@InProceedings{prather2023scaffolding,
author="Prather, James and Homer, John and Denny, Paul and Becker, Brett A. and Marsden, John and Powell, Garrett",
editor="Keane, Therese and Lewin, Cathy and Brinda, Torsten and Bottino, Rosa",
title="Scaffolding Task Planning Using Abstract Parsons Problems",
booktitle="Towards a Collaborative Society Through Creative Learning",
year="2023",
publisher="Springer Nature Switzerland",
address="Cham",
pages="591--602",
isbn="978-3-031-43393-1"}

Author's version of paper published in IFIP World Conference on Computers in Education, WCCE 2022: Towards a Collaborative Society Through Creative Learning pp 591–602 Version of record: https://dx.doi.org/10.1007/978-3-031-43393-1_53

Scaffolding Task Planning Using Abstract Parsons Problems

James Prather[1 [0000−0003−2807−6042]], John Homer[1 [0000−0002−5067−4009]], Paul Denny[2 [0000−0002−5150−9806]], Brett A. Becker[3 [0000−0003−1446−647X]], John Marsden[4 [0000−0001−9584−035X]], and Garrett Powell[1 [0000−0002−3221−7015]]

[1] Abilene Christian University, Abilene Texas, USA james.prather@ucd.ie
[2] The University of Auckland, Auckland, New Zealand p.denny@aukland.ac.nz
[3] University College Dublin, Belfield Dublin 4, Ireland brett.becker@ucd.ie
[4] North Carolina State University, Raleigh North Carolina, USA jmmarsde@ncsu.edu

Abstract. Interest is growing in the role of metacognition in computing education. Most work to-date has examined metacognitive approaches of novices learning to code. It has been shown that novices navigate through discernible stages of a problem-solving process when working through programming problems, and that scaffolding can be beneficial. In this paper, we describe a novel scaffolding task aimed at guiding novices through a crucial stage of developing and evaluating a problem-solving plan. We presented novices with a problem statement before working through an Abstract Parsons Problem, where the blocks present structural elements rather than complete code, to aid high-level planning before writing code. Comparing groups who experienced this approach with those that did not, revealed that novices who worked on an Abstract Parsons Problem before coding were more successful in solving the task and demonstrated improved metacognitive knowledge related to task planning when asked to identify useful future problem-solving strategies. Our observations from two courses over two years suggest that scaffolding students through a planning step prior to coding can be beneficial for students. We provide directions for future work in exploring strategies for providing this type of guidance, including the use of different types of planning activities, and studying these effects at scale.

Keywords: Automated assessment tools · CS1 · Introductory programming · Novice programmers · Metacognition · Metacognitive awareness · Parsons problems

1 Introduction

Metacognition, or thinking about thinking, is an increasingly important topic in computing education [1]. It is an essential set of skills necessary for efficient and elegant programming problem-solving [2], and most successful novice students tend to display more metacognitive behaviors than their less successful

peers [3]. However, many novices lack metacognitive insight [4] making problem-solving laborious and difficult, particularly in programming which is inherently complex [5, 6] and could be made easier to learn [7]. Without the ability to step back and think through one's progress (or lack thereof) during problem-solving, novices may waste time on incorrect approaches or become lost and frustrated [8]. Applying metacognitive scaffolds in the context of novices learning to program, holds promise but is under-investigated [9].

We present a novel classroom activity for scaffolding novice programmers' metacognition through a specific stage of problem-solving as they envision their solution *before writing code*. We utilize a variation on Parsons Problems [18] for students to explicitly engage in planning out the structure of solutions after reading a problem prompt. Unlike other forms of Parsons Problems, we used abstract code statements (e.g. "while loop", "assignment", generic input/output, etc.) rather than syntactically complete lines of code. This novel use of the Parsons Problem framework, which we call "Abstract Parsons Problems", served two purposes. First, it removed syntactic clues from the blocks which researchers have previously observed being used to construct solutions without having a full understanding of the problem [20]. Second, these abstract statements aim to get students to plan the general structure of solutions before jumping into code – supported by prior research showing that prematurely writing code often inhibits success [13]. We hypothesize that this approach will: (1) assist students in successfully solving the corresponding programming task; and (2) encourage them to plan solutions to future programming tasks once the scaffolding is removed.

We explored the use of Abstract Parsons Problems in two introductory courses taught over two years, and have observed students solving programming problems both with and without their use. We frame our evaluation around two high-level goals: **Goal 1:** Explore whether students are more likely to successfully solve a programming task when they first work through a related Abstract Parsons Problem. **Goal 2:** Discover in what ways the Abstract Parsons Problem step affects students' perceptions of task difficulty and their identification of future problem-solving strategies. We address the first by comparing task completion rates when students work through an Abstract Parsons Problem beforehand with a control group. To address the second, we analyze student comments from treatment group surveys for evidence of metacognitive thoughts and behaviors related to task difficulty and strategies students plan to use in the future.

2 Background

Metacognition describes knowledge about a person's own cognitive control, including identifying past strategies that have been successful (or not), monitoring emotions and self-efficacy, and evaluating the validity of metacognitive knowledge based on feedback [1]. It has been used in psychology-based research for decades but has only begun to be researched in the context of computing education [1]. Regardless of discipline, metacognitive skills are accepted as aiding learning, but such skills do not necessarily transfer easily across different pur-

suits [10]. It is likely that the development of metacognitive skills needs to take place within the context of a student's learning in order to be effective.

Few studies have tried to explicitly attempt this. VanDeGrift et al. reported helping students think through their design process, arguing that programming courses should not only teach language/syntax but also metacognitive skills associated with programming [11]. More recent work has required students to solve a test case problem (i.e. converting input to output) after reading the problem prompt before they began coding [12]. Their results indicate that students who first solved the test case problem were more successful at solving the subsequent programming problem, avoided early misunderstandings that could potentially derail their problem-solving process, and verbalized more metacognitive behaviors than those in a control group. Their findings were confirmed in two separate replications [13, 14]. Another recent study found that both high- and low-performers can exhibit weak metacognitive accuracy, illuminating the potential for metacognitive interventions to benefit students of all skill levels [15].

2.1 Loksa's problem-solving framework

Loksa et al. proposed a six-stage programming problem-solving process based on psychology of programming literature [16, 17] which serves as a theoretical framework for novice metacognition while solving programming problems. Although nominally sequential, these stages are in practice revisited frequently as programmers refine a solution iteratively. The six stages are: (1) Reinterpret problem prompt; (2) Search for analogous problems; (3) Search for new solutions or adapt existing solutions; (4) Evaluate a potential solution; (5) Implement a solution; and (6) Evaluate implemented solution.

When solving programming problems, beginners are often not aware of where they are in the problem-solving process – demonstrating a lack of metacognition. In the present work, we focus on stage 4 of Loksa's framework *Evaluate a potential solution*, described as [17]: "With a solution in mind, programmers must evaluate how well this solution will address the problem. ... Without evaluating potential solutions, programmers may waste time writing code and integrating it into their program only to find that it does not actually solve their problem".

One way to scaffold novice programmer metacognition during stage 4 of the problem-solving process is with Abstract Parsons Problems (see Section 3). Loksa suggested that novice programmers could benefit from some sort of explicit scaffolding during the murky middle of the problem-solving process [17], and it has been suggested that Parsons Problems could be used for this purpose [8].

2.2 Parsons Problems

Parsons & Haden first introduced Parsons Programming Puzzles, a kind of drag-and-drop exercise involving code fragments, in 2006 [18]. Since then "Parsons Problems" have received much attention, appearing in dozens of computing education research papers [19]. Parsons Problems have been used as a scaffolding

step before students learn to code as they remove the requirement for constructing syntactically correct statements at the character level. However, this can oversimplify the problem-solving process. Weinman et al. observed that some students use syntactic clues in the blocks to find solutions without necessarily understanding the problem, and thus introduced Faded Parsons Problems in which parts of the provided code are incomplete [20]. Garcia et al. used Parsons Problems to scaffold the programming design process [21]. However, they defined blocks at a coarser level of granularity than what we propose, and did not seek to discuss the intervention in the context of novice programmer metacognition.

3 Methodology

Our observations were made during two consecutive years (referred to as Year 1 and Year 2) of a typical introductory programming or "CS1" [22] course using C++ taught at a small US university. The Abstract Parsons Problems were introduced approximately half-way through the course. Fig. 1 shows a schematic view of the evaluations where students completed a survey after attempting a programming problem. In each year, students solved the same programming problem both with and without the Abstract Parsons Problem step. The Year 1 evaluation used a between-subjects design, where each student was assigned to one of two groups, with only one prompted to solve the Abstract Parsons Problem after reading the problem statement. When presenting and discussing our results, we will refer to these two groups for Year 1 as the "Parsons" group and the "non-Parsons" group. The Year 2 evaluation used a within-subjects design where all students initially solved a programming problem (problem #1) without the Abstract Parsons Problem step, and then approximately one week later solved a problem of similar complexity (problem #2) that included the Abstract Parsons Problem step. In both years, students had a whole class period (50 minutes) to complete each programming task and the survey.

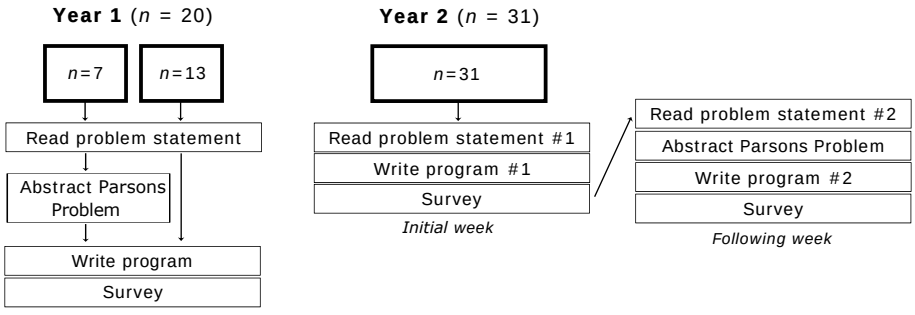

Fig. 1. Overview of the use of Abstract Parsons Problems to scaffold programming tasks and student surveys.

The programming problem in Year 1 asked students to write a program to display the set of distinct values coming from standard input until encountering -1. This problem prompt can be seen in Fig. 2. In Year 2, programming problem #1 asked students to print out a table of exponential numbers, and programming problem #2 (presented to students one week later) asked students to sequentially search for a value in an array. The difficulty of the two problems in Year 2 was roughly similar for where students were in the course when they encountered them, and commensurate with the topics covered in the two weeks.

The survey that students were prompted to complete after each programming problem was designed to help us address Goal 2 (see Section 1). In both years, the survey was identical and consisted of the following two reflective questions: 1) What did you find most difficult about this programming task? 2) What strategies do you think might be useful for solving similar problems in the future?

The Abstract Parsons Problem was delivered via a web-based interface, using js-parsons [23] integrated into Canvas, illustrated in Fig. 2. Students could drag individual blocks, corresponding to structural, data assignment, and input/output statements, from the left side of the screen (where they were initially presented in a random order) to the right side of the screen. We included more blocks than students would need since there are multiple solutions using sequential loops or nested loops, one array or two arrays, etc. Most of these blocks are not visible in Fig. 2. As described earlier, this "Parsons step" was designed to focus students on planning their solution prior to writing code. The Parsons interface did not provide students with feedback on the correctness of their plans, and students were free to end the planning step and begin coding at any time. Thus, it was acting as a supporting scaffold, and not feedback or otherwise explicit guidance towards a correct answer.

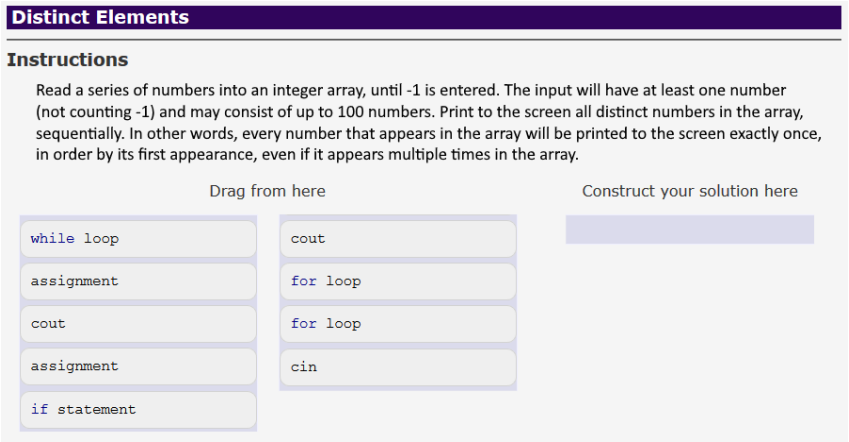

Fig. 2. Screenshot of the Abstract Parsons Problem shown to students in the Parsons group prior to coding.

4 Results

In Year 1, the 33 students (4 women, 29 men) enrolled in the course were originally assigned to the non-Parsons ($n = 16$) and Parsons ($n = 17$) groups alphabetically by surname, resulting in the four women being divided evenly across the two groups. We include in our data analysis all students who responded to the survey and who attempted the coding task. In total, we have complete data for 13 students in the non-Parsons group and 7 students in the Parsons group (as illustrated in Fig. 1). In Year 2, there were 40 students (8 women and 32 men) enrolled in the course. We include in our data analysis all students who responded to both surveys as well as attempted programming problems #1 and #2. In total, we have complete data for 31 students.

4.1 Task success (Goal 1)

A key measure of success for a programming task is solution correctness. Our first goal explores if the Parsons planning step helped students produce successful solutions. Two observations suggest that this step may have been helpful: in Year 1 data, the only students to successfully complete the programming task were in the Parsons group. In Year 2 data, a greater proportion of students solved the programming task when it followed the Abstract Parsons Problem step. We now explore these results in more detail.

Most Year 1 students made just a single submission to the grading system, towards the end of the timed task. All except one student made their first submission within 10 minutes of the session's end, with 65% of students making their first submission in the final 5 minutes. This pattern is consistent with the fact that many students were unable to complete the problem in the time allowed – a theme that emerged from our analysis of the survey responses (see Section 4.2), particularly for the non-Parsons group. In Year 2, nearly twice as many students successfully solved Problem #2 (which followed the Parsons planning step) compared to Problem #1.

In Year 1, only three students in the course successfully solved the programming task, all of whom were in the Parsons group. Given so few students completed the task, we manually examined the final code submissions in both groups to further understand the level of progress made. We coded the submissions for the presence of two patterns: a *nested loop*, and the use of *two arrays*. Given that the programming task required students to identify distinct elements in an input stream, we would expect a nested loop to be used either to check for repeated values in a record of prior inputs as new inputs are being read or to examine an array of all input values that have been stored once the stream is complete. Presence of a nested loop structure in the submitted code is therefore evidence of progress towards a viable solution. Although an elegant solution to the problem requires just a single array (either to store all values, or to keep track of only unique values), some students introduced two arrays – one to store all input values from the stream and the second to store the distinct values. Although use of a second array is not necessary, it is evidence of students executing a plan

to store the distinct values separately for printing the solution. In addition to manually coding the submissions for these two patterns, we also computed the number of lines of code and the number of variables that were used. Table 1 compares the final submissions across students in Year 1 with respect to these metrics, and shows the rate of success for students in Year 2.

Table 1. Comparison of metrics across the final code submissions made by students in each group. Where counts are given, percentages are shown (in brackets).

Year 1	non-Parsons ($n = 13$)	Parsons ($n = 7$)
Solved task (count)	0 (0.0%)	3 (42.9%)
Nested loop (count)	2 (15.4%)	5 (71.4%)
Two arrays (count)	3 (23.1%)	4 (57.1%)
Lines of code (mean)	33.38	38.66
Number of variables (mean)	3.15	4.22
Year 2	Problem #1	Problem #2
Solved task (count)	8 (25.8%)	13 (41.9%)

In addition to their greater success at solving the task in Year 1, we also found that a larger proportion of students in the Parsons group produced code that was closer to a working solution – particularly in terms of employing a nested loop – as well as producing more lines of code, and making more use of variables, on average. The number of students in each group is relatively small, and so we are cautious about drawing any conclusions regarding the generalizability of this result. Additionally we are not assuming that more lines and more variables are good without taking context into account. However here, given that most students did not submit a working solution, we see this as positive. Regardless, students in the Parsons group had to divide their time between the planning step and the coding step, and so we view it as promising that they achieved greater success on the coding task in this study.

4.2 Difficulties & future strategies (Goal 2)

Our second goal was to understand how the Abstract Parsons Problem, when used as a scaffolding step, might affect student perceptions of the programming task. In particular, we were interested in what students found most *difficult* about the task, and how the scaffolding might impact these perceived challenges. We were also interested in understanding how the planning step might influence the ways that students approach future programming problems, and in particular the *strategies* they choose to adopt. Our post-survey targeted both of these ideas. To analyze qualitative responses, we undertook a thematic analysis to identify the main patterns of meaning. We followed guidelines described by Braun &

Clarke [24], beginning with data familiarisation. After reading all responses, we assigned codes to each response and synthesized these into main themes.

4.3 Year 1

All students responded to both questions, with the exception of one student in the non-Parsons group who did not respond to the second, yielding 39 qualitative responses. We found two clear themes, one of which was prominent in the non-Parsons group and the other prominent in the Parsons group.

The first theme related to *time pressure*, which was reported as a difficulty of the task. This theme was very common in responses from students in the non-Parsons group, but *not* from students in the Parsons group. This is surprising, because both groups had an identical time limit for the programming task, yet students in the Parsons group also had to engage with the Parsons Problem step prior to coding, leaving less time for the coding itself. In other words, the students in the Parsons group were asked to do more in the same time, yet they raised time pressure as a challenge less frequently. Time pressure was mentioned as the most difficult aspect of the programming task by 6 of 13 students in the non-Parsons group (46%), compared with just 2 of the 7 students in the Parsons group (29%). As an example, one student in the non-Parsons group expressed feeling time pressure as: *"I felt like the most difficult part was the time constraint. The entire time I was working I could feel the pressure of the time running out. I feel like if I had more time to work and debug I could have solved the problem"*.

The second theme related to *task planning* emerged from responses about strategies identified as being useful for solving similar problems in the future. In this case, students in the Parsons group were more likely to identify that some form of planning would be useful to them in the future. Of the 7 students in the Parsons group, 4 identified task planning as being an important future strategy (57%), compared with just 2 of the 13 students in the non-Parsons group (15%). A student from the Parsons group expressed the importance of task planning as: *"I think planning out what I am going to do before I actually write the code will help. It allows for me to think out the problem and I might even be able to figure out the problem without even writing code first"*.

4.4 Year 2

All students responded to both questions for problems #1 and #2, for a total of 62 qualitative responses. In addition to the same themes from Year 1, we saw some new themes which we present below.

Problem #1. The first theme for Problem #1 was about *domain knowledge*. Fourteen of the 31 students wrote something that indicated they did not have enough domain knowledge to complete the task. Interestingly, none of these students completed the program within the time limit. From understanding nested loops to proper formatting using the `setw()` function (a function used in programs since week 2 of the course), almost half of students verbalized a lack of course content understanding. One student exemplified this as: *"Having more*

experience and practicing more. If I had more practice with nested for loops I might've been able to figure it out".

Although there was no metacognitive intervention in Problem #1, the second theme that emerged was related to *task planning* as vague metacognitive statements. Six of the 31 students wrote about breaking the problem down or ensuring they understood the prompt. Five completed the program within the time limits; one did not. We labeled these as vague because there were no specific examples of what to do, only evidence that they understood some kind of strategy was necessary as exemplified by this response: *"Understanding how to put multiple different ideas into a complex idea. I was able to understand the basic concepts, but had trouble combining those concepts to solve the problem."*

Problem #2. The first theme to emerge from Problem #2 was also about a lack of *domain knowledge*. Thirteen of the 31 students wrote about their lack of understanding of course content, such as arrays. This is similar to what we saw in Problem #1. Similarly, the second theme for Problem #2 was about *task planning*. In contrast to the vague statements from Problem #1 a week earlier, 10 of 31 students responded with concrete ideas about how to aid their problem-solving process in the future. One example was: *"Writing down the goals of the code with pen/paper, and structuring where you might need a loop using Parson's is something I will try to practice in future coding projects as well".*

We also tagged all student statements with one of three codes in Problems #1 and #2: non-cognitive, cognitive, and metacognitive. A statement was *non-cognitive* if students wrote things like "I don't know."; *cognitive* if the student mentioned needing to better understand course concepts or other domain knowledge; and *metacognitive* if the student showed a reflective stance toward their own mental problem-solving process. We then mapped the change from Problem #1 to #2. We believe that non-cognitive is the least desired response type, followed by cognitive, then metacognitive. Therefore, a positive change is movement towards metacognitive (non-cognitive to cognitive, non-cognitive to metacognitive, or cognitive to metacognitive). A negative change is movement toward non-cognitive. Neutral change is where the student gave the same type of response both times. In total we recorded 12 positives, 7 negatives, and 13 neutrals.

5 Limitations & Future Work

In Year 1, the COVID-19 pandemic proved highly disruptive due to a switch to a hybrid mode of teaching in the middle of the semester. Originally, we envisaged running the task as an in-class, supervised activity, however due to the switch some students attended class on campus while others studied remotely. We found that the students studying remotely, and thus unsupervised, were generally less likely to engage with course activities and abandoned some more readily. During the session in which we ran this study, around 40% of the class either did not engage or did not complete the tasks, resulting in less data than originally envisioned. Nevertheless, our allocation of students into the two groups was such

that a similar percentage of students in each group were studying remotely. In Year 2, students were supervised in-class for both of the programming problems.

In Year 1, we did not randomly assign our original participant pool to the two groups, but allocated students to them based on surname. A post-hoc analysis indicated no significant differences in the course marks of students in these groups prior to the study commencing. Additionally, we have no data regarding interactions between students and the interactive Parsons Problem tool. Thus, we do not know how long students in this group spent with the Parsons Problem, or how sophisticated their planning was prior to programming.

One avenue of future work focuses on metacognitive strategies. This involves further investigating the use of Parsons Problems to scaffold the problem-solving process. We introduced Abstract Parsons Problems to do this, but as discussed in Section 2, there are many different variations of Parsons Problems. Possible questions include: *Do other types of Parsons Problems cause an increase in verbalization of metacognitive strategies when used as a pre-coding scaffold?* and *Is there a similar effect with other types of planning activities or is there something specific about Parsons Problems causing the effect?* Additionally, our findings warrant a deeper, qualitative, exploration into the types of metacognitive strategies that design activities like Parsons Problems elicit.

Another avenue of future work involves scale, invigilation, and complexity. As noted in Section 5, our sample size was constrained by the limitations of running a live classroom study during the COVID-19 pandemic. Future work should replicate this study at scale. This experiment could also be run as a non-invigilated homework activity with a multi-day time window. Finally, increasing the complexity of the study itself, including crossover designs or conducting it longitudinally, could help establish related factors and what types of programming problems benefit from this kind of activity [14].

6 Implications & Conclusions

Our original hypothesis was that novice programmers would benefit from explicit scaffolding during the problem-solving process with higher task completion and an increase in metacognitive behaviors. Using a Parsons Problem to provide this metacognitive scaffolding is a novel contribution of our work. Our findings not only support our hypothesis, but they are somewhat surprising – despite having more time for writing code, when not presented with the Parsons Problem students tended to be less successful at completing the programming task, and were more likely to report being under time pressure. These findings could positively contribute to student learning in several ways. First and foremost it provides evidence that scaffolding learning during explicit metacognitive problem-solving stages can benefit students.

Recent research has shown metacognition to be a critical skill for novice programmers to develop alongside their domain knowledge. These results suggest that inserting a planning step with an interactive tool, in this case Abstract Parsons Problems, to guide students through this stage of the problem-solving

process could help to scaffold metacognitive skills in novices. Due to the limitations of our study, we interpreted these findings as positive indicators that warrant future work in metacognitive strategies, non-invigilated assignments, and increasing the scale and complexity of future investigations.

References

1. Prather, J., Becker, B.A., Craig, M., Denny, P., Loksa, D., Margulieux, L.: What do we think we think we are doing? Metacognition and self-regulation in programming. In: Proc. of the 2020 ACM Conference on International Computing Education Research. p. 2â€“13. ICER '20, ACM, NY, NY, USA (2020).
2. Loksa, D., Ko, A.J.: The role of self-regulation in programming problem solving process and success. In: Proc. of the 2016 ACM Conference on International Computing Education Research. p. 83â€“91. ICER '16, ACM, NY, NY, USA (2016).
3. Bergin, S., Reilly, R., Traynor, D.: Examining the role of self-regulated learning on introductory programming performance. In: Proc. of the 1st International Workshop on Computing Education Research. p. 81â€“86. ICER '05, ACM, NY, NY, USA (2005).
4. Roll, I., Holmes, N.G., Day, J., Bonn, D.: Evaluating metacognitive scaffolding in guided invention activities. Instructional Science **40**(4), 691–710 (Jul 2012).
5. Luxton-Reilly, A., Simon, Albluwi, I., Becker, B.A., Giannakos, M., Kumar, A.N., Ott, L., Paterson, J., Scott, M.J., Sheard, J., Szabo, C.: Introductory programming: A systematic literature review. In: Proc. Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education. pp. 55–106. ITiCSE 2018 Companion, ACM, NY, NY, USA (2018).
6. Becker, B.A.: What does saying that 'programming is hard' really say, and about whom? Commun. ACM **64**(8), 27â€“29 (Jul 2021).
7. Karvelas, I., Li, A., Becker, B.A.: The effects of compilation mechanisms and error message presentation on novice programmer behavior. In: Proceedings of the 51st ACM Technical Symposium on Computer Science Education. p. 759â€“765. SIGCSE '20, ACM, NY, NY, USA (2020).
8. Prather, J., Pettit, R., McMurry, K., Peters, A., Homer, J., Cohen, M.: Metacognitive difficulties faced by novice programmers in automated assessment tools. In: Proc. of the 2018 ACM Conference on International Computing Education Research. pp. 41–50. ICER '18, ACM, NY, NY, USA (2018).
9. Loksa, D., Margulieux, L., Becker, B.A., Craig, M., Denny, P., Pettit, R., Prather, J.: Metacognition and self-regulation in programming education: Theories and exemplars of use. ACM Trans. Comput. Educ. **22**(4) (Sep 2022).
10. Bandura, A.: Perceived self-efficacy in cognitive development and functioning. Educational psychologist **28**(2), 117–148 (1993).
11. VanDeGrift, T., Caruso, T., Hill, N., Simon, B.: Experience report: Getting novice programmers to THINK about improving their software development process. In: Proceedings of the 42nd ACM Technical Symposium on Computer Science Education. p. 493â€“498. SIGCSE '11, ACM, NY, NY, USA (2011).
12. Prather, J., Pettit, R., Becker, B.A., Denny, P., Loksa, D., Peters, A., Albrecht, Z., Masci, K.: First things first: Providing metacognitive scaffolding for interpreting problem prompts. In: Proc. of the 50th ACM Technical Symposium on Computer Science Education. pp. 531–537. SIGCSE '19, ACM, NY, NY, USA (2019).

13. Denny, P., Prather, J., Becker, B.A., Albrecht, Z., Loksa, D., Pettit, R.: A closer look at metacognitive scaffolding: Solving test cases before programming. In: Proc. of the 19th Koli Calling International Conference on Computing Education Research. Koli Calling '19, ACM, NY, NY, USA (2019).
14. Craig, M., Petersen, A., Campbell, J.: Answering the correct question. In: Proc. of the ACM Conference on Global Computing Education. pp. 72–77. CompEd '19, ACM, NY, NY, USA (2019).
15. Lee, P., Liao, S.N.: Targeting metacognition by incorporating student-reported confidence estimates on self-assessment quizzes. In: Proc. of the 52nd ACM Technical Symposium on Computer Science Education. pp. 431–437. SIGCSE '21, ACM, NY, NY, USA (2021).
16. Loksa, D., Ko, A.J., Jernigan, W., Oleson, A., Mendez, C.J., Burnett, M.M.: Programming, problem solving, and self-awareness: Effects of explicit guidance. In: Proc. of the 2016 CHI Conference on Human Factors in Computing Systems. pp. 1449–1461. ACM (2016).
17. Loksa, D.: Explicitly Training Metacognition and Self-Regulation for Computer Programming. Ph.D. thesis, University of Washington (2020).
18. Parsons, D., Haden, P.: Parson's programming puzzles: A fun and effective learning tool for first programming courses. In: Proc. of the 8th Australasian Conference on Computing Education - Volume 52. p. 157â€“163. ACE '06, Australian Computer Society, Inc., AUS (2006).
19. Du, Y., Luxton-Reilly, A., Denny, P.: A review of research on Parsons problems. In: Proc. of the 22nd Australasian Computing Education Conference. pp. 195–202. ACE'20, ACM, NY, NY, USA (2020).
20. Weinman, N., Fox, A., Hearst, M.: Exploring challenging variations of Parsons problems. In: Proc. of the 51st ACM Technical Symposium on Computer Science Education. p. 1349. SIGCSE '20, ACM, NY, NY, USA (2020).
21. Garcia, R., Falkner, K., Vivian, R.: Scaffolding the design process using Parsons problems. In: Proc. of the 18th Koli Calling International Conference on Computing Education Research. Koli Calling '18, ACM, NY, NY, USA (2018).
22. Becker, B.A., Quille, K.: 50 years of CS1 at SIGCSE: A review of the evolution of introductory programming education research. In: Proc. of the 50th ACM Technical Symposium on Computer Science Education. pp. 338–344. SIGCSE '19, ACM, NY, NY, USA (2019).
23. Ihantola, P., Karavirta, V.: Open source widget for Parson's puzzles. In: Proc. of the 15th Annual Conference on Innovation and Technology in Computer Science Education. p. 302. ITiCSE '10, ACM, NY, NY, USA (2010).
24. Braun, V., Clarke, V.: Using thematic analysis in psychology. Qualitative Research in Psychology **3**(2), 77–101 (2006).