Course Proposal: Debugging Fundamentals and Techniques

CS4991 Capstone Report, 2024

David Lin
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
lindavid404@gmail.com

ABSTRACT

Software developers spend a large portion of their time debugging software, so the software developers that the University of Virginia produces would greatly benefit from more exposure to efficient debugging techniques during their time at the university. I propose that the computer science curriculum include more exposure to the debugging process in the CS curriculum through a required class or elective for both BACS and BSCS, where students learn the fundamentals of debugging and practice applying them. Alternatively, a section dedicated to debugging should exist in one of the appropriate foundation courses: Introduction to Programming (CS 111x) or Software Development Essentials (CS 3140). These changes to the CS curriculum will prepare students for potential software issues and help them understand the importance of writing maintainable code. Currently, the proposal needs to reach the CS curriculum designers, who would then design the necessary optional debugging exposure and eventually determine its viability before making it mandatory to all CS students.

1. INTRODUCTION

According to an article on Undo.io (n.d.), software developers spend about 25-50% of their time debugging software. Some sources claim the time spent goes as high as 90% (Detre, 2020). Since debugging is so time-consuming for developers, CS students at UVA would benefit from exposure to efficient debugging methods within their curriculum.

Amazon Web Services defines debugging as: "…the process of finding and fixing errors or bugs in the source code of any software. When software does not work as expected, computer programmers study the code to determine why any errors occurred" (AWS, n.d.). Debugging is a critical part of updating and maintaining software, especially old software as it is common for the original developers to be absent from the current team responsible for maintaining it.

2. RELATED WORKS

Instructors from the University of Illinois at Urbana-Champaign ran an experiment to see the effects of giving some students debugging exercises. Students, who were given debugging exercises for four out of five programming assignments, kept an optional debugging log, and documented "design decisions, development plans, and overall debugging experiences" (Chmiel & Loui, 2004). There was a statistically significant difference between students that received the treatment versus the control group: the treatment group were able to complete the assignments an hour earlier, on average, compared to the control group.

Instructors from the University of Nottingham conducted an experiment with their students who are novice programmers. After assessing the student's competency, they were placed in overlapping categories: 59 were good programmers, 56 were weak debuggers, and 35 were good debuggers. They found that 66% of the good debuggers were also good programmers, but surprisingly only 39% of the good programmers were good debuggers (Ahmadzadeh, et al., 2005).

The experiment, done by instructors from the University of Illinois at Urbana-Champaign, points out the effectiveness of exposing students to debugging exercise while the experiment conducted by Instructors from the University of Nottingham demonstrates that programming competency does not correlate with debugging competency. These works motivated my proposal to incorporate a more formal debugging education into UVA's CS curriculum.

3. PROPOSED DESIGN

My proposal for supplementing debugging exposure is divided into three sections: 1) create a stand-alone one-credit debugging course, 2) add onto existing CS course, and 3) the pros and cons of either approach. This would give the curriculum designers options when considering the incorporation of debugging exposure. The following design draws inspiration from reputable institutions such as Google and Massachusetts Institute of Technology (MIT) and the study done by the instructors of University of Illinois at Urbana-Champaign.

3.1 Option 1: Stand-alone Course

The first option is to introduce a one-credit course dedicated to debugging. This class is intended to be low-stress and taken the semester after Introduction to Programming (CS 111x) to help novice developers learn the tools and develop the mindset for debugging. The course will meet once a week to discuss the week's debugging topic and provide related exercises for students to work on as homework. Topics this new course may cover, which I will go more in depth in this proposal, include: 1) how to effectively use a debugger, 2) how to write bug-proof code, and 3) common bugs. Other potential topics, which will not be covered in this proposal, may also include how to deal with ambiguity in instructions and avoiding introducing new bugs when debugging.

3.1.1 Teaching the Debugger

Debuggers are built into many integrated development environments (IDE). They provide a convenient interface for seeing the current state (values of different variables) and which instruction is executing. Since students have written python in the PyCharm IDE during CS111x, this new course can either expose students to debuggers through PyCharm or Eclipse, which is the IDE students would use for the class following CS111x (Data Structures and Algorithms 1 or DSA1).

Lecture(s) to introduce debuggers should explain basic debugging tools such as breakpoints, stepping into, and stepping over. It should also give guidance and provide examples for effective breakpoint placement. After the lecture, students would debug a couple of pre-written programs of moderate complexity (complex enough that it would be difficult to find the bug just by reading the code) to practice stepping through the code.

3.1.2 Writing Bug-Proof Code

Handling bugs can happen before they arise by writing clear and maintainable code. Lectures for this section can reference existing coding conventions of reputable companies, such as Google. Some coding conventions from Google's Python Style Guide (n.d.) provide guidance on how to document functions, determine if certain lines of code need to be commented, and name variables. Another useful convention from Google's C++ Style Guide (n.d.) provides guidance on determining the scope of a variable on declaration.

Aside from writing clear and maintainable code, students should be vigilant for potential bugs as they program. Lectures can provide guidance on where to check the parameters of a function or what the ideal data structure is for different scenarios. MIT's Software Construction course's reading (MIT.edu, n.d.) provides other strategies for avoiding potential bugs: use immutable types whenever possible, localize bugs using runtime assertion, develop incrementally though unit and regression testing, and modularize and encapsulate the code. Although this reading was intended for novice programmers learning Java, many of these ideas are universal and are applicable to other programming languages and the software development process.

3.1.3 Recognizing Common bugs

Another way to prevent injecting bugs into the code requires knowledge of the potential bugs. This knowledge will create awareness of where and how those bugs can be introduced. Since "a few bug types account for a lot of the mistakes made by students learning to program" (Spohrer & Soloway, 1986), a section of the course will be dedicated to covering the common bugs and errors.

Instructors from the University of Nottingham found that, of the 108,652 records of student Java errors, 32% were syntax (programming language's grammar) errors, 63% were semantic (not consistent with the language) errors, and 1% were lexical (unrecognized token) errors. They also found that the most common semantic errors were "field not found," "use of non-static variable inside the static method," and "type mismatch" (Ahmadzadeh, et al., 2005). This course can prepare students for many errors by explaining the possible bugs causing that can cause semantic errors and providing examples during the lecture. If there is time, other common errors, including "using a non-initialized variable," "method call with wrong arguments," and "method name not found," (Ahmadzadeh, et al., 2005) could also be covered. Although syntax and lexical errors made up the rest of the errors, they can only be consistently avoided once the developer is familiar with the language, which requires time and practice. So, the lecture(s) on this section will only cover common semantic errors and the potential bugs that may cause them.

3.2 Option 2: Addition to Existing Course

The second option is to add a section dedicated to debugging in an existing CS course. Since debugging is a skillset that is beneficial to programmers of all levels, this addition should complement a class CS students take early in their curriculum. The two most relevant classes that could benefit from additional debugging instructions are Introduction to Programming (CS 111x) and Software Development Essentials (CS 3140).

Topics addressed in the previous sections of the proposal can be added into a new section of the syllabus of either CS 111x or CS 3140. It is unlikely that all topics addressed in this proposal would be thoroughly covered within a short period of time, so only the most important topics should be covered. I believe that these sections are how to use a debugger and write bug-proof code.

3.3 New Class vs. Enhancing an Existing Class

UVA CS curriculum designers should weigh time against the importance of having a solid debugging foundation when deciding whether to dedicate a class to debugging or supplement a pre-existing course with the content outlined in the previous sections. Dedicating a class to debugging ensures students receive proper guidance on this critical skill. This class would compete for time against other courses on students' schedule, but it would place debugging in the spotlight rather than mentioning many of its concepts as a side note. Highlighting this skill as the main focal point draws students' focus, thus helping them build the correct foundation for software development.

4. ANTICIPATED RESULTS

Although students may not continue to use the programming language or debugging tools that this course or section teaches the different debugging concepts with, developing the correct mindset would be beneficial regardless of what language or tools the student programs with. Exposing students to these concepts and tools opens their minds to looking for similar concepts or tools that would aid them in their programming endeavors.

Another benefit of having experience and understanding of debugging is improving the chances of a student receiving return offers from an internship. Currently, the University relies on internships to help translate theoretical concepts taught in classrooms into applicable knowledge and experience. This is indeed an efficient approach, but not all concepts taught in classrooms are applicable for each unique company. Understanding how to approach debugging a piece of software, however, will almost always be applicable. Ensuring that a student understands efficient debugging techniques will certainly allow them to work more efficiently and place them in the best possible light for a chance at a return offer from their internship.

5. CONCLUSION

UVA CS curriculum will benefit from incorporating more debugging exposure as a new class or as a supplement to existing classes. Novice developers will greatly benefit from formal debugging training as it would help them develop the proper mindset to efficiently resolve bugs and write code in a way that makes introducing bugs difficult. By developing this mindset, students will find building large projects or working with prewritten code to be less overwhelming.

6. FUTURE WORK

Since this is currently a proposal, nothing has been forwarded to the CS curriculum designers. However, a conversation can be started with any CS professors, explaining the benefits of including more debugging exposure in their curriculum. The more curriculums that include debugging as part of their syllabus, the more likely the curriculum designers would take notice.

After the curriculum designers recognize the benefits of formal debugging concepts and exposure to the curriculum, they can make it optional. They can survey or use other methods to gauge the success and comfortability of CS students that have debugging exposure versus students that do not have the debugging exposure. Once there is data suggesting that formal guidance in debugging is beneficial to the success of CS students, the debugging class (or addition to an existing class) should be made mandatory.

REFERENCES

Ahmadzadeh, M., Elliman, D., & Higgins, C. (2005). An analysis of patterns of debugging among novice computer science students. *SIGCSE Bulletin*, *37*(3), 84–88. https://doi.org/10.1145/1151954.1067472

Detre, G. (2020, August 10). *How to write good software faster (we spend 90% of our time debugging)*. Making Data Mistakes. https://www.makingdatamistakes.com/how-to-write-good-software-faster-we-spend-90-of-our-time-debugging/

Chmiel, R., & Loui, M. C. (2004). Debugging: from novice to expert. *SIGCSE Bulletin*, *36*(1), 17–21. https://doi.org/10.1145/1028174.971310

Google C++ Style Guide. Google C++ style guide. (n.d.).

https://google.github.io/styleguide/cppguide

Google Python Style Guide. (n.d.). https://google.github.io/styleguide/pyguide

MIT.edu. (n.d.). *Reading 08: Avoiding Debugging*. 6.005: Software Constructionhttps://web.mit.edu/6.005/www/fa15/classes/08-avoiding-debugging/

Undo. (n.d.). *Reduce time spent debugging*. https://undo.io/solutions/developer-productivity/reduce-time-spent-debugging

Spohrer, J., & Soloway, E. (1986). Novice mistakes: are the folk wisdoms correct? *Communications of the ACM, 29*(7), 624–632. https://doi.org/10.1145/6138.6145

AWS. (n.d.). *What is debugging?* https://aws.amazon.com/what-is/debugging/