Lecture 19 — Debugging

Jeff Zarnett & Patrick Lam
jzarnett@uwaterloo.ca & p.lam@ece.uwaterloo.ca

Department of Electrical and Computer Engineering
University of Waterloo

November 14, 2015

Debugging

After testing, it's time to look at debugging.

Programmers once believed it couldn't be done systematically.

The "Richard Feynman Method":

1. Write down the problem.
2. Think very hard.
3. Write down the answer.

Not very realistic, especially for code someone else wrote.

Two Kinds of Software

There are two kinds of software:

Those that are so simple they obviously have no bugs.

Those that are so complex they have no obvious bugs.

More simply stated: all software has bugs.

The Systematic Approach

There are two kinds of software:

Those that are so simple they obviously have no bugs.

Those that are so complex they have no obvious bugs.

More simply stated: all software has bugs.

Debugging Strategy

Based on the scientific method.

1. Observe a failure.
2. Invent a hypothesis.
3. Make predictions.
4. Test the predictions using experiments and observations.
   - Correct? Refine the hypothesis.
   - Wrong? Try again with a new hypothesis.
5. Repeat steps 3 and 4 as needed.

Note: be explicit!

Observing a failure

Basic problem: output is not as expected.

Write down:

- the circumstances;
- expected output; and,
- actual output.

Example: When I enter a negative number like -5 (input) into my app (circumstances), it loops infinitely (output).

Why is this happening?

Make a hypothesis.

Your hypothesis guesses at a cause of the failure, consistent with the observations.

Example: The stopping condition in my program is when `counter == 0`, which never occurs as I'm decrementing `counter` and it is negative.

Make predictions

What else would happen if your prediction was correct?

Example: My program would also loop infinitely on an input of 1.5, as decrementing 1.5 would also never hit 0.

Test and Refine/Discard

Perform an experiment to see if your hypothesis is correct.

- Yes? OK, you can refine the hypothesis.
- No? Try something else.

Example: Yes, feeding 1.5 to my app loops 30-odd times before it crashes. Hypothesis seems correct.

Test and Refine/Discard

When testing your hypothesis, also try something that would *disprove* your theory.

It's a natural human tendency (called confirmation bias).

Repeat as Needed

Until you have an actionable hypothesis, continue to refine or discard it, conducting experiments along the way.

Fix the problem

Modify the code so that the failure can no longer occur.

Example: Instead of checking `counter == 0`, check `counter > 0`.

Key part of the hypothesis:

where is the problem?

Example: The failure was caused at an incorrect test for `counter == 0`.

Isolate the failure to a specific subsystem or module.

Can swap out with known-good versions of modules as an experiment.

Debugging Tactics

Four key tactics:

- Code review.
- Code instrumentation.
- Single-step execution.
- Take a Break.

Take a Break

Take a break?!

Sometimes when you're stumped, a break is what you need.

Even taking 15 minutes to get coffee can help.
Other times, going home and getting a good night's sleep.

Your subconscious continues to work on the problem.
The answer might come to you when you don't expect it.

Code Review

Just stare at the code.

- I find this most effective.
- You need to have a good idea of where the fault lies.
- Get a friend to help; rubber duck debugging?

Code instrumentation

Works very well with the strategy above.

- Use `print` or `Log.d` statements to get information on program state.
- Verify hypotheses based on this information.

Single-step execution

Use a debugger to manually inspect program state.

- Low-level view of variable contents.
- Easy to get bogged down.

Tactics for Bug Localization

- Supply different inputs;
- Instrument the program;
- Run the program;
- Set breakpoints;
- Examine internal state.

Assertions

A statement about the world.

- Should always be true.
- Not really for debugging; more for in-line documentation.
- Aids debugging when it fails—something to fix.

Assertion Examples

```
/* example 1: i is odd */
if (i % 2 == 0)
{
  ...
}
else
{
  assert i % 2 == 1;
  ...
}

/* example 2: doubly-linked list */
assert this.next.prev == this :
  (this+" fails doubly-linked node invariant");
```

Assertion Gotcha

An assertion should never have a side effect.

Single-step execution

Helps with both error localization and hypothesis testing.

Interactively monitor and change program values as the program is executing.

Problem with Single-step execution

Too many steps!

Problem with Single-step execution

Too many steps!

Solution: breakpoints.

Kinds of breakpoints

- Line breakpoints
- Exception breakpoints
- Watchpoints
- Method breakpoints

Part I

Types of Bugs

Types of Bugs

Bugs are our enemies.

Knowing our enemies helps us defeat them.

Fixing a bug might be impossible if we can't reproduce it.

How to reproduce it? Depends on the type.

The Common Bug

The basic type.

In the source code, and behaves predictably.

The Sporadic Bug

Not so consistent as the common bug.

Challenge: finding the right test case.

The Heisenbug

Recall Werner Heisenberg's uncertainty principle in physics:

> *The more precisely the position of a particle is known,*
> *the less precisely the momentum is, and vice versa.*

The harder we try to debug, the better the bug hides.

The Heisenbug

Heisenbugs are usually one of the following problems:

- Race Conditions
- Memory Errors
- Optimization

Heisenbug: Race Condition

The cause of most Heisenbugs.

Two things are running in parallel and program behaviour depends on the order in which they finish.

Adding statements changes how long it takes to execute;
this might suppress the bug!

Heisenbug: Memory Error

Reading an uninitialized variable, reading off the end of an array, reading an area of memory after it has been freed.

When we read an uninitialized variable, what value do we get?

Not an issue in Java.

Heisenbug: Optimization

... premature optimization is the root of all evil.

Donald Knuth

Heisenbug: Optimization

Optimizations are necessary, but sometimes it's a shortcut.

The shortcut might cause an error.

To find out, try disabling the optimization.

Bugs Hiding Behind Bugs

Sometimes we have multiple bugs interacting in some way.

Fix the bugs in the order of their occurrence in execution.

Look for the first error message.

If solving more than one at a time, keep good notes.

The Secret Bug

The customer is using the software but they can't give you a bug report.

May have confidential information (or they don't know how).

Try to reproduce the problem locally.

Solicit anonymized data or visit the customer site.

Configuration Bugs

Sometimes there's nothing wrong with the software.

Problem is the environment. Example: user lacks permissions.

Can change the environment oconfiguration procedure.

Consider changing software to handle the situation gracefully.

Hardware Bugs

Embedded systems often have hardware bugs.

Proving that the problem in hardware is a challenge.

Might need to create a software workaround.

Hardware Bugs

A famous example of a hardware bug: the Pentium FDIV bug.

The processor returned incorrect floating point division values.

Pentium FDIV Bug

$$\frac{4195835.0}{3145727.0} = 1.3338204491362410025 \text{ (Correct value)}$$

$$\frac{4195835.0}{3145727.0} = 1.3337390689020375894 \text{ (Flawed Pentium)}$$

Pentium FDIV Bug

Intel fixed the problem in future processors.

They also recalled the flawed chips.

The suggested interim workaround: multiply numerator & denominator by $\frac{15}{16}$.

The Not-a-Bug

Sometimes when examining a bug, you discover it's not a bug.

The software does what the spec says it should.

How to explain this to the customer?

Eclipse Demo

To close out I'll show you a bit of debugging in Eclipse.