International Society of the Learning Sciences

# Supporting Third Graders' Use of Subroutines in Programming through Play Versus Worked Examples

Sezai Kocabas, Purdue University, SKocabas@purdue.edu
Laura Bofferding, Purdue University, LBofferd@purdue.edu

**Abstract:** We conducted an in-depth analysis to better understand the role of playing versus analyzing worked examples when learning programming commands. Our findings, focused on two pairs of third graders, demonstrated that students did not use complex programming commands when only playing; whereas, when supported through analysis of worked examples, they did use subroutines. Both pairs started to identify repeating patterns in their code once they had a worked example where a subroutine was used to repeat a set of commands. Yet, having time to play before analyzing worked examples in programming contexts may provide subtle benefits, especially for more abstract commands such as subroutines. Unlike prior studies, our findings suggest that looping concepts can be taught within the concept of subroutines, even with young students.

Tangible programming devices are popular for introducing younger students how to program because students can see the direct results of the commands they program (e.g., Horn & Jacob, 2007). Depending on the type of device, students might program a robot (or other object) to make noises, light up, turn, and move forward or backward (e.g., Sullivan & Bers, 2016); all of these actions are easy to hear or see. However, some elements of programming, such as conditionals, loops, and subroutines, are still abstract in tangible forms because some of the steps or logic are not heard or seen in the same way. For example, Sullivan and Bers (2016) explored pre-kindergarten through second grade students' learning of programming involving KIBO robotic kits combined with a CHERP tangible programming language. At the end of the eight-week robotics curriculum, students struggled more with loop and conditional tasks than sequencing tasks.

There are several ways to help young students make sense of programming commands. One option, based on a constructionist philosophy, is to let students play and explore with the device, providing them with the opportunity to question and discover on their own (Monga et al., 2018). In a study where first and third graders played versus explained their program and goals while playing a tangible programming game, just playing the game supported students' (particularly females') improvement the most (Bofferding et al., 2020). On the other end of the spectrum, students could receive explicit instruction on commands and how to use them and then have targeted practice using the concepts. However, Lee et al. (2013) explored kindergarten students' social interactions through unstructured and structured programming curricula and did not find an impact of instruction on robotic skills and programming concepts. In the middle of the spectrum, students could use worked examples to reason about how a particular command works. In one case, worked-examples supported nine- to ten- year-old students' learning to program. (Joentausta & Hellas, 2018). To better understand such practices, we used an in-depth analysis of two pairs of third graders, unpacking their understanding of subroutines and programs depending on whether they engaged in playing or analyzing worked examples.

## Play versus worked examples

If students have sufficient domain knowledge, play might be at least as good or even better than using worked examples (Tuovinen & Sweller, 1999). Playing that involves no explicit instruction would be helpful to learn programming (McCoy-Parker et al., 2017; Mitamura et al., 2012; Monga et al., 2018) by reducing extraneous cognitive load (Hawlitschek & Joeckel, 2017). However the benefits of play might differ based on the context; playing was more effective on first and fifth graders' creative scores on robot programming but less effective on their technical scores than explicit instruction (McCoy-Parker et al., 2017).

On the other hand, worked examples are guided instruction techniques which provide students with step-by-step problem-solving instructions used to teach problem-solving processes (Atkinson et al., 2000) and can help students focus on understanding, reasoning, and encoding (Ward & Sweller, 1990); they are appropriate to guide students to solutions for difficult problems (Pirolli & Anderson 1985) so that students do not waste time on unhelpful strategies. However, the design of worked examples is important (Sweller et al., 1998). Examples with errors (i.e., incorrect worked examples) support students to use more self-explanations when interpreting the concept, identifying errors, and correcting bugs (Zhi et al., 2018). Zhi and colleagues (2018) found that analyzing incorrect examples can effectively support older students' learning about loops; however, analyzing a mixture of correct and incorrect worked examples can be helpful for students who have sufficient prior knowledge (Große & Renkl, 2007).

## Current study

In this study, we used play and worked examples to help third graders understand and reason about programming commands, with an emphasis on abstract concepts related to subroutines and loops. Although worked examples can help middle school students learn more abstract programming commands (e.g., Zhi et al., 2018) and play can help younger students learn more concrete programming commands (e.g., Mitamura et al., 2012), our aim in this study was to identify the relative benefits of both methods for third grade students and determine when either method is most helpful. Therefore, a further aim was to gain clarity on whether analyzing worked examples was more beneficial during their initial exposure to programming or after given some time to play as we explored two research questions: How do students, depending on their prior experience playing the tangible programming game, interpret the worked examples and the meaning of the subroutine programming block? How do they use information from the worked examples in their pair and their own programming?

## Methods

### Setting, participants, study design, and materials

This study took place at a school in a midwestern district of the United States where approximately 45% of students qualified for free or reduced-price lunch. To elevate the voices of females in early programming, we selected two female pairs of third graders for this in-depth analysis (out of 26 students). We chose the two pairs from the same classroom and who both attempted to use the subroutine block the most in their sessions. Students Sheep7 and Sheep8 played during sessions 1-3 and analyzed worked examples in sessions 4-6. Students Sheep5 and Sheep6 analyzed worked examples in sessions 1-3 and played in sessions 4-6.

To examine the role of worked examples versus play, we employed a repeated-measures, between-groups design (see Figure 1). For this paper, we report on students' descriptions of the subroutine programming block as well as on their programs to move Awbie to a specific spot (on the posttest, they wrote a second version of their program using the subroutine block). Additionally, we report on an item where students explained a program involving grab and walk blocks repeated within a subroutine. Between the pretest and midtest, students worked in pairs to either play the programming game or analyze worked examples plus play, and students switched activity-foci between the midtest and posttest.

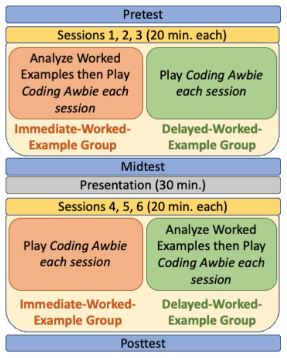
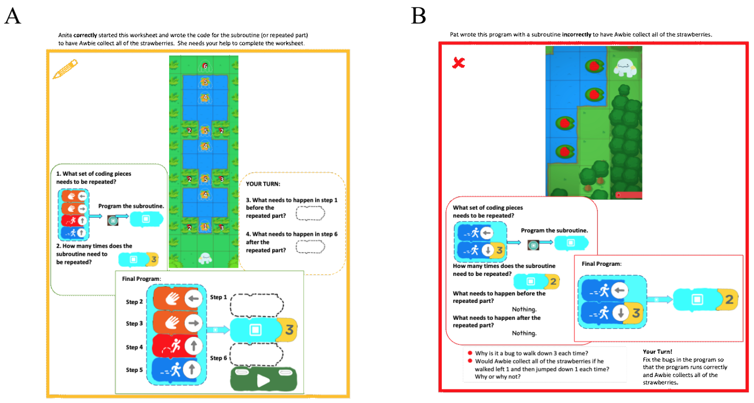

Pretest
Sessions 1, 2, 3 (20 min. each)
Analyze Worked Examples then Play Coding Awbie each session
Play Coding Awbie each session
Immediate-Worked-Example Group
Delayed-Worked-Example Group
Midtest
Presentation (30 min.)
Sessions 4, 5, 6 (20 min. each)
Play Coding Awbie each session
Analyze Worked Examples then Play Coding Awbie each session
Immediate-Worked-Example Group
Delayed-Worked-Example Group
Posttest

Figure 1. Study design.



Anita **correctly** started this worksheet and wrote the code for the subroutine (or repeated part) to have Awbie collect all of the strawberries. She needs your help to complete the worksheet.

1. What set of coding pieces needs to be repeated?

Program the subroutine.

2. How many times does the subroutine need to be repeated?

YOUR TURN:

3. What needs to happen in step 1 before the repeated part?

4. What needs to happen in step 6 after the repeated part?

Final Program:

Step 1
Step 2
Step 3
Step 4
Step 5
Step 6

Pat wrote this program with a subroutine **incorrectly** to have Awbie collect all of the strawberries.

What set of coding pieces needs to be repeated?

Program the subroutine.

How many times does the subroutine need to be repeated?

What needs to happen before the repeated part?

Nothing.

What needs to happen after the repeated part?

Nothing.

Final Program:

- Why is it a bug to walk down 3 each time?
- Would Awbie collect all of the strawberries if he walked left 1 and then jumped down 1 each time? Why or why not?

Your Turn!
Fix the bugs in the program so that the program runs correctly and Awbie collects all of the strawberries.

Figure 2. Example (A) incomplete and (B) incorrect worked examples.

In the worked example sessions, students spent the first 5-8 minutes analyzing and answering questions about worked examples. In session 2 (or 5), students analyzed a correct, worked example to show how to use a subroutine block to repeat a set of grab and walk movements. Then they had to identify missing problem blocks in an incomplete worked example (see Figure 2A), and they found and fixed an error with a subroutine (see Figure 2B). In session 3 (or 6), students analyzed a correct worked example about using a warning block within a subroutine, and they fixed a worked example that incorrectly repeated a jump within a subroutine.

### Analysis

Across subroutine items, we identified shifts in students' explanations (focusing on accuracy and conceptual understanding) and differences between the two pairs (i.e., the subroutine block can be programmed to do a set of commands or the subroutine can be repeated by adding numbers to the subroutine block or by making a subroutine of the subroutine). In order to characterize how pairs used the information from the worked examples, we identified the extent to which the pairs implemented subroutine ideas (and their effectiveness in doing so) within the sessions. Finally, at the individual level, we performed a similar analysis to the sessions using a programming item from the pretest, midtest, and posttest.

## Findings

Overall, students showed a better understanding of subroutines after analyzing worked examples (see Table 1).

Table 1: Students interpretations and use of the subroutine on pretest, midtest, and posttest

| **Group:** | **Pretest** | **Midtest** | **Posttest** |
|---|---|---|---|
| **Intervention** | | **Explaining the Function of the Subroutine** | |
| **-First** | ------- | Repeating (Sheep5, Sheep6) | Repeating (Sheep5, Sheep6) |
| **-Second** | Related to repeating (Sheep7) | ------- | Can be programmed, involves repeating (Sheep7, Sheep8) |
| **Intervention** | | **Programming the Character, Awbie** | |
| **-First** | ------- | Correct (Sheep5, Sheep6) | Correct, recognized repeating pattern but did not program subroutine block (Sheep5, Sheep6) |
| **-Second** | Correct, not efficient (Sheep7); Debug (Sheep8) | Correct (Sheep7, Sheep8) | Correct, programmed subroutine with full program, did not have it repeat (Sheep7, Sheep8) |

During sessions one to three, when analyzing the worked examples, Sheep5 and Sheep6 explained that the subroutine could repeat programmed commands by adding number blocks to the block; however, they did not explain that the subroutine block can be programmed to do a set of commands. They did not find the correct commands to complete the program in Figure 2A; but they were able to explain the problem with the program in Figure 2B. When they tried to fix an incorrect worked example that repeated a jump too many times within a subroutine. They explained the subroutine and fixed the bugs, although they created a new program that did not involve a subroutine. When playing, although the intervention-second group did not recognize repeating patterns, the intervention-first group started to recognize repeating patterns after analysis of the worked examples (which emphasized repeating patterns in the code). However, rather than programming the subroutine block with the part to repeat (*"walk right 1, walk down 1"*) and putting a number on the subroutine button, they placed the blocks to repeat after the subroutine button in the sequence of code. Sometimes they tried to use the subroutine button but forgot to program it first.

During sessions four to six, when analyzing the worked examples, Sheep7 and Sheep8 now explained that the subroutine could repeat programmed commands by adding number blocks to the block, and they mentioned they could repeat the programmed commands by using number blocks on the subroutine. Due to an implementation error, Sheep5 and Sheep6 reanalyzed one set of worked examples and, this time, identified the missing commands in Figure 2A, while Sheep7 and Sheep8 correctly identified the first missing command. They also fixed the subroutine with too many jumps by changing the code and not using the subroutine to do the new repeated part. When playing, both groups recognized repeating patterns and tried to use the subroutine block, but both groups forgot to program the subroutine block; instead, they put it at the end of the part they wanted to repeat. The intervention-second group eventually remembered to program the subroutine block and used it to *repeat jump right 1* three times, and they even put additional code after the block. By session six, they also used the subroutine to repeat by making a subroutine of the subroutine!

## Discussion

Although this analysis focuses on two pairs of students, a limitation of this analysis, the results follow larger trends from our research for those students who used the subroutine in their programming. Using the worked examples was beneficial for helping students understand and use the subroutine block. Sheep5 and Sheep6 were better equipped to try using the subroutine block in their second and third sessions because they engaged with worked examples involving the subroutine block; whereas, Sheep7 and Sheep8 did not try using the subroutine block until they had their first worked example involving the subroutine block. This is particularly interesting because both pairs had been exposed to examples on the pretest and midtest that required them to interpret the action of the subroutine block; however, they only tried to use the block when they had a worked example with one. Although prior research suggested that students would have difficulty learning abstract programming concepts (Sullivan & Bers, 2016; Zhi et al., 2018), our results suggest that students might be less likely to learn these more complex programming commands through play only; they can learn more abstract programming concepts with the support of varied worked examples (Lee et al., 2013).

Based on this study, students who initially played made just as many gains as students who started with worked examples, and arguably, their use of the subroutine button was more complex than the students who started out with worked examples. Therefore, having some time to play before analyzing worked examples in programming contexts may serve some subtle benefits, especially for more abstract commands, an area that should be explored further. Lastly, unlike the previous studies that were teaching loops (Sullivan & Bers, 2016; Zhi et al., 2018), we taught the repeating idea within the concept of the subroutine. Our study showed that students developed an understanding about programming a set of commands and calling them back *while* developing reasoning about repeating a set of commands at the same time.

## References

Atkinson, R. K., Derry, S. J., Renkl, A., & Wortham, D. (2000). Learning from examples: Instructional principles from the worked examples research. *Review of Educational Research*, *70*(2), 181-214.

Bofferding, L., Kocabas, S., Aqazade, M., Chen, L., & Haiduc, A. M. (2020, Apr 17 - 21) Exploring Practices to Support Commenting and Debugging in Early-Years Tangible Programming [Structured Poster Session]. AERA Annual Meeting, http://tinyurl.com/yyd7ayh4 (Conference Canceled)

Große, C. S., & Renkl, A. (2007). Finding and fixing errors in worked examples: Can this foster learning outcomes? *Learning and Instruction, 17*(6), 612-634.

Hawlitschek, A., & Joeckel, S. (2017). Increasing the effectiveness of digital educational games: The effects of a learning instruction on students' learning, motivation and cognitive load. *Computers in Human Behavior, 72*, 79-86.

Horn, M. S., & Jacob, R. J. (2007, April). Tangible programming in the classroom with tern. In *CHI'07 Extended Abstracts on Human Factors in Computing Systems* (pp. 1965-1970).

Joentausta, J., & Hellas, A. (2018, February). Subgoal labeled worked examples in K-3 education. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (pp. 616-621)

Lee, K. T., Sullivan, A., & Bers, M. U. (2013). Collaboration by design: Using robotics to foster social interaction in kindergarten. *Computers in the Schools, 30*(3), 271-281.

McCoy-Parker, K. S., Paull, L. N., Rule, A. C., & Montgomery, S. E. (2017). Challenging elementary learners with programmable robots during free play and direct instruction. *Journal of STEM Arts, Crafts, and Constructions, 2*(2), 100-129.

Mitamura, T., Suzuki, Y., & Oohori, T. (2012, October). Serious games for learning programming languages. In *2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (pp. 1812-1817).

Monga, M., Lodi, M., Malchiodi, D., Morpurgo, A., & Spieler, B. (2018). Learning to program in a constructionist way. In *Proceedings of Constructionism*, Vilnius, Lithuania. https://hal.inria.fr/hal-01913065

Pirolli, P. L., & Anderson, J. R. (1985). The role of learning from examples in the acquisition of recursive programming skills. *Canadian Journal of Psychology/Revue Canadienne de Psychologie, 39*(2), 240-272.

Sullivan, A., & Bers, M. U. (2016). Robotics in the early childhood classroom: Learning outcomes from an 8-week robotics curriculum in pre-kindergarten through second grade. *International Journal of Technology and Design Education, 26*(1), 3-20.

Sweller, J., Van Merrienboer, J. J., & Paas, F. G. (1998). Cognitive architecture and instructional design. *Educational Psychology Review, 10*(3), 251-296.

Tuovinen, J. E., & Sweller, J. (1999). A comparison of cognitive load associated with discovery learning and worked examples. *Journal of Educational Psychology*, *91*(2), 334-341.

Ward, M., & Sweller, J. (1990). Structuring effective worked examples. *Cognition and Instruction, 7*(1), 1-39.

Zhi, R., Lytle, N., & Price, T. W. (2018, February). 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* (pp. 747-752).

## Acknowledgement

This research was supported by an NSF DRL ITEST Grant #1759254.