PRIMM: Exploring pedagogical approaches for teaching text-based programming in school

Sue Sentance
sue.sentance@kcl.ac.uk
King's College London
London, UK

Jane Waite
jane.waite@kcl.ac.uk
King's College London
London, UK

ABSTRACT

Many teachers are able to recognise that students can find programming difficult – it is not as easy for teachers to know how to help struggling students to gain confidence and a secure understanding of programming concepts.This is particularly acute where the curriculum requires the teaching of text-based programming from age 11. In this paper we describe an approach to teaching programming we call PRIMM – Predict-Run-Investigate-Modify-Make. This builds on three areas of research: the Use-Modify-Create methodology, levels of abstraction used in programming, and tracing and code comprehension research. The PRIMM approach has been trialled with teachers new to programming and is now being implemented in a pilot study in secondary schools.

CCS CONCEPTS

• **Social and professional topics → K-12 education**;

KEYWORDS

K-12 education, pedagogy, programming

ACM Reference format:
Sue Sentance and Jane Waite. 2017. PRIMM: Exploring pedagogical approaches for teaching text-based programming in school. In *Proceedings of WiPSCE '17, Nijmegen, Netherlands, November 8–10, 2017,* 2 pages.
https://doi.org/10.1145/3137065.3137084

1 INTRODUCTION TO PRIMM

The recent introduction of computer science into the school curriculum means that children are being taught programming from a young age. There are many visual environments that can support learners of programming, but sooner or later students will be required to write code in a text-based language, and this can cause many difficulties, frustration, and a tendency to try to get code working with a trial and error approach. Relatively inexperienced computer science teachers may need support in planning appropriate lessons. In this short paper, we introduce a new approach known as PRIMM, which draws directly on a body of research around the learning of programming, including work on use-modify-create, levels of abstraction and code comprehension. PRIMM stands for Predict, Run, Investigate, Modify and Make:

- **Predict**: summarise what given code will do on execution
- **Run**: execute code to test prediction
- **Investigate**: explain, trace, annotate, debug
- **Modify**: edit program to change its functionality
- **Make**: design a new program

A study is being carried out to implement and evaluate this approach in the classroom

2 PEDAGOGY AND PROGRAMMING

The PRIMM approach draws on three areas of research: Use-Modify-Create, Levels of Abstraction, and reading and tracing code for understanding.

2.1 Use-Modify-Create

Use-modify-create (UMC) is a teaching framework for supporting progression in learning to program [4]. 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'. Once students start to create their own programs they employ an iterative process of refine, test, and analyse [4]. 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*. However, within each of the PRIMM phases attention is drawn to the level of abstraction at which the student is working.

2.2 Levels of abstraction

Developed to teach abstraction in introductory programming courses, a Levels of Abstraction (LOA) framework has been proposed consisting of four levels, namely: *execution*; *program*; *algorithm* and *problem* [1]. There are synergies between the LOA model and the *block model* – a three-dimensional educational model of program comprehension [8, 9], as well as the *Abstraction Transition Taxonomy* – a discourse intensive teaching model of student understanding of programs [3].

The focus of the LOA framework is on learners knowing what level they are working at and being able to transition between the levels.

2.3 Tracing and read-before-you-write

A well-known body of work relates to code comprehension in novice undergraduate programmers; comparing tracing skills to code writing, their reports have concluded a direct correlation [5, 7, 11], including that novices require a 50% tracing code accuracy before they could independently write code with confidence [6, 11].

WiPSCE '17, November 8–10, 2017, Nijmegen, Netherlands

ACM ISBN 978-1-4503-5428-8/17/11.
https://doi.org/10.1145/3137065.3137084

Learning to program is sequential and cumulative, with tracing requiring students to draw on accumulated knowledge to conceive a big picture. Novice learners should be focused on very small tasks with single elements, with emphasis on reading and tracing code before they are expected to write code snippets [10]. Another study concluded that as well as the importance of inferring meaning from code from its structure, the first step should be to make inferences about the execution of the program [2].

There is more relevant research in these three areas which has impacted on our work but limited space to report on it in a short paper. In the following section, we describe the beginnings of a study to implement this in the classroom.

2.4 The Study

The PRIMM study has three phases:

- Phase 1: teacher training
- Phase 2: pilot study in school
- Phase 3: larger-scale study in school

Prior to the first stage, the framework was discussed with groups of teachers and pre-service teachers in a variety of fora.

2.5 Phase 1: teacher training pilot

In the first stage, PRIMM was implemented with 15 non-specialist teachers attending an 8-week course on the teaching of computer science at Grades 6-8. The course also included unplugged methods and active learning approaches to teaching computer science as well as learning to program using PRIMM. A survey instrument was designed to evaluate teachers' perception of their learning and the pedagogical approaches used at the end of the course.

2.6 Phase 2: pilot study in school

In the second phase of the study, 7 experienced teachers were recruited to use the PRIMM approach in their classrooms. This pilot study involved three stages: training of the teachers in the PRIMM method, the intervention in school over a period of 4 - 6 weeks, and an interview. Teachers took part in a session where they became familiar with the PRIMM method and adapted materials so that they were suitable for their own classes. Multiple-choice questions were used as pre-test and post-test for the students and teachers completed a journal while they were using the PRIMM lessons. The teachers were interviewed at the end of the 4-6 week period.

2.7 Phase 3: larger-scale study in school

In the third phase, still to be implemented, a study will be run in more schools over a whole academic year. This phase is in the early stages of development.

3 INITIAL RESULTS FROM PHASE 1

In phase 1, 15 teachers completed the end-of-course questionnaire. The course modelled good pedagogy during all activities, as well as teaching computer science concepts and Python programming for novices. The feedback from all aspects of the course was highly positive. As regards PRIMM, teachers were asked whether they found learning programming using PRIMM was helpful. 47% reported that it was very helpful, another 47% reported that it was helpful and only 1 respondent (7%) reported that this approach was slightly helpful. As regards their own teaching, 80% of teachers stated that they were planning to use the PRIMM method, 13% said that they possibly would and only 7% (the same respondent) said that they would not. Although a small sample, this feedback is encouraging as we move on to Phase 2.

4 CONCLUSION

The PRIMM project is an attempt to not only propose, but rigorously evaluate, an approach to teaching programming that can be directly implemented in the secondary classroom by teachers, experienced or otherwise. This classroom method directly builds on influential work in the computer science education literature. This work is important on two levels: on one level it is important that as computer science is increasingly taught in schools around the world we draw together what the research says and implement it in schools. At another level, we have an opportunity to find out on a large scale whether the findings reported are valid when used over a period of time in classrooms, and not simply in experimental studies. A third important feature of this work is that by engaging teachers in the project, we provide a focus for them to reflect on their teaching, engage directly with research, and contribute to the debate on how to teach programming effectively.

REFERENCES

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

[2] 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.

[3] 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.

[4] 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.

[5] Raymond Lister, Tony Clear, Simon, Dennis J. Bouvier, Paul Carter, Anna Eckerdal, Jana Jackov\, \#225, Mike Lopez, Robert McCartney, Phil Robbins, Otto Sepp\, \#228, l\, and Errol Thompson. 2009. Naturally occurring data as research instrument: analyzing examination responses to study the novice programmer. *SIGCSE Bull.* 41, 4 (2009), 156–173.

[6] 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.

[7] Mike Lopez, Jacqueline Whalley, Phil Robbins, and Raymond Lister. 2008. Relationships Between Reading, Tracing and Writing Skills in Introductory Programming. In *Proceedings of the Fourth International Workshop on Computing Education Research (ICER '08)*. ACM, New York, NY, USA, 101–112.

[8] 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.

[9] Carsten Schulte, Tony Clear, Ahmad Taherkhani, Teresa Busjahn, and James H. Paterson. 2010. An Introduction to Program Comprehension for Computer Science Educators. In *Proceedings of the 2010 ITiCSE Working Group Reports (ITiCSE-WGR '10)*. ACM, New York, NY, USA, 65–86.

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

[11] 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.