Debugging

SWE 261P

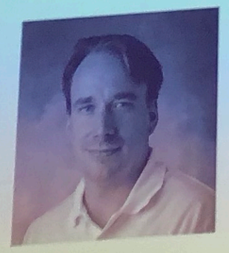
Material partially adapted from Cornell

The truth of today's debugging, as per the experts…

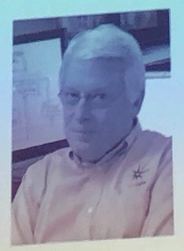
"On The Dichotomy of Debugging Behavior Among Programmers" — Beller, et al.

What do these nerds have in common?

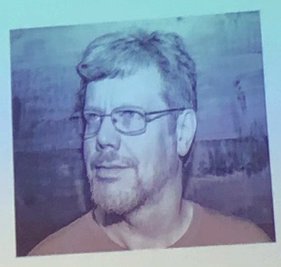
They all dislike debugging.



Linus Torvalds does not use debuggers.



Robert C. Martin thinks that debuggers are a wasteful timesink.



Guido van Rossum uses printf for 90% of his debugging.

Thanks, Daniel Lemire! https://lemire.me/blog/2016/06/21/i-do-not-use-a-debugger/

RQ1 What do developers know about debugging?

81% of developers **use the IDE debugger.**
Only **8.5% explicitly do not.**

72% of developers **examine log files** and **print statements.**

Testing is **integral part** of the debugging process.

Experience has no impact on self-reported **debugging knowledge.**

Debugging Strategies

Incremental Development

- Implement the program incrementally
- Unit test it thoroughly
- If a piece that was just written contains a bug, it is likely to be found quickly and early
- Each portion of code (i.e., a method) can be small and thus less likely to contain bugs and they may be easier to identify, understand, and debug

Instrument Program to Log Information

- Print statements!
- Or, better yet, use logging frameworks
- Remember the “visibility” principle
- May need additional tools to help filter the output

Instrument Program with Assertions

- Assertions check if expectations are met within the code
- These are often of the “sanity check”-kind of nature — it ***should*** be true, but let’s just check… just in case
- In many languages there is built-in “assert” support, such that all assertions can be enabled or disabled
- For example, in development assertions are enabled, but during deployment (i.e., customers are using it), they are disabled

Use Debuggers

- Using a “symbolic debugger” (e.g., those that are built into your IDE)
  - Place breakpoints
  - Watch variable values
  - Watchpoints or conditional breakpoints
  - Can replace the need for print statements or logging

Backtracking

- Start where you find a symptom of the problem
- Set a breakpoint before that, and try to incrementally walk backward on the execution trace until you reach the bug that caused the manifestation (observed symptom)
- Reverse debuggers are just starting to become possible (but they are typically computationally expensive)

Binary Search

- Divide and conquer: either in the code or in the test inputs
- For example, can comment out part of the code and see if the problem persists
- Or, can give part of the input
- Try to find the root of the cause of the problem

Form Hypotheses and Test Them

- You think you know what is happening, but you might be wrong
- Ensure that what you think is happening, IS happening
- Also, ensure what you think is NOT happening, IS NOT
- Brainstorm what you think could be going wrong, and those

The Future?

- Greater use and more sophisticated reverse debuggers
- Coverage-based fault localization / Spectra-based fault localization
- New visualization and diagnostic tools, yet to be invented…

Breakpoints Demo

&

Tarantula Demo