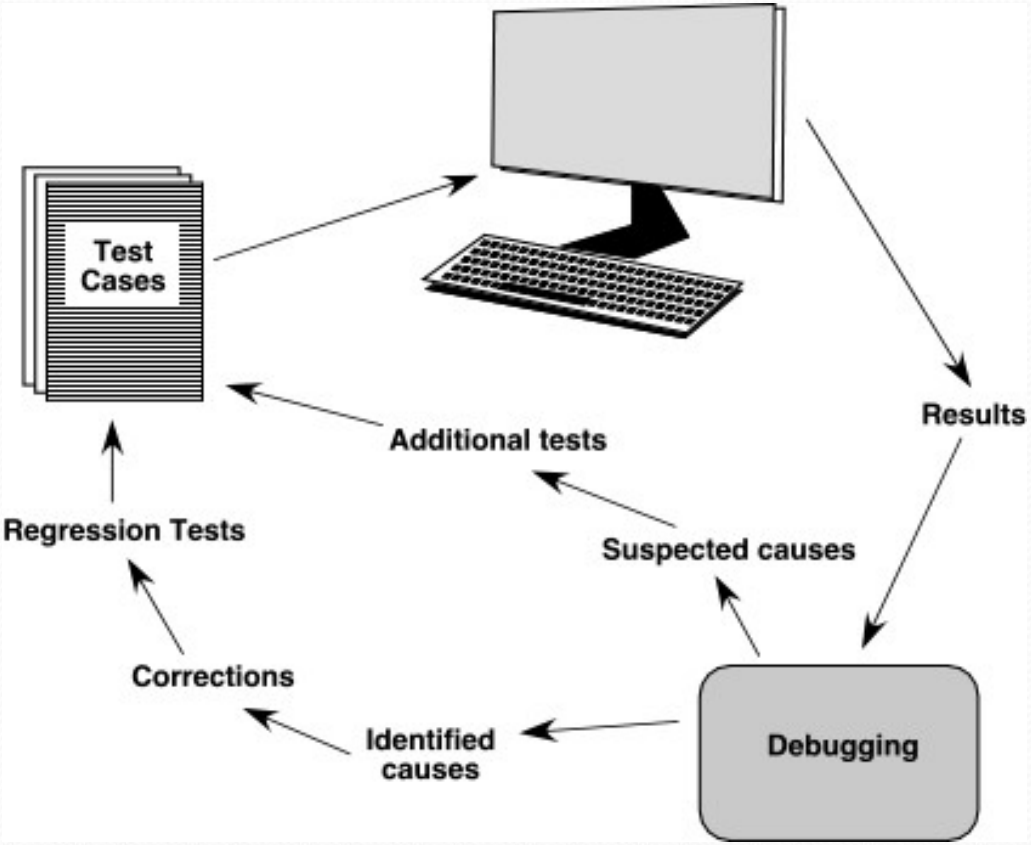
Debugging

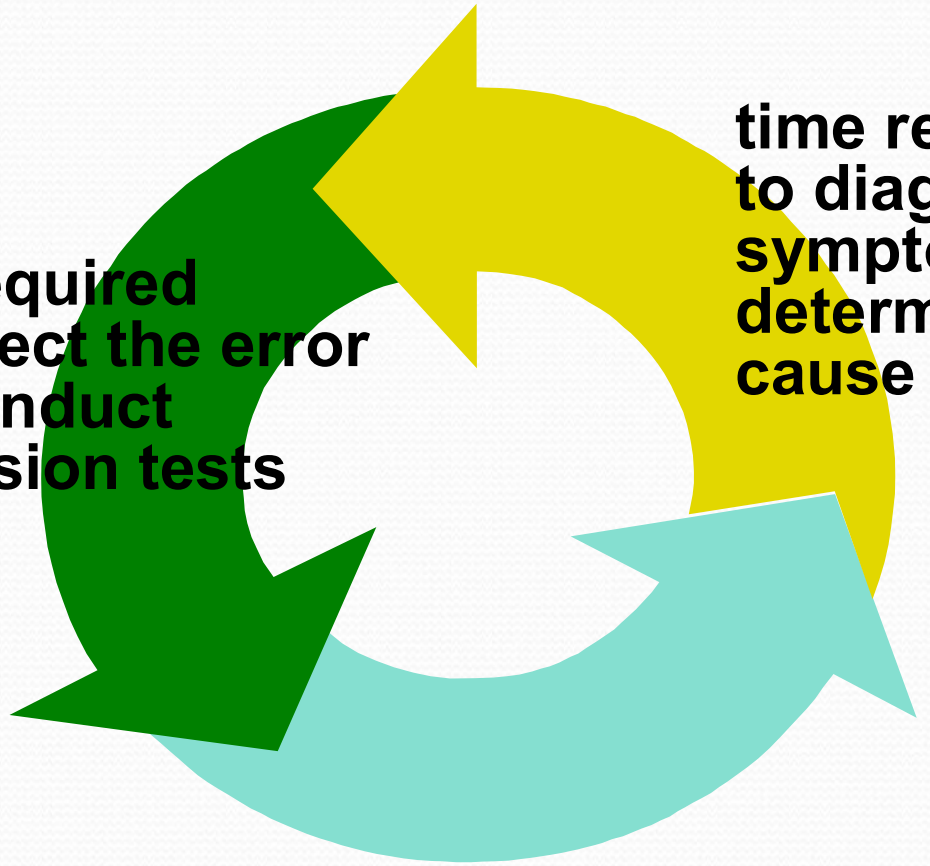
Debugging: A Diagnostic Process

These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

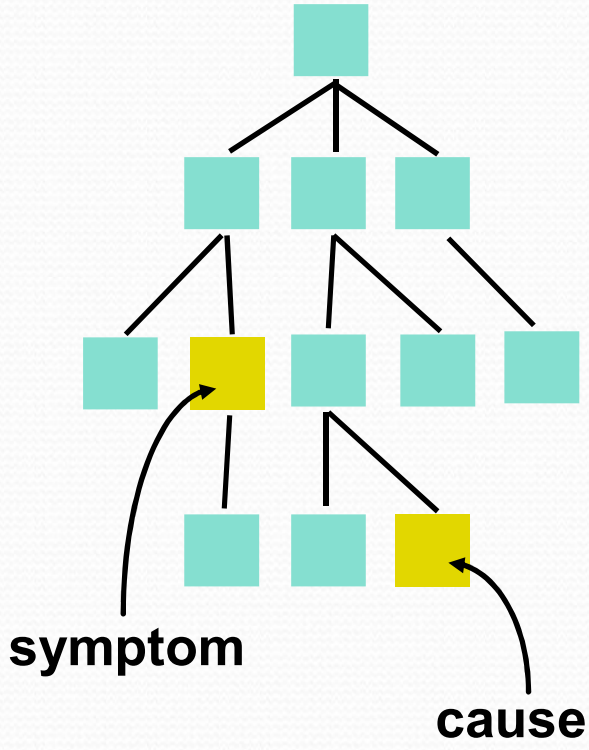
The Debugging Process



Debugging Effort

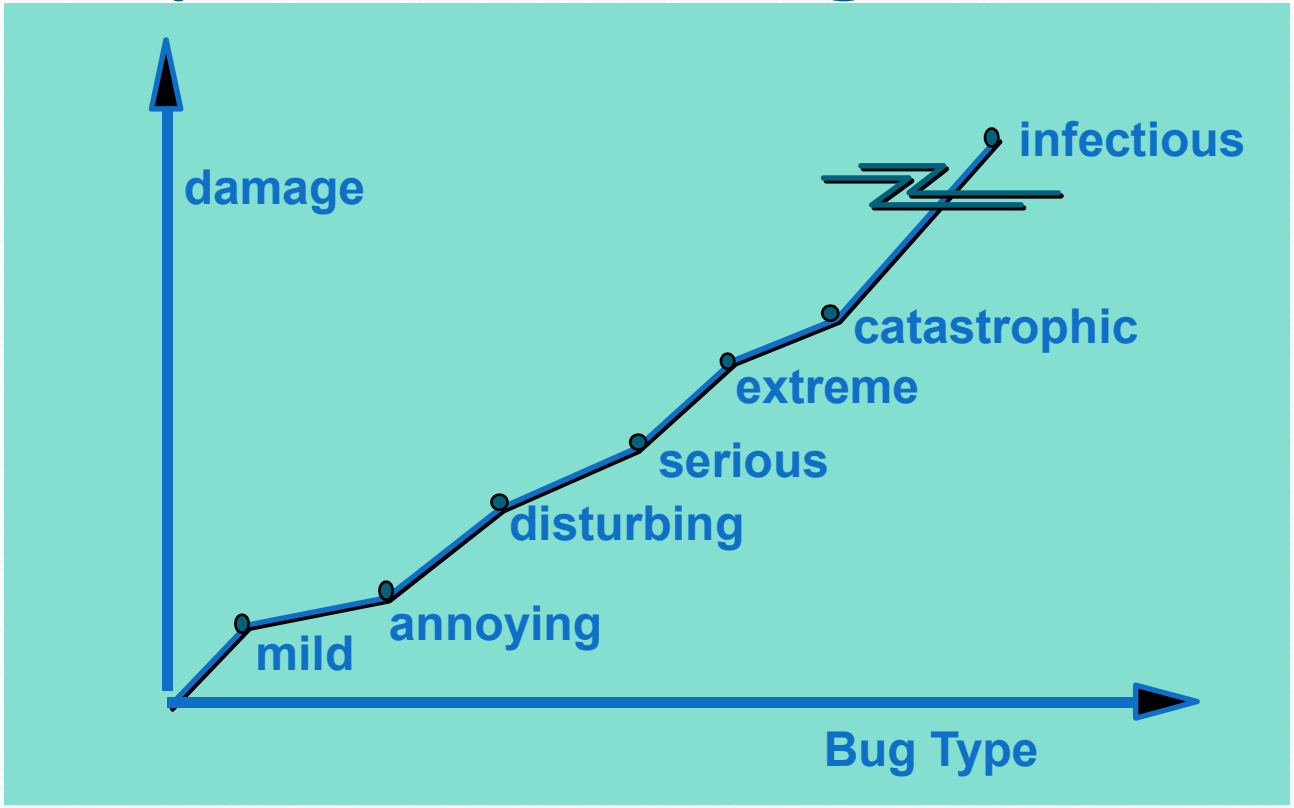


Symptoms & Causes



- symptom and cause may be geographically separated
- symptom may disappear when another problem is fixed
- cause may be due to a combination of non-errors
- cause may be due to a system or compiler error
- cause may be due to assumptions that everyone believes
- symptom may be intermittent

Consequences of Bugs



Bug Categories: **function-related bugs, system-related bugs, data bugs, coding bugs, design bugs, documentation bugs, standards violations, etc.**

Debugging Techniques

- brute force / testing
- backtracking
- induction
- deduction

Correcting the Error

- *Is the cause of the bug reproduced in another part of the program?* In many situations, a program defect is caused by an erroneous pattern of logic that may be reproduced elsewhere.
- *What "next bug" might be introduced by the fix I'm about to make?* Before the correction is made, the source code (or, better, the design) should be evaluated to assess coupling of logic and data structures.
- *What could we have done to prevent this bug in the first place?* This question is the first step toward establishing a statistical software quality assurance approach. If you correct the process as well as the product, the bug will be removed from the current program and may be eliminated from all future programs.

Bugs

- Term has been around a long time
  - Mark I – moth in machine
- Mistake made by programmers
- Also (and maybe better) called:
  - Errors
  - Defects
  - Faults

Sources of Bugs

- Bad Design
  - Wrong/incorrect solution to problem
  - From system-level to statement-level
- Insufficient Isolation
  - Changes in one area affect another
- Typos
  - Entered wrong text, chose wrong variable
- Later changes/fixes that aren't complete
  - A change in one area affects another

Debugging in Software Engineering

- Programmer speed has high correlation to debugging speed
  - Best debuggers can be more than to 10 times as fast
- Faster finding bugs
- Find more bugs
- Introduce fewer new bugs

Ways **NOT** to Debug

- Guess at what's causing it
- Don't try to understand what's causing it
- Fix the symptom instead of the cause
  - Special case code
- Blame it on someone else's code
  - Only after extensive testing/proof
- Blame it on the compiler/computer
  - Yes, it happens, but almost never is this the real cause

An Approach to Debugging

1. Stabilize the error
2. Locate the source
3. Fix the defect
4. Test the fix
5. Look for similar errors

- Goal: Figure out *why* it occurs and fix it completely

1. Stabilize the Error

- Find a simple test case to reliably produce the error
  - Narrow it to as *simple* a case as possible
- Some errors resist this
  - Failure to initialize
  - Pointer problems
  - Timing issues

1. Stabilizing the Error

- Converge on the actual (limited) error
  - Bad: “It crashes when I enter data”
  - Better: “It crashes when I enter data in non-sorted order”
  - Best: “It crashes when I enter something that needs to be first in sorted order”
- Create hypothesis for cause
  - Then test hypothesis to see if it’s accurate

2. Locate the Source

- This is where good code design helps
- Again, hypothesize where things are going wrong in code itself
  - Then, test to see if there are errors coming in there
  - Simple test cases make it easier to check

When it's Tough to Find Source

- Create multiple test cases that cause same error
  - But, from different "directions"
- Refine existing test cases to simpler ones
- Try to find source that encompasses all errors
  - Could be multiple ones, but less likely
- Brainstorm for sources, and keep list to check
- Talk to others
- Take a break

Finding Error Locations

- Process of elimination
  - Identify cases that work/failed hypotheses
  - Narrow the regions of code you need to check
  - Use unit tests to verify smaller sections
- Process of expansion:
  - Be suspicious of:
    - areas that previously had errors
    - code that changed recently
  - Expand from suspicious areas of code

Alternative to Finding Specific Source

- Brute Force Debugging
  - “Guaranteed” to find bug
  - Examples:
    - Rewrite code from scratch
    - Automated test suite
    - Full design/code review
    - Fully output step-by-step status
- Don’t spend more time trying to do a “quick” debug than it would take to brute-force it.

3. Fix the Defect

- Make sure you understand the *problem*
  - Don't fix only the symptom
    (e.g. no magic "subtract one here" fixes)
- Understand what's happening in the program, not just the place the error occurred
  - Understand interactions and dependencies
- Save the original code
  - Be able to "back out" of change

Fixing the Code

- Change only code that you have a good reason to change
  - Don't just try things till they work
- Make one change at a time

4. Check Your Fix

- After making the change, check that it works on test cases that caused errors
- Then, make sure it still works on other cases
  - Regression test
  - Add the error case to the test suite

5. Look for Similar Errors

- There's a good chance similar errors occurred in other parts of program
- *Before* moving on, think about rest of program
  - Similar routines, functions, copied code
  - Fix those areas immediately