Chapter 8
Components and Methods of Evaluating Computational Thinking for Fostering Creative Problem-Solvers in Senior Primary School Education

Siu-Cheung Kong

Abstract The challenge of introducing Computation Thinking (CT) education to K-12 is how to evaluate learners' CT development. This study used information from previous studies to identify essential components and methods for evaluation. A review of literature from 2010 onwards aimed to identify all studies related to CT evaluation to contribute to the development of appropriate evaluation components and methods for senior primary school education. To facilitate the design of a comprehensive evaluation approach, the CT framework of concepts, practices and perspectives developed by Brennan and Resnick (2012) was adopted in this study. This study has two highlights: (1) problem formulation is proposed to be included as a component of CT practices and (2) CT perspectives are refined to include computational identity and programming empowerment as components. CT concepts could be evaluated by test designs with multiple choice questions to identify students' learning outcomes. CT practices could be evaluated by designing rubrics to rate programming projects and using test designs with task-based questions to identify learners' learning outcomes. Finally, CT perspectives could be measured by designing survey instruments and administrating surveys to learners.

Keywords Computational thinking · Creative problem-solver · Evaluation · Senior primary school education · Visual programming

8.1 Introduction

To keep pace with the Fourth Industrial Revolution, which has integrated physical, digital and biological spheres in all aspects of life (World Economic Forum, 2016), it is necessary to nurture the next generation to become creative problem-solvers in the digital era. Computational Thinking (CT) is widely accepted as a fundamental practice for equipping young people to formulate and solve problems in the dig-

S.-C. Kong (✉)
Centre for Learning, Teaching and Technology, The Education University of Hong Kong, 10 Lo Ping Road, Tai Po, Hong Kong
e-mail: sckong@eduhk.hk

S.-C. Kong and H. Abelson (eds.), *Computational Thinking Education*,
https://doi.org/10.1007/978-981-13-6528-7_8

ital world (Computer Science Teachers Association, 2011), and it is unsurprising that many scholars have called for the integration of CT into K-12 education (e.g. Barr & Stephenson, 2011; Grover & Pea, 2013; Lye & Koh, 2014). To successfully implement CT education in schools, designing appropriate assessing components and methods for evaluating learners is of paramount importance. The National Research Council (2011) pointed out that the purposes of assessing learners' CT development are to evaluate the curriculum, teaching materials and pedagogy, to judge individuals' progress and to manage teacher development and support. Educators are responsible for determining what should be measured and what methods should be used to measure learners' CT development (Duncan & Bell, 2015). Although researchers have explored evaluation components and methods for young learners in recent years, there is no consensus on which are most effective (e.g. Brennan & Resnick, 2012; Grover, Pea, & Cooper, 2015; Zhong, Wang, Chen, & Li, 2016). There is a pressing need to clarify the evaluation components and to propose appropriate approaches and methods to measure learners' CT development.

This study used information from previous studies to identify and propose essential components and methods for evaluating senior primary school learners' CT development. In terms of CT concepts, proposed components to be evaluated include loops, conditionals, sequences, parallelism, data structures, mathematics operators, functions and Boolean logic, event handling, procedures and initialisation. These can be measured by test designs with multiple choice questions to identify students' learning outcomes. In terms of CT practices, proposed components include problem formulating, planning and designing, abstracting and modularising, algorithmic thinking, reusing and remixing, being iterative and incremental, and testing and debugging; these could be evaluated by designing rubrics to rate programming projects and using test designs with task-based questions to identify learners' learning outcomes. In terms of CT perspectives, proposed components include computational identity, programming empowerment and the perspectives of expressing, connecting and questioning, which could be measured by designing survey instruments and administering surveys to learners.

8.2 Background

8.2.1 Computational Thinking

CT is not a new idea; Papert (1980) advocated that children can learn to think expressively about thinking and can foster procedural thinking through programming. He argued that 'computational technology and computational ideas can provide children with new possibilities for learning, thinking, and growing emotionally as well as cognitively' (Papert, 1980, pp. 17–18). The idea of CT has gained global awareness and discussion after Wing's introduction in 2006. She suggested that everyone—not only computer scientists—should learn and use CT (Wing, 2006). According to Wing, CT

includes problem-solving, systems design and understanding human behaviour, and it draws on the concepts that are fundamental to computer science. CT is further defined as 'the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent' (Cuny, Snyder, & Wing, 2010, p. 1). Aho (2012) echoed the viewpoint that CT refers to problem formulation and that solutions can be represented as computational steps and algorithms. In sum, CT is a process of formulating and solving computational problems.

Because CT helps equip the next generation with problem-solving skills, a number of scholars have called for the integration of CT into K-12 education (e.g. Barr & Stephenson, 2011; Grover & Pea, 2013; Lye & Koh, 2014). Researchers have varying ideas with respect to K-12 curriculum because of their diverse interpretations of CT. One viewpoint is that CT should be promoted by means of programming courses because it can be incited by programming knowledge (Araujo, Andrade, & Guerrero, 2016) and learners can demonstrate their CT practices through programming tools. For example, Grover and Pea (2013) introduced nine essential components of CT practices: abstraction and pattern generalisations; systematic processing of information; symbol systems and representations; algorithmic notions of flow of control; structured problem decomposition; iterative, recursive and parallel thinking; conditional logic; efficiency and performance constraints and debugging and systematic error detection. Another viewpoint is that CT practices should be elicited by other subjects and disciplines. For instance, the International Society for Technology in Education (ISTE) emphasises data representations and working with automated solutions and simulations (Barr, Harrison, & Conery, 2011). ISTE's framework does not emphasise the building of artefacts through programming environments, as learners can display CT practices through data representations and can work with automated solutions and simulations (Lye & Koh, 2014). For a CT curriculum that adopts the constructionism approach (Papert, 1980), the acquisition of programming concepts and practices through programming is considered as the most effective way to learn CT. This study adopts this approach to propose measures to evaluate learners' understanding of CT development instead of using ISTE's framework.

8.2.2 The Adopted Framework for Computational Thinking Evaluation

This study adopted a CT framework in the context of programming proposed by Brennan and Resnick (2012) as the evaluation framework. The framework consists of three dimensions: CT concepts, CT practices and CT perspectives. The dimension of CT concepts refers to the computational concepts that learners develop in programming, the dimension of CT practices refers to the problem-solving practices that learners demonstrate repeatedly in the programming process and the dimension of CT perspectives refers to learners' understanding of themselves and their relationships

to others and the technological world that they develop by expressing, connecting and questioning during programming. According to the National Research Council (2012), in the twenty-first century, it is important to assess a person's competence in the cognitive, intrapersonal and interpersonal domains. CT concepts and practices represent the cognitive domain, such as learners' programming knowledge and practices, while CT perspectives represent the intrapersonal and interpersonal domains, such as learners' expression using programming tools and connection to the digitalised world. Lye and Koh (2014) pointed out that most studies focused on CT concepts, and fewer on practices and perspectives. This framework helps to provide a comprehensive picture for evaluating CT development in schools.

This framework is also appropriate for young learners to begin learning CT. In Piaget's model of the four stages of development, learners in the 7–11 age groups are developing operational thinking (Piaget, 1972). Thus, senior primary school learners in the 9–11 age groups are at the suitable starting point to learn basic programming knowledge. As the context of this framework is in visual programming environments such as Scratch and App Inventor, it enables learners to 'learn programming concepts and practice CT abilities while avoiding the syntax hurdle associated with text-based programming languages' (Araujo et al., 2016, p. 6). Within this context, Brennan and Resnick's framework facilitates the design of evaluation components and methods to capture the progress and achievements of learners studying CT in senior primary education.

8.3 Methodology

To facilitate the promotion of CT education in K-12 schools, it is necessary to conduct a comprehensive review of literature to identify the evaluation components and methods of CT for young learners. To achieve the aim of this study, two research questions are raised: (1) What evaluation components should be included in the CT concepts, practices and perspectives dimensions according to the CT framework proposed by Brennan and Resnick (2012) in K-12? (2) What methods are appropriate for evaluating the CT components in each dimension in this context? As each study has put forth its own components and methods for evaluating learners' CT development, this study used information from previous studies to identify the essential components and methods for evaluation. This review of the literature from 2010 to the present aimed to identify every study related to programming and CT evaluation to contribute to the proposal of appropriate evaluation components and methods for senior primary school education. Table 8.1 summarises the 24 studies reviewed in this study.

Table 8.1 List of 24 studies reviewed in this study

	Study	Dim[a]		Study	Dim[a]		Study	Dim[a]
1.	Brennan and Resnick (2012)	1, 2, 3	9.	Grover, Cooper, and Pea (2014)	1, 2	17.	Ruf, Mühling, and Hubwieser (2014)	1, 3
2.	Burke (2012)	1, 2	10.	Grover et al. (2015)	1, 2, 3	18.	Seiter and Foreman (2013)	1, 2
3.	Denner, Werner, Campe, and Ortiz (2014)	2, 3	11.	Kukul and Gökçearslan (2017)	3	19.	Sherman and Martin (2015)	1, 2
4.	Duncan and Bell (2015)	2, 3	12.	Maguire, Maguire, Hyland, and Marshall (2014)	3	20.	Werner, Denner, Campe, and Kawamoto (2012)	2, 3
5.	Ericson and McKlin (2012)	1, 3	13.	Meerbaum-Salant, Armoni, and Ben-Ari (2013)	1	21.	Wilson, Hainey, and Connolly (2012)	1
6.	Fessakis, Gouli, and Mavroudi (2013)	2	14.	Mueller, Beckett, Hennessey, and Shodiev (2017)	2	22.	Wolz, Stone, Pearson, Pulimood, and Switzer (2011)	3
7.	Giordano and Maiorana (2014)	1, 3	15.	Rodriguez, Kennicutt, Rader, and Camp (2017)	2	23.	Zhong et al. (2016)	1, 2
8.	Gouws, Bradshaw, and Wentworth (2013)	2	16.	Román-González, Pérez-González, and Jiménez-Fernández (2016)	1, 2	24.	Zur-Bargury, Pârv, and Lanzb erg (2013)	1

[a]The numbers refer to the dimensions discussed in the reviewed literature, according to the CT framework proposed by Brennan and Resnick (2012). '1' refers to CT concepts, '2' refers to CT practices and '3' refers to CT perspectives

8.4 Results and Discussion Based on Literature Review

Grover et al. (2015, p. 199) wrote that 'a system of assessments is beneficial to get a comprehensive picture of learners' CT learning.' In other words, there is no single evaluation component or method that can entirely measure learners' CT achievements. Researchers, therefore, strive to develop both quantitative and qualitative approaches to evaluate learners' learning outcomes. This section sheds light on the essential evaluation components and assessment methods of each of the three dimensions proposed by Brennan and Resnick (2012): CT concepts, CT practices and CT perspectives.

8.4.1 CT Concepts

8.4.1.1 Literature Review Results

The 14 studies on evaluating programming concepts in block-based environments provided 9 CT concepts to be included: (1) repetition/loops/iteration, (2) conditionals, (3) data structures, (4) sequences, (5) parallelism/concurrency, (6) mathematical operators, functions and Boolean logic, (7) event handling, (8) procedures and (9) initialisation. Table 8.2 summarises components of CT concepts that were evaluated by past studies and the tabulated results were sorted according to the frequency with which they were discussed.

Brennan and Resnick (2012) pointed out that the first seven concepts in Table 8.2 were useful in the Scratch programming environment and stated that learners should be able to transfer these CT concepts to other programming and non-programming contexts. The reviewed studies generally concurred on the importance of these seven components, in particular, all 14 studies agreed that repetition (or loops or iteration) is an essential component in evaluating CT concepts. Conditionals were identified by 11 studies as a core component of CT concepts. Ten studies put forth data as a key programming concept, which includes important knowledge such as variables and lists. The concept of sequences and parallelism (or concurrency), including sending or passing messages and handling coordination, was mentioned in eight studies. Eight studies also suggested that learners should understand the concepts of mathematical operators such as 'addition, subtraction, multiplication, division, as well as functions like sine and exponents' (Brennan & Resnick, 2012, pp. 5–6) and Boolean logic operations such as 'AND, OR, NOT'. Six studies investigated learners' ability to handle events in programming. In addition, the ability to apply procedures in programming tasks was identified as a component of CT concepts in two studies. It helps to avoid the repetition of codes and duplication of commands (Marji, 2014). One study attempted to evaluate learners' ability to deal with initialisation.

The 14 evaluation studies used both quantitative and qualitative methods to measure learners' understanding of CT concepts. Table 8.3 summarises the methods

Table 8.2 Components of CT concepts in past studies

Component	Study	Frequency
1. Repetition/loops/iteration	Brennan and Resnick (2012), Burke (2012), Ericson and McKlin (2012), Giordano and Maiorana (2014), Grover et al. (2014, 2015), Meerbaum-Salant et al. (2013), Román-González et al. (2016), Ruf et al. (2014), Seiter and Foreman (2013), Sherman & Martin (2015), Wilson et al. (2012), Zhong et al. (2016), Zur-Bargury et al. (2013)	14
2. Conditionals	Brennan and Resnick (2012), Ericson and McKlin (2012), Giordano and Maiorana (2014), Grover et al. (2014, 2015), Román-González et al. (2016), Ruf et al. (2014), Seiter and Foreman (2013), Sherman and Martin (2015), Wilson et al. (2012), Zur-Bargury et al. (2013)	11
3. Data structures (e.g. variables, lists)	Brennan and Resnick (2012), Ericson and McKlin (2012), Giordano and Maiorana (2014), Grover et al. (2014, 2015), Meerbaum-Salant et al. (2013), Seiter and Foreman (2013), Sherman & Martin (2015), Wilson et al. (2012), Zhong et al. (2016)	10
4. Sequences	Brennan and Resnick (2012), Giordano and Maiorana (2014), Grover et al. (2014, 2015), Román-González et al. (2016), Zhong et al. (2016), Seiter and Foreman (2013), Wilson et al. (2012)	8
5. Parallelism/concurrency (e.g. messages, coordination)	Brennan and Resnick (2012), Burke (2012), Ericson and McKlin (2012), Giordano and Maiorana (2014), Meerbaum-Salant et al. (2013), Seiter and Foreman (2013), Wilson et al. (2012), Zhong et al. (2016)	8
6. Mathematical operators, functions and Boolean logic	Brennan and Resnick (2012), Giordano and Maiorana (2014), Grover et al. (2014, 2015), Román-González et al. (2016), Seiter and Foreman (2013), Wilson et al. (2012), Zhong et al. (2016)	8
7. Event handling	Brennan and Resnick (2012), Burke (2012), Ericson and McKlin (2012), Giordano and Maiorana (2014), Seiter and Foreman (2013), Wilson et al. (2012)	6
8. Procedures	Giordano and Maiorana (2014), Seiter and Foreman (2013)	2
9. Initialisation	Meerbaum-Salant et al. (2013)	1

Table 8.3 Methods used by studies to evaluate CT concepts

Method	Study	Frequency
1. Test designs with multiple choice type questions in programming context	Ericson and McKlin (2012), Giordano and Maiorana (2014), Grover et al. (2014, 2015), Meerbaum-Salant et al. (2013), Román-González et al. (2016), Ruf et al. (2014), Zur-Bargury et al. (2013)	8
2. Task/project rubrics	Seiter and Foreman (2013), Sherman and Martin (2015), Wilson et al. (2012), Zhong et al. (2016)	4
3. Interviews	Brennan and Resnick (2012), Grover et al. (2014, 2015), Meerbaum-Salant et al. (2013)	4
4. Project analysis	Brennan and Resnick (2012), Burke (2012)	2
5. Observations	Burke (2012), Meerbaum-Salant et al. (2013)	2
6. Reflection reports	Zhong et al. (2016)	1

by studies used to evaluate CT concepts. The tabulated results are sorted according to the frequencies with which the methods were used. In terms of quantitative approaches, most of the studies measured learners' understanding of CT concepts using test items, most of which were designed with multiple choice questions in the programming context (Ericson & McKlin, 2012; Ruf, Mühling, & Hubwieser, 2014; Zur-Bargury et al., 2013). Task and project rubrics were also commonly used so that teachers could assess learners' project work with scoring guidelines.

Among the studies using qualitative approaches, interviews were the most common method of evaluating CT concepts. Interviews are conducted to understand the CT concepts that learners use to complete programming tasks or projects. Project analysis was another method used to evaluate learners' development in CT concepts. Evaluators used the Scrape tool to analyse the programming blocks of projects. Scrape can provide a record of the CT concepts that the learner utilised in his/her projects (Brennan & Resnick, 2012; Burke, 2012). Studies also observed programming lessons to investigate whether learners could correctly handle the CT concepts in programming activities. One study asked learners to write reflective reports on what kinds of CT concepts they used to complete their programming tasks and projects (Zhong et al., 2016).

8.4.1.2 Proposed Evaluation Components and Methods

Duncan and Bell (2015) analysed the computing curricula for primary school in England and recommended that key stage 2 learners in the 7–11 year age group acquire

Table 8.4 Proposed evaluation components of CT concepts at the primary school level

Component of CT concepts
1. Loops
2. Conditionals
3. Sequences
4. Parallelism
5. Data structures such as variables and lists
6. Mathematics operators, functions and Boolean operators
7. Event handling
8. Procedures
9. Initialisation

concepts such as loops, conditionals, sequences and variables. One study found that loops, conditionals, variables and operators were the fundamental programming concepts used by children in the Scratch programming environment (Maloney, Peppler, Kafai, Resnick, & Rusk, 2008). Wilson et al. (2012) found that over half of children's game-making projects used concepts such as loops, conditionals, sequences and parallelism. Because event handling is commonly used in block-based programming environments such as Scratch and App Inventor, the concept of event handling should be included (Brennan & Resnick, 2012). Although some may believe that procedure and initialisation are difficult for senior primary learners, they are essential concepts in programming. Procedure can avoid the repetition of codes and duplication of commands (Marji, 2014), and initialisation helps a programme reach its initial state. Therefore, if teachers introduce procedure and initialisation in class, they can consider assessing learners' abilities regarding these two concepts. In line with the above discussion and the results tabulated in Table 8.2, the following CT concepts should be evaluated in the K-12 CT curriculum: (1) loops, (2) conditionals, (3) sequences, (4) parallelism, (5) data structures such as variables and lists, (6) mathematical operators, functions and Boolean operators, (7) event handling, (8) procedures and (9) initialisation. Table 8.4 summarises the proposed CT concepts at the primary school level.

Because CT concepts are made up of different kinds of components, they can be evaluated by an aggregation of learners' achievement across components. Therefore, CT concepts can be evaluated by designing quantitative test instruments with itemised pieces that each evaluates a particular component. Evaluators can use these test instruments to measure the learning outcomes of learners and to administer pre- and post-test designs to measure learners' progression. Evaluators can also assess whether learners apply CT concepts appropriately by looking into their tasks/projects with the rubrics. Quantitative data can thus be obtained by accumulating the scores of learners generated from the rubrics. It is proposed that test designs with multiple choice questions be used to measure learners' progression for large numbers of learners. Evaluators can show programming segments in the test items and ask questions related to the segments, such as finding the output from the programming segment,

Please read the blocks below. If the answer is 'No', how will the programme respond?

A. Say "Great!"
B. Say "Cheer up!"
C. I don't know
D. No response
E. An error will occur

Fig. 8.1 An example of testing the CT concept of conditionals in the form of a multiple choice question

which can demonstrate learners' understanding of CT concepts. Figure 8.1 shows an example of testing conditionals in the form of a multiple choice question.

CT concepts can also be evaluated by qualitative methods such as interviews, project analyses, observations and reflection reports. Researchers found that when learners were asked about how parts of their code work, their explanations showed large conceptual gaps (Grover et al., 2014). Therefore, interviews with learners can show how they explain their work. It is also suggested that evaluators can evaluate learners' CT concepts by using automated tools to analyse 'the number, the range, and the frequency' of different programming blocks being used in their projects (Burke, 2012). These blocks could provide information on learners' development in CT concepts. In addition, classroom observations can enable evaluators to determine how learners use CT concepts to tackle computational tasks. Asking primary school learners to write simple reflective reports on what kinds of CT concepts they used in completing their programming tasks and projects is another alternative. Because qualitative approaches require time and effort to explore learners' learning outcomes, they should be regarded as supplementary for evaluators seeking to comprehend the CT concepts of learners. Table 8.4 summarises the proposed evaluation components of the CT concepts at the primary school level.

Table 8.5 Components of CT practices proposed by research studies

Component	Study	Frequency
1. Abstraction/abstracting, modelling/abstracting and modularising	Brennan and Resnick (2012), Denner et al. (2014), Gouws et al. (2013), Grover et al. (2015), Mueller et al. (2017), Rodriguez et al. (2017), Seiter and Foreman (2013), Sherman and Martin (2015), Werner et al. (2012), Zhong et al. (2016)	10
2. Algorithmic thinking	Denner et al. (2014), Duncan and Bell (2015), Gouws et al. (2013), Mueller et al. (2017), Rodriguez et al. (2017), Seiter and Foreman (2013), Werner et al. (2012)	7
3. Testing and debugging	Brennan and Resnick (2012), Burke (2012), Fessakis et al. (2013), Grover et al. (2015), Mueller et al. (2017), Román-González et al. (2016), Zhong et al. (2016)	7
4. Being incremental and iterative	Brennan and Resnick (2012), Grover et al. (2015), Mueller et al. (2017), Zhong et al. (2016)	4
5. Problem decomposition	Grover et al. (2015, 2014), Mueller et al. (2017), Seiter & Foreman (2013)	4
6. Planning and designing	Burke (2012), Zhong et al. (2016)	2
7. Reusing and remixing	Brennan and Resnick (2012), Mueller et al. (2017)	2

8.4.2 CT Practices

8.4.2.1 Literature Review Results

Fifteen studies in the literature provide seven CT practices: (1) abstraction/abstracting and modelling/abstracting and modularising, (2) algorithmic thinking, (3) testing and debugging, (4) being incremental and iterative, (5) problem decomposition, (6) planning and designing and (7) reusing and remixing. Table 8.5 summarises the components of CT practices identified by past studies, and the tabulated results were sorted according to the frequency with which they were discussed.

Abstraction is one of the most fundamental practices in CT (Wing, 2006). Ten studies are related to learners' ability to think abstractly. Brennan and Resnick (2012, p. 9) suggested that modularisation ability should be merged with abstraction so that learners can avoid complexity and 'build something large by putting together collections of smaller parts'. Modelling, another skill that is often linked with abstracting, allows learners to 'organize data, structure their thoughts, describe relationships, and analyse proposed designs' (Voland, 1999, p. 215). Seven of the studies regarded algo-

rithmic thinking as the foundation of CT practices. Learners are required to define the steps and develop instructions to solve a problem. Seven studies also suggested that testing and debugging are the core of CT practices because it is 'rare for a program to work correctly the first time it is tried, and often several attempts must be made before all errors are eliminated' (Peterson, 2002, p. 93). Four past studies suggested that problem decomposition and the practices of being incremental and iterative are indispensable in the programming process. Learners should learn to repeatedly 'develop a little bit, then try it out, then develop more' until the programme is complete (Brennan & Resnick, 2012, p. 7). Problem decomposition involves breaking down problems into smaller, more manageable tasks (Waller, 2016). Several researchers found that planning and designing were part of CT practices. They were used to investigate whether learners plan their solutions before they write the code or use trial and error during programming. Additionally, researchers noted that the reuse and remix of the works of other programmers are crucial in the online communities of Scratch and Alice (Brennan & Resnick, 2012), and they encouraged novices to produce more complicated creations by building on existing projects or ideas (Brennan & Resnick, 2012).

For evaluation methods, because of the difficulties in finding out how learners tackle problems during programming, evaluators use both quantitative and qualitative methods to measure learners' CT practices. Table 8.6 summarises the methods adopted by the studies to evaluate CT practices. The tabulated results are sorted by the frequency with which the methods were used. Seven studies used task/project rubrics to evaluate learners' CT practices, as teachers marked the programming outcomes of tasks or projects based on rubrics to evaluate the CT practices of learners. Four projects used tests with task-based questions in coding and non-coding contexts to assess learners' proficiency in CT practices. The qualitative methods proposed are favourable for demonstrating their learning process. Four studies proposed the use of interviews to understand learners' CT practices, as interviews enable teachers to understand learners' thoughts behind the code (Grover & Pea, 2013). Two studies proposed the use of observations: researchers observe in classrooms, take notes and videorecord the entire programming process to understand learners' CT practices (Grover & Pea, 2013). One study used learners' reflection reports to understand their programming practices. Reflection reports are self-assessments that are distributed to learners after they finish a task or a project (Zhong et al., 2016), in which they write out how they accomplished their problem-solving tasks.

8.4.2.2 Proposed Evaluation Components and Methods

Programming is not a simple process, and it might not have a clear order. However, the components of CT practices can be divided into two categories: design practices and programming practices. In the design practices category, learners design programmes. This category includes three practices: problem decomposition, abstracting and modularising, and algorithmic thinking. Because people often perceive planning as a 'highly specialized skill for solving problems' that requires scarce resources

Table 8.6 Methods adopted by studies to evaluate CT practices

Method	Study	Frequency
1. Task/project rubrics	Denner et al. (2014), Rodriguez et al. (2017), Román-González et al. (2016), Seiter and Foreman (2013), Sherman and Martin (2015), Werner et al. (2012), Zhong et al. (2016)	7
2. Tests designed with task-based questions	Duncan and Bell (2015), Gouws et al. (2013), Grover et al. (2014, 2015)	4
3. Interviews	Brennan and Resnick (2012), Grover et al. (2014, 2015), Mueller et al. (2017)	4
4. Observations	Burke (2012), Fessakis et al. (2013)	2
5. Reflection reports	Zhong et al. (2016)	1

(Lawler, 1997), planning and designing are not recommended for inclusion in the programming process. Learners are supposed to decompose problems into sub-tasks first (Waller, 2016). The practice of modularising is recommended to be merged with abstracting. Learners will be able to connect the parts of the whole so that they can test and debug different parts of the programme incrementally (Brennan & Resnick, 2012). The Center for Computational Thinking, Carnegie Mellon (2010) asserted that algorithmic thinking is essential, as it helps learners to produce efficient, fair and secure solutions. In sum, in line with the results in Table 8.5, problem decomposition, abstracting and modularising, and algorithmic thinking are essential CT practices for evaluation at this stage.

In the programming practices category, learners implement their designs to produce a concrete programme artefact. This stage includes three practices: reusing and remixing, being iterative and incremental, and testing and debugging. The first practice helps novices create their own programmes by building on others' works and developing their own works incrementally (Brennan & Resnick, 2012). The practice of being iterative and incremental is tied to testing and debugging: programmers have to develop part of the programme and test it to ensure that it works; they then repeat these steps and continue to develop the programme. In sum, in line with the results in Table 8.5, reusing and remixing, being iterative and incremental, and testing and debugging are essential CT practices for evaluation at this stage.

Problem formulating is proposed to be included in the CT practices component in the study of this paper. Although Cuny et al. (2010, p. 1) defined CT as the 'thought processes involved in formulating problems and their solutions', none of the reviewed studies proposed assessing learners' abilities in problem formulation. In the early twentieth century, Einstein argued that raising questions was more important than solving problems. In his classic study, *The Evolution of Physics*, he wrote, 'the formulation of a problem is often more essential than its solution, which may be merely a matter of mathematical or experimental skill. To raise new questions, new possibilities, to regard old problems from a new angle require creative imagina-

Table 8.7 Proposed evaluation components of CT practices at the primary school level

Component of CT practices
1. Problem formulating
2. Problem decomposition
3. Abstracting and modularising
4. Algorithmic thinking
5. Reusing and remixing
6. Being iterative and incremental
7. Testing and debugging

tion and marks real advances in science' (Einstein & Infeld, 1938, p. 92). Because learners' creativity can be demonstrated by formulating creative problems, there is a need to add this component to CT practices to produce creative problem-solvers in the digitalised world. The aims of CT practices are thus to develop both creative thinkers and problem-solvers (Brugman, 1991). Therefore, this study proposes to include 'problem formulating' as one of the components of CT practices. Table 8.7 summarises the proposed evaluation components of the CT practices of this study: (1) problem formulating, (2) problem decomposition, (3) abstracting and modularising, (4) algorithmic thinking, (5) reusing and remixing, (6) being iterative and incremental and (7) testing and debugging.

Like CT concepts, CT practices involve several components; these can be measured by an aggregation of learners' achievement across components. One quantitative method to evaluate CT practices is designing rubrics to assess the final programming projects of learners. The criteria of the rubrics are the components of CT practices, and there is a need to design descriptors of performance levels for each CT practice. These rubrics help evaluators to review learners' abilities to formulate and solve problems. In addition to using rubrics to assess the CT practices of learners, SRI International proposed a task-based approach to assess CT practices. SRI International also published a report titled 'Assessment Design Patterns for Computational Thinking Practices in Secondary Computer Science' (Bienkowski, Snow, Rutstein, & Grover, 2015), which presented an overview of four CT practice design patterns and two supporting design patterns. The CT practice design patterns include analysing the effects of developments in computing, designing and implementing creative solutions and artefacts, designing and applying abstractions and models, and supporting design patterns analysing learners' computational work and the work of others. SRI International developed focal knowledge, skills and other attributes for each CT practice, which illustrate the core abilities that learners need to acquire and what should be assessed. Its report provided direction for evaluating CT practices using the task-based approach. In this approach, evaluators' first step is to define the fundamental abilities of each CT practice. The next step is to design tasks that include a scenario and to ask questions about the scenarios to test learners' abilities related to CT practice. Learners need to understand the scenarios and to answer questions about selecting options, organising sequences of activities or performing

1. Scenario

An app developer developed a mobile location tracking app that can show the location of a place on a map using latitude and longitude. However, a user reported that the app does not show the location correctly on the map. Please identify the reason(s) why the error occurred.

- ☐ (a) The mobile app developer was unable to find the correct latitude and longitude of the location
- ☐ (b) The mobile app developer was unable to mark the location correctly on the map
- ☐ (c) The mobile app developer committed both mistakes (a) and (b)

Fig. 8.2 An example of testing the CT practice of testing and debugging using a task with a scenario

categorisations to demonstrate their abilities. For example, one of the focal skills of the CT practice of testing and debugging practice is the ability to explain the cause of an error. Figure 8.2 demonstrates an example of testing learners' understanding of testing and debugging using a task with a scenario.

With this approach, tests with sets of task-based questions can be designed to evaluate learners' abilities with the various components of CT practices. Pre- and post-test designs with task-based questions can enable evaluators to identify learners' progression in CT practices.

Measuring learners' CT practices by directly analysing their processes of designing and building programmes requires great effort. Qualitative approaches such as observing learners in designing and building programmes are one such method. Another is to interview them after they formulate, design and build their programmes and to ask them semi-structured questions. It is difficult to ask primary school learners to write detailed reflection reports on how they formulate, design and build their programmes; however, evaluators can ask learners to write simple reflection reports on how they can improve their programmes after solving their computational tasks. These qualitative methods can serve as supplements for evaluators seeking to understand what practices learners used in their programming tasks. Table 8.7 summarises the proposed evaluation components of CT practices at the primary school level.

8.4.3 CT Perspectives

8.4.3.1 Literature Review Results

The literature review indicated not only that CT concepts and practices are of paramount importance in measuring learners' CT development, but learners' perspectives on learning programming are also vital. Brennan and Resnick (2012) proposed three kinds of perspectives in the programming process—expressing, connecting and questioning—to investigate learners' understanding of themselves and their relationships to others and the technological world. Based on the literature review, CT perspectives include not only Brennan and Resnick's (2012) suggestions

Table 8.8 Components of CT perspectives proposed to be evaluated by research studies

Component	Study	Frequency
1. Attitudes such as interest in programming and computing	Denner et al. (2014), Duncan and Bell (2015), Ericson and McKlin (2012), Giordano and Maiorana (2014), Grover et al. (2015), Maguire et al. (2014), Ruf et al. (2014), Werner et al. (2012), Wolz et al. (2011)	9
2. Confidence in programming and computing, programming self-efficacy and programmers	Denner et al. (2014), Giordano and Maiorana (2014), Kukul and Gökçearslan (2017), Maguire et al. (2014), Werner et al. (2012), Wolz et al. (2011)	6
3. Expressing, connecting and questioning	Brennan and Resnick (2012)	1

(i.e. expressing, connecting and questioning) but also learners' motivation beliefs, namely, value and expectancy (Wigfield & Eccles, 2000; Chiu & Zeng, 2008), in their understanding of themselves. Value refers to learners' intrinsic motivations, such as their attitudes towards and interest in learning programming. Expectancy refers to learners' programming confidence, which includes their programming self-efficacy and self-concept. Programming self-efficacy is 'people's judgments of their capabilities to organize and execute courses of action required to attain designated types of performance' in the context of programming (Bandura, 1986, p. 391; Kong, 2017), while programming self-concept is the belief in one's programming competence (Chiu & Zeng, 2008). Previous studies indicated that researchers investigated both learners' motivation beliefs regarding learning programming and their perspectives on the technological world.

The 10 related studies in the literature identify three components of CT perspectives: (1) attitudes such as interest in programming and computing, (2) confidence in programming and computing, programming self-efficacy and programming competence and (3) expressing, connecting and questioning. Table 8.8 summarises the components of CT perspectives proposed to be evaluated by previous studies. The tabulated results are sorted according to the frequency with which they were discussed.

Most researchers agree that learners' attitudes, such as their interest in programming, should be included in this evaluation dimension. Researchers have focused on learners' programming confidence, self-efficacy and competence, and most have used quantitative instruments to measure learners' CT perspectives, although some have used qualitative data to analyse their attitudes. Table 8.9 summarises the methods adopted by studies to evaluate CT perspectives. Surveys are commonly used, with five-point Likert scales used most frequently (e.g. Ericson & McKlin, 2012; Ruf et al., 2014). Conducting surveys before and after a learning programming might facilitate the investigation of learners' CT perspectives. Qualitative methods such

Table 8.9 Methods adopted by studies to evaluate CT perspectives

Method	Study	Frequency
1. Survey	Denner et al. (2014), Duncan and Bell (2015), Ericson and McKlin (2012), Giordano and Maiorana (2014), Grover et al. (2015), Kukul and Gökçearslan (2017), Maguire et al. (2014), Ruf et al. (2014), Werner et al. (2012), Wolz et al. (2011)	10
2. Interview	Brennan and Resnick (2012)	1

as interviews can provide a more in-depth understanding of learners' perspectives towards programming because programming courses and related activities can be reviewed.

8.4.3.2 Proposed Evaluation Components and Methods

To fully understand learners' intrinsic values regarding programming, asking whether they like or dislike programming is far from sufficient. Deci and Ryan (1985, p. 34) argued that 'when people are intrinsically motivated, they experience interest and enjoyment, they feel competent and self-determined'. Learners' interest in programming can be assessed by their engagement in programming activities, sense of affiliation and career orientation. In addition to their attitudes, self-actualisation indicates whether learners feel competent after learning programming. These four factors can be used to explore learners' computational identity, which they form through networking and sharing knowledge and experience in programming activities.

Wenger (1998, pp. 4–5) argued that engagement is 'the most immediate relation to a practice-engaging in activities, doing things, working alone or together, talking, using and producing artifacts', and that it creates 'an identity of participation or non-participation'. When learners are more interested in programming tasks, their engagement in activities is deeper. Learners' self-formation through engagement is thus a critical factor that can directly reflect their interest in and enjoyment of learning programming. Measuring learners' sense of affiliation can also indicate whether they have a feeling of belongingness regarding their interest in programming. They can then develop a computational identity by the 'participation of sharing' (Gee, 2000).

It is believed that learners' interests affect their career choices. The construct of career orientation explores learners' motivation to work with people who share the same interests. Learners' self-actualisation is another crucial subcomponent. According to Maslow (1943, p. 382), self-actualisation is 'the desire for self-fulfilment, namely, to the tendency for one to become actualized in what he is potentially'. In other words, it is the development of competencies in knowledge, attitudes, skills and character (Huitt, 2007). In sum, these factors can be used to extensively investigate learners' intrinsic motivation, especially in their programming interest and competence. Thus, computational identity is proposed to be measured as the first

Table 8.10 Proposed evaluation components of CT perspectives at the primary school level

Component of CT perspectives
1. Computational identity
2. Programming empowerment
3. Perspectives of expressing, connecting and questioning

component of CT perspectives in this study. If learners have positive perceptions regarding these four factors, then they have a strong computational identity.

Concerning expectancy, programming empowerment, the second component of the CT perspectives proposed in this study, can help evaluate learners' programming confidence more comprehensively. Borrowed from the concept of digital empowerment (Makinen, 2006), programming empowerment can be defined as a person's experiences creating and designing programmes that enable them to tackle real-life problems and confidently participate in the digital world. Learners' belief in their competence to acquire the necessary concepts and practices to handle programming tasks is the primary factor that evaluators must measure. Programming empowerment also emphasises the meaningfulness, impact and creativity of programming, where meaningfulness refers to a person's perceived value—that is, the importance of their programming—and impact is a person's perception of the influence of the programming. Because this study proposes evaluation components to foster creative problem-solvers, creativity can be used to measure learners' beliefs in their ability to produce new ideas in the digitalised world through programming and related digital technologies. In sum, learners will contribute confidently in the digital world if they feel empowered concerning their programming experiences.

The third component is a broader horizon that stresses learners' relationships with others and the technological world. It explores how learners use their innovative thinking after learning programming to contribute to society through expressing, connecting and questioning the digitalised world (Brennen & Resnick, 2012). Expressing means learners' abilities to express themselves by creating with programming. Connecting means learners' understanding of the value of creating with and for others by programming, and it highlights the linkage between programming and real life. Questioning means that learners are empowered to ask questions about the technological world based on their programming experiences. It also means that learners are empowered to formulate problems in the computational context. The idea of questioning could also be expanded to formulate problems before programming. Learners are encouraged to think of questions that could be solved computationally.

As mentioned, CT perspectives represent a person's competence in intrapersonal and interpersonal domains. Therefore, this study proposed the evaluation of learners' (1) computational identity, (2) programming empowerment and (3) perspectives of expressing, connecting and questioning. Table 8.10 shows the proposed evaluation components of the CT perspectives of this study.

It is suggested that CT perspectives can be evaluated by well-designed survey instruments. This study proposed the design of survey instruments to measure

	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
Computational Identity					
1. I feel associated with my classmates when participating in computer programming activities with them.	○	○	○	○	○
2. Learning computer programming with my classmates gives me a strong sense of belonging.	○	○	○	○	○
Programming Empowerment					
1. I have confidence in my ability to use digital technologies.	○	○	○	○	○
2. I can solve problems with digital technologies.	○	○	○	○	○
Perspectives of expressing, connecting, and questioning					
1. I feel excited to express new ideas through programming.	○	○	○	○	○
2. I think carefully about potential problems in the process of programming.	○	○	○	○	○

Fig. 8.3 Sample items of the computational identity, programming empowerment and perspectives of expressing, connecting and questioning survey instruments

(1) computational identity, (2) programming empowerment and (3) perspectives of expressing, connecting and questioning. These surveys should be conducted at different time points to capture the development of learners' CT perspectives. Learners are required to rate their agreement or disagreement with the statements in these instruments, and it is convenient to analyse a large number of learners' replies using quantitative methods. Figure 8.3 shows sample items for the computational identity, programming empowerment and perspectives of expressing, connecting and questioning survey instruments. Qualitative methods such as interviews can be used to obtain more details on learners' perspectives of programming, but these require time and effort.

8.5 Conclusion

Every country/region requires creative problem-solvers who can contribute to every aspect of life in the digitalised world. CT is a way to cultivate creativity and problem-solving skills in young people. Evaluation is important to facilitate the implementation of CT education in schools because it enables teachers to identify the progress and outcomes of learners and help them attain their learning goals. Based on the literature review, this study proposed nine components of CT concepts, seven components of CT practices and three components of CT perspectives for evaluation.

As the goal of the CT curriculum is to nurture creative problem-solvers, it is essential to evaluate learners' abilities to formulate and solve problems in the computational context. This study has two highlights. First, this study proposed 'problem formulating' as an additional component of evaluating CT practices to encourage learners to raise questions before they programme. Problem formulation is also

closely linked with questioning in CT perspectives, which means that learners are empowered to formulate problems in the computational context. In addition, this study discussed the ideas of computational identity and programming empowerment and proposed them as components for evaluating the perspectives of expressing, connecting and questioning proposed by Brennan and Resnick (2012) to capture learners' motivation beliefs in a more comprehensive manner.

Past studies indicated that no single method could effectively measure learners' CT development in the three dimensions; multiple methods are needed to evaluate learning outcomes. To assess a large number of learners' CT development, quantitative methods are more feasible in terms of resources. This study proposed the design of tests with multiple choice questions to evaluate the CT concepts of learners. Evaluators could also assess learners' CT concepts by analysing their programming tasks and projects with rubrics. Qualitative methods such as interviews, project analyses, observations and simple reflection reports can be conducted as supplements to quantitative approaches. CT practices can be evaluated by measuring programming project outcomes with rubrics and the design of tests with task-based questions. Qualitative methods such as interviews, observations and simple reflection reports can be conducted given the in-depth understanding of a number of learners. CT perspectives can be measured by well-designed survey instruments, and qualitative methods such as interviews can be conducted if in-depth understanding is needed.

The future work of evaluating learners' CT development is to design instruments for measuring primary school learners in these three dimensions. Regarding CT concepts and practices, researchers should design programming project rubrics and test instruments to capture learners' learning outcomes in these areas. Regarding CT perspectives, the constructs of computational identity, programming empowerment and perspectives of expressing, connecting and questioning should be further explored and established. Researchers are recommended to develop and validate instruments for measuring learners' computational identity and programming empowerment, and how they express, connect and question in the digitalised world. The proposed evaluation components and methods will be implemented in senior primary schools when these instruments are developed. With appropriate evaluation components and methods, it is believed that schools will be in a better position to motivate and nurture young learners to become creative problem formulators and problem-solvers in the digitalised world.

References

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

Araujo, A. L., Andrade, W. L., & Guerrero, D. D. (2016, October 12–15). A systematic mapping study on assessing computational thinking abilities. In *Proceedings of the IEEE Frontiers in Education Conference* (pp. 1–9). Erie, PA.

Bandura, A. (1986). *Social foundations of thought and action: A social cognitive theory*. Englewood, Cliffs, NJ: Prentice-Hall.

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? *ACM Inroads, 2*(1), 48–54.

Barr, V., Harrison, J., & Conery, L. (2011). Computational thinking: A digital age skill for everyone. *ISTE Learning & Leading, 38*(6), 20–22.

Bienkowski, M., Snow, E., Rutstein, D. W., & Grover, S. (2015). *Assessment design patterns for computational thinking practices in secondary computer science: A first look* (SRI technical report). Menlo Park, CA: SRI International. Retrieved November 21, 2016, from http://pact.sri.com/resources.html.

Brennan, K., & Resnick, M. (2012, April). New frameworks for studying and assessing the development of computational thinking. In A. F. Ball & C. A. Tyson (Eds.), *Proceedings of the 2012 Annual Meeting of the American Educational Research Association* (25 pp.). Vancouver, Canada: American Educational Research Association.

Brugman, G. M. (1991). Problem finding: Discovering and formulating problems. *European Journal of High Ability, 2*(2), 212–227.

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.

Center for Computational Thinking, Carnegie Mellon. (2010). Retrieved November 15, 2017, from https://www.cs.cmu.edu/~CompThink/.

Chiu, M. M., & Zeng, X. (2008). Family and motivation effects on mathematics achievement: Analyses of students in 41 countries. *Learning and Instruction, 18*(4), 321–336.

Computer Science Teachers Association. (2011). *K-12 computer science standards*. Retrieved November 21, 2016, from http://csta.acm.org/Curriculum/sub/K12Standards.html.

Cuny, J., Snyder, L., & Wing, J. M. (2010). *Demystifying computational thinking for non-computer scientists. Unpublished manuscript in progress*. Retrieved April 16, 2017, from http://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf.

Deci, E. L., & Ryan, R. M. (1985). *Intrinsic motivation and self-determination in human behavior*. New York, NY: Plenum.

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

Duncan, C., & Bell, T. (2015, November 9–11). A pilot computer science and programming course for primary school students. In *Proceedings of the 10th Workshop in Primary and Secondary Computing Education* (pp. 39–48). London, England.

Einstein, A., & Infeld, L. (1938). *The evolution of physics: The growth of ideas from early concepts to relativity and quanta*. New York, NY: Simon and Schuster.

Ericson, B., & McKlin, T. (2012, February 29–March 3). Effective and sustainable computing summer camps. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (pp. 289–294). Raleigh, NC.

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.

Gee, J. P. (2000). Identity as an analytic lens for research in education. *Review of Research in Education, 25,* 99–125.

Giordano, D., & Maiorana, F. (2014, April 3–5). Use of cutting edge educational tools for an initial programming course. In *Proceedings of IEEE Global Engineering Education Conference* (pp. 556-563). Istanbul, Turkey.

Gouws, L., Bradshaw, K., & Wentworth, P. (2013, October 7–9). First year student performance in a test for computational thinking. In *Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference* (pp. 271–277). East London, South Africa.

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

Grover, S., Cooper, S., & Pea, R. (2014, June 21–25). Assessing computational learning in K-12. In *Proceedings of the Conference on Innovation & Technology in Computer Science Education* (pp. 57–62). Uppsala, Sweden.

Grover, S., Pea, R., & Cooper, S. (2015). Designing for deeper learning in a blended computer science course for middle school learners. *Computer Science Education, 25*(2), 199–237.

Huitt, W. (2007). *Maslow's hierarchy of needs. Educational psychology interactive*. Valdosta, GA: Valdosta State University. Retrieved November 8, 2017, from http://www.edpsycinteractive.org/topics/regsys/maslow.html.

Kong, S. C. (2017). Development and validation of a programming self-efficacy scale for senior primary school learners. In S. C. Kong, J. Sheldon & K. Y. Li (Eds.), *Proceedings of the International Conference on Computational Thinking Education 2017* (pp. 97–102). Hong Kong: The Education University of Hong Kong.

Kukul, V., & Gökçearslan, Ş. (2017). Computer programming self-efficacy scale (cpses) for secondary school students: Development, validation and reliability. *Eğitim Teknolojisi Kuram ve Uygulama, 7*(1), 158–158.

Lawler, R. W. (1997). *Learning with computers*. Exeter: Intellect Books.

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

Maguire, P., Maguire, R., Hyland, P., & Marshall, P. (2014). Enhancing collaborative learning using pair programming: Who benefits? *All Ireland Journal of Teaching and Learning in Higher Education, 6*(2), 1411–1425.

Makinen, M. (2006). Digital empowerment as a process for enhancing citizens' participation. *E-learning, 3*(3), 381–395.

Maloney, J. H., Peppler, K., Kafai, Y., Resnick, M., & Rusk, N. (2008, March 12–15). Programming by choice: Urban youth learning programming with scratch. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education* (pp. 367–371). Portland, OR.

Marji, M. (2014). *Learn to program with Scratch: A visual introduction to programming with games, art, science, and math*. San Francisco, CA: No Starch Press.

Maslow, A. H. (1943). A theory of human motivation. *Psychological Review, 50*(4), 370.

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2013). Learning computer science concepts with scratch. *Computer Science Education, 23*(3), 239–264.

Mueller, J., Beckett, D., Hennessey, E., & Shodiev, H. (2017). Assessing computational thinking across the curriculum. In P. J. Rich & C. B. Hodges (Eds.), *Emerging research, practice, and policy on computational thinking* (pp. 251–267). Cham, Switzerland: Springer International Publishing.

National Research Council. (2011). *Report of a workshop of pedagogical aspects of computational thinking*. Retrieved November 21, 2016, from https://www.nap.edu/catalog/13170/report-of-a-workshop-on-the-pedagogical-aspects-of-computational-thinking.

National Research Council. (2012). *Education for life and work: Developing transferable knowledge and skills in the 21st century. Committee on defining deeper learning and 21st century skills*. Retrieved December 6, 2016, from http://www.p21.org/storage/documents/Presentations/NRC_Report_Executive_Summary.pdf.

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

Peterson, I. (2002). *Mathematical treks: From surreal numbers to magic circles*. Washington, DC: Mathematical Association of America.

Piaget, J. (1972). Intellectual evolution from adolescence to adulthood. *Human Development, 15*(1), 1–12.

Rodriguez, B., Kennicutt, S., Rader, C., & Camp, T. (2017, March 8–11). Assessing computational thinking in CS unplugged activities. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 501–506). Seattle, Washington.

Román-González, M., Pérez-González, J., & Jiménez-Fernández, C. (2016). Which cognitive abilities underlie computational thinking? Criterion validity of the computational thinking test.

Computers in Human Behavior. Retrieved 22 March, 2017, from http://www.sciencedirect.com/science/article/pii/S0747563216306185.

Ruf, A., Mühling, A., & Hubwieser, P. (2014, November 5–7). Scratch vs. Karel: Impact on learning outcomes and motivation. In *Proceedings of the 9th Workshop in Primary and Secondary Computing Education* (pp. 50–59). Berlin, Germany.

Seiter, L. & Foreman, B. (2013, August 12–14). Modeling the learning progressions of computational thinking of primary grade students. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research* (pp. 59–66). San Diego, CA.

Sherman, M., & Martin, F. (2015). The assessment of mobile computational thinking. *Journal of Computing Sciences in Colleges, 30*(6), 53–59.

Voland, G. (1999). *Engineering by design*. Reading, MA: Addison-Wesley.

Waller, D. (2016). *Gcse computer science for OCR student book*. Cambridge, England: Cambridge University Press.

Wenger, E. (1998). *Communities of practice: Learning, meaning, and identity*. Cambridge, England: Cambridge University Press.

Werner, L., Denner, J., Campe, S., & Kawamoto, D. C. (2012, February 29–March 3). 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). Raleigh, NC.

Wigfield, A., & Eccles, J. S. (2000). Expectancy-value theory of achievement motivation. *Contemporary Educational Psychology, 25,* 68–81.

Wilson, A., Hainey, T., & Connolly, T. M. (2012, October). Evaluation of computer games developed by primary school children to gauge understanding of programming concepts. In *Proceedings of the 6th European Conference on Games-based Learning* (10 pp.). Cork, Ireland.

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

Wolz, U., Stone, M., Pearson, K., Pulimood, S. M., & Switzer, M. (2011). Computational thinking and expository writing in the middle school. *ACM Transactions on Computing Education, 11*(2), 1–22.

World Economic Forum. (2016). *The Fourth Industrial Revolution: What it means, how to respond*. Retrieved April 15, 2017, from https://www.weforum.org/agenda/2016/01/the-fourth-industrial-revolution-what-it-means-and-how-to-respond/.

Zhong, B., Wang, Q., Chen, J., & Li, Y. (2016). An exploration of three-dimensional integrated assessment for computational thinking. *Journal of Educational Computing Research, 53,* 562–590.

Zur-Bargury, I., Pârv, B., & Lanzberg, D. (2013, July 1–3). A nationwide exam as a tool for improving a new curriculum. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education* (pp. 267–272). Canterbury, England.