Lecture 8: Debugging Strategy

Engineering Design with Embedded Systems

Patrick Lam
University of Waterloo

January 22, 2013

Part I

Debugging Strategy

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.

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`.

Bug Localization

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.

Part II

Debugging Tactics

Debugging Tactics

Three key tactics:

- Code review.
- Code instrumentation.
- Single-step execution.

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.

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!

Solution: breakpoints.

Problem with Single-step execution

Too many steps!

Solution: breakpoints.

Kinds of breakpoints

- Line breakpoints
- Exception breakpoints
- Watchpoints
- Method breakpoints

Part III

Debugging for Embedded Systems

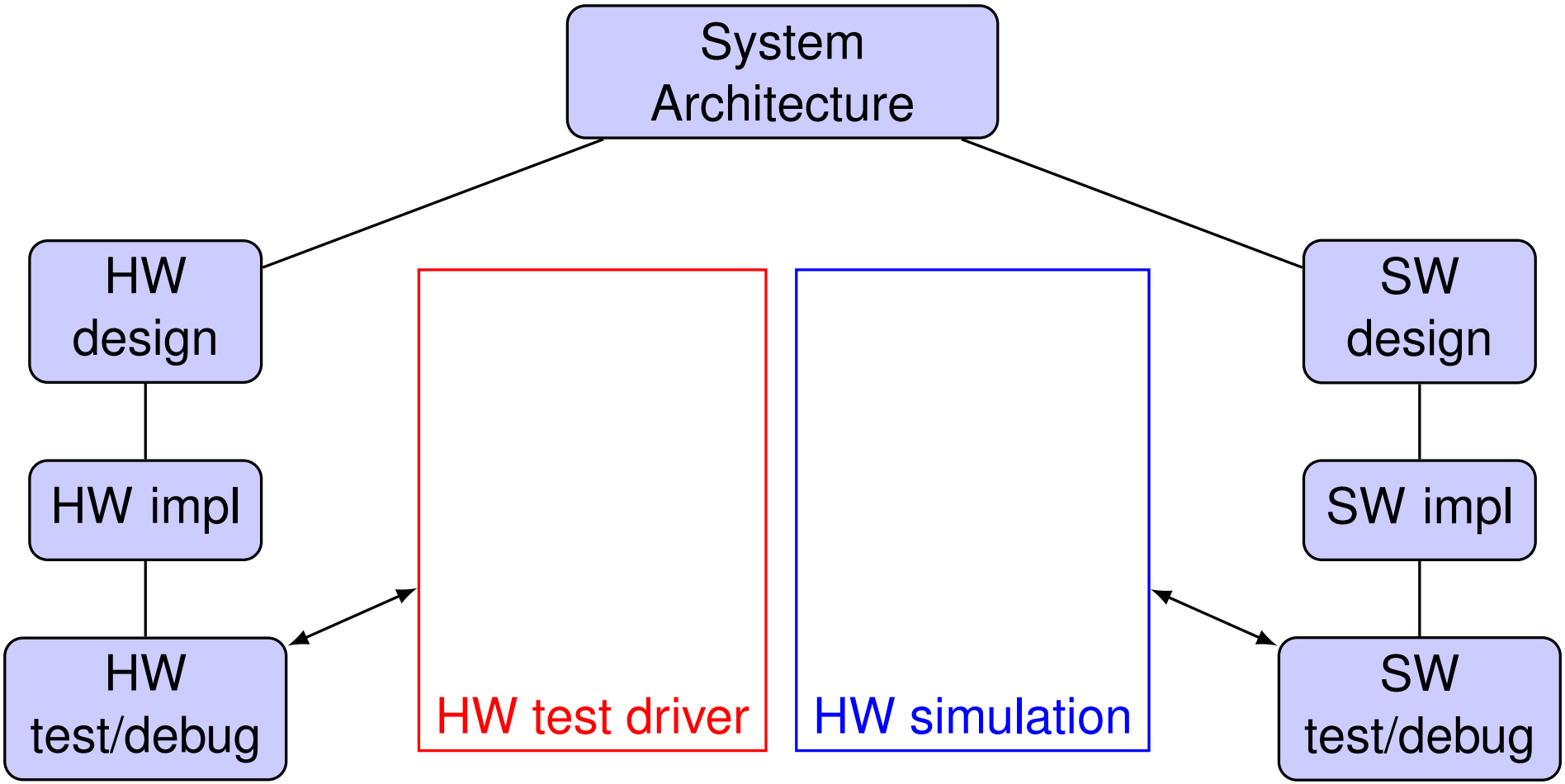
Challenges with embedded systems

Everything's harder: no visibility, no control.

Strategy: divide and conquer.

- to debug software: use a hardware simulator;
- to debug hardware: use test drivers & harness.

Embedded Systems diagram



Challenges in debugging embedded systems

After integrating hardware and software, you'll face more challenges.

- Fault localization: hardware or software?
- Instrumentation: what's the system state?
- Breakpoints are hard!

Debugging post-deployment failures

You won't remove all faults before shipping.

Aviation: black box.

(`http://en.wikipedia.org/wiki/File:Fdr_sidefront.jpg`)

Windows: upload diagnostic information on crash.