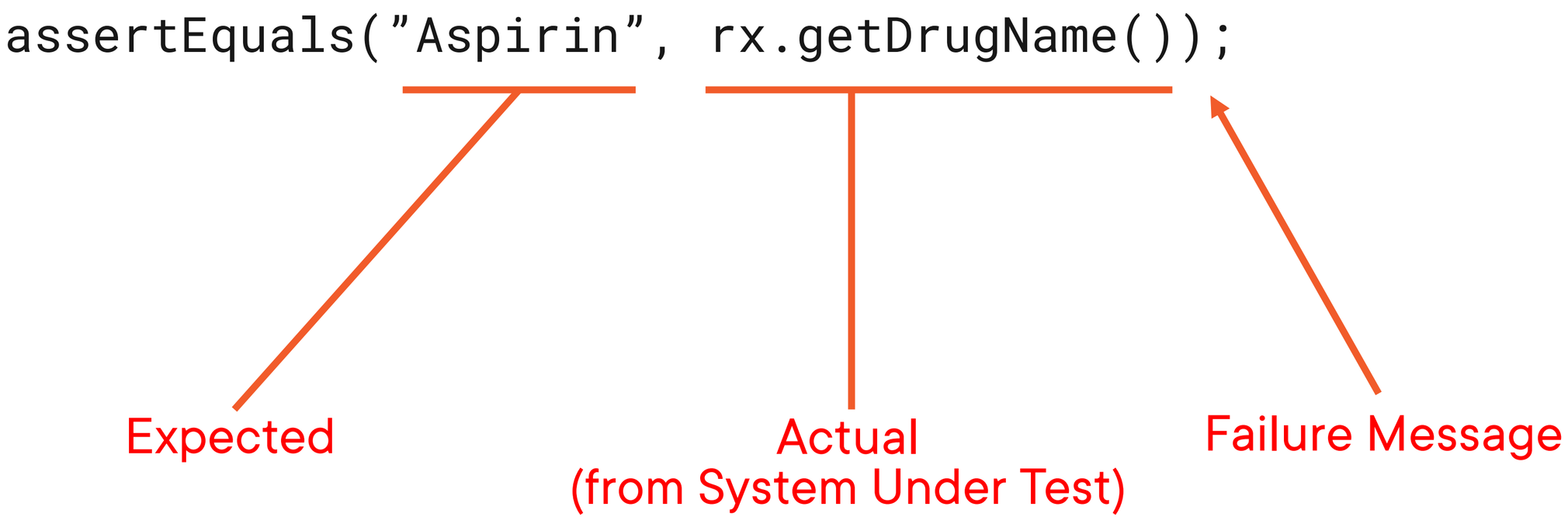
Writing Tests and Using Assertions

Using Junit Assertions

Jim Weaver
Developer, Trainer and Author

www.codeweaver.org

A JUnit Assertion



An Assertion Failure Short-Circuits a Test

```
assertEquals(”Aspirin”, rx.getDrugName());
assertEquals(”Tablet”, rx.getDoseForm());
```

This is one reason why it’s preferred to just have a single assertion per test method

Asserting Equality and Identity

Asserting Boolean Values

Asserting Collections and Streams

Understanding Common Test Method Structure

```
@Test
void drugInConceptIfOneClassMatches() {
    DrugConcept testConcept = new DrugConcept(
            new DrugClassification[] {DrugClassification.ANTIANXIETY,
            DrugClassification.ANALGESICS_NARCOTIC});
    DispensableDrug drug = new DispensableDrug("adrug",
            new DrugClassification[] {DrugClassification.ANALGESIC,
                    DrugClassification.ANTIANXIETY}, false);
    boolean drugInConcept = testConcept.isDrugInConcept(drug);
    assertTrue(drugInConcept);
}
```

Common Test Steps

- Setup: Prepare to call production code – often involves setting up expected results, initializing production code to be called, or setting up arguments to pass.
- Kick: Call the production code.
- Verify: Verify code returns expected results (assertions).
- Teardown: Any between test cleanup needed – rarely needed for typical unit tests.

State vs. Behavioral Verification

State-based Testing

Verify returned values from the code under test

Value returned from a method call

State of some object returned from a method call

Behavioral / Interaction-based Testing

Verify what the production code does after it's called – what its interactions are

Other methods called, especially on other objects

Often used to verify that third-party libraries or remote systems are called properly by the code under test

Mock frameworks help with this kind of testing

Asserting Expected Exceptions

Grouping Assertions with assertAll

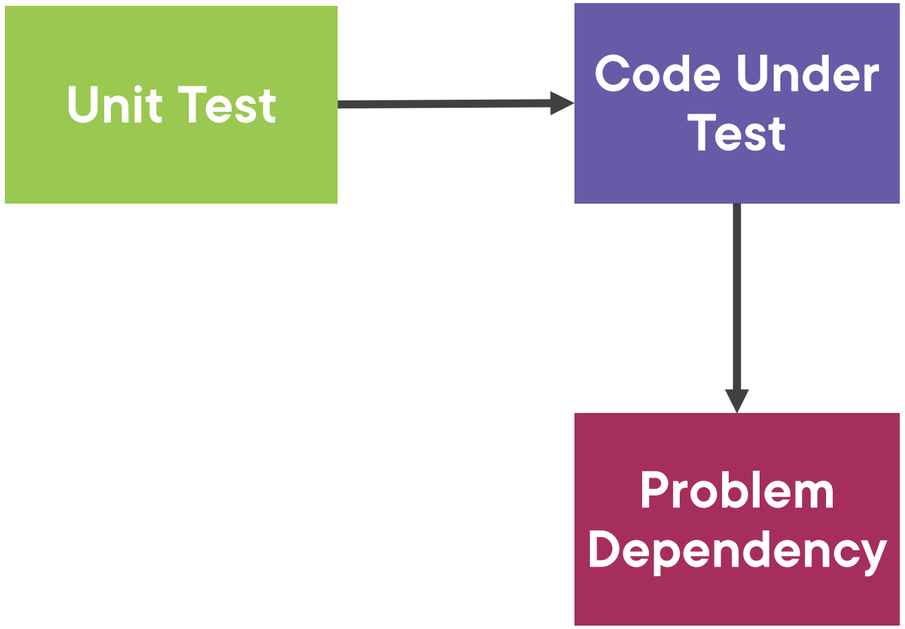
Understanding Test Doubles

Dependencies of the code under test can be problematic

Access an external system or database

Data returned is unreliable

Allowing the test to flow through the dependency makes the test unreliable

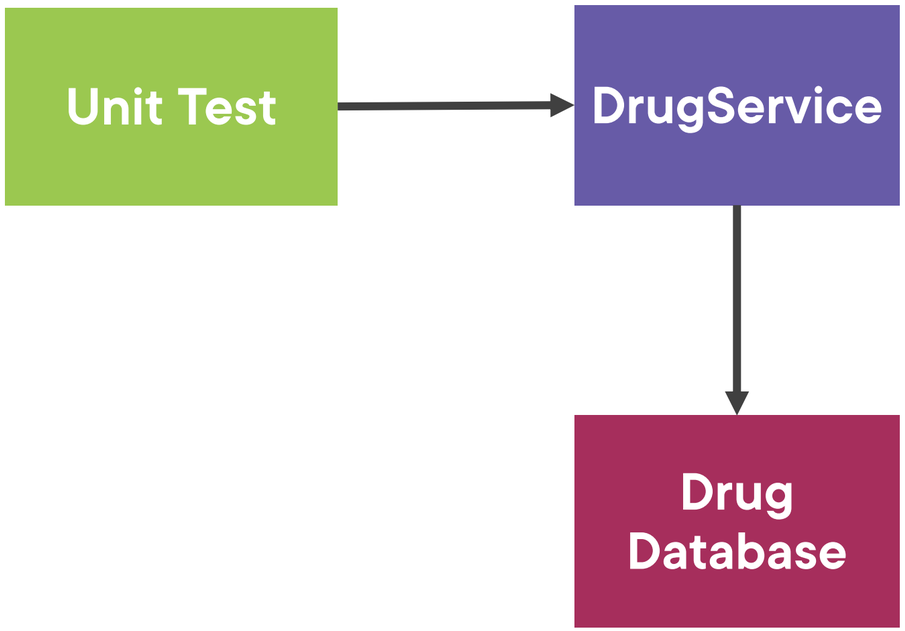


Dependencies of the code under test can be problematic

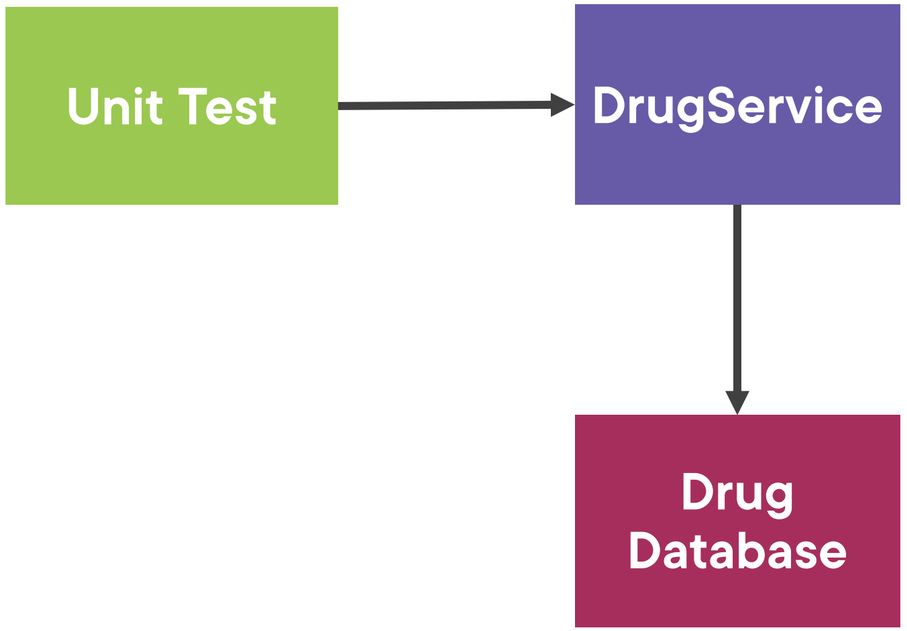
Access an external system or database

Data returned is unreliable

Allowing the test to flow through the dependency makes the test unreliable

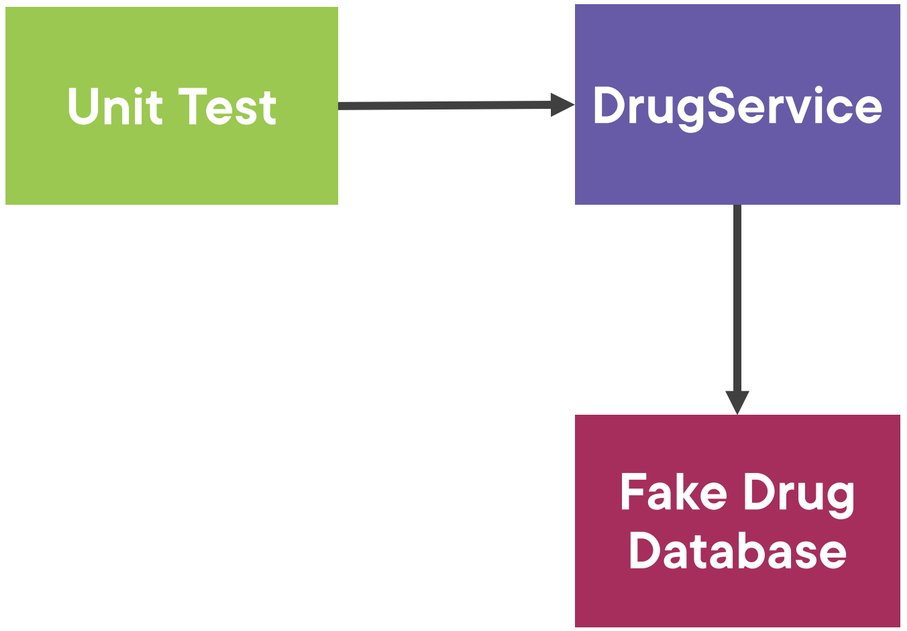


Test Double Solution



The unit tests swaps out the real dependency for a fake substitute, allowing the test to control what the substitute does.

Test Double Solution



This technique relies on a Java interface and dependency injection.

Using Test Doubles

Up Next:
Leveraging Test Lifecycle