BUILDING COMPUTATIONAL THINKING THROUGH PROGRAMMING IN K-6 EDUCATION: A NEW ZEALAND EXPERIENCE

Garry Falloon

The Faculty of Education, University of Waikato (NEW ZEALAND)

Abstract

The recent inclusion of computational skills in core curriculum by governments in the UK and Australia, has been linked to industry calls for schools to better equip young people with capabilities and dispositions aligned with needs of future high-tech industries and rapidly changing workplaces. This move has stimulated much interest in New Zealand, and while lacking any compulsory curriculum mandate, many teachers in K-12 classrooms are exploring the potential of coding tasks for developing computational skills as part of their mathematics, science and technology curricula.

This paper reports findings from a study that used a unique data capture app embedded in iPads to record 9&10 year old students while they used two apps of very different designs for coding tasks. Using Studiocode video analysis software, data were analysed using a framework developed from Brennan and Resnick's [1] three dimensions of computational thinking, to learn more about how these apps constituted useful environments for developing computational thinking. Results suggest that coding apps of a 'teaching' design complement apps where computational concepts and practices are applied to project creation, and indeed may be more efficient if the desired outcome specifically targets the learning of concepts. Implications for teachers exploring coding apps for computational thinking development are drawn.

Keywords: computational, coding, thinking, practice, programming.

1   INTRODUCTION AND BACKGROUND

Back in 1970, education visionary Seymour Papert, at a symposium held at MIT entitled 'Teaching Children Thinking', presented arguments for young children using computers as thinking tools – that is, learning to programme computers "rather than being programmed by it" [2, p. 2]. Supporting this vision, Papert and his team at MIT developed the LOGO programming language that was used extensively for nearly twenty years by schools across the globe. LOGO was designed to teach young children the basics of computational thinking, through engagement in programming tasks that allowed them to create simulations and projects (Microworlds) using programmable spites. LOGO's design was one of a "low threshold and no ceiling" [3, p. 1], being flexible enough to be used by very young children, while offering enough challenge for more sophisticated and experienced users. However, despite LOGO's early success and inclusion at curriculum level in England in the late 1980s, it, and Papert's original vision for *children teaching computers,* never gained universal acceptance. Criticism of LOGO's complexity, low levels of teacher capability and poor access to computers in schools, at the time conspired against its visionary ideals becoming part of mainstream education.

Fast-forward 30 years. In March 2006, Jeannette Wing from Carnegie Mellon, in a Viewpoint article published in CACM, rekindled some of Papert's original vision for computational thinking, describing it as "a fundamental skill for everyone, not just computer scientists. To reading, writing and arithmetic, we should add computational thinking to every child's analytical capability" [4, p. 33]. Her arguments did not promote teaching humans to think like computers, but were broadly based, focusing on building the capacity of human cleverness when combined with computer capabilities to "tackle problems we would not dare take on before the age of computing" [4, p. 35]. Wing emphasised the importance of abstraction and conceptualisation, commenting that computational thinking is more than merely learning to programme to produce software artefacts. She saw it as a complex intellectual endeavour drawing on many disciplines – one that builds human capacity to organise, evaluate and solve problems, to manage our lives, and effectively interact with others. Wing's ideas are timely, and coincide with calls from technology industries for schools to play a greater role in seeding interest in computational activities in students from a young age, with the hope that they will continue later into technology-related careers, helping to address dire shortages of skilled workers in this area.

Supporting such moves, education authorities in a number of countries have implemented standalone compulsory digital technologies curricula, incorporating significant components focused on the development of computational thinking, represented mainly through student engagement in basic coding activities. These countries include England, Australia, Israel, Denmark and some German states, with others, such as the United States and New Zealand, looking to do likewise. Significant technological improvements and the greater ubiquity of digital devices in schools capable of being used to advance the goals of such curricula, mean that, in many cases, earlier barriers affecting schools' engagement with coding and other computational activities have been diminished. In many countries, initiatives such as BYOD (Bring Your Own Device) and digital tablet-supported classrooms, have provided teachers with more options for pursuing a wider range of learning activities involving technology. Affordable tablet devices and their huge array of supporting apps, means it is now viable to explore the potential of including computational thinking in K-6 education.

1.1 Defining computational thinking

According to Brennan and Resnick, "there is little agreement about what computational thinking encompasses, and even less agreement about assessing the development of computational thinking in young people" [1, p. 1]. However, a review of recent literature suggests that computational thinking is an intellectual process involving the development and exercise of a range of cognitive and applied knowledge and skills, including problem formulation, definition, analysis, abstraction, and logic, in the creation of solutions "that can be effectively carried out by an information-processing agent [5, p. 1]. Bers, Flannery, Kazakoff and Sullivan argue that instead of attempting to define computational thinking, it is more useful to conceptualise it as a set of capabilities and dispositions, "encompassing a broad and somewhat debated range of analytical and problem-solving skills, dispositions, habits and approaches used in computer science" [6, p. 145].

While historically learning to think computationally has been almost exclusively associated with coding or programming, researchers such as Howland, Good and Nicholson argue that its usefulness is much broader than that, commenting that many computational skills "easily translate to non-computing contexts" [7, p. 148]. They illustrate this by listing four generic skills that can be developed through computational activities. Briefly, these are an ability to: define clear and specific instructions for carrying out a process; design systems made of components each with a specific responsibility; design systems where components only reveal information or action related to their purpose; and understand that complex systems can result from the simple interactions of many components. Howland et al. [7] suggest such skills can be valuable in everyday life and work, such as generating accurate instructions for completing specialised tasks, or understanding the responsibilities of each department of a large business, and how these contribute to the functioning of the organisation. Lu and Fletcher adopt a similar stance, commenting that learning to think computationally should be seen as a valuable life skill, and that "we need to start teaching computational thinking early and often" [8, p. 261].

1.2 Learning and evaluating computational thinking

With improved access to digital devices and curriculum imperatives, teachers are increasingly exploring how computational thinking skills can be developed in their students through project-based programming tasks. Challenges exist, however, in assessing and evaluating this learning – that is, collecting and evaluating data that indicates what is being learnt, and what progress in computational thinking skill development looks like. Brennan and Resnick [1] propose a three-dimensional computational thinking framework, that they suggest provides guidance for what we should be looking for when evaluating student development in this area. The framework comprises student learning of computational *concepts* (concepts used when programming or developing code, such as sequences, parallelism, conditionals etc.); computational *practices* (strategies and techniques applied when developing code, such as debugging, reusing or sharing code etc.); and computational *perspectives* (views formed when and from building programmes, such as self expression, collaborating and connecting with others, and questioning). Their framework was developed from research on students' interaction with Scratch, "a programming environment that enables young people to create their own interactive stories, games and simulations, and then share their creations in an online community with other young programmers from around the world" [1, p.1].

They argue that when engaged in computational tasks, students iteratively and interactively draw upon and develop conceptual knowledge through adopting practices aligned with the work of programme designers. In the process, they also develop greater appreciation of the nature of the work as a form of self-expression, and the value of working with others when generating solutions. However, they comment challenges exist in unpacking the relationship between the three computational dimensions, and identifying how each contributes to and interacts with the others during the students' work [1]. Their evaluation of three approaches to this (project portfolio analysis using the Scrape analytical tool; artefact-based student interviews; and premade design scenarios students reviewed, debugged and offered suggestions for improvement) suggests this is indeed complex, with each method providing only part of the picture. Regardless, what Brennan and Resnick's [1] framework does provide, are some very useful descriptors that signal the sort of concepts and practices students engage with when completing computational tasks. These dimensions have been used in this study to evaluate student interaction with two different iPad apps, selected by their teachers to help build computational thinking.

2　RESEARCH GOAL AND QUESTION

The research goal was to evaluate two iPad apps of very different designs, in terms of their efficacy as environments for young students to develop and exercise computational thinking.

The question guiding data collection was:

To what extent do the iPad apps CargoBot and Pyonkee provide young students with environments within which to build and exercise computational thinking?

3　RESEARCH CONTEXT

Data were collected in a K-6 primary (elementary) school in a small semi-rural town in the Waikato region of New Zealand. The researcher had been working in the school for over 3 years, exploring its development of iPad-supported and BYOD classrooms since 2011. Outcomes from these earlier studies have been published extensively elsewhere (9, 10, 11, 12, 13, 14) with the innovative data method used in the earlier research being refined and adapted for this study. The student group comprised 63 nine and ten year olds (30 boys and 33 girls). They were not learning in single classrooms, but were part of the school's movement towards government-supported modern learning environments[1] (MLEs), where multiple classes work collaboratively together in a common space, supported by a number of teachers. Two teachers worked with the students in the MLE - they were Leesa who was a year 15 teacher, and Margaret who had been teaching for 35 years. The school had a BYOD policy operating in the MLE, where it was mandatory for students to have personal access to a digital tablet (in this case an Apple iPad). While most students used personally-owned devices, the school made available a number of loan devices for those who were unable to afford or bring their own. The spacious physical environment, furniture, and supporting technical infrastructure (wifi, large visual displays, Apple TV) had been specifically designed for student collaboration, and to optimise benefits from ubiquitous access to large numbers of mobile devices (Fig. 1).

The curriculum in the MLE was strongly student-focused, and included a mixture of more conventional learning tasks designed to build foundational competencies in numeracy and literacy, and topic-based, thematic studies designed around problem and inquiry-based learning models. Across all activities the emphasis was on building student learning independence and higher order and critical thinking capabilities, with both teachers being suitably skilled in appropriate pedagogy and questioning to support these goals. Data were collected over a total of 8 hour-long (approx.) sessions comprising part of the students' numeracy programme, but this was the first time they had engaged in tasks of this nature. The teachers had linked learning objectives to the geometry and measurement strand of New Zealand's National Curriculum in numeracy.

---

[1] See http://www.minedu.govt.nz/NZEducation/EducationPolicies/Schools/PropertyToolBox/StateSchools/Design/ModernLearningEnvironment.aspx

Fig. 1 The Modern Learning Environment (MLE)

3.1 The apps and their introduction

Two apps of very different designs were selected. These were CargoBot and Pyonkee, both being free apps available on Apple's App Store. Using CargoBot, students learn and apply a range of computational concepts including abstraction, conditionals, loops and subroutines to develop scripts that programme a robotic crane to move coloured crates to different arrangements on palettes, as specified by goals set by the app (Fig. 2). The goals and complexity of programming get progressively more difficult as the challenges progress. Points (stars) are allocated according to successful completion of each challenge, and are factored against the 'efficiency' of the programme's design.

Fig. 2 The apps used: Cargobot (left) & Pyonkee (right)

Pyonkee is based on the popular PC application Scratch, and is built from Scratch 1.4 open source code. It was selected due to its compatibility with iPads, and because it presented a very different approach to CargoBot for developing computational concepts and practices. In Pyonkee, students use pre-made blocks of visual code to create and animate scenes comprising characters and props. Students can opt to use characters, backgrounds, sounds etc. from the built in libraries, or import their own using the iPad's camera and microphone. Unlike CargoBot, students start with a 'blank canvas' (stage) upon which to build their environments using a wide range of code blocks to programme characters and props. These include motion, control, variables, sound, looks, operators, pen and sensing (Fig. 2).

Prior to commencing independent work, the teachers introduced each app separately to the students. CargoBot was introduced first, with learning goals shared and basic instructions provided about to how work the app (eg., drag and drop), what the various programme blocks did, and how to test and run procedures. Students were also introduced to and encouraged to access the built in tutorials. The introductory session was approximately 30 minutes in duration and supported by visuals delivered to large screen televisions using Apple TV. Following the introduction, students worked independently in pairs on the challenges in three hour-long sessions, over the course of a week. Debriefing and sharing took place at the end of each session. A similar process was used for Pyonkee, although the introduction was more comprehensive and focused on drawing parallels between computational concepts and practices used in CargoBot, and their 'adaptation' to Pyonkee. The students worked in pairs with Pyonkee for 5 hour-long (approx.) sessions over the course of 2 weeks. The first two sessions were exploratory, where students experimented with code blocks and other features building procedures of their own design, while in the remaining sessions they worked to a teacher-specified brief to build a learning game for younger students.

4   DATA COLLECTION AND SELECTION

Data were collected using a specially-developed recording app that ran in the background on the iPads, while students worked. Although the classrooms were BYOD, a set of 10 iPad Airs with the recording app pre-installed were provided by the university for data collection purposes, due to issues installing the recording system on personally owned devices. The app recorded as .mov files the display, finger placement (indicated as a white dot) and audio via the device's microphone. Although students had been informed of its use and ethical clearance had been gained via normal informed consent and assent processes, once activated, no visible indication of the operation of the recorder was available to the students. Using this system enabled the collection of very natural data - not influenced by researcher presence, from a large number of students engaged in the same task, over a relatively brief period. Recordings were stored on each iPad, and later extracted to the researcher's laptop using iExplorer.

Due to the exceedingly large volume of data produced by the recorder, files were selected for analysis based on the following criteria:

1. They totalled five hours of recordings of students using each app (10 hours in total);
2. They represented a balance of students' interactions from across the sessions (ie., early work, later work);
3. They were reflective of the gender balance of the classes;
4. They included a balance of year 5 and year 6 students' interactions.

4.1 Coding

A coding system based on Brennan and Resnick's [1] evaluation framework was developed into a template built in Studiocode video analysis software, and was applied to the selected data (Fig. 3). Studiocode supports analysis of events contained in video data according to user-generated codes, presenting them as timelines from which statistical data such as average and total duration, event counts, and percentages of total time (etc.) can be calculated. Fig. 3 shows part of the coding template, timeline and recorded video for two students (R&M). The coloured arrows in the template are activation, deactivation and dependency links that enable concurrent event coding against different codes. These were important for mapping relationships between events, such as students collaborating (purple) when engaged in building or testing procedures (orange and green) or conceptualising the task (blue); or using experimentation (yellow) as a strategy when building procedures (orange).

The coding template was adjusted many times before finalising, responding to behaviours, strategies and occurrences revealed in data. To assist with this, a post-doctoral research assistant was employed to blind review data, in addition to the researcher. The ensuing discussion led to five main code categories being agreed upon, each comprising a range of sub codes against which data were coded. These are outlined and colour linked to the code template in Table 1. Some of Brennan and Resnick's [1] original dimensions were not included in the template as little or no evidence of them could be found, while others were added that reflected or described the actual strategies and practices used by these students (eg., stringing, problem conceptualisation/analysis and different ways this occurred).
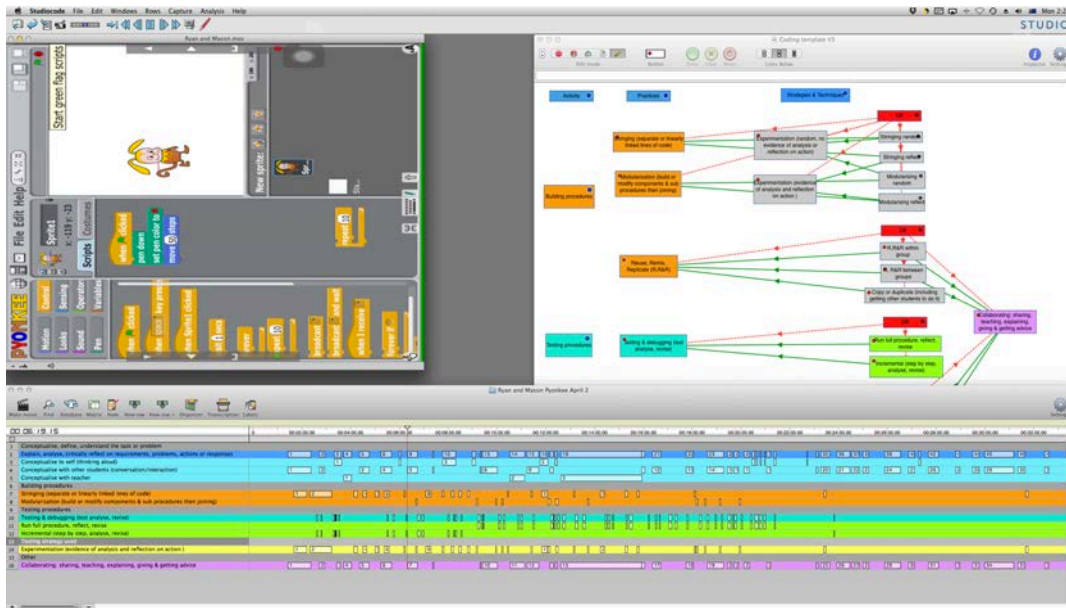
Fig. 3 The coding template (right), video (left) and event timeline (bottom)

Data were then double blind coded in Studiocode by the researcher and the assistant, following which a selection of 200 occurrences were subjected to a Kappa rater-agreement calculation. While these occurrences were selected at random, care was taken to ensure all codes were represented, and that at least some data aligned with each selection criterion were chosen. Only data both coders had identified were used, to avoid underestimation of agreement probability through inclusion of missing data [15]. The calculation yielded agreement across 168 occurrences with a Kappa of κ=.671 (SE=0.053; CI=.95 from 0.567-0.665). This rated as 'good' on Landis and Koch's scale [16].

Table 1. Code categories, descriptors and codes

Category	Description	Codes (generated from data)
Concept/Practice		
Building procedures	Selecting and assembling code elements or blocks into sequences	Stringing (separate or linearly-linked code elements)
		Modularisation (grouping elements into modules and linking modules to form procedures)
		Reuse, recycle, remix within or between pairs (R,R&R)
Testing and debugging procedures	Evaluating the performance of sequences by running procedure. Detecting & rectifying programming errors	Run full procedure, reflect (or not) on result and revise
		Incremental (test procedure by element or line using stepper or similar function), reflect (or not) on result and revise
Conceptualisation	Explain, analyse, critically reflect on requirements, problems, actions or responses	Conceptualise to self (thinking aloud)
		Conceptualise with other students (interacting/conversing)
		Conceptualise with teacher
Collaboration	Sharing, teaching, explaining, giving & getting advice	Collaboration between members of pairs and/or between pairs
Experimentation/ trialling	Strategy used mainly when testing & debugging procedures	Evidence of analysis and reflection on outcome used when debugging (deliberate/considered)
		No evidence of analysis and reflection on outcome used when debugging (random/guessing)

5 RESULTS

Statistical summaries for the sample data against each code were generated in Studiocode and exported to Excel for further analysis (see example in Table 2). These contained total count times, total time spent, percentage of overall task time, and the mean time per code for each pair. In most cases the percentage of task time exceeds 100%, as events often triggered multiple codes. This was particularly the case with data coded as 'experimentation' and 'collaboration', where these were linked as practices students applied to other computational activities, such as debugging, conceptualising or building procedures. Table 2 contains data summaries for four different pairs, two from CargoBot and two from Pyonkee. The far right columns separated from the pair data contain average count and percentage summaries *for all coded data*.

Table 3 provides illustrative data that were coded under some of the categories. It contains the category and code, an explanation of the scenario surround the event, a thumbnail from the display video, and a transcription of the dialogue recorded at the time. Due to space constraints not all categories and their codes have been included, but the table demonstrates the nature of data coded and illustrates how coding decisions were made.

Table 2. Example of summary data for four pairs. Averages for all groups at right

		Mania & Shae Pyonkee				Munit & Natalie Pyonkee				Averages	
		Count	total time	%	mean time	Count	total time	%	mean time	Count	Time (%)
	Conceptualise, define, understand the task or problem										
	Explain, analyse, critically reflect on requirements, problems, actions or responses	31	00:15:55.00	54.25	00:00:30.80	23	00:16:02.38	47.13	00:00:41.84	31.8	49.356
	Conceptualise to self (thinking aloud)	2	00:00:39.47	2.24	00:00:19.73	1	00:00:13.88	0.68	00:00:13.88	5	2.564
	Conceptualise with other students (conversation/interaction)	22	00:10:51.26	36.99	00:00:29.60	15	00:08:07.23	23.86	00:00:32.48	22	32.946
	Conceptualise with teacher	7	00:04:29.66	15.32	00:00:38.52	7	00:07:41.25	22.59	00:01:05.89	5	14.058
	Building procedures										
	Stringing (separate or linearly linked lines of code)	11	00:01:46.05	6.02	00:00:09.64	15	00:01:45.48	5.17	00:00:07.03	13.6	7.782
	Modularisation (build or modify components & sub procedures then joining)	0	00:00:00.00	0	00:00:00.00	10	00:01:01.05	2.99	00:00:06.10	8	4.164
	R,R&R within or between groups	0	00:00:00.00	0	00:00:00.00	2	00:00:20.28	0.99	00:00:10.14	2	0.99
	Testing strategy used										
	Experimentation (evidence of analysis and reflection on action)	11	00:01:46.05	6.02	00:00:09.64	25	00:02:46.54	8.16	00:00:06.66	19.6	9.946
	Testing procedures										
	Testing & debugging (test analyse, revise)	11	00:00:40.59	2.31	00:00:03.69	35	00:03:39.75	10.76	00:00:06.27	25.4	6.362
	Run full procedure, reflect, revise	0	00:00:00.00	0	00:00:00.00	22	00:02:37.60	7.72	00:00:07.16	13.4	3.694
	Incremental (step by step, analyse, revise)	11	00:00:40.59	2.31	00:00:03.69	13	00:01:02.15	3.04	00:00:04.78	12	2.67
	Other										
	Collaborating: sharing, teaching, explaining, giving & getting advice	29	00:15:16.76	52.08	00:00:31.61	23	00:15:09.57	44.54	00:00:39.54	27.4	46.56
		Ben & Cameron Cargobot				BJ & Charles Cargobot				Averages	
		Count	total time	%	mean time	Count	total time	%	mean time	Count	Time (%)
	Conceptualise, define, understand the task or problem										
	Explain, analyse, critically reflect on requirements, problems, actions or responses	89	00:28:37.32	48.88	00:00:13.52	82	00:17:13.31	29.88	00:00:12.60	63.6	29.126
	Conceptualise to self (thinking aloud)	55	00:06:39.94	11.38	00:00:05.71	43	00:06:22.77	11.07	00:00:08.90	29.2	5.744
	Conceptualise with other students (conversation/interaction)	33	00:22:21.28	38.18	00:00:23.53	36	00:10:13.81	17.75	00:00:17.05	29.8	21.008
	Conceptualise with teacher	1	00:00:16.18	0.46	00:00:16.18	3	00:00:27.10	0.78	00:00:13.55	4.6	2.566
	Building procedures										
	Stringing (separate or linearly linked lines of code)	24	00:03:16.40	5.59	00:00:08.18	24	00:04:52.95	8.47	00:00:12.20	31.2	8.158
	Modularisation (build or modify components & sub procedures then joining)	45	00:06:01.28	10.28	00:00:08.02	25	00:06:21.50	11.03	00:00:15.26	36.8	9.972
	R,R&R within or between groups	2	00:01:17.46	2.2	00:00:38.73	1	00:00:24.98	0.72	00:00:24.98	2.2	2.254
	Testing strategy used										
	Experimentation (evidence of analysis and reflection on action)	66	00:08:50.93	15.11	00:00:08.04	45	00:10:55.93	18.97	00:00:14.57	63	17.208
	Experimentation (random, little or no evidence of analysis or reflection on action)	3	00:00:19.91	0.57	00:00:06.63	4	00:00:18.51	0.54	00:00:04.62	4.8	0.752
	Testing procedures										
	Testing & debugging (test analyse, revise)	80	00:04:35.08	7.83	00:00:03.43	62	00:07:16.74	12.63	00:00:07.04	82.8	13.822
	Run full procedure, reflect, revise	6	00:00:42.30	1.2	00:00:07.05	16	00:02:19.80	4.04	00:00:08.73	23.4	4.9
	Incremental (step by step, analyse, revise)	74	00:03:52.77	6.63	00:00:03.14	46	00:04:55.00	8.53	00:00:06.41	59.6	8.79
	Other										
	Collaborating: sharing, teaching, explaining, giving & getting advice	60	00:22:46.39	38.89	00:00:22.77	39	00:11:20.56	19.68	00:00:17.45	45.2	24.594

6 DISCUSSION

When considering these results and reflecting on the research goal, it is important to remember the purpose of this study was not to rate one app as being *better or worse* than the other, but rather to evaluate their usefulness for developing computational thinking in young students. To that end these results suggest both apps are useful, but for building different dimensions of computational thinking. Specifically, they tentatively suggest that where the main learning objective is to develop a more technical understanding of computational concepts, such as coding procedures or scripting, and some practices such as testing and debugging, a more structured environment such as that offered by apps like CargoBot, may be more efficient.

Examining the averages across all pairs, significantly more time was spent applying computational concepts including stringing and building code modules, and some practices such as reworking or reusing code, analysing, testing and debugging etc. in CargoBot (approx. 34%), than in Pynokee (approx.18.5%). This also applied to strategies students used when engaged in these practices, particularly experimentation involving analysis and reflection on action or outcome (CargoBot approx.

17.2%; Pyonkee, approx. 9.9%). However, when evaluating other practice dimensions such as conceptualisation and levels of collaboration, Pyonkee (approx. 49% & 46% respectively) appeared to hold some advantages over CargoBot (approx. 29% & 24.5%).

Table 3. Data samples: Scenario, thumbnail and dialogue coded by category

Category	Code	Scenario description	Thumbnail	Recorded dialogue
Building procedures	Sequencing/ Stringing	Students G&J required to sort coloured crates according to goal. They had built multiple single code lines (PROGs 1-3) for each movement stage, linking lines using the appropriate PROG block.		*I think we are going to run out of space… (long pause)… that'll give us three piles… but we've still got to move the last 2 across (to the right-end pile) (J).* *Ummm... this is really hard... Why have they cut off the bottom one? (PROG 4). It should have the same spaces as the others (laughing)…. (L)* *We could do it easy then! (J).*
Conceptualisation	Conceptualise with other students	Students G&J required to sort coloured crates according to goal. They had developed two string sequences (PROGs 1&2) and tested them, They were creating a third sequence (PROG 3) following the same pattern, but had realised doing this would not allow them to move all crates in the available code space. J is building sequence.		*It's not going to work (G)…. (pause) you can't do that….* *Do what? (J)…* *Make it like that… in rows like that… and it's wrong…anyway…. (G). Why? (J)* *'Cos it won't do it… see… what's it going to do after PROG 1? Think about it stupid (sic)… it's going to go to PROG 2 and then pick it up and go off the end… 'cos you've told it to go across (to the right)… (G) (pause) OK… I get it… (pause)… it needs to go back and do it again… (J) Yeah, you have to repeat the same thing four times… (G).*
Experimentation	Experimentation – evidence of analysis or reflection on action or result	Students R&M were building their shape drawing game. They were trying to get their spite to draw a square, and had trialled sequence as per thumbnail. They were discussing the result.		*Ummm… it didn't do 10 steps (R)…* *Yes it did… see the line… it drew that… (M).* *That's not big enough… you can hardly see it!… OK… so we need to make it bigger… what number should we try? (R).* *Let's make it 100… we'll try 100 and see how far it goes… (M).* *Do you think that'll be enough? See how little 10 is… and it'll only be 10 lots of that… so it'll still be pretty tiny… (R).* *Go for 200 then… (M).*

Testing & debugging	Incremental – step by step	N&M are finalising their shape game and had scripted the chipmunk sprites. They were testing the length of 'wait' time before each sprite began to draw its shape. They had set this initially to 4 seconds and had run the relevant part of the script and were discussing the result.		*I think 4's too long for the juniors… they'll think nothing's happening (N)…* *But we need to give them a bit of time to look… and you know… they muck around a bit (M)... (they test 4 seconds again) (pause)….* *OK… it's probably too long… umm… 2? (pause)… shall we give 2 a try? (M)… (they test 2 seconds)… (pause)…* *That might be a bit quick… do 3… (N).*

The recorded display and audio data offered insights into possible reasons for these differences. Although both apps were developed to promote computational thinking, their designs approached and structured this in fundamentally different ways. First, CargoBot presented the students with fewer distractive options that could draw their attention away from the core task of programming the robotic crane. Unlike Pyonkee, there was no capacity to change any elements of the task, its appearance, or the environment, and apart from different difficulty levels, students had few customisable options. In Pyonkee, however, the students were able to develop and programme environments of their own design. They were confronted with multiple options for sprites, backgrounds, costumes, props and so on that they could use for this purpose, and then even customise content once selected, using the app's built in art tools. Alternatively, they could choose to import and manipulate their own content from the iPad's visual media tools. Display data indicated considerable time was spent by some pairs negotiating and conceptualising their project's content, and while this is an important part of building a project, in this limited trial it appeared to take time away from more formal coding activities.

Second, Pyonkee's design appeared to stimulate higher levels of collaborative interaction than CargoBot. While some of this linked to the students' decision-making when conceptualising their projects and deciding how to go about it, there were also differences revealed by recordings that appeared related to what they were required to do, and the extent of expertise each could contribute to completing parts of the task. There was a tendency in some CargoBot data for one pair member to lead the problem solving, while the other served more as an observer or checker, less often offering input to coding decisions. This observation is supported by the significantly higher average percentage of data coded as *conceptualise to self-thinking aloud* in CargoBot, where one student was recorded verbalising problems or planning next steps quietly to themselves, with no obvious engagement or participation from the other. Display data also indicated limited passing of the device between pair members, suggesting that the process of building code sequences was physically less collaborative. While reasons for this are yet to be explored, it may have been that one of the pair was judged to posses higher levels of 'computational capability' that could be applied to the challenge, or simply it could be that one didn't find as much motivation in solving the challenge as the other did, thus choosing to minimally engage at times. On the other hand, Pyonkee offered a much more diverse array of options of a less specialised or technically-complex nature, that potentially both students could contribute to. Designing stage layouts, deciding on sprites and costumes and developing procedures to animate characters appeared highly engaging for all students, who displayed high levels of intra-pair collaboration whilst doing so. The open, 'blank canvas' nature of Pyonkee undoubtedly contributed to this, as students were unconstrained by the pre-programmed structures and parameters of CargoBot, which limited their choices.

Third, notably higher average percentages of time spent testing, debugging and editing procedures in CargoBot, appeared also to be related to the demands of the task and the app's design and functionality. While both apps employed 'drag and drop' to build and edit sequences, students appeared to find debugging easier in CargoBot due to its single-screen layout, and the presence of an easy to use 'stepper' tool that supported command-by-command testing and debugging of sequences. While this was possible in Pyonkee it was more difficult to carry out, and the need to navigate through multiple code windows (control, sensing, motion etc.) to debug issues on a sprite-by-sprite basis, was more challenging. To a large extent these design influences could be expected, given the different nature of the task requirement and challenges presented in each case, and the openness (or not) of how students could go about achieving them.

Fourth, it was encouraging to note across students' interaction with both apps exceptionally low levels of random experimentation or trial and error based on guesswork. Unlike earlier studies by the author evaluating student interaction with learning game apps [14], recordings revealed almost no occurrences of this in CargoBot (0.75%) and none at all in Pyonkee, suggesting problem solving and debugging was a far more deliberate and reflective process. This conclusion finds support in the oral exchanges coded as *conceptualise with other students* and *conceptualise with teacher,* where substantial evidence exists of students using higher order thinking (reflection, analysis, evaluation) to develop sequences and revise them after trialling. Table 3 provides one illustration of this in sample data coded as *Experimentation*. Recordings also provide some evidence of computational concept and practice transfer between the apps, although the exact number of occurrences of this was not logged. Evidence that transfer of computational concepts such as repeats, loops and conditionals had taken place, was recorded in the oral exchanges between some students. Although relatively infrequent, comments such as "remember, you need to repeat it... like we did with the crane… you know how we did it with PROG2 and stuff… don't just pile them up…use the repeat block…" (students BJ&R, 21.05) indicated at least some conceptual transfer. While this finding appears promising, further research is needed to determine the extent of this, and if and how it might translate to other computational activities based on coding.

7   SUMMARY AND CONCLUSION

Recognising the obvious limitations of this study, results suggest that there is a place for apps of both teaching and creative designs in the development of computational thinking in students. They signal the likely value of students learning basic computational concepts and practices in more structured environments as provided by apps like CargoBot and LightBot, and for the very young, Kodable and Daisy the Dinosaur, before their creative application in project building using apps such as Pyonkee, Scratch and Scratch Jnr. Restrictions in the array of options and greater emphasis on learning technical concepts and dimensions of coding in more structured, challenge-based apps, appeared to support efficient acquisition of computational concepts and practices useful for project building. The emerging evidence of concept and practice transfer revealed in this study, provides some support for this position.

Using display and audio capture to gather data provided fascinating insights into the unsupervised work of students on their computational tasks. As evidenced by the recorded dialogue, it was apparent most pairs either forgot about or disregarded the operation of the recorder while engrossed in their challenges. There was no evidence of 'staged' performances that may have resulted from other data collection methods such as observation or external video, and the method was highly efficient, enabling collection of a substantial volume of data within a relatively defined timeframe. An interesting side benefit from using this data method, was that it drew attention to how engaged these students were in their work. While formal on/off task analysis was not the purpose of this research, data strongly indicated exceedingly high levels of task application, with minimal time wastage or unconnected activity. Although some expressed their frustration and annoyance at not being able to solve problems or easily debug scripts (leading to temporary disengagement in a few cases), this was the exception, not the rule. The vast majority displayed much tenacity and perseverance as they confronted challenges, applying at times quite sophisticated thinking and collaboration skills to solving them. Reflecting on this in relation to evaluating computational thinking performance, an expansion to Brennan and Resnick's [1] framework could acknowledge the application of higher order thinking, the importance of dispositional characteristics, and the role of collaboration in developing computational thinking capability. While understanding computational concepts and practices is important, these alone may be of limited use in the absence of cognitive strategies and dispositional characteristics that support their effective application to tasks. Using display captures to gather data for evaluating computational thinking development could be a useful means of unpacking the relationship between these dimensions.

Future research as part of this project will adopt a similar methodology to investigate the nature and type of thinking 5 and 6 year olds engage in when working in pairs with basic coding apps. It will combine a thinking types framework from an earlier study by the author [9] with ideas from a computational thinking analysis model developed by Selby [17], to explore the nature of student thinking when engaged in the different dimensions of computational tasks. It is hoped these studies will shed light on if and how computational tasks may act as vehicles for developing a range of higher-order thinking capabilities.

ACKNOWLEDGEMENT

REFERENCES

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

[2] Blikstein, P. 2013. Seymour Papert's Legacy: Thinking about learning, and learning about thinking. *Seymour Papert Tribute at IDC 2013*. Available: https://tltl.stanford.edu/content/seymour-papert-s-legacy-thinking-about-learning-and-learning-about-thinking

[3] Logo Foundation. 2013. *What is Logo?* Available: http://el.media.mit.edu/logo-foundation/logo/

[4] Wing, J. 2006. Computational Thinking. *Communications of the ACM, CACM*, vol. 49, no. 3, pp. 33-35.

[5] Cuny, J., Snyder, L. & Wing, J. 2010. *Demystifying computational thinking for non-computer scientists.* Unpublished manuscript, referenced in http://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf

[6] Bers, M., Flannery, L., Kazakoff, E. & Sullivan, A. 2014. Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. *Computers & Education*, vol. 72, pp. 145-157.

[7] Howland, K., Good, J. & Nicholson, K. 2009. Language-based support for computational thinking. *In 2009 IEEE Symposium on Visual Languages and Human-Centric Computing.* Corvallis, OR, USA, pp. 147-150.

[8] Lu, J. & Fletcher, G. 2009. Thinking about computational thinking. *Paper presented at SIGCSE '09*, Chattanooga, Tennessee, USA.

[9] Falloon, G.W. in press. iPads, apps and student thinking skill development. *Educational Technology & Society.*

[10] Falloon, G.W. 2015. What's the difference? Learning collaboratively using iPads in conventional classrooms. *Computers & Education*, vol. 84, pp. 62-77.

[11] Falloon, G. & Khoo, E. 2014. Exploring young students' talk in iPad-supported collaborative learning environments. *Computers & Education,* vol. 77, pp. 13-28.

[12] Falloon, G.W. 2013. Creating content: Building literacy skills in year 1 students using open format apps. *Computers in New Zealand Schools: Learning, Teaching, Technology,* vol. 25, no. 1-3, pp. 77-95.

[13] Falloon, G.W. 2013. What's going on behind the screens? Researching young students' learning pathways using iPads. *Journal of Computer-Assisted Learning*, vol. 30, no. 4, pp. 318-336.

[14] Falloon, G.W. 2013. Young students using iPads: App design and content influences on their learning. *Computers & Education*, vol. 68, pp. 505-521.

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

[16] Landis, J. R. & Koch, G. G. 1977. The measurement of observer agreement for categorical data. *Biometrics,* vol. 33. no. 1, pp. 159–174.

[17] Selby, C. 2012. Promoting computational thinking with programming. *Proceedings of the 7th Workshop in Primary and Secondary Computing,* pp. 74-77. Available: http://dl.acm.org/citation.cfm?id=2481466