Planning and implementing *coding in the junior classroom for competency and thinking-skill development*

GARRY FALLOON, PAULA HALE, AND TONIA FENEMOR

KEY POINTS

- Computational tasks such as coding can provide teachers with an effective means of building a range of general and higher order thinking skills in their students.
- Computational tasks can be used to complement the teaching of basic mathematics concepts such as angles, distance, and direction, and to support the development of positional language in young children.
- Using physical movement, instruction-based activities are effective for introducing young children to computational concepts and procedure building.
- Working collaboratively on problem-based computational challenges can support the development of a range of key competencies.

http://dx.doi.org/10.18296/set.0031

Recent moves within New Zealand and internationally have called for the inclusion of computational learning through activities such as coding, in school curricula. However, including activities such as coding in school curricula is a bold move, and one that will require significant support if it is to successfully achieve its goals. This article reports on outcomes from the first year of a TLRI-supported study exploring how teachers planned and integrated coding into their numeracy programme, and the types of thinking students employed when completing coding tasks. Findings suggest that coding can provide teachers with an effective means of exercising an array of general and higher order thinking skills and learning competencies with their students, but careful attention needs to be given to the planning and systematic implementation of these activities. The article concludes with a series of recommendations for teachers considering exploring coding in the classroom.

Introduction

Recent changes in many countries have seen the learning of basic coding included as part of core curricula, or encouraged by governments and their education ministries to be included as components of other curricula, such as mathematics, science, or technology (eg., Australian Curriculum Assessment & Reporting Authority, 2014; Department for Education, 2013; Education Scotland, 2015). Much of the thinking behind these moves relates to promoting interest in technology careers, responding to the well-documented shortage of skilled professionals engaging in high-tech and supposedly high economic value work (Careers NZ, 2015; Doesburg, 2013; Rosenbaum, 2015).

Coding falls under the broad umbrella of activities commonly known as *computational thinking,* which Jeanette Wing (2010) defines as "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" (p. 1). At its most basic, this may involve students generating simple code to program sprites (a two-dimensional image or animation) to perform certain movements or actions, often in response to "challenges" or problems posed by teachers. Indeed, many object-oriented apps and programs have been developed to help teachers introduce computational concepts to their students (CSTA, 2011). These simple, graphically-driven "drag and drop" applications include Scratch and its junior school equivalent, Scratch Jnr; Pyonkee (a Scratch "clone" for iPads), Tickle (for use with physical devices such as Spheros and Ollies), and CargoBot, Lightbot and Daisy the Dinosaur ("challenge-based" apps suitable for introducing basic coding concepts). The advent of these cheap apps, and moves towards bring your own device (BYOD) learning environments, means that engaging in computational learning through coding activities is now a viable option for many teachers and their students. However, teachers, already burdened by heavy workloads and an overcrowded curriculum, could be excused for questioning the merits of adding coding to their students' learning programmes. Implications from such a move are considerable, including the need for professional development and curriculum support to enable them to successfully integrate the new area of learning into their programmes.

While early work by by Pea (1983), Pea and Kurland (1984a, 1984b) and Mayer, Dyck and Vilberg (1986) explored possible links between computational activities and general thinking-skill development, little appears to have been done since investigating this relationship. Most recent research has focused on the development of computational thinking itself—and the practical skills involved in developing code, and how these can best be developed in students. However, the early work of Pea and his colleagues provided useful direction for this study, as it alluded to the potential of computational work for building useful general thinking capability. As Mayer et al. (1986) succinctly put it, "the most fruitful way to search for a relationship between thinking skills and programming is to focus on thinking skills that are cognitive components of programming” (p. 610).

Two case studies were completed investigating coding in the primary school as part of an ongoing Teaching and Learning Research Initiative (TLRI) project, entitled Exploring Student Thinking in iPad-Supported Learning Environments. One study is detailed in this article. It reports key stages in the planning and integration of coding tasks within the geometry topic, and analyses data gathered relating to the thinking types employed by the students, as they solved the computational challenges set by their teachers.

The elements of computational thinking

In 2012 Brennan and Resnick presented a framework identifying what they saw as the main elements of computational thinking built through coding. They developed this from an analysis of 7–14 year old students’ use of Scratch, a graphical coding application designed for schools and developed at Massachusetts Institute of Technology. They identified three main elements: computational thinking concepts, computational thinking practices, and computational thinking perspectives (Table 1).

Brennan and Resnick’s framework provided useful direction informing data collection for the second research question in this study, by pointing to the types of activities within the students’ coding work where the exercise of different thinking types might be apparent. Classification of thinking types within these was made using Krathwohl’s (2002) revision of Bloom’s taxonomy (cognitive domain).

Research questions

Data were gathered responding to these research questions:

1. What strategies did these primary teachers use to plan for and integrate coding into their numeracy programme?
2. What thinking types did these young students use while completing their coding tasks?

The school and classroom context

Data were gathered in a junior primary (Year 1 and 2) innovative learning environment (ILE) from February to September 2015. The school was a decile 7 contributing primary located south-east of Hamilton, with a roll of nearly 500. Two teachers worked collaboratively in the ILE and were experienced practitioners, averaging 22 years’ teaching experience. Their 36 students had permanent access to around 20 iPads supplied by the university.

The coding apps and curriculum links

A combination of the apps Daisy the Dinosaur (Daisy) and Scratch Jnr was used during the study. These apps were chosen as they were free, easy to use, and graphically based, removing the need for students to wrestle with complicated interfaces or technically complex programming languages. These two were selected as they were similarly designed in the way code was built (i.e., graphically-oriented, “drag and drop”). It was anticipated this would mean less relearning for students, as they transitioned between the structured “code-teaching” environment of Daisy, and the more open, creative environment of Scratch Jnr.

The apps were used in a geometry unit taught to help students learn basic shapes, master positional language, and give and respond to instructions.

TABLE 1. SUMMARY OF THE MAIN ELEMENTS OF BRENNAN & RESNICK’S (2012) COMPUTATIONAL THINKING EVALUATION FRAMEWORK

Element	Description	Coding application examples
Technical and conceptual knowledge	Technical and conceptual understanding of the basic “building blocks” of code, what they do, and how they can be used.	Sequencing Events and triggers Parallelism (running processes in parallel) Using conditionals, operators and variables
Practices	The techniques and strategies used when building code.	Incremental and iterative (‘step-by-step” code building, testing, modifying) Debugging code Remixing or reusing code (own or others) Modularisation (assembling code into modular “blocks’, each contributing to a larger procedure)
Perspectives	Dispositions and attitudes displayed while building code	Sharing code with others Collaborating to solve problems Coding as a personal creative outlet Understanding of technology as a problem solving tool

Basic computational knowledge such as code sequencing, learning about step size and distance (calibrating), and triggering procedures was developing using Daisy, through a series of "sandpit-like" exploratory activities and challenges. These were followed by more formal learning activities linked to Level 1 mathematics objectives from the Geometry and Measurement strand of *The New Zealand Curriculum* (Ministry of Education, 2007) (*NZC*).

The learning goals were:

- Position and Orientation
  - Give and follow instructions for movement that involves distances, directions, and half or quarter turn;
  - Describe their position relative to a person or object.
- Shape
  - Sort objects by their appearance.

The geometry coding challenges

Using Scratch Jnr., students were challenged to develop code to make their sprite (a cat) draw a range of basic geometric and letter shapes. These were:

- two squares of different dimensions;
- two rectangles of different dimensions;
- a range of upper and lower case letters (eg., T, L, U)
- a triangle (extra for experts).

Students could choose the order they completed the challenges—although most elected to follow the order recorded by the teachers on the whiteboard. As they successfully completed each challenge and had it verified by a teacher, they logged their results on the whiteboard by placing their initials beside the appropriate shape (Figure 1).

Data collection and analysis

Data relating to question 1 (planning and integration) were collected via interview, document analysis, and iPad display and audio capture. Interview questions probed teachers' organisation and planning strategies, how they viewed the activity as contributing to wider school competency-based objectives, and the specific geometry learning outcomes they were expecting the children to master. This was supported by document analysis (teacher planning, group organisation notes, information on the school's learner virtues framework) and data from the iPad display capture app. These data were inductively coded into themes that have been used to structure the findings.

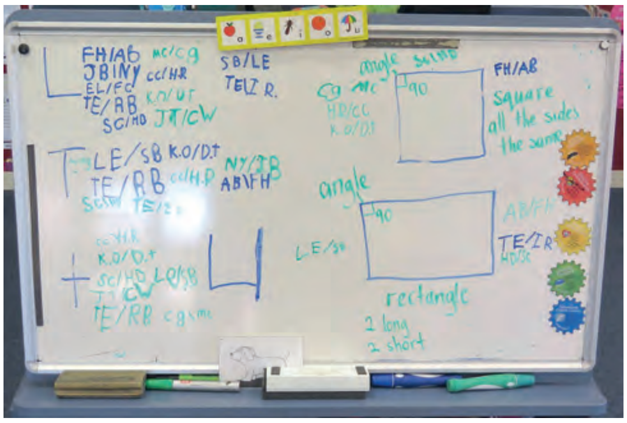

FIGURE 1. STUDENTS INITIALLED SUCCESSFUL OUTCOMES AFTER TEACHER VERIFICATION

Data for question 2 (thinking types) were collected using a specially-developed app installed on 20 university-supplied iPads.[1] The app recorded each device's display and student discussion and interaction while they worked on their coding challenges. These recordings were downloaded onto a laptop for later analysis, and selected samples were double-blind coded against a "thinking types" framework developed from Krathwohl's (2002) revision of Bloom's Taxonomy (Table 2), using Studiocode video data analysis software. The thinking type is listed in the left column, and descriptions of how each thinking type was interpreted in the context of the students' coding work is provided in the right column. Quantitative information relating to each of these thinking types was exported from Studiocode

TABLE 2. A SUMMARY OF THE "THINKING TYPES" FRAMEWORK USED TO CODE DISPLAY DATA

Thinking type	**Description of activity within coding work**
Remembering	*Retrieving information relating to what needed to be done and/or outcome attributes;* *Retrieving information related to the features, tools or operation of the app.*
Understanding	*Deconstructing task or problem into stages or activities to aid understanding of how to solve it;* *Interpret attributes of successful outcomes and how these will be evaluated.*
Applying & Creating	*Using computational knowledge to create and test code using "build and test" strategy;* *Using computational knowledge to create and test code using incremental strategy.*
Analysing	*Using general thinking & computational knowledge to understand challenges, & predictive thinking to identify and rectify possible errors, prior to creating & testing code;* *Using general thinking & computational knowledge to analyse and rectify errors after testing code.*
Evaluating	*Analysing how well the outcome meets success criteria;* *Analysing how well the outcome meets success criteria, and modifying code if needed.*

Note: Applying and creating are combined, as applying computational knowledge generally resulted in the creation of code.

and analysed in Microsoft Excel, to determine the average total times pairs spent applying each of the thinking types during their coding work.

Discussion of findings

Findings and related discussion are presented below, aligned with each of the research questions. For question 1, three themes emerged from data. These were: student organisation and developing learner virtues; the importance of preparatory activities; and the roles of the teachers and students.

Student organisation and developing learner virtues

An important outcome from this exercise was to further the teachers' efforts to develop dispositional, social, and collaborative competencies in these young students, aligned with the school's learner virtues framework (Figure 2). These virtues aim to develop students who are active learners, technologically capable learners, effective thinkers, and effective communicators, who are able to make a difference. The virtues were developed from *NZC* key competencies, and permeate activities across all levels of the school. Given the brief time most of these children had been at school, the teachers were keen to use this opportunity to build knowledge of these virtues, and how students can put them into practice.

To facilitate this, students were organised into teacher-selected pairs of mixed ability and year levels. These pairings allowed students to work together to solve problems by sharing their understanding and knowledge, and supported broader competencies such as building collaborative skills and encouraging acceptance of others' views. Mixing abilities also avoided the assumption that because a child may have been of lower ability for number and literacy, that they did not have something of value to add when working alongside a child of higher ability in this type of task. As initial trials progressed, pairings changed to ensure particular individuals did not dominate. It was important to build combinations based on trusting relationships, where each group member respected and valued the input of others.

Analysis of display data revealed the effect of grouping decisions on the children's work performance, and the value of close teacher observation and a willingness to change groupings, should issues become apparent. While pairs of mixed ability were initially selected, early analysis of display audio data revealed limitations to this in cases where personalities conflicted, or where one student dominated and the other became little more than a passive observer. Frequently, outputs from these pairs reflected the work of the dominant individual, or at best, was the result of separate interactions with the device on a more or less equal "time share" arrangement. The display data were valuable for revealing these issues—despite teachers' best efforts, they were unable to monitor all students all the time. The display data helped teachers learn more about their students' independent work characteristics and habits, and provided them with useful information for regrouping decisions that in the end reflected a balance between ability and social considerations.

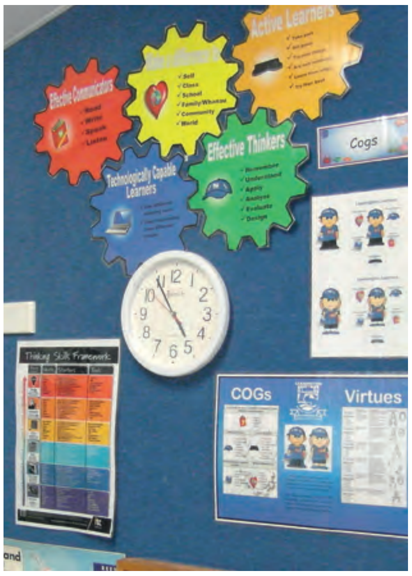

FIGURE 2. THE SCHOOL'S LEARNING VIRTUES FRAMEWORK

The importance of preparatory activities

Recognising the abstract nature of coding and the conceptual challenge this may present to very young children working in a 2-dimensional, screen-based environment, the teachers decided to complete an array of practical introductory tasks. These were designed to introduce "coding essentials" such as sequencing instructions, calibration (distance per unit or step), directions (left, right, forwards, backwards) and angles; and build understanding of the attributes of the shapes students were going to create (number and length of sides, angles, corners and the like). The activities comprised students moving in different directions responding to instructions given by one another and their teachers, creating patterns of objects by following instructions and using positional language, and producing the required shapes from pieces of paper. The shapes were also used later on as models by some students during their coding, with many physically comparing them to the shapes drawn by their sprite.

Again, the value of completing these activities was apparent when screen-capture data were analysed. Clear evidence was found of students referencing the practical activities by transferring understandings developed from them to their on-screen coding work.

An example of this was students C & M, who applied an incremental strategy to coding their square which they had practised in the playground activities:

> …we need to make all the sides the same… they need to be the same size… they need to be 7… remember… when we were outside we tested it bit by bit to see if we got it right … we've got to do the same here… (C & M, display capture 23.15–24)

In another example, students F & A were coding for an upper case "T" (Figure 3). They had correctly coded the up stroke and the left cross bar, but had become stuck at creating the right cross bar.

> …it's only gone back to the middle, A… it hasn't gone across enough! (F) Ummm… (*pause*)… but when we did it outside we got it! (A)… Yeah…what did we do again? (F) (*pause*). It has to keep going… we've got to make it go another 5… remember… you needed to keep going and not stop in the middle, when we did it the other day (*in the playground*) (A) (F & A, display capture, 17.37–50)

There was also considerable display capture evidence suggesting using Daisy for learning the "mechanics" of building code, was worthwhile. All students seamlessly applied the techniques learnt using Daisy to their work in Scratch Jnr. These included building and editing sequences, triggering events, selecting appropriate turning and movement-direction blocks, and so on. Furthermore, the more structured and "less-optioned" environment of Daisy provided fewer potential distractions, as the students mastered the basic concepts they needed.

The roles of the teachers and students

As could be expected, very active teacher engagement was required during these activities. However, given the task's dual purposes of geometry concept development and building collaborative work practices, elements of peer feedback were also incorporated at various points. During these occasions students met with other pairs and provided feedback on progress, a few pairs assisting by advising on code improvements or helping with debugging (Figure 4). On one occasion, a pair of students were recorded directly sharing their code with another pair who were having trouble coding their upper case "Z" (R & T and S & L, display capture, 46.25–47.09). However, display data revealed much of the feedback was not of a highly formative nature—providing affective rather than task attribute-related support. Despite this, data indicated the students took this process very seriously, and given it was the first time it had been attempted, the teachers were pleased with the outcome, commenting that "it (coding) was an ideal activity to build these sort of skills, because it's about solving problems together" (Teacher A, interview, July 2015).

Teacher explanation and modelling were critical for demonstrating to the students the skills and behaviours needed to work collaboratively, and to respect and value the contributions of others. This involved clearly setting expectations by continually referencing the learner virtues framework, diligently checking on students' progress and work habits, and modelling practical strategies they could use to help them work together, such as sitting beside rather than opposite each other while they worked with the iPads. It was important that if groupings were changed that students were made aware of why they were changed, and what they needed to do to improve their performance. Task success criteria were collaboratively developed with the students beforehand, and these were continually referenced as "aim points" to work towards, while students worked. A number of students were recorded comparing their emerging outcomes with the shape attributes drawn and recorded on the whiteboards, indicating the value

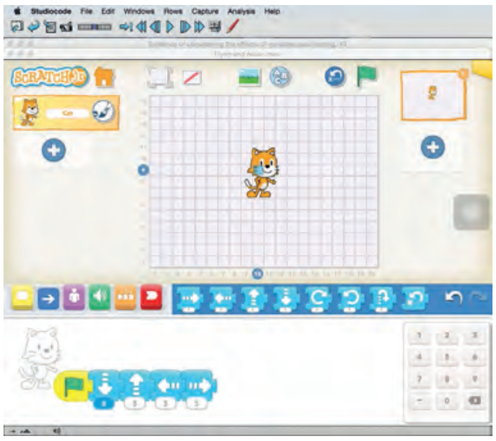

FIGURE 3. STUDENTS F & A TRANSFERRED KNOWLEDGE FROM THE PRACTICAL TASKS TO DEBUG THEIR UPPER CASE "T" CODE

FIGURE 4. PEER FEEDBACK ON CODE EFFORTS WAS TRIALLED FOR THE FIRST TIME

of this strategy. Modelling extended to helping students understand ways of working, and the sort of strategies and behaviours they should be using to enable them to successfully complete the tasks together. This was achieved via direct intervention while students were working, often to model the sort of attitude, language, or questioning students should use when collaboration is a goal. Doing this was essential to fulfil the teachers' key competency objectives, and required careful and continuous monitoring of the class, which at times, was challenging.

Computational activity and the exercise of thinking skills

To answer the second question, display data were analysed using Studiocode video analysis software. Due to the time-consuming nature of coding video and the large volume of data produced, data from nine purposively selected pairs were analysed. These pairs were chosen following an initial review of all data, to ensure that students with a range of work strategies and capabilities were included. Data from 6 boy/girl, 2 girl/girl, and 1 boy/boy combination were analysed.

Figure 5 provides the average total times for these pairs, coded against each of the thinking type categories summarised in Table 2. During coding, note was also taken of the computational element aligned with Brennan and Resnick's framework that each pair was engaged in at the time. An additional category labelled *teacher interaction* has been added. This was decided upon following the initial review of display data, that revealed the exceptionally high levels of teacher engagement needed to support these young students while they were completing their work. Details of the nature of this engagement, and why it was needed, have been summarised earlier.

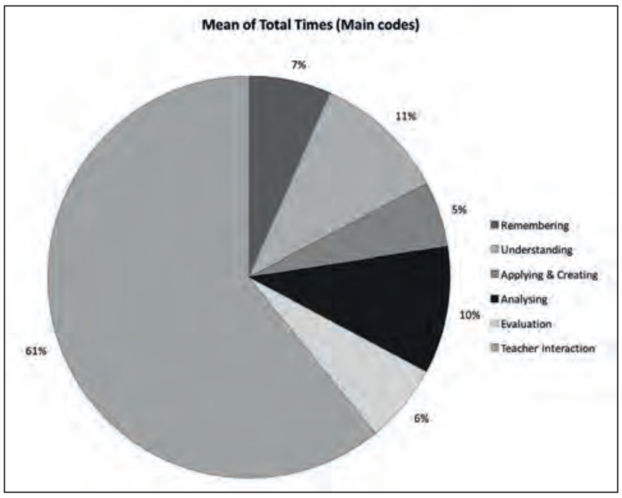

FIGURE 5. AVERAGE TOTAL TIMES FOR PAIRS CODED AGAINST THINKING TYPES

Table 3 below maps Brennan and Resnick's (2012) computational elements to the thinking types the pairs were exercising at the time coding decisions were made. Of note is the inclusion of *conceptualising and planning* and analysing (*predicting*) that were not included in the original framework. The former refers to thinking that focused on building *conceptual understanding* of what the challenge required, and on *strategising* how it might be solved. Of the 18 percent of total time coded as *remembering* and *understanding,* nearly two thirds of this was spent on *conceptualising* and *planning.* On average, these students spent over twice as long discussing and planning what they had to do (and how they might do it), than they did actually doing it (i.e., *applying* and *creating*). While conceptualising and planning appeared very important elements in these students' computational work, they were very time-consuming. Teachers should be aware of balancing these elements with encouraging students to test and then evaluate and modify code, according to outcomes. Both of these thinking types are important in computational work.

Predictive thinking (*predicting*) was often linked with analysing, and it describes a type of analysis where students predicted a likely outcome from running their code, *before* they ran it. Of the 10 percent average total time coded as *analysing,* over half of that was spent *predicting.* While predicting represented quite sophisticated thinking for such young students, like conceptualising, doing this was something of a "double-edged sword" as it consumed significant time, often limiting opportunities to test and evaluate ideas through actions. As above, it is important teachers encourage students to "give it a go" and evaluate the outcome, rather than spend lengthy periods trying to write perfect code from the outset.

Clear evidence of a full array of thinking types being exercised during this work, was present in the display data from these students. While there is insufficient space here to provide "code by code" examples of this, they can be found in other publications (Falloon, in review). Suffice to say, data indicates carefully designed, systematically implemented, and actively scaffolded computational activities like coding, can provide teachers with an ideal platform to support the exercise of different thinking types and a range of general learning competencies in their students.

Summary and conclusion

Acknowledging the limitations of this study in terms of its scope and duration, the quality and volume of data generated by the display capture tool and other methods provided unique insights into the thinking types students used in their coding tasks, providing useful information for teachers considering such activities. While results from this study indicate the efficacy of coding for thinking-skill development, it is not simply a matter of giving the students an iPad and expecting this to happen.

Firstly, a careful balance needs to be struck between teacher intervention using open, strategic questioning to guide student thinking, and allowing sufficient time for students to work out problems for themselves. To achieve this, students need space and time to trial developing strategies modelled to them by their teachers. Of course, one of the challenges for teachers working in digital environments is knowing if and when to intervene, and in what way. Unless continuously observing students, it is impossible to detect repeated errors or fully understand flaws in strategies students may be applying, as no visible "trail of evidence" is available. Teachers need to be particularly vigilant in their work around the classroom, to avoid students falling into cycles of repeated mistakes, frustration, and learning stagnation.

Second, while Scratch Jnr. proved reasonably useful for helping these students learn about shapes, it needed to be supplemented by other, concrete materials. It was difficult for the students to visualise the shape and dimensions of objects their code created. Drawing these on small whiteboards as they ran their procedure, or using their whiteboards to plan directions and calculate dimensions beforehand, were valuable strategies. Some also used as references the shapes they had made with paper in the introductory activities.

Third, teachers should take great care when selecting student work pairs, and carefully monitor these arrangements and be prepared to make changes, should issues become apparent. While in this study there was minimal evidence of deliberate "off task" behaviour, a few pairs struggled to work collaboratively, often vying for "hands on" time with the iPad, or attempting to prioritise their ideas over those of others. While this may not be unexpected, given the very young age of the students, it can be highly beneficial to deliberately teach and model collaborative strategies and use tasks to reinforce and practise them.

Finally, and related to building collaborative skills, this activity proved to be an ideal foil for introducing the students to the process of providing peer feedback. Strategies for doing this had been modelled by the teachers, and the phrasing of suitable questions to ask, and how to ask them, had been discussed in advance. While on this occasion the process may not have yielded the quality of feedback desired for all students, at the very least it introduced them to the concept, and served as a valuable starting point upon which to build.

In closing, carefully planned computational activities like coding can provide teachers with motivating and productive learning opportunities for students. However, as pointed out by Mayer et al. (1986), much more needs to be done to identify the types and sophistication of thinking students use while coding, so teachers are in a better position to design and implement tasks that will optimise their development.

Acknowledgements

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

Note

1 Full details of the system, its operation and rationale for use have been published extensively elsewhere (see Falloon 2013a, 2013b, 2014, 2015; Falloon & Khoo, 2014). Use of the display capture app was approved by

TABLE 3. THINKING TYPES MAPPED AGAINST ELEMENTS OF COMPUTATIONAL ACTIVITY
(adapted from Brennan & Resnick, 2012)

Element	Description	Coding application examples	Thinking types exercised
Technical & Conceptual knowledge	Technical and conceptual understanding of the basic "building blocks" of code, what they do, and how they can be used.	Sequencing Events and triggers Parallelism (running processes in parallel) Using conditionals, operators and variables	Remembering Understanding *(conceptualising & planning)*
Practices	The techniques and strategies used when building code.	Incremental and iterative ('step-by-step" code building, testing, modifying) Debugging code Remixing or reusing code (own or others) Modularisation (assembling code into modular "blocks', each contributing to a larger procedure)	Applying and creating Analysing *(predicting)* Analysing Understanding Evaluating
Perspectives	Dispositions and attitudes displayed while building code	Sharing code with others Collaborating to solve problems Coding as a personal creative outlet Understanding of technology as a problem solving tool	Analysing *(predicting)* Analysing Evaluating

the University of Waikato Education Research Ethics committee in March 2014.

References

Australian Curriculum Assessment & Reporting Authority. (2014). Creative and critical thinking. *F-10 Curriculum, General Capabilities*. Retrieved from http://www.australiancurriculum.edu.au/generalcapabilities/critical-and-creative-thinking/introduction/introduction

Brennan, K., & Resnick, M. (2012, April). *New frameworks for studying and assessing the development of computational thinking*. Paper presented at the American Educational Research Association Annual Meeting, Vancouver, BC. Retrieved from http://web.media.mit.edu/~kbrennan/files/Brennan_Resnick_AERA2012_CT.pdf

Careers NZ. (2016). *Jobs in skill shortage*. Retrieved from https://www.careers.govt.nz/jobs-database/whats-happening-in-the-job-market/skill-shortage-jobs/result/it-and-telecommunications

Computer Science Teachers' Association (CSTA). (2011). *The Computational thinking leadership toolkit.* International Society for Technology in Education (ISTE). Arlington: VA

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

Doesburg, A. (2013, March 15). Does not compute—Where are the IT workers? *The New Zealand Herald*. Retrieved from http://www.nzherald.co.nz/business/news/article.cfm?c_id=3&objectid=10871184

Education Scotland. (2015). *Digital learning and teaching: Our vision for digital learning.* Retrieved from http://www.gov.scot/Topics/Education/Schools/ICTinLearning

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

Falloon, G. (2013b). Creating content: Building literacy skills in year 1 students using open format apps. *Computers in New Zealand Schools: Learning, Teaching, Technology, 25*(1–3), 77–95.

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

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

Falloon, G. (2016). *Building general thinking skills though computational tasks: Evaluating young students' experiences using Scratch Jnr.* Manuscript submitted for publication.

Krathwohl, D. R. (2002). A revision of Bloom's taxonomy: An overview. *Theory into Practice, 41*(4), 212–225.

Mayer, R.E., Dyck, J. & Vilberg, W. (1986). Learning to programme and learning to think: What's the connection? *Communications of the ACM, 29*(7), 605–610.

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

Pea, R. (1983). Logo programming and problem solving. *ERIC Technical Report No. 12.* Retrieved from http://eric.ed.gov/?id=ED319371

Pea, R. & Kurland, D.M. (1984a). Logo programming and the development of planning skills. *ERIC Technical Report No. 16.* Retrieved from http://files.eric.ed.gov/fulltext/ED249930.pdf

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

Rosenbaum, M. (2015). How to fix the tech talent shortage. *Infoworld*. Retrieved from http://www.infoworld.com/article/2969298/agile-development/how-to-fix-the-tech-talent-shortage.html

Wing, J. (2010). Computational thinking: What and why? *Carnegie Melon School of Computer Science Discussion Papers.* Retrieved from https://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf

Dr Garry Falloon is an associate professor in Te Kura Toi Tangata Faculty of Education, The University of Waikato. Garry is presently involved in post-graduate teaching, supervision, and research in digital technologies and eLearning. He is principal investigator for a 2-year Teaching and Learning Research Initiative (TLRI) project exploring student thinking development when using tablet devices collaboratively within inquiry and problem-based learning scenarios in the primary school

Email falloong@waikato.ac.nz

Paula Hale has 24 years' teaching experience. This was Paula's first year working in a collaborative environment and working for the first time working on a TLRI project. She is the assistant principal at Leamington Primary School in Cambridge. She has worked with all levels in a variety of primary schools, predominately in the junior school.

Tonia Fenemor is ICT team leader at Leamington Primary School, Cambridge. Tonia has 16 years' teaching experience, working mostly in the junior school with some time spent across levels. This was her first year working in a collaborative, innovative learning environment. She has a particular interest in teaching and learning with technology.