Pedagogy in teaching Computer Science in schools: A Literature Review

Jane Waite

Queen Mary University of London and King's College London

Table of Contents

Table of Contents 2

List of Tables 3

List of Figures 3

1 Introduction 4

2 Research Methodology 5

3 Findings 8

4 Discussion and Recommendations 41

5 Summary. 52

Bibliography 53

Appendix A 75

Appendix B 85

Appendix C 87

Appendix D 88

Glossary 89

List of Tables

Table 1 Theme specific search terms 5
Table 2 Study attributes 6
Table 3 Teaching phases 7

List of Figures

Figure 1 Pedagogy literature review categories 8
Figure 2 The Block Model as explained by Clear 2012 (Clear, 2012) 11
Figure 3 Use-Modify-Create framework (Lee et al., 2011) 12
Figure 4 New combined taxonomy (Meerbaum-Salant et al., 2013) 14
Figure 5 UDL Hypothetical learning progression (Dwyer et al., 2014) 15
Figure 6 Interpretation of path diagram (Lopez et al., 2008) 19
Figure 7 The PTD framework (Bers, 2010, p. 5) 25
Figure 8 Primary and Secondary Programming Languages from CAS Survey 2015 (Sentance, 2015) 33

1 Introduction

As commissioned by the Royal Society, this work provides a supplementary addendum to the Royal Society Computing Education Project Report (Crick, 2017) by providing a review of current literature of pedagogy in teaching computer science in schools.

This literature review summarises, **what is known about pedagogies for teaching computing in schools**. The overall intention is to identify **what recent clear evidence is emerging from research** and identify **potential gaps where useful pedagogy research could be carried out to support teaching computing in the UK**.

The ICT curricula across the UK have been, or are in the process of being, changed to incorporate computer science, including programming and computational thinking. These changes require teachers to understand not only the content but also consider how it should be delivered. How much is understood about how programming and the other elements of the computing curriculum should be taught is not clear. This report adds to this body of knowledge by collating evidence from existing research.

2 Research Methodology

We adopt the approach of Gough, Sandy, & James (2013, p. 26) who suggest that a combined review of reviews with an additional synthesis of new studies can provide an overview of a study area.

The following approach was adopted:

1. Identification of recent literature reviews;
2. Selection of those reviews with the best coverage and methodology and with most clarity;
3. Identification of new empirical studies undertaken after the literature reviews were written;
4. Summarise the literature reviews and new studies;
5. Make recommendations for future research.

Literature reviews and studies were identified using searches of research databases alongside tracing of citations of papers. The research databases searched were: ACM Digital Library, IEEE, Taylor and Francis, Wiley Online Library and Eric. A search for the term "pedagogy" and "school" was used to collate an initial set of papers. For emerging themes, additional specific searches were undertaken using search terms as shown in Table 1. These extra searches were included as the generic searches retrieved limited coverage of the themes under consideration.

Theme	Generic search terms	Specific search terms
Contexts, Physical computing	School	Physical computing
Student Engagement, Pair programming	School	Pair programming
Programming Languages, Block to text-based programming transition	School	Transition, block

Table 1 Theme specific search terms

The search results were de-duplicated, and the abstracts of the remaining studies were read to decide whether to retain or discard the paper using the following selection criteria:

- Papers were included for further consideration if they related to school age students (under 18 years of age) unless there was no school age research for a field of interest, or the work was a significant contribution, based on our knowledge of the field;
- Papers were included for further consideration if they were literature reviews, qualitative or quantitative studies. Opinion pieces were removed as the focus of this report is to gather evidence-based research. However, several significant commonly cited opinion pieces which provide context have been retained to situate studies;

- Papers published since 2007 were included for further consideration; studies older than 10 years were removed as the focus of this report is to gather recent research. However, older work has been included where there was a significant contribution and it situated a theme;
- Papers with a strong evidence base were included for further consideration. Studies with weak empirical evidence were removed, as the focus of this report is to review research with clear evidence on which to base recommendations. However, where there was a lack of studies for a theme, some less rigorous reports were retained to situate discussion;
- Material already highlighted in the main Royal Society Report (Crick, 2017) was excluded, such as discussion of computational thinking.

To aid review of the selected studies, the attributes shown in Table 2 were summarised for each paper.

Attribute	Description	Notes
Theme	The theme for the study as categorised for this report	
School Phase	The phase of school learning	Primary, Secondary HE (Higher Education), KS1, KS2, KS3, KS4, KS5. See Table 3 Teaching phases
Country of origin	The country of most authors	Where there was no majority, the first author's country was selected
Research approach	The approach taken for the research	Literature Review, Qualitative, Quantitative, Mixed
Participation scale	The number of participants of the study	Small: 1-50 participants, Medium: 51-151 participants, Large: more than 151 participants
Study Context	The context of the study	Specific details, such as setting

Table 2 Study attributes

This initial search resulted in over 700 papers. The numbers of studies retrieved by data source are shown in Appendix D. Using the selection criteria explained above the number of papers was reduced to 86, the list of selected studies is shown in Appendix A. The papers are grouped by theme and shown in the same order that they are referenced in the main body of this report. The remainder of this study is organised as follows. For each theme, an overview of the studies is presented. Emerging evidence is then summarised, and potential gaps where useful pedagogy research could be carried out to support teaching computing in schools in the UK is outlined.

	England and Wales key stage	England and Wales	Scottish Grades	Northern Ireland Grades	USA Pupil Grades	Student age (years old)
	Early Years	Early Years	Primary 1	Nursery		4-5
Primary	Key Stage 1	Year 1	Primary 2	P1	Kindergarten (K)	5-6
		Year 2	Primary 3	P2	Grade 1 (K1)	6-7
	Key Stage 2	Years 3 to 6	Primary 4 to 7	P3-P7	Grades 2 to 5 (K2-K5)	7- 11
Secondary	Key Stage 3	Years 7 to 9	S1 to S3	Year 8 to 10	Grades 6 to 8 (K6-K8) (middle school)	12-14
	Key Stage 4	Years 10 to 11	S4 to S5	Year 12 to 12	Grades 9 to 10 (K9-K10)	15-16
	Key Stage 5	Years 12 to 13	S6	Year 13 to 14	Grades 11 to 12 (K11-K12)	17-18

Table 3 Teaching phases

3 Findings

The literature has been grouped into the themes shown in Figure 1 to support the process of review. These broad groupings became evident as the literature was synthesised. The material was aligned to these emergent themes using our knowledge of the field. The groupings exemplify some of the choices that teachers make as they decide how to meet a computer science learning objective. They draw upon significant models, select specific instructional techniques, choose a context to situate the learning, select a programming language and select a method of student engagement to maximise learning for their cohort of students.

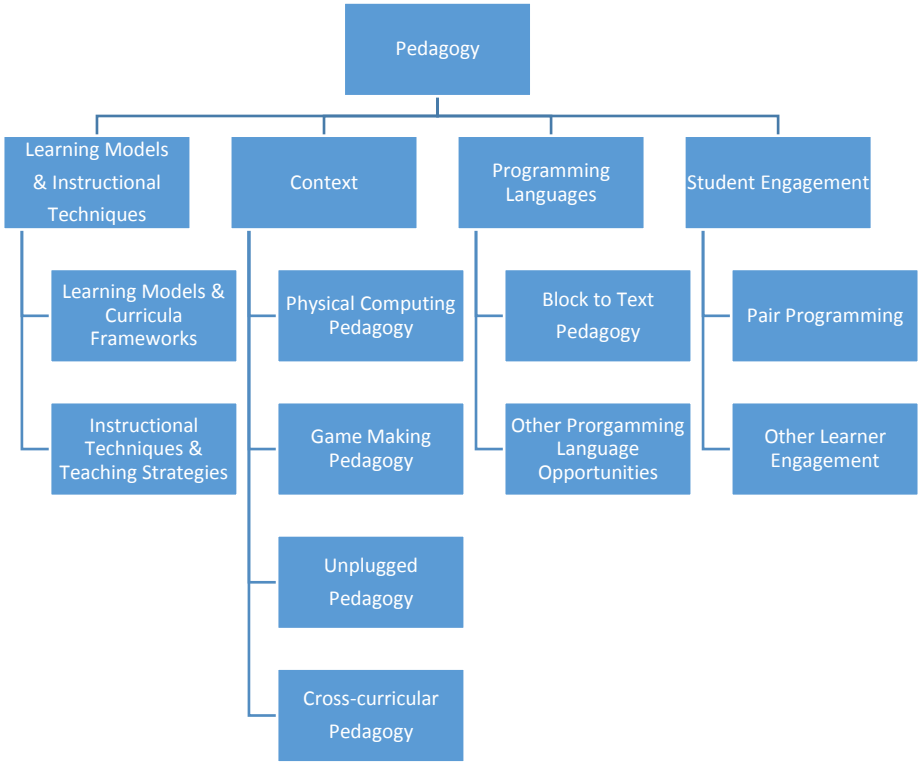

Figure 1 Pedagogy literature review categories

3.1 Learning Models and Instructional Techniques

Four literature reviews are first summarised, followed by a review of studies not covered by these papers.

In a literature review of teaching and learning of computational thinking through programming in K-12, Lye & Koh (2014) concluded that further research should be situated in class settings as less than half of the studies synthesised were in primary or secondary classrooms. The authors recommended that learners should be taught to verbalise their understanding using think-aloud techniques and that qualitative data should be framed within established programming study categories. Further, they

proposed instruction should use a constructionism-based authentic problem-solving approach with information processing, scaffolding and reflection activities. The authors emphasised that students ought to be 'thinking doing and not just doing'.

Falkner & Vivian (2015), in their systematic review of K-12 resources for the Australian Curriculum, identified that pedagogical support was generally missing from the resources reviewed, noting the focus was on content knowledge. The authors stated there was a need to develop teachers' resources that demonstrated and modelled learning progression and implementation plans. The authors reported further resources were required to support data and functional requirements analysis; algorithm design and evaluation; programming as an element of this process; evaluation and critical analysis and readymade unplugged resources for younger primary learners.

In a 2015 systematic literature review of computing education in K-12 schools, Garneli, Giannakos, & Chorianopoulos (2015) concluded that despite challenges, computing could be an effective and enjoyable learning experience and that there was no one pedagogical solution for all classes. The authors cited a range of instructional approaches used in the reviewed studies, including using visual programming languages to introduce young learners to programming and the use of authentic text-based languages. Game-making and physical computing were mentioned to motivate learners and using physical computing to explore concepts. Further, they suggested making creative products with much scaffolding to start with, combining kinaesthetic activities with programming activities and the careful selection of educational contexts. Studying, modifying and extending code samples, as well as using demonstrations and tutorials and debugging tools was recommended. The Use-Modify-Create strategy was mentioned (Lee et al., 2011) as well as problem and project-based learning and an 'in-time' approach to present new learning as and when needed.

In a K-8 study to provoke discussion rather than to raise immediate research agendas, Rich, Strickland, & Franklin's (2017) literature review catalogued computer science learning goals that experts had theorised as important to teach, and compared this to learning goals that had been explored and researched to identify discrepancies between the two. The most commonly cited unmatched goal was designing solutions. This goal related to the high-level planning of solutions in applied scenarios, usually before coding, requiring the abstraction away of details, clearly stating problems, reformulating problems so they can be solved computationally and avoiding implementation constraints. The authors raised the question of how focused on coding, rather than applied problem-solving computer science might be.

Rich et al. (2017) also noted that there was ample literature related to teaching learners about the mechanics of loops and functions. However, the authors could not find any research on K-8 learners concerning how to decide whether to create loops or functions, asking when to introduce considerations of clean, efficient, readable code and whether this needs to be explicitly taught. Further, they raised the issue of extremely fine-grained programming (EFGP) and the impact on debugging and testing, asking how thorough we might expect young learners to be.

3.1.1 Learning Models & Curricula Frameworks

Theoretical models for teaching and learning are now outlined that situate pedagogical choices.

Du Boulay (1986) highlights the concept of a programmers' mental model of a 'notational machine': the behaviour of static code as it becomes a running process. Opinion is divided on the significance of this model to novice programmers, and how it might come to be appreciated or taught to learners. There are calls for further research into this area (Ben-Ari, 1998; Berry & Kölling, 2013; Greening, 2000; Sorva, 2013).

Schulte (2008) and Schulte, Clear, Taherkhani, Busjahn, & Paterson (2010) suggest a holistic model of learner understanding of programming, the Block Model. A diagrammatic representation of the Block Model is presented in Figure 2. Using the Block Model, Schulte et al. (2010) suggest teaching and learning sequences: micro sequences that focus on one example, such as a single activity to implement an algorithm; and macro sequences that focus on a course of many activities. They concluded with a summary of three critical aspects of teaching and learning to program 1) domain knowledge is underestimated in pedagogy, 2) experts having a flexible understanding of programs based on more than just reading the program text is rarely discussed, 3) there are lots of possible learning tasks for reading and comprehending programs, such as tracing examples of code or explaining the purpose of a piece of code in plain English. An analogy cited by the authors, is that the process of learning to program is like sewing a patchwork quilt, with each cell in the model being one of the squares, and each knowledge layer like the stuffing. As knowledge is acquired the quilt becomes more robust and coherent, with novice programmers having a 'holey knowledge' (Schulte et al., 2010) or a 'holey quilt' (Clear, 2012).

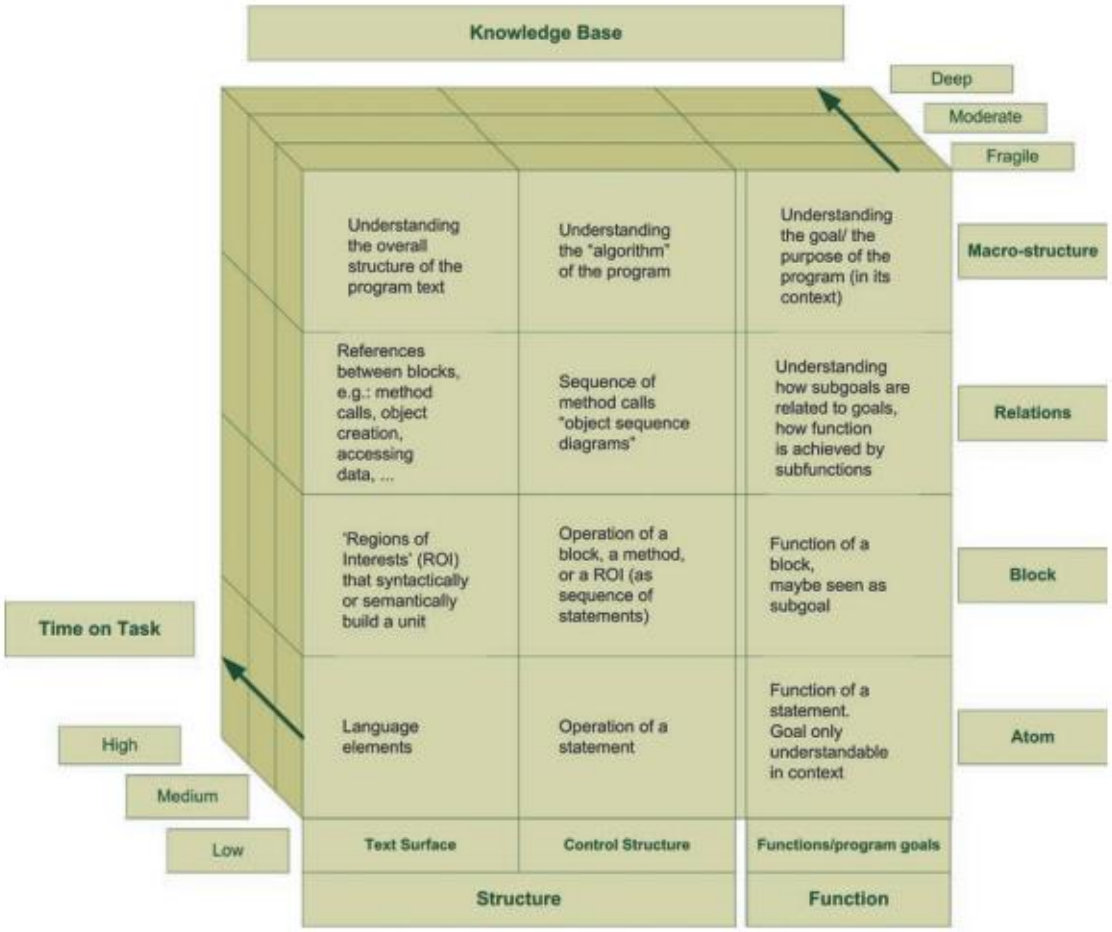

Figure 2 The Block Model as explained by Clear 2012 (Clear, 2012)

The Block Model's distinction between a novice programmer's understanding of the structural atomic detail of a program, the code, the functional goals of the program, and the problem (Schulte et al., 2010) resonate with the development of the levels of abstraction model (Armoni, 2013; Cutts, Esper, Fecho, Foster, & Simon, 2012; Hazzan, 2003; Perrenet & Kaasenbrood, 2006; Statter & Armoni, 2016; Taub, Armoni, & Ben-Ari, 2014; Waite, Curzon, Marsh, & Sentance, 2016). Here initial work focused on university students' understanding of algorithms and data structures in terms of levels of abstraction (Aharoni, 2000; Cutts et al., 2012; Hazzan, 2003), but more recently attention has turned to younger learners (Armoni, 2013; Statter & Armoni, 2016; Waite et al., 2016).

Statter & Armoni (2016) reported on a study of 119 grade 7 pupils taught the levels of abstraction model; namely the execution level, program level, algorithm and problem level. The authors reported that the experimental group attended more to the algorithm level, using more written and verbal descriptions

than the control group. There are synergies here with the call from Rich et al. (2017) for research related to design, as the algorithm level of abstraction maps to this stage of problem-solving (Waite et al., 2016).

In a separate study also looking at the understanding of abstractions when programming, Cutts et al. (2012) created an Abstraction Transition (AT) taxonomy. They described the taxonomy as having three main levels of: code; CS speak, and English and claim using it will develop students' programming. An example transition goal given by the authors was 'Given a technical description (CS Speak) of how to achieve a goal, choose code that will accomplish that goal'.

Grover & Pea (2013a) investigated a discourse-intensive pedagogy and highlighted the value of the social aspect of learning and how the deliberate introduction of talk into a curriculum can shape the 'process of development' (Grover & Pea, 2013a, p. 726) concluding with a call for further research on 'computational discourse'.

Returning to the Block Model, Schulte et al. (2010) proposed the problem of 'holey' knowledge and suggested time on task as the dimension to practise and develop depth of understanding. Lee et al. (2011) suggested a framework to afford developing depth of understanding, related to time on task, but with a finer progression of learning of 'Use- Modify-Create' as shown in Figure 3.

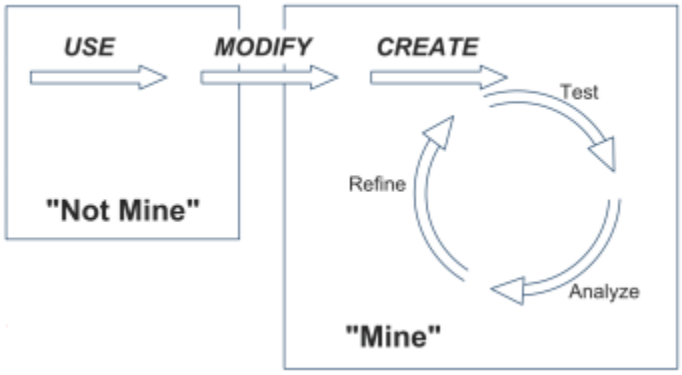

Figure 3 Use-Modify-Create framework (Lee et al., 2011)

Lister (2011) reported on cumulative work investigating learning of programming in higher education and proposed a model for learning that informs pedagogy. Based on Neo-Piagetian theory, the model maps stages of programming development to pre-operational, operational and formal operational reasoning. This theory has been used as the lens for some of the university student studies by Teague, Lister and Corney. They acknowledge limitations of generalisation of their findings, as studies are either of a small population or qualitative. However, they suggested that there is evidence that novices need more support to secure basic programming constructs, requiring educators to assess learners' stage of development

and tailor teaching to their needs. They suggest that further quantitative research is needed (Corney, Teague, Ahadi, & Lister, 2012; Teague & Lister, 2014a, 2014b).

Lister summarised his Neo-Piagetian theory and concluded that the gap between academics and teachers needed to be bridged else computing curricula would be developed that would not work in class. He called for practical day to day material to be created that was informed by research and incorporated learners' stages of development (Lister, 2016). This leads to consideration of pedagogical themes inherent in recent studies of teaching and learning material, developed by academic teams for practical use in classrooms.

Meerbaum-Salant, Armoni, & Ben-Ari (2013) reported on the evaluation of teaching materials they had developed and used to teach programming in Israeli middle schools using Scratch. The report concluded that Scratch was a suitable platform for teaching programming as most students reached a reasonable level of computer science understanding. However, the authors reported difficulties in teaching certain topics such as repeated execution, variables and concurrency. The authors urged caution, advising that close and effective mentoring was needed by teachers for effective learning to be achieved, as left to their own devices learners created media and learnt very little.

The same research team combined Bloom's taxonomy with the Solo taxonomy to create a new hierarchical taxonomy to support their curriculum development. The authors claimed that higher levels of the taxonomy imply deeper comprehension than the superficial, lower levels. The report provided the dimensions for their new taxonomy and three examples as depicted in Figure 4. The authors asserted that learners progress from 'unistructural understand' for easiest student performance to 'relational create' the highest level of mastery (Meerbaum-Salant et al., 2013).

Rather than teaching Scratch features, the pedagogy for delivery of teaching sequences was for concepts to be taught. These 'concepts based teaching sequences' were taught in a carefully considered order. For example, concurrency was taught early and variables late. Programming constructs were introduced as needed, with problems carefully selected so that constructs were not needed earlier than they had been introduced (Meerbaum-Salant et al., 2013).

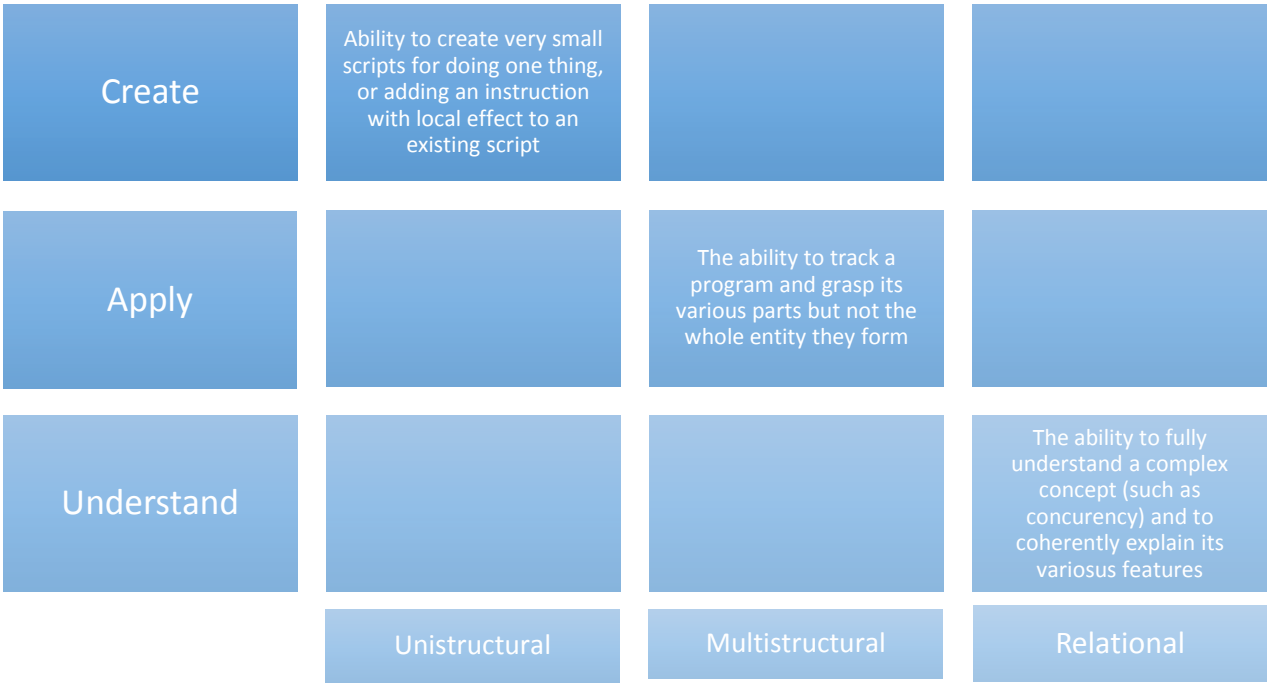

Figure 4 New combined taxonomy (Meerbaum-Salant et al., 2013)

Hansen, Hansen, Dwyer, Harlow, & Franklin (2016) reported on a curriculum they had developed for slightly younger learners, grades 4 to 6, again using a block-based programming language, but this time using LaPlaya, a specially developed 'Scratch like' environment. Franklin & Harlow's interdisciplinary team at the University of Chicago and UC Santa Barbara spent some five years working with local schools to design, implement and evaluate their new programming environment (LaPlaya) and associated curriculum. In support of this, the group developed the Universal Design of Learning (UDL) framework which underpinned the curriculum development and informed teachers' ongoing use of the curriculum provided. Informed by research on differentiated learning, the UDL framework was created to support the needs of all learners, meeting their cognitive, language and mathematical needs, incorporating gender neutral and appropriate ethnic and linguistic curricula content.

The UDL team have written over a dozen academic papers related to the curriculum development. Here five studies with significant pedagogical findings are mentioned. Dwyer, Hill, Carpenter, Harlow, & Franklin (2014) explored progression in algorithmic thinking and programming using unplugged activities and suggested a strand of their hypothetical learning progression as shown in Figure 5.

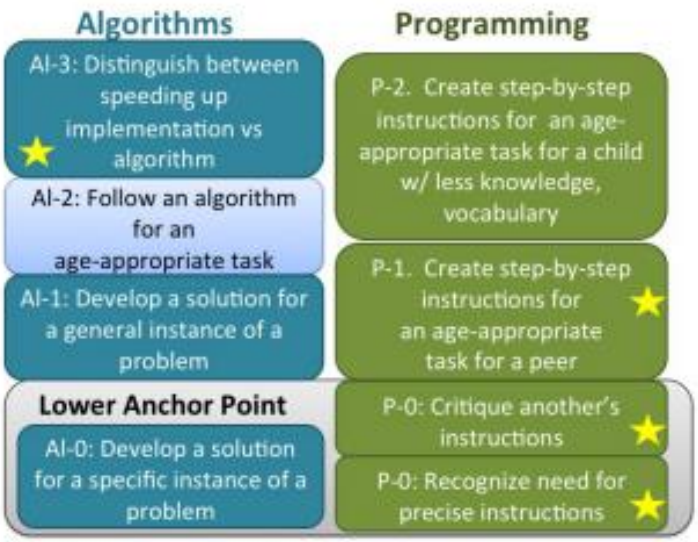

Figure 5 UDL Hypothetical learning progression (Dwyer et al., 2014)

Hansen, Iveland, et al. (2016) reported that user-centred design in block-based programming languages was more complex than originally thought and that it required explicit instruction. In a further study Hansen, Hansen, et al. (2016) suggested creating a sandbox, for learners to apply new skills, as some learners completed activities early and spent time changing the appearance of their work. There are synergies here with comments made by the Israeli curriculum team on focused learning activities (Meerbaum-Salant et al., 2013) and the Use-Modify-Create model (Lee et al., 2011).

Franklin et al. (2016) suggested that it is important to design gradations of task from simple to complex. They also argued and there is a need to analyse and understand all aspects of introductory computer science instruction and not to take atomic units such as initialisation for granted. There are links here with the 'holey knowledge' and the Block Model (Clear, 2012; Schulte, 2008). Franklin et al. (2017) reported that activities with precise mathematics caused undesirable difficulty and were a barrier for some students to learn the underlying computer science.

Further work is needed to understand the UDL framework and its commonalities with other models, curriculum frameworks, instructional techniques and strategies.

Grover, Pea, & Cooper (2015) created and tested a blended computer science course for middle school students, called 'Foundations of Advancing Computational Thinking' (FACT). Their 2015 study stated the material had been developed for 'deeper learning', focusing on pedagogical strategies to support and assess the transfer from block to text-based programming, including materials to remedy misconceptions

and provide systems of assessment. The authors reported that students using FACT achieved substantial gains in algorithmic thinking; could transfer their learning from Scratch to text-based languages and achieved significant growth in a more mature understanding of computing. Maths ability and prior computing experience were found to be strong predictors of learning outcomes. This association of maths ability and computing aptitude supports Franklin et al.'s comments on mathematics being a barrier for some students (Franklin et al., 2017) and merits further investigation.

The FACT pedagogy is clearly documented by the study, including a rationale for the choices made. The authors explain that no single pedagogical approach is employed. Instead, a blended approach is advocated. This approach incorporates inquiry-based learning, scaffolding (Pea, 2004), cognitive apprenticeship (Collins & others, 1987), code reading and tracing, think aloud, use of computing language, directed and open-ended projects, independent and pair work and a carefully controlled introduction of concepts in a pre-determined order. There are similarities here with the carefully ordered curriculum of Meerbaum-Salant et al. (2013) and opportunity to compare the progression advised by each curriculum.

Grover et al. (2015) refer to the contention raised by Mayer (2004) that: a minimally guided discovery approach often makes for higher learner engagement and agency but misses out on helping students develop mental models of concepts. Referring to Papert & Harel (1991) , they balance this with the idea that an 'instructionist' approach does not engage prior learning. The report cites a central theme for the curriculum of incorporating 'preparation for future learning' PFL, including assessment of this.

Probably the most widely cited academic generated framework for block-based programming curriculum is that of Brenan & Resnick's framework for studying and assessing the development of computational thinking (Brennan & Resnick, 2012). The concepts, practises and perspectives of this framework are often mentioned by research studies when considering progression for both curricula development and assessment.

Similarly, Seiter & Foreman (2013) propose a model for understanding and assessing progression in computational thinking, the Progression of Early Computational Thinking (PECT) model. To pilot test the model the authors analysed 150 Scratch projects, concluding that as learners get older their computational thinking skills increase. Further research is currently being conducted to validate the PECT model, to define a prototype progression in computational thinking.

Picking up this reoccurring theme of computational thinking, the National Curriculum in England requires teachers to 'equip pupils to use computational thinking and creativity to understand and change the

world' (DfE, 2013a, p. 1, 2013b, p. 1). Despite a lack of consensus on exactly what computational thinking is (Barr & Stephenson, 2011; CSTA, 2011a; Grover & Pea, 2013b; Lye & Koh, 2014; Selby & Woollard, 2014) and its merit (Tedre & Denning, 2016), proponents (Wing, 2011) advocate its importance and emerging guidance on teaching computing incorporates computational thinking in a variety of forms in computing materials and curricula (Benton, Hoyles, & Noss, 2016; Berry, 2015a; Berry et al., 2015; Bers, Flannery, Kazakoff, & Sullivan, 2014; Brennan & Resnick, 2012; Google, 2016; Grover et al., 2015; Gujberova & Kalas, 2013; Hansen, Hansen, et al., 2016; Kafai & Burke, 2015; Lee et al., 2011; Repenning et al., 2015; Rodriguez, Kennicutt, Rader, & Camp, 2017; Seiter & Foreman, 2013; Weintrop, Holbert, Horn, & Wilensky, 2016). The remit of this literature review is not to reflect upon the veracity of computational thinking, rather highlight significant pedagogical themes. Clearly, computational thinking is one such theme and as such any ongoing research on computing pedagogy requires review of what computational thinking is, how it can be developed, how it impacts on teaching and learning and its role within the pedagogy advocated.

3.1.2 Instructional Techniques & Teaching Strategies

The work of Papert runs as a thread throughout the curriculum frameworks devised by the various research communities. Reference is made to learners constructing knowledge as they explore and develop a personal understanding of newly introduced concepts or devices (Papert, 1980). However, balanced with this, is a call for guided instruction to ensure that learners circumnavigate a carefully constructed progression to develop a complete mental model (Garneli, Giannakos, & Chorianopoulos, 2015; Grover et al., 2015; Lye & Koh, 2014; Meerbaum-Salant et al., 2013; Schulte, 2008). Grover et al. (2015) suggest that to foster deep learning a combination of guided discovery and instruction rather than pure discovery and 'tinkering' would be more successful. Attention is now turned from generic models and frameworks to strategies and instructional techniques which develop depth of understanding, teach difficult concepts and address specific misconceptions.

A 6th to 8th-grade (n=100) study assessing understanding after an introductory programming course in Scratch revealed that learners were unfamiliar with the use of variables, and had trouble with loops and Boolean Logic. The authors suggested that constructionist activities should be combined with targeted conceptual learning for foundational constructs (Grover & Basu, 2017). This sentiment is echoed by a number of studies with emerging evidence that some of the more difficult concepts such as initialisation, variables and loops need to be explicitly taught (Hubwieser, Armoni, Giannakos, & Mittermeir, 2014; Kirschner, Sweller, & Clark, 2006; Statter & Armoni, 2016; Sweller, Kirschner, & Clark, 2007). Other studies

raise the need for learners' cognitive load to be managed by more closely controlling learning opportunities and learning experiences (Alexandron, Armoni, Gordon, & Harel, 2014; Paas, Renkl, & Sweller, 2003; Tsai, Yang, & Chang, 2015; Van Merrienboer & Sweller, 2005). As a body of studies there are implications here for pedagogy in school, with suggestion that targeted teaching is needed for difficult concepts within a controlled progression of learning experiences.

Meerbaum-Salant, Armoni, & Ben-Ari (2011) asserted that Scratch promotes certain 'habits of mind' such as a bottom-up approach to development where using trial and error novice programmers build up programs which are 'extremely fine grained'. The authors reported such programs are not well-structured, hard to understand and debug, leading to the question of whether this might cause problems as learners' advance in programming. They concluded that while Scratch is motivating and easy to use, the question of whether learners should start with 'the right way' or learn this later requires further qualitative and quantitative research.

A recent analysis of 250,000 Scratch projects, found most programs were small and included dead code. The authors reported code duplication was common and procedures, an essential component of well-structured programs, were rare (Aivaloglou & Hermans, 2016). This theme of quality of code and what learners perceive to be correct was focused upon in a large-scale study of how grade 10 to 12 learners evaluated the correctness of programs. Kolikant & Mussai (2008) concluded that learners' notions of partial or relative correctness of programs included 'a grain of correctness'. So, if a program met some of the requirements, it was deemed as correct. Only if there were no 'grains of correctness' was a program considered incorrect. The authors commented that the older participants were less tolerant of logic errors. However, they advised that younger students should be educated that even a small error means that the program is incorrect. Further research on the idea that non-working programs are considered as incorrect was recommended as well as an exploration of teachers' beliefs about what makes a 'correct program'. This leads to the notion of code comprehension, reading and tracing code.

In an Australian university study, Lopez, Whalley, Robbins, & Lister (2008) compared code tracing performance to results from code writing tasks, comparing students' ability to explain programs in plain English to code writing outcomes. The authors suggested a path of related tasks and understanding, a path diagram, to support programming development. Figure 6, depicts an interpretation of the path. There are synergies here with the Block Model (Schulte, 2008).

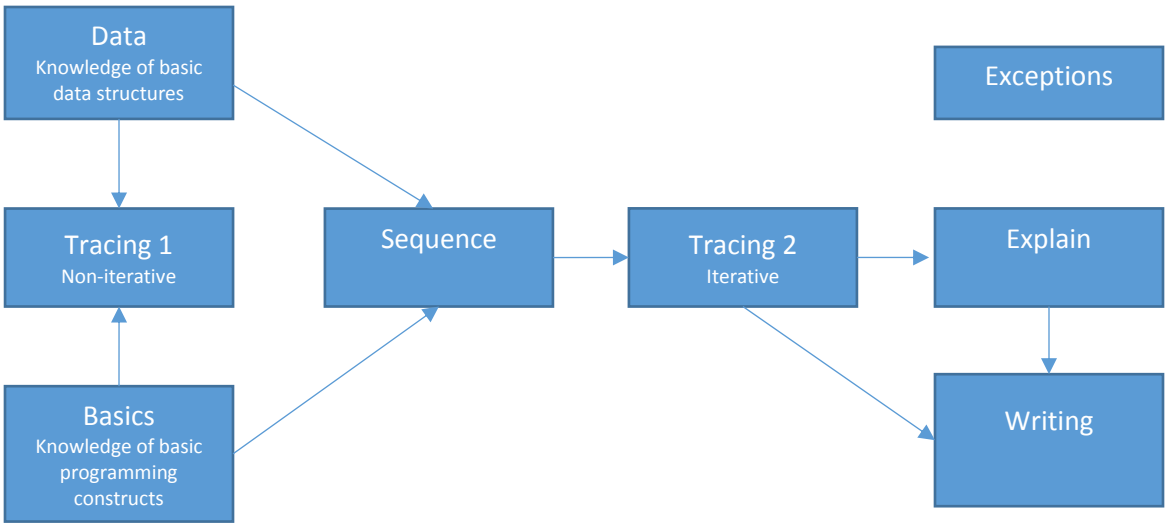

Figure 6 Interpretation of path diagram (Lopez et al., 2008)

Teague and Lister have created a body of work related to code comprehension in novice undergraduate programmers linking progression to their work on a Neo-Piagetian model of abstraction. In studies comparing tracing skills to code writing, reports have concluded a direct correlation (Lister, Fidge, & Teague, 2009; Lopez et al., 2008; Venables, Tan, & Lister, 2009). Lister described novices requiring 50% tracing code accuracy before they could independently write code with confidence (Lister, 2011).

Teague & Lister (2014c) reported that learning to program is sequential and cumulative, with tracing requiring students to draw on accumulated knowledge to conceive a big picture. The authors suggested novice learners should be focused on very small tasks with single elements, with emphasis on reading and tracing code before they are expected to write code snippets; scaffolded sequences of carefully chosen tasks should then be used to facilitate further progress. They concluded that the challenge lay with identifying at what stage students were, and then giving them time to master skills within their stage of development, calling for other academics to investigate this area. In a longitudinal study of a single student, Teague & Lister (2014a) evidenced these stages of development using a case study of a single student. Similar conclusions were made from a study of variables and assignment statements, the authors stating that teaching approaches needed to change to better identify learners' understanding, calling for further quantitative work (Teague & Lister, 2014b). This theme of assessing learners' current understanding, and tailoring work to build upon it resonates with the UDL philosophy of differentiation and supporting the needs of all learners (Hansen, Hansen, et al., 2016).

Studies related to code comprehension have highlighted the importance of reading code to address misconceptions of algorithm efficiency (Gal-Ezer & Zur, 2004) and the use of worked examples to understand how variables change over time (Sudol-DeLyser, Stehlik, & Carver, 2012).

Busjahn & Schulte (2013) interviewed high school teachers on code reading and concluded that further research was required to investigate improved reading strategies. In a higher education study, they investigated eye movements and the differences between how novice and expert programmers read source code. They reported that experts read code LESS linearly than novices. The authors suggested that cuing visual attention to locations that experts might attend to could be an avenue for further research, concluding that instructors and students could monitor their own progress using eye tracking tools (Busjahn et al., 2015).

In work related to younger students and how they look at code, Dwyer et al. (2015) reported several significant findings. Firstly, reading code in such environments is complex and secondly the visual nature of block-based environments impacts on reading in both intended and unintended ways. They reported that as well as using the code itself, learners used information from the 'stage' where the program was running or had run, as well as information about blocks from the code editing areas. In some cases, this information was helpful, in others not. The report suggested that younger learners may need explicit instruction on how different features work independently and together. The authors concluded further work was needed to analyse the reading of a variety of projects as well as to look at what visual cues were used by different groups of students.

Gujberova & Kalas (2013), working with primary students using a route-based programming environment, recommended a sequence of carefully graded learning activities to improve programming and computational thinking. Within these gradations was a stage where learners read and interpreted each line of code, as well as a stage for reading the entire program and predicting the outcome. The authors were cautious to interpret the results of their study, due to the test questions' similarity to intervention activities, concluding that further work is needed to understand the difficulty and similarities of programming tasks.

A further idea related to looking carefully at code is that of subgoal modelling, where meaningful labels are added to worked examples to visually group steps into subgoals thereby highlighting the structure of code. Two higher education studies, Margulieux & Catrambone (2016) and Morrison, Margulieux, Ericson, & Guzdial (2016), used this strategy with exemplar text, worked examples and problems. Both reports

concluded that those students given subgoals performed significantly better than those who had no subgoals or who added their own subgoals. The authors linked their work with the concept of cognitive load, and suggested the labels reduced the extraneous load related to the detail of the example, and learning was improved as the intrinsic load was also reduced by providing a way to organise the problem in memory.

In a KS3 study, learners were asked to add notes explaining their code by annotating it within the programming environment. The authors reported those adding annotations performed significantly better than those who did not. However, the intervention group not only used the annotation tool but also problem-based learning. Therefore, it is difficult to conclude that the annotation strategy resulted in the performance gains rather than the other changes (Su, Yang, Hwang, Huang, & Tern, 2014).

In a different line of research, a comparison was made between university students reviewing static code versus their instructor modelling coding 'live'. Rubin (2013) concluded that for the end of term projects, students' grades were significantly higher if they were part of the live coding version of the course. Furthermore, in midterm assessments, the live coding cohort performed as well as the static code group (Rubin, 2013).

The strategy of a teacher modelling the creation of a piece of work is a common instructional technique used in non-computing primary settings, as the teacher thinks aloud to explain the choices in writing a story or choosing a method to use to solve a mathematics problem. This form of apprenticeship is mentioned by Grover et al. (2015) in their description of the FACT framework as teachers think aloud as they model creating solutions including the writing of pseudo-code.

Cutts, Connor, Michaelson, & Donaldson (2014) reported on the effectiveness of pseudo-code as an instrument in formal computer science examinations and recommended that it be replaced with a reference language. They refer to the Block Model (Schulte, 2008) and the distinction between understanding the functional domain that a program is situated in, and the structural features of the program (see Figure 2). They revealed how natural language is overlaid to facilitate understanding and suggested it may be this overlay that confuses novice programmers.

As well as the confusion caused by using natural language to describe problems and programming solutions, novice programmers may have misconceptions about the detail of how programming constructs and commands work. A Finish group, at the University of Turku, (Veerasamy, D'Souza, & Laakso, 2016) revealed a range of misconceptions that impact novice programmers' comprehension. In a

study of university students (n=39) completing an end of course Python programming e-exam, they reported students misunderstood the meaning of inbuilt functions and their application and students were confused about the use of return statements and the data type of parameter passing. Further, students misunderstood the process of flow of control statements, particularly nested if and the for-loop process, index positions and referencing list elements. Critically, students who had misconceptions made knowledge errors and failed to complete the task. They recommended that further research should be conducted to measure the correlation between types of errors and misconceptions, particularly considering if there are any gender effects. However, they urged caution about generalising their results due to small sample size, and limitations in not investigating other possible influencing factors. Finally, in apparent opposition to the earlier theme raised that being able to trace a concept should mean improved ability to write code using the concept, the authors found that 83% of the students who failed to trace a loop **could** write a program using a loop, suggesting this was because students had remembered this code from an earlier experience.

The same Finnish team undertook another undergraduate study comparing experts to novices as they solved a simple Java programming problem (Lokkila et al., 2016). They concluded that experts seemed to abstract the task more than students, suggesting this was most likely due to them being able to draw upon existing templates and /or plans on how to solve a certain type of problem. The authors recommended that students would benefit from instruction and strategies on how to abstract tasks highlighting the importance of teaching students 'learning templates' and a process for problem-solving.

These two Finnish studies emphasise the point that coding does not occur in a vacuum, that it is situated in a problem domain. Therefore, our attention is turned next to the context in which computing activities occur.

3.2 Contexts

Teaching activities take place in situated contexts, such as making games, using physical computing or embedded within cross-curricular topics. These contexts are not distinct: an activity may engage in several contexts. For example, a KS1 year class might be learning about direction in maths and use a programmable toy to deepen understanding. A KS2 class might create a computer game for a history topic, augmenting the input with a modelling dough game board to learn about conductivity and science at the same time. In KS3, a class might use a microcontroller to make a sensor to measure speed for maths and science learning. A KS4 class might use e-textiles to design a product in design and technology. In KS5 students could create a games app that teaches about politics.

3.2.1 Physical Computing Pedagogy

Physical computing is often linked to Papert's constructionist framework, where a learner builds meaning through making (Papert, 1980). It is implied in the English national curriculum at Key Stage 2 where learners are required to design and write programs including controlling or simulating physical systems (DfE, 2013b).

The term 'control', as used in this programme of study statement, relates to controlling actions such as turning a motor on, activating a speaker, or turning a light on and off. Associated with a physical output is the idea of a physical input, such as a button being pressed, or movement, light or sound detected. Examples of physical computing devices range from programmable robots such as the Bee-Bot[1]; robotic kits such as Lego Mindstorms[2], programmable input devices or output devices such as the Makey Makey[3]; educational microcontrollers such as the crumble[4] and micro:bit[5], tangible interfaces such as the KIBO[6]; electronic and maker kits such as LittleBits[7] and single board computers such as the Raspberry Pi[8]. A list of example devices is shown in Appendix B exemplifying the wide range of products available to teachers. This list is not exhaustive and new devices are constantly being introduced and withdrawn from the educational technology market.

Four systematic literature reviews are summarised that contribute to a view of research related to programmable robots and robotic kits in primary and secondary education. A review of the educational potential of robotics in school presented only 10 studies, reporting that empirical evidence was limited with some learners making progress in Science Technology, Engineering and Maths (STEM) and others not (Benitti, 2012). Major, Kyriacou, & Brereton (2012) reported on 36 studies of which 11 were primary or secondary and similarly concluded that there was a need for large-scale, high-quality research to determine the effectiveness of using robotics to teach programming. More recently, Toh et al. (2016) synthesised research from the previous ten years and presented 27 studies. The authors concluded that quantitative analysis and experimental methods were lacking, with only four articles cited as evidencing an increase in academic development. Across these reviews the conclusions are the same, there is limited

[1] http://www.tts-group.co.uk/bee-bot-rechargeable-floor-robot/1001794.html accessed 5/4/2017
[2] https://www.lego.com/en-us/mindstorms accessed 5/4/2017
[3] http://www.makeymakey.com/ accessed 6/4/2017
[4] http://redfernelectronics.co.uk/crumble/ accessed 14/4/2017
[5] http://microbit.org/ accessed 5/4/2017
[6] http://www.shop.kinderlabrobotics.com/KIBO-Sets_c7.htm accessed 5/4/2017
[7] http://littlebits.cc/ accessed 5/4/2017
[8] https://www.raspberrypi.org/ accessed 5/4/2017

clear evidence that using robotics results in progress in STEM or programming, with calls for high quality, larger scale research.

Falkner & Vivian (2015) highlighted that despite the growing availability of physical computing technology, they found few resources to support robotics and physical devices. They also noted that these resources did not exploit opportunities to integrate with Design and Technology; were exemplars demonstrating use; and that tutorial resources for teachers were needed as well as illustrations of appropriate pedagogy.

Across the systematic literature reviews reported here, the most cited resource set was Lego Mindstorms[9] kit. How representative this research is of what is happening in UK schools now is not clear, nor whether the approach to teach and learn with this product is the same as teaching and learning with others.

The literature reviews presented focused on robotics with the most recent study reviewing papers published up until 2013. However, since that time, there has been a seed change in the types of research being addressed, with a new interest in tangible interfaces and microcontroller studies as evidenced by the additional reports added here.

Bers and her team, at Tufts University in the US, over a five-year period, have contributed to the field of teaching and learning in early robotics and tangible interfaces developing the Positive Technological Development (PTD) framework (shown in Figure 7) and associated TangibleK curriculum (Bers et al., 2014; Bers, 2010; Horn, Crouser, & Bers, 2012; Kazakoff & Bers, 2012; Strawhacker & Bers, 2015). Their research uses products such as the Lego Education WeDo Construction sets[10], the CHERP programming language [11] (a visual block-based language with a restricted number of commands to control the WeDo components), and a tangible interface of physical blocks that represent each CHERP command. Bers' studies include control groups, with the pupils' teachers delivering the physical computing lesson material, usually of in-class courses of 20 hours. Both qualitative and quantitative data measures are used to evaluate aspects of teaching and learning.

Kazakoff & Bers (2012) concluded that pupils improved their sequencing skills as a result of learning how to program robots using the TangibleK programme. A study in 2014, concluded learners using their PTD framework and curriculum, were both interested in, and able to learn, many aspects of robotics, programming and computational thinking (Bers et al., 2014). A later 2015 study (Strawhacker & Bers,

[9] https://www.lego.com/en-us/mindstorms accessed 5/4/2017
[10] https://education.lego.com/en-gb/primary/explore/computing?CMP=KAC-EDUK16JunWeDo2campaign 5/4/2017
[11] https://ase.tufts.edu/DevTech/tangiblek/research/cherp.asp 5/4/2017

2015) compared three types of interface: tangible blocks, on-screen programming and a combination of the two. Although the results were not conclusive, the study indicated that using a tangible interface may enhance understanding of repeat loops and other abstract concepts for younger learners. These three studies showcase a systematic approach to in-school research, whereby the research community works closely with educators to create, test and refine frameworks and resources over time.

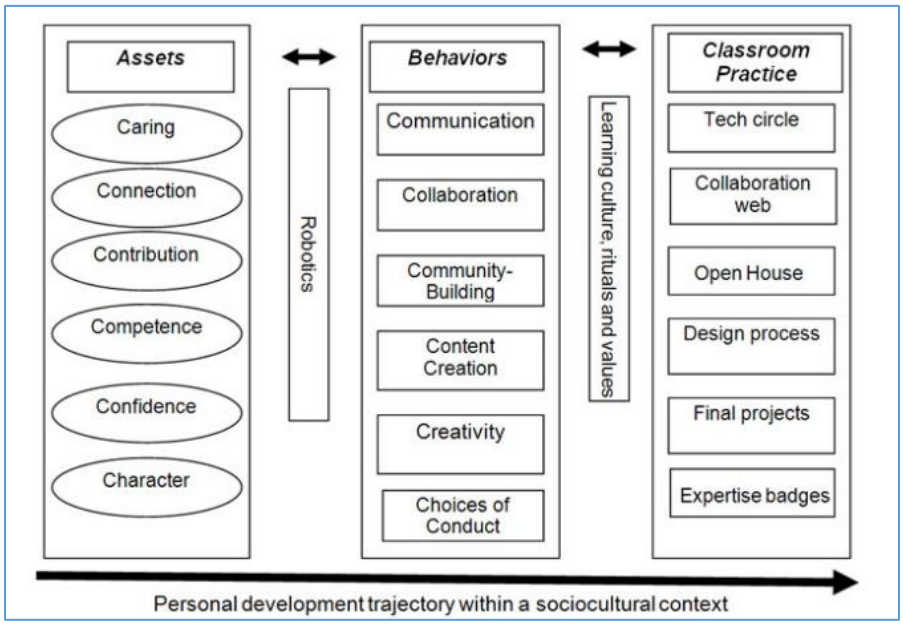

Figure 7 The PTD framework (Bers, 2010, p. 5)

In an Australian study, McDonald & Howell (2012) taught Lego WeDo robot construction lessons over a 6-week period with a small cohort of 16 KS1 pupils following an approach of 'model, explore, evaluate'. The authors cited development in emergent literacy and numeracy, digital access and basic engineering concepts. The authors recommended further investigation of the balance between teacher control and pupil autonomy. How cognitive load might be controlled by teaching sequences was explored by Jin, Haynie, & Kearns (2016) who proposed that physical computing can have a high cognitive overload and trialled a teaching sequence to reduce this. They suggested a sequence including introduction, program demonstration, learner guided activities with support from the teacher (including program design, development, testing) and post discussion. The authors concluded that cognitive load was managed and content knowledge increased. However, there was no control group to compare the outcomes to.

Our attention is turned next to microcontrollers and input and output devices, the pedagogical aspects of recent research for these areas is next outlined.

A small-scale study of KS2 learners using the Makey Makey[12] with modelling dough and other crafting materials to make game controllers and augmented game boards, led to a conclusion that crafting and coding was a valid approach for novice programmers as most learners went beyond surface changes in remixing code and design. The authors highlighted the importance of learners playing each other's games to provide an authentic audience, concluding that further research and implementation studies were needed to support the introduction of crafting and coding activities into schools (Kafai & Vasudevan, 2015).

In a recent UK focus group study, 54 KS3 students were interviewed concerning their use of the micro:bit[13]. Sentance, Waite, Hodges, MacLeod, & Yeomans (2017) reported that the device encouraged students to work creatively and motivated the learners because of its physical nature and novelty. Further, the authors suggested understanding was supported by the tangibility of the device. Teachers interviewed reported using a range of pedagogical approaches to incorporate the device in the curriculum. The authors concluded that further research was needed to support the claim that simply using such a device would guarantee the benefits illustrated by the interviews.

In a US high-school e-textiles study of 15 students over a 10-week elective course using Lilypad Arduino[14], Kafai, Lee, Searle, Fields, Kaplan & Lui (2014) reported a range of pedagogies used to support students. Suggested strategies included the use of a starter kit and starter code to learn basic skills, short code concept lessons, reading code and debugging code activities. Sample code and remixing design challenges, as well as drawing designs, was recommended as well as expert support and flexible lessons. While the students had engaged in learning through various strategies, the authors concluded that a significant finding from their study was the role of remixing in students' learning and personal and creative expression. They advised that future research should further analyse remixing of ideas and crafting techniques as well as remixing of circuit designs and code. The authors claimed the biggest success was that both girls and boys were equally engaged in the crafting, circuitry and coding.

The theme of remixing was echoed in 2017 work, again of electronic textiles using Arduino[15] for teaching 23 US high school students. The emphasis of the study was on reading, remixing and writing codable

[12] http://www.makeymakey.com/ accessed 6/4/2017
[13] http://microbit.org/ accessed 5/4/2017
[14] https://www.arduino.cc/en/Main/ArduinoBoardLilyPad accessed 14/4/2017
[15] https://www.arduino.cc/ accessed 14/4/2017

circuits. Litts, Kafai, Lui, Walker, & Widman's (2017) concluded that additional research was needed on designing and developing resources and tools to support making activities.

DesPortes, Anupam, Pathak, & DiSalvo (2016), in a small-scale study of 44 high school students using alternative breadboards, reported that cognitive load was reduced with a simplified design. They concluded that further research of cognitive load and physical computing learning environments should be undertaken.

In keeping with this theme of small-scale KS5 studies, Brinkmeier & Kalbreyer (2016) studied students (n=25) assembling and programming a model goods conveyer system. Most student time was spent dealing with problems with the model. The authors ultimately asked whether 'premade designs' might be used as a tutorial to lead to new ideas for subsequent projects and stimulate creativity.

Creativity and progression of support were also suggested by Sentance & Schwiderski-Grosche (2012). In a cross-phase study with KS3 to KS5 students, they reported that the tangible nature of the .NET Gadgeteer[16] microcontroller kit encouraged creativity, engagement, and students valued an exploratory bricolage approach to learning. They recommended future work should investigate more staged support and learning of programming concepts.

Recent work by Przybylla & Romeike (2014) and Przybylla (2016) has explored situating physical computing in secondary education. As well as relating the importance of creativity and making in physical projects, these authors explore how other aspects of the computer science curriculum can be taught through physical computing. They have suggested teaching embedded systems, control, interactive systems, memory, processor, inputs, outputs, interaction, hardware design and ubiquitous computing with physical computing. Similarly, Eickholt & Shrestha (2017) reported on opportunities to use physical computing to teach big data by learners having access to physical clusters. However, there have been no empirical studies in classroom settings concerning these curricula material.

That teachers have been using physical computing for some time, and are using a range of physical computing in class, is implied by previous English programmes of study, product donations to schools, and teacher resources. The 1999 to 2014 ICT programme of study in England specifically mentioned programmable toys, requiring pupils 'to plan and give instructions to make things happen [for example, programming a floor turtle, placing instructions in the right order]' (DfE, 1999). In 2016, all year 7, KS3,

[16] http://www.netmf.com/gadgeteer/ accessed 14/4/2017

pupils in England state schools received a microcontroller for free as part of the 2016 BBC micro:bit[17] programme (Sentance et al., 2017). The Barefoot Computing Programme (Berry et al., 2015), and the QuickStart Primary Project (Berry, 2015b) also include activities using programmable toys. The Computing At School June 2016 survey confirmed this use: 38% of some 750 teachers said they used physical computing often, 47% sometimes and 15% never (Sentance, 2016).

Therefore, there is evidence that teachers are using physical computing in class. However, the literature reviews and studies described here indicate there is limited empirical research to inform teachers' choices of what devices to use, what pedagogy to use and how effective these approaches are for learners to make progress.

3.2.2 Game-making Pedagogy

Game-making as a context for learning computing is next considered, here one recent literature review is outlined followed by a notable large scale programme that has developed a distinct pedagogy for teaching computing through games and simulations.

Kafai & Burke (2015), in their analysis of 55 studies of game-making in primary and secondary settings, found that most studies focused on teaching coding and academic content. The authors contended that making games more genuinely introduced children to technical skills and connected them to each other, countering the issue of access and diversity in traditional digital gaming cultures. Results from the review were overwhelmingly positive, but also raised four concerns. Firstly, it was difficult to synthesise findings from such diverse studies. Secondly, the studies reviewed had viewed learning differently, and the framework used to analyse them did not support all that was needed to conceptualise and assess computational thinking. Thirdly, the studies varied in the provision of basic data on research conducted, with much data missing. Fourth, few negative findings were observed in their review, apart from the lack of success for constructionist gaming to raise career aspirations in girls. Kafai and Burke suggested further work is needed for a wider view of participation, including social and cultural dimensions. They also recommended that short and long time frame studies be included in further research and collaborative arrangements should be studied. In conclusion, they called for a joining of instructionist efforts and constructionist approaches to create an approach with no distinction between players and designers coining the phrase 'connected gaming'.

[17] http://microbit.org/ accessed 5/4/2017

Over a fifteen-year period, Repenning et al.'s Scalable Games Design (SGD) project has engaged over 10,000 students in the making of games. The Colorado University led programme has been used by schools predominantly in the USA but more recently also in Mexico and Switzerland. SGD requires learners to design and program games and then transfer skills as they move on to design and program simulations in Science and other cross-curricular subjects. Rather than teaching programming by focusing on programming constructs such as loops, if-then statements and data structures, SGD focusses on teaching 'computational thinking patterns'. These patterns are based on game and simulation design patterns, such as generation, absorption, diffusion and transportation. The software used to code these design patterns is AgentSheets or AgentCubes. In a study with over 10,000 middle school students, Repenning et al. (2015) claim rapid adoption of SGD by teachers from multiple disciplines, high student motivation, high levels of participation by women, and interest regardless of demographic background. They also note the importance of learner ownership of created projects for motivation and the transition from following tutorials to creating new games.

In one study of SGD, Webb, Repenning, & Koh (2012) concluded that the way that it is taught influenced the motivation of girls and boys differently. The authors categorised the 27 teachers in the study as either delivering SGD through direct instruction, a highly scaffolded approach, or a guided discovery approach that was less scaffolded. Reporting on the results of 1420 completed student surveys, they concluded that the girls were less likely to be motivated by direct instruction. They also pointed to a negative impact on girls' motivation due to a higher ratio of boys to girls in class but stated that this influence could be overcome by using guided discovery scaffolding rather than direct instruction. The authors accepted that conclusions should not be generalised but called for further research on the impact of different teaching approaches on motivation and gender differences.

3.2.3 Unplugged Pedagogy

Teaching computing without a computer, or 'unplugged pedagogy', could be classified as an instructional technique. However, here it is reported as a context to highlight the interest and apparent popularity of the approach.

In a recent UK survey of computing teachers, unplugged pedagogy was cited as one of the most used and most successful strategies (Sentance & Csizmadia, 2016). Similarly, in curriculums and computing articles unplugged is heralded as an effective pedagogical approach (Berry, 2015b; CSTA, 2011b).

Perhaps the most well-known unplugged activities are those provided by Bell, Alexander, Freeman, & Grimley (2009), the CS Unplugged team who introduce a wide range of computer science concepts, such as binary, sorting algorithms and cryptography without the use of a computer (Bell et al., 2009).

Further unplugged activities can be found on a number of online websites including cs4fn[18] (Curzon, 2013), Teaching London Computing[19], CSTA[20] (CSTA, 2011a), Barefoot[21] (Berry et al., 2015), Digital School House[22] (Digital School House, 2016), Code.org[23] (Code.org, 2016) and Google[24] (Google, 2016).

Research related to university outreach programs is often descriptive (Bell et al., 2009; Bell & Newton, n.d.; Curzon, McOwan, Cutts, & Bell, 2009; Curzon, McOwan, Plant, & Meagher, 2014; Curzon, 2013; Cutts, Brown, Kemp, & Matheson, 2007) rather than rigorous and evaluative of long-term impact on learning. Some studies have questioned the effectiveness of CS Unplugged activities (Feaster, Segars, Wahba, & Hallstrom, 2011; Taub, Armoni, & Ben-Ari, 2012; Thies & Vahrenhold, 2012, 2016) with recommendations of the need to adapt learning for specific class settings.

Several recent studies indicate increased interest in exploiting and better understanding unplugged approached. In a small-scale study, of 11 grade 3 to 5 learners, Aggarwal, Gardner-McCune, & Touretzky (2017) investigated the use of physical manipulatives to support learners' understanding of Kodu. They reported that students who used the physical manipulatives performed better at the rule construction, whereas the students who engaged with the programming environment had a better mental simulation of the rules and a better understanding of the concepts. In separate recent study of 36 high school students in a summer camp, Ford, Siraj, Haynes, & Brown (2017) reported students showed increased understanding of cyber-related material following an unplugged project. They concluded that further unplugged cybersecurity activities would contribute to growing interest in cybersecurity education.

Rodriguez et al. (2017), reported on a 3-year research project developing and refining CS unplugged material for middle school learners. They concluded that further work was needed to understand how computer science techniques might map to computational thinking concepts. The authors urged caution

[18] http://www.cs4fn.org/ last accessed 14/5/2017
[19] https://teachinglondoncomputing.org/ last accessed 14/5/2017
[20] https://www.iste.org/explore/articleDetail?articleid=152&category=Solutions&article=Computational-thinking-for-all last accessed 14/5/2017
[21] http://barefootcas.org.uk/ last accessed 14/5/2017
[22] http://www.digitalschoolhouse.org.uk/ last accessed 14/5/2017
[23] https://code.org/curriculum/unplugged last accessed 14/52017
[24] www.google.com/edu/computational-thinking last accesed 14/5/2017

that forcing a mapping of every assessment of computing skills to computational thinking might be counterproductive.

Despite mixed evidence of the impact of unplugged activities on student learning, the approach appears to be popular with teachers as a pedagogical approach. Much more research is needed to determine how best to use it effectively.

3.2.4 Cross-curricular Pedagogy

Papert proposed, in a 2005 interview:

> 'programming is the most powerful medium of developing the sophisticated and rigorous thinking needed for mathematics, for grammar, for physics, for statistics, for all the 'hard' subjects.... In short, I believe more than ever that programming should be a key part of the intellectual development of people growing up' (Kestenbaum, 2005, p. 38).

Papert's championing of programming as a way to develop thinking in other subjects provides an attractive rational for teachers to situate computing in cross-curricular contexts. However, to what extent cross-curricular programming, or computing in more general, is occurring in the UK is not clear, nor how effective a cross-curricular approach is, nor what pedagogies are being used to implement it. Here, one related systematic literature review is presented followed by an overview of a notable cross-curricular initiative.

Moreno-León & Robles (2016) presented a systematic literature review of studies using Scratch to teach non-computing subjects. They reported on 15 papers and concluded that 8 of these studies indicated that programming could be a tool to improve learning in other subjects. However, they stated that these studies lacked rigour. The other 7 studies were more rigorous and evidenced improvements in pupils' problem-solving, logical reasoning and creativity. The authors recommended that more empirical research in classrooms, with larger samples of students, was needed to obtain clear conclusions about the types of learning that could be enhanced through programming. They did not summarise the detail of pedagogical approaches used in cross-curricular computing, but focused on evidence of effectiveness.

A notable recent cross-curricular computing initiative is the ScratchMaths programme. Funded by the Education Endowment Fund, it involved a 2-year intervention for learners aged 9-11 years. The outcome of the intervention will be measured in the summer of 2017 by national standardised mathematics test scores. The underlying pedagogy for the intervention was described as the '5Es': Explore; Explain; Envisage; Exchange; and bridgE. In an early study of the intervention before final evaluation, Benton,

Hoyles, & Noss (2017) note that live coding was likely to lead to deep learning. They also suggested that relating learning to previous maths work and returning to concepts may result in more learners successfully achieving outcomes. The authors suggested that the '5Es' gave teachers the flexibility to adapt the material in a way that met pupils' needs but that further time in CPD might be given to sharing more teaching strategies. They finished by asking what the conceptual and pedagogical obstacles to teaching maths through programming might be and how they might be addressed. Further work is needed to understand the instructional techniques available for the teaching of other subjects through computing and research to evaluate the merits and effectiveness of cross-curricular computing.

3.3 Programming Languages

In this section, research related to the pedagogy associated with the transition from block to text-based languages is presented. Following this, other programming language opportunities are reported including: toolsets which visualise programs as they are running; collaborative programming environments and research of less common programming languages.

3.3.1 Block to Text Pedagogy

Sometimes called block-based, visual or graphical programming languages, these languages use graphical images to represent programming commands (Wu, Tseng, & Huang, 2008). Educational block-based languages have been developed to be easy to get started with but to be powerful enough to create advanced programs, and have been available since the 1990's. Alice was released in 2000, Scratch in 2005, Kodu in 2009, Blockly in 2012 and GP a new graphical programming language is due out in mid-2017[25].

Block-based programming languages are currently advocated as being the most appropriate type of programming environment for young learners, such as those at primary schools, with a prediction that this will remain so for the foreseeable future (Kölling, 2015). With over twenty educational block-based languages currently available (see Appendix C) teachers must decide which is the best for their learners both for in terms of the learners' current level of expertise and how this will support their next steps in learning.

In England, learners, as they move to secondary school, are required in computing lessons to learn text-based programming languages (DfE, 2013a) and therefore, any pupil who has used a block-based language

[25] https://harc.ycr.org/project/gp/ last accessed 14/5/2017

is required to transition to text. Text-based languages include those developed for education, such as LOGO and those developed for industry, such as Python, Java and C.

In the 2015 Computing at School (CAS) annual survey, (Sentance, 2015) 96% of primary teachers (n=318) and 84% of secondary teachers who responded reported they were teaching Scratch (or that it was being taught in their school) (n=767[26]) as shown in Figure 8. Whether participants of the survey, CAS members, were representative of the entire teaching population is not known. However, it seems likely that many secondary teachers are required to transition their pupils from a block-based language to a text based one (most likely Scratch to Python, the most popular text-based secondary school language in the survey).

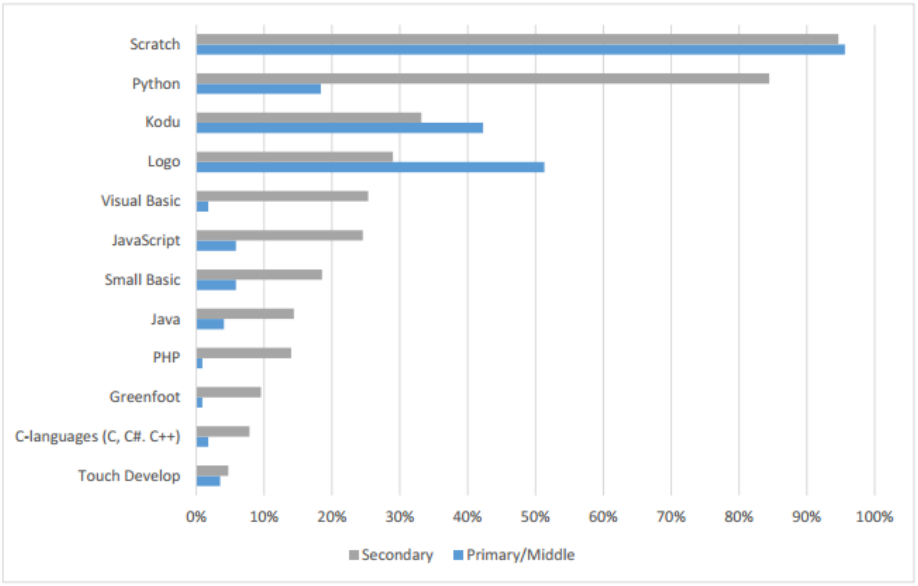

Figure 8 Primary and Secondary Programming Languages from CAS Survey 2015 (Sentance, 2015)

Despite anecdotal reports that the transition from block to text is a significant challenge for learners and teachers (Garneli, Giannakos, & Chorianopoulos, 2015; Sentance & Csizmadia, 2015), there is limited empirical research in this area, and evidence from studies is mixed (Armoni, Meerbaum-Salant, & Ben-Ari, 2015; Garlick & Cankaya, 2010; Kölling, Brown, & Altadmri, 2015; Kölling, 2015; Powers, Ecott, & Hirshfield, 2007; Weintrop et al., 2016; Weintrop & Holbert, 2017; Weintrop & Wilensky, 2015). Irrespective of the lack of conclusive academic evidence, pedagogies (Armoni et al., 2015; Dorling & White, 2015; Franklin et al., 2016; Grover & Basu, 2017; Lukkarinen & Sorva, 2016) and toolsets (Kölling et al.,

[26] Of the 137 teachers that selected 'Others', a small percentage were from further education or sixth form colleges, many were at schools that crossed the primary and secondary phases.

2015; Price, Dong, & Lipovac, 2017; Weintrop & Holbert, 2017; Weintrop & Wilensky, 2015) have been developed to attempt to support the transition.

In Falkner & Vivian's (2015) systematic review of computer science resources for primary and secondary classes, the authors stated they could find no classroom resources addressing transition. However, Garneli, Giannakos, & Chorianopoulos (2015), in the same year, in a systematic literature review of computing education in K-12 schools, cited a range of instructional approaches to support transition from block to text, including creating a solution in more than one language and using physical computing, but called for further research on pedagogies for transitioning between text, visual and tangible tools.

Dorling & White (2015) suggested using unplugged activities and side by side code to support block to text transition and presented anecdotal evidence of the effectiveness of this pedagogy in class.

Franklin et al. (2016) in an investigation of teaching initialisation in Scratch, suggested that the initialisation concept should be incorporated into the block-based programming curriculum so as to support understanding of the concept in text-based languages. However, they provided no empirical evidence to substantiate their proposal. They also recommended analysis of other operations and constructs to identify further transition opportunities. In keeping with this suggestion, that transition to text based languages should be considered when teaching block based programming, Grover et al. (2015) reported on 54 students using their FACT curriculum. It was developed specifically to support transfer from block to text-based programming. The authors concluded that students could transfer from block to text but students' ability to transfer learning was dependent on earlier learning and the depth of understanding of underlying concepts and constructs.

There is an implication here that prior learning of block-based programming may impact on students' ability to learn text-based languages. This was evidenced in a study by Armoni et al. (2015) on students learning C# and Java. Students with prior learning in Scratch performed better in a range of text-based programming tests and recognised text-based programming concepts earlier in the teaching process than students with no prior experience. In their extensive study of 120 students the authors also reported anecdotal evidence that the teaching process was shortened; there were reduced difficulties in teaching and learning and higher self-confidence for those learners with Scratch experience. They suggested further research was needed in to why Scratch improves short-term learning of some concepts, such as repeated loops, but not others, such as variables. Further, the authors called for investigation of the longer-term impact of having learned to program with blocks before text.

In a mixed KS5 and higher education study, Weintrop & Holbert (2017) investigated the use of Pencil Code, a hybrid programming language, where learners can switch between block and text modes. They concluded that novice programmers used both modalities throughout their programming experience. They also reported that all learners started with the block-based mode first; with some quickly moving to text while others stayed in block-based mode, and all learners successfully completed the tasks given. Weintrop & Holbert suggested that the opportunity to switch between modalities provided a means by which all learners could participate while keeping the more experienced programmers engaged.

In a separate two-year, higher education study, Dann, Cosgrove, Slater, Culyba, & Cooper (2012) employed an adapted version of Alice, to help learners transition between Alice and Java. The pedagogy combined several approaches and included:

- instructors showing sample code and directly comparing individual commands that accomplished the same outcome in Alice 3 and Java;
- learners reading code and learners writing code equivalents on paper;
- learners modifying code and undertaking debugging activities.

The intervention was implemented across two successive years, with consistent improvements of learners' achieving a full grade higher in both trials.

Price, Brown, Lipovac, Barnes, & Kölling's (2016) small scale evaluation of Stride, a frame-based programming language, concluded that learners using Stride completed tasks set more quickly than those writing Java using a traditional editor. They also reported the Stride users spent less time on syntactic edits to their code and significantly less time with non-compilable code. The study was limited due to sample size, sample selection and the short-term nature of the out of school study. The authors called for further research on the relationship between block, frame and text editors, how the transition might be mediated, with a focus on long-term effects and impact on learning gains, as well as a more in-depth investigation of learner perception of the editors.

Hybrid programming languages such as edublocks[27] (a block to Python text hybrid), pencilcode[28], j2code[29], and applab[30] (block to JavaScript hybrids) are now available for teachers to use in class. Some are free;

[27] http://edublocks.org/ accessed 17/4/2017
[28] https://pencilcode.net/ accessed 17/4/2017
[29] https://www.j2e.com/visual.html?edit accessed 17/4/2017
[30] https://code.org/educate/applab accessed 17/4/2017

others require subscription. How these products are being used in school and with what success in classroom settings has not yet been studied.

3.3.2 Other Programming Language Opportunities

To address the development of learners' understanding of the notional machine (Du Boulay, 1986), toolsets have been created that visualise the behaviour of programs as they run. Sorva, Karavirta, & Malmi (2013) reviewed such systems, recommending research on their integration in to introductory programming pedagogy.

A notable stream of such research is that by a Finnish research team on the ViLLE tool. Over a series of university student studies, they reported that program visualisation is particularly useful to novice programmers (Rajala, Laakso, Kaila, & Salakoski, 2008) and when learners are familiarised with the tool before use (Laakso, Rajala, Kaila, & Salakoski, 2008). They found the tool is more effective when used collaboratively (Rajala, Kaila, Laakso, & Salakoski, 2009) and when learners are actively engaged using it in exercises (Kaila, Laakso, Rajala, & Salakoski, 2009). In a 2010 study of the long-term effects of ViLLE on high school students (aged 16 to 19) learning Python, Kaila, Rajala, Laakso, & Salakoski (2010) reported significantly better final exam results for those students using ViLLE throughout the course. They concluded that program visualisations could be a highly beneficial method for teaching basic programming concepts, particularly when tracing the execution of function calls. They also pointed to the opportunity the tool afforded for lots of code reading activity, the significance of learners being given real-time feedback and program visualisation supporting independent rehearsal of code execution(Kaila et al., 2010).

Online toolsets to support collaborative learning are popular in professional domains and have been trialled in university settings. However, this has not yet been explored with the younger school aged group. Al-Jarrah & Pontelli (2014) developed a version of Alice, AliCe-ViLlagE (Alice as a Collaborative Virtual Learning Environment), for collaborative learning specifically to support 'virtual pair programming' where two learners remotely share their virtual world and interact in its construction. Empirical investigation is needed to evaluate the usefulness and effectiveness of this approach.

Further strands of research have investigated other programming languages such Flip (Good, 2011), Processing (Colubri & Fry, 2012; Parrish, Fry, & Reas, 2016), route based programming (Gujberova & Kalas, 2013) and NetsBlox for teaching distributed programming (Broll et al., 2017).

3.4 Student Engagement

Teachers can choose from a variety of approaches how learners will contribute to, and be engaged in, a learning activity. Firstly, pair programming is reviewed followed by an overview of other student engagement approaches including various problem-solving approaches.

3.4.1 Pair Programming

Used in industry and education, pair programming is a collaborative approach to programming where two people work at one computer to complete a single design, algorithm, coding or testing task (Williams & Kessler, 2000). One person takes the role of the driver, having control over the keyboard and mouse, and the second person is the navigator or observer. The navigator constantly reviews the code written, keeps track of progress against the design (McDowell, Werner, Bullock, & Fernald, 2006) and continuously collaborates (Williams & Kessler, 2000). While working on a task, the driver and navigator swap roles after a certain period of time and code is only changed with the agreement of both parties (McDowell et al., 2006).

Studies of pair programming as a pedagogical tool in primary and secondary computing education are scarce. A literature review of game-making that references pair programming (Kafai & Burke, 2015), a systematic literature review of resources (Falkner & Vivian, 2015) and two higher education literature reviews on pair programming (Hanks, Fitzgerald, McCauley, Murphy, & Zander, 2011; Salleh, Mendes, & Grundy, 2011) are summarised below.

In Kafai & Burke's (2015) analysis of 55 studies of game-making in primary and secondary settings, collaborative aspects were infrequently attended to, with the exception of two studies which referred to pair programming, both by US researchers Werner and Denner. They found insufficient research into the way that collaboration in game-making can be constructed and supported. Falkner & Vivian (2015) study found that lesson plans rarely included discussion of the pedagogy used, such as pair programming and there was little guidance for teachers on student project management or pedagogy for student teamwork. The authors recommended more rigorous research within conventional classroom settings. However, there is evidence that pair programming, compared to independent coding, improves higher education learners' grades on assignments, although it did not improve final exam scores (Salleh et al. 2011).

Hanks et al. (2011) concurred that there was evidence that pair programming led to improved learning for both task outcome and code quality in a unversity setting. They recommended further study of partner compatibility and investigation of the detail of how and why pair programming works or does not work.

They suggested broadening research to investigate pair programming in K-12 education and research of less strictly defined collaborative strategies.

Werner and Denner, have co-authored on US research studies for more than 10 years to build a developing body of computing education work with middle school learners. Included in their research, is consideration of pair programming. A number of research studies has culminated in the development of a 'pair effectiveness score'. They concluded that programming knowledge increased over time but that the greatest increases in knowledge occurred for confident partners who were paired with a friend who has relatively more initial programming knowledge. However, the findings were limited due to how 'friendship' was assessed and the small sample size (Werner et al., 2013).

In a 2014 paper, comparisons were made between those learners working in pairs and those working 'solo' leading to the conclusion that working with a partner had advantages for building programming knowledge and computational thinking (Denner, Werner, Campe, & Ortiz, 2014). Using video recordings, their most recent 2016 study, compared in detail the body language and verbal communication of Latino, White and mixed pairs of girls' while they pair programmed. Ruvalcaba, Werner, & Denner (2016) reported evidence of subtle differences in approaches to collaboration related to ethnicity, but raised questions about the validity of their conclusions due to the small sample size, calling for further research.

In a separate summer school study of 40 grade 6 learners, Lewis (2011) also compared pair programming to solo programming, concluding that pair programming resulted in pairs completing **less work**, and **did not increase overall progression in learning** compared to the solo programmers. These findings appear to contradict the findings of the Denner and Werner team. However, Lewis' solo programmers did not work in isolation; they worked collaboratively through peer support with an assigned partner. Also, the pairings of Lewis' learners' were set and changed every day by the course leader. Whereas in Denner et al.'s studies learners were involved in choosing their pair assignment, were then given an initial trial period, and finally assigned a partner for the duration of the course. In the Lewis study, roles were swapped every 5 minutes, whereas in the Denner et al. studies this happened every 20 minutes. There were other differences between the studies, including the pedagogy used to deliver material (Denner et al., 2014; Lewis, 2011; Werner et al., 2013). How levels of participation and agency were influenced by the different working practises in the studies merits further research.

In a recent 2016 Italian study of the use of the agile software methodology in high school (KS5), Missiroli, Russo, & Ciancarini (2016) concluded that pair programming was motivational, improved code quality and

for some learners their grades. However, despite a study population of over 80 students, the length of study, of one day, leads to a question of long-term impact.

In Sentance & Csizmadia's (2016) survey of UK primary and secondary teachers, teachers were asked what successful strategies they used in computing classes. Of the 339 survey respondents to this question, peer mentoring was mentioned 32 times, team coding/pair programming 23 times, and collaboration 22 times. However, the survey population might not have been representative as it was sourced from the UK computing teachers' association, Computing At School, membership.

Pair programming appears to be an attractive method for engaging students in the process of programming, with evidence that it can improve teaching and learning. How effective the approach is within UK classroom settings is yet to be determined.

Industry research of pair programming (Plonka, Segal, Sharp, & Linden, 2011) reported professional programmers switched role in a fluid way, without a complicated timing routine, and that on average 33% of the time was 'non-driving' time, including waiting-time, discussions, use of external representations, searching for advice from others and interruptions. The pairs with the highest non-driving time had complex problems which led to use of external representations and discussion. How off-screen time is used by school pupils, as they tackle complex problems may be a valuable avenue of research within studies of collaborative pedagogy.

3.4.2 Other Learner Engagement

Beyond pair programming, there is also an opportunity to explore other pedagogy for learner engagement in computing lessons. The Digital Leaders Network[31] advocates collaborative learning through pupil peer support and apprenticeship. Passey (2014) reported a case study of Digital Leaders, in which they concluded that the initiative involved some students who tended not to be normally involved in leadership activities. The authors highlighted that digital leaders provided technological support and advice for peers and teaching staff. However, Passey did not quantitatively evaluate the impact on teaching and learning nor define the pedagogical approaches used. The authors recommended further research on the balance of activities undertaken and outcomes of interventions as well as an investigation of perceptions of the programme on digital leaders themselves. Ching & Kafai (2008), reported on peer pedagogy, similarly

[31] http://www.digitalleadernetwork.co.uk/ accessed 16.4.2017

recommending further research on learners' explicit motivations and reasoning about peer pedagogy as well as the effect on learning for the pupils providing the peer support.

Several studies have investigated the use of agile methodologies in high schools. Missiroli et al. (2016) concluded groups with mixed skills performed better than those with the same skill level, and there was a general increase in code quality and student satisfaction. In another study, Kastl & Romeike (2015) reported teachers as saying that learners using agile in problem-based learning (PBL) projects were more self-sufficient. In a summary of agile projects across schools, Kastl, Kiesmüller, & Romeike (2016) stated that the objectives for all learners had been met. However, these three studies were not quantitative, had small populations, and did not have a control to compare against. Therefore, conclusions about the pedagogy associated with agile methodologies and their effectiveness are currently promising but limited.

Research into structured problem-, process- and project-based approaches which claim to improve learner engagement and teaching outcomes provide promising results. However, studies are often of small number of learners, or opinion pieces and call for more rigorous work to be undertaken. Problem-based learning (PBL), originally developed for medical training, has different interpretations of what constitutes a PBL project (Michaelson, 2015). However, evidence from higher education indicates improved motivation and increases in generic skills from using the approach (Nuutila, Törmä, & Malmi, 2005). Similarly, Process Oriented Guided Inquiry Learning (PoGiL) originally developed for chemistry education, appears to provide opportunities for application in computing lessons (Kussmaul, 2012) with claims of its effectiveness in a comparative case study of two USA middle school teachers' experiences (Griffin, Pirmann, & Gray, 2016). Garneli, Giannakos, Chorianopoulos, & Jaccheri (2015), in a study of 53 middle school learners, compared learning to program in three scenarios, namely using a Project Based Learning (PjBL) strategy, a traditional learning strategy and a game development strategy. They reported that the PjBL students completed their activity with fewer mistakes, while the traditional group experimented with more complex concepts, though not always successfully. The authors acknowledged generalisation from the study were limited due to the specificity of the population. They called for further research to be undertaken to explore the three approaches.

There is also opportunity to build upon the 'student contribution pedagogy' framework originally developed for older learners (Hamer et al., 2008). As well as merit in investigating code reviews (Bergin et al., n.d.), community and computational participation (Ching & Kafai, 2008; Kafai & Burke, 2013), peer instruction (Porter et al., 2016), value of peer interaction (Cajander, Daniels, & McDermott, 2012) and collaborative problem-solving (Cukurova, Avramides, Spikol, Luckin, & Mavrikis, 2016).

4 Discussion and Recommendations

Generally, there is limited clear empirical evidence to support advice on pedagogies for use in schools. The research focus has been on older learners in higher education with that for school aged learners predominantly lacking applicability or rigour, due to small scale of studies, short time frames, out of school settings or methodological shortfalls (Benitti, 2012; Falkner & Vivian, 2015; Garneli, Giannakos, & Chorianopoulos, 2015; Kafai & Burke, 2015; Lye & Koh, 2014; Major et al., 2012; Moreno-León & Robles, 2016; Toh et al., 2016). However, there are some gems of research, contributed by communities undertaking longer term programmes (Benton et al., 2016; Bers et al., 2014; Grover et al., 2015; Hansen, Hansen, et al., 2016; Kafai & Vasudevan, 2015; Kaila et al., 2010; Meerbaum-Salant et al., 2013; Repenning et al., 2015; Werner, Denner, & Campe, 2015) with promising results on which UK research could build. Similarly, there are many rich threads of research with novice programmers at university which provide starting points for classroom research. For each theme, we provide discussion and specific recommendations but first we state generic recommendations that hold across all themes.

1. **Why is programming difficult?**

   First of all, investigation of why programming is so difficult, in any context, for any learner should be a focus as well as exploring what concepts are difficult to grasp and what barriers to learning and misconceptions prevail (Gal-Ezer & Zur, 2004; Grover et al., 2015; Sudol-DeLyser et al., 2012; Veerasamy et al., 2016).

2. **How to support teachers?**

   As well conducting investigations in classrooms with pupils, research focusing on teachers is also recommended. Consideration should be given of teachers' perceptions and understanding of pedagogy and how they can be involved in their own ongoing professional development (Buchholz, Saeli, & Schulte, 2013; Menekse, 2015; Rahimi, Barendsen, & Henze, 2016; Rolandsson, 2012; Sentance & Csizmadia, 2015; Yadav, Gretter, Hambrusch, & Sands, 2016).

3. **Which pedagogy for which learner?**

   Attention should be given to the effectiveness of pedagogies in different phases of education and for different learners with consideration of gender, diversity and inclusion (Hansen, Hansen, et al., 2016; Teague & Lister, 2014b; Webb et al., 2012).

4. **What role might vocabulary and tools play?**

Similarly, the role of vocabulary (Grover & Pea, 2013a; Statter & Armoni, 2016), manipulatives (Aggarwal et al., 2017; Benton et al., 2016), tools (Busjahn et al., 2015; Dwyer et al., 2015; Kaila et al., 2010; Sorva et al., 2013) and resources to support, augment and transform learning should be considered.

5. How can computational thinking be effectively embedded?

Despite a lack of consensus on the merit of computational thinking and exactly what it (Barr & Stephenson, 2011; CSTA, 2011a; Grover & Pea, 2013b; Lye & Koh, 2014; Selby & Woollard, 2014; Tedre & Denning, 2016), emerging guidance for teachers incorporates computational thinking in a variety of forms in computing materials and curricula (Benton et al., 2016; Berry, 2015a; Berry et al., 2015; Bers et al., 2014; Brennan & Resnick, 2012; Google, 2016; Grover et al., 2015; Gujberova & Kalas, 2013; Hansen, Hansen, et al., 2016; Kafai & Burke, 2015; Lee et al., 2011; Repenning et al., 2015; Rodriguez et al., 2017; Seiter & Foreman, 2013; Weintrop et al., 2016). Therefore, any ongoing research on computing pedagogy requires review of what computational thinking is, how it impacts on teaching and learning and its role within the pedagogy advocated.

6. What is the current practice? What is already known?

For each of the review categories, it would be useful to survey current practice as a precursor to more substantial research. These surveys should include a review of the associated computing curricula and resources used by, and available to, teachers. These materials should be correlated to learning models and instructional techniques as outline in 4.1. Similarly, systematic literature reviews of each category are recommended as first steps of any significant programme of investigation.

4.1 Learning Models and Instructional Techniques

Despite UK curricula requiring classroom practitioners to teach computing[32] [33] [34] (DfE, 2013a, 2013b) there is limited rigorous empirical research related to the underpinning pedagogy that teachers should use to inform teaching and learning of computing (Falkner & Vivian, 2015; Garneli, Giannakos, & Chorianopoulos, 2015; Lye & Koh, 2014; Rich et al., 2017). Studies mention that resources available to teachers focus on

[32] http://learning.gov.wales/resources/browse-all/digital-competence-framework/?lang=en accessed 13/05/2017
[33] https://www.education.gov.scot/Documents/Technologies-es-os.pdf accessed 13/05/2017
[34] http://www.nicurriculum.org.uk/curriculum_microsite/uict_ks1_and_ks2/what_is_UICT/index.asp
http://ccea.org.uk/sites/default/files/docs/curriculum/area_of_learning/statutory_requirements/statutory_curriculum_ks3.pdf
http://ccea.org.uk/curriculum/key_stage_4/areas_learning/science_and_technology
accessed 13/05/2017

coding and content, rather than problem-solving and pedagogy (Falkner & Vivian, 2015; Kafai & Vasudevan, 2015; Rich et al., 2017), and that computing education research is rarely situated in school settings (Lye & Koh, 2014). Most attention, to-date, appears to have been focused on investigations with university students or small groups of pupils. Notable exceptions include several long-term, centrally-funded curricula development programmes such as the Israeli curriculum (Meerbaum-Salant et al., 2013), the Universal Design for Learning (UDL) initiative (Hansen, Hansen, et al., 2016), Grover et al.'s (2015) Foundations for Advancing Computational Thinking (FACT) blended pedagogy, the Positive Technological Development (PTD) framework (Bers et al., 2014),the ScratchMaths intervention funded by the Education Endowment Fund (Benton et al., 2016, 2017), the Scalable Games Design project (Repenning et al., 2015) and Brennan & Resnick's (2012) work on Scratch. However, these programs vary in confidence of outcomes, scale, focus and coverage. There is clearly promising work to build upon, but how these programmes relate to the requirements of UK school teachers is not clear.

Synergies between the notational machine model (Du Boulay, 1986); levels of abstraction framework (Armoni, 2013; Perrenet & Kaasenbrood, 2006; Statter & Armoni, 2016; Taub et al., 2014); abstraction transition taxonomy (Cutts et al., 2012); discourse intensive pedagogy (Grover & Pea, 2013a) and Use-Modify-Create approach (Lee et al., 2011) should be explored to create a cohesive view. There are opportunities here to address calls for more focus on design (Falkner & Vivian, 2015; Rich et al., 2017) and learning how to abstract problems (Lokkila et al., 2016).

Similarly, learning models and instructional techniques should be audited against emerging primary and secondary frameworks (Benton et al., 2016; Grover et al., 2015; Hansen, Hansen, et al., 2016; Meerbaum-Salant et al., 2013; Repenning et al., 2015).

A recurrent theme across studies was the debate related to how scaffolded teaching should be, with tension between constructivist exploration (Ackermann, 2001; Piaget, 1951; Solomon, 1986), constructionist making (Brennan & Resnick, 2012; Lye & Koh, 2014; Papert, 1980), and a controlled progression of the teaching of more difficult concepts (Hubwieser et al., 2014; Kirschner et al., 2006; Lourenço, 2012; Meerbaum-Salant et al., 2013; Sentance & Schwiderski-Grosche, 2012; Statter & Armoni, 2016; Sweller et al., 2007; Teague & Lister, 2014a, 2014b). A blended approach encompassing a range of pedagogies is advocated by some (Garneli, Giannakos, & Chorianopoulos, 2015; Grover et al., 2015; Grover & Basu, 2017; Hansen, Hansen, et al., 2016; Kafai & Burke, 2015). With others highlighting the importance of differentiation and access for all (Hansen, Hansen, et al., 2016; Teague & Lister, 2014b) and management of cognitive load (Alexandron et al., 2014; DesPortes et al., 2016; Jin et al., 2016; Margulieux

& Catrambone, 2016; Paas et al., 2003; Tsai et al., 2015; Van Merrienboer & Sweller, 2005). It is imperative that this continuum of scaffolding be explored and guidance provided for teachers to better understand the choices available to them.

Furthermore, there is merit in the investigation of extremely fine-grained programming and understanding of what program correctness means (Aivaloglou & Hermans, 2016; Kolikant & Mussai, 2008; Meerbaum-Salant et al., 2011; Rich et al., 2017) for both primary and secondary learners and their teachers.

There is also an opportunity to build upon a rich seam of research with novice university programmers related to the relationship between code reading, tracing and writing (Busjahn & Schulte, 2013; Busjahn et al., 2015; Corney et al., 2012; DeLyser, Mascio, & Finkel, 2016; Dwyer et al., 2015; Gal-Ezer & Zur, 2004; Gujberova & Kalas, 2013; Lister et al., 2009; Lister, 2011; Lopez et al., 2008; Teague & Lister, 2014c; Venables et al., 2009) for both primary and secondary learners and align this perhaps to the Use-Modify-Create framework (Lee et al., 2011), the Block Model (Schulte, 2008) and path diagram (Lopez et al., 2008).

Investigation is recommended in primary and secondary school settings of worked examples (Sudol-DeLyser et al., 2012), subgoal modelling (Margulieux & Catrambone, 2016; Morrison et al., 2016), code annotation (Su et al., 2014), live coding and think aloud techniques (Grover & Pea, 2013a; Lye & Koh, 2014; Rubin, 2013), pseudo code and reference languages (Cutts et al., 2014) and both what the main misconceptions are and how to overcome them (Gal-Ezer & Zur, 2004; Lokkila et al., 2016; Veerasamy et al., 2016).

A high priority should be to audit the pedagogical foundations of centrally developed programmes that are currently recommended to teachers such as the Barefoot Programme[35], QuickStart[36], Tenderfoot[37] and PlanC[38] materials as well as other popular curriculum resources.

Further, the role of computational thinking in primary computing should be reviewed particularly related to how it is incorporated in teaching programming. There are risks that computational thinking in primary may not be incorporated in programming tasks and only taught through unplugged cross curricular activities.

[35] http://barefootcas.org.uk/ accessed 14/4/2017
[36] http://primary.quickstartcomputing.org/ accessed 14/4/2017
[37] https://www.computingatschool.org.uk/custom_pages/56-tenderfoot accessed 14/7/2017
[38] http://www.cas.scot/plan-c/ accessed 14/4/2017

Recommended research opportunities

In summary, it is recommended that there is need to evaluate and develop **learning models, curricula frameworks, specific instructional techniques and teaching strategies** for computing in school.

Studies related to specific instructional techniques and frameworks should include classroom investigation of the impact and/or effectiveness of:

- different views on program correctness (of both teachers and students);
- extremely fine graining programming;
- code reading;
- code tracing;
- subgoal modelling;
- code annotation;
- live coding;
- worked program examples;
- using a reference language;
- techniques for addressing misconceptions;
- specific misconceptions such as algorithm efficiency, variables & assignment;
- the Use Modify Create framework;
- copy code and other direct instruction approaches;
- the role of design in programming projects;
- tinkering and exploratory learning;
- guided discovery;
- adapting and remixing;
- think aloud techniques;
- learning templates;
- computational thinking in programming activities.

Specific attention should be given to investigating the gaps outlined by Falkner & Vivian (2015) of:

- data and functional requirements analysis;
- algorithm design and evaluation;
- programming as an element of this process;
- evaluation and critical analysis.

4.2 Contexts

The contexts reported here are not exhaustive as puzzle based activities, route based problems, simulation tasks, working on contests are other examples of contexts for programming and computing projects. A review of contexts available to teachers and potential new contexts would be a good starting point for ongoing research. This would provide a framework to investigate and evaluate types of pedagogy particularly associated with different contexts, perhaps linked to underlying concepts, models and approaches such as in Section 3.1 and recommendations in Section 4.1 and vice versa.

4.2.1 Physical computing

Using programmable robots to teach programming is not new. The work of Papert (1980) in the 1970's and 80's inspired several devices such as the Roamer[39] and Bee-Bot. Similarly, the Raspberry Pi[40], Arduino[41] and similar products have been available for use in education for many years. However, the emergence of the maker community and the development of low-cost educational microcontrollers and block-based programming languages has created renewed interest and new opportunities for teachers to consider. In line with these recent changes to the physical computing landscape, research has started to emerge, but it is fragmented and limited (Benitti, 2012; Falkner & Vivian, 2015; Major et al., 2012; Toh et al., 2016) . Bers (2010), Przybylla & Romeike (2014) and Kafai et al. (2014) are starting to develop physical computing frameworks and approaches that can be built upon. However, further work is needed to validate these approaches in UK class settings. Without large-scale, robust empirical studies that have evaluated physical computing pedagogies, there is a risk that schools will invest in resources that they do not use effectively and do not fully exploit.

Recommended research opportunities

With respect to physical computing, research is needed to develop and evaluate pedagogies for primary and secondary school use of:

- **tangible interfaces;**
- **microcontrollers;**
- **programmable robots**;
- **other subjects** (such as art & design, design & technology, science, maths and music) using physical computing;

[39] http://www.valiant-technology.com/uk/pages/roamertoohome.php?cat=8&8 accessed 14/4/2017
[40] https://www.raspberrypi.org/ accessed 14/4/2017
[41] https://www.arduino.cc/ accessed 14/4/2017

- **computer science concepts** (such as networks, cybersecurity, big data, hardware) using physical computing.

Further, there are opportunities to compare and evaluate the teaching strategies employed in physical computing studies with a focus on:

- remixing;
- creativity;
- the distinction between the design phase and coding phase;
- cognitive load;
- learning about crafting and behaviour of the components and devices.

4.2.2 Game-making

Using games to learn how to program is cited as being highly motivational (Kafai & Burke, 2015; Repenning et al., 2015). However, what pedagogies are particularly suited to game-making rather than other contexts is not clear.

There are opportunities to compare the teaching strategies of game-making studies (Kafai & Burke, 2015; Repenning et al., 2015) to the techniques, models and approaches outlined in Section 3.1 and recommendations in Section 4.1 and vice versa.

Recommended research opportunities

Research is needed to develop and evaluate pedagogies for **game-making in primary and secondary schools**.

Studies should include classroom investigation:

- transition from following tutorials to creating new games;
- using a design pattern specific pedagogy for teaching gaming rather than teaching programming constructs;
- social and cultural dimension of gaming;
- gender differences.

4.2.3 Unplugged

Despite mixed evidence of the effectiveness of the unplugged approach to teaching computing (Bell et al., 2009; Curzon, 2013; Feaster et al., 2011; Thies & Vahrenhold, 2016). There is some new evidence of positive outcomes (Ford et al., 2017; Rodriguez et al., 2017). However, this is limited. Teachers claiming the effectiveness of unplugged pedagogy (Sentance & Csizmadia, 2015, 2016) may be doing so because they are adapting activities, and are situating them within a planned progression. Therefore, there is a need for rigorous classroom research to evaluate how teachers are using unplugged activities, how they can most effectively be used, as well as an evaluation of effectiveness. Underlying theory of why unplugged approaches are believed to work and the validation or otherwise of such theory is urgent.

Recommended research opportunities

Research is needed to develop and evaluate teaching and learning pedagogies for **unplugged approaches in primary and secondary schools.**

Studies should include classroom investigation of:

- how teachers are successfully embedding unplugged activities in programming projects;
- how teachers are successfully embedding unplugged activities to teach computational thinking;
- how teachers are successfully embedding unplugged activities to teach computer science concepts;
- how best to use unplugged activities, and the effectiveness of the different types of unplugged activities, including how they should be best combined with other approaches such as in teaching programming.

4.2.4 Cross-curricular Teaching

Moreno-León & Robles (2016) reported promising evidence that cross-curricular learning can be achieved through computing contexts. However, they did not report on the pedagogies used and called for empirical and larger scale research to provide clear conclusions on the effectiveness of using programming to teach other subjects. Falkner & Vivian (2015) noted a lack of pedagogical advice related to the integration of design and technology in physical computing resources.

Cross-curricular opportunities were mentioned in several studies. In Repenning et al.'s (2015) Scalable Games Design Programme the pedagogy is predicated on using science or other subjects for the making of simulations as a context in which to apply and develop knowledge, skills and understanding acquired during preceding learning through game-making. McDonald & Howell (2012) cited the development of

emergent literacy and mathematics in a study using physical computing. However, neither of these studies provide quantitative evidence of progress made in the 'other subjects', nor detail the underlying learning models and instructional techniques used to teach the 'other subjects'.

If learners taking part in the ScratchMaths intervention (Benton et al., 2016, 2017) show improved exam results in national Maths tests in 2017, then this research, may have a profound impact on interest in using 'coding to learn' (Resnick, 2013) and afford a high-profile opportunity to champion cross-curricular computing. This intervention has clearly stated pedagogical foundations and detailed instructional approaches which could be built upon to develop further maths and 'other subject' curricula material.

Recommended research opportunities

Research is needed to develop and evaluate teaching and learning pedagogies for **cross-curricular computing in primary and secondary schools**.

Studies should include classroom investigation of:

- the instructional techniques for teaching 'other subjects' through computing;
- the merits and effectiveness of cross-curricular computing both for computing and the paired subject.

A suggested priority is to evaluate the pedagogies used by the Barefoot Programme[42] as this initiative provides a range of cross-curricular computing resources that are recommended to primary schools (Berry et al., 2015). Similarly, the pedagogies used by popular cross-curricular computing materials produced by universities, local authority teams, commercial groups, schools and individual teachers should be evaluated to provide additional information to teachers so they can make more informed choices and adapt material as needed.

4.3 Programming Languages

There are significant opportunities within the UK to add to the body of understanding in the transition of learners from block to text-based programming. Kölling's team at King's College London have developed a toolset, Greenfoot and Stride (Kölling et al., 2015; Kölling, 2015), specifically to address this challenge, and are keen to support researchers undertaking trials in school. There are opportunities to build upon the work by Weintrop and Price (Price & Barnes, 2015; Price et al., 2016; Weintrop & Holbert, 2017;

[42] http://barefootcas.org.uk/ accessed 14/4/2017

Weintrop & Wilensky, 2015), Grover et al. (2015), Armoni et al. (2015) and Dann et al. (2012) who have completed promising work in this area.

Further, the release of GP[43] due in 2017 will generate much interest and afford an opportunity to study its implementation in schools. It is expected this product may be attractive to primary schools looking for progression beyond Scratch. There are opportunities to compare the pedagogies associated with Scratch, 2Code[44], Espresso Coding[45], Code.org[46] and other tools popular in the UK. This work could build upon the recommendations and experiences of Franklin et al. (2016) and Armoni et al. (2015).

There are also opportunities to review the effectiveness of, and how to best use: program visualisation tools (Kaila et al., 2009, 2010; Laakso et al., 2008; Rajala et al., 2008; Rajala, Salakoski, Laakso, Kaila, & others, 2009; Sorva et al., 2013); online programming collaboration tools (Al-Jarrah & Pontelli, 2014); other programming languages such as Flip (Good, 2011) and Processing (Colubri & Fry, 2012; Parrish et al., 2016); puzzle and route based environments (Gujberova & Kalas, 2013) and NetsBlox for teaching distributed programming (Broll et al., 2017).

Recommended research opportunities

Research is needed to develop and evaluate the pedagogies associated with:

1. **Transition from block to text programming in secondary schools;**

   Studies should include classroom investigation of the best way to use and effectiveness of:

   - frame-based editors;
   - hybrid program languages;
   - use of physical computing to support transition;
   - unplugged, side-by-side code and other instructional approaches;
   - specially designed transition curricula.

2. **Preparing primary pupils for the transition from block to text-based programming;**

   Studies should include classroom investigation of:

   - block-based curricula which have been specially designed with transition in mind;

[43] https://harc.ycr.org/project/gp/ accessed 17/4/2017
[44] http://www.2simple.com/2Code accessed 17/4/2017
[45] http://www.discoveryeducation.co.uk/what-we-offer/discovery-education-coding#newlook accessed 17/4/2017
[46] https://code.org/ accessed 17/4/2017

- why specific programming constructs transition easily and other are difficult.

3. **Use of program visualisation tools.**

4.4 Student Engagement

Despite limited studies of younger learners' use of pair programming, recommendations regarding student engagement from reviews of pedagogy (Falkner & Vivian, 2015; Kafai & Vasudevan, 2015) align with university studies' calls for further research on understanding in more detail why and how pairings do or do not work (Hanks et al., 2011; Salleh et al., 2011). There are opportunities to build upon work on 'pair effectiveness' (Denner et al., 2014) and influences on pair effectiveness (Ruvalcaba et al., 2016). Investigation of off-screen activity during pair programming (Plonka et al., 2011) has merit perhaps aligned to research on discourse (Grover & Pea, 2013a), levels of abstraction (Armoni, 2013; Statter & Armoni, 2016), abstraction transition (AT) taxonomy (Cutts et al., 2012) and calls for design to be included in programming projects (Falkner & Vivian, 2015; Rich et al., 2017).

Recommended research opportunities

Research is needed to develop and evaluate pedagogies for **student engagement** in primary and secondary schools.

Studies should include classroom investigation of:

- **pair programming** including building upon work on pair effectiveness and influences on collaborative engagement and off-screen time collaboration;
- **apprenticeship, digital leaders and peer instruction**;
- **other forms of student contribution** including collaborative problem-solving.

A suggested priority is to review pedagogy employed with digital leaders(Passey, 2014), apprenticeship and peer instruction (Cajander et al., 2012; Ching & Kafai, 2008; Porter et al., 2016) as, despite a lack of research related to these approaches, they appear to be popular in UK schools.

Further, studies to develop the 'student contribution pedagogy' (Hamer et al., 2008), investigate problem-solving approaches (Garneli, Giannakos, Chorianopoulos, et al., 2015; Griffin et al., 2016; Kastl et al., 2016; Kastl & Romeike, 2015; Kussmaul, 2012; Lokkila et al., 2016; Missiroli et al., 2016; Nuutila et al., 2005) and build upon recent work on collaborative problem solving (Cajander et al., 2012; Cukurova et al., 2016) is suggested as changes in this dimension of the pedagogy of computing teaching may have profound impact on both academic progress and motivation of girls and boys.

5 Summary.

The tension between exploratory (constructivist), making (constructionist) and direct teaching needs to be quickly addressed. Teachers are currently presented with a plethora of educational technology resources that lack pedagogical instruction. Research with university students indicates that targeted pedagogies teaching specific skills such as tracing code and subgoal modelling are essential to successful programming learning. Emerging research with school aged pupils indicates that a blended pedagogy, including guided exploration, targeted tasks and creative open problem solving provides a more effective learning scenario. However, these indications need to be verified in UK school settings in rigorous studies.

Benefits from physical computing are cited, but evidence to justify these claims are very limited. As physical computing often requires funding and there is much interest in this context for learning, there is an urgent need for practical guidance for teachers on what pedagogy should be employed to maximise investment and minimise risk. Similarly, teaching computing through game-making is cited as being highly motivational. However, there is limited and mixed evidence of the long-term progression of pupils when being taught computing in this context. Therefore, research is needed to trial different pedagogical approaches in different game-making programming environments and compare outcomes. The effectiveness of unplugged activities seems to be evidenced by teacher adoption of this approach. However, research seems to counter this confidence with mixed results from (mostly small scale or qualitative) existing studies. Robust research is required to verify teacher adoption. In the same vein, cross-curricular contexts are cited as being an effective context for learning computing. However, research here is very sparse, results mixed and studies often lack in rigour. A notable exception to this is the recent ScratchMaths programme, which could prove to be a template for further research studies.

There are clear opportunities for building upon promising work related to the transition from block to text-based languages. Both in secondary schools at the point of transition and in primary schools for preparation through changes to block based curriculum. Similarly, visualisation tools hold much promise, but research with school aged pupils is needed.

Teachers are already employing a range of student engagement approaches, including pair programming, problem-based learning, digital leaders and apprenticeship. However, whether teachers or learners are getting the most out of these strategies is not clear, nor even what the optimal arrangement might be for those approaches. There is promising research from industry, older learners and from research

communities across the world to build upon. Research of pedagogy of student engagement could be transformative in both informing pupil progress and increasing motivation for girls and boys in computing.

In conclusion, existing research related to computing pedagogy has generally focused on older learners or is not robust, due to small populations, short time frames or methodology. Where investigations have been effective, research has often been associated with longitudinal studies of research teams working with schools to produce curricula materials. In doing this, these communities have designed pedagogical frameworks and tested instructional techniques in situ with teachers. A similar approach is recommended, of long-term, collaborative in-class studies. Contrasting pedagogies need to be evaluated and clear practical guidance on how to teach computing should be created. Teachers need robust pedagogical frameworks built on verified foundational theories, with clearly identified learning models and effective instructional techniques. Differing approaches may be needed for different contexts. However, all approaches must provide for progression for all students and afford flexibility of student engagement. So, that teachers can plan and deliver lessons that are motivational for all students irrespective of their prior experience in computing, interests and gender.

Bibliography

Ackermann, E. (2001). Piaget's constructivism, Papert's constructionism: What's the difference. *Future of learning group publication*, *5*(3), 438.

Aggarwal, A., Gardner-McCune, C., & Touretzky, D. S. (2017). Evaluating the Effect of Using Physical Manipulatives to Foster Computational Thinking in Elementary School. *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 9–14). ACM.

Aharoni, D. (2000). Cogito, Ergo sum! cognitive processes of students dealing with data structures. *ACM SIGCSE Bulletin*, *32*(1), 26–30.

Aivaloglou, E., & Hermans, F. (2016). How kids code and how we know: An exploratory study on the scratch repository. *Proceedings of the 2016 ACM Conference on International Computing Education Research* (pp. 53–61). ACM.

Al-Jarrah, A., & Pontelli, E. (2014). AliCe-ViLlagE" Alice as a Collaborative Virtual Learning Environment. *Frontiers in Education Conference (FIE), 2014 IEEE* (pp. 1–9). IEEE.

Alexandron, G., Armoni, M., Gordon, M., & Harel, D. (2014). Scenario-based programming: reducing the cognitive load, fostering abstract thinking. *Companion Proceedings of the 36th International Conference on Software Engineering* (pp. 311–320). ACM.

Armoni, M. (2013). On Teaching Abstraction in Computer Science to Novices. *Journal of Computers in Mathematics and Science Teaching*, *32*(3), 265–284.

Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (2015). From Scratch to "real" programming. *ACM Transactions on Computing Education (TOCE)*, *14*(4), 25.

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

Bell, T., Alexander, J., Freeman, I., & Grimley, M. (2009). Computer science unplugged: School students doing real computing without computers. *The New Zealand Journal of Applied Computing and Information Technology*, *13*(1), 20–29.

Bell, T., & Newton, H. (n.d.). USING COMPUTER SCIENCE UNPLUGGED AS A TEACHING TOOL. *Accessed Online [5th Oct. 2014] http://nzacditt. org. nz/system/files/Bell,% 20Newton*.

Ben-Ari, M. (1998). Constructivism in computer science education. *ACM SIGCSE bulletin* (Vol. 30, pp. 257–261). ACM.

Benitti, F. B. V. (2012). Exploring the educational potential of robotics in schools: A systematic review. *Computers & Education*, *58*(3), 978–988.

Benton, L., Hoyles, C., & Noss, I. K. anRichard. (2016). Building mathematical knowledge with programming: insights from the ScratchMaths project. Constructionism.

Benton, L., Hoyles, C., & Noss, I. K. anRichard. (2017). Bridging Primary Programming and Mathematics: some findings of design research in England. *Digital Experiences in Mathematics Education*.

Bergin, J., Duvall, R. C., Mercer, R., Wallingford, E., Gabriel, R. P., West, D., & Rostal, P. M. (n.d.). A Snapshot of Studio Based Learning.

Berry. (2015a). QuickStart Primary Handbook. Swindon.

Berry. (2015b). QuickStart Computing Primary Handbook. Retrieved from http://primary.quickstartcomputing.org/resources/pdf/qs_handbook.pdf

Berry, M., & Kölling, M. (2013). The design and implementation of a notional machine for teaching introductory programming. *Proceedings of the 8th Workshop in Primary and Secondary Computing Education* (pp. 25–28). ACM.

Berry, Woollard, J., Hughes, P., Chippendal, J., Ross, Z., & Waite, J. (2015). Barefoot computing resoruces. Retrieved from http://barefootcas.org.uk/

Bers, M. (2010). The TangibleK Robotics program: Applied computational thinking for young children. *Early Childhood Research & Practice*, *12*(2), n2.

Bers, M., Flannery, L., Kazakoff, E. R., & Sullivan, A. (2014). Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. *Computers & Education*, *72*, 145–157.

Brennan, K., & Resnick, M. (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*.

Brinkmeier, M., & Kalbreyer, D. (2016). A Case Study of Physical Computing in Computer Science Education. *Proceedings of the 11th Workshop in Primary and Secondary Computing Education* (pp. 54–59). ACM.

Broll, B., Lédeczi, A., Volgyesi, P., Sallai, J., Maroti, M., Carrillo, A., Weeden-Wright, S. L., et al. (2017). A Visual Programming Environment for Learning Distributed Programming. *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 81–86). ACM.

Buchholz, M., Saeli, M., & Schulte, C. (2013). PCK and reflection in computer science teacher education. *Proceedings of the 8th workshop in primary and secondary computing education* (pp. 8–16). ACM.

Busjahn, T., Bednarik, R., Begel, A., Crosby, M., Paterson, J. H., Schulte, C., Sharif, B., et al. (2015). Eye movements in code reading: Relaxing the linear order. *Program Comprehension (ICPC), 2015 IEEE 23rd International Conference on* (pp. 255–265). IEEE.

Busjahn, T., & Schulte, C. (2013). The use of code reading in teaching programming. *Proceedings of the 13th Koli Calling International Conference on Computing Education Research* (pp. 3–11). ACM.

Cajander, Å., Daniels, M., & McDermott, R. (2012). On valuing peers: theories of learning and intercultural competence. *Computer Science Education*, *22*(4), 319–342.

Ching, C. C., & Kafai, Y. B. (2008). Peer pedagogy: Student collaboration and reflection in a learning-through-design project. *Teachers College Record*, *110*(12), 2601–2632.

Clear, T. (2012). The hermeneutics of program comprehension: a 'holey quilt' theory. *ACM Inroads*, *3*(2), 6–7.

Code.org. (2016). Abstraction. Retrieved from https://studio.code.org/s/20-hour/stage/14/puzzle/1

Collins, A., & others. (1987). Cognitive Apprenticeship: Teaching the Craft of Reading, Writing, and Mathematics. Technical Report No. 403.

Colubri, A., & Fry, B. (2012). Introducing Processing 2.0. *ACM SIGGRAPH 2012 Talks* (p. 12). ACM.

Corney, M., Teague, D., Ahadi, A., & Lister, R. (2012). Some empirical results for neo-Piagetian reasoning in novice programmers and the relationship to code explanation questions. *Proceedings of the Fourteenth Australasian Computing Education Conference-Volume 123* (pp. 77–86). Australian Computer Society, Inc.

Crick. (2017, April). Royal Society Computing Education Project Review of Literature: Literature on effective computing pedagogy.

Crouch, C. H., & Mazur, E. (2001). Peer instruction: Ten years of experience and results. *American journal of physics*, *69*(9), 970–977.

CSTA. (2011a). Computational Thinking Teacher Resources 2nd Edition. Retrieved from https://www.iste.org/explore/articleDetail?articleid=152&category=Solutions&article=Computational-thinking-for-all

CSTA. (2011b). K-12 Computer Science Standards Revised 2011. Retrieved from https://csta.acm.org/Curriculum/sub/K12Standards.html

Cukurova, M., Avramides, K., Spikol, D., Luckin, R., & Mavrikis, M. (2016). An analysis framework for collaborative problem solving in practice-based learning activities: A mixed-method approach. *Proceedings of the Sixth International Conference on Learning Analytics & Knowledge* (pp. 84–88). ACM.

Curzon, P. (2013). cs4fn and computational thinking unplugged. *Proceedings of the 8th Workshop in Primary and Secondary Computing Education* (pp. 47–50). ACM.

Curzon, P., McOwan, P. W., Cutts, Q. I., & Bell, T. (2009). Enthusing & inspiring with reusable kinaesthetic activities. *ACM SIGCSE Bulletin* (Vol. 41, pp. 94–98). ACM.

Curzon, P., McOwan, P. W., Plant, N., & Meagher, L. R. (2014). Introducing teachers to computational thinking using unplugged storytelling. *Proceedings of the 9th Workshop in Primary and Secondary Computing Education* (pp. 89–92). ACM. doi:10.1145/2670757.2670767

Cutts, Connor, R., Michaelson, G., & Donaldson, P. (2014). Code or (not code): separating formal and natural language in CS education. *Proceedings of the 9th Workshop in Primary and Secondary Computing Education* (pp. 20–28). ACM.

Cutts, Esper, S., Fecho, M., Foster, S., & Simon, B. (2012). The abstraction transition taxonomy: developing desired learning outcomes through the lens of situated cognition. *Proceedings of the ninth annual international conference on International computing education research* (pp. 63–70). ACM.

Cutts, Q. I., Brown, M. I., Kemp, L., & Matheson, C. (2007). Enthusing and informing potential computer science students and their teachers. *ACM SIGCSE Bulletin* (Vol. 39, pp. 196–200). ACM.

Dann, W., Cosgrove, D., Slater, D., Culyba, D., & Cooper, S. (2012). Mediated transfer: Alice 3 to java. *Proceedings of the 43rd ACM technical symposium on Computer Science Education* (pp. 141–146). ACM.

DeLyser, L. A., Mascio, B., & Finkel, K. (2016). Introducing Student Assessments with Evidence of Validity for NYC's CS4All. *Proceedings of the 11th Workshop in Primary and Secondary Computing Education* (pp. 17–26). ACM.

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

DesPortes, K., Anupam, A., Pathak, N., & DiSalvo, B. (2016). BitBlox: A Redesign of the Breadboard. *Proceedings of the The 15th International Conference on Interaction Design and Children* (pp. 255–261). ACM.

DfE. (1999). The national curriculum for England ICT. Retrieved from http://webarchive.nationalarchives.gov.uk/20100202100434/http://curriculum.qcda.gov.uk/key-stages-1-and-2/subjects/ict/keystage1/index.aspx

DfE Computing programmes of study: key stages 3 and 4 National curriculum in England (2013). Department for Education. Retrieved from https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/239067/SECONDARY_national_curriculum_-_Computing.pdf

DfE Computing programmes of study key stages 1 and 2 National Curriculum in England (2013). Department of Education. Retrieved from https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study

Digital School House. (2016). Digital School House, range of unplugged lessons. {Digital School House}. Retrieved from http://archive.digitalschoolhouse.org.uk/data/227-toptrumps

Dorling, M., & White, D. (2015). Scratch: A way to logo and python. *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (pp. 191–196). ACM.

Du Boulay, B. (1986). Some difficulties of learning to program. *Journal of Educational Computing Research*, *2*(1), 57–73.

Dwyer, H., Hill, C., Carpenter, S., Harlow, D., & Franklin, D. (2014). Identifying elementary students' pre-instructional ability to develop algorithms and step-by-step instructions. *Proceedings of the 45th ACM technical symposium on Computer science education* (pp. 511–516). ACM.

Dwyer, H., Hill, C., Hansen, A., Iveland, A., Franklin, D., & Harlow, D. (2015). Fourth Grade Students Reading Block-Based Programs: Predictions, Visual Cues, and Affordances. *Proceedings of the eleventh annual International Conference on International Computing Education Research* (pp. 111–119). ACM.

Eickholt, J., & Shrestha, S. (2017). Teaching Big Data and Cloud Computing with a Physical Cluster. *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 177–181). ACM.

Falkner, K., & Vivian, R. (2015). A review of computer science resources for learning and teaching with K-12 computing curricula: An Australian case study. *Computer Science Education*, *25*(4), 390–429.

Falkner, K., Vivian, R., & Falkner, N. (2015). Teaching Computational Thinking in K-6: The CSER Digital Technologies MOOC. *Proceedings of the 17th Australasian Computing Education Conference (ACE 2015)* (Vol. 27, p. 30).

Feaster, Y., Segars, L., Wahba, S. K., & Hallstrom, J. O. (2011). Teaching CS Unplugged in the High School (with Limited Success). *Proceedings of the 16th Annual Joint Conference on Innovation and*

Technology in Computer Science Education, ITiCSE '11 (pp. 248–252). Darmstadt, Germany: ACM. doi:10.1145/1999747.1999817

Ford, V., Siraj, A., Haynes, A., & Brown, E. (2017). Capture the Flag Unplugged: an Offline Cyber Competition. *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 225–230). ACM.

Franklin, D., Hill, C., Dwyer, H. A., Hansen, A. K., Iveland, A., & Harlow, D. B. (2016). Initialization in Scratch: Seeking Knowledge Transfer. *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (pp. 217–222). ACM.

Franklin, D., Skifstad, G., Rolock, R., Mehrotra, I., Ding, V., Hansen, A., Weintrop, D., et al. (2017). Using Upper-Elementary Student Performance to Understand Conceptual Sequencing in a Blocks-based Curriculum. *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 231–236). ACM.

Gal-Ezer, J., & Zur, E. (2004). The efficiency of algorithms—misconceptions. *Computers & Education*, *42*(3), 215–226.

Garlick, R., & Cankaya, E. C. (2010). Using alice in CS1: a quantitative experiment. *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education* (pp. 165–168). ACM.

Garneli, V., Giannakos, M. N., & Chorianopoulos, K. (2015). Computing education in K-12 schools: A review of the literature. *Global Engineering Education Conference (EDUCON), 2015 IEEE* (pp. 543–551). IEEE.

Garneli, V., Giannakos, M. N., Chorianopoulos, K., & Jaccheri, L. (2015). Serious game development as a creative learning experience: lessons learnt. *Proceedings of the Fourth International Workshop on Games and Software Engineering* (pp. 36–42). IEEE Press.

Good, J. (2011). Learners at the wheel: Novice programming environments come of age. *International Journal of People-Oriented Programming (IJPOP)*, *1*(1), 1–24.

Google. (2016). Exploring Computational Thinking. Retrieved from www.google.com/edu/computational-thinking

Gough, D. A., Sandy, O., & James, T. (2013). *Learning from research: systematic reviews for informing policy decisions: a quick guide*. Nesta London, UK.

Greening, T. (2000). Emerging Constructivist Forces in Computer Science Education: Shaping a New Future? In T. Greening (Ed.), *Computer Science Education in the 21st Century* (pp. 47–80). New York, NY: Springer New York. doi:10.1007/978-1-4612-1298-0_5

Griffin, J., Pirmann, T., & Gray, B. (2016). Two Teachers, Two Perspectives on CS Principles. *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (pp. 461–466). ACM.

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

Grover, S., & Basu, S. (2017). Measuring Student Learning in Introductory Block-Based Programming: Examining Misconceptions of Loops, Variables, and Boolean Logic. *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 267–272). ACM.

Grover, S., & Pea, R. (2013a). Using a discourse-intensive pedagogy and android's app inventor for introducing computational concepts to middle school students. *Proceeding of the 44th ACM technical symposium on Computer science education* (pp. 723–728). ACM.

Grover, S., & Pea, R. (2013b). Computational Thinking in K–12 A Review of the State of the Field. *Educational Researcher*, *42*(1), 38–43. doi:10.3102/0013189X12463051

Gujberova, M., & Kalas, I. (2013). Designing productive gradations of tasks in primary programming education. *Proceedings of the 8th Workshop in Primary and Secondary Computing Education* (pp. 108–117). ACM.

Hamer, J., Cutts, Q., Jackova, J., Luxton-Reilly, A., McCartney, R., Purchase, H., Riedesel, C., et al. (2008). Contributing student pedagogy. *ACM SIGCSE Bulletin*, *40*(4), 194–212.

Hanks, B., Fitzgerald, S., McCauley, R., Murphy, L., & Zander, C. (2011). Pair programming in education: a literature review. *Computer Science Education*, *21*(2), 135–173.

Hansen, A., Hansen, E., Dwyer, H., Harlow, D., & Franklin, D. (2016). Differentiating for Diversity: Using Universal Design for Learning in Elementary Computer Science Education. *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (pp. 376–381). ACM.

Hansen, A., Iveland, A., Carlin, C., Harlow, D. B., & Franklin, D. (2016). User-Centered Design in Block-Based Programming: Developmental & Pedagogical Considerations for Children. *Proceedings of the The 15th International Conference on Interaction Design and Children* (pp. 147–156). ACM.

Hazzan, O. (2003). How students attempt to reduce abstraction in the learning of mathematics and in the learning of computer science. *Computer Science Education*, *13*(2), 95–122.

Horn, M. S., Crouser, R. J., & Bers, M. U. (2012). Tangible interaction and learning: the case for a hybrid approach. *Personal and Ubiquitous Computing*, *16*(4), 379–389.

Hubwieser, P., Armoni, M., Giannakos, M. N., & Mittermeir, R. T. (2014). Perspectives and visions of computer science education in primary and secondary (K-12) schools. *ACM Transactions on Computing Education (TOCE)*, *14*(2), 7.

Jin, K. H., Haynie, K., & Kearns, G. (2016). Teaching Elementary Students Programming in a Physical Computing Classroom. *Proceedings of the 17th Annual Conference on Information Technology Education* (pp. 85–90). ACM.

Kafai, Y. B., & Burke, Q. (2013). The social turn in K-12 programming: moving from computational thinking to computational participation. *Proceeding of the 44th ACM technical symposium on computer science education* (pp. 603–608). ACM.

Kafai, Y. B., & Burke, Q. (2015). Constructionist gaming: Understanding the benefits of making games for learning. *Educational psychologist*, *50*(4), 313–334.

Kafai, Y. B., Lee, E., Searle, K., Fields, D., Kaplan, E., & Lui, D. (2014). A crafts-oriented approach to computing in high school: Introducing computational concepts, practices, and perspectives with electronic textiles. *ACM Transactions on Computing Education (TOCE)*, *14*(1), 1.

Kafai, Y. B., & Vasudevan, V. (2015). Constructionist Gaming Beyond the Screen: Middle School Students' Crafting and Computing of Touchpads, Board Games, and Controllers. *Proceedings of the Workshop in Primary and Secondary Computing Education* (pp. 49–54). ACM.

Kaila, E., Laakso, M.-J., Rajala, T., & Salakoski, T. (2009). Evaluation of Learner Engagement in Program Visualization. *12th IASTED International Conference on Computers and Advanced Technology in Education (CATE 2009)*.

Kaila, E., Rajala, T., Laakso, M.-J., & Salakoski, T. (2010). Effects of course-long use of a program visualization tool. *Proceedings of the Twelfth Australasian Conference on Computing Education-Volume 103* (pp. 97–106). Australian Computer Society, Inc.

Kastl, P., Kiesmüller, U., & Romeike, R. (2016). Starting out with Projects: Experiences with Agile Software Development in High Schools. *Proceedings of the 11th Workshop in Primary and Secondary Computing Education* (pp. 60–65). ACM.

Kastl, P., & Romeike, R. (2015). Now they just start working, and organize themselves First Results of Introducing Agile Practices in Lessons. *Proceedings of the Workshop in Primary and Secondary Computing Education* (pp. 25–28). ACM.

Kazakoff, E., & Bers, M. (2012). Programming in a robotics context in the kindergarten classroom: The impact on sequencing skills. *Journal of Educational Multimedia and Hypermedia*, *21*(4), 371–391.

Kestenbaum, D. (2005). The Challenges of IDC: What Have We Learned from Our Past? *Commun. ACM*, *48*(1), 35–38. doi:10.1145/1039539.1039566

Kirschner, P. A., Sweller, J., & Clark, R. E. (2006). Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. *Educational psychologist*, *41*(2), 75–86.

Kolikant, Y. B.-D., & Mussai, M. (2008). So my program doesn't run! Definition, origins, and practical expressions of students' (mis) conceptions of correctness. *Computer Science Education*, *18*(2), 135–151.

Kölling, M. (2015). Lessons from the Design of Three Educational Programming Environments: Blue, BlueJ and Greenfoot. *International Journal of People-Oriented Programming (IJPOP)*, *4*(1), 5–32.

Kölling, M., Brown, N. C., & Altadmri, A. (2015). Frame-based editing: Easing the transition from blocks to text-based programming. *Proceedings of the Workshop in Primary and Secondary Computing Education* (pp. 29–38). ACM.

Kussmaul, C. (2012). Process oriented guided inquiry learning (POGIL) for computer science. *Proceedings of the 43rd ACM technical symposium on Computer Science Education* (pp. 373–378). ACM.

Laakso, M.-J., Rajala, T., Kaila, E., & Salakoski, T. (2008). The impact of prior experience in using a visualization tool on learning to program. *Proceeding of Cognition and Exploratory Learning in Digital Age (CELDA 2008)*, 13–15.

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J., et al. (2011). Computational thinking for youth in practice. *ACM Inroads*, *2*(1), 32–37.

Lewis, C. M. (2011). Is pair programming more effective than other forms of collaboration for young students? *Computer Science Education*, *21*(2), 105–134.

Lister, R. (2011). Concrete and other neo-Piagetian forms of reasoning in the novice programmer. *Proceedings of the Thirteenth Australasian Computing Education Conference-Volume 114* (pp. 9–18). Australian Computer Society, Inc.

Lister, R. (2016). Toward a Developmental Epistemology of Computer Programming. *Proceedings of the 11th Workshop in Primary and Secondary Computing Education* (pp. 5–16). ACM.

Lister, R., Fidge, C., & Teague, D. (2009). Further evidence of a relationship between explaining, tracing and writing skills in introductory programming. *ACM SIGCSE Bulletin* (Vol. 41, pp. 161–165). ACM.

Litts, B. K., Kafai, Y. B., Lui, D., Walker, J., & Widman, S. (2017). Understanding High School Students' Reading, Remixing, and Writing Codeable Circuits for Electronic Textiles. *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 381–386). ACM.

Lokkila, E., Rajala, T., Veerasamy, A., Enges-Pyykönen, P., Laakso, M. J., & Salakoski, T. (2016). How students' programming process differs from experts – a case study with a robot programming exercise. *EDULEARN16 Proceedings*, 8th International Conference on Education and New Learning Technologies (pp. 1555–1562). Barcelona, Spain: IATED. doi:10.21125/edulearn.2016.1308

Lopez, M., Whalley, J., Robbins, P., & Lister, R. (2008). Relationships between reading, tracing and writing skills in introductory programming. *Proceedings of the fourth international workshop on computing education research* (pp. 101–112). ACM.

Lourenço, O. (2012). Piaget and Vygotsky: Many resemblances, and a crucial difference. *New Ideas in Psychology*, *30*(3), 281–295.

Lukkarinen, A., & Sorva, J. (2016). Classifying the tools of contextualized programming education and forms of media computation. *Proceedings of the 16th Koli Calling International Conference on Computing Education Research* (pp. 51–60). ACM.

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

Major, L., Kyriacou, T., & Brereton, O. P. (2012). Systematic literature review: Teaching novices programming using robots. *IET software*, *6*(6), 502–513.

Margulieux, L. E., & Catrambone, R. (2016). Improving problem solving with subgoal labels in expository text and worked examples. *Learning and Instruction*, *42*, 58–71.

Mayer, R. E. (2004). Should there be a three-strikes rule against pure discovery learning? *American psychologist*, *59*(1), 14.

McDonald, S., & Howell, J. (2012). Watching, creating and achieving: Creative technologies as a conduit for learning in the early years. *British journal of educational technology*, *43*(4), 641–651.

McDowell, C., Werner, L., Bullock, H. E., & Fernald, J. (2006). Pair programming improves student retention, confidence, and program quality. *Communications of the ACM*, *49*(8), 90–95.

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2011). Habits of programming in Scratch. *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education* (pp. 168–172). ACM.

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

Menekse, M. (2015). Computer science teacher professional development in the United States: a review of studies published between 2004 and 2014. *Computer Science Education*, 1–26.

Michaelson, G. (2015). Teaching Programming with Computational and Informational Thinking. *Journal of Pedagogic Development*, *5*(1).

Missiroli, M., Russo, D., & Ciancarini, P. (2016). Learning Agile software development in high school: an investigation. *Proceedings of the 38th International Conference on Software Engineering Companion* (pp. 293–302). ACM.

Moreno-León, J., & Robles, G. (2016). Code to learn with Scratch? A systematic literature review. *Global Engineering Education Conference (EDUCON), 2016* (pp. 150–156). IEEE.

Morrison, B. B., Margulieux, L. E., Ericson, B., & Guzdial, M. (2016). Subgoals help students solve Parsons problems. *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (pp. 42–47). ACM.

Nuutila, E., Törmä, S., & Malmi, L. (2005). PBL and computer programming—the seven steps method with adaptations. *Computer Science Education*, *15*(2), 123–142.

Paas, F., Renkl, A., & Sweller, J. (2003). Cognitive load theory and instructional design: Recent developments. *Educational psychologist*, *38*(1), 1–4.

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

Papert, S., & Harel, I. (1991). Situating constructionism. *Constructionism*, *36*(2), 1–11.

Parrish, A., Fry, B., & Reas, C. (2016). *Getting Started with Processing.Py: Making Interactive Graphics with Python's Processing Mode* (1st ed.). USA: Maker Media, Inc.

Passey, D. (2014). Intergenerational learning practices—Digital leaders in schools. *Education and Information Technologies*, *19*(3), 473–494.

Pea, R. D. (2004). The social and technological dimensions of scaffolding and related theoretical concepts for learning, education, and human activity. *The journal of the learning sciences*, *13*(3), 423–451.

Perrenet, J., & Kaasenbrood, E. (2006). Levels of abstraction in students' understanding of the concept of algorithm: the qualitative perspective. *ACM SIGCSE Bulletin*, *38*(3), 270–274.

Piaget, J. (1951). *The Psychology of Intelligence*. (K. Paul, Ed.). Routledge.

Plonka, L., Segal, J., Sharp, H., & Linden, J. van der. (2011). Collaboration in pair programming: driving and switching. *International Conference on Agile Software Development* (pp. 43–59). Springer.

Porter, L., Bouvier, D., Cutts, Q., Grissom, S., Lee, C., McCartney, R., Zingaro, D., et al. (2016). A multi-institutional study of peer instruction in introductory computing. *ACM Inroads*, *7*(2), 76–81.

Powers, K., Ecott, S., & Hirshfield, L. M. (2007). Through the looking glass: teaching CS0 with Alice. *ACM SIGCSE Bulletin*, *39*(1), 213–217.

Price, T. W., & Barnes, T. (2015). Comparing textual and block interfaces in a novice programming environment. *Proceedings of the eleventh annual International Conference on International Computing Education Research* (pp. 91–99). ACM.

Price, T. W., Brown, N. C., Lipovac, D., Barnes, T., & Kölling, M. (2016). Evaluation of a Frame-based Programming Editor. *Proceedings of the 2016 ACM Conference on International Computing Education Research* (pp. 33–42). ACM.

Price, T. W., Dong, Y., & Lipovac, D. (2017). iSnap: Towards Intelligent Tutoring in Novice Programming Environments. *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 483–488). ACM.

Przybylla, M. (2016). Situating Physical Computing in Secondary CS Education. *Proceedings of the 2016 ACM Conference on International Computing Education Research* (pp. 287–288). ACM.

Przybylla, M., & Romeike, R. (2014). Physical computing in computer science education. *Proceedings of the 9th Workshop in Primary and Secondary Computing Education* (pp. 136–137). ACM.

Rahimi, E., Barendsen, E., & Henze, I. (2016). Typifying Informatics Teachers' PCK of Designing Digital Artefacts in Dutch Upper Secondary Education. *International Conference on Informatics in Schools: Situation, Evolution, and Perspectives* (pp. 65–77). Springer.

Rajala, T., Kaila, E., Laakso, M.-J., & Salakoski, T. (2009). Effects of Collaboration in Program Visualization. *Proceedings of 2009 Technology Enhanced Learning Conference (TELearn 2009)*.

Rajala, T., Laakso, M.-J., Kaila, E., & Salakoski, T. (2008). Effectiveness of Program Visualization: A Case Study with the ViLLE Tool. *Journal of Information Technology Education*, *7*, 15–32.

Rajala, T., Salakoski, T., Laakso, M.-J., Kaila, E., & others. (2009). Effects, experiences and feedback from studies of a program visualization tool. *Informatics in Education-An International Journal*, (8 /1), 17–34.

Repenning, A., Webb, D. C., Koh, K. H., Nickerson, H., Miller, S. B., Brand, C., Basawapatna, A., et al. (2015). Scalable Game Design: A strategy to bring systemic Computer Science Education to schools through game design and simulation creation. *ACM Transactions on Computing Education (TOCE)*, *15*(2), 11.

Resnick, M. (2013). Learn to code, code to learn. *EdSurge, May*.

Rich, K., Strickland, C., & Franklin, D. (2017). A Literature Review through the Lens of Computer Science Learning Goals Theorized and Explored in Research. *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 495–500). ACM.

Rodriguez, B., Kennicutt, S., Rader, C., & Camp, T. (2017). Assessing Computational Thinking in CS Unplugged Activities. *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 501–506). ACM.

Rolandsson, L. (2012). *Changing Computer Programming Education; The Dinosaur that Survived in School: An explorative study of educational issues based on teachers' beliefs and curriculum development in secondary school*. KTH Royal Institute of Technology.

Rubin, M. J. (2013). The effectiveness of live-coding to teach introductory programming. *Proceeding of the 44th ACM technical symposium on Computer science education* (pp. 651–656). ACM.

Ruvalcaba, O., Werner, L., & Denner, J. (2016). Observations of Pair Programming: Variations in Collaboration Across Demographic Groups. *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (pp. 90–95). ACM.

Salleh, N., Mendes, E., & Grundy, J. (2011). Empirical studies of pair programming for CS/SE teaching in higher education: A systematic literature review. *IEEE Transactions on Software Engineering*, *37*(4), 509–525.

Schulte, C. (2008). Block Model: an educational model of program comprehension as a tool for a scholarly approach to teaching. *Proceedings of the Fourth international Workshop on Computing Education Research* (pp. 149–160). ACM.

Schulte, C., Clear, T., Taherkhani, A., Busjahn, T., & Paterson, J. H. (2010). An introduction to program comprehension for computer science educators. *Proceedings of the 2010 ITiCSE working group reports* (pp. 65–86). ACM.

Seiter, L., & Foreman, B. (2013). Modeling the learning progressions of computational thinking of primary grade students. *Proceedings of the ninth annual international ACM conference on International computing education research* (pp. 59–66). ACM.

Selby, C., & Woollard, J. (2014). *Refining an understanding of computational thinking*. Retrieved from http://eprints.soton.ac.uk/372410/1/372410UnderstdCT.pdf

Sentance, S. (2015, May). Computing At School Annual Survey 2015. http://community.computingatschool.org.uk/files/6098/original.pdf.

Sentance, S. (2016). Computing At School Annual Survey 2016. Retrieved from http://community.computingatschool.org.uk/files/8106/original.pdf

Sentance, S., & Csizmadia, A. (2015). Teachers' perspectives on successful strategies for teaching computing in school. *IFIP TC3 Working Conference*, *2015*.

Sentance, S., & Csizmadia, A. (2016). Computing in the curriculum: Challenges and strategies from a teacher's perspective. *Education and Information Technologies*, 1–27.

Sentance, S., & Schwiderski-Grosche, S. (2012). Challenge and creativity: using. NET gadgeteer in schools. *Proceedings of the 7th Workshop in Primary and Secondary Computing Education* (pp. 90–100). ACM.

Sentance, S., Waite, J., Hodges, S., MacLeod, E., & Yeomans, L. (2017). Creating Cool Stuff: Pupils' Experience of the BBC micro: bit. *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 531–536). ACM.

Solomon, C. (1986). Papert: Constructivism and Piagetian Learning.

Sorva, J. (2013). Notional machines and introductory programming education. *ACM Transactions on Computing Education (TOCE)*, *13*(2), 8.

Sorva, J., Karavirta, V., & Malmi, L. (2013). A review of generic program visualization systems for introductory programming education. *ACM Transactions on Computing Education (TOCE)*, *13*(4), 15.

Statter, D., & Armoni, M. (2016). Teaching Abstract Thinking in Introduction to Computer Science for 7th Graders. *Proceedings of the 11th Workshop in Primary and Secondary Computing Education* (pp. 80–83). ACM.

Strawhacker, A., & Bers, M. U. (2015). I want my robot to look for food Comparing Kindergartner's programming comprehension using tangible, graphic, and hybrid user interfaces. *International Journal of Technology and Design Education*, *25*(3), 293–319.

Su, A., Yang, S. J., Hwang, W.-Y., Huang, C. S., & Tern, M.-Y. (2014). Investigating the role of computer-supported annotation in problem-solving-based teaching: An empirical study of a Scratch programming pedagogy. *British Journal of Educational Technology*, *45*(4), 647–665.

Sudol-DeLyser, L. A., Stehlik, M., & Carver, S. (2012). Code comprehension problems as learning events. *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education* (pp. 81–86). ACM.

Sweller, J., Kirschner, P. A., & Clark, R. E. (2007). Why minimally guided teaching techniques do not work: A reply to commentaries. *Educational Psychologist*, *42*(2), 115–121.

Taub, R., Armoni, M., & Ben-Ari, M. (2012). CS unplugged and middle-school students' views, attitudes, and intentions regarding CS. *ACM Transactions on Computing Education (TOCE)*, *12*(2), 8.

Taub, R., Armoni, M., & Ben-Ari, M. M. (2014). Abstraction as a bridging concept between computer science and physics. *Proceedings of the 9th Workshop in Primary and Secondary Computing Education* (pp. 16–19). ACM. doi:10.1145/2670757.2670777

Teague, D., & Lister, R. (2014a). Longitudinal think aloud study of a novice programmer. *Proceedings of the Sixteenth Australasian Computing Education Conference-Volume 148* (pp. 41–50). Australian Computer Society, Inc.

Teague, D., & Lister, R. (2014b). Manifestations of preoperational reasoning on similar programming tasks. *Proceedings of the Sixteenth Australasian Computing Education Conference-Volume 148* (pp. 65–74). Australian Computer Society, Inc.

Teague, D., & Lister, R. (2014c). Programming: reading, writing and reversing. *Proceedings of the 2014 conference on Innovation & technology in computer science education* (pp. 285–290). ACM.

Tedre, M., & Denning, P. J. (2016). The long quest for computational thinking. *Proceedings of the 16th Koli Calling Conference on Computing Education Research* (pp. 24–27).

Thies, R., & Vahrenhold, J. (2012). Reflections on outreach programs in CS classes: learning objectives for unplugged activities. *Proceedings of the 43rd ACM technical symposium on Computer Science Education* (pp. 487–492). ACM.

Thies, R., & Vahrenhold, J. (2016). Back to School: Computer Science Unplugged in the Wild. *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 118–123). ACM.

Toh, L. P. E., Causo, A., Tzuo, P. W., Chen, I.-M., Yeo, S. H., & others. (2016). A Review on the Use of Robots in Education and Young Children. *Educational Technology & Society*, *19*(2), 148–163.

Tsai, C.-Y., Yang, Y.-F., & Chang, C.-K. (2015). Cognitive Load Comparison of Traditional and Distributed Pair Programming on Visual Programming Language. *Proceedings of the International Conference of Educational Innovation through Technology (EITT),* (pp. 143–146). IEEE.

Van Merrienboer, J. J., & Sweller, J. (2005). Cognitive load theory and complex learning: Recent developments and future directions. *Educational psychology review*, *17*(2), 147–177.

Veerasamy, A. K., D'Souza, D., & Laakso, M.-J. (2016). Identifying Novice Student Programming Misconceptions and Errors From Summative Assessments. *Journal of Educational Technology Systems*, *45*(1), 50–73.

Venables, A., Tan, G., & Lister, R. (2009). A closer look at tracing, explaining and code writing skills in the novice programmer. *Proceedings of the fifth international workshop on Computing education research workshop* (pp. 117–128). ACM.

Waite, J., Curzon, P., Marsh, W., & Sentance, S. (2016). Abstraction and common classroom activities. *Proceedings of the 11th Workshop in Primary and Secondary Computing Education* (pp. 112–113). ACM.

Webb, D. C., Repenning, A., & Koh, K. H. (2012). Toward an emergent theory of broadening participation in computer science education. *Proceedings of the 43rd ACM technical symposium on Computer Science Education* (pp. 173–178). ACM.

Weintrop, D., & Holbert, N. (2017). From Blocks to Text and Back: Programming Patterns in a Dual-Modality Environment. *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 633–638). ACM.

Weintrop, D., Holbert, N., Horn, M. S., & Wilensky, U. (2016). Computational thinking in constructionist video games. *International Journal of Game-Based Learning (IJGBL)*, *6*(1), 1–17.

Weintrop, D., & Wilensky, U. (2015). To block or not to block, that is the question: students' perceptions of blocks-based programming. *Proceedings of the 14th International Conference on Interaction Design and Children* (pp. 199–208). ACM.

Werner, L., Denner, J., & Campe, S. (2015). Children programming games: a strategy for measuring computational learning. *ACM Transactions on Computing Education (TOCE)*, *14*(4), 24.

Werner, L., Denner, J., Campe, S., Ortiz, E., DeLay, D., Hartl, A., & Laursen, B. (2013). Pair programming for middle school students: does friendship influence academic outcomes? *Proceeding of the 44th ACM technical symposium on Computer science education* (pp. 421–426). ACM.

Williams, L. A., & Kessler, R. R. (2000). All I really need to know about pair programming I learned in kindergarten. *Communications of the ACM*, *43*(5), 108–114.

Wing, J. M. (2011). Computational thinking. *VL/HCC* (p. 3).

Wu, C.-C., Tseng, I.-C., & Huang, S.-L. (2008). Visualization of program behaviors: Physical robots versus robot simulators. *International Conference on Informatics in Secondary Schools-Evolution and Perspectives* (pp. 53–62). Springer.

Yadav, A., Gretter, S., Hambrusch, S., & Sands, P. (2016). Expanding computer science education in schools: understanding teacher experiences and challenges. *Computer Science Education*, 1–20.

Appendix A

Research studies per theme are shown in the tables below. Papers are ordered in the same order as they are referenced in the report.

Concepts and techniques

Sub heading	Phase	Articles included	Country	Study Type	Study Size	Study context	
Literature reviews	Primary and Secondary	Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? Computers in Human Behavior, 41, 51–61.	USA	Literature review	27 papers reviewed	Computational thinking through programming	1
		Falkner, K., & Vivian, R. (2015). A review of computer science resources for learning and teaching with K-12 computing curricula: An Australian case study. Computer Science Education, 25(4), 390–429	Australia	Systematic Resource Review	65 resources	Curriculum resources	2
		Garneli, V., Giannakos, M. N., & Chorianopoulos, K. (2015). Computing education in K-12 schools: A review of the literature. Global Engineering Education Conference (EDUCON), 2015 IEEE (pp. 543–551). IEEE.	Greece	Systematic literature review	47 papers	Computing Education	3
		Rich, K., Strickland, C., & Franklin, D. (2017). A Literature Review through the Lens of Computer Science Learning Goals Theorized and Explored in Research. Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (pp. 495–500). ACM.	USA	Literature review		Raising questions paper	4
LOA & discourse	KS3 13/14 years' old	Statter, D., & Armoni, M. (2016). Teaching Abstract Thinking in Introduction to Computer Science for 7th Graders. Proceedings of the 11th Workshop in Primary and Secondary Computing Education (pp. 80–83). ACM	Israel	Mixed	Medium 129		5
	University	Cutts, Q., Esper, S., Fecho, M., Foster, S. R., & Simon, B. (2012). The abstraction transition taxonomy: developing desired learning outcomes through the lens of situated cognition. Proceedings of the 9th Annual International Conference on International Computing Education Research (pp. 63–70). ACM.	UK	Post hoc analysis	133 Peer instruction questions		6
	KS3	Grover, S. & Pea, R., 2013. Using a discourse-intensive pedagogy and android's app inventor for introducing computational concepts to middle school students. In Proceeding of the 44th ACM technical symposium on Computer science education. ACM, pp. 723–728.	USA	Mixed	Small 7 students		7

Curriculum developed by research community	KS3	Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2013). Learning computer science concepts with Scratch. Computer Science Education, 23(3), 239–264.	Israel	Mixed	Medium 108 + 40		8
	KS2/3 (grades 4-6)	Hansen, A., Hansen, E., Dwyer, H., Harlow, D., & Franklin, D. (2016). Differentiating for Diversity: Using Universal Design for Learning in Elementary Computer Science Education. Proceedings of the 47th ACM Technical Symposium on Computing Science Education (pp. 376–381). ACM.	USA	Theory	N/A	Explains UDL	9
		Dwyer, H., Hill, C., Carpenter, S., Harlow, D., & Franklin, D. (2014). Identifying elementary students' pre-instructional ability to develop algorithms and step-by-step instructions. Proceedings of the 45th ACM technical symposium on Computer science education (pp. 511–516). ACM.	USA	Qualitative	Medium 55		10
		Hansen, A.K. et al., 2016. User-Centered Design in Block-Based Programming: Developmental & Pedagogical Considerations for Children. In Proceedings of the 15th International Conference on Interaction Design and Children. ACM, pp. 147–156.	USA	Qualitative	Medium (123 students)		11
		Franklin, D., Hill, C., Dwyer, H. A., Hansen, A. K., Iveland, A., & Harlow, D. B. (2016). Initialization in Scratch: Seeking Knowledge Transfer. Proceedings of the 47th ACM Technical Symposium on Computing Science Education (pp. 217–222). ACM.	USA	Review of field notes analysis of code	Not stated		12
		Franklin, D., Skifstad, G., Rolock, R., Mehrotra, I., Ding, V., Hansen, A., Weintrop, D., et al. (2017). Using Upper-Elementary Student Performance to Understand Conceptual Sequencing in a Blocks-based Curriculum. Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (pp. 231–236). ACM.	USA	Mixed	Medium (123 students)		13
	KS3 11-14	Grover, S., Pea, R. & Cooper, S., 2015. Designing for deeper learning in a blended computer science course for middle school students. Computer Science Education, 25(2), pp.199–237.	USA	Mixed	Medium 54		14
	N/A	Seiter, L., & Foreman, B. (2013). Modeling the learning progressions of computational thinking of primary grade students. *Proceedings of the ninth annual international ACM conference on International computing education research* (pp. 59–66). ACM	USA	Quantitative	N/A 150 programs		15
Guided Discovery	KS3	Grover, S. & Basu, S., 2017. Measuring Student Learning in Introductory Block-Based Programming: Examining Misconceptions of Loops, Variables, and Boolean Logic. In Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education. ACM, pp. 267–272.	USA	Quantitative	medium (100 school students)		16
Extremely Fine	KS4	Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2011). Habits of programming in Scratch. Proceedings of the 16th annual joint conference on Innovation and	Israel	Qualitative	Small 46	Findings arose in a	17

Grained		technology in computer science education (pp. 168–172). ACM.				study. EFPG	
	N/A	Aivaloglou, E., & Hermans, F. (2016). How kids code and how we know: An exploratory study on the Scratch repository. Proceedings of the 2016 ACM Conference on International Computing Education Research (pp. 53–61). ACM.	Holland	Quantitative	250,000 Scratch programs		18
	KS4/5 grades 10-12	Kolikant, Y. B.-D., & Mussai, M. (2008). "So my program doesn't run" Definition, origins, and practical expressions of students' (mis) conceptions of correctness. Computer Science Education, 18(2), 135–151.	Israel	Qualitative	Large (159 students)		19
Tracing & Reading	HE	Teague, D., & Lister, R. (2014c). Programming: reading, writing and reversing. Proceedings of the 2014 conference on Innovation & technology in computer science education (pp. 285–290). ACM	Australia	Qualitative	Small 4		20
	HE	Teague, D., & Lister, R. (2014a). Longitudinal think aloud study of a novice programmer. Proceedings of the Sixteenth Australasian Computing Education Conference-Volume 148 (pp. 41–50). Australian Computer Society, Inc.	Australia	Qualitative	Small 1		21
	HE	Teague, D., & Lister, R. (2014b). Manifestations of preoperational reasoning on similar programming tasks. Proceedings of the Sixteenth Australasian Computing Education Conference-Volume 148 (pp. 65–74). Australian Computer Society, Inc	Australia	Qualitative	Small 11		22
	KS5 grade 10/11	Gal-Ezer, J., & Zur, E. (2004). The efficiency of algorithms—misconceptions. Computers & Education, 42(3), 215–226.	Israel	Quantitative	Large 319		23
	HE	Sudol-DeLyser, L. A., Stehlik, M., & Carver, S. (2012). Code comprehension problems as learning events. Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education (pp. 81–86). ACM.	Finland	Quantitative	Large 345	Code comprehension for learning	24
	KS4 high school	Busjahn, T., & Schulte, C. (2013). The use of code reading in teaching programming. Proceedings of the 13th Koli Calling International Conference on Computing Education Research (pp. 3–11). ACM.	Germany	Qualitative	Small 6 teachers	Teachers interviewed in 3 countries using block model	25
	HE and adult	Busjahn, T., Bednarik, R., Begel, A., Crosby, M., Paterson, J. H., Schulte, C., Sharif, B., et al. (2015). Eye movements in code reading: Relaxing the linear order. Program Comprehension (ICPC), 2015 IEEE 23rd International Conference on (pp. 255–265). IEEE.	Germany	Qualitative	Small 20	eye movements	26
	KS2	Gujberova, M. & Kalas, I., 2013. Designing productive gradations of tasks in primary programming education. In	Slovenia	Quantitative	Medium (128)	Route based aimed at	27

	Age 8 to 10	Proceedings of the 8th Workshop in Primary and Secondary Computing Education. ACM, pp. 108–117				improving Bebras	
	KS2/3 (grades 4-6)	Dwyer, H., Hill, C., Carpenter, S., Harlow, D., & Franklin, D. (2014). Identifying elementary students' pre-instructional ability to develop algorithms and step-by-step instructions. Proceedings of the 45th ACM technical symposium on Computer science education (pp. 511–516). ACM.	USA	Qualitative	Medium 55		28
Subgoal	HE	Margulieux, L. E., & Catrambone, R. (2016). Improving problem-solving with subgoal labels in expository text and worked examples. Learning and Instruction, 42, 58–71.	USA	Quantitative	medium 120		29
	HE	Morrison, B. B., Margulieux, L. E., Ericson, B., & Guzdial, M. (2016). Subgoals help students solve Parsons problems. Proceedings of the 47th ACM Technical Symposium on Computing Science Education (pp. 42–47). ACM.	USA	Quantitative	Medium 119		30
Annotating code and PBL	KS3	Su, A., Yang, S. J., Hwang, W.-Y., Huang, C. S., & Tern, M.-Y. (2014). Investigating the role of computer-supported annotation in problem-solving-based teaching: An empirical study of a Scratch programming pedagogy. British Journal of Educational Technology, 45(4), 647–665.	Taiwan	Quantitative	Medium 135		31
Live coding	HE	Rubin, M. J. (2013). The effectiveness of live-coding to teach introductory programming. Proceeding of the 44th ACM technical symposium on Computer science education (pp. 651–656). ACM.	USA	Mixed	Large 166 students		32
Pseudo code	KS4/5	Cutts, Connor, R., Michaelson, G., & Donaldson, P. (2014). Code or (not code): separating formal and natural language in CS education. Proceedings of the 9th Workshop in Primary and Secondary Computing Education (pp. 20–28). ACM.	UK	N/A	N/A	Review of exam questions	33
Misconceptions	University	Veerasamy, A. K., D'Souza, D., & Laakso, M.-J. (2016). Identifying Novice Student Programming Misconceptions and Errors From Summative Assessments. Journal of Educational Technology Systems, 45(1), 50–73.	Finland	Mixed	Small 39	Review of results from an e-exam	34
		Lokkila, E., Rajala, T., Veerasamy, A., Enges-Pyykönen, P., Laakso, M. J., & Salakoski, T. (2016). How students' programming process differs from experts – a case study with a robot programming exercise. *EDULEARN16 Proceedings*, 8th International Conference on Education and New Learning Technologies (pp. 1555–1562). Barcelona, Spain: IATED.	Finland	Mixed	3 experts 197 students	Compared expert programs to novice	35

Contexts

Theme	Phase	Articles included	Country	Study Type	Study Size	Study context	
Physical	Primary and Secondary	Benitti, F.B.V., 2012. Exploring the educational potential of robotics in schools: A systematic review. Computers & Education, 58(3), pp.978–988.	Brazil	Literature Review	10 papers		1
	Primary Secondary and HE	Major, L., Kyriacou, T. & Brereton, O.P., 2012. Systematic literature review: Teaching novices programming using robots. IET software, 6(6), pp.502–513.	UK	Literature Review	36 papers		2
	Primary and Secondary	Falkner, K., & Vivian, R. (2015). A review of computer science resources for learning and teaching with K-12 computing curricula: An Australian case study. Computer Science Education, 25(4), 390–429	Australia	Systematic Resource Review	65 resources		3
		Toh, L.P.E. et al., 2016. A Review on the Use of Robots in Education and Young Children. Educational Technology & Society, 19(2), pp.148–163.	Singapore	Systematic Literature Review	55 papers		4
	KS1	Kazakoff, E. & Bers, M., 2012. Programming in a robotics context in the kindergarten classroom: The impact on sequencing skills. Journal of Educational Multimedia and Hypermedia, 21(4), pp.371–391.	US	Mixed With control group	Medium 54	2 groups in class time CHERP Lego WeDo	5
		Bers, M.U. et al., 2014. Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. Computers & Education, 72, pp.145–157.	US	Mixed With control group	Medium 53	3 classes in class time CHERP Lego WeDo	6
		Strawhacker, A. & Bers, M.U., 2015. "I want my robot to look for food" Comparing Kindergartner"s programming comprehension using tangible, graphic, and hybrid user interfaces. International Journal of Technology and Design Education, 25(3), pp.293–319.	US	Mixed With control group	Medium 53	3 classes in class time CHERP Lego WeDo	7
		McDonald, S. & Howell, J., 2012. Watching, creating and achieving: Creative technologies as a conduit for learning in the early years. British journal of educational technology, 43(4), pp.641–651.	Australia	Qualitative no control group	Small 16	1 class Lego WeDo	8
	KS2	Jin, K.H., Haynie, K. & Kearns, G., 2016. Teaching Elementary Students Programming in a Physical Computing Classroom. In Proceedings of the 17th Annual Conference on Information Technology Education. ACM, pp. 85–90.	US	Mixed	Small 30 (age 8 to 10)	Summer school Lego Mindstorms	9
	KS3	Kafai, Y.B. & Vasudevan, V., 2015. Constructionist Gaming Beyond the Screen: Middle School Students' Crafting and Computing of Touchpads, Board Games, and	USA	Qualitative Descriptive	Small 28 students	In class but taught by researcher	10

		Controllers. In Proceedings of the Workshop in Primary and Secondary Computing Education ACM, pp. 49–54.				Scratch, Makey Makey;	
		Sentance, S., Waite, J., Hodges, S., MacLeod, E., Yeomans, L., 2017. Creating Cool Stuff: Pupils' Experience of the BBC micro: bit, in: Proceedings 2017 ACM SIGCSE Technical SymposiumComputerScienceEducation. ACM, pp. 531–536.	UK	Qualitative Descriptive	Small 28 students	Focus groups Microbit	11
	17 to 18 High school KS5	Kafai, Y.B., Lee, E., Searle, K., Fields, D., Kaplan, E., Lui, D., 2014. A crafts-oriented approach to computing in high school: Introducing computational concepts, practices, and perspectives with electronic textiles. ACM Transactions on Computing Education (TOCE) 14, 1.	USA	Qualitative	Small 15	Lilypad Arduino, in class	12
		Litts, B.K., Kafai, Y.B., Lui, D., Walker, J., Widman, S., 2017. Understanding High School Students' Reading, Remixing, and Writing Codeable Circuits for Electronic Textiles, in: Proceedings 2017 ACM SIGCSE Technical SymposiumComputerScienceEducation. ACM, pp. 381–386.	USA	Mixed	Small 23	Arduino in class	13
		DesPortes, K., Anupam, A., Pathak, N., DiSalvo, B., 2016. BitBlox: A Redesign of the Breadboard, in: Proceedings The 15th International ConferenceInteractionDesign Children. ACM, pp. 255–261.	USA	Qualitative	Small 44	BitBlox vs Breadboard In class	14
		Brinkmeier, M., Kalbreyer, D., 2016. A Case Study of Physical Computing in Computer Science Education, in: Proceedings 11th WorkshopPrimary Secondary Computing Education. ACM, pp. 54–59.	Germany	Qualitative	Small 25	Abbozza! And Adruino	15
	11 to 17	Sentance, S., Schwiderski-Grosche, S., 2012. Challenge and creativity: using. NET gadgeteer in schools, in: Proceedings 7th WorkshopPrimary Secondary Computing Education. ACM, pp. 90–100.	UK	Qualitative	Small 16 interviewed	.NET gadgeteer afterschool	16
Game-making	Primary	Kafai, Y.B. & Burke, Q., 2015. Constructionist gaming: Understanding the benefits of making games for learning. Educational psychologist, 50(4), pp.313–334.	USA	Literature Review	55 papers	Making Games	17
	KS3 11-14	Repenning, A., Webb, D. C., Koh, K. H., Nickerson, H., Miller, S. B., Brand, C., Basawapatna, A., et al. (2015). Scalable game design: A strategy to bring systemic computer science education to schools through game design and simulation creation. *ACM Transactions on Computing Education (TOCE)*, *15*(2), 11.	USA	Qualitative	Large 10,000		18
		Webb, D. C., Repenning, A., & Koh, K. H. (2012). Toward an emergent theory of broadening	USA	Qualitative	Large 1420		19

		participation in computer science education. *Proceedings of the 43rd ACM technical symposium on Computer Science Education* (pp. 173–178). ACM.					
Unplugged	KS3 Grade 7	Rodriguez, B., Kennicutt, S., Rader, C., & Camp, T. (2017). Assessing Computational Thinking in CS Unplugged Activities. Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (pp. 501–506). ACM.	USA	Quantitative	Medium 141 students		20
	KS5 (High school)	Ford, V., Siraj, A., Haynes, A., & Brown, E. (2017). Capture the Flag Unplugged: an Offline Cyber Competition. Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (pp. 225–230). ACM.	USA	Mixed	Small 36		21
Cross curricula	N/A	Moreno-León, J., & Robles, G. (2016). Code to learn with Scratch? A systematic literature review. Global Engineering Education Conference (EDUCON), 2016 (pp. 150–156). IEEE.	Spain	Systematic literature review	15 papers	Learning non-computing subjects using Scratch	22
	KS2	Benton, L., Hoyles, C., & Noss, I. K. anRichard. (2017). Bridging Primary Programming and Mathematics: some findings of design research in England. Digital Experiences in Mathematics Education.	UK	Qualitative	Medium (55 students)		23

Programming Languages

	Phase	Articles included	Country	Study Type	Study Size	Study context	
Pedagogy based transition	Primary and KS3	Dorling, M., & White, D. (2015). Scratch: A way to logo and Python. Proceedings of the 46th ACM Technical Symposium on Computer Science Education (pp. 191–196). ACM.	UK	Theoretical	N/A		1
		Franklin, D., Hill, C., Dwyer, H. A., Hansen, A. K., Iveland, A., & Harlow, D. B. (2016). Initialization in Scratch: Seeking Knowledge Transfer. Proceedings of the 47th ACM Technical Symposium on Computing Science Education (pp. 217–222). ACM.	USA	Review of field notes, analysis of code	Not stated		2
	KS3	Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (2015). From Scratch to "real" programming. ACM Transactions on Computing Education (TOCE), 14(4), 25.	Israel	Mixed + Control group	Medium (120 students)	Scratch to C# or Java across 5 classes, 4 schools, 4 teachers	3
Hybrid	KS5 and older	Weintrop, D., & Holbert, N. (2017). From Blocks to Text and Back: Programming Patterns in a Dual-Modality Environment. Proceedings of the 2017 ACM SIGCSE Technical Symposium on	US	Quantitative	Small 23 (13 KS5 +	Could choose which	4

		Computer Science Education (pp. 633–638). ACM.			10 HE)	modality to use	
Hybrid and pedagogy	University	Dann, W., Cosgrove, D., Slater, D., Culyba, D., & Cooper, S. (2012). Mediated transfer: Alice 3 to java. Proceedings of the 43rd ACM technical symposium on Computer Science Education (pp. 141–146). ACM.	USA	Quantitative	Medium 78	2 cohorts	5
Frame based	KS3	Price, T. W., Brown, N. C., Lipovac, D., Barnes, T., & Kölling, M. (2016). Evaluation of a Frame-based Programming Editor. Proceedings of the 2016 ACM Conference on International Computing Education Research (pp. 33–42). ACM.	USA	Quantitative	Small 32 students	Elective outreach 1 hour lesson	6
Notational Machine and visualisation	N/A	Sorva, J., Karavirta, V., & Malmi, L. (2013). A review of generic program visualization systems for introductory programming education. ACM Transactions on Computing Education (TOCE), 13(4), 15.	Finland	Product review	Approx. 40 Visual programming systems		7
	HE HE	Rajala, T., Laakso, M.-J., Kaila, E., & Salakoski, T. (2008). Effectiveness of Program Visualization: A Case Study with the ViLLE Tool. *Journal of Information Technology Education*, *7*.	Finland	Quantitative	Medium 72 students	VILLE tool	8
		Laakso, M.-J., Rajala, T., Kaila, E., & Salakoski, T. (2008). The impact of prior experience in using a visualization tool on learning to program. *Proceedings of Cognition and Exploratory Learning in Digital Age (CELDA 2008)*, 13–15.	Finland	Quantitative	Small 24 students	VILLE tool	9
		Rajala, T., Kaila, E., Laakso, M.-J., & Salakoski, T. (2009). Effects of Collaboration in Program Visualization. *Proceedings of 2009 Technology Enhanced Learning Conference (TELearn 2009)*.	Finland	Quantitative	Small	VILLE tool	10
		Kaila, E., Laakso, M.-J., Rajala, T., & Salakoski, T. (2009). Evaluation of Learner Engagement in Program Visualization. *12th IASTED International Conference on Computers and Advanced Technology in Education (CATE 2009)*.	Finland	Quantitative	Small	VILLE tool	11
	KS5 16-19 years' old	Kaila, E., Rajala, T., Laakso, M.-J., & Salakoski, T. (2010). Effects of course-long use of a program visualization tool. Proceedings of the Twelfth Australasian Conference on Computing Education-Volume 103 (pp. 97–106). Australian Computer Society, Inc.	Finland	Quantitative (with control group)	Small 23	ViLLE tool embedded in high school Python course	12

Student Engagement

	Phase	Articles included	Country	Study Type	Study Size	Study context	
Pair programming	Primary and Secondary	Kafai, Y.B. & Burke, Q., 2015. Constructionist gaming: Understanding the benefits of making games for learning. Educational psychologist, 50(4), pp.313–334.	USA	Literature Review	55 papers		1
		Falkner, K., & Vivian, R. (2015). A review of computer science resources for learning and teaching with K-12 computing curricula: An Australian case study. Computer Science Education, 25(4), 390–429	Australia	Systematic Resource Review	65 resources		2
	HE	Hanks, B. et al., 2011. Pair programming in education: a literature review. Computer Science Education, 21(2), pp.135–173.	USA	Literature Review	43 papers		3
		Salleh, N., Mendes, E. & Grundy, J., 2011. Empirical studies of pair programming for CS/SE teaching in higher education: A systematic literature review. IEEE Transactions on Software Engineering, 37(4), pp.509–525.	USA	Literature Review	74 papers		4
	KS3	Werner, L. Denner, J. Campe, S. Ortiz, E. DeLay, D. Hartl, A. Laursen, B. 2013. Pair programming for middle school students: does friendship influence academic outcomes? In Proceeding of the 44th ACM technical symposium on Computer science education. ACM, pp. 421–426.	USA	Mixed	Large 189 students	Alice in school 2009-2011 study Focus friendship	5
		Werner, L. Denner, J. Campe S. and Ortiz E 2014. Pair programming: Under what conditions is it advantageous for middle school students? *Journal of Research on Technology in Education*, 46(3), pp.277–296.	USA	Mixed	Large 320 students	Alice in school 2009-2011 Focus Pair vs Solo	6
		Ruvalcaba, O., Werner, L. & Denner, J., 2016. Observations of Pair Programming: Variations in Collaboration Across Demographic Groups. In Proceedings of the 47th ACM Technical Symposium on Computing Science Education. ACM, pp. 90–95.	USA	Qualitative (Video analysis)	Large 158 students	Alice in school 2009-2011 Focus ethnicity	7
		Lewis, C.M., 2011. Is pair programming more effective than other forms of collaboration for young students? Computer Science Education, 21(2), pp.105–134.	USA	Mixed	Small 40 students	Summer school	8
	KS4	Missiroli, M., Russo, D. & Ciancarini, P., 2016. Learning Agile software development in high school: an investigation. In Proceedings of the 38th International Conference on Software Engineering Companion. ACM, pp. 293–302.	Italian	Qualitative	Medium 84 students	In school	9
	KS3	Passey, D., 2013. inspire – Wolverhampton's Local Education Partnership: evaluating the	UK	2 Case Studies	Not clear	Digital Leaders	10

		development and practices of digital leaders in Wolverhampton schools, Lancaster University. Passey, D., 2014. Intergenerational learning practices—Digital leaders in schools. Education and Information Technologies, 19(3), pp.473–494.					
	KS2	Ching, C.C. & Kafai, Y.B., 2008. Peer pedagogy: Student collaboration and reflection in a learning-through-design project. Teachers College Record, 110(12), pp.2601–2632.	USA	Design based research Qualitative	Medium 63 students	Peer apprenticeship	11
	N/A	Al-Jarrah, A. & Pontelli, E., 2014. AliCe-ViLlagE" Alice as a Collaborative Virtual Learning Environment. In Frontiers in Education Conference (FIE), 2014 IEEE. IEEE, pp. 1–9.	USA	Theoretical	N/A	Toolset research	12
PjBL	KS3	Garneli, V. et al., 2015. Serious game development as a creative learning experience: lessons learnt. In Proceedings of the Fourth International Workshop on Games and Software Engineering. IEEE Press, pp. 36–42	Greece	Quantitative (of code created)	medium (53 school students		13
Agile	KS4	Missiroli, M., Russo, D. & Ciancarini, P., 2016. Learning Agile software development in high school: an investigation. In Proceedings of the 38th International Conference on Software Engineering Companion. ACM, pp. 293–302.	Italian	Qualitative	Medium 84 students	In school	14
	KS5 (High school)	Kastl, P., Kiesmüller, U., & Romeike, R. (2016). Starting out with Projects: Experiences with Agile Software Development in High Schools. Proceedings of the 11th Workshop in Primary and Secondary Computing Education (pp. 60–65). ACM.	Germany	Qualitative	Medium 140		15
	KS2	Aggarwal, A., Gardner-McCune, C., & Touretzky, D. S. (2017). Evaluating the Effect of Using Physical Manipulatives to Foster Computational Thinking in Elementary School. Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (pp. 9–14). ACM.	USA	Mixed	Small 11		16

Appendix B

List of example physical computing programmable devices.

Product name	Device type	Weblink
Bee-Bot	Programmable Robot	http://www.tts-group.co.uk/bee-bot-rechargeable-floor-robot/1001794.html
Blue Bot	Programmable Robot	http://www.tts-group.co.uk/blue-bot-bluetooth-programmable-floor-robot/1007812.html?gclid=CJqHgpLdos8CFQw6Gwods5kNYw
Codybot	Programmable Robot	http://www.makeblock.com/codeybot
Cubetto	Programmable Robot	https://www.primotoys.com/
Dash and Dot	Programmable Robot	https://www.makewonder.com/dash
Edison	Programmable Robot	https://meetedison.com
Finch	Programmable Robot	http://www.finchrobot.com/
KIBO	Programmable robot and Tangible programming interface	http://www.shop.kinderlabrobotics.com/KIBO-Sets_c7.htm
mbot	Programmable Robot	http://www.makeblock.com/mbot-v1-1-stem-educational-robot-kit
Moway	Programmable Robot	http://moway-robot.com/en/
Nao	Programmable Robot	https://www.ald.softbankrobotics.com/en/cool-robots/nao
Ollie	Programmable Robot	http://www.sphero.com/ollie
Ozobot	Programmable Robot	http://ozobot.com/
Pro- Bot	Programmable Robot	http://www.tts-group.co.uk/pro-bot-rechargeable-floor-robot/1009825.html
Roamer	Programmable Robot	http://www.valiant-technology.com/uk/pages/roamertoohome.php?cat=8&8
Sphero	Programmable Robot	http://www.sphero.com/
Arduino	Microcontroller	https://www.arduino.cc/
Bareconductive	Microcontroller	https://makerclub.org/product/bare-conductive-touch-board/
BBC micro:bit	Microcontroller	https://www.microbit.co.uk/
Codebug	Microcontroller	http://www.codebug.org.uk/
Crumble	Microcontroller	http://redfernelectronics.co.uk/crumble/
Engduino	Microcontroller	http://www.engduino.org/html/index.html
.NET Gadgeteer	Microcontroller	http://www.netmf.com/gadgeteer/
GoGO board	Microcontroller	http://gogoboard.org/
gpio box	Control box for microcontrollers and Raspberry Pi	http://www.gpio.co.uk/
hornet board	Microcontroller	https://makerclub.org/product/the-hornet-board/

Hummingbird robotics kit	Programmable kit (electronics/maker kit)	http://www.hummingbirdkit.com/
Lego Mindstorms	Programmable Kit (robotics)	http://www.lego.com/nl-nl/mindstorms
LEGO WeDo	Programmable input/output device	https://education.lego.com/en-us/elementary/shop/wedo-2
little bits	Programmable kit (electronics/maker kit)	https://littlebits.cc/bits/w6-arduino
Meccano robots	Programmable kit (electronics/maker kit	http://www.meccano.com/meccanoid-about
Picoboard	Programmable input/output device	http://www.picocricket.com/picoboard.html
Raspberry Pi	Single board Computer and programmable electronics/maker kits	https://www.raspberrypi.org/
Tech will save us kits	Programmable kit (electronics/maker kit)	https://www.techwillsaveus.com/
Makey Makey	Programmable input device	http://makeymakey.com/
Scratch controller	Programmable input device	http://www.tts-group.co.uk/Scratch-controller-input-device/1010503.html
Scratch LED matrix	Programmable input device	http://www.tts-group.co.uk/Scratch-led-rainbow-matrix/1011571.html
Bloxels	Tangible programming interface	http://www.bloxelsbuilder.com/education-overview/
Makeblock	Tangible programming interface	http://www.makeblock.com
Osmo	Tangible programming interface	https://www.playosmo.com/en/coding/
puzzlet	Tangible programming interface	https://www.digitaldreamlabs.com/educators/
lightup	Tangible programming interface	http://www.lightup.io/

Appendix C

List of example educational block-based programming languages.

Language	Weblink	Notes
AgentSheets	http://www.agentsheets.com/	
Alice	https://www.alice.org/	
AppInventor	http://appinventor.mit.edu/explore/	
AppLab	https://code.org/educate/applab	Hybrid (JavaScript)
Blockly	https://blockly-games.appspot.com/	
Bubble	https://bubble.is/	
BYOB/ Snap	https://snap.berkeley.edu/	
Code.org	https://code.org/	
codecombat	https://codecombat.com/	
CTSiM	http://www.ctsim.org/	
Daisy the Dinosaur	http://www.daisythedinosaur.com/	
edublocks	http://edublocks.org/	hybrid (Python)
Espresso coding	http://www.discoveryeducation.co.uk/what-we-offer/discovery-education-coding#newlook	
Etoys	http://www.squeakland.org/	
Flowgorithm	http://www.flowgorithm.org/	
Gamefoot	http://gamefroot.com/knowledgebase/how-to-use-scripts-to-program-game-objects/	
GameMaker	http://www.yoyogames.com/gamemaker?utm_source=google_adwords&utm_medium=text_ads&utm_campaign=Game_Making_UK&utm_term=Game_Maker&gclid=COOurq6tq9MCFXEz0wodGCMM2w	
GameSalad	http://gamesalad.com/	
GP	https://harc.ycr.org/project/gp/	
Greenfoot	https://www.greenfoot.org/door	visual tools to learn java
Hopscotch	https://www.gethopscotch.com/	
J2Code	https://www.j2e.com/visual.html?edit	Hybrid (JavaScript)
Kodu	https://www.kodugamelab.com/	
LaPlaya	http://people.cs.uchicago.edu/~dmfranklin/kelpcs/why.html	
LaPlaya	https://discover.cs.ucsb.edu/kelpcs/why-kelp-cs.html	
Lego NXT [1]	https://www.lego.com/en-gb/mindstorms	
Modkit	http://www.modkit.com/	
NetsBlox	https://netsblox.org/	
PencilCode	https://pencilcode.net/	Hybrid (JavaScript, HTML, CSS)
PicoBlocks	http://www.picocricket.com/download.html	
PocketCode	https://play.google.com/store/apps/details?id=org.catrobat.catroid&hl=en_GB	
Raptor	http://raptor.martincarlisle.com/	
Scratch	https://Scratch.mit.edu/	
Sketchware	http://sketchware.io/	
Stagecast	http://acypher.com/creator/	
StarLogo	http://education.mit.edu/portfolio_page/starlogo-tng/	
Stencyl	http://www.stencyl.com/	
Toontalk	http://www.toontalk.com/	
Tynker	https://www.tynker.com/	
Visual Logic	http://www.visuallogic.org/	

Appendix D

List of literature studies per source by theme, showing literature counts by sources.

Theme	Number of papers retrieved from initial search per source						Papers included
	ACM	IEE	Taylor	Wiley	Eric	Total	
Pedagogy	215	7	131	88	22	485	35
Contexts	31	12	22	17	29	112	23
Programming	18	12	0	9	3	42	12
Student Engagement	20	10	2	3	79	114	16
Totals						733	86

Glossary

Term	Definition
Abstraction Transition (AT) Taxonomy	A taxonomy suggested by Cutts et al. (2012) to support the learning of programming, which classifies kinds of student knowing and practices. The model includes three main levels of code; CS speak; and English and nine transitions across the three levels each with a how and why goal defined. These 18 goals are claimed to develop students' programming. An example transition goal given by the study was 'Given a technical description (CS Speak) of how to achieve a goal, choose code that will accomplish that goal'.
Block Model	A three-dimensional educational model of program comprehension, suggested by Schulte et al. (2010). The model includes a vertical axis of levels of code detail, a horizontal axis of the continuum of structure (including text surface and notional machine) and function axis and a third dimension representing time on task depicting a range of understanding from fragile to deep.
Foundations of Advancing Computational Thinking (FACT)	Grover et al.'s (2015) blended computer science course for middle school students developed for 'deeper learning' focusing on pedagogical strategies to support and assess the transfer from block to text-based programming, including materials to remedy misconceptions and provide systems of assessment. (Grover et al., 2015).
Levels of abstraction	A framework, depicting programming projects in terms of a problem level, a design/object/algorithm level, a code level and a code running level (Armoni, 2013; Perrenet & Kaasenbrood, 2006; Statter & Armoni, 2016; Waite et al., 2016).
New combined taxonomy	A programming progression taxonomy which combines the Solo taxonomy (horizontal axis) and elements of Bloom's taxonomy (vertical axis). Created to support an Israeli middle school Scratch curriculum, the authors claimed that higher levels of the taxonomy imply deeper comprehension than the superficial lower levels as learners progress from 'unistructural understand' for easiest student performance to 'relational create' the highest level of mastery (Meerbaum-Salant et al., 2013).
Pair Programming	A collaborative approach to programming where two people work at one computer to complete a single design, algorithm, coding or testing task (Williams & Kessler, 2000). One person takes the role of the driver, having control over the keyboard and mouse, and the second person is the navigator or observer, constantly reviewing the code written, keeping track of progress against the design (McDowell et al., 2006) and continuously collaborating (Williams & Kessler, 2000). Whilst working on a task, the driver and navigator swap roles after a certain period of time, code is only included or removed with agreement between parties (McDowell et al., 2006).
Path Diagram	A path of related task and understanding to support programming development, including knowing about data structures, programming constructs, tracing, explaining and writing programs (Lopez et al., 2008).
Peer Instruction	A research based teaching method where students apply, discuss and explain concepts by engaging students independently and collaboratively with carefully designed questions. The method was pioneered by Eric Mazur, professor at Harvard University for teaching undergraduate Physics. Simply put, the teacher formulates a question that addresses a misconception or concept. This is presented for students to independently answer, sometimes by using a voting system. Students then work in small groups to arrive at a consensus answer, requiring the students to explain and clarify their understanding. Each student is then asked to vote again. Finally, the teacher leads a class discussion to review answers and address any misunderstandings (Crouch & Mazur, 2001).

Positive Technological Development (PTD) framework	Developed to support the TangibleK curriculum, this framework supports the development of learning of robotics. The framework incorporates assets, behaviours and classroom practise to situate progression in a sociocultural context (Bers et al., 2014).
Process Oriented Guided Inquiry Learning[47] (POGiL)	A user-centred, guided inquiry, problem-solving approach originally developed for chemistry students that guides learners to construct new knowledge. Students work in small groups and are assigned specific roles to ensure they are fully engaged in the learning process.
Program visualisation tools	Tools that visually illustrate the behaviour of a program in different states as it executes. Program visualisation tools typically show the values of variables, expression evaluation or object and function dependencies. Often they include options to step forwards and backwards in a program. They can be used by teachers as whole class demonstrations or independently by learners. Some toolsets allow teachers to create embedded questions that actively engage students in the tool use.
Progression of Early Computational Thinking (PECT) model	Seiter & Forman's model for understanding and assessing progression in computational thinking. The model includes computational thinking concepts, Design Pattern Variables (ability to recognise and use commands and programming constructs for a particular purpose) and Evidence Variables (code blocks) (Seiter & Foreman, 2013). The model is intended to be used to analyse programs, such as Scratch code, to evaluate the programmers' progression in computational thinking.
Subgoal modelling	A teaching approach whereby meaningful labels are added to programs to highlight the structure of the code (Margulieux & Catrambone, 2016; Morrison et al., 2016).
Universal Design of Learning (UDL) framework	The UDL framework is a teaching and learning framework created to support the needs of all learners, meeting their cognitive, language and mathematical needs, incorporating gender neutral and appropriate ethnic and linguistic curricula content (Hansen, Hansen, et al., 2016)
Use-Modify-Create	A teaching framework for supporting progression in learning to program. Learners move along a continuum where the start using programs made by someone else to finally create their own programs. Between these points they modify work made by someone else so that the modified material becomes 'theirs'. Once students start to create their own programs they employ an iterative process of refine, test, analyse (Lee et al., 2011).
Scalable Games Design (SGD)	A teaching and learning initiative that includes teacher training, online authoring tools (AgentSheets and AgentCubes), an environment for sharing work created and curriculum materials. This is a long-term project-based at the University of Colorado, which uses a project first approach for learning object oriented programming. Over 10,000 learners have been involved, mainly in the US and more recently in Mexico and Switzerland. Rather than teaching programming constructs such as loops, if-then statements or data structures the approach is to teach computational thinking patterns which are common in the design of games, such as generation, absorption, collision. These patterns are first learned in the making of games and then reapplied in the making of simulations for science, maths and other subjects (Repenning et al., 2015).

[47] https://pogil.org/about last accessed 13/5/2017