Teachers' Experiences of using PRIMM to Teach Programming in School (Author Pre-Print)

SUE SENTANCE*, King's College London
JANE WAITE, Queen Mary University of London
MARIA KALLIA, King's College London

PRIMM is an approach to teaching programming at K-12 that facilitates the structure of lessons in a purposeful way. PRIMM stands for Predict-Run-Investigate-Modify-Make, and draws on recent research in programming education. In particular the PRIMM approach recognises that starting with existing code and being able to explain what it does gives novice programmers the confidence to write their own programs. Using the PRIMM approach, teachers can devise scaffolded and targeted tasks for students which helps engender understanding, particularly for those who may have previously struggled to understand programming concepts. In this techniques paper, we consider what PRIMM is, and the experiences that teachers have had of using the structure in the classroom. PRIMM materials have been trialled in schools in a study involving around 500 students aged 11-14. From interviews with nine participating teachers we have found that teachers particularly value the collaborative approach taken in PRIMM, the structure given to lessons, and the way that resources can be differentiated. We propose that PRIMM is an approach that could be adopted in all phases of programming education as well as in teacher training.

CCS Concepts: • **Social and professional topics → K-12 education**; **Computer science education**;

Additional Key Words and Phrases: K-12 education, programming education, K-12 teachers

ACM Reference Format:
Sue Sentance, Jane Waite, and Maria Kallia. 2019. Teachers' Experiences of using PRIMM to Teach Programming in School (Author Pre-Print). 1, 1 (January 2019), 13 pages. https://doi.org/10.1145/3287324.3287477

1 INTRODUCTION

The reformation of school computing in England has been a welcome development given that it is essential that we prepare all young people with the digital skills they need to fully participate in society. The introduction of new computer science concepts and skills into the curriculum, particularly in the area of computer programming, has been challenging for students and teachers, firstly because models for pedagogy are either not fully formed or shared with teachers, and secondly, because computer programming can be difficult to learn [24]. It is thus recognised that more research needs to be conducted with a focus on appropriate pedagogy [31].

Computing teachers benefit from access to proven teaching strategies and pedagogies relating to programming. Much research has been carried out in this area, mostly in higher education settings, but only recently in schools,

*The first author now works at the Raspberry Pi Foundation. Paper published by ACM at https://doi.org/10.1145/3287324.3287477

Authors' addresses: Sue Sentance, King's College London, London, UK, sue@raspberrypi.org; Jane Waite, Queen Mary University of London, London, UK, j.l.waite@qmul.ac.uk; Maria Kallia, King's College London, London, UK, maria.kallia@kcl.ac.uk.

Manuscript submitted to ACM

and this has not been widely translated into usable structures for teachers. Consequently, computing teachers are being called to deliver a challenging subject with insufficient knowledge of effective teaching strategies and on how to develop and enhance vital competencies to accomplish this task. To address these issues, we have developed and are evaluating a new pedagogical model for teaching and learning programming (PRIMM) [26]. PRIMM can be used to structure lessons and sequences of lessons with the following activities:

- **Predict** what code will do
- **Run** the code to test predictions
- **Investigate** the structure of code
- **Modify** the code to add functionality
- **Make** a new program using the same/modified structures.

PRIMM stands for **Predict**, **Run**, **Investigate**, **Modify** and **Make**. Using PRIMM, classroom activities can be designed which involve predicting the output of code, code comprehension and gradually making new programs. It is a method of teaching programming that counters the known problem of novices trying to write programs before they are able to read them [15]. It incorporates activities that scaffold learning for students and provides a structure for lessons.

In this techniques paper, we describe the rationale for this approach, exemplify it, and describe teachers' experiences. In a study of the PRIMM methodology we interviewed nine experienced K-12 Computing teachers about their teaching of programming to students aged 11-14 using PRIMM. From our analysis we identified that teachers particularly value predicting what code does, the potential for differentiation to a wide range of abilities, and the structure provided by PRIMM.

2 TEACHING PROGRAMMING: INSTRUCTIONAL APPROACHES

Much research associated with the teaching of programming has focused on pedagogy and instructional approaches to teaching. From the 1980s we see an emphasis on learners constructing knowledge as they explore [22]; applying ideas from Papert's constructionism we see instructional approaches based around open-ended activities, through which students can develop a personal understanding of newly introduced concepts or devices.

More recent research has highlighted the need for guided instruction to ensure that learners circumnavigate a carefully constructed progression to develop a complete mental model [8, 9, 17, 20, 25]. 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[9]. 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 [11, 12, 20]. Other studies raise the need for learners' cognitive load to be managed by more closely controlling learning opportunities and learning experiences [1, 32, 33]. This research has implications for pedagogy in school, suggesting that targeted teaching is needed for difficult concepts within a controlled progression of learning experiences [35].

Research in the teaching of programming has also included a focus on the levels of abstraction involved in understanding how to write programs. Cutts et al. reviewed university students use of vocabulary when solving multiple choice questions [6] and suggested a teaching model of student understanding of programs called the Abstraction Transition Taxonomy (ATT) which included three levels of language in programming: English, CS Speak and Code. The recommendation from this research was to support learners to be able to transition across all levels. Another framework, the Levels of Abstraction (LOA) framework, has been developed for introductory programming courses

[2, 23, 28]. Four levels are described: *execution*; *program*; *algorithm* and *problem* [2]. The focus of the LOA framework is on learners knowing what level they are working at and being able to transition between the levels.

Another focus has been on reading and tracing code. 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[15, 16]. Comparing tracing skills to code writing, they demonstrated that novices require a 50% tracing code accuracy before they can independently write code with confidence [16, 34]. Learning to program is sequential and cumulative, and tracing requires students to draw on accumulated knowledge to conceive a big picture. Work by Teague and Lister in this area suggests that novice learners should be focused on very small tasks with single elements [30]. Another study concluded that as well as inferring meaning from code from its structure, the first step should be to make inferences about the execution of the program [5].

Studies related to code comprehension have also highlighted the importance of reading code to address misconceptions of algorithm efficiency [7] and the use of worked examples to understand how variables change over time [29]. Gujberova and Kalas 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 [10]. Another approach 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 [18, 21] 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.

Another approach used in the teaching of programming is Use-modify-create (UMC). UMC is a teaching framework for supporting progression in learning to program [14]. 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'.

3 THE PRIMM APPROACH

The PRIMM approach builds on some of the research cited above. In particular it draws primarily on three areas of research:

- **Tracing and read-before-you-write**[15, 16]. PRIMM draws on tracing and reading code as an important principle for teaching programming [15]. The *predict* phase of PRIMM encourages students to practice reading code and working out what it will do when executed.
- **Use-Modify-Create** [14]. PRIMM is influenced by the work on Use-Modify-Create (UMC) [14]. PRIMM's *predict*, *run* and *investigate* phases map to the *use* stage. *Modify* is the same across both frameworks. PRIMM's *make* phase is equivalent to *create*. PRIMM has partly built on UMC to gradually transfer ownership of the program to the student. It supports the student's confidence as they are not burdened by the prospect of failure until they understand how the program works.
- **Levels of Abstraction** [6, 23]. Thirdly, PRIMM draws on work relating to abstraction [6, 23], in that the different activities focus on different levels of abstraction. The *Predict* and *Run* phases focuses very much on the execution of code, whereas the *Investigate* stage is about the program, or Cutts et al's *Code* level. When students reach the *Make* stage they have developed skills to focus on the 'problem' that needs to be solved.

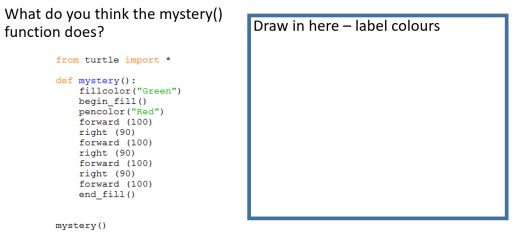

Fig. 1. A predict activity from one of the first lessons

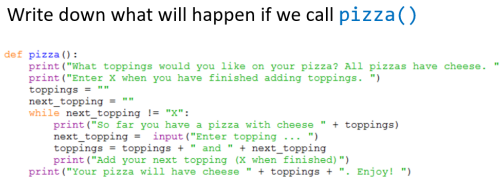

Fig. 2. A predict activity from a more advanced lesson

4 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.

4.1 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. Our examples use Python; two examples are shown in Figures 1 and 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.

4.2 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 [25] 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. In the pizza example shown in Figure 2, the student may be asked what happens if the user does not add any toppings. In addition 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 [6].

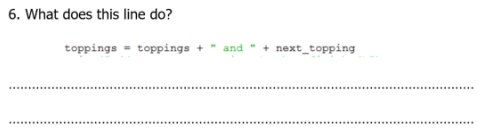

Fig. 3. Sample question in the investigate phase

4.3 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, following the Pizza example in Figure 2, a *modify* task may to 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 [6].

5 THE STUDY

As part of our research into the effectiveness of PRIMM, we applied design-based research methodology [4] to evaluate and test the materials implementing the PRIMM approach. The goal of design-based research (DBR) is to *"use the close study of learning as it unfolds within a naturalistic context that contains theoretically inspired innovations, usually that have passed through multiple iterations, to then develop new theories, artifacts, and practices that can be generalized to other schools and classrooms."* [p.151][3]. DBR takes materials that have been developed through a particular theoretical perspective and implements them in a naturalistic setting, iterating with the results of the materials in context. The PRIMM materials are on their third iteration: firstly in a small study in a CPD setting [26], secondly used in a pilot study with six teachers (Section 5.1), and the third iteration developed for the main study (Section 5.2). We used DBR to consider both the structure of PRIMM as a teaching aid and the actual materials produced to exemplify PRIMM.

5.1 The pilot study

We designed and implemented a short pilot study to evaluate the effectiveness of PRIMM and understand it better. The pilot study involved 6 teachers and approximately 80 students over 4-7 lessons, followed by individual interviews with the teachers to consider their views of PRIMM and the materials. Interviews were recorded, transcribed, coded and analysed. Teachers were enthusiastic about the PRIMM structure; our analysis of the interviews enabled us to further develop the materials and to devise appropriate research instruments for the subsequent main study. For the pilot we developed PRIMM-style worksheets for teachers to use with the intention of teachers developing them further and becoming co-creators of PRIMM materials. We concluded 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.

5.2 The main study

For the main study we built on the pilot study in refining both the study materials and the analysis process. 14 teachers were recruited, of whom 13 completed the trial and taught just less than 500 students using PRIMM for a period of three to four months at the beginning of 2018. As a separate part of the study we used a quasi-experimental approach to

investigate the impact of PRIMM on students' performance [27], with an experimental group of 493 students performing significantly better ($p<.05$) than a comparison group of 180 students in a post-test. In this paper, we focus on the qualitative analysis, and the teachers' experiences of PRIMM.

5.3 Participants

Teachers were recruited to the study via a number of channels, being selected on the basis of having students undertaking an appropriate programming module within the timeframe of our study. Teachers attended a one-day training session on the PRIMM approach and materials, which included the rationale behind DBR; teachers were invited to adapt, edit and upload edited materials as they wished during the study. Teachers completed evaluations of the PRIMM lessons that they taught, and interviews were arranged at the end of the trial with PRIMM materials. All ethical procedures were adhered to. Information about the teachers is shown in Table 1.

Table 1. Summary of teacher and lesson characteristics

ID	Teacher gender	Grade	Class gender in study	No. classes	No. topics covered	No. lessons \topic
A	Female	8th	Girls	2	10	1
B	Male	8th	Boys	3	7	2
C	Male	7th	Mixed	2	8	2
D	Female	6th	Mixed	1	5	2-3
E	Male	8th	Mixed	2	10	2-3
F	Female	7&8th	Mixed	4	4	2
G	Male	7th	Girls	3	7	1
H	Male	7th	Mixed	6	7-9	1-2
I	Male	8th	Girls	2	8	1

5.4 Data collection: interviews

Semi-structured interviews were designed and conducted. After the pilot study (see Section 5.1) we reviewed the interview questions for their contribution and usefulness and adapted them for the follow up study interview. In the pilot study teachers were asked questions on general information about sessions with PRIMM and questions on PRIMM experiences. The general questions were retained as they were useful to verify subject material covered; the experience questions were expanded and grouped over three sections, resulting in four sections of questions for the main study:

- Section A General information about PRIMM sessions
- 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

In total nine interviews were held with teachers from the 13 participating schools: these 9 were representative of the sample of 13 (see Table 1). All interviews and focus groups were completed in the spring of 2018.

5.5 Data analysis

All interviews were conducted online, audio recorded and transcribed. A thematic qualitative data analysis (QDA) approach was used to analyse the transcribed interviews and outcomes of tasks based on the methodology detailed by

Table 2. Summary of codes

Overarching theme	% segments coded to this theme
A Practical details of implementation	9%
B Skills needed to program	4%
C. Stages of PRIMM	18%
D. Impact and use of PRIMM	23%
E. Differentiation and assessment	24%
F. Adaptation and future use	12%
G. Teachers' feelings and emotions	11%

[13]. NVivo was used to support the process of coding text segments. Two of the authors worked on the coding of the interviews.

In the pilot study we had first generated high level categories deductively from the research questions [13]. We had also created sub-categories at this stage, based on knowledge of the field [19]. However, in the main study we started by coding inductively from the interviews [19]. The rationale for this decision was to investigate whether there were any significant differences in the emergent themes. 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 1/5th of the overall transcripts in line with recommendations from [13] of 10 to 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 [13, 19]. Once the more elaborate category system had been created, we checked that all data adhered to the new coding structure and recoded as necessary [13]. 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.

6 RESULTS: TEACHERS' EXPERIENCES

Through an iterative coding process seven key themes emerged from the data, as shown in Table 2, which shows the themes and the % of coded segments for each theme. In total there were 1603 coded segments. Within the 6 themes, categories were divided and further sub-divided to capture teachers' contributions. In total there were 87 separate sub-categories of themes. The 20 most commonly occurring of these 101 sub-categories is shown in Table 3.

From the 20 most common sub-categories and from our own theming exercise, we can see that teachers were commenting largely on the structure and use of PRIMM, differentiation for different groups of students, and also their own response to teaching in this way as teachers.

6.1 The structure and use of PRIMM

Teachers commented on a range of aspects of PRIMM that supported their teaching. Several teachers liked the way the lessons were divided up into different activities and maintained interest of students of different abilities:

Table 3. Most common sub-categories

No	Sub-category	No. segments
1	Use and adaptation of PRIMM	147
2	Use for lower ability children	91
3	Predict aspect of PRIMM	78
4	Teacher emotions	70
5	Progress made by pupils	63
6	Modify aspect of PRIMM	59
7	Investigate aspect of PRIMM	58
8	Use for higher ability children	50
9	PRIMM being fun	46
10	Future use of PRIMM	42
11	Make aspect of PRIMM	35
12	Routine nature of PRIMM	34
13	Differentiation by group	32
14	Assessment	32
15	Run aspect of PRIMM	31
16	Repeating or interwoven concepts	30
17	Differentation by task	30
18	Flow of control and tracing	27
19	Differentiation by teacher guidance	25
20	Pupil/pupil communication	23

> *"...it's chunking it up ... and each lesson having a new challenge that builds on the previous one so it keeps the interest of the ones that have raced ahead in the previous lesson."* (Teacher A)

One teacher commented that the **predict and run** part of the lesson promoted student engagement:

> *"I think the Predict bit drew them into the lesson from the start and then they were focused and that's what made them want to get involved."* (Teacher I)

Generally, asking students to predict what code would do without writing any was a new strategy for teachers. Teacher C commented that he felt it set much higher expectations of his students. Several other teachers referred to the collaborative nature of PRIMM:

> *" I think the starter activity, where they actually have a clear activity, that was very good because it got them to work with each other, help them help each other, really. And then, moving on, [we had] the code to be given to the students to actually work with."* (Teacher H)

Teachers also commented on the independence and thinking skills generated by predicting what the code would do:

> *"With the starters, quite often I used a whiteboard where they actually had to write down what they thought the program was going to do, what values they thought the program was going to create. Because they were having to work out and write down what the program was doing, they were having to actually think about for themselves, rather than me telling them what the program was doing."* (Teacher C)

Teachers could see the value of asking questions to demonstrate code comprehension, although the **investigate** part of the process was felt to be a difficult part of the lesson for some students. In part this was caused by the pitch of the

materials (written by the research team) being too high, and including some challenging questions. However alongside these concerns teachers acknowledged that they helped students gain an understanding of what was going on in the program:

> *" They were spotting how things happened or the significance of particular bits of the code and realising that essentially that the task was easier than they might previously have thought."* (Teacher I)

Teacher I linked enjoyment of the **modify** stage and independence as he said:

> *"The less able ones [students] enjoyed it (modify) because they got what they were doing when they were at that stage. They were more sure of themselves than they have been in previous Python lessons where they've relied on my telling them. This was them doing it themselves … the difference was tangible."* (Teacher I)

Eight of the nine teachers mentioned how clear the PRIMM process was, how easy it was to follow the PRIMM structure or the routine repeatable nature of PRIMM:

> *"By the third lesson they were familiar with the structure and they were happy to go with it. They knew what was coming next, if you like, without necessarily knowing what the content was going to be, they knew what was coming up next."* (Teacher I)

Some of the content provided in the PRIMM materials was felt to be too advanced for students at some schools, and teachers felt they were pushing students through it. For example, Teacher D pointed out that he felt pressurized to get learners to complete work:

> *"In one sense, it was very well structured because it meant that the high-performance students never run out of work. In none of the lessons did I have students who completed every single thing. But it also meant that I was sort of pushing on a lot."* (Teacher D)

The PRIMM **modify** stage was linked to learner confidence:

> *"…previously, when they're writing their own programs, we have so much trouble with syntax errors and half the lesson is just them sorting out syntax. So actually being able to modify it, they can think a little bit more about what their code is doing rather than whether they've got a colon in the right place or whatever. And then I think, moving on, once they get to making their own, they've got that little bit more confidence that they've got a starting point to move on from"* (Teacher A)

Teachers reported valuing and making time for the *make* aspect, despite being short of time:

> *"One of the reasons I didn't get through ten activities is that the geography quiz was a really interesting make project for them, yet most of them barely started their first question within the time available in one hour across that lesson. I actually sacrificed the next lesson … it was a nice lesson to have as an extension without introducing new concepts. We actually gave up another hour to making and peer reviewing the geography quiz"* (Teacher A)

6.2 Accessibility to a range of learners

One of the motivations behind PRIMM was that mandatory computing needs to be accessible to students with a wide range of abilities, and those who may not have any innate interest in the subject. For example, Teacher D mentioned that if students were not interested at all in programming, they could get into the content of the lesson much more quickly *"because they didn't have to go through the process of typing it out themselves"*.

Three of the teachers described how they adapted some of the resources in the **investigate** stage, for example making missing word activities. Some teachers described how their lower-ability learners did not get on to the final **make** activities but that they had still achieved some of the learning outcomes:

> *"I found that the resources for each lesson had enough stretch for those that got it straight away and wanted to go ahead, but also there was enough so that those that didn't get it. They could carry on modifying, not necessarily getting onto the make part . . . and they didn't have that sense of frustration that they weren't doing anything."* (Teacher D)

Teacher D also saw starting with an existing program an advantageous way to make progress:

> *"It's like a writer, isn't it? You write an essay. If you start with a blank bit of paper, that scares some people. I think the same can be true of a blank IDLE screen. So I think having a structure to hinge what they're doing on and not starting blank and the idea of tinkering is actually a good one."* (Teacher D)

Many of the lower-ability students lack confidence in programming but teachers felt that the PRIMM structure enabled students to gradually build up confidence through the repetitive nature of the exercises, and be more comfortable about making mistakes:

> *"At first, they weren't confident by themselves to predict, and so I'd ask them to talk in pairs, for example, as I think they were scared of getting it wrong . . . But as we went on and it became a routine that we did it, and I had to build up the fact that they probably will fail as resilience . . . as a normal way of working. Once they got to that point, they realised if they got it wrong, 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."* (Teacher D)

6.3 Teachers' personal experience

Teachers talked a lot about how they experienced teaching with PRIMM personally. Several teachers discussed gaining more confidence or insight into their own teaching methods.

> *"I felt that the structure, it's something that I could hand over to someone that wasn't such a confident coder and they could go with it as well. "* (Teacher A)

Other teachers mentioned that using PRIMM made them more confident in their own students ability to learn new concepts:

> *"I think that's one of the things that I'd take away from PRIMM. It's something that had changed my practice and it made me I suppose more confident in kids' ability to grasp functions or user-defined functions and procedures. I'd say that was one of the few things in which I'd changed in terms of confidence."* (Teacher B)

Another teacher explained how they had more idea of students' progress during the lesson, which made them more effective:

> *"I found I could get round the room much more quickly, and I was seeing everyone's screen or I was being shown things. Things were being explained to me, instead of me asking for explanations from them."* (Teacher I)

One of the most confident and experienced teachers in the study, who found keeping to the PRIMM structure through the whole trial quite constraining, talked about the PRIMM approach as something he would use amongst

other approaches, and repeated several times that he thought of PRIMM as a 'philosophy' rather than a 'methodology'. From this we infer that the structure was restrictive to him, but that the inherent ideas implicit were useful:

> *"I think the structure of PRIMM every lesson is a bit constraining. But I think as a philosophy, it's great. And I hope that the post-test against the control group shows that PRIMM is successful because it seems to make sense. It seems to me that, as a philosophy, it would be great. As a methodology, it's quite constricting."* (Teacher B)

Teachers also talked about adapting the resources for future use, which implies a confidence to develop their teaching further. Finally, teachers had comments on improvements to the materials, or how they might adapt the approach for their particular students. Overall teachers were positive: *"It really does work! "* (Teacher I)

6.4 Summary

Our study to date indicates that teachers value the structure provided by PRIMM. Many comments related to the *Predict* phase particularly. This aligns with the work around the importance of reading and tracing code [15, 30, 34] as well as the importance of understanding transition between different levels of a program [2, 6, 23, 28]. Teachers like the idea of students modifying existing code, supported by the UMC approach [14], although in some cases the materials were too weighty for the teachers to reach those sections in the time available. In terms of the materials we developed to exemplify PRIMM, our research indicates that while the *structure* of the lessons suited the lower ability students in the class, the quantity of *content* in the materials meant teachers could not complete all the activities planned. This does not necessarily reflect on the PRIMM structure but indicates that both the structure and content of PRIMM lessons are equally important.

Some teachers in our study were very willing to adapt materials and create new ones for students, for topics that we had not yet covered, in particular. Other teachers were less confident to do this and felt constrained by being in a research study. We reflected that, although we had built up good relationships with teachers in the study, in future studies we would ensure that materials were co-constructed and owned more definitively by teachers.

7 CONCLUSION

In this paper we have described the structure of the PRIMM approach and the materials we developed to exemplify it. Through interviews with teachers we have gained an understanding of the extent to which PRIMM is useful in school.

From teacher feedback, we can conclude that teachers find the approach useful and indicate that it helps students to gain a better understanding of programming, as well as structuring productive lessons. We have shown that teachers particularly value the collaborative approach taken in PRIMM, and the structure given to the lessons. Teachers also could see the potential for differentiation to a wide range of abilities, and the focus on subject-specific vocabulary that PRIMM engenders.

Although there has been much research around programming pedagogy, with a range of strategies suggested, our experience to date suggests that this approach offers teachers a clear and easy-to-follow structure for lessons that other approaches have not provided. For this reason, in our future work we plan to revisit the methodology used to ensure that teachers feel able to engage more with the materials design and take an enhanced participatory role in the research. We are collecting PRIMM-style resources that teachers have developed to use in their classrooms to use as exemplars for a wider body of teachers. We anticipate that this approach may be of use to other researchers of school-based computing education.

REFERENCES

[1] Giora Alexandron, Michal Armoni, Michal Gordon, and David Harel. 2014. Scenario-based Programming: Reducing the Cognitive Load, Fostering Abstract Thinking. In *Companion Proceedings of the 36th International Conference on Software Engineering (ICSE Companion 2014)*. ACM, New York, NY, USA, 311–320. https://doi.org/10.1145/2591062.2591167 00023.

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

[3] Sasha Barab. 2014. Design-based research: A methodological toolkit for engineering change. In *The Cambridge Handbook of the Learning Sciences, Second Edition*. Cambridge University Press.

[4] Sasha Barab and Kurt Squire. 2004. Design-based research: Putting a stake in the ground. *The journal of the learning sciences* 13, 1 (2004), 1–14.

[5] Teresa Busjahn and Carsten Schulte. 2013. The Use of Code Reading in Teaching Programming. In *Proceedings of the 13th Koli Calling International Conference on Computing Education Research (Koli Calling '13)*. ACM, New York, NY, USA, 3–11.

[6] Quintin Cutts, Sarah Esper, Marlena Fecho, Stephen R. Foster, and Beth Simon. 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 (ICER '12)*. ACM, New York, NY, USA, 63–70.

[7] Judith Gal-Ezer and Ela Zur. 2004. The efficiency of algorithmsâĂŤmisconceptions. *Computers & Education* 42, 3 (April 2004), 215–226. https://doi.org/10.1016/j.compedu.2003.07.004

[8] Varvara Garneli, Michail N. Giannakos, and Konstantinos Chorianopoulos. 2015. Computing education in K-12 schools: A review of the literature. In *Global Engineering Education Conference (EDUCON), 2015 IEEE*. IEEE, 543–551. 00017.

[9] Shuchi Grover, Roy Pea, and Stephen Cooper. 2015. Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education* 25, 2 (April 2015), 199–237. https://doi.org/10.1080/08993408.2015.1033142

[10] Monika Gujberova and Ivan Kalas. 2013. Designing Productive Gradations of Tasks in Primary Programming Education. In *Proceedings of the 8th Workshop in Primary and Secondary Computing Education (WiPSE '13)*. ACM, New York, NY, USA, 108–117. https://doi.org/10.1145/2532748.2532750 00012.

[11] Peter Hubwieser, Michal Armoni, Michail N. Giannakos, and Roland T. Mittermeir. 2014. Perspectives and Visions of Computer Science Education in Primary and Secondary (K-12) Schools. *Trans. Comput. Educ.* 14, 2 (June 2014), 7:1–7:9. 00041.

[12] Paul A. Kirschner, John Sweller, and Richard E. Clark. 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 (June 2006), 75–86. https://doi.org/10.1207/s15326985ep4102_1 05793.

[13] Udo Kuckartz. 2014. *Qualitative text analysis: A guide to methods, practice and using software*. Sage.

[14] Irene Lee, Fred Martin, Jill Denner, Bob Coulter, Walter Allan, Jeri Erickson, Joyce Malyn-Smith, and Linda Werner. 2011. Computational thinking for youth in practice. *ACM Inroads* 2, 1 (2011), 32.

[15] Raymond Lister, Elizabeth S Adams, Sue Fitzgerald, William Fone, John Hamer, Morten Lindholm, Robert McCartney, Jan Erik Moström, Kate Sanders, Otto Seppälä, et al. 2004. A multi-national study of reading and tracing skills in novice programmers. In *ACM SIGCSE Bulletin*, Vol. 36. ACM, 119–150.

[16] Raymond Lister, Colin Fidge, and Donna Teague. 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 (ITiCSE '09)*. ACM, New York, NY, USA, 161–165.

[17] Sze Yee Lye and Joyce Hwee Ling Koh. 2014. Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior* 41 (2014), 51–61. 00303.

[18] Lauren E. Margulieux and Richard Catrambone. 2016. Improving problem solving with subgoal labels in expository text and worked examples. *Learning and Instruction* 42 (2016), 58–71. 00015.

[19] P Mayring. 2000. Forum: Qualitative Social Research Sozialforschung, 2. History of Content Analysis. In *Forum: Qualitative Social Research. Sozialforschung*, Vol. 1.

[20] Orni Meerbaum-Salant, Michal Armoni, and Mordechai (Moti) Ben-Ari. 2013. Learning computer science concepts with Scratch. *Computer Science Education* 23, 3 (Sept. 2013), 239–264. https://doi.org/10.1080/08993408.2013.832022

[21] Briana B. Morrison, Lauren E. Margulieux, Barbara Ericson, and Mark Guzdial. 2016. Subgoals Help Students Solve Parsons Problems. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE '16)*. ACM, New York, NY, USA, 42–47. https://doi.org/10.1145/2839509.2844617 00025.

[22] Seymour Papert. 1980. *Mindstorms: children, computers and powerful ideas*. Vol. Harvester studies in cognitive science. Harvester, Brighton.

[23] Jacob Perrenet, Jan Friso Groote, and Eric Kaasenbrood. 2005. Exploring students' understanding of the concept of algorithm: levels of abstraction. *ACM SIGCSE Bulletin* 37, 3 (2005), 64–68.

[24] Anthony Robins, Janet Rountree, and Nathan Rountree. 2003. Learning and Teaching Programming: A Review and Discussion. *Computer Science Education* 13, 2 (2003), 137–172.

[25] Carsten Schulte. 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 (ICER '08)*. ACM, New York, NY, USA, 149–160.

[26] Sue Sentance and Jane Waite. 2017. PRIMM: Exploring pedagogical approaches for teaching text-based programming in school. In *Proceedings of the 12th Workshop in Primary and Secondary Computing Education*. ACM. https://doi.org/10.475/123_4

[27] Sue Sentance, Jane Waite, and Maria Kallia. [n. d.]. Teaching computer programming with PRIMM: a sociocultural perspective. Paper in review. ([n. d.]).

[28] David Statter and Michal Armoni. 2016. Teaching Abstract Thinking in Introduction to Computer Science for 7th Graders. In *Proceedings of the 11th Workshop in Primary and Secondary Computing Education (WiPSCE '16)*. ACM, New York, NY, USA, 80–83. https://doi.org/10.1145/2978249.2978261 00010.

[29] Leigh Ann Sudol-DeLyser, Mark Stehlik, and Sharon Carver. 2012. Code comprehension problems as learning events. In *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education*. ACM, 81–86. http://dl.acm.org/citation.cfm?id=2325319

[30] Donna Teague and Raymond Lister. 2014. Programming: Reading, Writing and Reversing. In *Proceedings of the 2014 Conference on Innovation and Technology in Computer Science Education (ITiCSE '14)*. ACM, New York, NY, USA, 285–290.

[31] The Royal Society. 2017. After the Reboot: Computing Education in UK Schools. Policy Report.

[32] Chia-Yin Tsai, Ya-Fei Yang, and Chih-Kai Chang. 2015. Cognitive Load Comparison of Traditional and Distributed Pair Programming on Visual Programming Language. In *Educational Innovation through Technology (EITT), 2015 International Conference of*. IEEE, 143–146. 00001.

[33] Jeroen J. G. van MerriÃńnboer and John Sweller. 2005. Cognitive Load Theory and Complex Learning: Recent Developments and Future Directions. *Educational Psychology Review* 17, 2 (June 2005), 147–177. https://doi.org/10.1007/s10648-005-3951-0 00000.

[34] Anne Venables, Grace Tan, and Raymond Lister. 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 (ICER '09)*. ACM, New York, NY, USA, 117–128.

[35] Jane Waite. 2017. Pedagogy in teaching Computer Science in schools: A Literature Review. https://royalsociety.org/ /media/policy/projects/computing-education/literature-review-pedagogy-in-teaching.pdf The Royal Society.