



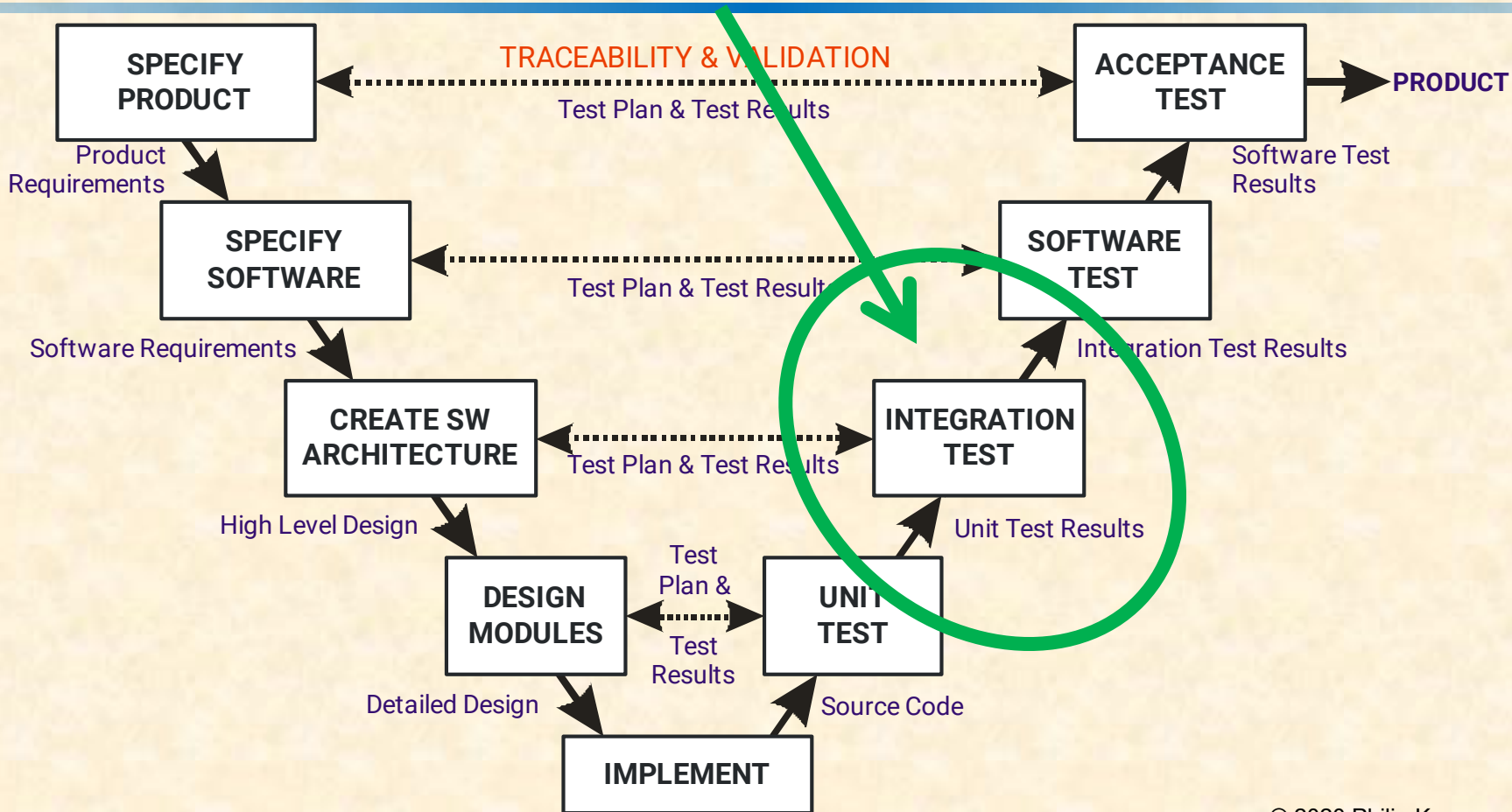
Prof. Philip Koopman

Integration Testing

“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*

YOU ARE HERE

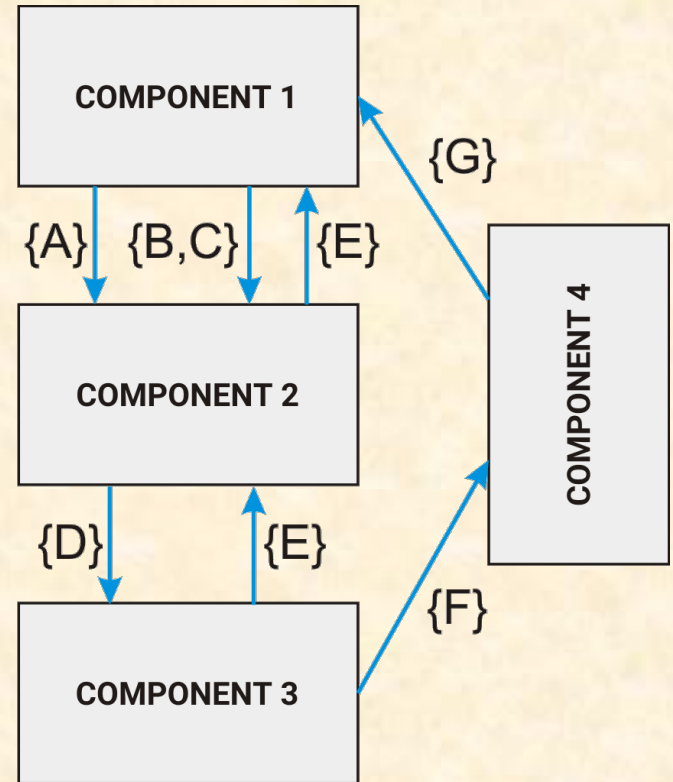


■ Anti-Patterns:

- **Skipping straight to 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, including SDs**
 - Assume unit test has vetted each component
 - Concentrate on component interactions

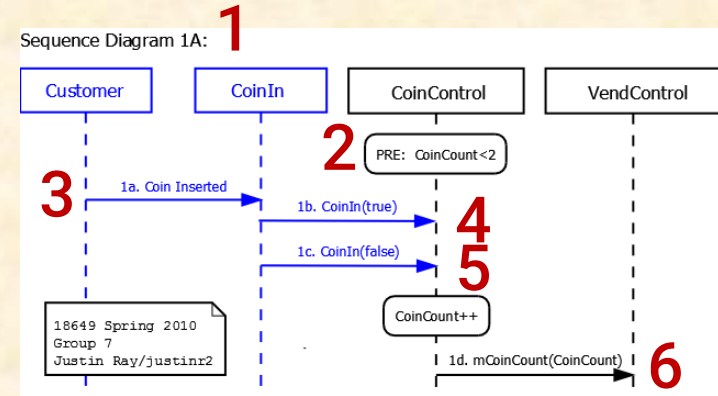


■ Exercise all interfaces

- 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?
 - Extraneous outputs?



Integration Test IT-1a:

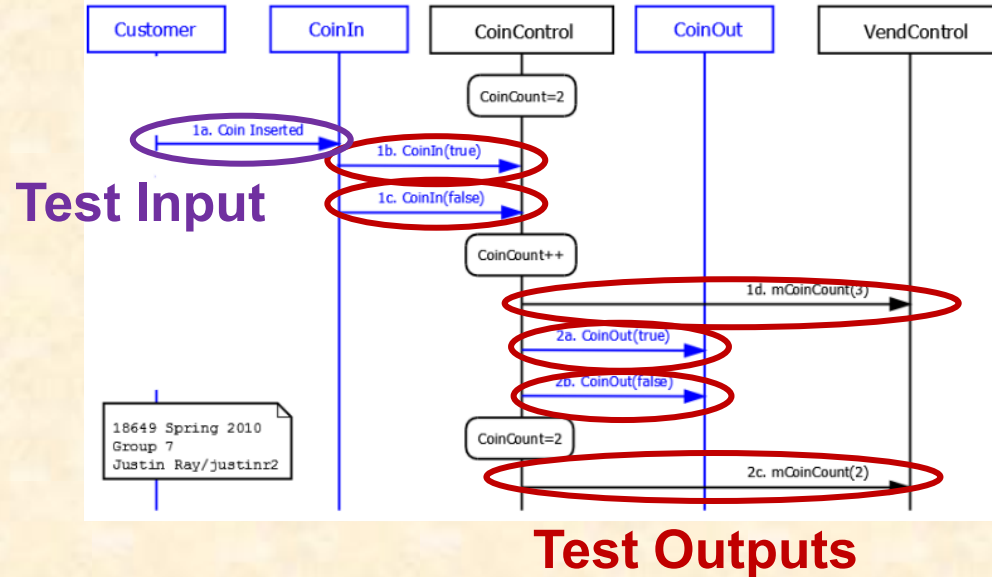
1. Initialize modules
2. Test setup: CoinCount to zero
3. Insert coin (1a)
4. Observe CoinIn(true) (1b)
5. Observe CoinIn(false) (1c)
6. Observe mCoinCount == 1 (1d)

Tracing Integration Tests to SDs

■ Observe module interactions

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

Sequence Diagram 1B:



■ 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 the message dictionary
 - Defines message types, formats, etc.
 - Accompanied by a validation test suite

Mode 01 [edit]

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

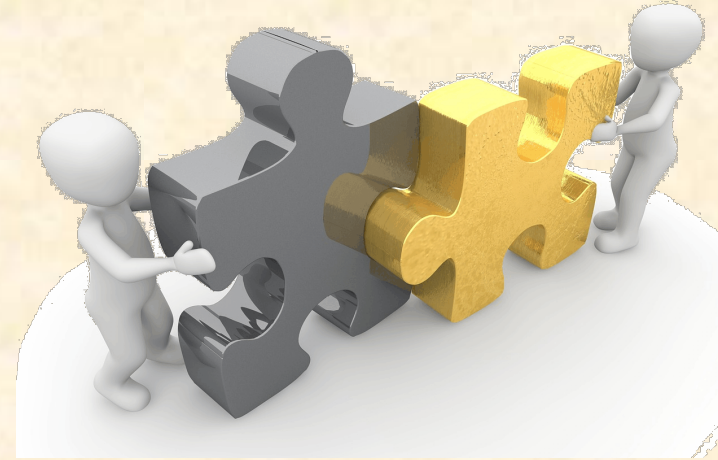
OB迪 Parameter ID message dictionary (CAN Network Messages)

[https://en.wikipedia.org/wiki/OBD-II_PIDs]

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 edge 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 can't trace Integration Tests back to design

Solutions that might fix the problem without breaking anything



Essential

Hoping This Works