TDD with JUnit

Test-Driven Development (TDD)

- TDD Life Cycle
- TDD vs DLP (Debug Later Programming)
- Why TDD is matter
- Unit Testing with F.I.R.S.T
- Code and Test Coverage
- Structure of good unit testing (GUT)

Unit Testing with JUnit

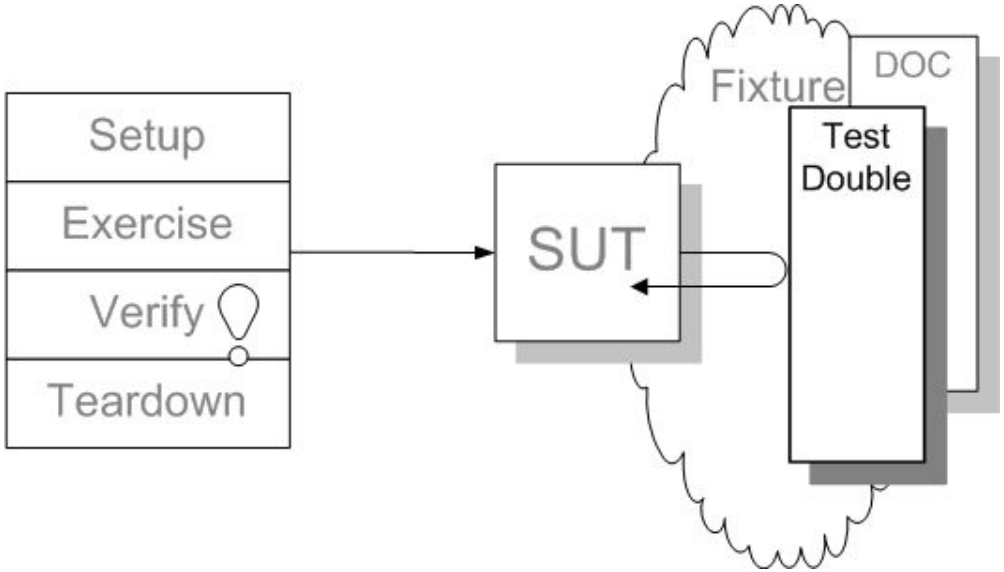
- JUnit lifeCycle
- Assertion
- Data-Driven Test with JUnit
- JUnit features
- Timeout
- Conditional
- **Category**
- **Suite**
- Running testing

Test Double

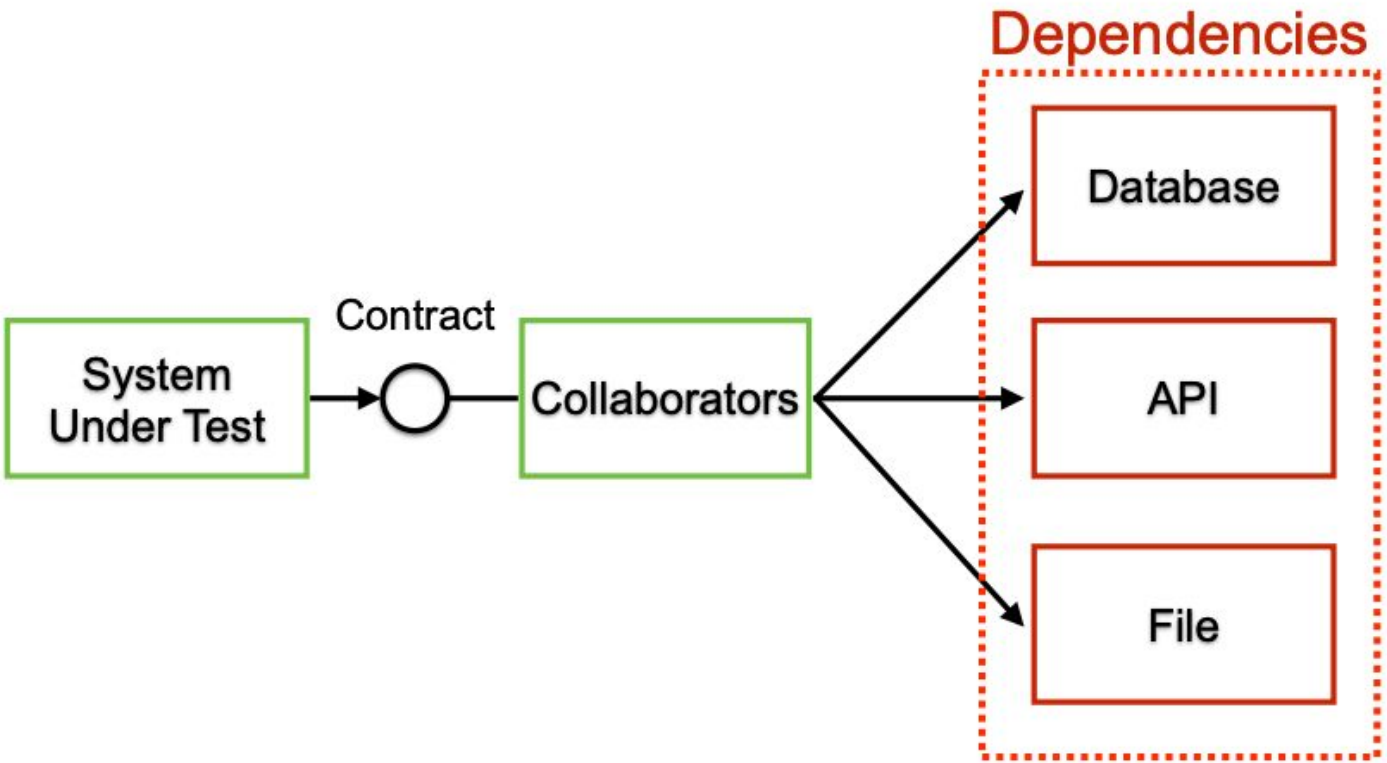
- **Dummy**
- **Stub**
- **Spy**
- **Mock**
- **Fake**

Test Double

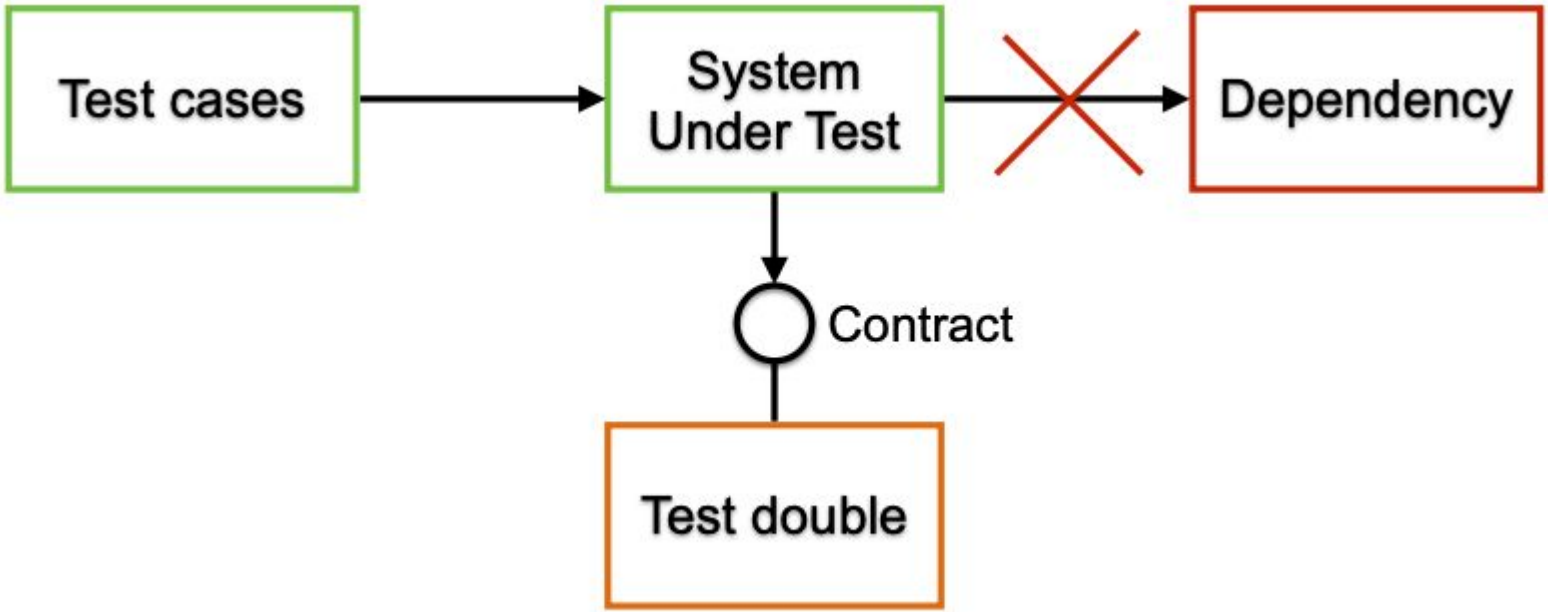
How can we verify logic independently ?
How can we avoid Slow tests ?



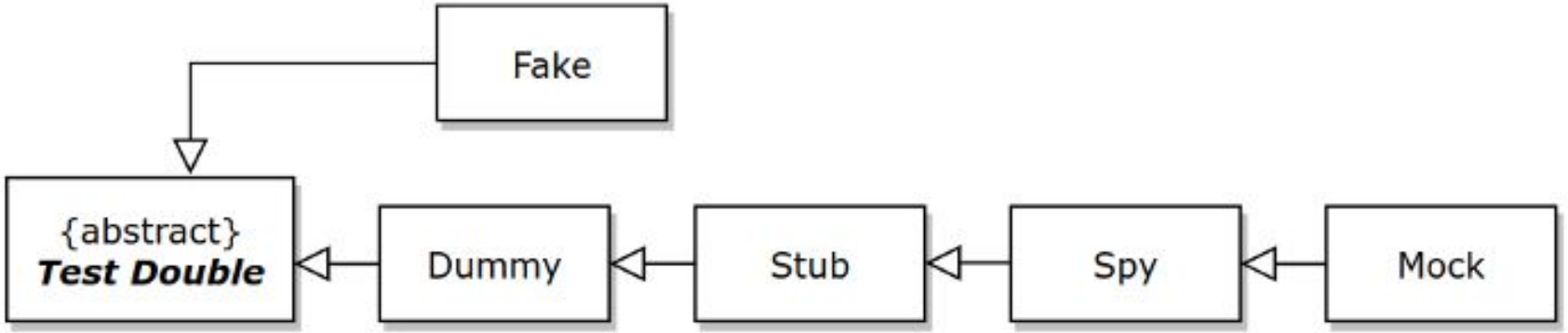
Production Code



With Test double



The five types of Test Doubles



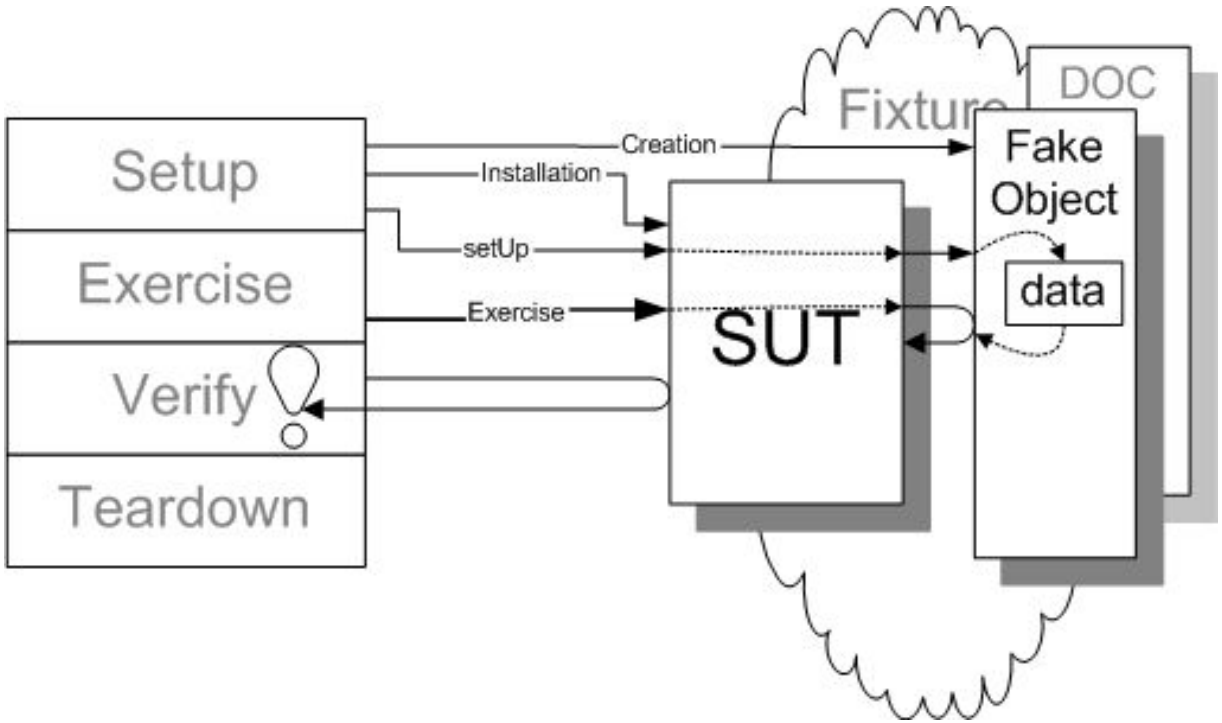
A Test Double is an object that can stand-in for a real object in a test, similar to how a stunt double stands in for an actor in a movie.

https://martinfowler.com/bliki/TestDouble.html

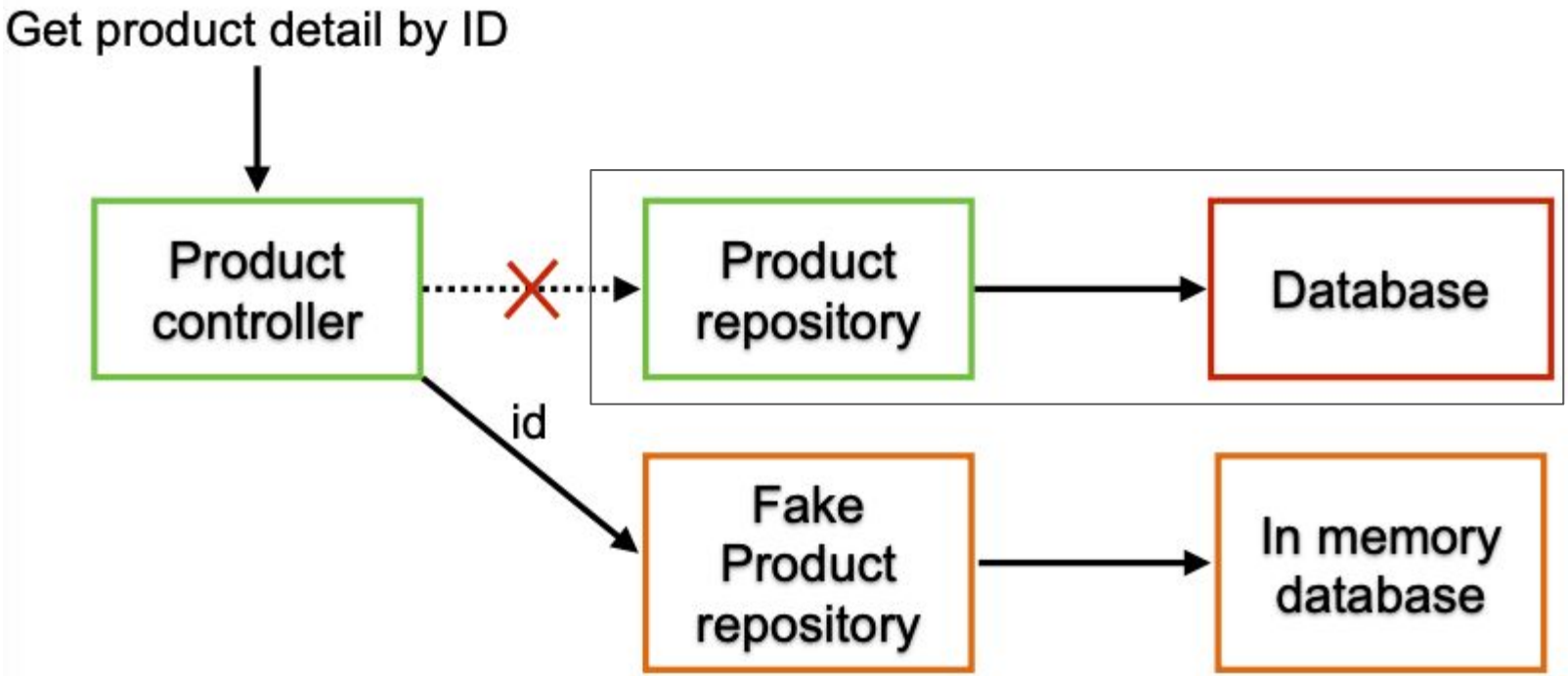
Fake

Working with implementation but take some shortcut
Not suitable for production
Use with read and write operations

E.g. In-memory database, Fake API server

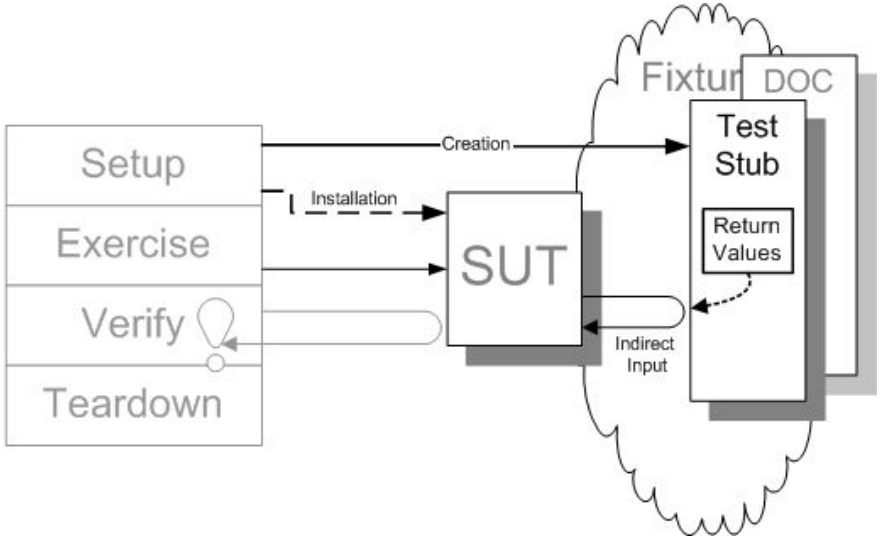

Setup
Exercise
Verify
Teardown
Creation
Installation
setUp
Exercise
SUT
Fixture
DOC
Fake
Object
data

Fake

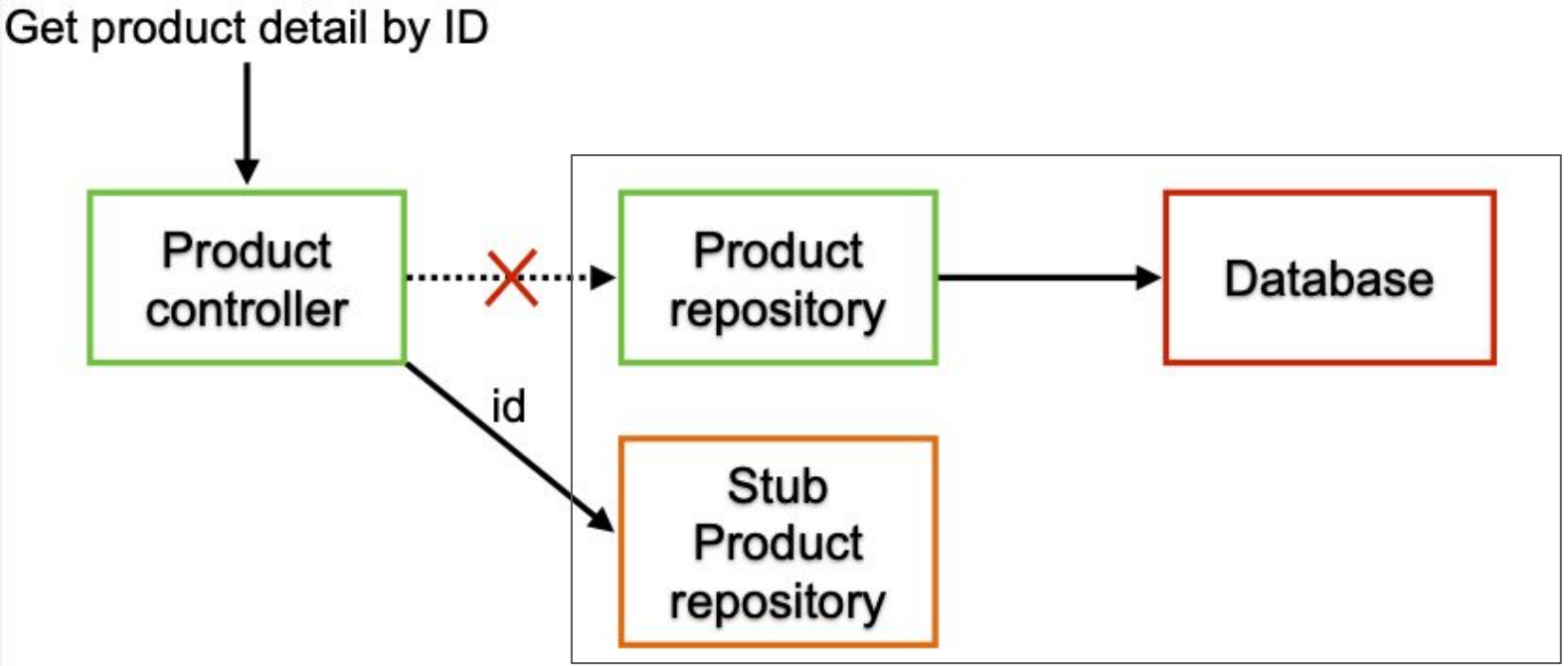

Get product detail by ID
Product controller
Product repository
Database
id
Fake Product repository
In memory database

Stub

Provide answers to calls made during the test
A double with hardcode return values



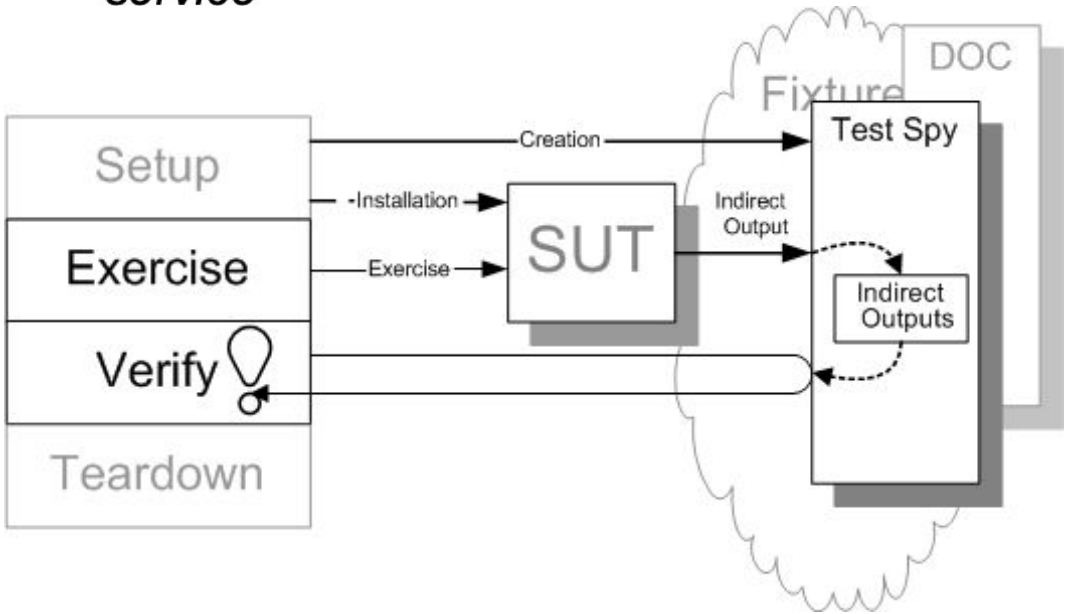
Stub


Get product detail by ID
Product controller
id
Product repository
Database
Stub Product repository

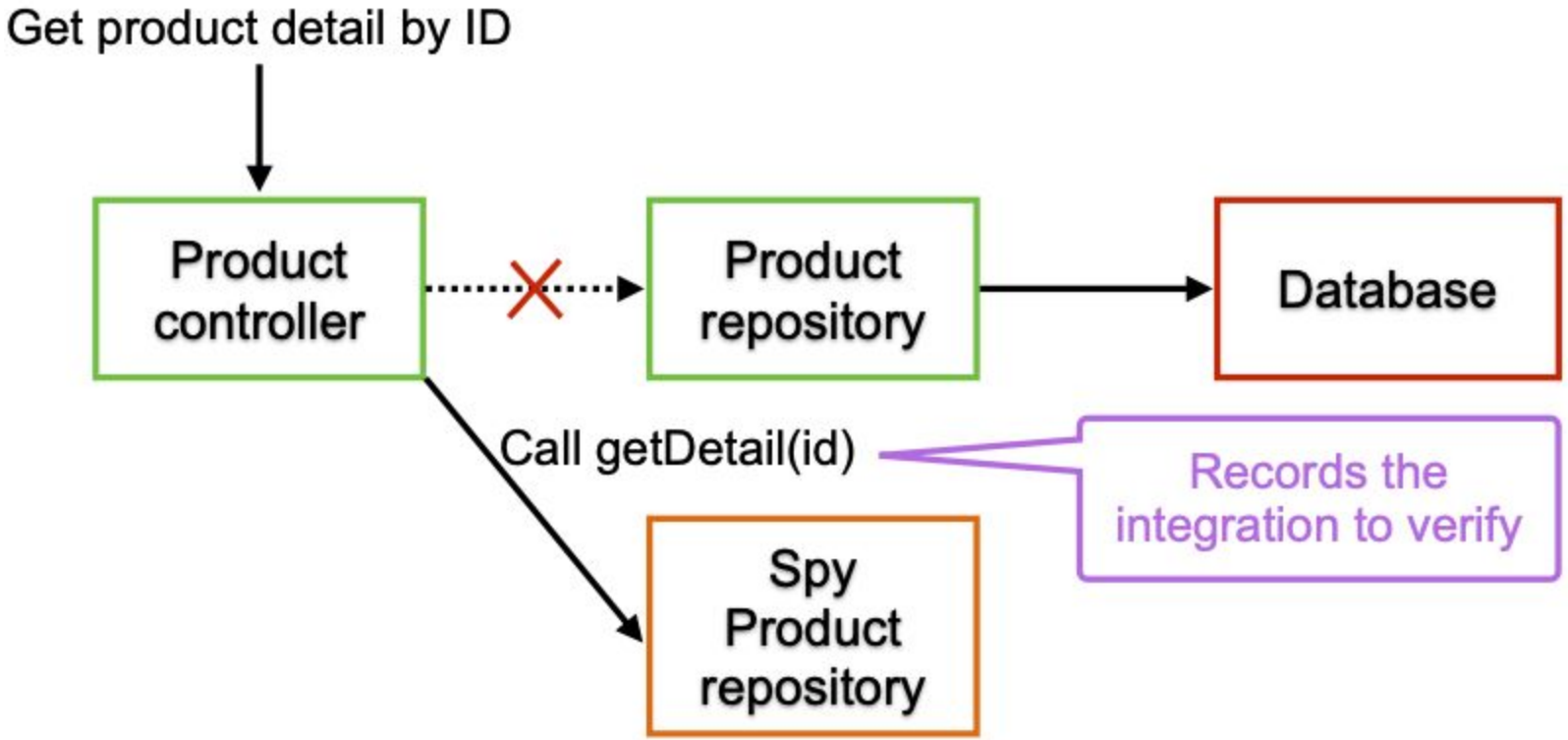
Spy

Like stub
Record some information based on how its called

E.g. how many message it was sent via email service

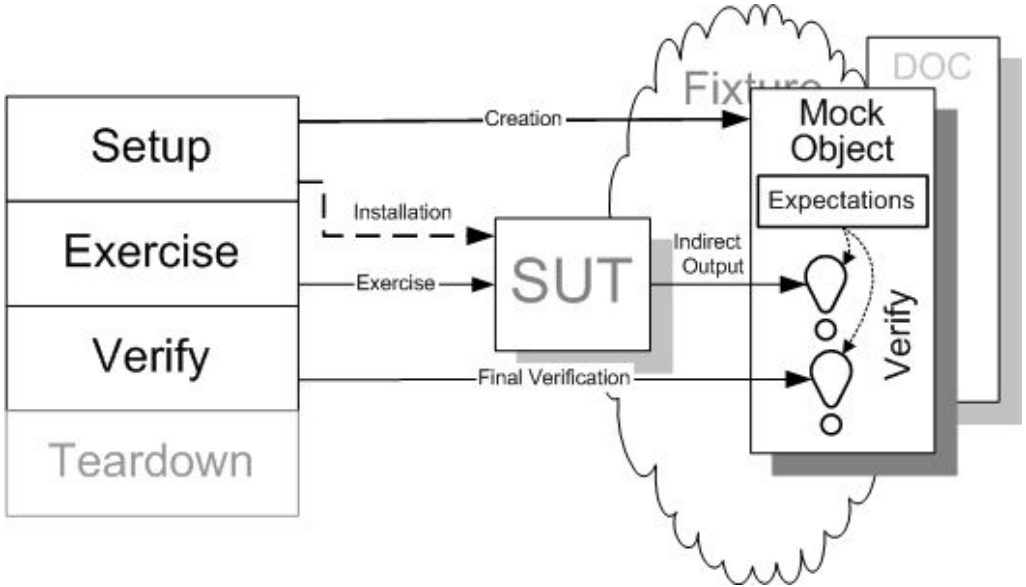


Spy

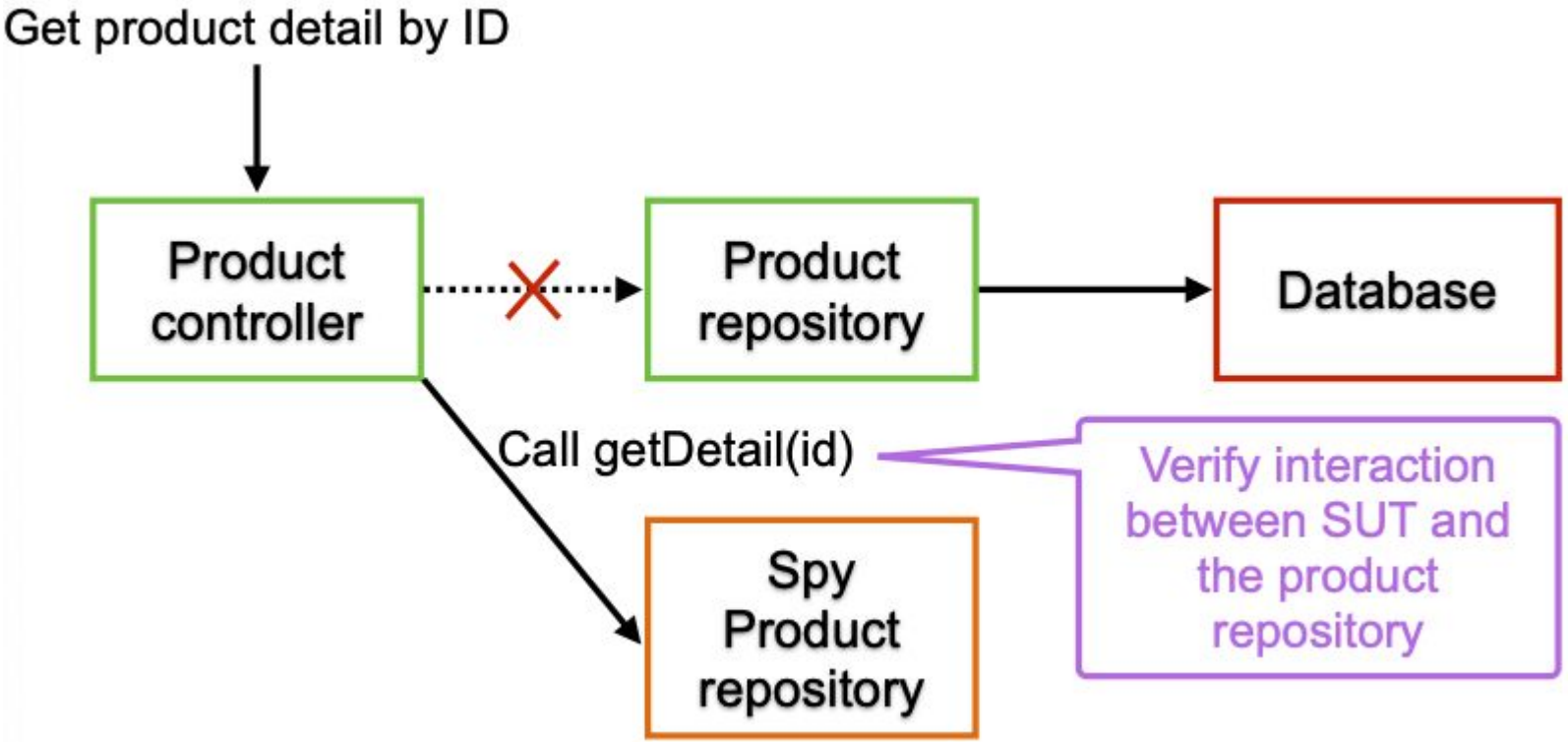

Get product detail by ID
Product controller
Product repository
Database
Call getDetail(id)
Records the integration to verify
Spy Product repository

Mock object

Pre-programmed with expectations with spec
Mock object can throw an exception if receive a call that don't expect



Mock object

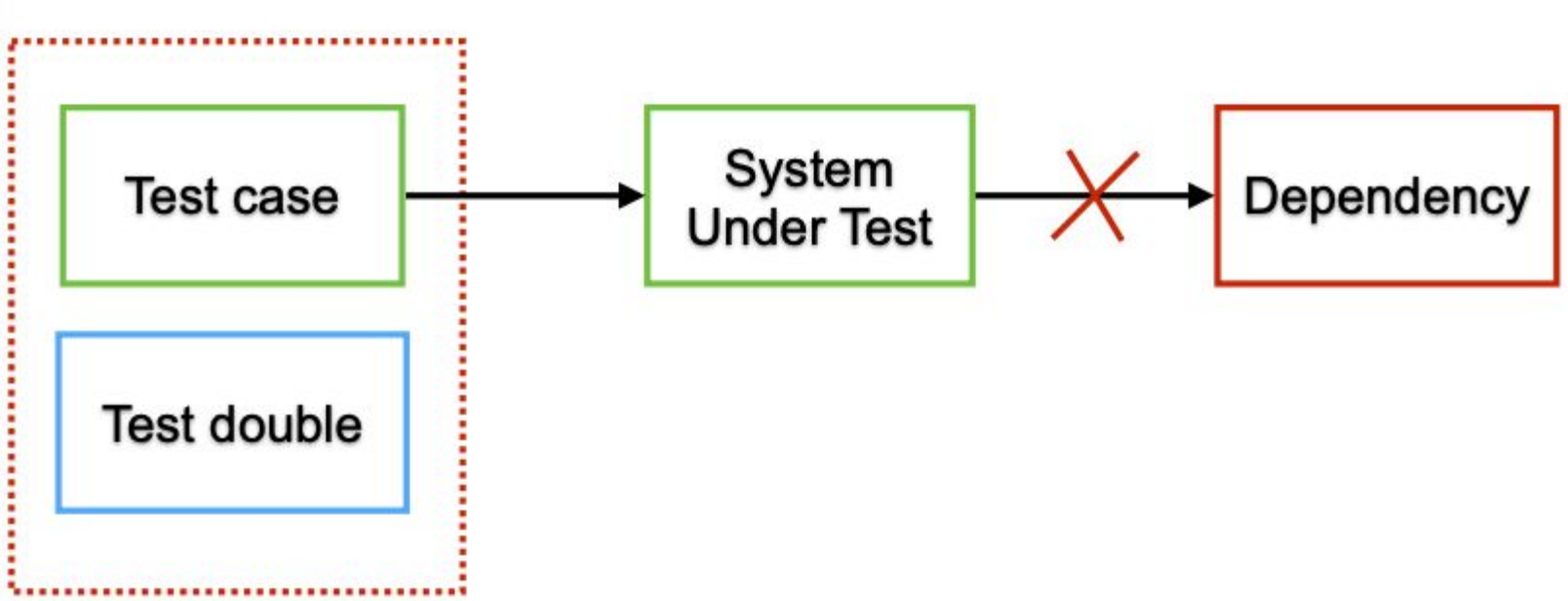

Get product detail by ID
Product controller
Product repository
Database
Call getDetail(id)
Spy Product repository
Verify interaction between SUT and the product repository

Dummy object

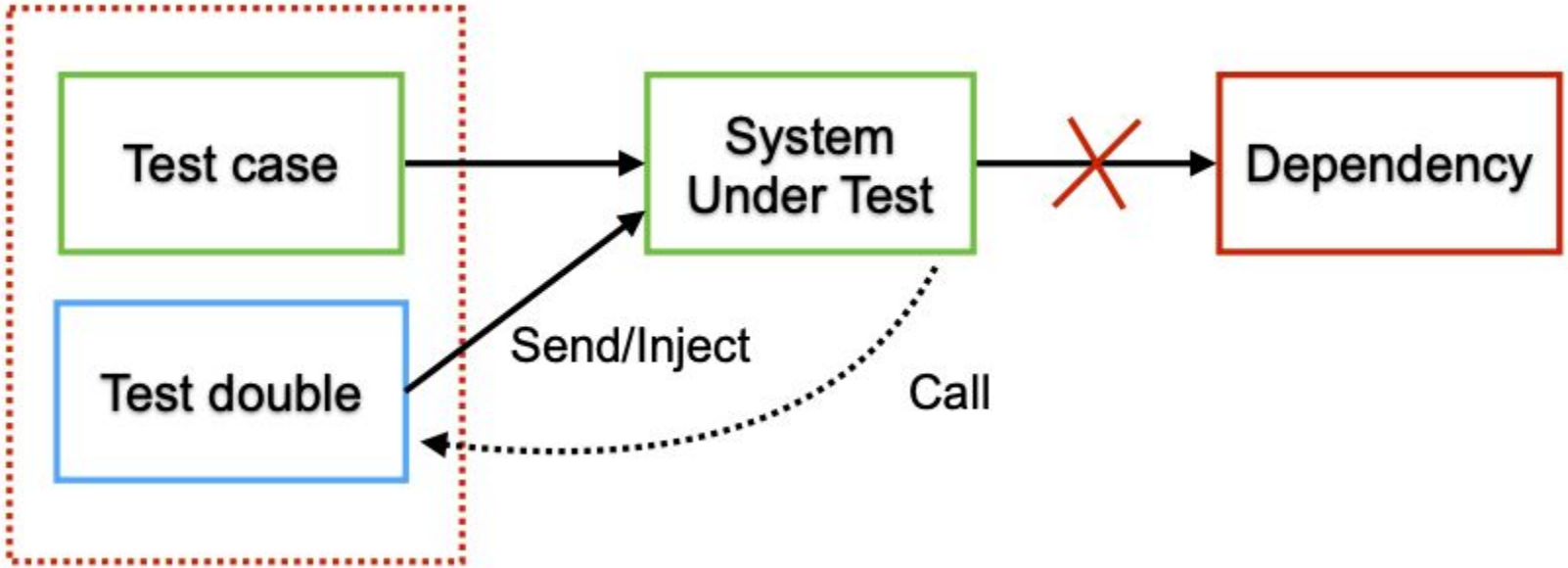
Passed around but never actually used
Used to fill parameter lists

Test double ?

Create test double



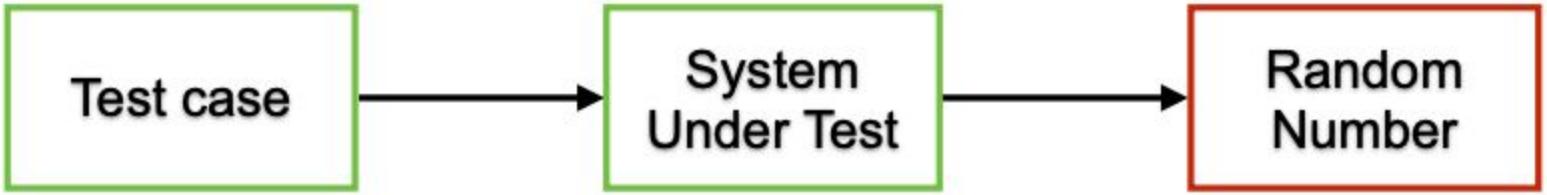
Send/inject test double to SUT



Workshop

Test random number = 5 ?

Workshop



Test random number must called 1 time ?

Mockito

Using Mockito with JUnit 5

```
@ExtendWith(MockitoExtension.class)
public class DemoWithMockito {

    @Mock
    Random random;

    @Test
    public void usingMockito() {
        // Create stub
        when(random.nextInt(10))
                .thenReturn(5);
    }

}
```

Break

10:35

Test suite

Test suite is used to bundle a few unit test cases and run them together. In JUnit, both **@RunWith** and **@Suite** annotations are used to run the suite tests. This chapter takes an example having two test classes, **TestJunit1 & TestJunit2**, that run together using Test Suite.

```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;

@RunWith(Suite.class)

@Suite.SuiteClasses({
   TestJunit1.class,
   TestJunit2.class
})

public class JunitTestSuite {
}
```

JUnit Category (JUnit 4)

We can create as many categories by implementing marker interfaces where the name of the marker interface represents the name of the category.

```
public interface UnitTest {
}
```

```
public interface IntegrationTest {
}
```

```
@Test
@Category(IntegrationTest.class)
public void testAddEmployeeUsingSimpelJdbcInsert() {
}

@Test
@Category(UnitTest.class)
public void givenNumberOfEmployeeWhenCountEmployeeThenCountMatch() {
}
```

JUnit 5 Tags

JUnit 5 we can filter tests by tagging a subset of them under a unique **tag** name. For example, suppose we have both unit tests and integration tests implemented using JUnit 5. We can add tags on both sets of test cases:

```
@Test
@Tag("IntegrationTest")
public void testAddEmployeeUsingSimpelJdbcInsert() {
}

@Test
@Tag("UnitTest")
public void givenNumberOfEmployeeWhenCountEmployeeThenCountMatch() {
}
```