Teaching Debugging Collaboratively—Midway Report

Sammy Furr

May 11, 2020

1 Introduction

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 we leave them to learn the tools they use to code by themselves. This approach often works well—the programmer's choice of editor is *very* personal, students figure out how to configure an individualized workflow. Perhaps because debuggers are tools, they often get lumped into the "teach yourself" category. 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].

The use of a debugger is particularly important in a low-level programming classes, which is often the first time that students encounter concepts like assembly instructions and memory addresses. Debuggers such as GDB offer incredibly powerful tools to step through and inspect running code. Sadly, low level debugging tools are often less than intuitive, and provide little to no means for collaboration. Powerful tools such as RR exist that enable collaborative "record and replay" debugging [11], but they lack tools to visualize address spaces and control flow of programs. Research shows that visualization of these previously unexplored spaces aids students taking low-level or systems programming classes [12].

The necessity for accessible, collaborative tools for teaching all aspects of computer science, not the least debugging, has been exemplified by the current COVID-19 crisis. Many students don't have access to a high powered computer (or even a computer at all) to run tools that allow for collaboration and replay by capturing the entire state of a virtual machine [4]. Though the limitations of interacting with a debugger through a phone or low-powered laptop seem daunting, I believe they are actually constraints that can help better shape a visualized, collaborative approach.

For my senior project, I want to develop a front-end interface to RR that enables collaborative debugging and process visualization on a variety of non-traditional platforms.

2 The Value of Teaching Debugging

Before creating a platform for teaching debugging it's of course important to make sure that teaching debugging is actually valuable. There is not a lot 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 are useful in other subject areas: the UK's computer science curriculum considers debugging an essential "transferable skill" [1]. Though there seems to be confidence that the problem-solving techniques used in debugging are widely applicable, I am more interested in whether systematically teaching debugging actually benefits computer science students. 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 their research 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 they 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]. They weren't able to find conclusive evidence towards better performance on tests or assignments, though they did find that students in the treatment group felt more confident in their debugging abilities. Unfortunately their 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.

Despite the lack of higher education-specific research, I feel that the value of teaching debugging has still been demonstrated. The research discussed all finds that K12 and college students alike commonly resort to sporadic debugging techniques when beginning to learn. Given this, I see no reason why the conclusive and positive research toward the benefit of explicitly teaching debugging to K12 students should not apply to their collegiate counterparts.

Research into how to best teach debugging is also self-admittedly sparse. Chan et al. allow that "in general research on how to improve debugging is

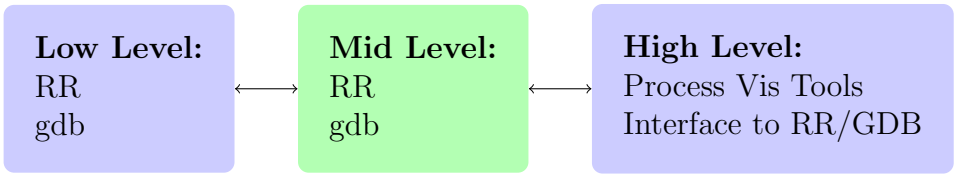
sporadic"—an observation that leads them to research a framework to reduce the complexity of teaching debugging [8]. To organize their framework, they split debugging knowledge into 5 categories: Domain, System, Procedural, Strategic, and Experiential. They then review different debugging tools and teaching 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 two of which I seek to address follow:

1. A lack of back-tracing ability/coverage.
2. A lack of tools addressing system knowledge (an understanding of the program to be debugged).

Basing my collaborative debugging platform on RR solves the problem of back-tracing ability and coverage. RR is a "time traveling" addition to GDB, that allows for unlimited recording and replaying code execution deterministically, both backwards and forwards [11]. By integrating the debugger with tools to visualize the execution and code space in an accessible way, I hope to address the issue of system knowledge. Though the system knowledge gained may be primarily applicable to low level programming, skills such as visualizing code execution and memory space are applicable to all types of coding. These solutions are discussed in depth in the following sections.

3 Collaborative Debugger

3.1 Overview



3.2 Low Level

Overview of RR Time traveling debugging—stepping backwards and forwards through a program's execution—is supported naively by GDB through the use of commands such as `reverse-continue` and `reverse-stepi`. [5]

What RR brings to the table is the ability to deterministically record and replay the execution of even very complex programs. This allows single users to capture and debug events that may not happen every time a program runs and revisit debugging later. Essentially for my planned collaborative debugger, execution can be replayed and debugged even *on a different computer* than the computer that recorded the original execution. 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]

Limitations of RR RR comes with two major limitations: it 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. For my purposes the first limitation is fairly irrelevant–I expect students to be connecting to a central server in order to execute and replay code, and most systems classes are taught in Linux. The second limitation poses more of a problem for instruction. Because recording and replay is limited to x86 CPUs, classes cannot use a debugger based on RR to teach ARM assembly, which is becoming increasingly popular. Though this isn't a limitation that solutions like the aforementioned full-VM recorder PANDA face, RR still maintains the advantage of being a debugger first and foremost. Students can learn RR, GDB, and debugging techniques simultaneously. Platforms that require entire VM replay come with far greater performance and instructional overhead. Because of RR's ease of use and tight integration with GDB, I feel it is the best choice of platform to base a collaborative debugger on.

3.3 High Level

There are many existing tools to visualize parts of the process space during execution, such as Python Tutor's C Tutor [7]. Likewise, there are excellent platforms like Matt Godbolt's Compiler Explorer [6] that allow students to easily see the mapping of C code to assembly. Unfortunately there are no student friendly tools such as these that integrate fully with GDB. There are plugins that enhance the features of GDB to offer some further visualization, like GEF [3], but these are typically designed with experts in mind. While I feel that it is essential for students to learn real-world tools like GDB, research shows that visualization tools designed with teaching in mind significantly

increase students' understanding [12]. I want to marry the teaching-oriented interface of tools like C Tutor and the Compiler Explorer with the real-world power of GDB and RR.

3.4 Mid Level

4 Next Steps

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. 1721.

[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. 7986.

[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), WiPSCE19, 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] 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 Symposium on Computer Science Education* (New York, NY, USA, 2020), SIGCSE 20, Association for Computing Machinery, p. 10331039.