InterACt

a Tool for Unit Test based Integration of Component-based Software Systems

(GitHub)

Nils Wild
wild@swc.rwth-aachen.de

Horst Lichter
lichter@swc.rwth-aachen.de

IARIA SWC RWTH AACHEN UNIVERSITY

Birth

Early Interest in Computers
Lego Robots (RCX & NXT)
Javaland & DOAG

1994

1. Business

Webdevelopment for
Small Businesses around
My Hometown

2010

2. Business

Sinnovate GmbH
Digitialization for Childcare
Institutions

2016

Research Assistant

Test Automation
RWTH Aachen University
Research Group Software
Construction

2019

1. REQUIREMENTS

Analyse the requirements and plan the project

2. DESIGN

Make design and architecture decisions

3. IMPLEMENTATION

Implement the requirements according to design and architecture decisions

4. TESTING

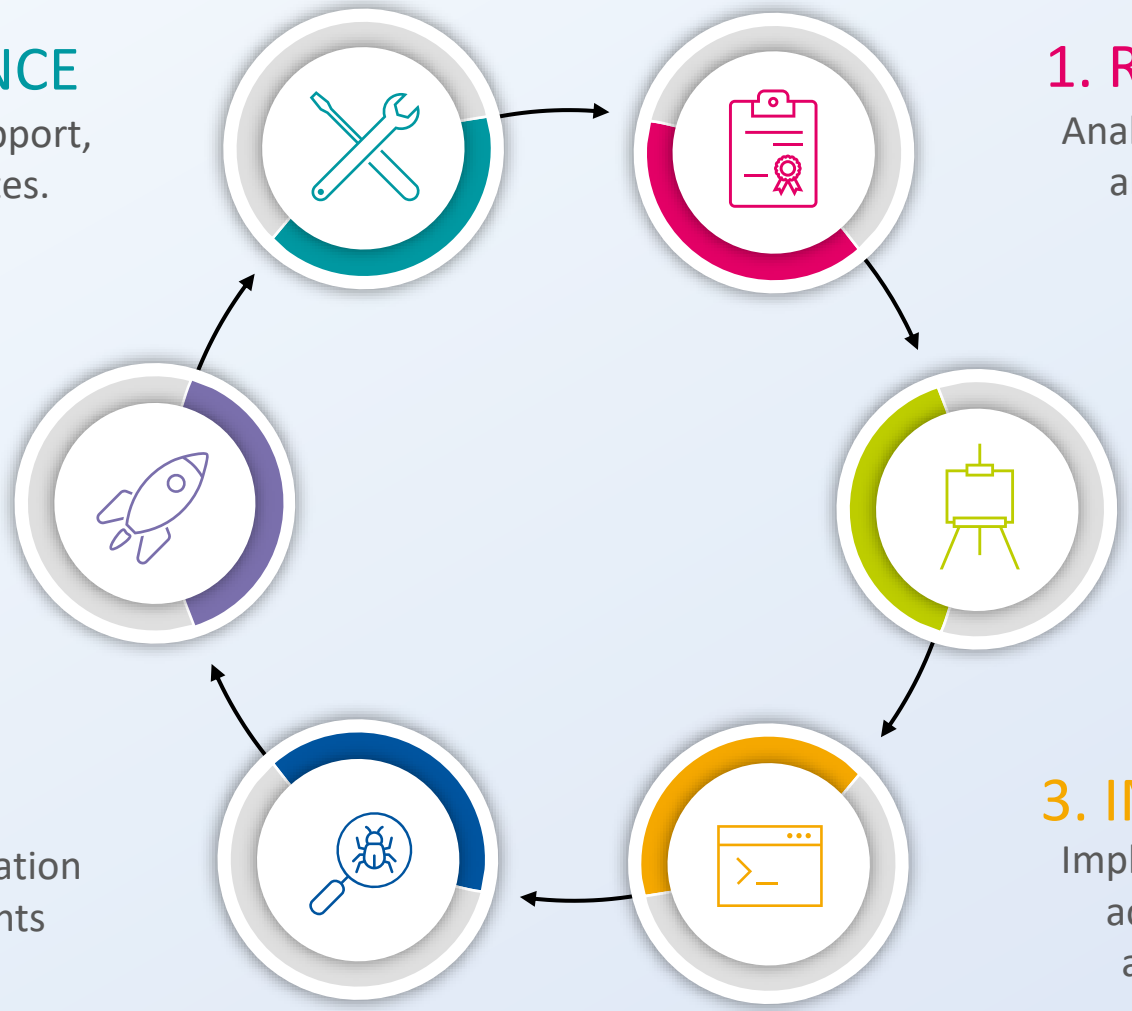
Check that the implementation meets all the requirements

5. DEPLOYMENT

Release & deploy to production

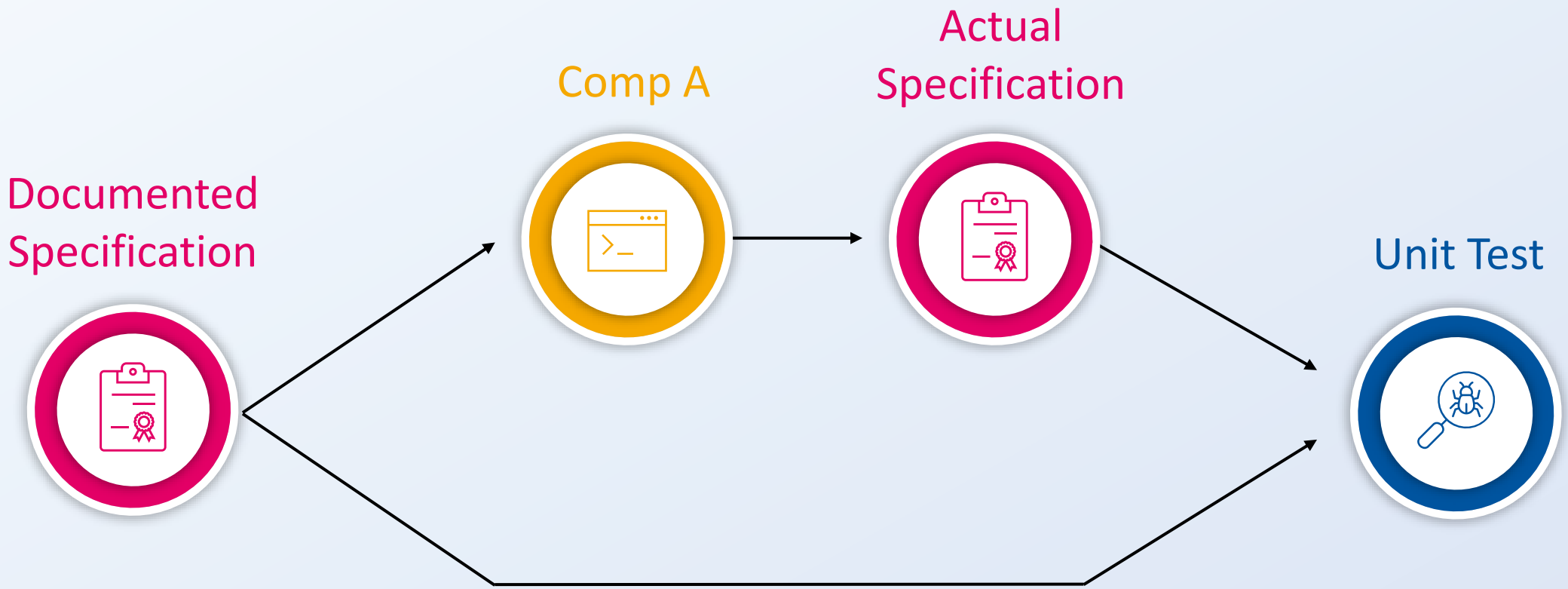
6. MAINTENANCE

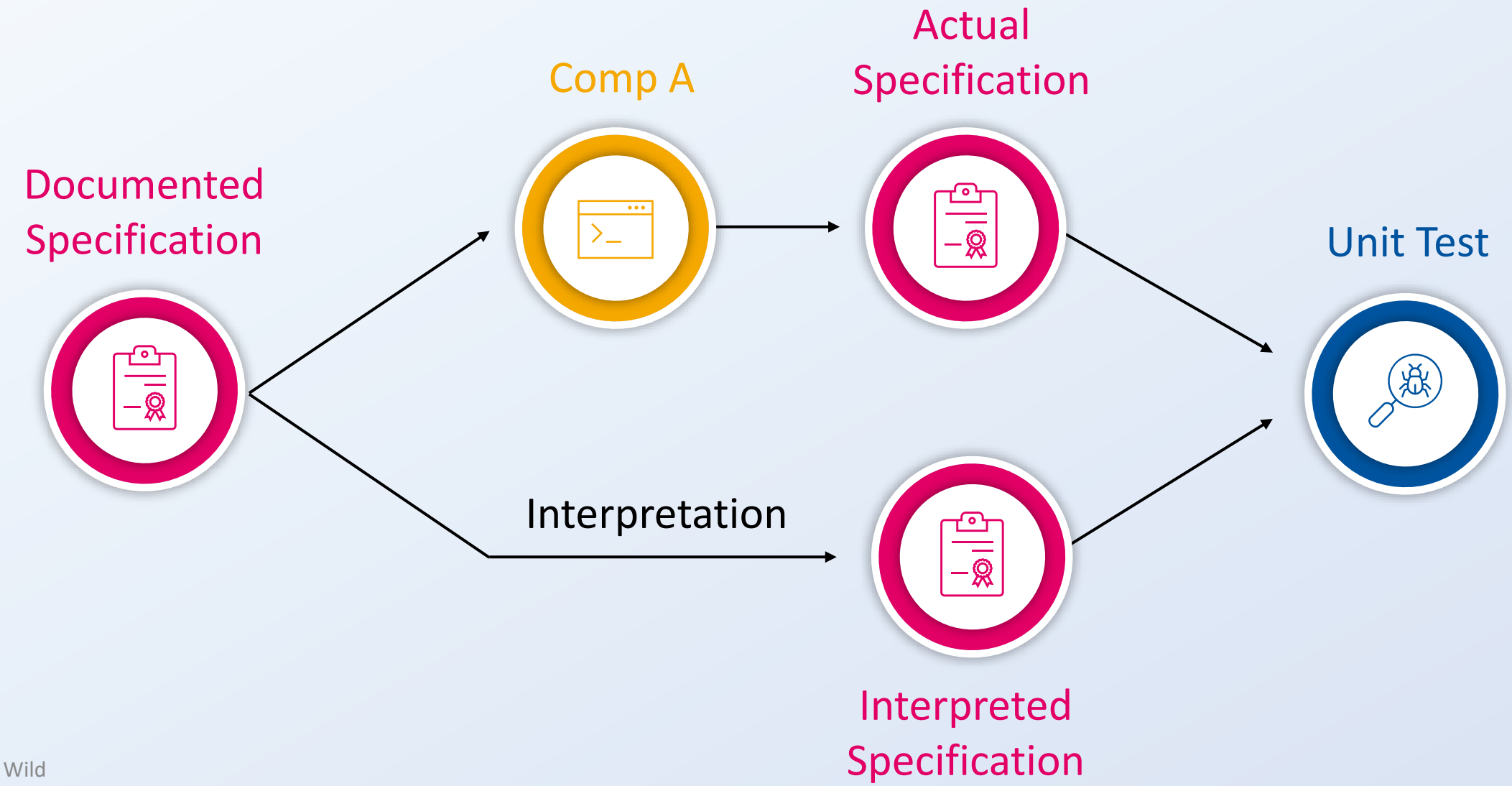
Ensuring continued support, upgrades, and updates.

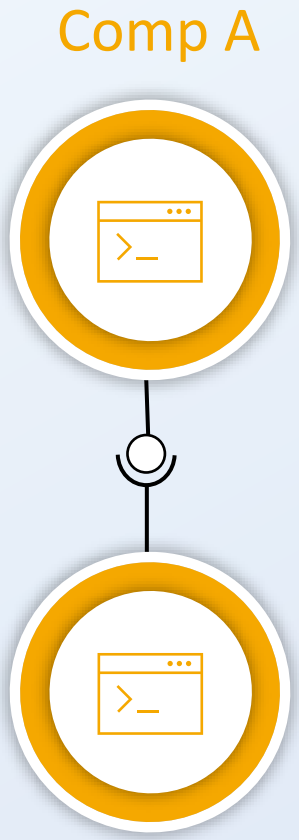


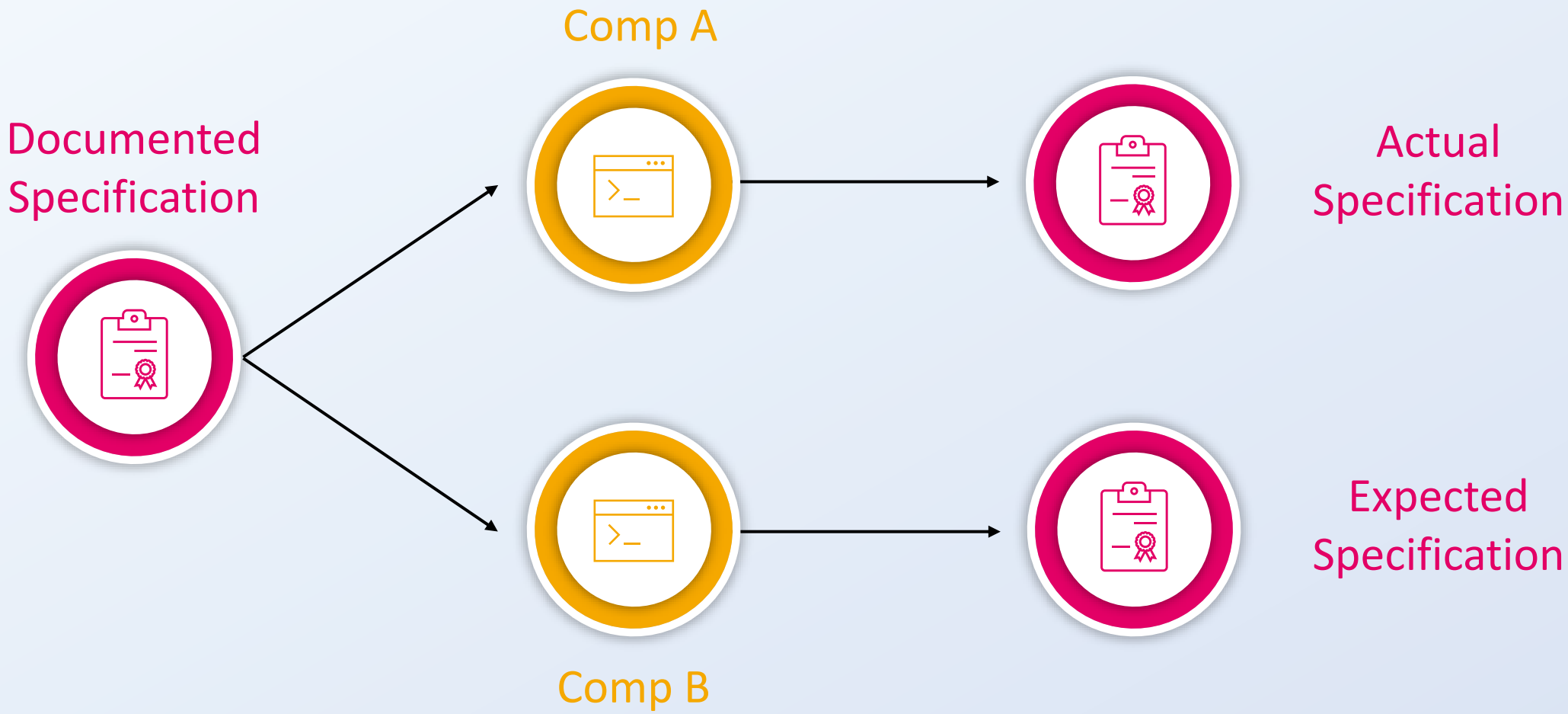
Documented Specification

Unit Test


Documented Specification
Comp A
Actual Specification
Unit Test

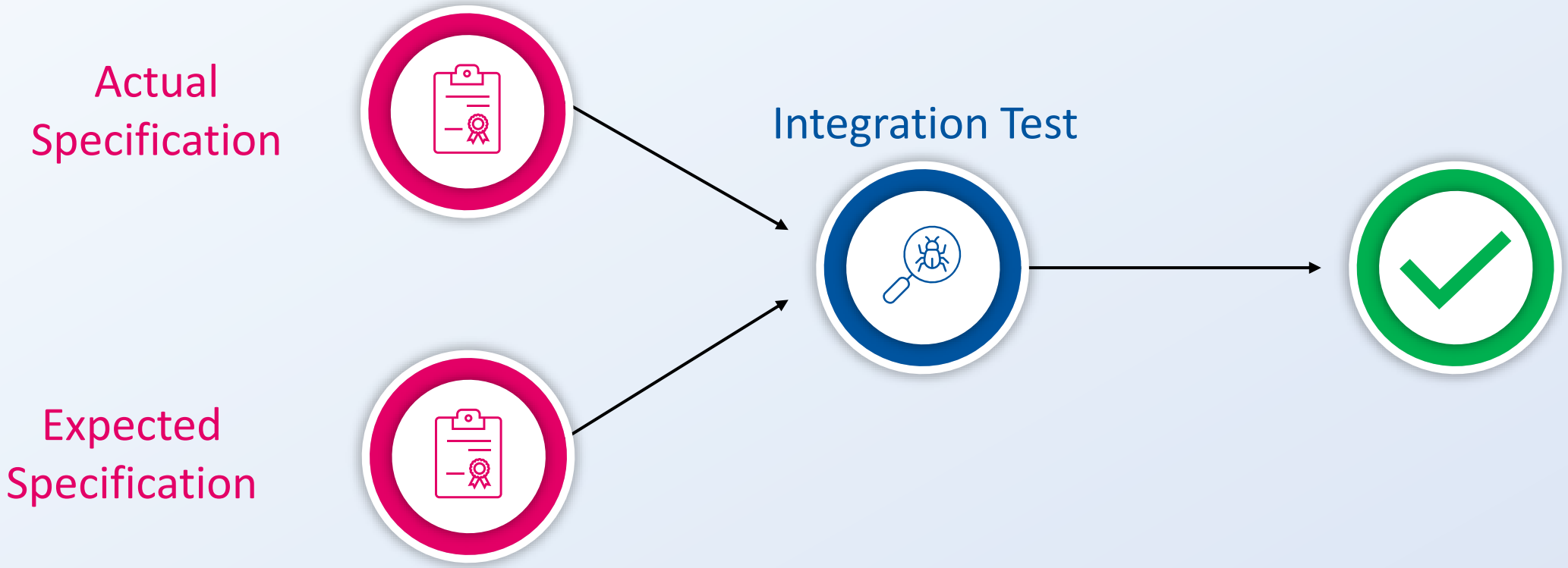

Documented Specification
Comp A
Actual Specification
Unit Test
Interpretation
Interpreted Specification

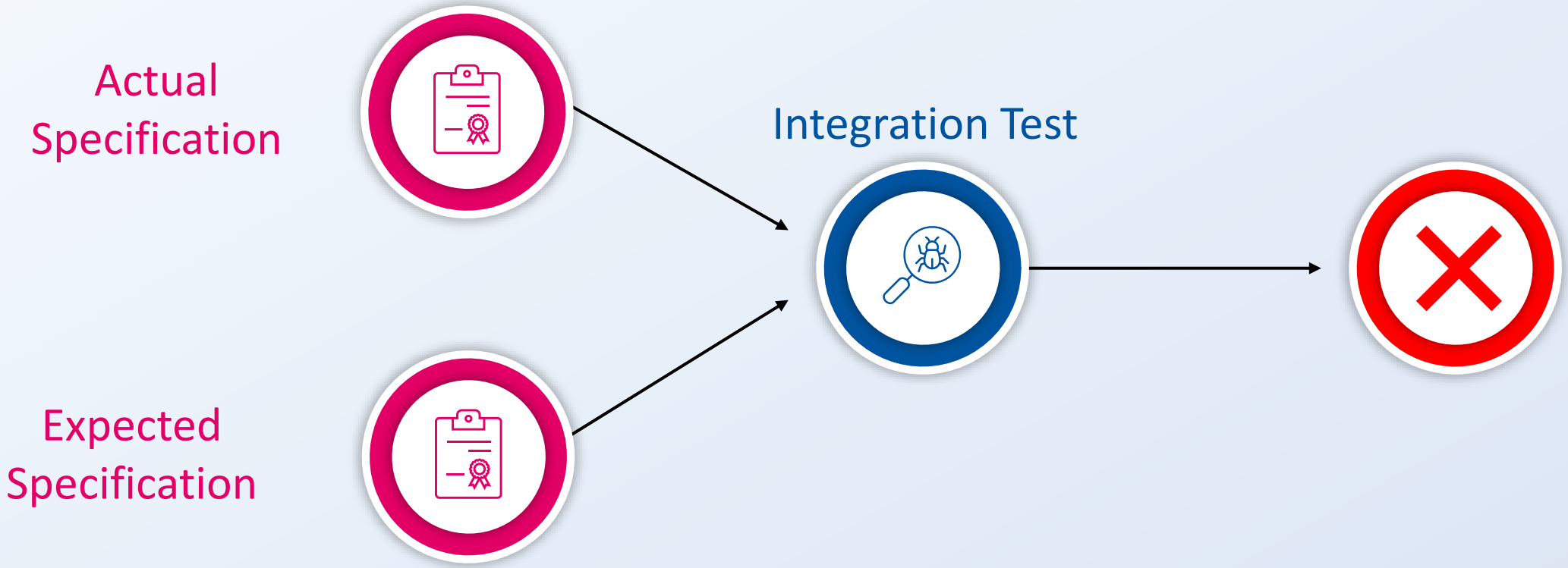



Comp A
Documented Specification
Actual Specification
Expected Specification
Comp B

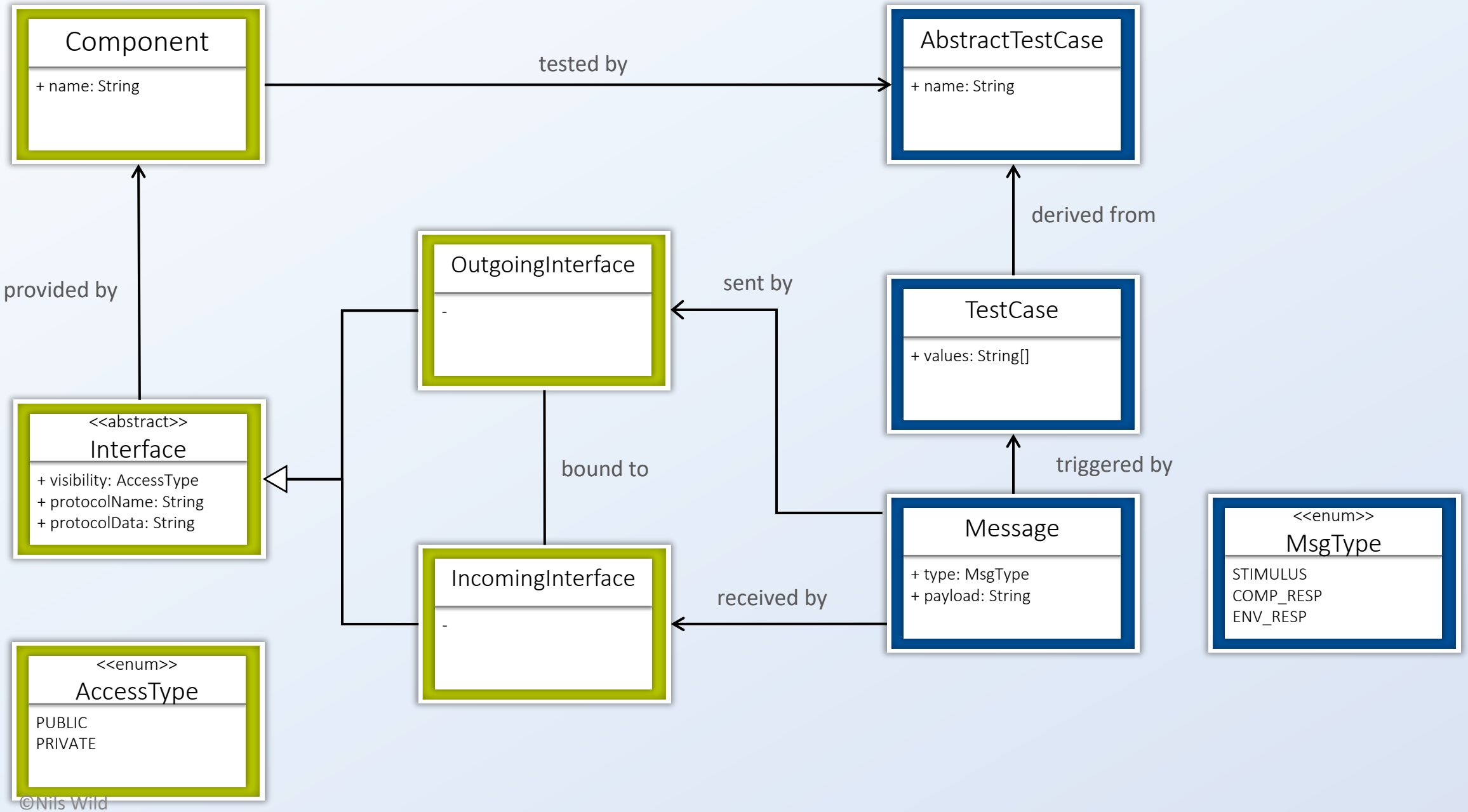
Actual Specification

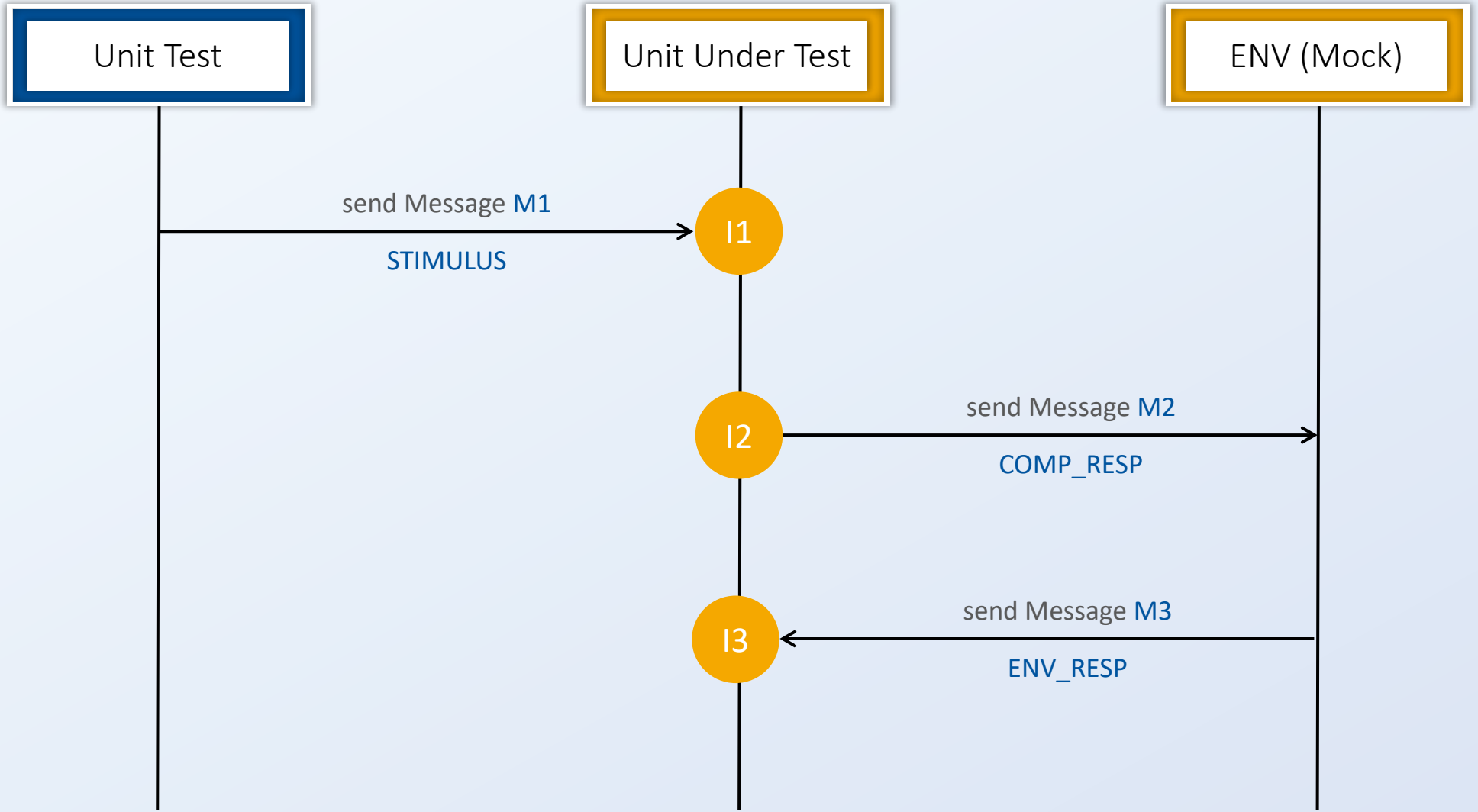
Expected Specification

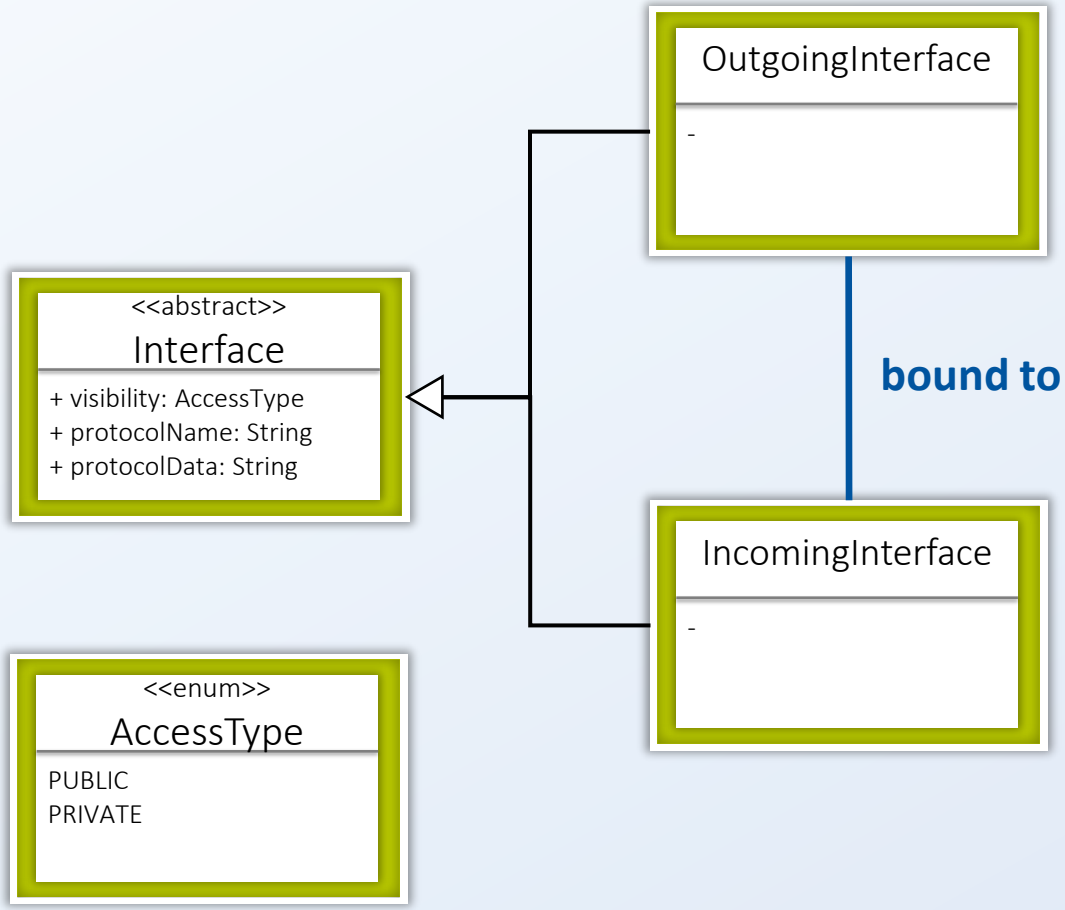

Actual Specification
Expected Specification
Integration Test


Actual Specification
Expected Specification
Integration Test

Automated Iterative Integration Testing based on Unit Test Cases


Component
+ name: String
tested by
AbstractTestCase
+ name: String
derived from
provided by
OutgoingInterface
-
sent by
TestCase
+ values: String[]
<<abstract>>
Interface
+ visibility: AccessType
+ protocolName: String
+ protocolData: String
bound to
triggered by
Message
+ type: MsgType
+ payload: String
<<enum>>
MsgType
STIMULUS
COMP_RESP
ENV_RESP
IncomingInterface
-
received by
<<enum>>
AccessType
PUBLIC
PRIVATE


Unit Test
Unit Under Test
ENV (Mock)
send Message M1
STIMULUS
I1
I2
send Message M2
COMP_RESP
I3
send Message M3
ENV_RESP

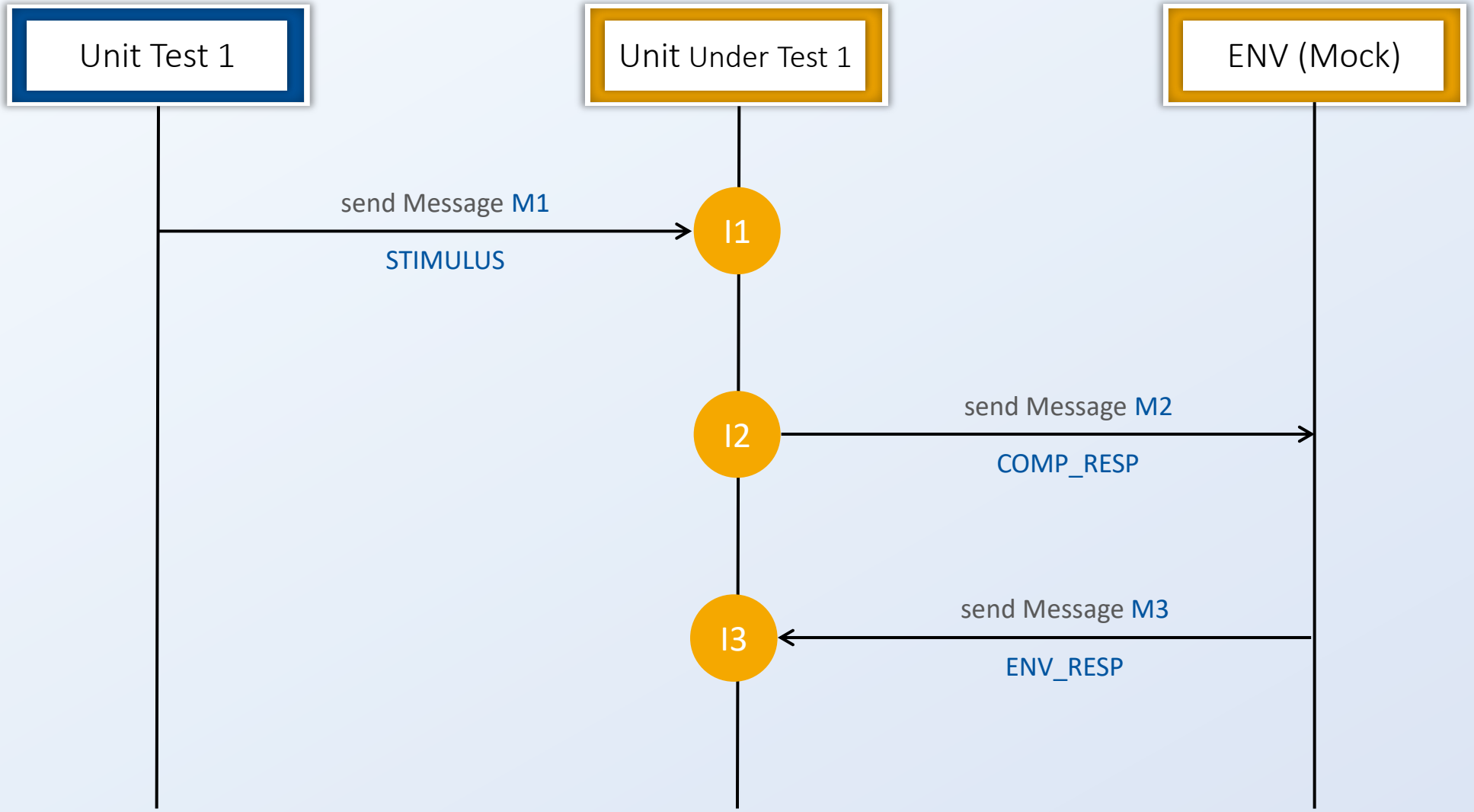


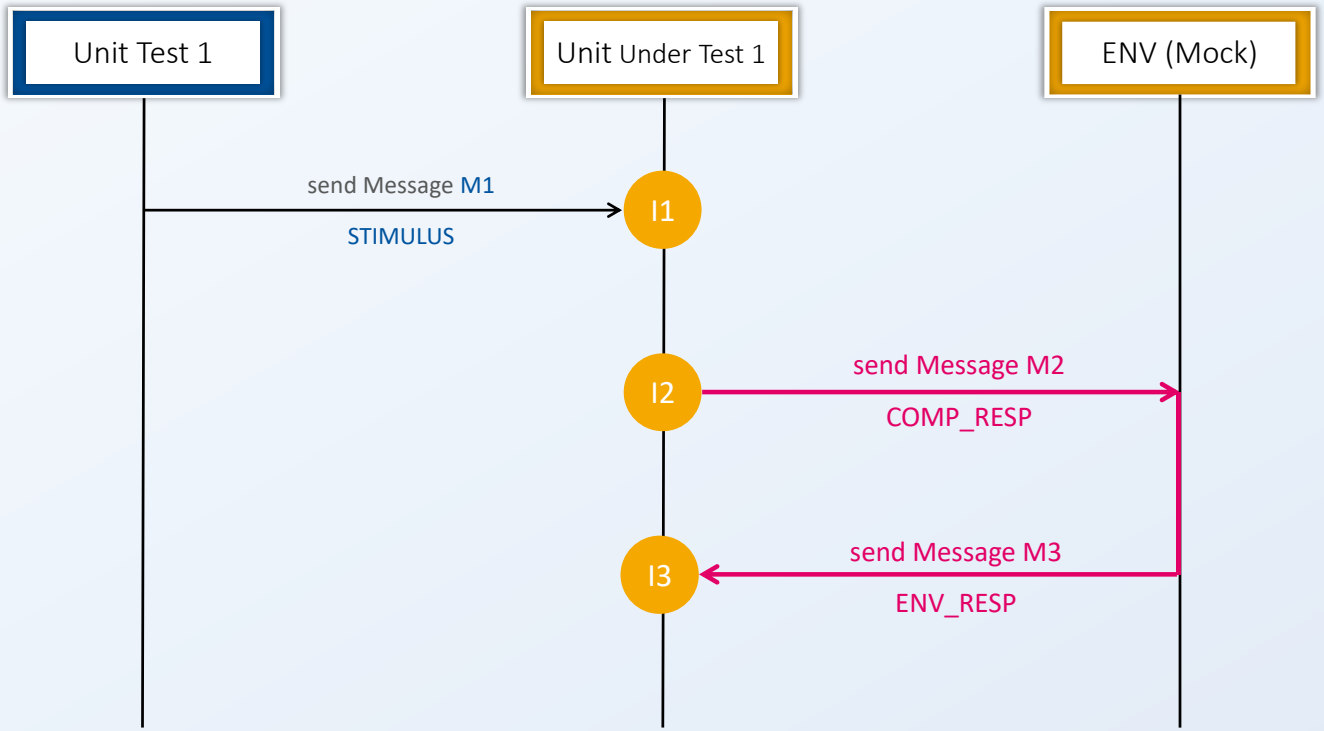
Interpretation Faults

Missing Function Fault*

Extra Function Fault*

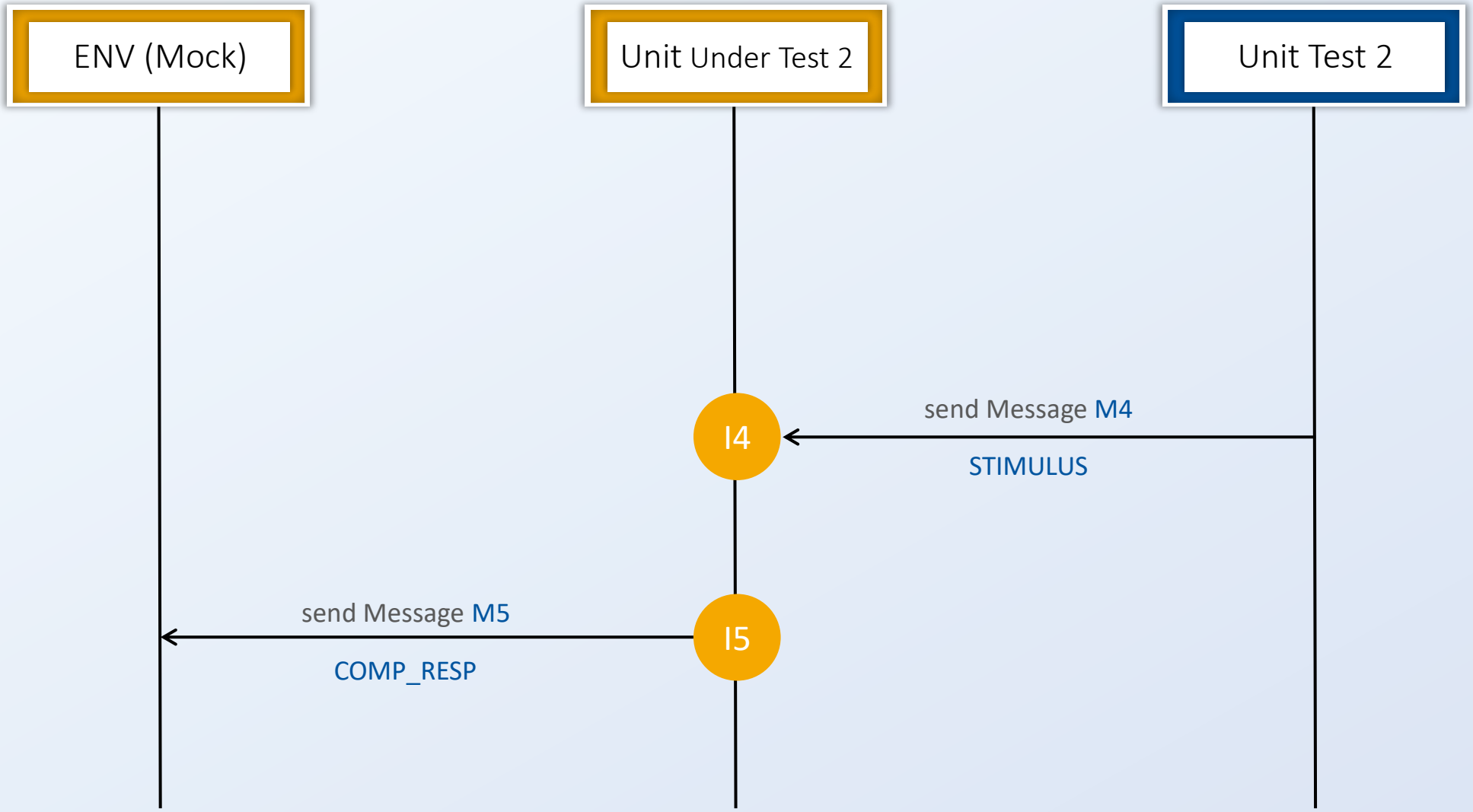
Test Gap

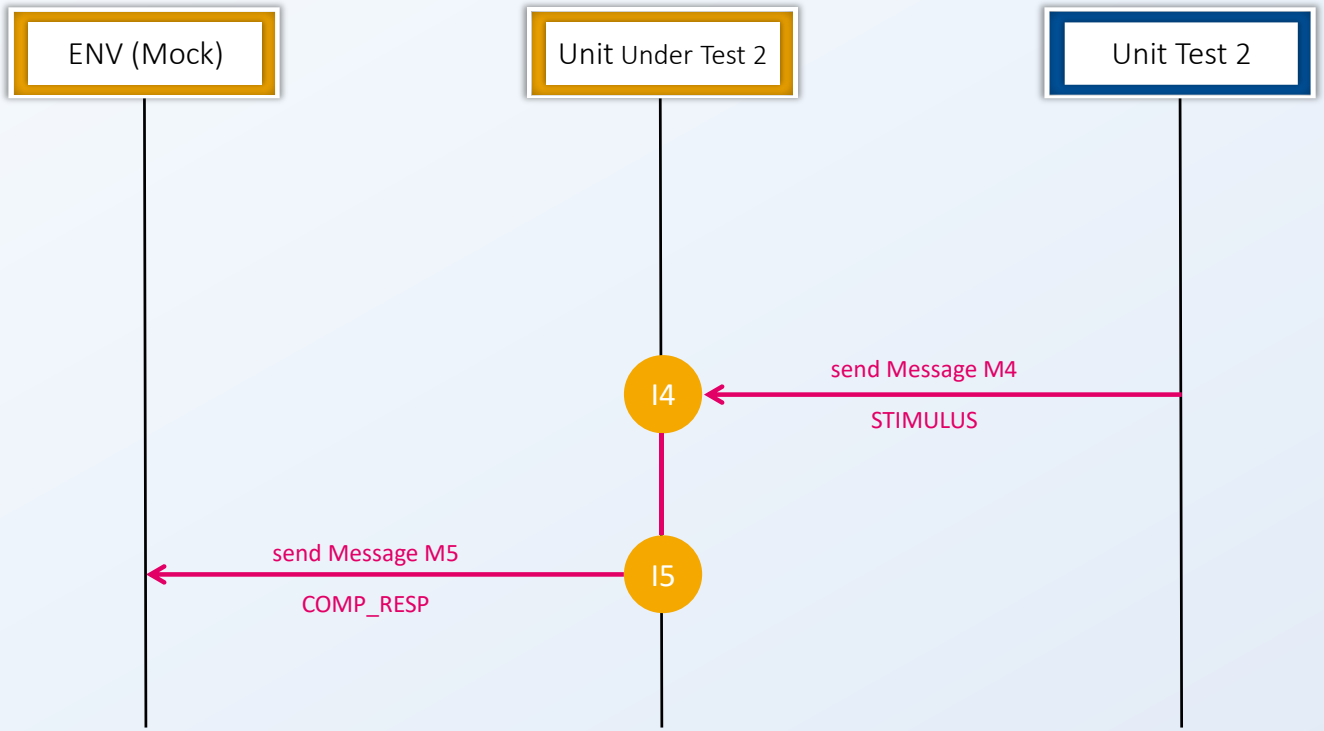

Unit Test 1
Unit Under Test 1
ENV (Mock)
send Message M1
STIMULUS
I1
I2
send Message M2
COMP_RESP
I3
send Message M3
ENV_RESP



Expected Specification

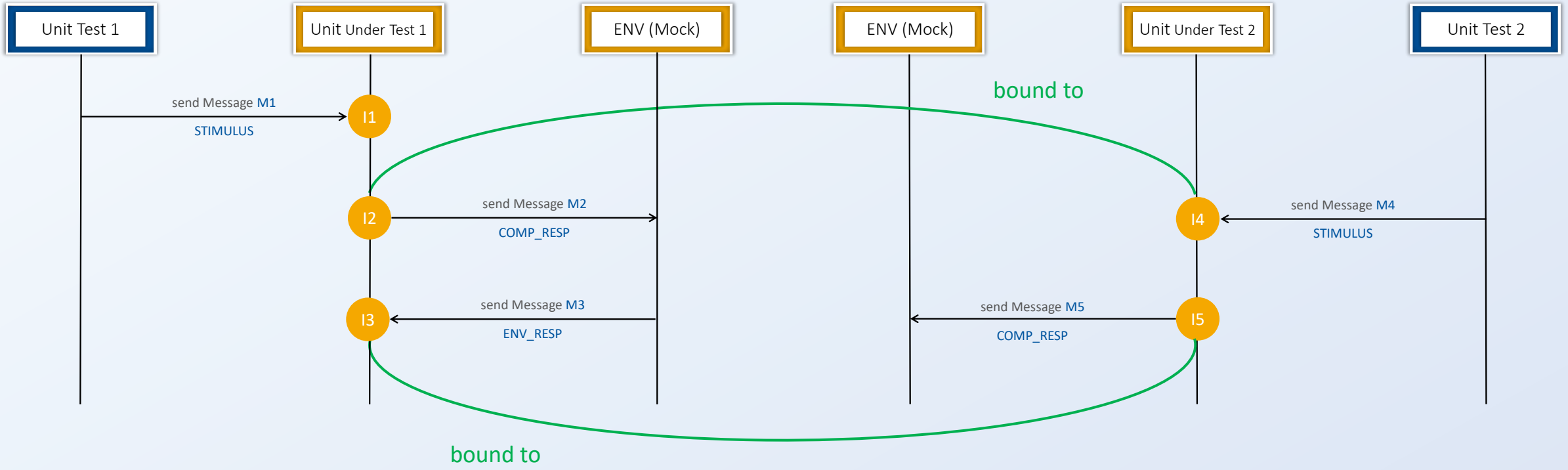
When a message like M2 is sent to the environment, a message like M3 is expected as response with respect to the unit test's expectations.

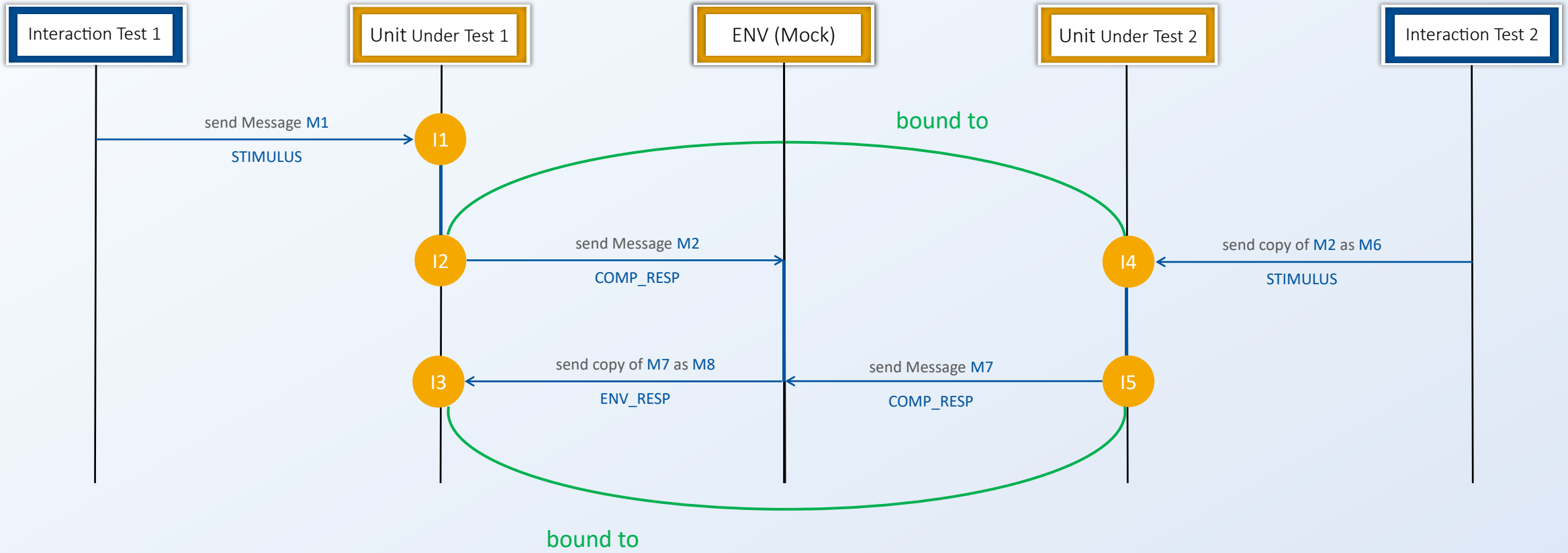

ENV (Mock)
Unit Under Test 2
Unit Test 2
I4
send Message M4
STIMULUS
I5
send Message M5
COMP_RESP



Actual Specification

When a message like M4 - with respect to the units behavior - is received, a message like M5 is sent with respect to the unit test's expectations.


Unit Test 1
Unit Under Test 1
ENV (Mock)
ENV (Mock)
Unit Under Test 2
Unit Test 2
bound to
send Message M1
STIMULUS
I1
I2
send Message M2
COMP_RESP
I4
send Message M4
STIMULUS
I3
send Message M3
ENV_RESP
send Message M5
COMP_RESP
I5
bound to



Interpretation Faults

Wrong Function Fault*

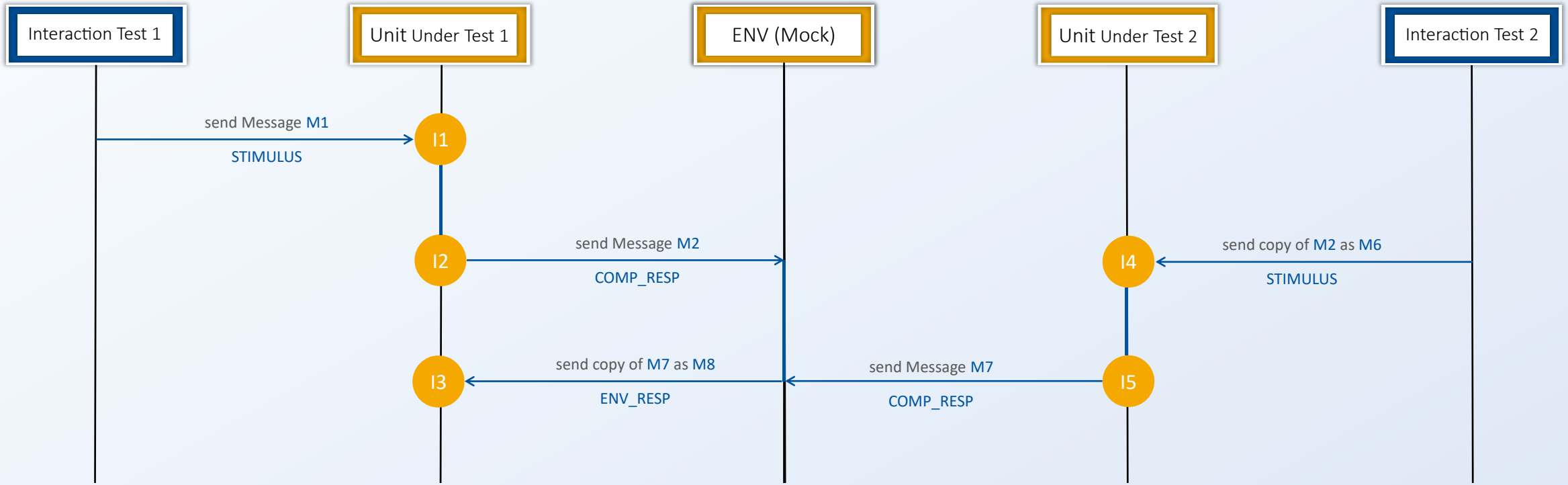
Test Gap

Miscoded Call Faults

Missing Call Fault*

Test Gap

Interface Faults*



Interpretation Faults

Wrong Function Fault*

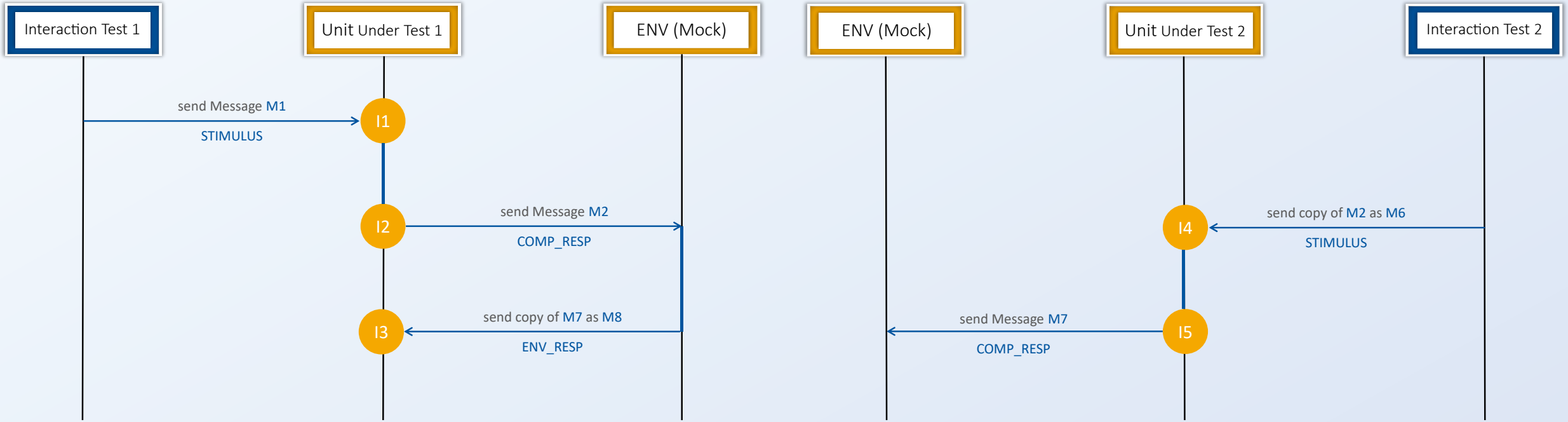
Test Gap

Miscoded Call Faults

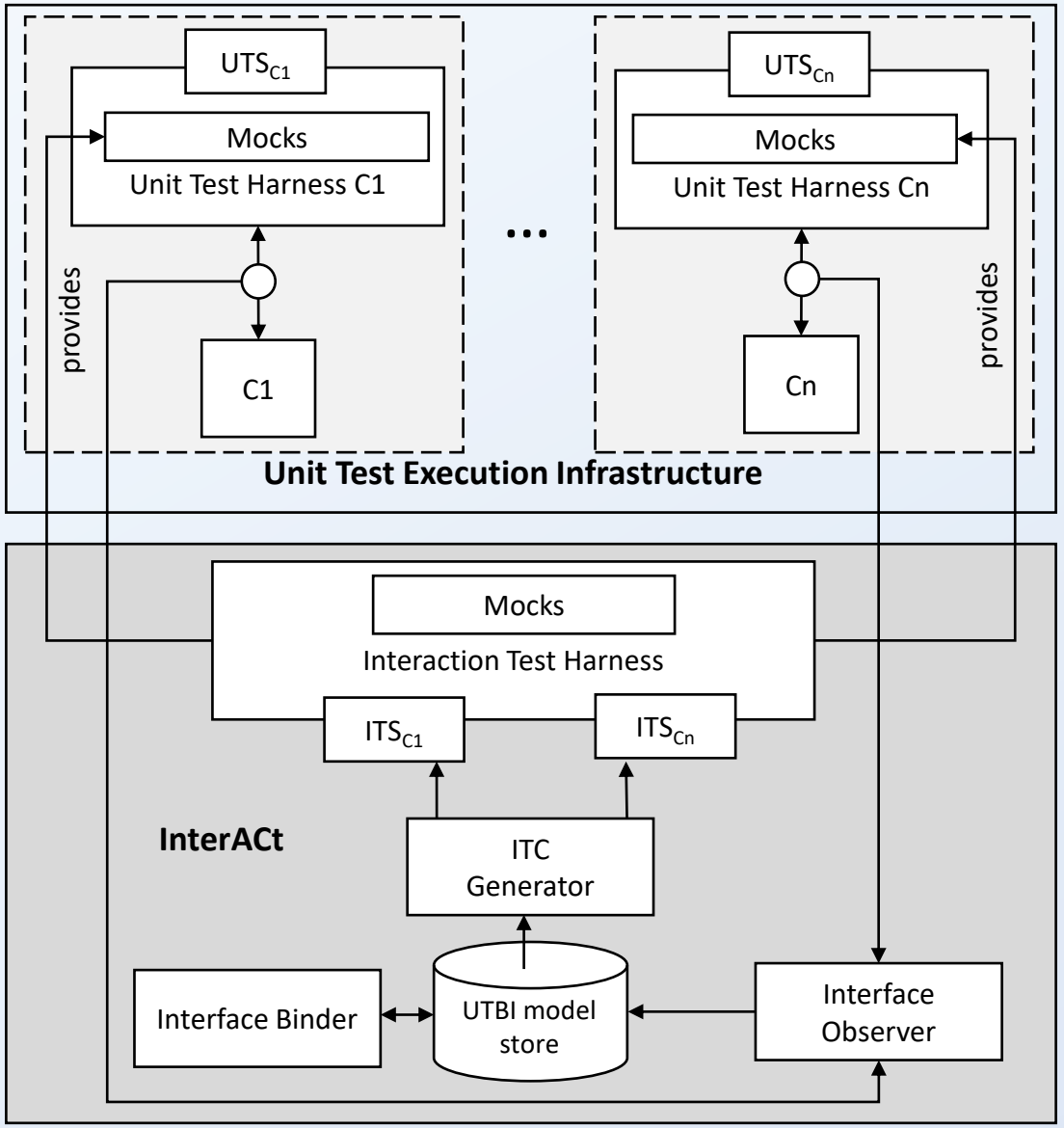
Missing Call Fault*

Test Gap

Interface Faults*


Interaction Test 1
Unit Under Test 1
ENV (Mock)
ENV (Mock)
Unit Under Test 2
Interaction Test 2
send Message M1
STIMULUS
I1
I2
send Message M2
COMP_RESP
I3
send copy of M7 as M8
ENV_RESP
send Message M7
COMP_RESP
I5
I4
send copy of M2 as M6
STIMULUS

InterACt


UTS$_{C1}$
Mocks
Unit Test Harness C1
provides
C1
UTS$_{Cn}$
Mocks
Unit Test Harness Cn
provides
Cn
Unit Test Execution Infrastructure
Mocks
Interaction Test Harness
ITS$_{C1}$
ITS$_{Cn}$
InterACt
ITC Generator
Interface Binder
UTBI model store
Interface Observer

```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
@Import(InterACtObserverConfig.class)
public class MyUnitTestClass {
        ...

        @InterACtTest
        @CsvSource("Stimulus,MockResponse,AnotherMockResponse,ExpectedValue“)
        void aUnitTest(
                @AggregateWith(StimulusAggregator.class) Type stimulus,
                @AggregateWith(MockAggregator.class) Type response1,
                @AggregateWith(MockAggregator.class) Type response2,
                SimpleType expectedValue
        ) {
                ...
        }
}
```

Run the Unit Tests

Store the Components Behavior

Extract the Expectations

Search for Paths matching the Expectations

Generate Validation Plans

Execute Interaction Tests

PROS

- Reuse existing unit-tests
- Focus on unit-testing leads to test early
- Adapting to architectural and behavioral changes
- Resource utilization capped at max resources for unit-testing

CONS

- Requires unit-test suites with high coverage
- Can not cover all integration scenarios
- Currently no support for pass-through APIs and state expectations
- Requires special test engineering