Chapter 23: Debugging

- Debugging is the process of identifying the root cause of an error and correcting it
- It contrasts with testing, as testing is the process of detecting the error initially
- "Debugging is twice as hard as writing code in the first place. Therefore, if you wrote the code as cleverly as possible, you are, by definition, not smart enough to debug it" because the "smart" code you wrote is too complex for you to understand
- Big idea is to minimize the number of things that need to be debugged

Overview of Debugging Issues

Role of Debugging in Software Quality

- Like testing, debugging isn't a way to improve quality of code
    - Rather, it's a way to diagnose effects
- Debugging is a last resort

Variations in Debugging Performance

- Studies show that experienced programmers can find bugs about 20x faster than inexperienced ones
    - Aka debugging is a skill

Defects as Opportunities

- You don't need to learn how to fix defects, you need to learn how to avoid them in the first place
- Opportunities to learn
    - Learn about the program you're working on
        - If you knew it 100% you wouldn't have made the defect
    - Learn about the kinds of mistakes you make
        - If you wrote the program, you inserted the error
        - Ask yourself how and why it was made?
    - Learn about the quality of your code from the point of view of someone who has to read it
    - Learn how you solve problems
        - Does your approach to debugging give you confidence?
        - Does it work?
        - Do you find defects quickly?
        - The inverse of these?
    - Learn about how you fix defects
        - Do you make the easiest possible corrections?
        - Do you make systematic corrections?

An Ineffective Approach

- **The Devils Guide to Debugging**
    - Lowest circle of hell is for those who don't learn to debug effectively
    - Satan tortures programmers by making them use the common debugging approaches:
        - Find the defect by guessing
            - Scattering random print statements throughout the program
        - Don't spend time trying to understand the problem
            - Assuming the problem is trivial
        - Fix the error with the most obvious fix
- Debugging by Superstition
    - Satan has leased a part of hell to programmers who debug by superstition
    - Every group of programmers has the one programmer who has endless problems with
        - Demon machines
        - Mysterious compiler defects
        - Hidden language defects that appear when the moon is full
        - Bad data
        - Losing important changes
        - A possesses editor that saves programs incorrectly
    - If you have a fault with a program you've written, its your fault
        - Not the computers fault
        - Not the compilers fault
    - Assume all problems are your fault and are fixable

Finding a Defect

The Scientific Method of Debugging

- Steps
    - Gather data through repeatable experiments
    - Form a hypothesis that accounts for the relevant data
    - Design an experiment to prove or disprove the hypothesis
    - Prove or disprove the hypothesis
    - Repeat as needed
- Effective approach to finding a defect
    - Stabilize the error
    - Locate the source of the error/fault
        - Gather the data that produces the defect
        - Analyze the data that has been gathered, and form a hypothesis about the defect
        - Determine how to prove or disprove the hypothesis, either by testing the program or by examining the code
        - Prove or disprove the hypothesis by using the procedure identified above
    - Fix the defect

    - Test the fix
    - Look for similar errors

Tips for Finding Defects

- Think of it as “triangulating the defect”
- Want test cases that pass and fail for negative examples
- Come up with lots of hypothesis
- Make notes on your debugging process
- Narrow down the suspicious region of the code
    - Don’t move haphazardly between sections
    - Think like a binary search tree
- Be suspicious of classes and routines that have had defects before
- Check code that’s changed recently
- Integrate code incrementally
    - Debugging is easy if you add pieces to the system one at a time
- Rubber duck
- Take a break from the problem
- **Brute Force Debugging**
        - A technique that might be ridiculous, but one that is **guaranteed** to work
    - Perform a full design and/or code review on the broken code
    - Throw away the section of code and redesign/recode it from scratch
    - Throw away the whole program and redesign/recode it from scratch
    - Compile the code with full debugging information
    - Compile the code at the pickiest warning level and fix all the picky compiler warnings
    - Strap on a unit test harness and test the new code in isolation
    - Create an automated testing suite and run it all night
    - Step through a big loop in the debugger manually until you get to the error condition
    - Instrument the code with print, display, or other logging statements
    - Compile the code with a different debugger
    - Compile and run the program in a different environment
    - Link or run the code against special libraries or execution environments that produce warning when code is use incorrectly
    - Replicate the end-users full machine configuration
    - Integrate new code in small pieces, fully testing each piece as its integrated
- Set a maximum time limit for the “quick and dirty” debugging session

Fixing a Defect

- The hard part of debugging is finding the defect
    - Fixing the defect is the easy part
- However, as with easy tasks, the fact its easy makes it particularly error prone
- Guidelines for reducing change of error
    - Understand the problem before you fix it
        - Really understand it to the core
    - Understand the program, not just the problem

        - Understanding global behavior leads to much better understanding than local behavior
    - Confirm the defect diagnosis
        - Do this before rushing in to fix a defect
    - DO NOT RUSH
        - “Never debug standing up” – our boi Weinberg
        - Relax long enough to be sure your solution is right
    - Save the original source code
    - Fix the problem, not the symptom
    - Rotating the tires of a car doesn’t fix engine problems
    - Before making a change, be confident it will work
        - Being wrong about a change should leave you astonished
    - Make one change at a time
    - Check your fix
    - Add a unit test that exposes the defect
    - Look for similar defects

Psychological Considerations in Debugging

Debugging Blindness

- Those stupid facebook trash posts where words are repeated, and you don’t notice it
- Avoid stuff like this with
    - Good formatting
    - Good commenting
    - Good variable names
    - Good routine names
- Also helps to slice away components that aren’t directly relevant to the debug session
- Make an effort to have variable names look very different
    - Shape
    - Letters
    - Etc

Tools

- Use diff on source code
    - Find changes you forgot you made
- Use linters
    - Good for administrative crap you can overlook
- Use test frameworks bruh
- Find an actually good debugger for real
    - Important to be able to fully examine the data
    - “Thinking isn’t a substitute for a good debugger”