17: Testing
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

General comments

- Make sure there is some complexity (whole project shouldn’t just link an input to an output – what kind of control can you introduce?)
- Check for compatibility with Arduino (SAMD architecture for libraries, 3.3V for parts)
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
  - WDTs should not be used as regular timers!
- Actively detecting a failure (such as malformed input) 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

- Be sure to say what is *triggering* the interrupt, not just the result
- 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/Serial API does not expose an interrupt for communication (have to poll)
Today
What is testing?

Inputs or workload

Oracle ("ground truth")

Program

Match??
V model: artifacts guide testing

At each level: testing asks, “does the implementation match the design?”
Why not just system/acceptance testing?

Cost of fixing defect

Requirements → Design → Implementation → Unit test → Integration test → System test → Acceptance test → Production
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
<table>
<thead>
<tr>
<th>Functional vs. structural testing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Functional</strong></td>
</tr>
<tr>
<td>“Black box” testing</td>
</tr>
<tr>
<td>No underlying knowledge of code</td>
</tr>
<tr>
<td>Example goal: exercise every requirement for module, or every transition in FSM</td>
</tr>
<tr>
<td><strong>Structural</strong></td>
</tr>
<tr>
<td>“White box” testing</td>
</tr>
<tr>
<td>Knowledge of structure of code - guides testing</td>
</tr>
<tr>
<td>Example: exercise every line of code in function call</td>
</tr>
</tbody>
</table>
What are the tradeoffs between black box and white box testing?
Want to test updateFSM’s implementation as-is (without making changes to it)
Test for transition 1-2

Test: transition correctly taken based on inputs, variables/outputs set correctly

```
endState = updateFSM(STOPPED, true, 35)
assert(endState == MOVE_UP)
assert(motor is moving up)
```

guided by FSM/spec (to test that code matches FSM): black-box testing
Mock out functions

// #define TESTING // uncomment to test
#ifndef TESTING // means TESTING is not defined
void setMotorControl(MotorEnum me) { ...normal operation ... }
#else
MotorEnum motorState;
void setMotorControl(MotorEnum me) { motorState = me;}
#endif
Updated test of FSM transition 1-2

endState = updateFSM(STOPPED, true, 35)
assert(endState == MOVE_UP)
assert(motorState == UP)
Edge case/unexpected inputs

What should this do?
updateFSM(STOPPED, true, 5000)
updateFSM(STOPPED, true, -2)

What about this?
updateFSM(DONT_MOVE, true, 40)
int some_fun(int x, int y, int z) {
    if (x == 3 && y < 0 ) {
        // do something;
    } else {
        // do something else
    }
    q = x + z;
    if (q < y) {
        if (x == z) {
            // do another thing
        }
        // do a fourth thing
    }
}
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