Understanding the Different Types of Automated Tests

Overview

Types of automated tests

Unit, integration, subcutaneous, and functional user interface tests

Test breadth versus depth

The logical phases of an automated test

Isolating code with mock objects

Data-driven tests

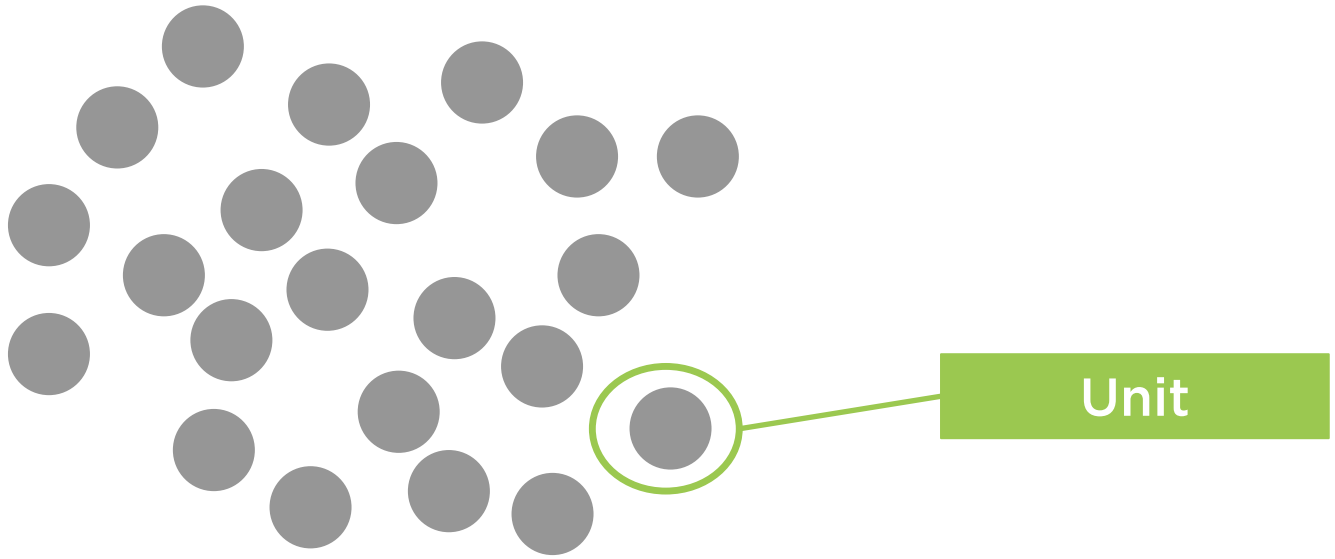
Business-readable tests

How many tests of each type?

- Testing Pyramid model
- Beyond the Test Pyramid

Characteristics of good automated tests

Unit Tests



Unit Tests

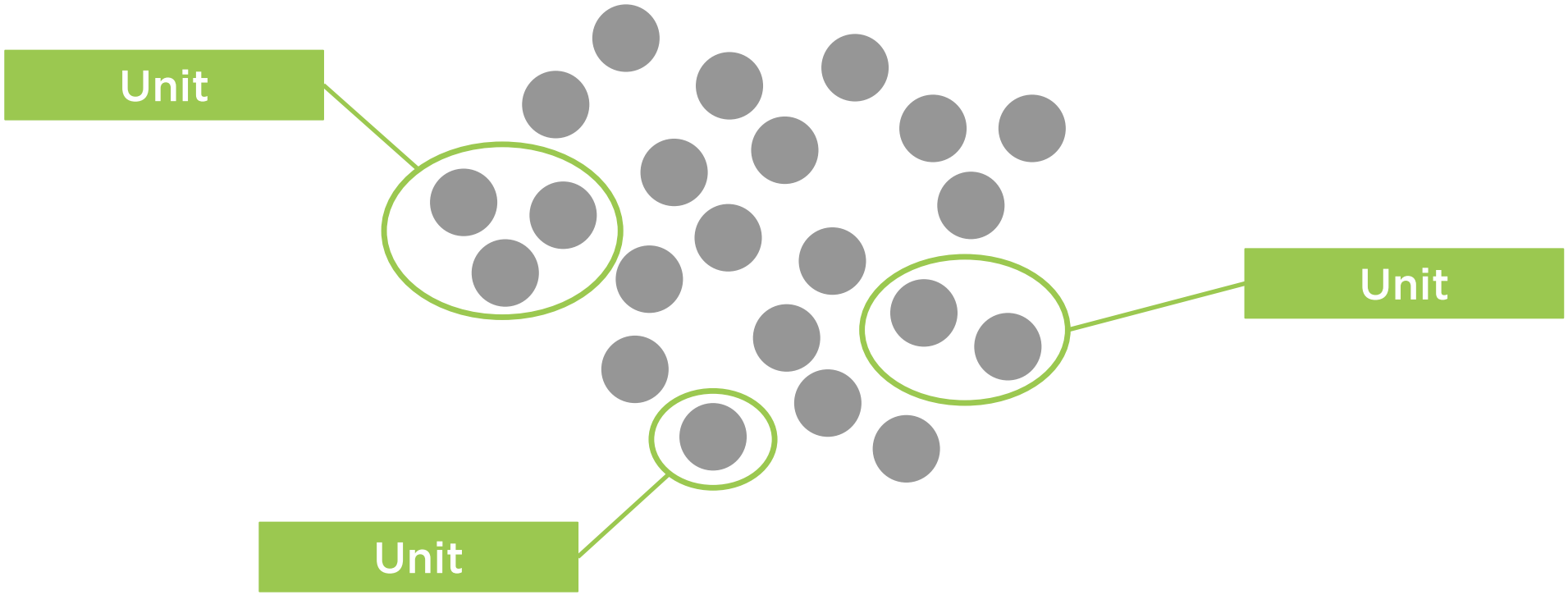
Low level

Highly focused

Quick to execute

Easy to test all parts / paths of code

Unit Tests



“...it's a situational thing - the team decides what makes sense to be a unit for the purposes of their understanding of the system and its testing”

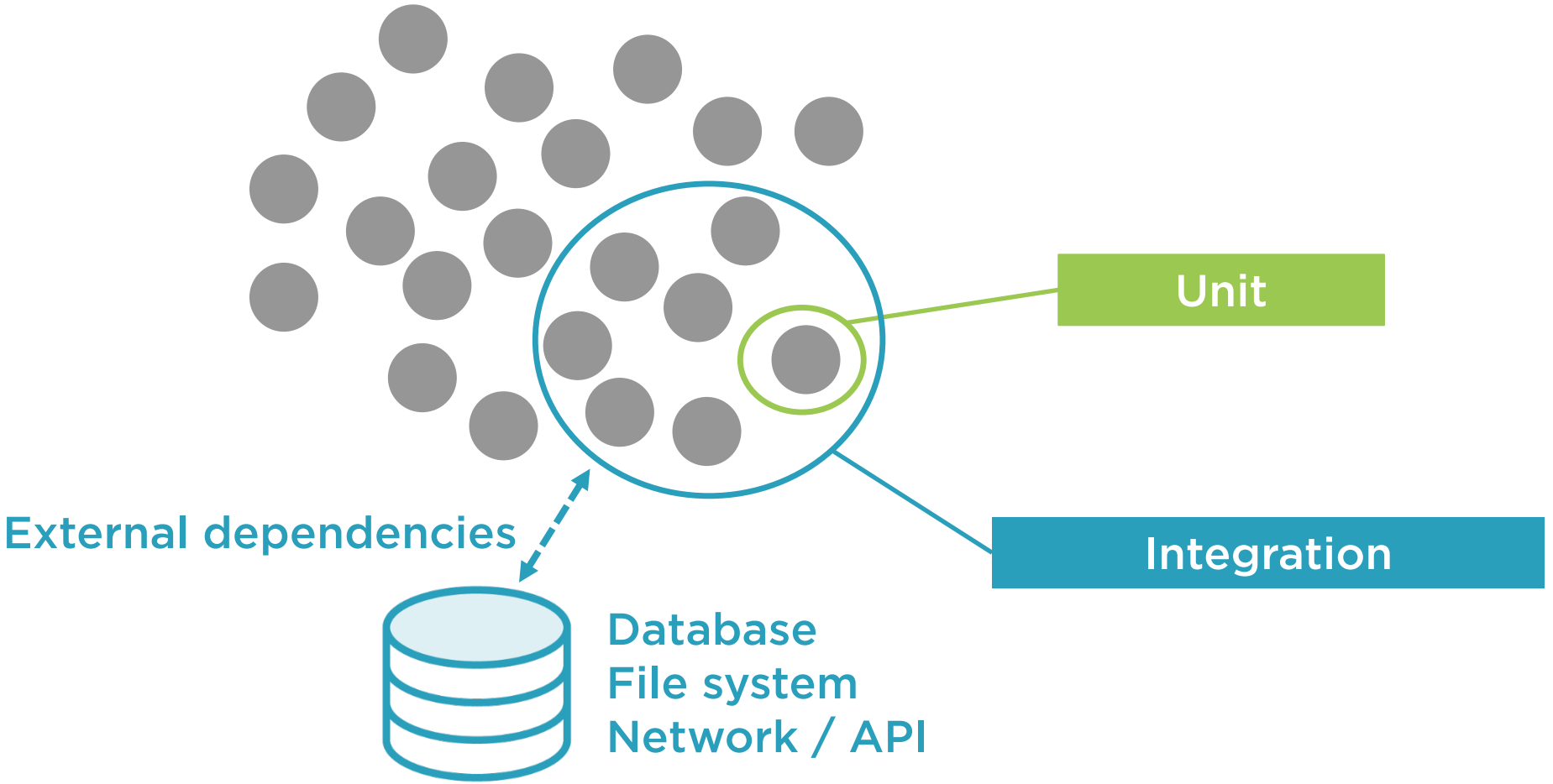
Martin Fowler

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

Units of behavior

```
BEGIN TEST Addition

    Create new Calculator instance


    Call Add functionality, passing values 5 and 2



    IF result is equal to 7
        Report test as passed
    ELSE
        Report test as failed
    END IF

END TEST
```

Integration Tests



Integration Tests

Higher level than unit

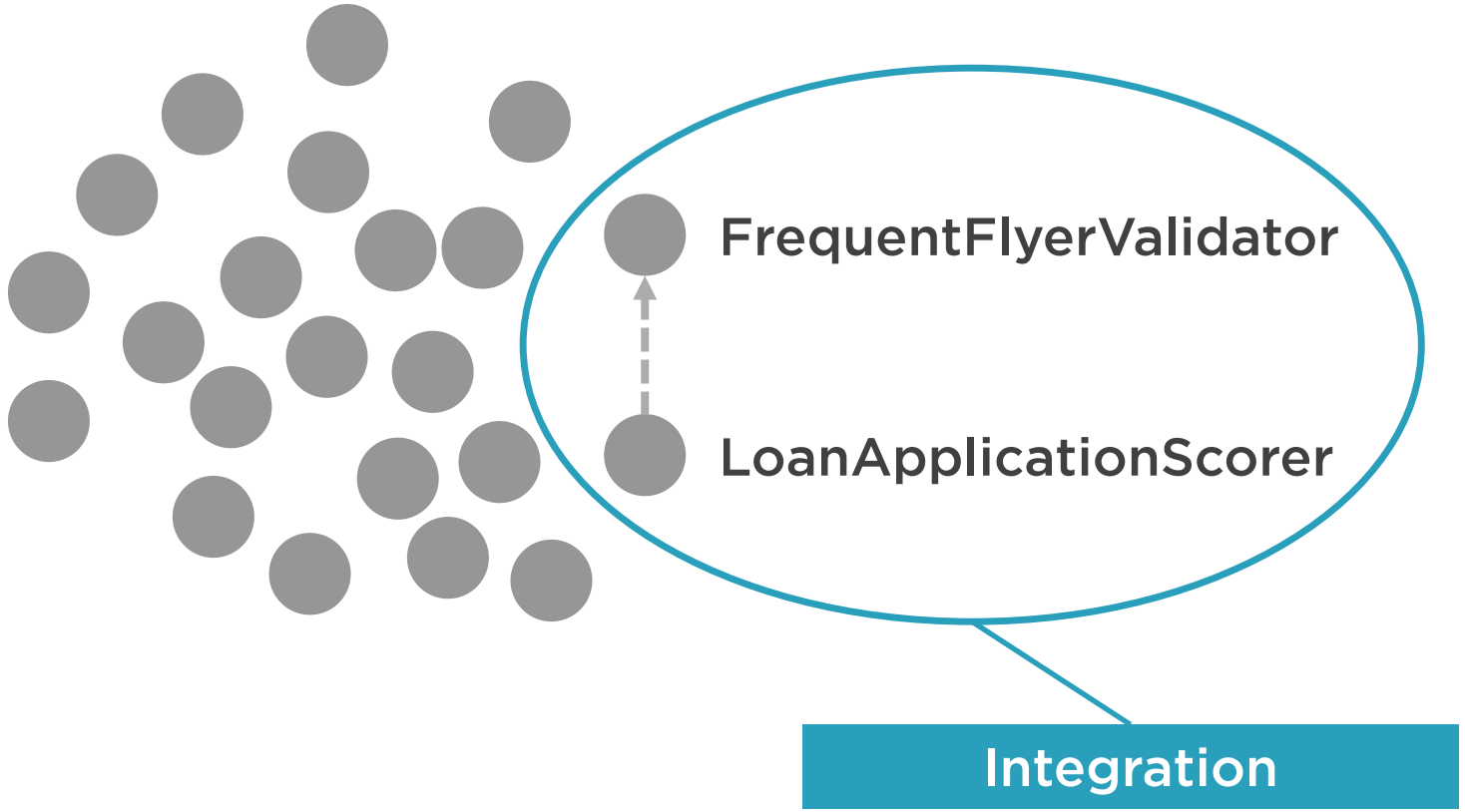
Things working together

May be slow if external dependencies involved (e.g. database)

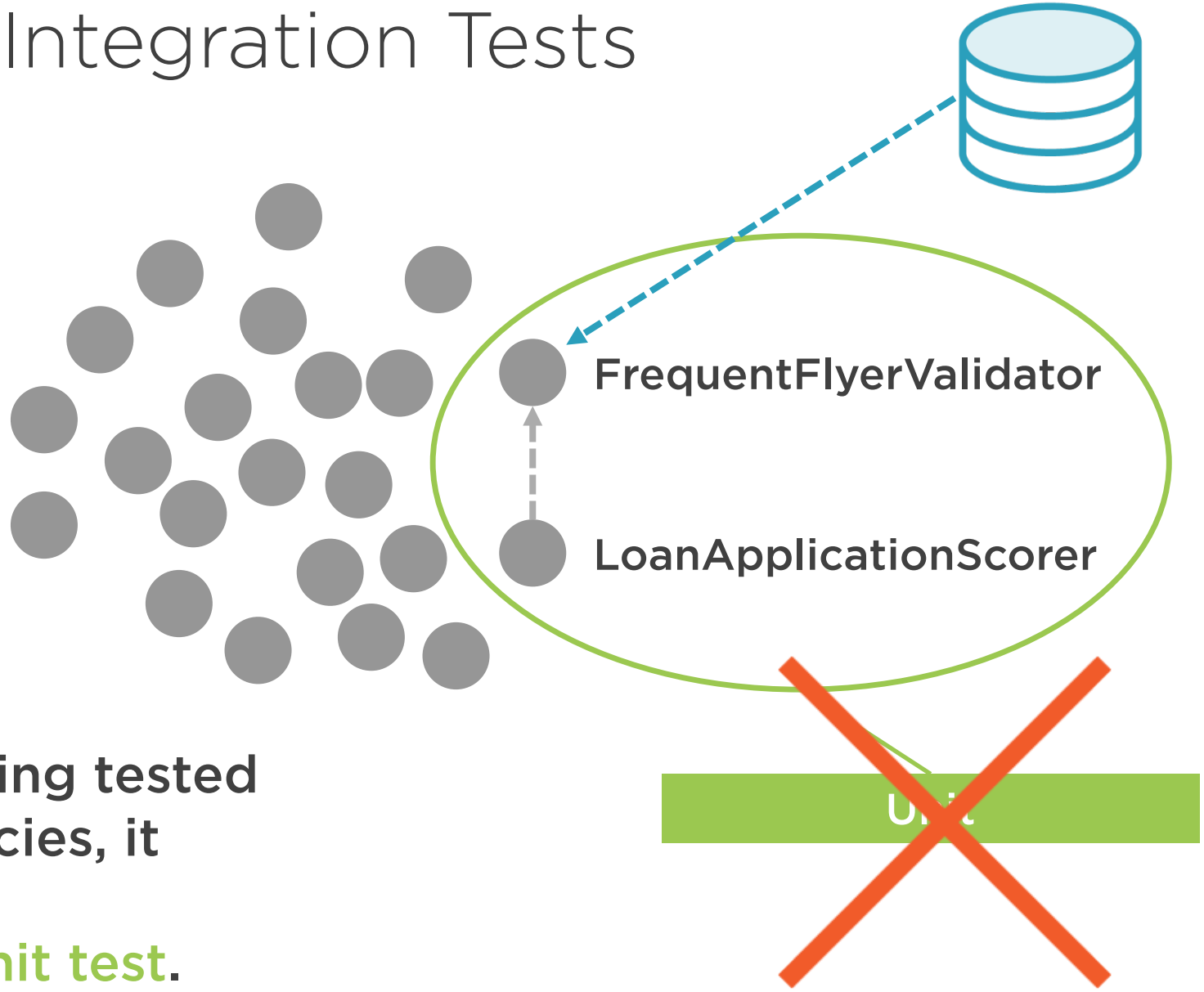
Harder to test all parts / paths of code

```
BEGIN TEST Loan Application Successful

    Create new FrequentFlyerValidator instance v

    Create new LoanApplicationScorer instance s, passing v as
    the dependency


    Call s score loan functionality, passing “good” applicant



    IF result is equal to LOANGRANTED
        Report test as passed
    ELSE
        Report test as failed
    END IF

END TEST
```

Integration Tests

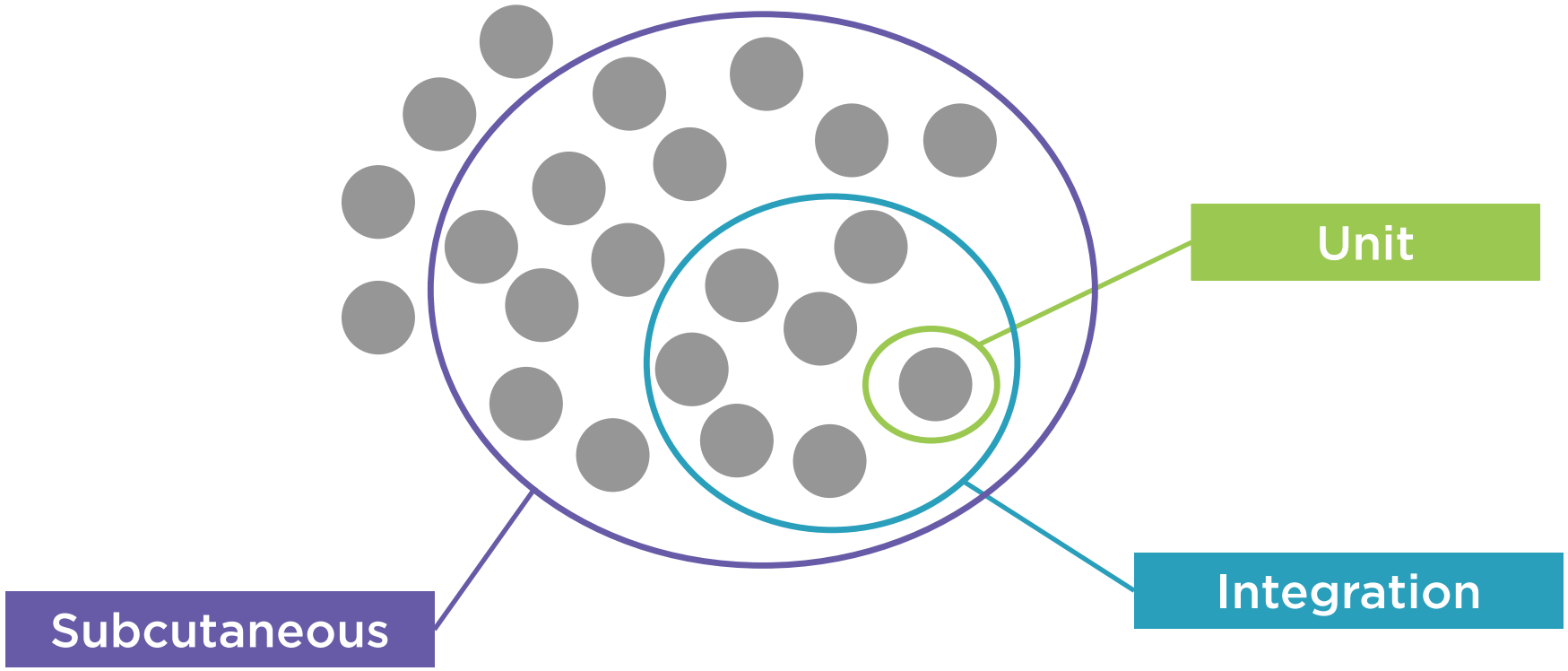


Integration Tests

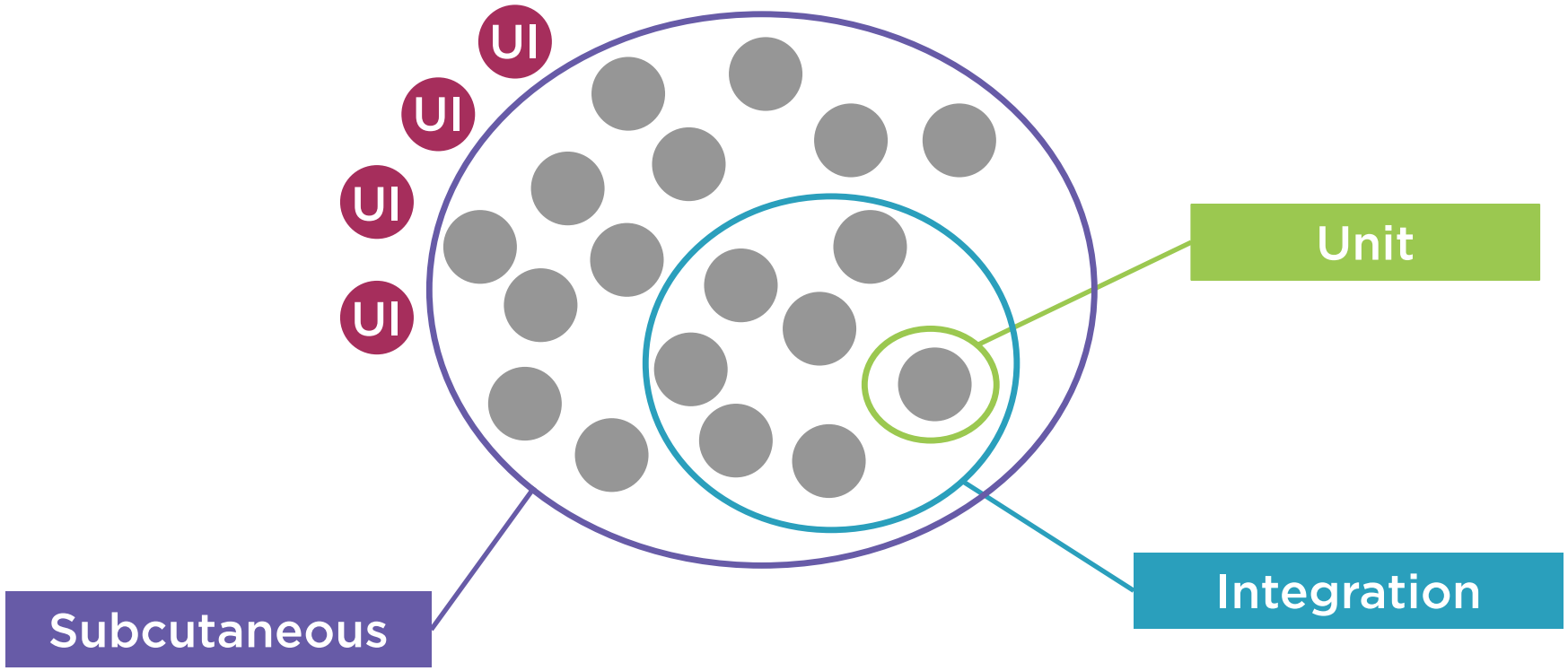


Generally if the code being tested uses external dependencies, it would be considered an integration test, not a unit test.

Subcutaneous Tests



Subcutaneous Tests



Subcutaneous Tests

Just below the surface of the UI

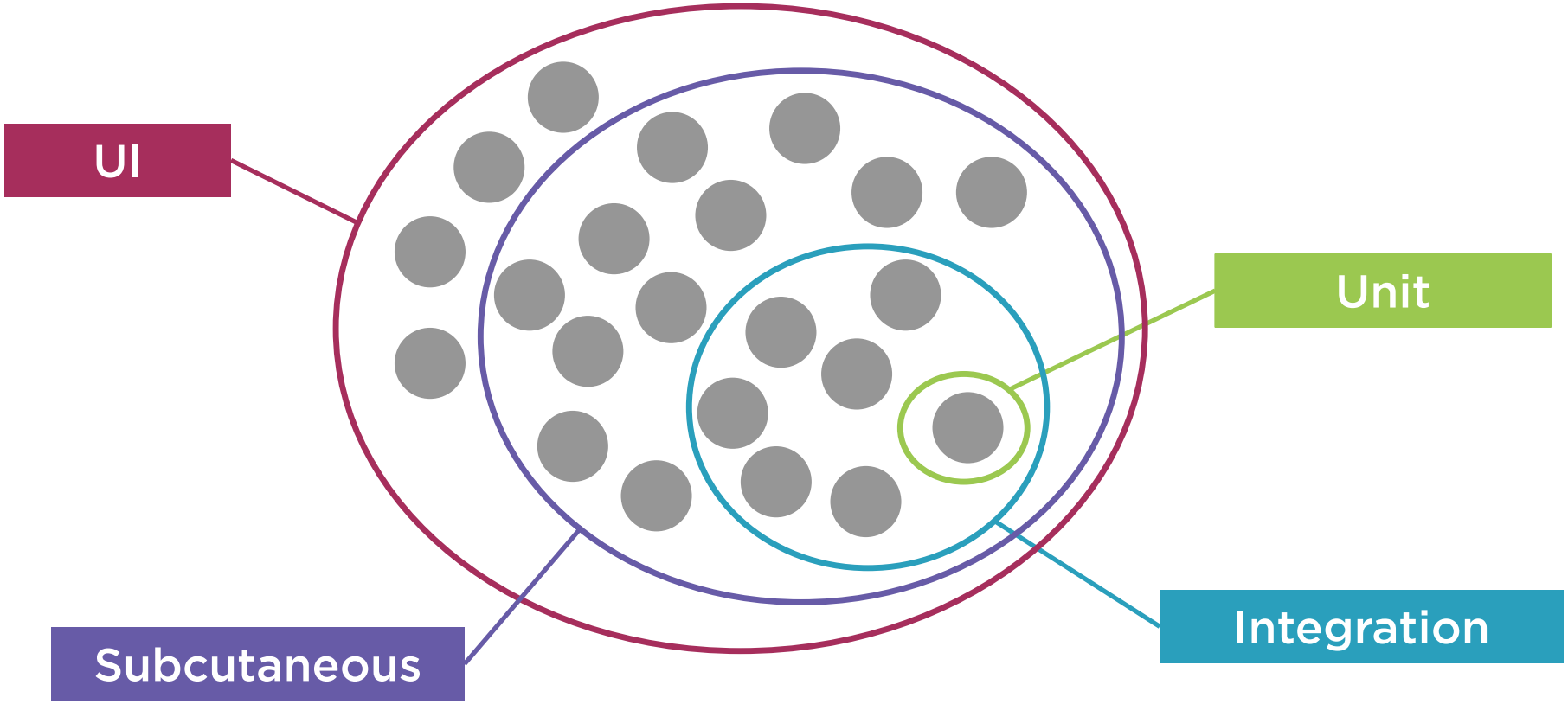
Can test all the non-UI components working together

Potentially quicker than automated UI tests

Difficult to test UI

Doesn't test logic coded into the UI

Functional UI Tests



Functional User Interface Tests

Testing as if we are end-user

Manipulating UI elements (e.g. clicking buttons)

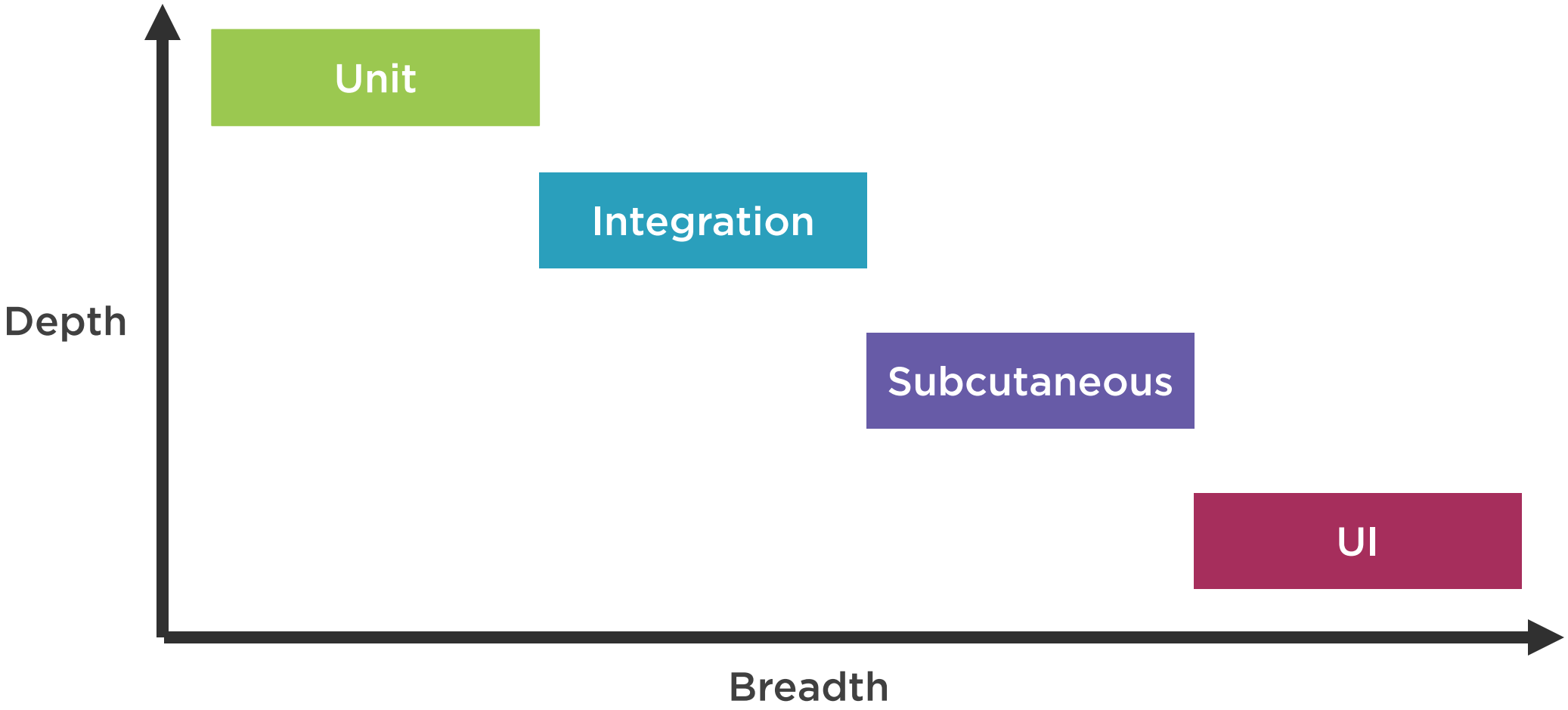
"Full stack" testing

Not testing look and feel

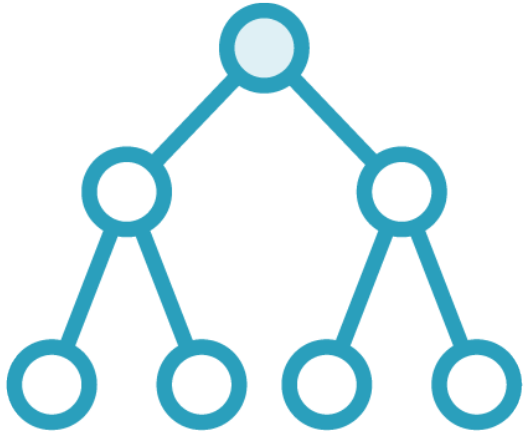
Slow to execute

Potentially brittle

Test Breadth Versus Depth



The Logical Phases of an Automated Test



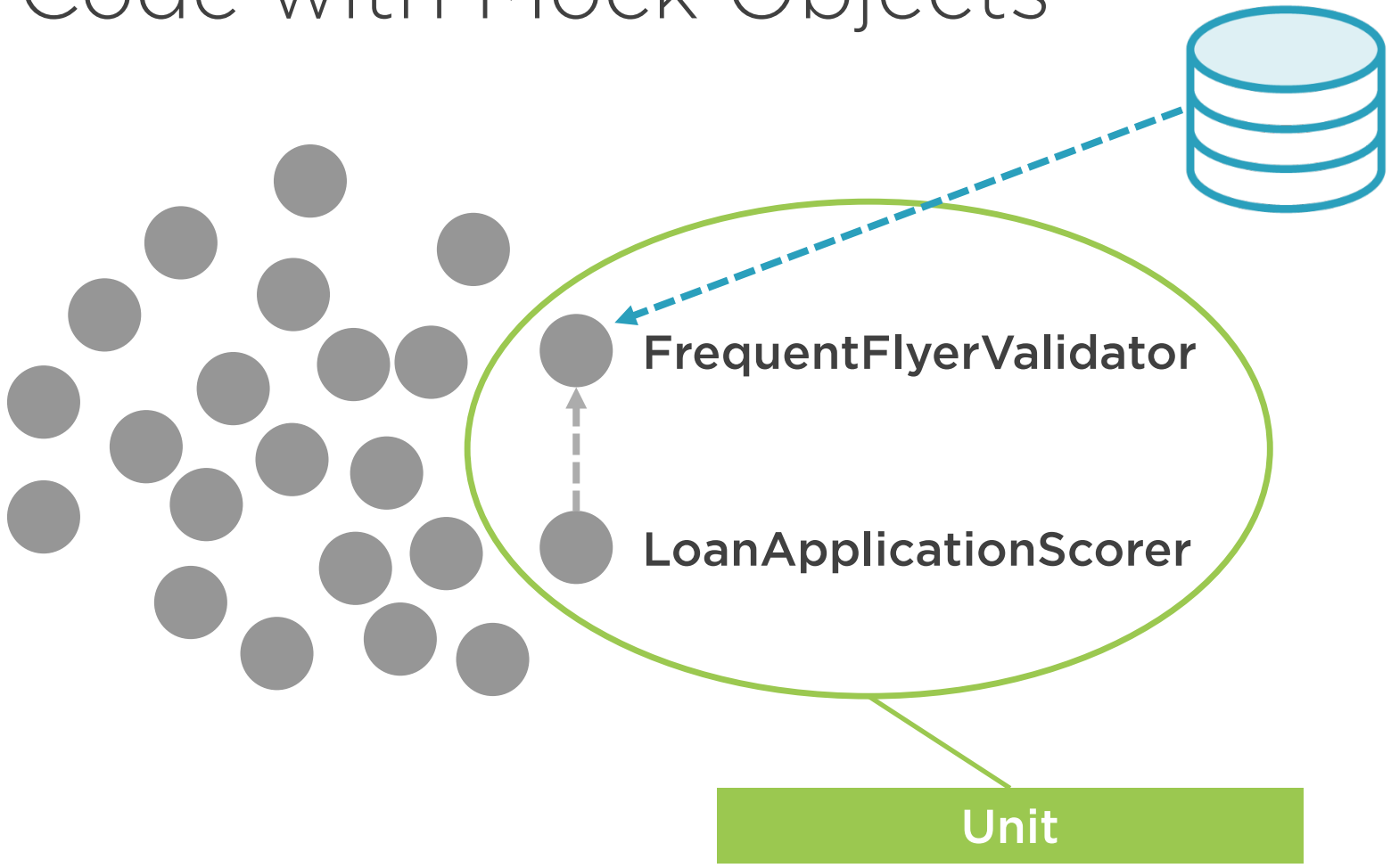
Arrange

Act

Assert

```
BEGIN TEST Loan Application Successful

    Create new FrequentFlyerValidator instance v

    Create new LoanApplicationScorer instance s, passing v as
    the dependency

    Call s score loan functionality, passing “good” applicant

    IF result is equal to LOANGRANTED
        Report test as passed
    ELSE
        Report test as failed
    END IF

END TEST
```

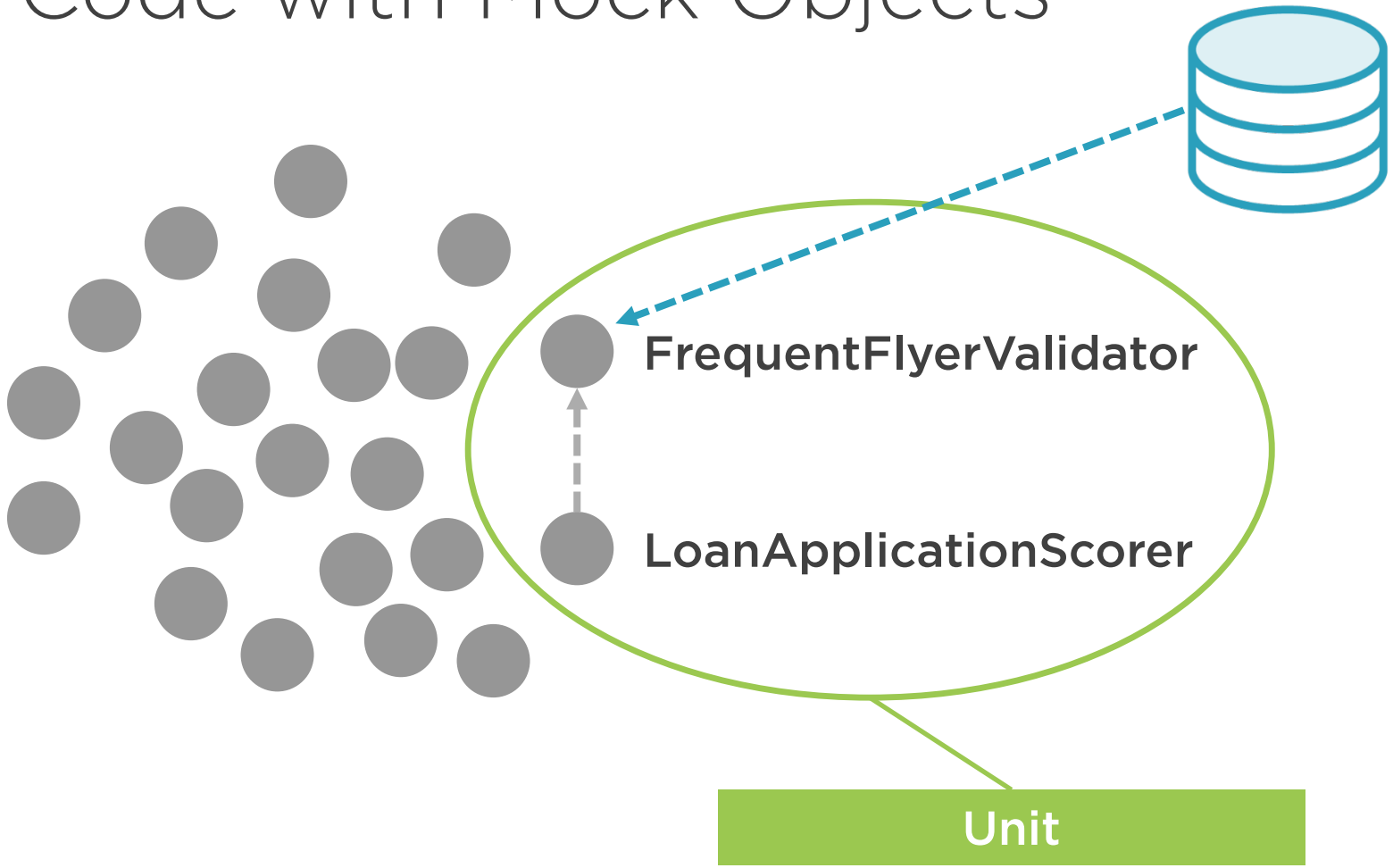
Isolating Code with Mock Objects



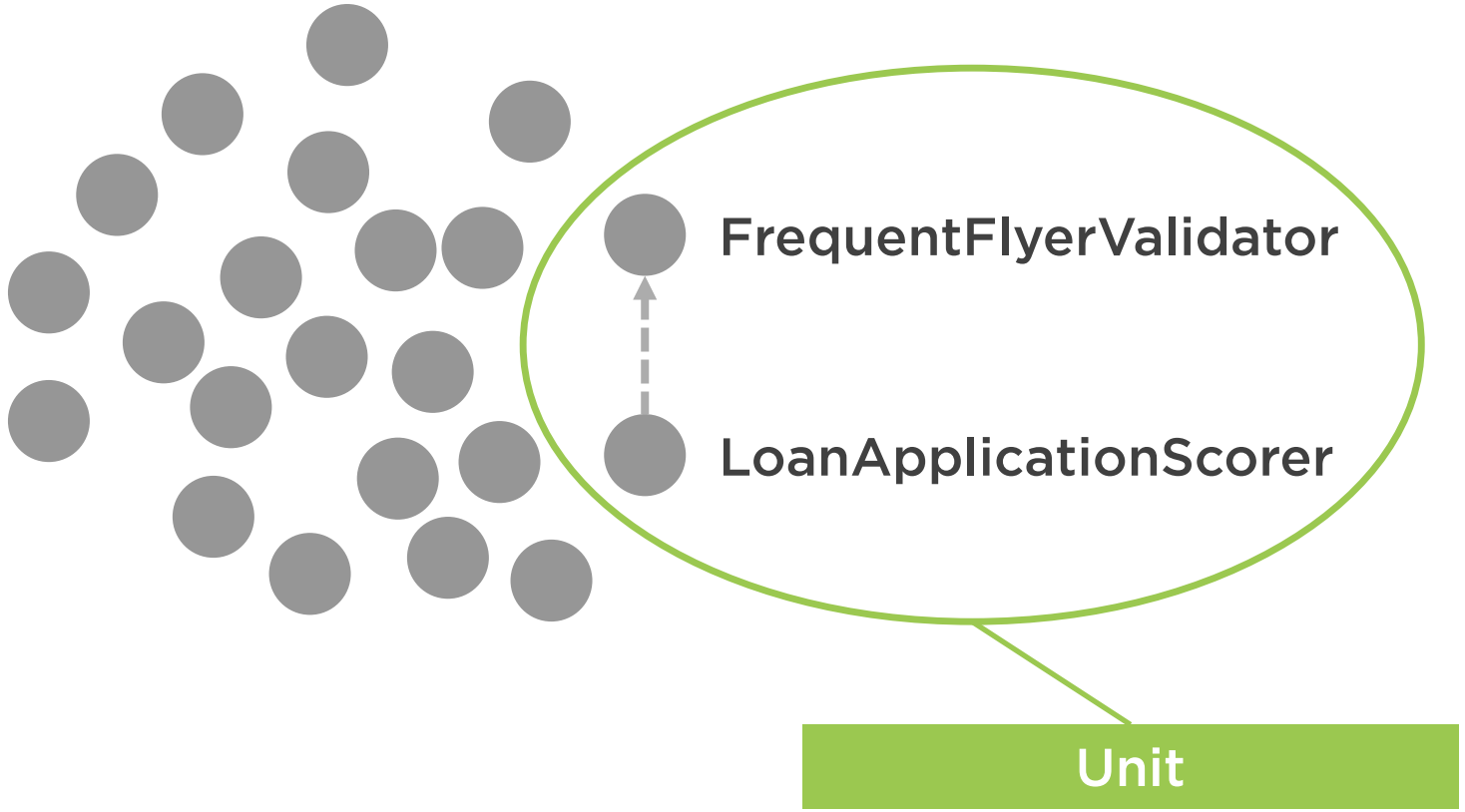
Mocks
Stubs
Test Doubles
Fakes

Replacing the actual dependency that would be used at production time, with a test-time-only version that enables easier isolation of dependencies.

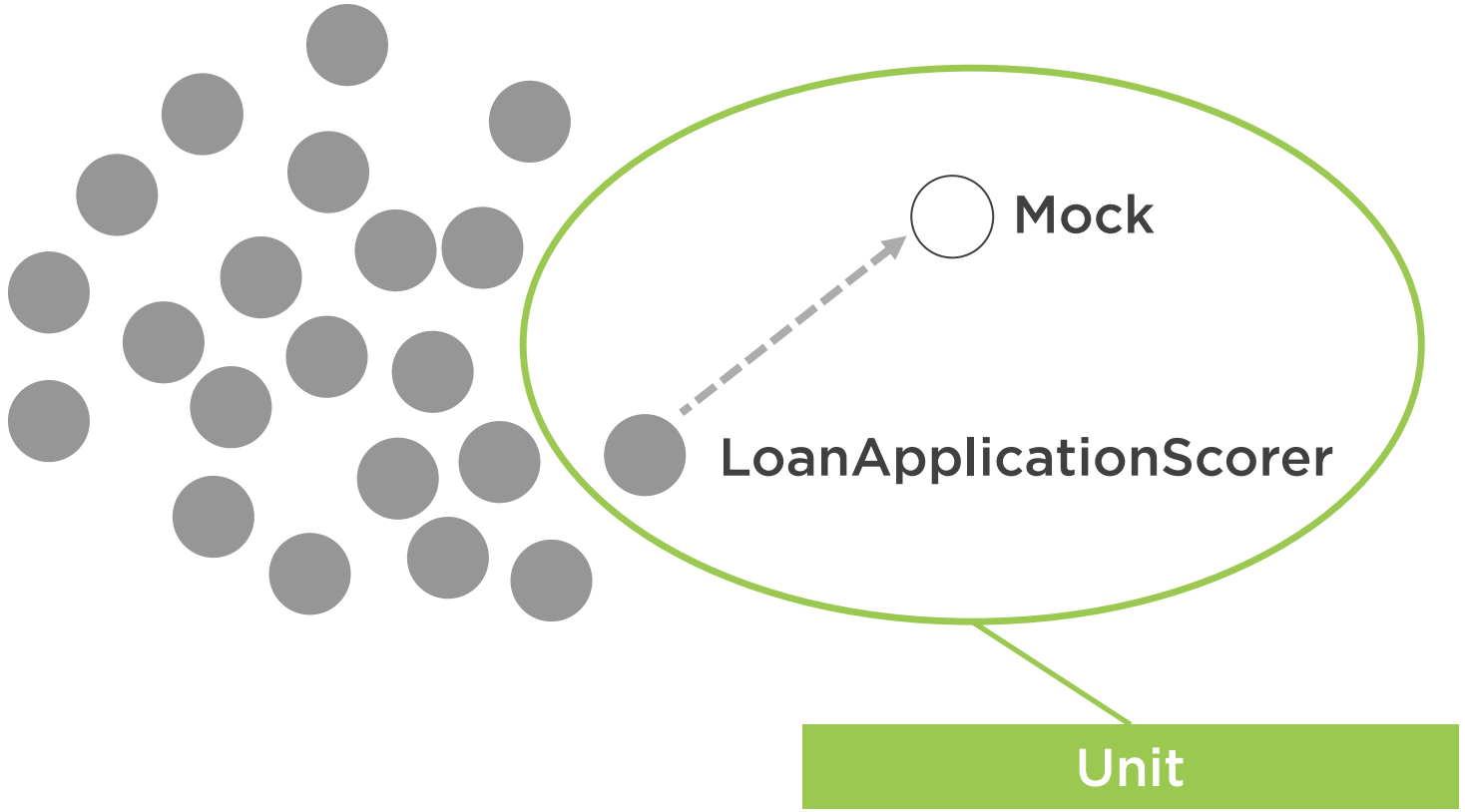
Isolating Code with Mock Objects



Isolating Code with Mock Objects



Isolating Code with Mock Objects



```
BEGIN TEST Loan Application Successful

    Create new mock FrequentFlyerValidator instance mv

    Configure mv to simulate valid frequent flyer number

    Create new LoanApplicationScorer instance s, passing mv
    as the dependency


    Call s score loan functionality, passing “good” applicant


    IF result is equal to LOANGRANTED
        Report test as passed
    ELSE
        Report test as failed
    END IF

END TEST
```

Data-driven testing can be combined with any type of automated test.

```
BEGIN TEST Add Positive Numbers

    Create new Calculator instance


    Call Add functionality, passing values 5 and 2



    IF result is equal to 7
        Report test as passed
    ELSE
        Report test as failed
    END IF

END TEST
```

```
BEGIN TEST Add Negative Numbers

    Create new Calculator instance


    Call Add functionality, passing values -5 and -2



    IF result is equal to -7
        Report test as passed
    ELSE
        Report test as failed
    END IF

END TEST
```

```
BEGIN TEST Add Mixed Numbers

    Create new Calculator instance


    Call Add functionality, passing values -5 and 2



    IF result is equal to -3
        Report test as passed
    ELSE
        Report test as failed
    END IF

END TEST
```

```
BEGIN TEST Add Numbers

    Create new Calculator instance

    Call Add functionality, passing values @a and @b


    IF result is equal to @c
        Report test as passed
    ELSE
        Report test as failed
    END IF

END TEST
```

@a	@b	@c
5	2	7
-5	-2	-7
-5	2	-3

Business Readable Tests

Business,
client,
stakeholders

Developers /
development team

Business Readable Tests

Business,
client,
stakeholders

Business
Readable
Tests

Developers /
development team

Why Business Readable Tests?

Document application

Non-technical

Onboarding

Executable specs

Source controlled

Stays accurate

Living documentation

Better communication

Common/high-level language

Correct features

Reduce wasted effort

Business Readable Tests help ensure that the right thing is being built and that it's being built right.

```
BEGIN TEST Add Positive Numbers

    Create new Calculator instance


    Call Add functionality, passing values 5 and 2



    IF result is equal to 7
        Report test as passed
    ELSE
        Report test as failed
    END IF

END TEST
```

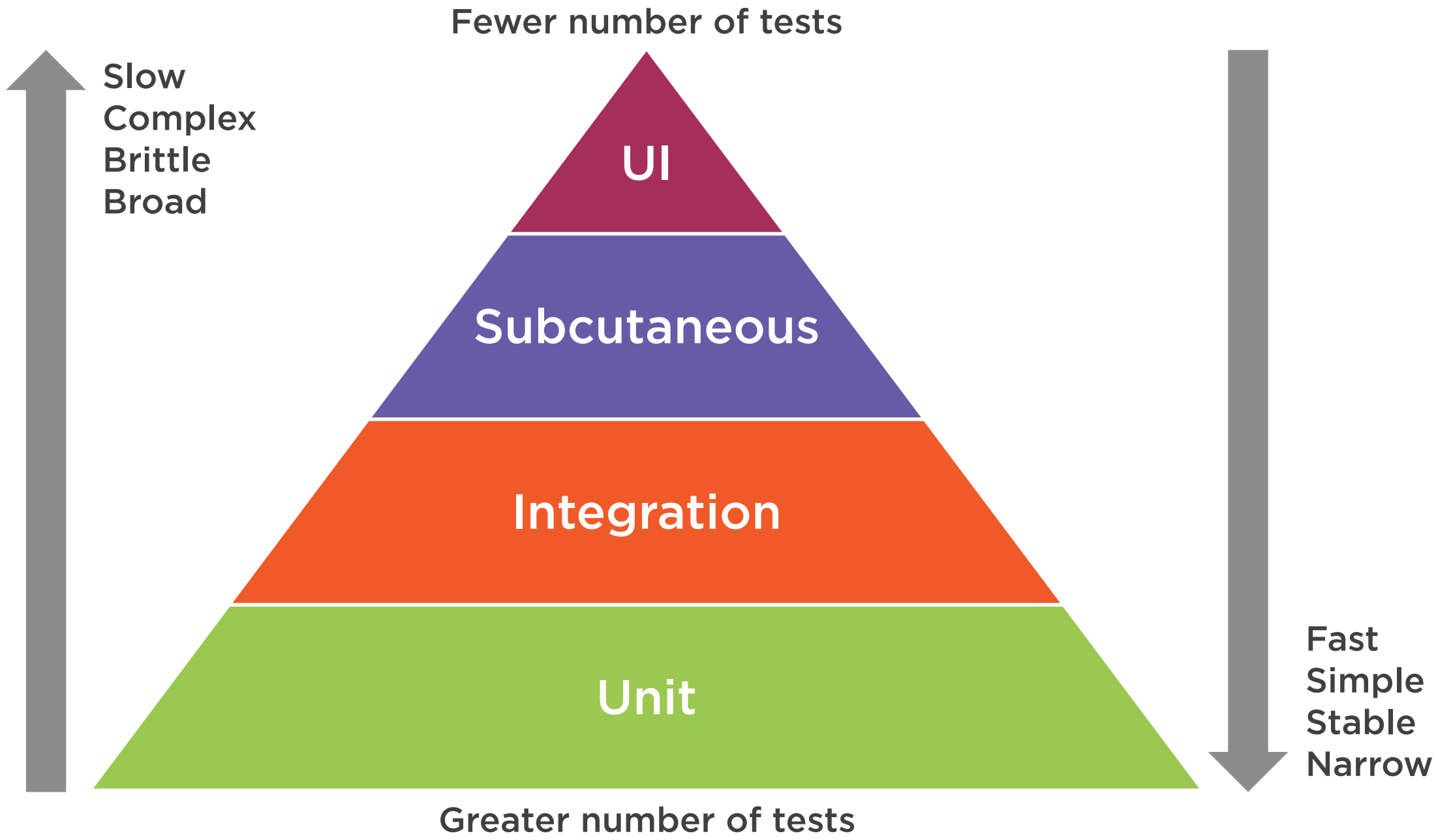
```
Feature: Addition
    Scenario: Positive number addition
        Given I have a new zeroed calculator
        When I add 5 and 2 together
        Then the result should be 7

    Scenario: Negative number addition
    ...
```

Gherkin

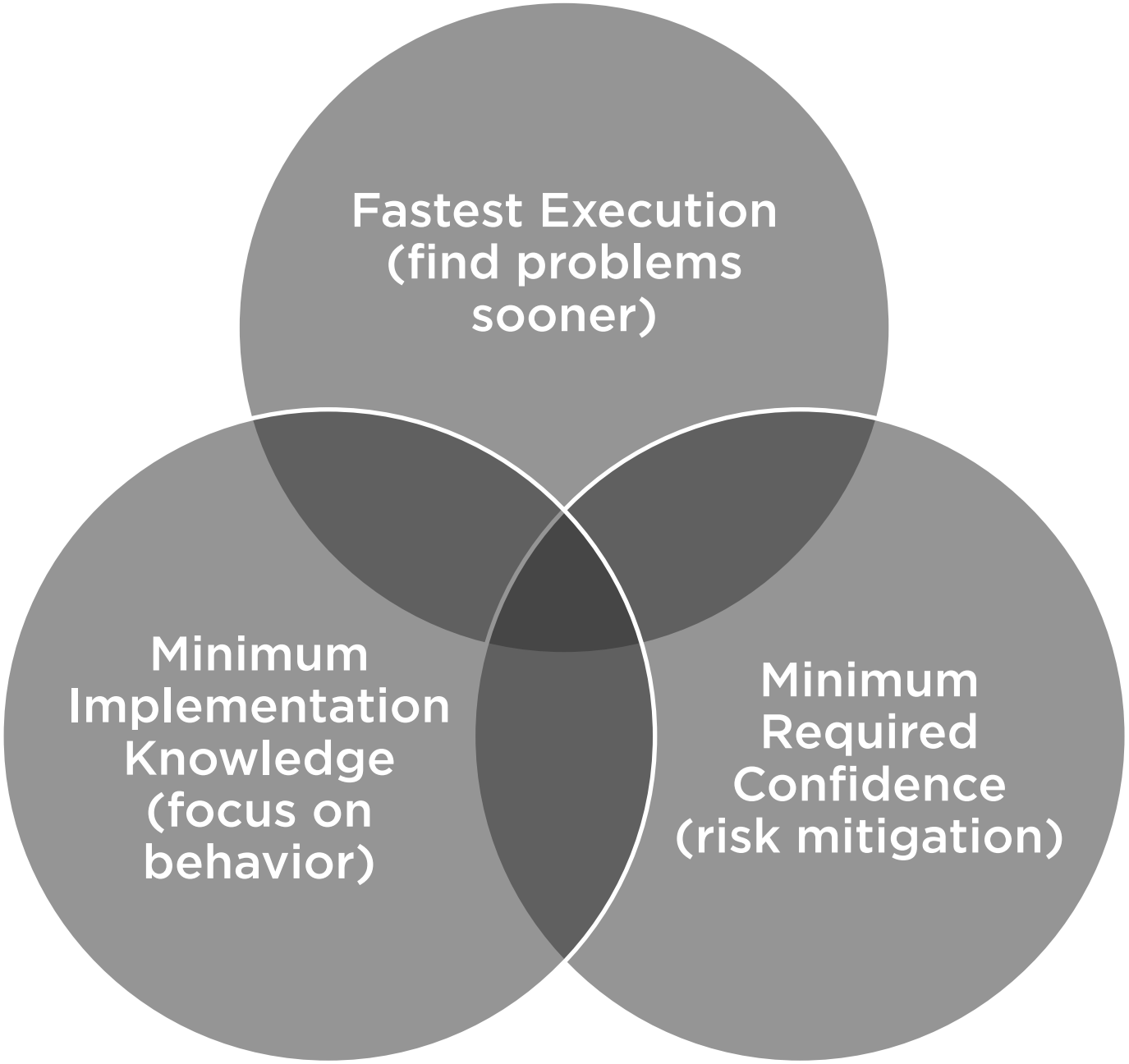
“Gherkin is...a Business Readable, Domain Specific Language that lets you describe software’s behaviour without detailing how that behaviour is implemented.

Gherkin serves two purposes — documentation and automated tests.”

https://github.com/cucumber/cucumber/wiki/Gherkin


Fewer number of tests
Slow
Complex
Brittle
Broad
UI
Subcutaneous
Integration
Unit
Fast
Simple
Stable
Narrow
Greater number of tests

Beyond the Testing Pyramid



Write the **smallest number** of tests possible to reach the **required** level of **quality** or **confidence** in the system being developed.

Characteristics of Good Automated Tests

Isolated
(no side effects on other tests)

Independent
(can be run in any order)

Repeatable
(always pass or fail)

Maintainable
(easy to change)

Valuable

Summary

- Types of automated tests
- Unit, integration, subcutaneous, and functional user interface tests
- Test breadth versus depth
- Arrange, Act, Assert
- Isolating code with mock objects
- Data-driven tests
- Business-readable tests
- Testing Pyramid and beyond
- Characteristics of good automated tests

Next:

Automated Testing Within the Software Development Process