School of Computing & Communications | Lancaster University

SCC.111 Software Development – Lecture 24: Runtime Code Inspection and Debugging

Adrian Friday, Hansi Hettiarachchi and Nigel Davies

Introduction

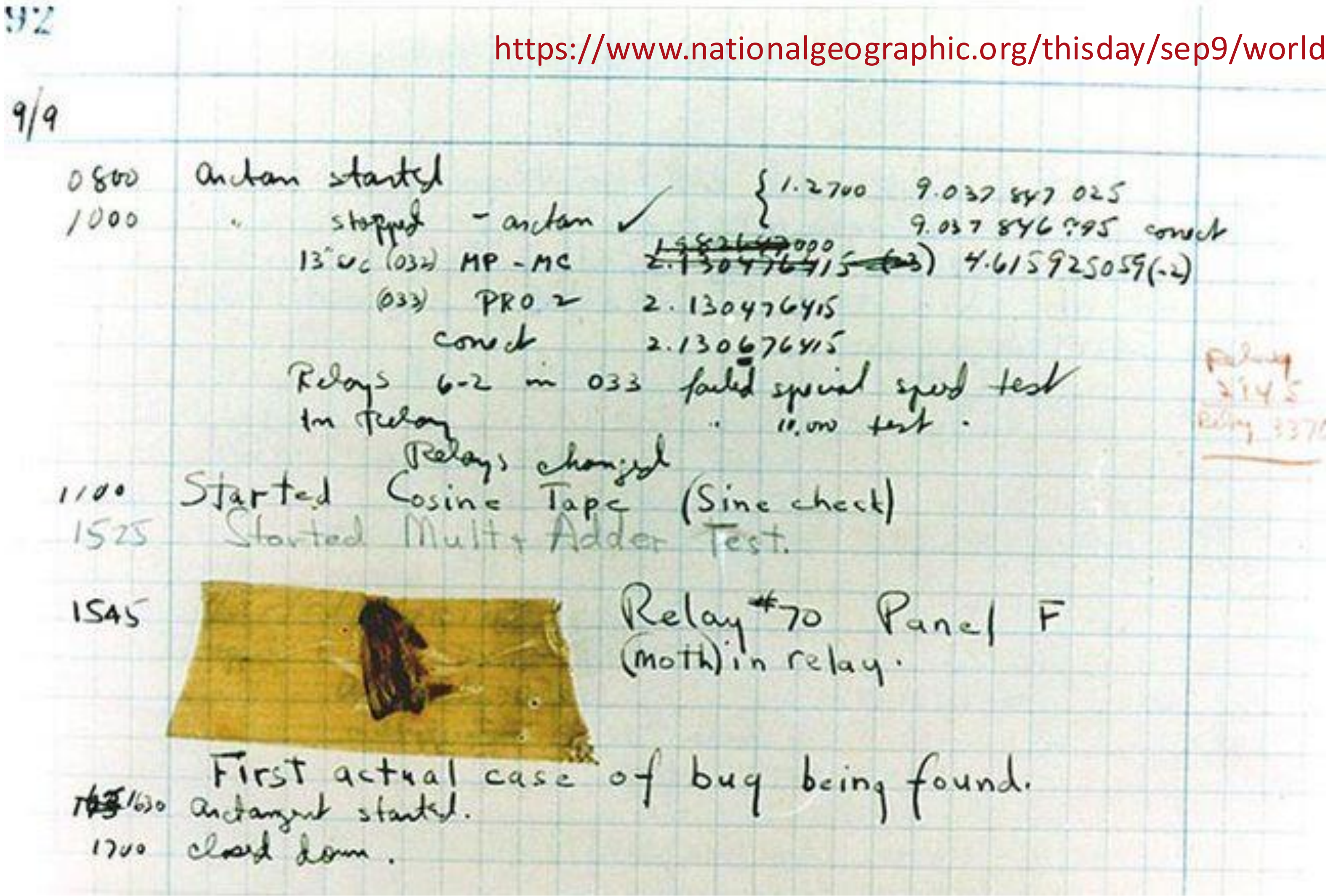
- Last lecture, we looked at:
  - Encapsulation: How to protect and initialize your objects in C++
  - Some worked examples

- Today we’re going to explore how to debug your programs!
  - Bugs in code are inevitable
  - Identifying and correcting bugs is standard practice
  - There are techniques to help find and fix them

The first ‘bug’

Photo # NH 96566-KN (Color) First Computer “Bug”, 1947

https://www.nationalgeographic.org/thisday/sep9/worlds-first-computer-bug/



92

9/9

0800 Anctan started

1000 stopped - anctan ✓ {1.2700 9.037 847 025

9.037 846 995 conct

13" uc (032) MP - MC ~~1.982147000~~ ~~2.130476415~~ (-3) 4.615925059(-2)

(033) PRO 2 2.130476415

conct 2.130676415

Relays 6-2 in 033 failed special speed test

In relay . 11.000 test .

Relays changed

Relay 2145

Relay 3370

1100 Started Cosine Tape (Sine check)

1525 Started Mult + Adder Test.

1545 Relay #70 Panel F

(moth) in relay.

First actual case of bug being found.

1630 Antangent started.

1700 closed down.

> “Everyone knows that debugging is twice as hard as writing a program in the first place. So if you're as clever as you can be when you write it, how will you ever debug it?”

- Brian Kernighan, *The Elements of Programming Style*, 2nd edition, chapter 2

“Generative AI tools will mean that typical code written by humans will be harder to write, and they will spend a larger proportion of their time analysing and debugging.”

- Joe Finney (2023)

“You can't win until you're not afraid to lose”

- John Francis Bongiovi (2000)

Fail fast, fail often.

- Never be afraid of failure
  - **Finding bugs is a good thing**
  - Compile often. Test frequently.
  - Add one feature at a time

- A compiler is your **friend** when it comes to syntax errors.
  - **Compiler messages tell you the file and line of code where the syntax error is found**

- **Debugging is like finding a needle in a haystack… so don’t add more hay until you know there is no needle**

Code inspection

- Read through the code you have written. Be thorough.
  - **Dry run it in your head before you execute the code.**
  - One of the cheapest approaches (in terms of your time)
  - But does rely on experience (you get better with practice)

- Start at the beginning of the method where the program goes wrong
  - **Another good reason to keep your code modular.**

- If you can't see anything obvious after about 3-5 minutes, then **stop and move on to a more systematic approach.**

Know when you are data deficient

- If you can’t find a bug by code inspection, then you **don’t have enough knowledge to fix it**
  - Stop trying to fix the problem
  - Start trying to identify the problem

- Start logging diagnostic data about your program
  - **Use printf()**
  - Entry/exit of methods, loops, conditionals
  - Output the value of key variables: parameters, loop conditions, return values…

- If your code is complex, consider making debug code more permanent so it can be reused whenever you need it.

Runtime Debuggers

- Debuggers allow you to visualize what your program is doing.
  - See each line of code being executed in real-time
  - See the values change variables as your program is running
  - VS Code has a great debugger… **Install the C++ extensions**

  - Programs normally run much too fast for humans to observe
  - **Breakpoints can be added to programs**
  - Breakpoints pause the program in the debugger when a specified line of code is reached. You can have many breakpoints at the same time.

Runtime Debugging in VS Code

Breakpoints, single stepping, variable inspection, stack frames.

Reproducibility

- Some bugs are transient
  - **Never ignore them and hope they don't come back. They will.**

- Stop trying to fix the problem
  - **Focus on creating a reproducible test case**
  - Document everything you do
  - Github issues are perfect for this…
    - https://github.com/lancaster-university/codal-microbit-v2/issues/102

Thinking outside the box

- There are many external factors that can cause problems
  - Is the file you are editing the same as the one you are testing?
  - Have you compiled all your C++ files? Are there any files missing?
  - **Case sensitivity is important in most languages**
  - But some file systems are not case sensitive e.g. Windows…
  - **Did you know your H: drive is a Windows file system?**

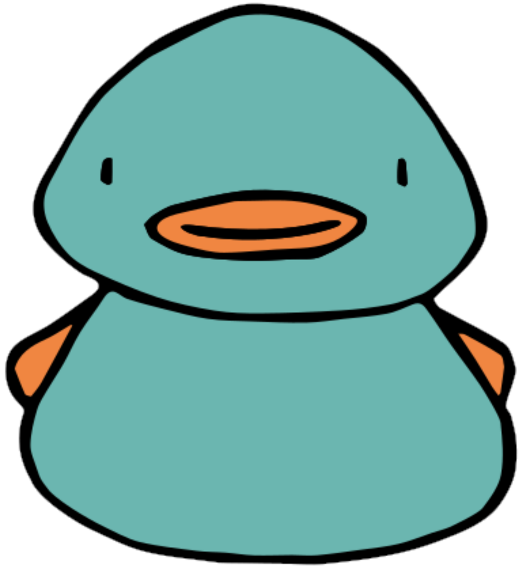
  - **Time can even be a factor.** Some of the hardest bugs to fix relate to temporal anomalies.
  - Are you providing the same data input to your program?
  - In the same order?

Rubber duck debugging

- Explaining the problem and code to yourself out loud
  - Duck can be a real duck (or any inanimate object) that you talk to
  - The act of talking about it often helps your brain think differently

- But be aware that you will have unconscious bias
  - **If you wrote your code then you will likely think it is correct**

- **Talk the behaviour through with another developer in your team**

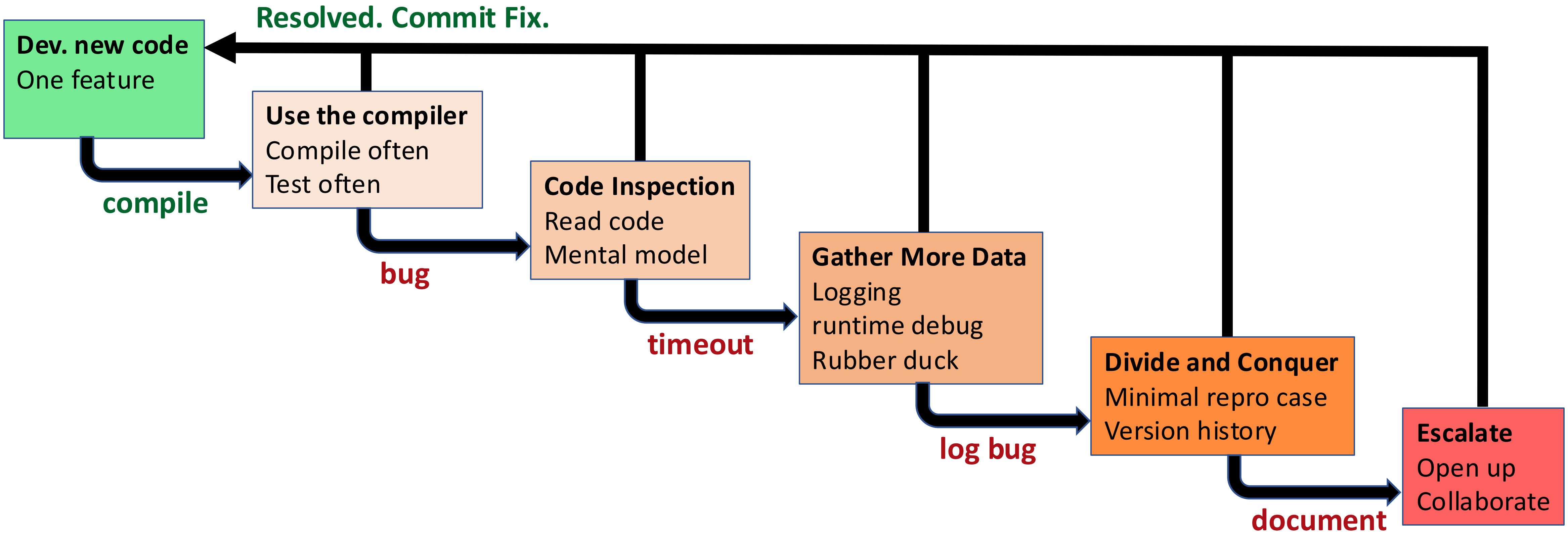


Divide and Conquer: Space and Time

- Reduce the places where the bug can hide
  - **Create a minimal reproducible to test case**
  - Remove as much unnecessary code as possible, whilst still demonstrating the bug

  - **Review GitHub commits from you and your team**
  - Identify changes to relevant parts of the codebase
  - Revert to earlier version of the code. Isolate which commit introduced the bug.
  - Look at the diff from that commit

Typical Debugging Workflow



Resolved. Commit Fix.

Dev. new code
One feature

compile

Use the compiler
Compile often
Test often

bug

Code Inspection
Read code
Mental model

timeout

Gather More Data
Logging
runtime debug
Rubber duck

log bug

Divide and Conquer
Minimal repro case
Version history

document

Escalate
Open up
Collaborate

Summary

- In today’s lecture…
  - We’ve explored structured ways of debugging
  - We explored debuggers in more detail to allow us to step through our code and inspect it
  - We saw further examples of why modular programming and version control helps us create better software