Evaluating Parsons Problems as a Design-Based Intervention

Rita Garcia
University of Adelaide
Adelaide, South Australia
rita.garcia@adelaide.edu.au

***Abstract*—Parsons problems are a type of assessment that helps introductory programming (CS1) students learn programming by arranging fragments of code to form working programs. Parsons problems enables students to focus on the learning concepts by minimising their focus on extrinsic operations while coding, such as syntax details. Because of the success in teaching programming with Parsons problems, we investigate the possibility of using it as a design-based intervention. We examine how Parsons problems can be used by procedural programming students during the design process, and investigate how the assessment supports the use of Self-Regulated Learning (SRL) strategies during the problem-solving process. We performed usability testing, comprising a mixed-methods approach of pre-test, think-alouds, and interviews, to collect students' behaviours and experiences. The results show a variety of approaches used by students when solving the Parsons problems, including engaging SRL strategies to better understand the problem and to verify their work. We offer future research opportunities to further explore Parsons problems as a design-based intervention.**

***Index Terms*—Parsons Problems, Design Process, Design Knowledge**

I. Introduction

When given a programming assignment to complete, novices either do not devote enough time [41] or completely neglect [44] the design process when solving the problem. Novices might avoid the design process because of their fragile *design knowledge* — knowledge that helps them identify plans and goals within a programming problem [46]. When novices implement the design process, they sometimes misapply *design strategies* [13]; — components in the cognitive activity used for planning and setting goals [19]. These strategies include Self-Regulated Learning (SRL) strategies which are 'actions and processes directed at acquiring information or skill' [64]. Interventions have been shown to raise awareness in strategy usage during the problem-solving process, which can result in a higher completion rate of programming assignments [42]. Along with raising awareness in strategy usage, interventions could also help students correctly apply these strategies, which can help them improve their learning process [64].

This study investigates the use of Parsons problems as a design-based intervention and how the assessment supports students in their use of SRL strategies. Parsons problems are a type of assessment that has students arrange fragments of code to construct working programs [39]. Though students' use of Parsons problems as fragments of code to form working programs has been previously evaluated [21], to the best of our knowledge, this is the first application of Parsons problems that has students using the assessment to organise programming plans at the design level for a programming problem.

We are interested in evaluating Problems problems as a design-based intervention because of the positive outcomes prior research [33], [39] reported on when using Parsons problems to teach programming. Parsons problems allows students to focus on the learning task, and minimises other factors in software development that might impede students' learning, such as programming syntax [12]. Parsons problems can help students focus on solution space [63]; this gave us the idea to examine how the assessment might help students focus on the design process. Because the programming paradigm impacts how a solution is formulated and structured [1], we evaluate Parsons problems during the design process within the procedural programming paradigm. We seek to answer the following research questions:

RQ1: *How do CS1 procedural programming students use the Parsons problems when presented as a design-based intervention?*

RQ2: *How are Self-Regulated Learning (SRL) strategies applied by students when interacting with the Parsons problems?*

The study performs usability testing, employing a mixed-methods approach of pre-test, think-alouds, and interviews. The results from the year-long study show different approaches to solving the Parsons problems, supporting students' SRL strategies when solving procedural programming problems. This paper concludes with future research opportunities, to further evaluate the use of Parsons problems as a design-based intervention.

II. Background

A. Design Strategies

Design strategies are actions taken by novices to find answers when designing a programming solution [22]. Design strategies are useful in solving programming problems because they help in the development of plan structures that contain multiple plans that translate into basic elements that form a program [44]. CS1 students have acknowledged that design strategies can help them solving programming problems, but felt they lacked the abilities to effectively apply this strategy [16]. However, as novices gain more experience, their design knowledge — metaknowledge containing methods for finding solutions to problems [22] — improves, which results in better designed software [2] and enables them to 'perform abstract thinking and to exhibit abstraction skills' [26].

Novices have been shown to use their existing knowledge and generalised problem-solving strategies [47], but through guidance of CS-specific design strategies, they can build their knowledge coordinating goals and plans for a program [56]. Planning helps the student identify the subproblems in the textual description, and outline the purpose of the problem [5]; but for novices, they have difficulty scoping the problem, and identifying the plans for solving it [58]. To identify the goals, students have to decompose the problem [23]. Novices have been found to use a depth-first decomposition approach, investigating a specific subgoal before evaluating the other subgoals [37]. This approach works for novel problems [35], but not for more complex ones with subgoals that work together [37]. With depth-first, novices spend less cognitive effort during the design process than experts [55].

The background literature has demonstrated students are aware of the need for design strategies to solve problems, but have been found to either lack the necessary skills to apply strategies or felt their existing abilities are limited. By providing students an intervention to develop design strategies, novices might adopt good practises. For example, supporting students during the design process could help them design a solution using a breath-first decomposition approach

that systematically explores the problem before focusing on task details [45]. This approach is used by experts to evaluate subgoals at a higher level, giving them opportunities to explore the design [37] to build the design knowledge and confidence to apply as independent learners.

B. Self-Regulated Learning

Self-Regulated Learning (SRL) strategies are strategies that can help improve the learning process [64]. The development of SRL strategies by students is associated with their self-perceptions of their abilities, and the perceived purpose of engagement with the activity [38]. When applied in problem-solving situations, SRL strategies tend to improve the students' programming performance [5]. The original SRL taxonomy was formed by assessing learners' behaviours in a classroom environment. The study identified 14 SRL categories, which included self-evaluation, organising and transforming, goal setting and planning, and seeking information. From these 14 strategies, Zimmerman went on to develop a triadic model [65] that is comprised of three stages: *forethought*, *performance control*, and *self-reflection*. During *self-regulation*, CS students will likely spend more time in the design process [6] and decompose programming problems into subgoals [16]. In the final phase, *self-regulation*, students evaluate their work on tasks to determine whether they successfully accomplished them.

When students develop their SRL strategies, they reflect on the strategy they apply to determine whether the strategy was successful. This process of reflection and evaluation helps them to form new strategies for future problem-solving contexts [9]. Students' adoption of SRL strategies is not simply associated with those that are available to them, but those strategies they consider to be relevant to the situation and purpose [36]. Effective learners have been shown [5] to perform well in programming tasks by frequently using SRL strategies to increase their understanding of learning materials. When students are encouraged to use SRL strategies, there is an increase in their motivation, resulting in higher grades [25], helping them to become expert programmers [8].

C. Parsons Problems

Parsons problems are a type of assessment that provides novices with code fragments to arrange to form a working program [39]. The assessment helps novices build programs within a scaffolded learning environment consisting of curated components that guide them through the correct problem-solving path [62]. Parsons problems can increase students' learning of CS programming concepts, while reducing their cognitive load [33] — the effort placed on the students' working memory when performing a task [59]. Students use less trial and error strategies when solving programming problems, because Parsons problems engage the students' logical problem-solving skills [12]. Because Parsons problems remove syntax errors from the problem-solving process, the assessment can identify the 'logic and syntax issues students have not yet mastered' [12], which can also be applied in exam settings.

Available Parsons problem features include distractors, subgoal labeling, and feedback. Distractors are extraneous fragments that can identify students' misconceptions [20]. When distractors are included, students' learning has been found to decrease while their cognitive load increases [20]. Subgoal labeling presents the purpose of the code fragment, resulting in increased comprehension on 'the meaning and sequence of programs without having to also generate syntax' [33]. Parsons problems can provide initiated and immediate feedback. Initiated feedback is provided upon the student's request. Immediate feedback was provided in the original iteration of Parsons problems [39], which places the student's incorrect choice back to the original position in the list. Immediate feedback is discouraged if the goal of an intervention is to promote metacognitive skill development [34], such as the learning of design strategies that encourage students to evaluate their solutions.

The scaffolding within Parsons problems provides students with an environment that enables them to focus on the solution space by minimising other operations that might distract from the learning process [63]. Parsons problems can provide educators with insight into areas students misinterpret by identifying repeated patterns of mistakes related to logic and syntax concepts. The Parsons problems learning environment can also assist students in guiding them through the programming process without the need for them to focus on fine-grained details, such as syntax.

III. Related Work

This section presents related work that has investigated ways to support student development of effective design strategies in programming through learning tools. For example, *Coached Program Planning* (CPP) is an Intelligent Tutoring System (ITS) designed to help students in the early stages of programming by having them identify the problem's goals and generate steps to achieve these goals [28]. CPP is a dialogue-based tutoring system that encourages students to use pseudocode to describe the solution to the problem. The results from using CPP showed students making fewer mistakes to the structure of their programs, and reduced erratic behaviour when programming a solution. Another ITS, Program Planner (ProPL), similar to CPP, uses questioning during the design process to encourage CS1 students to use pseudocode to deconstruct problems into subgoals [29]. ProPL is a web-based ITS written in Java, and designed to help students with the decomposition of programming goals. ProPL's interface allows students to view the problem description and pseudocode while developing their programming solution. The system provides a questioning session after the student reads the problem description, which prompts them to discuss how they will achieve the problem's goals. Results from ProPL showed improvements to students' solving composition problems, partially due to the their reflecting on and thinking about problems at an abstract level; but this did not seem to help improve their skills decomposing problems. A tutoring system developed by Hu et al. [23] uses visual notation to teach students about goals and to plan strategies [53]. The tool uses the visual notation to identify the problem's goals and plans. Results from using the tool showed improvements in students' learning of programming skills, but did not result in any conclusions about the students' usage of strategies during the design process.

Exercises have been used to encourage students to develop design strategies [16]. These exercises gave students the opportunity to practise SRL design and planning strategies [15]. Students in the study were given two compulsory exercises that encouraged them to describe their software development processes. The results showed additional scaffolding assistance is needed to support students' mastery of SRL strategies as they progress in their studies.

The related work shows different approaches used to support students learning of design strategies, where the approaches include learning tools and exercises. These activities were designed to get students to further reflect on the problem, either through questioning or self-reflection activities.

IV. Study Method

This study applies usability testing to evaluate how students use Parsons problems as a design-based intervention. This section

describes the design of the intervention and the usability testing performed, describing the context, pre-test, think-alouds, and interviews.

A. Intervention Design

The design-based Parsons problems intervention was designed for procedural programming assignments used in a CS1 course offered at a large university in Australia, where the course offers minimal exposure to design strategies. The interventions were designed to work in combination with the assignment's problem description, and followed the description when presented in Canvas, a Learning Management System (LMS). The programming assignments were designed for students to practise learning objectives covered during the week, and to build on prior learning objectives. The interventions were designed to allow students to interact with them throughout the development of programming assignments.

The programming problems' goals were deconstructed into plans for use in the Parsons problems, shown in Figure 1. The plans allowed students to practise decomposition problem-solving strategies, where the software implementation order is determined by the dependencies between goals and tasks [18]. The decomposition problem-solving strategy is a proposed model with four phases: *1) Construct a Representation*, *2) Search for a Solution*, *3) Implement Solution*, and *4) Stop* [18]. The interventions were designed to help the student with the first two phases that are part of the design process.

The interventions are designed to support students' development of design strategies, such as decomposition of programming goals. Decomposition of programming goals can be performed at the task [61] and plan [53] levels. Task-level design strategies focus on algorithms and data structures; while at the plan level, the design strategies identify the software components and interrelationships for the programmed solution [48]. For this study, plan-level design was selected, because 'a plan corresponds to a fragment of code that performs actions to achieve a goal' [23]. There are different plans: *strategic*, *tactical*, and *implementation* plans [54]. *Strategic* plans define the algorithm's overall strategy, *tactical* define a local strategy for solving the problem, and *implementation* focus on programming language-specific approaches for actualising the strategic and tactical plans. *Strategic* plans were selected to support novices in organising their approach to achieving the problem's goals.

When developing the *strategic* plans for the interventions, sentence structures that resemble task variation were avoided [12]. In mathematical word problems, sequence sentences provide guidance in the form of simple phrase-by-phrase translation of the problem, and though sequential sentences might minimise students' frustration and mistakes [51], using sentence structure might provide students with ordering clues. Minimising these cues can help students think about the purpose of each plan, instead of forming the correct order based on sentence structure.

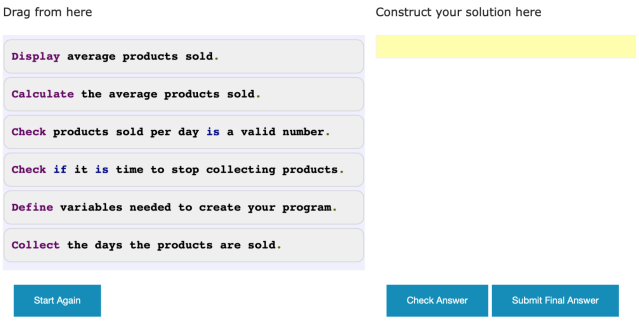

Figure 1: Intervention for Assignment 4

The developed interventions are contextualised with the aim to help students relate the plans to the assignment problem description. Figure 1 presents Assignment A4 intervention, showing six *strategic* plans that are closely aligned with the assignment's subgoals for averaging products sold over a given number of days. The intervention is presented in `js-parsons` [24], a library that supports Parsons problems within the Canvas LMS.

Through the `js-parsons` library, students can select the *Check Answer* button with line-based feedback. Students select the *Submit Final Answer* button when they are ready to submit their interactions, which sends their answers, a de-identified student ID, and the complete date to Google Sheets for further analysis.

B. Usability Testing

Usability testing is a type of observational study that records participants' interactions with the system, which researchers can later analyse [57]. This section presents the study's context and data-collection and analysis methods used within the usability testing.

1) Context: The usability testing was conducted over two semesters by a CS education researcher (lead author) with student volunteers enrolled in a 12-week Introductory Programming (CS1) course at a university. The study involved three interventions in three separate assignments; two interventions related to usability testing and were conducted in a lab environment three weeks apart, in weeks 8 and 11 of the semester.

Two research groups were involved in the usability testing. Groups V1 (August 2018) with six (1 female, 5 male) volunteers and V2 (February 2019) with four male volunteers were recruited from a class announcement. Group V1 students were selected based on their successful completion of the intervention for Assignment A4. For group V2, any student responding to the announcement was included in the study. Both groups were given vouchers and movie posters as incentives for their participation. Students received instructions on how to use the Parsons problems prior to the start of the first study session.

2) Pre-Test: The pre-test was administered to the students in groups V1 and V2 prior to the think-alouds and interviews, and was designed to have students self-assess their prior programming experience and known problem-solving strategies. The pre-test was administered in paper format, with answers transcribed to Google Sheets for further analysis.

The pre-test contained six questions. Three 5-point Likert questions asked students to self-assess their programming experience, comparing it with the experiences of their classmates and experts. The scale ranged from 'Very Experienced' (5 points) to 'Very Inexperienced' (1 point). The analysis of means approach was performed on the Likert scale questions, to identify the students' most frequent responses. Internal consistency reliability [11] was used on these questions, to measure consistency across the groups' answers.

Three open-text questions were designed to ascertain students' existing problem-solving strategies and their prior usage of programming languages. The students' responses to the open-text questions were exported from Google Sheets and imported into NVivo version 12, to code responses and extract from NVivo as a matrix to identify common themes. Thematic content analysis [32] was used on the open-text questions, to provide insight into how students might use the intervention and report on previously used programming languages. The thematic content analysis started with fourteen nodes representing the SRL strategies.

Coding accuracy on the open-text questions involved revisiting the coded responses four weeks after the initial coding with the 14 SRL

strategies, to include five Emotional Regulation (ER) [7] strategies. The adaptation was made because some of the strategies reported by students included ER, such as expressing and suppressing emotions, when encountering frustration in the problem-solving process. The adaptation also allowed inter-rater reliability to be performed using the process defined by Mackey and Gass [31], ensuring consistency in the coding system. When the five ER strategies were added to the coding framework four weeks later, previous coded segments were revisited, to determine if coded utterances contained ER strategies. The primary author coded the responses to the pre-test and the co-authors ensured trustworthiness by providing feedback on the interpreted data [11].

3) Think-Alouds: The 30-minute think-alouds were conducted following the pre-test. The think-alouds were audio and visually recorded using `SimpleScreenRecorder` [3], an application located on the lab computers. The recorded audiovisual materials were used by the authors to review and analyse the students' interactions with the Parsons problem. The primary author conducted the think-alouds, taking observational notes and prompting students when extended periods of time lapsed with no utterances.

Procedural analysis was applied to the think-aloud recordings to identify students' procedural steps, and to generate a flowchart [52]. Displaying results displayed in a flowchart is an established visualisation approach [49], presenting a list of the procedural tasks that are mapped to the students' behaviours, demonstrating their common actions and problem-solving approaches.

Upon completion of the procedural analysis, the cognitive task analysis was performed by comparing the think-aloud transcripts to the procedural tasks students performed, as observed in the audiovisual files. Cognitive tasks analysis is a type of analysis used to understand tasks that require cognitive processing [10]; it uses procedural analysis methods [52] that involves temporal ordering of procedural tasks to define the mental and physical steps used by the students to complete an activity. The result is a cognitive process protocol fragment explaining a student's actions [49].

4) Interviews: Task-based interviews were conducted after the think-alouds. The interviews were audio recorded by the interviewer. As a guide, nine questions were prepared that related to the assignment description and the Parsons problem. A set of questions were prepared, and the interview was conducted using a retrospective protocol, a form of protocol analysis [14] that discusses the observations made after the problem-solving process. The pre-set interview questions guided the interview sessions, along with notes made by the interviewer during the think-alouds.

Directed content analysis was used to analyse the interview data, where the analysis can place the data in the constructs of an established theory [32]. The constructs used in the analysis were the 14 SRL strategies developed by Zimmerman to determine the extent that students adopted SRL during the activity [64]. The interview responses were coded using the established theory to generate coding frequencies, to interpret the findings from the conversations. The Kappa value was 0.87, indicating excellent agreement for the inter-rater reliability [27]. Coding frequencies from the directed content analysis results were extracted using the matrix table from NVivo version 12.

To verify the coding, the authors discussed the results from the initial coding. The initial coding contained the SRL strategies, but it was decided to include the five Emotional Regulation (ER) [7] strategies in the coding framework because the strategies were addressed in the pre-test. As a result of adding the ER strategies, all the existing coded responses were revisited, to determine if the ER strategies were included in the coded utterances.

Group	V1 N=6 (Aug 2018)	V2 N=4 (Feb 2019)
Self Assess	Somewhat Experienced (2.17)	Inexperienced (1.8)
Classmates	Experienced (3.33)	Inexperienced (2.4)
Experts	Very Inexperienced (1.5)	Very Inexperienced (1.0)
Languages	Python (n=1), C++ (n=2)	Python (n=1), Mathlab (n=1)

Self-Regulated Learning Strategies			
Strategy	**V1**	**V2**	**Examples**
Organising & transforming	4 (21.1%)	3 (15.8%)	*"I try to divide the problems in parts for easier understanding then solve each of those"* and *"Draw out and expand the problem. Explain each part to myself."*
Goal-setting & planning	2 (10.5%)	2 (10.5%)	*"Collect the data for the assignment, and then plan how I will write it down."*
Seeking information	2 (10.5%)	1 (5.3%)	*"I refer to online resources and books to solve the problem."*
Keeping records & monitoring	0 (0.0%)	1 (5.3%)	*"Using techniques I've been taught - refer to notes."*
Other	3 (15.8%)	1 (5.3%)	*"First I will try to solve it by myself. After that I will get some help from tutors, online resources, friends."*
Emotional Regulation Strategies			
Strategy	**V1**	**V2**	**Examples**
Re-appraise situation	3 (25%)	4 (33.3%)	*"Take a break, and do something recreational to get my mind off of it"* and *"Revisit it at a later date. Fresh perspective can help."*
Social support	2 (16.7%)	0 (0.0%)	*"I refer a lot to online resources and books to solve the problem."*
Denial & distraction	2 (16.7%)	1 (8.3%)	*"...if it's a very difficult problem, I usually leave it based on the time left"* and *"Often move onto other problems."*
Expressing emotions	1 (8.3%)	0 (0.0%)	*"Try to make negative things into positive."*

Table 1: Identified SRL and Emotional Regulation Strategies

V. Usability Testing Results

A. Pre-Test Results

This section presents the results from the pre-test. Table 1 presents the students' responses to the Likert scale (1—5) questions with the analysis of means. In the reliability analysis for these questions, Cronbach's alpha was found to be 0.97, a relatively high internal consistency. Table 1 shows the participants we report on for the results. Though the study started with five participants in group V2, one participant was excluded from the results because she was behind in her assignments and could not participate in the think-alouds. Table 1 shows group V1 participants (n=6) ranking their programming experience higher (Δ_{V1V2}=0.37) than group V2 participants (n=4), possibly due to two V1 students stating 2 — 3 years experience using Python and C++. The remaining V1 students and group V2 stated their programming experience started with this course. Group V1 rated themselves experienced (V1=3.33) compared to their classmates, while group V2 rated themselves a bit below experienced (V2=2.4). The higher rating from V1 was due to the two students with prior programming experience. When comparing their experience to experts, both groups responded as very inexperienced (V1=1.5, V2=1.0).

The pre-test asked students to provide their known problem-solving strategies. Table 1 presents example student responses within the coded framework with SRL and Emotional Regulation (ER) strategies. The most common SRL strategy for both groups (V1=4 21i%, V2=3 15.8%) was *Organising & transforming*, 'student-initiated overt or covert rearrangement of instructional materials to improve learning' [64]. Students also stated they seek assistance through online resources and peers (V1=2, V2=1), and review notes (V1=1, V2=0). They also used *Goal-setting & transforming*, such as *"Collect the data for the assignment, and then plan how I will write it down"*, and time management (V1=1, V2=2), for example, *"Try and evenly distribute the workload over the week to give myself the least stress to completion"*. The students' statements show that during the problem-solving process they used the ER strategy *Re-appraise situation* (n_{V1}=3, n_{V2}=3), where they take breaks and revisit the problem later. Most of the responses related to ER strategies show that they help the student complete the problem, except for *Denial & distraction* (n_{V1}=8.3%, n_{V2}=0.0%), a procrastination strategy that can interfere with the learning process [60] and promote task avoidance [7], which can result in an incomplete solutions. From the responses, it is unclear whether the participants used *Denial & distraction* in combination with another strategy, such as *Re-appraise situation*, to complete problems, or whether the *Denial & distraction* ER strategy was used at the end of the problem-solving process.

	Students					
Procedural Tasks	**V1.2**	**V1.4**	**V2.1**	**V2.2**	**V2.3**	**V2.4**
Environmental structuring	•	•	•			
Reads problem description	•	•	•	•	•	
Begins programming		•	•			
Seek information			•			
Evaluates all plans		•		•		•
Identifies first plan	•		•		•	•
Evaluates remaining plans	•		•		•	•
Arranges plans	•	•	•	•	•	•
Initiate self-evaluation	•		•	•	•	
Re-organises incorrect plan			•		•	
Initiate self-evaluation			•		•	
Submit answer	•	•	•	•	•	•

Table 2: Task Analysis for Assignment A6

B. Think-Aloud Results

We present the think-aloud results using two views. The first view shows the procedural steps taken by the students interacting with the third intervention, Assignment A6, shown in Table 2. The table presents the procedural steps in temporal order, showing students' interactions in the order they were performed. All four students in group V2 are represented in the table, while two out of the six students in group V1 are reported. Two V1 students were excluded because they completed the intervention before the think-alouds. These two students reported in the pre-test having 2-3 years prior programming experience, so their prior experiences might have helped them complete the activities faster. Another V1 student was excluded because she was behind in her assignments, while the fourth V1 student's data was corrupted and could not be analysed.

All the students presented in Table 2 were observed arranging the plans in the Parsons problems in linear order, where plans are positioned in textual order. Students arranging the plans linearly might suggest mental arrangement of the plans. A potential reason for the students' mentally organising the plans can be gleaned from the pre-test, where *Organising & transforming* SRL strategy was stated (36.9%) as a prior approach used when solving problems. The students' prior usage of this strategy might suggest it contributed to their ability to organise the plans mentally, instead of the using the intervention as a supplement for the organisation process.

Table 2 shows students using SRL strategies. Three (50%) students used the *Environmental structuring* SRL strategy, preparing their environments for better learning by adjusting their computer desktops to view the assignment description and the Integrated Development Environment (IDE). One student (16.7%) applied the *Seeking information* SRL strategy to better understand the assignment's problem description and plans in the Parsons problems. The *Self-evaluation* SRL strategy was used (66.7%) by students to check their answers with the Parsons problem's feedback feature.

Student's Comments	**Analysis**
1: `I'm not making a full-on plan when I see them,`	Student acknowledges how he is viewing the activity to help organise the plans.
4: `besides probably the very last one, because those ones are very obvious.`	Student identifies the placement of the last plan.
8: `But the ones in the middle, they're usually just not the first and they're not the last.`	Student reviews all the plans, acknowledging the middle plans are not first in the organised list.
13: `Once I see the first one, then I move onto the next.`	Student visits all the plans, and identifies the first one in the organised plan.
16: `But the other ones aren't already in my head yet.`	Student has not reflected on the middle plans.
19: `I don't put them into an order in my head first.`	Student does not reflect on middle plans.
22: `But the middle ones, I don't order them.`	Student articulates an organisation strategy, except the middle ones.
24: `If it's more simple then I compare them to each other, but I don't keep track of which one should be in which order until I get to the next one.`	Student uses negotiating tactics to determine the ordering of the middle plans, because they require more programming logic to perform.

Figure 2: Protocol Fragment for Student V2.1

The next analysis explored the students' cognitive processes, to help understand students' behaviours and experiences better during the use of the Parsons problems. The results from the task analysis (See Table 2) show students arranging the plans in linear order. The cognitive tasks are layered over the procedural steps to form protocol fragments, showing various approaches performed by the students when mentally arranging the plans in linear order. Figures 2, 3, and 4 are protocol fragments from students interacting with the activity without referencing the problem's context, while Figures 5, 6, and 7 are examples with students leveraging the problem's context. The figures display the students' transcribed think-alouds in the left-hand column with line numbers, and the right-hand column provides an explanation of their verbalisations.

Figures 2 and 3 show the cognitive processes for Students V2.1 and

Student's Comments	Analysis
1: I would have at least, maybe, two... Depending on how many options there are, two of these grey boxes, one here and one here.	Student begins to evaluate the overall layout of the activity, to determine plans he needs to arrange.
7: And then I would, yes, organise what I would think has to come first, what has to come second.	Student begins to negotiate the ordering of plans.
12: And then, I would look in this and see...how to organise this.	Student evaluates the order is correct.
Student validates answer using 'Check Answer' button.	
15: If something doesn't add up in this way, I would then check here	Student uses feedback to determine if his work is correct.
18: and see what's in here that's missing from here, and then I would swap it out.	Student uses the corrective feedback from the activity, to adjust the plan he organised.

Figure 3: Protocol Fragment for Student V2.2

V2.2 working on Assignment A6. Student V2.1 is using means-ends analysis [59] to examine the plans within the current state to reach the ultimate goal state. Student V2.1 applies his prior experiences to help him identify plans previously encountered, and understand where these plans reside in the organised list [43], [47]. Student V2.2 performs the organisation process by searching for the first plan, instead of using prior experiences to identify familiar plans (line 7 Figure 3). To help with the organisation process, Student V2.2 uses the Parsons problem feedback feature for help during the organisation process (line 15 Figure 3), while Student V2.1 closely inspects the plans to decide the order (line 22 Figure 2). Both students' problem-solving approaches show them thinking about the overall problem, a similar problem-solving approach to an expert [50], [17].

The protocol fragment presented in Figure 4 shows Student V1.1 using a trial and error problem-solving approach for Assignment A5, an approach also observed [21] when students solved Parsons problems to construct working programs. She was observed not thinking about the plans when interacting with the activity, stating *"what you end up doing is trying to get it all green (correct), rather than thinking what you want to do it"* (lines 3-8 Figure 4). The

Student's Comments	Analysis
1: I guess it gives an idea, I think	Student acknowledges the activity helps.
3: and what you end up doing is trying to get it all green,	Student performs trial & error strategy.
6: rather than actually thinking what you want to do it.	Student acknowledges not thinking about plan organisation to solve the problem.
9: The hard part was actually getting a pattern to work.	Student acknowledges the trial & error strategy is not an easy problem-solving approach.

Figure 4: Protocol Fragment for Student V1.1

Student's Comments	Analysis
1: So we'll set up a basic grid. Setting up the grid obviously would have to come first.	Student identifies and arranges the first plan in the list.
5: That one has to be last, resetting. We don't want to check the square field yet.	Student determines the next plan in the ordered list.
9: That probably and then that. Implementing the highlighting, then the filling.	Student identifies and arranges the next plans in the list.
13: Then we have to see when the grid is full. When that happens, we have to reset.	Student identifies the final plans in the ordered list.
Student validates answer using 'Check Answer' button.	
17: Yes, seems good.	Student checks work.

Figure 5: Protocol Fragment for Student V1.2

statement shows she is aware the approach is not the best approach for understanding the problem (line 6 Figure 4). She did find the intervention encouraged her to think about the problem, stating *"I guess it gives an idea, I think"* (lines 1-2 Figure 4).

Figure 5, like Figure 2, shows Student V1.2 using relationships between the plans to help with the organisation process, but Student V1.2 includes the plan's purpose when negotiating the plan's placement (lines 2-4 Figure 5). Unlike Student V2.1 in Figure 2, who negotiates placement without context, Student V1.2 compares the relationship between the plans in the context of the problem's workflow. Student V1.2 rationalises his selection of the first plan *Set up a basic grid*, stating *"setting up the basic grid obviously would have to come first"*, while providing reasons for the last plan's placement, *Reset grid when filled*, stating *"That one has to be last, resetting"*. This statement demonstrates that he understands the completion of the problem. Between identifying the first and last plan, he negotiates the placement of the middle plans in the list, stating *"We don't want to check the square field yet"*.

Student's Comments	Analysis
1: All right, so place ball on game board.	Student assumes the second plan is the first in the list.
3: Actually, construct basic game... Okay, that one is first.	Student realises his mistake, and adjusts the ordered list.
6: Okay, and then try building another and check the ball is placed in hole.	Student identifies the remaining plans to add to the list.
Student validates answer using 'Check Answer' button.	
10: Yes, done.	Student checks work.

Figure 6: Protocol Fragment for Student V2.4

Student V2.4, shown in Figure 6, negotiates (line 1-2 Figure 6) the plan order for Assignment A6 by drawing on the problem's context. Student V2.4 starts with evaluating the first plan presented in the intervention. During the think-aloud, *Place ball on gameboard* was the plan presented in the first randomised list. He assumed that was

the first plan, but he adjusts his decision after evaluating the other plans in the list, stating *"construct basic game... Okay, that one is first"*. Had the *Place ball on gameboard* plan been presented at the bottom of the list, the student might have approached the organisation process differently, such as reflecting on the remaining plans before making a decision.

Student's Comments	Analysis
1: `You have to make the basic setup, where the ball's position is, that's what the first thing is.`	Student identifies the initial plan, and associates it with other plans.
6: `And then, yes, because, based on the ball's position, the hole is random, and so the game changes according to that.`	Student continues to bring in other requirements to the problem, such as randomising the location of the initial hole in the game.
12: `And then, you have to check the ball's position to see if it's in there. There's no point in tracking it around if the ball's already in the hole.`	Student considers ways to make his solution robust.
Student begins to interact with the activity by arranging plans.	
19: `I think the meaning of this plan, the construct basic game board, is just creating the playing space.`	Student evaluates the first plan to organise.
24: `Okay. Oh, yes, that... If it's just that then it's... This should be in the front.`	Student is justifying the ordering of the first plan.
28: `Yes. Ball is placed in the hole. That makes sense. Yes, because I would try either way; it depends on how you see it.`	Student examines how the plans relate through his interpretation of the problem.

Figure 7: Protocol Fragment for Student V2.3

Figure 7 demonstrates Student V2.3 using the intervention for Assignment A6, to better understand the problem. This is demonstrated by him first reflecting on the plans to construct the workflow of the program (lines 1-18 Figure 7). Student V2.3 describes the workflow by using the plans to initiate thoughts on cases that will make his solution more robust. For example, the *Place ball on gameboard* plan has him consider how the ball will be tracked on the gameboard and cases where tracking is not required (lines 12-15 Figure 7). Upon reaching the end of the workflow (lines 15-18 Figure 7), Student V2.3 interacts with the plans in the intervention. When interacting with the plans, his mental model of the solution does not align with the plans to organise, stating *"Ball is placed in the hold. That makes sense. Yes, because I would try either way; it depends on how you see it"* (line 28-33 Figure 7). This example shows that though the student mentally constructed a workflow for solving the problem, his understanding of how to approach it was different from the correct path, and the interactions with the intervention helped adjust his understanding.

ID	V1	V2	Theme	Examples
T1	0 (0.0%)	1 (3.8%)	Environmental structuring	*"...have it split screen, and have the coding open here...I have a problem of getting distracted and going off course occasionally, so I've gotten into the habit of having what I need to be doing very clear and obvious to me."* Student V1.1
T2	2 (7.7%)	3 (11.5%)	Organising & transforming	*"helps to find the start in the program"* Student V1.3, and *"Helps organise thoughts and what to do next"* Student V1.4
T3	3 (11.5%)	5 (19.2%)	Self-evaluation	*"validate as hints"* Student V1.1, and *"I guess in doing that also showed me what doesn't work in a way"* Student V2.2
T4	2 (7.7%)	2 (7.7%)	Goal setting & planning	*"ordering plans helpful"* Student V1.2, and *"Just what the process was to get to my end goal was the way I was thinking about it, like a logical order of how to get there"* Student V2.2
T16	3 (11.5%)	0 (0.0%)	Motivation	*"easier understanding and easier working while we do the problem"* Student V1.3
T17	1 (3.8%)	0 (0.0%)	Poor Learning Behaviours	*"making it green"* Student V1.1
T18	4 (15.4%)	0 (0.0%)	Too easy	*"too basic"* Student V1.5, and *"would like to have more task"* Student V1.4

Table 3: Narrative Interview Coded Framework

C. Interview Results

Table 3 shows the interview results divided into groups V1 and V2, and lists the total number and percentage of narrative segments for the seven themes that emerged from the interviews. The results show four SRL strategies were identified and no Emotional Regulation (ER) strategies. The identified SRL strategies are *T1 Environmental structuring* (3.8%), *T2 Organising & transforming* (19.2%), *T3 Self-evaluation* (30.7%), *T4 Goal setting & planning* (15.4%). The feedback also shows positive, *T16 Motivation* (11.5%) and negative, *T18 Too easy* (15.4%), feedback.

The most common response was the use of the SRL strategy *T3 Self-evaluation* (n=8, 30.7%). For example, Student V1.6 used the intervention to check his understanding of the problem; while Student V2.2 used it to verify he was done with the assignment, stating *"To me, feels like the end point. I'm going to figure out what steps I need to take to get to the end-point."*

One student discussed preparing his environment for learning, an SRL strategy, but in the think-alouds, three (50%) students were observed using this strategy (See Table 2). It is possible students might not have recognised setting up their environment as an important step in their learning process.

One student acknowledged using Poor Learning Tendencies, habits that compensate for the student's lack of understanding [4]. She interacted with the intervention for the purpose of *"making it green"*, a visual feedback from the Parsons problem that denotes success. The motivation to complete the activity over understanding how to solve the problem is a behaviour previously observed when students

debug code [40], where they alter code without understanding the implications to their changes, in the hopes of getting the program to work.

The students provided feedback on how to improve the intervention. Students stated the intervention was *"too easy"* (T18, V1=15.4%), and suggested more plans to make it *"more challenging"* and *"probably would use that to organise"*. Their suggestion that the intervention was easy might explain their approach to mentally organising the plans observed during the think-alouds.

VI. Discussion

The study results provided outcomes to help answer the research questions. The first research question *How do CS1 procedural programming students use the Parsons problems when presented as a design-based intervention*, while the second question asked *How are Self-Regulated Learning (SRL) strategies applied by students when interacting with the Parsons problems?* The results showed the participants using the Parsons problems to better understand the problem, but also applying trial and error to simply find the correct answer. Participants were also using SRL strategies to solve the Parsons problems. The SRL strategies, such as self-explanation and planning, used by the participants are strategies applied in a previous study [30] that examine students' iterative problem-solving behaviours while coding programming solutions. Through the think-alouds, we observed students' interactions with the Parsons problems, using it to design solutions and utilising the Parsons problem's feedback to validate their problem-solving approaches. Students' use of SRL strategies in both the design and implementation processes show that the design-based Parsons problem could serve as supplemental support for students to practise these strategies.

VII. Threats to Validity

The study has limitations and contextual variables. The study had ten volunteers, but for the think-alouds, the sample size was smaller (n=6) with male-only volunteers. Future studies can employ different recruiting strategies to diversify the study group with a balanced gender representation. In addition, think-alouds place additional cognitive load on students [14], and may have influenced the collected data. However, think-alouds are necessary because they are the closest approximation for collecting thoughts while performing tasks. Students were not asked in this study if they had prior experiences with Parsons problems. Prior usage might have influenced how they used the activity. Future studies will ask students about their previous Parsons problems experiences.

VIII. Conclusion

This study examined a novel application of Parsons problems as a design-based intervention, observing how students adopted problem-solving approaches when interacting with the activity. Students' approaches to arranging *strategic* plans showed positive usage, such as their reflection in developing robust solutions. From this study, we learned the intervention was easy due to the small number of *strategic* plans. Future research can present more plans to determine if there are changes to problem-solving approaches. Another future research opportunity could explore the Parsons problems with *tactical* and *implementation* plans, to determine whether these plans might generate different approaches to solving the activity. Another future research opportunity can evaluate Parsons problems as a design-based intervention for the functional programming paradigm. Future research could draw comparisons on how students approach the design process based on the programming paradigm.

Students involved in this study were observed using design knowledge when solving the intervention, such as analysing the relationship between plans to organise an implementation order, but were also seen gaming the activity. Students used the intervention to better understand the problem, to consider the program's robustness, and to reflect on the implementation order. The results demonstrated Parsons problems as a design-based intervention can have learning benefits outside of constructing working programs, such as promoting the use of SRL strategies like self-evaluation. The results suggest Parsons problems can be used as an activity to scaffold students through the design process.

References

[1] A. Ambler, M. Burnett, and B. Zimmerman. Operational versus definitional: A perspective on programming paradigms. *Computer*, 25(9):28–43, Sept 1992.

[2] C. Atman, J. Chimka, K. Bursic, and H. Nachtmann. A comparison of freshman and senior engineering design processes. *Design Studies*, 20:131–152, 1999.

[3] M. Baert. Simplescreenrecorder. https://www.maartenbaert.be/simplescreenrecorder, 2019. [Online; accessed 11-Feb-2019].

[4] J. Baird and J. Northfield. *Learning from the PEEL experience*. Peel Publications, 2nd edition, 1995.

[5] S. Bergin, R. Reilly, and D. Traynor. Examining the role of self-regulated learning on introductory programming performance. *ICER '05*, pages 81–86, 2005.

[6] C. Bishop-Clark. Cognitive style, personality, and computer programming. *Computers in Human Behavior*, 11:241–260, 1995.

[7] M. Boekaerts. Self-regulated learning at the junction of cognition and motivation. *European Psychologist*, 1:100–112, 1996.

[8] J. Buckley and C. Exton. Bloom's taxonomy: A framework for assessing programmers' knowledge of software systems. *Proceedings of the 11th IEEE International Workshop on Program Comprehension*, pages 165–174, 2003.

[9] R. Cantwell and P. Moore. The development of measures of individual differences in self-regulatory control and their relationship to academic performance. *Contemporary Educational Psychology*, 21:500–517, 1996.

[10] B. Crandall, G. Klein, and R. Hoffman. *Working minds: A practitioner's guide to cognitive task analysis*. MIT Press, 2006.

[11] J. Creswell. *Educational research: Planning, conducting, and evaluating quantitative and qualitative research*. Educational Research: Planning, Conducting, and Evaluating Quantitative and Qualitative Research. Pearson, 2012.

[12] P. Denny, A. Luxton-Reilly, and B. Simon. Evaluating a new exam question: Parsons problems. *Proceedings of the Fourth International Workshop on Computing Education Research*, pages 113–124, 2008.

[13] F. Détienne. Software design: Theoretical approaches. In F. Bott, editor, *Software design—Cognitive aspects*, pages 21–41. Springer-Verlag, Berlin, Heidelberg, 2002.

[14] K. A. Ericsson and H. A. Simon. *Protocol Analysis: Verbal Reports as Data*. A Bradford Book, London: The MIT Press, 1993.

[15] K. Falkner, C. Szabo, R. Vivian, and N. Falkner. Evolution of software development strategies. *ICSE '15*, pages 243–252, May 2015.

[16] K. Falkner, R. Vivian, and N. Falkner. Identifying computer science self-regulated learning strategies. *ITiCSE*, pages 291–296, 2014.

[17] A. Gerdes, J. Juering, and B. Heeren. An interactive functional programming tutor. *ITiCSE*, pages 250–255, July 2012.

[18] M. Gick. Problem-solving strategies. *Educational Psychologist*, 21:99–120, 1986.

[19] J. Greeno and H. Simon. Problem solving and reasoning. In R. Atkinson, R. Herrnstein, G. Lindzey, and R. L. (Eds.), editors, *Stevens' handbook of experimental psychology: Perception and motivation*, pages 589–672. Oxford, England: John Wiley & Sons, 1988.

[20] K. Harms, J. Chen, and F. Kelleher. Distractors in Parsons problems decrease learning efficiency for young novice programmers. *ICER '16*, pages 241–250, 2016.

[21] J. Helminen, P. Ihantola, V. Karavirta, and M. Malmi. How do students solve parsons programming problems? - an analysis of interactions traces. *International Computing Education Research Conference (ICER) '12*, pages 119–126, 2012.

[22] C. Hoadley and C. Cox. What is design knowledge and how do we teach it. In *Educating Learning Technology Designers*, pages 19–34. Routledge, January 2009.

[23] M. Hu, M. Winikoff, and S. Cranefield. Teaching novice programming using goals and plans in a visual notation. *14th Australian Computing Education Conference (ACE 2012)*, pages 43–52, Jan 2012.

[24] V. Karavirta, P. Ihantola, J. Helminen, and M. Hewner. js-parsons - a JavaScript library for Parsons Problems. https://js-parsons.github.io, 2018. [Online; accessed 19-July-2018].

[25] R. Kizilcec, M. Pérez-Sanagustín, and J. Maldonado. Self-regulated learning strategies predict learner behavior and goal attainment in massive open online courses. *Computers & Education*, 104:18–33, 2017.

[26] J. Kramer. Is abstraction the key to computing? *Communication of the ACM*, 50:36–42, 2007.

[27] J. Landis and G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33(1):159–174, 1977.

[28] C. Lane and K. VanLehn. Coached program planning: Dialogue-based support for novice program design. *SIGCSE '03*, pages 148–152, 2003.

[29] C. Lane and K. VanLehn. Teaching the tacit knowledge of programming to novices with natural language tutoring. *Computer Science Education*, 15:183–201, 2005.

[30] D. Loksa and A. J. Ko. The role of self-regulation in programming problem solving process and success. In *Proceedings of the 2016 ACM Conference on International Computing Education Research*, ICER '16, page 83–91, New York, NY, USA, 2016. Association for Computing Machinery.

[31] A. Mackey and S. Gass. *Second language research: Methodology and design*. Lawrence Erlbaum Associates, Mahwah, NJ, 2005.

[32] C. Marshall and G. Rossman. *Designing Qualitative Research*. Sage Publications, London, 3rd edition, 1999.

[33] B. Morrison, L. Margulieux, B. Ericson, and M. Guzdial. Subgoals help students solve Parsons problems. *SIGCSE 2016*, pages 42–47, 2016.

[34] S. Narciss. Feedback strategies for interactive learning tasks. In *Handbook of Research on Educational Communications and Technology*, pages 125–143. Mahaw, NJ: Lawrence Erlbaum Associates, 2007.

[35] A. Newell and H. Simon. *Human Problem Solving*. Prentice-Hall, Englewood Cliffs, NJ, 1972.

[36] J. Nicholls. Students as educational theorists. In D. Schunk and J. Meece, editors, *Student Perceptions in the Classroom*, pages 267–286. Lawrence Erlbaum Associates, Hillsdale, NJ, 1992.

[37] T. Ormerod and J. Ridgway. Developing task design guides through cognitive studies of expertise. *European Conference on Cognitive Science*, pages 401–410, 1999.

[38] S. Paris and J. Turner. Situated motivation. In P. Pintrich, D. Brown, and C. Weinstein, editors, *Student Motivation, Cognition and Learning: Essays in Honor of Wilbert J. McKeachie*, pages 213–237. Lawrence Erlbaum Associates, Hillsdale, NJ, 1994.

[39] D. Parsons and P. Haden. Parson's programming puzzles: A fun and effective learning tool for first programming courses. *Proceedings of the 8th Australian Conference on Computing education*, pages 157–163, 2006.

[40] D. Perkins and F. Martin. Fragile knowledge and neglected strategies in novice programmers. *Papers presented at the first workshop on empirical students of programmers*, pages 213–229, October 1985.

[41] P. Pintrich, C. Berger, and P. Stemmer. Students' programming behavior in a Pascal course. *Journal of Research in Science Teaching*, 24:451–466, 1987.

[42] J. Prather, R. Pettit, B. Becker, P. Denny, D. Loksa, A. Peters, Z. Albrecht, and K. Masci. First things first: Providing metacognitive scaffolding for interpreting problem prompts. *SIGCSE 2019*, pages 531–537, February 2019.

[43] D. Riley. Teaching problem solving in an introductory computer science class. *SIGCSE 1981*, pages 244–251, February 1981.

[44] R. Rist. Program structure and design. *Cognitive Science*, 19:507–562, 1995.

[45] P. Robillard. The role of knowledge in software development. *Communications of the ACM*, 42(1), 1999.

[46] A. Robins, J. Rountree, and N. Rountree. Learning and teaching programming: A review and discussion. *Computer Science Education*, 13:137–172, 2003.

[47] I. Roll, V. Aleven, B. Mclaren, and K. Koedinger. Designing for metacognition—applying cognitive tutor principles to the tutoring of help seeking. *Metacognition and Learning*, 2:125–140, December 2007.

[48] H. Rombach. Design measurement: Some lessons learned. *IEEE Software*, pages 17–25, March 1990.

[49] A. Schaafstal. Diagnostic skill in progress operation: A comparison between experts and novices. *PhD Thesis*, 1999.

[50] C. Schulte, T. Busjahn, T. Clear, J. Paterson, and A. Taherkhani. An introduction to program comprehension for computer science educators. *Proceedings of the 2010 ITiCSE Working Group*, pages 65–86, 6 2010.

[51] J. Silbert and M. Stein. *Direct Instruction Mathematics*. Columbus, OH: Merrill, 2nd edition, 1990.

[52] P. Smith and T. Ragan. *Instructional design*. John Wiley & Sons, Inc, New York, 1999.

[53] E. Soloway. Learning to program = learning to construct mechanisms and explanations. *Communications of the ACM*, pages 850–858, Sept 1986.

[54] E. Soloway, J. Bonar, J. Greenspan, and K. Ehrlich. What do novices know about programming? *Directions in Human-Computer Interactions*, 1982.

[55] T. Song, K. Becker, J. Gero, S. DeBerard, O. Lawanto, and E. Reeve. Problem decomposition and recomposition in engineering design: A comparison of design behavior between professional engineers, engineering seniors, and engineering freshman. *Journal of Technology Education*, 27(2):37–56, Spring 2016.

[56] J. Spohrer and E. Soloway. *Studying the novice programmer*. Hilldale, NJ. Lawrence Erlbaum, 1989.

[57] J. Stasko and C. Hundhausen. Algorithm visualization. In S. F. . M. P. (Eds.), editor, *Computer Science Education Research*, pages 589–672. Taylor & Francis, January 2004.

[58] A. Sutcliffe and N. Maiden. Analysing the novice analyst: Cognitive models in software engineering. *International Journal Man-Machine Studies*, 36:719–740, 1992.

[59] J. Sweller, P. Ayres, and S. Kalyuga. Intrinsic and extraneous cognitive load. In J. M. Spector and S. P. Lajoie, editors, *Cognitive Load Theory*, pages 57–69. Springer, 2011.

[60] E. Webster and A. Hadwin. Emotions and emotion regulation in undergraduate studying: Examining students' reports from a self-regulated learning perspective. *Educational Psychology: An International Journal of Experimental Educational Psychology*, 35:794–818, 2014.

[61] L. Winslow. Programming pedagogy – a psychological overview. *SIGCSE Bulletin*, 28(3):17–22, 1996.

[62] D. Wood, J. Bruner, and R. Gail. The role of tutoring in problem solving. *The Journal of Child Psychology and Psychiatry*, 17(2):89–100, April 1976.

[63] R. Zhi, M. Chi, T. Barnes, and T. Price. Evaluating the effectiveness of parsons problems for block-based programming. *ICER '19*, pages 51–59, 2019.

[64] B. Zimmerman. A social cognitive view of self-regulated academic learning. *Journal of Educational Psychology*, 81:329–339, 1989.

[65] B. Zimmerman. Attaining self-regulation: A social cognitive perspective. In M. Boekaets, M. Zeidner, and P. Pintrich, editors, *Handbook of Self-Regulation*, pages 13–39. Elsevier Academic Press, London, UK, 2000.