, pinpoint where the error would occur in the code, identify whether it is actually present in the code and if so, fix it. If that particular error was not present, or if it was present but fixing it did not result in the target code behaviour, the programmer must move on to consider another possible reason for the flaw. Beginner programmers are disadvantaged because they do not have the experience to quickly identify or prioritise possible errors. From this point of view, the debugging exercises used in this study were designed to start with examples where the learner need only solve one bug at a time and progress to finding and correcting multiple bugs.

## 3. A Classroom Study

We conducted an initial study with a class of 11-year-old children to begin the process of testing the theorised link between EF and CT with empirical data. The aims of the study were to quantiatively explore the relationship between EF and CT, specifically: the overall relationship between creative programming and debugging with aspects of executive function, as measured by BRIEF2. A further goal for this study was to use case studies from the empirical data to qualitatively illustrate the relationship between EF and CT.

### *3.1 Participants*

This study involved twenty-five children (16 boys, 9 girls) aged 11-12 years from the same Primary 7 class at a Scottish Primary School. All children had experience of programming using Scratch as part of their computing curriculum work. The group were recruited by their class teacher and received written information sheets and consent forms (in age appropriate language) two weeks before the study to be read, signed, and returned before they could take part in the sessions. The ethical procedures were approved by the [*blank for review*] ethics committee. Of the twenty-five children who gave their consent to be part of the study, we were able to collect data on all EF/CT measures for twenty-three.

### *3.2 Data Collection*

The following data was collected in this study.

*Executive functions*

The Behavioural Rating Inventory of Executive Function (BRIEF2) is a rating scale used to assess everyday behaviours associated with executive functions at home and school. It is considered to be an ecologically valid method of assessing the extent to which individuals are able to successfully pursue their own goals in complex every-day problem solving tasks (Toplak, West, & Stanovich, 2013) . It is used for clinical assessment of children for whom there may be concerns about self- regulation (e.g those with autistic spectrum disorders, attention disorders, depression and other conditions). The BRIEF2 is a questionnaire completed by teachers about individuals' behaviour and emotional regulation, aspects of EF which are also important for classroom learning. A recent review of BRIEF2 considered it to be a theoretically and psychometrically sound measure of executive functioning for children and adolescents (Dodzik, 2017), with internal consistency for the teachers form in the range of alpha coefficient =0.88 to 0.98, and a test/re-test reliability of 0.82.

In this study, the BRIEF2 teacher rating scale is used to assess behaviours which might impact on typically developing children's ability to complete complex creative programming and debugging tasks. BRIEF2 has three indices: behavioural regulation (consisting of inhibit and self-monitor scales), emotion regulation (consisting of shift and emotional control scales) and cognitive regulation (consisting of initiate, working memory, plan/organise, task-monitor, and organisation of materials scales).

The class teacher filled in a 63 item scale for each pupil, indicating whether the statement is true of the child *never* (scored as 1), *sometimes* (scored as 2), or *often* (scored as 3). The raw score was then converted to a T-score which is normalised for age and gender according to. T-scores range between 36 and 90 for 11-13 year old girls and between 37 and 88 for 11-13 year old boys (Gioia, Isquith, Guy, & Kenworthy, 2015). Higher scores indicate higher level of difficulty in a specific domain of executive function. The Global Executive Composite score is reported here as it is considered as a useful summary measure.

*Creative programming task*: We used an automatic assessment of aspects of computational thinking as manifested in the source code of a Scratch program collected from the participants (flow control, data representation, abstraction, user interaction, synchronisation, parallelism and logic). The code was analysed using Dr Scratch, software which performs static analysis of Scratch source code (Moreno-León & Robles, 2015). Assessment from Dr Scratch has shown to be consistent with other software metrics of code complexity (Moreno-Leon, Robles, & Roman-Gonzalez, 2016) and correlate strongly with the assessments of expert human evaluators (r=0.82) (Moreno-León, Harteveld, Román-González, & Robles, 2017). Overall scores range between 0 and 21, with higher scores indicating higher proficiency.

*Debugging task*: A set of 7 custom Scratch debugging tasks were developed by the authors, based on the Debug It! Exercises available on Scratch Studio[2]. The seven exercises required debugging examples involving conditionals, fixed loop, variables, conditional loop, parallelism (simple problems), and two integrated examples which brought together several of these concepts (complex problems). Participants were given a specification of what an example Scratch program should do, a description of the buggy behaviour of the code when it runs, and the code itself. They were then asked to locate and fix the error. Each simple problem was scored with 2 points for a complete solution, and one point for a partial solution (where a clue had been given), while the two complex problems which had a maximum score of 6 points because there were multiple bugs. The maximum overall score was 22, with higher scores indicating higher proficiency. The tasks were scored by the first and second authors; discrepancies in the scores were resolved through discussion.

### *3.3 Procedure*

Data collection sessions were undertaken by the second author and a research assistant at the Primary School. These staff members both had up-to-date certification to work with children.

#### 3.3.1 Scratch Creative Programming Session

The creative programming sessions were hosted in the children's classroom using school Windows laptops and facilitated by the second author. He first explained to the children that they were being asked to use Scratch online to individually create a program of their choosing within a 60-minute time limit and gave the children an opportunity to ask questions about the exercise. The children had a written document of a description of the task and instructions to refer to and could ask for clarification and assistance if required.

#### 3.3.2 Scratch Program Debugging Session

The debugging session was again facilitated by the second author but the research assistant was also present to help support the children. The session began with the second author explaining to the children that they were being asked to fix a series of 7 broken Scratch programs within a 60-minute time limit. The children were asked to work alone without discussion.

The children were given a worksheet with a description of each problem, stating what the program should ideally look like (the specification) and the problems with the current version of the program (see supplementary materials). The children also had access to videos of the ideal of each program and the problematic version.

## 4. Results

### *4.1 Descriptive Statistics*

The descriptive statistics indicate that this class of children had room for improvement in debugging (with an average just slightly over half marks) and creative programming (with a mean under half marks). Note that the number of participants N varies due to student absences on different data collection days.

Table 5. Descriptive Statistics for Overall Measures

| | **N** | **Mean** | **Standard deviation** |
|---|---|---|---|
| **Debugging score** | 22 | 7.5 | 4.13 |
| **Creative programming score** | 23 | 9.35 | 4.53 |
| **BRIEF2 T-score** | 25 | 51.12 | 15.55 |

[2] https://scratch.mit.edu/projects/10437439/

The pearson correlation between the Creative Programming task score and the Debugging score is r=0.6 (95% CI 0.26, 0.82]). This can be interpreted as a large correlation between these two measurements of computational thinking (Cohen, 1992).

### *4.2 Findings*

As shown in Table 3, programs which have higher Dr Scratch scores are produced by children who have lower BRIEF2 scores (i.e. those who have more mature executive functioning): BRIEF2 scores explain 60% of the variance in Dr Scratch scores. Similarly, learners with a greater level of debugging skills have better developed EF skills, explaining 40% of the variance in debugging scores. Focusing on how the Scratch score relates to the BRIEF2 sub-scales, the Behaviour Regulation Index (BRI) correlates with r=-0.69, the Emotional Regulation Index (ERI) correlates with r=-0.57, and the Cognitive Regulation Index (CRI) correlates with r = -0.55. Given that the BRI seems to have the strongest relationship with the Scratch score, it was worth examining the relationship of the further subscales within it: the Inhibit subscale is r=-0.7, and the Self-monitor subscale is r=-0.67. The definition of Inhibit from the BRIEF2 manual is "the inhibit scale assesses inhibitory control (i.e. the ability to inhibit, resist, or not act on impulse), including the ability to stop one's behaviour at the appropriate time." (Dodzik, 2017, p. 33). The definition of Self-monitor is "the self-monitor scale assesses awareness of the impact of one's own behaviour on other people and outcomes". It includes "awareness of one's own effectiveness in problem solving and the ability to monitor important outcomes" (Dodzik, 2017, p. 34).

Table 6. Relationships between Measures of CT and "Reflective" Aspects of EF

| **Measures** | **Pearson correlation r** | **95% CI** |
|---|---|---|
| **Creative Programming total and BRIEF2 global composite** | -0.6 | [-0.8, -0.25] |
| **Debugging total and BRIEF2 global composite** | -0.4 | [-0.70, -0.02] |
| **Creative Programming and BRIEF2 behavioural regulation index (BRI)** | -0.69 | [-0.86, -0.40] |
| **Creative Programming and BRIEF2 emotional regulation index (ERI)** | -0.56 | [-0.79, -0.19] |
| **Creative Programming and BRIEF2 cogntivie regulation index (CRI)** | -0.55 | [-0.79,-0.18] |
| **Creative Programming and BRIEF2 BRI Inhibit** | 0.7 | [-0.86, -0.40] |
| **Creative Programming and BRIEF2 BRI self-monitor** | -0.67 | [-0.85, -0.36] |

### *4.3 Qualitative Illustrations of How EF Relates to Programming Tasks*

It is instructive to examine the games produced by learners with different EF profiles in order to identify sorts of help which teachers could provide when supporting similar tasks in the future (see Table 7 for a summary of their numerical scores).

Table 7. Summary of Case Study Learners' Scores

| **Participant number** | **Sex** | **Age** | **BRIEF2 Total score** | **Creative programming score** | **Debugging score** |
|---|---|---|---|---|---|
| P18 | M | 11 | 164 | 1 | 7 |
| P9 | M | 11 | 160 | N/A[3] | N/A |
| P5 | M | 11 | 83 | 0 | 3 |
| P1 | F | 12 | 69 | 14 | 12 |
| P11 | F | 11 | 63 | 8 | 5 |
| P17 | F | 11 | 62 | 12 | 10 |

[3] Scratch analyser crashes when this program is run; no score is output.

P18 is an eleven year old boy with one of the highest Behavioural Regulation Index scores in the class, scoring highly on both the inhibit and self-monitor scales indicating serious issues with these areas of functioning. Indeed, P18 has extremely elevated EF profiles across all BRIEF2 subscales. This learner's Scratch file has no code. It is hand-drawn stick man sprite and some vertical black lines on a backdrop. It is possible that the black lines were intended to form a maze like some of the other children's. The game does not convey personal interests or show exploratory behaviour within the tool like some of the other games with very little functionality. This learner has an elevated score in Initiate, suggesting that he finds it difficult to get started on a task. P18 scored 7 out of 21 on the debugging test, which illustrates that he has some basic understanding of Scratch constructs but he was unable to put them into practice. He was unable to marshall his efforts to produce a plan for a meaningful program in an open ended creative task. This learner would benefit from being given a specific task which would consolidate his knowledge of the Scratch concepts which he has been taught.

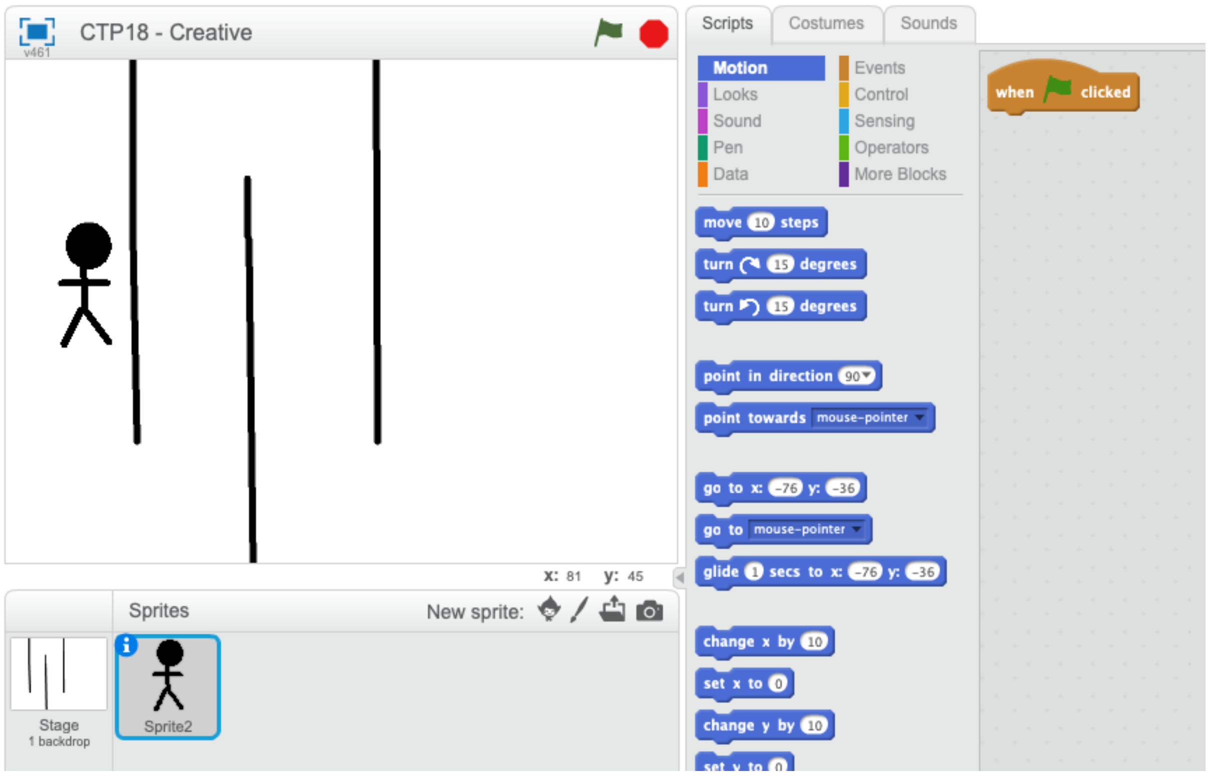
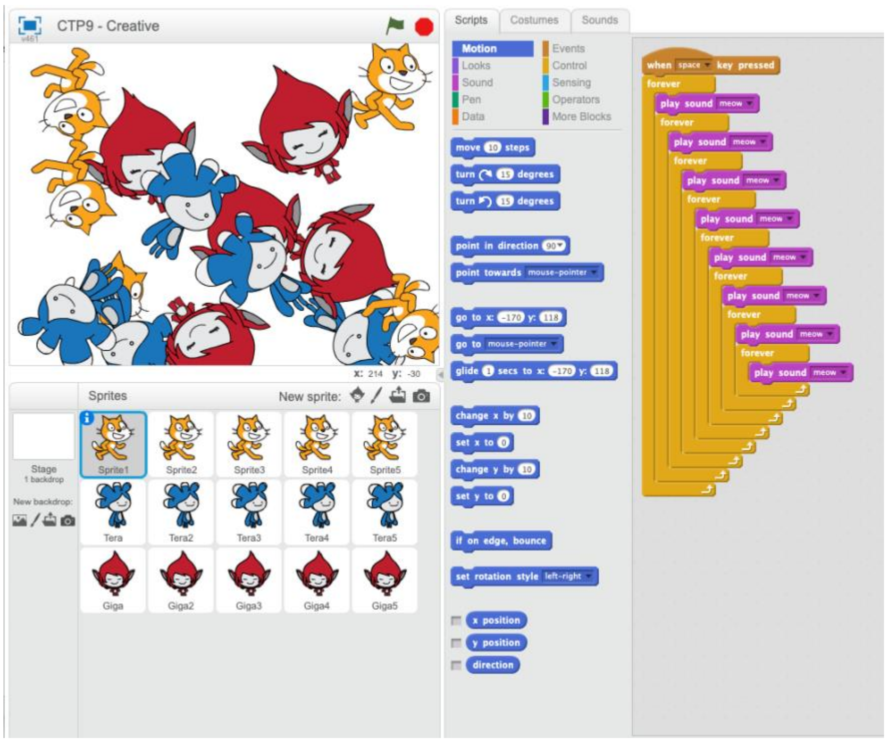


Figure 3. A Screen Shot of P18's Game

Figure 4 shows a screen shot of P9's game. P9 is also an eleven year old boy with an elevated EF score across all the subscales and was joint with P18 in having a particularly high score for the self-monitoring scale again, indicating difficulties across numerous areas of EF. In this game, there a multiple copies of each sprite with the same code copied between sprites. When the code runs, the sprites rapidly rotate and make a noise. The overall effect is strikingly colourful and overwhelming. There is quite a lot of code, but it is repeated sequences of animation instructions and loops which do not serve a clear purpose. There are multiple unnecessary nested loop constructs which suggest that the learner does not understand that only one forever block would have caused the cat sprite to repeatedly miaow. The learner does not appear to have been following a plan to create a particular interactive program, but rather gives the impression of exploring the Scratch interface to produce an entertaining visual result. He was absent on the day of the debugging session so there is no additional information about his Scratch knowledge. While children often find it fun to experiment with visual effects in Scratch when they initially encounter it, this learner would benefit from support in developing a specific goal and identifying how this could be accomplished in Scratch.



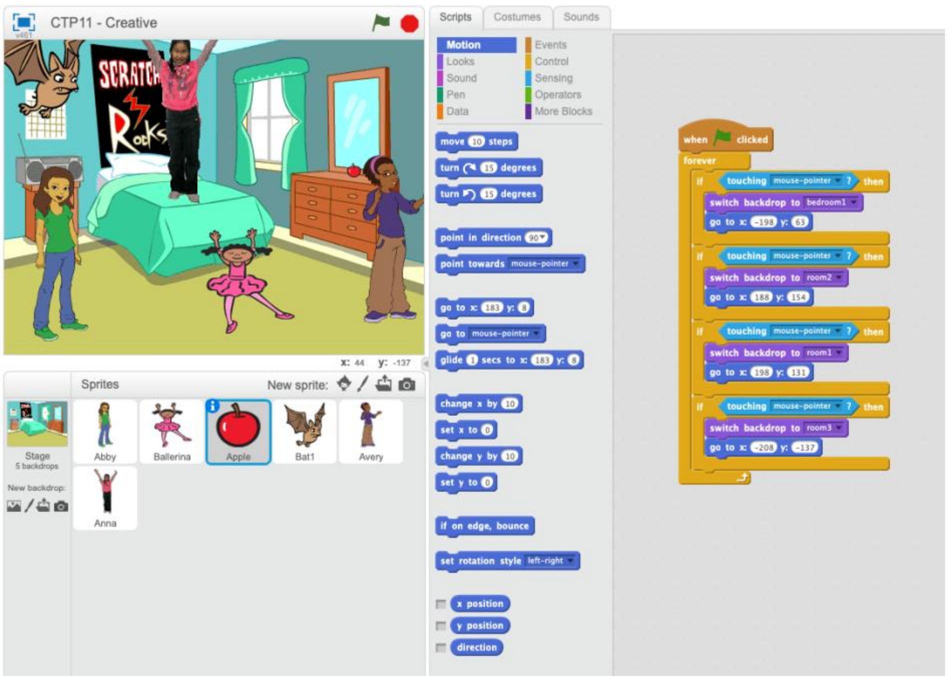
```
when space key pressed
forever
  play sound meow
  forever
    play sound meow
    forever
      play sound meow
      forever
        play sound meow
        forever
          play sound meow
          forever
            play sound meow
            forever
              play sound meow
              forever
                play sound meow
```

Figure 4. A Screenshot of P9's Game

As an example of a child who did have a plan but who did not write any code, P5 made a backdrop which reads "in this game you have to get your green ball to the yellow ball and you will progress to the next round". This is a plan for a potentially entertaining game, but it has not been implemented at all, with no sprites representing either colour of ball. His debugging score of 8 suggests that he understands how Scratch concepts work although he did not use them in his own game. P5 is in the 59% percentile for his overall EF score, but he has an extremely elevated score for Inhibit which indicates that in general he is distractable and may be diverted from executing plans.

In the case of P1, a twelve year old girl, there is an indication that she thought of a plan but did not prioritise finishing the subtasks which were essential for the user to play the game. According to the instructions for the user, the purpose is to jump up and collect apples from a tree. The event handling code to implement this is not present, although the learner spent time creating scene changes to transfer from a bedroom to an orchard and back. P1 has a mid-range EF score in general, but it is elevated for the Shift subscale. Shift includes the ability to switch or alternate attention and change focus from one topic to another. Here, the learner may have failed to switch attention to the apple gathering mechanic in time to complete the game. The game would have been more successful if the learner had some support in identifying which aspects were essential to get a working prototype of the game completed.

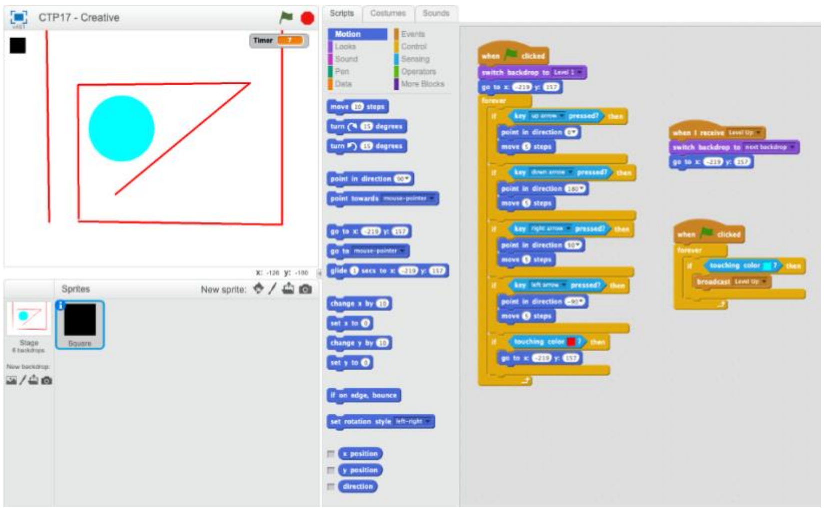
P11 (an eleven year old girl) has developed a plan for her game although her knowledge of Scratch appears to be insufficient to put it fully into practice. P11 has low scores for EF, benchmarked in the 26th percentile for a girl of her age according to the BRIEF2 manual indicating no substantive difficulties with EF in comparison to peers. In the game, there are clear instructions which tell the user to hunt for the apple in each scene. There are multiple scenes, and multiple sprites for decoration. Code is copied across sprites but is not relevant to them. The code does not work because when the user clicks on the apple it switches to only one other room. There is also redundant code with "if" statements which will never execute. Her debugging score of 5 also suggests that her knowledge of Scratch concepts could be improved. In this case, the learner could progress with some support in learning conditional language constructs to help her achieve her initial plan.



```
when [flag] clicked
forever
  if <touching [mouse-pointer] ?> then
    switch backdrop to [bedroom1]
    go to x: (-198) y: (63)
  if <touching [mouse-pointer] ?> then
    switch backdrop to [room2]
    go to x: (188) y: (154)
  if <touching [mouse-pointer] ?> then
    switch backdrop to [room1]
    go to x: (198) y: (131)
  if <touching [mouse-pointer] ?> then
    switch backdrop to [room3]
    go to x: (-208) y: (-137)
```

Figure 5. A Screen Shot of P11's Game

Like many of the children, P17 (an eleven year old girl) made a maze game (see Figure 6) but it is unusually well executed. P17 has low EF scores indicating no difficulties in EF (in the 23rd percentile) and her debugging score of 8 indicates that she has some working knowledge of Scratch constructs. The maze game has several levels of progressing difficulty implemented, indicating some advance planning of the maze features. The code is concise and elegant in comparison to that of her peers, using broadcast to generically level up. A step forwards for this learner might be to try implementing more complex game mechanics such as a scoring mechanism which would require additional computational thinking skills.



```
when [flag] clicked
switch backdrop to [Level 1]
go to x: (-219) y: (157)
forever
  if <key [up arrow] pressed?> then
    point in direction (0)
    move (5) steps
  if <key [down arrow] pressed?> then
    point in direction (180)
    move (5) steps
  if <key [right arrow] pressed?> then
    point in direction (90)
    move (5) steps
  if <key [left arrow] pressed?> then
    point in direction (-90)
    move (5) steps
  if <touching color [🟥] ?> then
    go to x: (-219) y: (157)

when I receive [Level Up]
switch backdrop to [next backdrop]
go to x: (-219) y: (157)

when [flag] clicked
forever
  if <touching color [🟦] ?> then
    broadcast [Level Up]
```

Figure 6. A Screen Shot of P17's Game

## 5. Discussion and Future Research

Initial empirical results suggest that there is a relationship between EF and both creative programming and debugging. This is consistent with the recent finding that computational thinking assessments correlate with general cognitive abilities (r= 0.67) (Román-González et al., 2017). We do not interpret this to mean that CT is just “ordinary” problem solving and that the current worldwide emphasis on developing CT is mistaken. Rather, we see it is one step further in establishing the *nomological net* (Román-González et al., 2017) which links CT to other cognitive variables.

The case studies give an indication of how programming skills and executive functions are both required for learners to be successful in open ended creative programing tasks. Our study suggests EF is linked with the CT abilities of children but metacognition (in terms of self-regulation and self-monitoring) may also contribute to the ability of children to engage in CT. Our findings from a standardized test of EF and analysis of the code produced by the children along with their debugging performance are complementary to previous evidence about children’s metacognition when programming. Allsop’s study in a primary school classroom (in a similar educational context and age group) triangulated evidence from semi-structured interviews, learner journals, observations as well as completed games to examine the planning, monitoring and evaluation skills used by the children (Allsop, 2019). As with our case study learners, Allsop found that debugging required a degree of skill in monitoring and evaluation. Interestingly, the learners in her study reported that their methods of planning their games (such as sketching) transferred into their ability to plan for other learning domains. Allsop’s methodology gives some insight into the children’s thought processes which is beneficial, although in our case the use of standardized tests of EF alongside the assessment of programming enabled us to investigate the relationship between EF and CT. Future research could further examine the impact of MC in the emerging EF-CT relationship.

An important finding from Allsop’s work is that the children in a natural classroom setting used language as an instrument for making decisions, evaluating and regulating activities in conversation with their peers. This suggests that attempts to objectively assess individual performance on creative programming and debugging in a “test” situation where peer collaboration does not occur may underestimate the children’s capabilities. That is, Scratch code and debugging scores from “test” conditions reveal what the learner is capable of without the assistance of a more able peer, whereas more naturalist methodologies show what the learner is able to do within their zone of proximal adjustment (i.e. the human and artefact resources that are the most appropriate form of assistance for a given learner at a particular moment in time (Luckin, 2008)).

As this was exploratory work, further studies are required to confirm these findings, and establish the direction of the effect. This study should be replicated with a larger sample size to establish whether the relationships between these variables hold. While the automated assessment of Scratch programs using Dr Scratch is convenient, hand analysis of the games suggests that there are discrepancies between the inclusion of a code construct and its correct usage. Future versions of Dr Scratch may address such problems such as the inclusion of “dead” code which is never used.

There have been recent calls for improvements in standardised methods for assessing computational thinking (Román-González et al., 2017). This study focussed on programming and debugging, by analysing the product of these processes (program source code). Future work could use assessments of wider computational thinking skills, perhaps using tools such as the CTt (Román-González et al., 2017). We would anticipate that individual test items would be shorter and they would require less sustained concentration on a single problem than an open-ended programming tasks.

## 6. Implications for Educators

The results of this pilot work may be of benefit to classroom teachers who are planning how to support their learners during Scratch projects. It would appear that learners who have difficulties with executive functions, particularly behavioural regulation (including not being able to successfully self-monitor or inhibit their behaviour) could find an open creative task challenging. For such learners, it could be beneficial to introduce a variety of support mechanisms for a zone of proximal adjustment (Luckin, 2008). The teacher could introduce regular check points where each class member reflects (perhaps with a classmate) on the extent to which they have achieved their initial plan. Some learners may need help to come up with a coherent plan in the first place (beyond exploring the interface to find serendipitous effects), and could potentially benefit from lessons in structuring and prioritising the required steps to achieve an initial game idea. This can be scaffolded through activities in which the learners devise a plan to implement a given specification, or are asked to prioritise a list of tasks which need to be achieved to make a specified game.

Other learners in this study appeared to have the executive skills to conceive, execute and monitor a plan but lacked the knowledge of programming language constructs to carry it out. A possible remedy for this would be to recommend project specifications to learners to develop their current level of Scratch knowledge e.g. a learner who has successfully implemented a maze game could be encouraged to try developing a scoring system for it once they have studied a lesson on variables.

While teachers are skilled in scaffolding tasks according to learners' stages for other areas of learning, they may find it harder to achieve if they are themselves new to programming and have not yet developed an intuition for how difficult tasks are to achieve in the target programming language. This emphasises the need for learning materials which are clearly graded in terms of computational thinking difficulty.

If future work was to confirm a strong relationship between EF and aspects of CT, what would be the implications? Firstly, it would be useful for those designing curricula and teaching materials for CT, because previous empirical results about the developmental trajectory of EF would give some guidance of the stage at which it would be appropriate to introduce particular problem solving tasks in CT. Well established empirical results about the development of the ability to look ahead when planning, the development working memory and shifting could be applied. These could be used to design external representations, software scaffolding or pedagogical approaches which would assist learners whose EFs are still developing.

Secondly, knowledge of how certain types of CT task rely on secure EFs could help teachers to plan tasks which are appropriate to their learners. Research into teachers' knowledge of executive functions in mathematics learning illustrates that experienced teachers are aware of the importance of working memory, inhibition and shifting from observation during their practice (Gilmore & Cragg, 2014). However, Gilmore and Cragg found that it may take some years for this understanding to develop and student teachers may not encounter these concepts during their studies. It is therefore important that developers of classroom learning materials think carefully about the executive function demands of their programming and debugging activities and indicate clearly the stage of learning for which each task is suitable. Plain language indications of the underlying skills would also be helpful (such as the statements used in the BRIEF2 tool), as Gilmore and Cragg found that even the experienced teachers who understood the concepts were not familiar with the technical terms from the psychology literature. An introduction to EF and how it relates to CT could be a useful part of initial teacher education programmes.

Lastly, it is possible that CT activities could be a motivating and engaging way to help learners improve their EFs. Because EF is a predictor of life success, academic success and health in later life, interventions which successfully improve EF in young learners are very valuable (Diamond, 2012). Attempts to train EFs (such as working memory) in isolation have shown limited effectiveness when transferred to improving maths proficiency (Titz & Karbach, 2014). More holistic curriculum based interventions have met with more success, for example Diamond's model of the routes to developing EF emphasises the importance of joy; social belonging and support; and the building of confidence, pride and self-efficacy (Diamond, 2012). The design goals of the Scratch community are strikingly similar (Resnick et al., 2009), with the emphasis on fun, low floor, high ceiling and wide walls (as a route to building confidence and pride in achievements), and a large online community for sharing and support[4]. For these reasons, practice during motivating, authentic and appropriately challenging computational activities could be a rich environment in which to develop the executive functions which will be crucially important to children's lives.

## References

Allsop, Y. (2019). Assessing computational thinking process using a multiple evaluation approach. *International Journal of Child-Computer Interaction*, *19*, 30–55. https://doi.org/10.1016/j.ijcci.2018.10.004

Altemeier, L. E., Abbott, R. D., & Berninger, V. W. (2008). Executive functions for reading and writing in typical literacy development and dyslexia. *Journal of Clinical and Experimental Neuropsychology*, *30*(5), 588–606. https://doi.org/10.1080/13803390701562818

Anderson, P. (2002). Assessment and Development of Executive Function (EF) During Childhood. *Child Neuropsychology*, *8*(2), 71–82. https://doi.org/10.1076/chin.8.2.71.8724

Aron, A. R., Robbins, T. W., & Poldrack, R. A. (2004). Inhibition and the right inferior frontal cortex. *Trends in Cognitive Sciences*, *8*(4), 170–177. https://doi.org/10.1016/j.tics.2004.02.010

Best, J. R., Miller, P. H., & Naglieri, J. A. (2011). Relations between executive function and academic achievement from ages 5 to 17 in a large, representative national sample. *Learning and Individual Differences*, *21*(4),

[4] Scratch is the most obvious example, but it is not unique. Unplugged approaches and other programming environments often share the same features.

327–336. https://doi.org/10.1016/j.lindif.2011.01.007

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. *Annual American Educational Research Association Meeting, Vancouver, BC, Canada*, 1–25. https://doi.org/10.1.1.296.6602

Bryce, D., Whitebread, D., & Szűcs, D. (2015). The relationships among executive functions, metacognitive skills and educational achievement in 5 and 7 year-old children. *Metacognition and Learning*, *10*(2), 181–198. https://doi.org/10.1007/s11409-014-9120-4

Bull, R., & Scerif, G. (2010). Developmental Neuropsychology Callosal Contribution to Procedural Learning in Children. *Developmental Neuropsychology*, *19*(3), 37–41. https://doi.org/10.1207/S15326942DN1903

Cohen, J. (1992). A power primer. *Psychological Bulletin*, *112*(1), 155–159. Retrieved from http://psycnet.apa.org/journals/bul/112/1/155/

Cragg, L., & Gilmore, C. (2014). Skills underlying mathematics: The role of executive function in the development of mathematics proficiency. *Trends in Neuroscience and Education*, *3*(2), 63–68. https://doi.org/10.1016/j.tine.2013.12.001

Daly, M., Delaney, L., Egan, M., & Baumeister, R. F. (2015). Childhood Self-Control and Unemployment Throughout the Life Span: Evidence From Two British Cohort Studies. *Psychological Science*, *26*(6), 709–723. https://doi.org/10.1177/0956797615569001

Denckla, M. B. (1996). Research on executive function in a neurodevelopmental context: Application of clinical measures. *Developmental Neuropsychology*, *12*(1), 5–15. https://doi.org/10.1080/87565649609540637

Diamond, A. (2012). Activities and Programs That Improve Children’s Executive Functions. *Current Directions in Psychological Science*, *21*(5), 335–341. https://doi.org/10.1177/0963721412453722

Diamond, A. (2013). Executive functions. *Annual Review of Psychology*, *64*, 135–168. https://doi.org/10.1146/annurev-psych-113011-143750

Dodzik, P. (2017). Behavior Rating Inventory of Executive Function, Second Edition Gerard A. Gioia, Peter K. Isquith, Steven C. Guy, and Lauren Kenworthy. *Journal of Pediatric Neuropsychology*. https://doi.org/10.1007/s40817-017-0044-1

Dolan, C. V, & Molen, M. W. Van Der. (2006). Age-related change in executive function : Developmental trends and a latent variable analysis. *Neuropsychologia*, *44*, 2017–2036. https://doi.org/10.1016/j.neuropsychologia.2006.01.010

Gilmore, C., Attridge, N., Clayton, S., Cragg, L., Johnson, S., Marlow, N., … Inglis, M. (2013). Individual Differences in Inhibitory Control, Not Non-Verbal Number Acuity, Correlate with Mathematics Achievement. *PLoS ONE*, *8*(6), 1–9. https://doi.org/10.1371/journal.pone.0067374

Gilmore, C., & Cragg, L. (2014). Teachers’ Understanding of the Role of Executive Functions in Mathematics Learning. *Mind, Brain and Education : The Official Journal of the International Mind, Brain, and Education Society*, *8*(3), 132–136. https://doi.org/10.1111/mbe.12050

Gioia, G., Isquith, P., Guy, S., & Kenworthy, L. (2015). *BRIEF 2 Behaviour Rating Inventory of Executive Function: Professional Manual*. Lutz, FL.

Grover, S., & Pea, R. (2013). Computational Thinking in K–12: A Review of the State of the Field. *Educational Researcher*, *42*(1), 38–43. https://doi.org/10.3102/0013189x12463051

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. https://doi.org/10.1080/08993408.2015.1033142

Knouse, L. E., Feldman, G., & Blevins, E. J. (2014). Executive functioning difficulties as predictors of academic performance: Examining the role of grade goals. *Learning and Individual Differences*, *36*, 19–26. https://doi.org/10.1016/j.lindif.2014.07.001

Luckin, R. (2008). The learner centric ecology of resources: A framework for using technology to scaffold learning. *Computers & Education*, *50*(2), 449–462. https://doi.org/10.1016/j.compedu.2007.09.018

Luria, A. (1966). *Higher cortical functions in man*. London: Tavistock.

Lyons, K. E., & Zelazo, P. D. (2011). Monitoring, metacognition, and executive function. Elucidating the role of self-reflection in the development of self-regulation. In *Advances in Child Development and Behavior* (1st ed., Vol. 40). https://doi.org/10.1016/B978-0-12-386491-8.00010-4

Mayer, R. E. (2004). Should There Be a Three-Strikes Rule Against Pure Discovery Learning? *American Psychologist*, *59*(1), 14–19. https://doi.org/10.1037/0003-066X.59.1.14

McLean, J. F., & Hitch, G. J. (1999). Working memory impairments in children with specific arithmetic learning difficulties. *Journal of Experimental Child Psychology*, *74*(3), 240–260. https://doi.org/10.1006/jecp.1999.2516

Miyake, A., Friedman, N. P., Emerson, M. J., Witzki, A. H., Howerter, A., & Wager, T. D. (2000). The Unity and Diversity of Executive Functions and Their Contributions to Complex “Frontal Lobe” Tasks: A Latent Variable Analysis. *Cognitive Psychology*, *41*(1), 49–100. https://doi.org/10.1006/cogp.1999.0734

Moreno-León, J., Harteveld, C., Román-González, M., & Robles, G. (2017). On the Automatic Assessment of Computational Thinking Skills: A Comparison with Human Experts. *Conference on Human Factors in Computing Systems (CHI)*, 2788–2795. https://doi.org/10.1145/3027063.3053216

Moreno-León, J., & Robles, G. (2015). Dr. Scratch. *Proceedings of the Workshop in Primary and Secondary Computing Education on ZZZ - WiPSCE ’15*, (November), 132–133. https://doi.org/10.1145/2818314.2818338

Moreno-Leon, J., Robles, G., & Roman-Gonzalez, M. (2016). Comparing computational thinking development assessment scores with software complexity metrics. *IEEE Global Engineering Education Conference, EDUCON*, *10-13-Apri*(April), 1040–1045. https://doi.org/10.1109/EDUCON.2016.7474681

Palumbo, D. B. (1990). Programming Language/Problem-Solving Research: A Review of Relevant Issues. *Review of Educational Research*, *60*(1), 65–89. https://doi.org/10.3102/00346543060001065

Papert, S. (1991). Situating Constructionism. *Constructionism*, 1–11. https://doi.org/10.1111/1467-9752.00269

Parham, J., Gugerty, L., & Stevenson, D. E. (2010). Empirical evidence for the existence and uses of metacognition in computer science problem solving. *SIGCSE’10 - Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, 416–420. https://doi.org/10.1145/1734263.1734406

Pennington, B. F., & Ozonoff, S. (1996). Executive functions and developmental psychopathology. *Journal of Child Psychology and Psychiatry*, *37*(I), 51–87. https://doi.org/10.1111/j.1469-7610.1996.tb01380.x

Resnick, M. (2007). Sowing the Seeds for a More Creative Society. *Learning and Leading With Technology*, *35*(4), 18–22.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., … Kafai, Y. (2009). Scratch: programming for all. *Communications of the ACM*, *52*(11), 60–67. https://doi.org/10.1145/1592761.1592779

Rich, K. M., Andrew Binkowski, T., Strickland, C., & Franklin, D. (2019). A k-8 debugging learning trajectory derived from research literature. *SIGCSE 2019 - Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 745–751. https://doi.org/10.1145/3287324.3287396

Roebers, C. M. (2017a). Executive function and metacognition: Towards a unifying framework of cognitive self-regulation. *Developmental Review*, *45*, 31–51. https://doi.org/10.1016/j.dr.2017.04.001

Roebers, C. M. (2017b). Executive function and metacognition: Towards a unifying framework of cognitive self-regulation. *Developmental Review*, *45*, 31–51. https://doi.org/10.1016/j.dr.2017.04.001

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. https://doi.org/10.1016/j.chb.2016.08.047

Royal Society. (2017). *After the reboot : computing education in UK schools*.

Scherer, R., Siddiq, F., & Sanches Viveros, B. (2018). Learning to Code—Does It Help Students to Improve Their Thinking Skills? In S. KONG (Ed.), *International Conference on Computational Thinking Education* (pp. 37–40). Retrieved from http://www.eduhk.hk/cte2018/doc/CTE2018 Proceeding_Full_20180604.pdf

St Clair-Thompson, H. L., & Gathercole, S. E. (2006). Executive functions and achievements in school: Shifting, updating, inhibition, and working memory. *Quarterly Journal of Experimental Psychology*, *59*(4), 745–759. https://doi.org/10.1080/17470210500162854

The Royal Society. (2012). Shut down or restart? The way forward for computing in UK schools. In *Technology*. https://doi.org/10.1088/2058-7058/25/07/21

Titz, C., & Karbach, J. (2014). Working memory and executive functions: effects of training on academic achievement. *Psychological Research*. https://doi.org/10.1007/s00426-013-0537-1

Toplak, M. E., West, R. F., & Stanovich, K. E. (2013). Practitioner Review: Do performance-based measures and ratings of executive function assess the same construct? *Journal of Child Psychology and Psychiatry and Allied Disciplines*, *54*(2), 131–143. https://doi.org/10.1111/jcpp.12001

Van der Ven, S. H. G., Kroesbergen, E. H., Boom, J., & Leseman, P. P. M. (2012). The development of executive functions and early mathematics: A dynamic relationship. *British Journal of Educational Psychology*, *82*(1), 100–119. https://doi.org/10.1111/j.2044-8279.2011.02035.x

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining Computational Thinking for Mathematics and Science Classrooms. *Journal of Science Education and Technology*, *25*(1), 127–147. https://doi.org/10.1007/s10956-015-9581-5

*International Journal of Computer Science Education in Schools, April 2020, Vol. 3, No. 4*
*ISSN 2513-8359*

# Investigation of Computational Thinking in the Context of ICT and Mobile Technologies

**Didem Alsancak Sırakaya**

Kırşehir Ahi Evran University

DOI: 10.21585/ijcses.v3i4.73

**Abstract**

This research aims to determine the change in students' computational thinking skills according to their ICT and mobile technology experience and frequency of use. The sample of the study, designed with the survey model, consisted of 269 students attending a vocational school of higher education. Data were collected using the Computational Thinking Scale and the Personal Information Form. Descriptive statistics, independent samples t-test and one-way ANOVA were used in data analysis. According to results, it was determined that students' computational thinking skills differs according to their internet experience, mobile device experience, mobile internet experience and period of daily mobile Internet use, while no differences were found based on computer experience, the number of times they checked their mobile devices a day and purpose of mobile technology usage.

**Keywords:** computational thinking, ICT experience, mobile technology experience

## 1. Introduction

The increase in the space that technology occupies in our lives in terms of volume and function brings about the necessity to update the features that individuals should have (Sırakaya, 2019). Today, regardless of age, every individual is expected to have basic computer skills. Kalelioğlu, Gülbahar and Kukul (2016) state that today, all individuals should have some basic computational skills. Drawing attention to a similar topic, Kalelioğlu (2015) and Sáez-López, Román-González and Vázquez-Cano Lopez (2016), emphasize that individuals in 21st century should not only use technology but also produce technology. In this context, computational thinking skills come to the forefront as among the important skills students should acquire. The concept of computational thinking skills which gained popularity in 2006 with the research conducted by Wing, is essentially a concept that has been discussed in the literature for many years. Wing (2006) pointed out that computational thinking is a necessary competence for every individual.

Although it has been discussed for a long time, it can be argued that there is no consensus on the definition of computational thinking (Grover & Pea, 2013; Demir & Seferoğlu, 2017). It is seen that similar definitions are produced for the concept of computational thinking. Some of these definitions are based on computer sciences. Korkmaz, Cakır and Özden (2017) and Wing (2008) point out that the concepts and applications that constitute computational thinking are based on the basic concepts of computer science. Using computer science concepts, Wing (2006) defines computational thinking as problem solving, systems design, and human behaviour analysis. Similarly, Sengupta, Kinnebrew, Basu, Biswas and Clark (2013) state that computational thinking utilizes the basic subjects and concepts found in computer sciences. In different definitions, computational thinking is associated with concepts such as problem solving (Lye & Koh, 2014), algorithmic thinking (Barr and Stephenson, 2011; Lee et al., 2011) and abstraction (Wing, 2008). Kalelioğlu, Gülbahar and Kukul, (2016) identify three most accepted components of computational thinking as abstraction, algorithmic thinking and problem solving. ISTE

(International Society for Technology in Education) (2019) examines computational thinking skills under the categories of fragmentation, patterning, abstraction and algorithm. Shute et al. (2017) cite skills such as fragmentation, abstraction, generalization, algorithmic design, debugging and iteration along with thinking and acting as computational thinking skills. From a more general perspective, Pulimood, Pearson and Bates (2016) describe the reasoning process in the solution of abstract problems as computational thinking.

For a better understanding of computational thinking and its concepts, the operational definitions are provided along with its conceptual definition. In their operational definition that considers computational thinking as a problem-solving process, ISTE and CSTA (Computer Science Teachers Association) state that computational thinking includes, but not limited to, the following activities (ISTE, 2019):

- Re-formulating problems in order to solve them with computers and other tools.
- Organizing and analyzing data logically.
- Re-presenting data in manners of abstraction, such as models and simulations.
- Automating solutions through algorithmic thinking.
- Identifying, analyzing and implementing possible solutions to ensure the most effective and efficient combination of steps and resources.
- Generalizing and transferring the problem-solving process to a wide range of problems.

In order to understand the concept of computational thinking more clearly, it may be useful to look at the process from the reverse. In this context, it is useful to take the characteristics of individuals who have computational thinking skills as a reference. Accordingly, individuals with computational thinking skills have the following characteristics (Lee et al. 2011; Wing, 2006, 2008, 2011):

- Making problems solvable by using technological tools.
- Organizing and analyzing data logically.
- Making the data abstract.
- Developing solutions through algorithmic thinking.
- Identifying, analyzing and applying possible solutions and resources.
- Adapting the solution to different problems.

Computational thinking a key skill for the 21st century (Pérez-Marín, Hijón-Neira, Bacelo & Pizarro, 2018), is a necessary skill for every individual just like literacy and basic mathematical skills (Wing, 2014). Educators are researching how to ensure that students acquire computational thinking skill that is regarded to be highly important (Wing, 2006). Many researchers suggest that computational thinking skill should be added to the curriculum in order to ensure students are given an opportunity to acquire computational thinking skills (Juškevičienė & Dagienė, 2018; Karal et al., 2017; Yadav et al., 2017). As a matter of fact, many countries include computational thinking in their curricula in order to instruct students starting from early ages (Küçük & Şişman, 2017; Webb et al., 2017; Wong & Cheung, 2018). It is aimed to improve students' computational thinking skills through activities such as courses, projects and competitions organized as a supplement to the curriculum. However, it is not clear how to acquire and evaluate this skill, since the definition and limits of computational thinking skills are not clear (Pérez-Marín et al., 2018; Werner, Denner, Campe & Kawamoto, 2012). In addition, the required level is not achieved yet in terms of resources and informed teachers that are needed to ensure this skill is acquired by students (Brackmann et al., 2016; Pérez-Marín et al., 2018). Literature review shows that different methods and tools are used to develop computational thinking skills. Various methods such as computer-free activities (Takaoka, Fukushima, Hirose & Hasegawa, 2014), block-based programming (Kalelioğlu, 2015; Yünkül et al., 2017; Oluk & Korkmaz, 2016; Oluk, Korkmaz & Oluk, 2018), text-based programming (Alsancak-Sırakaya, 2019) and robotic sets (Karaahmetoğlu & Korkmaz, 2019) are used for the development of computational thinking skills.

Although different tools are used in the development of computational thinking skills, it is remarkable that most of these tools are technological. Technological tools such as computers, mobile devices, programming languages and robotic sets play an important role in the process of acquiring computational skills. Pellas and Peroutseas (2016) state that computer sciences are an important resource in the acquisition of computational thinking skills. Similarly, in their studies, Yıldız Durak and Sarıtepeci (2018) report that experience in using information and communication technologies (ICT) may influence computational thinking skills. Juškevičienė and Dagienė (2018) state that research on the relationship between digital competence and computational thinking is needed. Based on these, this study aims to determine the change in students' computational thinking skills according to their experience and frequency of ICT and mobile technology usage. For this purpose, answers to the following sub-problems are be sought:

- Do students' computational thinking skills significantly differ according to their experience in computer and Internet use?
- Do students' computational thinking skills significantly differ according to their experience in using mobile technologies?
- Do students' computational thinking skills significantly differ according to the frequency of their mobile technology use?
- Do students' computational thinking skills significantly differ according to the purpose of using mobile technology?

## 2. Method

### *2.1 Research Design*

Screening model was used in the study. Screening model reveals a group's attitudes, beliefs, thoughts, expectations, attitudes, and characteristics (Creswell, 2012). Creswell (2012) defines screening model as "quantitative research processes that researchers apply to a specific sample to define attitude, opinion, behavior or characteristic features related to the universe". Generally, the aim of screening studies conducted with larger sample groups compared to other types of research is to reveal the situation in question as is (Büyüköztürk, Kılıç Çakmak, Akgün, Karadeniz & Demirel, 2008).

### *2.2 Universe and Sample*

The universe of the study consisted of vocational school of higher education students at a state university and the sample is composed of 269 students from a vocational college at the same state university. According to gender, 18.6% (50) of the participants were female and 81.4% (219) were male. Of these, 23% (62) were in their first year and 77% (207) were in their second year. According to department, 31.6% (85) were students at Computer Technologies, 48.3% (130) were studying Construction Technology, 20.1% (54) attended the Department of Electricity and Energy. Convenient sampling method was used in determining the study group. In the convenient sampling method, the researcher tries to reach the number of samples that is needed by starting with the participants that he/she can reach most easily (Büyüköztürk et al., 2008). Ethical permit document has been obtained from the educational institution that the students are affiliated with.

### *2.3 Data Collection and Data Collection Tools*

The process of data collection began with informing participants verbally about the purpose of the study. Then, data were collected from volunteer participants through data collection tools. The data collection tools used in this study are described below:

Computational Thinking Scale: The Computational Thinking Scale developed by Korkmaz, Çakır and Özden (2017) was used to determine the computational thinking skills of the participants. The scale, with a total of 29 items, is collected under 5 factors. The internal consistency coefficient of the whole scale was calculated as 0.822 and the internal consistency coefficients of the factors were stated as follows: Creativity (0.843), Algorithmic thinking (0.869), Cooperativity (0.865), Critical thinking (0.784) and Problem solving (0.727). The reliability analyses of the scale were re-performed with the data collected within the scope of this study. Accordingly, the overall reliability coefficient of the scale was calculated to be 0.869 with the following internal consistency coefficients for the factors: Creativity, 0.855; Algorithmic thinking, 0,913; Cooperativity, 0.77; Critical thinking, 0,818 and Problem solving, 0.817. Korkmaz, Çakır and Özden (2017) stated that the scale is a valid measurement tool based on the exploratory and confirmatory factor analyses.

Personal Information Form: Personal Information Form prepared by the researchers was used to determine the gender, department, school year, ICT experience, mobile technology experience, frequency of mobile technology usage and purpose of mobile technology use. The form, which was prepared in accordance with the opinions of two subject areas experts, consists of a total of 16 questions.

### *2.4 Data Analysis*

Since there was a small amount of data loss (1%) in the responses to the scale items, mean substitution technique was used based on the recommendation of Schumacker and Lomax (2004). Since the sample size was greater than 50, Kolmogorov-Smirnov test (Büyüköztürk, 2007) and Q-Q Plot graphs were used to determine whether the data showed normal distribution or not. As a result of the test, it was found that the normal distribution value was not statistically significant (p> .05) and the graphical analysis showed that the data showed normal distribution. For this reason, data analysis included descriptive statistics along with parametric tests such as independent samples t-test and one-way ANOVA. Levene test (variance homogeneity of groups) was taken into consideration in determining which groups caused the difference as a result of ANOVA test. Since the variance was homogeneously distributed in all variables (p> .05), LSD test was preferred (Büyüköztürk, 2007).

## 3. Results

### *3.1 Findings Related to Computer and Internet Experience*

In order to determine whether students' computational thinking skills changed according to their computer and internet experiences, one-factor analysis of variance was used for independent samples. The test results are given in Table 1.

Table 1. Change of computational thinking skills according to computer and internet experience

| | | | N | $\overline{X}$ | S | | | | N | $\overline{X}$ | S |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Computer experience** | 1 | Less than 1 year | 19 | 87.06 | 21.64 | **Internet experience** | 1 | Less than 1 year | 9 | 94.44 | 17.64 |
| | 2 | Between 1-2 years | 18 | 98.90 | 16.21 | | 2 | Between 1-2 years | 28 | 91.54 | 20.51 |
| | 3 | Between 2-3 years | 36 | 100.53 | 15.20 | | 3 | Between 2-3 years | 43 | 96.01 | 16.99 |
| | 4 | Between 3-4 years | 46 | 98.92 | 18.04 | | 4 | Between 3-4 years | 59 | 97.42 | 16.06 |
| | 5 | Between 4-5 years | 52 | 100.67 | 15.36 | | 5 | Between 4-5 years | 58 | 98.69 | 14.65 |
| | 6 | More than 5 years | 88 | 98.58 | 14.12 | | 6 | More than 5 years | 64 | 102.92 | 15.55 |
| | | Total | 259 | 98.51 | 16.22 | | | Total | 261 | 98.08 | 16.58 |

| Variable | Source of variance | Sum of squares | SD | Mean square | F | p | Significant difference |
|---|---|---|---|---|---|---|---|
| **Computer experience** | Between groups | 2890.152 | 5 | 578.030 | | | |
| | In-groups | 65017.289 | 253 | 256.985 | 2.249 | .050 | ---- |
| | Total | 67907.441 | 258 | | | | |
| **Internet experience** | Between groups | 3047.367 | 5 | 609.473 | | | Between 6 and 2, 3 |
| | In-groups | 68446.342 | 255 | 268.417 | 2.271 | .048 | |
| | Total | 71493.709 | 260 | | | | |

According to the Table, it was determined that students' computational thinking mean scores did not show significant differences based on computer experience. ($F_{(5-253)} = 2.249$; $p <=.05$).

As a result of the analysis, it was determined that internet experience caused a significant difference in computational thinking mean scores ($F_{(5-255)} = 2.271$; $p <.05$). According to the LSD test, it was found that students with more than 5 years of internet experience had significantly higher computational thinking scores than those with 1-2 years and 2-3 years of experience.

### *3.2 Findings on Mobile Technology Experience*

One-factor analysis of variance for independent samples was used to determine whether students' computational thinking skills changed based on mobile technology experience. The test results are given in Table 2.

Table 2. Change of computational thinking skills based on mobile technology experience

| | | | N | $\overline{X}$ | S | | | | N | $\overline{X}$ | S |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Mobile device experience** | 1 | Less than 1 year | 9 | 89.26 | 22.60 | **Mobile Internet experience** | 1 | Less than 1 year | 12 | 86.88 | 18.57 |
| | 2 | Between 1-2 years | 15 | 87.99 | 21.22 | | 2 | Between 1-2 years | 24 | 90.48 | 18.98 |
| | 3 | Between 2-3 years | 26 | 94.54 | 14.06 | | 3 | Between 2-3 years | 40 | 94.95 | 18.28 |
| | 4 | Between 3-4 years | 40 | 98.39 | 18.36 | | 4 | Between 3-4 years | 50 | 99.48 | 14.05 |
| | 5 | Between 4-5 years | 51 | 98.15 | 15.50 | | 5 | Between 4-5 years | 69 | 98.05 | 16.00 |
| | 6 | More than 5 years | 126 | 100.67 | 15.07 | | 6 | More than 5 years | 68 | 103.19 | 14.54 |
| | | Total | 267 | 98.15 | 16.48 | | | Total | 263 | 97.98 | 16.52 |

| Variable | Source of variance | Sum of squares | SD | Mean square | F | p | Significant difference |
|---|---|---|---|---|---|---|---|
| **Mobile device experience** | Between groups | 3400.874 | 5 | 680.175 | 2.576 | .027 | Between 1 and 6<br>Between 2 and 4, 5, 6 |
| | In-groups | 68913.698 | 261 | 264.037 | | | |
| | Total | 72314.572 | 266 | | | | |
| **Mobile Internet experience** | Between groups | 5154.202 | 5 | 1030.840 | 3.990 | .002 | Between 1 and 4, 5, 6<br>Between 2 and 4, 5, 6 |
| | In-groups | 66404.305 | 257 | 258.383 | | | |
| | Total | 71558.507 | 262 | | | | |

When Table 2 was examined, it was found that students' computational thinking skills mean scores differed significantly according to mobile device experience ($F_{(5-261)} = 2.576$; $p < .05$). The results of the LSD test conducted to determine which groups caused the difference show that computational thinking skills mean scores of students with less than 1 year of mobile device experience were lower compared to those with more than 5 years of experience and computational thinking skills mean scores of students with 1-2 years of experience were significantly lower than those with 3-4 years, 4-5 years and more than 5 years experience in mobile devices.

It was concluded that mobile internet experience caused a significant difference in computational thinking skills scores ($F_{(5-257)} = 3.990$; $p < .05$). According to the LSD test, it was found that the computational thinking skills mean scores of students with less than 1 year and 1-2 years of mobile internet experience were significantly lower than those with 3-4 years, 4-5 years and more than 5 years experience.

### *3.3 Findings Related to Frequency of Mobile Technology Use*

One-factor analysis of variance for independent samples was used in order to determine whether students' computational thinking skills differed based on the frequency of mobile technology use. Table 3 presents the test results.

Table 3. Changes in computational thinking skills based on frequency of mobile technology use

| | | | N | $\overline{X}$ | S | | | | N | $\overline{X}$ | S |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Number of daily checks for the mobile | 1 | 1-19 | 69 | 93.44 | 19.66 | Duration of daily mobile internet use | 1 | < 1 hour | 38 | 93.52 | 19.44 |
| | 2 | 20- 39 | 48 | 102.53 | 13.64 | | 2 | 1-2 hours | 53 | 101.21 | 15.02 |
| | 3 | 40-59 | 46 | 97.85 | 14.72 | | 3 | 2-3 hours | 54 | 98.10 | 15.62 |
| | 4 | 60-79 | 29 | 97.77 | 15.37 | | 4 | 3-4 hours | 28 | 91.89 | 17.77 |
| | 5 | 80-99 | 29 | 102.21 | 17.70 | | 5 | 4-5 hours | 35 | 101.73 | 17.05 |
| | 6 | >99 | 46 | 98.28 | 14.22 | | 6 | More than 5 hours | 56 | 99.99 | 12.17 |
| | | Total | 267 | 98.09 | 16.47 | | | Total | 264 | 98.29 | 16.11 |

| Variable | Source of variance | Sum of squares | Sd | Mean square | F | p | Significant difference |
|---|---|---|---|---|---|---|---|
| **Number of daily checks for the mobile** | Between groups | 2938.162 | 5 | 587.632 | 2.214 | .053 | ----- |
| | In-groups | 69284.439 | 261 | 265.458 | | | |
| | Total | 72222.601 | 266 | | | | |
| **Duration of daily mobile internet use** | Between groups | 3041.923 | 5 | 608.385 | 2.406 | .037 | Between 1 and 2, 5 |
| | In-groups | 65238.895 | 258 | 252.864 | | | Between 2 and 4 |
| | Total | 68280.818 | 263 | | | | Between 4 and 5, 6 |

According to Table 3, it was found that students' computational thinking skills mean scores did not show significant difference based on how many times they checked their mobile devices ($F_{(5\text{-}261)} = 2.214$; $p>.05$).

Based on the conducted analyzes, it was determined that duration of daily mobile internet use caused a significant difference in computational thinking skills mean scores ($F_{(5\text{-}258)} = 2.406$; $p <.05$). According to LSD test results, computational thinking skills of students with less than 1 hour of daily mobile internet use were significantly lower than those with 1-2 hours and 4-5 hours; computational thinking skills of students with 3-4 hours were significantly lower than those with 1-2 hours and computational thinking skills of students with 3-4 hours were significantly lower than those with 4-5 hours and more than 5 hours.

### *3.4 Findings Regarding Purpose of Use of Mobile Technology*

Independent samples t-test was applied to determine whether students' computational thinking skills differed based on their purpose for using a mobile technology. Table 4 presented the obtained results.

Table 4. Change of computational thinking skills based on purpose of using mobile technology

| **Purpose of Use** | **At present** | **N** | **$\bar{X}$** | **S** | **SD** | **t** | **p** |
|---|---|---|---|---|---|---|---|
| Connecting to social networks | Yes | 239 | 98.02 | 15.79 | 260 | .768 | .070 |
| | No | 23 | 95.29 | 20.72 | | | |
| Playing games | Yes | 146 | 98.20 | 15.86 | 245 | .433 | .387 |
| | No | 101 | 97.27 | 17.53 | | | |
| Keeping up to date with current news | Yes | 222 | 98.49 | 16.30 | 251 | 1.309 | .995 |
| | No | 31 | 94.37 | 16.90 | | | |
| Doing homework/research | Yes | 221 | 98.20 | 16.68 | 254 | 1.027 | .224 |
| | No | 35 | 95.14 | 14.01 | | | |
| Listening to music | Yes | 243 | 98.24 | 15.84 | 257 | .410 | .106 |
| | No | 16 | 96.51 | 23.31 | | | |
| Online shopping | Yes | 144 | 99.16 | 15.64 | 235 | 1.298 | .487 |
| | No | 93 | 96.39 | 16.69 | | | |
| Watching videos | Yes | 237 | 98.17 | 15.80 | 252 | .445 | .404 |
| | No | 17 | 96.38 | 19.60 | | | |

Table 4 shows that while computational thinking skill mean scores of students who used mobile technology to connect to social networks ($\bar{X}$=98.02) were higher than the mean score of the students who did not use mobile technology for this purpose ($\bar{X}$=95.29), the difference was not significant ($t_{(260)}$= .768, p>.05). Similarly, it was concluded that while computational thinking skill mean scores of students who used their mobile technology to play games ($\bar{X}$=98.22) were higher than the mean scores of students who did not use mobile technology purpose ($\bar{X}$=97.27), the difference was not significant ($t_{(245)}$= .433, p>.05).

Although the computational thinking skills mean scores of students who used their mobile technology to follow the current developments ($\bar{X}$=98.49) were higher than the computational thinking skills mean scores of students who did not use their devices purpose ($\bar{X}$=94.37), the difference was not significant ($t_{(251)}$= 1.309, p>.05). It was found that although the computational thinking skills mean scores of students who used their mobile technology for doing homework/research ($\bar{X}$=98.20) were higher than the computational thinking skills mean scores of students who did not use their mobile technology for this purpose ($\bar{X}$=95.14), the difference was not significant ($t_{(254)}$= 1.027, p>.05). While the computational thinking skills mean scores of students who used their mobile technology to listen to music was higher ($\bar{X}$=98.24) than the computational thinking skills mean scores of students who did not use their devices for this purpose ($\bar{X}$=96.51), the difference was not significant ($t_{(257)}$= .410, p>.05). It was also found that the computational thinking skills mean scores of students who used their mobile technology to do online shopping were higher ($\bar{X}$=99.16) than he computational thinking skills mean scores of students who did not use their devices for this purpose ($\bar{X}$=96.39); the difference was not significant ($t_{(235)}$= 1.298, p>.05). The findings also show that while the computational thinking skills mean scores of students who used their mobile technology to watch videos were higher ($\bar{X}$=98.17) than the computational thinking skills mean scores of students who did not use their devices for this purpose ($\bar{X}$=96.38), the difference was not significant ($t_{(252)}$= .445, p>.05). Evaluation of these results in general demonstrates that students' computational thinking skills did not differ based on the purpose of mobile technology use and that their mean scores were very close to one another.

## 4. Results and Discussion

Based on the results of the analyses, it was determined that students' computational thinking skills did not differ based on computer experience. Arriving at a similar finding, Oluk and Korkmaz (2016) stated that computational thinking skills did not differ according to the duration of daily computer use. Another result obtained in the study demonstrated that internet experience affected computational thinking skills. Accordingly, students with 1-2 years internet experience had significantly lower computational skills than those with more than 5 years experience. Korucu et al. (2017) concluded that middle school students' computational thinking skills did not change based on their weekly internet use. Yıldız Durak and Sarıtepeci (2018) concluded that ICT experience did not predict computational thinking skills.

The analyses demonstrated that mobile technology experience differentiated computational thinking skills. Accordingly, students with less mobile device experience and less mobile Internet experience had significantly lower computational thinking skills than those with more experience. This finding can be interpreted to suggest that mobile technology experience can increase computational thinking skills. Reaching a different conclusion, Korucu et al. (2017) stated that those with only 2 years of mobile technology experience had significantly higher computational thinking skills than with longer experience. Differences in the levels of samples in studies may cause variations in the obtained results.

The analyses conducted based on the frequency of mobile technology use concluded that computational thinking skills did not differ according to how many times a person checked his/her mobile device but varied according to the duration of daily mobile Internet use. Accordingly, students who use less mobile Internet daily had significantly lower computational thinking skills than those who used mobile Internet more. According to Korucu et al. (2017), who similarly studied the ability to use mobile devices as a variable, computational thinking skills did not differ. Yıldız Durak and Sarıtepeci (2018), who examined the duration of daily internet use of secondary school students, concluded that this variable did not affect computational thinking skills.

The study also aimed to determine whether students' computational thinking skills differed based on their purposes while using their mobile technology. According to the analyses, it was concluded that using mobile technology to connect to social networks, play games, follow the current developments, do homework/research, listen to music, shop and watch videos did not change students' computational thinking skills. While computational thinking skills differed according to mobile device experience and frequency of use, it is a remarkable finding that computational thinking skills did not change based on purpose of use. There are no other studies in the literature that explored these variables. Future studies may consider filling this gap.

This research is limited to 269 vocational school students in terms of participants. One of the limitations of the study is that self-reported instruments were used. Self-reported instruments may not reflect the actual measure as students' perceptions might be differ from their actual levels. Qualitative data collection tools such as observation and interview can be used in future research. In addition, experimental studies examining CT and ICT (mobile technology) can be conducted.

## References

Alsancak-Sırakaya, D. (2019). Programlama öğretiminin bilgi işlemsel düşünme becerisine etkisi. *Turkish Journal of Social Research/Turkiye Sosyal Arastirmalar Dergisi, 23*(2), 575-590.

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? *Inroads*, *2*(1), 48-54.

Brackmann, C., Barone, D., Casali, A., Boucinha, R., & Muñoz-Hernandez, S. (2016, September). Computational thinking: Panorama of the Americas. In *2016 International symposium on computers in Education (SIIE)* (pp. 1-6). IEEE.

Brennan, K., & Resnick, M. (2012, April). New frameworks for studying and assessing the development of computational thinking. *Paper presented at the 2012 annual meeting of the American Educational Research Association*, Vancouver, Canada.

Büyüköztürk, Ş. (2007). *Sosyal Bilimler İçin Veri Analizi El Kitabı*. Ankara: PegemA Yayıncılık.

Büyüköztürk, Ş., Kılıç Çakmak, E., Akgün, Ö., E., Karadeniz, Ş., & Demirel, F. (2008). *Bilimsel Araştırma Yöntemleri*. Ankara: Pegem Akademi.

Creswell, J. W. (2012). *Educational research: Planning, conducting and evaluatingquantitative and qualitative research*. Boston: Pearson Education.

Demir, G. Ö., & Seferoğlu, S. S. (2017). Yeni kavramlar, farklı kullanımlar: Bilgi-işlemsel düşünmeyle ilgili bir değerlendirme (Eds: Akkoyunlu, B., Odabaşı, F., & İşman, A.). In *Eğitim teknolojileri okumaları*, 801-830.

Grover, S., & Pea, R. (2013). Computational thinking in K12 a review of the state of the field. *Educational Researcher, 42*(1), 38-43.

ISTE (2019). Operational definition of computational thinking. Retrieved from https://id.iste.org/docs/ct-documents/computational-thinking-operational-definition-flyer.pdf on 28.11.2019

Juškevičienė, A., & Dagienė, V. (2018). Computational thinking relationship with digital competence. *Informatics* *in Education, 17*(2), 265-284.

Kalelioglu, F., Gülbahar, Y., & Kukul, V. (2016). A framework for computational thinking based on a systematic research review. *Baltic Journal of Modern Computing, 4*(3), 583

Kalelioğlu, F. (2015). A new way of teaching programming skills to K-12 students: Code.org. *Computers in Human Behavior*, 52, 200-210.

Karaahmetoğlu, K., & Korkmaz, Ö. (2019). The effect of project-based arduino educational robot applications on students' computational thinking skills and their perception of Basic Stem skill levels. *Participatory Educational Research*, *6*(2), 1-14.

Karal, H., Şılbır, G.M., & Yıldız, M. (2017). *STEM eğitiminde bilişimsel düşünme ve kodlamanın rolü.* (Ed: Salih Çepni). In *Kuramdan Uygulamaya STEM Eğitimi.* Ankara: Pegem Akademi, 389-411.

Karasar, N. (2008). *Bilimsel araştırma yöntemi.* (18. Baskı). Ankara: Nobel Yayın Dağıtım.

Korkmaz, Ö., Çakir, R., & Özden, M. Y. (2017). A validity and reliability study of the Computational Thinking Scales (CTS). *Computers in Human Behavior*, *72*, 558-569.

Korucu, A.T., Gençtürk, A.T. & Gündoğdu, M.M. (2017). Examination of the computational thinking skills of students. *Journal of Learning and Teaching in Digital Age, 2*(1), 11-19.

Küçük, S., & Şişman, B. (2017). Behavioral patterns of elementary students and teachers in one-to-one robotics instruction. *Computers & Education*, 111, 31-43.

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J., & Werner, L., 2011. Computational thinking for youth in practice. *Acm Inroads,* 2(1), 32-37.

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

Oluk, A., Korkmaz, Ö., & Oluk, H.A. (2018). Scratch'ın 5. sınıf öğrencilerinin algoritma geliştirme ve bilgi-işlemsel düşünme becerilerine etkisi. *Turkish Journal of Computer and Mathematics Education, 9*(1), 54-71.

Oluk, A., & Korkmaz, Ö. (2016). Comparing students' scratch skills with their computational thinking skills in terms of different variables. *Online Submission*, *8*(11), 1-7.

Pellas, N., & Peroutseas, E. (2016). Gaming in Second Life via Scratch4SL: Engaging high school students in programming courses. *Journal of Educational Computing Research*, *54*(1), 108-143.

Pérez-Marín, D., Hijón-Neira, R., Bacelo, A., & Pizarro, C. (2018). Can computational thinking be improved by using a methodology based on metaphors and scratch to teach computer programming to children? *Computers in Human Behavior*.

Pulimood, S. M., Pearson, K., & Bates, D. C. (2016, February). A study on the impact of multidisciplinary collaboration on computational thinking. In *Proceedings of the 47th ACM technical symposium on computing science education* (pp. 30-35). ACM.

Sáez-López, J. M., Román-González, M., & Vázquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school: A two years case study using "Scratch" in five schools. *Computers & Education*, *97*, 129-141.

Schumacker, R.E., & Lomax, R.G. (2004). *Data Entry and Data Editing Issues.* In A beginner's guide to structural equation modeling (pp. 13-33). Mahwah, NJ: Lawrence Erlbaum

Sengupta, P., Kinnebrew, J. S., Basu, S., Biswas, G., & Clark, D. (2013). Integrating computational thinking with K-12 science education using agent-based computation: A theoretical framework. *Education and Information Technologies*, *18*(2), 351-380.

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, *22*, 142-158.

Sırakaya, M. (2019). İlkokul ve ortaokul öğretmenlerinin teknoloji kabul durumları. *İnönü Üniversitesi Eğitim Fakültesi Dergisi, 20*(2), 578-590.

Webb, M., Davis, N., Bell, T., Katz, Y. J., Reynolds, N., Chambers, D. P., & Sysło, M. M. (2017). Computer science in K-12 school curricula of the 21st century: Why, what and when?. Education and Information Technologies, 22(2), 445-468.

Werner, L., Denner, J., Campe, S., & Kawamoto, D. C. (2012, February). The fairy performance assessment:

measuring computational thinking in middle school. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education* (pp. 215-220). ACM.

Wing, J. M. (2014). *Computational thinking benefits society*, 40th Anniversary Blog of Social Issues in Computing, 2014

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

Wing, J. M. (2011). Computational thinking. In G. Costagliola, A. Ko, A. Cypher, J. Nichols, C. Scaffidi, C. Kelleher, et al. (Eds.), *2011 IEEE symposium on visual languages and human-centric computing* (p. 3).

Wing, J. M., (2008). Computational thinking and thinking about computing. *Philosophical Transactions of The Royal Society of London A: Mathematical, Physical And Engineering Sciences*, *366*(1881), 3717-3725.

Yıldız Durak, H., & Saritepeci, M. (2018). Analysis of the relation between computational thinking skills and various variables with the structural equation model. *Computers & Education*, *116*, 191-202.

Yunkül, E., Durak, G., Çankaya, S., & Mısırlı, Z.A. (2017). The effects of Scratch software on students' computational thinking skills. *Necatibey Eğitim Fakültesi Fen ve Matematik Eğitimi Dergisi, 11(*2), 502-517.

Volume 3, No: 4

April 2020

ISSN 2513-8359