Investigating the Affect and Effect of Adaptive Parsons Problems

Barbara Ericson
University of Michigan
Ann Arbor, Michigan, USA
barbarer@umich.edu

Austin McCall
University of Michigan
Ann Arbor, Michigan, USA
austin.mccall32@gmail.com

Kathryn Cunningham
University of Michigan
Ann Arbor, Michigan, USA
kicunn@umich.edu

ABSTRACT

In a Parsons problem the learner places mixed-up code blocks in the correct order to solve a problem. Parsons problems can be used for both practice and assessment in programming courses. While most students correctly solve Parsons problems, some do not. Unsuccessful practice is not conducive to learning, leads to frustration, and lowers self-efficacy. Ericson invented two types of adaptation for Parsons problems, *intra-problem* and *inter-problem*, in order to decrease frustration and maximize learning gains. In *intra-problem* adaptation, if the learner is struggling, the problem can dynamically be made easier. In *inter-problem* adaptation, the next problem's difficulty is modified based on the learner's performance on the last problem. This paper reports on the first observational studies of five undergraduate students and 11 secondary teachers solving both intra-problem adaptive and non-adaptive Parsons problems. It also reports on a log file analysis with data from over 8,000 users solving non-adaptive and adaptive Parsons problems. The paper reports on teachers' understanding of the intra-problem adaptation process, their preference for adaptive or non-adaptive Parsons problems, their perception of the usefulness of solving Parsons problems in helping them learn to fix and write similar code, and the effect of adaptation (both intra-problem and inter-problem) on problem correctness. Teachers understood most of the intra-problem adaptation process, but not all. Most teachers preferred adaptive Parsons problems and felt that solving Parsons problems helped them learn to fix and write similar code. Analysis of the log file data provided evidence that learners are nearly twice as likely to correctly solve adaptive Parsons problems than non-adaptive ones.

CCS CONCEPTS

• **Social and professional topics** → **Informal education**; **Student assessment**; **K-12 education**; **Adult education**.

KEYWORDS

Parsons problems, Parson's problems, adaptation, self-efficacy

ACM Reference Format:
Barbara Ericson, Austin McCall, and Kathryn Cunningham. 2019. Investigating the Affect and Effect of Adaptive Parsons Problems. In *19th Koli Calling International Conference on Computing Education Research (Koli Calling '19), November 21–24, 2019, Koli, Finland*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3364510.3364524

Koli Calling '19, November 21–24, 2019, Koli, Finland

ACM ISBN 978-1-4503-7715-7/19/11…$15.00
https://doi.org/10.1145/3364510.3364524

1 INTRODUCTION

Learning to program can be difficult. Drop out and failure rates in many introductory computing classes at the college level are high with an average pass rate worldwide of only 67% [7, 40]. Novices have spent many frustrating hours trying to figure out why their program does not compile [6]. College students that encounter errors while programming experience negative emotions that can impact their self-efficacy [28]. Negative experiences in courses tend to affect women more than men, which may be one reason that women are underrepresented in computing [16, 30].

Parsons problems may help novices learn to recognize and fix common errors as well as learn common algorithms. In a Parsons problem the correct code to solve a problem is provided, but is divided into blocks of statements, and the blocks are mixed up [31]. The blocks must be placed in the correct order as shown in Figure 1. Problems can also contain additional blocks called *distractors*, which are not needed in a correct solution. These distractors may contain common syntactic or semantic errors. *Two-dimensional Parsons problems* also require the learner to indent the blocks correctly [26]. Some languages, like Python, use indentation to indicate the body of a block for control structures.

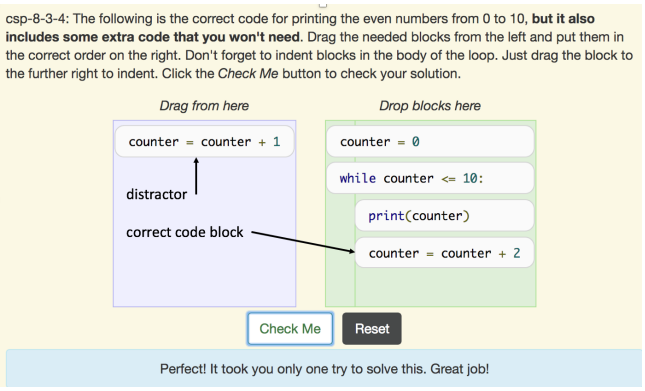

Figure 1: A two-dimensional Parsons problem with the solution on the right and a distractor on the left

Prior research on Parsons problems embedded in a free ebook showed that while most students successfully solved the Parsons problems (80 - 90%), some students gave up and never solved them [20]. Unsuccessful practice is not conducive to learning [9]. It can lead to frustration [28] and lower self-efficacy [5]. The goal is to keep the learner in Vygotsky's zone of proximal development (ZPD)

[39]. The zone of proximal development is defined as the difference between what a learner can accomplish with help versus without help. Learning is optimized in the ZPD because the problem is challenging for the learner, but not frustrating.

Intelligent tutoring systems (ITS) also attempt to keep the learner in the zone of proximal development [1], but these systems take a long time to develop [12] and are not widely used [2]. Conversely, a Parsons problem can be created in minutes and thousands of students are already solving them in free interactive ebooks. Ericson invented two types of adaptation for Parsons problems to optimize learning. In *intra-problem adaptation* if the learner is struggling to solve the current problem, it can dynamically be made easier. In *inter-problem adaptation* the difficulty of the next problem is modified based on the learner's performance on the previous problem.

This study attempted to answer the following research questions:

- RQ1: Do learners understand the intra-problem adaptation process?
- RQ2: Do learners prefer adaptive or non-adaptive Parsons problems?
- RQ3: Do learners perceive that solving Parsons problems with distractors helped them learn to fix code with similar errors and write similar code?
- RQ4: What is the effect of adaptation (both intra-problem and inter-problem) on learners' ability to correctly complete Parsons problems?

This paper uses both qualitative and quantitative methods to study the affect and effect of non-adaptive Parsons problems and adaptive Parsons problems. It used an observational study to answer RQ1 - RQ3 and a log file analysis to answer RQ4. It reports on 1) the first observational studies of undergraduate students and secondary teachers solving intra-problem adaptive Parsons problems. It also reports on 2) a log file analysis of over 8,000 users who solved adaptive (both intra-problem and inter-problem) and non-adaptive Parsons problems in two free interactive ebooks. The observational studies were part of a series of studies for a dissertation on the effectiveness and efficiency of solving adaptive and non-adaptive Parsons problems [18, 19, 21].

2 RELATED WORK

Practice is crucial in developing expertise [33, 36]. However, it needs to be the right kind of practice. It is possible to spend many hours practicing without any improvement in ability. It is important to help learners succeed during practice because unsuccessful practice is not conducive to learning [13]. Novice programmers have reported spending hours trying to find a simple syntax error in a program [6].

During learning new information must be processed in working memory and then added to the knowledge representations (*schemas*) that exist in long-term memory [35]. However, cognitive load theory posits that working memory has a limited capacity, and if that capacity is needed entirely to process new information, it cannot be used to build schemas [34]. Instructional material can be designed to reduce cognitive load. Parsons problems, as a type of code completion problem, should have a lower cognitive load than a problem that requires the learner to write the code from scratch, because the problem space is more constrained [37].

2.1 Adaptive Practice

Dynamically adaptive practice, where the practice problems are adapted based on the learners prior performance, improves learning, takes less time, and increases engagement compared to non-adaptive practice problems [11]. Successful adaptive practice should keep the learner in Vygotsky's zone of proximal development, which is optimal for learning [8].

Intelligent tutoring systems (ITS) also try to keep the learner challenged, but not frustrated [1]. They include both an inner loop and outer loop [38]. The inner loop executes once per step taken by a student when solving a problem and provides feedback and hints. This is similar to intra-problem adaptation, except that the hints in adaptive Parsons problems are implicit rather than explicit. The inner loop can also be used to assess the student's mastery of concepts and update the student model, which is used by the outer loop to select the next problem based on the student's performance. This is different than inter-problem adaptation, which does not select the next problem based on the student's past performance, but instead modifies the difficulty of the next problem.

2.2 Research on Parsons Problems

Knowing the features that make Parsons problems more or less difficult is essential for successful adaptation. Several researchers have studied the impact of distractors in Parsons problems. These studies provided evidence that Parsons problems with distractors are harder to solve than those without distractors [23, 24], that visually pairing a distractor block and its corresponding correct block makes the problem easier [15], and that adding more distractors makes a problem harder [15]. Several studies have indicated that providing the indentation or structure of the problem makes the problem easier, while requiring the learner to provide the indentation makes the problem harder [15, 26]. Increasing the number of blocks in a Parsons problem increases the difficulty of the problem, since it increases the number of possible combinations.

3 OBSERVATIONAL STUDY MATERIALS

The observational study used the first five chapters of a free online teacher ebook for the Advanced Placement (AP) Computer Science Principles (CSP) course [17]. The AP CSP course is offered in secondary schools and is intended to be equivalent to a college level course in computer science for non-majors. This course includes basic programming concepts such as variables, loops, conditionals, and functions. The teacher ebook was created to provide secondary teachers with free professional development in order to increase their knowledge of programming and confidence in their ability to teach programming [17]. To create sufficient material for this study, Ericson added material to chapter five from the later ebook chapters on loops and nested loops and also created new Parsons problems.

Two versions of chapter five of the ebook were created, each with 10 intra-problem adaptive Parsons problems and 10 non-adaptive Parsons. The only difference between the two versions was which Parsons problems were adaptive. In one version the sequence was "ANNAANNAANNAANNAANNA" where "A" means adaptive and "N" means non-adaptive. In the other version the sequence was "NAANNAANNAANNAANNAAN". Notice that if a problem is

adaptive in one version, it is non-adaptive in the other version. Within the ebook content, problems were grouped into pairs of adaptive and non-adaptive (after the first problem), for future tests of inter-problem adaptation, since inter-problem adaptation only affects the next adaptive Parsons problem. Participants were randomly assigned to one of the two versions of the ebook.

Chapter five included Parsons problems, fix code problems, and write code problems. In a fix code problem the learner must fix errors in the code as shown in Figure 2. One of the write code problems is shown in Figure 3. There were several Parsons problems with distractors, before two fix code problems, which were followed by a write code problem. The distractors included common errors such as the wrong case, unmatched parentheses, missing parentheses, and missing colons. The goal was to use distractors in the Parsons problems to help novices learn to recognize and fix the types of syntax errors that experts easily fix, but that can cause novices hours of frustration [6].

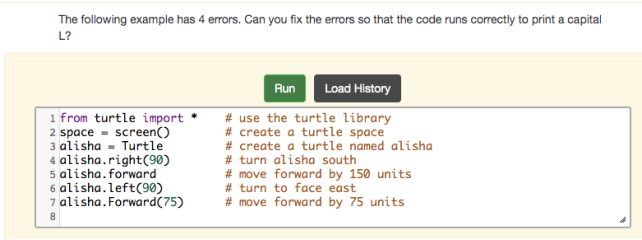

Figure 2: One of the fix code problems with four errors.

Figure 3: One of the write code problem

4 ADAPTIVE PARSONS PROBLEMS

The observational study only used intra-problem adaptation since it is visible to the learner and initiated by the learner. Inter-problem adaptation happens automatically before the next problem is presented to the learner, which means that it is invisible to the learner. Both types, intra-problem and inter-problem adaptation were used in the adaptive Parsons problems in the log file analysis.

4.1 Intra-Problem Adaptation

In intra-problem adaptation, if the learner is struggling to solve the current Parsons problem it can dynamically be made easier by disabling distractors, providing indentation, and/or combining blocks. Adaptation was only available after the learner had checked three full solutions. A full solution has at least the required number of blocks. When the learner clicked the *Check Me* button to check an incorrect solution for the third time an alert notified the user that help was available. The learner had to click on the *Help Me* button to initiate the adaptation. If the learner clicked on the *Help Me* button before completing three full attempts, an alert was shown that said that, *"You must make at least three distinct full attempts at a solution before you can get help"*.

When the learner initiated an adaptation, an alert notified the user which type of change it was about to perform: disable a distractor, provide indentation, or combine two blocks. The change did not occur until the learner clicked the *Close* button on the alert window. This was intended to help the learner understand and track the change. Only one change was made each time the user clicked on the *Help Me* button. The system would disable a distractor, provide indentation, or combine two blocks into one.

Figure 4 shows a Parsons problem when it is first displayed. This problem has four distractor blocks shown paired with the correct code blocks. Notice the purple edges that pair the correct code block and the incorrect (distractor) code block. The correct code is randomly shown above or below the distractor block. This problem asks the learner to stamp three turtle shapes in a line. The learner constructs the solution on the right side.

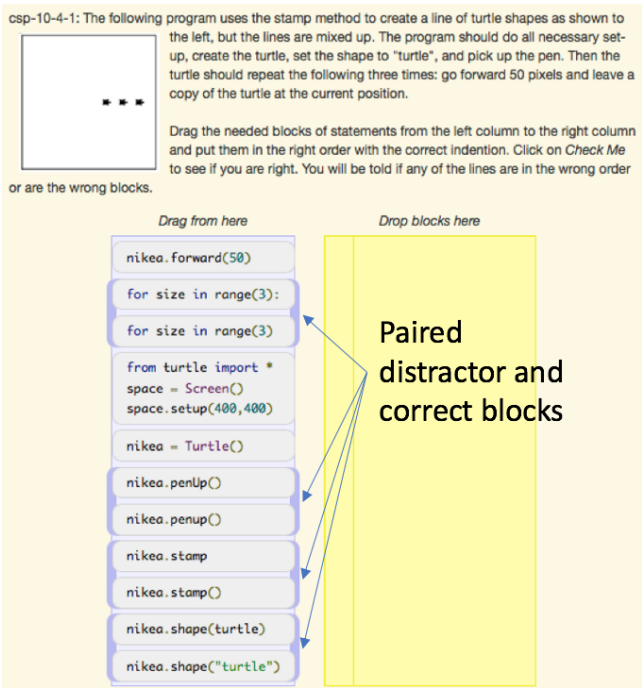

Figure 4: Problem 13 in the experiment with four distractor blocks shown paired with the correct code blocks.

If the learner had made at least three full attempts at a solution when trying to solve this problem, the learner could ask for help. If help (adaptation) was requested and any distractor was still enabled it would disable a distractor, meaning that it grayed out over time and would not respond to further attempts to move it as shown in Figure 7. If the distractor was in the solution area, it would first move slowly from the solution area on the right to the source area on the left as shown in Figure 5 before being disabled. Animation is useful for grabbing attending and conveying a change over time [4, 10].

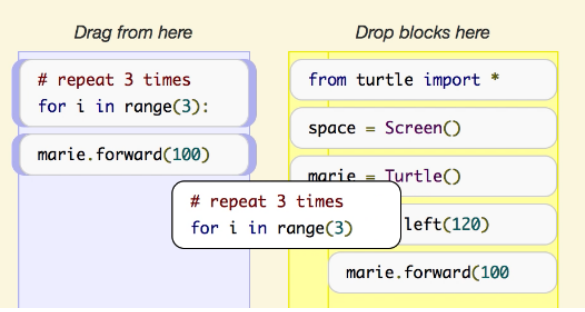

Figure 5: A distractor (missing colon) moving back to the source area on the left from the solution area on the right

If the distractor was originally shown paired with the correct code, and that correct code block was still in the source area on the left, then the distractor was again paired with the correct code as shown in Figure 6. This was an implicit hint that the learners should use the paired correct block rather than the disabled distractor. If no distractors were used in the solution, it would disable (gray out) a distractor in the source area on the left.

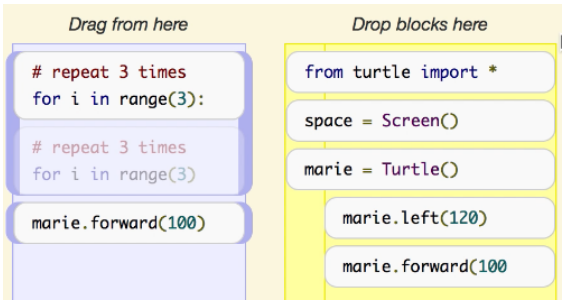

Figure 6: A distractor (missing colon) after it has been paired with the correct code block on the left and disabled (grayed out)

If the user asked for help and there were no active distractors, indentation was provided as shown in Figure 7. Space was slowly added to the left before the code. After indentation was provided the blocks could no longer be indented by the learner.

If the user asked for help when there were no active distractors and indentation was not needed or had already been provided, then two blocks were combined into one. The block that was about to be added below another block slowly moved into place. The blocks were then redrawn as one block as shown in Figure 8. This type of adaptation would only happen if there were more than three blocks left. In a problem with three blocks, there are only six possible combinations. Since the first block of the solution is usually in the correct position, this means that there are really only two possible combinations to try.

4.2 Inter-problem Adaptation

Inter-problem adaptation was added to the adaptive Parsons problem software after the observational study. In inter-problem adaptation the user's performance on the previous problem is used to

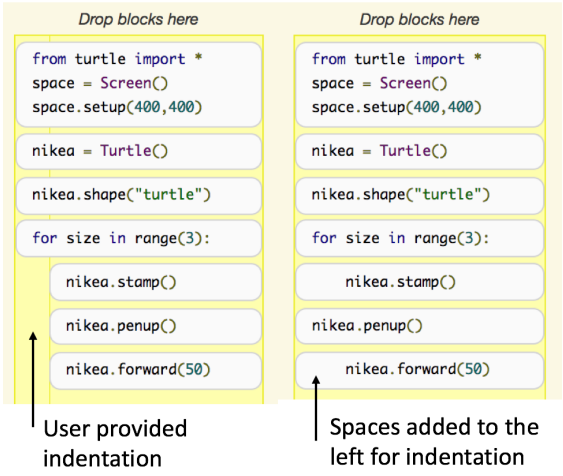

Figure 7: User provided indentation on the left and after indentation has been provide on the right.

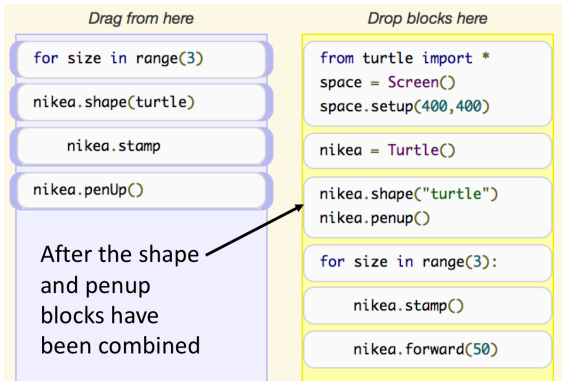

Figure 8: After the shape and penup blocks have been combined into one block.

modify the difficulty of the next problem. If the learner solved the last Parsons problem in only one attempt, then the next Parsons problem was made more difficult by un-pairing distractors (randomly mixing them in with the correct code) and by using all available distractors as shown in Figure 9.

If it took the learner four or five attempts to solve the last Parsons problem, then on the next problem the distractors would be shown paired with the correct code blocks as shown in Figure 4. If it took the learner six to seven attempts to solve the last problem, then 50% of the available distractors were removed and the remaining distractors were shown paired with the correct code blocks on the next problem. If it took the learner eight or more attempts to solve the last problem, then all distractors were removed from the next problem as shown in Figure 10. Inter-problem adaptation is similar to the outer loop of intelligent tutoring systems, which chooses the next problem from a set of possible problems based on the user's prior performance. However, in inter-problem adaptation the difficulty of the next problem is modified rather than which problem is next.

Figure 9: Parsons problem with all distractor blocks randomly mixed in with the correct code blocks.

Figure 10: Parsons problem without any distractor blocks.

5 PILOT STUDY

To test that the study materials were the right level of difficulty, three undergraduate students were observed as they individually worked through the materials. The undergraduates received five points of extra credit for taking part in the study. They had all previously taken an undergraduate CS1 course in Python, so they were more experienced than the intended audience. None of the three students used the adaptation, which indicated that the problems might be too easy. Four of the Parsons problems were dropped since the students solved those problems easily. More distractors were added to the remaining problems and half of the problems were changed to randomly mix in the distractor blocks with the correct code blocks, in order to make the problems harder. Ericson observed two more undergraduate students and found that these students used the adaptation. This indicated that the problems were at a level of difficulty that would make it likely that the teachers, who had less textual programming experience than the students, would also need to use intra-problem adaptation to solve at least some of the problems.

6 OBSERVATIONAL STUDY

Observations provide rich details on individual learners' understanding and perceptions. The goals for this study were to 1) check if the learners understood the intra-problem adaptation process and could make use of the implicit hints provided by the adaptation, 2) determine learners' preference for adaptive or non-adaptive Parsons problems, and 3) understand learners' perception of the usefulness of solving Parsons problems with distractors for helping them learn to fix and write similar code.

This was a within-subjects observational study of 11 teachers with less than three months of textual programming experience. They were randomly assigned to one of two versions of chapter five of an ebook for teachers. The only difference between the two versions was which Parsons problems were adaptive.

6.1 Recruitment

Ericson sent email to a list of over 500 secondary teachers and also sent email to other instructors who lead teacher professional development to ask them to forward the email to their participants. The email stated that the study was for teachers with less than three months of textual programming experience over the last two years. Teachers who completed the study earned a $75 gift card. Teachers had to complete the first four chapters of the ebook on their own and then were observed as they worked through the fifth chapter. They were asked to complete the first four chapters by the end of August 2017.

6.2 Procedure

Interested teachers first gave consent online. Next they filled out a survey, which asked for demographic information including gender, age, race, certification/license, number of years teaching and the number of years teaching computing courses. This information is shown in Table 1. The survey also asked if they had less than three months of textual programming experience, and their familiarity with computing concepts such as variables, loops, conditionals, and lists. Qualified teachers were randomly assigned to one of the two versions of the study ebook and were sent an email welcoming them to the study. The email contained the URL for the study ebook, their login, and their password. They were instructed to work through the first four chapters on their own and then contact Ericson to arrange for a two-hour observation using the Zoom videoconferencing software. Teachers were sent an email every week, which reminded them that they could quit the study at any time. They were also asked to contact Ericson to arrange an observation when they had completed chapter four.

Twenty-six teachers applied to be in the study. Some of these had more than three months experience in a textual programming language and were disqualified. Some teachers had years of experience teaching computing courses, but did not have at least three months of experience in a textual programming language. This

experience could have been in other courses like web design or in block-based languages. Eighteen teachers enrolled in the study, but five withdrew from the study because they did not think that they had time to finish before the deadline. Another two teachers did not complete the first four chapters in time. This paper is reporting on the 11 teachers (five in one group and six in the other) that were observed working through two versions of the fifth chapter. The two-hour observations were recorded with the teachers' consent. The videos from the recordings were transcribed.

Table 1: Teacher id, gender, race, age, certification/license, and years teaching computing/IT courses

Id	M/F	Race	Age	Certification/License	Years
T1	F	White	40	Business	10
T2	F	White	26	Biology	0
T3	F	White	30	English, Math and Science	1
T6	F	Hispanic	55	Math	1
T8	M	Arab	37	Biology and Chemistry	2
T9	M	White	33	History, English, and Tech.	2
T10	M	White	42	Instructional Tech.	15
T11	M	White	42	Language Arts and CS	0
T13	M	White	43	No Answer	1
T14	F	Asian	26	Math	1
T18	F	White	59	Math - Alt.	1

As you can see in Table 1 the majority of the teachers identified as white (63%), female (55%), and had been teaching computing courses for two years or less (82%). The teachers came from seven different states in the United States. Eight of the teachers (73%) had prior experience in Scratch [29]. The rest had experience in App Inventor [41], Java [3], Alice [14], JavaScript [22], Squeak [27], Snap [25], and Python [32]. Eight teachers (73%) had less than one year of experience programming in a drag and drop environment. Nine of the teachers (82%) had used variables and conditionals in a program. Seven of the teachers (64%) had used a loop in a program.

6.3 Use of Intra-Problem Adaptation

Of the 11 teachers, seven (63%) used the intra-problem adaptation by clicking the *Help Me* button at least once during the two-hour observation as shown in Table 2 (T1, T6, T9, T10, T11, T14, and T18). Three teachers (T1, T6, and T9), used the adaptation on four problems. Four teachers used the adaptation on problem 13 (T18, T1, T9, and T14). Table 2 shows the problem id with a short descriptor, the number of blocks followed by the number of distractors in parentheses, and then the teacher id followed by dash and then a code for the last type of adaptation that was used (D is distractor disabled, I is indentation provided, and C is combined blocks). The number after the code is the number of times that type of adaptation was used before the teacher correctly solved the problem. For example, T1 and T9 were both able to solve P1 (draw an L) after just one distractor was removed. T9 was able to solve P4 (draw an A) after one block was combined with another. Teacher T1 was able to solve P9 (draw a rectangle) after four distractors were disabled. T6 was able to solve P11 (draw a spirograph type pattern) after the indentation was provided. As you can see from this table, some teachers used the adaptation more than others and teachers were able to solve a problem after each type of adaptation.

Two teachers did not solve one of the non-adaptive problems. Teacher T9 gave up on problem 11 after 11 attempts saying, *"I don't know what I am doing on this one."* He had the block that starts with setting the color to blue in the outer loop and the block that starts with setting the color to red in the inner loop as shown on the right side in Figure 11 and they should be swapped. Teacher T6 gave up on P16 after 15 attempts to solve it and asked, *"Why isn't it letting me do the help?"* She was missing one of the for loops in a nested for loop problem and had added an extra left turn instead. See Problem 16 in Figure 12.

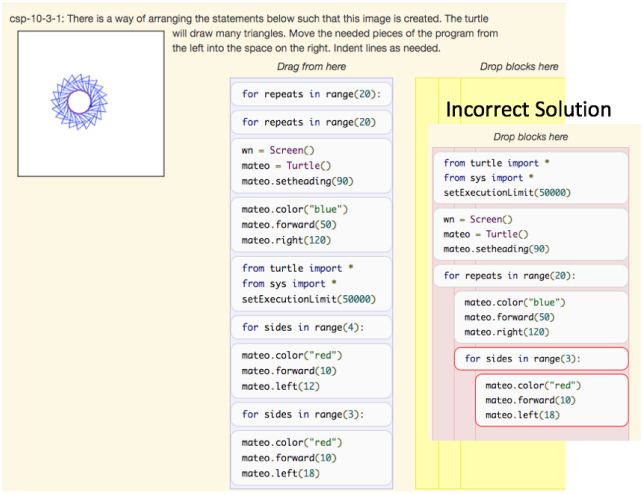

Figure 11: Problem 11 (P11) in the teacher observations.

Table 2: The problems that teachers successfully solved after using adaptation.

Id	Descriptor	# Blocks (# Dis.)	Teachers & Last Adaptation
P1	L	7(4)	T1-D1, T9-D1
P4	A	7(4)	T9-C1
P6	N	7(4)	T6-D1
P7	F	7(4)	T6-C1
P9	Rectangle	6(4)	T1-D4
P11	Spiro	6(3)	T6-I, T10-C3, T11-C3
P13	Line	7(4)	T18-D1, T1-I, T9-C1, T14-C1
P15	X	9(4)	T6-C2
P16	Squares	9(4)	T1-C3, T9-C4

6.4 An In-Depth Look at Problem 13

To examine understanding of the adaptation process, it is helpful to take an in-depth look at one problem. Four teachers used the adaptation on P13 shown in Figure 4, which was the largest number of teachers who used the adaptation on any problem. Problem 13 was also the only problem that was solved after each type of

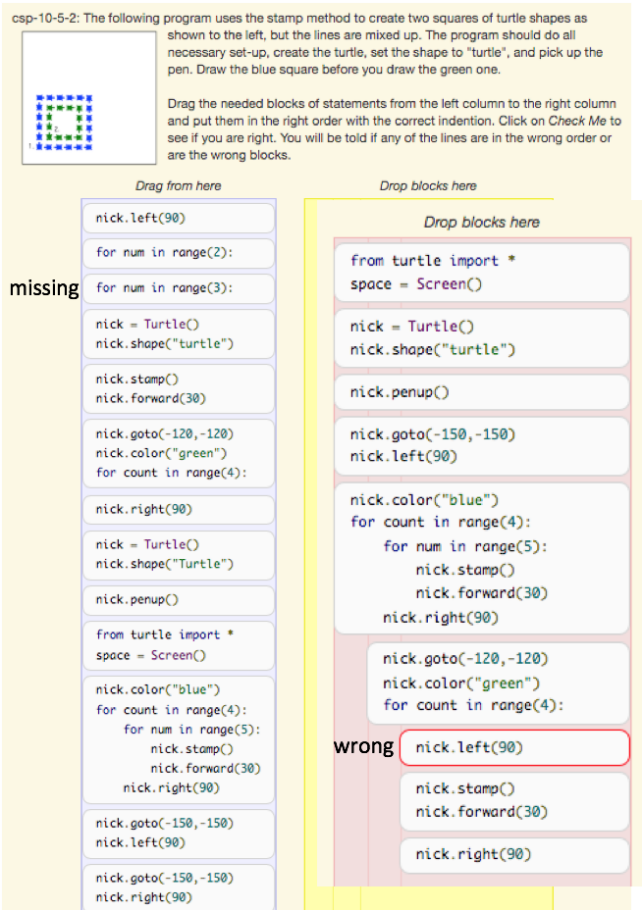

Figure 12: Problem 16 (P16) in the teacher observations.

adaptation. Problem 13 asked the learner to stamp three turtle shapes in a line. The distractor blocks had common syntax errors like the wrong case, missing parentheses, missing quotes for the string "turtle", and a missing colon at the end of the for statement.

6.4.1 Solving the problem after a distractor was disabled. Teacher T18 solved this problem after one distractor was removed. Her first solution was missing the block to set the turtle shape. She said, *"I don't have my other program to look at to see if I have to put a shape turtle in there. I am just going to run this and see what happens."* This is interesting because it indicates that she did not realize that the correct and incorrect blocks were shown paired with purple edges outlining the pair as shown in Figure 4, since that should have been a clue that she needed to use at least one of the two shape blocks. The solution check told her the solution was too short and that she needed to add more blocks. She said, *"I think I must need the shape in there somewhere."* She added the correct *shape* block.

When she checked her solution the software highlighted the distractor *stamp* block (the one that was missing the parentheses) as needing to be moved or replaced. She tried moving the *stamp* block before the forward block, but that was still wrong. The alert then said that help was available. She clicked on the *Help Me* button and an alert said that it would remove (disable) an incorrect block. She clicked on *Close* and the distractor stamp block moved slowly back to the source area on the left and was paired again with the correct *stamp* block and then grayed out. She said, *"Ah, okay I need the one with the thing [parentheses]. Oh gosh, I should have noticed that."* As mentioned earlier, experts pick up on details like missing parentheses that novices do not [13]. This example shows that this teacher understood the implicit hint provided by moving the distractor back to the source and pairing it with the correct code block before graying it out. In fact, all of the teachers in the observation study immediately moved the correct code block to the solution after the system disabled a distractor block that had been used in the solution.

6.4.2 Solving the problem after the indentation was provided. Teacher T1 solved this problem after all the distractors were disabled and the indentation was provided. She clicked on the *Help Me* button four times to disable the four distractors, but she had not used any of them in her solution. Then she clicked on the *Help Me* button again and the alert said that indentation would be provided. She clicked on the *Close* button and space was slowly added before the text in the blocks to provide the indentation as shown in Figure 7. Notice that the *penup* block should not be in the loop since it is not indented. This should have been an implicit hint to move that block outside the loop. However, this teacher did not understand this implicit hint.

After the indentation had been provided, she picked up the *stamp* block and tried to indent it, but could not. She then asked Ericson, *"So these last three [blocks] aren't supposed to be indented?"* Ericson answered, *"Not the penup."* She then said, *"But, these two are?"* and moved her cursor over the *stamp* and *forward* blocks. Ericson answered, *"So, what it did was provide the indentation."* She then moved the *penup* block before the loop and checked her solution. The last block was highlighted as needing to be moved. She then moved the *forward* before the *stamp* and that solution was correct. She had not realized that she should pick up the pen using the penup block before the turtle moved forward or it would draw a line when the turtle moved, even though the example before this problem showed the *penup* block before the loop.

Other teachers also put the *penup* block in the loop and were not able to solve the problem until after code blocks had been combined. This indicates that novice users may not be able to use the implicit hint given by providing the indentation.

6.4.3 Solving the problem after combining blocks. Teacher T9 originally dragged the *penUp* (wrong case) distractor into his solution, but then immediately moved it back to the source and dragged out the correct *penup* block instead. This distractor was shown paired with the correct code as shown in Figure 4 and this implies that the pairing helped him focus on what was different between the two blocks. He then checked a solution that did not include the block to set the turtle shape, and was told that the solution was too short. He, like teacher T18, did not seem to realize that because the distractor and correct blocks were shown in pairs, he should use at least one of the two blocks in the pair to set the shape.

After being told that the solution was too short, he dragged in the distractor block for setting the shape (the one that was missing the double quotes around the string). When he checked the solution, it highlighted the distractor and two blocks in the loop that had

to be moved for a correct solution. Like teacher T1, he also had included the *penup* block in the loop, even though the previous example had shown it before the loop.

He used the *Help Me* button to disable all of the distractor blocks and provide the indentation. After the indentation had been provided, he started to move the *penup* block, but stopped and clicked the *Help Me* button again, which moved the *penup* block below the *shape* block and then combined them as shown in Figure 8. The problem was now correct. This example shows that this teacher found the combining of two blocks into one block useful for helping him solve the problem.

6.5 Preference and Perception of Usefulness

If the teacher never used the adaptation during the observation, Ericson demonstrated it after the teacher had finished working through the fifth chapter. The teacher was asked 1) What type of Parsons problem did the teacher prefer (adaptive or non-adaptive) and 2) Did the teacher feel that solving Parsons problems helped them learn to fix or write similar code?

Nine (81.8%) of the eleven teachers said that they preferred adaptive to non-adaptive Parsons problems. One of the teachers still wanted help, but she suggested providing an example or providing audio feedback instead. She was worried that the intra-problem adaptation could become a "crutch" and that some students might abuse it. Another teacher said that the adaptation messed him up. He was confused when the distractors that he had not used in his solution were disabled. His attention was on his solution on the right, not the source area on the left. After this study, the adaptation process was changed to only disable distractors that had been used in the solution.

All of the teachers said that they felt that solving Parsons problems with distractors helped them when they had to fix code with the same type of errors as the distractors. One teacher said, *"Yes, it kept reminding me. Hey, that requires a colon, that requires parentheses, that requires lower case or upper case. That is what I was struggling with in the beginning."* All of the teachers said that they felt that solving Parsons problems with distractors also helped them when they had to write code from scratch. One teacher said, *"I had made a mistake with one [problem] and then went back to switch it [the distractor block for the correct block]. I realized that okay, the only difference there is that it is capitalized, or I left a parenthesis off or something."*

During the observations several teachers used a distractor that was missing the colon at the end of a *for* loop. One teacher even asked why there were two of the exact same block in the source area. That teacher didn't notice that one block was missing the colon. The first fix code problem also was missing a colon at the end of a *for* loop and all of the teachers quickly found that error. This provides evidence that solving Parsons problems with distractors with common syntax errors can help novice programmers learn to spot and fix common syntax errors.

7 TESTING ADAPTATION AT SCALE

While observations can provide an in-depth look at learners' understanding of the adaptation process, they do not provide sufficient evidence of the effect of intra-problem and inter-problem adaptation on learners' ability to successfully solve Parsons problems. Quantitative data derived from logs of user interactions can provide this evidence.

The log file was an anonymous log file with all the user and institution information replaced with numbers. The log file contained data from two free ebooks from 8/13/2017 to 10/31/2018. Users of these ebooks are notified that their usage data will be used for educational research. One ebook was for AP CSP and the other for AP CSA. The AP CSP course is intended to be equivalent to a first computer science course for non-majors. It covers programming basics in Python including variables, conditionals, loops, and functions. The AP CSA course is intended to be equivalent to a first course for computer science majors at the college level. It covers object-oriented programming in Java and includes programming fundamentals, arrays (1D and 2D), lists, recursion, searching, sorting, inheritance, interfaces, and polymorphism.

The Parsons problems used in the observation study were added to the public version of the Advanced Placement (AP) CS Principles (CSP) ebook. It is interesting to examine the log file data for just the Parsons problems that were used in the observational study. These Parsons problems were in pairs of problems of similar difficulty where one of the pair was adaptive and the other was not. Table 3 shows the problem id, a short descriptor, whether the problem was adaptive, the number of users who attempted the problem, and the percentage of users who got the problem correct.

Table 3: Problem Id, descriptor, whether the problem was adaptive, the number of users who attempted the problem, the percentage of users who got the problem correct

Id	Description	Adaptive?	Num. attempted	% correct
P1	L	N	839	83.75%
P2	Check	Y	708	93.27%
P3	T	Y	668	92.76%
P4	A	N	592	79.11%
P5	Z	N	548	93.17%
P6	N	Y	501	94.46%
P7	F	Y	488	93.05%
P8	A2	N	425	93.51%
P9	Rectangle	N	297	82.86%
P10	Triangle	Y	238	96.60%
P11	Spiro	Y	247	93.72%
P12	Spiro2	N	196	91.75%
P13	Line	N	212	89.22%
P14	Circle	Y	191	91.49%
P15	X	Y	194	90.96%
P16	Squares	N	165	77.22%

The average percentage of users who got the non-adaptive Parsons problems correct was 86.32% with a standard deviation of 6.44. The average for the adaptive Parsons problems was 93.29% with a standard deviation of 1.75. This provides evidence that a higher average percentage of users correctly solved adaptive Parsons problems than non-adaptive Parsons problems.

Table 4 shows the percentage of users who used adaptation on these problems varied from 9.7% for problem P2 to 44.7% for problem P15. It also shows that the percentage who used adaptation, but never got the problem correct ranged from 10% to 16.4%. This indicates that the adaptation process could be improved.

Table 4: Problem Id, descriptor, the number of users who used adaptation (the percentage of users), number of users who used adaptation, but never solved the problem (the percentage of users who used the adaptation, but did not solve the problem)

Id	Descriptor	# Used Adaptation (%)	Never Solved (%)
P2	Check	66 (9.7%)	10 (15.2%)
P3	T	149 (23.0%)	16 (10.7%)
P6	N	61 (12.5%)	10 (16.4%)
P7	F	84 (17.7%)	12 (14.3%)
P10	Triangle	26 (11.1%)	4 (15.4%)
P11	Spiro	79 (33.1%)	8 (10.1%)
P14	Circle	70 (37.2%)	7 (10.0%)
P15	X	84 (44.7%)	11 (13.1%)

7.1 Adaptation Increases the Odds that Users will Correctly Solve the Problem

To test if ebook users were more likely to correctly solve adaptive Parsons problems than non-adaptive Parsons problems, an analysis was performed with all 154 Parsons problems in both the AP CSP ebook (97 adaptive problems, 9 non-adaptive problems) and the AP CS A ebook (14 adaptive problems, 34 non-adaptive problems). This analysis included 8,675 unique users who attempted to solve one or more Parsons problems.

Other factors besides the adaptation could contribute to students' ability to get these problems correct. The analysis controlled for the effect of the problem, the user, and the ebook they were using. The problem and the user were modeled as random effects, while the book was modeled as a fixed effect.

In order to accommodate both fixed and random effects, as well as categorical data, a generalized linear mixed model was used. The dependent variable was whether the user got the problem correct or not, and the predictors were whether the problem was adaptive or non-adaptive, the problem id, the user id, and the ebook being used. The glmer function in R was used to fit the model using maximum likelihood (Laplace Approximation).

The coefficient for adaptiveness was 0.69, which was significantly different from 0 ($p = 0.005$). Since adaptiveness is a binary variable (the problem was either adaptive or non-adaptive), 0.69 is the log odds ratio of correctness for adaptive problems compared to non-adaptive problems. Transforming this coefficient from the log odds ratio to the odds ratio gives the result that ebook users were 1.99 times as likely to get a problem right when the Parsons problem was adaptive than when it was not.

8 DISCUSSION

RQ1: Did the learners understand the intra-problem adaptation process? The observational study provided evidence that the teachers understood part of the adaptation process, but not all of it. Teachers understood when a distractor that they had used in their solution was disabled (animated moving back to the source area on the left and then grayed out), but not when a distractor that they had not used was disabled (just grayed out). After this study the adaptation process was modified to no longer disable distractors that were not used in the solution. Most teachers did not seem to understand the implicit hint given by providing the indentation. Perhaps this type of adaptation could be removed. Teachers did understand when two blocks were combined into one (one block was moved slowly under the other and then the two blocks were redrawn as one block).

RQ2: What type of Parsons problem did learners prefer? Nine (82%) of the 11 teachers preferred adaptive Parsons problems to non-adaptive Parsons problems. One teacher still wanted help, but wanted a different type of help. The other teacher did not like the adaptation because he was confused by the disabling of distractors that he had not used in his solution.

RQ3: Did the learners perceive that solving Parsons problems helped them learn to fix and write similar code? All of the teachers reported that solving Parsons problems helped them when they had to fix code with similar errors to the distractors or write similar code. They felt that the distractors helped them learn to recognize common syntax errors. They also said that the Parsons problems helped them learn common sequences.

RQ4: What was the effect of the type of Parsons problem on the learners ability to correctly complete the problem? The log file analysis of over 8,000 users shows that users are almost twice as likely to get an adaptive Parsons problem correct than a non-adaptive one.

9 THREATS TO VALIDITY

The observed participants' actions and perceptions may have been influenced by knowing that they were being observed. Other studies should be done to test the reported perceptions. This study only reports on observations of secondary teachers from the United States. The results may differ for teachers from other countries. The log file analysis does not break out the effect of intra-problem adaptation versus inter-problem adaptation. More studies are needed to determine the effect of each type of adaptation on learners' ability to correctly complete problems. Learners may have used the adaptation to correctly complete a problem, but may not have learned much in the process. However, in another study, learners had equivalent learning gains from pretest to posttest after solving adaptive Parsons problems versus non-adaptive Parsons problems versus writing the equivalent code [19]. The learners who solved adaptive Parsons problems had significantly higher learning gains than learners in an off-task control group [19].

10 CONCLUSION

This paper is the first to report on an observational study of intra-problem adaptive Parsons problems in which the difficulty of the problem was dynamically reduced by disabling distractors, providing indentation, and combining blocks. It provides evidence that the secondary teachers understood what was happening when distractors were disabled that they had used in a solution, but not when

distractors were disabled that they had not used. In addition, not all teachers were able to use the implicit hints given by providing the indentation. These findings resulted in changes to the adaptation process. Distractors that have not been used in the solution are no longer disabled.

Teachers perceived that solving Parsons problems with distractors helped them learn to recognize and fix common syntax errors. All of the teachers reported that solving Parsons problems with distractors helped them learn to fix and write code. Perception is important since a positive perception is likely to increase motivation. However, more research should be done to test these perceptions.

Finally, a log file analysis of data from over 8,000 users showed that learners are nearly twice as likely to correctly complete an adaptive Parsons problem than a non-adaptive one. This indicate that adaptation is successfully helping learners correctly solve Parsons problems.

These results suggest that using adaptive Parsons problems could reduce the frustration that many novices feel when learning programming, and perhaps improve retention rates in college-level introductory programming courses. We encourage researchers to test these hypotheses.

11 ACKNOWLEDGMENTS

Thanks to Jochen (Jeff) Rick for his work on the Parsons problem adaptation process. Thank you to the undergraduate students and secondary teachers who volunteered for observations. Thank you to the reviewers for their suggestions. This study was supported by the National Science Foundation under grants 1138378 and 1432300.

REFERENCES

[1] John R Anderson, C Franklin Boyle, and Brian J Reiser. 1985. Intelligent tutoring systems. *Science* 228, 4698 (1985), 456–462.

[2] John R Anderson, Albert T Corbett, Kenneth R Koedinger, and Ray Pelletier. 1995. Cognitive tutors: Lessons learned. *The journal of the learning sciences* 4, 2 (1995), 167–207.

[3] Ken Arnold, James Gosling, and David Holmes. 2005. *The Java programming language.* Addison Wesley Professional.

[4] Ronald Baecker and Ian Small. 1990. Animation at the interface. *The art of human-computer interface design* (1990), 251–267.

[5] Albert Bandura, WH Freeman, and Richard Lightsey. 1999. Self-efficacy: The exercise of control.

[6] Klara Benda, Amy Bruckman, and Mark Guzdial. 2012. When life and learning do not fit: Challenges of workload and communication in introductory computer science online. *ACM Transactions on Computing Education (TOCE)* 12, 4 (2012), 15.

[7] Jens Bennedsen and Michael E Caspersen. 2007. Failure rates in introductory programming. *ACM SIGcSE Bulletin* 39, 2 (2007), 32–36.

[8] Laura E. Berk and Adam Winsler. 1995. *Scaffolding Children's Learning: Vygotsky and Early Childhood Education.* National Association for the Education of Young Children.

[9] John D Bransford, Ann L Brown, Rodney R Cocking, et al. 2000. *How people learn.* Vol. 11. Washington, DC: National academy press.

[10] Fanny Chevalier, Nathalie Henry Riche, Catherine Plaisant, Amira Chalbi, and Christophe Hurter. 2016. Animations 25 Years Later: New Roles and Opportunities. In *Proceedings of the International Working Conference on Advanced Visual Interfaces.* ACM, 2909255, 280–287. https://doi.org/10.1145/2909132.2909255

[11] Gemma Corbalan, Liesbeth Kester, and Jeroen JG Van Merriënboer. 2008. Selecting learning tasks: Effects of adaptation and shared control on learning efficiency and task involvement. *Contemporary Educational Psychology* 33, 4 (2008), 733–756.

[12] Albert T Corbett, Kenneth R Koedinger, and John R Anderson. 1997. Intelligent tutoring systems. In *Handbook of human-computer interaction.* Elsevier, 849–874.

[13] National Research Council et al. 2000. *How people learn: Brain, mind, experience, and school: Expanded edition.* National Academies Press.

[14] Wanda P Dann, Stephen Cooper, and Randy Pausch. 2008. *Learning to program with Alice.* Prentice Hall Press.

[15] Paul Denny, Andrew Luxton-Reilly, and Beth Simon. 2008. Evaluating a new exam question: Parsons problems. In *Proceedings of the fourth international workshop on computing education research.* ACM, 113–124.

[16] Carol S Dweck. 1986. Motivational processes affecting learning. *American psychologist* 41, 10 (1986), 1040.

[17] Barbara Ericson, Mark Guzdial, Briana Morrison, Miranda Parker, Matthew Moldavan, and Lekha Surasani. 2015. An eBook for teachers learning CS principles. *ACM Inroads* 6, 4 (2015), 84–86.

[18] Barbara Jane Ericson. 2018. *Evaluating the Effectiveness and Efficiency of Parsons Problems and Dynamically Adaptive Parsons Problems as a Type of Low Cognitive Load Practice Problem.* Ph.D. Dissertation. Georgia Institute of Technology.

[19] Barbara J. Ericson, James D. Foley, and Jochen Rick. 2018. Evaluating the Efficiency and Effectiveness of Adaptive Parsons Problems. In *Proceedings of the 2018 ACM Conference on International Computing Education Research (ICER '18).* ACM, New York, NY, USA, 60–68. https://doi.org/10.1145/3230977.3231000

[20] Barbara J Ericson, Mark J Guzdial, and Briana B Morrison. 2015. Analysis of interactive features designed to enhance learning in an ebook. In *Proceedings of the eleventh annual International Conference on International Computing Education Research.* ACM, 169–178.

[21] Barbara J Ericson, Lauren E Margulieux, and Jochen Rick. 2017. Solving parsons problems versus fixing and writing code. In *Proceedings of the 17th Koli Calling Conference on Computing Education Research.* ACM, 20–29.

[22] David Flanagan. 2006. *JavaScript: the definitive guide.* " O'Reilly Media, Inc.".

[23] Stuart Garner. 2007. An Exploration of How a Technology-Facilitated Part-Complete Solution Method Supports the Learning of Computer Programming. *Issues in Informing Science & Information Technology* 4 (2007).

[24] Kyle James Harms, Jason Chen, and Caitlin L Kelleher. 2016. Distractors in Parsons Problems Decrease Learning Efficiency for Young Novice Programmers. In *Proceedings of the 2016 ACM Conference on International Computing Education Research.* ACM, 241–250.

[25] Brian Harvey, Daniel Garcia, Josh Paley, and Luke Segars. 2012. Snap!:(build your own blocks). In *Proceedings of the 43rd ACM technical symposium on Computer Science Education.* ACM, 662–662.

[26] Petri Ihantola and Ville Karavirta. 2011. Two-dimensional Parson's Puzzles: The Concept, Tools, and First Observations. *Journal of Information Technology Education* 10 (2011), 119–132.

[27] Dan Ingalls, Ted Kaehler, John Maloney, Scott Wallace, and Alan Kay. 1997. Back to the future: the story of Squeak, a practical Smalltalk written in itself. In *ACM SIGPLAN Notices*, Vol. 32. ACM, 318–326.

[28] Paivi Kinnunen and Beth Simon. 2010. Experiencing programming assignments in CS1: the emotional toll. In *Proceedings of the Sixth international workshop on Computing education research.* ACM, 77–86.

[29] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. 2010. The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)* 10, 4 (2010), 16.

[30] Jane Margolis and Allan Fisher. 2003. *Unlocking the clubhouse: Women in computing.* MIT press.

[31] Dale Parsons and Patricia Haden. 2006. Parson's programming puzzles: a fun and effective learning tool for first programming courses. In *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52.* Australian Computer Society, Inc., 157–163.

[32] Michel F Sanner et al. 1999. Python: a programming language for software integration and development. *J Mol Graph Model* 17, 1 (1999), 57–61.

[33] John A Sloboda, Jane W Davidson, Michael JA Howe, and Derek G Moore. 1996. The role of practice in the development of performing musicians. *British journal of psychology* 87, 2 (1996), 287–309.

[34] John Sweller. 1988. Cognitive load during problem solving: Effects on learning. *Cognitive science* 12, 2 (1988), 257–285.

[35] John Sweller. 1994. Cognitive load theory, learning difficulty, and instructional design. *Learning and instruction* 4, 4 (1994), 295–312.

[36] Michael Tuffiash, Roy W Roring, and K Anders Ericsson. 2007. Expert performance in SCRABBLE: Implications for the study of the structure and acquisition of complex skills. *Journal of Experimental Psychology: Applied* 13, 3 (2007), 124.

[37] Jeroen JG Van Merriënboer. 1990. Strategies for programming instruction in high school: Program completion vs. program generation. *Journal of educational computing research* 6, 3 (1990), 265–285.

[38] Kurt Vanlehn. 2006. The behavior of tutoring systems. *International journal of artificial intelligence in education* 16, 3 (2006), 227–265.

[39] Lev Semenovich Vygotsky. 1980. *Mind in society: The development of higher psychological processes.* Harvard university press.

[40] Christopher Watson and Frederick WB Li. 2014. Failure rates in introductory programming revisited. In *Proceedings of the 2014 conference on Innovation & technology in computer science education.* ACM, 39–44.

[41] David Wolber, Hal Abelson, Ellen Spertus, and Liz Looney. 2011. *App Inventor.* " O'Reilly Media, Inc.".