The Development of a Collaborative Tool to Teach Debugging

Sammy Furr

October 27, 2020

Abstract

TODO: write an abstract

Contents

1 Introduction **3**

1.1 Motivation 3

1.1.1 The Value of Teaching Debugging 3

1.1.2 Methods for Teaching Debugging 5

1.1.3 The Value of Collaborative Programming 6

1.1.4 Tools that Enable Collaborative Programming 6

1.1.5 Collaborative Programming and Teaching Debugging . 6

1.2 Project Components 6

1.2.1 Mozilla's rr 6

1.2.2 Kubernetes and Docker 7

1.2.3 React 7

1.2.4 Xterm.js 8

1.3 Project Overview 8

2 Design **9**

3 Next Steps **10**

1 Introduction

1.1 Motivation

Debugging is invaluable in writing and understanding code, yet it is rarely formally taught [9]. We typically teach students programming structures, concepts, and languages, but leave them to learn the tools they use to write code by themselves. This approach often works well—a programmer's choice of tools is often *very* personal and students figure out how to configure an individualized workflow. Perhaps because debuggers are tools, students are often expected to learn them with minimal guidance. Unlike editors or reference guides however, effectively using a debugger requires a set of high-level, platform agnostic, teachable skills. Teaching these skills is effective, and translates into better, faster, debugging and programming [8] [10].

1.1.1 The Value of Teaching Debugging

There is an unfortunate lack of research specifically into the efficacy of teaching debugging for computer science students, despite a recent rise in the inclusion of debugging in "computational thinking" curriculums [10]. These curriculums attempt to teach skills in computer science classes that translate into other subject areas: the UK's computer science curriculum considers debugging an essential "transferable skill" [1].

There seems to be confidence that the problem-solving techniques used

in debugging are widely applicable, but of greater interest to computer science teachers is whether teaching debugging directly benefits student programmers. Michaeli and Romeike conducted a good, albeit somewhat small, study on the efficacy of teaching a systematic debugging process to K12 students. They found that students who have been taught a specific debugging framework performed better in debugging tests and were more confident in their own debugging skills [10]. Their result is positive evidence towards the efficacy of teaching debugging, though it doesn't include college or university students.

As Michaeli and Romeike point out, there is a lack of research into the value of teaching debugging in higher education. None of the research these authors found placed much focus on explicitly teaching debugging. Chmiel and Loui studied whether students who were provided with debugging tools and frameworks performed better on tests or spent less time on assignments than those who were not [2]. Though this research wasn't able to find conclusive evidence towards better performance on tests or assignments, it did find that students in the treatment group felt more confident in their debugging abilities. Unfortunately Chmiel and Loui's study didn't involve extended explicit teaching of debugging—use of the tools was voluntary, and variations in the students' individual abilities made the data difficult to evaluate.

Though there is a lack of higher-education research, the value of teaching debugging is still demonstrable. The research discussed all finds that K-12 and college students alike commonly resort to sporadic debugging techniques

when beginning to learn. Since this pattern of behavior that explicitly teaching debugging corrects exists in college as well as in K-12 students, it seems logical that the benefit of explicitly teaching debugging to K-12 students should be realized equally by their collegiate counterparts.

1.1.2 Methods for Teaching Debugging

Similarly to research on the value of teaching debugging, research into how to best teach debugging is self-admittedly sparse. Chan et al. allow that "in general research on how to improve debugging is sporadic"—an observation that leads them to research a framework to reduce the complexity of teaching debugging [8]. To organize their framework, they split debugging knowledge into 5 categories: *Domain*, *System*, *Procedural*, *Strategic*, and *Experiential*. They then review different debugging tools and teaching aids—from those that involve writing code to games—and map tools to the knowledge areas they seek to address. After an evaluation of a host of different tools, they claim to find a few significant faults in current debugging teaching platforms. The primary two which this project seeks to address are as follows:

1. A lack of back-tracing ability/coverage.

2. A lack of tools addressing system knowledge (an understanding of the program to be debugged).

1.1.3 The Value of Collaborative Programming

TODO: find some basic research that backs up the claim that collaborative/-pair programming is worthwhile.

1.1.4 Tools that Enable Collaborative Programming

TODO: write about glitch, repl.it, etc.

1.1.5 Collaborative Programming and Teaching Debugging

Debuggers exist at an intersection of tools and skills similar to programming languages themselves. By becoming familiar with a specific debugger, students learn techniques and paradigms necessary to use all debuggers effectively.

1.2 Project Components

1.2.1 Mozilla's rr

Overview rr is "a lightweight tool for recording, replaying and debugging execution of applications". [12] rr allows a programmer to record the execution of a program on any compatible machine and replay the execution later. This enhances GDB's ability to "time-travel" when debugging, using commands such as `reverse-continue` and `reverse-stepi` [5] to step backwards and forwards through a program's execution. Through a novel encapsulation of the execution space, rr is able to deterministically record and replay the

execution of syscalls and other process behavior that differs run-to-run. This is invaluable when trying to debug behavior that is not entirely dependent on the code being debugged. A typical workflow in rr consists of recording an inexplicable error, replaying execution to find the area in which the error occurs, and then narrowing in on the bug not by re-running the entire program, but by progressing back and forth through execution in the problem area.

Limitations In comparison to solutions like PANDA [4] that rely on capturing the entire state of of a virtual machine to replay execution, rr records and replays faster, produces far smaller files, and doesn't force execution inside of a VM. [11] The tradeoff for these benefits are two major system limitations: rr is only compatible with the Linux kernel, and it's deterministic recording and replay relies on a feature that is only found on modern *Intel* x86 CPUs. These limitations influenced the development of this project as a webapp similar to existing tools for collaborative programming. Luckily, the speed and size benefits of rr lend themselves well to non-local execution.

1.2.2 Kubernetes and Docker

TODO: Write about k8s and docker.

1.2.3 React

TODO: Write about react.

1.2.4 Xterm.js

TODO: Write about Xterm.js if I end up using it (which seems overwhelmingly likely ATM).

1.3 Project Overview

TODO: a brief overview linking aspects of the project to background research discussed above.

2 Design

TODO: the big piece of writing, detailing components of the project, the system as a whole, and motivations for design choices.

3 Next Steps

TODO: write about additional/necessary unimplemented features (ATM this is looking like auth stuff, unless OAuth2 proves to be less of a pain in the ass then when I've used it before in Javascript. Maybe some of the vis stuff as well, though I'd really like to get the basics done here.) Write about desire to actually do research and see whether the tool helps students learn debugging.

References

[1] Brown, N. C. C., Sentance, S., Crick, T., and Humphreys, S. Restart: The resurgence of computer science in uk schools. *ACM Trans. Comput. Educ. 14*, 2 (June 2014).

[2] Chmiel, R., and Loui, M. C. Debugging: From novice to expert. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education* (New York, NY, USA, 2004), SIGCSE '04, Association for Computing Machinery, p. 17–21.

[3] crazy rabbidz. GDB Enhanced Features, May 2020.

[4] Dolan-Gavitt, B., Hodosh, J., Hulin, P., Leek, T., and Whelan, R. Repeatable reverse engineering with panda. In *Proceedings of the 5th Program Protection and Reverse Engineering Workshop* (New York, NY, USA, 2015), PPREW-5, Association for Computing Machinery.

[5] Free Software Foundation. *Debugging with GDB*, 2020.

[6] Godbolt, M. Compiler Explorer, May 2020.

[7] Guo, P. Python Tutor, Feb. 2020.

[8] Li, C., Chan, E., Denny, P., Luxton-Reilly, A., and Tempero, E. Towards a framework for teaching debugging. In *Proceedings of*

the Twenty-First Australasian Computing Education Conference (New York, NY, USA, 2019), ACE '19, Association for Computing Machinery, p. 79–86.

[9] McCauley, R., Fitzgerald, S., Lewandowski, G., Murphy, L., Simon, B., Thomas, L., and Zander, C. Debugging: a review of the literature from an educational perspective. *Computer Science Education 18*, 2 (2008), 67–92.

[10] Michaeli, T., and Romeike, R. 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* (New York, NY, USA, 2019), WiPSCE'19, Association for Computing Machinery.

[11] O'Callahan, R., Jones, C., Froyd, N., Huey, K., Noll, A., and Partush, N. Engineering record and replay for deployability: Extended technical report. *CoRR abs/1705.05937* (2017).

[12] O'Callahan, R., Jones, C., Froyd, N., Huey, K., Noll, A., and Partush, N. rr, Dec. 2019.

[13] Walker, J., Wang, M., Carr, S., Mayo, J., and Shene, C.-K. A system for visualizing the process address space in the context of teaching secure coding in c. In *Proceedings of the 51st ACM Technical Sym-*

posium on Computer Science Education (New York, NY, USA, 2020), SIGCSE '20, Association for Computing Machinery, p. 1033–1039.