The Development of a Collaborative Tool to Teach Debugging

Sammy Furr

November 27, 2020

Abstract

TODO: write an abstract

Contents

1 Introduction

1.1 Motivation

Debugging is invaluable in writing and understanding code, yet it is rarely formally taught [16]. 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 [15] [17].

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 [17]. 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" [10].

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 [17]. 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 [11]. 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 [15]. 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 Tools Used

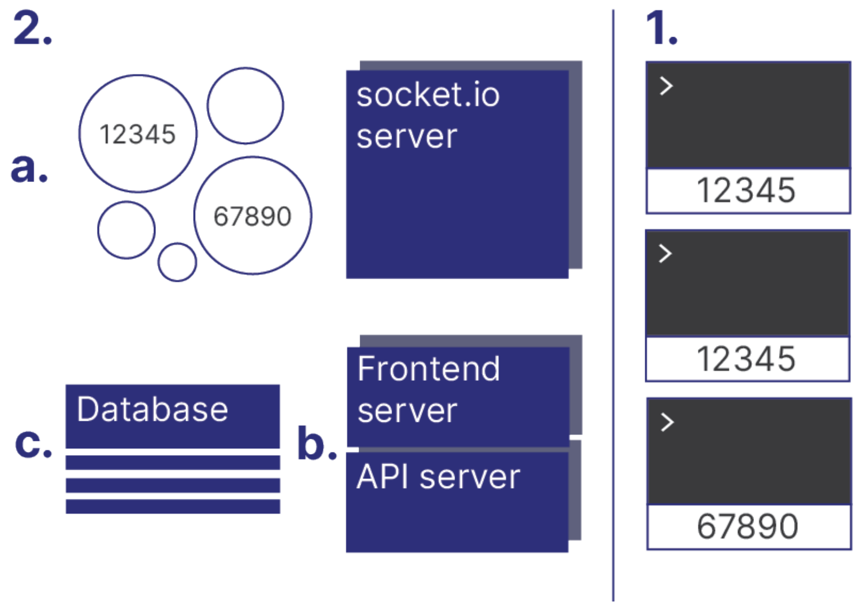

Figure 1: Overview of the Collaborative Debugger

The next sections give an overview of the various tools used to create the collaborative debugger. The debugger consists of:

1. A frontend web app built using React (1.2.7) that presents a debugging interface to the end user.

2. A distributed backend managed by Kubernetes (1.2.1), split into three parts:

    (a) A pod for each debugging instance which runs the rr debugger (1.2.2). These communicate directly with users through WebSocket server pods. (1.2.4).

(b) Pods running a frontend server written in Node.js which works in tandem with an API server created using Flask (1.2.6). The API server manages creation and destruction of debug sessions, as well as authentication.

(c) A MongoDB (1.2.5) database.

1.2.1 Kubernetes

Kubernetes is the defacto standard in container orchestration software. It provides a layer of abstraction on top of normal containers, like those created by Docker. By bundling one or more closely linked containers into a "pod", Kubernetes is able to manage deployment and re-deployment of applications running inside containers. It is trivial to create new pods, or to create multiple pods running the same application as needed within a Kubernetes cluster [4]. The speed at which even relatively large pods can be created and the inherent security provided by containerization drove the decision to create a new pod on the fly for each debugging instance in the collaborative debugger.

Kubernetes also provides services to facilitate load balancing, manage storage volumes, and contain secrets. The abstraction provided by these features, in tandem with the ease of Kubernetes deployment on a managed Kubernetes service [1] greatly accelerated development.

1.2.2 Mozilla's rr

Overview rr is "a lightweight tool for recording, replaying and debugging execution of applications" [19]. 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` [13] 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.

rr is an ideal tool for teaching debugging because it allows instructors to record execution of a program and design a debugging example with the knowledge that normally non-deterministic events will be repeatable, and that any input they provide to the program will be exactly replicated. With the collaborative debugger, teachers can record a program's execution and design a debugging lesson which students can work on together. The repeatablity of rr means that students can focus on debugging, and teachers can create as specific examples as they please.

Limitations In comparison to solutions like PANDA [12] 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. [18] 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. In conjunction with Kubernetes, it takes a few seconds to create a new container running rr and connect to web clients.

1.2.3 pygdbmi

In order to "support the development of systems which use the debugger as just one small component of a larger system", GDB provides a machine-oriented interface called GDB/MI [13]. rr supports interaction through GDB/MI, and using the interface was a natural choice for the collaborative debugger. In addition to being far easier to interact with from within a program, the structured, machine-friendly output of GDB/MI lends itself in particular to future development of visualization aids in the collaborative debugger.

To parse rr output into Python dictionaries and to easily control rr as a subprocess, pygdbmi [20] is used in each debugging pod. pygdbmi's abstraction simplifies programatically controlling rr. A pod can receive a command from the client, pass it to rr, and respond without having to deal with parsing GDB/MI output or managing the rr process.

1.2.4 Socket.IO

To speed communication, the collaborative debugger uses WebSockets to directly connect web clients and the pods running rr. Socket.IO is a library that extends WebSockets. It provides backup incase a WebSocket connection cannot be established, enables automatic reconnection and disconnection detection, and adds support for namespaces [8]. The collaborative debugger uses the standard JavaScript implementation of Socket.IO on the client side. Messages are passed through a server to individual debugging pods, both of which use the Python implementation of Socket.IO, python-socket.io [14].

1.2.5 MongoDB

The collaborative debugger uses a database to store information about users, pods, and example debugging sessions. Due to it's speed of deployment and natural interaction with the object-oriented languages used to create the project, MongoDB was chosen as database software [2].

1.2.6 Flask and Node.js

The primary server for the collaborative debugger is split into two sections: a simple Node.js [6] server that serves the frontend webapp, and an API server created using Flask [3]. While in development, the builtin React (see next section) development server is used to serve the frontend. This makes debugging the frontend far easier.

An API server is necessary to authenticate users and to provide a means to create/delete debugging sessions. Since the rest of the backend was created using Python, Flask was chosen to create the API server. Flask is a lightweight web application framework which lends itself perfectly to interacting with the Python MongoDB and Kubernetes APIs.

1.2.7 React

React is JavaScript library that simplifies creating user interfaces and managing state [7]. React's state management is of particular importance to the collaborative debugger's frontend. State constantly changes as users create/delete debugging sessions, join existing sessions, and communicate with rr. React allows classes to encapsulate components such as a list of existing debugging sessions, a view of the current program's source code, and the terminal interface with rr. Instances of these classes maintain state and update efficiently.

The frontend makes extensive use of JSX, syntax which allows the inclusion of segments of HTML code within a React app written in JavaScript. This makes it easy for each component of the one-page webapp to hide/show subcomponents as state changes.

1.2.8 Monaco and Xterm.js

After joining or creating a debugging session, users spend most of their time interacting collaboratively with rr. Their primary interface to rr is through Xterm.js, a frontend component that makes it easy to emulate terminal behavior in the browser [9]. With a few control methods, it is simple to provide a terminal interface to rr that is virtually indistinguishable from a local session. By using the Xterm.js based interface, students can learn to use rr (and by extension gdb) collaboratively, and directly translate that knowledge to individual work.

In addition to the terminal interface, the frontend shows a view of the current source file being debugged. The Monaco Editor [5] is used to display this source view. Though more complex than strictly necessary to display code, Monaco makes it easy to format and syntax-highlight. Using Monaco also simplifies the future addition of editing source code, should the need arise. React's state management allows updating text in the editor as efficiently as possible.

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] Managed Kubernetes on DigitalOcean. https://www.digitalocean.com/products/kubernetes/.

[2] Mongodb. https://www.mongodb.com.

[3] Flask, Nov. 2020. https://github.com/pallets/flask.

[4] kubernetes/kubernetes, Nov. 2020. https://github.com/kubernetes/kubernetes.

[5] Monaco editor, Nov. 2020. https://github.com/microsoft/monaco-editor.

[6] Nodejs, Nov. 2020. https://github.com/nodejs/node.

[7] React, Nov. 2020. https://github.com/facebook/react.

[8] socket.io, Nov. 2020. https://github.com/socketio/socket.io.

[9] xtermjs/xterm.js, Nov. 2020. https://github.com/xtermjs/xterm.js.

[10] 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).

[11] Chmiel, R., and Loui, M. C. Debugging: From novice to expert. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Sci-*

ence Education (New York, NY, USA, 2004), SIGCSE '04, Association for Computing Machinery, p. 17–21.

[12] 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.

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

[14] Grinberg, M. python-socketio, Nov. 2020. https://github.com/miguelgrinberg/python-socketio.

[15] 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.

[16] 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.

[17] 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.

[18] 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).

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

[20] Smith, C. pygdbmi, Nov. 2020. https://github.com/cs01/pygdbmi.