Section 3:
Computing topics

Teaching programming with PRIMM: the importance of classroom talk

Sue Sentance (Raspberry Pi Foundation, UK)

Sentance, S. (2021). Teaching programming with PRIMM: the importance of classroom talk. In Understanding computing education (Vol 1). Proceedings of the Raspberry Pi Foundation Research Seminar series.

Available at: rpf.io/seminar-proceedings-2020

Raspberry Pi

Section 3: Computing topics

Teaching programming with PRIMM: the importance of classroom talk

Sue Sentance (Raspberry Pi Foundation, UK)

Abstract

PRIMM is an approach to structuring programming lessons with a focus on working with extracts of code in depth to understand both structure and function and doing so in collaboration with peers, through dialogue. Previous research has shown that teaching using a PRIMM approach can improve learner outcomes. In this paper I introduce the PRIMM approach to structuring lessons and how it can impact on productive classroom talk. A qualitative study was conducted with 20 programming teachers in primary and secondary schools. Early findings indicated that in PRIMM lessons teachers' talk differs in quality and content at different stages of the lesson, and highlights the importance of students' use of programming vocabulary. A focus on language and talk could be a productive area of research in our quest to improve our understanding of effective teaching strategies for young novice programmers.

Introduction

PRIMM is an approach to teaching programming that came about because teachers who were new to teaching, or new to teaching programming, were expressing frustration that they could not effectively support young students who had difficulty with programming. Computing teachers benefit from access to proven teaching strategies and pedagogies relating to programming. Much research has been carried out in programming education, and only recently in schools, 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, I and my colleagues have developed and are evaluating a new pedagogical model for teaching and learning programming (PRIMM) (Sentance & Waite, 2017, Sentance, Waite and Kallia, 2019).

PRIMM stands for Predict, Run, Investigate, Modify, and Make. Using PRIMM, classroom activities can be designed that 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 (Lister et al., 2004). It provides a staged and gradual approach to building an understanding of programming concepts alongside the development of confidence, with a focus on program comprehension over completed artefacts. It is an appropriate approach for young students where we need to minimise excessive cognitive load and helps teachers to engage each student when teaching large mixed-ability classes.

This paper focuses on one aspect that is a key feature of every PRIMM lesson: productive classroom talk. Despite a surge of interest in programming education in school in recent

years, the use of talk and language has not been a particular focus, with little literature in computing education on this topic. Research in mathematics and science education around dialogue has increased our understanding of both the nature of productive classroom talk, and how teachers can encourage this in their classes. What is of interest here is how this work relates to the programming classroom, and whether talking together about programs can really support learning. In this paper I outline what PRIMM is, why language and talk is important to the learning of programming, and report on some of the findings from a recent study.

Teaching programming

Novices can find programming difficult; research abounds on this topic. For example, it has been asserted that, beyond the syntax and semantics of particular programming concepts, novices may struggle to put these together 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). Actually writing code (as opposed to reading) is particularly hard for novice programmers (Denny et al., 2008; Qian & Lehman, 2017), and it is commonly believed that code tracing is easier than code writing (Denny et al., 2008). However, 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 also be viewed through cognitive load theory (van Merriënboer & 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). However, we may inadvertently use teaching methods which don't help this situation at all. A reliance on programming textbooks and "show me" approaches to teaching coding means that novices may end up being asked to copy in a section of code that has no meaning to them at all. Add this to the fact that younger learners will be developing their literacy and keyboard skills, the process of copying in can be incredibly frustrating and dispiriting. Another practice might be to model writing a program from the front while learners watch, and then ask learners to go ahead and write a similar program themselves: this leaves a huge chasm for the novice programmer to fill in themselves which many simply cannot manage.

What is PRIMM?

PRIMM stands for, Predict, Run, Investigate, Modify, and Make. It is based on the following five principles;

Principle 1: Read code before you write code. The excitement of writing a new program and creating something that works can mean we don't spend enough time at the beginning reading and learning from simple, well-written programs. PRIMM draws on tracing and reading code as an important principle for teaching programming (Lister et al., 2009). The predict phase of PRIMM encourages students to practise reading code and working out what it will do when executed.

Principle 2: Work collaboratively to talk about programs. Dialogue and classroom talk are an important aspect of teaching and learning.

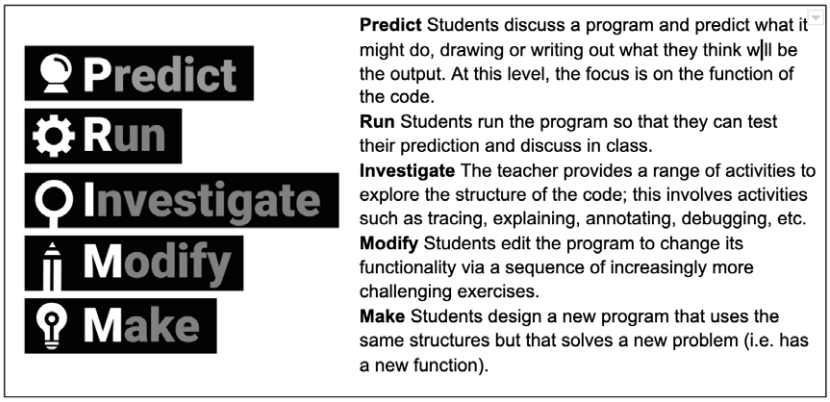

Figure 1. The five stages of PRIMM

PRIMM particularly focuses on classroom discussion, specific questioning about code, use of vocabulary, and asking students to talk to each other about code. PRIMM draws on sociocultural theory which helps us to understand how language can support learning. Language can be seen as a central form of mediation that enables thinking and internalisation of concepts to take place (Vygotsky, 1962). In PRIMM lessons, students are encouraged to discuss with each other; a social construction of knowledge formed through collaborative, program-focused tasks.

Principle 3: Focus on code comprehension. Languages like Python (commonly used in schools in England) are often celebrated because you can write a program in a short number of lines. However, that usually means there are lots of concepts in one line. One way to unpack what the code is doing is to align comprehension exercises to the Block Model (Schulte et al., 2010; Cruz et al., 2019). The Block Model distinguishes 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). Unpacking and focusing on understanding the code also reduces cognitive load on the learner (Sweller, 1994).

Principle 4: Use existing starter programs. Again drawing on sociocultural theory, learning can be seen as a transition from the social plane to the cognitive plane (Walqui, 2006; Sentance et al., 2019), through the use of 'starter' programs that students can work with before taking ownership themselves. A PRIMM lesson starts with an activity whereby learners examine some existing code and predict what it might do.The learner does not have responsibility for the code and does not suffer emotionally if the code has errors in. Learners can test their predictions by running the code.

Principle 5: Gradually take ownership of programs. Learners should move along a continuum from where they first use programs made by someone else to finally create their own programs. In this way, PRIMM has partly built on Use-Modify-Create (UMC) (Lee et al., 2011) 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.

PRIMM provides a structure for one of a series of lessons, with the intention that teachers can develop their own PRIMM-like materials at an appropriate level for their students (Figure 1).

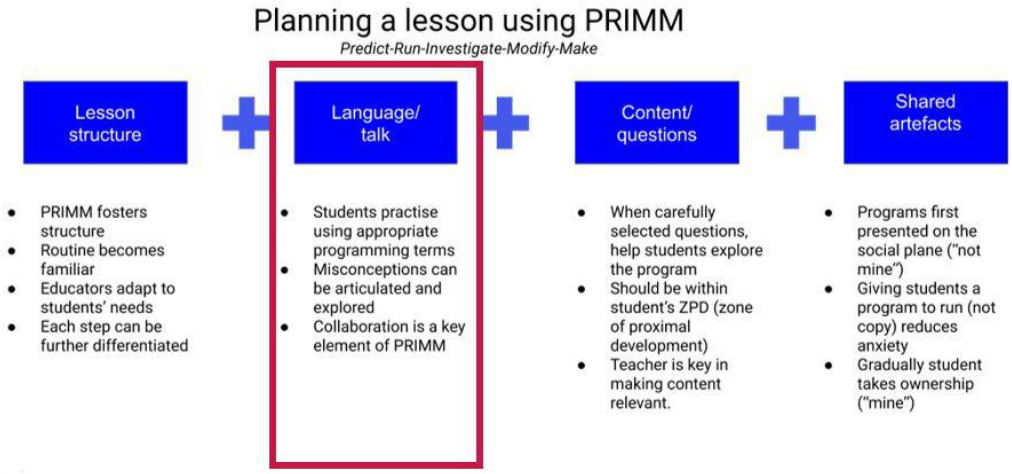

Figure 2. Planning a PRIMM lesson

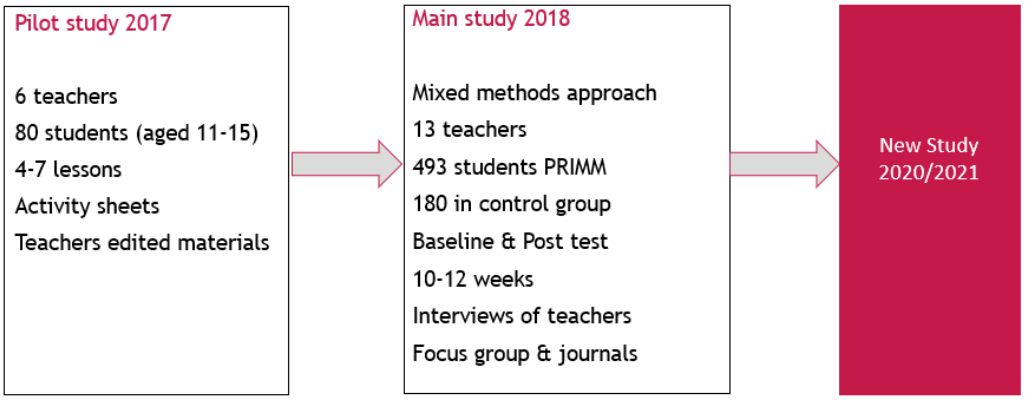

Figure 3. Research on the effectiveness of PRIMM approach

In terms of planning a PRIMM lesson, teachers will consider not only the structure of the lesson (as described in Figure 1), but also the opportunity for language and talk, the content and level of questioning, and the shared artefacts that are used in the lesson. These elements of planning are shown in Figure 2. This paper focuses on the language and talk that takes place in a PRIMM lesson.

PRIMM and learning outcomes

A number of studies have been employed to investigate the impact of PRIMM (see Figure 3). To date the largest of these was a mixed methods study conducted in 2018 involving around 500 students aged 11 to 14.

In this study, a type of quasi-experimental design known as the non-equivalent control group post-test design (Campbell & Stanley, 1963) was

used to investigate the impact of a series of PRIMM-structured lessons on learner outcomes. Following this methodology, the treatment, or experimental, group were classes being taught using PRIMM materials provided by the researchers, with the control group consisting 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. To ensure that students did not differ significantly in their computer programming attainment, both groups were baseline tested before the start of the intervention.
Teachers were given full sets of materials, including starter tasks, presentations, worksheets, starter programs, and answers, for ten lessons (including extension material) covering the basic programming constructs of sequence, selection, and iteration in Python. Teachers then delivered programming lessons using the PRIMM approach for 8 to 12 weeks. Data was collected via a combination of a baseline test, a post test to compare control and experimental groups, and teacher interviews.

The post-test score of the experimental group was compared with that of the control group. 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. The results showed a statistically significant difference in the score between the control and experimental groups for all students in favour of the experimental group (see Sentance et al. (2019) for further details).

The quantitative results were further supported by the qualitative data. From interviews with nine participating teachers the research found that teachers particularly value the collaborative approach taken in PRIMM, the structure given to lessons, and the way that resources can be differentiated. This led to the assertion that PRIMM is an approach in school classrooms to improve learner outcomes in programming (Sentance et al., 2019).

PRIMM and classroom talk

In this paper I am focusing on a specific aspect of PRIMM, the role of language. According to Vygotsky, social interaction plays a critical role in children's learning (Vygotsky, 1978). Mediated activity promotes higher mental processes in three major forms of mediation: material tools, psychological tools (including language), and interaction with other human beings.

Classroom talk

Classrooms are full of talk — instructions, questions, explanations, as well as student–student and student–teacher dialogue. Teachers have an impact on the quality of the dialogue in their classroom and are an important model for pupils' use of language for reasoning (Mercer & Sams, 2006).

A range of models have been proposed to describe effective dialogue in the classroom. Dialogically organised instruction (Nystrand et al., 2003) sets out three ways the teacher can promote effective dialogue: through uptake (incorporating student ideas into subsequent questions of other students), through authentic questioning (used to explore views not test knowledge), and through high-level evaluation (where the teacher incorporates the response into elaborative comments).

This demonstrates that questioning is a key part of establishing effective dialogue, but teachers may limit their questions to the Initiation-Response-Feedback (IRF) style (Sinclair & Coulthard, 1975) to elicit answers from students where the answers to the questions are already known. Although a valid component of some lessons, these types of questions have been criticised for inhibiting classroom talk and the

development of ideas (Dawes, 2004; Wilkinson, 2013). A more dialogic approach focuses on open, exploratory questions.

Mercer and colleagues developed the idea of exploratory talk (Mercer, 1995), in which partners engage critically but constructively with each other's ideas. To measure the impact of exploratory talk, a series of research projects were conducted under the banner of Thinking Together. The research involved interventions that gave both teachers and students new skills in using language for reasoning. In the context of mathematics, this was shown to enable them to use language more effectively as a tool for working on maths problems together.

A recent study found that improving the quality of children's use of language for reasoning together improves their learning and understanding of mathematics (Mercer & Sams, 2006). Another study found that three aspects of teacher–student dialogue strongly predicted the performance of pupils aged 10 to 11 in standardised assessments: elaboration (building on contributions), querying (challenging a contribution) , and student participation (Howe et al., 2019).

In computing education, most of the literature relating to language and communication as a vehicle for learning centres on pair programming and peer instruction (Vahrenhold et al., 2019), both privileging classroom talk and purposeful dialogue. Research has shown that peer instruction positively impacts learning outcomes (Porter et al., 2011; Zingaro et al., 2014). Pair programming has been shown to improve program quality and confidence (Braught et al., 2008; McDowell et al., 2006), but in the school context it may depend on the way that the collaborative work is instantiated (Lewis, 2011.) An in-depth study of six pairs of 5th grade students in the context of pair programming revealed specific dialogue strategies used by students such as 'Let me help you' or 'Make suggestion' (Tsan et al., 2018). Another study which looked at interaction mechanisms in computing students' talk identified collaborative problem solving, conversations expressing excitement, and more social conversations (Israel et al., 2017). I am not aware of studies in programming education in school that specifically focus on dialogue and programming vocabulary.

Diethelm and Goschler (2015) highlight the lack of attention to computing-specific vocabulary and consider that specific items of computing vocabulary may be ambiguous or have different meanings in everyday life from their scientific meaning. They suggest a need for a meta-discourse around language such that pupils in school can learn to distinguish between everyday and scientific meanings of terms and that teachers should be more deliberate about vocabulary (Diethelm et al., 2018). There is clearly scope for more detailed investigation into how young learners acquire and use the technical vocabulary in programming.

The current study

In a PRIMM lesson, the intention is that a teacher facilitates productive classroom talk — encouraging discussion, modelling vocabulary use, asking in-depth questions. Having a common language to talk about programming constructs is important. Talking about a program and how it works helps learners to find the right vocabulary to use to articulate their understanding. Actually verbalising out loud the steps of a program that is difficult to understand can help learners to focus on atomic or smaller elements at a time. The analysis of data in the 2018 study inspired a new phase in research around PRIMM specifically focusing on the use of talk in the classroom and how it could support a deep understanding of programming constructs.

In the current study I am focusing specifically

on classroom talk in programming lessons in the context of PRIMM, seeking to investigate the quantity, quality, and content of classroom talk in programming lessons and teachers' perceptions of the impact of PRIMM on classroom talk. This work is in progress.

In the first phase of the study, I conducted interviews with 20 teachers who have been using PRIMM for different amounts of time in their classrooms. The findings are obviously impacted by the fact that much of the teaching in the last six months has been either remote or under varying degrees of social distancing in the classroom. Teachers were asked a number of questions around the following topics:

- The types of talk that take place in programming lessons
- The impact, if any, of PRIMM on the quantity and quality of talk in programming lessons
- Teachers' experience of students' use of programming terminology and vocabulary
- Approaches teachers use to foster discussion amongst students

To ensure that the study aligned to ethical guidelines (BERA, 2018) participants gave consent to the use of their data for specific purposes and full information was given. After transcription, participants were able to check their interview transcripts.

Early findings

The data was transcribed and analysed using thematic analysis (Braun & Clarke, 2006). The interviews were coded through an iterative and inductive process of coding, merging, and refining codes and re-coding (Nowell et al., 2016; Braun & Clarke, 2006).

There were some initial findings relating to the impact of PRIMM on classroom talk. Many teachers referred to the difference between 'pre-PRIMM' teaching and using the PRIMM approach. They commented that in PRIMM lessons there was less whole-class talk by the teacher, enhanced student-student dialogue, and that there was an increased focus on programming vocabulary.

Less talk by the teacher

One teacher, Teacher O[13], had found that when he initially taught programming he found that the approaches he was using were ineffective for his lower secondary school students, who were struggling. Since using PRIMM, he talks less now from the front of the class at the beginning of the lesson and gives students tasks to do that focus on the content of the code:

"In non-PRIMM lessons, I'm more talking about fundamentals and just talking through some real basics, like how to use a particular statement, and I'm talking to a whole group and then I find myself repeating myself going around the whole group. With PRIMM lessons, I'm getting kids to get onto the work and then I'm able to talk at a much higher level about what's going on in those particular programs." (Teacher O, secondary)

Both secondary and primary teachers noted the difference in the amount and nature of the whole-class talk:

"So I guess it lessens the me standing and talking at the front of the classroom because traditionally before this approach I probably would have put the code up on the board and then talked through it block by block and said, this is going to do this and this is going to do that, and so on and so forth, whereas it throws it out [and] it gets them in the driving seat straightaway…" (Teacher N, primary)

Student-student dialogue

Other teachers could specifically see the impact of the PRIMM approach in facilitating a more

[13] The 20 teachers in the study are referred to as Teacher A through to Teacher T.

questioning approach amongst students:

"And they'll go and say, but how did that work, why does that work, why is mine not doing that? And I think that PRIMM scaffolds that and allows them to have those discussions. Whereas, before, even with differentiation, they just could either do it or they couldn't do it." (Teacher C, secondary)

Several teachers highlighted the impact of verbalising on pupils' understanding. This aligns with research indicating that peer interaction improves learning:

"They are more engaged in the code itself and talking more about the code itself and what it does and that use of language definitely does aid their understanding." (Teacher L, primary)

Students' use of programming vocabulary

In the way that teachers discussed the use of programming-specific terms there was an indication that the use of PRIMM facilitated a more confident use of programming vocabulary:

"But what I have found is moving to PRIMM is the language the students are using is more improved because they know… Well, what's the variable? There's the variable… That is embedded over a period of time as well. " (Teacher G, secondary)

Other teachers were able to articulate why they thought it was important to use talk to verbalise how a program works, in that it gives learners a language with which they can express their understanding and supports the creation of a mental model. Finally, a teacher reflects on the fact that the focus on function and structure of code was enabling them to ask more advanced questions of the class or of individuals:
"I'm talking at a more advanced level to the whole group, but for less time. When I'm asking questions, they're usually much more useful and probing questions… " (Teacher O, secondary)

This study is in its early stages and I plan to report on it more in full in future publications. There are also plans to corroborate indicative findings with more research into actual classroom dialogue. However at this stage it appears that teachers believe that the use of PRIMM to structure lessons, with the collaborative, investigative exercises, gives an opportunity for more, and potentially more productive, dialogue. Teachers across the data set reflected that they have found this way of working enhances vocabulary use and a higher level of conceptual understanding.

Conclusion

Ad hoc reports indicate that the use of PRIMM to structure programming lessons has been widely adopted across schools in England, and also further afield in Australia, USA, and Malaysia. Teachers are able to create their own PRIMM materials by reworking their existing programming lessons around the PRIMM structure, or they can use or adapt resources that are being developed and shared by resource creators, including through the free, government-funded Teach Computing Curriculum[14] in England, which uses PRIMM in many of the programming units of work.

PRIMM is certainly a popular approach but further research is needed to examine what specific elements of it make a difference to learner outcomes. Variations of PRIMM are emerging which adapt the structure in different ways, some with more emphasis on keywords at the beginning (KPRIDE[15]), and others with a stage for evaluation at the end (TIME[16]).

What PRIMM has achieved for many teachers is an opportunity to reflect on, and examine, the value of the different activities that they use in the programming classroom. As all teachers know, it is being a reflective practitioner, and trying out different strategies, that improves teaching over time. To this extent it doesn't

[14] http://teachcomputing.org/curriculum

[15] https://blog.withcode.uk/2019/06/k-pride-tips-for-teaching-programming-so-everyone-can-make-progress/

[16] https://craigndave.org/programming-with-time/

really matter if every programming teacher uses a different acronym or variation on the theme, if they are able to reflect on the process of teaching and the impact on the individual students with whom they are working. Where PRIMM really comes into its own is to support new computing teachers, either new to teaching or new to computing, who are struggling with a class of young novice programmers, with varying levels of interest and engagement, where there is the potential for all the children to be "stuck" at exactly the same time and all be in need of teacher attention. The staged, gradual approach of PRIMM builds confidence and ownership of code one step at a time and focuses on understanding not completed artefacts.

In this paper, the particular focus has been on language due to the way that PRIMM promotes the practice, both by teachers and students, of talking out loud about what a program might do (function) and how it might do it (structure). The social and psychological functions of language are both drawn on to promote confidence as well as understanding, through talk and dialogue. An initial study into this aspect of PRIMM has shown some particular aspects of classroom talk that are facilitated by the PRIMM structure:

- Specific questioning about code leads to productive dialogue between students about programming code
- Teachers use whole class teaching differently at different stages of the PRIMM cycle
- Learning to use vocabulary to explain how a program works is challenging for students
- Teachers using PRIMM see part of their role as facilitating and focusing productive classroom talk

More research is needed on the way that classroom talk can support young novices learning programming, and beyond the context of PRIMM. It would be interesting to investigate whether an intervention based on exploratory talk (Mercer, 1995) would improve learning outcomes in computing as it has done in other subjects.

References

British Educational Research Association (BERA) (2018) *Ethical guidelines for educational research*. 4th edn. Available at: https://www.bera.ac.uk/researchers-resources/publications/ethical-guidelines-for-educational-research-2018 (Accessed: 20 December 2020).

Braught, G. Ebay, L. M. and Wahls, T. (2008). The effects of pair programming on individual programming skill. ACM SIGCSE Bulletin 40, 1(2008), 200–204.

Braun, V. and Clarke,V. (2006). Using thematic analysis in psychology. Qualitative research in psychology 3, 2 (2006), 77–101.

Campbell, D. T., and Stanley, J.C, Experimental and Quasi-Experimental Designs for Research on Teaching. In N. L. Gage (ed.), Handbook of Research on Teaching. Chicago: Rand McNally, 1963.

Dawes, L. (2004). Talk and Learning in classroom science. International Journal of Science Education 26, 6 (2004), 677–695.

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*, 113-124.Diethelm, I., & Goschler, J. (2015). Questions on Spoken Language and Terminology for Teaching Computer Science. Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education, 21–26.

Diethelm, I., Goschler, J., & Lampe, T. (2018). Language and Computing (In Sentance, S., Barendsen, E. and Schulte, C. Computer Science Education. Perspectives on Teaching and Learning in School. Bloomsbury Academic.. p. Chapter 15, 207-219).

Howe, C., Hennessy, S., Mercer, N., Vrikki, M., & Wheatley, L. (2019). Teacher–Student Dialogue During Classroom Teaching: Does It Really Impact on Student Outcomes? Journal of the Learning Sciences, 28(4–5), 462–512.

Israel, M., Wherfel, Q.M., Shehab, S., Melvin, O. & Lash, T. (2017). Describing Elementary Students' Interactions in K-5 Puzzle-based Computer Science Environments using the Collaborative Computing Observation Instrument (C-COI). In *Proceedings of the 2017 ACM Conference on International Computing Education Research (ICER '17). Association for Computing Machinery*, New York, NY, USA, 110–117.

Izu, C., Schulte, C., Aggarwal, A., Cutts, Q., Duran, R., Gutica, M., ... & Weeda, R. (2019). Fostering program comprehension in novice programmers-learning activities and learning trajectories. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education* (pp. 27-52).

Lahtinen, E., Ala-Mutka, K., J, H.-M., & rvinen. (2005). A study of the difficulties of novice programmers. Caparica, Portugal.

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

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

Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Mostr, J. E., Sanders, K., Sepp, O., I, Simon, B., & Thomas, L. (2004). A multi-national study of reading and tracing skills in novice programmers. Leeds, United Kingdom, 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 (ITiCSE '09). ACM, New York, NY, USA, 161–165.

Lopez, M., Whalley, J., Robbins, P., & Lister, R. (2008). Relationships Between Reading, Tracing and Writing Skills in Introductory Programming. 101–112.

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

Mercer, N. (1995). The guided construction of knowledge: *Talk amongst teachers and learners*. Multilingual matters.

Mercer, N., & Sams, C. (2006). Teaching Children How to Use Language to Solve Maths Problems. Language and Education, 20(6), 507–528.

Nowell, L. S., Norris, J. M., White, D. E., & Moules, N. J. (2017). Thematic analysis: Striving to meet the trustworthiness criteria. *International journal of qualitative methods, 16*(1), 1609406917733847.

Nystrand, M., Wu, L. L., Gamoran, A., Zeiser, S., & Long, D. A. (2003). Questions in Time: Investigating the Structure and Dynamics of Unfolding Classroom Discourse. Discourse Processes, 35(2), 135–198.

Qian, Y., & Lehman, J. (2017). Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education (TOCE), 18*(1), 1-24.

Porter, L., Bailey Lee, C., Simon, B., & Zingaro, D. (2011, August). Peer instruction: Do students really learn from peer discussion in computing?. In *Proceedings of the seventh international workshop on Computing education research* (pp. 45-52).

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. Computer Science Education, 13(2), 137–172.

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* (pp. 65-86).

Sentance, S. and 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. ACM. https://dl.acm.org/doi/10.1145/3137065.3137084

Sentance, S., Waite, J., & Kallia, M. (2019). Teaching computer programming with PRIMM: a sociocultural perspective. Computer Science Education, 29(2–3), 136–176.

Sinclair, J.M. & Coulthard, M. (1975). Towards an analysis of discourse: The English used by teachers and pupils. Oxford Univ Press.

Sweller, J. (1994). Cognitive load theory, learning difficulty, and instructional design. *Learning and instruction, 4*(4), 295-312.

Tsai, C.-Y., Yang, Y.-F., & Chang, C.-K. (2015). Cognitive Load Comparison of Traditional and Distributed Pair Programming on Visual Programming Language. *Educational Innovation through Technology (EITT), 2015 International Conference Of*, 143–146.Tsan, J., Lynch, C. F., & Boyer, K. E. (2018). "Alright, what do we need?": A study of young coders' collaborative dialogue. International Journal of Child-Computer Interaction, 17, 61–71

Vahrenhold, J., Cutts, Q. & Falkner, K. (2019). Schools (K–12). In The Cambridge Handbook of Computing Education Research, Fincher, S.A & Robins, A.V. (Eds.). Cambridge University Press, 547–583

Vainio, V., & Sajaniemi, J. (2007). Factors in Novice Programmers' Poor Tracing Skills. *Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, 236–240.

van Merriënboer, J. J. G., & Sweller, J. (2005). Cognitive Load Theory and Complex Learning: Recent Developments and Future Directions. *Educational Psychology Review*, 17(2), 147–177. https://doi.org/10.1007/s10648-005-3951-0

Vygotsky, L.S. (1962). Thought and word. In Studies in communication. Thought and Language, Vygotsky,L.S., Hanfmann, £. & Vakar, G. (Eds.). MIT Press, 119–153.

Vygotsky, L. S. (1978). *Mind in society: The development of higher psychological processes* Cambridge, Mass.: Harvard University Press.

Walqui, A. (2006). Scaffolding Instruction for English Language Learners: A Conceptual Framework. International Journal of Bilingual Education and Bilingualism, 9(2), 159–180. https://doi.org/10.1080/13670050608668639

Wilkinson, I. & Nelson, K. (2019). Role of Discussion in Reading Comprehension. In Hattie, J., Anderman, E.M. Visible Learning Guide to Student Achievement: Schools Edition. Routledge

Zingaro, D. (2014, March). Peer instruction contributes to self-efficacy in CS1. In *Proceedings of the 45th ACM technical symposium on Computer Science Education* (pp. 373-378).

Raspberry Pi