Rodrigo, M. M. T. et al. (Eds.) (2021). Proceedings of the 29th International Conference on Computers in Education. Asia-Pacific Society for Computers in Education

Integrating Parsons Puzzles with Scratch

Jeff BENDER[a*], Bingpu ZHAO[b], Lalitha MADDURI[a], Alex DZIENA[a], Alex LIEBESKIND[a] & Gail KAISER[a]
[a]*Programming Systems Laboratory, Columbia University, USA*
[b]*Department of Computer Science, Barnard College, USA*
*jeffrey.bender@columbia.edu

Abstract: We surveyed grade 6-9 teachers to learn teacher perceptions of student engagement with computational thinking (CT) and how well their needs are met by existing CT learning systems. The results and a literature review lead us to extend the trend of balancing Scratch's agency with structure to better serve learners and reduce burden on teachers aiming to learn and teach CT. In this paper, we integrate Parsons Programming Puzzles (PPPs) with Scratch and analyze the effects on adults, who crucially influence the education of their children. The results from our small pilot study suggest PPPs catalyze CT motivation, reduce extraneous cognitive load, and increase learning efficiency without jeopardizing performance on transfer tasks.

Keywords: computational thinking, parsons programming puzzles, scratch, motivation, cognitive load

1. Introduction

In response to a crisis in CS teacher certification and a deficit of student exposure to CS in grades K-12 (Wilson et al., 2010; Leyzberg et al., 2017), governments are enacting policies requiring CT in schools (Whitehouse.gov, 2016; The Royal Society, 2016). Additional argument (Wing, 2006, 2008) and evidence (Grover et al., 2013) provide rationale for ensuring children achieve CT competency during the formative cognitive and social development cycles throughout grades K-12. Parsons programming puzzles (PPPs), which enable learners to practice CT by assembling into correct order sets of mixed-up blocks that comprise samples of well-written code focused on individual concepts, are one approach used to introduce CT efficiently (Parsons et al., 2006; Ericson et al., 2018).

These scaffolded program construction tasks facilitate learning of syntactic and semantic language constructs underlying a CT concept. As the learner solves carefully designed single-solution puzzles, she arranges constructs from a curated assortment in a cycle of deliberate practice that exposes and addresses misconceptions (Kaczmarczyk et al., 2010; Emerson et al., 2020). Among the correct code fragments, she might find distractors which provoke cognitive conflict that reinforces learning (Karavirta et al., 2012). The partial suboptimal path distractor type, for example, might tempt a learner toward faulty progress without enabling her to solve the problem fully, thereby triggering recognition of a misconception and productive backtracking toward the correct solution (Harms et al., 2016).

Research indicates this structured approach to learning CT can lead to more efficient concept learning than alternatives such as learning by tutorial, or writing/fixing code (Harms et al., 2015; Ericson et al., 2017; Zhi et al., 2019). To measure efficiency, researchers often leverage cognitive load theory, which helps to distinguish between the complexity of the material, the instructional design, and the strategies used for knowledge construction. Since PPPs provide constrained problem spaces, they can induce lower cognitive load than that experienced when writing code with open-ended agency.

In the current study, we seek further evidence of their efficiency by integrating PPPs into Scratch, a block-based environment initially designed for informal learning that invites exploration, collaboration, and knowledge construction through personally meaningful creation (Maloney et al., 2010). K-8 teachers use Scratch more than any other coding language internationally (Rich et al., 2019), resulting in an ecosystem with over 74 million users (MIT Media Lab, 2021), and more research focus than any other environment in K-12 from 2012-2018 (McGill et al., 2020). However, historical findings indicate Scratchers infrequently demonstrate skill increases over time (Scaffidi et al., 2012), misconceive loops, variables, Booleans, nested conditionals, and procedures (Grover et al., 2017, 2018), and often adopt habits unaligned with accepted CS practice (Meerbaum-Salant et al., 2011). In a recent study of 74,830 Scratch projects, 45% contained at least one bug pattern (Frädrich et al., 2020). Instead

of problem-solving algorithmically, Scratchers often engage in bricolage (Harel et al., 1991), which involves bottom-up tinkering that does not necessarily prove productive (Dong et al., 2019).

To balance this agency with structure as recommended in (Brennan, 2013), and to encourage the development of desired habits when learning CT concepts without stifling learner creativity, researchers have designed external Scratch curricula (Brennan et al., 2014; Franklin et al., 2020), created introductory Scratch Microworlds with reduced functionality (Tsur et al., 2018), and devised learning strategies based on the Use->Modify->Create pedagogy to scaffold instruction (Salac et al., 2020). We extend this trend by integrating with Scratch PPPs with explicit goals that offer gameful scoring targets and per-block feedback to disincentivize trial-and-error behavior and steer learners toward correct solutions. We reason that if learners initially can internalize CT concepts efficiently via PPPs, they can better deepen their understanding in less-restrictive interest-driven projects such as those described in (Kong et al. 2020) that embrace Scratch's roots in constructionism (Brennan et al., 2014).

To test this reasoning, we ran a pilot study targeting adults, who comprise a general population that might not only benefit from learning CT, but who might most effectively mobilize the advancement of teaching and learning CT for all. We investigated the following research questions: **R1**) what are the effects on motivation and cognitive load when training occurs via: PPPs; PPPs with distractors (PPPDs); programming with access to all blocks and without feedback (limited-constraint-feedback or LCF)?; **R2**) what are the effects on learning efficiency for training via PPP, PPPD, and LCF? Although the 75-participant sample limits the number of statistically significant results, findings indicate: **F1**) participants self-report higher motivation when training via PPPs and PPPDs, and less extraneous cognitive load when training via PPPs than via PPPDs or via LCF; **F2**) participants training via PPPs and PPPDs experience increased learning efficiency compared with those training via LCF.

We first review the background and the required software development. We then document the study purpose, formative and summative evaluations, and results before previewing future work.

2. Background

Since PPPs emerged in the CS literature as a new form of program completion problem in 2006, the community has investigated their strengths and weaknesses. Strengths include: scaffolded support of syntax and semantics learning; solvers with prior experience perform better and need less time (Harms et al., 2016); quicker grading and less grading variability than code writing problems (Ericson et al., 2017); easier detection of learning differences between students compared to code writing and code fixing problems (Morrison et al., 2016); a moderate correlation between PPP proficiency and code writing proficiency in an exam setting (Denny et al., 2008); less completion time required than for code writing exercises with equivalent performance on transfer tasks (Ericson et al., 2017; Zhi et al., 2019); higher enjoyment and less completion time required than for tutorial users with better performance on transfer tasks (Harms et al., 2016); and a lack of significant differences in performance across gender. Weaknesses include: constriction of puzzle-design surface to maintain single-solution structure (not strictly required, but commonly enforced to maintain strengths); the invitation of trial-and-error behavior in PPPs with excessive corrective feedback (Helminen et al., 2013); and a potential ceiling effect when feedback guides most learners to solve PPPs correctly, resulting in the need to evaluate learner process in addition to learner product when assessing (Helminen et al., 2012).

The community also has explored differences in learning outcomes resulting from using different PPP elements. Evidence suggests that 2D puzzles, in which the student must not only correctly order programming constructs but also indent them correctly, are more difficult than 1D (Ihantola et al., 2011). Similarly, PPPs that conceal the number of lines of code needed for each solution section and those that include distractors are more difficult, require more time to complete, and produce higher cognitive load during training than those that specify section sizes and those without distractors (Garner, 2007; Harms et al., 2015). Learning differences continue to emerge when researchers vary these elements. For example, learners struggle more when distractors are randomly distributed among the correct code constructs than when they are paired with correct constructs (Denny et al., 2008).

To identify these strengths, weaknesses, and learning differences between PPP elements, researchers often leverage Cognitive Load Theory (CLT) (Sweller, 2010). According to CLT, the brain provides limited short-term memory and processing capability along with infinite long-term memory,

and learning occurs via schema construction and elaboration that leads to automation. Construction ensues by combining new, single elements into one larger element, and elaboration follows by adding new elements to an existing, larger element. Through intensive practice, individuals can automate their processing of these larger elements so that they execute without controlled processing.

CLT helps distinguish characteristics of and between PPP systems by offering a framework with tools to measure the three types of cognitive load experienced: intrinsic, extraneous, and germane. The total number of interacting elements perceived by the learner determines intrinsic load; the sometimes-impeding organization and presentation of the content determines extraneous load; and the instructional features necessary for schema construction determine the germane load. PPP designers aim to reduce extraneous load to free learners' capacity to contend with germane load when attempting to maximize learning efficiency. For example, the pairing of distractors with correct constructs might increase germane load by focusing student attention on the intended, misconception-revealing differences between two solution options, while also reducing extraneous load by eliminating the need to search for and identify the two relevant options amidst a random distribution of constructs.

To measure relative learning efficiency quantitatively across conditions, researchers calculate instructional and performance efficiency (van Gog et al., 2008). These calculations account for learners who compensate for an increase in mental load by committing more mental effort, thereby maintaining constant performance while load varies. The data recorded often include empirical estimates of mental effort during instruction (EI) and transfer (ET) tasks and the performance (P) on transfer tasks. The EI and P calculation measures the instructional efficiency of the learning process, while the ET and P calculation measures the performance efficiency of the learning outcome. For example, in a study that included interactive puzzles in the transfer phase, results indicate PPPs with randomly distributed distractors decrease performance efficiency (Harms et al., 2016). In our study, we measure instructional efficiency with a focus on learning process economy.

3. Software Development

Figure 1. Design, Play, and Assessment Functionality integrated via PPP in Scratch.

To investigate our research questions, we modified Scratch to facilitate the design, play, and assessment of PPPs. Aligned with the gamification strategy described in (Tahir, Mitrovic, & Sotardi, 2020), in which the game elements were added to SQL-Tutor, and similar to recent iSnap integrations offering progress panels and adaptive messages (Zhi et al., 2019; Marwan et al., 2020), we augmented Scratch to influence the behavior of learners. As shown in Figure 1a, we first established a design mode which enables content developers to assign points to individual blocks and select blocks for inclusion in a new PPP palette. Equipped with this functionality, teachers can assign higher point values to blocks relevant to the CT concept studied and can isolate in a single palette blocks pertinent to the puzzle.

As presented in Figure 1b, we next established a play mode which enables students to load PPPs in a manner that displays the designed animated elements in the Scratch stage, but none of the blocks in the scripts pane authored as the solution. Technical detail is reported in (Sulaiman et al., 2019), but relevant to this study is an assessment system that includes a gameful scoring algorithm intended to encourage deliberate practice and discourage trial-and-error behavior. The longest common

subsequence feedback algorithm described in (Karavirta et al., 2012) inspired this approach; ours differs in that we leverage block points, use them and subsequence length as multipliers, and sum the multiples from all subsequences matching the single correct solution while also deducting for incorrectness in absolute position. The closer the participant is to the solution, the higher the score.

Lastly, we added auto-initialization and auto-execution to reflect progress visually after each block placement during puzzle play. These mechanisms enable the display of gameful animations while an avatar presents per-block correctness feedback. They also calculate completion progress so that the learner receives appropriate feedback when she correctly solves the puzzle or the allotted time expires.

4. Study Purpose

This extended functionality positioned us to fill gaps in existing research. One study purpose was to explore the adult-use of CT learning system functionality primarily designed for children. Recent research has: 1) found significant correlation of motivation and previous programming experience with self-efficacy and inclination toward a CS career in elementary students (Aivaloglou et al., 2019); 2) indicated drag-and-drop programming can increase three CS motivational factors in middle school (Bush et al., 2020); 3) suggested computing experiences prior to university can affect the world-image of computing habits, perceptions, and attitudes which enable or inhibit pathways into CS (Schulte et al., 2007); and 4) illuminated benefits of community commitment and a CS/CT focused ecosystem inclusive of the home and community (Cao et al., 2020; DeLyser, 2018). Since demographic factors can drive communal values, and perceptions of how computing fulfills those values can affect sense of belonging and student retention (Lewis et al., 2019), we measure adult motivation and cognitive load while probing for attitudinal change that might influence the CT inclination for participants' children.

A second purpose was to further identify PPP elements that optimize learning efficiency, since the behavior of programming environments can affect novices' learning (Karvelas et al., 2020). While many researchers have hypothesized (Denny et al., 2008) and less often produced evidence (Ericson et al., 2018) that PPPs can result in more efficient learning than alternatives such as writing or fixing code, recently some have attempted to measure the contributions of various PPP elements (Kumar, 2017, 2019a, 2019b; Sirkia, 2016). We measure PPP learning efficiency with and without distractors, while offering a comparison to programming with LCF. Derived from the literature, our hypotheses were: **H1**) PPP and PPPD training increase motivation and reduce extraneous cognitive load compared to training via programming with LCF; **H2**) PPP training yields highest learning efficiency.

5. Formative Evaluation

As an early step in a roadmap of studies intended to explore the efficacy of adding gameful systems to novice programming environments, and with an aim to reinforce construct validity, we engaged in a formative evaluation with grade 6-9 educators. Through design thinking activities, we advanced our learning design technique, similar to the approach described in (Kashmira & Mason, 2020). Our goals included: 1) identifying the CT concepts receiving focus; 2) eliciting the pedagogical needs of practicing teachers; 3) and refining puzzle and feedback systems. We focus discussion here on goal 1.

5.1 Participants

The participants included 21 teachers from learning organizations such as Girls Who Code and codeHER, and 17 from U.S. schools. 11% had taught with Scratch for at least 2-4 years, 63% for 6-18 months, and 26% had instructed with Scratch for less than 6 months. 34% of the teachers used Scratch for at least 51% of their curriculum, 29% used it for 26-50%, and 29% used Scratch for at least 11-15%.

5.2 CT Concept Engagement

(Ihantola et al., 2016) highlights the concerning status quo in which most studies in the field focus on a single institution and a single course, without validation by subsequent replicating research, leading to

limited understanding of the reasons results occur. To contribute replication results, and to identify the CT concepts receiving focus, we distributed a survey that included a question from a survey previously distributed to K-9 teachers in five European countries (Mannila et al., 2014). This question asks teachers to respond with their perceptions of student engagement in nine facets of CT. Since we targeted a narrower set of teachers in the U.S., it is perhaps unsurprising that the results do not match the earlier international study, in which teachers reported their students most frequently use CT concepts related to data (e.g. analysis). However, we present this finding to reinforce the replication concerns raised, and to underscore the challenges the community faces when attempting to disseminate CT globally.

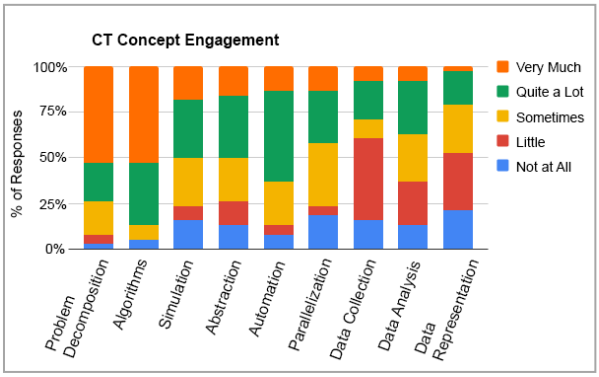

Figure 2. How a Small Sample of U.S. Teachers Perceive Student Engagement in CT Concepts.

Our findings in Figure 2 indicate teachers perceive their students engage in data CT concepts less than others such as abstraction and algorithms. Aside from the differences in population samples, and the associated threat to internal validity due to implicit differences in curricula (Barendsen et al., 2015) notes a low ratio of data knowledge in K-9 U.S. CSTA materials, 2%, compared with the English national curriculum, 14%, English Computing at School, 16%, and Italian guidelines, 25%), an extra explanation for this contrast could be related to the respondent recruitment process, as we specifically targeted Scratch teachers, whereas the earlier one did not. Since the small sample introduces a threat to external validity, future studies could try to replicate these results while controlling for technology and teacher pedagogical content knowledge utilizing a Content Representation approach like the one described in (Grgurina et al., 2014). Regardless, the lack of student engagement with data warrants investigation, as it is an alarming result for an increasingly data-driven society.

6. Summative Evaluation

6.1 Study Design

The formative evaluation helped us prioritize development, craft learning materials, and organize an initial summative evaluation via a 10-step between-subjects study through Amazon Mechanical Turk (Amazon, 2021). Participants used written instructions to guide their solving of four puzzles, and responded to a validated CS cognitive load component survey (CS CLCS) (Morrison et al., 2014), to a programming attitude Likert scale survey derived from categorized text-based responses by adults learners in (Charters et al., 2014), and to an intrinsic motivation Task Evaluation Questionnaire (TEQ) (SDT). Protocol materials are publicly available to facilitate replication at https://bit.ly/3uhSUd7.

We randomly assigned participants to one of three independent variable conditions: 1) PPP; 2) PPPD; 3) LCF. The dependent variables included time and performance on the pre/posttests, time and block moves in puzzles, and the cognitive load, programming attitude, and TEQ results.

6.2 Materials

We tested and refined our materials in collaboration with a high school teacher, 16 of her freshman physics students with little prior exposure to CT, and eight undergraduates with diverse majors. Tests

included trials of the surveys and puzzles, and think-alouds in which the participant would interact with puzzles while verbalizing her thoughts. Results led to refinements such as puzzle theme modification, normalization of pre/posttest difficulty, and simplification of language used in survey questions.

6.3 Participants

Given Wing's mobilizing declaration that CT is a "fundamental skill for everyone, not just for computer scientists" (Wing, 2006), and with the interventionist spirit of design-based research (Barab, 2014), we sought a learner population that might not otherwise encounter an opportunity to engage with CT but might influence its trajectory in the lives of children. We recruited 75 adults with varying degrees (24% high school, 60% undergraduate, 16% graduate) and a variety of self-reported programming experience (low: 38; medium: 26; high: 11). 46 men and 29 women comprise the sample population sourced from eight countries including the U.S. (60%), India (20%), and Brazil (11%).

7. Analysis & Results

7.1 Data Collection & Processing

We created seven surveys in Qualtrics to capture data not directly collected by our CT learning system. To help measure performance and efficiency, we added instrumentation to: 1) record time from puzzle start until submission; 2) trace each block moved; and 3) calculate score. Since the data did not exhibit Shapiro-Wilk normality ($p<0.05$), we used non-parametric statistics, including Kruskal-Wallis and Pairwise Comparison of Condition tests between-subjects, and Wilcoxon tests within-subjects, to address skewness and kurtosis. We used guidelines for characterizing effect sizes in (Fritz et al., 2012).

7.2 Cognitive Load

We did not find significant differences in overall cognitive load during training between conditions. Upon analyzing subtypes, we found no notable differences in intrinsic and germane load, but moderate differences in extraneous load (PPP: M=3.12, SD= 3.26; PPPD: M=3.55, SD=3.62; LCF: M=3.90, SD=3.62). This result supports **H1**, as PPP participants self-reported lower extraneous load than PPPD participants, while LCF participants reported the highest. Since the LCF condition presented far more block choices and block search options than the PPPD condition, which in turn presented more choices than the PPP condition, this result indicates reducing impediments to block identification frees capacity for intrinsic and germane load. The higher extraneous cognitive load for training via PPPDs than with PPPs aligns with the findings in (Harms, et. al, 2016). Since pedagogically, distractors can challenge the learner to address misconceptions, we recommend further study to track cognitive load as agency increases, as misconceptions not addressed during structured learning could amplify in open-ended environments, resulting in higher cognitive load if measured in sum across a longitudinal span.

7.3 Performance

Though we did not find significant training performance differences across conditions, participants in the PPP and PPPD conditions interacted with the blocks significantly more with a relatively strong effect ($H(2)=21.14$, $p<0.001$, $\varepsilon^2=0.29$; PPP vs. LCF: $p=0.001$; PPPD vs. LCF: $p<0.001$). The fewer block moves made by participants in the LCF condition indicates some may have perceived the task as sufficiently overwhelming to decrease exploratory programming behavior probability.

In addition to analyzing aggregate puzzle performance, we compared individual puzzles. Participants in the PPP and PPPD conditions correctly solved puzzle 3 with a significantly higher score ($H(2)=18.44$, $p<0.001$) and executed more moves ($H(2)=14.772$, $p=0.001$) than those who trained with LCF. Since the solution to puzzle 3 involved 14 blocks, the second-highest count, this result suggests PPPs and PPPDs, which help learners focus on smaller block selection sets, might increase training performance and motivation as the difficulty of the puzzle increases.

Although participants in each condition solved more posttest than pretest questions correctly (PPP:M=0.65, SD=2.03; PPPD: M=0.82, SD=1.76, LCF: M=0.32, SD=2.25), with those in the PPPD condition yielding the highest increase, there is no significant difference on posttest performance across conditions (H(2)=1.335, p=0.513). This lack of transfer performance disparity between PPP and PPPD conditions ostensibly replicates findings in (Harms et al., 2016), and is similar to findings on PPP inter-problem and intra-problem adaptation in (Ericson et al., 2018).

7.4 Efficiency

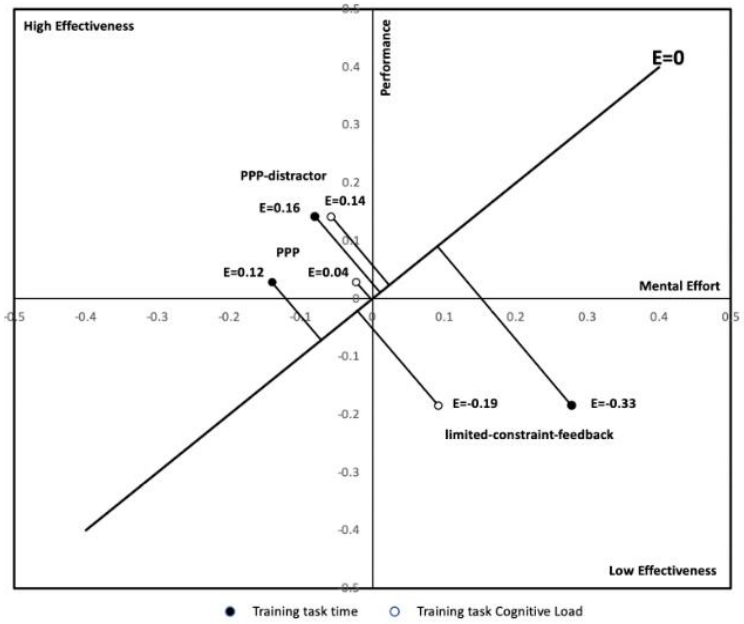

Figure 3. Instructional Efficiency (E) for each of the Three Conditions.

To measure efficiency, we analyzed training and transfer task time across conditions. During training, participants in the LCF condition, despite making fewer block moves, required significantly more time than those in the PPP and PPPD conditions with a moderate effect ($H(2)=6.203$, $p=0.045$, $\varepsilon^2=0.08$; PPP vs. LCF: $p=0.030$; PPPD vs. LCF: $p=0.021$). Since transfer task performance did not vary significantly across conditions, this result suggests training via PPPs and PPPDs enables more efficient CT learning. We did not, however, find a significant difference in the transfer task time ($H(2)=0.883$, $p=0.643$).

To emphasize the opportunity for efficient CT learning, we calculated instructional efficiency, using pre/posttest improvement to measure transfer performance and both time and cognitive load as measurements of mental effort during training (Pass & Merrienboer, 1993). Figure 3 presents areas of high and low effectiveness separated by the effort line E=0. The chart depicts higher instructional efficiency for training with PPPs and PPPDs than with LCF. However, this result refutes **H2**, as the PPPD condition yielded the highest instructional efficiency. This result contrasts with findings in (Harms et al., 2016), which found evidence of decreased learning efficiency from PPPDs, but it aligns with hypotheses regarding distractor learning benefits in (Parsons et al., 2006; Karavirta et al., 2012).

7.5 Motivation

To analyze motivation quantitatively, we scored the TEQ and calculated the within-subject change in programming attitude that occurred between the start and end of the study. Although there was no significant difference in TEQ results across conditions, for the perceived competence subscale, participants who trained with PPPs (M=4.89, SD=1.18) and PPPDs (M=4.91, SD=1.36) scored marginally higher than those who trained with LCF (M=4.53, SD=1.77).

We also found significant positive attitude changes from the start to end of the study. PPP participants' attitude shifted most by: *perceiving programming as more fun* with a small effect size ($H(2)=2.392$, $p=0.017$, $r=0.07$), *more enjoyable* with a medium effect ($H(2)=2.392$, $p=0.017$, $r=0.44$), *easier to start* with a large effect ($H(2)=3.038$, $p=0.002$, $r=0.55$), *less difficult to understand* with a large

effect (H(2)=-3.343, p=0.01, r=-0.6), and *less of a foreign concept* with a large effect (H(2)=-3.074, p=0.002, r=-0.55). PPPD participants' attitudes also shifted positively by: *perceiving programming as easier to start* with a large effect (H(2)=2.514, p=0.012, r=0.54). LCF participants shifted least by: *perceive programming as more enjoyable* with a medium effect (H(2)=2.514, p=0.012, r=0.46). When including only those with little prior programming experience, PPP participants reported *programming more as something they want to learn* with a medium effect (H(2)=1.997, p=0.046, r=0.48) and *less boring* with a medium effect (H(2)=-1.961, p=0.050, r=0.48) in addition to the attitude shifts described above. Although these results indicate attitude improvement and support **H1**, the lack of longitudinal data poses a threat to internal validity, as we cannot claim change at study conclusion persists.

Table 1. *Within-subject Attitude Change. Positive shifts (p), negative shifts (n).* *p<0.05, **p<0.01

Programming is...	PPP	PPP-distractor	limited-constraint-feedback
something I've wanted to learn (p)	M=0.19, SD=1.40	M=0.27, SD=1.31	M=0, SD=1.19
fun (p)	M=0.74, SD=1.67*	M=0.40, SD=1.74	M=0.36, SD=1.43
Enjoyable (p)	M=0.90, SD=1.83*	M=-0.05, SD=1.68	M=0.68, SD=1.76*
important to know (p)	M=0.25, SD=1.48	M=-0.05, SD=1.17	M=0.09, SD=1.19
easy to start (p)	M=1.35, SD=2.29*	M=0.68, SD=1.13*	M=0.45, SD=1.71
something that takes practice (p)	M=0.065, SD=1.09	M=0.05, SD=1.29	M=-0.32, SD=1.17
too difficult to understand (n)	M-1.48=, SD=2.03**	M=-0.77, SD=1.77	M=-0.64, SD=1.89
boring (n)	M=-0.41, SD=1.6	M=-0.32, SD=1.17	M=-0.54, SD=1.90
a foreign concept (n)	M=-1.13, SD=1.83*	M=-0.27, SD=1.55	M=0, SD=2.07
too time consuming (n)	M=-0.35, SD=2.09	M=-0.09, SD=1.27	M=-0.09, SD=2.44

To supplement the quantitative results, we sought qualitative feedback by requesting that participants describe their attitude or view toward programming after the learning experience. For both those who self-reported low and high prior programming experience, we recorded more hesitant responses from those who trained via limited constraint and feedback than those who trained via PPPs and PPPDs. One LCF participant who selected "have tried programming activities, but have not taken a class" in the demographic survey, reflected on sustained struggle: "I still feel like programming is insanely complex. When I was in college I dropped out of computer science as soon as we started python. I just couldn't understand what we were doing, and maybe I could understand it if I really tried. It just seems to be better geared towards certain people." One PPP participant who recorded the same prior programming experience noted: "[T]his activity was somewhat easy but programming is really much harder than this. [B]ut this is a good way for a kid to start learning." One PPPD participant who selected "never attempted to program before" revealed potential for future pursuit of CT: "I would love

to learn more about programming and encourage my son to start learning programming early." These results support **H1** and those in (Charters, Lee, Ko, & Loksa, 2014), which found significant attitude improvement regardless of gender and education level after a brief online programming experience.

8. Conclusion & Future Work

Our survey of grade 6-9 teachers exposed teacher perceptions of limited student engagement with data concepts central to CT. These results led us extend the trend of balancing Scratch's agency with structure to better serve learners and reduce burden on teachers. A small pilot study of an adult population using a learning system that integrates PPPs with Scratch yielded results indicating the structure provided by PPPs catalyzes motivation for CT, reduces extraneous cognitive load, and increases learning efficiency without sacrificing performance on transfer tasks.

While these results reveal opportunities to advance the teaching and learning of CT via augmentations to block-based programming environments, we remain cautious due to external validity limitations: the single CT concept, sequences, and small summative evaluation population (75 adults), threaten generalizability. In future work, we intend to study additional CT concepts, functionality variation, and participants, to identify factors supportive of reliably efficient and effective CT learning.

Acknowledgments

The Programming Systems Laboratory is supported in part by DARPA N6600121C4018, NSF CCF-1815494 and NSF CNS-1563555.

References

Aivaloglou, E., & Hermans, F. (2019). Early programming education and career orientation: the effects of gender, self-efficacy, motivation and stereotype. *ACM SIGCSE*, (pp. 679-685).

Amazon. (2021). Retrieved from Amazon Mechnical Turk: https://www.mturk.com/

Barab, S. (2014). Design-based resarch: a methodological toolkit for engineering change. In K. Sawyer (Ed.), *The Cambridge Handbook of the Learning Sciences.* Cambridge University Press.

Barendsen, E., Mannila, L., Demo, B., Grgurina, N., Izu, C., Mirolo, C., . . . Stupurienė, G. (2015). Concepts in K-9 computer science education. *ACM ITiCSE-WGR*, (pp. 85-116).

Brennan, K. (2013). *Best of both worlds: issues of structure and agency in computational creation.* MIT.

Brennan, K., Balch, C., & Chung, M. (2014). *Creative computing.* Harvard Graduate School of Education.

Bush, J. B., Gilmore, M. R., & Miller, S. B. (2020). Drag and drop programming experiences and equity: analysis of a large scale middle school student motivation survey. *ACM SIGCSE*, (pp. 664-670).

Cao, L., Rorrer, A., Pugalee, D., Maher, M. L., Dorodchi, M., Frye, D., . . . Wiebe, E. (2020). Work in progress report: a STEM ecosystem approach to CS/CT. *ACM SIGCSE*, (pp. 999-1004).

Charters, P., Lee, M., Ko, A., & Loksa, D. (2014). Challenging stereotypes and changing attitudes: the effect of a brief programming encounter on adults' attitudes toward programming. *ACM SIGCSE*, (pp. 653-658).

DeLyser, L. (2018). A community model of CSforALL. *ACM ITiCSE*, (pp. 99-104).

Denny, P., Luxton-Reilly, A., & Simon, B. (2008). Evaluating a new exam question: parsons problems. *ICCE.*

Dong, Y., Marwan, S., Catete, V., Price, T., & Barnes, T. (2019). Defining tinkering behavior in open-ended block-based programming assignments. *SIGCSE*, (pp. 1204-1210).

Emerson, A., Smith, A., Rodriguez, F., W. E., & Bradford, W. (2020). Cluster-based analysis of novice coding misconceptions in block-based programming. *ACM SIGCSE*, (pp. 825-832).

Ericson, B., …Rick, J. (2018). Evaluating the efficiency and effectiveness of adaptive parsons problems. *ICER.*

Ericson, B., … Rick, J. (2017). Solving parsons problems versus fixing and writing code. *Koli Calling CER.*

Frädrich, C., Obermüller, … Fraser, G. (2020). Common bugs in Scratch programs. *ACM ITiCSE*, (pp. 89-95).

Franklin, D., Weintrop, D., Palmer, J., Coenraad, M., Cobian, M., Beck, K., . . . Crenshaw, Z. (2020). Scratch Encore: the design and pilot of a culturally-relevant. *ACM SIGCSE*, (pp. 794-800).

Fritz, C. O., Morris, P. E., & Richler, J. J. (2012). Effect size estimates. *Experimental Psychology, 141*(1), 2-18.

Garner, S. (2007). An exploration of how a technology-facilitated part-complete solution method supports the learning of computer programming. *Informing Science & Information Technology.*

Grgurina, N., Barendsen, E., Zwaneveld, B., van Veen, K., & Stoker, I. (2014). Computational thinking skills in dutch secondary education: exploring pedagogical content knowledge. *ACM Koli Calling*, (pp. 173-174).
Grover, S., & Basu, S. (2017). Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and boolean logic. *ACM SIGCSE*, (pp. 267-272).
Grover, S., & Pea, R. (2013). Computational thinking in K–12: a review of the state of the field. *Edu. Res.*, *42*(1).
Grover, S., Basu, S., & Schank, P. (2018). What we can learn about student learning from open-ended programming projects in middle school computer science. *ACM ITiCSE*, (pp. 999-1004).
Harel, I., & Papert, S. (1991). *Constructionism.* Ablex Publishing.
Harms, K., Balzuweit, E., Chen, J., & Kelleher, C. (2016). Learning programming from tutorials and code puzzles: children's perceptions of value. *IEEE Visual Languages and Human-Centric Computing*, (pp. 59-67).
Harms, K., Chen, J., & Kelleher, C. (2016). Distractors in parsons problems decrease learning efficiency for young novice programmers. *International Computing Education Research*, (pp. 241-250).
Harms, K., Rowlett, N., & Kelleher, C. (2015). Enabling independent learning of programming concepts through programming completion puzzles. *IEEE Visual Lanugages and HCC*, (pp. 271-279).
Helminen, J., … Alaoutinen, S. (2013). How do students solve parsons programming problems? *L. & T. in Comp*.
Helminen, J., … Malmi, L. (2012). How do students solve parsons programming problems? *ICCE*.
Ihantola, P., & Karavirta, V. (2011). Two-dimensional parson's puzzles *IT Education*, *10*, 119-132.
Ihantola, P., Vihavainen, A., Ahadi, A., Butler, M., Börstler, J., Edwards, S., . . . Rubio, M. (2016). Educational data mining and learning analytics in programming: literature review and case studies. *ITiCSE WGR.*
Kaczmarczyk, & Petrick. (2010). Identifying student misconceptions of programming. *SIGCSE*, (pp. 107-111).
Karavirta, V., Helminen, J., & Ihantola, P. (2012). A mobile learning application for parsons problems with automatic feedback. *International Conference on Computing Education Research*, (pp. 11-18).
Karvelas, I., Li, A., & Becker, B. A. (2020). The effects of compilation mechanisms and error message presentation on novice programmer behavior. *ACM SIGCSE*, (pp. 759-765).
Kashmira, D., & Mason, J. (2020). Empowering learning designers through design thinking. *ICCE* (pp. 497-502).
Kong, S., Lai, M., & Siu, C. (2020). Development of CT concepts in Scratch programming. *ICCE*, (pp. 652-657).
Kumar. (2017). The effect of providing motivational support in parsons puzzle tutors. *AI in Ed.*, (pp. 528-531).
Kumar, A. N. (2019a). Helping students solve Parsons puzzles better. *ACM ITiCSE*, (pp. 65-70).
Kumar, A. N. (2019b). Mnemonic variable names in Parsons puzzles. *ACM CompEd*, (pp. 120-126).
Lewis, C., Bruno, P., Raygoza, J., & Wang, J. (2019). Alignment of goals and perceptions of computing predicts students' sense of belonging in computing. *ACM ITiCSE*, (pp. 11-19).
Leyzberg, D., & Moretti, C. (2017). Teaching CS to CS teachers. *ACM SIGCSE*, (pp. 369-374).
Maloney, J., .. Eastmond, E. (2010). The Scratch programming language and environment. *Computing Education*.
Mannila, L., Dagiene, V., Demo, B., Grgurina, N., Mirolo, C., Rolandsson, L., & Settle, A. (2014). Computational thinking in K-9 education. *Innovation & Technology in Computer Science Education.*
Marwan, S., Gao, G., Fisk, S., Price, T., & Barnes, T. (2020). Adaptive immediate feedback can improve novice programming engagement and intention to persist in computer science. *ACM ICER*, (pp. 194-203).
McGill, M. M., & Decker, A. (2020). Tools, languages, and environments used in primary and secondary computing education. *ACM ITiCSE*, (pp. 103-109).
Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2011). Habits of programming in Scratch. *ACM ITiCSE*.
MIT Media Lab. (2021). *Scratch Statisitcs*. Retrieved from https://scratch.mit.edu/statistics/
Morrison, B., Dorn, B., & Guzdial, M. (2014). Measuring cognitive load in introductory CS. *ICER*, (pp. 131-138).
Morrison, B., … Guzdial, M. (2016). Subgoals help students solve Parsons problems. *ACM SIGCSE*, (pp. 42-47).
Parsons, D., & Haden, P. (2006). Parson's programming puzzles: a fun and effective learning tool for first programming courses. *Australian Conference on Computing Education.*
Pass, F., & Merrienboer, J. (1993). The efficiency of instructional conditions. *Human Factors, 35*(4), 737-743.
Prolific. (2021). Retrieved from Prolific | Online participant recruitment: https://www.prolific.co/
Rich, P. J., Browning, S., Perkins, M., Shoop, T. Y., & Belikov, O. M. (2019). Coding in K-8: international trends in teaching elementary/primary computing. *Tech Trends, 63*(3), 311-329.
Salac, J., Thomas, C., Butler, C., Sanchez, A., & Franklin, D. (2020). TIPP&SEE: a learning strategy to guide students through use-modify Scratch activities. *SIGCSE*, (pp. 79-85).
Scaffidi, C., & Chambers, C. (2012). Skill progression demonstrated by users in the Scratch animation environment. *International journal of Human-Computer Interaction, 28*, 383-398.
Schulte, C., & Knobelsdorf, M. (2007). Attitudes towards computer science-computing experiences as a starting point and barrier to computer science. *ACM ICER*, (pp. 27-38).
SDT. (n.d.). Self-determination Theory: https://selfdeterminationtheory.org/intrinsic-motivation-inventory/
Sirkia, T. (2016). Combining parson's problems with program visualization in CS1 context. *Koli Calling*.
Sulaiman, J., Dziena, A. B., & Kaiser, G. (2019). SAGE-RA: a reference architecture to advance the teaching and learning of computational thinking. *Embedding AI in Education Policy and Practice for Southeast Asia.*
Sweller, J. (2010). *Cognitive Load Theory: Recent Theoretical Advances.* Cambridge University Press.
Tahir, F., Mitrovic, A., & Sotardi, V. (2020). Investigating the effects of gamifying SQL-Tutor. *ICCE*.

The Royal Society. (2016). *After the Reboot: Computing Ed. in UK Schools*. Retrieved from https://royalsociety.org/~/media/policy/projects/computing-education/computing-education-report.pdf
Tsur, M., & Rusk., N. (2018). Scratch microworlds: designing project-based introductions to coding. *SIGCSE*.
van Gog, T., & Paas, F. (2008). Instructional efficiency. *Educational Psychologist*, (pp. 16-26).
Whitehouse.gov. (2016). *CS for All* https://obamawhitehouse.archives.gov/blog/2016/01/30/computer-science-all
Wilson, C., Sudol, L., Stephenson, C., & Stehlik, M. (2010). *Running on Empty: The Failure to Teach K-12 CS in the Digital Age.* CSTA. http://www.acm.org/runningonempty/fullreport2.pdf
Wing, J. (2006). Computational Thinking. *Communications of the ACM, 49*(3), 33-35.
Wing, J. (2008). Computational thinking and thinking about computing. *Philosophical Transactions.*
Zhi, R., Chi, M., Barnes, T., & Price, T. W. (2019). Evaluating the effectiveness of parsons problems for block-based programming. *ACM ICER*, (pp. 51-59).