Bugs in Large Software Systems

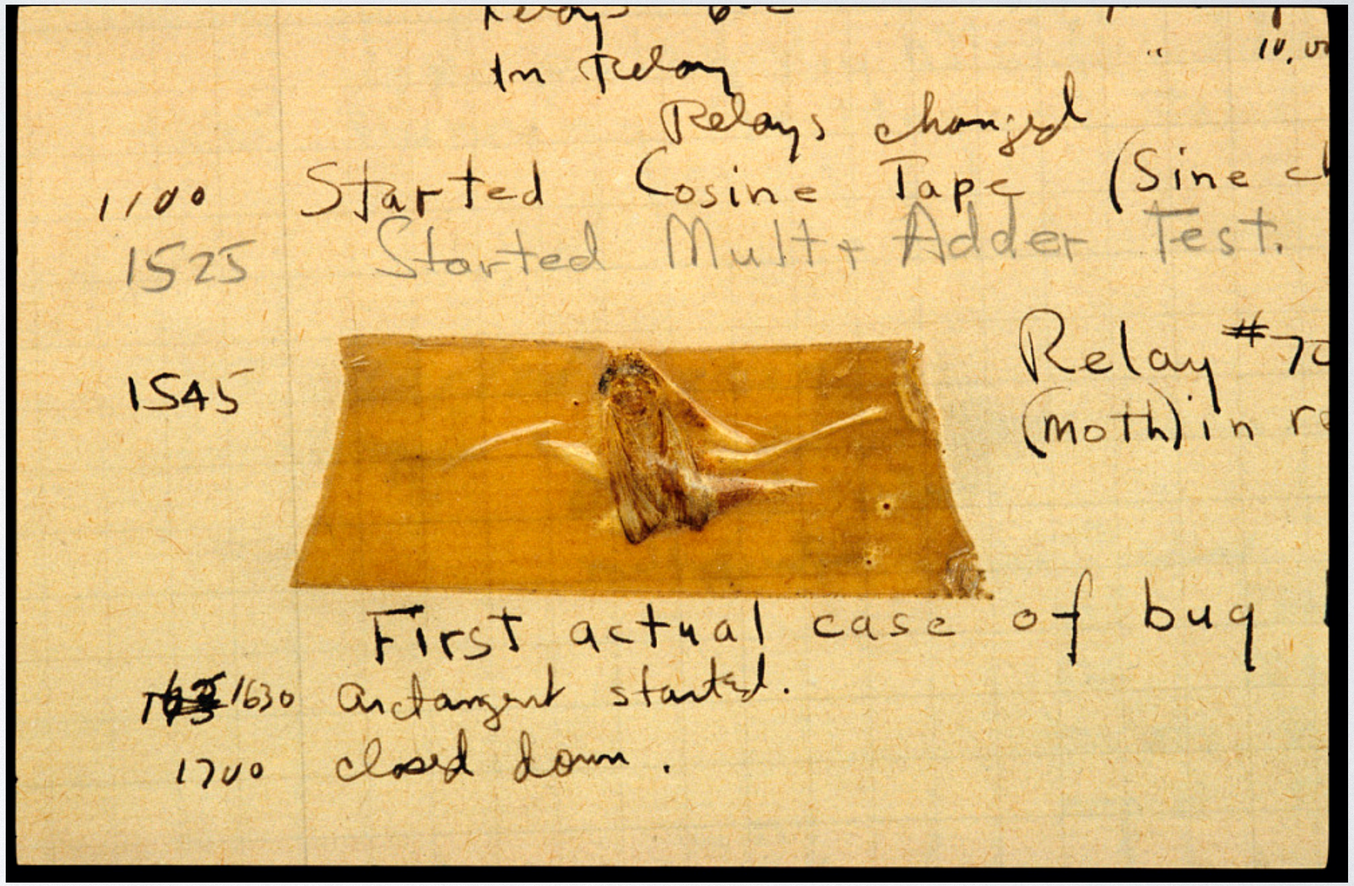
Practice and Research

Debugging Strategies

- Goal today: debugging strategies for large systems
- Large means "too big to fit in your head all at once"
- If you are using a framework, your system is large!

Origin of "Bug"

- Wikipedia: Middle English word *bugge* is the basis for the terms "bugbear" and "bugaboo" as terms used for a monster.
- The term "bug" to describe defects has been a part of engineering jargon since the 1870s
- Thomas Edison wrote in a letter to an associate in 1878:
    - ".. difficulties arise—this thing gives out and [it is] then that "Bugs"—as such little faults and difficulties are called—show themselves"



Bug found in Mark II computer in 1947 (Harvard University)

https://americanhistory.si.edu/collections/search/object/nmah_334663

A Bug Report

- Should say:
  - *How* to reproduce bug
    - (otherwise you won't know whether you've fixed it!)
  - What the observed behavior is
  - What the expected behavior is
- Don't assume the expected behavior is the correct behavior, either!

Do I Really Have To Fix This Bug?

Unfortunately

- You (usually) can't fix every bug.
  - There are too many
  - Lots of bugs don't really matter
- But you'd better fix the important ones!

It's a Feature, Not a Bug?

- Bugs represent *discrepancies* between expectations and the implementation
  - Some (but in most systems, not all!) expectations are encoded in specifications
- Two ways to fix bugs
  - Change code
  - Change the specification

Priorities

- Manager: "Please fix this performance bug. It's super important."
- You: "No problem. I'll re-architect module X."
- Manager: "How long will that take?"
- You: "Two weeks."
- Manager: "If we do that, we won't have time to fix ten other bugs. Actually, that bug wasn't so important after all."

Risk

- Manager: "Please fix this performance bug. It's super important."
- You: "No problem. I'll re-architect module X."
- Manager: "What might break if you do that?"
- You: "Modules Y and Z depend on X, so we'd have to re-test them."
- Manager: "Ugh. Let's fix it in the next release."

In Contrast: Severity

- You: "We need to fix this bug."
- Manager: "How long will it take?"
- You: "Two weeks."
- Manager: "No way."
- You: "But otherwise we might leak private customer data to the Internet."
- Manager: "Ugh, okay, go ahead."

Moral

- Consider *cost*, *risk*, and *severity* before fixing a bug.

On Culture

- "Whose fault is this bug?"
- Leads to a culture of blame.
- Incentivizes bad behavior. Instead, want all bugs to all get reported/logged
- Report and prioritize all bugs
  - Your "very serious" bug may be low priority or actually a feature!

Fixing Bugs

- Two phases
  - Fault localization ("which code is buggy?")
  - Fault repair ("what do I do about it?")
- Most of the work (in my experience) is usually in fault localization

Be Deliberate

- The turtle wins the race
- Fix one bug at a time
- Write test cases for each bug
- Commit after fixing each bug

Two Approaches (of Many)

- Often, first you need to identify a starting point (find relevant code)
- Then, choose:
  - Hypothesis Testing
  - Backward Tracing
- Lots more approaches, but we will focus on these in depth

Debugging Lab

- You will practice using these two approaches by trying to fix two bugs.
- We will randomly assign you bugs and approaches.
- If you stay for all three hours, we guarantee completion (grading is effort-based)
- We hope you'll allow us to use your data for research purposes
  - We want to teach debugging in the most effective way possible!

First Steps

- Identify *somewhere* in the code that's relevant (because you cannot read the whole codebase)
- Begin to understand the code structure at a high level
- Ways to get started:
  - Search the source code for relevant terms (file names & contents)
  - Read **relevant** documentation, if it exists (don't just read everything you can find)
  - Set breakpoints, add print statements

Hypothesis Testing

- Approach: use the *scientific method* to understand the cause of your bug.
- Form hypothesis
- Test your hypothesis

} Repeat until you understand the root cause

- Propose a fix
- Test your fix

Forming a Hypothesis

- **I suspect that _____ (function / class / module / configuration / logic) is most likely responsible, because _____**
- Example sources of suspicion:
  - Recent change (worked okay until yesterday? check yesterday's changes)
  - Suspicious value (why is it printing `0x8badf00d`? where can I find `0x8badf00d` in the code?)
  - Pattern in logs (`A, B, A, B, A, B, …`)
  - Error message (`dynamicroles.atj: removeDynamicRoles() failed: Error: Must be an admin to use this feature… line: 180…`)

Testing a Hypothesis

- **To test this hypothesis, I will _____**
  - Add print statements
  - Use debugger (like adding print statements)
  - Write a minimal reproduction
  - Run a specific test
  - Remove suspicious code
  - Set something to a known-good constant
- **If my hypothesis is correct, I will observe _____ (specific expected evidence).**

Hypothesis Testing Result

- I tested the hypothesis and observed _____ (actual result).
  - A. The hypothesis is confirmed – this is likely the cause.
  - B. The hypothesis is falsified – I will return to observation or form a new hypothesis.

Root Cause

- Repeat steps 2 - 4 until the root cause is clear
- The defect originates at _____ (line of code / function / module) because _____ (reason: wrong logic, missing check, incorrect assumption, etc.).

Fix and Validate

- To fix the issue, I _____
- To confirm my fix worked, I _____.
- If it didn't work:
  - Go to step 1 (form a hypothesis about why).
  - Is the fix in error, or did I make a mistake earlier?

Demo

Backward Tracing

- Assumptions:
  - You identified *some* relevant code
  - That code runs *after* the bug has already manifested
- You know where a null pointer is dereferenced, but why was it null?
- The data structure is bad, but why?
- After you click the button, the resulting web page is wrong.

Backward Tracing

- Ask:
  - Where is this value produced? (file:line / function)
    - `x = a + b;`
    - `print(x); // if x is bad, either a, b, or the + must be wrong`
  - Which function(s)/module(s) write it? (study those)
  - What aspects of the input, configuration, or environment influence it? (experiment with them)
  - What invariants should hold here? (add assertions)
- How did you know (what is your thought process, or what actions did you take) ?

Narrowing the Source

- Based on tracing, I narrow down the issue to _____ (file / class / function / module).

Repeat

- Repeat steps 2 - 4 until the root cause is clear
- The defect originates at _____ (line of code / function / module) because _____ (reason: wrong logic, missing check, incorrect assumption, etc.).

Fix and Validate

- To fix the issue, I _____
- To confirm my fix worked, I _____.
- If the fix does not work, **trace forward** to understand the effect of your change.

Demo

More Fault Localization Tricks

- What follow are some techniques I've found helpful in my many years of debugging experience.

Test Case Minimization

- Remove all elements of the test case that are unnecessary.
- In industry: maybe your QA staff can help you with this.

Narrowing Down the Responsible Code

- Replace modules with mock modules that do the right thing
- Try to show the bug is in a framework you're using: build a minimal broken example
  - Either you file a bug report against the framework, or you learn a key ingredient in the bug and a possible workaround
- Descend a layer of abstraction (debug into the framework)

Divide and Conquer

- The bug is because either:
  - (A) a component does not do what it's supposed to do
  - (B) a component DOES do what it's supposed to, but that is not what some OTHER component (or the user) needed.
- THEREFORE:
  - Be explicit about assumptions (preconditions)
  - Be explicit about expectations (postconditions)

Regressions

- Did this use case previously work, but now it's broken?
  - Then you have a *regression*
- Try: find out which specific change broke it
  - *git bisect*
- Now you know at least some of the relevant code.

Bad State

- Bug: after doing X, some state is wrong.
- Doing X involves running a lot of code.
- Plan: Sprinkle assertions throughout code for X.
- Drill down.

```
assert(state correct);
foo();
assert(state correct);
bar();
```

(OK)

(assertion failed. Bug must be in `foo()`.)

Next step: sprinkle assertions inside `foo()`. Avoid reading `bar()`.

"I Have No Idea Where to Start."

- Search code for relevant-sounding words
- Add breakpoints, trace through relevant code
- Anything hit?
  - If so, you may have found something relevant

Ask an Expert

- "Can you give me a pointer to where I might start looking?"
  - Not asking someone else to do your job
  - You will get up to speed faster and be more helpful if you take a little advice

Which Expert?

- If you can find remotely-related code: git blame

```
c3264e536 (Michael Coblenz      2019-09-06 15:41:37 -0400   81)     def removeThisFieldType(fieldName: String): Context =
c3264e536 (Michael Coblenz      2019-09-06 15:41:37 -0400   82)         Context(contractTable,
c3264e536 (Michael Coblenz      2019-09-06 15:41:37 -0400   83)             underlyingVariableMap,
c3264e536 (Michael Coblenz      2019-09-06 15:41:37 -0400   84)             isThrown,
ff40088b2 (Michael Coblenz      2019-11-25 14:00:30 -0500   85)             transitionFieldsDefinitelyInitialized,
ff40088b2 (Michael Coblenz      2019-11-25 14:00:30 -0500   86)             transitionFieldsMaybeInitialized,
c3264e536 (Michael Coblenz      2019-09-06 15:41:37 -0400   87)             localFieldsInitialized,
c3264e536 (Michael Coblenz      2019-09-06 15:41:37 -0400   88)             thisFieldTypes - fieldName,
c3264e536 (Michael Coblenz      2019-09-06 15:41:37 -0400   89)             valVariables)
c3264e536 (Michael Coblenz      2019-09-06 15:41:37 -0400   90)
c8f738622 (Michael Coblenz      2019-04-02 11:10:28 -0400   91)     def updatedMakingVariableVal(variableName: String): Context =
c8f738622 (Michael Coblenz      2019-04-02 11:10:28 -0400   92)         Context(contractTable,
c8f738622 (Michael Coblenz      2019-04-02 11:10:28 -0400   93)             underlyingVariableMap,
c8f738622 (Michael Coblenz      2019-04-02 11:10:28 -0400   94)             isThrown,
ff40088b2 (Michael Coblenz      2019-11-25 14:00:30 -0500   95)             transitionFieldsDefinitelyInitialized,
ff40088b2 (Michael Coblenz      2019-11-25 14:00:30 -0500   96)             transitionFieldsMaybeInitialized,
c8f738622 (Michael Coblenz      2019-04-02 11:10:28 -0400   97)             localFieldsInitialized,
c8f738622 (Michael Coblenz      2019-04-02 11:10:28 -0400   98)             thisFieldTypes,
c8f738622 (Michael Coblenz      2019-04-02 11:10:28 -0400   99)             valVariables + variableName)
```

Print Statements

- Both for logging data and for monitoring control flow
  - Did this code run?
- Especially useful for debugging race conditions

Narrowing Down the Problem

- Compare how you think it *should* work to how it *does* work
- Plan A: read the code (like reading English)
- Plan B: trace through the code very carefully

Unusual Situations

- "Heisenbugs": bugs that disappear when you try to debug them
- Usual suspects:
  - Race conditions (try using print statements or lightweight logging)
  - Compiler optimizations (either due to performance changes or due to compiler bugs)
- Hardware failures, configuration errors (does it reproduce on another machine?)

Careful Recording

- If you realize you can't keep everything in your head:
- Get out a notebook.
- Record:
  - Each hypothesis
  - Test inputs and results (every test) and what you conclude
- Change only one thing at a time

Fault Repair

- Complaint: `NullPointerException` raised on last line of `foo()`
- Add null check in `foo()`?
- Avoid passing null in `cause()`?
- Usually, you want to fix the root cause. Which is it?

```
void cause() {
  String x = foo(null);
}

int foo(String s) {
  List<String> l  = new List<>();
  l.add(s);
  Map<Integer, List<String>> m = new Map<>()
  m.put(42, l);
  // a bunch more computation
  String p = m.get(42).get(0);
  return p.length();
}
```

Credit: https://www.cs.tufts.edu/comp/121/08-debugging.pdf

Fix Both?

- Wearing both belt and suspenders prevents disaster…

https://www.pexels.com/photo/man-in-shirt-with-bow-tie-and-pants-with-suspenders-18236780/

Best Fix Depends on Risk Tolerance

https://pxhere.com/en/photo/895059

https://commons.wikimedia.org/wiki/File:Tweezers_2019.jpg

Git Blame, Again

```
c3264e536 (Michael Coblenz      2019-09-06 15:41:37 -0400   81)     def removeThisFieldType(fieldName: String): Context =
c3264e536 (Michael Coblenz      2019-09-06 15:41:37 -0400   82)         Context(contractTable,
c3264e536 (Michael Coblenz      2019-09-06 15:41:37 -0400   83)             underlyingVariableMap,
c3264e536 (Michael Coblenz      2019-09-06 15:41:37 -0400   84)             isThrown,
ff40088b2 (Michael Coblenz      2019-11-25 14:00:30 -0500   85)             transitionFieldsDefinitelyInitialized,
ff40088b2 (Michael Coblenz      2019-11-25 14:00:30 -0500   86)             transitionFieldsMaybeInitialized,
c3264e536 (Michael Coblenz      2019-09-06 15:41:37 -0400   87)             localFieldsInitialized,
c3264e536 (Michael Coblenz      2019-09-06 15:41:37 -0400   88)             thisFieldTypes - fieldName,
c3264e536 (Michael Coblenz      2019-09-06 15:41:37 -0400   89)             valVariables)
c3264e536 (Michael Coblenz      2019-09-06 15:41:37 -0400   90)
c8f738622 (Michael Coblenz      2019-04-02 11:10:28 -0400   91)     def updatedMakingVariableVal(variableName: String): Context =
c8f738622 (Michael Coblenz      2019-04-02 11:10:28 -0400   92)         Context(contractTable,
c8f738622 (Michael Coblenz      2019-04-02 11:10:28 -0400   93)             underlyingVariableMap,
c8f738622 (Michael Coblenz      2019-04-02 11:10:28 -0400   94)             isThrown,
ff40088b2 (Michael Coblenz      2019-11-25 14:00:30 -0500   95)             transitionFieldsDefinitelyInitialized,
ff40088b2 (Michael Coblenz      2019-11-25 14:00:30 -0500   96)             transitionFieldsMaybeInitialized,
c8f738622 (Michael Coblenz      2019-04-02 11:10:28 -0400   97)             localFieldsInitialized,
c8f738622 (Michael Coblenz      2019-04-02 11:10:28 -0400   98)             thisFieldTypes,
c8f738622 (Michael Coblenz      2019-04-02 11:10:28 -0400   99)             valVariables + variableName)
```

- Maybe change ff40088b2 is suspicious.

Git Show

```
commit ff40088b2466d724295a4c7e1d6f8385644d8de2
Author: Michael Coblenz <mcoblenz@cs.cmu.edu>
Date:   Mon Nov 25 14:00:30 2019 -0500

    Track state field assignments properly so we can give the right errors when one branch
assigns but fails to transition.

diff --git a/src/main/scala/edu/cmu/cs/obsidian/typecheck/Checker.scala b/src/main/scala/
edu/cmu/cs/obsidian/typecheck/Checker.scala
index 8c93fe4b..627376a6 100644
--- a/src/main/scala/edu/cmu/cs/obsidian/typecheck/Checker.scala
+++ b/src/main/scala/edu/cmu/cs/obsidian/typecheck/Checker.scala
@@ -18,7 +18,8 @@ import scala.collection.immutable.TreeMap
 case class Context(table: DeclarationTable,
                    underlyingVariableMap: Map[String, ObsidianType],
                    isThrown: Boolean,
-                   transitionFieldsInitialized: Set[(String, String, AST)],
+                   transitionFieldsDefinitelyInitialized: Set[(String, String, AST)],
+                   transitionFieldsMaybeInitialized: Set[(String, String, AST)],
                    localFieldsInitialized: Set[String],
                    thisFieldTypes: Map[String, ObsidianType],
                    valVariables : Set[String]) {
@@ -28,7 +29,8 @@ case class Context(table: DeclarationTable,
         Context(contractTable,
             underlyingVariableMap.updated(s, t),
             isThrown,
-            transitionFieldsInitialized,
+            transitionFieldsDefinitelyInitialized,
+            transitionFieldsMaybeInitialized,
             localFieldsInitialized,
             thisFieldTypes,
             valVariables)
```

Fixing the Bug

- Write a test case for the bug (which initially fails)
- Fix the bug
- Search for additional instances of the bug
- Run all the tests
- Get your change reviewed

A Classic Job Interview Question

- Tell me about a tough bug you fixed.

Conclusion

- Narrowing down the test case and the possibly-relevant code can help you identify the root cause
  - Even in unfamiliar code!
- Asking experts is often a good plan.