Debugging and Error Management Lessons for Undergraduate Introductory Computer Science Classes

Joseph Brooks

Gardner-McCune

This paper introduces extra assignments and coursework into an undergrad introductory programming course in order to emphasize and develop the skills to manage compiler errors, debug, and test code. The emphasis is on 20 minutes of TA or lecture time followed by 1 hour programming assignments. Three different lessons have been created. The first teaches how to read and deal with syntax and compiler errors, the second emphasizes block testing code, and the third teaches how to properly use a debugger.

The created lessons can be found on github at:

https://github.com/brooksturtle/error_management_lessons.git

The lessons were written in Java but could easily be translated to python or another introductory language. The pacing and expected ability of the students is based on the COP 2800 introductory Java course at the University of Florida. Small assignments are used to focus on only one aspect at a time, as intro students may be overwhelmed by a large assignment that requires learning too many skills at once.

Many students in intro to programming classes become frustrated when their programs have errors that could take hours of struggling to solve. Many students mistakenly assume a lack of compiler errors translates to a working program, and do not test their programs (Kazemian, 2008). This causes students to feel frustrated and overwhelmed when trying to debug their programs. Debugging and validation is given very few hours in Computer Science courses (Laporte, 2007). But these skills are extremely important and could help combat the 30% attrition rate for Computer Science freshman (ACM, 2018).

In a survey of freshman and sophomores, only 23% consistently unit test their code, and only 40% of students consistently trace the logic flow of their code before testing (Kazemian, 2008). Further, since the increase in usage of automatic grading and test code by instructors, it is felt that students do not take the initiative to test their own code, relying only on the instructors given code

(Kazemian, 2008). Students equate passing the instructors test code as the only necessary benchmark to succeed (Edwards, 2004). This causes students to never develop the ability or mindset to test their code, making any advancement into more challenging classes or a professional computer science degree difficult. Further, constant testing of code encourages and aids students in tracing through the logic of their code, which is extremely important in developing students ability to read and comprehend a program. Without this focus, some students will blindly program and wait until the end to compile, causing a “big bang” of messy errors (Edwards, 2004). It is no surprise so many students drop Computer Science every year if every time they program they are overwhelmed by a mountain of errors and never develop the good habits to deal with them. The lessons taught are simple but help create beneficial lifelong habits.

Lesson 1: Error Detection

The first assignment is a simple class with two methods. The first method multiplies two integers and the second prints the result. The code has 7 compiler errors that the students must fix to get the code to run. The students will be required to make a separate document where they write each error they find, describe it, and how to solve it. They will be highly encouraged to google each error and find the solution on stack overflow. Some errors like brackets and semi colons are simpler fixes and don’t require google searches.

The seven errors were all part of a series on the most common java errors (Stringfellow, 2017) and when copying the terminal message into Google, it was checked that the solution to each error comes up within the first three search results.

The purpose of this assignment is to develop the ability to deal with compiler errors. Many students are unaware of the vast resources available to programmers on the web and may never even think to look things up online. The lesson reinforces how convenient sites like StackOverflow can be and how searching for a compiler issue verbatim is a simple and effective tool.

Further, beginner programmers may be overwhelmed with multiple compiler errors that can crop up at the same time. This assignment would be prefaced by a 20 minute TA or in class lecture demonstrating debugging an example java program with errors. The program would preferably have multiple errors when compiled and the lecturer would emphasize starting from the first compiler error and fixing them one at a time while recompiling each time. Students will also be given a link to a guide of the 50 most common Java Errors and How to Avoid Them, another useful resource for beginner programmers (Stringfellow, 2017).

Scaffolding/ Weaknesses: Students may find sifting through a large amount of online resources and finding reliable sources for an error difficult. Good sources were made sure to be in the first three google results for each error, and a link to a reliable guide will also be given. Because students may still be overwhelmed by the number of compiler errors, a list of all the errors within the program will be given a few days before the assignment is due.

Student Current Knowledge: As the students are first getting used to methods, 2-3 weeks into the course. Assumes knowledge of basic variable types, but before the first major programming assignment is given.

Lesson 2: Unit Testing

According to a survey of first year CS undergrads, only 23% consistently unit tested their code and only 40% consistently traced a codes logic flow before testing (Kazemian, 2008). Some students may mistakenly believe a lack of syntax errors is the only necessary proof that their program works. Further, with the increase in instructors giving automatic testing for programs, students rely less on themselves to check their code and only use what is given by instructors (Edwards, 2004). Without students constantly testing and going through their code, opportunities to increase code and logic comprehension are lost as students pay less attention to what they have typed before moving on.

This lesson has students fill in several preset methods dealing with adding, deleting, and modifying strings from an array list. Test code is given along with the assignment and as each method is completed the students will copy and paste the test test code into main to test the methods individually, teaching the value of isolating and testing snippets of code. Each method builds on the last and the students will be told to do the methods in order.

This lesson is important because it will emphasize the fact that a program can be written and tested in snippets, using print statements. Because if everything is tested and run at once it could become impossible to debug. It also teaches that by making sure basic methods work, one can build up to fairly complex functions that rely on those basic methods. Finally, the assignment introduces the students to edge cases.

The assignment would be prefaced by a 20 minutes of TA or lecture time. The talk would emphasize the need to constantly compile and run your program, and how just because a program compiles and runs does not mean it will have the desired output. 2-3 examples of code that compiles but has unplanned output would also be demonstrated. The lecture will also complete the first method in class, and demonstrate how to copy and paste the test code into the main method and read the printed output in terminal.

Student Knowledge: 3-4 weeks into an introductory course. The students should be comfortable calling methods from within a class, and have a basic knowledge of arrays.

Scaffolding/ Weakness: Students may be unfamiliar with ArrayList, so a guide is given at the beginning of the program. Students may not know what to do with the test code, so TA will demonstrate how to to complete and test the first method in class. Students may have trouble printing out an ArrayList, so they are given a method that does this in the program. Students may have trouble following multiple method calls and determining what the correct output should be, so each method has a description of what it does, and each method only adds or deletes one string at a time.

In Class Demo: A demonstration of lesson 2 was given in class. The students were given the test code and program and told to complete the assignment. There was much confusion about what they were required to do. To fix this, the test code was turned into a text file instead of a java file so they do not mistakenly try to compile it. I also completed the first method for the students and have already inserted the test code for that method into the program so that they can see what needs to be done. More detailed feedback and how it was integrated is in the github repository.

Lesson 2 teaches the students the usefulness of test code and print statements, and is a step towards the students creating their own test code for each project. While outside the scope of this paper, later lessons would introduce Test Driven Development and require students to submit test code along with every project (Edwards, 2004).

Lesson 3: Learning to use a Debugger

Students dealt with syntax errors and print statements in the first two lessons, this lesson focuses on variable tracking and run time errors by introducing a debugger.

University of Florida CS students are not taught in class how to use a debugger. This tool is extremely useful when tracking variables change over time and helps students visualize what their code is doing line by line, increasing code comprehension. It can also be much more convenient then print statements when attempting to find an error.

This assignment requires students to debug two already made methods. One method is a binary search, and the other finds the first repeated value in an array. Each method has exactly one error. The methods and the error were chosen so that students need step by step variable tracking to be found and that would be difficult using just print statements. If the students attempt to use print statements one method will give an infinite loop and the other will have an IndexOutOfBounds error. The students will be required to write down the value of each variable in a separate file each time the program loops, and to do so again after they fix the error. The methods were chosen because they both require a good deal of variable tracking and should emphasize the usefulness of a debugger. This also goes over infinite loops and array out of bounds, common programming mistakes

This exercise would be precluded by an in class demonstration on how a debugger works. A lecturer would use it in a program by setting break points and moving in and out of methods while demonstrating how the variables change. If this is done in a TA lab, the TA should check that every student can correctly use the debugger. Also, a brief presentation on a binary search works should be given.

Technology wise the students should all be using the same IDE. Visual Studios, Eclipse, and many more have Debugger capabilities.

Current Knowledge: Students should be 3-4 weeks into class and be comfortable calling methods and using print statements to test code. They should also have been given lectures on how debuggers and binary search works.

Scaffolding/ Weaknesses: Students may have trouble finding the errors in the methods. Test code and hints on how to solve each method is given in the program. Students may just rely on print statements which they are more comfortable with, so both methods make using print statements tedious. If the students still have difficulty finding the error, the solution should be given out a few days before it is due, but still require variable tracking using a debugger. Technology wise students may have trouble getting the debugger set up, so a TA should check that every student can use it correctly.

A strength of these lessons that I got from feedback was that they focused on teaching one skill at a time and made apparent how that skill was being taught. Another strength is that the students ability level at each stage was taken into consideration and appropriate scaffolding was given so that students could avoid common roadblocks and focus solely on the task at hand.

The purpose of these lessons is to increase the 70% retention rate in introductory CS courses by giving programmers the tools to test their programs and avoid unnecessary frustration and roadblocks. The lessons also aid in programming comprehension and teach lifelong skills of unit testing and tracking variables.

For now, I am content leaving the project on my github and do not plan to continue it further. If this project was to be expanded I would emphasize students creating and submitting their own test code for each assignment and test driven development. The two main papers I used both have great ideas for integrating error management into a full class and I highly encourage interested parties to read them (Edwards, 2004) (Kazemian, 2008).

Works Cited

Association of Computer Machinery. "Retention in Computer Science Undergraduate Programs in the U.S." 2018. https://www.acm.org/binaries/content/assets/education/retention-in-cs-undergrad-programs-in-the-us.pdf

Edwards, Stephen. "Using software testing to move students from trial-and-error to reflection-in-action" Virginia Polytechnic Institute and State University. 2004. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.102.4393&rep=rep1&type=pdf

Kazemian, Fereydoun and Trudy Howles. "Teaching Challenges: Testing and Debugging Skills for Novice Programmers." Rochester Institute of Technology. 2008. https://pdfs.semanticscholar.org/0126/499e56801bf7ce9b67aef45fba619912720d.pdf

Laporte, C. Y., A. April, and K. Bencherif. "Teaching software quality assurance in an undergraduate software engineering program." Software Quality Professional, 4-10. 2007.

Stringfellow, Angela "50 common Java Errors and how to avoid them." 2017. https://dzone.com/articles/50-common-java-errors-and-how-to-avoid-them-part-1