Example Modules

Motor module: Implements control of all motors.

Sensor module: Hides the interface to all sensors.

Choose Modules

- Design decision: hide the ones that are most likely to change
- Design decision: pick at least 2
- Design decision: pick at least 2

Steps in Software Engineering

- Plan the project
- Design the system
- Decompose into components (modules/classes)
- Design interfaces
- Implement
- Verify and validate

Possible interactions with other systems (OS, operating system, other programs, etc.) are essentially infinite:
- Doesn't wear out or break.
- Interoperability is rarely valid.
- Interoperability is never valid.
- Doesn't consist of materials that obey physical laws.
- Components can interact in many ways.
- Requires special skills to choose designs.
- No natural (internal) boundaries.

How is Software Different from Other Engineering
- Vectors may include other modules
- Use case/user’s guide
- Specification

- How are the outputs related to the inputs (i.e., functional)

- Behaviour
  - Does it interact with the environment?
  - Data type/interpretation
  - Methods and their arguments, input and output variables (be specific)
  - Interface — How can the module/class be used?

Module/Class Design Documentation

Example Module Design

Test plan & results

Outputs
- HW signals to drive motors.
- Right, reverse, stop.

Inputs
- Direction command: forward, left, reverse, hand left, hand right, hand off.
- Steer ing module: Controls movement of vehicle.
- Front, off, all
- Outputs: Signals relative to tape (e.g., on tape, off left, off right, off)
- Outputs: HW signals from tape detection sensors.

Software Design Documentation

Better Modules

What’s wrong with this?

Both wheel directions are set to logic 0 and the PWMs are set to 0% duty cycle. The wheels are turned in opposite directions as applicable. For reversal, when the right wheel is stopped by setting the right wheel speed to 0% and the left wheel is stopped by setting the left wheel speed to 0% and the right wheel speed to 80% duty cycle. If the “start left” command is given, then the left wheel which is stopped by setting the left wheel speed to 0% and the right wheel speed to 80% duty cycle. If the “start right” command is given, then the right wheel which is stopped by setting the right wheel speed to 0% and the left wheel speed to 80% duty cycle. If the “start” command is given, the PWMs for both wheel speeds (pins 15 and 17) are set to logic 1 and the PWMs for both wheel speeds (pins 16 and 18) are set to logic 0. The steering module is used to control the direction of the motor. If the “forward” command is given, both wheel directions (pin 15 and 17) are set to logic 1 and the PWMs for both wheel speeds (pins 15 and 16) are set to logic 1. The steering module is used to control the direction of the motor. If the “forward” command is given, both wheel directions (pin 15 and 17) are set to logic 1 and the PWMs for both wheel speeds (pins 15 and 18) are set to logic 1. The steering module is used to control the direction of the motor.
Steering Module Internal Design

Hardware signals:

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>L_Dir</td>
<td>Left wheel direction, 1 = forward, 0 = reverse</td>
<td>PIC:15, HB:2, HB:7</td>
</tr>
<tr>
<td>R_Dir</td>
<td>Right wheel direction, 1 = forward, 0 = reverse</td>
<td>PIC:18, HB:15, HB:10</td>
</tr>
<tr>
<td>L_Speed</td>
<td>Left wheel speed, PWM signal</td>
<td>PIC:16, HB:1</td>
</tr>
<tr>
<td>R_Speed</td>
<td>Right wheel speed, PWM signal</td>
<td>PIC:17, HB:9</td>
</tr>
</tbody>
</table>

Outputs:

<table>
<thead>
<tr>
<th>State</th>
<th>L_Dir</th>
<th>R_Dir</th>
<th>L_Speed (% duty)</th>
<th>R_Speed (% duty)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stop</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Forward</td>
<td>1</td>
<td>1</td>
<td>80</td>
<td>80</td>
</tr>
<tr>
<td>AdjLeft</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>80</td>
</tr>
<tr>
<td>AdjRight</td>
<td>1</td>
<td>1</td>
<td>80</td>
<td>0</td>
</tr>
<tr>
<td>HardLeft</td>
<td>0</td>
<td>1</td>
<td>80</td>
<td>80</td>
</tr>
<tr>
<td>HardRight</td>
<td>1</td>
<td>0</td>
<td>80</td>
<td>80</td>
</tr>
<tr>
<td>Reverse</td>
<td>0</td>
<td>0</td>
<td>80</td>
<td>80</td>
</tr>
</tbody>
</table>

Verification & Validation

- Testing
- Reviews/Inspections
- Does it do what the client wants?
- Does it do what you said it should do?
- Does it do what the client needs?

Better Module Design Documentation

Name: Steering module

Exported types:

```c
enum Direction { Forward, AdjLeft, AdjRight, HardLeft, HardRight, Reverse, Stop };
```

Methods: void steer(Direction dir) — sets the direction of travel to dir.

Behaviour:

- Initial state is Stop.
- Each call to steer changes the output state to that given by its argument, as follows:

<table>
<thead>
<tr>
<th>dir</th>
<th>Left wheel</th>
<th>Right wheel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stop</td>
<td>stopped</td>
<td>stopped</td>
</tr>
<tr>
<td>Forward</td>
<td>forward, full</td>
<td>forward, full</td>
</tr>
<tr>
<td>AdjLeft</td>
<td>stopped</td>
<td>forward, full</td>
</tr>
<tr>
<td>AdjRight</td>
<td>forward, full</td>
<td>stopped</td>
</tr>
<tr>
<td>HardLeft</td>
<td>reverse, full</td>
<td>forward, full</td>
</tr>
<tr>
<td>HardRight</td>
<td>forward, full</td>
<td>reverse, full</td>
</tr>
<tr>
<td>Reverse</td>
<td>reverse, full</td>
<td>reverse, full</td>
</tr>
</tbody>
</table>
Reviews/Inspections

Testing

Don't forget to test the error cases.

Aim for high coverage.

The goal is to find errors — try hard to break the system.

- Expected results
- Input
- Test environment
- Plan your tests in advance (when component interface is designed)