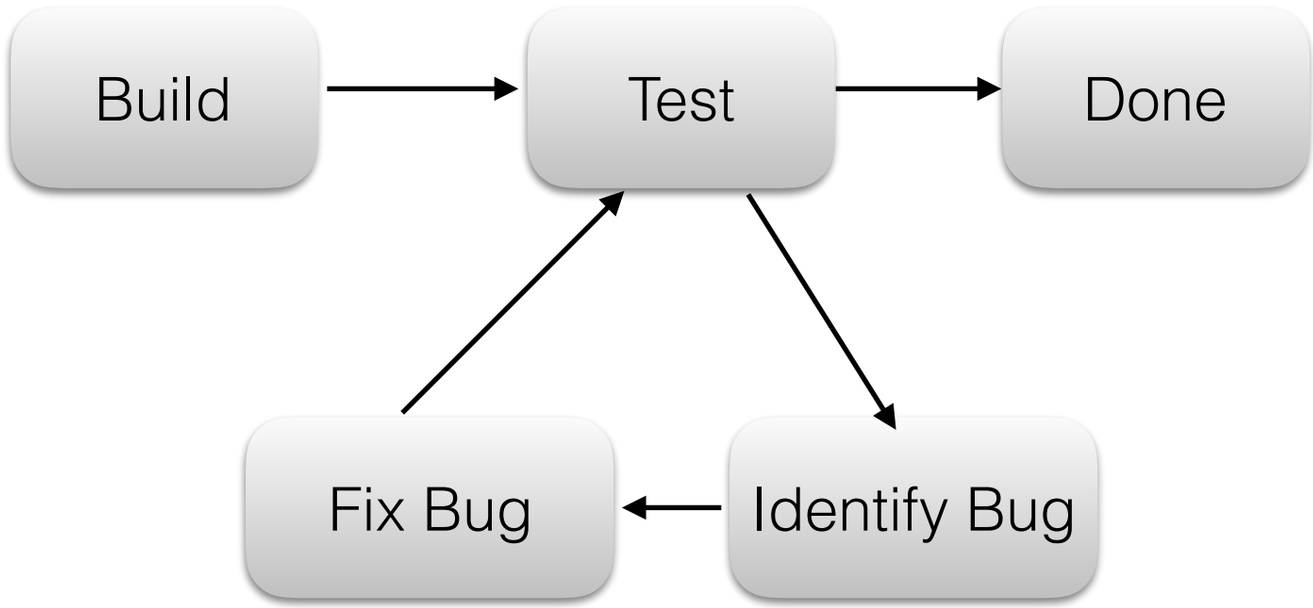
Lecture 4:
Testing & Debugging

Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.

- Kernighan's Law

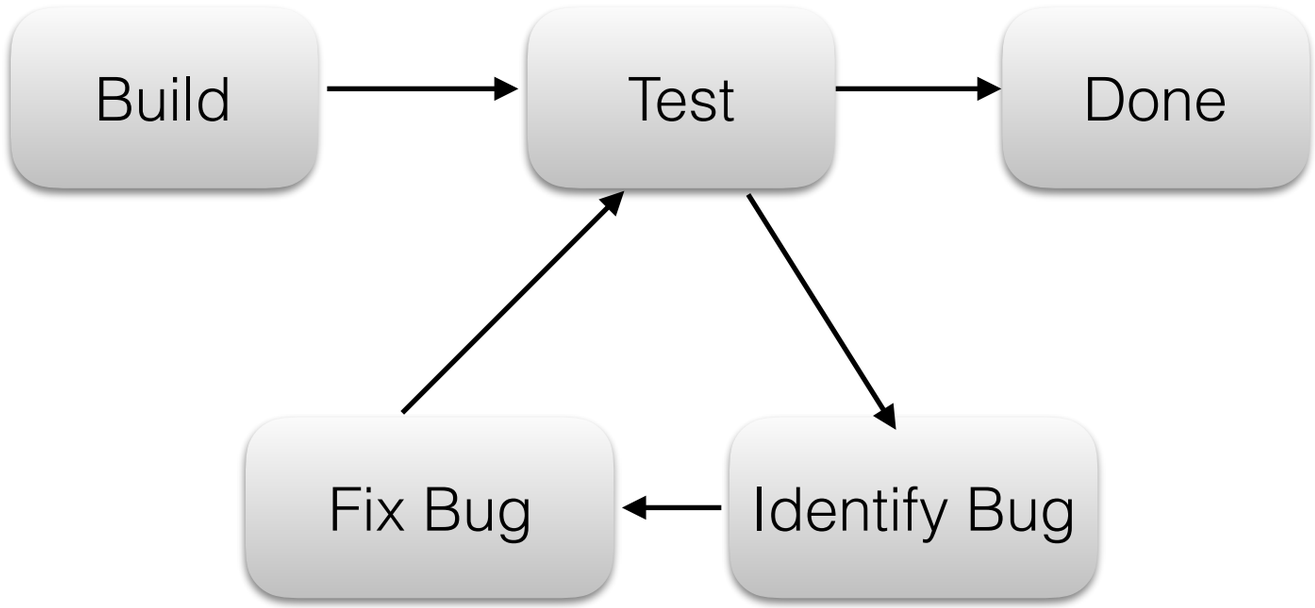
Basic Testing/Debugging

Basic Testing/Debugging



Debugging hardware & software is hard!

Basic Testing/Debugging

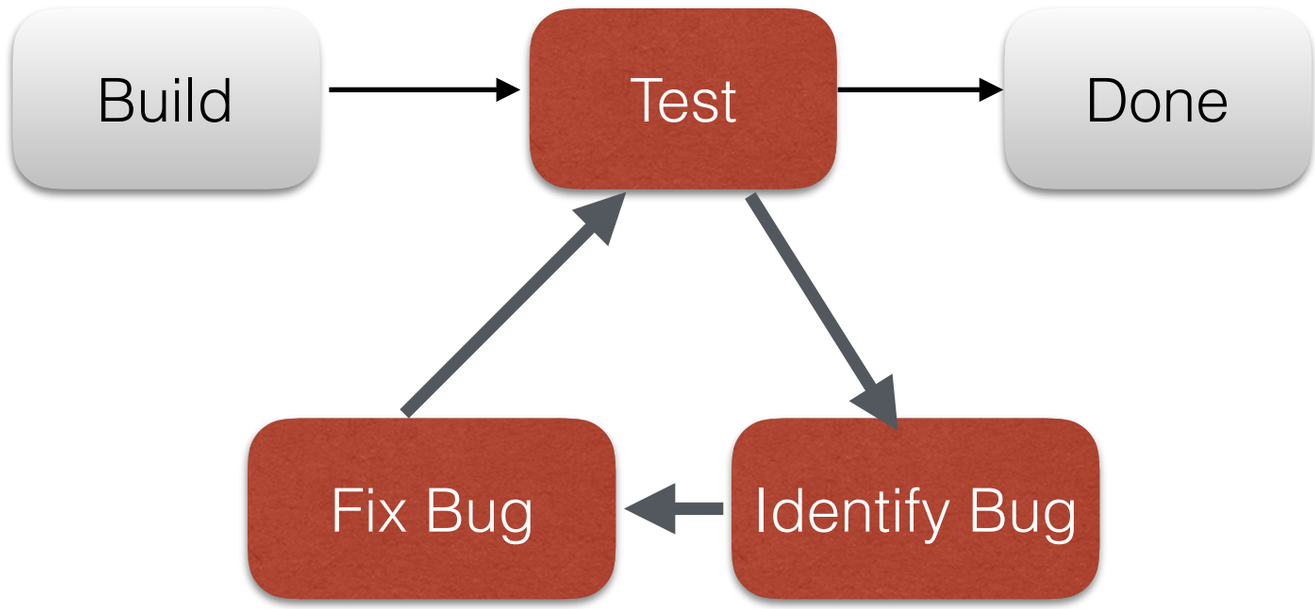


Debugging hardware & software is hard!

- You can't see bits or electrons
- Each component may have complex behavior
- Errors can be at any level (wires, h/w, s/w, algorithms)

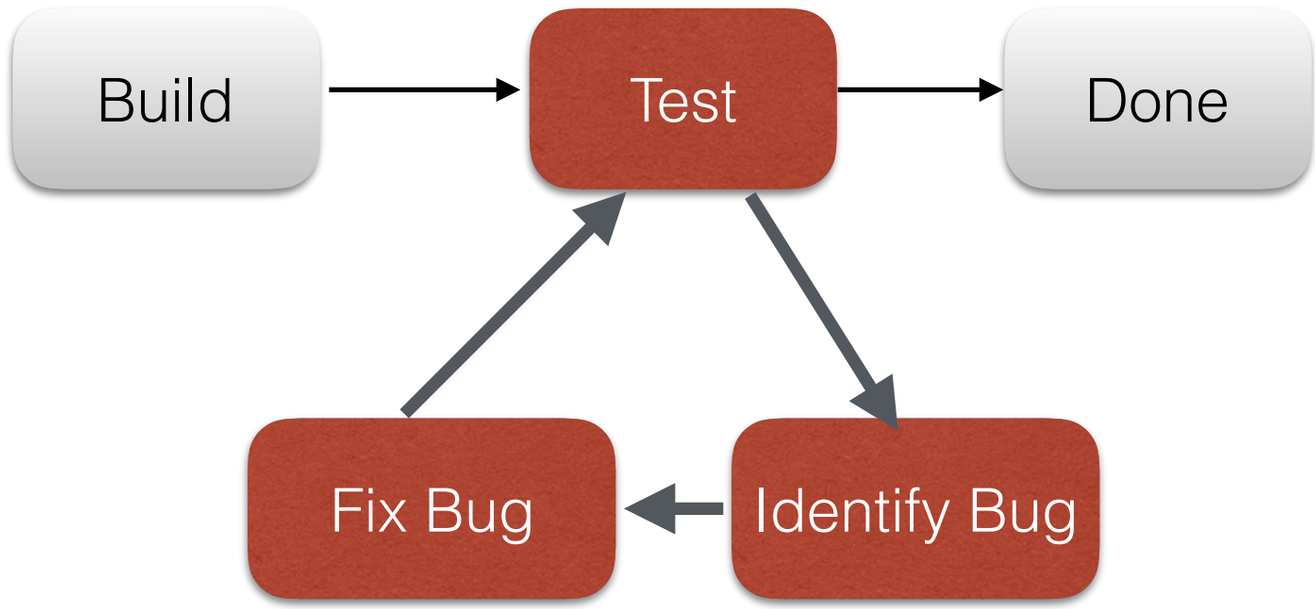
Basic Testing/Debugging

Time spent: 20% ? **80% ?**



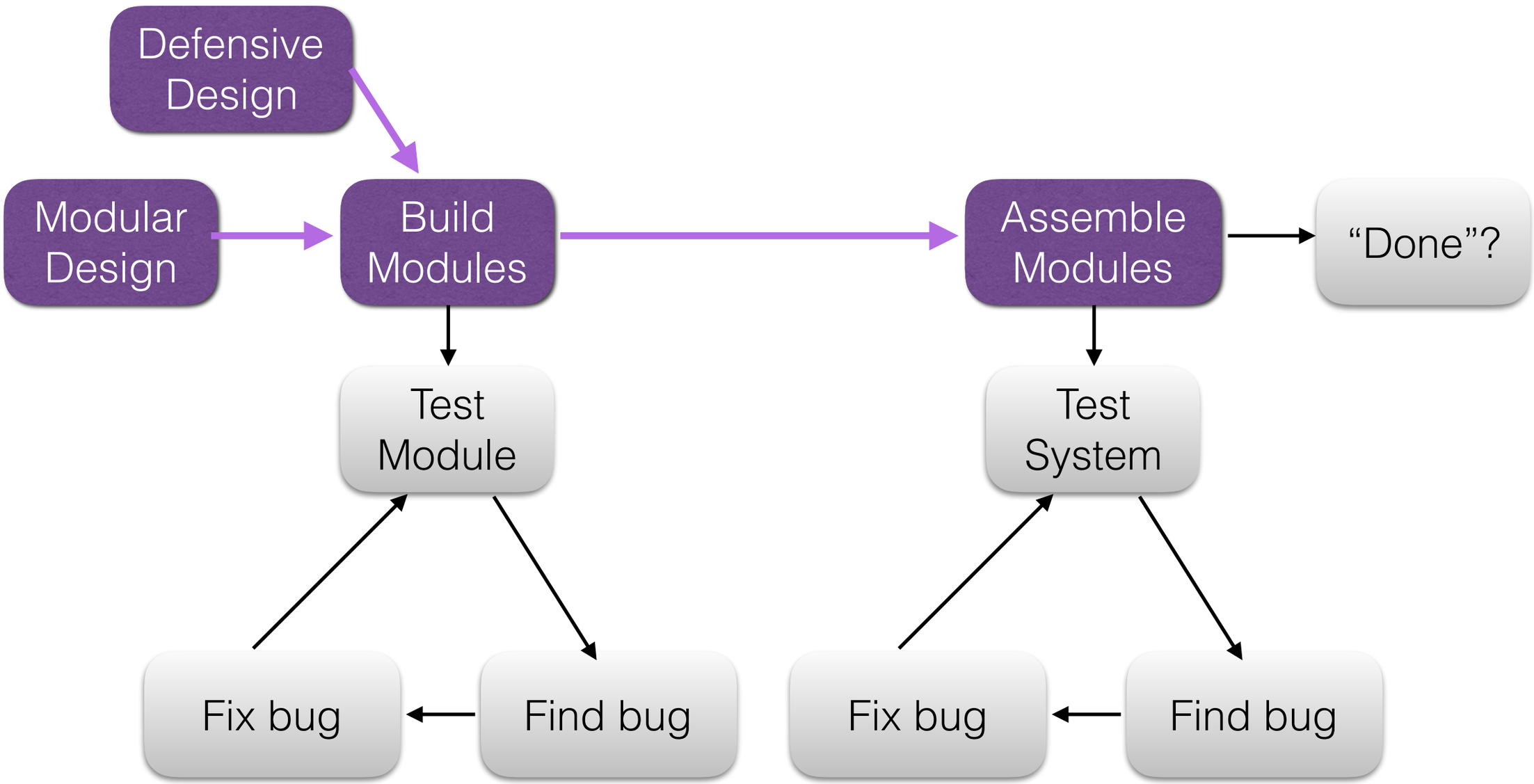
Basic Testing/Debugging

Time spent: 20% ? **80% ?**



What about preventing bugs in the first place?

Advanced Testing & Debugging



Debugging generalizes to other disciplines

E.g., Biology research:

Small details can matter. A lot.

Hard to observe underlying mechanisms

- helps to have mental model
- specialized tools to observe what's happening

Test experimental modules separately, then combine:

- E.g, Genetics, physiology, and behavior
- Test under controlled conditions: *in vitro vs in vivo*, model cells, fluorescent beads

1 Modular Design

- Applies to hardware and software
- Make sure you *have* a design
- Build and test each subsystem separately
- Consider including extra hardware/software specifically for testing modules in isolation

2 Defensive Development

Develop software and hardware to minimize bugs

- Don’t rush; make sure you understand what you’re doing
- Reuse existing modules when possible (e.g., software libraries, circuit designs, pre-made circuits)
- KISS (keep it simple, stupid)

Flexibility / Allow for minor modifications & bug fixes

- Don’t hard-code values, use named constants instead
- DRY (don’t repeat yourself)
- Leave unused IO pins accessible

2 Defensive Development

Plan for Debugging & Minor Modifications:

```
void loop() {
  int potVal;

  for (int i = 0; i < 4; i++) {
    potVal = analogRead(4);  // get pot value
    Serial.print("Potentiometer: ");
    Serial.print(potVal / 1024.0 * 5); // convert to Volts
    Serial.println(" Volts.");
  }

  delay(2000); // do some other stuff here

  for (int i = 0; i < 4; i++) {
    potVal = analogRead(4);
    Serial.print("Potentiometer: ");
    Serial.print(potVal / 1024.0 * 5);
    Serial.println(" Volts.");
  }

}
```

2 Defensive Development

Plan for Debugging & Minor Modifications:

```
void loop() {
  int potVal;

  for (int i = 0; i < 4; i++) {
    potVal = analogRead(4);  // get pot value
    Serial.print("Potentiometer: ");
    Serial.print(potVal / 1024.0 * 5); // convert to Volts
    Serial.println(" Volts.");
  }

  delay(2000); // do some other stuff here

  for (int i = 0; i < 4; i++) {
    potVal = analogRead(4);
    Serial.print("Potentiometer: ");
    Serial.print(potVal / 1024.0 * 5);
    Serial.println(" Volts.");
  }

}
```

Oops, I want to make a few changes:

- move potentiometer to pin 3
- print value in millivolts

2 Defensive Development

Plan for Debugging & Minor Modifications:

```
void loop() {
  int potVal;

  for (int i = 0; i < 4; i++) {
    potVal = analogRead(4);  // get pot value
    Serial.print("Potentiometer: ");
    Serial.print(potVal / 1024.0 * 5); // convert to Volts
    Serial.println(" Volts.");
  }

  delay(2000); // do some other stuff here

  for (int i = 0; i < 4; i++) {
    potVal = analogRead(4);
    Serial.print("Potentiometer: ");
    Serial.print(potVal / 1024.0 * 5);
    Serial.println(" Volts.");
  }

}
```

Oops, I want to make a few changes:

- **move potentiometer to pin 3**
- print value in millivolts

Find-replace won't help

2 Defensive Development

Plan for Debugging & Minor Modifications:

```
const int potPin = 3;
const int numPotReads = 4;

void loop() {
  int potVal;

  for (int i = 0; i < numPotReads; i++) {
    potVal = analogRead(potPin);  // get pot value
    Serial.print("Potentiometer: ");
    Serial.print(potVal / 1024.0 * 5); // convert to Volts
    Serial.println(" Volts.");
  }

  delay(2000); // do some other stuff h

  for (int i = 0; i < numPotReads; i++)
    potVal = analogRead(potPin);
    Serial.print("Potentiometer: ");
    Serial.print(potVal / 1024.0 * 5);
    Serial.println(" Volts.");
  }
}
```

Oops, I want to make a few changes:

- **move potentiometer to pin 3**
- print value in millivolts

2 Defensive Development

Plan for Debugging & Minor Modifications:

```
const int potPin = 3;
const int numPotReads = 4;

void loop() {
  int potVal;

  for (int i = 0; i < numPotReads; i++) {
    potVal = analogRead(potPin);  // get pot value
    Serial.print("Potentiometer: ");
    Serial.print(potVal / 1024.0 * 5); // convert to Volts
    Serial.println(" Volts.");
  }

  delay(2000); // do some other stuff h

  for (int i = 0; i < numPotReads; i++)
    potVal = analogRead(potPin);
    Serial.print("Potentiometer: ");
    Serial.print(potVal / 1024.0 * 5);
    Serial.println(" Volts.");
  }
}
```

You may also see:

```
#define potPin 3
#define numPotReads 4
```

Oops, I want to make a few changes:

- **move potentiometer to pin 3**
- print value in millivolts

2 Defensive Development

Plan for Debugging & Minor Modifications:

```
const int potPin = 3;
const int numPotReads = 4;

void loop() {
  int potVal;

  for (int i = 0; i < numPotReads; i++) {
    potVal = analogRead(potPin);  // get pot value
    Serial.print("Potentiometer: ");
    Serial.print(potVal / 1024.0 * 5); // convert to Volts
    Serial.println(" Volts.");
  }

  delay(2000); // do some other stuff h

  for (int i = 0; i < numPotReads; i++)
    potVal = analogRead(potPin);
    Serial.print("Potentiometer: ");
    Serial.print(potVal / 1024.0 * 5);
    Serial.println(" Volts.");
  }
}
```

Oops, I want to make a few changes:

- move potentiometer to pin 3
- **print value in millivolts**

2 Defensive Development

Plan for Debugging & Minor Modifications:

```
const int potPin = 3;
const int numPotReads = 4;

void printPotValueInMillivolts() {
    int potVal = analogRead(potPin);  // get pot value
    Serial.print("Potentiometer: ");
    Serial.print(potVal / 1024.0 * 5000); // convert to mv
    Serial.println(" Volts.");
}

void loop() {
  for (int i = 0; i < numPotReads; i++) {
    printPotValueInMillivolts();
  }
  delay(2000); // do some other stuff here
  for (int i = 0; i < numPotReads; i++) {
    printPotValueInMillivolts();
  }
}
```

2 Defensive Development

Plan for Debugging & Minor Modifications:

```
const int potPin = 3;
const int numPotReads = 4;

void printPotValueInMillivolts() {
    int potVal = analogRead(potPin);  // get pot value
    Serial.print("Potentiometer: ");
    Serial.print(potVal / 1024.0 * 5000); // convert to mv
    Serial.println(" Volts.");
}

void loop() {
  for (int i = 0; i < numPotReads; i++) {
    printPotValueInMillivolts();
  }
  delay(2000); // do some other stuff here
  for (int i = 0; i < numPotReads; i++) {
    printPotValueInMillivolts();
  }
}
```

These should also be named constants, e.g.:

```
const float maxAnalogValue = 1024.0;
const int maxAnalogInputMillivolts = 5000;
```

2 Defensive Development

Plan for Debugging & Minor Modifications:

```
const int potPin = 3;
const int numPotReads = 4;

void printPotValueInMillivolts() {
    int potVal = analogRead(potPin);  // get pot value
    Serial.print("Potentiometer: ");
    Serial.print(potVal / 1024.0 * 5000); // convert to mv
    Serial.println(" Volts.");
}

void loop() {
  for (int i = 0; i < numPotReads; i++) {
    printPotValueInMillivolts();
  }
  delay(2000); // do some other stuff here
  for (int i = 0; i < numPotReads; i++) {
    printPotValueInMillivolts
  }
}
```

Preventing bugs:

- DRY (don't repeat yourself)
- Avoid hard-coded values
- Write readable code

2 Defensive Development

Make things easy to debug

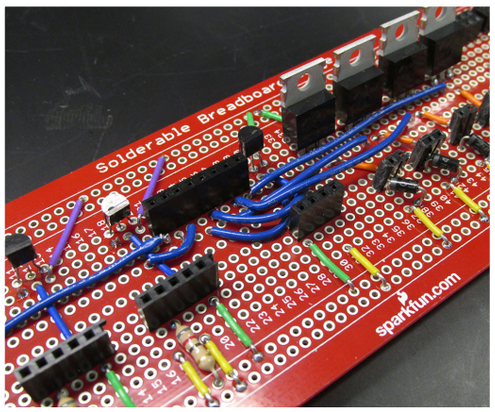
- Write readable code
  - include comments
  - use meaningful variable names:
    - **BAD:** `value = analogRead(2);`
    - **GOOD:** `cageTemperature = analogRead(tempSensorPin);`
- Follow standard practices
  - wire coloring, code indentation, etc
- Include additional documentation if necessary
- Make it easy for someone else (including future you) to understand what you were trying to do

> *"Any code of your own that you haven't looked at for six or more months might as well have been written by someone else."*
>
> - Eagleson's law

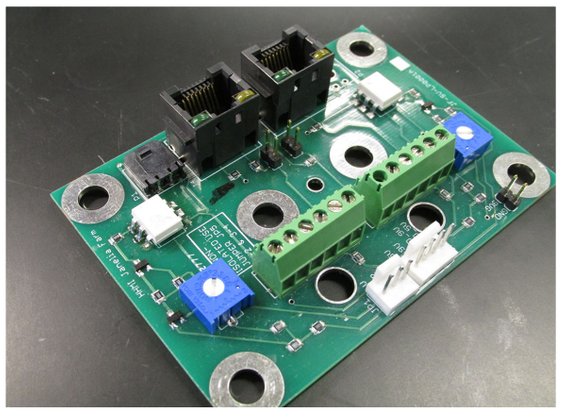
2 Defensive Development

Protect your hardware

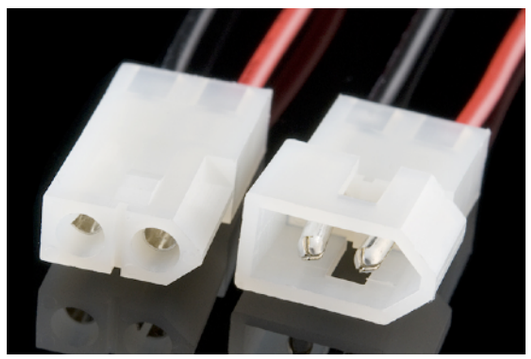
- Invest in good connectors & wiring
- Protect from moisture, spills, etc
- Protoboards and printed circuit boards (PCBs) are more robust than breadboards



Protoboard


PCB

- Use polarized connectors



3 Testing

- Test each module separately first, then assemble system
- Run module/system and observe any obvious errors
- Test for hidden bugs
  - Measure the properties that matter to you (e.g., timing)
  - Where are bugs more likely? (complex algorithms, new components)
  - Response to unexpected inputs/conditions?
- Test for robustness & reliability
  - run the system for hours/days
  - flex wires, reconnect all connectors, etc.
  - your project may eventually be used beyond the limits you designed for

4 Debugging

Defining the bug:

- Try to **reproduce** it reliably
  - What (combination of) factors results in incorrect operation?

- Try to **reduce** it to its simples case
  - Can you trace it back to a single module? A single line of code or hardware component?

- Now you have a clear and specific description of the problem:

  BAD: *“It doesn’t work.”*

  GOOD: *“The red LED does not turn on, but only at the start of even numbered trials.”*

- These steps are helpful for
  - focusing your efforts to find the underlying error
  - asking for help
  - testing that the bug is fixed

4 Debugging

Identifying the cause of the bug:

- Narrow down the scope
  - Can you quickly eliminate options?
  - Is it in software or hardware?
  - Does it persist after removing components (hardware or software)?
- Eliminate uncertainty
  - swap out questionable hardware components
- Check for common errors
  - Off-by-one errors, **=** vs **==**, etc.
  - check power & ground connections
- *Use your mental model*
  - Where does system behavior differ from your expectations?
    - Which one is correct?
  - RTFM (Documentation, data sheets)
  - Test all your assumptions

Debugging Tools

Use your senses (sight, touch, smell)

Use tools to display what you can't directly observe:

- Serial.println()
  - output timing, values of variables, detect when you reach a particular line of code
  - may slow down program execution
- digitalWrite() to unused pin
  - minimal effect on program speed
  - turn HIGH/LOW to measure duration of code block (or detect its execution)
  - observe with LED or oscilloscope (use pin 13 for internal LED)
- Digital multimeter (DMM)
  - directly measure voltages, resistance, connectivity, current
- Oscilloscope
  - Plots voltage vs time
  - Multiple channels, triggering, etc
- Specialized hardware (Logic analyzer, variable power supply, thermal imaging camera, …)
- **Ask for help!** (but do your homework first)