Supporting Collaborative Debugging Processes

Afaf Baabdullah, The Pennsylvania State University, afb5304@psu.edu
ChanMin Kim, The Pennsylvania State University, cmk604@psu.edu

Abstract: This qualitative case study reports how novice pairs debugged errors in block code while interacting with a CSCL tool called Debug Space. Findings suggest that utilizing visual cognitive resources integrated within Debug Space supported initiating conceptual models of desired and actual code and consolidating collective understanding of debugging tasks.

Introduction

Code comprehension is a key process in debugging. Novices tend to struggle with understanding code structure and its chunks (Li et al., 2019) and debug in an arbitrary manner. Even when successful, debugging often does not lead to understanding of how the bug was fixed (Kim et al., 2018). While pair debugging is an approach to alleviate novices' debugging difficulties (Chintakovid et al., 2006), it is not effective without code comprehension of both debuggers. Our inquiry in the present study was on pair debugging processes that especially involve code comprehension. We used a CSCL tool called Debug Space to facilitate productive collaboration between pair debuggers and examined their discourses and actions during debugging to answer the following research question: How does a novice pair solve a given debugging task while interacting with Debug Space?

Theoretical framework

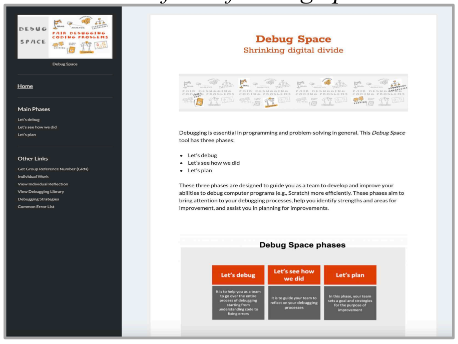
Gilmore's debugging model (1991) posits that debugging is a product of the processes of flexible, incomplete comprehension, mental representations of desired and actual code, and mismatch detection of bugs. The model explains how debuggers (re)construct conceptual models of desired code and compare them with actual code performance in term of *what the code does* and *how the code works*. *What the code does* can be understood by running the code and observing the output. *How the code works* requires understanding the relationship between/among multiple blocks within the code.

Methods

Participants and context

Thirty two undergraduates enrolled in an elective general education course at a large public university in the northeastern United States participated in the study. The programming unit was for 50 minutes per week for four weeks. For the first two weeks, participants learned to code using Scratch. For the other two weeks, they were invited to fix Scratch buggy code using a CSCL tool called Debug Space. One pair case is reported in this poster.

Figure 1
The Main Interface of Debug Space



Debug Space

Home

Main Phases
Let's debug
Let's see how we did
Let's plan

Other Links
Get Group Reference Number (GRN)
Individual Work
View Individual Reflection
View Debugging Library
Debugging Strategies
Common Error List

Debug Space
Shrinking digital divide

Debugging is essential in programming and problem-solving in general. This *Debug Space* tool has three phases:

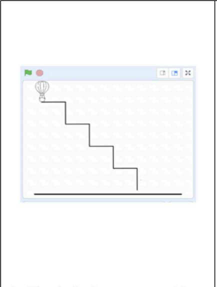
- Let's debug
- Let's see how we did
- Let's plan

These three phases are designed to guide you as a team to develop and improve your abilities to debug computer programs (e.g., Scratch) more efficiently. These phases aim to bring attention to your debugging processes, help you identify strengths and areas for improvement, and assist you in planning for improvements.

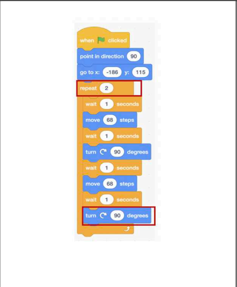
Debug Space phases

Let's debug	Let's see how we did	Let's plan
It is to help you as a team to go over the entire process of debugging starting from understanding code to fixing errors	It is to guide your team to reflect on your debugging processes	In this phase, your team sets a goal and strategies for the purpose of improvement

Figure 2
The Debugging Task Supported with Visual Resources

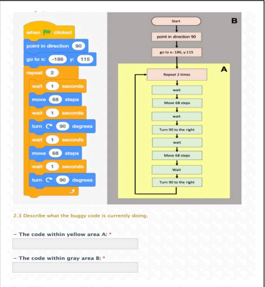


a) The desired movement video shows the sprite moving from up to down stairs



```
when [flag] clicked
point in direction 90
go to x: -186 y: 115
repeat 2
    wait 1 seconds
    move 68 steps
    wait 1 seconds
    turn ↻ 90 degrees
    wait 1 seconds
    move 68 steps
    wait 1 seconds
    turn ↻ 90 degrees
```

b) The buggy code has two seeded bugs: repetition value and the second turning block



2.3 Describe what the buggy code is currently doing.

– The code within yellow area A: *

– The code within gray area B: *

c) The graphical representations of the buggy code combined with question prompts

Debug Space

Debug Space was designed to support pair debugging among novice learners through their reflection on the processes of debugging and collaboration. It consists of three main phases called, *Let's debug*, *Let's see how we did*, and *Let's plan* (see Figure 1). Pairs were prompted to articulate their understanding of program requirements, code, discrepancies, and bugs. Visual resources were also provided to facilitate their discussions during collaborative debugging. For example, during their code review process, multiple graphics were provided to encourage them to construct mental representations of the code at various levels (Figure 2c). Borge et al.'s (2018) framework for group regulation was applied in this phase for productive collaboration. To build collective, shared understanding, pairs studied buggy code and responded to prompts individually first (Figure 2c), and then discussed code structure and responded to prompts together.

Data analysis

Pair debugging video recordings and interviews were transcribed. We identified collaborative debugging episodes, and analyzed structures of conversations in each episode. That is, we coded new ideas in conversations and actions and then overarching ideas across conversational turns. We then applied Kim et al.'s (2018) coding scheme to also identify debugging actions and discourse within each overarching idea. We had a series of collaborative coding sessions until they reached consensus.

Preliminary findings

Mason and Allison utilized the visual resources integrated within Debug Space to consolidate their collective understanding of the code. At the outset, the pair examined code requirements (Figure 2a). During their examination, they analyzed the sprite's movement and identified the subgoals of the program. The pair then examined the visual resources while responding to the prompts (Figure 2c). While examining the diverse representations of the code through the visual resources, the pair discussed specific blocks in connection with the overall programming goal. They also initiated building conceptual models of *what each chunk does* and *how it works*. The role of graphical representations in Debug Space seemed critical in that the control flow of the code guided not only the individual code review but also collaborative conversations and actions. During another debugging session, Allison proposed examining graphical representations as follows, "Maybe we should just go look at it [the flowchart], not here 'cause this [the code] is more confusing". Her interview comments in the following explains her use of graphical representations further: "I think Debug Space helps more because it makes … you to plan out and really watch the specifics of the code." The pair used their constructed conceptual models for discrepancies between the actual code and the desired code in detect a bug. Constrained by one hypothesis, Mason identified a potential bug and justified his identification by explaining the causal effect on the actual output while demonstrating the reasoning process on the provided visual resources. It is important to note that at this point, the pair only predicted the performance of the buggy code without running the code. That is, the pair studied *what the code did* and *how the code worked* through responding to the prompts on Debug Space without seeing the actual output of the buggy code. In doing so, the pair worked through their collaborative explanations from descriptive, to mapping, hypothetical, and analytical levels, which advanced their collective conceptual models of the buggy code.

Discussion and implications

Our findings illustrate how pair's collaborative debugging process was facilitated via engaging with a CSCL tool. The pair co-constructed shared conceptual representations of desired and actual states of the code. Bringing the two representations into the what-code-does and how-it-works knowledge enabled the pair to conduct comparisons and detect discrepancies. Educators can plan instruction to guide productive debugging process of novice pairs via utilizing various visual representations of different states of the code. More research is needed to further scaffold collaborative debugging.

References

Borge, M., Ong, Y. S., & Rosé, C. P. (2018). Learning to monitor and regulate collective thinking processes. *International Journal of Computer-Supported Collaborative Learning*, (13), 61–92. https://doi.org/10.1007/s11412-018-9270-5

Chintakovid, T., Wiedenbeck, S., Burnett, M., & Grigoreanu, V. (2006). Pair collaboration in end-user debugging. In *Proceedings - IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2006* (pp. 3–10). IEEE. https://doi.org/10.1109/VLHCC.2006.36

Gilmore, D. J. (1991). Models of debugging. *Acta Psychologica*, *78*(1–3), 151–172. https://doi.org/10.1016/0001-6918(91)90009-O

Kim, C., Yuan, J., Vasconcelos, L., Shin, M., & Hill, R. B. (2018). Debugging during block-based programming. *Instructional Science*, *46*(5), 767–787. https://doi.org/10.1007/s11251-018-9453-5

Li, C., Chan, E., Denny, P., Luxton-Reilly, A., & Tempero, E. (2019). Towards a framework for teaching debugging. In *Proceedings of the Twenty-First Australasian Computing Education Conference on - ACE'19* (pp. 79–86). Sydney, Australia: ACM. https://doi.org/10.1145/3286960.3286970