

Teaching computer programming with PRIMM: a sociocultural perspective

Sue Sentance, Jane Waite & Maria Kallia

To cite this article: Sue Sentance, Jane Waite & Maria Kallia (2019): Teaching computer programming with PRIMM: a sociocultural perspective, Computer Science Education, DOI: [10.1080/08993408.2019.1608781](https://doi.org/10.1080/08993408.2019.1608781)

To link to this article: <https://doi.org/10.1080/08993408.2019.1608781>



Published online: 26 Apr 2019.



Submit your article to this journal [↗](#)



View Crossmark data [↗](#)

ARTICLE



Teaching computer programming with PRIMM: a sociocultural perspective

Sue Sentence^a, Jane Waite^b and Maria Kallia^a

^aDepartment of Education, Communication and Society, King's College London, London, UK;

^bSchool of Electronic Engineering and Computer Science, Queen Mary University of London, London, UK

ABSTRACT

Background and Context: Vygotsky's sociocultural theory emphasises the importance of language, mediation, and the transfer of skills and knowledge from the social into the cognitive plane. This perspective has influenced the development of PRIMM (Predict, Run, Investigate, Modify, Make), a structured approach to teaching programming.

Objective: The objective of the study was to find out if using PRIMM to teach programming had an impact on learner attainment in secondary school, and the extent to which it was valuable for teachers.

Method: We evaluated the use of PRIMM in 13 schools with 493 students aged 11-14 alongside a control group, using a mixed-methods approach. Teachers delivered programming lessons using the PRIMM approach for 8-12 weeks. Data were collected via a combination of a baseline test, a post-test to compare control and experimental groups, and teacher interviews.

Findings: Learners performed better in the post test than the control group. Teachers reported several benefits of the PRIMM approach, including that PRIMM helped them to teach effectively in mixed-ability classes, enabling all learners to make progress.

Implications: PRIMM makes a contribution to programming education research by building on previous work in effective pedagogy for teachers, and encouraging the use of language and dialogue to facilitate understanding.

ARTICLE HISTORY

Received 6 August 2018

Accepted 15 April 2019

KEYWORDS

Computer programming; computer science education; K-12 education; pedagogy; sociocultural theory; Vygotsky

1. Introduction

With mandatory computing being introduced more widely, the prospect of every child in school studying computing throughout their school years is exciting. However, this change brings a challenge, particularly where computer programming is part of every school's core curriculum. To truly champion computing literacy, we need to understand how the learning of programming can be truly accessible to all (The Royal Society, 2017).

Looking at this existing body of programming education research, it can be seen that it has primarily been informed by cognitive rather than sociocultural approaches to learning (Tenenbergh & Knobelsdorf, 2014). Cognitive approaches to learning do not account for all the phenomena observed in teaching and learning (Machanick, 2007), whereas a sociocultural perspective enables us to consider how learning is impacted by society, its agents, the environment and social factors. Therefore, we wish to investigate how looking at the learning of programming through this lens can help us to develop an enhanced understanding.

Our particular focus is teachers, and how they can understand students' learning, and apply practicable and effective strategies in the computing classroom. This is particularly pertinent for new computing teachers who have yet to develop their pedagogical content knowledge (PCK) (Shulman, 1986) in the teaching of programming. Even experienced teachers faced with teaching programming for the first time will not have a toolkit of strategies applicable to this domain. Thus by looking at the learning of programming through a sociocultural lens we have developed a strategy for teaching programming that acknowledges the role of mediating factors, including language and tools, and aligns to a Vygotskian perspective. We believe highlighting classroom interaction, use of language, and the tools and techniques used, as well as cognitive factors, can support teachers in developing their PCK for the teaching of programming.

Drawing on both existing research into the learning of computer programming, and with our Vygotskian lens, we have developed and trialled a framework for teaching programming called PRIMM (Predict-Run-Investigate-Modify-Make) (Sentance & Waite, 2017; Sentance, Waite, & Kallia, 2019). PRIMM is a method of teaching programming that counters the known problem of novices writing programs before they are yet able to read them, and focuses on students talking about how and why programs work before they tackle editing and writing their own programs. We have conducted a mixed-methods study researching the impact of the PRIMM method for teaching programming. Through this study, we can see an impact of PRIMM on learners and teachers. Positive quantitative results of learner outcomes are supported by the teachers' explanations about why this method works for them. Through this research, we are formulating a model that suggests that developing a structured approach to teaching and learning programming draws on teachers, peers and carefully selected content. Our model is informed by and aligns with the Vygotskian concept of mediation and the zone of proximal development (ZPD). The results of the study point to the way that language is used in the learning of programming. Key to our research is not simply what works but why it works and how teachers' understanding of programming learning can support better outcomes.

This paper is structured as follows. Firstly we consider previous research around the teaching and learning of programming, focusing primarily on pedagogical

strategies teachers can use. Secondly we describe how a sociocultural perspective, particularly using the work of Vygotsky, can inform both an overarching framework and also the specific PRIMM model that we are describing in this paper. Thirdly, we describe a mixed-methods study of the use of PRIMM in the classroom and the results. Finally, we outline where future research can be directed to gain more understanding of effective teaching strategies for programming, particularly in the school classroom.

2. Teaching and learning programming

2.1. The challenges

For many students, computer programming is among the most challenging aspects of computer science (Jenkins, 2002). Many computing education researchers have endeavoured to identify the sources of students' difficulties in programming, some focusing on categorising problematic areas. For example, going back to the 1980s, Perkins & Martin (1986) identified four types of fragile knowledge: partial knowledge, inert knowledge, misplaced knowledge, and conglomerated knowledge. Du Boulay (1986) also explored and categorised curriculum areas that students experience difficulties: orientation, notional machine, notation, structures and pragmatics, and made a step further suggesting specific types of errors students make: misapplication of analogy, overgeneralisations and interactions. Other researchers have focused on bugs and misconceptions. As examples, Pea (1986) identified conceptual bugs that students often make in their attempt to communicate with the computer, and Spohrer and Soloway (1986) categorised students' errors into two categories: construct-based problems, which are problems that refer to the semantics of programming constructs, and plan composition problems, which refer to difficulties in putting plans together. Sorva, Karavirta, and Malmi (2013) describe a range of challenges facing novice programmers including static perceptions of programming, misconceptions, difficulties understanding the computer and tracing and program state. Actual concepts novice programmers struggle with include variables, loops, and parameter passing (Wiegand, Bucci, Kumar, Albert, & Gaspar, 2016), and recent work has identified a range of (41) specific programming misconceptions relating to these concepts (Sorva, 2018).

Generally, research agrees that novices' problems do not only centre around syntax or semantics but mostly on how to associate these to construct a program (Robins, Rountree, & Rountree, 2003); additionally, students have a surface knowledge of programming which is context specific and, thus, it is difficult to be applied in different contexts (Lahtinen, Ala-Mutka, & Järvinen, 2005). It has been argued that programming is a hierarchical skill, and that students who do not understand a topic cannot advance to the next one (Rahmat et al., 2012). For example, Linn and Dalbey (1985) suggested three

chained cognitive accomplishments that students should exhibit: single language features, design skills, and problem-solving skills. This complex hierarchy of dependent knowledge and skills compounds the difficulties faced by students when learning to program.

Actually writing code (as opposed to reading) is particularly hard for novice programmers (Denny, Luxton-Reilly, & Simon, 2008; Qian & Lehman, 2017), and it is commonly believed that code tracing is easier than code writing (Denny et al., 2008), with some research showing a correlation (Lopez, Whalley, Robbins, & Lister, 2008; Sheard et al., 2008), with others less conclusive (Simon, Lopez, Sutton, & Clear, 2009). Many students find code tracing challenging (Vainio & Sajaniemi, 2007) with particular difficulties being around single value tracing, confusion of function and structure, external representations and levels of abstraction.

The mental effort needed by learners as they embark on this complex journey of learning to program can be viewed through cognitive load theory (Van Merrinboer & Sweller, 2005). Cognitive load theory is a theory of instructional design that suggests that some instructional techniques assume a processing capacity greater than our limits and so are likely to be defective, and that students should instead engage in activities that are directed at schema acquisition and automation (Sweller, 1994). Working independently on programming has been suggested to have higher cognitive load than working collaboratively through pair programming (Tsai, Yang, & Chang, 2015). Similarly, some programming languages have been indicated to have a greater toll on cognitive load than others. Alexandron, Armoni, Gordon, and Harel (2014) point to object-oriented programming as requiring programmers to think about solving the parts of a problem and the integration of these at the same time whereas declarative scenario-based languages allow learners to have an initial simpler mental model as they attend to the parts first before being required to incorporate these into the whole.

With so many challenges we can appreciate what is described as the *“long and torturous cognitive development of the novice programmer”* (Corney, Teague, Ahadi, & Lister, 2012, p. 86). However to be able to teach programming effectively in school, we need to look beyond challenges to effective teaching strategies that take into account those difficulties, which can be used with all ages of learners.

2.2. Modelling learning

Over time, learning models have been developed to explain how novices learn programming. The two most influential to the development of PRIMM, the Block Model and Levels of Abstraction (LOA), are discussed here.

2.2.1. Levels of abstraction

The Levels of Abstraction (LOA) framework has been suggested to support novices in their learning to program (Perrenet, Groote, & Kaasenbrood, 2005; Perrenet & Kaasenbrood, 2006). Perrenet et al. suggested four levels, namely: *execution*; *program*; *object* and *problem*. Armoni (2013) renamed the *object* level as *algorithm*; using this Statter and Armoni (2016) evidenced impact on programming progress from high school students who understood the level at which they were working at. Another study using this framework reported teachers of K-6 students supporting learners to move between the LOA levels in teaching programming in similar ways to how they supported young students in learning how to write in English lessons (Waite, Curzon, Marsh, & Sentence, 2018). Related to this work is the Abstraction Transition Taxonomy (ATT) (Cutts, Esper, Fecho, Foster, & Simon, 2012). Cutts et al. reviewed university students use of vocabulary when solving peer instruction multiple choice questions and suggested ATT as a discourse intensive teaching model of student understanding of programs, including three levels of language in programming: English, CS Speak and Code. A clear recommendation of this research, which drew on situated cognition, was to support learners to be able to transition across all levels.

2.2.2. The block model

Schulte (2008) suggests a holistic model of learner understanding of programming called the Block Model. The Block Model has a horizontal dimension of *Function* (split into text surface and program execution) and *Structure*. Its vertical dimension has levels of *atoms*, *blocks*, *relations* and *macro structure*. Function is described as the goals of the program. Using the Block Model Schulte, Clear, Taherkhani, Busjahn, and Paterson (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 argue that 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). 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) resonates with the development of the LOA model described above, and has influenced the development of the PRIMM model.

2.3. Teaching strategies

There have been many strategies for teaching programming developed over the years. Work on teaching strategies has influenced the development of the PRIMM approach, particularly the work on code reading and tracing and Use-Modify-Create.

2.3.1. Structured and less structured approaches

With much research associated with the teaching of programming focused on pedagogy and instructional approaches to teaching, Papert has been very influential (Papert, 1980); from his work from the 1980s onwards we see instructional approaches based around open-ended activities, through which students can develop a personal understanding of newly introduced concepts or devices. This approach is still very popular with its emphasis on creativity and play (Resnick, 2007, 2017), and underlies much of the block-based programming environment design.

However, other research has highlighted the need for guided instruction to ensure that learners circumnavigate a carefully constructed progression to develop a complete mental model (Garneli, Giannakos, & Chorianopoulos, 2015; Grover, Pea, & Cooper, 2015; Lye & Koh, 2014; Meerbaum-Salant, Armoni, & Ben-Ari, 2013; Schulte, 2008). Grover *et al.* suggest that to foster deep learning a combination of guided discovery and instruction rather than pure discovery and “tinkering” would be more successful (Grover *et al.*, 2015). 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; Meerbaum-Salant *et al.*, 2013). In addition, researchers claim that learners’ cognitive load can be managed by more closely controlling learning opportunities and learning experiences (Alexandron *et al.*, 2014; Tsai *et al.*, 2015; Van Merrinboer & Sweller, 2005). The PRIMM approach recognises the need for structured approaches to the teaching of programming, particularly in a school setting.

2.3.2. Code reading vs code writing

Many authors have investigated the difference between code reading and code writing as instructional strategies. Back in 1987, Van Merrienboer and Krammer (1987) found that an approach to teaching programming based on code reading was more effective than the so-called Expert and Spiral approaches, which focused on top-down code-writing and incremental program design, respectively. Work by Lister and colleagues over many years has

highlighted the importance of reading code and being able to trace what it does before writing new code (Lister et al., 2004; Lister, Fidge, & Teague, 2009). Comparing tracing skills to code writing, they demonstrated that novices require a 50% tracing code accuracy before they can independently write code with confidence (Lister et al., 2009; Venables, Tan, & Lister, 2009). Learning to program is sequential and cumulative, and tracing requires students to draw on accumulated knowledge to conceive a big picture; consequently, novice learners should be focused on very small tasks with single elements (Teague & Lister, 2014). Busjahn and Schulte (2013) highlighted the importance of using the structure of code to infer meaning and that the first step should be to make inferences about the execution of the program (Busjahn & Schulte, 2013). Path diagrams, consisting of paths of related tasks (Lopez et al., 2008), and visualisation and tracing combined (Hertz & Jump, 2013), are related approaches. Parsons problems can offer a valid alternative activity to code writing and develop useful skills (Denny et al., 2008). Code-tracing activities can help students learn to write both syntactic and semantic components of code (Kumar, 2015).

2.3.3. *Worked examples*

Studies related to code comprehension have also highlighted the use of worked examples to understand how variables change over time (Sudol-DeLyser, Stehlik, & Carver, 2012). Gujberova and Kalas (2013) recommended a sequence of carefully graded learning activities for primary students to improve programming and computational thinking, including activities where learners read and interpreted each line of code, as well as a stage for reading the entire program and predicting the outcome (Gujberova & Kalas, 2013).

Another way of using worked examples is 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; 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, suggesting that providing more support is more effective for novice learners.

2.3.4. *Use-modify-create*

Another approach used in the teaching of programming is Use-Modify-Create (UMC), a teaching framework for supporting progression in learning to program (Lee et al., 2011). Learners move along a continuum from where they first *use* 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”. This work has some history, as it builds on

a range of related work (Caspersen, 2018), including the “call before write” suggestion by Pattis (Pattis, 1990) and “use-extend-create” (Caspersen & Bennedsen, 2007).

2.3.5. Differentiation

Differentiation means the adaptation of teaching strategies so that learning is accessible by all learners in a class. There are several approaches to differentiating instruction: differentiation by task, by outcome and by time allowed, differentiating by accommodating different learning styles and support needs, and differentiation by setting individual tasks and targets (Petty, 2004). Knowing how to differentiate lessons for students with different levels of ability was one of the five most commonly mentioned challenges described by Computing teachers in a recent study (Sentance & Csizmadia, 2017), particularly for teachers teaching Computing in secondary schools. In establishing the PRIMM methodology, we were aware of the need of teachers teaching mandatory computing to mixed-ability classes to be able to find approaches that would both stretch the most able learners and support those who needed additional help.

This review gives an indication of the wealth of suggestions for teaching programming that have been suggested. Still other approaches include annotation (Su, Yang, Hwang, Huang, & Tern, 2014) and live coding (Rubin, 2013). In this paper, we add to this body of research by suggesting PRIMM as a practical strategy that teachers can use to structure lessons and effectively teach programming. It builds primarily on the Use-Modify-Create research but is influenced by the need for language and talk to aid understanding.

3. Vygotsky and socio-cultural theory

Social constructivism, in particular, the work of the Soviet psychologist Vygotsky, can frame our understanding of novice programmers and their learning, and help us to develop effective pedagogical strategies, drawing on this interpretation of the learning process. Vygotsky reoriented learning theory from an individualistic to a sociocultural perspective with his sociocultural theory (SCT) (Kozulin, 2003).

3.1. Mediation

A key element of Vygotsky's work concerns mediation (Wertsch & Tulviste, 1992), which includes both the different kinds of mediational tools adopted and valued by society as well as the appropriation of mediational tools and how they are integrated into cognitive activity during the processes of an individual's development (Shabani, 2016). Such “tools” are appropriated to provide mediation within the learning process.

Vygotsky proposed that higher mental processes were functions of mediated activity and that there were three major classes of mediators: material tools, psychological tools (sometimes called signs and symbols), and other human beings (Kozulin & Presseisen, 1995). Psychological tools can include language, different forms of numeration and counting, mnemotechnic techniques, algebraic symbolism, works of art, writing, schemes, diagrams, maps, blueprints, and all sorts of conventional signs (Vygotsky, 1981, p. 140). Mediation enables a process of apprenticeship followed by internalisation, and SCT suggests that movement from the “social plane” to the “cognitive plane” supports the learning of skills and knowledge (Walqui, 2006, p. 160). This is pertinent to computer programming, where an existing program shared and discussed by learners can be seen to be on the social plane: only when understood and internalised can we describe the program as being on the cognitive plane. With the PRIMM approach, the “starter programs” that are shared and discussed can be seen as being on the social plane.

3.2. The zone of proximal development

In the context of school learning, Vygotsky states that a child’s development within a *zone of proximal development* (ZPD) involves social interaction, dialogue, and mediated activity between learners and with their teachers (Vygotsky, 1978). According to Vygotsky, the zone of proximal development is the distance between what a learner can do (problem solve) independently and what they can do with adult guidance or in collaboration with more capable peers (Vygotsky, 1978, p. 86). The term proximal (nearby) indicates that the assistance provided goes just slightly beyond the learner’s current competence complementing and building on their existing abilities. The ZPD has become synonymous in the literature with the term scaffolding. However, Vygotsky did not use the term in his writing – it was introduced later by Wood, Bruner, and Ross (1976), who describes the scaffolding process as that which enables a child to solve a problem which otherwise would be not possible unassisted.

The concept of ZPD implies that some agency is needed to bridge the gap. This can take many forms, including teachers or peers, but also artefacts such as books, wall displays and online learning environments (Brown et al., 1993). In programming education, there are many tools which provide scaffolding, for example, block-based environments which prevent students from making syntax errors. In the development of PRIMM, we considered that the computing teacher is essential in supporting the student and enabling them to do what they could not do alone, or by exploration, and that structured lessons containing a range of targeted tasks were tools which also formed mediating activity and would support learning.

3.3. *Language*

A key aspect of Vygotsky's sociocultural theory is that language is a central form of mediation that enables thinking and internalisation of concepts to take place; there are different types of talk including inner speech as well as talk that forms part of social interaction. For Vygotsky, social interaction is the basis of learning and development (Walqui, 2006). Through artefacts existing in the social plane that are shared by learners, we are able to use language to understand them and thus internalise that understanding over time. In programming education, the programming language, including the language environment, is the cultural tool for understanding the programming concepts (Kasto, 2016). As a sign system, we can consider the programming language to be a mediator in the same way as natural language.

Vygotsky also discussed the difference between writing in a language and oral expression through language (Vygotsky, 2004). He discusses several reasons for the difference in difficulty associated with written language, with its formal rules, in comparison to oral language, and also explains how when children face challenging tasks the difference between their written and oral language becomes more marked. This suggests that, if we can consider understanding what a program does as being a challenging task, being able to explain verbally what a program does will be easier than expressing that in writing. Some recent research is considering how children verbalise and vocalise program code (Hermans, Swidan, & Aivaloglou, 2018).

3.4. *Vygotsky and pedagogy*

Vygotsky's work is highly relevant to our work with classroom teachers as it was directly concerned with whole-class teaching in public schools (Guk & Kellogg, 2007), and relates to the social transformation that takes place through schooling (Daniels, 2016). Although referring to science teaching, Giest and Lompscher (2003) describe three progressive stages for teaching that apply equally to programming: creating conditions for learning within the zone of actual performance, so identifying what the students can and cannot do, through tasks, then facilitating learning through tasks carefully situated within the zone of proximal development (Vygotsky, 1978) which enable the student to carry out more complex cognitive tasks than they would be able to do on their own, with the support of a "more knowledgeable other" (MKO) and finally moving into the stage where the student is able to work independently and reach their own learning goals. However, according to Giest and Lompschrer, this approach can put high demands on the teachers' "educational and psychological competence".

In their review of the theories of mind underpinning computer science education research, Tenenberg and Knobelsdorf (2014) noted the historical

focus on individualistic cognitive theory with a trend over the last 20 years towards incorporating a sociocultural framing. Sociocultural approaches have strongly influenced a range of recent computer science education work (Bennedsen and Eriksen, 2006; Cajander, Daniels, & McDermott, 2012; Ryoo, 2013), and other research has specifically referenced or used Vygotsky's ZPD (Basawapatna, Repenning, Koh, & Nickerson, 2013; Kotsopoulos et al., 2017), so the influence of SCT is gradually increasing in computer science education. Recently Nadia Kasto's doctoral thesis drew on Vygotskian theory in a study of novice programmers (Kasto, 2016). Kasto describes Robin's learning edge momentum (Robins, 2010) as showing the closest influence of Vygotsky in computer science education research.

3.5. A framework for teaching programming

There have been calls to take a more sociocultural approach within computer science education and programming (Machanick, 2007; Tenenberg & Knobelsdorf, 2014) and particularly with regards to the role of language in teaching computer science (Diethelm and Goschler, 2015). Just as in science teaching, different ways of talking in class about concepts lead to internalisation of the necessary concepts (Leach & Scott, 2003), in computer science education we need to use talk to help students to internalise the difficult concepts they face in programming. In addition, by understanding the process of appropriation better we will be more able to support novice programmers to acquire both a better understanding of programming and confidence in their ability to carry out programming tasks.

From sociocultural theory we can draw on three key principles that can guide the teaching of programming:

- (1) **Mediation through language.** When learning to understand how programs work, students should be encouraged to discuss with each other through a social construction of knowledge. This can be through pair programming, or through collaborative tasks such as talking about segments of program code to identify their function. Teaching should facilitate focused discussion around programming constructs and concepts.
- (2) **Learning moves from the social plane to the cognitive plane.** Using starter programs and teaching the reading of code means that the program code exists on the social plane first, before being understood internally by the student. Programming tasks should be carefully scaffolded so they are within the zone of proximal development of the student. Gradually more complex programming tasks involving independent problem solving and creativity will be possible by students as the understanding becomes internalised.
- (3) **The role of the 'more knowledgeable other' (MKO) in ZPD.** Students need teachers, as More Knowledgeable Others, to show them (model) how to solve a problem. Students working together can be paired so that one peer is the More Knowledgeable Other. The materials and the structure are both mediating activity in Vygotsky's terms, but should be designed to be within the ZPD of the learners. This

requires a detailed understanding of progression and which concepts are easier or harder for students (Wiegand et al., 2016).

This framework can be seen in [Figure 1](#) with suggestions for implementation in [Table 1](#). Drawing on this framework we have developed PRIMM, outlined in the next section.

4. PRIMM

PRIMM is a method of teaching programming that counters the known problem of novices writing programs before they are yet able to read them, and incorporates discussion and investigation of sample code through scaffolded

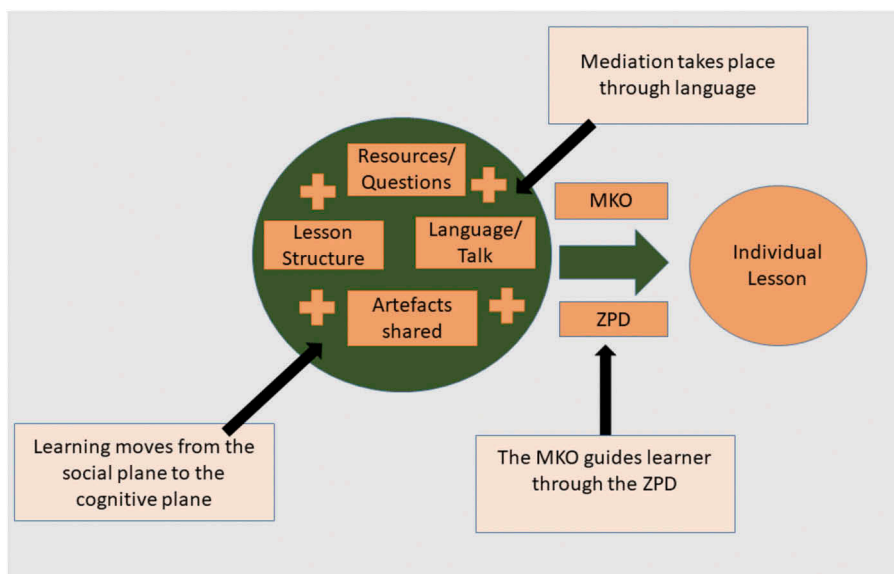


Figure 1. Framework for design of programming lesson.

Table 1. Implementation of key principles.

Principle	Implementation of the principle	Implications for practice in programming lessons
Mediation takes place through language	Focused discussion Collaborative work Questions and talk are part of activities	Work in pairs and small groups to talk about program structure and function.
Learning moves from the social plane to the cognitive plane	Artfacts that already exist Investigate artefacts made by another Gradually move ownership to learner	Work with existing code that already works. Predict what programs will do Gradually modify existing code to add ne functionality.
The MKO guides learner through the ZPD	Teachers model solutions Teachers provide guidance Material is within learners' ZPD Differentiated resources	Teachers structure lessons using PRIMM Activities are adaptable to different abilities Programming tasks are differentiated.

tasks. The PRIMM approach is a process that teachers can use to structure a lesson with five elements: Predict, Run, Investigate, Modify and Make:

- **Predict** Students discuss a program and predict what it might do, drawing or writing out what they think will be the output. At this level, the focus is on the function of the code.
- **Run** Students run the program so that they can test their prediction and discuss in class
- **Investigate** The teacher provides a range of activities to explore the structure of the code; this involves activities such as tracing, explaining, annotating, debugging, but with the scaffolding provided by an existing solution
- **Modify** Students edit the program to change its functionality via a sequence of increasingly more challenging exercises; the transfer of ownership moves from the code being not mine to partly mine as students gain confidence by extending the function of the code
- **Make** Students design a new program that uses the same structures but that solves a new problem (i.e. has a new function)

PRIMM draws on existing research in computer science education, particularly four areas of programming research: Use-Modify-Create (Lee et al., 2011), tracing and reading code before writing (Lister et al., 2004), the Abstraction Transition Taxonomy (Cutts et al., 2012) and the Block Model (Schulte, 2008). In PRIMM, students transition from the *program or code* level to the *execution* level and may also summarise in English to the *problem* or with CS speak to the *algorithm*. During the *run* stage they check to see if their prediction was correct using English, CS speak and code as they accommodate or assimilate their understanding with language and vocabulary becoming the oil to facilitate the transitions.

4.1. A PRIMM lesson

In this section, we briefly describe a PRIMM lesson or sequence of lessons, and the materials that exemplify this. The intention is that teachers can develop their own PRIMM-like materials at an appropriate level for their students.

4.2. Predict and run

At the beginning of a PRIMM lesson, students are given a short program on the board, or on paper, to look at in pairs. The task is for them to write down the output of the program. Most of our examples use Python; an example is shown in Figure 2.

The teacher discusses the students' answers with the class, and students then download code and run to check their prediction. It is important that they do not copy the code as this is a completely different process. Access to

a shared area where starter programs are stored is important, and multiple **predict** activities can be used.

4.3. Investigate

In this phase of the lesson or sequence of lessons, students are asked code comprehension questions about the same program or snippet of code. These questions pick out certain aspects of the program to develop understanding. Developing good questions in this section requires a good understanding of programming and student misconceptions, and the Block Model (Schulte, 2008) (see Figure 4) can help to structure questions. For example, students may be asked a question about the execution of the whole program, which requires an understanding of the underlying algorithm and program execution. Questions can be asked which enable the students to discuss individual snippets of codes, such as that shown in Figure 3. Discussion of the question should ideally take place in pairs or groups to enable students to develop the vocabulary they need to talk about the program (Cutts et al., 2012).

What do you think the mystery() function does?

```
from turtle import *

def mystery():
    fillcolor("Green")
    begin_fill()
    pencolor("Red")
    forward (100)
    right (90)
    forward (100)
    right (90)
    forward (100)
    right (90)
    forward (100)
    end_fill()

mystery()
```

Draw in here – label colours




Figure 2. A predict activity from one of the first lessons.

6. What does this line do?

```
toppings = toppings + " and " + next_topping
```

.....

.....

Figure 3. Sample question in the investigate phase.

Macro Structure	Overall structure of the program text	Understanding the algorithm of the program	Understanding the goal/purpose of the program in context
Relations	References between separate blocks	Sequence of related actions	Understanding how subgoals are related to overall goals
Blocks	Regions of interest-blocks like control structures or groups of adjacent statements	Operation of a block	Function of a block, maybe seen as a sub-goal
Atoms	Language elements	Operation of a statement	Function of a statement. Goal only understandable in context
	Text Surface	Program Execution (data and control flow)	Functions/goals of the program
	Structure		Function

Figure 4. The Block Model (adapted from Schulte, 2008).

4.4. Modify and make

In this phase of the lesson, the learners are able to build on the existing program to modify and create new programs. Carefully structured activities allow progression from simple changes to more substantial functional changes to the program. Having an existing program in place gives the student confidence and something to build on. Sometimes the *modify* task is to remove obvious glitches with the program. For example, one of the programming examples includes a programme to allow toppings to be added to a pizza. On running the program, students discover the output always ends with the word “and” so following this a *modify* task may be to improve the program so that the output does not end with “and”. Subsequently, in the *make* phase, the students will be asked to create a new program from a problem description, drawing on what they have learned about loops and string manipulation from the previous program (Cutts et al., 2012).

PRIMM could take one lesson for a simple concept, or there could be iteration around parts of PRIMM, or a number of lessons might be needed. It offers the teacher a structure to support students in gaining an understanding of core programming concepts.

We developed resources to exemplify how PRIMM might work in the classroom. The resources are not themselves PRIMM, but are the *content* that illustrates to teachers how the PRIMM elements work. The resources were developed by the first author and used by teachers. In summary, PRIMM is a way of structuring programming lessons that focuses on:

- Reading code before writing code starting from code that is ‘out there’ (social plane)
- Working collaboratively with learners to talk about programs – focused discourse and interaction

- Reducing cognitive load by unpacking and understanding what program code is doing – small, scaffolded steps within learners' ZPD
- Gradually allowing learners to take ownership of programs when ready by moving the learner on to what they can do on their own

In order to evaluate PRIMM we carried out a study with the following two research questions:

- RQ1: To what extent does being taught using the PRIMM approach impact on learners' programming attainment?
- RQ2: To what extent do teachers find the PRIMM approach an effective method to use in school?

The study is described in the next section.

5. The study

5.1. Research design

The aim of the study was both to consider the effectiveness of PRIMM in the teaching of programming and to increase our understanding of its impact on learning. To answer the first research question the plan was as follows:

- Carry out a quasi-experimental study in a number of schools, assessing programming knowledge after several months' of PRIMM lessons (via a post-test). A baseline test was used to establish a starting point for both control and experimental groups.
- To gather data from teachers (qualitative) about their experience of the approach and its impact on student learning

To answer our second research question our plan was:

- To gather data from teachers (qualitative) about the impact of PRIMM on their confidence as teachers, and their use of the PRIMM resources

We employed a type of quasi-experimental design known as the non-equivalent control group post-test design (Campbell & Stanley, 1963). The specific research design is a post-test design, but it lacks the random assignment. Following Campbell and Stanley (1963), the treatment was administered by the school system and the untreated classroom was the control group. In accordance with this design, the control group consisted of students who were to take the same number of programming lessons, covering the same topics, but using the teaching method normally used in the school. The experimental group included students who learned computer programming from teachers who adapted their teaching methods according to the PRIMM approach. To ensure that students did not differ significantly in their computer programming ability, both groups were baseline tested before the start of the intervention. Control groups were not available in all of the schools in the study for practical reasons.

5.2. Teacher participants

Teachers were recruited through an open invitation on social media and through computer science forums. Potential participants were informed that they would be required to be involved in a school-based study of a new approach to teaching programming. Teachers who responded were asked if they had colleagues who could be control classes. This approach to finding control classes enabled us to reduce some of the differences between pupils as the students attended the same school and would have had similar prior learning experiences to the test group.

A requirement was made for teachers to attend a full day's training, and they had to identify in advance groups of students to be used for the study. Ethical processes were adhered to and teachers consented to our use of data for all aspects of the study. Teachers had to ensure that they had the school principal's permission to work with the students, and they were formally released from school for the training. During the training day, teachers worked with the PRIMM resources and their involvement as participants was clearly explained. Fourteen teachers were recruited, of which 13 continued to participate.

Teachers were given full sets of materials, including starter tasks, presentations, worksheets, starter programs and answers, for 10 lessons (including extension material) covering the basic programming constructs of sequence, selection and iteration in Python.

5.3. Student participants

The participants in the study were students at key stage 3, in years 7–9 (see [Table 2](#) for age). The number of students per teacher varied: some classes were very small with only 15 pupils others were large with over 30 pupils. Some teachers had multiple classes, for example, in School 2. In total, 493 students were included in the experimental group and 180 students were part of the control group.

As shown in [Table 3](#), the control groups were in four state schools. In these schools, more than one teacher taught computing in the year group and the second teacher, who had not been trained to use PRIMM, delivered programming lessons using the school's pre-existing lesson plan materials to the control class pupils. When asking teachers to organise control groups we requested that they find similar classes in their school to be engaged. Pupils in years 7, 8 and 9 have not elected to specially take computing; at this stage, computing is a mandatory subject for all and is therefore taught through general mixed ability classes. Therefore, there is no reason to assume that the demographics of pupils, such as their spread of ages, abilities and backgrounds would be significantly different between the PRIMM and control

Table 2. Teaching phases, highlighting the students in our study.

England & Wales			USA	Student age
Phase	Key Stage	Year Group	Pupil Grades	(Years Old)
Early Years	Early Years			4–5
Primary	Key Stage 1	Year 1	Kindergarten (K)	5–6
		Year 2	Grade 1 (K1)	6–7
Secondary	Key Stage 2	Years 3–6	Grades 2–5 (K2-K5)	7–11
	Key Stage 3	Years 7–9	Grades 6–8 (K6-K8)	12–14
			(middle school)	
	Key Stage 4	Years 10–11	Grades 9–10 (K9-K10)	15–16
	Key Stage 5	Years 12–13	Grades 11–12 (K11-K12)	17–18

Table 3. Participating teachers and classes (those with * were not included in analysis).

ID	School	Teacher gender	Year group	Students in PRIMM trial	Students in control group	Interviewed
1 *	Mixed State	M	8			
2	Mixed State	F	8	114	103	
3	Girls State	F	9	45	36	Y
4	Boys State	M	9	43		Y
6	Mixed State	M	8	36		Y
7	Mixed State	F	7	25		Y
8	Mixed State	M	9	38		Y
9	Mixed State	M	8 and 9	23		
10	Mixed State	F	8 and 9	21	15	Y
11	Girls Independent	M	8	46		Y
12	Mixed State	M	8 and 9	45		Y
13	Mixed State	M	9	23	26	
14	Girls Independent	M	9	34		Y
TOTAL				493	180	9

classes. To confirm homogeneity we designed and ran a baseline test that compared the control and intervention groups.

5.4. Study materials

We had previously designed and implemented a short pilot study to trial the PRIMM materials. This involved 6 teachers and approximately 80 students over 4–7 lessons, and was followed by individual interviews with all the teachers to consider their views of PRIMM and the materials. Our analysis of the interviews enabled us to further develop a set of materials, and to devise appropriate research instruments for the subsequent main study. The pilot demonstrated that we needed to provide more comprehensive materials, and that teachers had limited confidence and time to develop their own resources in the PRIMM style. Thus for this study, we produced a set of 10 complete lessons, with starter exercises, presentations, starter programs and worksheets (all editable by the teachers) covering the

topics of sequence, selection and iteration for beginner programmers.¹ These were shared with the teachers on the first training day.

5.5. Research instruments

5.5.1. Baseline and post-test

The baseline test was designed by one of the researchers and tested students' basic understanding of programming since they had limited experience. The post-test was designed by the same researcher and included 7 programming tasks covering the topics in the PRIMM resources (selection, loops, etc.). In terms of the test characteristics listed in Margulieux, Ketenci, and Decker (2019), the baseline test included seven multiple choice questions, one code explaining question, one code tracing question and one code testing question. The post-test included one Parson's puzzle-style problem question (1 point), one code explaining question (2 points), two code writing questions (2 points), two multiple choice questions (1 point each), and one fill in the gaps question (2 points). Examples of a baseline test question and a post-test question are shown in Figures 5 and 6, respectively. Content validity was ensured by the researchers and other experts reviewing the tests and agreeing that the items of the test measured the intended concepts (Effendi, Matore, & Khairani, 2015).

5.5.2. Interview design

Interview questions were trialled during the pilot study which enabled us to select questions that informed our research questions effectively. A semi-structured interview design was used, and questions were grouped into four sections:

Question 8
Which of the following would make this statement TRUE?
`(tries < 3) and (password == "secret")`

- A) `tries = 3 password = secret`
- B) `tries = 2 password = secrets`
- C) `tries = 4 password = Secret`
- D) `tries = 1 password = secret`

Figure 5. A baseline test question.

Question 10

This Python program is supposed to add together the numbers the user enters at the keyboard until they type 999. When the user enters 999 it should stop. The program should then print the total.

However the statements are in the wrong order! Use the up and down arrows at the end of each line to put the program into the right order.

```
def adding():
    total = 0
    number = 0
    while number != 999:
        total = number + total
        number = int(input("Enter the next number "))
    print(total)
```

Total mark: 1 point (all the statements must be in the right order)

Figure 6. A post-test question including answer and marking criteria for the teacher.

- Section A General information about sessions with PRIMM
- Section B Impact of PRIMM on students programming skills
- Section C Impact of PRIMM on teachers confidence
- Section D Use of PRIMM resources and the future

5.6. Method

The baseline test was sent by email to all the computing teachers who administered the test on paper with their students. The teachers were responsible for marking their students' responses to the test. Specific marking guidelines were sent to the teachers along with the baseline test. We should note here that it was quite easy to mark these tests as all the questions were multiple choice and each correct answer contributed one mark to the final score. Both control groups and experimental groups completed the baseline test, in order to establish that they were equivalent groups.

At the end of the course and after almost six months of programming instruction, we administered the post-test to both groups. The post-test was administered in an online form and the link to the test was emailed to the teachers. Some of the teachers marked their students post-tests but others preferred to let the researchers mark the tests. We created specific marking guidelines for the test which were sent to all teachers to ease the assessment procedure ensuring a valid and reliable way of marking; we moderated a sample of the teachers' marking. In total 12 of the teachers provided data for use in the quantitative part of the study. At the end of the study and when the post-tests had been completed, teachers submitted their journals, were invited for interview and invited to an online focus group.

5.7. Threats to validity

Quasi-experimental designs have some validity concerns that researchers must take into consideration. Threats to internal validity include history, maturation, testing, instrumentation, statistical regression, selection, attrition, and diffusion. Although we cannot say much about history and therefore the extraneous variables that may have influenced the results, maturation was controlled by having students of the same age and class in both groups. To control for the selection bias we included a baseline test at the start of the study which tested students' ability in programming before the intervention took place. Testing and instrumentation were controlled by not having the same questions in the baseline test and in the post-test. The baseline test was used to ensure that the two groups did not differ significantly before the intervention and thus the statistical regression is not a problem in this design. Concerns about diffusion were controlled as we did not have any evidence of

communication between the control & experimental group, and attrition was minimal.

6. Data analysis

6.1. Quantitative

To examine if PRIMM had any effect on students' learning, we compared the post-test score of the experimental group with that of the control group. The data comprised the students' baseline test and post-test scores as well as their identifying class and teacher. SPSS² was used for the data analysis. Students' data were grouped depending on whether they belong to the experimental or the control group. Before employing any statistical test, we checked if the dependent variable was normally distributed. The test that we employed for this was the Shapiro–Wilk test for normality. The results indicated that our dependent variable (baseline test and post-test scores per group) was not normally distributed in any case. Therefore, the test we employed for comparing differences between our two independent groups was the nonparametric Mann–Whitney U test. We first used this test to examine if our two groups had significant differences in the baseline test score. As soon as this pre-condition was tested and assured we then moved on to compare the two groups' post-test scores. The results of this test are described in [Section 7.1](#).

6.2. Qualitative

Nine of the 13 teachers were interviewed as shown in [Table 3](#). The selection of those interviewed was based on a spread of male/female, ages taught and mixed and single-sex schools. Three of the four female teachers (75%) & six out of nine male teachers (66%) were interviewed. The boys only school teacher was interviewed, all three girls schools' teachers and five of the mixed schools' teachers were interviewed. The only year 7 teacher, 4 of the year 8 (50%) and 5 of the year 9 (55%) teachers were interviewed. All interviews were conducted online, were audio recorded and transcribed. The length of time of the interviews varied from 26 to 68 min. Four took between 26 and 36 min, three between 46 and 56 min and one took 68 min.

Two of the authors worked on the coding process. A thematic qualitative data analysis (QDA) approach was used to analyse the transcribed interviews and outcomes of tasks based on the methodology detailed by Kuckartz (2014). NVivo³ was used to support the process of coding text segments.

In the second study, we started by coding inductively from the interviews (Mayring, 2000). The overall objective at this point was to create main themes which would lead to a structure for reporting which would not be pre-

determined by any initial constraints. One author coded two of the interviews adding and amending categories and sub-categories inductively. After this first pass of coding, we reviewed and revised the resultant categories to confirm they matched the data coded. Two interviews provided approximately one-fifth of the overall transcripts in line with recommendations from Kuckartz (2014) of 10–20% for the first pass. Following this, all interviews were coded. Emergent patterns were recognized and new codes created to hierarchically group codes. This process was repeated across the categories creating, merging and splitting codes inductively (Kuckartz, 2014; Mayring, 2000). Once the more elaborate category system had been created, we checked that all data adhered to the new coding structure and recoded as necessary (Kuckartz, 2014). A second researcher then coded three of the nine interviews (33% of the text) a second time, with a Cohen's Kappa reliability score of 0.75, which is considered as good agreement between researchers.

Throughout the coding process, a consensual coding approach was used (Kuckartz, 2014). As categories and sub-categories were developed detailed definitions were maintained to support the category review and second coding. There was minimal disagreement in category review and coding but where this occurred the two researchers discussed the differences and resolved through changes to definitions or coding as necessary.

All teachers were offered the opportunity to take part in an online focus group to discuss general experiences. The focus group was audio recorded, transcribed and coded. Teachers 3, 8 and 14 took part. Teachers 2, 4, 6, 7, 8, 10, 12 and 14 also provided written notes of evaluation of their delivery of PRIMM lessons. These have also been coded. All interviews and focus groups were completed in the spring of 2018.

7. Results

7.1. Results – quantitative

All classes as shown in Table 3, except those of Teacher 1, were included in the quantitative analysis. Teacher 1's classes were not included as he had used a local school test rather than the study post-test. Table 3 depicts the descriptive information for the control and the experimental group for the baseline test and the post-test.

7.1.1. Differences before the intervention

Differences before the intervention were designed to show parity of the control and experimental groups. As Table 4 shows, in the baseline test, the mean and the median scores for the control group are 4.58 and 4.0, respectively, while for the experimental group the scores are 4.89 and 5.0. To test if there are significant differences between our two groups at the beginning of the intervention, the

Mann – Whitney test was employed since the data were not normally distributed. The null hypothesis that was tested was the following:

H01: there is no significant difference between the scores of the experimental and the control group in the baseline test

The results in Table 4 indicate that the students' performance on the baseline test and post test for all students did not differ significantly ($p.05$) and, thus, we could not reject the null hypothesis ($Z = -1.891$, $p.05$).

7.1.2. Differences after the intervention

Differences between the control and experimental groups after the programming lessons were examined to see if the PRIMM lessons had had an impact on programming attainment. Table 3 indicates that in the post-test, the experimental group ($M = 3.284$, $MD = 3.0$) scored higher than the control group ($M = 2.57$, $MD = 2.0$). To investigate if these differences are statistically significant, we test the following null hypothesis:

H02: There is no significant difference between the score of the experimental and the control group in the post-test.

The Mann and Whitney test (Table 5) employed showed that there is a statistically significant difference in the score between the control and experimental groups for all students in favour of the experimental group. This suggests that the experimental group scored statistically significant higher than the control group in the post-test tasks ($Z = -3.392$, $p.05$, $r = .13$) (r = effect size). The r value is the effect size and means that mean that 13% of the variance between groups was due to the intervention. From these results, we are able to reject the null hypothesis.

Table 4. Descriptive statistics.

Control	Baseline test	Posttest	Experimental	Baseline test	Posttest
Mean	4.58	2.575	Mean	4.89	3.284
Median	4.0	2.0	Median	5.0	3.0
Std. Deviation	1.911	2.1916	Std. Deviation	2.115	2.510
Skewness	.548 (se = .181)	1.237 (se = .181)	Skewness	0.88(se = .110)	.873(se = .110)
Gray Kurtosis	.301 (se = .360)	1.314 (se = .336)	Kurtosis	-.389(se = .220)	.301(se = .220)

Table 5. Mann-Whitney – comparing control & intervention groups – baseline test & posttest.

	baseline test	Posttest
Mann-Whitney U	40,195.0	36,822.5
Wilcoxon W	56,485.0	53,112.5
Z	-1.891	-3.392
Asymp. Sig.(2-tailed)	.59	.001

Table 6. Summary of codes.

Theme	% segments coded to this theme
1 Implementation of PRIMM in class	10%
2 Skills needed by learners	4%
3 Stages of PRIMM lessons	21%
4 The impact and particular aspects of PRIMM	26%
5 Differentiation and groups of learners	12%
6 Adaption and future use of PRIMM	13%
7 Feelings of motivation of teachers & learners	14%

In summary, these results indicate that the control and experimental groups were comparable before the study, and that after the intervention, the experimental group scored higher on the post-test than the control group. This indicates that the PRIMM lessons had a favourable impact on learner outcomes. We draw on the qualitative results to further support this conclusion.

7.2. Results: qualitative

Through an iterative coding process, seven key themes emerged from the data, as shown in Table 6, which shows the themes and the percentage of coded segments for each theme. Here we summarise and illustrate the themes and highlight the teachers' perspectives in so far as they enable us to understand the impact of the PRIMM method.

7.2.1. Implementation of PRIMM in class

The comments coded in this theme related to how the study was implemented in class by teachers and included the number and age of learners in the study and any control groups, number of lessons taught, PRIMM topics taught, length of PRIMM topic (lesson), test information, teachers involved in the study and how programming is taught.

7.2.2. Stages of PRIMM lessons

In their interviews, all nine teachers talked about every stage of the PRIMM lesson, with the most frequently mentioned stage being *Predict* with 28% of the coded segments. Followed by, *Modify* with 22%, *Investigate* with 21%, *Make* 12% & *Run* with 11%.

There were many comments about the advantages of having a structure to the lessons by following different stages of PRIMM, for example:

"The PRIMM, the reading and then the running and then the investigation was a really nice structure for them to follow. And because they're the same every week, they could see what was going to be coming." (Teacher 6)

Teacher 11 highlighted the impact of *predict* on all his students (all girls):

"Predict was really successful with all students at all levels, because ... during the sharing stage of the process, those that haven't really picked it up, because they didn't quite get the logic or they weren't such good readers, would feed off other people's ideas and develop their own thoughts, and you could see the impact of it really clearly as a teacher" (Teacher 11)

Six teachers alluded to a "modify ceiling", indicating for a variety of reasons that some learners did not get beyond the modify part of the lesson:

"I think a lot of kids didn't get on to make and that's why I made the sessions two lessons long to make sure that enough students got on to make. But even then, many students didn't get on to make." (Teacher 4)

Here we can see the distinction between the **structure** of the PRIMM lessons which teachers liked, and some of the **content** of the materials, which for some classes were rather advanced and lengthy. Amongst our schools, there was a wide range of types of schools and students' prior experience, despite our attempts to control for this in study design. In addition, some schools have more hours of computing each week which was another factor affecting the way that the content within the materials was received.

7.2.3. The impact of aspects of PRIMM

Six sub-themes emerged in the coding: progress made, speaking and listening, concepts and key skills, task design, expectations of students and starter programs. These are exemplified in Table 7.

Several teachers described how previously students had been taught to write code straight away whereas reading code first had really assisted the middle and lower ability students. Other teachers described the routine nature

Table 7. Examples of the sub-themes emerging around the impact of PRIMM.

Sub-theme	Example
Progress made by learners	<i>I think it's the whole PRIMM approach. I think it was the predict and the run and not just the modify, it just allowed them to get that concept a lot faster.</i> (Teacher 10)
Speaking and listening	<i>There was certainly more active talking and planned talking about the programming because of the way that the questions are worded in the worksheets and the resources</i> (Teacher 6)
Concepts and key skills	<i>When I taught functions prior to using PRIMM, I'd have to break each and every single part down. But with the PRIMM lessons the way it was structured, it was very easy to follow, and I think the kids understood it quite quickly as well.</i> (Teacher 12)
Task design	<i>"Modify" was reinforcing the understanding that they'd reached to get to that point .. the student is armed with much more knowledge when they attempt a task – it's all in the immediate confines of that lesson</i> (Teacher 14)
Expectations of students	<i>We've always got the learners to explain and to comment their code, but here it was done first. Right from the beginning it was, no, you need to be able to read this, and the expectation was far, far higher.</i> (Teacher 6)
Starter programs	<i>..having some code they could just run, and it gets it basically working, encourages them a little bit more than having to type it all in from scratch .. They've got a starting point from which they then could develop.</i> (Teacher 8)

of the PRIMM lessons giving students confidence and familiarity with a process. Teachers were generally positive about the progress made by students, and were aware that the material used within the PRIMM structure generated high expectations:

"I mean we're expecting a lot from the kids in those 14 hours, those 14 sessions. And for some of the kids, they coded functions, if statements, nested if statements, while loops, concatenation, casting. To get that in 14 hours in Year 9 is remarkable."(Teacher 4)

All teachers mentioned a speaking and listening aspect of PRIMM which was coded into different sub-themes around pupil-pupil communication, teacher-pupil communication, vocabulary, social aspects and the quality of the talk. Language was a very frequent topic in the interviews.

All teachers mentioned some form of paired and group talk. This talk was sometimes an implicit aspect of the structure of PRIMM stages or resulted from the specific questions in the content of the PRIMM resources. Talk was often linked to collaborative work:

"I noticed a big difference in terms of the girls collaborating with each other and trying to sort of out each other's problems. Usually that's been confined to one or two girls who feel quite confident, but with PRIMM with the tasks that they were doing, they always felt that they were closer to a solution that they might necessarily have felt in the past when I've taught them... So they were actively engaging in helping each other and looking at each other's code and making suggestions" (Teacher 14)

One teacher mentioned how content repeated across lessons supported learning, another that the pace of lessons was maintained by the repetition of content across a sequence of lessons. Jigsaw puzzle pieces were suggested as an analogy by another:

"So sometimes the third lesson would be the make section and during that section, I was teaching them how to magpie bits of code. So to see where the solution had worked in a similar domain and then copy and paste parts of it to almost use it like a jigsaw puzzle to put together a solution to a problem, and encourage them that that was okay. They could go and find bits of different problems, borrow it and put it together." (Teacher 7)

Five of the teachers liked the gradual addition of complexity and chunking up the learning. Teacher 14 explained:

"PRIMM gave them the opportunity to edit code and enjoy success very quickly, and that fed onto the next bit when you're looking at the next bit of code and how you add some more complexity to it... It starts off as a small snippet, which they understand, they get, and then they develop it." (Teacher 14)

He went on to compare PRIMM with the way he used to teach programming:

"I'd give them the whole code with conditions in there straightaway, and that might work for one or two of them, but be a sea of confusion for a lot of others. PRIMM

introduced them to programming in a much more systematic and measured way.”(Teacher 14)

7.2.4. Differentiation and groups of learners

As discussed in [Section 2.3.5](#), differentiation refers to the way that teachers adapt their teaching to support different groups of students particularly those who are lower ability or higher ability. Within this theme, there were a variety of approaches to differentiation mentioned. Some examples are given below to summarise this theme, but the striking element of data coded here is how much the teachers reflected on their teaching of programming and the adaptations needed to reach students of a range of abilities.

Teachers mentioned the use of a wide range of different methods to differentiate:

“I think this (PRIMM) has helped me to think more creatively about being more inclusive and adapting, for those lower ability students.” (Teacher 7)

The open-ended nature of the content was highlighted as being very important, particularly for higher ability students. This was mentioned by teachers 3, 8 and 12. Teacher 12 explained:

“the differentiation is already there. So, not all kids have to complete every single task. They can be doing the first couple of tasks. And if their ability allows them to move forward, that’s great. But if that’s where they get to, that’s also great.” (Teacher 12)

Eight teachers mentioned they supported learners with explanations through responses to questions or small group coaching and through modelling what they required students to do. Here teachers were explicitly using their own expertise, as a *“more knowledgeable other”* (MKO) (Vygotsky, 1978).

The role of the learning support assistant (LSA) to also help individuals and working with small groups was highlighted:

“it was differentiation through outcome and support, so I spent more time, and the LSA spent more time with those people, explaining. I’d also have small groups together towards the end, and we were getting a bit more meaty programming. And with those small groups, I would run through the program with them.” (Teacher 7)

This teacher also explained how modelling, motivation and linking to prior learning were required to differentiate the *make* stage:

“I think it (differentiation at the make) was probably through modelling and through encouragement. How would I characterise that? Using what they’d already done and could understand and applying it to a new problem, I guess. Trying to make it transparent that, just because you solved this problem this way, there were elements of that solution that could fit this new solution, and trying to make that really transparent and clear.” (Teacher 7)

Reading and writing skills were both mentioned by teachers. One teacher reflected on the relationship between reading and programming:

"If they can't read, then they're not going to be able to access what they need to do or realise that this word means something and it needs to obviously go into the computer in a particular way." (Teacher 6)

Teacher 4 linked the answering of *investigate* questions to familiarity with comprehension activities in other subject:

"And also, I found some kids were very happy going along through the Word document and filling in the gaps and getting all the right answers because they kind of felt it was similar to comprehension that they might do in say, Geography, or another subject where you do comprehension." (Teacher 4)

7.2.5. Adapting & future use of PRIMM

All teachers talked about how they adapted and used PRIMM in their classroom, with over 147 coded segments including the practical aspects of classroom management, differentiation, impact and aspects of PRIMM, and 42 coded segments focusing on the future use of PRIMM.

Changes and future uses included changing wording of tasks, *PRIMMing* other concepts and constructs, adding new questions for the *investigate* stage, using PRIMM with other year groups, adapting resources for exercise books and other formats, PRIMM being used by more teachers in their school, adding a hook or overarching context, increasing the design aspects of activities, simplifying tasks for younger and lower ability students.

Teacher 7 was very keen to have a *Pre-PRIMM vocabulary stage*, explaining:

"So I think an introduction section with vocabulary and the basics and then starting PRIMM would be good." (Teacher 7)

Teacher 10 wanted to change the frequency of teaching programming in general, having PRIMM embedded in more frequent but less long blocks of teaching, more time for *make* and a closer meshing of *investigate* and *modify*. She planned to do more discrete teaching on structure, particularly using the *investigate* stage, saying:

"I'm going to adapt this for the future, I will have discrete lessons on structure and make sure the investigate focus is on that structure." (Teacher 10)

One teacher explained that he was going to use PRIMM to increase pupil's take-up of computer science in later years.

"I'm going to argue with my colleague that there should be a term of PRIMM in each of years 7,8 and 9, because I think that will really crack it. I really do. It will really crack it in terms of our recruitment." (Teacher 14)

7.2.6. Feelings & emotions of teachers & learners

All teachers implied or directly mentioned both their own emotional response to PRIMM and that of their learners, with 70 coded segments for teacher

feelings and over 120 segments for pupil emotions. Teachers overall were very positive about their experiences with PRIMM, with some mentioning an increase in confidence and all but one clearly saying they were going to implement PRIMM as a pedagogical approach in their ongoing practice. One teacher explained:

"We'd never done reading before writing, we didn't think about that. Reading before writing was a really, really good revelation for us, and we thought it was really positive" (Teacher10)

One teacher mentioned that encouraging his colleagues to use PRIMM had built up the confidence of less experienced colleagues and another teacher said:

"I loved it because I just felt it made sense ... one of the challenges we've had previously is we've had booklets that they work through but because youve got these strugglers, racers, they would be working at such different speeds... So having that structure of each lesson [with PRIMM] I never felt that I was struggling to deal with the different abilities in the same way that I have done previously." (Teacher 3)

In terms of pupil emotional reactions, Teacher 3 explained how over time students become more confident with predicting, saying:

"There wasn't this big drama if they didn't get it (predict) right, because then we could run it and talk about it – 'this is why it's not what you thought it was'. It didn't knock their confidence." (Teacher 3)

Another teacher highlighted students being confident to share work which they hadn't been previously:

"Now a lot of those girls wouldn't have wanted to share their quiz in the past, because they might be convinced it wasn't going to work ... but everybody, in both sets, everybody did that task." (Teacher 14)

Other teachers described students being more confident to have a go. Having sample code to work with which enabled quick success was mentioned by Teacher 8 as he said:

"I knew that everyone would get some code running because they'd be given some code. And that would run and it would work. And I didn't have to spend half the lesson getting the typical problems out of the way. All the syntax and those other things, which get in the way of actually getting some success from learners to start with" (Teacher 8)

Teacher 11 explained how his students *"were really satisfied – lots of cheering when they managed to get one of the modify activities done, and they would share it with their friends."* (Teacher 11). However the same teacher described how students did still get frustrated if the material was too hard.

Speed of understanding was linked to enjoyment of the tasks, with a teacher commenting that the students had learned more quickly using this approach:

"I think it's the whole PRIMM approach. I think it was the predict and the run and not just the modify, it just allowed them to get that concept a lot faster." (Teacher 6)

Another teacher talked about a reduction in anxiety about getting things wrong, explaining that it *"didn't actually matter, because it actually gave us more to talk about and about why they thought it was wrong, and mistakes were completely acceptable and a normal part of computer science. Then, it was much more successful."* (Teacher 7). This is very encouraging as students are often very reluctant to make mistakes in computer programming (Sentance & Csizmadia, 2017). Finally, the elements of collaborative work built into the PRIMM structure where possible was associated with enjoyment of lessons, according to one of the teachers:

"That [paired and group talk] was different [from normal computing lessons], yes. It was the fact that they were talking and bouncing ideas off each other made it enjoyable and made it different." (Teacher 10)

Overall PRIMM was associated by the teachers with many positive emotions, particularly relating to the structure and style of activities. Students and teachers were more frustrated at times by some of the content within the materials, where it was too hard or lengthy, as will be discussed in the next section.

8. Discussion

We began this paper by asserting that a sociocultural perspective on learning would deepen our understanding of programming. We have now described in some detail our evaluation of the PRIMM strategies with teachers in mandatory computing classes for 11–14 year olds. We now return to the two research questions:

- RQ1: To what extent does being taught using the PRIMM approach impact on learners' programming attainment?
- RQ2: To what extent do teachers find the PRIMM approach an effective method to use in school?

8.1. Research question 1: impact on programming attainment

Our data show a difference in the control and experimental groups and indicates that those using the PRIMM lessons did better in the final test questions. This gives support to the structure of PRIMM as the results were

positive despite some reports about some of the content being too advanced or too lengthy. With more appropriately pitched content we might expect that the quantitative results would have shown stronger significance levels.

So how can this improvement in learner outcomes be explained? Teachers certainly reported that the familiarity of a routine with lessons starting with students discussing code was an advantage of the PRIMM method. Several teachers also specifically felt that reading code before having to write their own programs, as previously advocated (Lister et al., 2009), helped students' understanding as well as their confidence:

"I think they made more progress using this approach than the previous approach. My anecdotal evidence is individuals who struggled the first time back in September when we did it the old-fashioned way versus this time around using this approach. It really helped the people who like a structure and get comfortable once they know what's going to happen... the kids, the guinea pigs, have all got something out of this process. I'm actually convinced of that. And a few kids have got a lot. The ones at the top end have also got something out of it, but maybe not as much as some of the others." (Teacher 8)

Students with high ability may have previously had no problems learning to program in school, but the fact that teachers find that lower ability students find this approach beneficial implies that this is worth pursuing.

In terms of the content, teachers' feedback indicated that there were some issues with the content that we devised to go within the structure of PRIMM, which accelerated too quickly for some of the learners. In the training of the teachers, we had encouraged them to edit the content to meet the need of their learners, as long as they maintained the structure of PRIMM, but some teachers were very reluctant to do that, feeling that they would compromise the research. Two main difficulties emerged with the materials: firstly that there was too much material per lesson, and secondly that the difficulty of the *investigate* questions was too great for the learners.

In summary, teachers found that PRIMM was effective for a range of learners and we conclude that if the content is pitched correctly within learners' ZPD, PRIMM will impact learner outcomes.

8.2. Research question 2: do teachers find PRIMM effective?

Teachers used PRIMM in different ways. Although it was explained that their participation in the research meant they could adapt the content for their students several did not do this at all and pushed through material that was too advanced for their students. Others adapted the materials for their own students so that it included more scaffolding. Thus, some teachers were more aware of the need to work within their learners' ZPD than others.

The use of carefully constructed questions to guide student's developing understanding and relating to prior learning shows teachers being used as the MKO:

"So, I sort of scaffolded it for the lower ones. But I'd often be having a chat, asking them some questions to coax out maybe an answer from them. Or if they were struggling with getting the code extended, I'd refer them back to a previous lesson. So, remember the code from the last lesson, go and have a look at that, that's got some ideas on how to do this." (Teacher 8)

Some teachers adapted the material – as intended. Others didn't and we felt that more time spent on co-construction would have been better for some teachers. With more funding, holding workshops with time for teachers to write their own PRIMM materials including adapting appropriate content may be an effective solution.

A key finding from the study was around vocabulary in the PRIMM resources. Many of the teachers discussed the use of vocabulary and the way that the investigate questions ensured their students used technical terms. This not only highlights the use of language to explain what the program is doing functionally, but consistent use of technical words for programming constructs starts to give the students tools to talk accurately about their programs. The emphasis on the importance of the right vocabulary and use of language was very evident from our data.

Some teachers found the *investigate* part the hardest for their students to do, while others found the *predict* part the most difficult. Teachers generally found it helpful where the resource included content that revisited topics. The formative assessment opportunities of the *predict* stage led Teacher 3 to be able to target different student needs across classes and focus on discussion in the lesson. This highlights the importance of language as mediation:

"I found that I then had a different discussion, depending on what they came up with. So for example, one class might have got the predict part completely right whereas another class might have gone off on a bit of a tangent and come up with different responses. So it varied how I then taught that section or how much detail I went into with the starter, depending on what they've said in response to the predict." (Teacher 3)

Implementation issues exist in school that affect how methods can be effectively used. One of the teachers worked in a school where policy was that the starter exercise was done in silence, so this made it impossible to do the PRIMM predict task in pairs at the beginning of the lesson. Other factors that affected the implementation were the level of the content we provided, an issue we have raised earlier.

We found that teachers were enthusiastic about the extent to which PRIMM helped them teach programming. They were able to describe what they were doing using the language of PRIMM, using terms like *investigate*, *modify* and *predict* that they would not have used before. Being part of the PRIMM trial

had enabled them to reflect on how they were teaching programming, and we would suggest that any teacher using PRIMM would necessarily start to reflect on how they were teaching programming and start using a vocabulary for strategies they were using.

The data revealed that teachers had found PRIMM gave them more confidence in teaching programming, as well as giving them strategies for use in the future:

"I think I'm quite confident anyway, but I do think that it helped me in terms of understanding how they learn how to code, if that makes sense. How to adapt, to be more inclusive". (Teacher7)

During the research study, we were fortunate to be working with a dedicated and reflective group of teachers who made valuable contributions to our research. The implications for this in terms of future implementation are that PRIMM may need additional structuring to be accessible to teachers who have other priorities or less time to reflect in such depth.

Overall we believe that the PRIMM method has given teachers in the study some strategies to use in their classroom to teach programming and increased their PCK. Teachers need a range of strategies in a toolkit of approaches to use and this may be a useful one for many teachers in the future. In particular, we felt the PRIMM method gave teachers a language to talk about the way that they were teaching programming, and this increased their ability to reflect on how their students were learning.

A surprising outcome of the study was that several of the teachers went on to deliver professional development sessions to other teachers in their networks about PRIMM, and wrote about its use on social media. Subsequently many other teachers that we have come across on an ad hoc basis have started to develop their own resources using PRIMM and this gives us additional, although informal, evidence that it is an approach that teachers like to use.

8.3. Limitations of the study

We were fortunate to be able to trial the PRIMM lessons on a large group of nearly 500 students across England, but there are some improvements that could have been made to the research design that would have made the results more reliable and conclusive. Firstly, although they had been partially piloted, the resources still contained too much content per lesson. Secondly, the teachers were not representative of all computing teachers in England – having responded to a call to participate. We would like to repeat the study with a more representative sample of teachers. Thirdly, we did not have access to programming assignments which could be classed as validated instruments, as recommended by Margulieux et al. (2019), so we had to rely on expert guidance to ensure the appropriateness of these instruments. Fourthly, we did

not capture any metadata on the demographics of either test or control classes other than numbers and gender. There were no differences found with respect to gender so we have not reported that data here. Finally, we were not able to control the lessons received by the control group, as teachers use different schemes of work for teaching the programming concepts in Python, although we know that these will be from a small subset of options.

8.4. Further work

In attempting to answer the research questions, we are aware that there is more work to be done. We would next like to consider how to improve the investigative stage, perhaps with a stronger focus on the Block model. In order to develop learning within the ZPD, a more informed and detailed understanding of easy/hard concepts is needed (Wiegand et al., 2016), so understanding progression in programming education is an area where much more research is needed.

For future trials, we hope to develop more co-construction of materials with teachers, with more differentiation built in, with the same PRIMM structure. Materials must continue to be editable as students in the same stage of their education may vary, as might be expected, in their ZPD (Vygotsky, 1978). Resource development requires an in-depth understanding of programming and can be structured around the different levels of the block model (Schulte, 2008) (as our materials were) with varied questions being asked that referred to atoms of code, blocks of code and then whole sections of code and questions that referred to both structure and function. A further piece of work might be to support teachers in writing material around the block model structure, and then teaching it with a PRIMM structure, thus enabling teachers to distinguish between content and structure more clearly.

In our work, we have focused on programming only, without considering algorithm planning and design, and this is an area that we could develop further. The research focused on text-based programming using Python, but we have anecdotal evidence that some teachers are using this approach with block-based programming, which is an avenue for investigation. Finally, alongside this work, there is a need to embed the theoretical framework surrounding our approach more specifically, clarifying the different roles of different types of language as mediating activity in PRIMM lessons.

9. Conclusion

We have presented the PRIMM model for teaching programming, which seeks to recognise the need for mediation in the learning process, following the Vygotskian concept of mediation as tools, psychological tools including language, and human beings (Kozulin & Presseisen, 1995). PRIMM draws on

existing work in computer science education relating to the teaching of programming, while rooted in sociocultural theory. Key to our approach is having the program available in the social plane for students to work with and understand, which reduces some of the initial obstacles in learning to program. As the difficulties of novices learning to program are well documented (Sorva, 2013), within our PRIMM model we have built upon the rich seam of work on supporting development of code comprehension skills and moving learners from the use of others programs to independent creation of their own. Drawing on some excellent work from other researchers (Lee et al., 2011; Lister et al., 2004; Schulte, 2008) we have developed an approach which we believe is directly usable and effective in the classroom.

As well as presenting PRIMM, we have evaluated it through a mixed-methods study. Our research has shown that classes using PRIMM for 10 or more lessons do significantly better in a post-test than those in a control group. It has also demonstrated that teachers are able to use the approach in secondary school and that classroom implementation of our approach is viable. We have demonstrated that teachers have welcomed the PRIMM approach in their classes because they feel it offers routine, accessibility for lower ability students, and an enriched understanding of core programming constructs.

It is important that all children leave school with the digital skills they will need to contribute to our technology-driven world; of these, computer programming skills are becoming increasingly important. Until the inclusion of computer science as a mandatory subject, students, in informal and formal settings, were required to choose, at some level, to be involved with learning to program. These students had some degree of motivation or aptitude as they elected to take the subject. In the context of the mandatory teaching of programming, this situation has changed, and we should be focused on making computing accessible as well as available. We will not be able to bring programming to all students with any level of integrity unless we explore how to do this effectively from an educational point of view. The PRIMM method demonstrates the relevance of socio-culturally-inspired approaches to the teaching of programming. PRIMM supports the development of teacher's pedagogical content knowledge (Shulman, 1986) in which the mediating role of the teacher as the "more knowledgeable other" is seen as key. Teachers are crucial to whether students will learn programming in school or not, so teachers need both confidence and resources that embed research-led ways of teaching programming. We very much hope that researchers will continue with similar research endeavours to support children and teachers in the learning of programming.

Notes

1. Now available at <https://primming.wordpress.com/2018/08/23/primm-materials-2018/>.
2. <https://www.ibm.com/analytics/spss-statistics-software>.
3. <https://www.qsrinternational.com/nvivo>.

Disclosure statement

No potential conflict of interest was reported by the authors.

Notes on contributors

Dr Sue Sentance is the Chief Learning Officer at the Raspberry Pi Foundation and a Visiting Fellow at King's College London. Sue's research is around pedagogy for computer programming and teacher professional development in computing, and she has previously worked both as a computing teacher and teacher trainer. She currently leads a team developing content for a government-funded in-service teacher support programme for computing.

Jane Waite is a computer science education researcher, teacher trainer and school resource developer who currently works and studies at Queen Mary University of London. Jane has twenty years of experience in the IT industry and ten years of experience as a primary school teacher.

Maria Kallia is a research student at King's College London. Her My PhD is in the field of Computer Science Education and my research interests focus on the pedagogy of computer programming. Prior to her PhD, she worked as a computing teacher for six years in Greece.

ORCID

Sue Sentance  <http://orcid.org/0000-0002-0259-7408>

References

- Alexandron, G., Armoni, M., Gordon, M., & Harel, D. (2014). Scenario-based programming: Reducing the cognitive load, fostering abstract thinking. *in Companion Proceedings of the 36th International Conference on Software Engineering*, New York, NY, USA: ACM, ICSE Companion 2014, 311–320, 00023.
- Armoni, M. (2013). On teaching abstraction in computer science to novices. *Journal of Computers in Mathematics and Science Teaching*, 32(3), 265–284.
- Basawapatna, A. R., Repenning, A., Koh, K. H., & Nickerson, H. (2013). The zones of proximal flow: Guiding students through a space of computational thinking skills and challenges. *in Proceedings of the Ninth Annual International ACM conference on International Computing Education Research*, San Diego, California, ACM, 67–74.
- Bennedsen, J., & Eriksen, O. (2006). Categorizing pedagogical patterns by teaching activities and pedagogical values. *Computer Science Education*, 16(2), 157–172.
- Brown, A. L., Ash, D., Rutherford, M., Nakagawa, K., Gordon, A., & Campione, J. C. (1993). Distributed expertise in the classroom. In Salomon, G. (Ed.). (1993). *Distributed cognitions*:

- Psychological and educational considerations (pp. 188–228). Cambridge UK: Cambridge University Press.
- Busjahn, T., & Schulte, C. (2013). The use of code reading in teaching programming. in *Proceedings of the 13th Koli Calling International Conference on Computing Education Research*, New York, NY, USA: ACM, Koli Calling '13, 3–11.
- Cajander, Å., Daniels, M., & McDermott, R. (2012). On valuing peers: Theories of learning and intercultural competence. *Computer Science Education*, 22(4), 319–342.
- Campbell, D. T., & Stanley, J. C. (1963). Experimental and quasi-experimental designs for research. In Gage, N. L., *Handbook of research on teaching*. Rand McNally, Chicago (pp. 171–246).
- Caspersen, M. E. (2018). Teaching programming. In S. Sentance, E. Barendsen, & C. Schulte (Eds.), *Computer science education: Perspectives on teaching and learning in school* (pp. 109–130). London: Bloomsbury Academic.
- Caspersen, M. E., & Bennedsen, J. (2007). Instructional design of a programming course: A learning theoretic approach. in *Proceedings of the Third International Workshop on Computing Education Research*, New York, NY, USA: ACM, ICER '07, 111–122.
- Clear, T. (2012). The hermeneutics of program comprehension: A 'Holey Quilt' theory. *ACM Inroads*, 3(2), 6–7.
- 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. in *Proceedings of the Fourteenth Australasian Computing Education Conference - Volume 123*, Darlinghurst, Australia, Australia: Australian Computer Society, Inc., ACE '12, 77–86.
- 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. in *Proceedings of the Ninth Annual International Conference on International Computing Education Research*, New York, NY, USA: ACM, ICER '12, 63–70.
- Daniels, H. (2016). *Vygotsky and pedagogy*. London, UK: Routledge.
- Denny, P., Luxton-Reilly, A., & Simon, B. (2008). Evaluating a new exam question: Parsons problems. in *Proceedings of the Fourth International Workshop on Computing Education Research*, New York, NY, USA: ACM, ICER '08, 113–124, 00090.
- Diethelm, I., & Goschler, J., (2015). Questions on spoken language and terminology for teaching computer science. in *Proceedings of the 2015 ACM conference on innovation and technology in computer science education*, ITICSE '15, Vilnius, Lithuania.
- Du Boulay, B. (1986). Some difficulties of learning to program. *Journal of Educational Computing Research*, 2(1), 57–73.
- Effendi, M., Matore, E. M., & Khairani, A. (2015). Assessing content validity of IKBAR among field experts in Polytechnics. *Australian Journal Basic Application Science*, 7, 255–257.
- Garneli, V., Giannakos, M. N., & Chorianopoulos, K., 2015. Computing education in K-12 schools: A review of the literature. in *Global Engineering Education Conference (EDUCON), 2015 IEEE*, IEEE, 543–551, 00017. Tallin, Estonia: IEEE.
- Giest, H., & Lompscher, J. (2003). Formation of learning activity and theoretical thinking in science teaching. In Kozulin, A., Gindis, B., Ageyev, V. S., & Miller, S. M. (Eds.), *Vygotsky's educational theory in cultural context* (pp. 267–288). Cambridge, UK: Cambridge University Press.
- 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*, New York, NY, USA: ACM, SIGCSE '17, 267–272, 00040.
- 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), 199–237.

- Gujberova, M., & Kalas, I., (2013). Designing productive gradations of tasks in primary programming education. in *Proceedings of the 8th Workshop in Primary and Secondary Computing Education*, New York, NY, USA: ACM, WiPSE '13, 108–117, 00012.
- Guk, I., & Kellogg, D. (2007). The ZPD and whole class teaching: Teacher-led and student-led interactional mediation of tasks. *Language Teaching Research*, 11(3), 281–299.
- Hermans, F., Swidan, A., & Aivaloglou, E. (2018). Code phonology: An exploration into the vocalization of code in *Proceedings of the 26th Conference on Program Comprehension*, 308–311. Gothenburg, Sweden: ACM.
- Hertz, M., & Jump, M. (2013). Trace-based teaching in early programming courses. in *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, New York, NY, USA: ACM, SIGCSE '13, 561–566, 00023.
- 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. *Transactions on Computing Education*, 14(2), 7, 1–7:9, 00041.
- Jenkins, T. (2002). On the difficulty of learning to program. in *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*, Loughborough University, UK, 53–58.
- Kasto, N., (2016). *Learning to Program: The development of knowledge in Novice Programmers*, Ph.D. thesis, Auckland University of Technology.
- 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.
- Kotsopoulos, D., Floyd, L., Khan, S., Namukasa, I. K., Somanath, S., Weber, J., & Yiu, C. (2017). A pedagogical framework for computational thinking. *Digital Experiences in Mathematics Education*, 3(2), 154–171.
- Kozulin, A. (2003). Psychological tools and mediated learning. In A. Kozulin, B. Gindis, V. S. Ageyev, & S. M. Miller (Eds.), *Vygotsky's educational theory in cultural context* (pp. 15–38). Cambridge, UK: Cambridge University Press.
- Kozulin, A., & Presseisen, B. Z. (1995). Mediated learning experience and psychological tools: Vygotsky's and feuerstein's perspectives in a study of student learning. *Educational Psychologist*, 30(2), 67–75.
- Kuckartz, U. (2014). *Qualitative text analysis: A guide to methods, practice and using software*. London, UK: Sage.
- Kumar, A. N. (2015). solving code-tracing problems and its effect on code-writing skills pertaining to program semantics. in *Proceedings of the 2ITICSE '15*, New York, NY, USA: ACM, ITICSE '15, 314–319, 00005.
- Lahtinen, E., Ala-Mutka, K., & Järvinen, H. M. (2005). A study of the difficulties of novice programmers. *ACM Sigcse Bulletin*, 37(3), 14–18.
- Leach, J., & Scott, P. (2003). Individual and sociocultural views of learning in science education. *Science & Education*, 12(1), 91–113.
- Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., ... Werner, L. (2011). Computational thinking for youth in practice. *ACM Inroads*, 2(1), 32.
- Linn, M. C., & Dalbey, J. (1985). Cognitive consequences of programming instruction: Instruction, access, and ability. *Educational Psychologist*, 20(4), 191–206.
- Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., ... Seppälä, O., et al. (2004). A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin*, ACM, 36, 119–150.
- Lister, R., Fidge, C., & Teague, D. (2009). Further evidence of a relationship between explaining, tracing and writing skills in introductory programming. in *Proceedings of the*

- 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education, New York, NY, USA: ACM, ITiCSE '09, 161–165.
- Lopez, M., Whalley, J., Robbins, P., & Lister, R. (2008). Relationships between reading, tracing and writing skills in introductory programming. in *Proceedings of the Fourth International Workshop on Computing Education Research*, New York, NY, USA: ACM, ICER '08, 101–112.
- 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. 00303.
- Machanick, P. (2007). A social construction approach to computer science education. *Computer Science Education*, 17(1), 1–20.
- Margulieux, L., Ketenci, T. A., & Decker, A. (2019). Review of measurements used in computing education research and suggestions for increasing standardization. In *Computer Science Education*, 29(1). doi:10.1080/08993408.2018.1562145.
- Margulieux, L. E., & Catrambone, R. (2016). Improving problem solving with subgoal labels in expository text and worked examples. *Learning and Instruction*, 42, 58–71. 00015.
- Mayring, P. (2000). Qualitative content analysis. *Forum of Qualitative Social Research*, 1(2), 1–10.
- Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. M. (2013). Learning computer science concepts with scratch. *Computer Science Education*, 23(3), 239–264.
- Morrison, B. B., Margulieux, L. E., Ericson, B., & Guzdial, M. (2016). subgoals help students solve parsons problems. in *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, New York, NY, USA: ACM, SIGCSE '16, 42–47, 00025.
- Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York, NY: Basic Books, Inc.
- Pattis, R. E. (1990). A philosophy and example of CS-1 programming projects. in *Proceedings of the Twenty-first SIGCSE Technical Symposium on Computer Science Education*, New York, NY, USA: ACM, SIGCSE '90, 34–39.
- Pea, R. D. (1986). Language-independent conceptual bugs in novice programming. *Journal of Educational Computing Research*, 2(1), 25–36.
- Perkins, D., & Martin, F. (1986). Fragile knowledge and neglected strategies in novice programmers. In Soloway, E. & Iyengar, S. Sitharama (Eds.) *Empirical Studies of Programmers, 1st Workshop*, Washington, DC, (pp. 213–229).
- Perrenet, J., Groote, J. F., & Kaasenbrood, E. (2005). Exploring students' understanding of the concept of algorithm: Levels of abstraction. *ACM SIGCSE Bulletin*, 37(3), 64–68.
- 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.
- Petty, G. (2004). *Teaching today: A practical guide*. Cheltenham, UK: Nelson Thornes.
- Qian, Y., & Lehman, J. (2017). Students misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education (TOCE)*, 18(1), 1.
- Rahmat, M., Shahrani, S., Latih, R., Yatim, N. F. M., Zainal, N. F. A., & Ab Rahman, R. (2012). Major problems in basic programming that influence student performance. *Procedia-Social and Behavioral Sciences*, 59, 287–296.
- Resnick, M. (2007). All I really need to know (about creative thinking) i learned (by studying how children learn) in Kindergarten. in *Proceedings of the 6th ACM SIGCHI Conference on Creativity & Cognition*, New York, NY, USA: ACM, C&C '07, 1–6.
- Resnick, M. (2017). *Lifelong Kindergarten: Cultivating creativity through projects, passion, peers, and play*. Cambridge, MA: MIT Press.

- Robins, A. (2010). Learning edge momentum: A new account of outcomes in CS1. *Computer Science Education*, 20(1), 37–71.
- Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2), 137–172.
- Rubin, M. J. (2013). The effectiveness of live-coding to teach introductory programming. in *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, New York, NY, USA: ACM, SIGCSE '13, 651–656, 00041.
- Ryoo, J. J. (2013). *Pedagogy Matters: Engaging Diverse Students as Community Researchers in Three Computer Science Classrooms* Ph.D. thesis, UCLA.
- Schulte, C. (2008). Block model: An educational model of program comprehension as a tool for a scholarly approach to teaching. in *Proceedings of the Fourth International Workshop on Computing Education Research*, New York, NY, USA: ACM, ICER '08, 149–160.
- Schulte, C., Clear, T., Taherkhani, A., Busjahn, T., & Paterson, J. H. (2010). An introduction to program comprehension for computer science educators. in *Proceedings of the 2010 ITiCSE Working Group Reports*, New York, NY, USA: ACM, ITiCSE-WGR '10, 65–86.
- Sentance, S., & Csizmadia, A. (2017). Computing in the curriculum: Challenges and strategies from a teachers perspective. *Education and Information Technologies*, 22(2), 469–495.
- Sentance, S., & Waite, J. (2017). PRIMM: Exploring pedagogical approaches for teaching text-based programming in school. in *Proceedings of the 12th Workshop on Primary and Secondary Computing Education*, 113–114. Nijmegen, The Netherlands: ACM.
- Sentance, S., Waite, J., & Kallia, M. (2019). Teachers' experiences of using PRIMM to teach programming in school. in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 476–482. Minneapolis, MI: ACM.
- Shabani, K. (2016). Applications of Vygotsky's sociocultural approach for teachers professional development. *Cogent Education*, 3(1).
- Sheard, J., Carbone, A., Lister, R., Simon, B., Thompson, E., & Whalley, J. L. (2008). Going SOLO to assess novice programmers. in *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education*, New York, NY, USA: ACM, ITiCSE '08, 209–213, 00087.
- Shulman, L. (1986). Those who understand: Knowledge growth in teaching. *American Educational Review*, 15(2).
- Simon, Lopez, M., Sutton, K., & Clear, T. (2009). Surely we must learn to read before we learn to write!. In *Proceedings of the Eleventh Australasian Conference on Computing Education - Volume 95*, Darlinghurst, Australia, Australia: Australian Computer Society, Inc., ACE '09, 165–170.
- Sorva, J. (2013). Notional machines and introductory programming education. *ACM Transactions of Computing Education*, 13(2).
- Sorva, J. (2018). Misconceptions and the beginner programmer. In S. Sentance, E. Barendsen, & C. Schulte (Eds.), *Computer science education: Perspectives on teaching and learning in school* (pp. 171–186). London: Bloomsbury Academic.
- Sorva, J., Karavirta, V., & Malmi, L. (2013). A review of generic program visualization systems for introductory programming education. *Transactions on Computing Education*, 13(4), 1–15.
- Spohrer, J. C., & Soloway, E. (1986). Novice mistakes: Are the folk wisdoms correct? *Communications of the ACM*, 29(7), 624–632.
- Statter, D., & Armoni, M. (2016). Teaching abstract thinking in introduction to computer science for 7th graders. in *Proceedings of the 11th Workshop in Primary and Secondary Computing Education*, New York, NY, USA: ACM, WiPSCE '16, 80–83.
- Su, A. Y. S., Yang, S. J. H., Hwang, W. Y., Huang, C. S. J., & 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, 00000.
- Sudol-DeLyser, L. A., Stehlik, M., & Carver, S. (2012). Code comprehension problems as learning events. in *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education*, 81–86. Haifa, Israel: ACM.
- Sweller, J. (1994). Cognitive load theory, learning difficulty, and instructional design. *Learning and Instruction*, 4(4), 295–312.
- Teague, D., & Lister, R. (2014). Programming: Reading, writing and reversing. in *Proceedings of the 2014 Conference on Innovation and Technology in Computer Science Education*, New York, NY, USA: ACM, ITiCSE '14, 285–290.
- Tenenberg, J., & Knobelsdorf, M. (2014). Out of our minds: A review of sociocultural cognition theory. *Computer Science Education*, 24(1), 1–24.
- The Royal Society. (2017). After the reboot: Computing education in UK schools. Policy Report. Retrieved from <https://royalsociety.org/topics-policy/projects/computing-education/>.
- Tsai, C. Y., Yang, Y. F., & Chang, C. K. (2015). Cognitive load comparison of traditional and distributed pair programming on visual programming language. In *International Conference of Educational Innovation through Technology (EITT), 2015*, IEEE, 143–146.
- Vainio, V., & Sajaniemi, J. (2007). Factors in novice programmers' poor tracing skills. in *Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, New York, NY, USA: ACM, ITiCSE '07, 236–240, 00036.
- Van Merriënboer, J. J. G., & Krammer, H. P. M. (1987). Instructional strategies and tactics for the design of introductory computer programming courses in high school. *Instructional Science*, 16(3), 251–285, 00082.
- Van Merrinboer, J. J. G., & Sweller, J. (2005). cognitive load theory and complex learning. *Recent Developments and Future Directions, Educational Psychology Review*, 17(2), 147–177, 00000.
- Venables, A., Tan, G., & Lister, R. (2009). A closer look at tracing, explaining and code writing skills in the novice programmer. in *Proceedings of the Fifth International Workshop on Computing Education Research Workshop*, New York, NY, USA: ACM, ICER '09, 117–128.
- Vygotsky, L. S. (1978). *Mind in society*. Cambridge, MA: Harvard University Press.
- Vygotsky, L. S. (1981). The instrumental method in psychology. In J. V. Wertsch (Ed.), *The concept of activity in Soviet psychology* (pp. 134–143). Armonk, NY: Sharpe.
- Vygotsky, L. S. (2004). Imagination and creativity in childhood. *Journal of Russian & East European Psychology*, 42(1), 7–97.
- Waite, J., Curzon, P., Marsh, D., & Sentance, S. (2018). Comparing K-5 teachers' reported use of design in teaching programming and planning in teaching writing. in *Proceedings of the 13th Workshop in Primary and Secondary Computing Education*, WiPSCE '18, Potsdam, Germany.
- Walqui, A. (2006). Scaffolding instruction for english language learners: A conceptual framework. *International Journal of Bilingual Education and Bilingualism*, 9(2), 159–180.
- Wertsch, J. V., & Tulviste, P. (1992). LS Vygotsky and contemporary developmental psychology. *Developmental Psychology*, 28(4), 548, 00552.
- Wiegand, R. P., Bucci, A., Kumar, A. N., Albert, J. L., & Gaspar, A. (2016). A data-driven analysis of informatively hard concepts in introductory programming. in *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, ACM, 370–375, Memphis, Tennessee.
- Wood, D., Bruner, J. S., & Ross, G. (1976). The role of tutoring in problem solving. *Journal of Child Psychology and Psychiatry*, 17(2), 89–100.