International Journal of Computer Science Education in Schools, Oct 2017, Vol. 1, No. 4
ISSN 2513-8359

Classroom Talk and Computational Thinking

Craig W Jenkins

University of South Wales

craig.jenkins@southwales.ac.uk

DOI: 10.21585/ijcses.v1i4.15

Abstract

This paper is part of a wider action research project taking place at a secondary school in South Wales, UK. The overarching aim of the project is to examine the potential for aspects of literacy and computational thinking to be developed using extensible 'build your own block' programming activities. This paper examines classroom talk at an extracurricular programming club and focuses in particular on dialogue relating to computational thinking. Learners spent a number of weeks carrying out an activity designed using the Snap! programming tool. The activity was themed around language and the task was to devise a collection of fixed-form poetry.

The findings are in two parts. First is a dialogue analysis using the SEDA coding scheme. This analysis revealed a number of learner interactions showing evidence of reasoning. Second, examples of talk sequences are provided in order to examine how the reasoning identified in the interactions relate to what we may recognise as computational thinking. The paper concludes by considering how dialogic approaches in the classroom potentially have an important role to play in the process of teaching young people to think computationally.

Keywords: computational thinking, Snap!, dialogic teaching, digital competence, seda.

1. Introduction

Given its increased prominence as a national priority for school improvement and its statutory designation as a responsibility of all teachers, literacy figures as an important area of cross-curricular development for secondary school teachers in Wales. Computational thinking, meanwhile, is now higher on the educational agenda than at any time since the early 1980s with the introduction of the Digital Competence Framework (*see* Welsh Government, 2011; 2016). This study is part of a wider action research project investigating how defined aspects of literacy and computational thinking may be developed through extensible block-based programming activities. The institutional context is a secondary (11-18) school in South Wales, UK. This aim of this study in particular is to investigate the role of teacher-learner and peer classroom dialogue in the development of computational thinking.

Educators in computing education will be familiar with *Scratch*, a block-based programming tool that allows the learner to piece together chunks of a program in an experience just like slotting together Lego bricks. *Build Your Own Blocks* (BYOB) and its browser-based edition *Snap!* (Mönig and Harvey, 2009) is an offshoot of Scratch that enables teachers (and learners) to build their own custom programming blocks to supplement the standard set of Scratch blocks. This means that programming activities can be customised to make accessible particular ideas within domains of knowledge. Like Scratch, *Snap!* is a tool that is freely available to all.

The context of the study draws on experiments with text-based programming languages carried out by Sharples (1985) and Goldenberg and Feurzeig (1987). Sharples implemented three programs in LOGO that were aligned with a philosophy of exploratory learning and the subject domain of written English. One program that Sharples created was PAT: a word pattern generator. Learners first specify a structure of sentence components and a bank of different word classes. The computer, in turn, combines words and generates random phrases according to the specified sentence structures. By specifying different phrase structures and then generating sentences with random word selections, the idea is that learners are able to observe the regularities in language and gain an understanding of its linguistic rules. Sharples' work generated some evidence to support the idea that programming with PAT can lead to gains in written English. Using a feature analysis, Sharples was able to measure the maturity of writing features in essays that were written before and after his teaching scheme intervention. He saw an increase (though not a statistically significant one) in the mature writing features in comparison to the control group (*ibid.)*

While Sharples' work focused primarily on storytelling, Goldenberg and Feurzeig (1987) also approached programming from the perspective of simple poetry. Goldenberg and Feurzeig (1987) explored how one might go about creating a program to produce different lines of text in a poem, which is replicated below (p. 32). They use the famous 'Roses are Red' poem to illustrate.

```
TO VIOLETS
    PRINT    [ROSES ARE RED.]
    PRINT    [VIOLETS ARE BLUE.]
    PRINT    PRAISE
    PRINT    RHYMOO
END

TO RHYMOO
    OUTPUT (    SENTENCE    [AND]    WHO MOO    )
END

TO MOO
    OUTPUT PICK [    [DOES TOO]    [LOVES YOU]    [SNIFFS GLUE]    [SIPS DEW] ]
END
```

The program here is made up of three different procedures. The procedure VIOLET is used to create the overall structure of the poem. We can see that this simply prints four lines. Notice that the first two lines will always remain the same: these will always say 'roses are red' followed by 'violets are blue'. The third line refers to a procedure called PRAISE that we do not see here. PRAISE is where a compliment will be paid to the poem's recipient on Valentine's Day. The final line of VIOLETS calls RHYMOO. RHYMOO is used to produce a sentence with a random '-oo' sound ending each time. This is achieved in two stages. First, a sentence beginning is provided by RYHMOO. Second, MOO randomly picks a sentence end from, in this case, a choice of four: 'does too', 'loves you', 'sniffs glue' or 'sips due'.

The mechanism above is important because it enables poems to be created and analysed using multiple levels of abstraction. Wing (2006) reminds us that this is an important part of learning to think computationally. Here, the ability to think using multiple levels of abstraction is evident where learners create a set of rules to define both (i) the poetry form itself and (ii) the lines within that poetry form.

This paper adapts these pedagogical ideas for a block-based implementation using *Snap!*. Learners were provided with access to a bespoke Snap! block-kit in order to develop a collection of poems. The activity was called the 'Poem Generator'. The block-kit was designed to promote aspects of literacy *and* computational thinking development, though only the latter is reported in this paper.

There are a multiplicity of frameworks attempting to define *computational thinking*. Brennan and Resnick's

(2012) framework is specifically constructed with the block-based Scratch programming language in mind. The activity described here takes place using Snap! (an offshoot of Scratch) and so it is logical to use Brennan and Resnick's framework. The framework defines computational thinking along three dimensions: computational *concepts*, *practices* and *perspectives* (*see* ibid. for a discussion of these terms). The dialogue excerpts contained in the results focus on how this activity developed three specific computational thinking *practices*: (i) testing and debugging; (ii) abstracting and modularizing and (iii) reusing and remixing.

2. Method

This study takes places at an extracurricular programming club at a secondary school that ran over the course of one full term of around twelve weeks during the academic year.

2.1 School setting

In Wales, all schools are required to be periodically inspected by Estyn, Her Majesty's Inspectorate for Education and Training and Wales, under section ten of the 1996 Schools Inspection Act. The contextual data here summarizes a recent report into the quality and standards of provision at this education provider (Estyn, 2016).

School X is a large English-medium secondary school that is maintained by a Unitary Authority in the South East region of Wales, UK. It is a large school, with 1,521 mixed-gender learners on roll between the ages of 11 - 18.

The school admits students with a varying range of abilities. In Wales, students experiencing learning difficulties are placed on a register of Special Educational Needs (SEN). The percentile of compulsory school age learners appearing on the SEN register for School X was 17% at the time of the inspection. This is nearly half that of the National comparator of 25.1%.

In Wales, the percentage of learners entitled to free school meals (FSM) serves as a measure of social and economic deprivation within a school's intake. The FSM measure for School X is 6% of students on roll. This is significantly lower than the National comparator of 17.4%. A majority of learners on roll come from geographical areas of socioeconomic advantage.

2.2 Participants

Learners were recruited through advertisements in weekly year group assemblies so there is no element of random assignment in the participant group. Attendance at the club was variable with around six-to-ten Year 7 (ages 11 to 12) and Year 8 (ages 12 to 13) students voluntarily opting to attend each session. The learners who attended remained relatively constant from the beginning: a core group of around six learners attended every session. The sample was mixed gender and mixed ability.

As this formed part of an action research project, the teacher-researcher is employed as a teacher of ICT in the secondary school where the research takes place and was the leader of the intervention. Even though the club was carried out at the school where the teacher-researcher works, an independent academic from the University of South Wales also attended the club for the majority of the sessions. This adds an element of inter-judge reliability (*see* Black, 2002) to the reported findings.

2.3 Procedure

When attending the club, learners were placed into pairs. The rationale for this was clear: there is evidence to suggest that pairwork is an effective pedagogical tool when learning to program. According to Cockburn and Williams (2009), the pair programming approach can contribute to a faster solving of problems and also a greater enjoyment of the learning process. Further, other evidence has suggested that working in pairs to solve a programming problem can lead to participants acquiring the technical skills that are demonstrated to them by their collaborator (see Chong et al., 2005).

Learners were provided with a simple task brief as an ongoing project during a number of lunchtimes. They needed to construct a computer program for producing a collection of tanaga poetry. The tanaga is a type of short Filipino poem, consisting of four lines of seven syllables each with the same rhyme at the end of each line. The poem uses a 7-7-7-7 syllable form with an AAAA rhyme pattern. The activity they completed in Snap! can be described by referring to three key steps.

2.3.1 Step 1 – Working with the word lists

The first step of making the poems involved choosing the words to appear in them. There were several 'boxes'

or lists of different types of word sorted by either word class or word ending. Learners could choose to either add existing words to these lists or create their own lists with different types of words. The word lists appeared on the Snap! Stage (*see* figure 1).

Figure 1. Working with the word lists

2.3.2 Step 2 – 'Programming' the poems

After the words had been entered, sentence patterns for the poem need to be input. The 'generate' C-shaped block in the scripting area controls the poem generator on the stage. Multiple poems can be repeated using the numeric input contained within the block. Each time a 'write' block is used, a new line will be started. Within each line, some words can remain constant each time. It is also possible to pick a random item from one of the word lists (*see* figure 2).

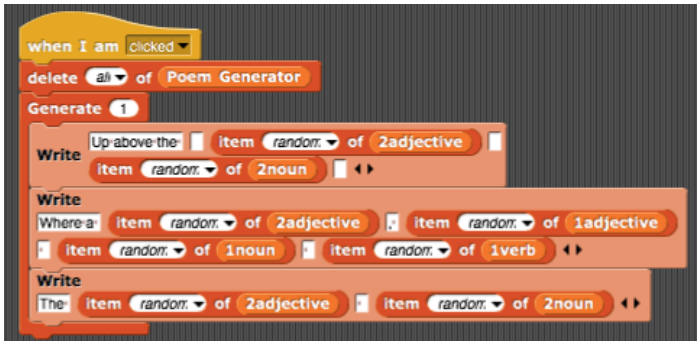


Figure 2. 'Programming' the poems

2.3.3 Step 3 – Generating poems

The final step is to link the sentence patterns to the words in the word lists. When learners click the 'generate' button, the poems are generated on the stage. Where an 'item random' block has been used, this will be substituted by picking one word from the corresponding word list (*see* figure 3).

Figure 3. Generating poems

Voice recordings were made of the pairs at key points during the club sessions. Each recording typically lasted for around ten minutes and were made with different pairs in the club at key points over the term.

Learners were made aware that that the recordings would be transcribed and then immediately deleted. Learners were also informed that all reporting would be anonymous. Ethical consent was gained from all participants and their parents/carers. Further, ethical approval was sought from the Senior Management Team at the school in addition to the University's Research Ethics Committee.

2.4 Data Analysis

Following the transcription of the audio recordings, the pairwork and learner-teacher sequences were analysed using the SEDA coding scheme (*see* Hennessy et al., 2016). The result of a three-year project in UK and Mexico, the SEDA scheme begins by establishing a hierarchical and nested model for analysing dialogue. The SEDA scheme identifies communication on a micro, meso and macro level of communicative acts (CA), communicative events (CE) and communicative situations (CS) respectively. A CA is one contribution within a conversation. Several CA about a particular topic or task form a CE. Each turn in the conversation to a different topic signals a change of CE. Finally, the overall general context of the conversation is the CS and this may be comprised of multiple CE.

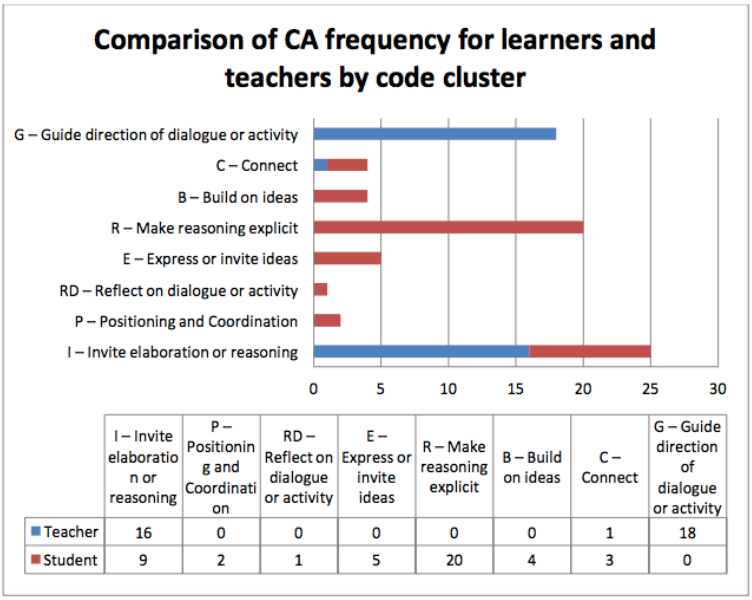
The analysis used in this paper is fine-grained in that it largely restricts itself to a micro-level analysis of CA only. The CS of the analysis is the conversation that takes place within the club session with the different excerpts of dialogue being separated into different CE. In order to carry out the micro-level analysis of the different CE, each interaction was assigned one of thirty-three codes that were in turn part of eight clusters. It was therefore possible to gain a deeper grain of understanding with respect to the kinds of peer talk exchanges that were taking place whilst the activities were being carried out.

3. Findings and discussion

The analysis of the dialogue between teacher and learners during the club begins with a preliminary statistical overview of the clusters for all CA recorded before moving to discuss examples of individual CE that were recorded.

3.1 Overview Cluster Analysis

Figure 4 shows the CA cluster frequencies organised by their agents of either teacher or learner.



	I – Invite elaboration or reasoning	P – Positioning and Coordination	RD – Reflect on dialogue or activity	E – Express or invite ideas	R – Make reasoning explicit	B – Build on ideas	C – Connect	G – Guide direction of dialogue or activity
Teacher	16	0	0	0	0	0	1	18
Student	9	2	1	5	20	4	3	0

Figure 4. Comparison of frequencies for learners and teachers by code cluster

It is clear to see that teacher dialogue was considerably less diverse in the scope of CA and clusters in comparison to that shown by learners. Teacher dialogue was largely restricted to either cluster G (where guidance was issued) or cluster I (where reasoning was invited). Table CE1 provides an example of an interaction in the former cluster: here we see how teacher prompting is used to help Ethan debug his program. By making use of careful questioning, a lot of teacher time was spent inviting learners to think carefully about the task in hand and the steps that should be followed next.

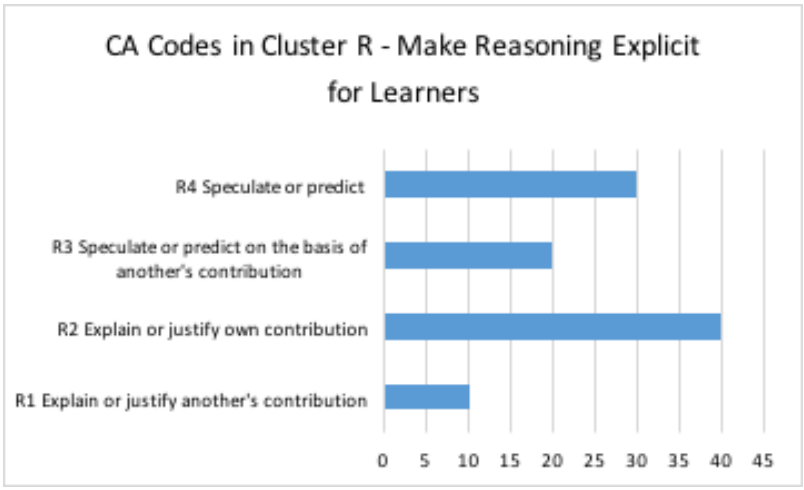


Figure 5. Breakdown of CA codes in cluster R – 'making reasoning explicit' interactions made by learners

The range of the interactions between learners, on the other hand, was greater. Still, however, there are two clusters that are particularly stronger than others. First, learners regularly asked their teacher and peers for help in order to take the next steps that were required. This explains the high invitation for elaboration (cluster code I) that is seen here. It is particularly encouraging that the activity, combined with teacher questioning and peer discussion, gave rise to interactions where learners verbally articulated their reasoning or formed hypotheses/predictions (cluster code R). Table CE2 provides a good example of this kind of interaction: Ovyaa verbally articulates a problem causing the entire list to be returned rather than a single random data item. The reasoning cluster and accounted for a near majority (45%) of all learner CA that were recorded.

Figure 5 breaks down the CA that were recorded for learners in cluster R. It is particularly pleasing to see that 70% of the recorded CA within this cluster (codes R3 and R4) focus on interactions where learners are verbally establishing theories or exploring possibilities regarding what will happen when a particular action is taken with the Poem Generator. In table CE1, for instance, Ethan correctly predicts the way to overcome a problem when attempting to make a pick from an empty word list.

Given these examples, there may be some evidence to suggest that the combination of programmability and the context of language learning, when scaffolded with teacher and peer talk, incites speculation and prediction. These findings are limited, however, by the relatively small number of communicative events that were recorded (twelve in total, not all are listed here) in addition to the small sample of learners that appear across the CE (six).

3.2 Examples of Computational Thinking Dialogue

From the overview, it can be seen that there is evidence of reasoning (particularly speculation and prediction) in the learner interactions. Following the overview, it is now possible to look at examples of specific CE in order to consider how these aspects of reasoning relate to what we may recognise as computational thinking using Brennan and Resnick's framework (2012) for computational thinking with Scratch. Specifically, the examples of dialogue are organised according to three *practices* that they identify: (i) testing and debugging; (ii) abstracting and modularizing and (iii) reusing and remixing.

3.2.1 Testing and Debugging

CE1 illustrates how the computational practice of *testing and debugging* is engaged with through the activity. It begins with Ethan's frustration that the poem generator stops each time it reaches a particular line of the poem. Guided by careful teacher-directed questioning, Ethan discovers in line four the reason for this error. He has typed an input into the random pick block that references a word list that does not exist. The CE ends with Ethan removing the block and testing that the change works. The questioning by the teacher in this conversation is particularly open at lines two and four: the problem solving skill demonstrated by Ethan here in locating and resolving this problem is particularly impressive.

Table CE1. Ethan debugging his poem

CE	Agent	Line	Programming club	CA		
				Code one	Code two	Code three
CE1 Debugging an issue that prevented the 'generate' block from functioning correctly	Ethan	1	Sir, why is this not working?	I4		
	Teacher	2	Have a look at your blocks – is anything stopping it?	G2	I4	
	Ethan	3	Oh.. Is it because of 'bad' (a custom word list)?	R4		
	Teacher	4	Why would that make a difference?	I4		
	Ethan	5	Its empty. No words in it.	R4	B2	
	Teacher	6	Try taking it out then.	G2		
	Ethan	7	It works now, thanks!	U		

CE2 is a conversation between Ovyaa and the teacher that provides a further example of a debugging process. At first in line one, Ovyaa wonders why the words generated have been displayed without spaces and all at once. By line five, however, Ovyaa makes the realisation that the words she is seeing is a complete list of all items within the word list. Ovyaa realises that she has not added a pick random block in order to choose one particular item from the list. As such, the entire word list has appeared in the poem generator on the stage. This example not only illustrates Ovyaa's debugging capability: she has also discovered an important programming concept that will prove useful later in the education cycle. This concept is the difference between lists and the data items that are contained within them.

Table CE2. Ovyaa debugging list handling in Snap!

CE	Agent	Line	Programming club	CA		
				Code one	Code two	Code three
CE2 Ovyaa uses debugging to correct her mistake and realises the computer's confusion with handling lists	Ovyaa	1	Sir, its not working. There's no spaces anywhere on here.	I4		
	Teacher	2	Which blocks have you used?	G6	I4	
	Ovyaa	3	Oh, there is no 'pick random' on there (*laughs*)!	R4		
	Teacher	4	So where have those words come from then?	I4		
	Ovyaa	5	It has returned all of the words rather than just one of them.	R2		

3.2.2 Abstracting and Modularizing

The creation of custom blocks featured as a key theme within the class discussions. CE3 is an account of conversation between Ethan and the teacher as Ethan creates his own custom block for generating a tanaga poem. Ethan initiates the conversation by seeking teacher approval of the custom tanaga block that he creates.

A key issue with the custom block Ethan made is that there is no method of incorporating bounded iteration to generate each poem a set number of times. In line 3, Ethan asks a very important question and explains (though not using the correct terminology) that a block variable is required in order to generate a series of poems. A block variable is a variable that exists locally to a block, effectively enabling the set of instructions inside the block to be looped a set number of times. The sequence ends with the teacher explaining how to do this and Ethan making his new block to solve the problem.

Table CE3. Ethan working with custom blocks

CE	Agent	Line	Programming club	CA		
				Code one	Code two	Code three
CE3 Ethan creates a custom block that generates a tanaga for a set number of times	Ethan	1	Sir! Look, (*pointing at the screen and a custom 'tanaga' block*) I've made a tanaga with only four blocks.	B2		
	Teacher	2	Great. Now can you also create a custom block containing the 'delete' and 'generate' blocks?	G2		
	Ethan	3	Ok. How do I make a block with a number space in for the number of poems?	R4	I4	
	Teacher	4	Good idea - you mean a block variable? You create a space like this *(demonstrates)* and then add this into the generate block, like this *(demonstrates)*.	G2		
	Teacher	5	Can you make your own and test if it works?	G2		
	Teacher	6	Yes, thanks!	U		

CE4 is a further conversation that takes place between Ethan and the teacher. In line 3, the teacher asks Ethan to explain the construction of the block he has created. Towards the end of the conversation, in line 5, Ethan makes a very important discovery about computation. Ethan realises that he has created a custom programming block that incorporates a different custom block he has also created. Ethan created a custom block called 'tanaga'. His second block, unusually named 'tanaga code', contains the tanaga block with the addition of a block variable to specify the number of times the generator will repeat. This block variable enables the user to define the number of the times a tanaga is generated. Through making his custom block within a block, Ethan has made a discovery about the modular and hierarchical nature of programming code in a computational practice that Brennan and Resnick call *abstracing and modularizing.* Further, by incorporating a block variable within this block, he has engaged with the computational concepts of *data* containers and *loops.*

Table CE4. Ethan working with custom blocks

CE	Agent	Line	Programming club	CA		
				Code one	Code two	Code three
CE4	Teacher	1	What block have you got there, Ethan?	I4		

During his presentation	Ethan	2	You type in the number of times you want a tanaga in the space in the block.	R2
at the end of	Teacher	3	What is inside that block?	I4
the challenge, Ethan	Ethan	4	There is another tanaga block that makes the four lines.	R2
explains the	Teacher	5	Show me.	G2
new block he has created.	Ethan	6	(*Demonstrating*) Here is a block within a block.	E2

3.2.3 Reusing and Remixing

James was a latecomer to the club and the conversation CE5 focuses on Rokas showing James how to use the Poem Generator. It was interesting to see here that Rokas appears to suggest *reusing and remixing* as a form of initiation. We can see in line four that James panics 'I broke it' as he struggles with the process of making a random pick from a word list. In line five, however, Rokas reassures James by advising him to 'copy' his code at first before expanding his own program later. Rokas appears to be suggesting a traditional pedagogical and 'cultural' approach that has been a way to learn to program.

Table CE5. Rokas introduces newcomer James to the club

CE	Agent	Line	Programming Club	CA		
				Code one	Code two	Code three
CE5 Rokas introduces newcomer James to the club and	Rokas	1	This is an haiku (gesturing towards James' screen). This is going to be one, two, three, four. Four syllables. You need seven in this line. You need to figure out a way of adding seven to this line.	P5	R2	
attempts to scaffold	James	2	Item random. How do I get item random?	I4		
James' learning	Rokas	3	You find it here (pointing towards the item block). You then get one of these boxes and put them in there.	E2		
	James	4	I broke it.	U		
	Rokas	5	James (reassuringly).. don't touch anything. Just try copying mine to start with.. don't worry James.	P3		

4. Discussion

From the analysis of teacher-learner and learner-learner dialogues some clear patterns emerge.

Teacher talk focused on situations where guidance or instruction was issued to learners; where the next steps required in order to make progress were suggested. This finding matches with expectations. Teacher dialogue also prompted learners to use reasoning to solve problems. During the activity, questions were characterised by key words and phrases such as 'how', 'why' and 'what would happen if?' in order to encourage reasoning.

A more surprising finding from the cluster analysis is the comparatively larger range of interactions that were witnessed when analysing the learner responses. The most frequent cluster code amongst learners, however,

was cluster R where reasoning was made explicit. Many of the CA within this cluster were derived from situations where learners explained or justified their thoughts through answering the questions posed by teachers and peers. Within the reasoning cluster, half of the CA recorded by learners were characterised by prediction, hypothesising or exploration.

The excerpts of dialogue provide insights into the ways in which reasoning manifested itself using the Poem Generator in relation to computational thinking concepts and practices. We have seen three computational thinking practices that were evident in the discussions taking place: *testing and debugging*, *abstracting and modularizing* and *reusing and remixing*. Further, computational thinking concepts like *data and containers* and *looping* can be seen in these excerpts.

More significant, however, is that the study revealed the importance of student-led classroom talk when learning programming.

Moving to a Dialogic Classroom

Monologic classroom talk is where the aims of classroom interactions are controlled by the teacher: 'communication geared towards achieving the teacher's goals' (Lyle, 2008, p. 225). Dialogic teaching, on the other hand, 'creates a space for multiple voices and discourses that challenge the asymmetrical power relations constructed by monologic practices' (ibid.). Alexander (2006) suggests that the movement towards dialogic talk in the classroom sees the learner taking more control over asking questions and commenting on ideas as he/she encounters opportunities for creating knowledge in the classroom.

The CE examples provided here show many examples of questioning initiated by the learner (*not* the teacher) as a result of the issues that they encountered when exploring with the Poem Generator. The answers that were provided by the teacher or peer to these questions were an integral part of the learning process. Dialogic talk, it could be suggested, was equally as important when learning to think computationally as the act of programming itself.

In CE3 and CE4, for instance, Ethan carefully constructs some complex questions to the teacher regarding how to incorporate looping into his block design. Teacher response led to the successful development of a new custom block that incorporated bounded iteration. CE2 provides a further example where careful teacher dialogue was necessary to enable Nimra to successfully debug her program. A key part of constructing new knowledge about computational thinking here was the learners taking control over discussing issues with others.

It is important to note that the 'other' is not *just* the teacher. There is already evidence to suggest that programming in pairs can help with the co-construction of knowledge (*see* Chong et al., 2005). Table CE5 corroborates this where Rokas attempts to initiate newcomer James to the club by suggesting *reusing and remixing* as a possible practice of entry. The pair programming approach, then, may be a useful tool for encouraging classroom dialogue when developing computational thinking in our classrooms.

Dialogic classrooms encourage the learner to pose questions and take ownership over steering the dialogue that takes place. The sources of these interactions are not only between peers: learner-teacher communication is equally as important. This study, though modest in scope, suggests that the dialogic approach could have a part to play in shaping future discussions about the pedagogy of programming.

Acknowledgments

Thanks to Torbjorn Dahl (University of Plymouth), Ceri Pugh (University of South Wales) and David Longman (independent academic) for their many helpful comments and insights. Further thanks to Gavin Jones, Victoria Corken, Dean Seabrook and Jemma Paull for giving up their time to support this research in what is always a very busy school day.

References

Alexander, R. (2006) *Towards Dialogic Teaching.* 3rd edn. New York: Dialogos.

Black, T. (2002) Understanding Social Science Research. 2nd ed. London: Sage.

Resnick, M. and Brennan, K. (2012) 'New Frameworks for Studying and Assessing the Development of Computational Thinking', *Proceedings of the 2012 Annual Meeting of the American Educational Research Association*, Vancouver, Canada, 13-17 April. Washington, DC: AERA [Online]. Available at: http://web.media.mit.edu/~kbrennan/files/ Brennan_Resnick_AERA2012_CT.pdf (Accessed 28 October 2013).

Chong, J., Plummer, R., Leifer, L., Klemmer, S., Eris, O. and Toye, G. (2005) 'Pair Programming: When and Why it Works', *Proceedings of the Seventeenth Annual Workshop of the Psychology of Programming Interest Group* (PPIG 2005), Sussex, Brighton, 29 June – 1 July. Sussex: University of Brighton [Online]. Available at: http://www.ppig.org/papers/17th-chong.pdf (Accessed: 22 September 2015).

Cockburn, A. and Williams, L. (2000) 'The Costs and Benefits of Pair Programming', *Proceedings of the First International Conference on eXtreme Programming and Flexible Processes in Software Engineering* (XP2000), Cagliari, Sardinia, Italy, 21 – 23 June. Calgary, CA: University of Calgary [Online]. Available at: http://collaboration.csc.ncsu.edu/laurie/ Papers/XPSardinia.PDF (Accessed: 22 September 2015).

Estyn. (2016) *Inspection under section 10 of the Schools Inspection Act 1996: School X.* Cardiff: Estyn.

Goldenberg, E. and Feurzeig, W. (1987) *Exploring Language with Logo.* Cambridge, MA: MIT Press.

Lyle, S. (2008) 'Dialogic Teaching: Discussing Theoretical Contexts and Reviewing Evidence from Classroom Practice', *Language and Education*, 22(3), pp. 222-240.

Mönig, J. and Harvey, B. (2009) 'Bringing "No Ceiling" to Scratch: Can One Language Serve Kids and Computer Scientists?', *Constructionism 2010,* Paris, France, 16-21 August. Paris, France: AUP [Online]. Available at: http://www.eecs.berkeley.edu/~bh/BYOB.pdf (Accessed 29 October 2013).

Sharples, M. (1985) *Cognition, Computers and Creative Writing*. West Sussex: Ellis Horwood.

Welsh Government (2011) *Raising School Standards* [Online]. Available at: http://gov.wales/docs/dcells/ publications/110629raisingschoolstandardsen.pdf (Accessed: 8 August 2016).

Welsh Government (2016) *Draft Digital Competence* Framework [Online]. http://learning.gov.wales/docs/learningwales/publications/160611-draft-digital-competence-framework.xlsx (Accessed: 8 August 2016).

Wing, J. (2006) 'Computational Thinking', *Communications of the ACM*, 49(3), pp. 33–35.

Copyrights