UNIVERSITEIT VAN AMSTERDAM

Debugging and Testing

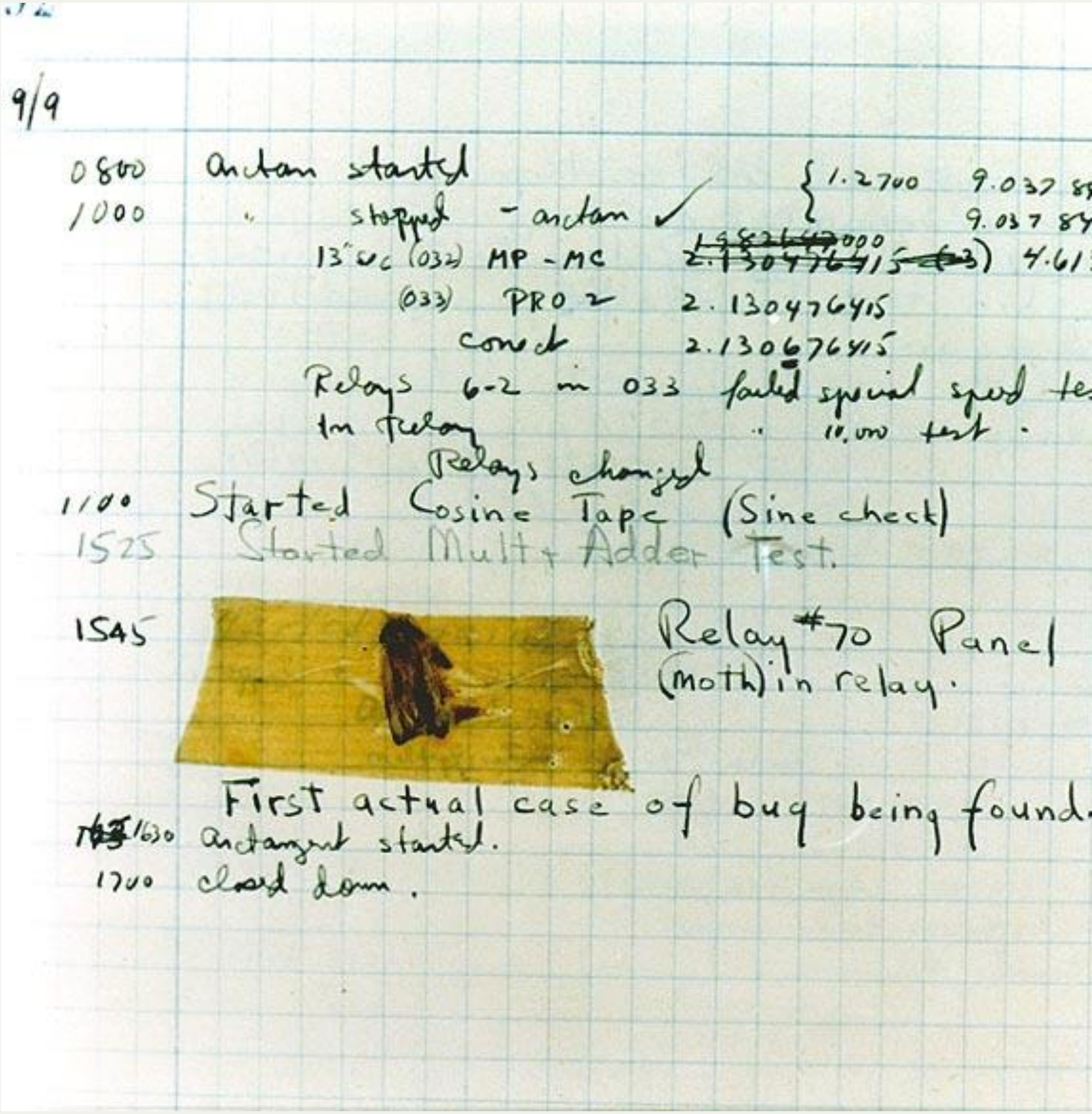
Software Construction 2018

Dr. Vadim Zaytsev aka @grammarware

raincode LABS compiler experts

What is a bug?

- Incorrect program code
- Incorrect program state
- Incorrect program execution



Andreas Zeller, *Why Programs Fail: A Guide to Systematic Debugging*, 2009.

What is a bug?

- Incorrect program code is a **defect**
- Incorrect program state is an **infection**
- Incorrect program execution is a **failure**

Andreas Zeller, *Why Programs Fail: A Guide to Systematic Debugging*, 2009.

From defects to failures

- The programmer creates a defect
- The defect causes an infection
- The infection propagates
- The infection causes a failure

Andreas Zeller, *Why Programs Fail: A Guide to Systematic Debugging*, 2009.

From failures to fixes

- Track the problem
- Reproduce the failure
- Automate and simplify the test case
- Find possible infection origins
- Focus on the most likely origins
- Isolate the infection chain
- Correct the defect

Andreas Zeller, *Why Programs Fail: A Guide to Systematic Debugging*, 2009.

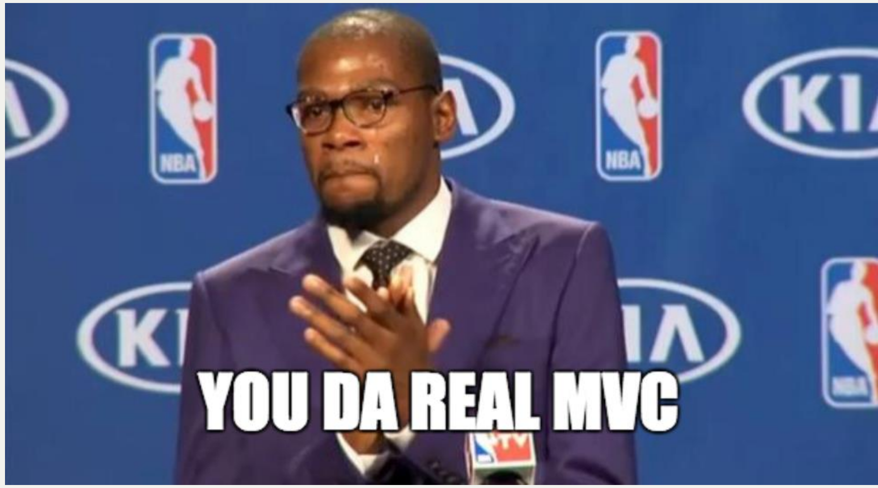
Step 1: Track the problem

- The user informs the vendor
- The vendor reproduces the problem
- The vendor isolates the circumstances
- The vendor locates and fixes the defect locally
- The vendor delivers the fix to the user

Andreas Zeller, *Why Programs Fail: A Guide to Systematic Debugging*, 2009.

Step 2: Reproduce the failure

- Create a test
  - layers: presentation, functionality, unit
- Software can be designed to be debugged
  - cf. MVC
- Reasons
  - observe the problem & check that it's fixed
- Reproduce
  - program environment & problem history



Andreas Zeller, *Why Programs Fail: A Guide to Systematic Debugging*, 2009.

Step 3: Automate and simplify

- A simplified test case facilitates debugging
  - easy to communicate, identify duplicates, etc
- Manual simplification
  - binary search
- Automatic simplification
  - automated tests + search strategy
- To speed up
  - use caching, stop early, at syntactic/semantic level, isolate differences

Andreas Zeller, *Why Programs Fail: A Guide to Systematic Debugging*, 2009.

Step 4: Find origins

- Isolate value origins
- Understand control flow
- Track dependencies
- Slice programs (also dynamically)
- Detect code smells
- Explore execution history (logs)
- Track down dependencies from the infected value

Andreas Zeller, *Why Programs Fail: A Guide to Systematic Debugging*, 2009.

Step 5: Focus on likely origins

- Known infections
  - assertions, invariants, correctness, specifications, proof obligations
- Causes in state/code/input [suggest fixes]
  - det.replay, test gen, isolation, relating causes to errors
- Anomalies
  - coverage comparison, statistical debugging, data monitoring, inferring invariants
- Code smells
  - like conventions & styles, but bad
- Earlier defect sources
  - mining authors/modules/times, during spec/dev/qa, predicting

Andreas Zeller, *Why Programs Fail: A Guide to Systematic Debugging*, 2009.

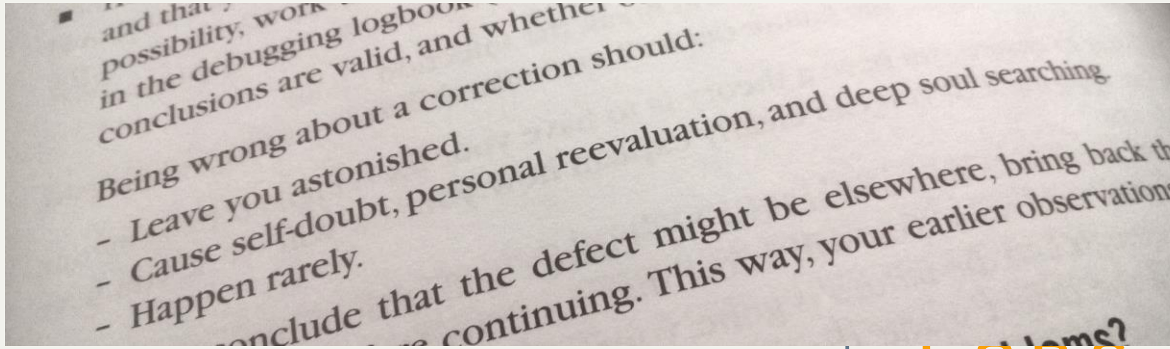
Step 6: Isolate the infection chain

- A failing program is a natural phenomenon
- Observe a failure
- Formulate a hypothesis
- Use it to make predictions
- Make experiments to test it
- Further observations
- Automate!
- Experimentation < Induction < Observation < Deduction

Andreas Zeller, *Why Programs Fail: A Guide to Systematic Debugging*, 2009.

Step 7: Correct the defect

- Think before you code
  - your change retrospectively validates causality
- Does the failure no longer occur?
- Did the correction introduce new problems?
- Was the same mistake made elsewhere?
- Track / integrate / patch / …
- Workarounds


possibility, work
in the debugging logbook
conclusions are valid, and whether
Being wrong about a correction should:
- Leave you astonished.
- Cause self-doubt, personal reevaluation, and deep soul searching.
- Happen rarely.
nclude that the defect might be elsewhere, bring back th
continuing. This way, your earlier observation

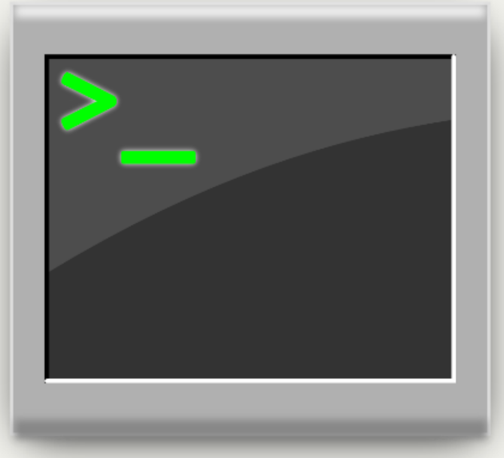
Andreas Zeller, *Why Programs Fail: A Guide to Systematic Debugging*, 2009.

Automated debugging is good!

- Simplified input / delta debugging
- Program slicing
- Observing state
- Watching state
- Assertions
- Anomalies
- Cause-effect chains
- Learning from defect density

Andreas Zeller, *Why Programs Fail: A Guide to Systematic Debugging*, 2009.

Conclusion



- Debugging can be systematic
- Track your issues
- Automate whenever possible
- Work with simple test cases in your environment
- Scan infection origins
- Use the scientific method on failing programs
- Correct the defect and issue a fix
- https://eu.udacity.com/course/software-debugging--cs259