CS 4350: Fundamentals of Software Engineering
CS 5500: Foundations of Software Engineering

Lesson 7.1 Bugs

Jon Bell, John Boyland, Mitch Wand

Khoury College of Computer Sciences

Outline of this lesson

1. How a project should handle bug reports
2. Effective debugging

Learning Objectives for this Lesson

- By the end of this lesson, you should be able to:
  - Describe best practices around bugs;
  - Distinguish the concept of where a failure is manifest versus where fault is in execution;
  - Contrast synchronic (in current code) versus diachronic (in changes) fault-locating;
  - Review the basic operations provided by debuggers.

What to Do When a Bug is Reported

- Log the issue in a tracking system
  - Many possibilities: github issues, Bugzilla, gerrit, … .
  - Connect to symptoms, tests, causes and fixes.
  - User-searchable helps prevent duplicate/invalid reports.
  - In an open-source project, recent activity gives confidence to users that the project is still live.

First Steps

- Assign a priority
  - How crucial it is to fix the problem.
- Assign a responsible party
  - Not blame: rather who will work on it first.
- Try to find a Short Self-Contained Correct Example

  S: remove everything that is not necessary;

  S-C: add everything needed to demonstrate problem;

  C: the example is valid;

  E: a concrete example of bug behavior.

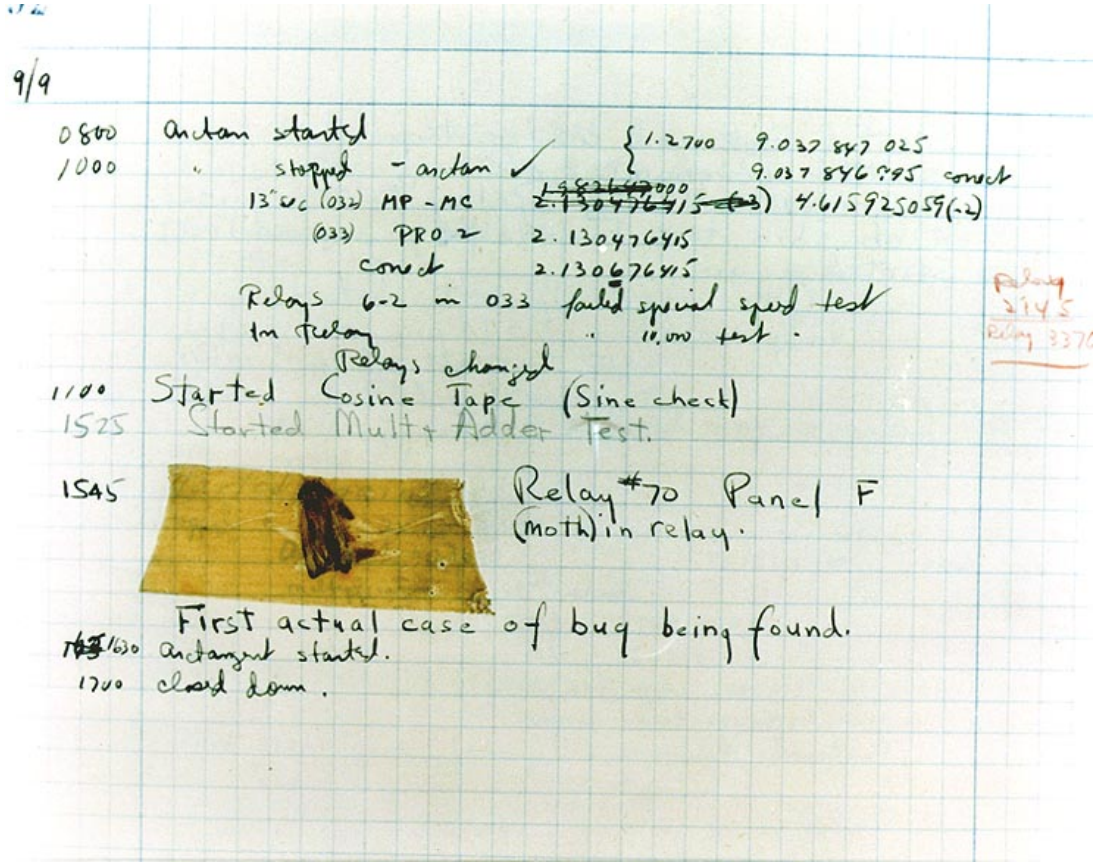
  (Ideally reporter gives one or gets close.)

Assignee tasks

- The assignee needs to investigate the report to see if it is actionable, or else it is …
  - Invalid: not relevant to the software
  - Duplicate: already reported
  - Fixed: in a recent version
  - Can’t reproduce: back to reporter for more data
- Otherwise, if the bug is reproducible:
  - Complete SSCCE if not already done.
  - Convert SSCCE into a (failing) test:
    - Small scope and automated, if possible.

You may recall this step from TDD

“Debugging”



- The term “bug” (or a “glitch”) dates at least back to Thomas Edison in the 19th century.
- ”Debugging” also predates the famous bug found in Mark II, as related by Admiral Grace Hopper.

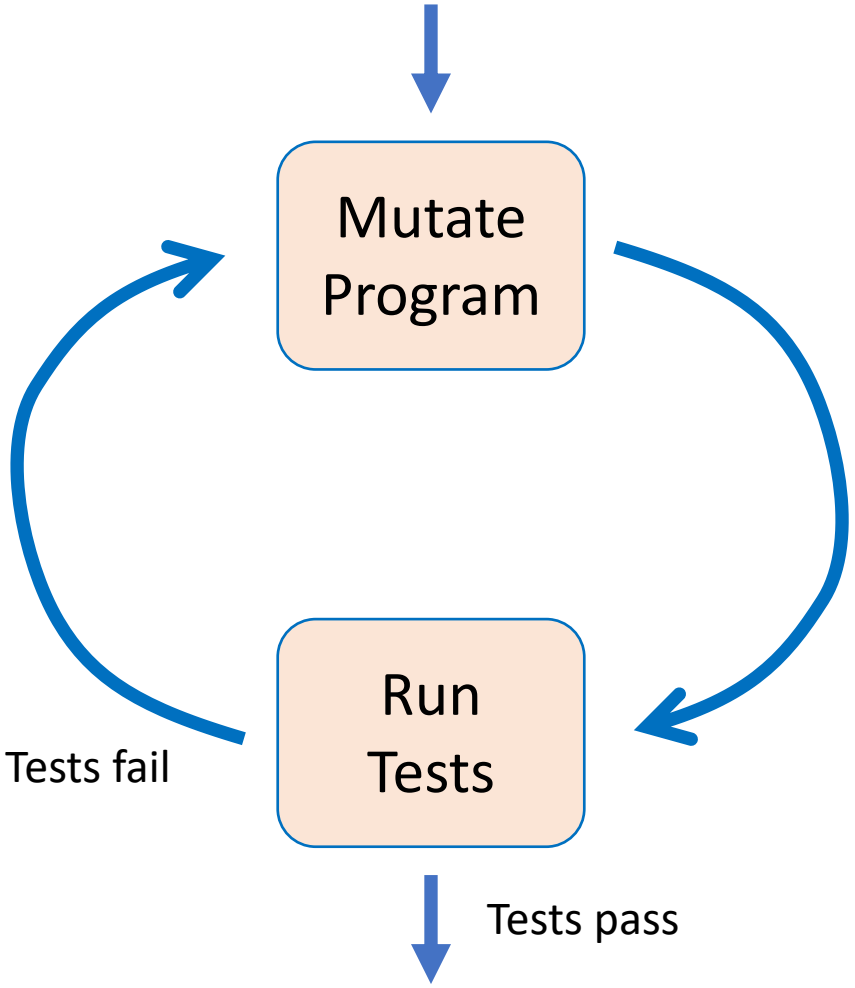
Debugging has Two Phases

- Diagnosis
  - Determine the fault causing the problem observed.
  - For example:
    - Faulty computation;
    - Incorrect assumption;
    - Mis-use of another component.
- Correction
  - Correct the fault so that test passes (and none fail).
  - For example:
    - Fix computation;
    - Add code to handle unhandled cases.

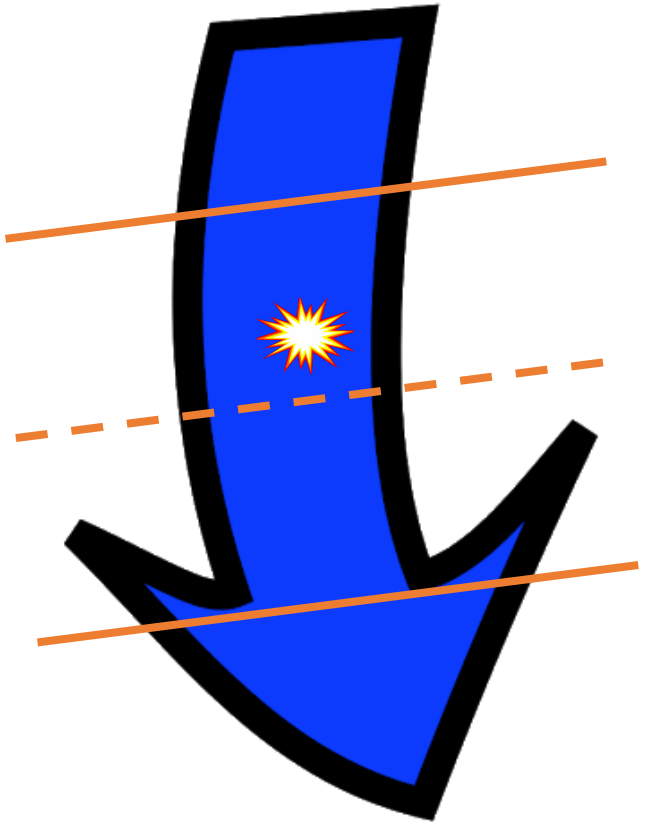
Basically, coding

Anti-Pattern: Mutation Debugging

- Avoiding diagnosis is a mistake.
  - If you don't understand what's going wrong, how can you fix it?
- Mutation debugging:
  - Make a change to the program
  - Re-run all tests.
  - Repeat unless all pass.
- A (poor) version of genetic programming, with similar (unmaintainable) results.



(Synchronic) Fault Location



- Initially, fault must lie between program start and failure point.
- Use tools to narrow dynamic scope.
- Iterate until fault located.
- ”Wolf Fence” technique of Edward Gauss

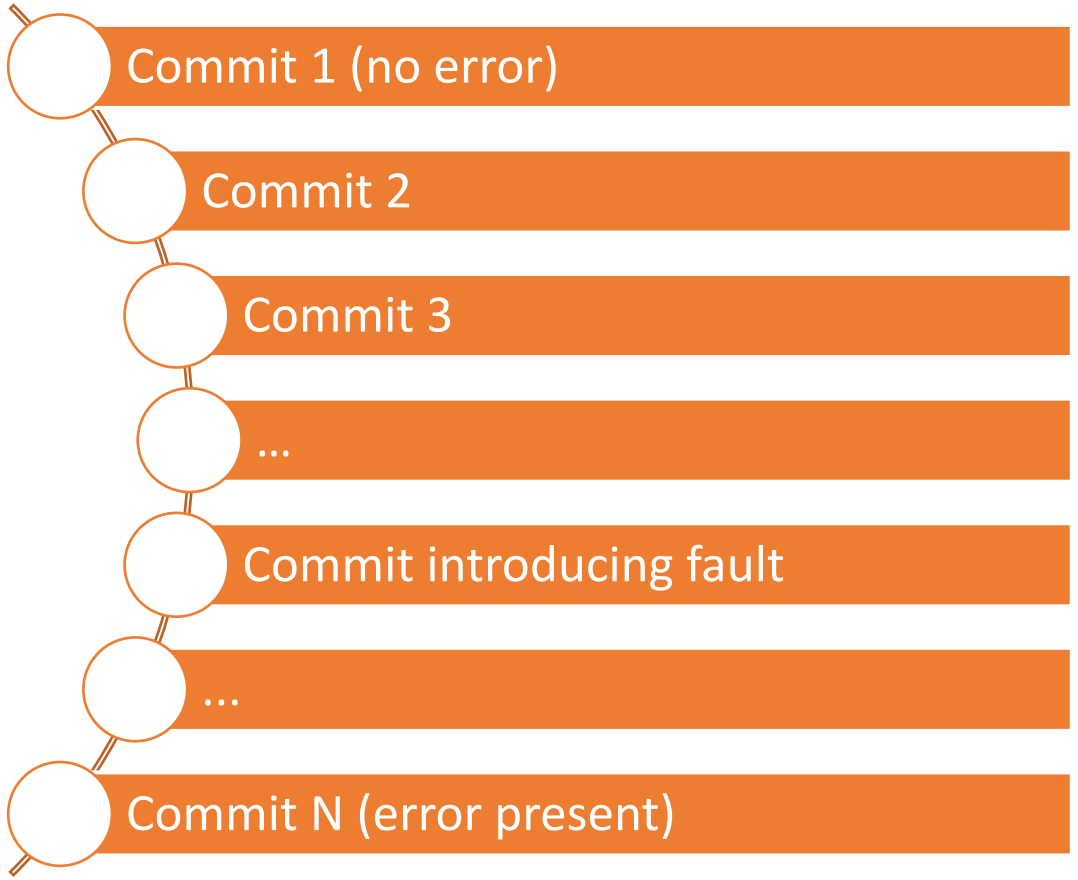
A form of Binary Search

Diachronic Fault Location

- If the fault was not present in an earlier version, we can use *diachronic fault location* to find when it was introduced.
- Same binary search approach, but this time over the versions of software.

```
git bisect
```

- This uses an automated test to indicate presence.



Debugging as Hypothesis Testing

- Guessing a diagnosis:
  - Experience (pattern matching);
  - Community (colleagues, stackoverflow.com);
  - Information from faulty run.
- Verbalize hypothesis:
  - Good for bug reporting system;
  - Direct debugging actions.
- Testing a diagnosis:
  - Use debugging techniques to test;
  - Better once fault located to a narrow area.

Using Debugging Tools

- Techniques:
  - Print statements,
  - Special-purpose code changes (e.g., assertions),
  - Breakpoints,
  - Stepping,
  - Watchpoints.
- Use tools for a purpose:
  - Fencing the wolf;
  - Testing the hypothesis.

Assisting Future Debugging

- Assisting fault location:
  - “fail-fast”: stop as soon as problem noticed;
  - Check invariants frequently during testing.
- Assisting hypothesis testing:
  - Make assumptions explicit (and checked!);
  - Log interactions with other components.

Review: Learning Objectives for this Lesson

- You should now be able to:
  - Describe best practices around bugs;
  - Distinguish the concept of where a failure is manifest versus where fault is in execution;
  - Contrast synchronic (in current code) versus diachronic (in changes) fault-locating;
  - Review the basic operations provided by debuggers.

Next steps...

- In our next lesson, we'll discuss using git in teams.