Engaging Students with Instructor Solutions in Online Programming Homework

Thomas W. Price
North Carolina State Univ.
Raleigh, NC, USA
twprice@ncsu.edu

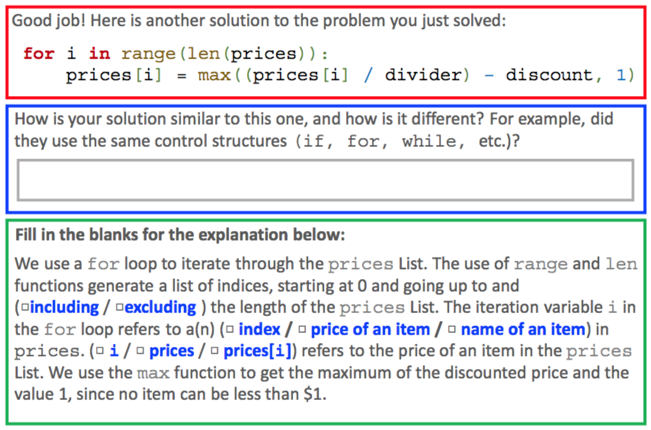
Joseph Jay Williams, Jaemarie Solyst
Univ. of Toronto
Toronto, Canada
williams@cs.toronto.edu, jsolyst@cs.toronto.edu

Samiha Marwan
North Carolina State Univ.
Raleigh, NC, USA
samarwan@ncsu.edu

ABSTRACT
Students working on programming homework do not receive the same level of support as in the classroom, relying primarily on automated feedback from test cases. One low-effort way to provide more support is by prompting students to compare their solution to an instructor's solution, but it is unclear the best way to design such prompts to support learning. We designed and deployed a randomized controlled trial during online programming homework, where we provided students with an instructor's solution, and randomized whether they were prompted to compare their solution to the instructor's, to fill in the blanks for a written explanation of the instructor's solution, to do both, or neither. Our results suggest that these prompts can effectively engage students in reflecting on instructor solutions, although the results point to design trade-offs between the amount of effort that different prompts require from students and instructors, and their relative impact on learning.

Author Keywords
Computing Education; Programming; Self-explanation; Comparison

INTRODUCTION
For university students who are brand new to programming, help in mastering key concepts is essential for developing critical computing skills that can be applied across many disciplines [3, 13]. Learning increasingly takes place during formative programming homework, often through web-based systems, where students spend substantial time practicing writing code, but have little support besides knowing which test cases their code passes [29]. For example, consider a student trying to write a function that takes in a list of prices and applies a discount, while ensuring that no item is sold for less than a dollar. Even if they eventually submit a solution that passes all the test cases, it does not mean they have fully understood key concepts (e.g. control structures like if statements and for loops), since students often use trial-and-error to get to a correct answer [2, 8]. In such cases, how can an instructor foster greater learning once the student has *some* working solution?


Good job! Here is another solution to the problem you just solved:

```
for i in range(len(prices)):
    prices[i] = max((prices[i] / divider) - discount, 1)
```

How is your solution similar to this one, and how is it different? For example, did they use the same control structures (`if, for, while,` etc.)?

Fill in the blanks for the explanation below:

We use a `for` loop to iterate through the `prices` List. The use of `range` and `len` functions generate a list of indices, starting at 0 and going up to and (□**including** / □**excluding**) the length of the `prices` List. The iteration variable `i` in the `for` loop refers to a(n) (□ **index** / □ **price of an item** / □ **name of an item**) in `prices`. (□ **i** / □ **prices** / □ **prices[i]**) refers to the price of an item in the `prices` List. We use the `max` function to get the maximum of the discounted price and the value 1, since no item can be less than $1.

Figure 1. An example instructor's solution (red, top) with Compare (blue, center) and Explain (green, bottom) prompts.

One approach would be to provide the student with an instructor's solution, so the student can compare it to their own solution. However, it is less clear what the best way to support such learning would be. Would it be necessary or worthwhile for the instructor to invest time and effort in designing some support to help students reflect on the solution?

We draw on prior work in psychology and education that could be extended to support students to engage with an instructor's solution. Literature on comparison suggests it can be beneficial to prompt students to articulate what is similar and different between examples, as this can help them clarify underlying principles and when to apply them [10]. Might such benefits extend to comparing a student's solution to that of the instructor? Another approach might be to provide an explanation of the instructor's solution, writing out elaborations of the concepts and terms, and how they function and relate to each other, as in a worked-out example [20]. Such approaches require differential investment by the instructor – explanations must be created separately for each problem, while a prompt to compare can easily be used for many different problems. They also might require differential investment by students – time students spend engaging in comparison or engaging with explanations is time they might instead spend on other beneficial activities, like practicing additional homework questions.

The extensive literature on comparison and explanation in psychology and education tends to focus on theories and laboratory studies [10, 18, 23], or creating entirely new materials [19, 31], such as mathematics worksheets with carefully designed contrasting examples for K12 students to compare in class [22]. In contrast, our current work seeks to incorporate comparison and explanation into existing programming homework, by helping students compare their own solution to that of an instructor on problems *they have already solved*, and we investigate whether this lower-effort intervention can still promote learning. We also test these ideas in university-level programming homework, where students have less support and more discretion in whether to engage with supplemental prompts to compare or explain, requiring us to understand how students perceive the benefits of these activities.

We evaluated two kinds of prompts to reflect on an instructor's solution, with the goal of supporting students to learn more from existing programming practice materials. We deployed a randomized controlled trial during an authentic programming homework assignment in a large, in-person introductory university course. We provided all students with an instructor's solution after they solved a problem, but we independently varied whether or not they received prompts to: (1) *Compare* their solution to an instructor's, (2) *Explain* the instructor's solution by filling in the blanks of a written explanation.

We found that students spent significantly more time with the instructor's solution if given Compare or Explain prompts. This is notable since, in contrast to a laboratory or classroom setting, engagement with the instructor's solution was optional, and our results suggest that prompts may be needed to encourage engagement. We also saw that time spent to Explain by filling in the blanks paid off in improved success in solving related problems. However, the evidence that Compare prompts were beneficial was more suggestive than definitive. Surprisingly, there was no evidence that combining Compare and Explain resulted in a learning benefit beyond no prompts at all, despite students spending substantially more time. Our results suggest that reflection on an instructor's solution *can* lead to learning, even for a problem a student has already solved, but it depends on how such reflection is scaffolded. Our results also provide a cautionary note and emphasize interesting directions for future research to understand the contexts under which multiple such prompts are effective.

This work makes novel contributions by translating laboratory findings on comparison and explanation into the design of prompts for a real-world, independent homework setting, and exploring how multiple prompts interact to impact learning. It highlights the trade-offs instructors can consider in designing such activities: between students' potential learning gains, the additional time students spend, and the instructor time and effort to create and implement additional activities.

RELATED WORK

Comparison of Examples

Several laboratory studies in cognitive science provide evidence that providing students with contrasting solution examples to compare improves their learning and transfer in various contexts [24, 10, 16, 18]. For example, Gentner et al. [10], presented participants with two written negotiation case studies, showing them either one at a time, or at the same time with a prompt to compare. They found that the comparison group performed better on post-test questions and were more capable of transferring skills to a new scenario [10]. Most of this work has focused on short, laboratory studies; however, Rittle-Johnson and Star have extended this work to mathematics classrooms, finding that students who compared multiple solution methods at the same time promoted greater learning than students who evaluated one example at a time [21]. However, this work required extensive curriculum development (e.g. worksheets, teacher training), and the authors noted that teachers had difficulty knowing when to use comparison, and what to compare, leading to low adoption and lack of student improvement [22]. This suggests a need to investigate how to incorporate comparison into existing problems, rather than carefully-constructed examples, and whether such comparison is still beneficial – such as in our current context of asking students' to compare their solution to that of an instructor's.

In the domain of computing education, far less work has been done on comparison. The primary work comes from Patitsas et al. [19], who replicated Rittle-Johnson and Star's results [21] in a CS class on algorithms and data-structures. They used worksheets that showed multiple code solutions to a problem, either one at a time or in parallel. They found that comparing multiple solutions improved students' ability to differentiate important problem features (procedural knowledge), and to consider multiple ways to solve a problem (flexibility) [19]. However, as Patitsas et al. note, they intentionally studied students with prior experience in programming, and it is unclear how these results would generalize to novices, which is the focus of this work. Building on these findings, Villeroy [31, 32] designed a game-based programming environment, "CodeStitch," to encourage students to share, compare and reflect on their solutions and those of their peers, but they did not compare this to other approaches. While this literature suggests the potential learning benefits of comparison in CS, it is still unclear how students would engage with comparison during homework, especially with an instructor's solution.

Scaffolded Self-Explanation Prompts

Another effective technique in engaging students to learn is by prompting them to self-explain or reflect on a worked example [5, 7, 20, 23, 34, 35]. Several studies have shown that novices do not always know how to effectively self-explain when prompted [6, 14]. Researchers have addressed this by *scaffolding* self-explanation, such as by asking student to complete blanks in an explanation [4], which can improve students' performance, particularly when they are at an early stage of learning [7, 15]. However, in programming courses, it is unclear how to design effective scaffolded self-explanation prompts to improve students' learning. For example, Simon et al. found that providing students with multiple-choice self-explanation prompts, instead of open-ended ones, does not help students in understanding the purpose of simple code fragments [25]. However, Vihavainen et al., studied the long-term learning benefits of scaffolded self-explanations, and found that embedding multiple choice questions in the self-explanation prompts benefited students' learning [30]. We build on this work to better understand how scaffolded self-explanation prompts can benefit programming novices.

EXPERIMENT COMPARING REFLECTION PROMPTS

We set out to understand how students engaged with prompts to reflect on an instructor's solution through a randomized controlled experiment during programming homework practice in a real-world classroom.

Design of Prompts. The goal of our prompts was to engage students in reflection after solving a programming practice problem to help them learn, specifically by generalizing principles to solve new, similar problems. We designed two prompts, shown in Figure 1: 1) Compare and Contrast (**Compare**) prompt: An open-ended prompt for the student to compare their code to the instructor's; 2) Scaffolded Self-Explanation (**Explain**) prompt: A step-by-step explanation of the instructor's code, with 3 blanks that the student could fill in using multiple-choice options. We chose comparison and self-explanation prompts because both have been shown to address our goal of helping learners generalize their knowledge to solve new problems [10, 33], at least with carefully constructed examples. The first author created the prompts by reviewing relevant literature on comparison, explanation and computing education, consulting with a co-author who had previously taught the course and run various experiments on comparison and self-explanation, and inviting input from the current instructor. The text of the Compare prompt was chosen based on literature emphasizing the importance of both similarity and difference in helping identify central rather than superficial features [10]. Because open-ended questions can be challenging for novice students [12, 4], the prompt included one specific example of a comparison students could make: the use of "control structures (if, for, while, etc.)."

The Explain prompts were constructed for *each problem* to convey the reasoning behind each line of the solution code. The design of the Explain prompt was motivated by robust findings that worked-out (explained) examples are an effective way to learn [27, 28], especially when the student also engages in self-explanation of the example [1, 23], rather than reading it passively (e.g. in [35]). To encourage this self-explanation process, we included blanks in the explanation, which the student had to fill in, testing their understanding of the solution. This scaffolded approach can be more effective for learning than writing out a whole explanation from scratch [4]. While prior work suggests both prompts can be effective with carefully constructed examples, ours are novel because they are designed to encourage reflection on an existing instructor solution to problems students have *already solved correctly*, imposing less authoring burden on instructors. We hypothesize that our reflection prompts will still be needed and useful, since students can solve problems without fully understanding their own solutions (e.g. through trial-and-error [2, 8]), but reflection can help students identify these underlying principles [33]. Even if the the student's and instructor's solutions are similar, comparison may still be useful, since small differences between examples can be challenging for novices [11] (e.g. different variable names), and self explanation, even of one's own work, is still associated with learning [17].

Population and Procedure: Our study took place in an in-person introductory Computer Science course at a large public university in Canada, consisting of CS-majors and non-majors who typically have little to no programming experience. Students were given an optional review assignment following their midterm exam, which gave them additional practice on two relevant concepts: loops and variables. The practice consisted of program writing tasks, completed in an online practice environment. Each problem presented students with a function stub, which students had to complete based on a brief description, and examples of correct input/output. When students submitted their code, it was checked with a series of test cases, which are reported to the student. The practice included 3 pairs of isomorphic problems (3-8 lines of code): 1) A variant of the rainfall problem [26, 9]; 2) Filtering items out of list; 3) Modifying each item in a list. After solving the first problem in each pair correctly, a window appeared below the problem with an intervention, allowing the student to compare their solution to the instructor's solution. Students had a 50% chance of seeing the Compare or Explain prompts, and these were independent conditions, so students could receive one, both or neither prompt. We also re-randomized students after each problem pair, so they could have seen 3 distinct combinations of prompts. In addition to 0-2 prompts, students were also asked to rate the helpfulness of this intervention to their learning on a scale of 1 (very unhelpful) to 9 (very helpful). Answering the prompts was not required, so we also measured students' completion rate for the intervention. After each intervention, students completed an isomorphic problem, which had a similar solution structure but used a different context, so students had to recognize how and when to apply programming concepts. These problems therefore measured whether prompts addressed our goal of helping students generalize and solve similar, new problems, which can be quite challenging for novices. After the study, students were asked to complete a survey with open-ended prompts asking them to write about their experience with the prompts, such as what they found helpful and unhelpful.

Analysis and Results

We analyzed 233 problem pair attempts from 109 students[1]. We set out to understand how students engaged with the different prompts (how frequently, for how long, and how willingly?), and the prompts' potential costs and learning benefits. When comparing a measure (e.g. helpfulness rating) across conditions, we used a 2-way ANOVA, based on a linear mixed-effects model, investigating the main effects of having Compare and Explain prompts, as well as the interaction effect between them. To account for analyzing multiple attempts

[1]We removed 73 attempts which were done in the wrong order or which lost data from a logging error (uniform across conditions).

Table 1. For each condition, the total number of problem pairs recorded, tests passed (mean and SD), time spent on the intervention, percent of students who completed the intervention, and its rating (1-9).

Prompts	n	Tests Passed	Time (m)	Compl.	Rating
None	61	57.8% (46.0)	0.40 (0.50)	–	6.89 (1.52)
Explain	52	74.5% (40.1)	1.88 (1.42)	59.6%	6.68 (1.42)
Compare	53	70.3% (42.2)	1.19 (0.86)	67.9%	7.06 (1.83)
Both	67	61.5% (44.5)	2.44 (1.38)	62.7%	7.02 (1.49)

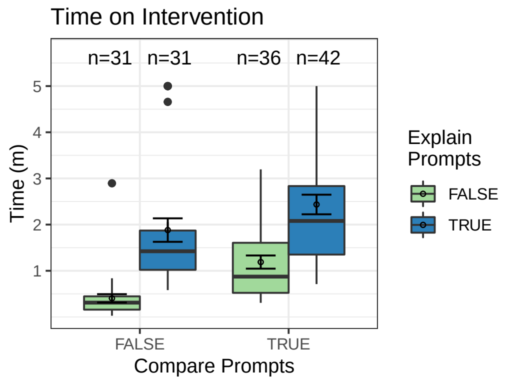


Figure 2. Boxplots of time spent completing the intervention, with mean and standard error bars, and the number of students who did so (n).

from each student, our model included a random effect for the student. Below we summarize our primary findings:

Most students completed the optional prompts and found the intervention helpful for their learning. As shown in Table 1, students completed prompts 60-68% of the time (submitting the open-ended or multiple-choice questions). Those students rated the intervention as helpful for their learning, with an average around 7 out of 9 and less than 3% of responses rating it below 5 (neutral). However, we found no evidence that the prompts impacted the ratings. An ANOVA showed no significant main effect of Compare ($\chi^2(1) = 0.023$; $p = 0.879$) or Explain ($\chi^2(1) = 0.349$; $p = 0.555$) prompts. While there was a significant interaction effect ($\chi^2(1) = 4.32$; $p = 0.038$), post-hoc t-tests showed no significant differences among conditions. A χ^2 test showed no significant relationship between the prompt condition and frequency of completing the intervention ($\chi^2(2) = 1.33$; $p = 0.514$). We also found that students who got the first problem pair right on their first try were as likely to complete the intervention ($n = 78$; 67.9%) as those who did not ($n = 94$; 59.6%): $\chi^2(1) = 0.952$, $p = 0.329$, suggesting that both more and less successful students were willing to engage with prompts.

Students with prompts spent significantly more time[2] *on the intervention.* As shown in Table 1 and Figure 2, students with no prompts spent less than 30s on average looking at solutions, suggesting minimal engagement with the instructor's solution, while those with prompts spent 3-6 times as long. An ANOVA shows that there is a significant main effect of receiving both Compare ($\chi^2(1) = 12.2$; $p < 0.001$) and Explain ($\chi^2(1) = 52.2$; $p < 0.001$) prompts, but no interaction

[2]We capped this time at 5 minutes for analysis (affecting 7% of attempts), since some students took 30+ minutes.

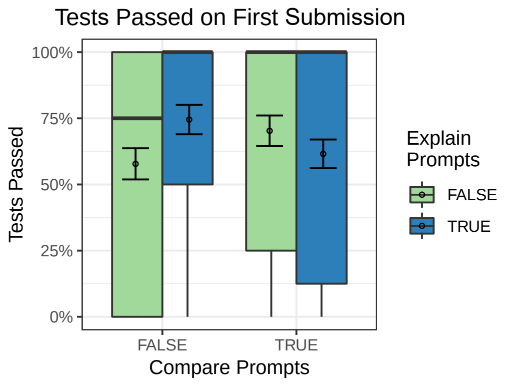


Figure 3. Boxplots of tests passed on students' first submission on the problem pair after the intervention, with mean and standard error bars.

effect ($\chi^2(1) = 0.233$; $p = 0.630$). This suggests that the each prompt independently increases the time spent, but there is not time cost or savings from seeing both together.

Students passed significantly more tests[3] *on subsequent, isomorphic problems when they received only Explain prompts.* As shown in Table 1 and Figure 3, students who received only Compare or Explain prompts performed better than those without, but those with both prompts did no better than those with none. An ANOVA shows a significant interaction effect ($\chi^2(1) = 5.19$; $p = 0.023$), so we therefore performed post hoc pairwise t-tests among each condition. This revealed a significant difference only between those with no prompts and those with only Explain prompts ($t(110.9) = 2.07$; $p = 0.041$; Cohen's $d = 0.386$) This suggests that a single Explain prompt may improve performance, but it is unclear whether additional prompts help, or even hinder performance.

Student's Open-ended Responses

Guided by key questions raised by our quantitative results, we analyzed students' responses to open-ended Compare prompts and survey questions and report prevalent themes below.

What did students write in Compare prompts? Responses were overall quite short (median 10 words), with most students simply stating that there were no differences between their solution and the instructor's besides variable names. For those who noted differences, many pointed to superficial, syntactic program differences, e.g. "I used two loops," rather than high-level differences in approach (e.g. *why* the student used two loops). Many specifically noted their use of `for` or `if` (19.7% of responses included both), which was suggested by the prompt itself. When students did note differences between their solution and the instructor's, most did so objectively "I used for-loops and if-statements, I did not use the max function," but a small number added value judgements, e.g. "I over-complicated the solution" or "this solution is cleaner than mine."

What value did students find in reflecting on an instructor's solution? Students noted that prompts were useful in confirm-

[3]We used the percentage of test cases passed on a student's *first* code submission, since students eventually succeeded 98.7% of the time.

ing the correctness of their solution process: "let me know my thoughts are in the right place." They were also perceived to help solidify or deepen existing knowledge: "makes me revisit the questions and understand the concepts further." This process may also elicit new perspectives on the problem: "They really helped me understand the material to the next level since they started getting me to think about something that I didn't realize and that might be a problem for me." Occasionally, students noted positive feelings that could arise from this process: "Looking at other people's solution inspired me a lot."

How did students choose whether to engage with prompts? When asked when they were likely to respond to prompts, students' responses made clear that they were actively weighing perceived costs and benefits when making this choice. One primary cost was time: "I didn't want to spend more time on the problems than I needed to." This cost was context-dependent, e.g. based on how late the student started the assignment: "I would write when there's enough time. However, if I forgot to do the exercise early and I am in a hurry, I would not write it." The perceived value of prompts was also context-dependent, particularly based on a students' confidence: "If we do know the answer, the prompt is mainly a waste of time."

DISCUSSION

Our goal was to understand how students engage with different prompts to reflect on an instructor's solution. Our results show that overall, students were willing to engage with the optional (ungraded) prompts (60-68% of the time), and rated them as helpful to their learning (7 out of 9). In particular, prompts engaged students with the instructor's solution for much longer (3-6 times as long, on average) than the solution alone. This suggests that comparison and explanation prompts, which can improve learning in *laboratory or classroom* settings [21, 19, 15], are also viable in online programming homework, where students have much more control over which learning content they engage with and answering prompts is optional. Given that students were engaging with solutions to problems they had *already completed*, one could have predicted that students would see this as redundant or a poor use of time. However, our results suggest that when instructors incorporate such prompts, even when they are optional, this can substantially increase the time students spend with the solution.

We also found that this time spent engaging with prompts *can* translate into increased learning for the student, but does not necessarily do so. We saw a significant benefit of Explain prompts. We hypothesize that this was due in part to the prompts helping students extract underlying principles and apply them in the isomorphic problems, as in prior work [33]. There was suggestive but less decisive evidence for the learning benefit of our implementation of Compare prompts. This highlights an important trade-off: students can learn more from prompts, but also spend additional time with such prompts (about 1-2 minutes on average). This time could have been spent on other learning content (e.g. additional problems). However, it is possible that these activities are not mutually exclusive, and prompts may simply increase the total time students are willing to spend on coursework. Students' survey responses show that they often weigh these time costs and learning benefits when considering how to engage with learning content.

Our results showed no evidence for a benefit of *combining* Compare and Explain prompts, as this took additional time but did not result in improved performance beyond no prompts at all. In fact, adding additional prompts may have even hindered performance compared to single prompts (Figure 3). The latter result is surprising, and we interpret it cautiously. We found no evidence that students were less likely to complete the combined prompts, or that they engaged less with them (e.g. wrote less in the Compare prompt), so further work is needed to verify this. Still, these results suggests that in a real-world setting, combining interventions may have unexpected results.

There are a number of possible explanations for why our design of Compare prompts had a less clear impact on learning compared to Explain prompts. Many students noted that their solution was nearly identical to the instructor's, and some noted that this diminished the prompts' usefulness. Prior work on comparison uses carefully constructed, contrasting example pairs [10, 21], and our results show that on real homework problems, there can be fewer contrasting strategies. Our particular choice of Compare prompt, which specifically suggested looking at differences in control structures (e.g. `for`, `if`), may have also led to shallow responses, since many students discussed only syntactic, rather than high-level features.

Another explanation for the difference between the impact of Compare and Explain prompts is that Explain prompts offered students additional, task-specific learning content (the explanation itself), while the Compare prompt was generic across problems. This highlights another potential trade-off between the usefulness of a prompt and the authoring burden on the instructor. Is the learning benefit worth the additional cost of authoring prompts for each existing problem, and maintaining those prompts as problems are updated? Designers and researchers may want to consider this trade-off when recommending interventions to instructors.

LIMITATIONS & FUTURE WORK

Our results, especially with respect to student performance, should be interpreted cautiously, since in this initial study with a smaller population we did not correct for multiple post hoc statistical tests. Additionally, our learning measure (tests passed on the first submission on a later, isomorphic problem) had limitations. A student may not always be trying to fully solve the problem in their first code submission; however, our results suggest this was rare and would have been evenly distributed across conditions. Our study also focused on prompts' impact on students' ability to solve similar problems and does not address farther transfer.

Despite these limitations, our results are still valuable, as they suggest clear hypotheses (Explain prompts promote learning) and open questions (when are Compare prompts useful?; do multiple prompts help?) that motivate future work on prompts to compare to instructors' solutions. This work should explore prompts' effectiveness for multiple programming concepts (beyond loops and lists), and other contextual factors (e.g. timing, difficulty). Future work should also explore the classes of differences between the students' solutions and the instructor's solution, and how these differences impact the prompts' usefulness. Students in each of our conditions had access to the same instructor solution, so we cannot evaluate how useful the solution itself was. Future work could therefore explore how similar prompts to encourage reflection could be used with *different* solutions, such as other students' correct or even *incorrect* solutions. Lastly, our analysis of students' open-ended responses only highlighted high-level themes, and future qualitative work may yield valuable insight into students' perspectives on our specific design decisions.

CONCLUSION

This paper presented a randomized controlled trial evaluating two optional prompts to encourage reflection on an instructor's solution during online programming practice. These prompts were designed based on prior literature from cognitive and learning sciences, as well as instructor experiences and knowledge of the specific domain and task students had to perform, which can help inform future work in the design of reflection prompts in related contexts. Our results offer insight into how students engage with comparison and self-explanation, and the interaction between the two, in the novel context of reflecting on an instructor's solution, rather than carefully designed examples. We found that despite being optional, students are generally willing to engage with prompts, rating them highly and spending considerable time with them. We ground those findings in students' written responses to prompts and survey questions, suggesting why students may see value in and benefit from further engagement with the instructor's solution. Our results also provide insight into how design choices in prompts (e.g. task specificity) can create trade-offs between student learning and student time, as well as instructor authoring burden.

ACKNOWLEDGEMENTS

This work was partially supported by a grant from the Office of Naval Research (#N00014-18-1-2755).

REFERENCES

[1] Robert K Atkinson, Alexander Renkl, and Mary Margaret Merrill. 2003. Transitioning from studying examples to solving problems: Effects of self-explanation prompts and fading worked-out steps. *Journal of educational psychology* 95, 4 (2003), 774.

[2] Tapio Auvinen. 2015. Harmful study habits in online learning environments with automatic assessment. *Proceedings - 2015 International Conference on Learning and Teaching in Computing and Engineering, LaTiCE 2015* (2015), 50–57. DOI: http://dx.doi.org/10.1109/LaTiCE.2015.31

[3] David Barr, John Harrison, and Leslie Conery. 2011. Computational Thinking: A Digital Age Skill for Everyone. *Learning & Leading with Technology* 38, 6 (2011), 20–23. DOI: http://dx.doi.org/10.1590/S1676-06032006000100002

[4] Kirsten Berthold, Tessa HS Eysink, and Alexander Renkl. 2009. Assisting self-explanation prompts are more effective than open prompts when learning with multiple representations. *Instructional Science* 37, 4 (2009), 345–363.

[5] Michelene TH Chi, Miriam Bassok, Matthew W Lewis, Peter Reimann, and Robert Glaser. 1989. Self-explanations: How students study and use examples in learning to solve problems. *Cognitive science* 13, 2 (1989), 145–182.

[6] Michelene TH Chi, Nicholas De Leeuw, Mei-Hung Chiu, and Christian LaVancher. 1994. Eliciting self-explanations improves understanding. *Cognitive science* 18, 3 (1994), 439–477.

[7] Cristina Conati and Kurt Vanlehn. 2000. Toward computer-based support of meta-cognitive skills: A computational framework to coach self-explanation. *International Journal of Artificial Intelligence in Education* 11 (2000), 398âĂŞ 4151.

[8] Stephen H. Edwards. 2004. Using software testing to move students from trial-andor to reflectlon-in-action. *SIGCSE Bulletin (Association for Computing Machinery, Special Interest Group on Computer Science Education)* 36, 1 (2004), 26–30. DOI: http://dx.doi.org/10.1145/1028174.971312

[9] Kathi Fisler. 2014. The recurring rainfall problem. In *Proceedings of the tenth annual conference on International computing education research*. ACM, 35–42.

[10] Dedre Gentner, Jeffrey Loewenstein, and Leigh Thompson. 2003. Learning and transfer: A general role for analogical encoding. *Journal of Educational Psychology* 95, 2 (2003), 393.

[11] Mary L Gick and Keith J Holyoak. 1983. Schema induction and analogical transfer. *Cognitive psychology* 15, 1 (1983), 1–38.

[12] Jimmie Leppink, Nick J Broers, Tjaart Imbos, Cees PM van der Vleuten, and Martijn PF Berger. 2012. Self-explanation in the domain of statistics: an expertise reversal effect. *Higher Education* 63, 6 (2012), 771–785.

[13] Joyce Malyn-Smith and Irene Lee. 2012. Application of the occupational analysis of computational thinking-enabled STEM professionals as a program assessment tool. *J Comput Sci Educ* 3 (2012), 2–10.

[14] Danielle S McNamara. 2001. Reading both high-coherence and low-coherence texts: Effects of text sequence and prior knowledge. *Canadian Journal of Experimental Psychology/Revue canadienne de psychologie expérimentale* 55, 1 (2001), 51.

[15] Danielle S McNamara. 2017. Self-explanation and reading strategy training (SERT) improves low-knowledge students' science course performance. *Discourse Processes* 54, 7 (2017), 479–492.

[16] Laura L Namy and Dedre Gentner. 2002. Making a silk purse out of two sow's ears: Young children's use of comparison in category learning. *Journal of Experimental Psychology: General* 131, 1 (2002), 5.

[17] Yair Neuman, Liat Leibowitz, and Baruch Schwarz. 2000. Patterns of verbal mediation during problem solving: A sequential analysis of self-explanation. *The journal of experimental Education* 68, 3 (2000), 197–213.

[18] Lisa M Oakes and Rebecca J Ribar. 2005. A comparison of infants' categorization in paired and successive presentation familiarization tasks. *Infancy* 7, 1 (2005), 85–98.

[19] Elizabeth Patitsas, Michelle Craig, and Steve Easterbrook. 2013. Comparing and contrasting different algorithms leads to increased student learning. In *Proceedings of the ninth annual international ACM conference on International computing education research*. ACM, 145–152.

[20] Alexander Renkl and Robert K Atkinson. 2002. Learning from examples: Fostering self-explanations in computer-based learning environments. *Interactive learning environments* 10, 2 (2002), 105–119.

[21] Bethany Rittle-Johnson and Jon R Star. 2007. Does comparing solution methods facilitate conceptual and procedural knowledge? An experimental study on learning to solve equations. *Journal of Educational Psychology* 99, 3 (2007), 561.

[22] Bethany Rittle-Johnson, Jon R. Star, and Kelley Durkin. 2017. *The Power of Comparison in Mathematics Instruction: Experimental Evidence From Classrooms*. Number January. Elsevier. 273–295 pages. DOI: http://dx.doi.org/10.1016/b978-0-12-805086-6.00012-6

[23] Marguerite Roy and Michelene TH Chi. 2005. The self-explanation principle in multimedia learning. *The Cambridge handbook of multimedia learning* (2005), 271–286.

[24] Daniel L Schwartz and John D Bransford. 1998. A time for telling. *Cognition and instruction* 16, 4 (1998), 475–5223.

[25] Susan Snowdon Simon. 2011. Explaining program code: giving students the answer helps-but only just. In *Proceedings of the seventh international workshop on Computing education research*. ACM, 93–100.

[26] Elliot Soloway. 1986. Learning to program= learning to construct mechanisms and explanations. *Commun. ACM* 29, 9 (1986), 850–858.

[27] John Sweller. 2006. The worked example effect and human cognition. *Learning and instruction* 16 (2006), 165–169.

[28] J Gregory Trafton and Brian J Reiser. 1993. Studying examples and solving problems: Contributions to skill acquisition. In *Proceedings of the 15th conference of the Cognitive Science Society*. Citeseer, 1017–1022.

[29] Arto Vihavainen, Matti Luukkainen, and Jaakko Kurhila. 2012. Multi-faceted support for MOOC in programming. In *Proceedings of the 13th annual conference on Information technology education*. ACM, 171–176.

[30] Arto Vihavainen, Craig S Miller, and Amber Settle. 2015. Benefits of Self-explanation in Introductory Programming. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. ACM, 284–289.

[31] Marleen Villeroy. 2016. Sharing as a Means for Reflection: Seeing Differences, Understanding Affordances of Peers' Programming Solutions. In *Proceedings of the 11th Workshop in Primary and Secondary Computing Education*. ACM, 88–91.

[32] Marleen Villeroy. 2017. CodeStitch: Leveraging Analogical Encoding in a Game Space. In *Proceedings of the 2017 Conference on Interaction Design and Children*. ACM, 575–581.

[33] Joseph J. Williams and Tania Lombrozo. 2010. The role of explanation in discovery and generalization: Evidence from category learning. *Cognitive Science* 34, 5 (2010), 776–806. DOI: http://dx.doi.org/10.1111/j.1551-6709.2010.01113.x

[34] Joseph Jay Williams, Tania Lombrozo, Anne Hsu, Bernd Huber, and Juho Kim. 2016. Revising learner misconceptions without feedback: Prompting for reflection on anomalies. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, 470–474.

[35] Rui Zhi, Nicholas Lytle, and Thomas W Price. 2018. Exploring Instructional Support Design in an Educational Game for K-12 Computing Education. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. ACM, 747–752.