

Debugging an Unfamiliar Codebase

COMP23311: Software Engineering

Week 3

Anas Elhag

Course Unit Roadmap

Skills for Small Scale Code Changes

Git basics
Automated build
Automated test
Code reading
Debugging

Working with Features

Software estimation
Git workflows
Test-first development
Code review

Larger-Scale Change

Design for Testability
Refactoring
Feature migration
Design patterns

Open Source Challenge

1

2

3

4

5

6

7

8

9

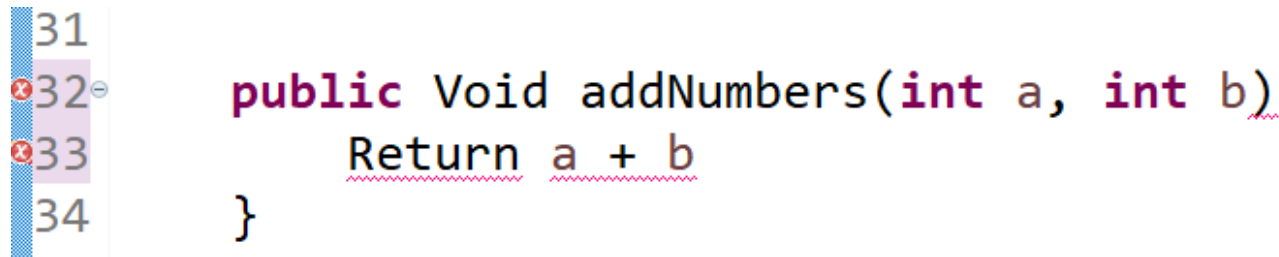
10

11

12

Syntactical Errors

- The language used to create the code is incorrect
- Capitalizing keywords, Missing parentheses, missing semicolons, forgetting to import a type, etc.
- Easy to deal with (most of the time).
- Advanced IDEs complain and won't compile.



The screenshot shows a code editor with line numbers 31 to 34 on the left. The code is as follows:

```
31  
32 public Void addNumbers(int a, int b)  
33     Return a + b  
34 }
```

There are three red 'x' marks indicating errors: one on line 32 before 'public', one on line 32 before 'Void', and one on line 33 before 'Return'. The code is syntactically incorrect due to these errors.

- How many syntactical error can you spot in this simple example?

Conceptual Errors (Bugs)

- The language used to create the code is correct
- There are no syntactical errors.
- The IDE is happy, and is able to compile the code.
- Logical errors: the code does not work as intended.
- Harder to spot (especially in large codebases).

```
31  
32 public int addNumbers(int a, int b) {  
33     return a - b;  
34 }
```

- This code is buggy. Can you tell why?

Software Bugs

- Refer to faults, errors or flaws in a codebase.
- Could be in the *source code* or in the *system design*.
- Lead to incorrect or unexpected results or behaviour.
- Origins of the term:
“The first actual case of bug being found”

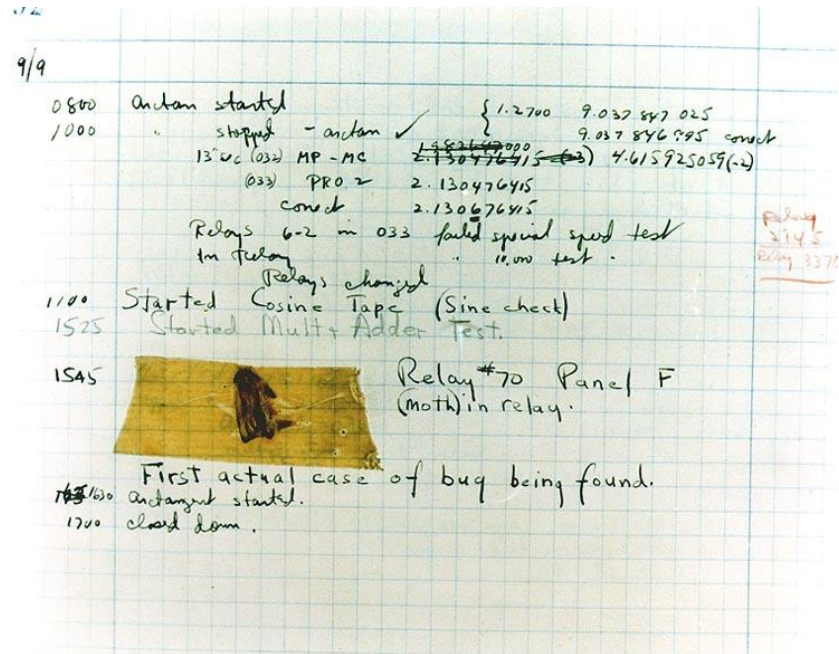


Image adapted from a U.S. Naval Historical Center
Online Library Photograph

Software Debugging

- The practise of *discovering, locating and fixing* bugs.
- Uses systematic approaches and formal techniques.

*“As soon as we started programming, we found to our surprise that **it wasn't as easy to get programs right as we had thought. Debugging had to be discovered.** I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs.”*

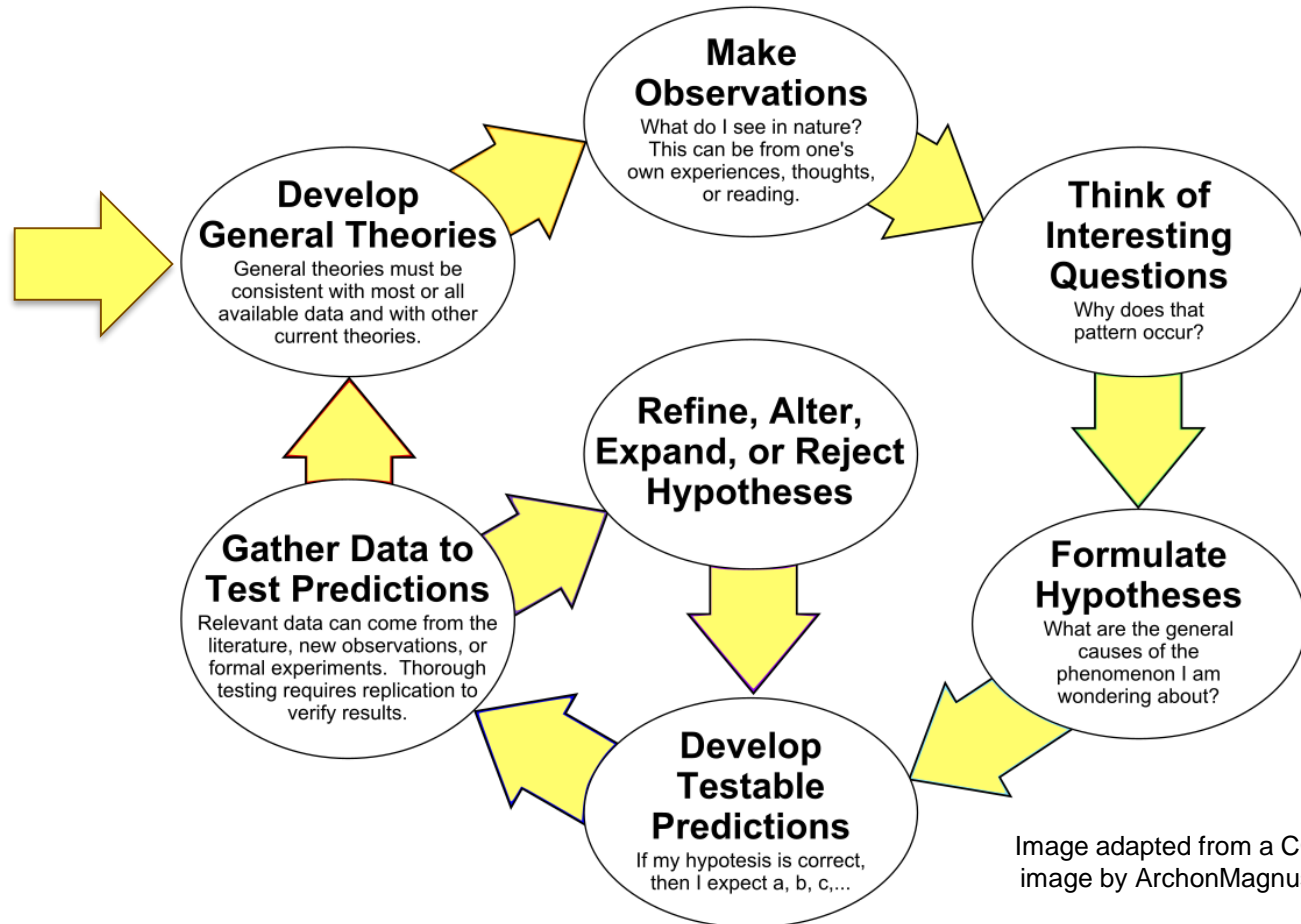
- Maurice Wilkes, 1949



Image adapted from a CC photo
from the University of Cambridge
Computer Laboratory Archive

Science or Art?

- Debugging is a skill that improves with experience.



A Systematic Approach

- Uses systematic approaches and formal techniques.
 1. Start with a problem.
 2. Stabilise the problem.
 3. Isolate the source of the problem.
 4. Fix the problem.
 5. Test the fix.
 6. Look for similar errors.

1- Start with a Problem

Confirm the reported problem by replicating it:

- Make sure you understand the expected correct behaviour of the system.
- Run the code to (using the same input data and conditions that created the reported problem).
- Observe the actual behaviour.

If the bug cannot be replicated:

- Make sure that you are running the code under the same conditions that created the bug.
- Request more info / discuss with the person who reported it.

2- Stabilise the Problem

Narrow the problem to as precise a statement as is possible.

- “The *save as* feature of this application does not work”.
 - “The *save as* feature causes the application to crash when the specified filename is long.”
 - “The *save as* feature causes the application to crash when the base name plus full specified file path exceeds 250 characters and the file does not already exist. In addition to crashing the application, the intended save operation is not achieved (that is, no new file is created).”
-
- Initially, this may be achieved using execution only.
 - Use test cases where appropriate

3- Isolate the Source of the Problem

Strategies:

- Brute force : work through the code from start to finish until you notice something odd.
- Back-tracking: start from the point at which a particular problem is detected and work backwards until you find the cause.
- Binary search: approximate the “middle” of the execution flow for this problem. Can you determine whether the error is manifest at that point? If so, find the middle of the first half of execution; if not find the middle of the second half — repeat.
- Cause elimination: identify as many possible causes of the problem as possible. Work through them systematically, only ruling one out when you can prove that it must not be the cause.

3- Isolate the Source of the Problem

Tactics:

- Print statements/console output.
- Rubber Ducking: just explain exactly what the code is doing to someone else (e.g. a rubber duck).
- Look for Common Problems: logic inversion, boundary errors, poor code isolation, etc.
- IDE debugging tools: break points.
- Test cases: Instrument any code you're suspicious of with tests.
- Git commit logs: Look at what's changed recently. Which commits altered which lines of each file.

3- Isolate the Source of the Problem

Use a combination: a strategy + a tactic:

- Cause elimination + tests - For each possible cause, write a test. Tests that pass rule out causes.
- Cause elimination + online debugger: - Step through the code associated with each possible cause — is the state correct or incorrect?
- Cause elimination + print statements - For each possible cause, write a print statement before/after — is the state correct or incorrect?
- Binary search + print statements - Print the state out at the mid-point of execution, mid-way through the first half of the execution flow, midway through the second quarter of the execution flow etc.

4- Fix the Problem

- This is usually the easy part (compared to finding the problem).
- Double check with a teammate to make sure that your work is correct.

5- Test the Fix

- Just run the tests you already have!

6- Look for Similar Errors

- Find at least one other occurrence of the method you've modified being called. Can you prove that this code behaves as intended?