2021 ASEE ANNUAL CONFERENCE
Virtual Meeting | July 26–29, 2021 | Pacific Daylight Time
Paper ID #34831

Collaborative Parsons Problems in a Remote-learning First-year Engineering Classroom

Brooke Morin, Ohio State University

Brooke Morin is a Senior Lecturer in the College of Engineering at Ohio State University, teaching First-Year Engineering for Honors classes in the Department of Engineering Education. Brooke earned her bachelor's degree and master's degree in Mechanical Engineering at Ohio State. Her research interests include implementing and evaluating evidence-based practices in the first-year engineering classroom.

Dr. Krista M. Kecskemety, Ohio State University

Krista Kecskemety is an Assistant Professor of Practice in the Department of Engineering Education at The Ohio State University. Krista received her B.S. in Aerospace Engineering at The Ohio State University in 2006 and received her M.S. from Ohio State in 2007. In 2012, Krista completed her Ph.D. in Aerospace Engineering at Ohio State. Her engineering education research interests include investigating first-year engineering student experiences, faculty experiences, and the connection between the two.

Collaborative Parsons Problems in a Remote-Learning First-Year Engineering Classroom

Introduction

This complete evidence-based practice paper examines the impact of a classroom activity to teach programming to first-year engineering students. Programming and logic are vital components of an engineering education. While some may assume programming is only important in computer science, many engineering disciplines use computer software, which requires programming and logic. This is why it is common to teach introductory programming and computation to all engineering disciplines [1]. However, students often struggle to learn programming and come into the first year of engineering with a wide range of prior programming experience[2], [3]. The differences can be due to access of computer science courses at the high school level [4], the level of instruction in these courses, and the self-selection bias in enrolling in these courses. It has been shown that women and under-represented minorities are less likely to have taken a high school computer science course [4]. Therefore, when these students with a wide range of prior experience start a first-year engineering course, many of them struggle.

One of the explanations for why students struggle in introductory programming classes is the high cognitive load associated with traditional programming tasks. Cognitive load theory uses an evolutionary framework to describe how novel information is learned and stored in long term memory [5]. A given learning task has both *intrinsic loads*, or the inherent interactivity of the elements of the information being learned, and *extraneous loads*, which are introduced primarily by the choice of instructional technique. These loads (with extraneous load often differentiated from *germane load*, which is also due to instructional technique but is beneficial to learning) act upon a learner's working memory [6], [7]. When the working memory is overloaded, the ability of a learner to process and retain information in long term memory is hindered.

Traditional programming instruction tends to create high extraneous cognitive loads [8], [9]. It often requires students to learn programming syntax and logic simultaneously, which is particularly challenging for students with a less robust computer science and programming background, as they do not have existing schema for either of these interacting elements. However, when students who do have existing schema in long term memory for one or both of these elements will experience lower cognitive loads in their working memory [5]. Thus, separating these two components should reduce the cognitive load of the students, allowing them to focus on developing a single skill at a time in order to build the necessary schemas to effectively complete traditional programming problems.

One way to implement this separation of syntax and logic is through the use of Parsons Problems. Parsons Problems are activities or assignments in which computer code is separated into individual lines or multi-line segments [10]. These segments are then scrambled and are to be placed in order by the students. These problems have been shown to have similar efficacy in improving programming skills as writing the programs themselves [11], [12].

Parsons Problems have been implemented into a first-year engineering course that teaches introductory computer programming through MATLAB and C/++. The problems, first implemented in Autumn 2019, were designed to be completed in person in the classroom. The students were provided slips of paper and asked to rearrange them into the correct order. Previous student feedback provided some areas of improvement for the next iteration including looking at the length and difficulty of the problems [13]. This process is described in the following section, *Background*.

While these challenges were being addressed, an additional challenge presented itself: the virtual class environment. All activities were forced to shift to a virtual environment due to the class becoming 100% online in AU 2020. Activities were converted and delivered via custom adaption of js-parsons, an online Parsons Problem JavaScript library [14]. This still allowed the collaboration that made the previous iteration unique through the use of Zoom breakout rooms and screen sharing. This paper discusses the development of the online library as well as discussing student feedback for this online version of the activity and compare that with the feedback that was obtained for the in-person activity in previous years. Additionally, we will highlight the plans for further research into the learning gains and the impact of activity formatting and mode of adoption has on the success of Parsons Problem and a learning tool.

Background

The first-year engineering program at The Ohio State University first used Parsons Problems in the first-semester problem solving and programming course in 2019. The decision to incorporate these activities was based on existing research that demonstrated the benefit in introductory programming [15]. The honors program was the first to implement these problems in all of their sections of the course in Autumn 2019, focusing on weekly in-class group activities as the mechanism for including them in the course. The delivery of the problems included physical slips of paper, shown in Figure 1, that students manipulated to create the finished program solution based on a provided prompt. Students worked in groups of 2 to 4 and teaching assistants were available to help guide the activity. The difficulty of the problems increased as the semester progressed. The initial problems were simple programs while later problems were more complex. This increase in complexity occurred for several reasons. First, the logic of more advanced programming concepts was inherently more challenging to understand. Second, as the programming concepts became more complex, the length of the problem also increased to provide the necessary context. Finally, distractors, logically or syntactically incorrect code segments aimed at correcting common errors, were also incorporated as the semester progressed.

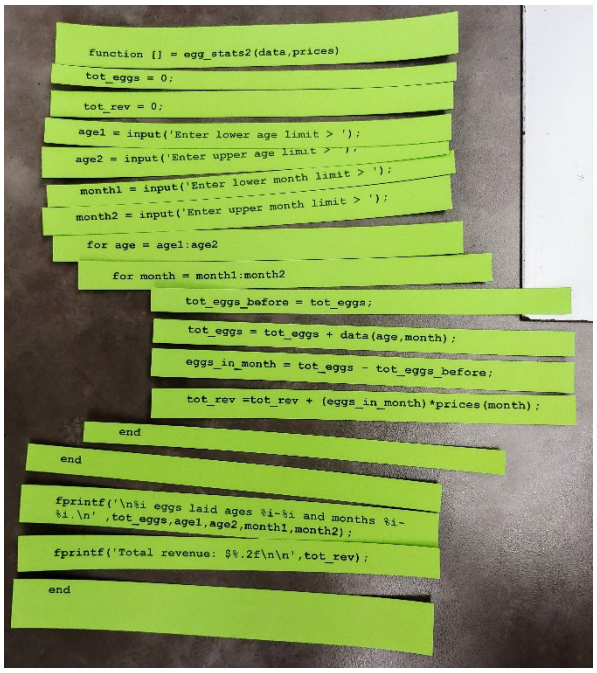

Figure 1: Paper example of the Parsons Problem used during in-person instruction. This problem tested nested loops logic in MATLAB.

Based on the pilot delivery, questions began to emerge related to what features were most beneficial to Parsons Problems. Faculty anecdotally noted that students, while generally engaged in the activity, disengaged when the problems were too complex. This is consistent with the cognitive load theory in that their working memory was likely becoming overloaded. However, which specific elements of that complexity were driving the cognitive load increases, as well as a clear idea of the threshold under which the complexity and interactivity were manageable, was not identifiable.

To gather student perceptions, an open-ended question was presented to all students to provide feedback on the Parsons Problems. In Autumn 2019, students indicated positive responses about the activities in 60% of the responses. At least one student even noted that "[Parsons Problems] allowed me to focus on the logic and theory behind different concepts without worrying about syntax". And while 40% had either a mixed or negative response to the problems, this is not dissimilar to what is seen with other educators who have implemented Parsons Problems [11]. Furthermore, many of the negative comments related directly to specific attributes of the Parsons Problems: length and difficulty. For quantitative feedback, students were asked a Likert scale question about the helpfulness of the distractors. Half of students indicated that the distractors were very helpful or helpful, 22% were neutral and 28% indicated they were unhelpful or very unhelpful.

From this data, the instructional staff intended to improve their delivery of the Parsons Problems, including reducing the length and complexity of several of the more difficult programs. With the Covid-19 pandemic forcing courses to transition to online delivery, it became clear that an alternate method of presenting these problems was also necessary. The following section details

the process by which the Parsons Problems were adapted for online delivery, followed by student responses to this change.

Methods

Online Parsons Problem Development

In Autumn 2020, the virtual nature of the courses necessitated a change to the delivery of the problems. Prior to identifying the appropriate platform for the new problems, priorities for the new format were identified.

- *Simple environment:* The new format had to be simple. The previous implementation used slips of paper and a page of instructions to implement the activities. The new environment needed to be similarly streamlined and intuitive.
- *Easy manipulation of the code segments:* The new format had to allow the students to move code segments easily. The segments needed to be easy to align, easy to reorder, and easy to indent.
- *Collaborative:* One of the features that made the paper strips so effective was that students could collaborate when solving the puzzles by handing pieces of paper back and forth and discussing where the strips belonged. The new solution had to replicate this process as closely as possible.
- *Easy to disseminate:* The instructional team responsible for the course were already transitioning courses and laboratories online, along with dealing with the inherent work overhead of an online course. The solution could not require an instructor to create duplicate documents, copy and paste code for multiple teams, or other work-intensive operations.
- *Self-checking:* In the in-person implementation, the instructional team (one faculty member and multiple undergraduate teaching assistants) would rotate around the classroom, helping students identify logic errors. In the online classroom, it was much more difficult to quickly move between groups and assess the students' solutions. In order to provide students quick and effective feedback, the tool had to have self-evaluation features.

Based on these priorities, an in-house adaptation of js-parsons, an online Parsons Problem JavaScript library [14], was identified as the platform. The problems were hosted on a university server. A backend was developed using JavaScript with NodeJS and Express that was able to evaluate the unscrambled MATLAB and C/C++ code segments and return the output to the screen. Students could then evaluate the output, as they would evaluate the output of a programming assignment, to see if the code was working properly. If their submission would return an error, the error text was returned.

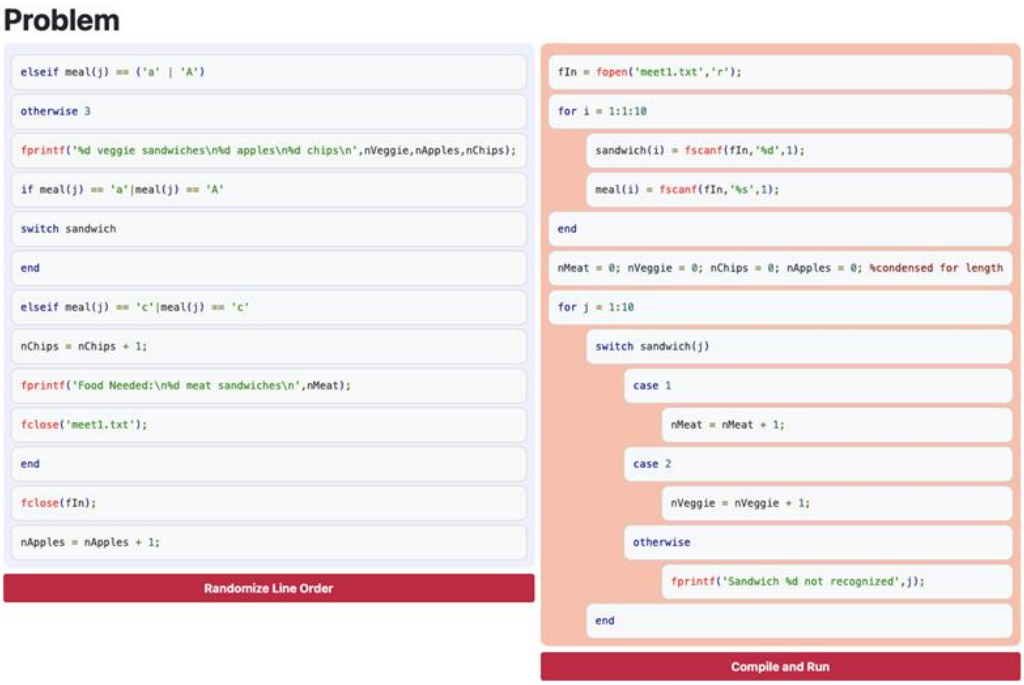

Figure 2: Example of a Parsons Problem used in the online environment. Students could drag code segments between regions, indent, and reorder. A box elsewhere on the screen showed output.

This online implementation, shown in Figure 2, above, used the same types of problems that had been delivered in person but now for a virtual environment. Students were sent to Zoom breakout rooms to work on the problems, where they were instructed to have one student screen share the problem website. The remainder of the students were shown how to use the annotate feature to indicate where the code segments belonged or to request control of the shared screen in order to move the segments themselves. Instructions and a general description (Figure 3) were provided at the top of the webpage.

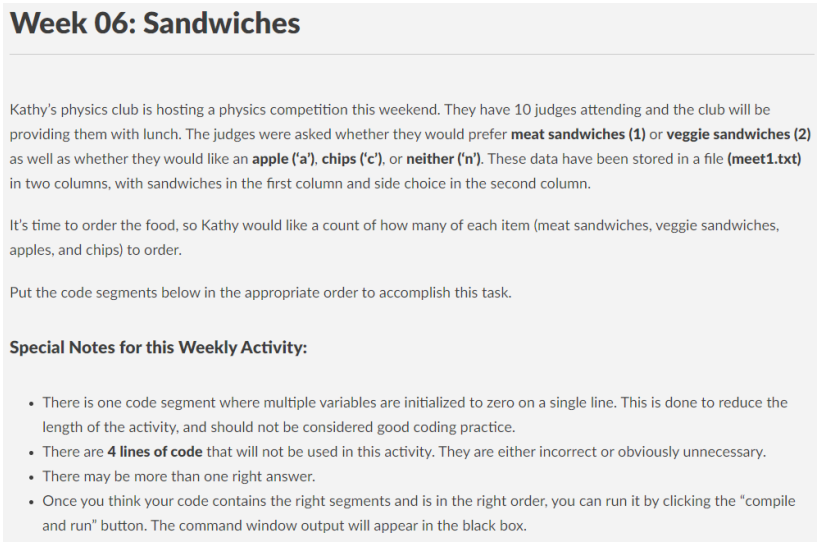

Figure 3: The students were provided with problem context and instructions for completing the problem.

The problems were similar to the previous year, but changes were made to adjust to the online platform and respond to students' previous criticisms. For instance, due to limited development time, the online platform could not evaluate software that required keyboard input that generated output files. Therefore, these structures were limited to only the most relevant week, where the answers had to be manually evaluated, and otherwise replaced with input files and writing to the screen. Additionally, the practical size of a computer screen provided natural limits to how many lines the final code could reasonably be. This limited the length, and often the complexity, of the programs. These changes likely resulted in some of the problems being less difficult than the previous year's counterpart.

Survey Instrument

Students were given the following survey questions at the end of the course asking about the Parsons Problem implementation. In this survey the Parsons Problems are referred to as "Weekly Activities", which was the name used for them in the course.

- Did you feel that the Weekly Activities helped you learn how to code? If not, why not? If so, how? [Open ended text response]
- Briefly describe ways in which the Weekly Activities were approached by different breakout rooms you were in this semester. Were some approaches more effective than others? If so, please describe. [Open ended text response]
- Did you ever access the Weekly Activities outside of class time? If so, how many? [Response Options: No; Yes, just one or two; Yes, some of them; and Yes, most or all of them]
- Some Weekly Activities included "distractors", or code segments that were incorrect or unnecessary. Rank the usefulness of these distractors, from very unhelpful (1) to very helpful (5). - Distractor usefulness

Of the 309 students enrolled in the course, 282 responses were recorded, although some participants chose not to answer all the questions or provide written responses. This resulted in different survey questions having different response counts, which will be indicated when the data are presented. The open-ended response to the first question was coded to indicate a positive, negative, or mixed response.

Results and Discussion

The survey question asking if the weekly activities were helpful in learning to code was coded to indicate the positivity of the response. Approximately 57% of the students who responded (N=270) had a positive response about the activities, with the rest having either a mixed or negative response. This was similar to the 60% positivity (N=289) found in the previous iteration in Autumn 2019 with the paper-based activities, though the number of confounding factors that arose when transitioning to online delivery prevented any concrete statistical claims. In both cases, a common concern cited by the students was whether completing the activities would help with writing complete programs. As previously discussed, this has been shown to be true in the literature, but even when these demonstrative studies are cited in the classroom students struggle to believe that Parsons Problems have an impact on their learning.

Students also provided feedback as to whether they found distractors helpful. As seen in Figure 4, 60% of students in Autumn 2020 indicated that distractors were very helpful or helpful, 27% were neutral and 13% indicated they were unhelpful or very unhelpful. Compared to the previous year, Autumn 2019, they were slightly more positive and less negative about the distractors. It is not clear what caused the positive shift from the student perspective as the distractors were similar in the Autumn 2020 semester. Some of the problems ended up being easier than the year before, which could have contributed to the distractors being slightly more helpful. This would be consistent with the cognitive load framework, as the students' existing schemas being more sufficient to handle the primary sorting task would allow for the addition of distractors without overwhelming their working memory.

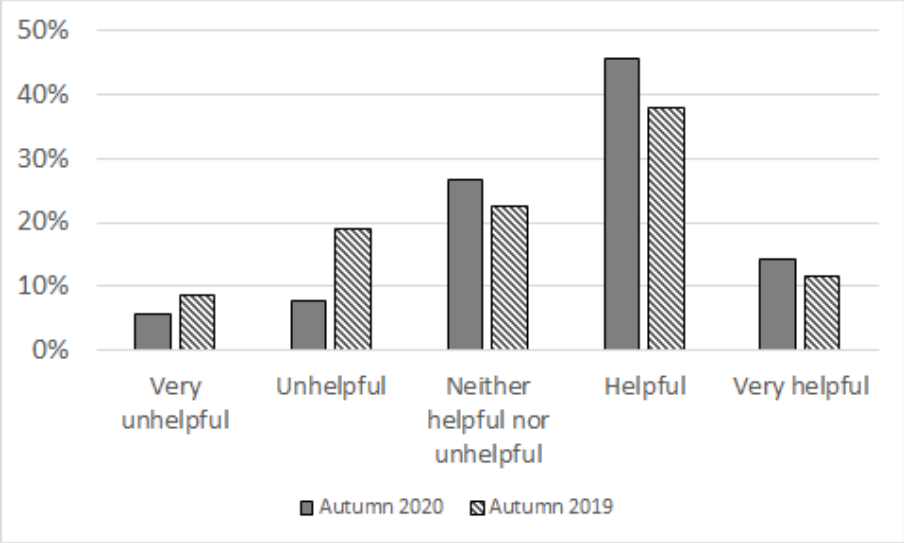

Figure 4: Student responses to helpfulness of distractors in Parsons Problems activities (N=281 for Autumn 2020, N=387 for Autumn 2019)

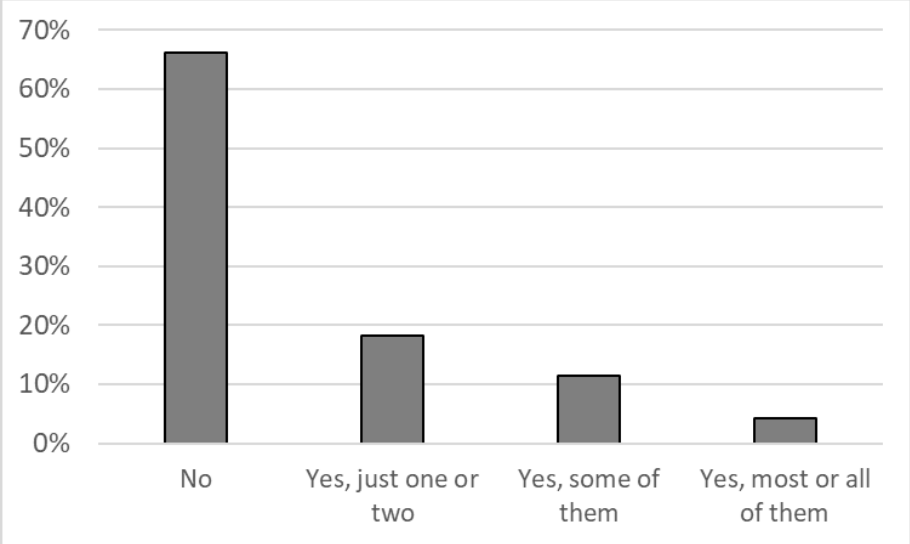

Figure 5: Student reported use of Parsons Problems outside of class (N=281)

In addition to completing the Parsons Problems during class time, students were encouraged to practice them out of class. While these problems were intended to be used to help students study and practice outside of class as well, we can see by Figure 5 that students did not often use these resources outside of class time. Future work will examine whether these responses compare with course and exam grades to see if there is any correlation between use or helpfulness and course grades, as well as whether the lack of adoption of these resources was due to poor advertising, lack of interest, or another factor.

The other open-ended question asked students about how they used the breakout rooms and worked with groups to accomplish the Parsons Problems. Through this response we see many responses indicating that student groups used the breakout rooms in different ways. Some students indicated that the group members would work independently and then share their answers with each other, while others indicated that they would work through it together with one student sharing their screen. Many indicated that when they worked together the activity was more useful. This was partially a natural outcome of the varied approaches students took to the online classroom. In a physical space, the instructional team could identify students who were not collaborating and encourage them to work together. Additionally, some students were highly interactive in the online breakout rooms while others were either more hesitant to or unable to participate verbally or with video. This inconsistency of approach also highlights the need for clearer instructions in the future if these problems are used in a virtual format. Since the problems were intended to be completed collaboratively rather than independently, the instructions should be clearer to the students about how to approach the teamwork.

Conclusion and Future Work

From this initial online delivery of collaborative Parsons Problems, we drew several conclusions: (1) Students had varied experiences with the Parsons Problems. Over the two years, the problems varied significantly as we tried to accommodate varying programming concepts, adjust to external constraints, and respond to student and instructor feedback. One consistent observation reported by the instructors was the idea that the students would “shut down”, or disengage, when the problem became too overwhelming. This behavior would be consistent with working memory being overloaded. (2) Though varied, the experiences skewed positive. This feedback suggested that, especially if we could reduce cognitive overload, students and faculty would be receptive to continued use of these activities in the classroom. Combined with the documented success of Parsons Problems in programming instruction, this anticipated adoption supports our decision to continue to investigate best practices for the problems. (3) In particular, the results concerning distractors support the proposed influence of cognitive load on Parsons Problem engagement. When the problems became easier during the second year, our largely novice student population seemed better equipped to handle the additional interacting elements that the use of distractors presents.

These results all support the desire to further investigate and optimize Parsons Problems, whether presented online or in person, through a cognitive load framework. Introductory programming instruction is an area where cognitive load is of particular concern [8], [9]. The intrinsic load of a programming course is inherently high, as programming syntax and logic exhibit high

interactivity, with syntax alone consisting of many interactive elements [16]. Extrinsic loads may be high as well, as varied programming backgrounds render information redundant for some learners and novel for others [5]. Previous authors have presented strategies for reducing cognitive load in programming courses [8], [16], [17] but these strategies often involve changing programming environment, using different languages, or other large-scale changes that may be burdensome on instructors with limited resources. Parsons problems are proposed to reduce cognitive load in several ways: (1) by already containing the correct syntax, they isolate the semantic component of programming from the syntactic component, mitigating intrinsic load; (2) the code segments are miniature worked examples, which have been shown to be effective, low-cognitive-load instructional methods, with the completed block being a larger worked example; and (3) the interface can be simple without the redundant information or distracting elements of a development environment [15]. They have been shown to be highly effective in providing programming instruction, at least comparable to traditional code-writing activities [11], [12]. By using cognitive load as a framework to examine different implementations of Parsons Problems we posit that we can identify best-practices for Parsons Problems reducing cognitive load and increasing learning gains.

By developing Parsons Problems that address the challenges novice programmers often face, we would contribute to closing the gap in instruction provided by under-resourced schools in order to encourage more equitable access to early programming experience. Particularly with paper-based activities, schools would be able to provide the groundwork for their students to build the schemas to accommodate the complex process of writing programs from scratch using low-cost materials. Additionally, if we can identify the factors that will cause cognitive loads to decrease when completing the activities, this will provide an opportunity for novice programmers to build new schemas. Together, these efforts will help disrupt two points at which educational institutions often fail to provide minoritized and low-income students with equitable access to programming education.

Acknowledgements

The authors would like to acknowledge Sean Messerly for his work in adapting, developing, and maintaining the online platform through which the Parsons Problems were offered to students.

References

[1] B. W. Char and T. T. Hewett, “A first year common course on computational problem solving and programming,” *ASEE Annu. Conf. Expo. Conf. Proc.*, 2014.

[2] R. Bualuan, “Teaching computer programming skills to first-year engineering students using fun animation in Matlab,” *ASEE Annu. Conf. Expo. Conf. Proc.*, 2006.

[3] D. Ronan and D. Cenk Erdil, “Impact on computing attitudes and career intentions in a rotation-based survey course,” *ASEE Annu. Conf. Expo. Conf. Proc.*, vol. 2020-June, 2020.

[4] Code.org, CSTA, and ECEP Alliance, “2020 State of Computer Science Education: Illuminating Disparities,” 2020.

[5] J. Sweller, “Cognitive load theory.,” in *The psychology of learning and motivation:*

Cognition in education, Vol. 55, San Diego, CA, US: Elsevier Academic Press, 2011, pp. 37–76.

[6] F. Paas, J. E. Tuovinen, H. Tabbers, and P. W. M. Van Gerven, "Cognitive Load Measurement as a Means to Advance Cognitive Load Theory," *Educ. Psychol.*, vol. 38, no. 1, pp. 63–71, Mar. 2003.

[7] J. Leppink, F. Paas, C. P. M. Van der Vleuten, T. Van Gog, and J. J. G. Van Merriënboer, "Development of an instrument for measuring different types of cognitive load," *Behav. Res. Methods*, vol. 45, no. 4, pp. 1058–1072, 2013.

[8] S.-S. Abdul-Rahman and B. du Boulay, "Learning programming via worked-examples: Relation of learning styles to cognitive load," *Comput. Human Behav.*, vol. 30, pp. 286–298, 2014.

[9] B. B. Morrison, B. Dorn, and M. Guzdial, "Measuring Cognitive Load in Introductory CS: Adaptation of an Instrument," in *Proceedings of the Tenth Annual Conference on International Computing Education Research*, 2014, pp. 131–138.

[10] D. Parsons and P. Haden, "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*, 2006, pp. 157–163.

[11] P. Denny, A. Luxton-Reilly, and B. Simon, "Evaluating a New Exam Question: Parsons Problems," in *Proceedings of the Fourth International Workshop on Computing Education Research*, 2008, pp. 113–124.

[12] B. J. Ericson, J. D. Foley, and J. Rick, "Evaluating the Efficiency and Effectiveness of Adaptive Parsons Problems," in *Proceedings of the 2018 ACM Conference on International Computing Education Research*, 2018, pp. 60–68.

[13] B. C. Morin, K. M. Kecskemety, K. A. Harper, and P. A. Clingan, "Work in Progress: Parsons Problems as a Tool in the First-Year Engineering Classroom." ASEE Conferences, Virtual On line, 2020.

[14] V. Karavirta, P. Ihantola, J. Helminen, and M. Hewner, "js-parsons - a JavaScript library for Parsons Problems." 2018.

[15] B. J. Ericson, L. E. Margulieux, and J. Rick, "Solving Parsons Problems versus Fixing and Writing Code," in *Proceedings of the 17th Koli Calling International Conference on Computing Education Research*, 2017, pp. 20–29.

[16] J. Stachel, D. Marghitu, T. Ben Brahim, R. Sims, L. Reynolds, and V. Czelusniak, "Managing Cognitive Load in Introductory Programming Courses: A Cognitive Aware Scaffolding Tool," *J. Integr. Des. Process Sci.*, vol. 17, pp. 37–54, 2013.

[17] J. Moons and C. De Backer, "The design and pilot evaluation of an interactive learning environment for introductory programming influenced by cognitive load theory and constructivism," *Comput. Educ.*, vol. 60, no. 1, pp. 368–384, 2013.