17: Testing

Your feedback from HW 9

Thank you for your encouraging and constructive feedback!

Labs: more guidance on what you're meant to be getting out of them

Project: more guidance as to what is expected/how to get started

Fill out the capstone form!

There is a post on Ed

Project proposals

Graded on completion (everyone got a 6/6)

Everyone got comments – some ask for changes to be made before the milestone report/demo, so **please read them**

Watchdog timers

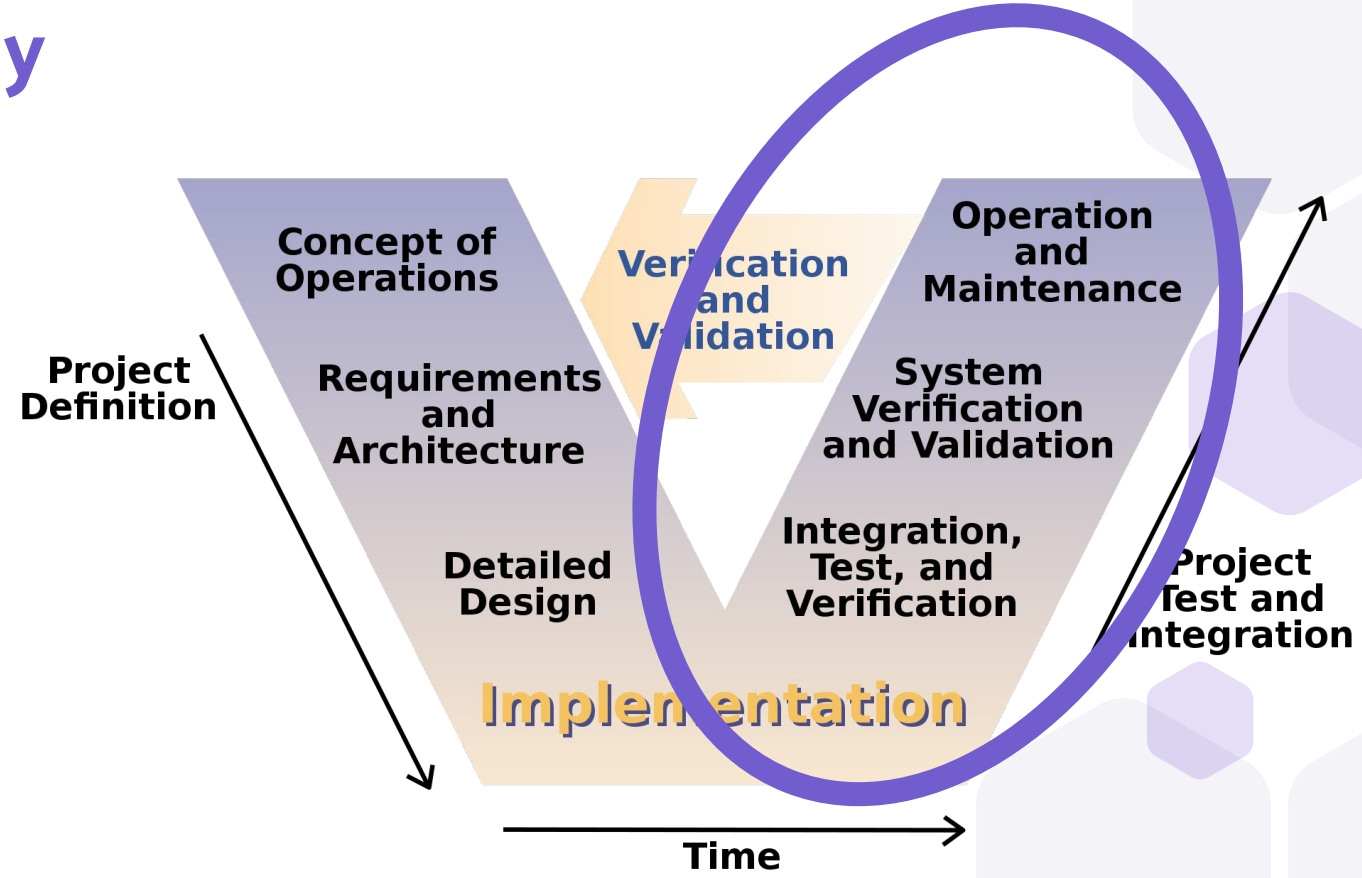
- Watchdogs are meant to detect system hangs
  - Pet them in specific places in the code
  - Successful pet = code still running = no watchdog reset
- Actively detecting a failure and acting upon that failure should not be handled by a watchdog
- Also think about what it means for the system to reset: is resetting safe behavior for your system?
  - Consider using early warning interrupt to warn user instead

Interrupts

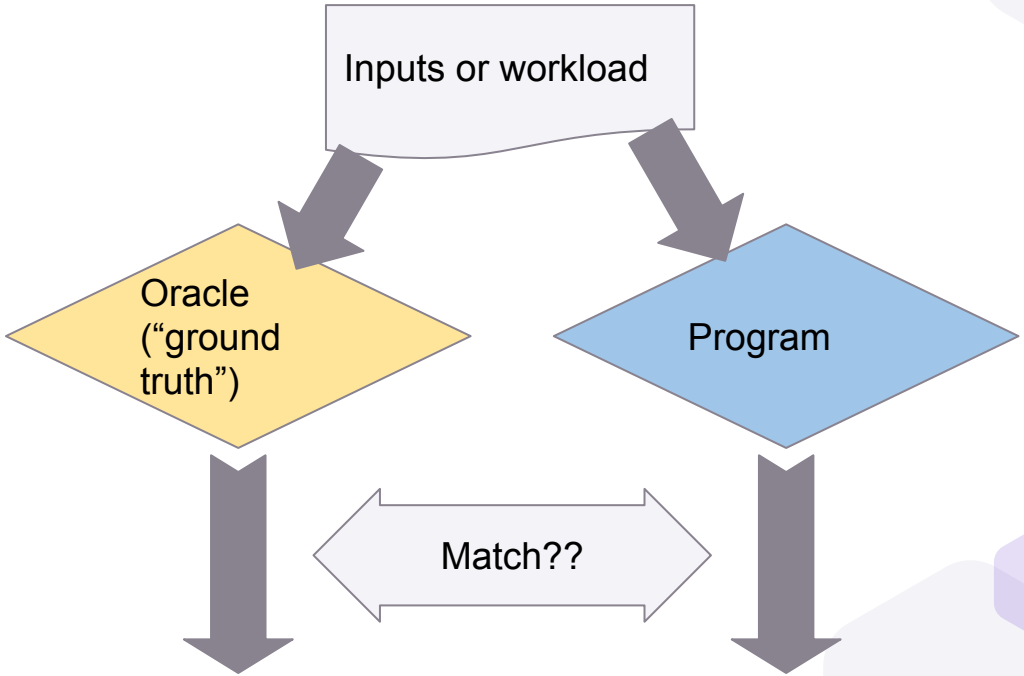
Read the datasheet to find out how your components work

- Some of the proposed “interrupt” ideas could only be accomplished with polling
  - Does it make sense to interrupt on an analog signal (or a “change” in something that’s not a digital electrical signal?)
  - MKR1000 WiFi API does not expose an interrupt for http request (have to poll)

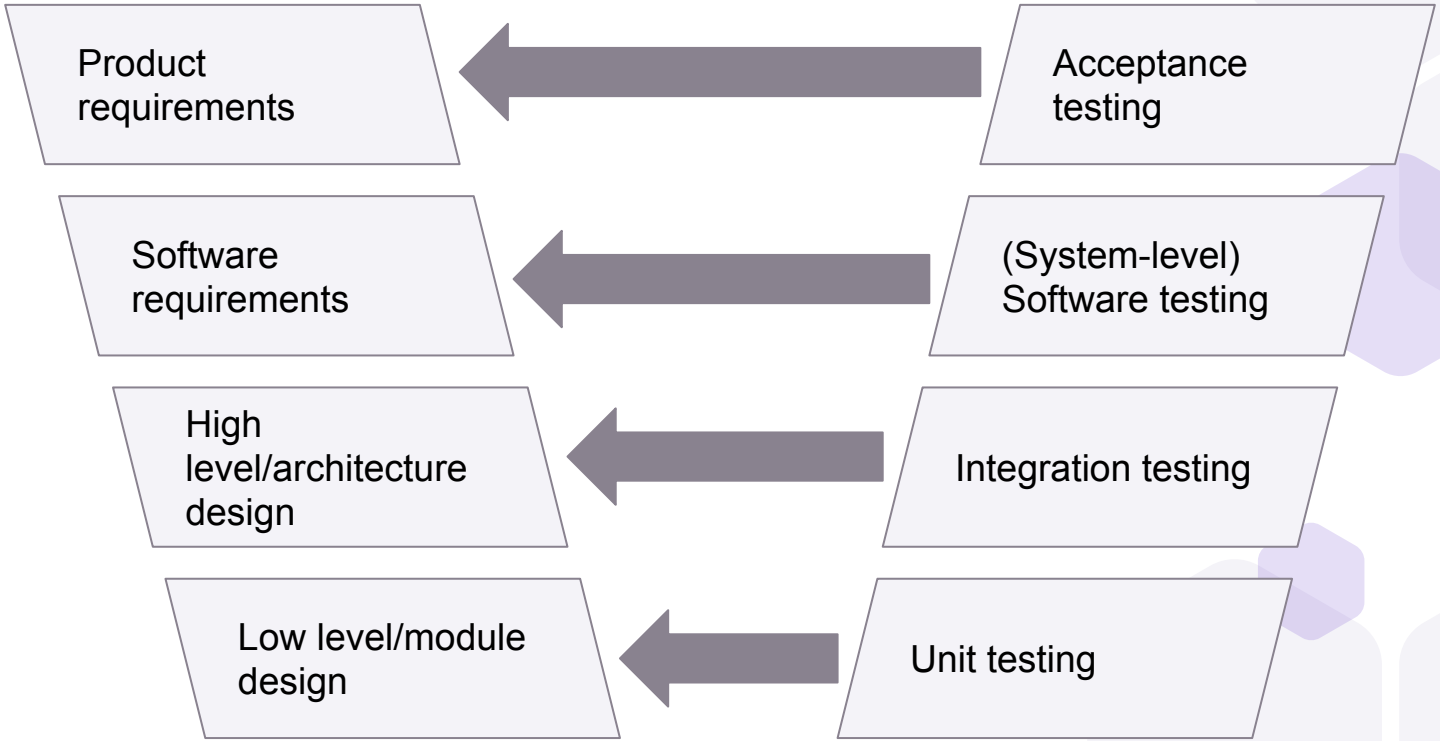
Today


Concept of Operations
Requirements and Architecture
Detailed Design
Implementation
Integration, Test, and Verification
System Verification and Validation
Operation and Maintenance
Project Definition
Time

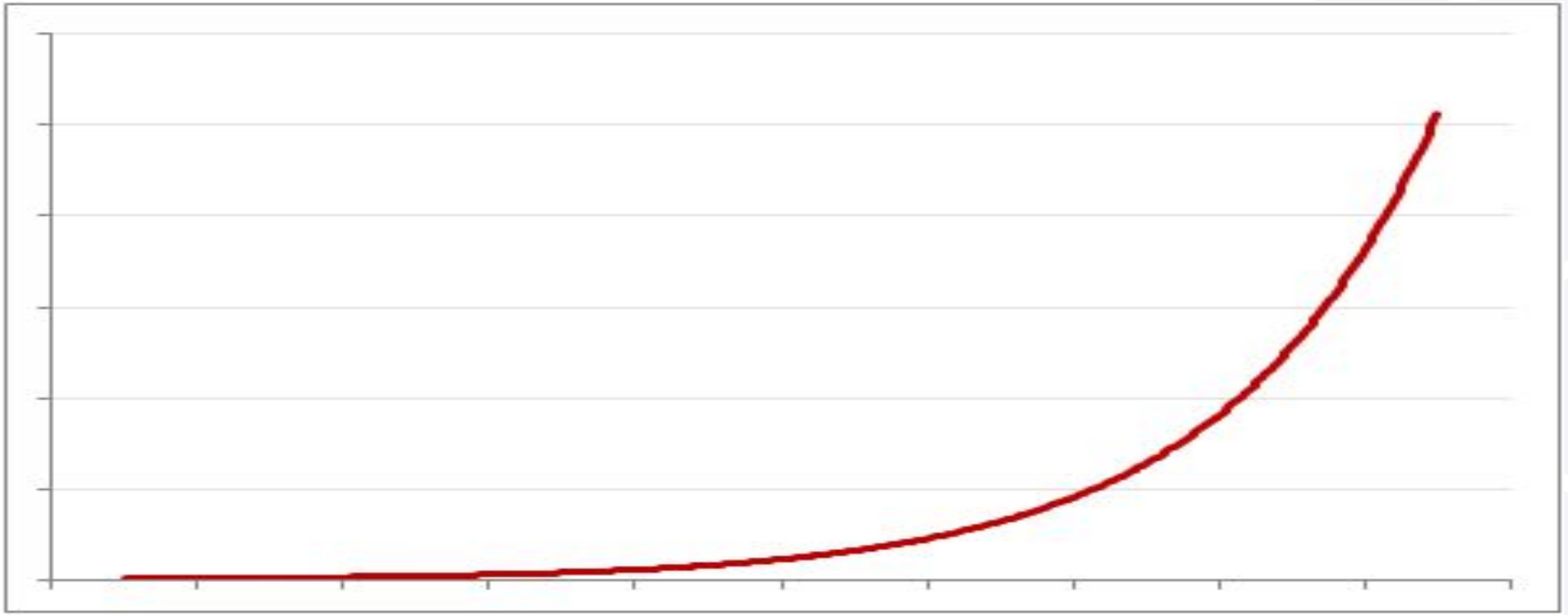
What is testing



Types of testing in the V model



Why not just system level testing?



Unit testing

Check correctness of a module

One unit test = test a single function/method/path

- Cannot test even single function calls exhaustively - consider f(int x, int y, int z)
- Best place to test edge case values

Both structural and functional testing

Functional vs. structural testing

Functional

"Black box" testing

No underlying knowledge of code

Example goal: exercise every requirement for module, or every transition in FSM

Pro: help verify that the code actually matches the spec, instead of the interpretation of the spec

Pro: test assumptions on external code

Pro: sometimes testing is outsourced, and you don't have access to the code

Structural

"White box" testing

Knowledge of structure of code - guides testing

Example: exercise every line of code in function call

Pro: identify code that is unreachable

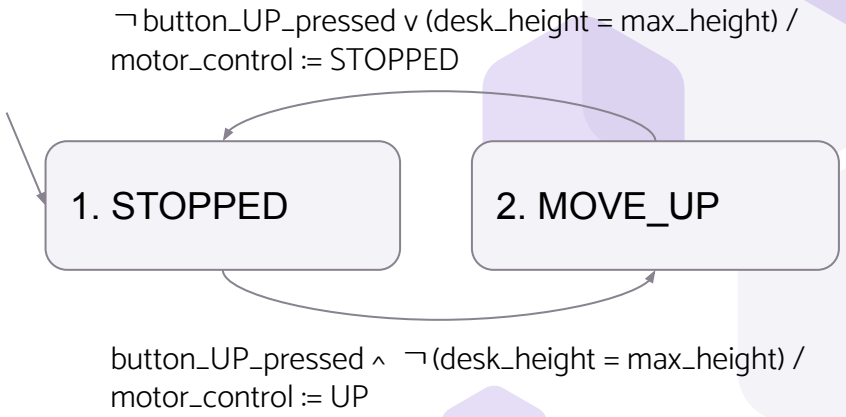
Pro: identify a problematic line of code that may have been missed in functional testing

Pro: identify spaghetti code

What are the tradeoffs between black box and white box testing?

How to unit test an implementation based on FSM?

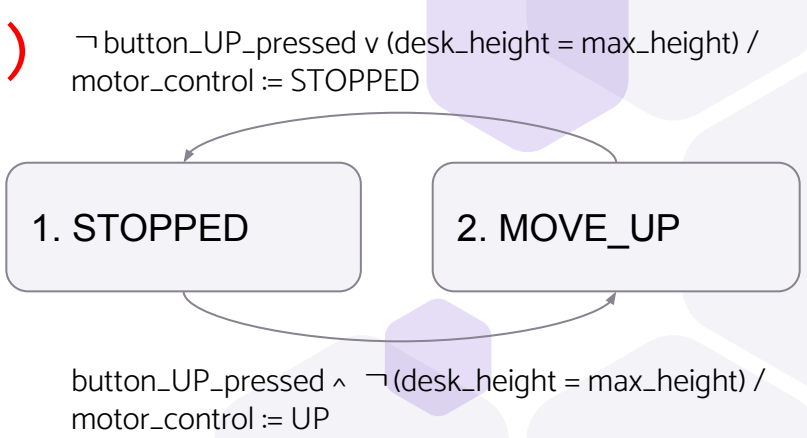
```
state update_fsm(state current_state, bool button_UP_pressed, int desk_height) {
  state next_state;
  switch(current_state) {
    case STOPPED:
    // transition 1-2
    if (button_UP_pressed && desk_height != max_height) {
        next_state = MOVE_UP;
        set_motor_control(MOTOR_UP)
    } else {
        next_state = current_state
    }
    break; // important, or this goes to the next case
    case MOVE_UP:
    ...
    default:
      error("wrong state!");
  }
  return next_state;
}
```



Want to test update_fsm's implementation as-is (without making changes to it)

Test for transition 1-2

```
end_state = update_fsm(STOPPED, true, 35)
assert(end_state == MOVE_UP)
assert(motor_movement == UP)
```



Mock out functions

```
// #define TESTING // uncomment to test
#ifndef TESTING // means TESTING is not defined
void set_motor_control(MotorEnum me) { ...normal operation … }
#else
MotorEnum motor_state;
void set_motor_control(MotorEnum me) { motor_state = me;}
#endif
```

Updated test of FSM transition 1-2

```
end_state = update_fsm(STOPPED, true, 35)
assert(end_state == MOVE_UP)
assert(motor_state == UP)
```

Edge case/unexpected inputs

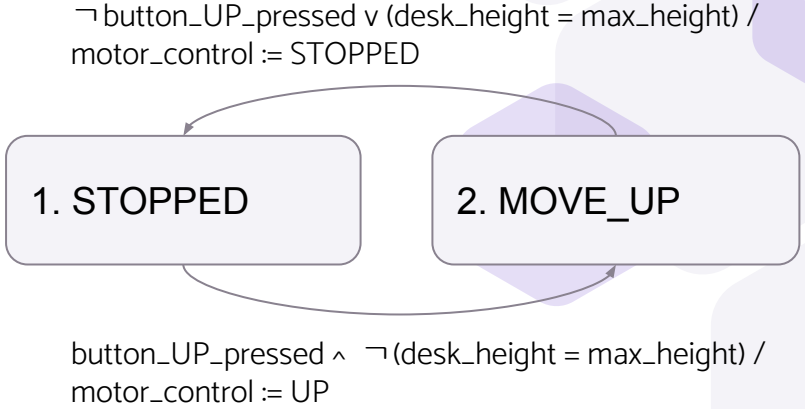
What should this do?

```
update_fsm(STOPPED, true, 5000)
```

```
update_fsm(STOPPED, true, -2)
```

What about this?

```
update_fsm(DONT_MOVE, true, 40)
```



Coverage (a preview)

Notion of how completely a piece of code has been tested with a particular set of tests, with respect to a specific metric

Examples:

- What % of requirements have been tested?
- What % of lines of code have been tested?

100% coverage does **not** mean 100% tested, but it's a start to assess testing thoroughness

Unit testing summary

Cheaper to catch defects here than at any other stage of testing

Perform structural (white-box) or functional (black-box) testing on modules/components/functions