Original article

doi: 10.1111/jcal.12155

Journal of Computer Assisted Learning

An analysis of young students' thinking when completing basic coding tasks using Scratch Jnr. On the iPad

G. Falloon

The Faculty of Education, University of Waikato, Hamilton, New Zealand

Abstract

Recent government moves in many countries have seen coding included in school curricula, or promoted as part of computing, mathematics or science programmes. While these moves have generally been associated with a need to engage more young people in technology study, research has hinted at possible benefits from learning to program including fostering general thinking skills. However, little research has been carried out exploring these ideas. This study analysed data collected while 5- and 6-year-old students in a New Zealand primary school were using Scratch Jnr. to learn about basic shapes, as part of a numeracy topic. Analysis combined Brennan and Resnick's (2012) computational thinking skills framework and Krathwohl's (2002) revision of Bloom's Taxonomy to evaluate any role general thinking skills played in these students' coding work. Results suggest including basic coding in primary curricula provides teachers with an effective means of exercising their students' general and higher order thinking skills. They build on Brennan and Resnick's (2012) framework by including conceptualization as an important element in students' computational work and highlight the role of predictive thinking in debugging code. Findings support historical arguments that more needs to be done to investigate students' cognitive processes when undertaking computational work.

Keywords coding, collaborative, competencies, computational, learning, thinking.

Introduction

This article reports outcomes from a study exploring the use of the iPad app Scratch Jnr. in a primary (elementary) school numeracy topic about learning basic shapes. It used a display capture app to record students' interactions while working in pairs, developing simple code to program a sprite to draw an array of shapes set by their teachers. Using Krathwohl's (2002) revision of Bloom's Taxonomy (cognitive process dimension) and Brennan and Resnick's (2012) computational thinking (CT) framework, the study explored the types of thinking students engaged in during their coding tasks. The study used Brennan and Resnick's (2012) framework to identify key components of the students' computational work, and Krathwohl's (2002) thinking skills to analyse the types of thinking the students used when engaged with these components. Its purpose was ***not*** to measure or assess progression in thinking skill development, nor was it to ***compare*** the efficacy of computational with other learning tasks for thinking skill development. Rather, its goal was to investigate the *types* of thinking young students used, and *when* they used them within computational tasks, to learn more about whether such tasks create useful opportunities for students to exercise and possibly enhance an array of thinking capabilities. This study is timely, given moves by many governments

Accepted: 26 June 2016
Correspondence: Garry Falloon, The Faculty of Education, University of Waikato, Hillcrest Rd, Hamilton, New Zealand. Email: falloong@waikato.ac.nz

© 2016 John Wiley & Sons Ltd

to introduce computational learning into the school curriculum.

Computational learning and the school curriculum

The past few years has seen many governments introduce computing curricula as part of core learning in compulsory education (e.g., Australian Curriculum Assessment & Reporting Authority, 2014; Dept. for Education, 2013; Education Scotland, 2015). A major component of these relates to the development of CT through activities such as coding, which, it is argued, 'are precisely the sort of skills which the jobs of the future – and, for that matter, the jobs of the present – demand' (Gove, 2014: p.1). The Computer Science Teachers' Association (CSTA) taskforce in the U.S broadens the argument for CT, by highlighting its value for 'improving problem solving and critical thinking… and helping students realise that computers can automate solutions that solve problems more efficiently and extend their own thinking' (CSTA, 2011: p.6–7). They also comment that applying elements of CT is inherent in all state curricula, pointing to their value for enhancing students' higher order thinking and general problem solving capabilities. Many curricula emphasize the learning of basic coding through mastering what Brennan and Resnick (2012) term 'computational thinking concepts' (p.3) – that is, the key understandings underpinning most programming languages. These include developing sequences, understanding triggers and events, parallel processes, and working with conditionals, operators and variables. Such learning also finds some support in academic circles (e.g., Barr & Stephenson, 2011; Lye & Koh, 2014; Seiter & Foreman, 2013; Yadav, Mayfield, Zhou, Hambrusch, & Korb, 2014).

However, some researchers and educational agencies point to wider benefits from including CT tasks such as coding in the school curriculum (e.g., CSTA, 2011; Lye & Koh, 2014; Wing, 2010). They identify advantages stemming from the type and sophistication of thinking needed to understand and solve computational problems, with Lye and Koh (2014) commenting that CT fosters 'the ability to think more systematically, and (assists with the) development of mathematical and scientific expertise' (p.52). Wing (2010) takes an even broader perspective, claiming CT to be 'the new literacy of the 21st century' (p.3) with educational benefits from its focus on problem solving, analysis, evaluation and reflection meaning 'computational thinking can transfer to any domain, by enhancing and reinforcing intellectual skills' (p.5). Such views closely echo those of earlier theorists such as Seymour Papert, who, in his 1980 book *Mindstorms*, described the turtle sprite in LOGO as 'an object to think with' (Papert, 1980: p.11). Indeed, much of Papert's pioneering work in this field underpins more contemporary theory and practice relating to building transferable cognitive skills through computational tasks.

While Wing's and Papert's notion of transferability holds much appeal, historical accounts cast doubt as to whether or not this is the case (e.g., Mayer, Dyck, & Vilberg, 1986; Pea, 1983; Pea & Kurland, 1984a; Pea & Kurland, 1984b; Pea, Kurland, & Hawkins, 1985). Pea and Kurland (1984b) for example, point to the lack of empirical studies exploring possible transfer of problem solving skills from computational tasks, to general problem solving in other domains. They highlight the importance of the learning environment, levels of existing student knowledge and the capabilities of teachers as important variables affecting instructional quality, and thereby the outcomes of computational learning. They make the point that little is known about 'characterisations of levels of programming proficiency' (Pea & Kurland, 1984b: p.162), and just what might be expected of students engaged in CT tasks at different levels of education. This, they claim, contributes to vagueness and a lack of specificity in the expected outcomes from CT activities. Similar concerns have been raised by others including Mayer *et al.* (1986), who contend that while learning to code may be useful for enhancing related thinking skills – for example more sophisticated thinking needed to learn difficult concepts *within* a specific programming language, 'there is not strong support for the idea that it will radically improve general thinking skills' (p.608). Despite these studies being somewhat dated, an extensive literature search revealed no other research exploring the questions they raise.

These early studies point to the need to build understanding of the thinking processes used by students when engaged in computational work. This knowledge is important for determining any *general learning value* from CT tasks that are becoming increasingly commonplace in schools. Technological advances including tablet computers, and provisioning systems such as Bring Your Own Device (BYOD), mean many teachers now have viable options to explore CT with their students.

© 2016 John Wiley & Sons Ltd

Cheap, object-oriented, 'drag and drop' apps such as Scratch, Scratch Jnr, Pyonkee and, more recently, Tickle, have removed much of the syntactic complexity from code development. This should mean that teachers and their students are better positioned to explore CT activities, freed from the technical intricacies of mastering text-based coding.

Computational and general thinking

In 2012, Brennan and Resnick presented a framework for studying and assessing CT. Using Scratch as an example, they proposed three essential elements to building CT through coding. These are learning and using computational concepts, computational design practices and displaying computational perspectives. Briefly, concepts comprise *understanding* key ideas relating to the 'mechanics' of building code, such as loops, events and triggers, using conditionals, operators, variables and so on. Practices focus on *how* this process is undertaken. This means the sort of strategies and techniques applied by students when coding. Practices include testing and debugging, remixing and reusing code, modularization and working iteratively and incrementally. Dispositional elements relate to *perspectives* displayed by students while coding. These include viewing coding as a form of creative self-expression, noting advantages from collaboration and engagement with others, and a developing capacity to question about and critique technologies. Brennan and Resnick's framework concentrates primarily on the sort of knowledge used, and how it is used, when students create code. Its principal purpose is not to describe cognitive dimensions, such as the types of thinking students use when completing these tasks, and when and how they use them. However, exploring this relationship is a useful endeavour, as it should provide insights into if and how CT activities might provide teachers with an effective means of allowing their students to exercise a range of general thinking skills.

Although Wing (2010) theorized that the power of computational activity lies in its ability to build intellectual capacity useful across disciplines and contexts, such claims find only tentative support in earlier literature (e.g., Mayer *et al.*, 1986; Pea *et al.*, 1985; Pea & Kurland, 1984b). Pea and Kurland (1984b), for example, point to a lack of empirical evidence for problem solving and planning skill transfer in studies of students' work with Logo. They comment that such claims are based on little more than enthusiastic anecdotes, and that results suggest 'the Logo programming experiences had no significant effects on planning performances' (p.160). While Mayer *et al.*'s (1986) study found limited evidence of a relationship between a person's thinking skills and their ability to learn BASIC programming – specifically problem representation and procedure comprehension, they too comment on the problematic nature of claiming generalisable skill transfer. Instead they state 'the most fruitful way to search for a relationship between thinking skills and programming is to focus on the thinking skills that are cognitive components of programming' (Mayer *et al.*, 1986: p.610). The present study responds to this challenge by exploring the types of thinking students use when solving problems embedded in coding activities.

Curriculum, computational work and the development of thinking capabilities

The ability to analyse and think critically and reflectively are seen as important capabilities for young people growing up in the 21st century (Ananiadou & Claro, 2009). This is reflected in curriculum statements worldwide (e.g., Australian Curriculum Assessment & Reporting Authority, 2010; Council for the Curriculum, Examinations and Assessment, 2014; Ministry of Education, 2007), and are viewed by organizations such as UNESCO, the OECD and CSTA as an important competency set required to lead a successful future life (Ananiadou & Claro, 2009; CSTA, 2011; OECD, 2013, 2015; Rychen & Salganik, 2003). Generally, these curricula emphasize what Bloom, Engelhart, Furst, Hill, and Krathwohl (1956) categorize as higher order thinking skills – namely analysis, synthesis and evaluation, that are often integrated into learning across subject areas. Solving ill-structured problems is seen as an important component of this, where basic processes (observing, inferring, predicting etc.) are combined with integrated processes (interpreting data, manipulating variables, hypothesizing and experimenting) in developing problem solutions (Lewis & Smith, 1993). These processes align well with the sort of capabilities linked to computational tasks. As Gouws, Bradshaw, and Wentworth (2013) observe, CT 'is not about thinking like a computer… but is an approach that includes (understanding) all aspects of a problem, considering the complexity of the problem, and finding an optimal solution that can be achieved with the available resources' (p.10).

© 2016 John Wiley & Sons Ltd

Gouws *et al.* (2013) developed a theoretical framework that mapped learning computational concepts (similar to Brennan and Resnick's) against cognitive levels developed from Bloom's Taxonomy of Educational Objectives. Four thinking skill levels were used to evaluate the 'cognitive demands' triggered by computational concepts such as abstraction, modelling, developing algorithms, generating automated processes etc. in the iPad app, LightBot. The thinking skills were Recognizing (recognize and recall knowledge relating to the problem); Understanding (interpret, compare and explain the problem); Applying (using CT to develop a solution) and Assimilating (critically analyse and decompose the problem). Although their matrix had not been field-tested, desk analysis suggested links existed between the exercise of computational concepts and different types and levels of thinking. This conclusion finds support in the earlier work of Selby (2012), whose research found 'implied connections between CT skills and programming activities, and implied relationships between CT skills and other taxonomies of learning' (p.74). The present study builds on both Gouws *et al.* (2013) and Selby's (2012) work by investigating this theoretical relationship through empirical analysis of field data.

Research goal and questions

Data collection was guided by these questions:

1. What types of general thinking skills were evident, and how were they applied, in these students' computational work?
2. What relationship exists between the exercise of general thinking skills and these students' computational work?

The research context and prior work

This study continues a series commenced by the author in 2011 (see Falloon, 2013a, 2013b, 2014; Falloon & Khoo, 2014; Falloon, 2015; Falloon, in press). The research involved two primary (K–6) year 1 and 2 classrooms (5 and 6-year-olds), from February 2015 to April 2015. The two participating teachers worked collaboratively in a large, shared classroom space, designed as an innovative learning environment (Figure 1). Earlier research had involved one of the teachers, but for the other, this was their first time. None of the students had been involved in any of the earlier studies.

The class curriculum was designed thematically, with learning in different subjects (language, mathematics etc.) being integrated into topics that combined knowledge and skills from multiple areas. Data were collected during a numeracy-focused topic (geometry), where student pairs used Scratch Jnr. to create a range of basic shapes and letters. These were two squares and two rectangles (each of different sizes), a triangle, and the upper and lower case letters L, U, E, T, F, V and Z. Working with the students, the teachers drew the shapes on two whiteboards, at the same time discussing and recording the desired attributes of each (the success criteria). Although students could choose which order they attempted the shapes, most followed the sequence as recorded by the teacher on the whiteboards. Once each shape was successfully completed and verified by a teacher, the students recorded their initials beside it on the whiteboard. Figure 2 shows one of the whiteboards with student initials.

Prior coding work included two introductory sessions using the iPad app *Daisy the Dinosaur*. During these, students were introduced to building and running basic code sequences using an object-oriented environment similar to Scratch Jnr. Prior teaching comprised revision of shape attributes (introduced previously), building the success criteria, and an introduction to using Scratch Jnr. (via Apple TV). The introduction explained the basic app tools, and revised the 'drag and drop' process of creating code.

Data collection and analysis

Data were gathered from 32 students (16 pairs), comprising 14-year 1 (average time at school = 2 months) and 18-year two students (average time at school = 14 months). Collection took place over five sessions, each lasting approximately 25 to 40 min. Pairs were used as the number of recording app-equipped iPads was limited, and because it also supported the teachers' key competency goals of building students' collaborative and cooperative learning skills. As some students were absent on some days, pairings occasionally changed between sessions, although most remained stable. Data were collected using an embedded display and audio capture app installed on iPad Air tablets, that recorded

© 2016 John Wiley & Sons Ltd

Figure 1 The Innovative Learning Environment (ILE)

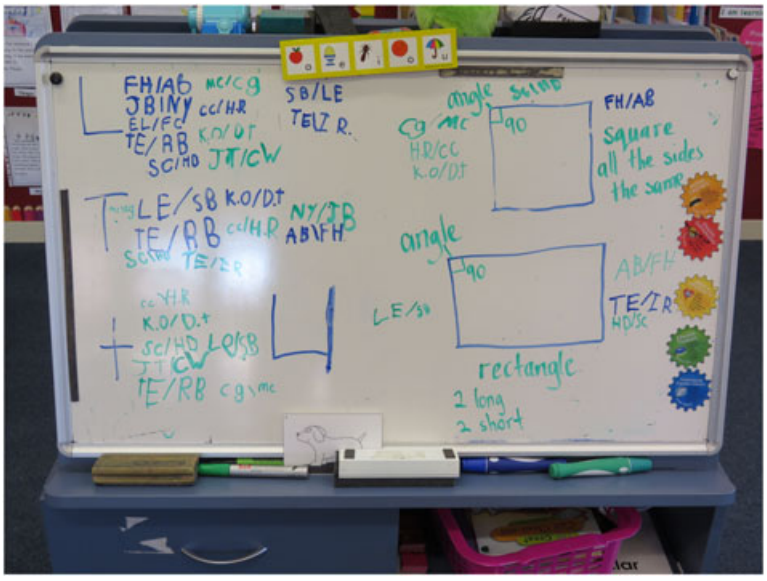

Figure 2 A Whiteboard was Used to Demonstrate Shapes and Log Successful Completions

the students' on screen activity and associated oral interaction.[1] In total just over 21 h of display capture data were recorded, but because of resource constraints and the time consuming nature of coding video data, 9½ h were selected for analysis, including the three selections used in developing the analysis framework. Data sample selection was based on these criteria:

1. Data from a balance of boy/boy, girl/girl and girl/boy pairs;

[1]Full explanation of this system can be found in Falloon (2013a).

© 2016 John Wiley & Sons Ltd

2. Data from both year 1 and year 2 students from all sessions;
3. Data representing students' work on all of the set tasks.

Data analysis framework

A framework comprising the general thinking types included in Krathwohl's (2002) revision of Bloom's Taxonomy (cognitive processes) was used to open code the range of data samples. To begin, Krathwohl's thinking type categories (remembering, understanding, applying etc.) were 'tested' against display data from three pairs, to determine first *if* any evidence existed of students using each thinking type, and second, *how* this use linked to their coding work – that is, to the components of Brennan and Resnick's framework. The initial review involved two coders reviewing the three samples, during which recorded activities aligning with each thinking type category, and the times they occurred, were noted. Following this, both coders met and generated single statement descriptors that were considered *generally representative* of the noted activities. Where relevant these were associated with the concepts, practices and perspectives detailed in Brennan and Resnick's framework. Table 1 illustrates the analysis framework that was generated, mapping the general thinking types and their contextualized descriptors (in italics) against the elements of Brennan and Resnick's framework, with a short explanation of the coded activities (right column).

In interpreting the framework, three aspects are of note. First, not all of Brennan and Resnick's original categories are represented. Principally this was because of a combination of very basic task design, and the extremely limited prior exposure these young students had had to activities of this type. Second, *conceptualizing* has been added as a CT practice in the fourth column. This was added as data indicated deliberate acts by students to systematically deconstruct problems (challenges) into manageable 'chunks', frequently aligning the achievement of each 'chunk' with specific tools (i.e., 'what we need to do this') or discussion about what successful completion of 'chunks' might look like (i.e., each challenge's success criteria). Third, the general thinking types *applying* and *creating* are combined. This is because the majority of data indicated the outcome of *applying* CT concepts, practices and perspectives was the *creation* and testing of code. These two thinking types were effectively used in tandem when students built code.

Data coding

Sample data were double blind coded using Studiocode video analysis software. Studiocode supports timeline coding of events in video data aligned with a code template. In this case, the code template was developed from the analysis framework generated from examination of the three data samples.

Recordings comprising the 9½ h of selected data from 11 of the 16 pairs were individually imported into Studiocode, and using the coding template, a research assistant (post doc) developed event timelines for each (see sample in Figure 3). While this was happening, the researcher separately coded 3 h of the same data to enable an inter-rater agreement calculation to be made. The selection criteria described previously were also applied to selecting the rater-agreement sample, which comprised work from eight different student pairs.

Rater agreement

In total, 436 events in the sample data across all codes were identified by both the researcher and the assistant. Rater agreement was calculated across events and codes collectively, as doing this 'by code' offered no analysis advantages. Following Gwet's (2012) recommendation, only events both coders had identified were used in calculating rater agreement, to minimize any distortion of agreement probability. After discussion between the researcher and assistant, 43 events upon which no agreement could be reached were discarded. Kappa results were rated at 'Good' according to Landis and Koch's (1977) scale. These results are summarized in Table 2.

Sample data and its alignment with the thinking type codes are included in Table 3. The table comprises the code, a description of the context surrounding the coded event, a thumbnail image from the display video and a verbatim transcript of student interaction taken from recorded audio.

Data analysis

Following coding, timeline data for each pair were exported to an Excel spreadsheet. Studiocode collates

© 2016 John Wiley & Sons Ltd

Table 1. The Analysis Framework

General thinking and main area of application	CT concepts	CT practice	CT perspectives	Coded activity
Remembering or recalling (*retrieving information relating to what needed to be done and/or outcome attributes*)		Conceptualizing (the task)	Sharing (connecting)	Remembering or recalling information about the task or specifications
Remembering or recalling (*retrieving information related to the features, tools or operation of the app*)		Conceptualizing (the toolset and resources)	Sharing (connecting)	Remembering or recalling information about tools, blocks or functions
Understanding (*deconstructing task or problem into stages or activities to aid understanding of how to solve it*)		Conceptualizing (the task)	Questioning Sharing (connecting)	Understanding or clarifying steps or stages needed to complete the task
Understanding (*interpret attributes of successful outcomes and how these will be evaluated*)		Conceptualizing (the task)	Questioning Sharing (connecting)	Understanding or clarifying the task specifications or success criteria
Applying and **creating** (*using knowledge to create and test code using 'build and test' strategy*)	Sequencing. Triggers and events. Working with data, values and variables.	Creating and testing code using non-incremental and/or non-iterative strategies	Sharing (connecting)	Applying knowledge to create and test code (strategy: 'build and test')
Applying and **creating** (*using knowledge to create and test code using incremental strategy*)	Sequencing. Triggers and events. Working with data, values and variables.	Creating and testing code, using incremental and/or iterative strategies		Applying knowledge to create and test code (strategy: 'step-by-step')
Analysing (*using general thinking* and *computational knowledge to understand challenges*, and *predictive thinking to identify and rectify possible errors, prior to creating* and *testing code*)	Working with data, values and variables	Conceptualizing tasks and criteria, and identifying possible code errors, prior to testing. Debugging, prior to testing.	Questioning	Analysing possible code errors and rectification prior to testing (predictive)
Analysing (*using general thinking and computational knowledge to analyse and rectify errors after testing code*)	Working with data, values and variables.	Conceptualizing code errors, after testing Debugging, after testing.	Questioning	Analysing code errors and rectification after testing
Evaluating (*analysing how well the outcome meets success criteria*)		Appraising the outcome from running code (not action-oriented)	Questioning	Evaluating not linked to changes or decision making
Evaluating (*analysing how well the outcome meets success criteria, and modifying code if needed*)		Appraising the outcome from running code and modifying, if needed (action-oriented)	Questioning	Evaluating linked to changes or decision making

© 2016 John Wiley & Sons Ltd

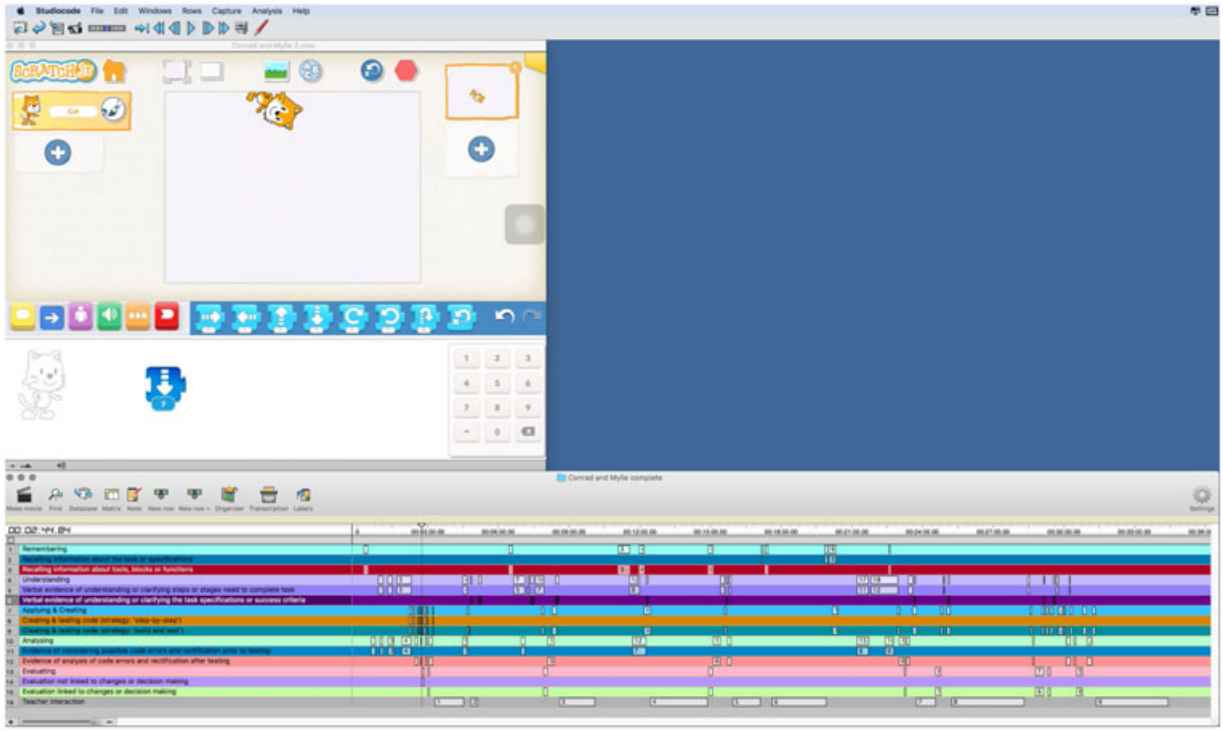

Figure 3 A Sample Timeline

Table 2. Inter-rater Agreement

	Agreement events (used in calculation)	Std. error (K)	Confidence interval (@95%)	Kappa (K)	Rating (Landis & Koch, 1977)
All codes	393	0.035	0.648–0.787	0.718	Good

data detailing event count, time per code, percentage time of all coded events (per event) and average time per event, which can be exported for analysis. The separate files for each pair were assembled into a single spreadsheet. While Studiocode exports actual times aligned to individual codes within single samples, it does not support the conversion of these to percentages to enable comparative charting across multiple samples. For this reason, raw time fields for each sample were converted into fractions of a day (i.e., 24 h = 100%) to enable accurate percentages of the total runtime spent on each coded activity to be charted by pair, and as a whole group. An example from the spreadsheet for one pair can be seen in Figure 4. It provides a summary of event counts against each code, total times for each, and mean times. The 'fraction of a day' column contains a formula that was used to convert total times to percentages for charting purposes, as described above.

Results

Charts were generated for each student pair, illustrating the percentage of runtime (i.e., time coded as 'on task') spent on activities aligned with each code. Figure 5, as an example, is the chart for pair R&T. A chart was generated for the average percentages for all pairs across all codes (Figure 6).

Discussion

What types of general thinking skills were evident, and how were they applied, in these students' computational work?

This section relates to the first research question. It discusses data relating to the types of general thinking skills students applied during different phases of their work, details the average percentage of time they spent

© 2016 John Wiley & Sons Ltd

Table 3. Sample Data Aligned with Codes

Code	Context	Image	Recorded dialogue
Remembering or recalling (*retrieving information relating to what needed to be done and/or outcome attributes*)	C&H had just completed their code for drawing a square and had successfully tested it. They were discussing their next steps.		'Yeah… we did it… (claps)… what's next… on the list? (C)… I'll go take a look on the board… she writ (sic) what we have to do up there (H)… (pause)… It says rectangle… we have to do a rectangle (H)… OK… and what's after that? (C)… We have to do an upper case T (H)… Now that ***will*** be tricky…' (C, laughing).
Remembering or recalling (*retrieving information related to the features, tools or operation of the app*)	R&T were sequencing their square. R had dragged the yellow 'tap sprite' block to the beginning of the sequence.		'… we don't use that one… we use the flag (T) … We ***can*** use this one, 'cos it does the same thing (R) … What d'ya mean? (T)… Well, it starts it off… like the flag… you just have to tap the cat and it goes (R)… Oh… that's different… we should try it (pause)… do you think we should make it go up or across first? (T)
Understanding (*deconstructing task or problem into stages or activities to aid understanding of how to solve it*)	S&L were drafting code to draw a capital T. They had managed to draw part of the shape and needed to move their sprite back to the centre.		'That gets us there… (referring to the right end of the T's top stroke) … but how do we get back? (L)… We need to get back to the middle first… so how do we… umm…? (S, pause)… …one thing at a time… let's see… if it went 9 that way… (to the right) we have to go the same back… to get us to the middle… (S, pause). Try it and see…. (L, tests 10)… OK… that's a wee bit far… but we can fix it… we need to work on the other side… (S)'

(*Continues*)

© 2016 John Wiley & Sons Ltd

Table 3. (Continued)

Code	Context	Image	Recorded dialogue
Understanding (*interpret attributes of successful outcomes and how these will be evaluated*)	H&S are finalizing their code to draw a rectangle. They had sequenced three sides (10–13–10) and were discussing the length of the final side.		'…we have to draw a rectangle… remember… the long sides need to be the same… (pause)… see, it's on the board… she's drawn one. (S). OK… so we need to make that (the fourth side) the same as the top… that'll mean it needs to be 13… (H). Yeah, it needs to go 13 back that way…' (S, referring to left).
Applying and **creating** (*using knowledge to create and test code using 'build and test' strategy*)	F&A were testing their code for drawing an upper case T. They had created the downstroke and the right side of the cross stroke, but couldn't work out how to move across and past the centre, to create the left cross stroke.		'It goes back to the middle… it just goes back to 10 (coordinate on x axis)(F)… Maybe we need to take out this one (removes right 7 block then tests code again – pause)… no, it went the other way then (A)… I think we need to make this bigger, too (the upstroke)… the up bit's longer than the cross bit… anyway (F)… What shall we make it? (pause) …say 10? Shall we try 10? (A) Try 10… but we still have to get the other bit right…' (F)
Applying and **creating** (*using knowledge to create and test code using incremental strategy*)	R&T were drafting code to draw an upper case T. They had worked out the length of the upstroke (6) and the left top crossbar (4), and had centred their sprite via a return (4).		'Come on puss… we want you to go across… right across (referring to drawing the right crossbar of T) (T)… (pause). We need to test it first to see if it's right (R)… (touches first code block, then others one at a time, in sequence – sprite moves accordingly)… OK… that gets you to there… (referring to centre)… but we've got to make you go further… umm… (pause)… if it's 4 that side… it must be 4 this side too…' (R).

(*Continues*)

© 2016 John Wiley & Sons Ltd

Table 3. (Continued)

Code	Context	Image	Recorded dialogue
Analysing (*using general thinking and computational knowledge to understand challenges, and predictive thinking to identify and rectify possible errors, prior to creating and testing code*)	C&M were working on their upper case T challenge. They had tested 6 for the upstroke, and were discussing the length of the left crossbar.		'I think we should make it 10, M (C)… No… 'cos if we make it 10 that way, it'll be too long… the top's smaller… (M) What d'ya mean? (C)… Well, look at the board… see, it's littler… the top bit… if we make it 10, it'll look like a cross, not a 'T'…(M). Oh…ok… (pause)… what d'ya think we should make it then?' (C) …
Analysing (*using general thinking skills and computational knowledge to analyse and rectify errors after testing*)	H&C were working on their upper case T. They had tested their code as shown, and were analysing the result. They were discussing the action of each block.		'That one's good, eh, C… (H) (referring to 'home' block). Yeah, it put it back to the start… (pause)… let's see… it goes down first… then up… across… back… ummm… back to the middle… (C). It didn't go far enough… it stopped! Try making it 10 (H) (referring to right crossbar. C changes 9 to 10, then tests)… Not enough, it needs to be more. Try 11 (H)'… (they continue this pattern, increasing by 1 until they reach 16)
Evaluating (*assessing how well the outcome meets success criteria*)	R&T had successfully completed their code to draw an upper case T. They were experimenting to turn it into a block letter shape.		'We did a good job, didn't we R… it was the hardest one, eh… (pause) … It was hard, but we did it (T, laughing)… We can use the yellow block again… it's good how it makes it work like the flag… (pause) it's cool how you just touch the cat and it goes (R)… What do we have to do now? … I think 10 is too long (T)…'

(*Continues*)

© 2016 John Wiley & Sons Ltd

Table 3. (Continued)

Code	Context	Image	Recorded dialogue
Evaluating (*assessing how well the outcome meets success criteria, and modifying code if needed*)	N&J were sequencing a rectangle. They had sequenced one side (10) and the top (9), and had tested the code. Their sprite had gone off screen top and side.		'What just happened… what'd we do wrong… where'd it go…? He went into the sky… and then came out of the sand (J, laughing)… We need him to go across first… (pause)… but not so far… 'cos remember that one (referring to home block) makes him start over there (N, pointing)… Say I make that one 5 that way…' (J drags away existing blocks and replaces first with 5 right)…

	A	B	C	D
1	**Statistics for**	**Chelsea and Hamish complete.**		
2	Number of rows:	16		
3				
4	Name	count	total time	**fraction of day (total)**
5				
6	**Remembering**	11	00:02:01.77	0.001409375
7	Remembering or recalling information about the task or specifications	6	00:00:44.79	0.000518403
8	Remembering or recalling information about tools, blocks or functions	5	00:01:13.49	0.000850579
9				
10	**Understanding**	11	00:03:09.31	0.002191088
11	Understanding or clarifying the steps or stages need to complete task	11	00:03:09.31	0.002191088
12	Understanding or clarifying the task specifications or success criteria	0	00:00:00.00	0
13				
14	**Applying & Creating**	21	00:02:17.56	0.00159213
15	Applying knowledge to create & test code (strategy: 'build and test')	21	00:02:17.56	0.00159213
16	Applying knowledge to create and test code (strategy: step-by-step)	11	00:01:05.53	0.000758449
17				
18	**Analysing**	7	00:00:53.89	0.000623727
19	Analysing possible code errors and rectification prior to testing (predictive)	3	00:00:24.39	0.000282292
20	Analysing code errors and rectification after testing	4	00:00:29.50	0.000341435
21				
22	**Evaluation**	8	00:01:01.83	0.000715625
23	Evaluating linked to changes or decision making	8	00:01:01.83	0.000715625
24	Evaluating not linked to changes or decision making	0	00:00:00.00	0
25				
26	**Teacher interaction**	11	00:10:58.37	0.007620023
27				
28	Sum of Percentage			
29	Sum of Main Codes			
30	Sum of Sub Tasks		00:10:26	0.00725

Figure 4 Sample Statistics Summary for C&H

applying them and briefly outlines their overall contribution. Figure 6 records the average percentages of total runtime for each thinking skill within its general area of application. It is acknowledged that these categorizations are not definitive, but should be interpreted as the *main* or *predominant* area of application within the work of this group.

While thinking types historically labelled 'lower order' were prominent (average 45% coded as *Remembering* and *Understanding*), exercising these at appropriate times was very important to these students' work. Timely application of these skills helped students understand what tasks demanded, how they might go about solving them and what tools were needed.

© 2016 John Wiley & Sons Ltd

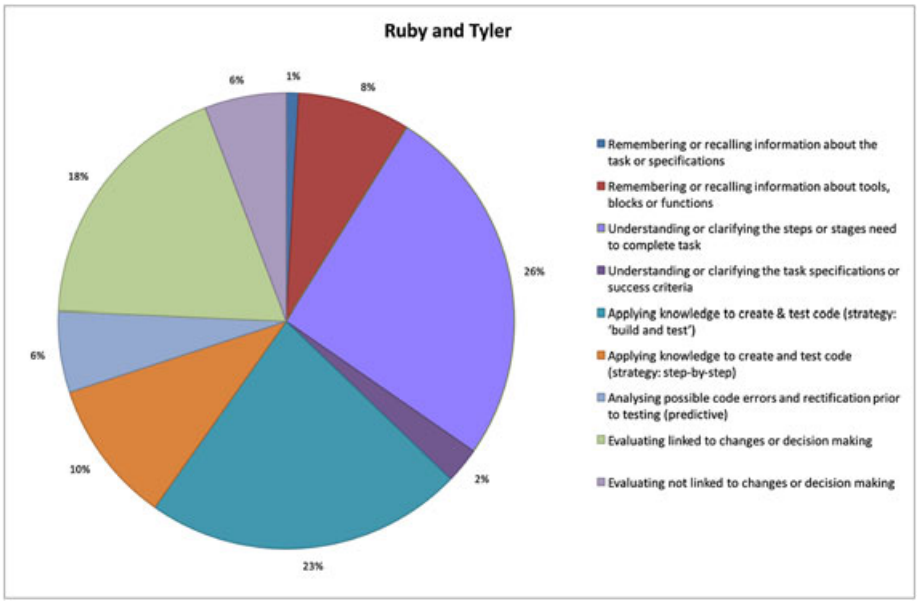

Figure 5 Sample Data Chart for R&T

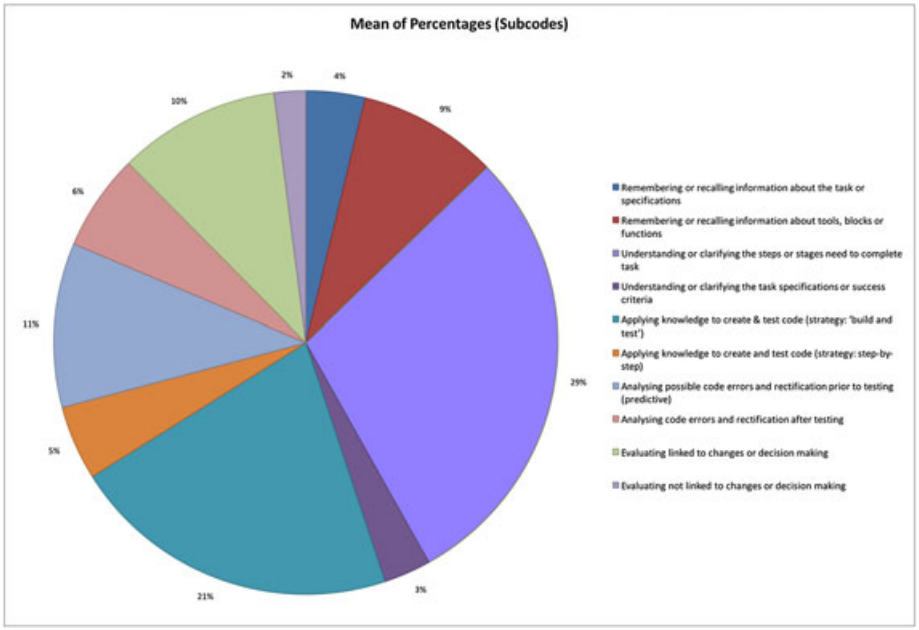

Figure 6 Average Percentages for All Pairs

Building this knowledge was often a collaborative process, involving the sharing of information and strategies within and between pairs. Sample data in Table 3 demonstrates this for two pairs (rows 1 and 4) and illustrates how students collaborated to generate a common understanding of a task and its success criteria.

Analysing (average 17%) featured in different aspects of these students' work according to what stage they were at with their tasks, although most *analysis* was associated with debugging. For example, *analysing* was linked with assessing the effect of different code combinations before testing (11%), and also with debugging code after testing (6%). Small amounts were also aligned

© 2016 John Wiley & Sons Ltd

with *evaluating*, predominantly when students appraised their outcomes against the success criteria. Again, *analysing* was frequently a collaborative process, with pairs giving to and receiving advice from others, to help solve problems. Sharing and connecting was a common strategy employed at all stages of these students' work.

Not originally included in the thinking types framework but evident in data aligned with debugging (*analysing*) was a skill that was termed *predictive* thinking. Although not coded separately, *predictive* thinking was analysis that occurred while students were building code, and can be described as thinking whereby they predicted or hypothesized a result from running code, before actually testing it. Data in Table 3, row 7 illustrates this, where students C&M used the example provided by the teacher as a 'yardstick' to predict the result from running the code they were creating. This analysis triggered a change in the length of the top stroke of the 'T', when they realized the measure they were going to use was incorrect. *Analysis* (*predictive*) was coded at 11% of average runtime and was apparent in the work of five pairs, and interestingly, rated at almost twice the level of *analysis* after testing (6%).

Given their young age, the presence of this relatively sophisticated type of thinking was surprising. However, deeper interrogation of display audio surrounding three of these events, suggested that rather than being a deliberate strategy to improve accuracy or efficiency, it was motivated more by a reluctance to take risks. It was interesting to observe these students spending significant time analysing code *before* testing, rather than trialling it and evaluating results, then using this information to make changes. Interestingly, particular features of Scratch Jnr. appeared to influence the extent to which some of these students were prepared to test their ideas. For example, one pair who were observed spending considerable time discussing their code without actually drafting it, commented that the absence of a visible trail made it challenging for them to work out where they had made mistakes, after they had run their code. As S put it, '…we're just trying to work it out in our heads… ' cos the cat doesn't make a mark when it moves… so… we can't see where we went wrong after it's done it…' (Student S, personal communication, recorded by display capture, 12:11). Acknowledging that Scratch Jnr.'s main purpose is not to help teach geometry, in countries such as New Zealand where activities like coding are likely to be integrated into other curriculum areas in primary schools, the minor addition of a 'sprite pencil' would be advantageous. While earlier LOGO-based programming languages had this facility, it seems to have been removed from most contemporary early education coding applications.

Predictive thinking is an important skill, and it was interesting to find emerging evidence of this in the work of some of these very young students. However, its presence appeared to be something of a 'double edged sword'. The amount of time spent by these students speculating on possible issues was considerable, and significantly slowed their overall progress. Encouraging predictive thinking needs to be balanced with encouragement to take risks. Students must be supported to evaluate the results of a particular course of action or activity, and use this information to action modifications or changes. While the design of apps like Scratch Jnr. may encourage experimentation and exploration, operationalizing this also relies on establishing supportive classroom environments.

Finally, an average of 12% of runtime was coded as students engaged in evaluative thinking (*evaluation*). Most activity coded in this category was associated with students appraising their outcomes against the success criteria. Most often this led to changes in their code when it was discovered outcomes did not meet the criteria (10% in data from six pairs), although 2% (in data from two pairs) did not make any changes or modifications. Display audio data suggested these pairs struggled to identify mistakes, or did not understand how to correct the mistakes they had made. A strategy these students frequently applied was to re-run the unaltered code, hoping that somehow it would have corrected itself!

Although most evaluative thinking resulted in changes to code (e.g., Table 3, row 10) not all of these were immediately successful, and the strategies used when making changes varied greatly. Some changes resulted from close evaluation of an outcome, some appeared more random in nature, while others represented a combination of both. Often students began debugging systematically (line-by-line), after discussing changes likely to lead to a successful result. However, if this process did not prove successful within a relatively short time period, data showed they adopted less systematic strategies, such as adjusting code randomly in the hope they stumbled across the right combination. Debugging code is a complex process requiring perseverance and a systematic approach. Dispositions like perseverance

© 2016 John Wiley & Sons Ltd

and learning systematic ways of approaching problems are valuable life skills. Teachers exploring coding with young children should be mindful of the important role they have in teaching systematic approaches to problem solving, and modelling dispositional traits such as tenacity and perseverance when implementing them.

What relationship exists between the exercise of general thinking skills and computational activity?

The following relates to the second question exploring the relationship between general thinking and elements of computational activity, as detailed in Brennan and Resnick's (2012) CT framework. Italics have been used to indicate where links were evident, and these are further detailed in Table 1.

In starting, it should be noted that for this group of students the additional practice of 'conceptualizing' was added to the original evaluation framework, responding to data indicating the use of general thinking skills to clarify and *understand* steps and stages needed to complete a task (average 29%); *recalling* or *remembering* information about a task or available tools (average 13%); and discussing and *understanding* success criteria (average 3%). Conceptualizing tasks, problems, criteria and resources consumed, on average, 45% of the working time for these students, but it was a very important part of their endeavours. Some may argue that doing this is simply a natural part of planning and undertaking any work. However, what is interesting here is not so much that students spent time thinking about and planning their work, but how many went about it. A common strategy used by students coded in the 29% of runtime categorized as *understanding steps or stages*, could be described as 'chunking'. Chunking refers to a process whereby tasks were deconstructed and (re)organized into manageable pieces, seemingly in an effort to make solving them easier (*understanding*). Often chunking was associated with discussion of which blocks (tools) should be used for the job, and what a successful outcome from creating each chunk might look like (*analysing*). Although unable to compare this process with strategies these students used for other tasks, the deliberate, systematic and organized manner in which they deconstructed tasks into manageable pieces, was impressive. While perhaps not constituting a purely 'technical' computational practice in the way Brennan and Resnick define them, the importance of conceptualization to these students' work was apparent. It helped them *analyse* and *understand* what a task demanded, strategize how to manageably meet the demands, and identify the resources they needed to do so.

As Table 1 indicates (and as illustrated in the sample data), conceptualization was often associated with questioning (*understanding*). Brennan and Resnick (2012) comment that this perspective means 'young people should feel empowered to ask questions about and with technology' (p.11). Results from the current study suggest the purpose and nature of questioning was broader than this, with questioning of self and others being an important strategy in solution development. In many ways questioning for these students was not so much a perspective, but more a practice, to the extent that it helped them *understand* tasks, *analyse* intended or developed solutions and *evaluate* outcomes.

Questioning was also integral to the collaborative approach these students adopted when developing solutions. In Table 1 this is referred to as 'Sharing', and is aligned with Brennan and Resnick's 'Connecting' perspective. Substantial interaction occurred between students that focused on conceptualizing tasks or problems (*understanding*), finding code bugs (*analysing*) or advising on 'next steps'. Consistent with findings of earlier studies (Falloon, 2015), the iPad's design and portability greatly assisted with this, as students easily transported their device to selected others to get help or advice. Interestingly the display recordings revealed no instances of students actually copying (reusing) someone else's code, and no evidence of students reusing or adapting their own code for different purposes (e.g., changing length variables in a square to create a rectangle) without teacher suggestion. It was interesting to note how these young students treated each task as a separate entity, perhaps suggesting a lack of concept-transfer ability.

Data revealed emerging knowledge of technical computational concepts such as working with variables (e.g., distance, direction), sequencing, triggers and events, and step-testing code (*in applying* and *creating*). While step-testing was limited to only a few pairs (5% average), it was notable that those who used it did so without any prior instruction, seeming to transfer the strategy from their earlier work with *Daisy the*

© 2016 John Wiley & Sons Ltd

Dinosaur. An example of this can be seen in Table 3, row 6, where student R adopts a block-by-block (incremental) approach to test her code. The students also displayed emerging knowledge of mathematics ideas such as distance (steps measured using the grid) and direction of sprite movement from its initial location, sometimes deciding to change these after reviewing the result of running their code (*analysing* and *evaluating*). Interestingly, unlike results of earlier studies of students' use of learning games (Falloon, 2013a), decisions about modifying code appeared somewhat more deliberate. *Analysis* occurred both before and after testing (average 11% and 6% respectively), with many of these occasions resulting in adjustments to distance and direction variables. Data in Table 3 rows 7 and 8, illustrate this process. They also show how valuable verbal interaction between the students was for determining changes.

Conclusion

Early studies by Pea and others take issue with the lack of empirical evidence for claims made about the transferability of general thinking capabilities from learning to code. The problematic nature of identifying transfer, and the variable conditions under which coding is learnt, add to the complexity of proving transfer theories. In some ways the fixation on transfer from computational learning has been something of a distraction. That is, it has diverted attention from the thinking processes *in operation* when students are involved in creating code to solve problems. Mayer *et al.* (1986) alluded to this in their comment about research needing to focus its attention on understanding the *cognitive dimensions of programming*, rather than being too concerned about the question of general transfer. It is surprising that little research since has attempted to unpack these dimensions.

Acknowledging the scope limitations of this study, data suggests integrating basic coding into primary school curricula can provide teachers with a useful medium for students to exercise a range of general, computational and higher order thinking skills. While this work did not seek to measure the development, progression or transfer of these, sufficient evidence exists that well-designed coding tasks embedded in collaborative, problem-based and/or thematic curriculum designs, can support a wide array of student thinking capabilities. These results broaden the base of evidence for including coding in core curricula. They move the focus away from the vocational arguments of politicians and industry groups, towards more cognitive dimensions that are relevant to *all* students. This is very important given the role of school curricula in helping prepare *all* young people for future citizenship, not just those wishing to pursue technology-related careers.

A key objective of the teachers in this study was using these coding tasks as a means of building collaborative, cooperative and self-management skills in their young students. This aligned closely with the Key Competency goals of the Ministry of Education (2007), which derive from Rychen & Salganik's, 2003 framework, *Definitions and Selection of Key Competencies for a Successful Life and Well-Functioning Society*. Teaching decisions relating to the learning task, the evaluation (success) criteria and the composition of student groups, were deliberately made to support curriculum key competency goals. Outcomes indicate these goals were successfully met, with exceedingly high levels of 'on task' engagement and student self-management, collaboration and knowledge sharing, being recorded.

As countries revise subjects and evolve curricula to incorporate future-focused skills and competencies such as those identified by Rychen and Salganik and the OECD (2013, 2015), the results of this study support including computational learning in these considerations. While further work needs to be undertaken to verify these results across different contexts and school age levels, evidence strongly supports the efficacy of coding for promoting broadly-based cognitive, collaborative, team work and self-management objectives. These competencies are seen as important outcomes for 21st century education (Bolstad *et al.*, 2012; Griffin, Care, & McGaw, 2012; OECD, 2015). Therefore, when undertaking curriculum reviews governments should expand their vision for coding beyond vocational arguments, acknowledging the much broader contribution it has to make to student learning.

Acknowledgement

The author gratefully acknowledges the funding support of the Teaching and Learning Research Initiative (TLRI) for undertaking this study.

© 2016 John Wiley & Sons Ltd

References

Ananiadou, K. & Claro, M. (2009) 21st century skills and competencies for new millennium learners in OECD countries. *EDU Working paper no.41*. [Online] Available from: http://www.oecd.org/officialdocuments/publicdisplaydocumentpdf/?cote=EDU/WKP(2009)20&doclanguage=en [Accessed 16th September 2015].

Australian Curriculum Assessment & Reporting Authority. (2010) Digital technologies in the Australian curriculum. [Online] Available from: http://www.australiancurriculum.edu.au/technologies/digital-technologies/curriculum/f-10?layout=1 [Accessed 2nd October 2015].

Australian Curriculum Assessment & Reporting Authority. (2014) Creative and critical thinking. *F-10 Curriculum*, *General Capabilities*. [Online] Available from: http://www.australiancurriculum.edu.au/generalcapabilities/critical-and-creative-thinking/introduction/introduction [Accessed 2nd October 2015].

Falloon, G.W. (2013a). Young students using iPads: App design and content influences on their learning pathways. *Computers & Education*, *68*, 505–521.

Falloon, G.W. (2013b). Creating content: Building literacy skills in year 1 students using open format apps. *Computers in New Zealand Schools: Learning, Teaching, Technology*, *25*, 77–95.

Falloon, G.W. (2014). What's going on behind the screens? Researching young students' learning pathways using iPads. *Journal of Computer-Assisted Learning*, *30*(4), 318–336.

Falloon, G.W. (2015). What's the difference? Learning collaboratively using iPads in conventional classrooms. *Computers & Education*, *84*, 62–77.

Falloon, G.W. (in press). iPads, apps and student thinking skill development. In N. Kucirkova & G.W. Falloon (Eds.), *Apps, technology and younger learners*. United Kingdom: Routledge.

Falloon, G.W., & Khoo, E. (2014). Exploring young students' talk in iPad-supported collaborative learning environments. *Computers & Education*, *77*, 13–28.

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K–12: What is involved and what is the role of the computer science community? *ACM Inroads*, *2*(1), 48–54.

Bloom, B., Engelhart, M., Furst, E., Hill, W., & Krathwohl, D. (1956). *Taxonomy of educational objectives: The classification of educational goals*. New York: David McKay.

Bolstad, R., Gilbert, J., McDowell, S., Bull, A., Boyd, S., & Hipkins, R. (2012). *Supporting future-oriented learning and teaching—A New Zealand perspective*. Ministry of Education: Wellington.

Brennan, K., & Resnick, M. (2012) New frameworks for studying and assessing the development of computational thinking. *Paper presented at AERA*, Vancouver, BC. [Online] Available from: http://web.media.mit.edu/~kbrennan/files/Brennan_Resnick_AERA2012_CT.pdf [Accessed 5th November 2015].

Computer Science Teachers' Association (CSTA). (2011) The computational thinking leadership toolkit. [Online] Available from: http://www.csta.acm.org/Curriculum/sub/CompThinking.html [Accessed 13th October 2015].

Council for the Curriculum, Examinations and Assessment. (2014) Assessing thinking skills and personal capabilities. *The School Curriculum for Ireland*. [Online] Available from: http://ccea.org.uk/curriculum/key_stage_3/assessment/assessing_thinking_skills_and_personal_capabilities [Accessed 13th September 2015].

Department for Education. (2013) National curriculum in England: Computing programmes of study. [Online] Available from: https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study/national-curriculum-in-england-computing-programmes-of-study [Accessed 12th November 2015].

Education Scotland. (2015) Digital learning and teaching: Our vision for digital learning. [Online] Available from: http://www.gov.scot/Topics/Education/Schools/ICTinLearning [Accessed 2nd July 2015].

Gouws, L., Bradshaw, K., & Wentworth, P. (2013). Computational thinking in educational activities. In J. Carter, I. Utting & A. Clear (Eds.), *The proceedings of the 18th Conference on Innovation and Technology in Computer Science Education* (pp. 10–15). Canterbury: ACM.

Gove, M. (2014) Michael Gove speaks about computing and educational technology. *Speech to the BETT Conference*, *London*. [Online] Available from: https://www.gov.uk/government/speeches/michael-gove-speaks-about-computing-and-education-technology [Accessed 17th November 2015].

Griffin, P., Care, E., & McGaw, B. (2012). The changing role of education and schools. In P. Griffin, E. Care & B. McGaw (Eds.), *Assessment and teaching of 21St century skills* (pp. 1–17). Heidelberg: Springer.

Gwet, K. L. (2012). *Handbook of inter-rater reliability* (3rd ed.). Advanced Analytics: Gaithersburg.

Krathwohl, D. R. (2002). A revision of Bloom's taxonomy: An overview. *Theory Into Practice*, *41*, 212–225.

Landis, J. R., & Koch, G. G. (1977). The measurement of observer agreement for categorical data. *Biometrics*, *33*, 159–174.

Lewis, A., & Smith, D. (1993). Defining higher order thinking. *Theory Into Practice*, *32*, 131–137.

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

Mayer, R. E., Dyck, J., & Vilberg, W. (1986). Learning to programme and learning to think: What's the connection? *Communications of the ACM*, *29*, 605–610.

© 2016 John Wiley & Sons Ltd

Ministry of Education (2007). *The New Zealand curriculum*. Learning Media Ltd: Wellington.

OECD. (2013) Innovative learning environments. [Online] Available from: http://www.oecd-ilibrary.org/education/innovative-learning-environments_9789264203488-en [Accessed 16th March, 2016].

OECD. (2015) Schooling redesigned: Towards innovative learning systems. [Online] Available from: http://www.oecd-ilibrary.org/education/schooling-redesigned_9789264245914-en [Accessed 16th March, 2016].

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

Pea, R. (1983) Logo programming and problem solving. *ERIC Technical Report No. 12*. [Online] Available from: http://eric.ed.gov/?id=ED319371 [Accessed 16th January 2016].

Pea, R. & Kurland, D.M. (1984a) Logo programming and the development of planning skills. *ERIC Technical Report No. 16*. [Online] Available from: http://files.eric.ed.gov/fulltext/ED249930.pdf [Accessed 12th January 2016].

Pea, R., & Kurland, D. M. (1984b). On the cognitive effects of learning computer programming. *New Ideas Psychology*, *2*, 137–168.

Pea, R., Kurland, D. M., & Hawkins, J. (1985). Logo and the development of thinking skills. In M. Chen & W. Paisley (Eds.), *Children and microcomputers: Research on the newest medium* (pp. 193–212). California: Sage.

Rychen, D., & Salganik, L. (Eds.) (2003). *Key competencies for a successful life and a well-functioning society*. Hogrefe & Huber: Gottingen.

Seiter, L. & Foreman, B. (2013) Modeling the learning progressions of computational thinking of primary grade students. *Paper presented at ICER'13*. [Online] Available from: http://dl.acm.org/citation.cfm?doid=2493394.2493403 [Accessed 8th November 2015].

Selby, C. (2012) Promoting computational thinking with programming. In: Knobelsdorf, M. & Romeike, R. (Eds.), *The Proceedings of the 7th Workshop in Primary and Secondary Computing*. Hamburg, ACM. [Online] Available from: http://dl.acm.org/citation.cfm?id=2481466 [Accessed 19th November 2015].

Wing, J. (2010) Computational thinking: What and why? *Carnegie Melon School of Computer Science Discussion Papers*. [Online] Available from: https://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf [Accessed 18th November 2015].

Yadav, A., Mayfield, C., Zhou, N., Hambrusch, S., & Korb, J. (2014). Computational thinking in elementary and secondary teacher education. *ACM Transactions on Computing Education*, *5*, 3–15.

© 2016 John Wiley & Sons Ltd