IDENTIFYING AND FIXING DOUBLE COUNTING ERRORS IN MATHEMATICS AND PROGRAMMING

Sezai Kocabas
Purdue University
Skocabas@purdue.edu

Lizhen Chen
Purdue University
Lizchen@purdue.edu

Laura Bofferding
Purdue University
Lbofferd@purdue.edu

Mahtob Aqazade
Purdue University
Maqazade@purdue.edu

Ana-Maria Haiduc
Purdue University
Ahaiduc@purdue.edu

We investigated how 28 first graders and 27 third graders, who analyzed worked examples as part of a programming intervention, debugged (identified and fixed bugs) and reasoned about double-counting errors in mathematics and programming tasks. Students completed the tasks on a pretest, a midtest (only programming tasks), and a posttest. Results showed that identifying double-counting errors positively correlated with fixing those errors in both programming and mathematics tasks and that students made more gains if they had analyzed worked examples during their programming, game-playing sessions prior to the test. The results suggest the importance of two-dimensional coordination in programming and mathematics debugging.

Keywords: Computational Thinking, Computing and Coding, Number Concepts and Operations, Elementary School Education

Computational Thinking (CT) includes cognitive skills, e.g., abstraction, problem-solving, or debugging (e.g., Wing, 2006, 2011), which align to key computer science standards (NGSS Lead States, 2013) and mathematical practices (National Governors Association Center for Best Practices & Council of Chief State School Officers, 2010). Prior research in elementary education has shown a correlation between programing and mathematics scores (Grover et al., 2016; Lewis & Shah, 2012) and indicated that learning programming helped students extend mathematics content knowledge and develop problem-solving skills (Ahmed et al., 2011; Fessakis et al., 2013; Friend et al., 2018). At the same time, elementary students encountered difficulties in counting while debugging a program (Bofferding et al., 2020; Kocabas et al., 2019). We further explore the relation between debugging in programming and mathematics for early elementary students by focusing on this fundamental skill: counting.

Debugging and Counting "Bugs"

Debugging is difficult for students who have little programming experience (Fitzgerald et al., 2008; Murphy et al., 2008). Studies have reported that fixing errors in a program is harder than identifying them (Fitzgerald et al., 2008; Katz & Anderson, 1987; Lewis, 2012) and that fixing an error becomes easier if the error has already been identified, when students pay attention to relevant features (e.g., Lewis, 2012). On the other hand, having no or little programming experience might lead students to introduce new errors while trying to identify the existing error in a program (Gugerty & Olson, 1986; Nanja & Cook, 1987). Therefore, they are more likely to do extra, unneeded, modifications in a program (Ahmadzadeh et al., 2005; Nanja & Cook 1987).

Double counting, counting the same object or space twice, is a common difficulty for young students in programming (e.g., Kocabas et al., 2019) and mathematics (e.g., Fuson, 2012). Fuson

Olanoff, D., Johnson, K., & Spitzer, S. (2021). *Proceedings of the forty-third annual meeting of the North American Chapter of the International Group for the Psychology of Mathematics Education*. Philadelphia, PA.

(2012) found that three- to five-year-olds made more double-counting errors when objects were disorganized than when they were displayed ordinally. Kocabas et al. (2019) reported that first and third graders double counted the spaces on a programming path where it switched directions. Similarly, Battista and colleagues (Battista, 1999, 2010; Battista et al., 1998) reported that second graders without row and column structures may double count where rows and columns overlap. When counting down, as for solving 14 – 6, children say 13 while putting up one finger to indicate that one less than 14 is 13 and gradually say "12, 11, 10, 9, 8" while sequentially raising five more fingers (Maclellan, 1995; Wright et al., 2006); however, some students may count the 14 as one taken away. Encouraging students to debug could help draw their attention to such counting errors. We combine a focus on debugging and counting in this study through the following research questions: (1) How do first and third graders make sense of double counting errors in programming versus mathematics debugging tasks? (a) To what extent does success in debugging double-counting errors correlate between programming and mathematics tasks? (b) How does students' success in debugging change after counting to make programs in a coding game? Does analyzing worked examples earlier versus later affect the changes? (c) What are possible explanations for students' debugging reasoning (different or similar) in programming and mathematics?

Methods and Analysis

For this study, we analyze data from 28 first graders and 27 third graders from a public elementary school in the Midwest. The students completed a pretest, three 20-minute sessions playing *Osmo™ Coding Awbie* in pairs, a midtest, participated in a 30-minute presentation on programming applications, three additional 20-minute sessions of game play, and a posttest. Before the sessions, students were randomly assigned to either the immediate-worked-examples (immediate) group or the delayed-worked-examples (delayed) group. During the first three sessions, students in the immediate group analyzed a set of programming worked examples (<10 minutes) and then played the game without interruption (>10 minutes), while students in the delayed group just played the game. After the midtest, during the second three sessions, the immediate and delayed groups switched their activities.

In this paper, we focused on one programming debugging item (see Figure 1, left panel: bug 1) included on the pretest, midtest, and posttest and one mathematics debugging item (see Figure 1, right panel) included on the pretest and posttest. In both cases, students watched a video of the counting bug occurring and were asked to find and fix the bug. We interviewed the students individually. We ran correlational analyses to determine if there was an association between students' identifying and fixing (debugging) programming and mathematics counting errors.

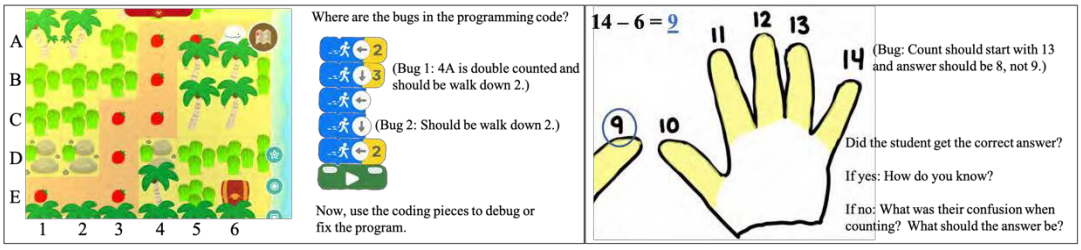

Figure 1: Program Debugging (left panel) and Mathematics Debugging (right panel) Items

Further, we used McNemar tests and Mann-Whitney U tests to determine whether there were significant differences between immediate and delayed groups from the pretest to posttest. For qualitative analysis, we then grouped students based on whether they (1) did not identify or fix, (2) identified but did not fix, (3) did not identify but did fix, or (4) identified and fixed the errors. To provide a clear picture of students' debugging performance, we identified qualitative descriptions of students' reasoning based on their interpretation of the given visuals and their use of additional strategies (e.g., creating their own code rather than modifying given code or counting on their own fingers to determine the answer). Within each group, across the mathematics and programming items, we looked for commonalities in their reasoning.

Findings

Students' identifying and fixing the mathematics bug were significantly correlated within the pretest (immediate group: r=.750, p<.001; delayed group: r=.727, p<.001) and within the posttest (immediate group: r=.806, p<.001; delayed group: r=.802, p<.001). Similarly, identifying and fixing the programming bug were significantly correlated within the pretest (immediate group: r=.650, p=.001; delayed group: r=.606, p=.002), midtest (immediate group: r=.793, p<.001; delayed group: r=.512, p=.009), and posttest (immediate group: r=.651, p=.001; delayed group: r=.592, p=.004). The only other significant correlation for the delayed group was fixing the pretest math counting bug with fixing the midtest programming bug (r=.421, p=.029). On the other hand, the immediate group had a significant correlation with identifying the pretest programming bug with identifying the posttest math bug (r=.430, p=.022) and identifying the math and programming bugs on the posttest (r=.426, p=.027).

Overall, based on a McNemar test of change, students in the immediate and delayed groups made significant gains in fixing the programming bug from pretest to posttest (χ^2= 5.06, p=.021 and χ^2= 8.64, p=.002 respectively), but did not make significant gains in fixing the mathematics bug (χ^2= 1.13, p=.289 and χ^2=.13, p=.727, respectively) (see Table 1). Further, based on Mann-Whitney U tests, the gains in fixing bugs between the two groups from pretest to posttest did not differ significantly on the programming debugging item, U=320.00, z=-.60, p=.550, or on the mathematics debugging item, U=416.00, z=1.46, p=.146. However, based on a McNemar test of change, students in the immediate group made significant gains in fixing the programming bug from pretest to midtest, χ^2 = 6.13, p=.008, unlike the delayed group, χ^2= 2.50, p=.109. Neither group made significant gains from midtest to posttest.

Table 1: Percent of Students Who Identified and Fixed the Math and Programming Bugs

	Mathematics				Programming					
	Identify		Fix		Identify			Fix		
Group	**Pre**	**Post**	**Pre**	**Post**	**Pre**	**Mid**	**Post**	**Pre**	**Mid**	**Post**
Immediate [a]	32%	54%	29%	43%	29%	42% [b]	54%	29%	54% [b]	64%
Delayed [c]	30%	28%	26% [d]	20% [d]	19%	22%	56% [d]	15%	37%	60% [d]

[a] n=28. [b] n=26 because two students missed the midtest. [c] n=27. [d] n=25 because two students moved before the posttest.

Programming and Mathematics Debugging Reasoning

For the program debugging item, students were more likely to fix the bug once they identified it. Interestingly, students who did not identify the bug sometimes inadvertently fixed

the bug when rewriting the code. Yet, when rewriting the code, some of these students created new double-counting or directional bugs. For instance, two first graders succeeded in correcting "walk down 3" to "walk down 2" but changed the first correct code "walk left 2" to "walk left *3*" or "walk *right* 2" (see Figure 1). A few students identified the bug but did not know how to fix it. Students who did not identify or fix the bug often double counted the space A4; they had difficulty structuring their counting, i.e., separating the horizontal code (walk left) from the vertical code (walk down), and identifying that space A4 was counted in the first line of code.

For the mathematics debugging item, although approximately 25% more of the immediate group identified and fixed the bug on the posttest compared to pretest, students in the delayed group did not show similar improvement. Similar to programming, once students identified the bug, most of them fixed it. Students who did not identify the math bug but still fixed it either knew the answer should be eight or correctly took six fingers away to get eight; however, they did not have a problem with the picture showing the count starting at 14. Students who neither identified nor fixed the error agreed that "six fingers are taken away." However, they did not realize that the first count incorrectly started with 14 instead of 13. On the other hand, about 10% of students did identify the bug but failed to fix it. These students indicated that the answer of nine was not correct (often by counting on their own fingers to check), but when they counted the fingers on the picture, they ended up agreeing with the counting strategy and did not fix it. The group of students who succeeded in identifying and fixing the bug often reasoned that "fourteen doesn't count" and avoided double counting.

Discussions and Implications

Our study confirmed previous findings that once identifying bugs, students could fix them in programming (e.g., Fitzgerald et al., 2008) and mathematics contexts. For the delayed group, there was a correlation between fixing the pretest mathematics bug and the midtest programming bug, building on similar correlational findings by Lewis and Shah (2012). On the other hand, these items were not correlated for the immediate group, possibly because thinking critically about the worked examples from the beginning helped students even if they had not fixed the mathematics bug on the pretest. Moreover, identifying the bugs in the mathematics and programming items on the posttest were correlated for this group, once again suggesting some relation between mathematics and programming. Future studies could balance programming and mathematics debugging experiences to further investigate how they relate.

Overall, we found that students struggled with coordinating horizontal movements with vertical movements (lines 1 and 2 of the programming code in Figure 1) and with aligning pictorial representations with their own finger counting. In both situations, students showed a lack of global structuring of the information (Battista et al., 1998; Battista & Clements, 1996). In the programming item, the grid organization may not have alleviated students' inclination to double count because they were asked to track the position and its result in their heads and may not have considered overlapping spaces. Likewise, for the mathematics item, they saw a static representation of the finger counting (with highlighting to show action), so they may have had difficulty tracking what the count corresponded to in relation to the picture. The fact that even some third graders, who had initially indicated the answer should be eight, ended up agreeing with the answer of nine because the picture looked right, highlights the need to help students analyze and reason about visuals (as was done with the programming worked examples).

Acknowledgement

This research was supported by the National Science Foundation grant #1759254.

References

Ahmadzadeh, M., Elliman, D., & Higgins, C. (2005). An analysis of patterns of debugging among novice computer science students. *ACM SIGCSE Bulletin, 37*(3), 84-88.

Ahmed, G., Nouri, J., Zhang, L., & Norén, E. (2020). Didactic Methods of Integrating Programming in Mathematics in Primary School: Findings from a Swedish National Project. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (pp. 261-267).

Battista, M. T. (1999). Fifth graders' enumeration of cubes in 3D arrays: Conceptual progress in an inquiry-based classroom. *Journal for Research in Mathematics Education*, *30*(4), 417-448.

Battista, M. T. (2010). Thoughts on elementary students' reasoning about 3-D arrays of cubes and 306anderbilt. In Z. Usiskin, K. Andersen, & N. Zotto (Eds.), *Future curricular trends in school algebra and geometry: Proceedings of a conference* (pp. 183-199). IAP.

Battista, M. T., & Clements, D. H. (1996). Students' understanding of three-dimensional rectangular arrays of cubes. *Journal for Research in Mathematics Education*, *27*(3), 258-292.

Battista, M. T., Clements, D. H., Arnoff, J., Battista, K., & Borrow, C. V. A. (1998). Students' spatial structuring of 2D arrays of squares. *Journal for Research in Mathematics Education, 29*(5), 503-532.

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 San Francisco, CA http://tinyurl.com/yyd7ayh4 (Conference Canceled)

Fessakis, G., Gouli, E., & Mavroudi, E. (2013). Problem solving by 5–6 years old kindergarten children in a computer programming environment: A case study. *Computers & Education*, *63*, 87-97.

Fitzgerald, S., Lewandowski, G., McCauley, R., Murphy, L., Simon, B., Thomas, L., & Zander, C. (2008). Debugging: finding, fixing and failing a multi-institutional study of novice debuggers. *Computer Science Education*, *18*(2), 93-116.

Friend, M., Matthews, M., Winter, V., Love, B., Moisset, D., & Goodwin, I. (2018, February). Bricklayer: Elementary Students Learn Math through Programming and Art. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (pp. 628-633).

Fuson, K. C. (2012). *Children's counting and concepts of number*. Springer Science & Business Media.

Gugerty, L., & Olson, G. (1986). Debugging by skilled and novice programmers. *ACM SIGCHI Bulletin, 17*(4), 171-174.

Katz, I. R., & Anderson, J. R. (1987). Debugging: An analysis of bug-location strategies. *Human-Computer Interaction*, *3*(4), 351-399.

Kocabas, S., Bofferding, L., Aqazade, M., Haiduc, A., & Chen, L. (2019). Students' directional language and counting on a grid. In S. Otten, A. G. Candela, Z. de Araujo, C. Haines, & C. Munter (Eds.), *Proceedings of the 41st annual meeting of the north American chapter of the international group for the psychology of mathematics education* (pp. 395-399).

Lewis, C. M. (2012). The importance of students' attention to program state: A case study of debugging behavior. In A. Clear, K. Sanders, & B. Simon (Eds.), *Proceedings of the ninth annual international conference on international computing education research* (pp. 127-134). Association for Computing Machinery

Lewis, C. M., & Shah, N. (2012). Building upon and enriching grade four mathematics standards with programming curriculum. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (pp. 57-62).

Maclellan, E. (1995). Counting all, counting on, counting up, counting down: The role of counting in learning to add and subtract. *Education 3-13, 23*(3), 17-21.

Murphy, L., Lewandowski, G., Mccauley, R., Simon, B., Thomas, L., & Zander, C. (2008). Debugging: The good, the bad, and the quirky – a qualitative analysis of novices' strategies. *ACM SIGCSE Bulletin, 40*(1), 163-167.

Nanja, M., & Cook, C.R. (1987). An analysis of the on-line debugging process. In G. Olson, S. Sheppard & E. Soloway (Eds.), Empirical studies of programmers: Second workshop (pp. 172–184). Ablex.

National Governors Association Center for Best Practices & Council of Chief State School Officers (2010). *Common Core state standards mathematics*. Washington, DC: National Governors Association Center for Best Practices & Council of Chief State School Officers.

NGSS Lead States. (2013). *Next generation science standards: For states, by states*. The National Academy Press.

Wing, J. M. (2011, March 06). *Research notebook: Computational thinking—What and why.* The Link. https://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33-35.

Wright, R. J., Martland, J., & Stafford, A. K. (2006). Early numeracy: Assessment for teaching and intervention. *School Library Journal, 52*(10), 84.