It's hard enough to find an error in your code when you're looking for it; it's even harder when you've assumed your code is error-free.

– Steve McConnell
Integration Testing

Anti-Patterns:
- Skipping unit test to do system test
- No traceability from integration test to High Level Design
- Integration test “pass” criterion based on system function, not interfaces

Testing component integration:
- Exercise all component interfaces
  - Correct responses to input sequences?
  - Handle all types of data on interfaces?
- Ensure modules match HLD and SDs
  - Assume unit test has vetted each component
  - Concentrate on component interactions
## Integration Test Approaches

### Exercise all interfaces
- All inputs result in correct outputs
- Every component interface exercised
  - With all relevant values
  - With all relevant timing & sequencing
- Use SDs and HLD info drive testing
  - Pass/fail: does it match SD?

### Integration test coverage:
- All arcs on all SDs exercised?
- Off-nominal behaviors tested?
  - Invalid sequencing and extraneous inputs?

---

**Integration Test IT-1a:**

1. Initialize modules
2. Test setup: CoinCount to zero
3. Insert coin (1a)
4. Observe CoinIn(true) (1b)
5. Observe CoinIn(false) (1c)
6. Observe mCoinCount == 1 (1d)
Tracing Integration Tests to SDs

- Observe module interactions
  - Set up test
    - Meet SD preconditions
  - Feed input arcs to modules
  - Observe intermediate arcs
  - Observe output arcs
  - Find a way to observe documented side effects (e.g., final CoinCount)

- Integration test “pass” is *not* just based on final output
  - Do all the arcs appear in expected sequence?
  - Is timing appropriate?
Integration Tests and Messaging

- Interfaces often look like “messages”
  - Categorical values (enums)
  - Data structures
  - Network packets
- Integration testing should exercise “message” structure
  - All types of messages
  - Valid and invalid field values
  - Timing, exception handling
    - e.g., bad checksum, bad sequence number
- HLD will have a message dictionary
  - Defines message types, formats, etc.
  - Accompanied by a validation test suite

### Table: OBDii Parameter ID message dictionary (CAN Network Messages)

<table>
<thead>
<tr>
<th>PID (hex)</th>
<th>PID (Dec)</th>
<th>Data bytes returned</th>
<th>Description</th>
<th>Min value</th>
<th>Max value</th>
<th>Units</th>
<th>Formula[a]</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>4</td>
<td>PIDs supported [01 - 20]</td>
<td></td>
<td></td>
<td></td>
<td>Bit encoded [A7..D0] == [PID $01..PID $20] See below</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>4</td>
<td>Monitor status since DTCs cleared. (Includes malfunction indicator lamp (MIL) status and number of DTCs.)</td>
<td></td>
<td></td>
<td></td>
<td>Bit encoded. See below</td>
</tr>
<tr>
<td>02</td>
<td>2</td>
<td>2</td>
<td>Freeze DTC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>3</td>
<td>2</td>
<td>Fuel system status</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>04</td>
<td>4</td>
<td>1</td>
<td>Calculated engine load</td>
<td>0</td>
<td>100</td>
<td>%</td>
<td>$100/255 A (or $A/2.55)</td>
</tr>
<tr>
<td>05</td>
<td>5</td>
<td>1</td>
<td>Engine coolant temperature</td>
<td>-40</td>
<td>215</td>
<td>°C</td>
<td>A - 40</td>
</tr>
</tbody>
</table>

Integration Test Best Practices

**Trace Integration tests to HLD**
- Exercise all arcs on every SD
- Cover all modules; all interfaces
- Cover all message types and fields

**Integration test pitfalls**
- System testing alone misses system integration corner cases
  - Sometimes a misbehaving system appears to work at system test
  - Can be difficult to exercise off-nominal SDs at system level
- If you skip HLD, you have nothing to trace Integration Tests to