Accepted Manuscript, to appear in:
Koli Calling '25: Proceedings of the 25th Koli Calling International Conference on Computing Education Research

Analyzing Process Data of Students' Debugging Processes

Elena Spörer
elena.spoerer@tum.de
Technical University of Munich
Computing Education Research Group
Munich, Germany

Abstract

Debugging is an unavoidable challenge for novice programmers and often causes frustration. Even if a systematic debugging process helps students to be successful, debugging still remains challenging for them. Aiming at supporting students and reducing frustration, this doctoral research project investigates debugging processes of students using multi-modal process data. By triangulating this process data (e.g., screen recordings, eyetracking, log data), the project aims at making steps towards an automated analysis of debugging processes, facilitating large-scale process analysis. Eventually, this approach seeks to provide learners with personalized feedback and to advance the understanding of how students engage in debugging.

CCS Concepts

• **Social and professional topics** → **Computing education**; *Computer science education.*

Keywords

debugging, process analysis, theory-based, computing education

ACM Reference Format:
Elena Spörer. 2018. Analyzing Process Data of Students' Debugging Processes. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX).* ACM, New York, NY, USA, 2 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

In almost every computing education curriculum worldwide, learning to program is an important part [2]. Although several studies on teaching and learning programming have been conducted so far, students still struggle with learning programming [9]. As they need to deal with several new concepts, such as variables and conditionals, it is not surprising that students' code frequently contains bugs that need to be fixed. Consequently, debugging, the process of finding and fixing bugs, is an unavoidable challenge for novices when learning to program. Moreover, debugging is difficult for most students, regardless of their level of programming skills, because debugging skills differ from programming skills [3] and need to be taught explicitly [6] what only happens rarely [10]. As a result, students often end up being frustrated [12]. Teaching a systematic debugging process has proven to be beneficial for students [10].

However, to investigate how learners actually engage in debugging, analyzing debugging processes rather than only looking at performance outcomes is essential. This perspective highlights not only where learners encounter challenges but also how interventions or feedback can be tailored to support students in targeting difficulties more effectively.

2 Related Work

Debugging stems from the necessity of finding and fixing bugs, as most programs in educational and professional settings do not work as intended right from the beginning. To describe an ideal-typical way of debugging, research has proposed several debugging process models since the 1970s. Those models represent an ideal and successful sequence of debugging process steps and were developed either theoretically or empirically, e.g., [1, 4, 13]. Although the process models were developed over a long period of time using different methodologies, they can be summarized into the same four steps: (1) observe the failure, (2) set up a hypothesis, (3) verify the hypothesis, and (4) fix the bug and verify the solution. When debugging code from other developers, an additional step of code comprehension to familiarize with the program is necessary [5]. This step is often carried out in the beginning, but can also be included at other points in time of the debugging process. However, only some process models address this additional step, e.g., [4].

Focusing on educational contexts, several previous studies analyzed the debugging processes of students. Examining those studies in detail reveals that the data type used to capture the debugging processes has a great impact on the outcome. Capturing snapshots of a program at certain points in time is an easy way to collect data for analyzing students' debugging processes. However, snapshots lack all the actions that happen between the collection points. In contrast, screen recordings and log data are examples of data types that capture everything the students are doing when debugging. In light of this, it is surprising that the majority of the previous studies either used screen recordings (e.g., [7, 11]) or snapshots (e.g., [5]) as a basis for analyzing students' debugging processes. Studies analyzing log data are rare. For evaluating the data, most of the studies used inductive analysis approaches to investigate the debugging strategies the students applied (e.g., [3, 5]). In addition, there is a focus on assessing students' debugging performance (e.g., [3, 7]). In contrast, only a few studies evaluated screen-recorded debugging processes using deductive analysis methods, with debugging process steps from previous work as a foundation [8, 11]. As a result, the current body of research lacks theory-based analysis approaches, investigating students' debugging processes using an ideal-typical debugging process as a foundation.

Conference acronym 'XX, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

3 Current Work

As a first step, this doctoral research project has thus far focused on developing a qualitative theory-based analysis approach for investigating screen-recorded debugging processes. To identify students' the debugging processes, all actions performed by the students were mapped to the debugging process steps of an ideal-typical debugging process. Subsequently, the students' processes were contrasted with the ideal-typical debugging process to identify deviations. Preliminary results from a pilot study with five K-12 students working on debugging tasks using Scratch show that students often omitted important steps in the debugging process, like verifying the hypothesis or verifying the solution. In conclusion, the comparison with an ideal-typical debugging process offers deep insights into the students' processes and the possibility to identify deviations that might be the cause of the problems. Moreover, this approach offers a starting point for an extended analysis approach that takes more types of process data into account, facilitating the setup of a comprehensive code book for mapping actions to debugging process steps.

4 Future Work

As a next step, it is planned to capture multi-modal process data to be able to analyze the debugging processes in a more fine-grained way. Besides using screen recordings for capturing the debugging processes, also eyetracking, think-aloud protocols, and retrospective interviews should be used to get as many perspectives on the debugging process as possible. This offers to develop a very detailed code book for mapping actions to the debugging process steps. Moreover, pre- and post-surveys, assessing students' programming skills, their attitudes towards programming and debugging, and the emotions experienced while debugging, allow for the collection of background information about the participants.

As a second next step, it is planned to expand the approach to analyzing log data of debugging processes. For identifying which information is necessary for mapping logs to debugging process steps, first, the screen recordings should be examined in detail again, focusing on this aspect. Then the logs should also be analyzed qualitatively to develop a code book for mapping log data to actions. Afterwards, the results of the analyses of different data types should be contrasted to investigate whether log data miss relevant context information in comparison to screen recordings. Overall, the results of investigating debugging processes can serve as a foundation for automated debugging process analysis. Such automatization offers the chance to conduct large-scale analyses of students' debugging processes, investigating, e.g., common problems. Moreover, automated analysis can be expanded to provide personalized, timely feedback to students and to assess the effect of an intervention, analyzing the influence on the process level.

Acknowledgments

During the preparation of this work, the authors used ChatGPT 5 and Grammarly in order to revise the draft for readability and language. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

References

[1] Keijiro Araki, Zengo Furukawa, and Jingde Cheng. 1991. A general framework for debugging. *IEEE Software*, 8, 3, 14–20. doi:10.1109/52.88939.

[2] Katrina Falkner et al. 2019. An International Comparison of K-12 Computer Science Education Intended and Enacted Curricula. In *Proceedings of the 19th Koli Calling International Conference on Computing Education Research*. Koli Calling '19: 19th Koli Calling International Conference on Computing Education Research (Koli Finland). Petri Ihantola and Nick Falkner, (Eds.) ACM, New York, NY, USA, 1–10. ISBN: 9781450377157. doi:10.1145/3364510.3364517.

[3] Sue Fitzgerald, Gary Lewandowski, Renée McCauley, Laurie Murphy, Beth Simon, Lynda Thomas, and Carol Zander. 2008. Debugging: finding, fixing and flailing, a multi-institutional study of novice debuggers. *Computer Science Education*, 18, 2, 93–116. doi:10.1080/08993400802114508.

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

[5] Irvin R. Katz and John R. Anderson. 1987. Debugging: an analysis of bug-location strategies. *Human-Computer Interaction*, 3, 4, 351–399. doi:10.1207/s15327051hci0304_2.

[6] Claudius M. Kessler and John R. Anderson. 1986. A model of novice debugging in LISP. In *Papers Presented at the First Workshop on Empirical Studies of Programmers on Empirical Studies of Programmers*. Ablex Publishing Corp, USA, 198–212. ISBN: 089391388X.

[7] ChanMin Kim, Emre Dinç, Eunseo Lee, Afaf Baabdullah, Anna Y. Zhang, and Brian R. Belland. 2023. Revisiting Analogical Reasoning in Computing Education: Use of Similarities Between Robot Programming Tasks in Debugging. *Journal of Educational Computing Research*, 61, 5, 1036–1063. doi:10.1177/07356331221142912.

[8] ChanMin Kim, Jiangmei Yuan, Lucas Vasconcelos, Minyoung Shin, and Roger B. Hill. 2018. Debugging during block-based programming. *Instructional Science*, 46, 5, 767–787. doi:10.1007/s11251-018-9453-5.

[9] David J. Malan and Henry H. Leitner. 2007. Scratch for budding computer scientists. *ACM SIGCSE Bulletin*, 39, 1, 223–227. doi:10.1145/1227504.1227388.

[10] Tilman Michaeli and Ralf Romeike. 2019. Improving Debugging Skills in the Classroom - The Effects of Teaching a Systematic Debugging Process. In *Proceedings of the 14th Workshop in Primary and Secondary Computing Education*. WiPSCE'19: 14th Workshop in Primary and Secondary Computing Education (Glasgow Scotland Uk). ACM, New York, NY, USA, 1–7. ISBN: 9781450377041. doi:10.1145/3361721.3361724.

[11] Meghan M. Parkinson, Seppe Hermans, David Gijbels, and Daniel L. Dinsmore. 2024. Exploring debugging processes and regulation strategies during collaborative coding tasks among elementary and secondary students. *Computer Science Education*, 34, 4, 617–644. doi:10.1080/08993408.2024.2305026.

[12] David N. Perkins, Chris Hancock, Renee Hobbs, Fay Martin, and Rebecca Simmons. 1986. Conditions of Learning in Novice Programmers. *Journal of Educational Computing Research*, 2, 1, 37–55. doi:10.2190/GUJT-JCBJ-Q6QU-Q9PL.

[13] Andreas Zeller. 2009. *Why programs fail. A guide to systematic debugging*. Morgan Kaufmann an imprint of Elsevier and dpunkt.verlag, Amsterdam et al. ISBN: 978-0-12-374515-6.