View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by UEL Research Repository at University of East London

PRIDAM: a framework for teaching programming

Adrian Udenze and Marwan Elfallah,
University of East London, UK

ABSTRACT

The drive to teach programming across Key Stages 1–4 in UK schools has resulted in computer science teachers without computer science degrees. Many of these teachers find it difficult to teach an abstract concept like programming to children. The work presented here proposes a framework for teaching programming to aid teachers and pupil progress. PRIMM, a state-of-the-art framework, was implemented in two Year 9 classes of a comprehensive school in England. Results presented show that when it comes to solving problems, PRIMM performs well for simple single-statement problems but fails for more complex multi-statement problems. PRIDAM improves on PRIMM by introducing problem Decomposition and Arrangement which makes it more suitable for more complex multi-statement problems. The authors conclude that whereas the PRIMM framework is suitable for introducing concepts, PRIDAM is suitable both for introducing concepts and solving programming problems

KEYWORDS

Teaching Programming

Teaching Computer Science

Programming Framework

INTRODUCTION

In 2013, the UK Government introduced a computing curriculum for England that requires programming to be taught in Key Stages 1–4 (Gov.uk 2013). To make this happen, the Government pledged £78m of funding and went into partnership with Raspberry PI to train up to 40,000 teachers in England, many of whom do not have computer science degrees (Murgia 2018). Over the past six years, many teachers have discovered that pupils find the concepts of programming difficult, and teachers also find it difficult to know how to help struggling pupils (Sentance & Waite 2017).

Even before the introduction of England's new curriculum, how to teach programming had been studied by researchers with the aim of identifying and solving the problems involved (Saeli et al. 2011). Many approaches to teaching programming are currently in place: the use of robots (Merkouris & Chorianopoulos 2018), use of simulated robot environments (Cyberbotics.com 2020), e-learning (Tundjungsari 2016), the use of different tools (Nowicki et al. 2013; Yildiz 2020) and gamification (Papadakis & Kalogiannakis 2018), to name a few. A good review of current literature can be found in Garneli et al. (2015).

The many different approaches and tools to facilitate the teaching of programming in schools is testament to the fact that there is no magic bullet that guarantees pupils will learn how to program in a programming course or class. However, the software industry faced a similar problem through the 1980s and 90s, going from ad hoc, structured programming techniques that resulted in one-off error-prone software, to the modern software engineering concepts of reusable, object-oriented programming (Mall 2003) resulting in robust, reusable and maintainable software. Computer

science teachers now face the same order of problems as the software industry did, and the current focus on and research into delivering structured lessons that conform to best practice, resulting in optimal progress of pupils, will no doubt pay off in the future.

The aim of this research was to investigate, and devise a framework for, teaching programming in secondary schools. Such a framework has a number of advantages for teaching computer science. Not all computer science teachers have computer science degrees (Gibbs 2016; Murgia 2018), and a framework will help them plan lessons to a uniform standard that ensures pupil progress. A framework is also beneficial for pupils, as a standard way of planning lessons will allow pupils to prepare for lessons that they know will conform to a set pattern, thus improving pupil progress. Also, a framework will be the foundation of using teaching tools like artificial intelligence (AI) in the classroom. If lessons conform to a framework, then AI can be used to train machines on assessment, lesson planning and differentiation (Roll & Wylie 2016).

This research began with a literature review of existing techniques for teaching in general, and teaching programming in schools. A state-of-the-art existing framework called PRIMM (Predict, Run, Investigate, Modify, Make; Sentance et al. 2019) was identified, implemented and investigated as a first step. Results presented show that PRIMM worked well for small single-statement problems, but pupils found the Make step very challenging when it came to more complex multi-statement problems. Based on the results of PRIMM, the authors have developed PRIDAM (Predict, Run, Investigate, Decompose, Arrange, Make). PRIDAM improves on PRIMM for more complex problems with multi-statements. PRIDAM was trialed in a UK comprehensive school on two Year 9 classes. Results presented show that PRIDAM outperforms PRIMM for more complex problems with multi-statements in terms of pupil progress and learning outcomes. The authors conclude that PRIMM is adequate for demonstrating concepts and statements but falls short on problem solving, whereas PRIDAM works well for demonstrating concepts as well as problem solving.

LITERATURE REVIEW

Given that not all computer science teachers have computer science degrees (Murgia 2018) and also that the subject of programming is an applied knowledge, which ideally requires years of experience solving problems to develop a deep understanding, it is not surprising that teachers and pupils alike find programming daunting (Sentance & Wylie 2017). Those that have the necessary field experience, most likely in industry, are not teachers, and most teachers do not have industrial experience.

There has been an ongoing discussion in the teaching profession on how best to enable learning. There are those who advocate so-called minimalist guidance techniques (also sometimes referred to as problem-based learning, inquiry learning, discovery learning) where students are left to learn by themselves. Examples of this approach to learning include where science students are placed in inquiry learning contexts (laboratory environments) and asked to discover fundamental and well-known principles of science (by experimenting). Exponents of this idea include Van Joolingen et al. (2005), Papert (1980) and Rutherford (1964).

On the other hand, there are those who propose that rather than leaving students to learn by themselves, direct support should be given, and concepts and procedures explained. Proponents of this idea include Cronbach & Snow (1977), Klahr & Nigam (2004), Mayer (2004) and Kirschner et al. (2006). Then there are those who propose a combination of the two, where fundamental concepts are taught and then students are left to gain deeper knowledge by themselves, (Kirschner et al. 2006; Meerbaum-Salant et al. 2013; Hubwieser et al. 2014; Grover et al. 2015).

On the pedagogy of teaching programming and computer science in general, Cutts et al. (2012) detail a framework that involves three levels of abstraction described as English, CS speak and Code. Students are required to be able to move from one step to the next.

Another framework in which four levels are proposed, namely execution, program, algorithm and problem, termed the Levels of Abstraction framework, is proposed in Armoni (2013). Lister et al. (2004, 2009) emphasise the need to read code and decipher what it does before attempting to write new code. Teague & Lister (2014) propose that students new to programming should begin with very small tasks with single elements.

A comprehensive literature review of teaching computing in primary and secondary schools (K-12) can be found in Garneli et al. (2015). Of the three questions the authors try to answer, 'RQ3: What are the most common instructional practices and how are educators putting them into practice?' is of interest. Bennett et al. (2011) suggest that a problem-project based approach where students follow a step-by-step procedure helps students create their own projects but could have a negative impact on the learner's creativity. To boost creativity, Kacmarcik et al. (2009) suggest using a 'Study, Modify, Extend' model.

Sentance et al. (2019) extend this idea further, where students are required to Predict, Run, Investigate, Modify and then Make programs. PRIMM encapsulates the idea of providing students with a framework which is directed but also allows the student to develop creatively by giving them the time to investigate and modify on their own.

METHODOLOGY

The research was carried out in an English comprehensive school in Greater London. Following a recent inspection, the school was rated 'Good' across all areas by the Office for Standards in Education (Ofsted);

in the 2018/19 academic year, the Progress 8 score was 0.2, attainment 8 score 49.04, 46% of pupils achieved grade 5 or above in English and maths, 67% achieved grade 4 or above in English and maths, 65.4% of pupils entered Ebacc, with 35% of the entrants achieving it. Two Year 9 classes were chosen for the research. To be allowed to choose computer science as a subject, pupils must have a certain level of maths and so the abilities of the Year 9 classes were above average for the school. Of the two Year 9 classes, one was slightly more able academically, the more able class scoring between 4 and 6% better in class tests. The pupils did some block-based visual programming in Years 7 and 8; however, they were completely new to text-based programming and the Python programming language which was used in the research.

The literature suggests three techniques for collecting research data, namely quantitative, qualitative and a mixture of quantitative and qualitative data (Creswell 2003; Castellan 2013). For the research, the authors decided on the mixed approach. The research focus was to implement PRIMM, evaluate its performance for teaching Python programming to Year 9 pupils in a UK comprehensive school and, based on results, come up with appropriate improvements.

The performance metrics were:

- Given programming problems of varying complexity and difficulty, how well does PRIMM perform in terms of pupil scores for the different tasks?
- What was the pupil experience for the different tasks? Did they find the tasks easy or difficult and did they enjoy the challenge of completing the tasks?

For student scores, given various tasks, the authors decided to adopt a quantitative approach to collecting data. For pupil experience, we decided to adopt a qualitative approach to collecting data.

The research began with the authors preparing presentation slides and worksheets for the pupils based on PRIMM. As previously described , the authors adopted a mixture of guided learning and inquiry-based learning as a teaching approach. A typical lesson would be along the following lines:

- Introduce a programming concept to pupils
- Get the pupils to carry out the PRIMM (or PRIDAM)-based exercises in the worksheet
- Review the exercises with the pupils

The complexity of the problems was evaluated using Lines of Code (LOC), the idea being that optimum solutions for simple problems would have fewer LOC than more complex problems. To begin with, for simple problems with single elements, one or two LOC were given to the pupils. The pupils had to complete Predict, Run, Investigate, Modify and Make tasks. Following the Investigate and Modify steps of the PRIMM framework, pupils were expected to Make programs which involved applying the knowledge gained in previous steps to solving problems or Making a program. The Make step of the worksheets started off with simple one-statement problems and gradually increased to more complex multi-statement problems. At the end of each task, pupils were asked for feedback on the completed tasks. The feedback was in the form of selecting one of three choices on how difficult they found the task, whether it was Easy, Hard or Very Hard. There was also space provided for pupils to comment in their own words on the Make task.

Initially, PRIMM was used for tasks. When LOC got to 5 and above, the performance of pupils on Make tasks dropped to the point that they simply could not answer the questions. To aid pupils, the authors introduced the idea of problem Decomposition and Arrangement. One of the fundamental ideas of programming is problem decomposition, where a programming task is broken into smaller manageable tasks. For the Year 9 class, this amounted to decomposing problems until each task had a Python equivalent statement. At this point the problem could not be decomposed further and the next task was to arrange the sub-tasks (Python statements) to solve the given problem. The results in the next section will show a significant improvement in pupil performance when problem Decomposition and Arrangement is used. The new framework which improves on PRIMM the authors have called PRIDAM: Predict, Run, Investigate, Decompose, Arrange, Make. The Modify step of PRIMM was condensed into the Investigate step. The results section also shows a sample of pupil worksheets and pupil feedback for PRIMM and PRIDAM.

RESULTS

Figure 1 shows images of pupil worksheets using PRIMM and PRIDAM for multi-statement Make problems. The pictures in the top row are for PRIMM tasks. The pupils do nothing as they do not know how to progress with the Make questions after successfully completing the other PRIMM steps. Note that in all cases the pupils score 0 for the Make tasks since they have no idea how to progress. This was the case for most of the class after LOC got to greater than 5. The bottom row shows pupil worksheets using PRIDAM. Bottom left shows how problem Decomposition and Arrangement was taught. Bottom middle and right show the application of PRIDAM to solving problems. Note the methodical approach to solving the problems using PRIDAM and note that even if the final result isn't correct, the pupil still scores some marks for work done.

Table 1 shows the results of pupil ratings for the two classes for different LOC. The results show that there isn't a significant difference between the performance of the two classes.

Table 2 shows the performance of both classes for different PRIMM tasks. At 5 LOC and above, the average score of class

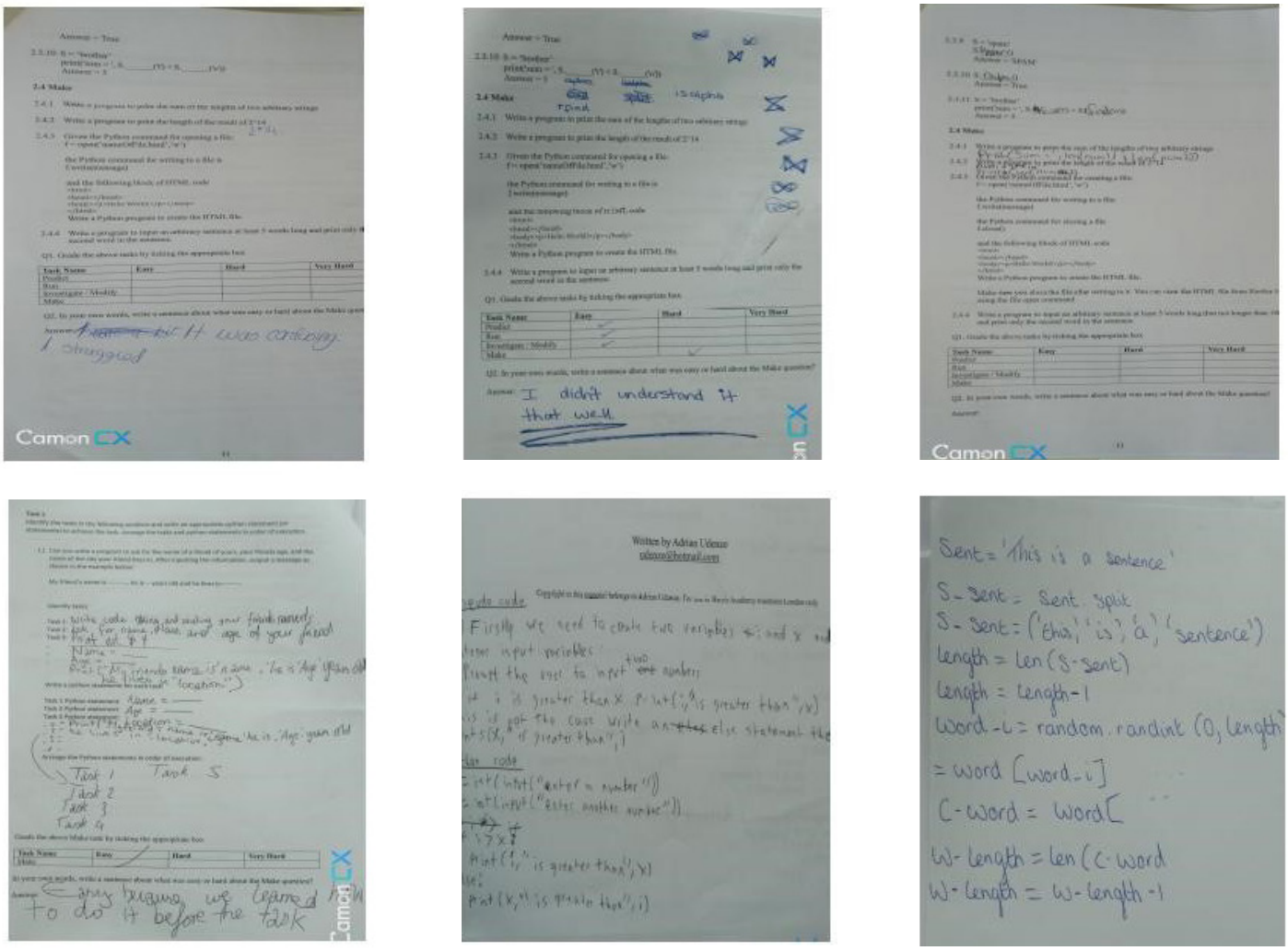


Figure 1: Pupil worksheets using PRIMM (top row) and PRIDAM (bottom row) for multiple-element Make questions

Class A						**Class B**				
Task	**LOC**	**Easy**	**Hard**	**Very Hard**		**Task**	**LOC**	**Easy**	**Hard**	**Very Hard**
Predict	1	15	3	0		Predict	1	12	3	0
Run	1	17	1	0		Run	1	15	0	0
IM	1	17	1	0		IM	1	12	3	0
Make	1	16	2	0		Make	1	11	3	1
Class A						**Class B**				
Task	**LOC**	**Easy**	**Hard**	**Very Hard**		**Task**	**LOC**	**Easy**	**Hard**	**Very Hard**
Predict	2	6	1	0		Predict	2	6	0	0
Run	2	4	2	1		Run	2	6	0	0
IM	2	2	4	1		IM	2	3	3	0
Make	2	4	2	1		Make	2	3	3	0
Class A						**Class B**				
Task	**LOC**	**Easy**	**Hard**	**Very Hard**		**Task**	**LOC**	**Easy**	**Hard**	**Very Hard**
Predict	3	9	8	0		Predict	3	6	5	0
Run	3	6	11	0		Run	3	7	4	0
IM	3	6	10	1		IM	3	6	5	0
Make	5	0	10	7		Make	5	1	2	8

Table 1: Comparison of pupil ratings for the two Year 9 classes.

		Class A	Class B
Task	LOC	Score	Score
Predict	1	92	95
Run	1	87	95
IM	1	98	89
Make	1	90	91
Predict	2	90	88
Run	2	87	94
IM	2	91	91
Make	2	85	89
Predict	3	87	86
Run	3	77	92
IM	3	83	90
Make	5	16	17

Table 2: Performance of class A and class B for different PRIMM tasks.

A dropped to 16% for the Make task and 17% for class B. For other tasks prior to Make, the scores remained consistently high for both classes.

At this stage, PRIDAM as discussed in the previous section was introduced. Table 2 shows the result of PRIDAM for Make tasks, where pupils are required to write programs for problems, for classes A and B. The results show a significant increase compared to PRIMM across both classes, with class B performing slightly better.

Plot 1 shows the performance for different LOC using the PRIMM and PRIDAM frameworks. PRIDAM shows an increase of between 40 and 50% for different LOC.

Plots 2 and 3 show the spread of scores and mean, median, mode, range for PRIMM versus PRIDAM for Make tasks. PRIDAM clearly outperforms PRIMM.

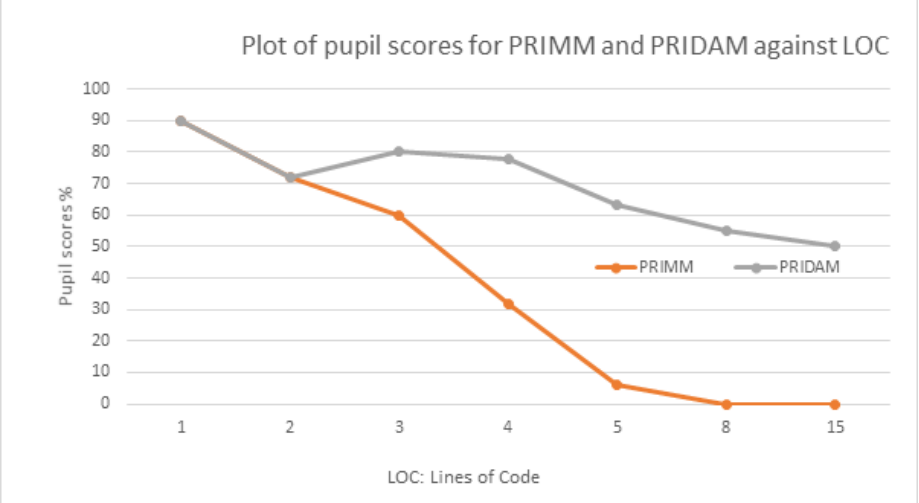
On the qualitative data collected, analysis showed that pupils were comfortable with simple tasks with single elements for PRIMM, with comments such as 'I found it easy because I did other tasks', whereas for more complex Make problems, pupils struggled to know where to begin, and comments like 'I don't know what to do' were common. For PRIDAM, pupils were more comfortable with more complex tasks and gave feedback such as 'Easy because I learned to do the tasks'.

CONCLUSIONS

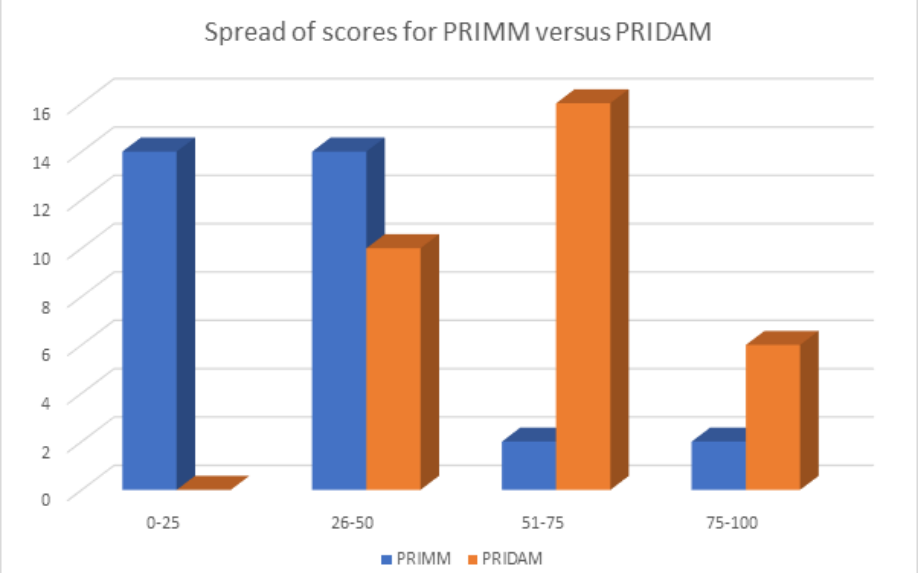
A framework for teaching programming can standardise lessons, making it easier

Class A						Class B					
Task	LOC	Easy	Hard	Very Hard	Score	Task	LOC	Easy	Hard	Very Hard	Score
Make 1	4	6	7	0	78	Make 1	4	8	6	0	81
Make 2	5	0	6	4	63	Make 2	5	2	9	3	72
Make 3	8	2	2	2	55	Make 3	8	2	3	1	70
Make 4	15	0	0	1	50	Make 4	15	0	0	3	56

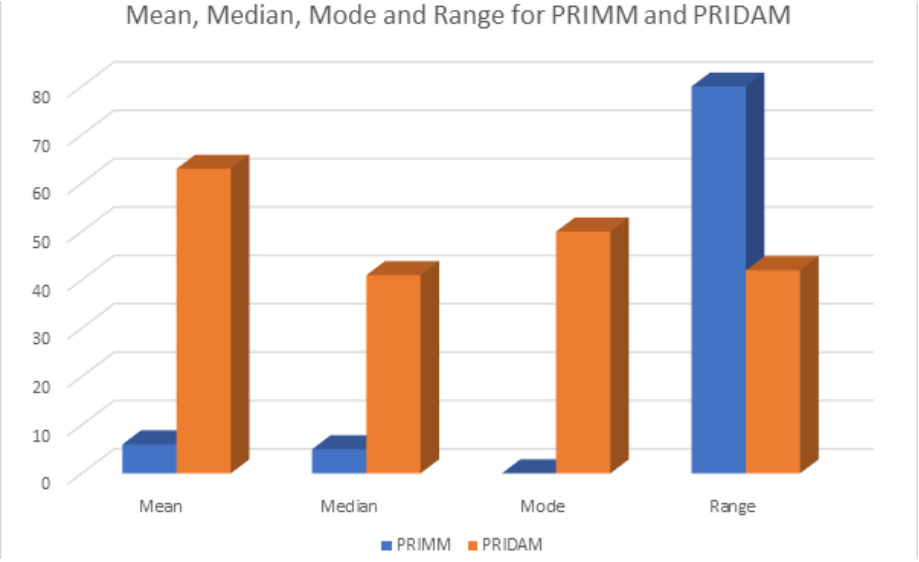
Table 3: Average performance scores for classes A and B for Make tasks.



Plot 1: Scores versus LOC for PRIMM and PRIDAM for Make tasks.



Plot 2: Spread of scores for PRIMM versus PRIDAM



Plot 3: Mean, median, mode and range of scores for PRIMM versus PRIDAM.

for teachers to teach and at the same time facilitating pupil progress. Results obtained from two Year 9 classes in a UK comprehensive school show that a framework called PRIDAM outperforms PRIMM, a state-of-the-art framework, when it comes to solving programming problems. PRIMM was shown to work well for teaching programming concepts that involve single statements but failed when it came to solving more complex problems involving multi-statements. PRIDAM was shown to match PRIMM for teaching concepts, and results also showed that PRIDAM outperformed PRIMM by as much as 50% for solving complex multi-statement problems.

The research was carried out in two Year 9 classes with a total of 60 pupils. In future the authors will be looking to expand the field trials to more schools and pupils. The PRIDAM framework was implemented with a lumped model for each step. Each model presents an opportunity for further research. How long should each step last for a given pupil? How is scaffolding to be done for each step? More granular models for PRIDAM will be researched. This work also lays the foundation for future work on AI-assisted teaching. A simulation environment in which different frameworks and modelling for frameworks can be experimented with to secure optimal pupil progress would be of great assistance to teachers. ■

REFERENCES

Armoni, M. (2013). 'On teaching abstraction in computer science to novices'. *Journal of Computers in Mathematics and Science Teaching*, 32(3), 265–84.

Bennett, V., Koh, K. & Repenning, A. (2011). 'CS education re-kindles creativity in public schools'. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education,* pp. 183–7. New York: ACM.

Castellan, C. M. (2013). 'Quantitative and qualitative research: a view for clarity'. *International Journal of Education,* 2(2), 1–14.

Creswell, J. W. (2003). *Research design: qualitative, quantitative and mixed methods approaches.* London: Sage Publications.

Cronbach, L. J., & Snow, R. E. (1977). *Aptitudes and instructional methods: a handbook for research on interactions*. New York: Irvington.

Cutts, Q., Esper, S., Fecho, M., Foster, S. & 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* (ICER '12), pp. 63–70. New York: ACM.

Cyberbotics.com. Cyberbotics official website. Online: https://cyberbotics.com/ [accessed 19 February 2020]

Garneli, V., Giannakos, M. and Chorianopoulos, K. (2015). 'Computing education in K-12 schools: a review of the literature'. In *IEEE Global Engineering Education Conference (EDUCON)*, 543–551. Piscataway, NJ: Institute of Electrical and Electronics Engineers.

Gibbs, D. (2016). Stem UK official website. Online: https://www.stem.org.uk/news-and-views/opinions/deep-end-%E2%80%93-non-specialists-teaching-computer-science [accessed 2 March 2020]

Gov.uk. (2013). UK Government official website. Online: https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study [accessed 20 March 2020]

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.

Hubwieser, P., Armoni, M., Giannakos, M. & Mittermeir, R. T. (2014). 'Perspectives and visions of computer science education in primary and secondary (K-12) schools'. *ACM Transactions on Computer Education*, 14(2), 7:1–7:9.

Kacmarcik, G. & Kacmarcik, S. (2009). 'Introducing computer programming via gameboy advance homebrew'. *ACM SIGCSE Bulletin*, 41(1), 281–5.

Kirschner, P., Sweller, J. & Clark, R. (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.

Klahr, D. & Nigam, M. (2004). 'The equivalence of learning paths in early science instruction: effects of direct instruction and discovery learning'. *Psychological Science*, 15, 661–7.

Lister, R., Adams, E., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J., Sanders, K., Seppälä, O., Simon B. & Thomal L. (2004). 'A multi-national study of reading and tracing skills in novice programmers'. *ACM SIGCSE Bulletin*, 36, 119–50.

Lister, R., Fidge, C. and 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 (ITiCSE '09)*, pp. 161–5. New York: ACM.

Mall, R. (2003). *Fundamentals of software engineering*, 4th edn. New Delhi: PHI Learning.

Mayer, R. (2004). 'Should there be a three-strikes rule against pure discovery learning? The case for guided methods of instruction'. *American Psychologist*, 59, 14–19.

Meerbaum-Salant, O., Armoni, M. & Ben-Ari, M. (2013). 'Learning computer science concepts with Scratch'. *Computer Science Education*, 23(3), 239–64.

Merkouris, A. & Chorianopoulos, K. (2018). 'Programming touch and full-body interaction with a remotely controlled robot in a secondary education STEM course'. In *Proceedings of 22nd Pan-Hellenic Conference on Informatics (PCI '18)*, 5pp. New York, ACM. Online: https://doi.org/10.1145/3291533.3291537 [accessed 19 February 2020]

Murgia, M. (2018). 'How the UK plans to teach computer science to every child'. Ft.com. *Financial Times* official website. Online: https://www.ft.com/content/a712f6de-ef37-11e8-89c8-d36339d835c0 [accessed 19 February 2020]

Nowicki, M., Matuszak, M., Kwiatkowska, A., Sysło, M. & Bała, P. (2013). 'Teaching secondary school students programming using distance learning: a case study'. In *X World Conference on Computers in Education Toruń, Poland, pp. 246–54.*

Papadakis S. & Kalogiannakis M. (2018). 'Using gamification for supporting an introductory programming course: the case of classcraft in a secondary education classroom'. In *Brooks A., Brooks E. &Vidakis N. (eds), Interactivity, game creation, design, learning, and innovation. ArtsIT 2017, DLI 2017. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, vol. 229. Cham: Springer.

Papert, S. (1980). *Mindstorms: children, computers and powerful ideas.* Brighton: Harvester.

Rutherford, F. J. (1964). 'The role of inquiry in science teaching'. *Journal of Research in Science Teaching*, 2, 80–4.

Roll, I. and Wylie R. (2016). Evolution and revolution in artificial intelligence in education. *International Journal of Education*, 26, 582–99.

Saeli, M. et al. (2011). 'Teaching programming in secondary school: a pedagogical content knowledge perspective'. *Informatics in Education*, 10(1), 73–88. Online: https://pdfs.semanticscholar.org/6502/3157d05f80b46d492bd4c75741027b94a895.pdf [accessed 19 February 2020].

Sentance, S., & Waite, J. (2017). 'PRIMM: Exploring pedagogical approaches for teaching text-based programming in school'. In *Proceedings of the 12th Workshop in Primary and Secondary Computing Education: WIPSCE '17.* Nijmegen.

Sentance, S., Waite, J. & Kallia, M. (2019). 'Teachers' experiences of using PRIMM to teach programming in school'. In *The 50th ACM Technical Symposium on Computing Science Education: SIGCSE 2019.* New York: ACM.

Teague, D. & Lister, R. (2014). 'Programming: reading, writing and reversing'. In *Proceedings of the 2014 Conference on Innovation and Technology in Computer Science Education (ITiCSE '14),* pp. 285–90. New York: ACM.

Tundjungsari, V. (2016). 'E-learning model for teaching programming language for secondary school students in Indonesia'. In *13th International Conference on Remote Engineering and Virtual Instrumentation (REV)*, pp. 262–6. Piscataway, NJ: Institute of Electrical and Electronics Engineers.

Van Joolingen, W., De Jong, T., Lazonder, A., Savelsbergh, E., & Manlove, S. (2005). 'Co-Lab: research and development of an online learning environment for collaborative scientific discovery learning'. *Computers in Human Behaviour,* 21, 671–88.

Yildiz, D. (2020). 'The effects of using different tools in programming: teaching of secondary school students on engagement, computational thinking and reflective thinking skills for problem solving tech know learn'. Online: https://doi.org/10.1007/s10758-018-9391-y [Accessed 19 February 2020].