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

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.

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 them and fix them? (Uses Scratch artefact).	**Debugging/testing** (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.

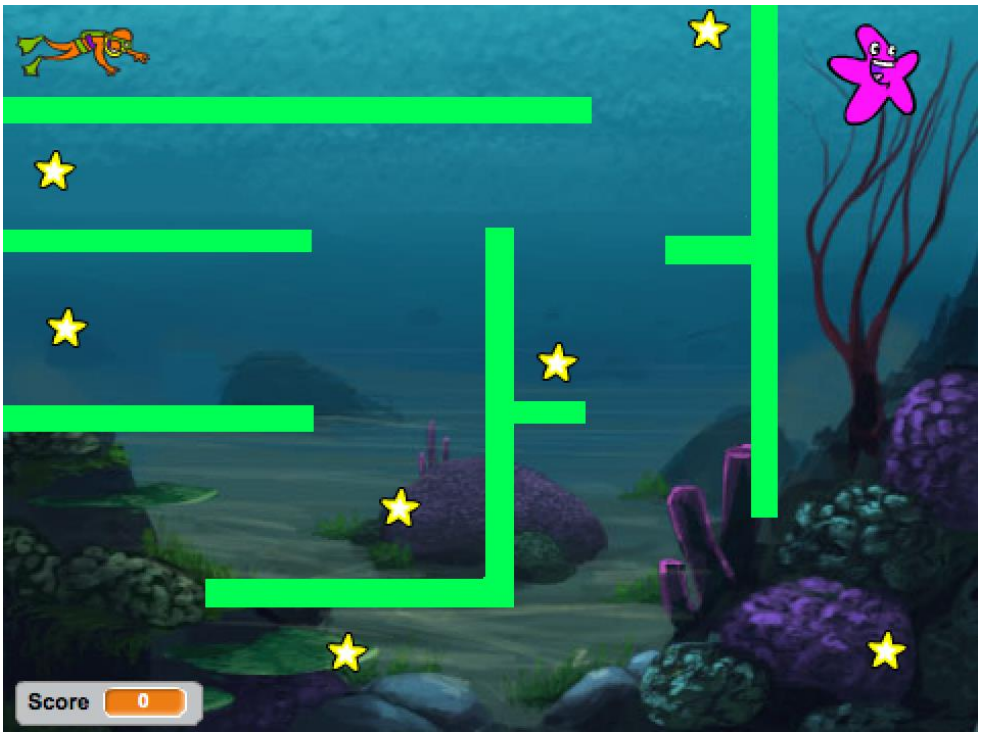

Figure 2. Screen Shot of the Scratch Maze Game Used In Questions 8-11

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:	Some found important details, and	Some found important details, and
a) Abstraction	some talked about all parts.	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. (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