Debugging

is like science
in a slightly wacky universe

Form an argument

Debugging starts from a correctness argument:
How your program was supposed to work, and why it should have worked correctly
You are looking for the flaw in the argument.

If you don’t have a correctness argument ...
• Congratulations, you’ve found the bug. The bug is that you don’t have a correctness argument.

Debugging

Testing is for finding bugs
So you succeeded. Your program failed.
Now what?

Debugging is for understanding and repairing bugs

If you ask for help debugging ...
I will ask you ...
• What is your program doing?
• What should it be doing?
• How is it supposed to work?

I am asking for your correctness argument, and the deviation you have observed.
Observe

We make observations of three kinds:

Assumption: I know $x$ (but let’s just check)
Conjecture: It seems like $y$ (but is it really?)
Hypothesis: If it works like $m$, then we should see $z$

*We start with a model of how the program should work ... what should we see?*

Anomaly

We expected to see something consistent with our mental model of how the program was supposed to work

We saw something else
an anomaly ... something that shouldn’t happen if the program worked as we thought it should

To see the anomaly, you must know what you expected to see. The anomaly is valuable because it breaks part of your theory, and demands an explanation.

Conjecture

*That was weird ... ... but it could happen if the flozzle was rerouted through the bargistator*

Physicians, auto mechanics, and expert debuggers conjecture explanations based on their understanding of how systems (bodies, cars, programs) work. Scientists too.

Hypothesis Testing

The new windshield was brilliant, but within a few days it was foggy again. The haze is on the inside, ...

**TOM:** The other possibility is that your heater core is leaking. If the heater core has a hole in it, the haze on your windshield could be a thin film of coolant.

**RAY:** If it’s coolant, it would have certain characteristics. It would be greasy to the touch. It would smell sweet. And it would likely be thicker at the bottom of the windshield, near where the vents are.

**TOM:** So try a couple of experiments. Try removing the seat covers for a week, and see what happens. If that fixes it, your son may need to go commando -- without his seat covers.

*Car Talk, May 2009*
When my bicycle makes a noise ...

I listen ...
- Does it happen only when I’m pedaling?
- Does it happen only when I’m moving?
- Only in hills? Only in the big ring?
- Only when I’m seated?
- Faster and slower depending on my speed?
- Faster and slower depending on cadence?

These are observations that can confirm some causes and and eliminate others.

Step by step

Make observations to confirm correct execution based on your correctness argument
Observe anomalies
Conjecture possible causes
Develop conjectures into hypotheses observe to (dis)confirm your theory

When stuck, check assumptions

Observing... low tech approach

Insert print statements
- Make them self-identifying
- Consider what you expect, so anomalies are obvious

Avoid “sipping from the firehose”
- Print just enough to confirm or contradict your conjecture
- Reduce, reduce, reduce the test case

Suggestor dump

```java
/** DEBUGGING for nearMatch */
static void showState( int maxEdits,
    String fromWord, int fromPos,
    String toWord, int toPos ) {
  System.out.format("%d : %10s %10s" +
    " %10s %10s%n", maxEdits,
    fromWord.substring(0,fromPos),
    fromWord.substring(fromPos,fromWord.length)

  Logical view of the “matched” and “unmatched” parts
```
Observing ... high tech approach

Development tools like Eclipse include debugging support
Breakpoints (stop here, look around)
Watchpoints (like temporary print statements)
Single-stepping
State inspection (including activation stack)

Very useful ... with a systematic approach
• A good cook sharpens his knives, but sharp knives won’t make you a good cook

Anti-patterns

Random program changes
• That is “programming by magic” ... and it turns a slightly broken program into a total wreck
• But changes to observe effects can be useful, as well as replacing mystery code by something simpler

Massive data dump
• Useful information is lost in the flood. Better to print selectively, with useful abstractions

Jumping to conclusions
• Don’t skip the experiment: Predict a behavior and observe it

Make debugging easier

Build and test incrementally
• Makes bugs smaller and more local

Trim failing test cases
• Make observation and diagnosis easier

Program defensively
• Make your programs crash grandly every time, instead of mostly working most of the time
• Make your program crash as soon as it’s wrong, not later. assert( ... ) is an excellent tool for this.

Learn from every bug

What did I do wrong?
How could I have prevented it?
What can I look for to identify this kind of bug?
– Example: Off-by-one errors in loops, forgetting to reset a flag (boolean), forgetting to advance an index, ...

Build your mental database of bug patterns and build your systematic debugging skills
Debugging is a key skill

As long as programs are written by people, debugging will be part of programming. It’s worth learning to do it well.

The basic methods are used across disciplines

• In medicine: Disease diagnosis
• In auto repair: Fault diagnosis
• In science: Theory building
• In teaching: Explanation (broken model diagnosis)

If you learn to debug well, you’ll use it in surprising ways.