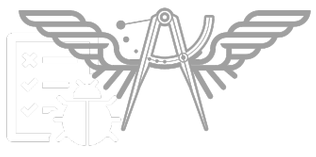
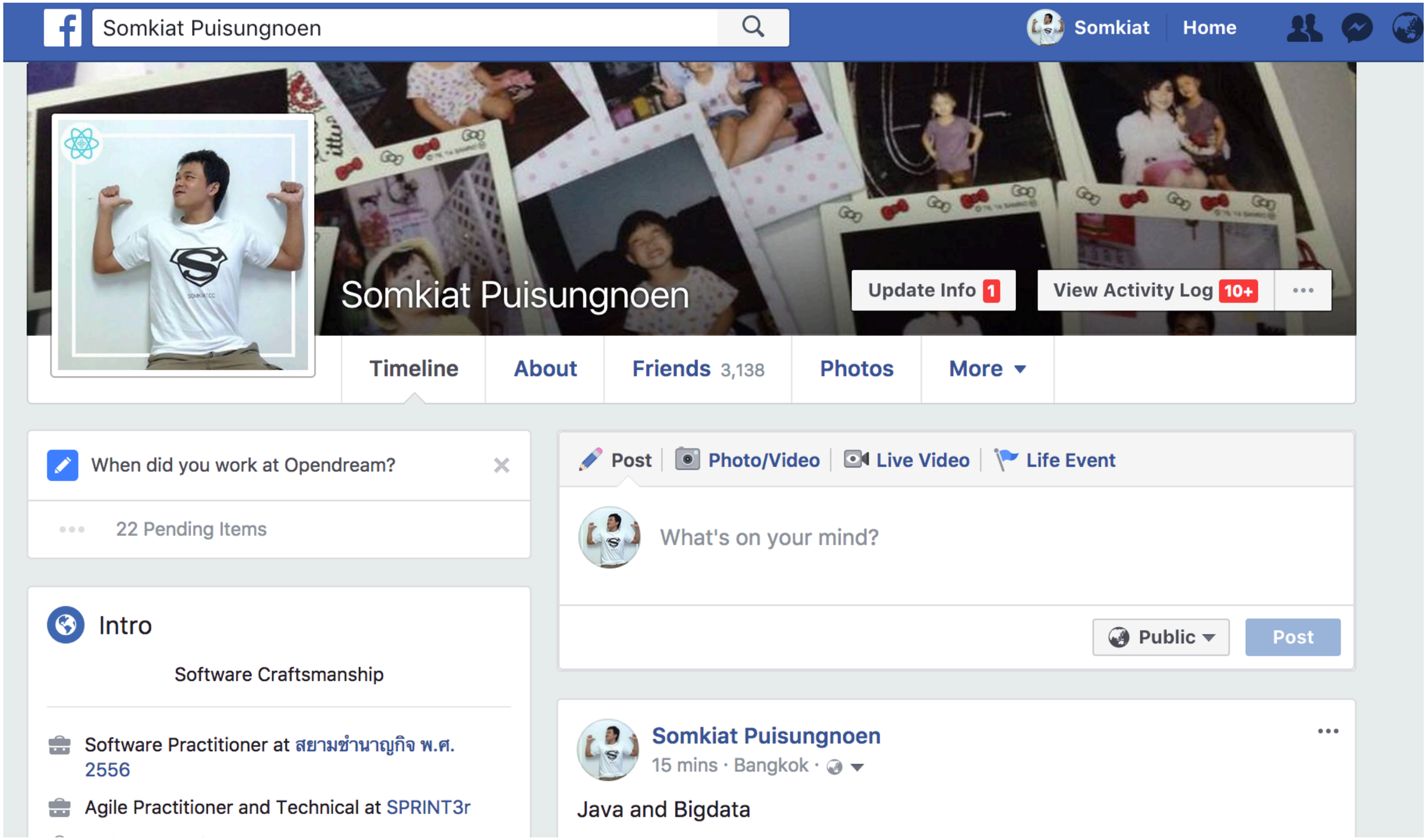
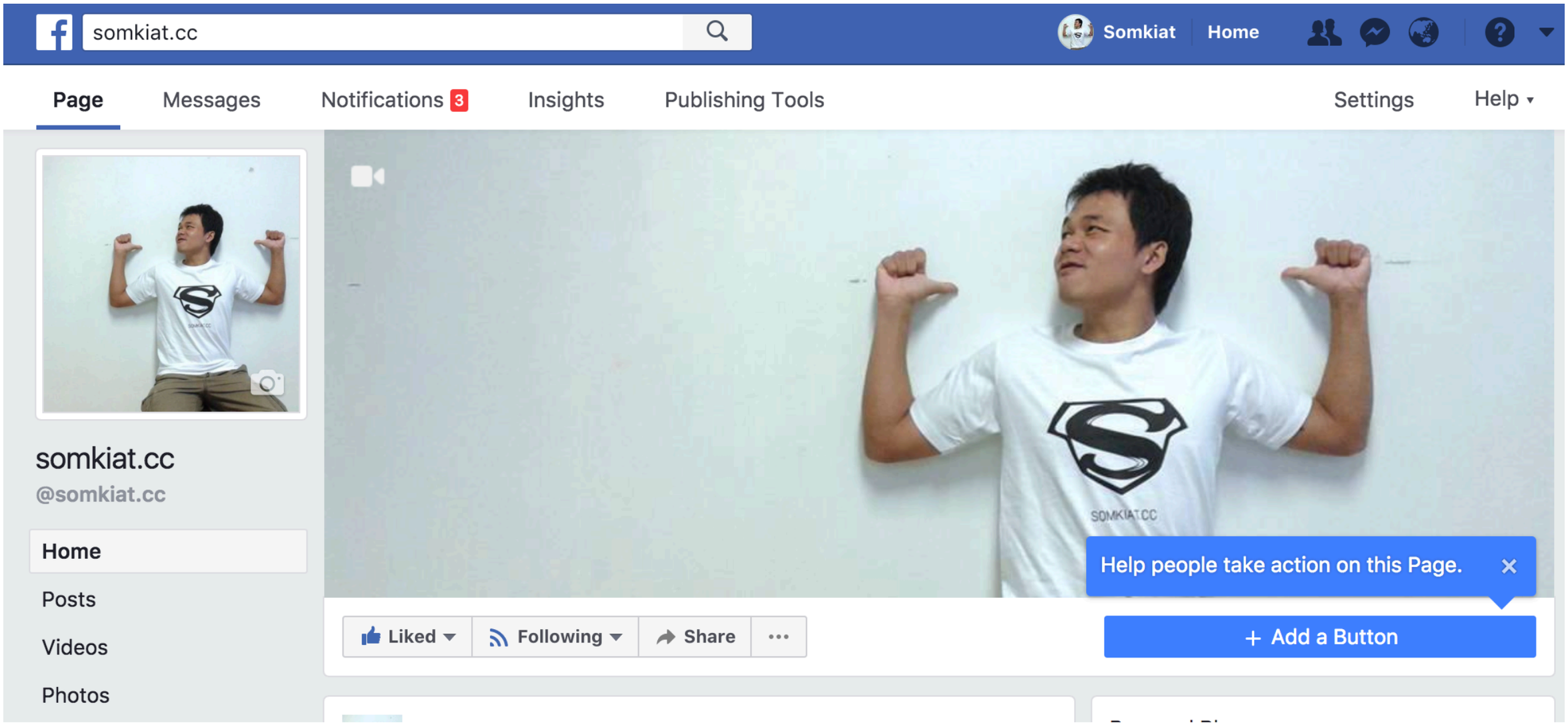
สยามชำนาญกิจ
SIAM CHAMNAN KIT

TDD with Java







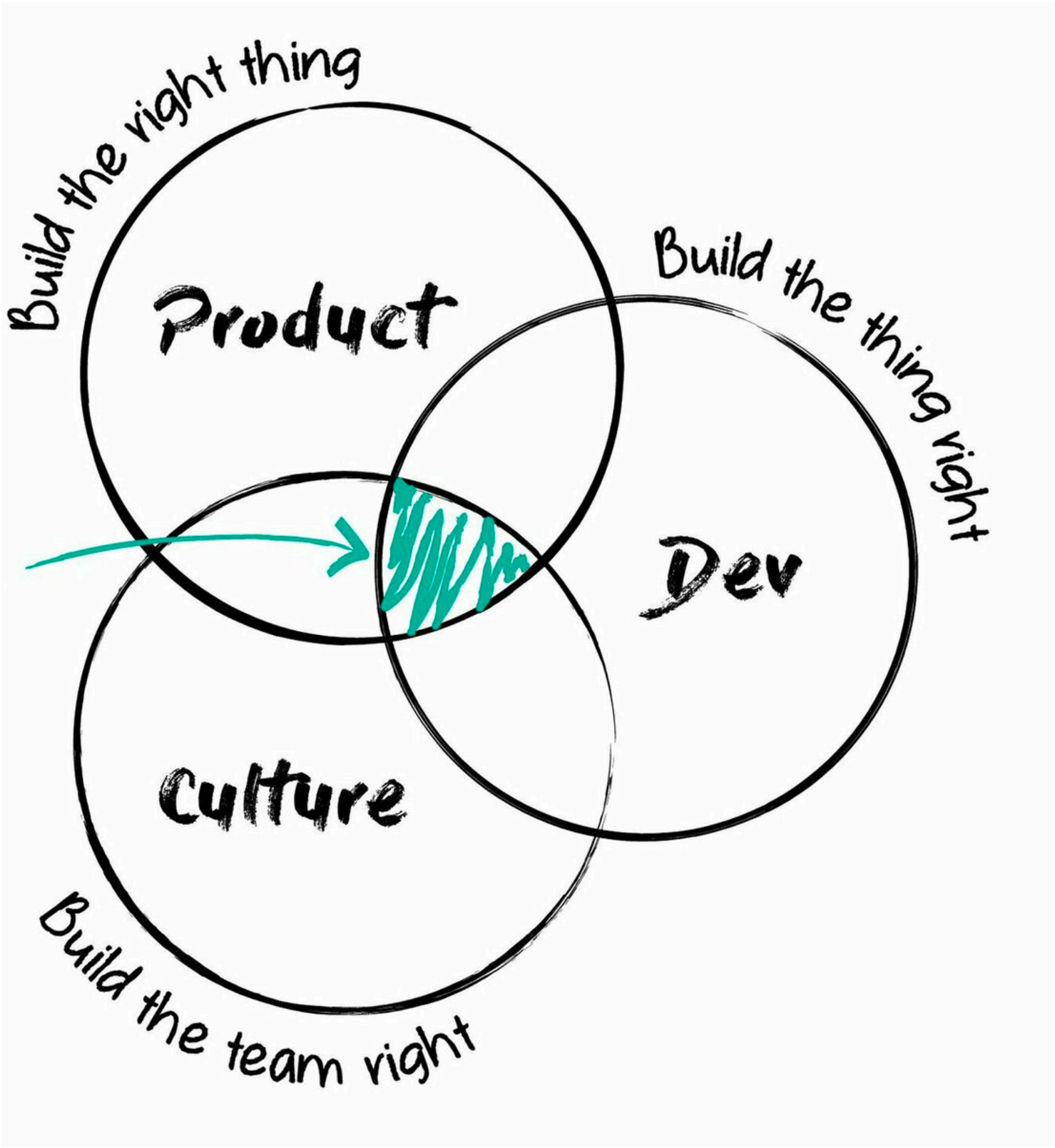
Agenda

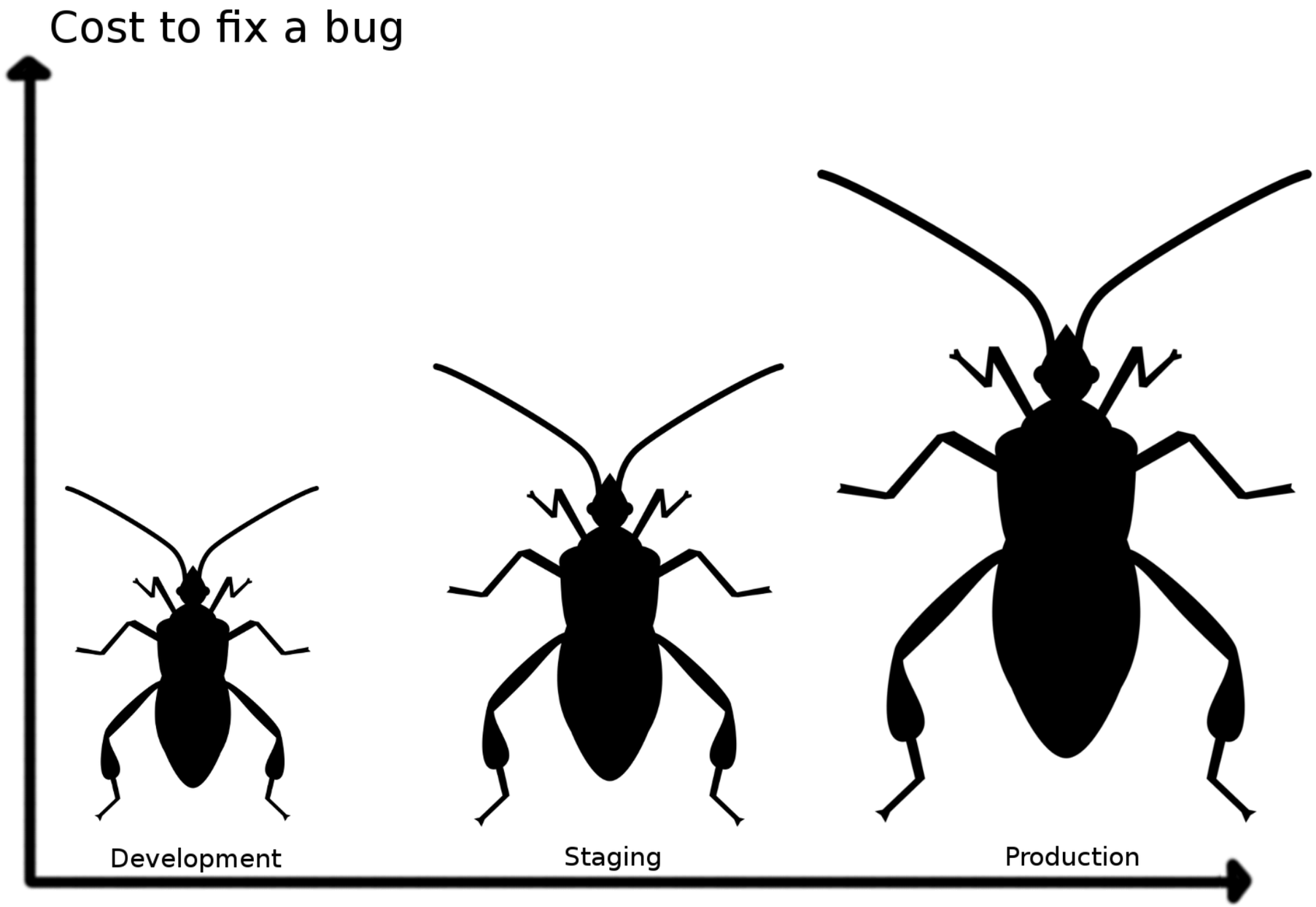
Agenda

TDD (Test-Driven Development)
TDD (Test-Driven Design)
TDD Life Cycle
Rules of TDD
Types of Testing
Workshop (think, design, test, develop)

Agenda

Code coverage
Code smell
Refactoring :: How to improve ?
Good design principle
BDD (Behavior-Driven Development)
Working with Legacy code
Workshop





🐛 Fixing Bugs 🐜

Bug

Find Code

Write Test

Fix Test

Deadline-Driven Development

Later = Never

SPECIFICATION BY EXAMPLE

CONTINUOUS INTEGRATION

CONTINUOUS DELIVERY

TEST AUTOMATION

TECHNICAL EXCELLENCE

ARCHITECTURE & DESIGN

ACCEPTANCE TESTING

CODE

CLEAN CODE

THINKING ABOUT TESTING

TEST-DRIVEN DEVELOPMENT

UNIT TESTING

https://less.works/less/technical-excellence/index.html

https://less.works/less/technical-excellence/index.html

SPECIFICATION BY EXAMPLE

CONTINUOUS INTEGRATION

CONTINUOUS DELIVERY

TEST AUTOMATION

TECHNICAL EXCELLENCE

ARCHITECTURE & DESIGN

ACCEPTANCE TESTING

CODE

CLEAN CODE

THINKING ABOUT TESTING

DEVELOPMENT

TEST-DRIVEN DEVELOPMENT

UNIT TESTING

https://less.works/less/technical-excellence/index.html

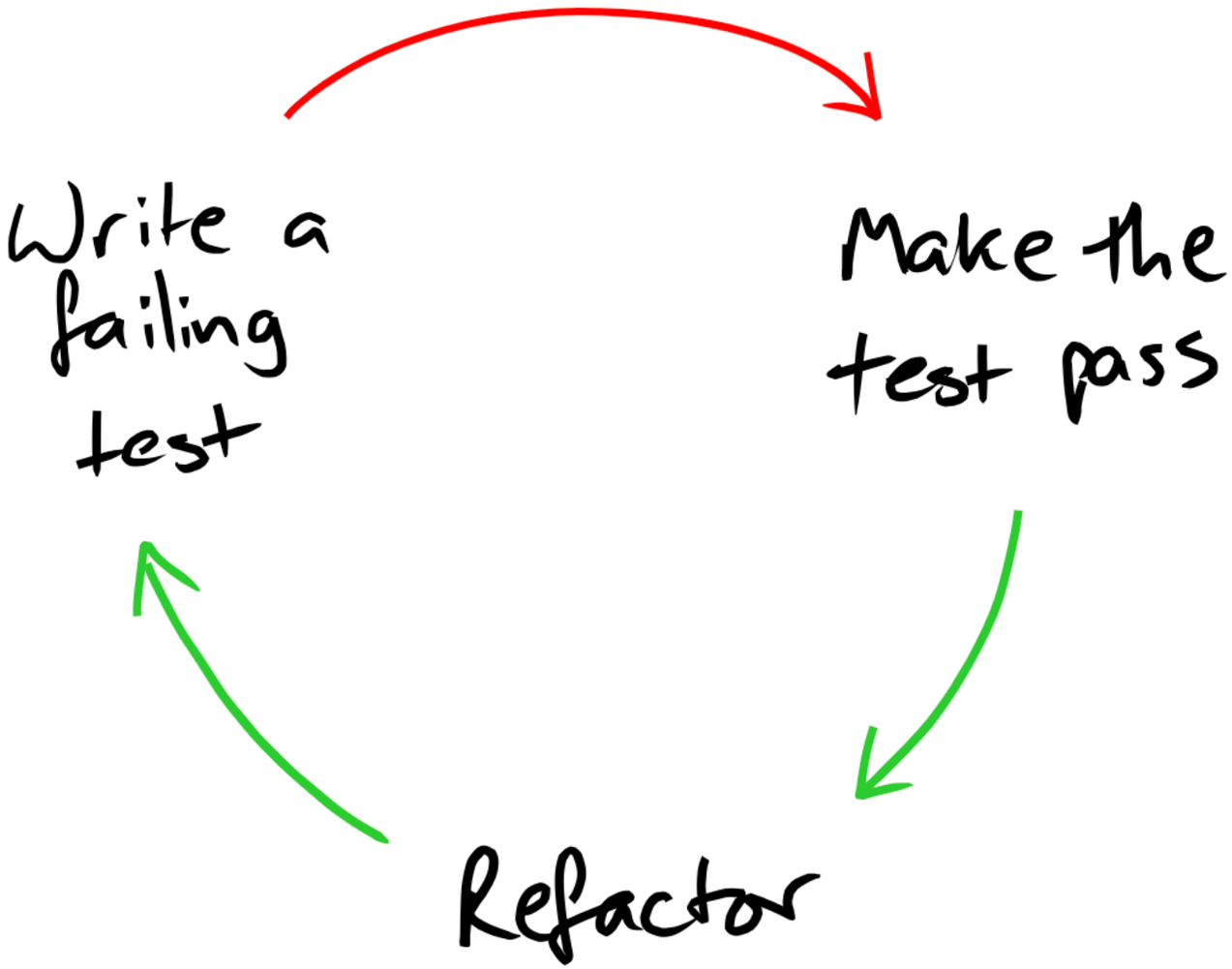
TDD

ALL CODE IS GUILTY UNTIL PROVEN INNOCENT

TDD

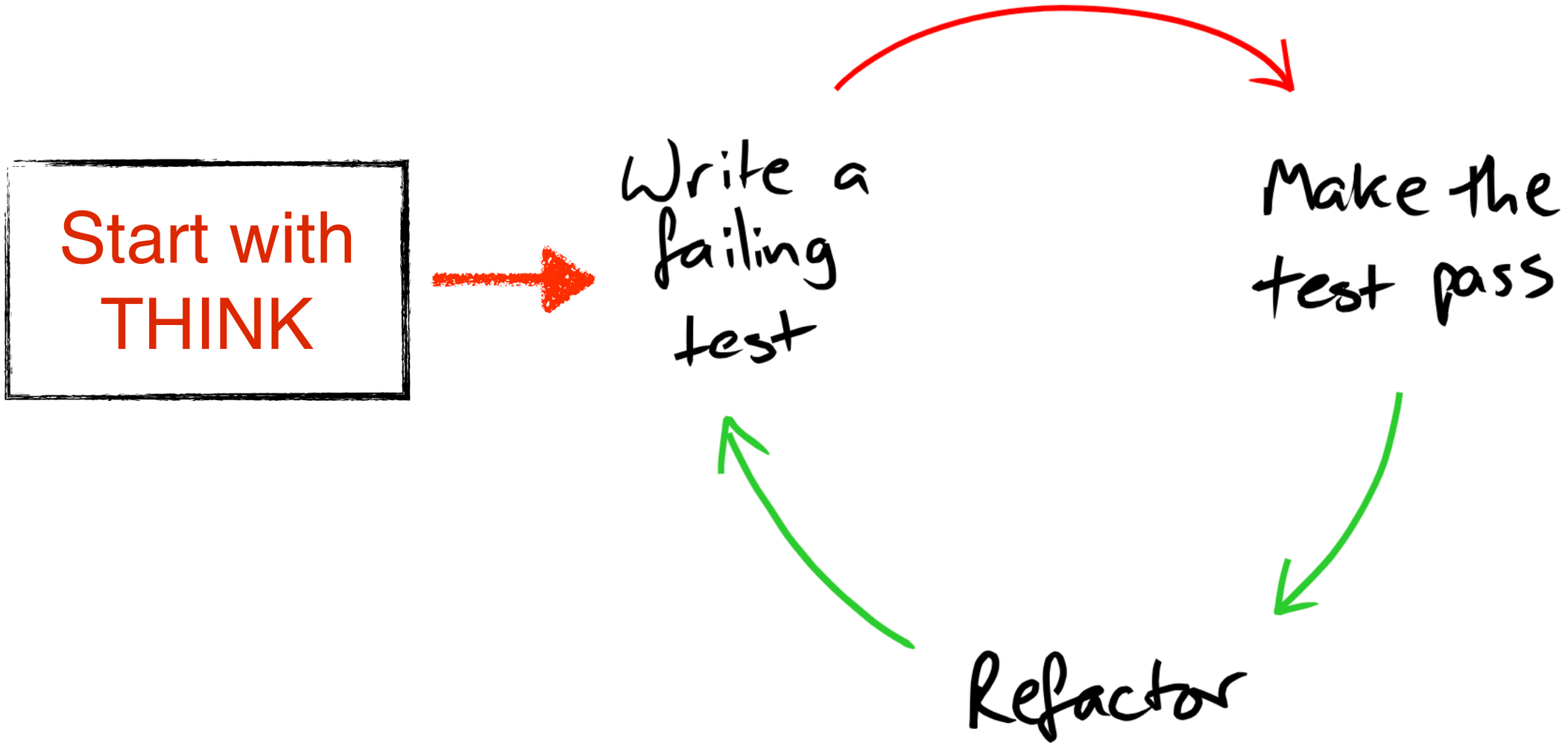
Test-Driven-Development
Test-Driven Design

TDD Cycle

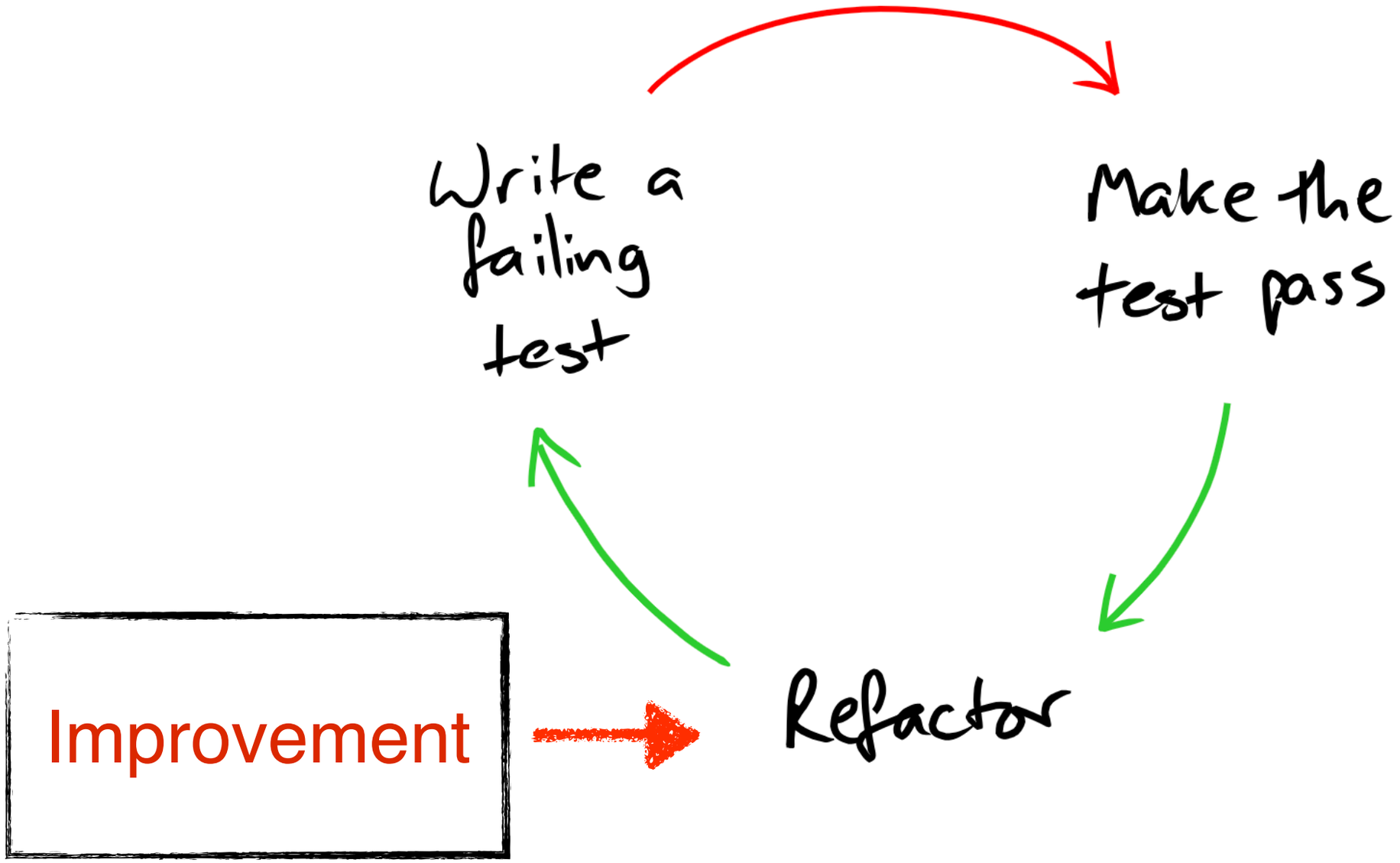


Red Green Refactor

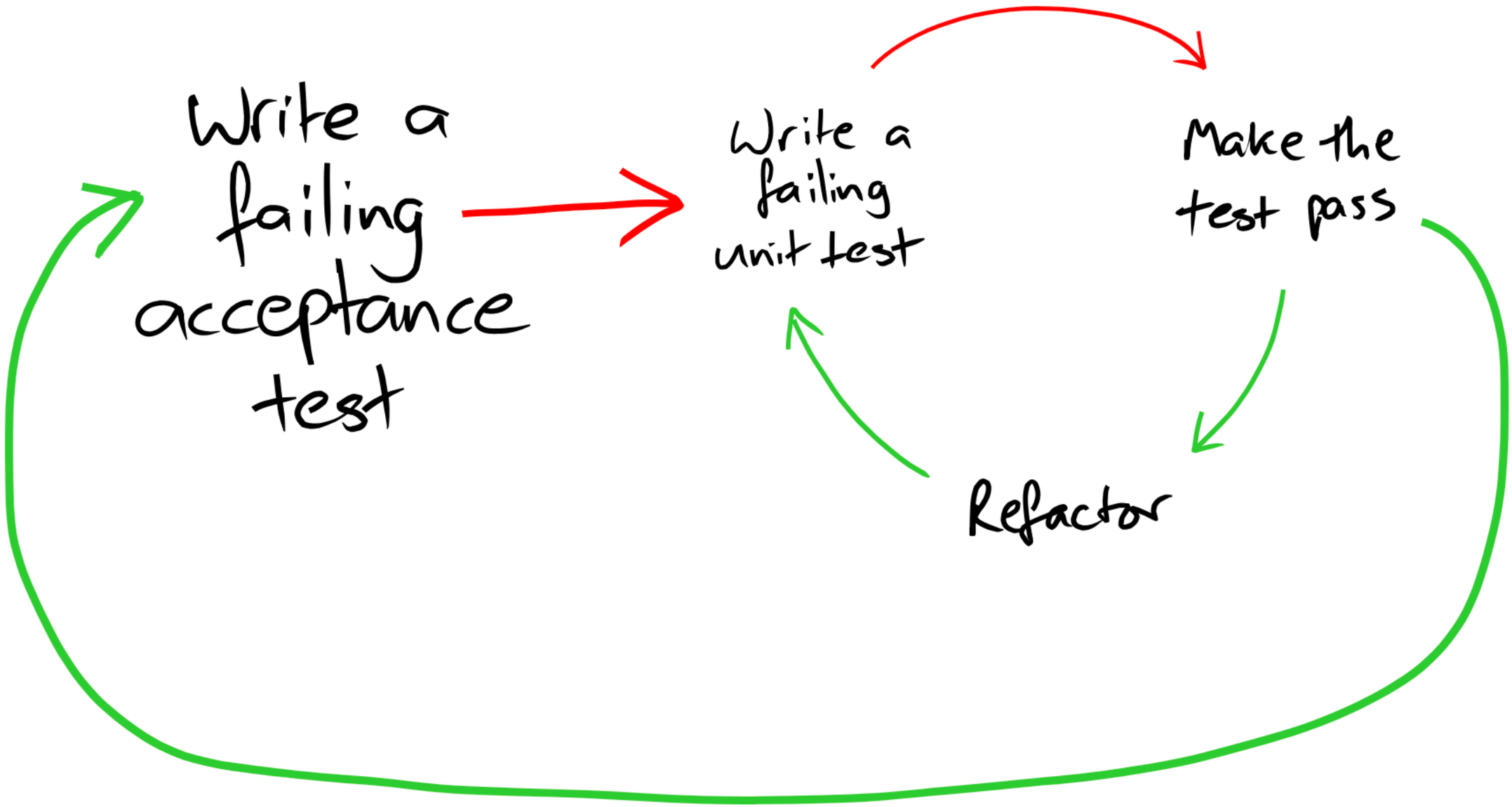
Improve TDD Cycle



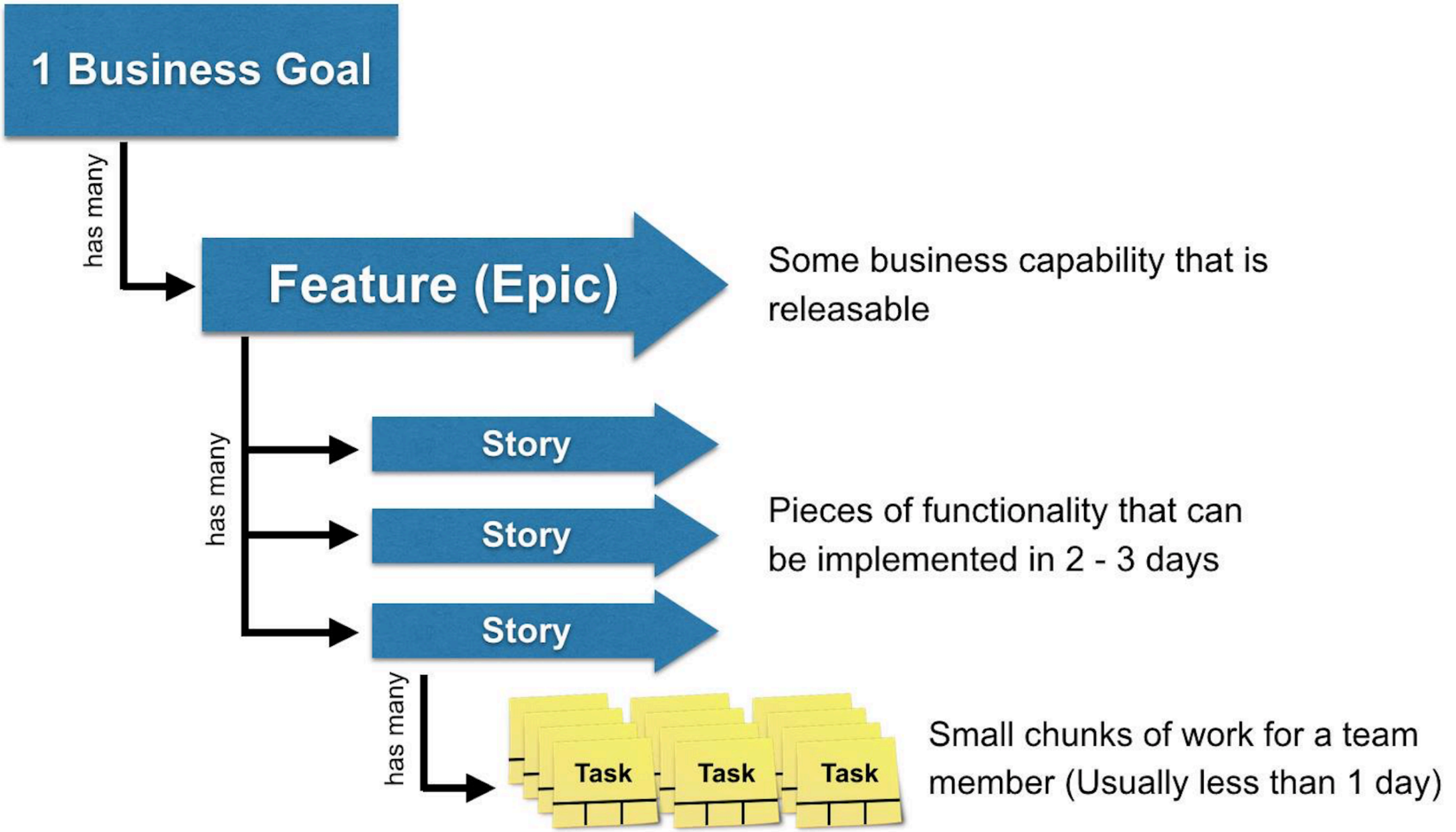
Improve TDD Cycle



Acceptance tests

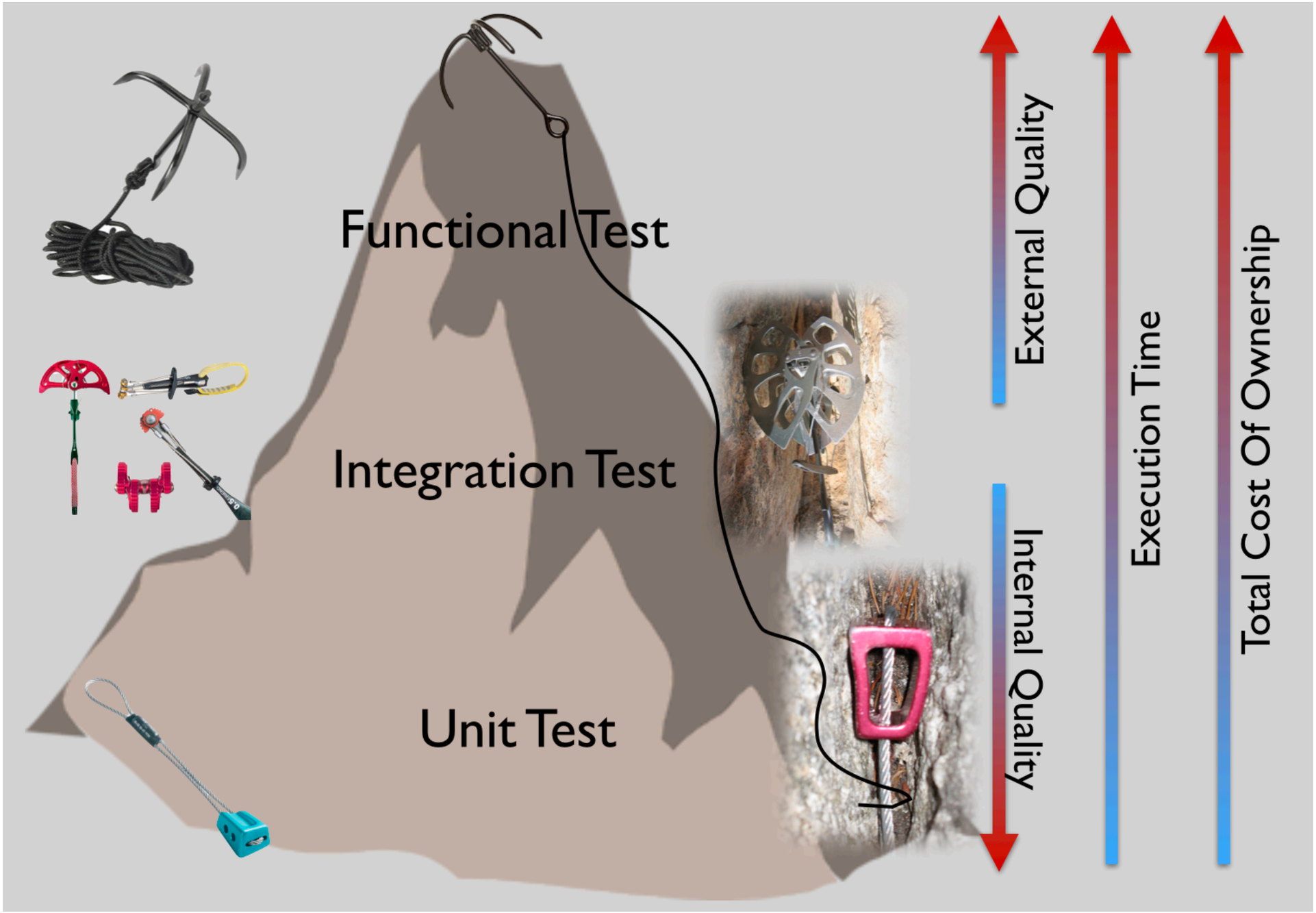


Start with Business Goal to Tasks



Source: The Whole Team Approach to Agile Testing by Janet Gregory and Lisa Crispin

The Golden rule of TDD



https://less.works/jp/less/technical-excellence/unit-testing.html

Types of tests

Unit test

Integration test

Component test

Contract test

End-to-End test

https://martinfowler.com/articles/microservice-testing/

The Golden rule of TDD

Discipline #1

You are not allowed to write production code until you have first written a failing unit test.

Discipline #2

You are not allowed to write more of a unit test than is sufficient to fail, and not compiling is failing.

Discipline #3

You are not allowed to write more production code than is sufficient to pass the currently failing test.

The Golden rule of TDD

"Never write a line of code without a failing test"

- Kent Beck -

Workshop

Input	Expected result
[1, 5]	1,2,3,4,5
[1, 5)	1,2,3,4
(1, 5]	2,3,4,5
(1, 5)	2,3,4

http://codingdojo.org/kata/Range/

Organize your tests

How to make your tests visually **consistent** ?

Keeping tests maintainable by testing **behavior**

The importance of **test naming**

Using **test's life cycle**

Good test structure

1. Setup the test data
2. Call your method under test
3. Assert that the expected results are returns

Good test structure

AAA (Arrange Act Assert)
Given When Then from BDD style

https://www.martinfowler.com/bliki/GivenWhenThen.html
https://xp123.com/articles/3a-arrange-act-assert/

Make the test pass !!

Fake or hard code
Triangulation
Professional code

Test should be **FIRST**

Use your **Right-BICEP**

Check **CORRECT** boundary

Good Unit Tests (F.I.R.S.T)

Fast

Independent/Isolate

Repeat

Self-validate

Thorough/Timely

What to test ?

Right-BICEP

Right

Are the results right ?

Boundary conditions

Are all the boundary conditions correct ?

With **CORRECT**

CORRECT

Conformance

Ordering

Range

Reference

Existence

Cardinality

Time

Conformance

Does the value conform to an expected format?

You should validate and test that it's in an expected format

You should also check what happens if it is not

Ordering

Is the set of values ordered or unordered as appropriate?

Range

Is the value within reasonable minimum and maximum values?

Reference

Does the code **reference** anything **external** that isn't under direct control of the code itself ?

Check how the test will work when the **external state** is as expected

How it will work if **external state** is changed

Existence

Does the value exist (is it non-null, nonzero, present in a set, and so on)?

Cardinality

Are there exactly enough values?

If you are putting collections into the methods, check how your code will act with different collection sizes

Zero, One and Many

Time

Is everything happening in order? At the right time? In time?

You should check for the order of actions

Absolute, relative and concurrency

Back to **Right-B**ICEP

Inverse relationships

Can you check inverse relationships ?

Cross check

Can you cross check results using the other means ?

Error conditions

Can you for error conditions to happen ?

Unhappy paths

Performance

Are performance characteristics within bounds ?

Test Life Cycle

@BeforeClass

@Before

@After

@AfterClass

Code coverage

"Code coverage can show the high risk areas in a program, but never the risk-free."

Paul Reilly, 2018, Kotlin TDD with Code Coverage

Code coverage

A tool to measure how much of your code is covered by tests that break down into classes, methods and lines.

Code coverage

Class coverage
Method coverage
Line coverage
Branch coverage

Code coverage

```
1.  public class MyRange {
2.      public boolean startWithInclude(String input) {
3.          return input.startsWith("[");
4.      }
5.
6.      public boolean endWithInclude(String input) {
7.          if(input == null) {
8.              return false;
9.          }
10.         return input.endsWith("]");
11.     }
12.
13.     public boolean startWithInclude2(String input) {
14.         return input.startsWith("[");
15.     }
16. }
```

Code coverage

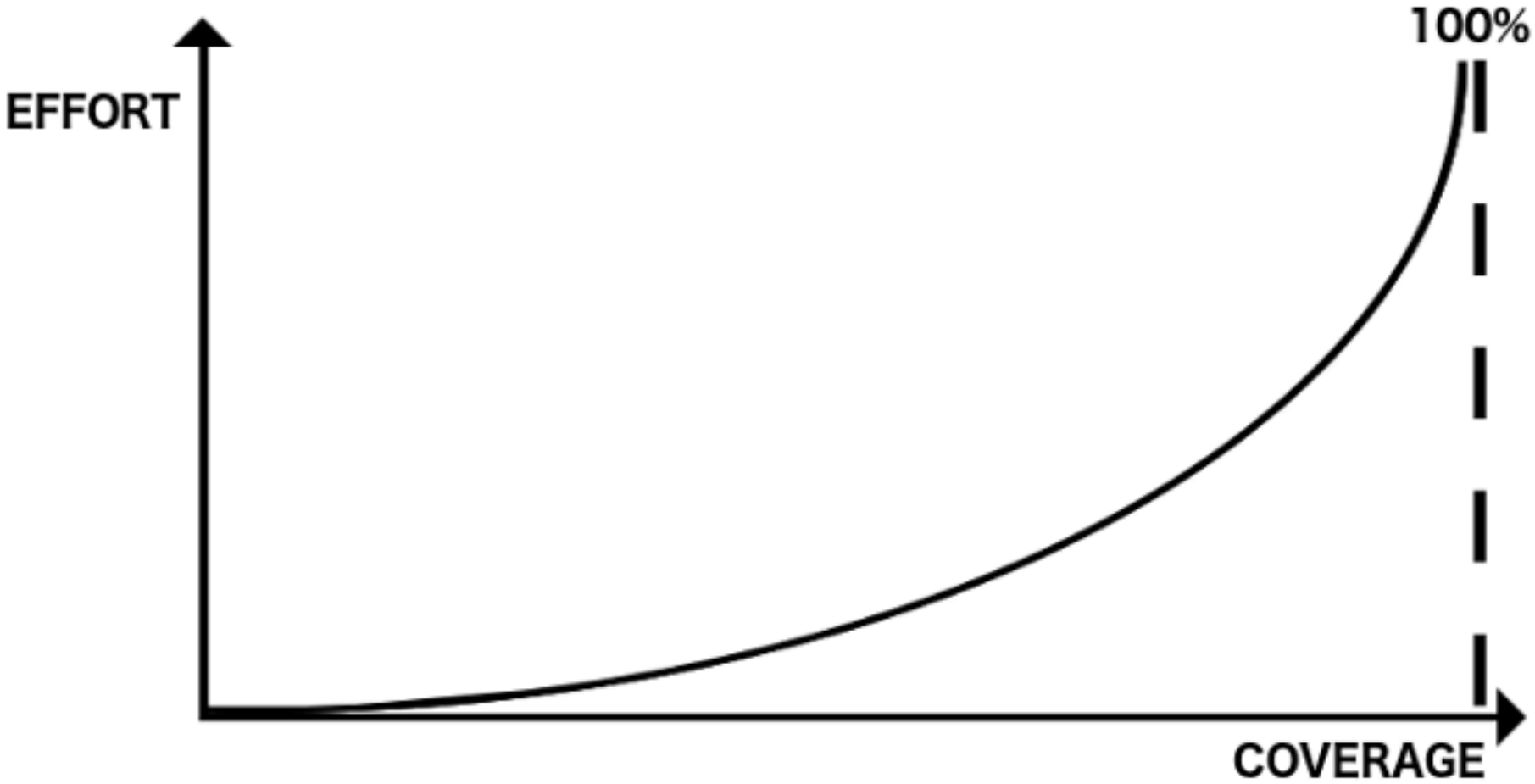
But 100% of code coverage **does not mean** that your code is 100% correct

Code coverage

Powerful tool to improve the quality of your code

Code coverage != quality of tests

Code coverage 100% ?



% of Code/Test coverage

Code coverage with Java

Cobertura
Jacoco

TDD in Iteration development

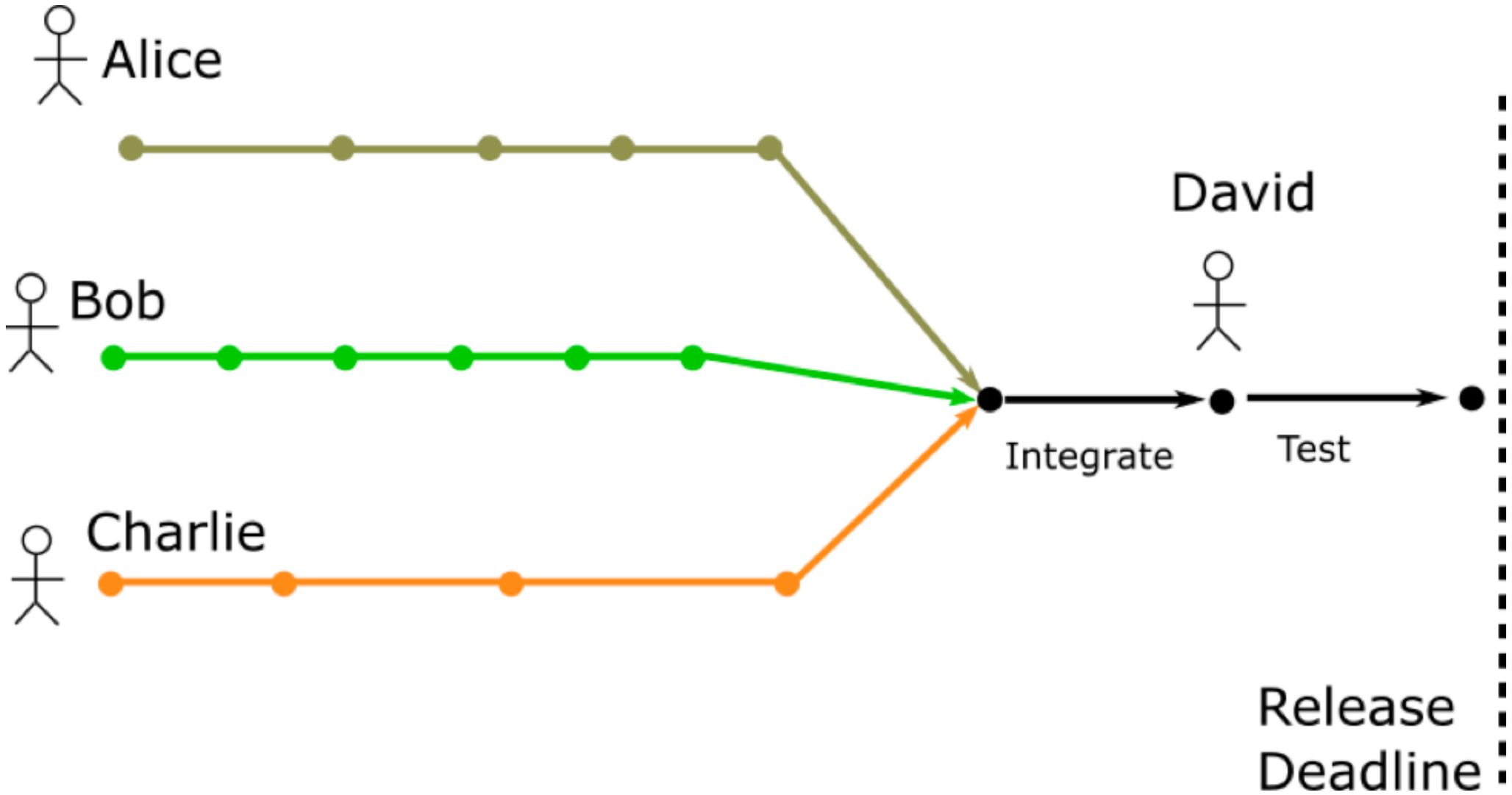
TDD in Iteration development

Iteration cycle 2-4 weeks

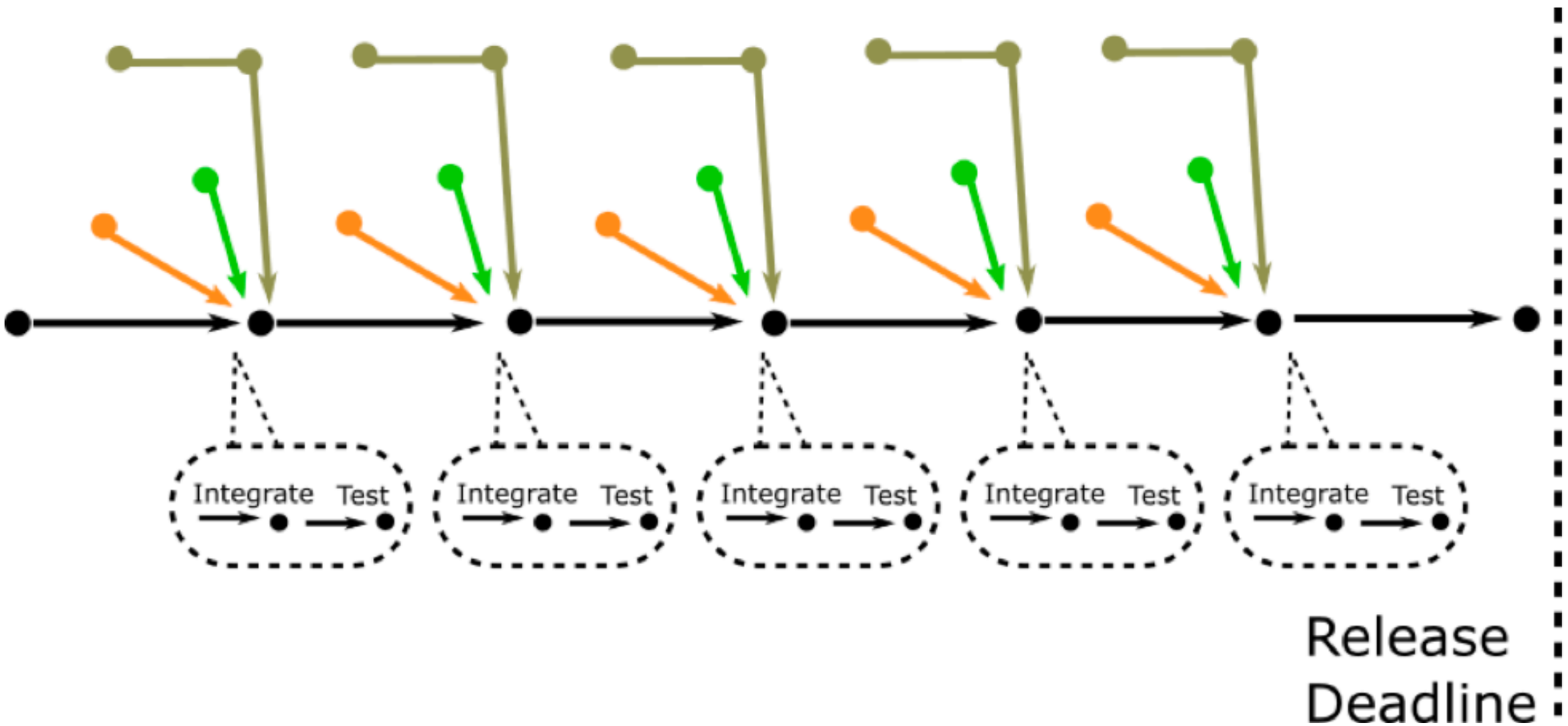
Continuous integration cycle
(multiple times a day)

TDD cycle in a few minutes (3-10 minutes)

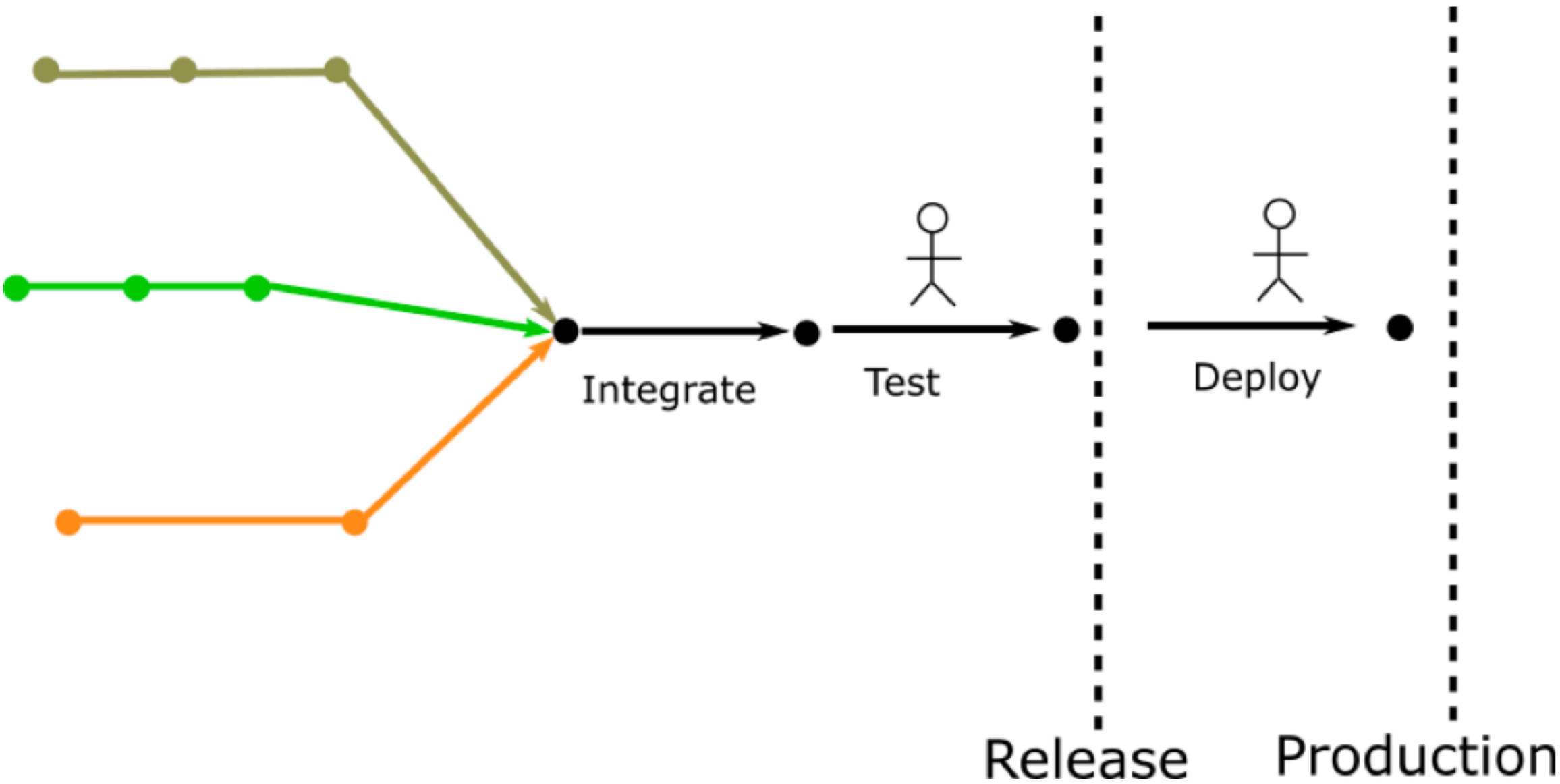
TDD in Iteration development



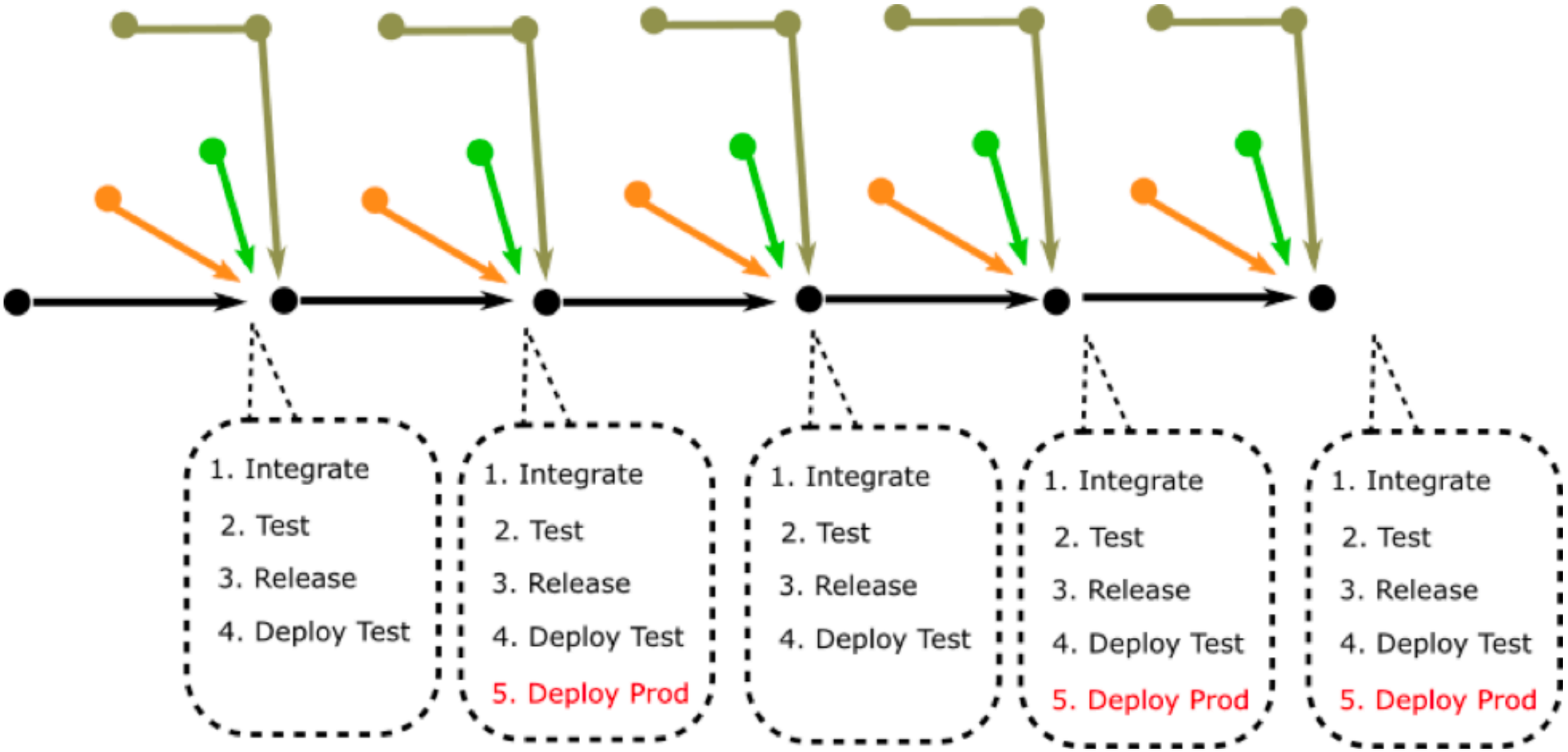
TDD in Iteration development



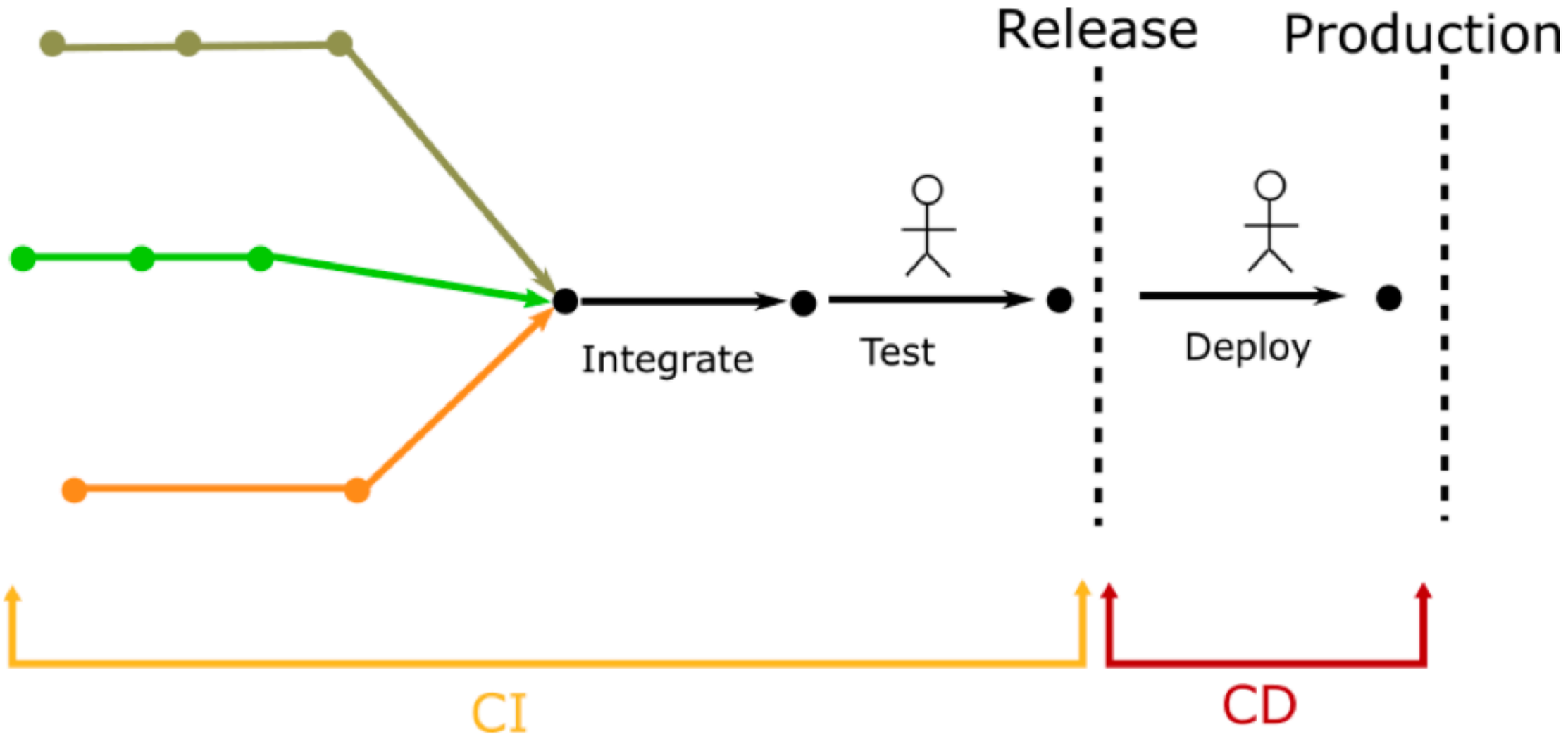
TDD in Iteration development



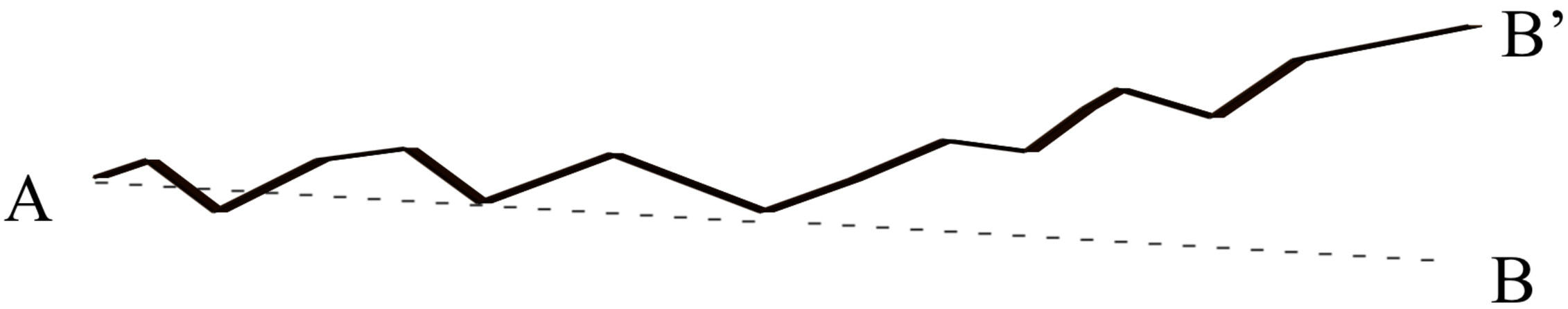
TDD in Iteration development



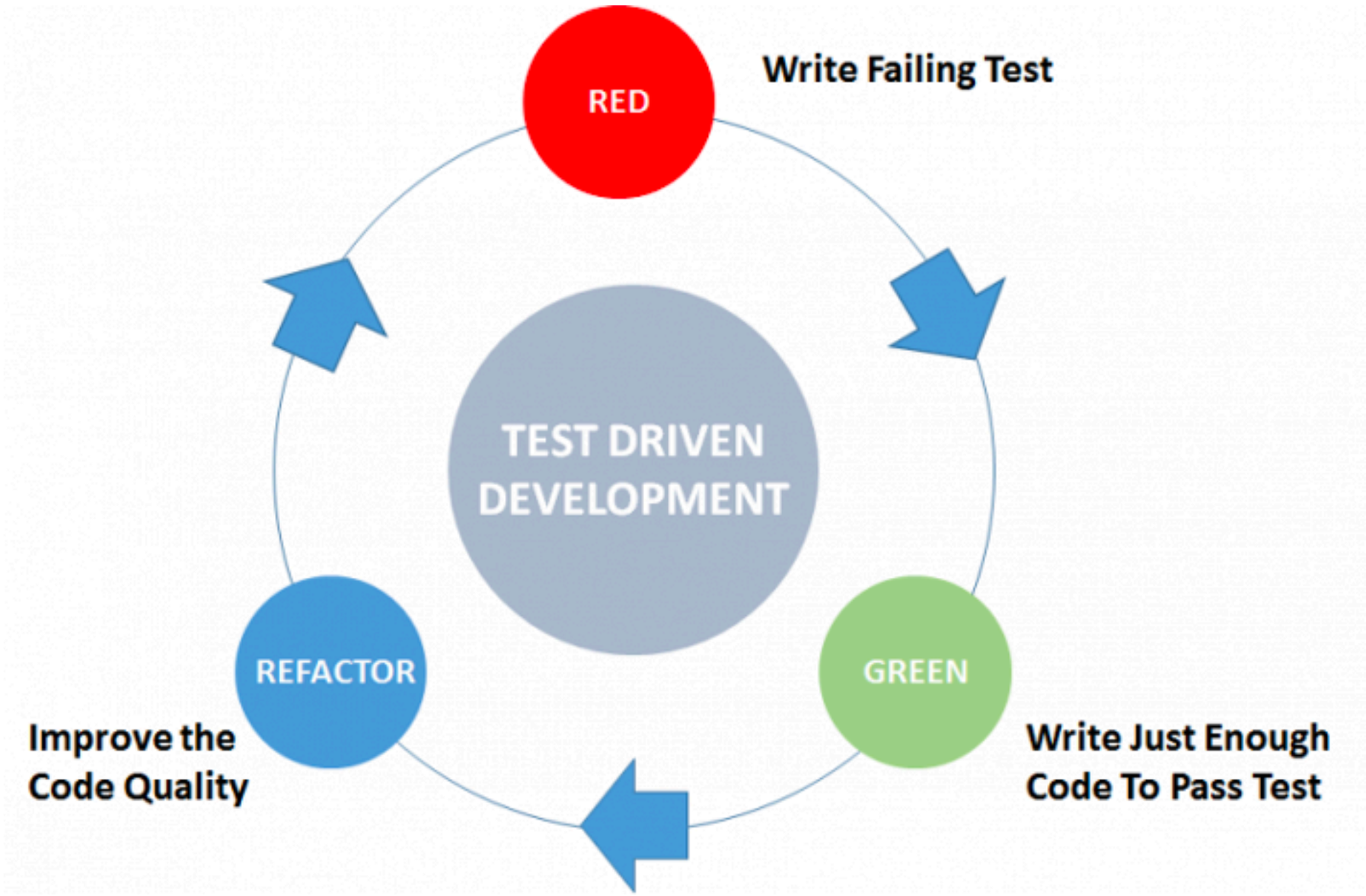
TDD in Iteration development



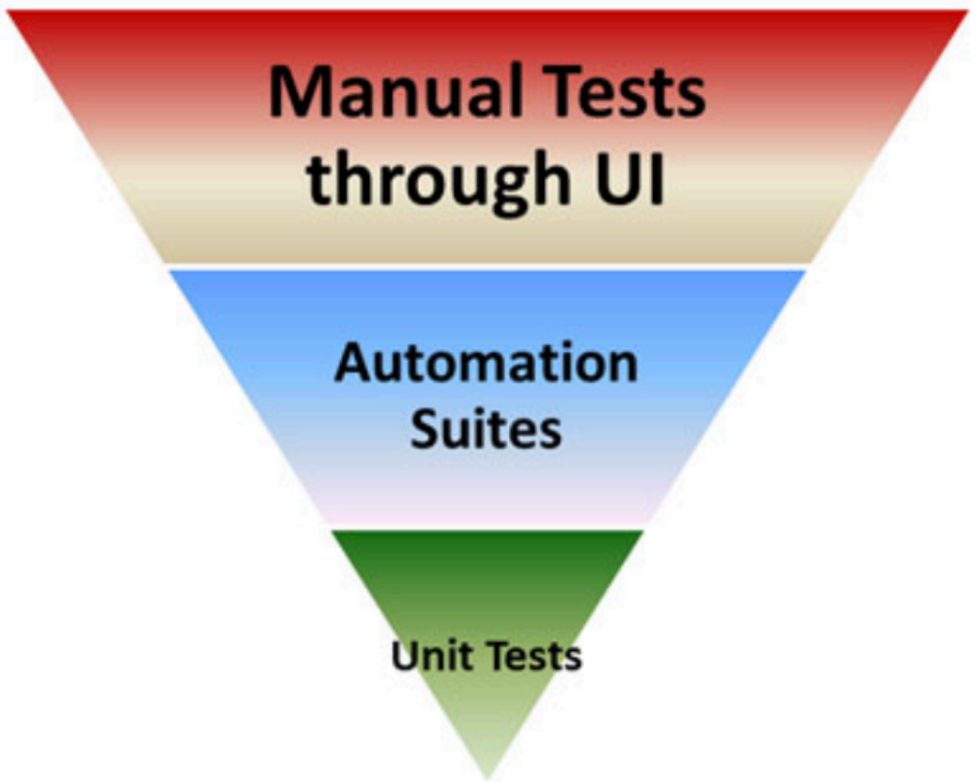
Small steps



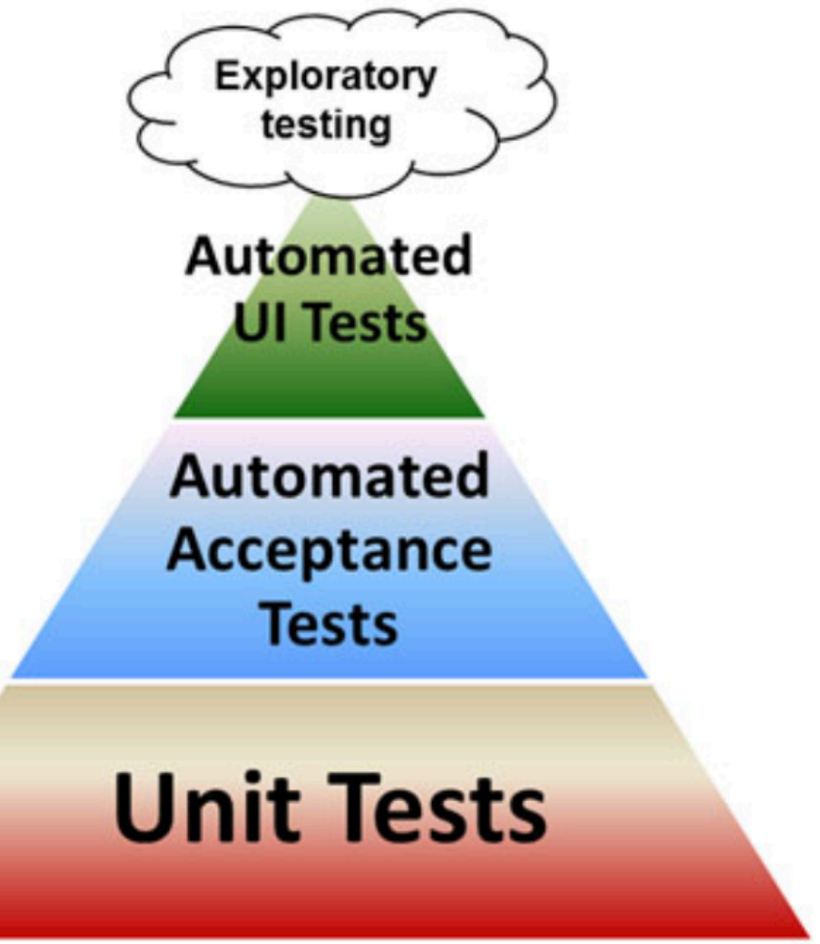
TDD = ทำดีดี



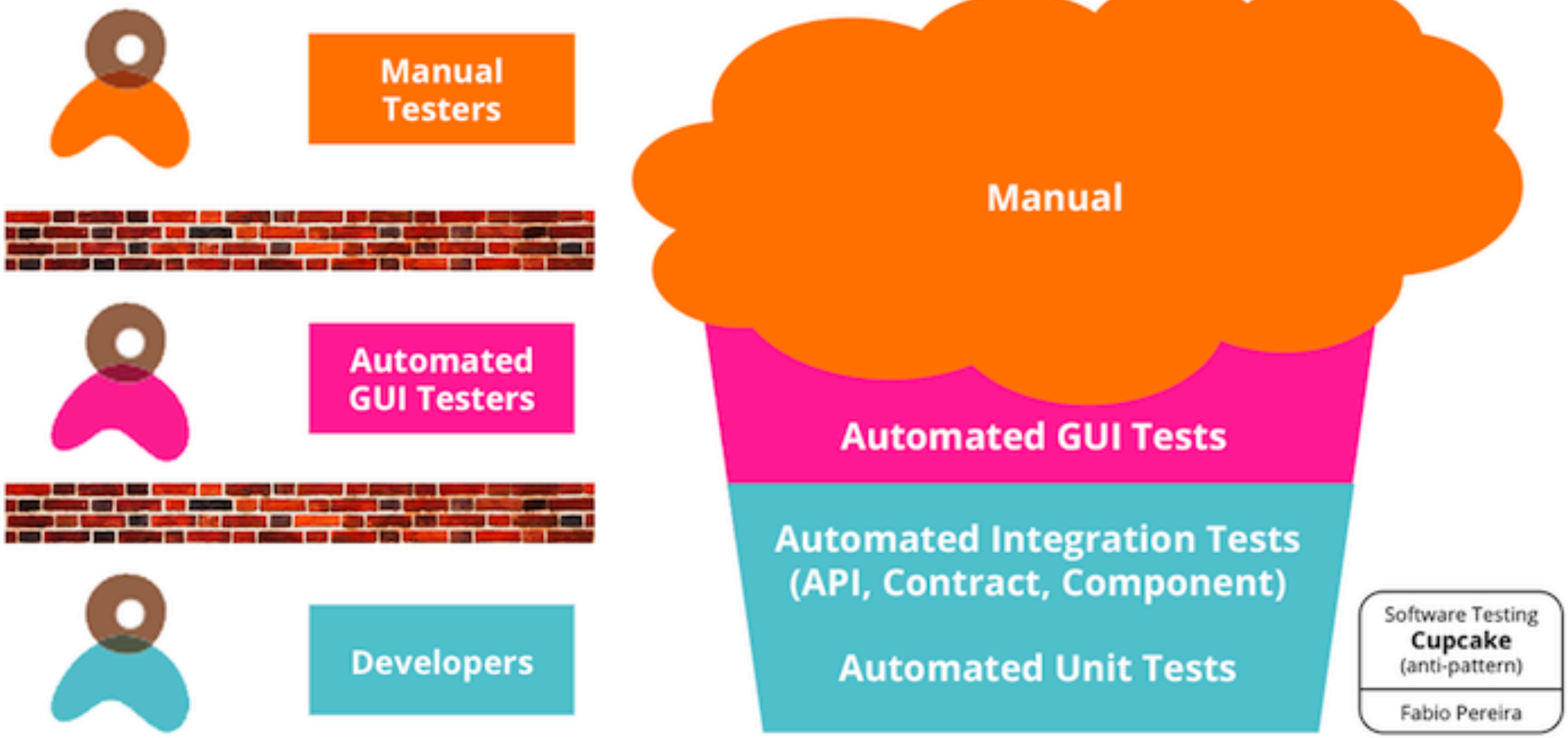
Types of Test



Traditional
(find bugs)



Agile
(prevent bugs)



https://www.thoughtworks.com/insights/blog/introducing-software-testing-cupcake-anti-pattern

THE FOUR TYPES OF TESTS

End to End

A helper robot that behaves like a user to click around the app and verify that it functions correctly.

Sometimes called “functional testing” or e2e.

Integration

Verify that several units work together in harmony.

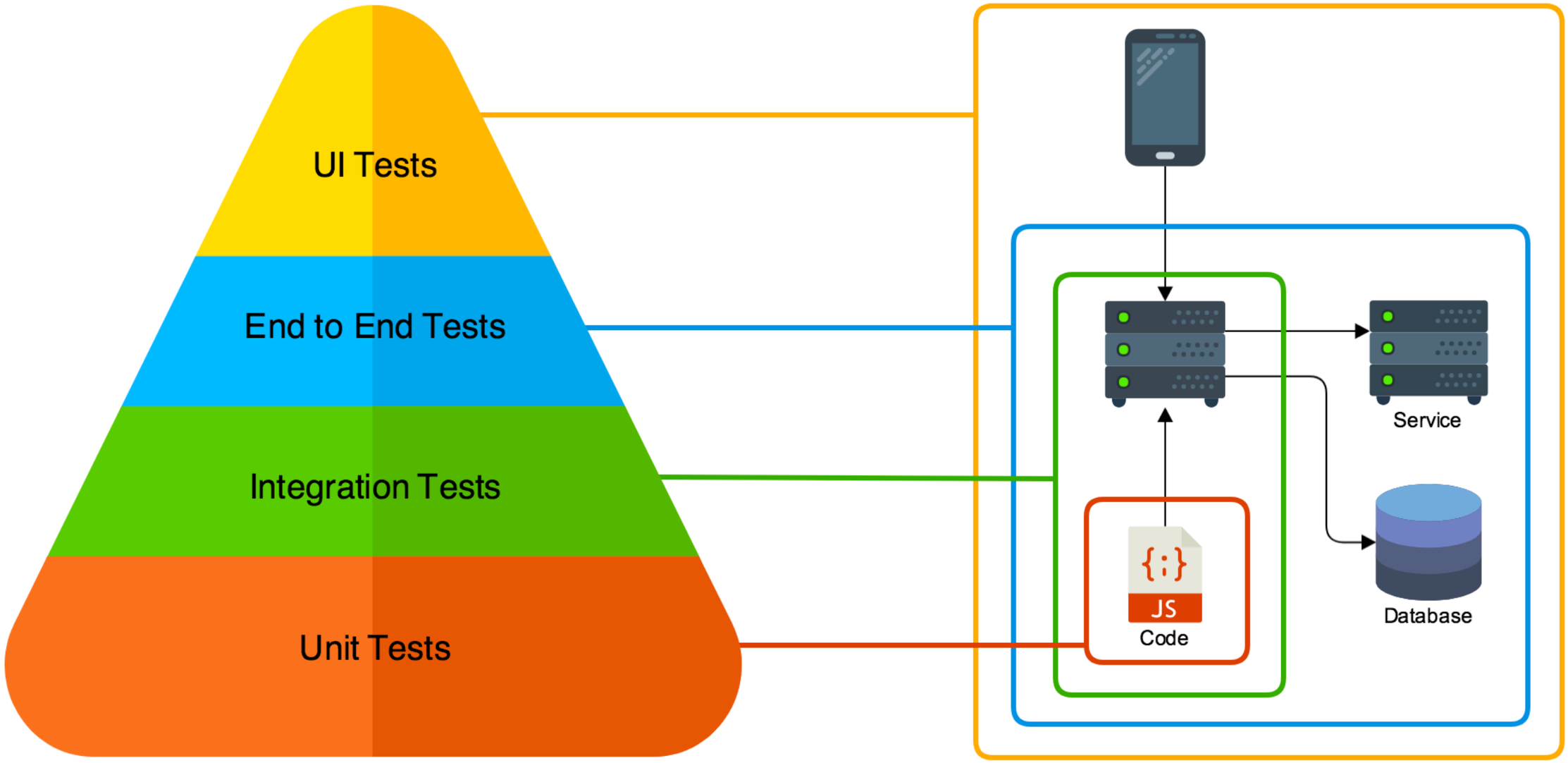
Unit

Verify that individual, isolated parts work as expected.

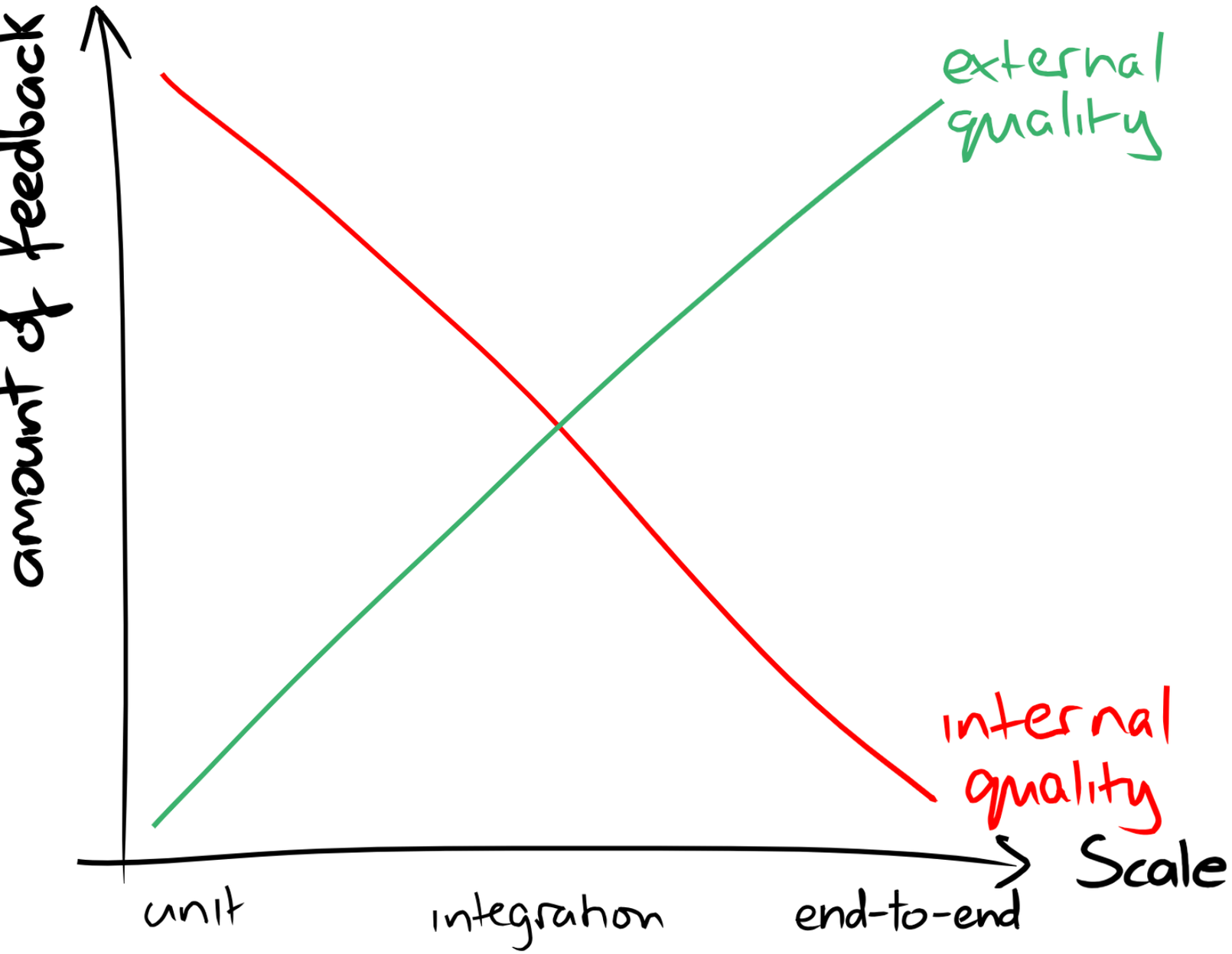
Static

Catch typos and type errors as you write the code.

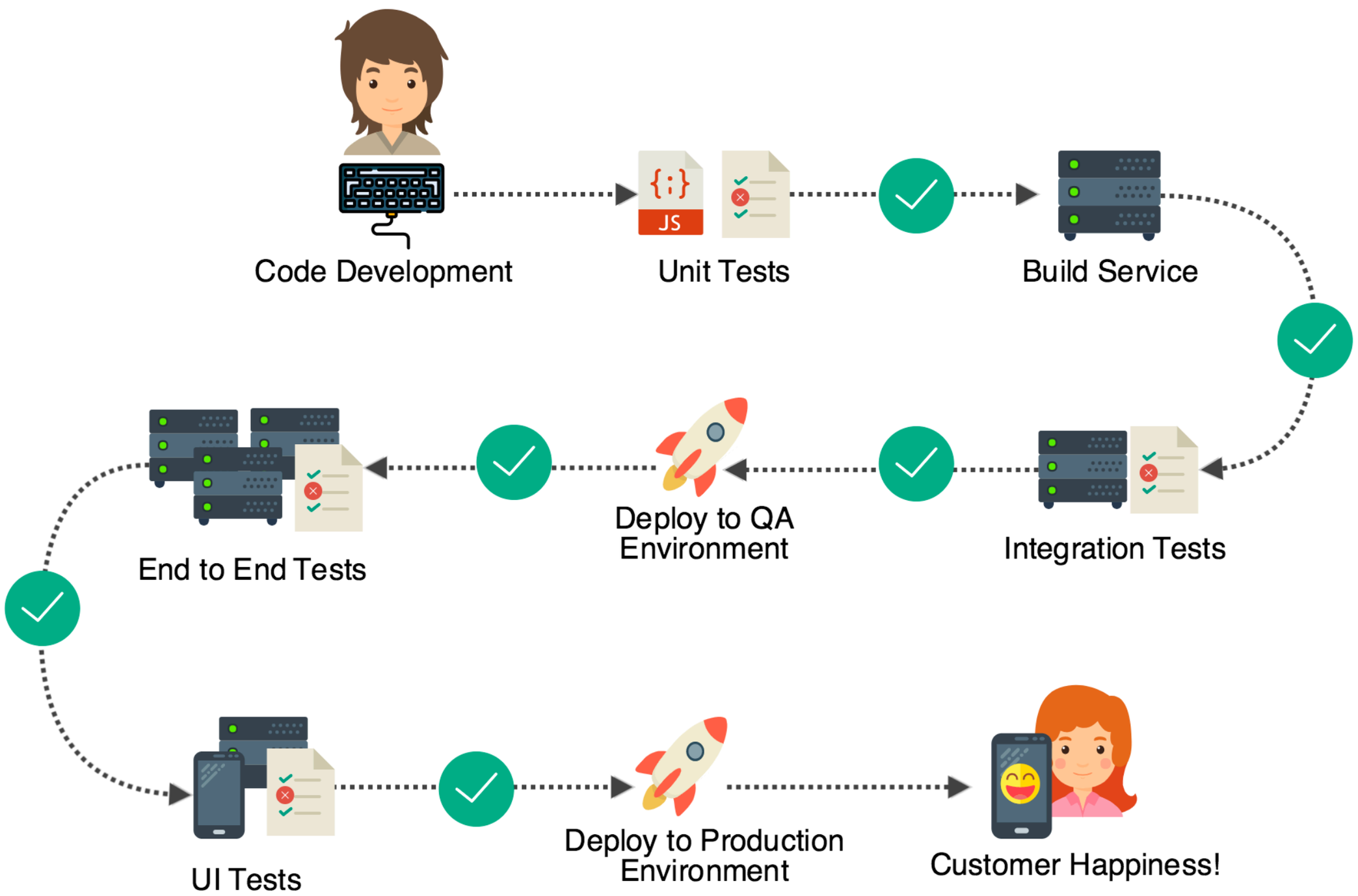
https://testingjavascript.com/



Internal and External quality



Delivery process



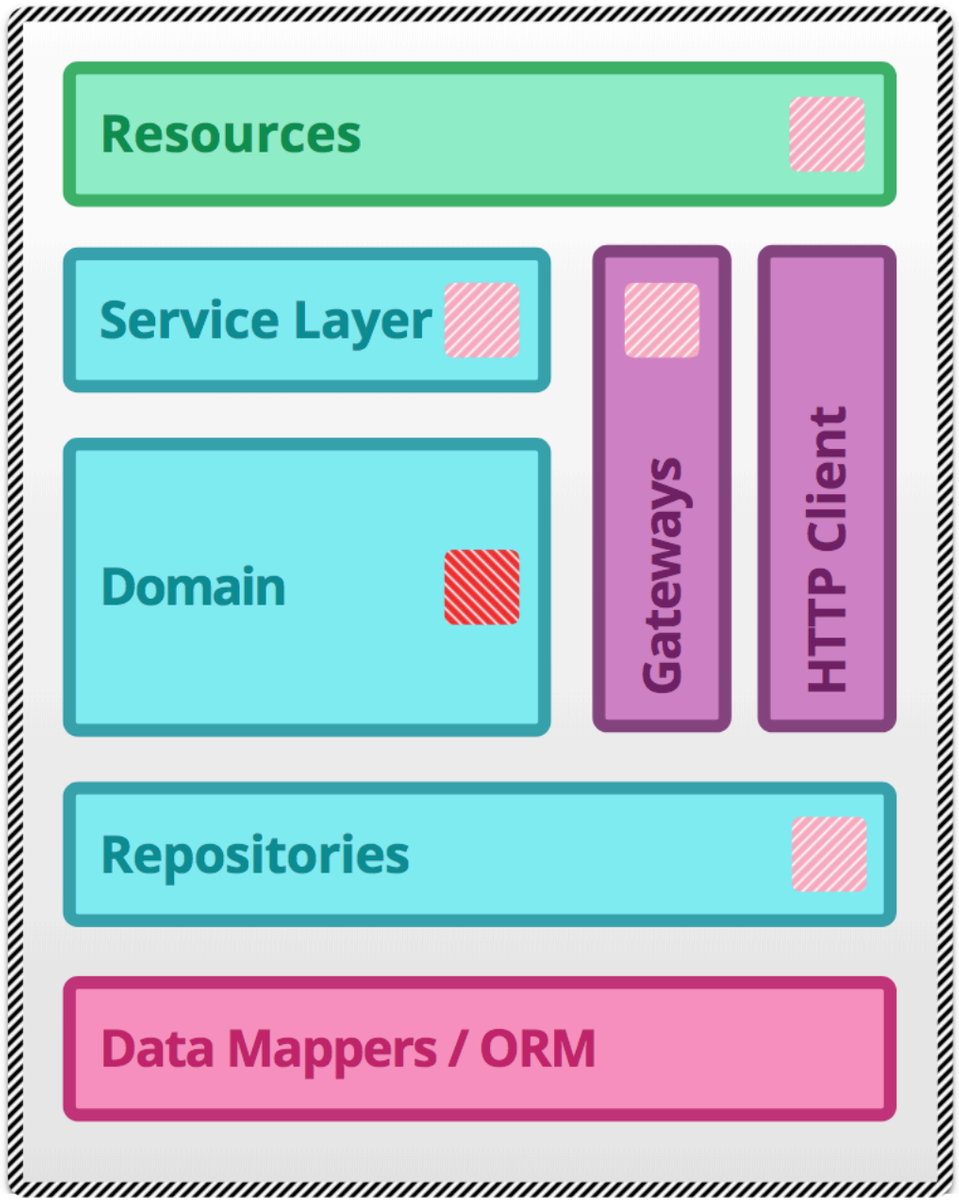
More with Microservices Testing

https://martinfowler.com/articles/microservice-testing/

Testing strategies

Unit
Integration
Component
Contract
End-to-End

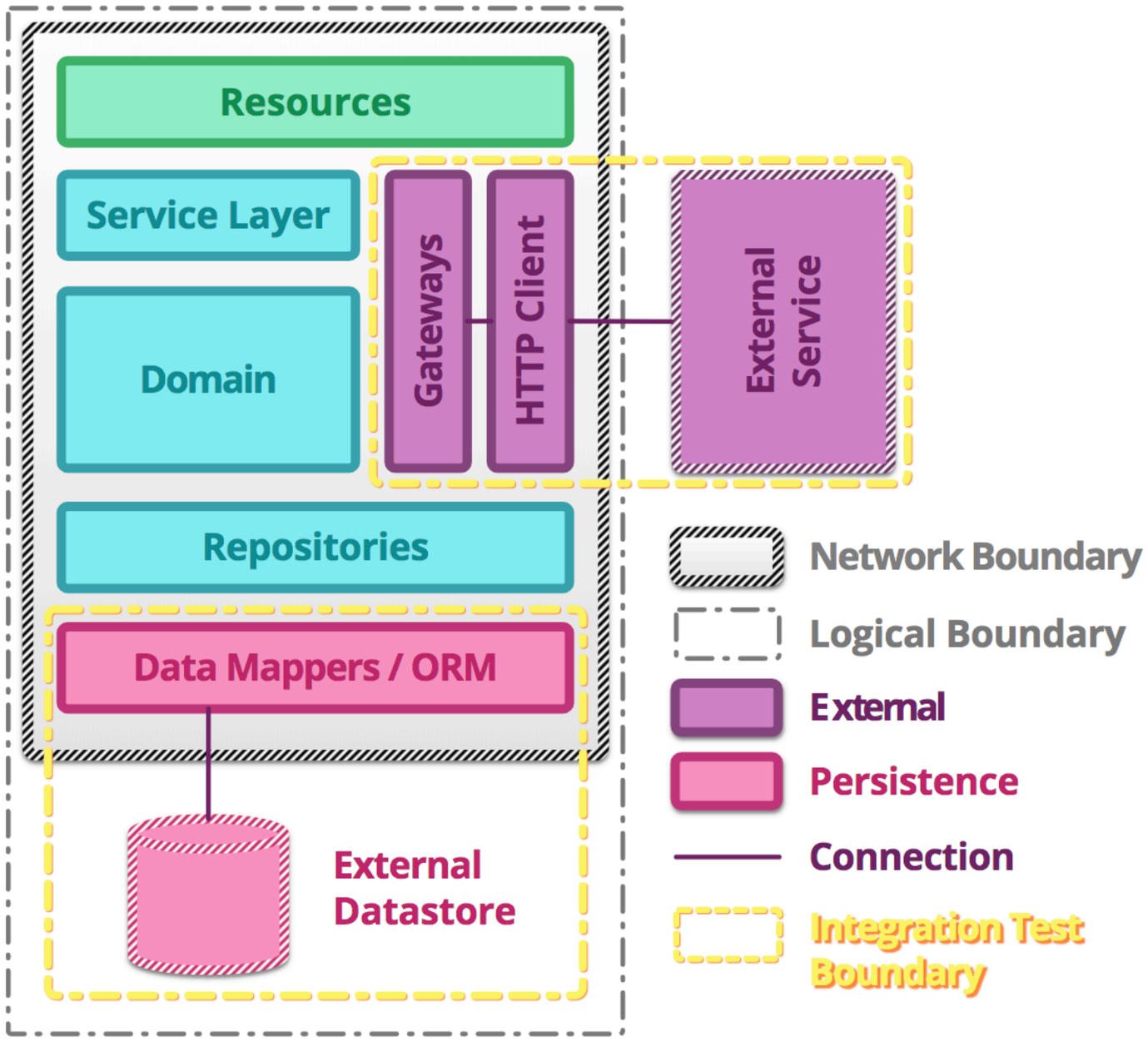
Unit testing



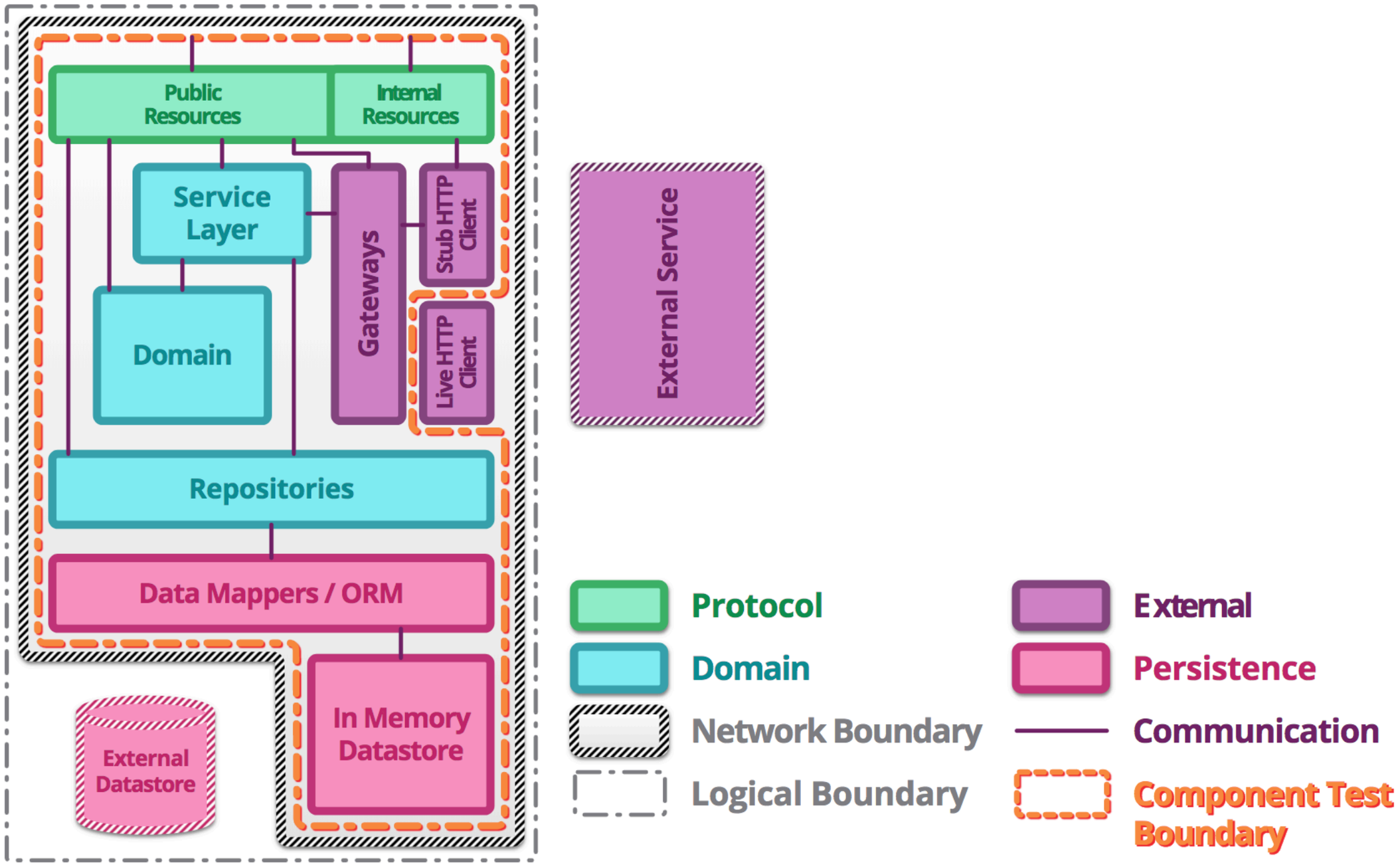
Unit - Solitary

Unit - Sociable

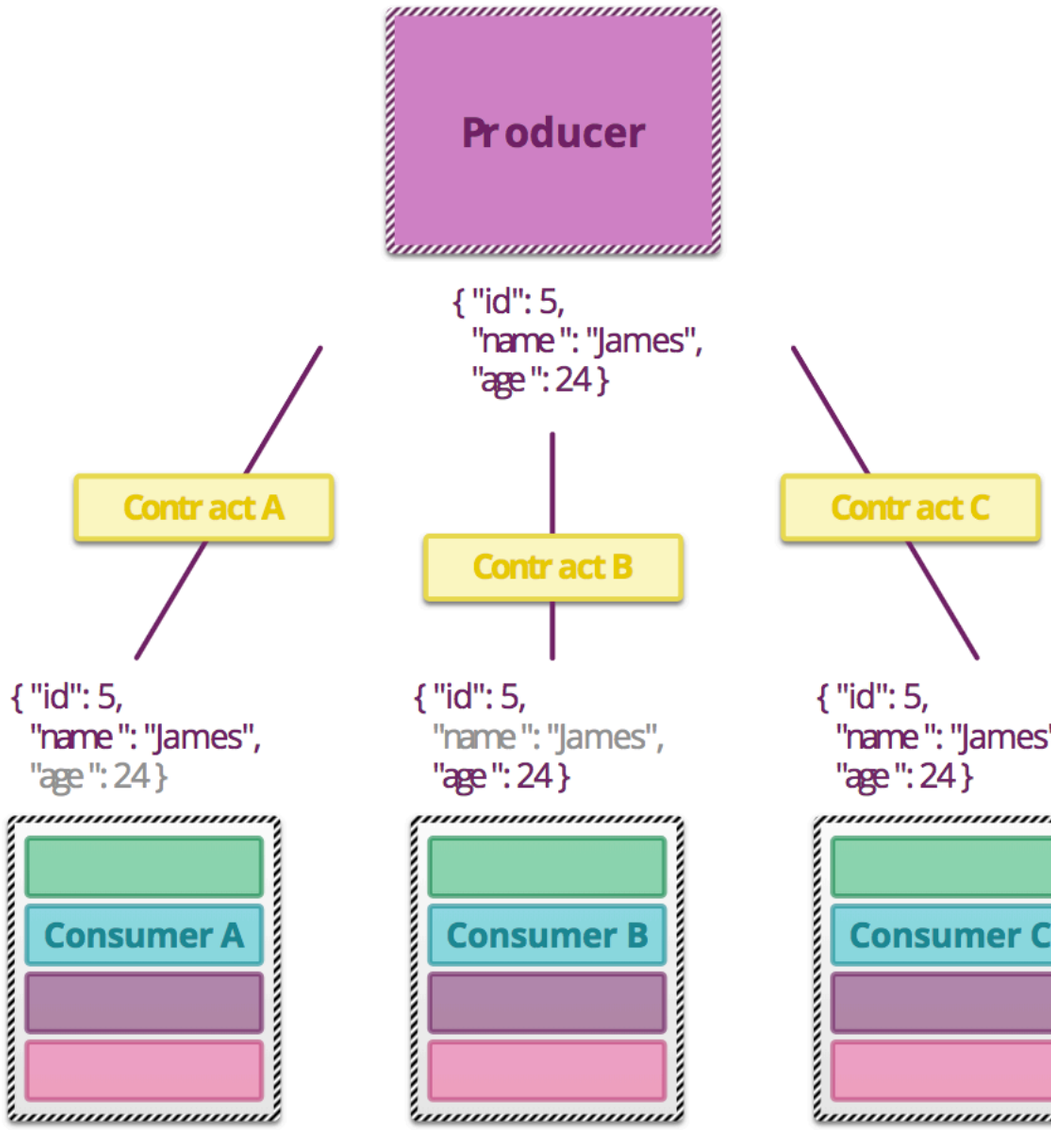
Integration testing



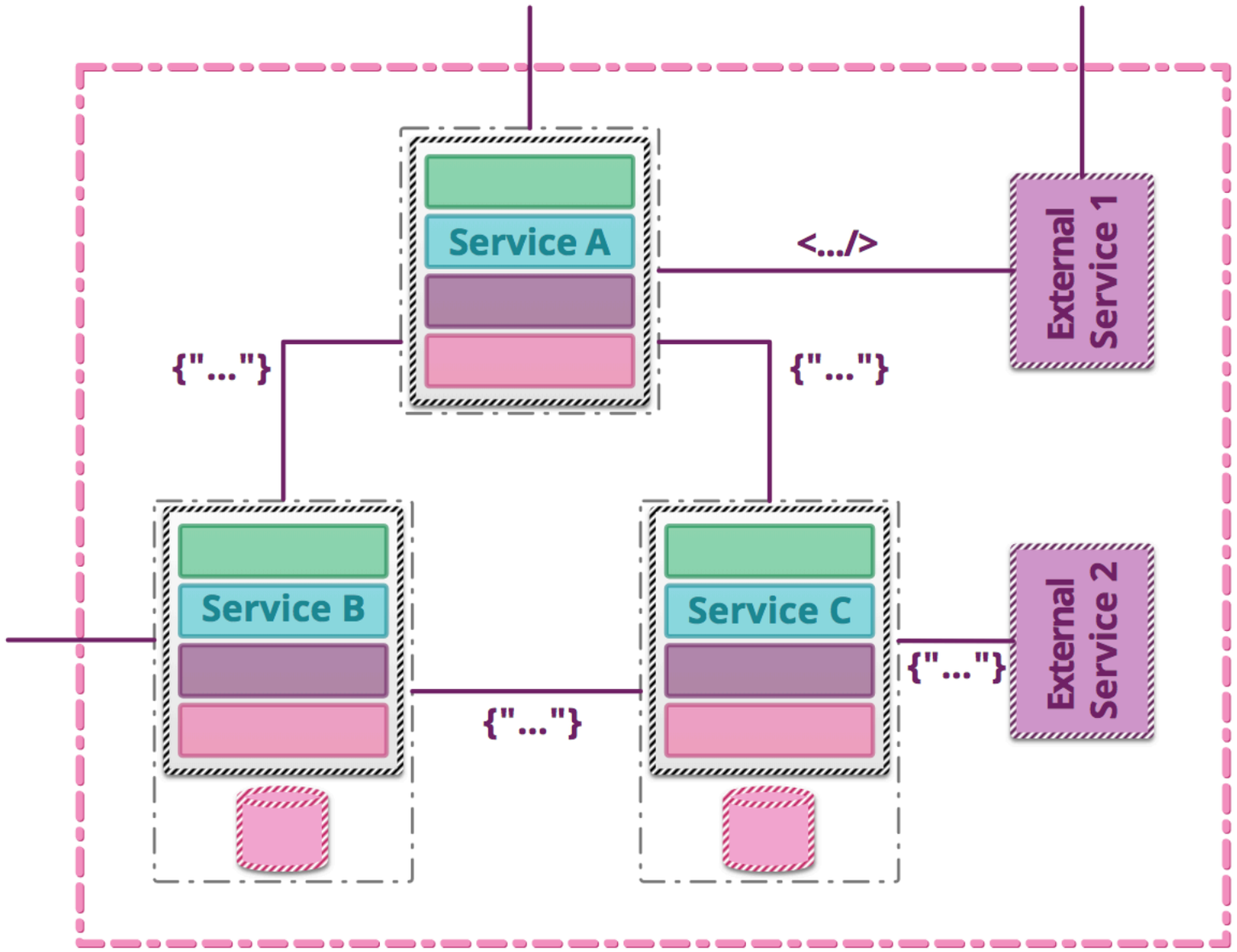
Component testing



Contract testing



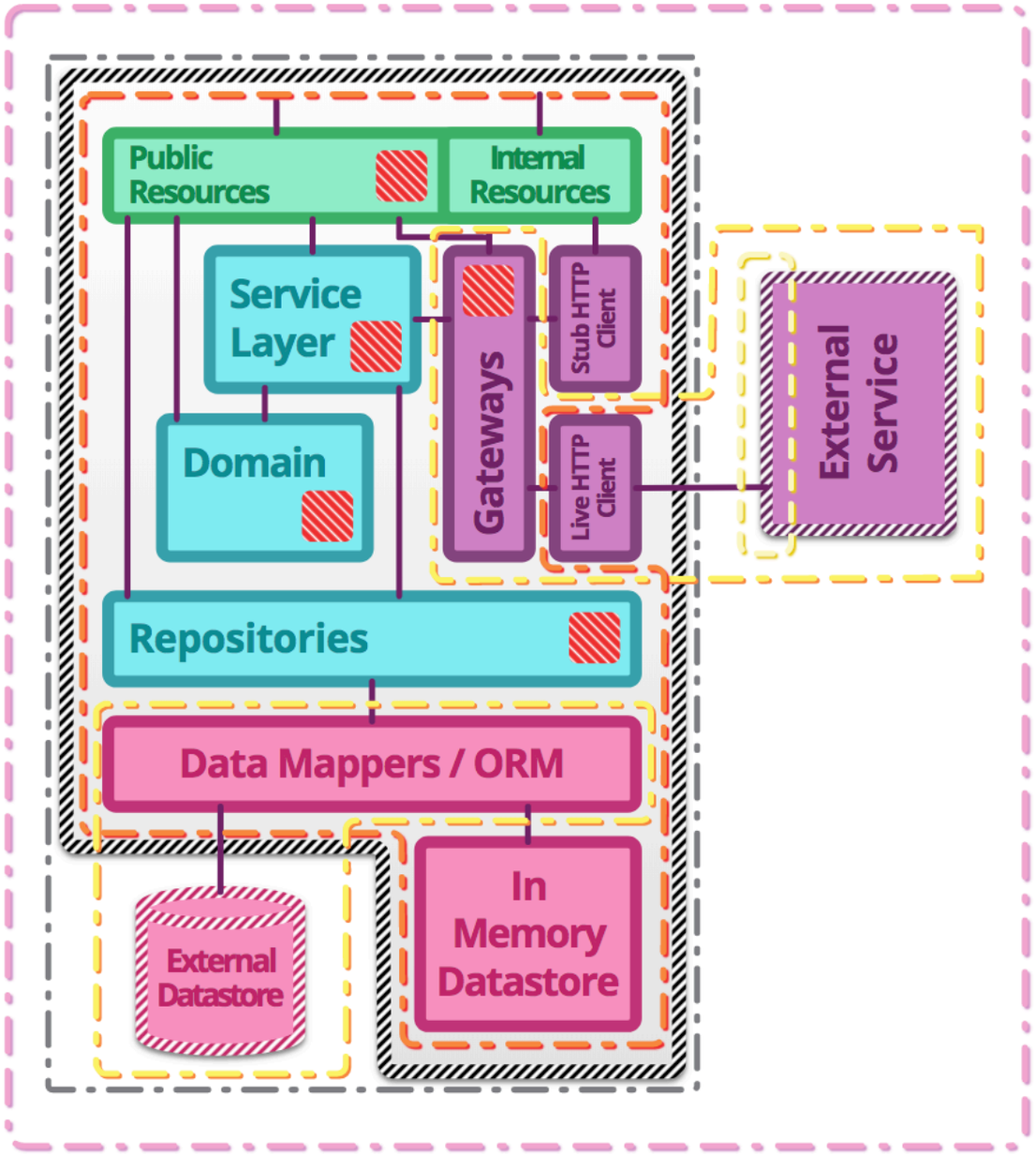
End-to-End testing



Summary

Unit tests : exercise the smallest pieces of testable software in the application to determine whether they behave as expected.

Integration tests : verify the communication paths and interactions between components to detect interface defects.

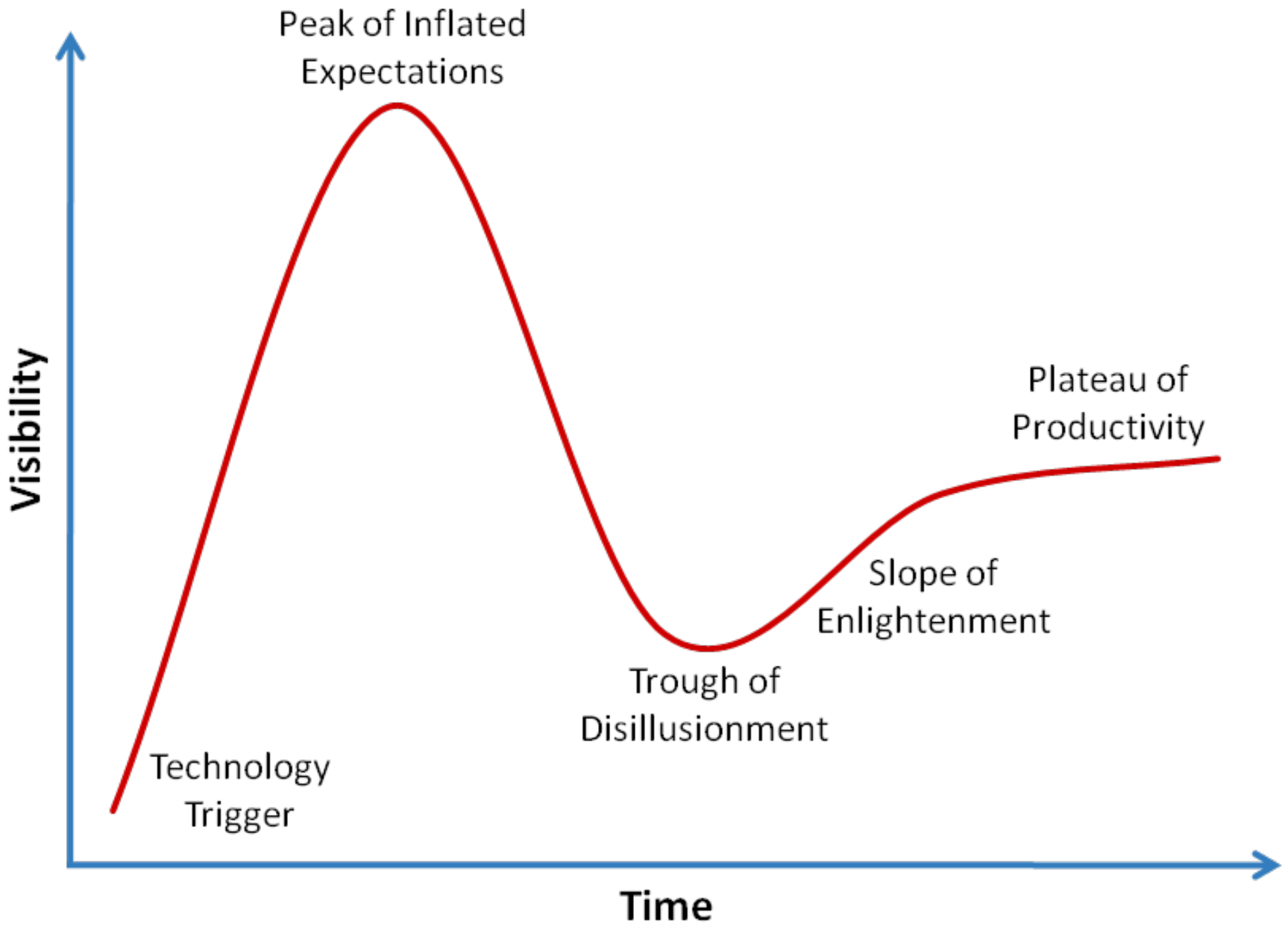


Component tests : limit the scope of the exercised software to a portion of the system under test, manipulating the system through internal code interfaces and using test doubles to isolate the code under test from other components.

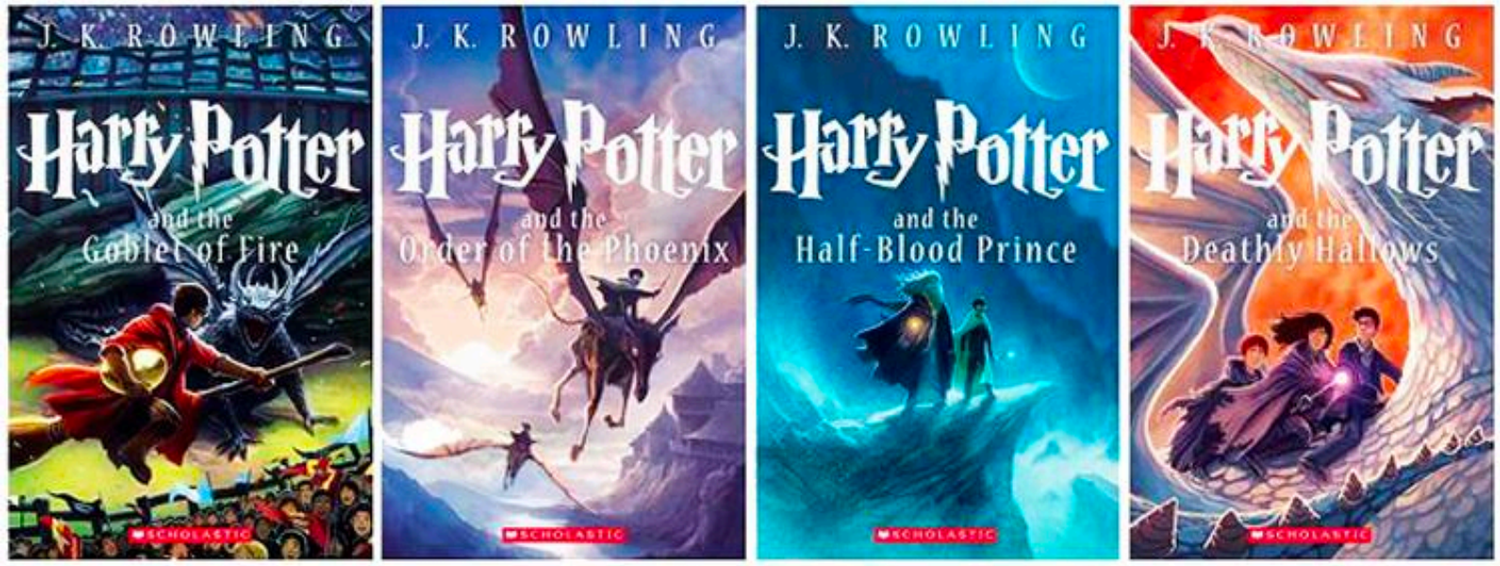
Contract tests : verify interactions at the boundary of an external service asserting that it meets the contract expected by a consuming service.

End-to-end tests : verify that a system meets external requirements and achieves its goals, testing the entire system, from end to end.

Hype cycle



Workshop

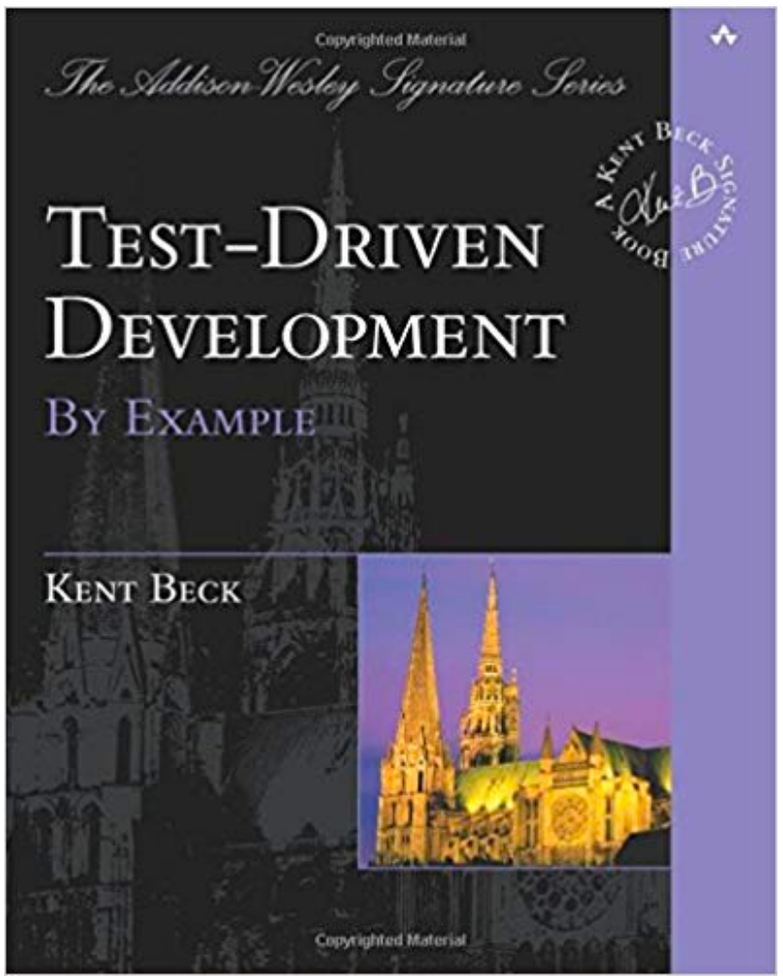


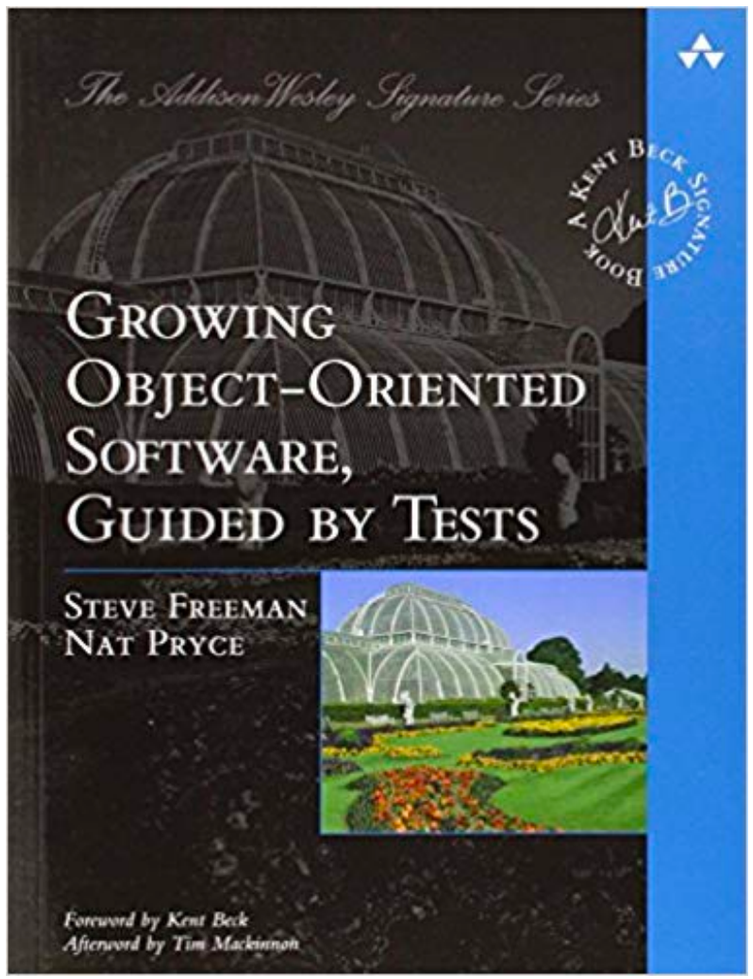
http://codingdojo.org/kata/Potter/

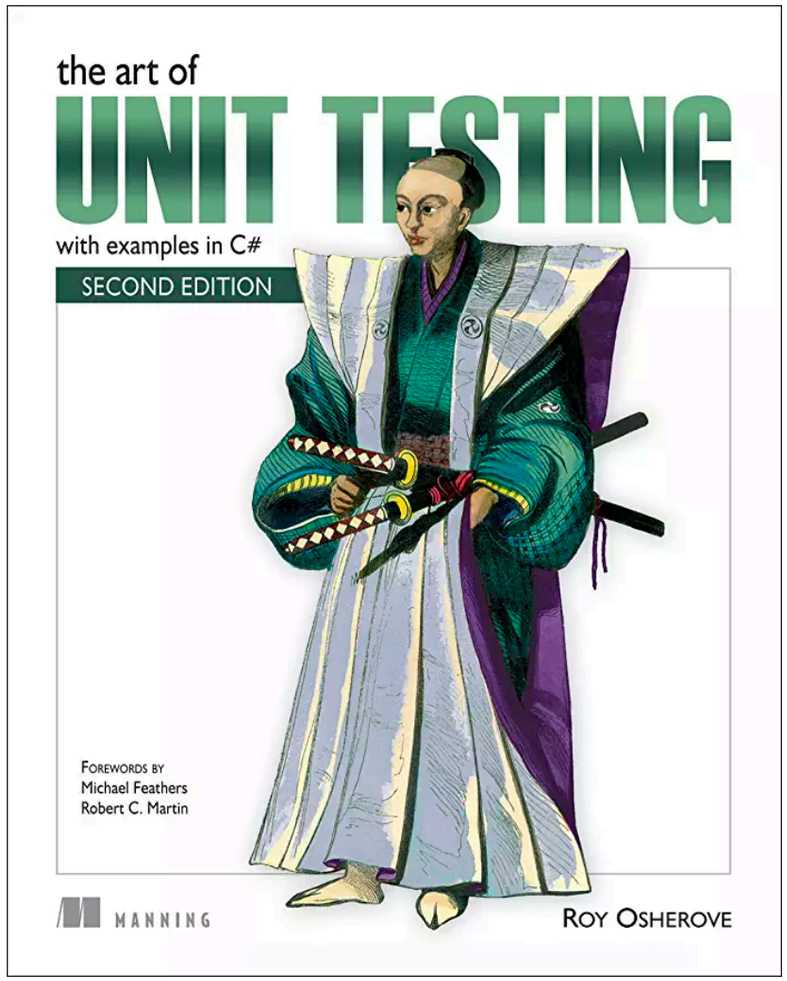
Don’t forgot Refactoring

Code Smell

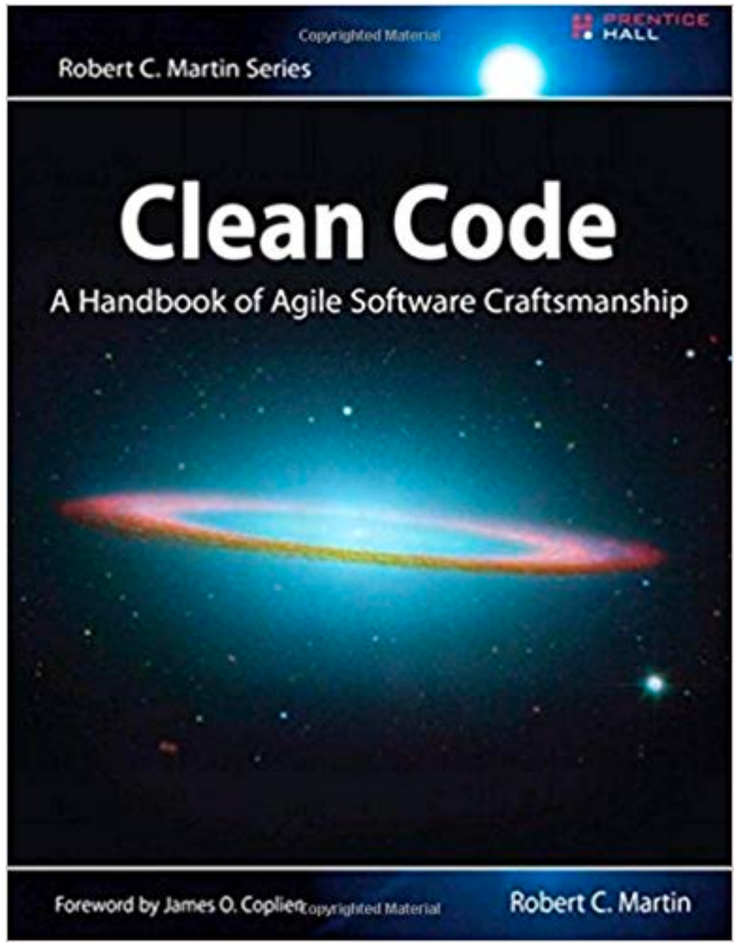
Books

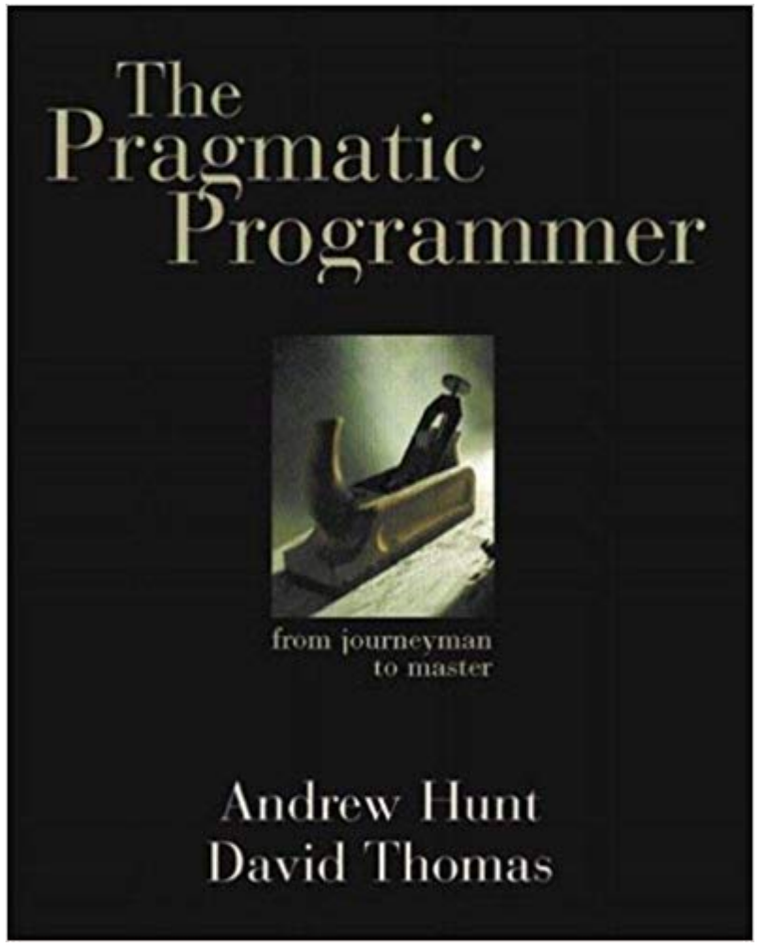


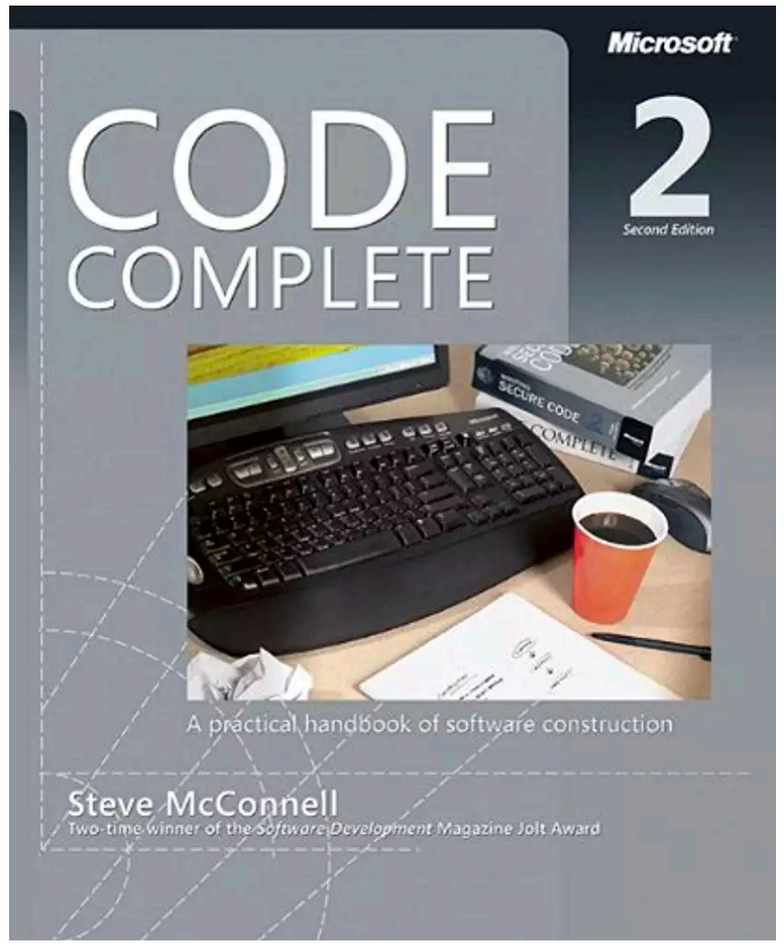




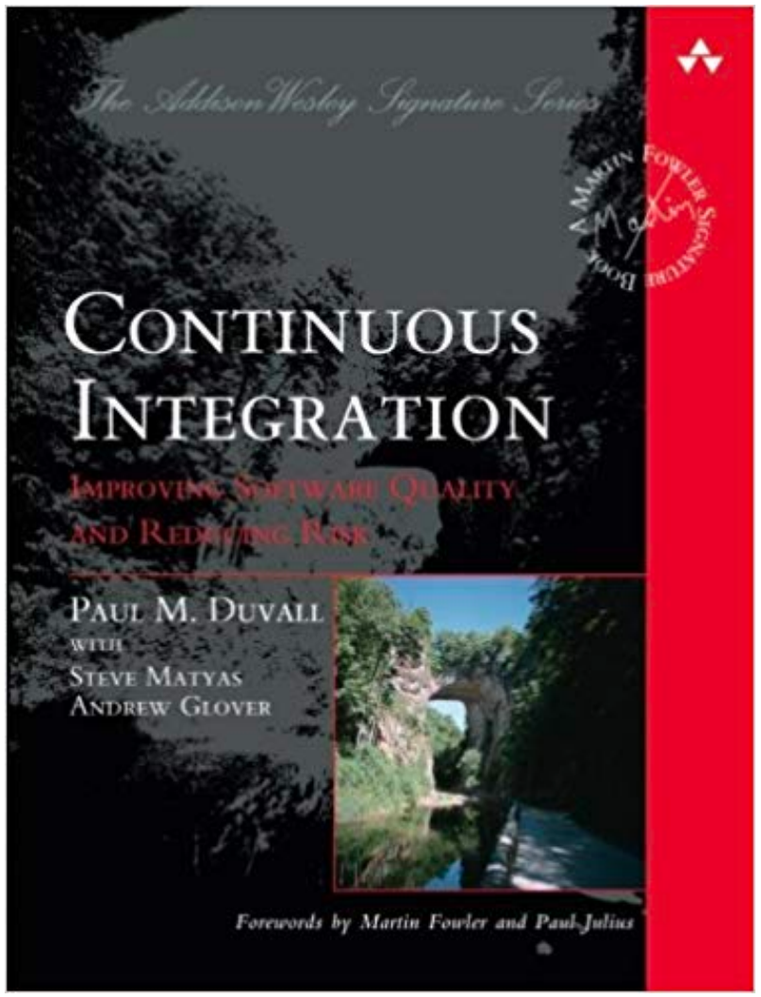
Books

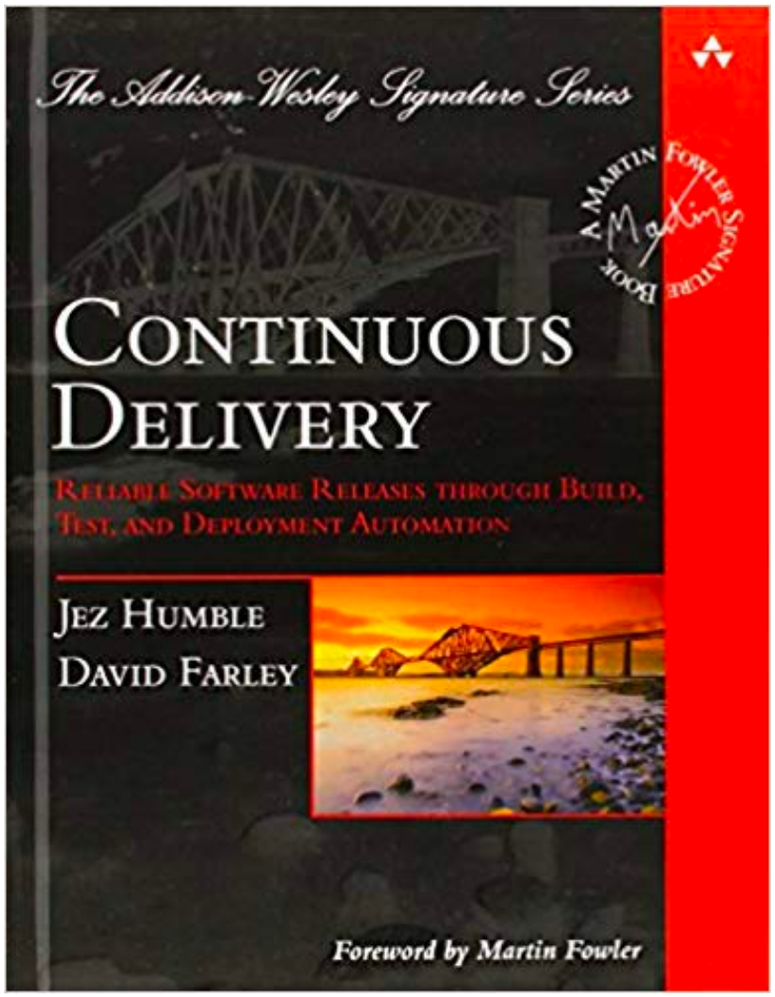






Books





SPECIFICATION BY EXAMPLE

CONTINUOUS INTEGRATION

CONTINUOUS DELIVERY

TEST AUTOMATION

ACCEPTANCE TESTING

ARCHITECTURE & DESIGN

CLEAN CODE

THINKING ABOUT TESTING

TECHNICAL EXCELLENCE

TEST-DRIVEN DEVELOPMENT

UNIT TESTING

https://less.works/less/technical-excellence/index.html

SPECIFICATION BY EXAMPLE

CONTINUOUS INTEGRATION

CONTINUOUS DELIVERY

TEST AUTOMATION

TECHNICAL EXCELLENCE

ARCHITECTURE & DESIGN

ACCEPTANCE TESTING

CODE

CLEAN CODE

THINKING ABOUT TESTING

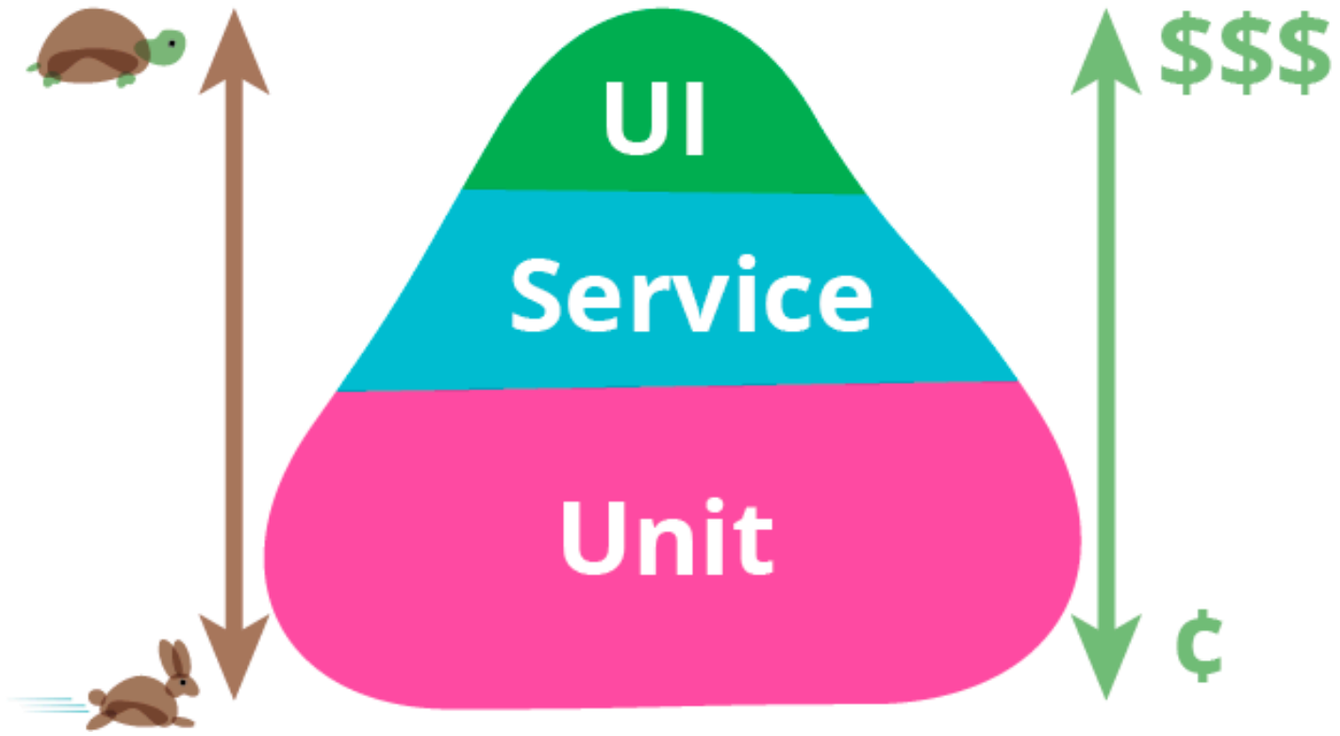
TEST-DRIVEN DEVELOPMENT

UNIT TESTING

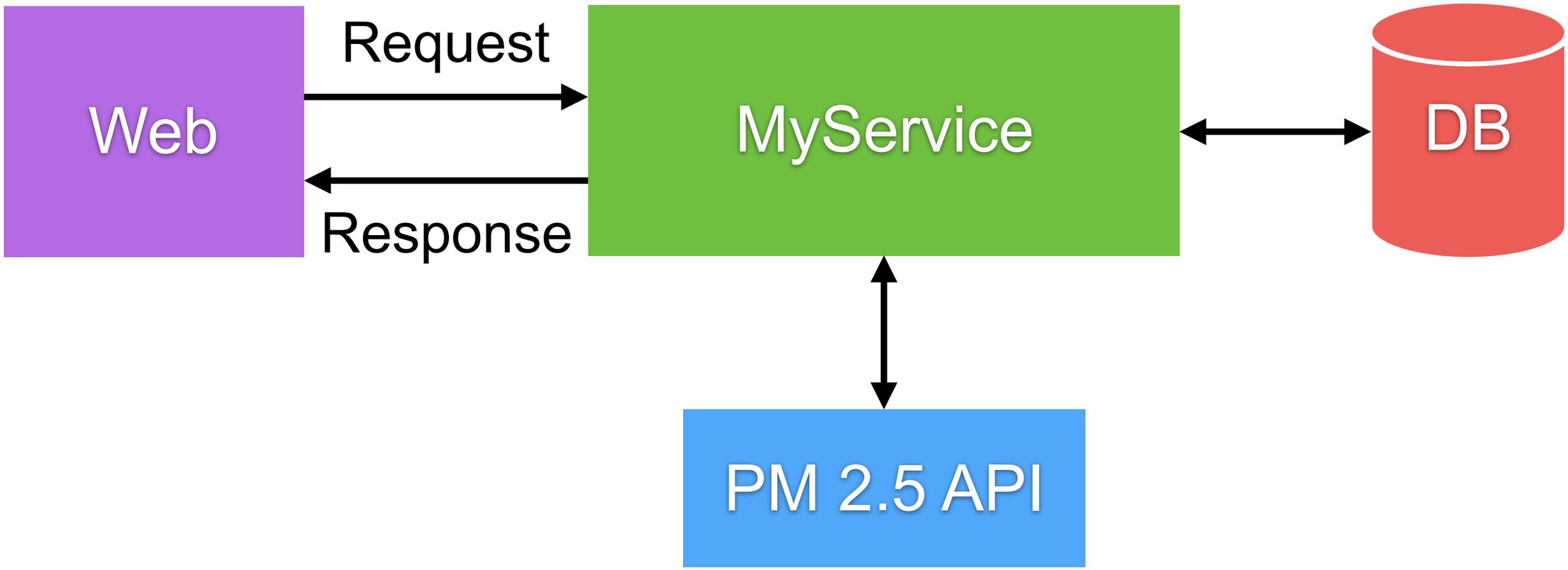
https://less.works/less/technical-excellence/index.html

Workshop with Testing

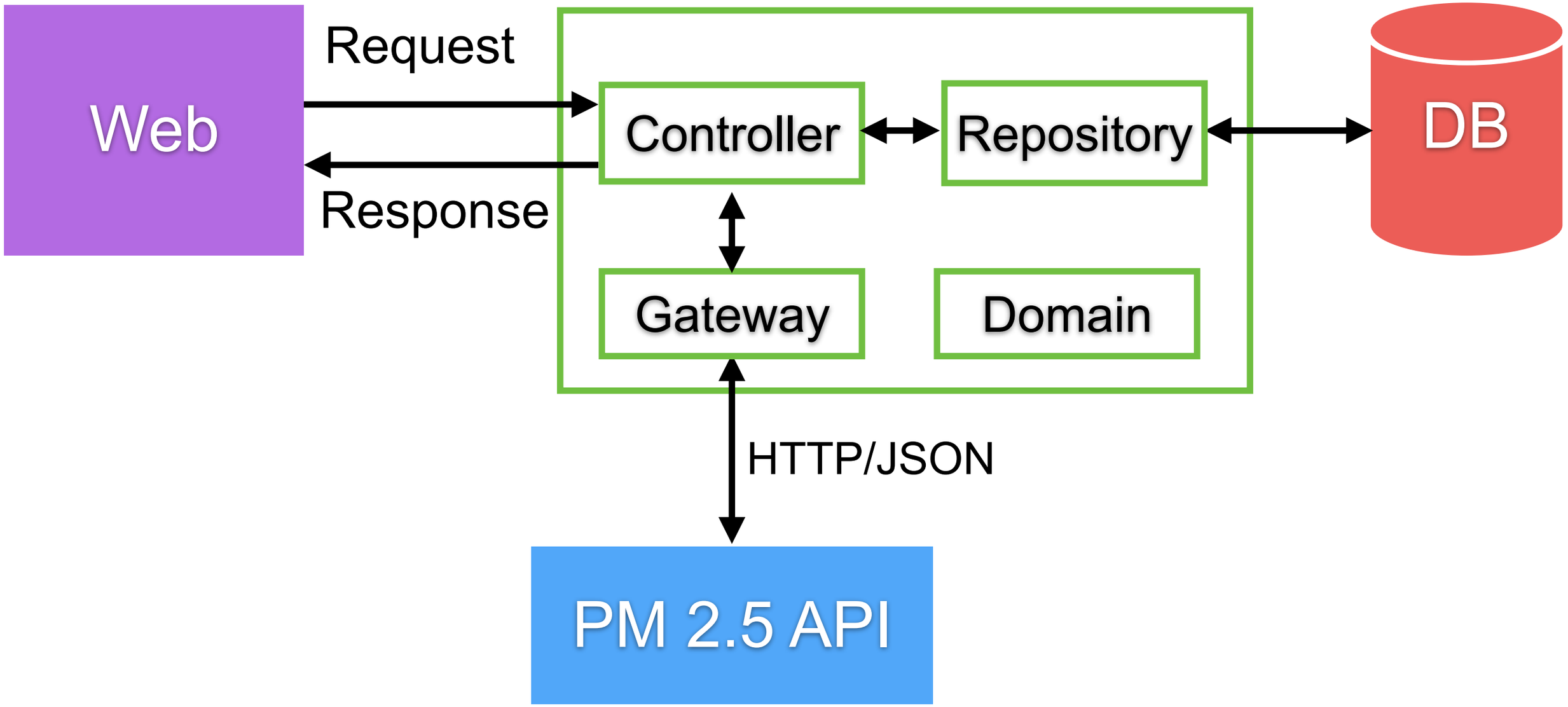
Test Pyramid



Workshop

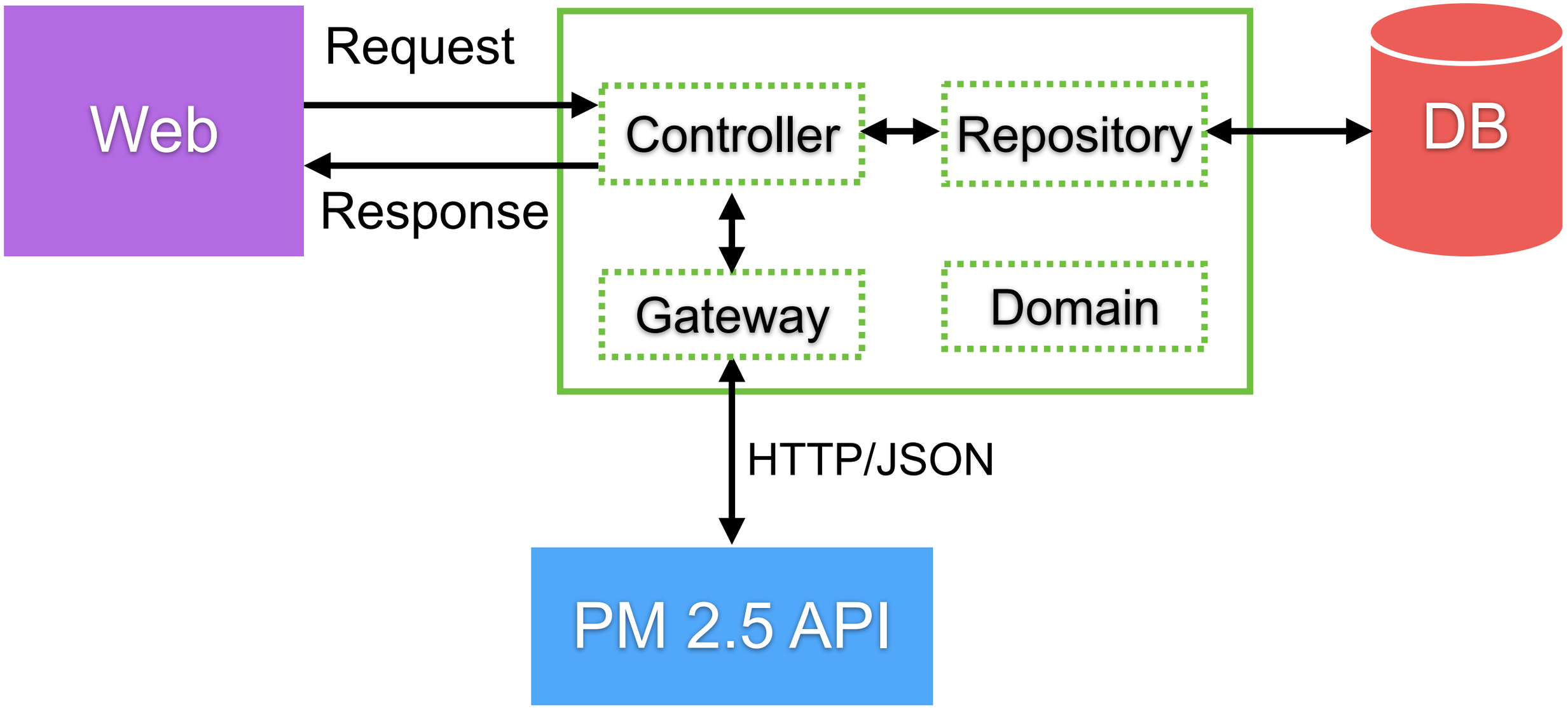


Structure of service



Unit testing

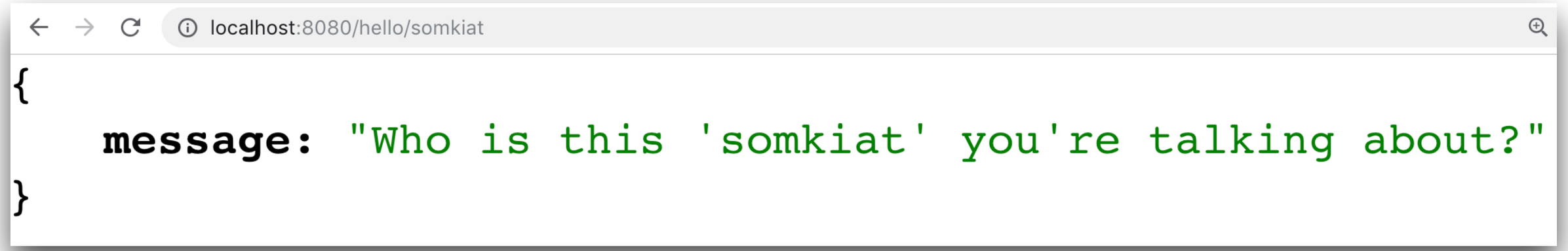
Unit testing



Controller testing ?

$mvnw spring-boot:run

http://localhost:8080/hello/somkiat

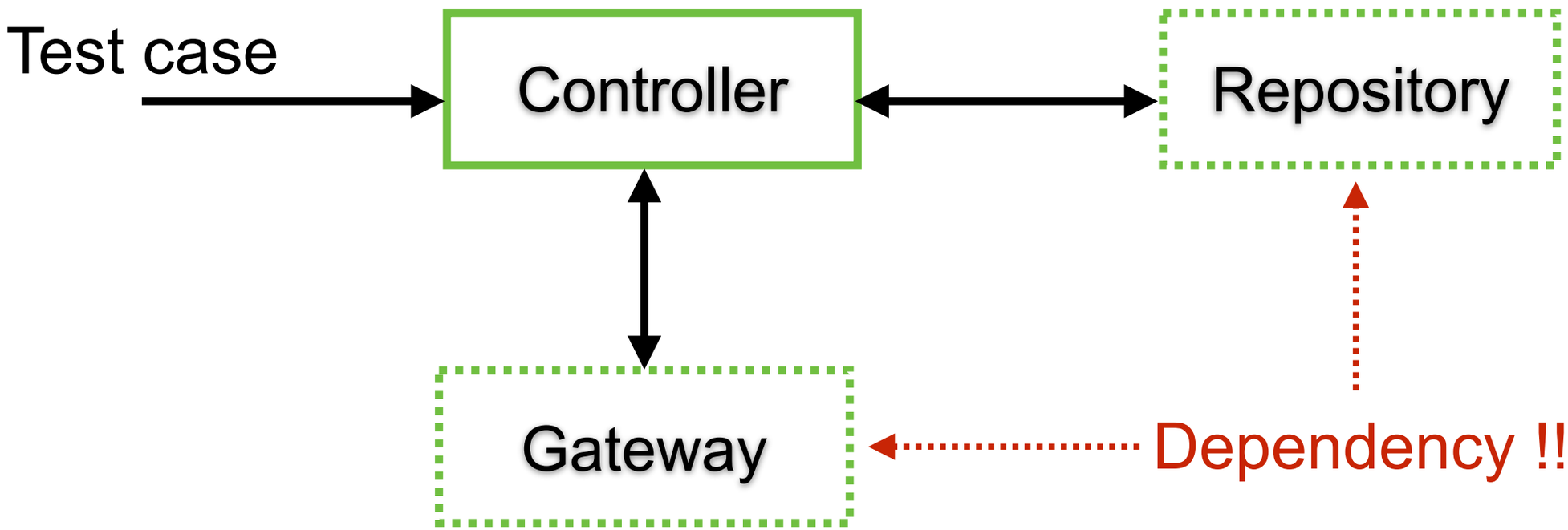


How to test Controller ?

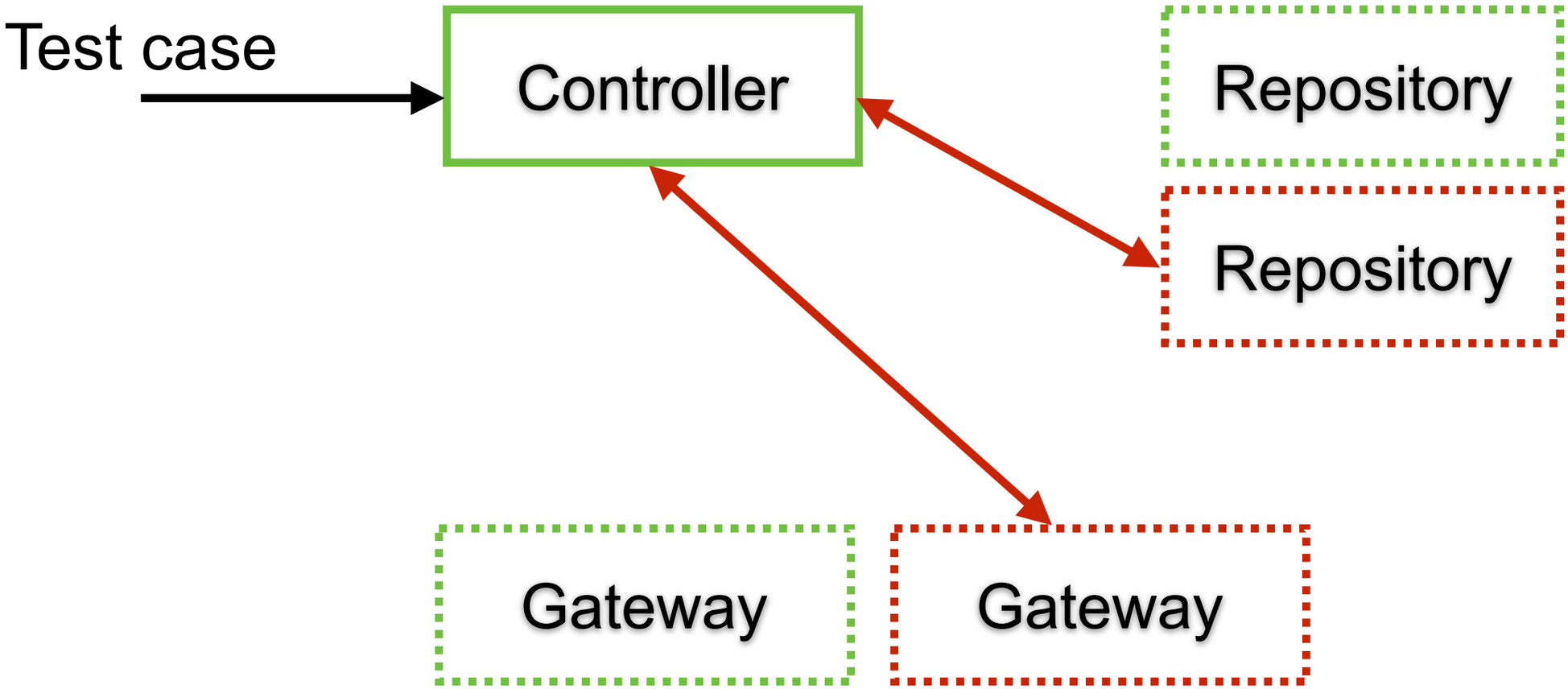
File /src/main/java/DemoController.java

```
@GetMapping("/hello/{lastName}")
public HelloResponse hello(@PathVariable String lastName) {
    Optional<Person> foundPerson
                = personRepository.findByLastName(lastName);

    return foundPerson
            .map(person -> new HelloResponse("Success"))
            .orElse(new HelloResponse("Failure"));
}
```

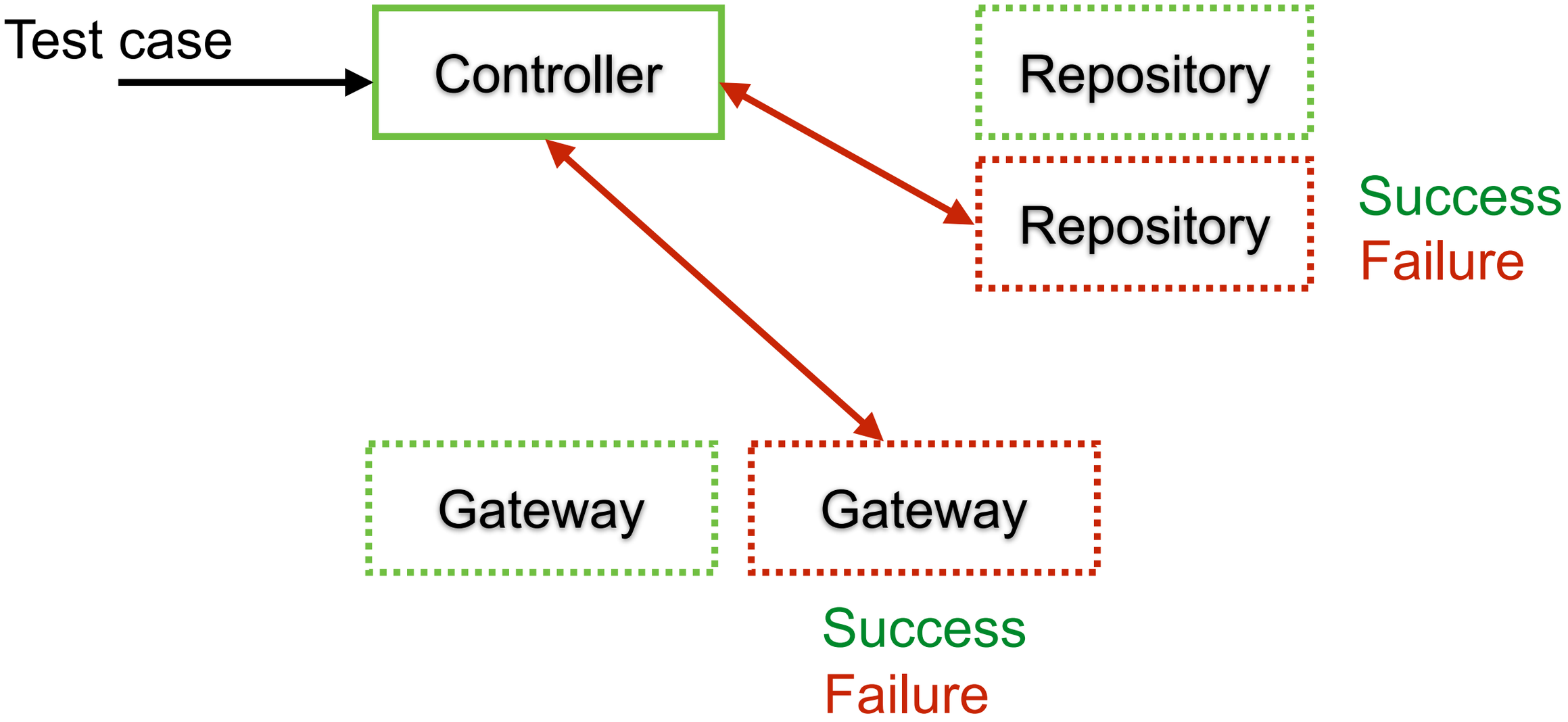
How to test Controller ?



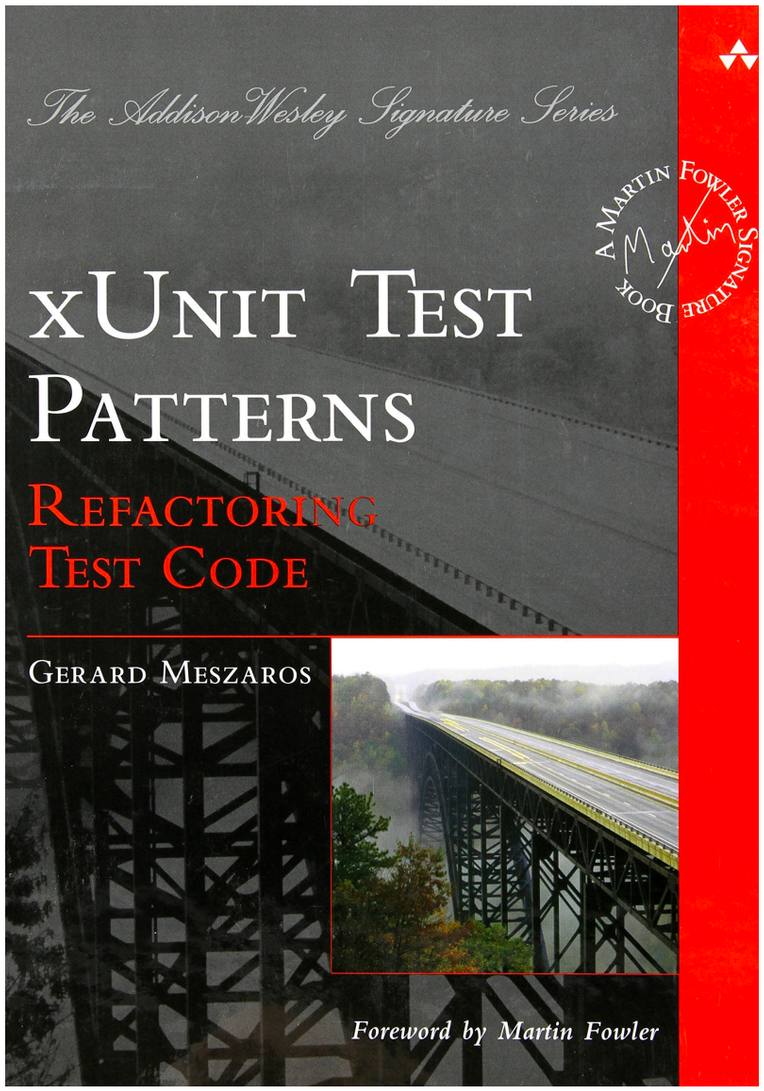
Working with Test Double



Working with Test Double

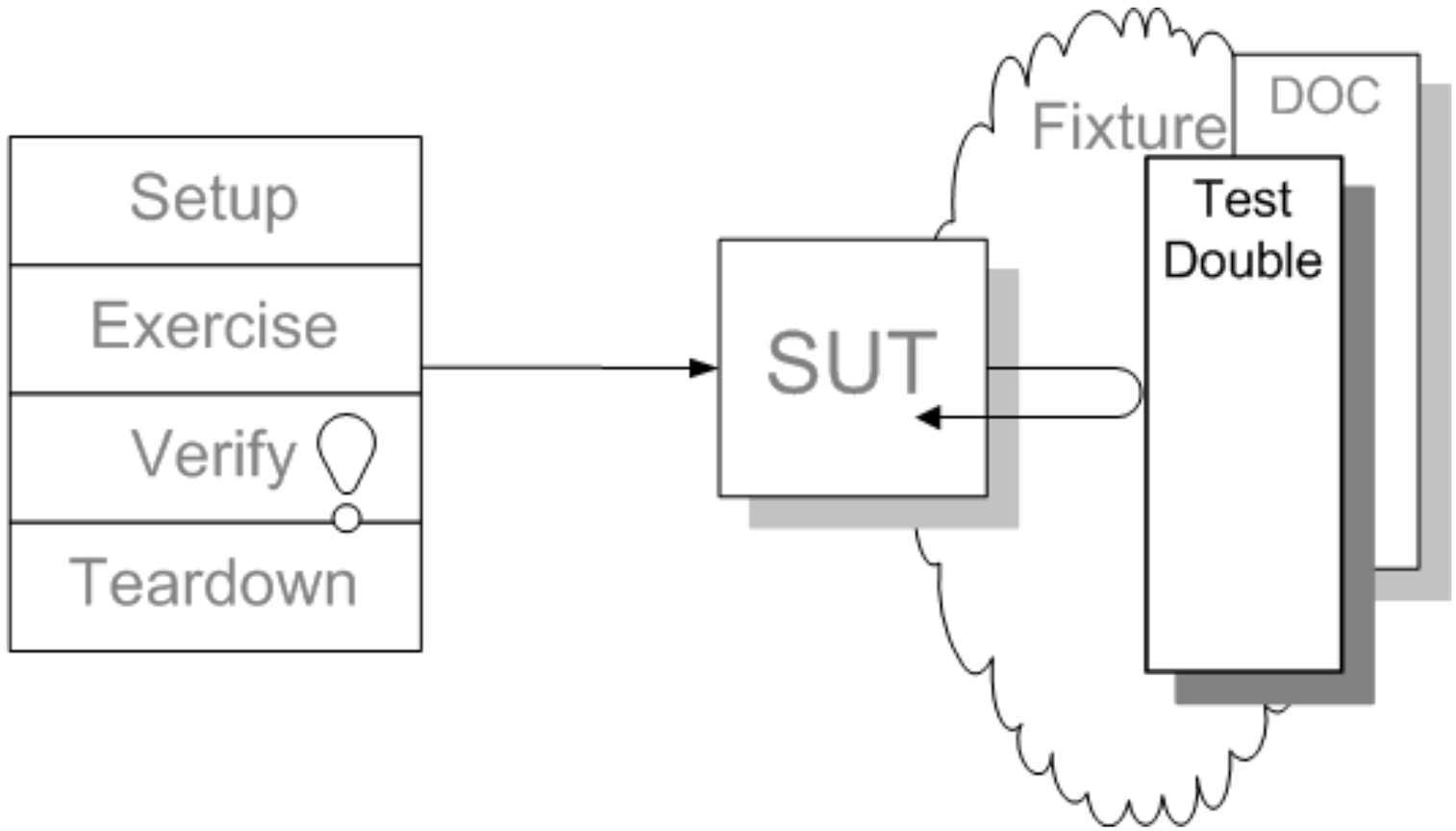


Test Double ?



http://xunitpatterns.com

Test Double ?



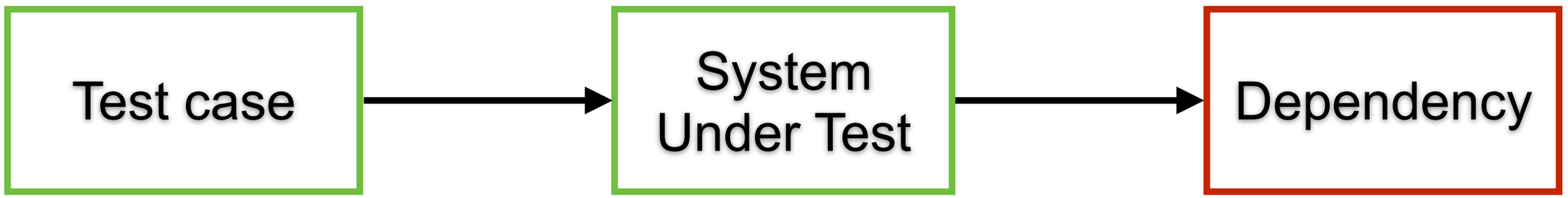
http://xunitpatterns.com/Test%20Double.html

Test Double ?

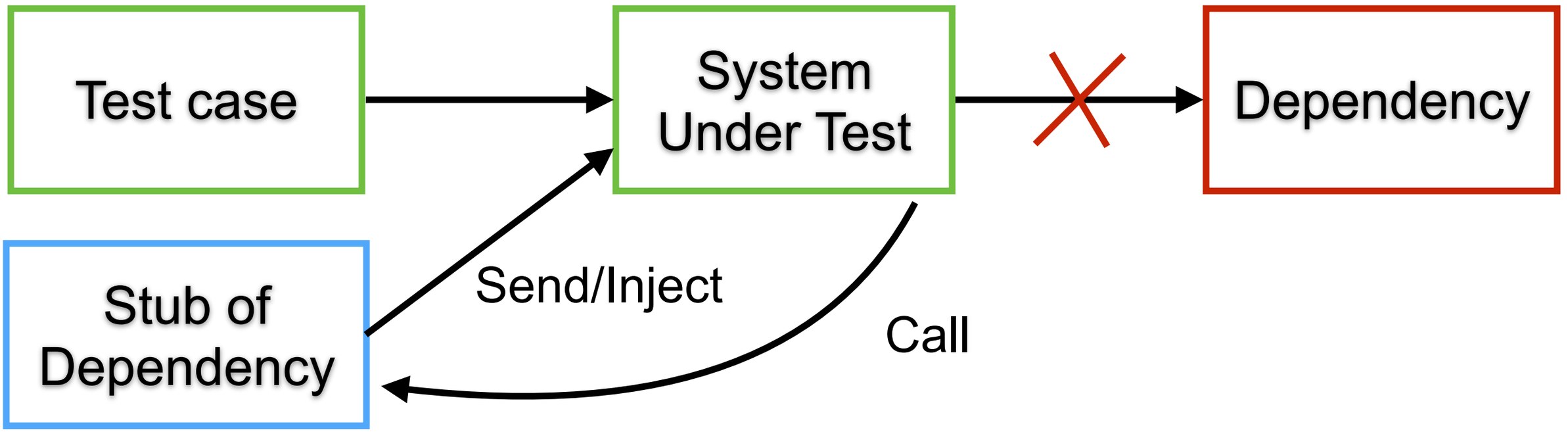
Dummy object	Stub	Spy	Mock object	Fake object

http://xunitpatterns.com/Test%20Double.html

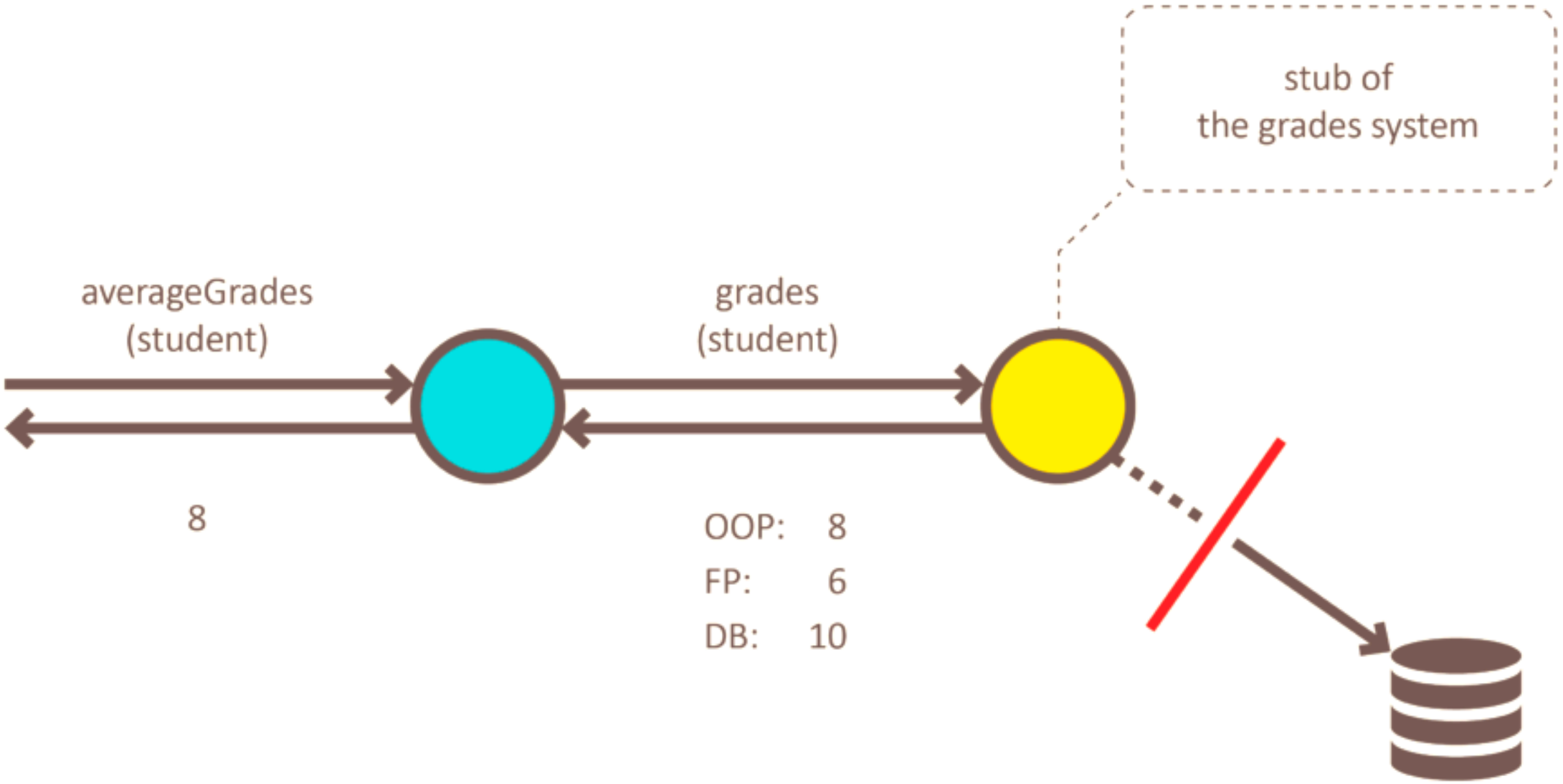
Stub



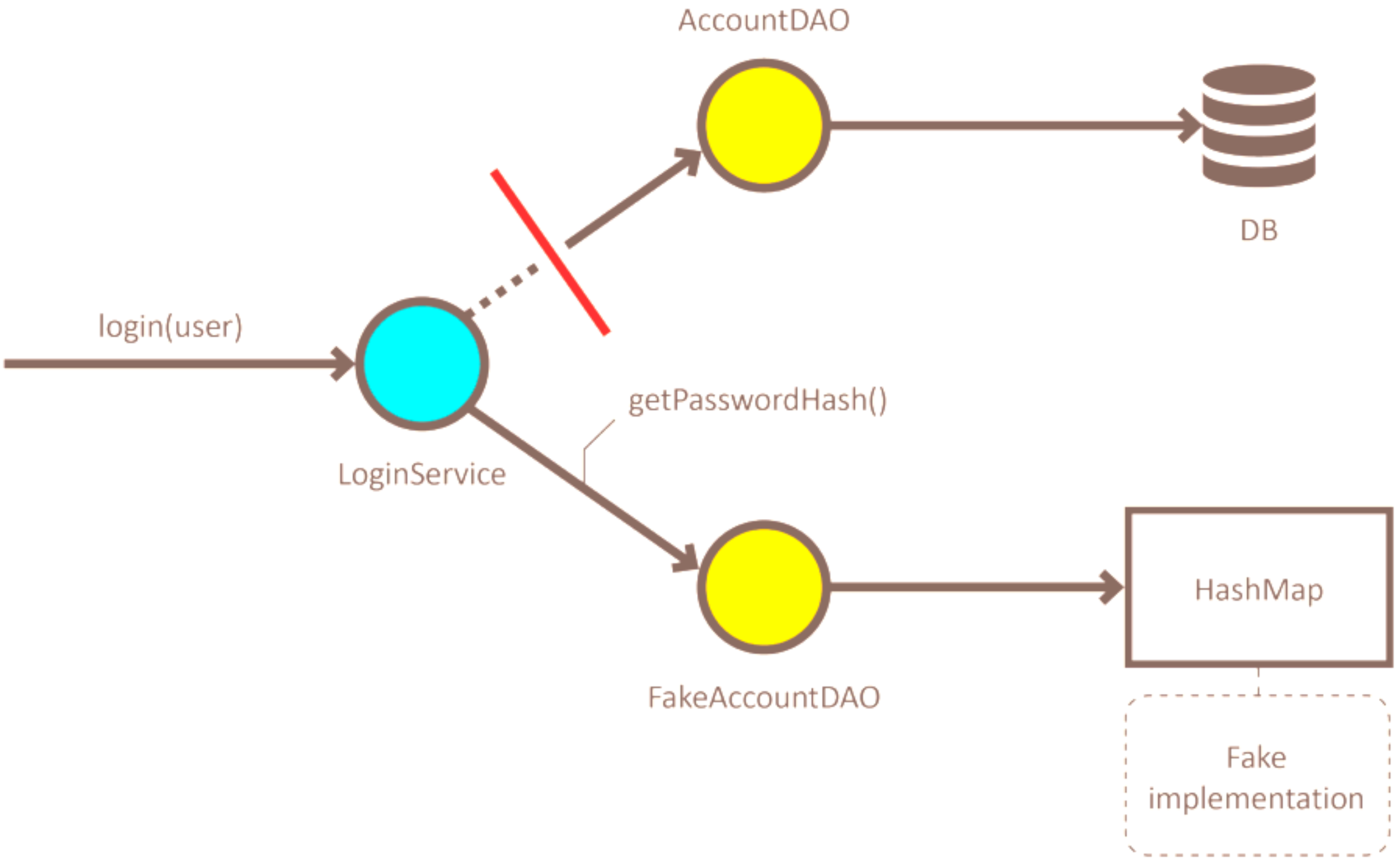
Create and Send Stub



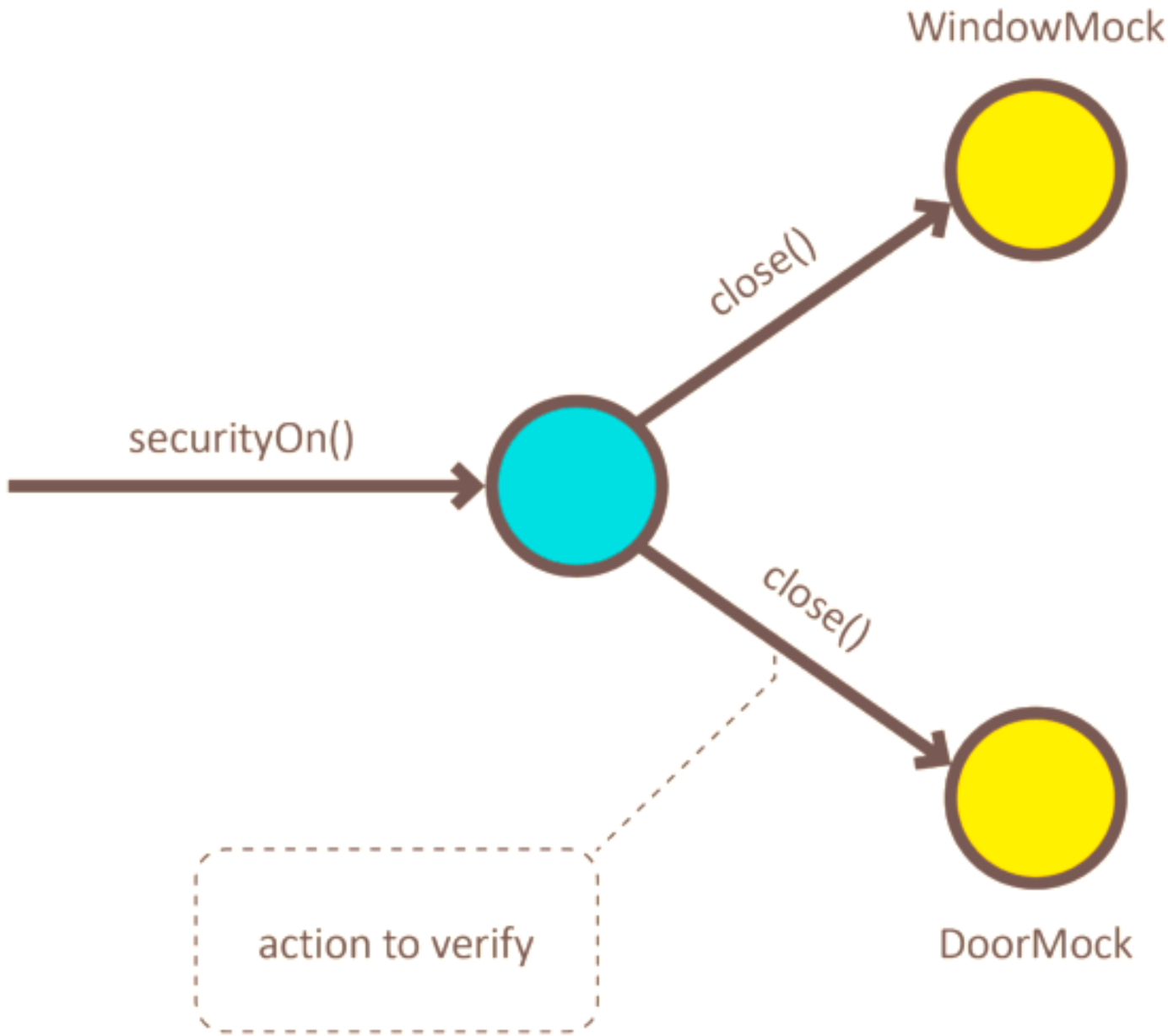
Stub



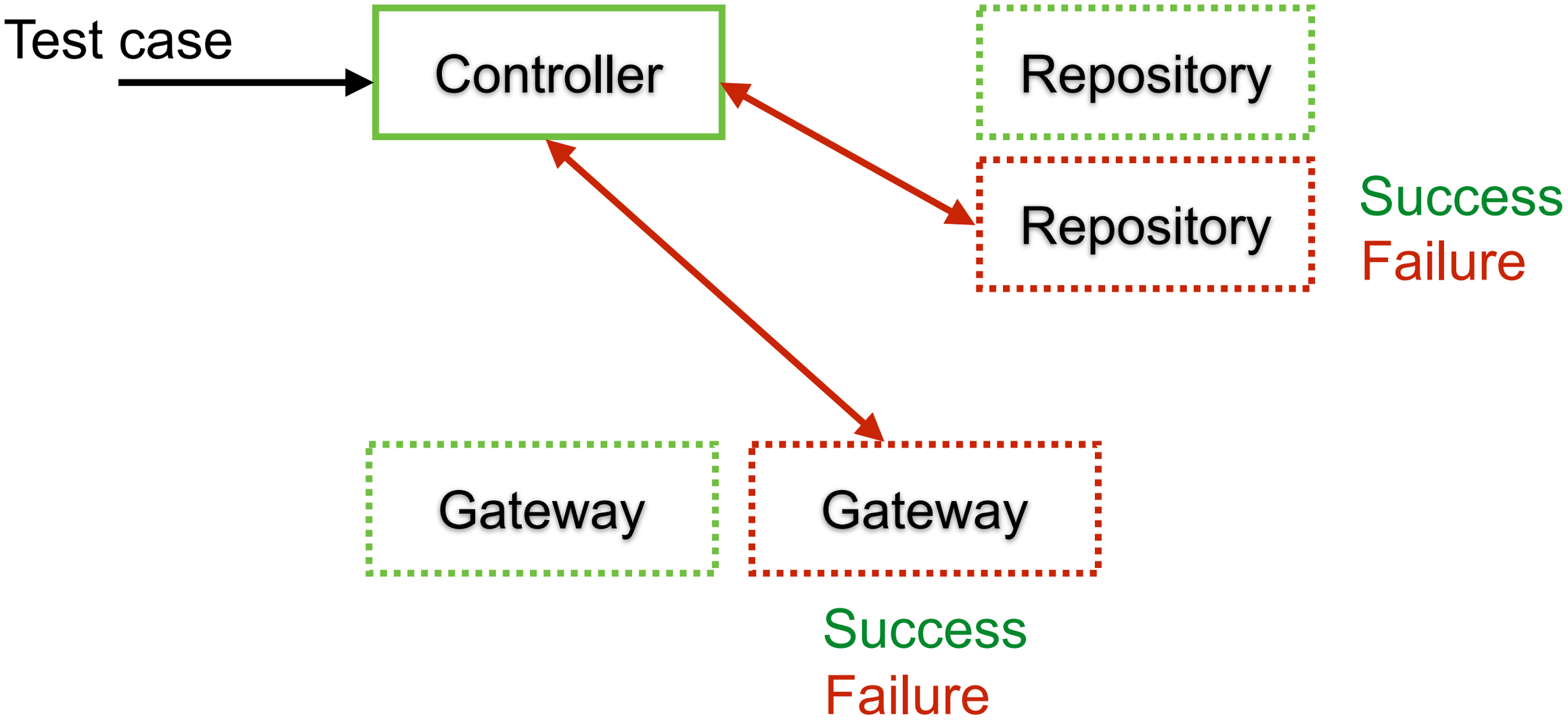
Fake



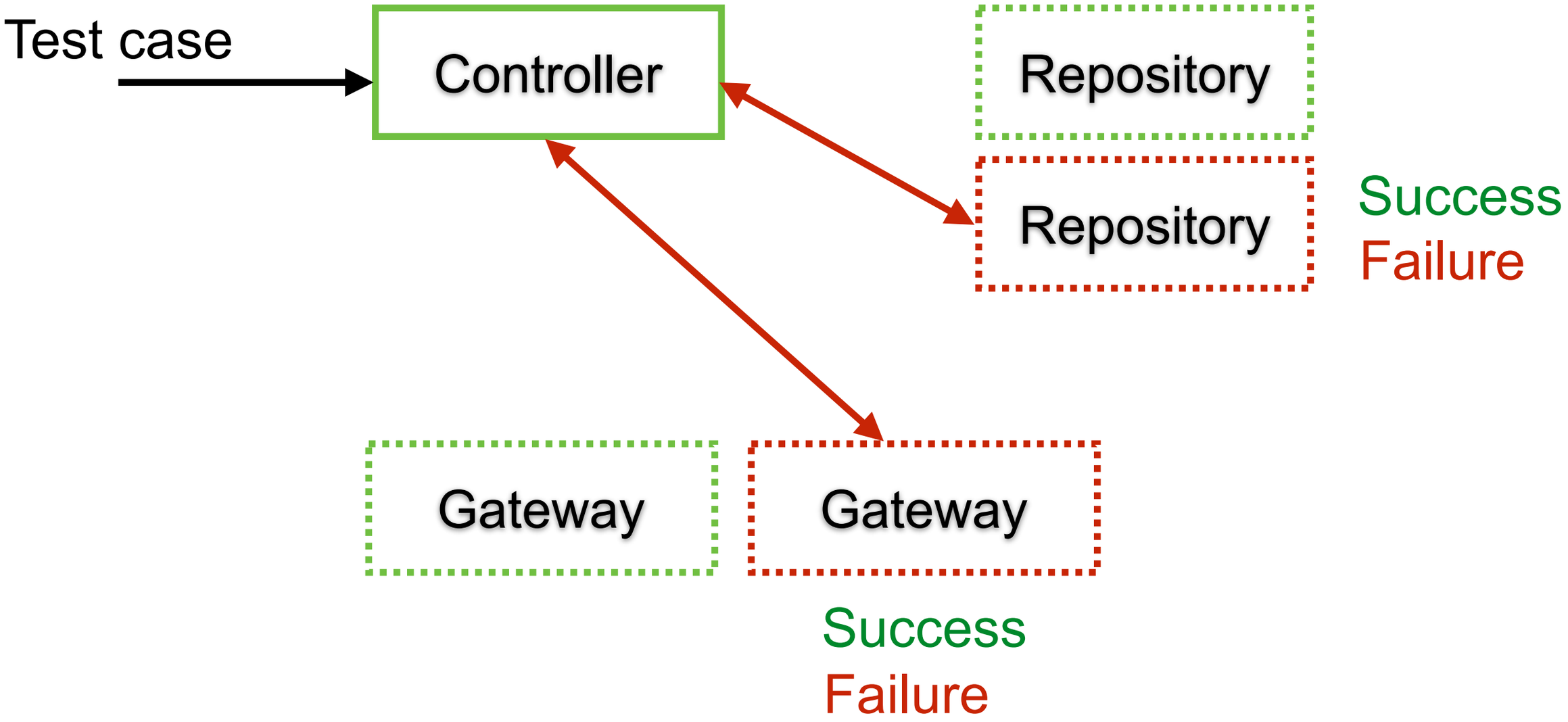
Mock



Working with Test Double ?



Stub



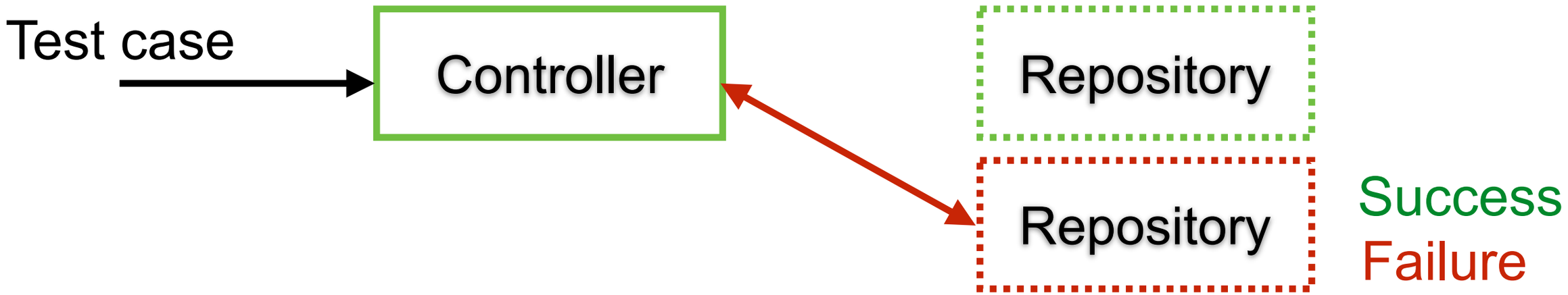
How to create Stub object ?

Manual

Library

https://site.mockito.org/

Working with Repository



Unit test for Controller #1

Prepare test environment with **Mockito**

```
public class DemoControllerUnitTest {

    private DemoController subject;

    @Mock
    private PersonRepository personRepository;

    @Mock
    private PM25Gateway pm25Gateway;

    @Before
    public void setUp() throws Exception {
        initMocks(this);
        subject
            = new DemoController(personRepository, pm25Gateway);
    }
```

JUnit Life Cycle ?

```
public class DemoControllerUnitTest {

    private DemoController subject;

    @Mock
    private PersonRepository personRepository;

    @Mock
    private PM25Gateway pm25Gateway;

    @Before
    public void setUp() throws Exception {
        initMocks(this);
        subject
            = new DemoController(personRepository, pm25Gateway);
    }
```

JUnit Life Cycle ?

@BeforeClass

For each @Test

Instantiate test class

@Before

Invoke the test

@After

@AfterClass

Unit test for Controller #2

Create Stub object for success case

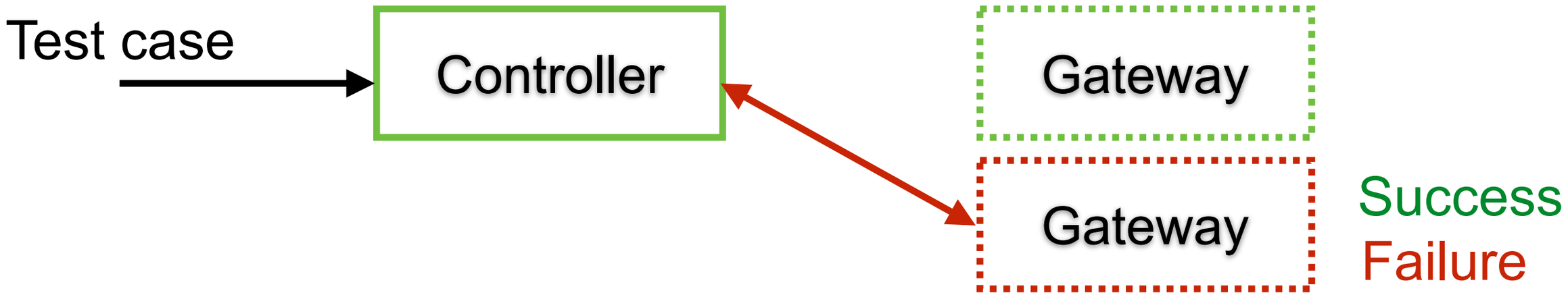
```
@Test
public void shouldReturnFullNameOfAPerson() throws Exception {
    // Create stub
    Person somkiat = new Person("Somkiat", "Pui");
    given(personRepository.findByLastName(“Pui"))
            .willReturn(Optional.of(somkiat));

    // Act
    HelloResponse greeting = subject.hello("Pui");

    // Assert
    assertThat(greeting.getMessage(), is("Hello Somkiat Pui!”));

}
```

Unit test for Controller #3

Create Stub object for failure case

```
@Test
public void shouldTellIfPersonIsUnknown() throws Exception {
    // Create stub
    given(personRepository.findByLastName(anyString()))
            .willReturn(Optional.empty());

    // Act
    HelloResponse greeting = subject.hello("Pui");

    // Assert
    assertThat(greeting.getMessage(),
        is("Who is this 'Pui' you're talking about?"));

}
```

Working with Gateway



Unit test for Controller

Create Stub object for success case

```
@Test
public void shouldReturnPM25Result() throws Exception {

    // Create stub
    PM25Response pm25Response = new PM25Response(100);
    given(pm25Gateway.fetchData())
            .willReturn(Optional.of(pm25Response));

    // Act
    String pm25 = subject.getDataPM25();

    // Assert
    assertThat(pm25, is(“100"));

}
```

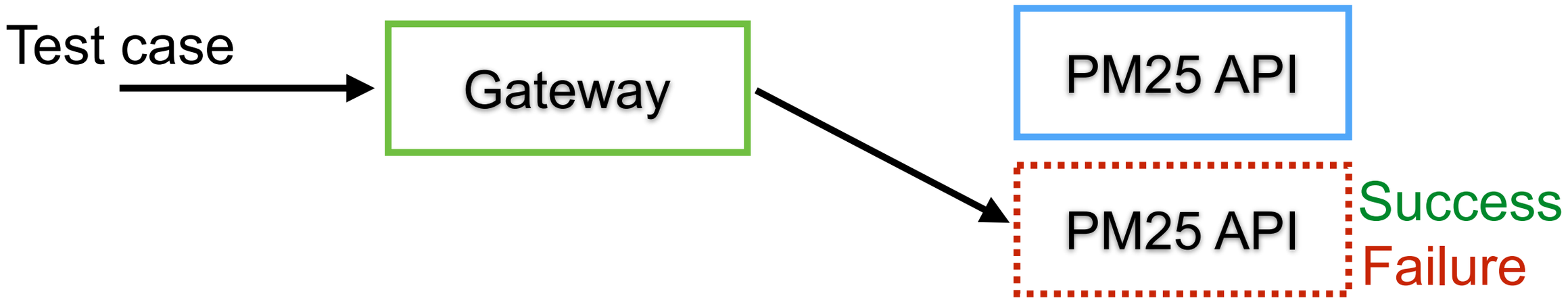
Unit test for Controller

Create Stub object for failure case

```
@Test
public void shouldReturnErrorMessageIfPM25IsUnavailable() throws
Exception {
    // Create stub
    given(pm25Gateway.fetchData())
            .willReturn(Optional.empty());

    // Act
    String pm25 = subject.getDataPM25();

    // Assert
    assertThat(pm25, is("Can not get PM2.5 from API"));
}
```

Unit test for Gateway ?

Unit test for Gateway



Unit test for Gateway #1

Initial environment for testing

```
@RunWith(SpringRunner.class)
public class PM25GatewayUnitTest {

    private PM25Gateway subject;

    @Mock
    private RestTemplate restTemplate;
```

```
    @Before
    public void setUp() throws Exception {
        initMocks(this);
        subject = new PM25Gateway(restTemplate, "http://localhost:8089");
    }
```

Unit test for Gateway #2

Create Stub object for success case

```
@Test
public void shouldCallPM25API() throws Exception {
    // Create Stub
    PM25Response expectedResponse = new PM25Response(1000);
    given(restTemplate.getForObject(
                    “http://localhost:8089/feed/", PM25Response.class))
            .willReturn(expectedResponse);

    // Act
    Optional<PM25Response> actualResponse = subject.fetchData();

    // Assert
    assertThat(actualResponse, is(Optional.of(expectedResponse)));
}
```

Unit test for Gateway #3

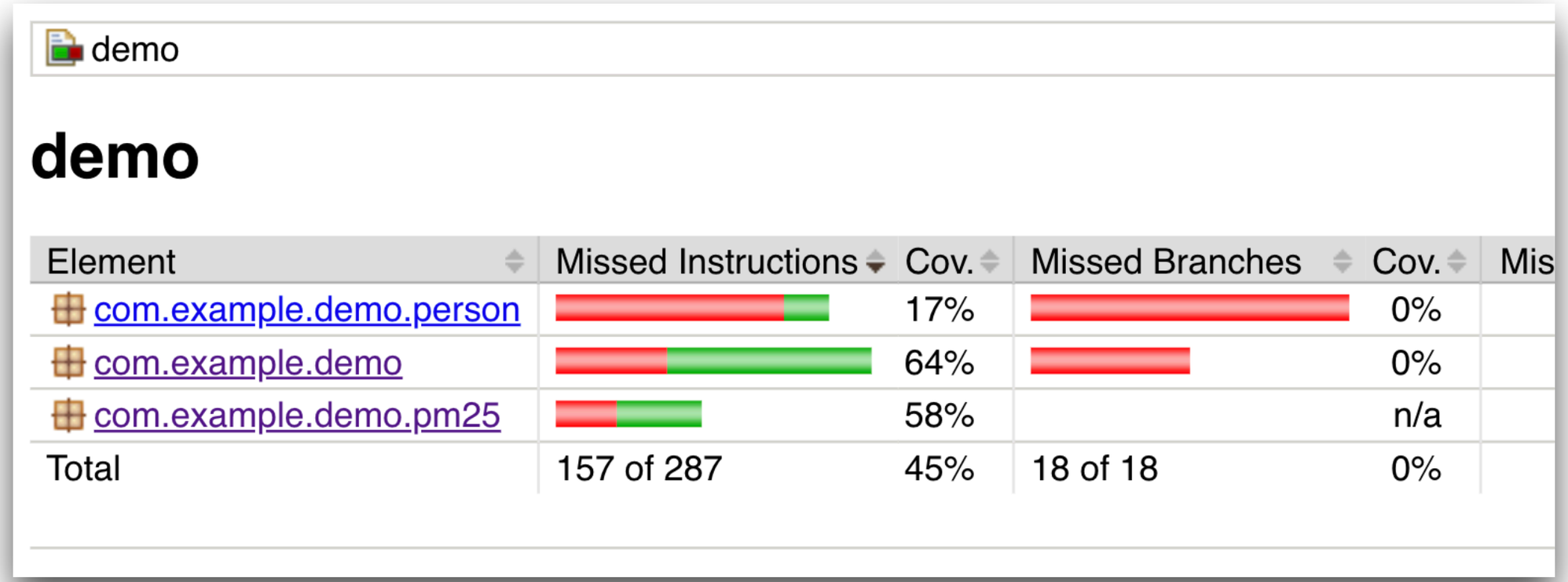
Create Stub object for failure case

```
@Test
public void shouldReturnEmptyOptionalIfPM25APIIsUnavailable() {
    // Create Stub
    given(restTemplate.getForObject(
            “http://localhost:8089/feed/thailand/", PM25Response.class))
            .willThrow(new RestClientException("something went wrong"));

    // Act
    Optional<PM25Response> actualResponse = subject.fetchData();

    // Assert
    assertThat(actualResponse, is(Optional.empty()));

}
```

Run all test + coverage

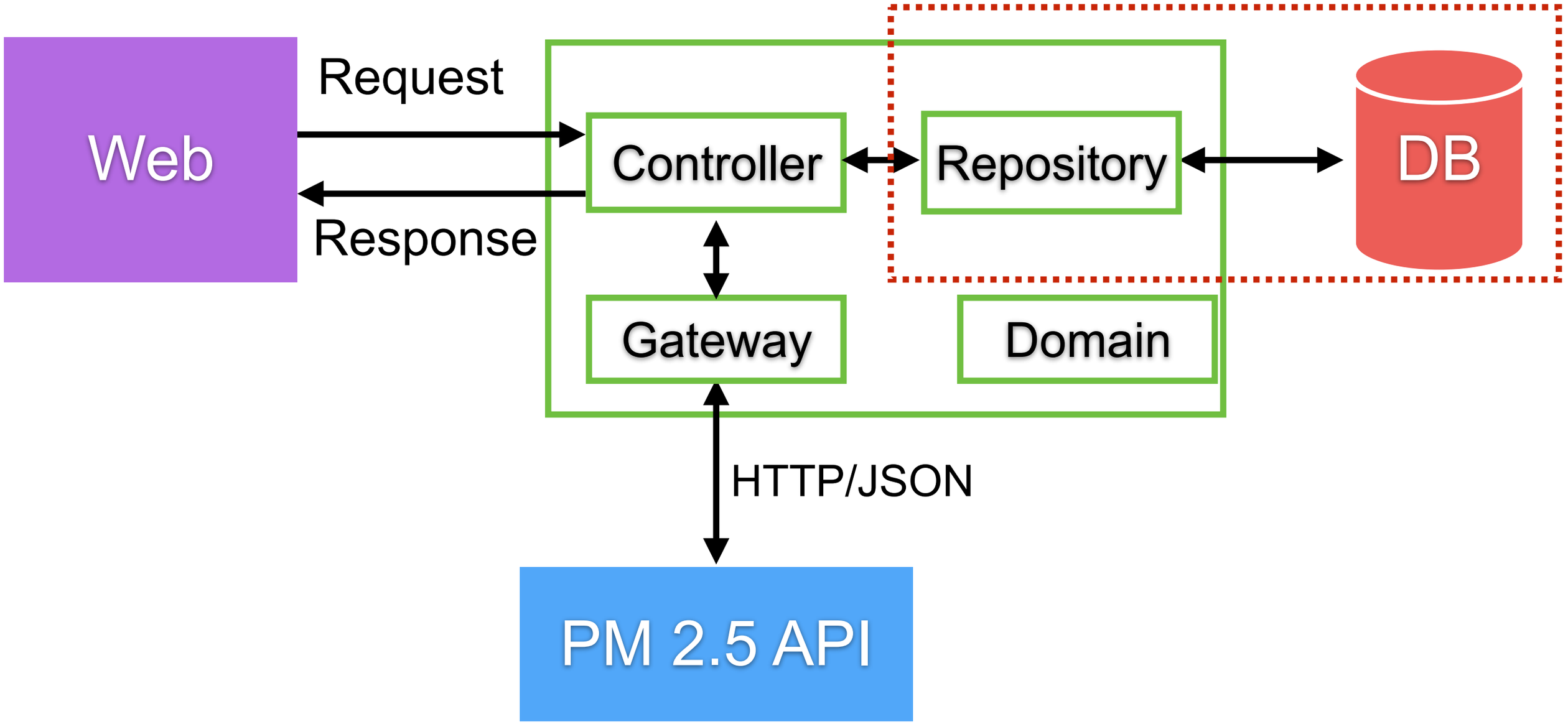
$mvnw clean package

Open file /target/site/jacoco/index.html

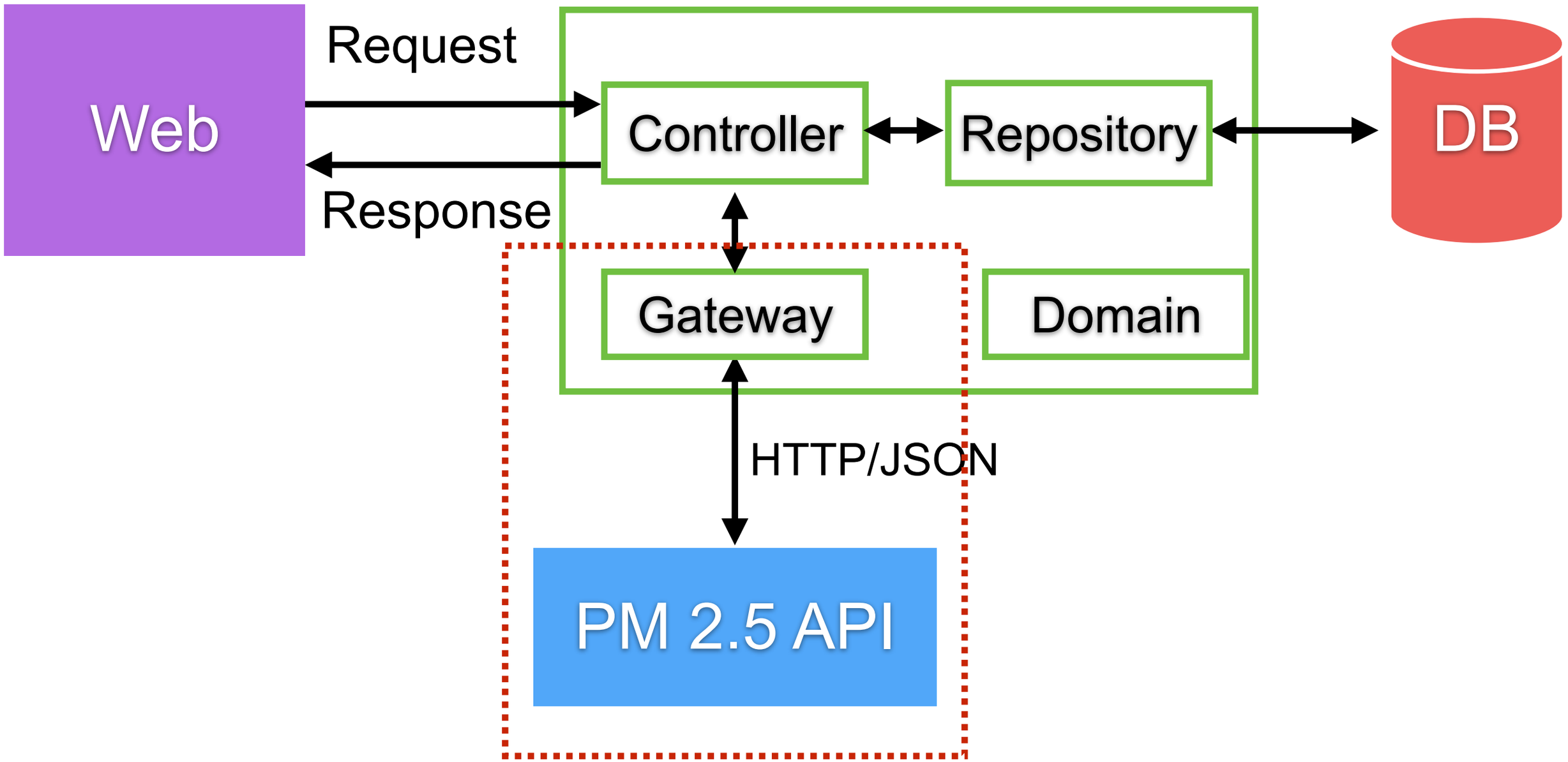


Integration testing

Integration testing

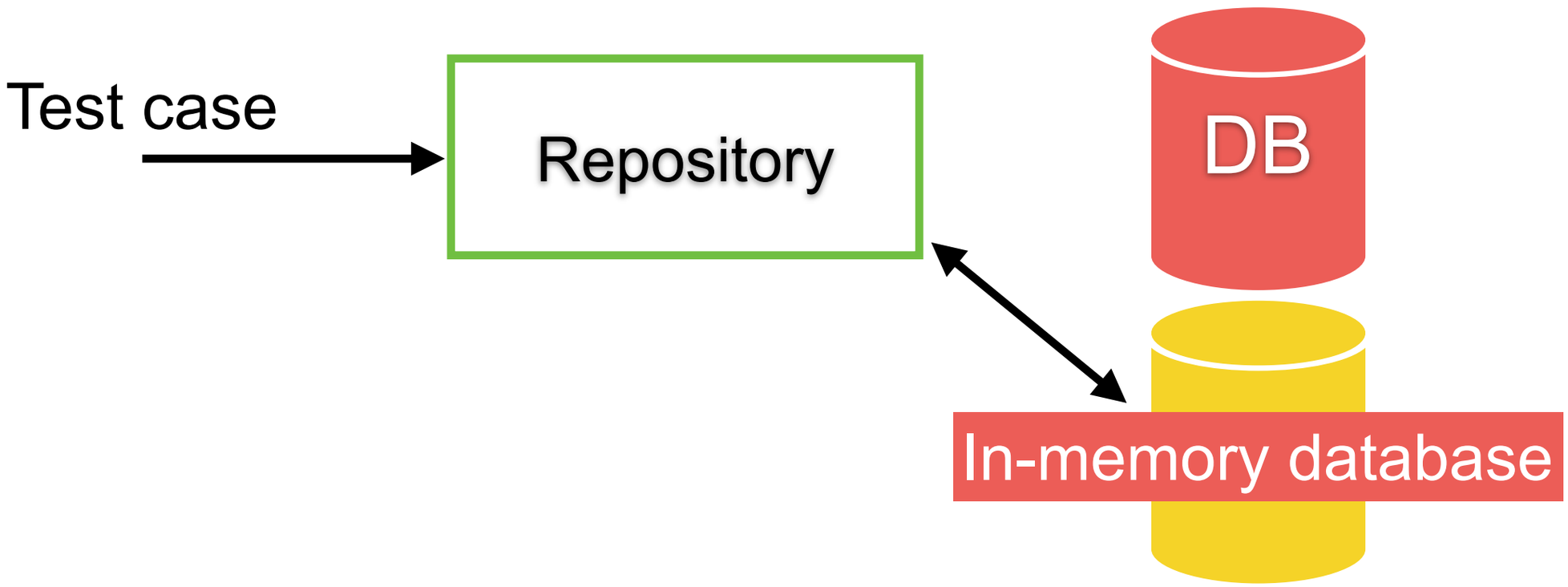


Integration testing



Integration testing with Database

1. Start database
2. Connect application to the database
3. Call function in code that read/write data to db
4. Check expected data has been written to DB



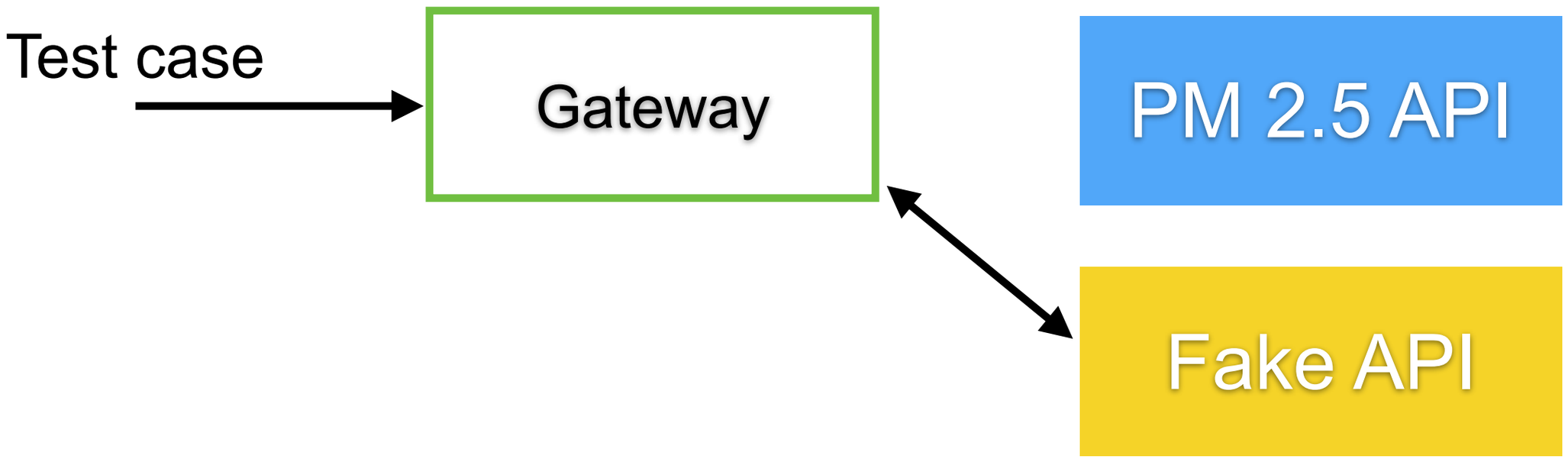
Integration test with database

File /src/test/java/PersonRepositoryIntegrationTest.java

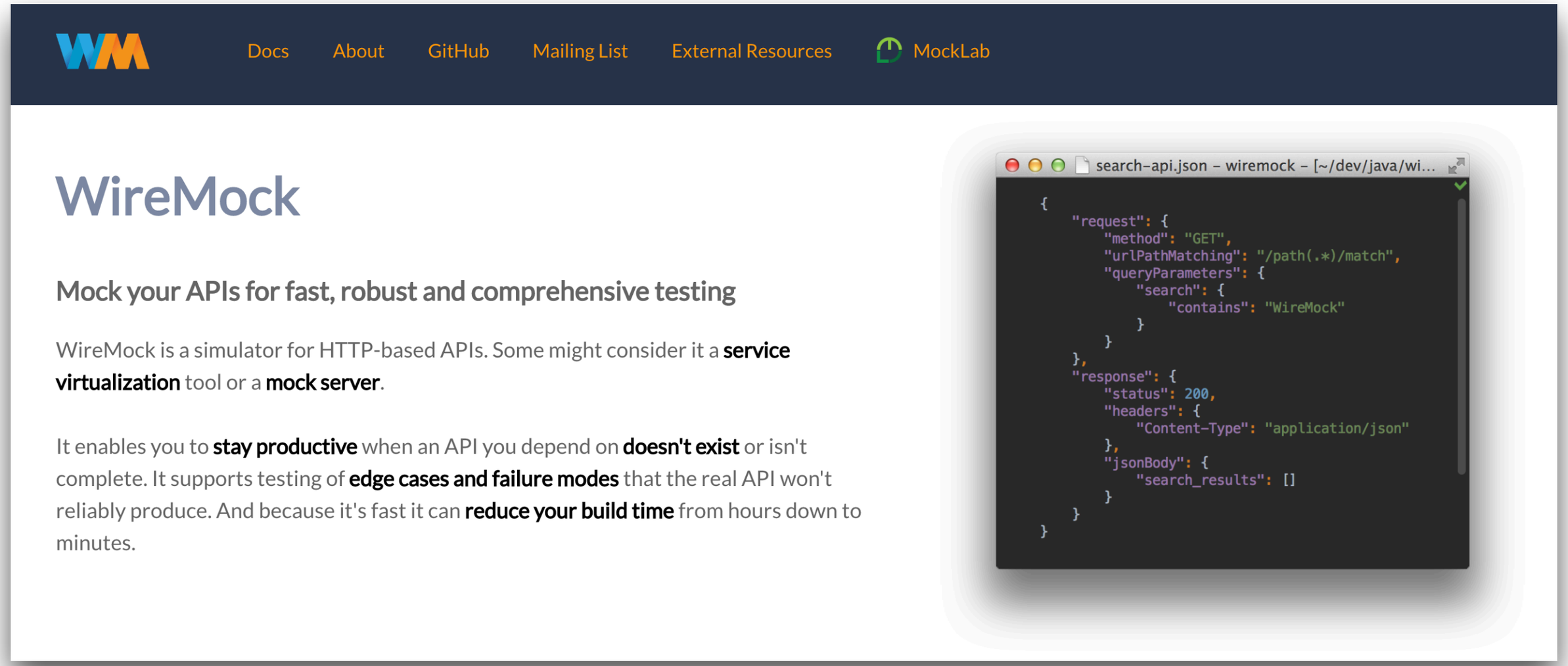
```
public class PersonRepositoryIntegrationTest {

    @Autowired
    private PersonRepository subject;

    @Test
    public void shouldSaveAndFetchPerson() throws Exception {
        // Insert data for testing
        Person somkiat = new Person("Somkiat", "Pui");
        subject.save(somkiat);

        // Act
        Optional<Person> maybePeter = subject.findByLastName("Pui");

        // Assert
        assertThat(maybePeter, is(Optional.of(somkiat)));
    }
}
```

Integration testing with API

1. Start application
2. Start API (test double)
3. Call function in code that read from API
4. Check that app can parse the response correctly



WireMock



http://wiremock.org/

Integration testing with API

File /src/test/java/PM25GatewayIntegrationTest.java

```
public class PM25GatewayIntegrationTest {

    @Autowired
    private PM25Gateway subject;

    @Rule
    public WireMockRule wireMockRule = new WireMockRule(8089);

    @Test
    public void shouldCallPM25APIWithSuccess() throws Exception {

    }

}
```

Integration testing with API

File /src/test/java/PM25GatewayIntegrationTest.java

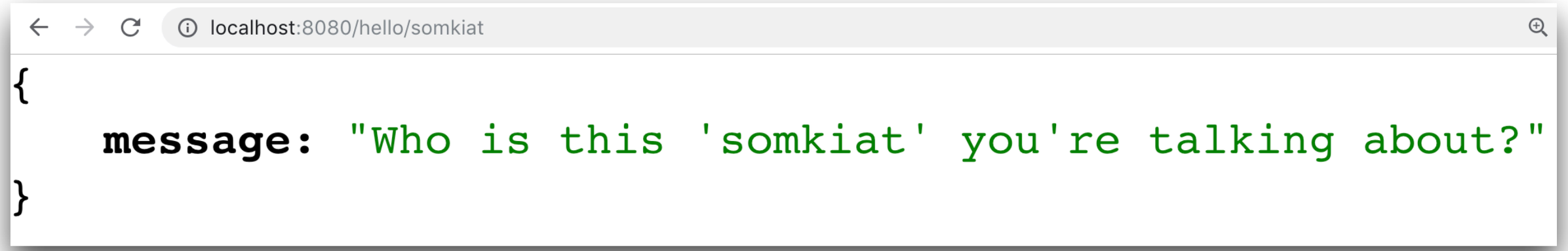
```
@Test
public void shouldCallPM25APIWithSuccess() throws Exception {
    // Create Fake API with WireMock
    wireMockRule.stubFor(get(urlPathEqualTo("/feed/thailand/pathum-thani/"))
            .withQueryParam("token", matching(“zzzzz"))
            .willReturn(aResponse()
                    .withBody(FileLoader.read("classpath:pm25Response.json"))
                    .withHeader(CONTENT_TYPE, MediaType.APPLICATION_JSON_VALUE)
                    .withStatus(200)));

    // Call API
    Optional<PM25Response> pm25Response = subject.fetchData();

    // Assert :: Check response from API
    Optional<PM25Response> expectedResponse = Optional.of(new PM25Response(1000));
    assertThat(pm25Response, is(expectedResponse));
}
```

UI testing

UI Testing

Open your browser

http://localhost:8080/hello/somkiat



End-to-End with Selenium

File /src/test/java/DemoControllerUIWithSeleniumTest.java

```
public class DemoControllerUIWithSeleniumTest {

    private WebDriver driver;

    @LocalServerPort
    private int port;

    @BeforeClass
    public static void setUpClass() throws Exception {
        WebDriverManager.chromedriver().version("73").setup();
    }

    @Before
    public void setUp() throws Exception {
        driver = new ChromeDriver();
    }
```

End-to-End with Selenium

File /src/test/java/DemoControllerUIWithSeleniumTest.java

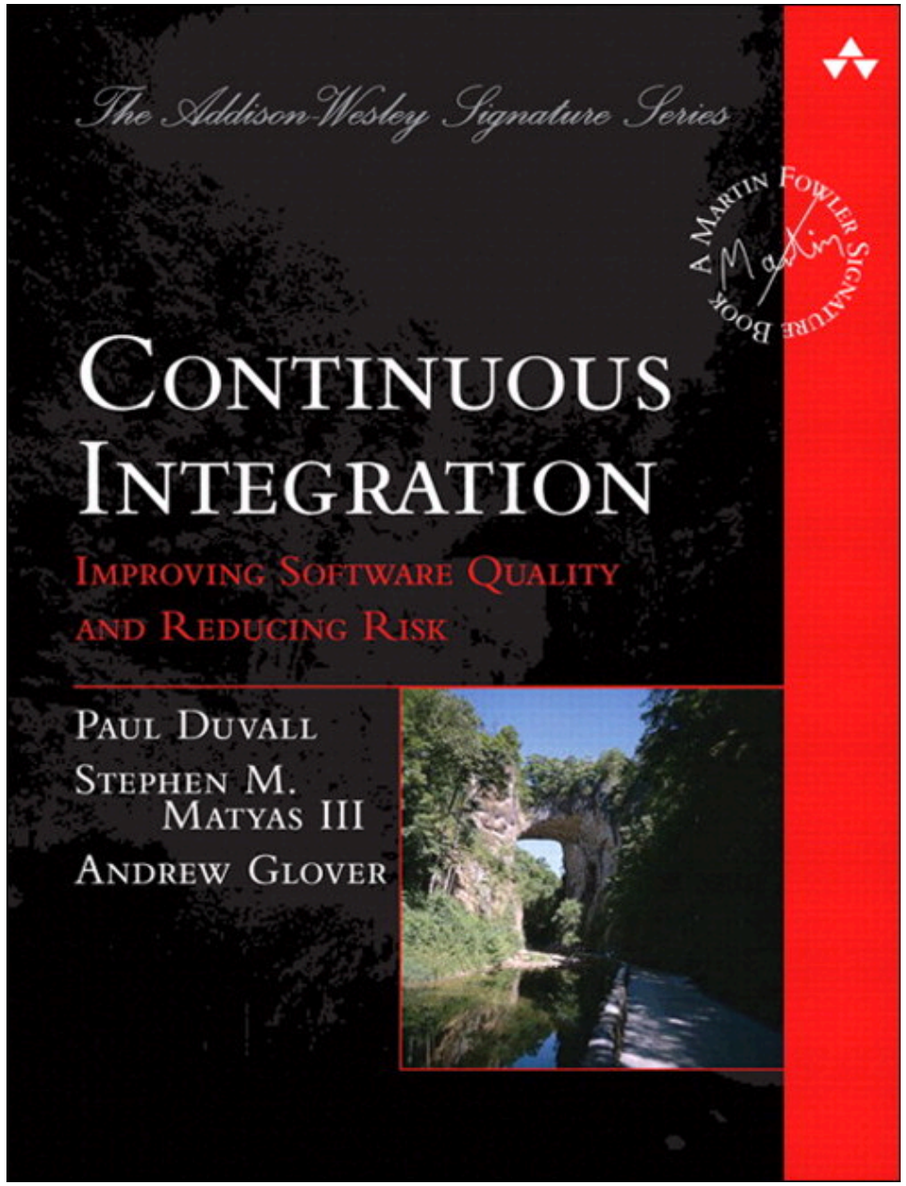
```
@Test
public void helloPageHasTextHelloWorld() {
    // Go to URL
    driver.navigate().to(
        String.format("http://localhost:%s/hello", port));

    // Find by tag name
    WebElement body = driver.findElement(By.tagName("body"));

    // Check data with expected result
    assertThat(body.getText(), containsString("Hello World!"));
}
```

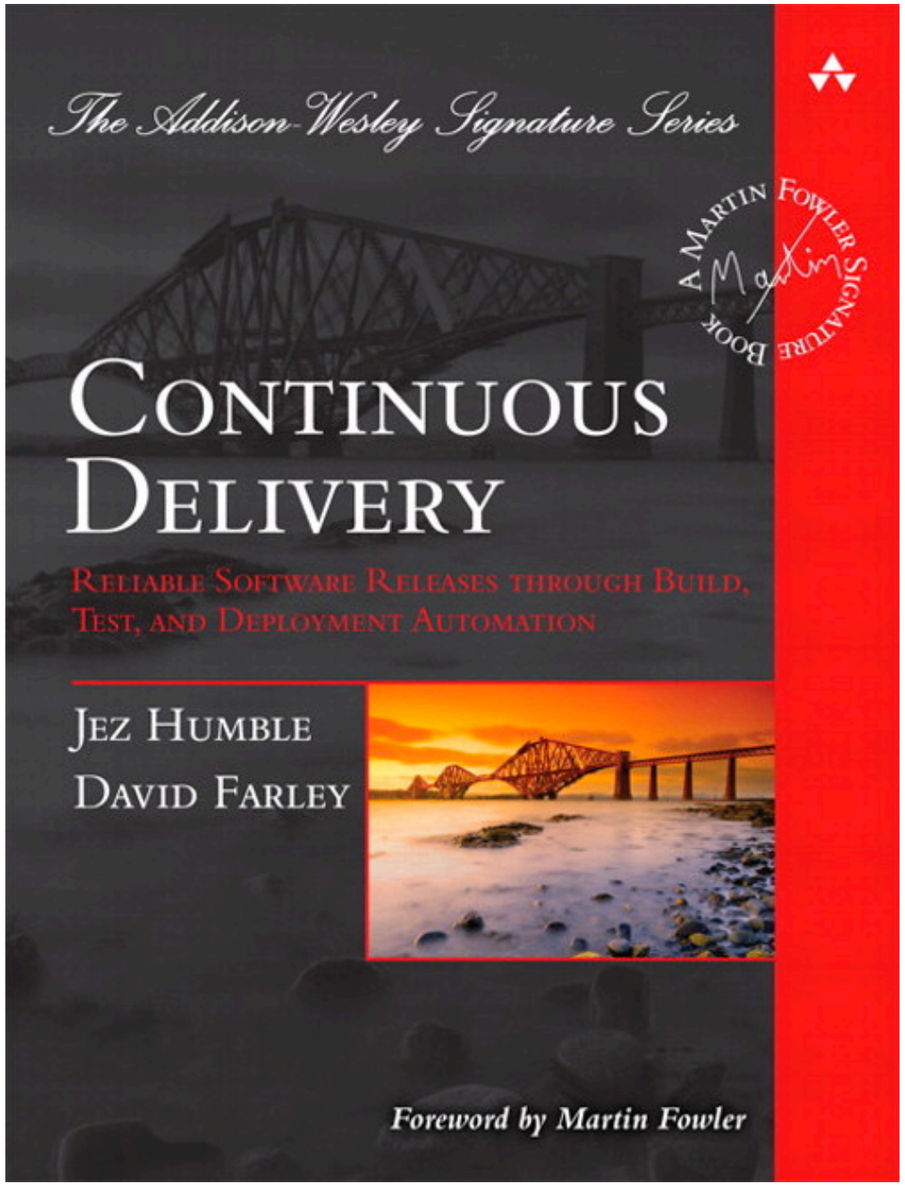
Continuous Integration

“Behind every successful agile project, there is a

Continuous Integration System”

Improve quality and reduce risk

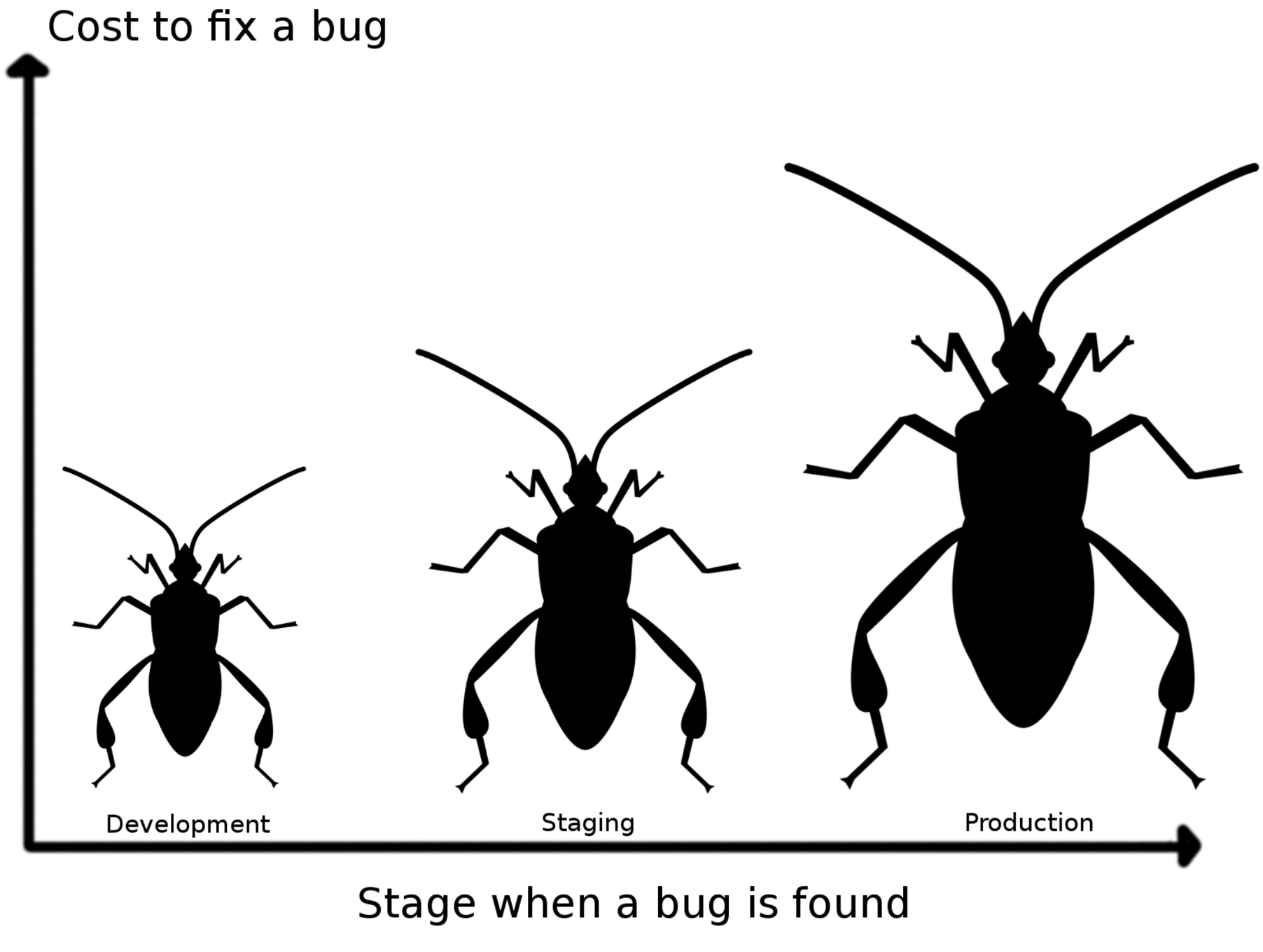


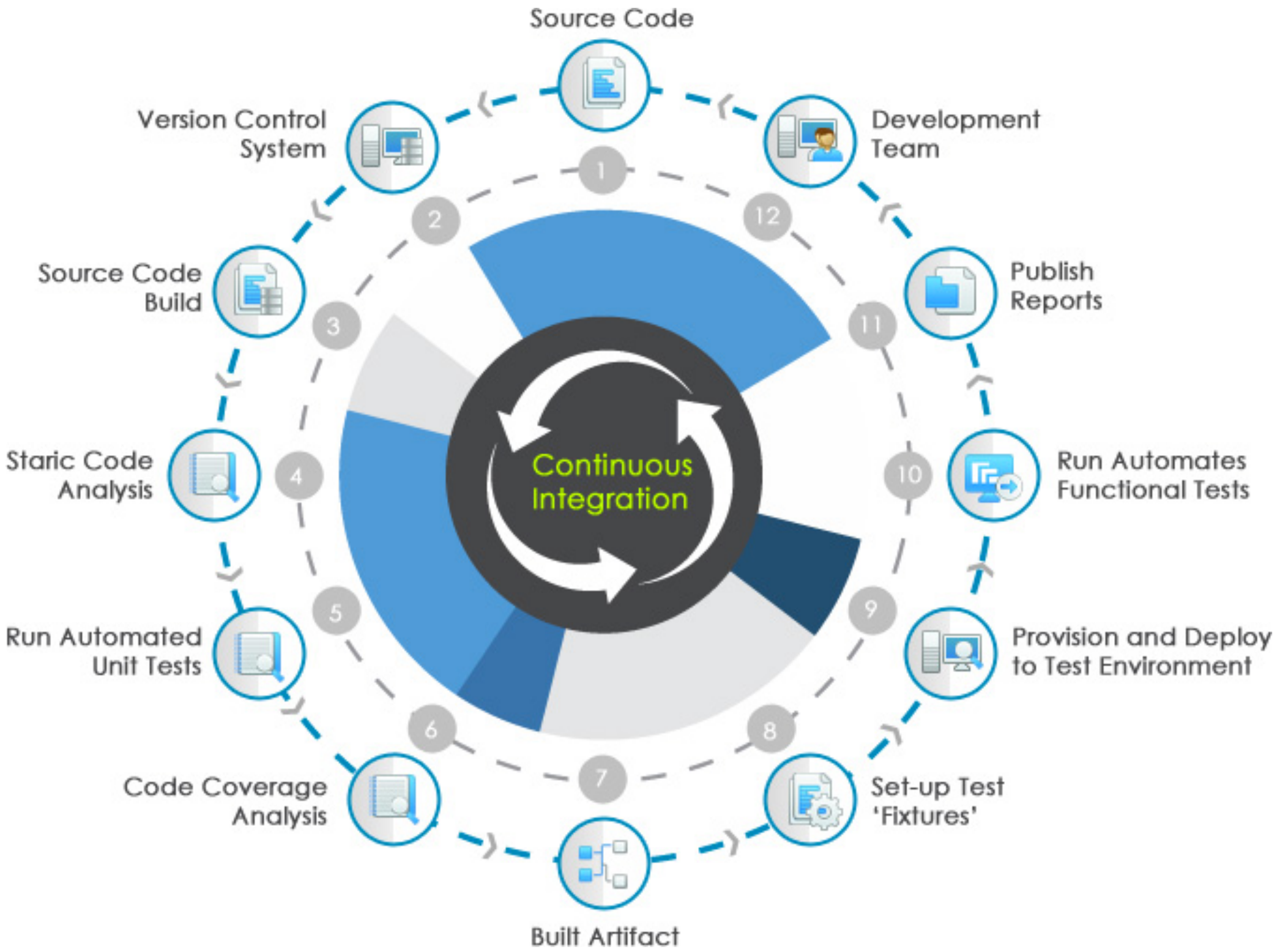


The cost of integration

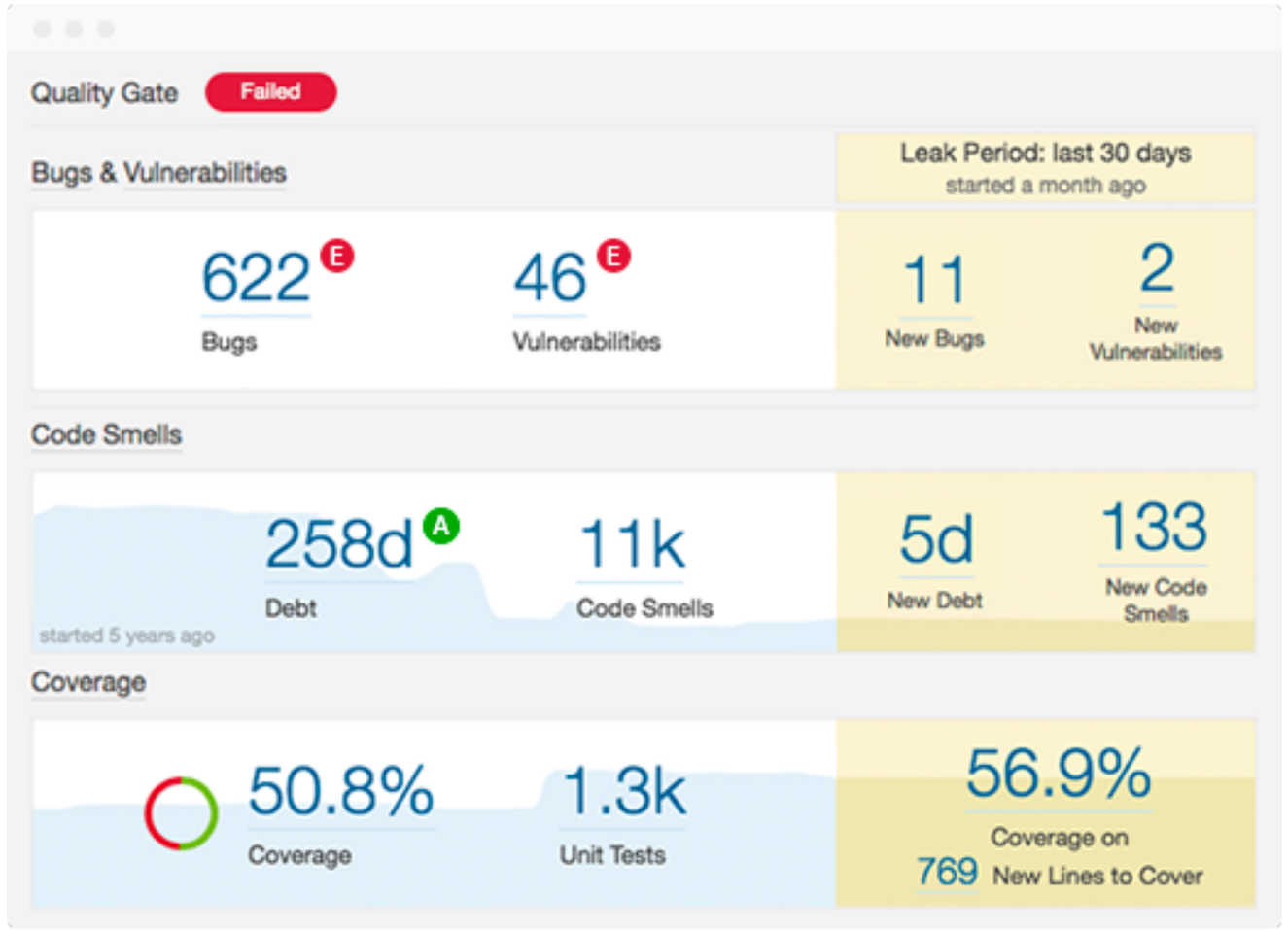
1. Merging the code
2. Duplicate changes
3. Test again again !!
4. Fixing bugs
5. Impact on stability

The cost of integration





Code Quality



https://www.sonarqube.org/

Jenkins

Bamboo

TeamCity

go

Visual Studio
Team Foundation Server

Hudson

Travis

wercker

circleci

CI is about what people do not about what tools they use

Continuous Integration

Discipline to **integrate frequently**

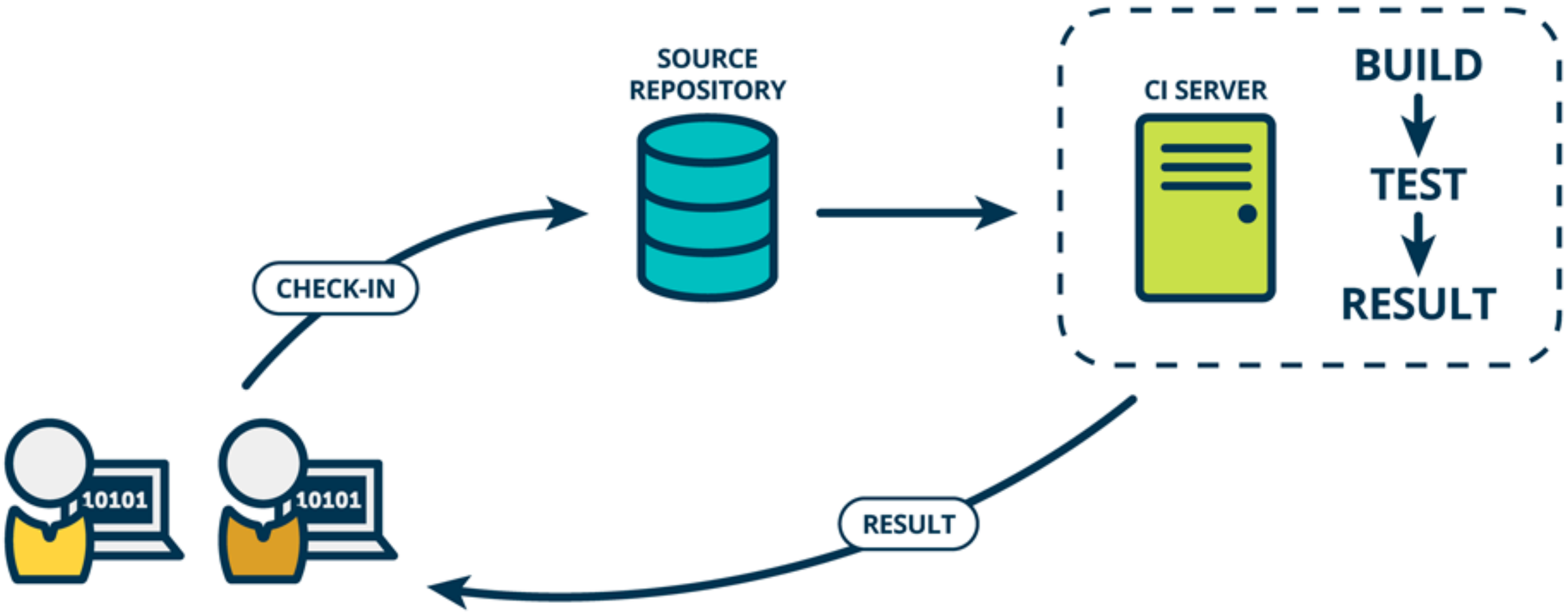
Continuous Integration

Strive to make **small change**

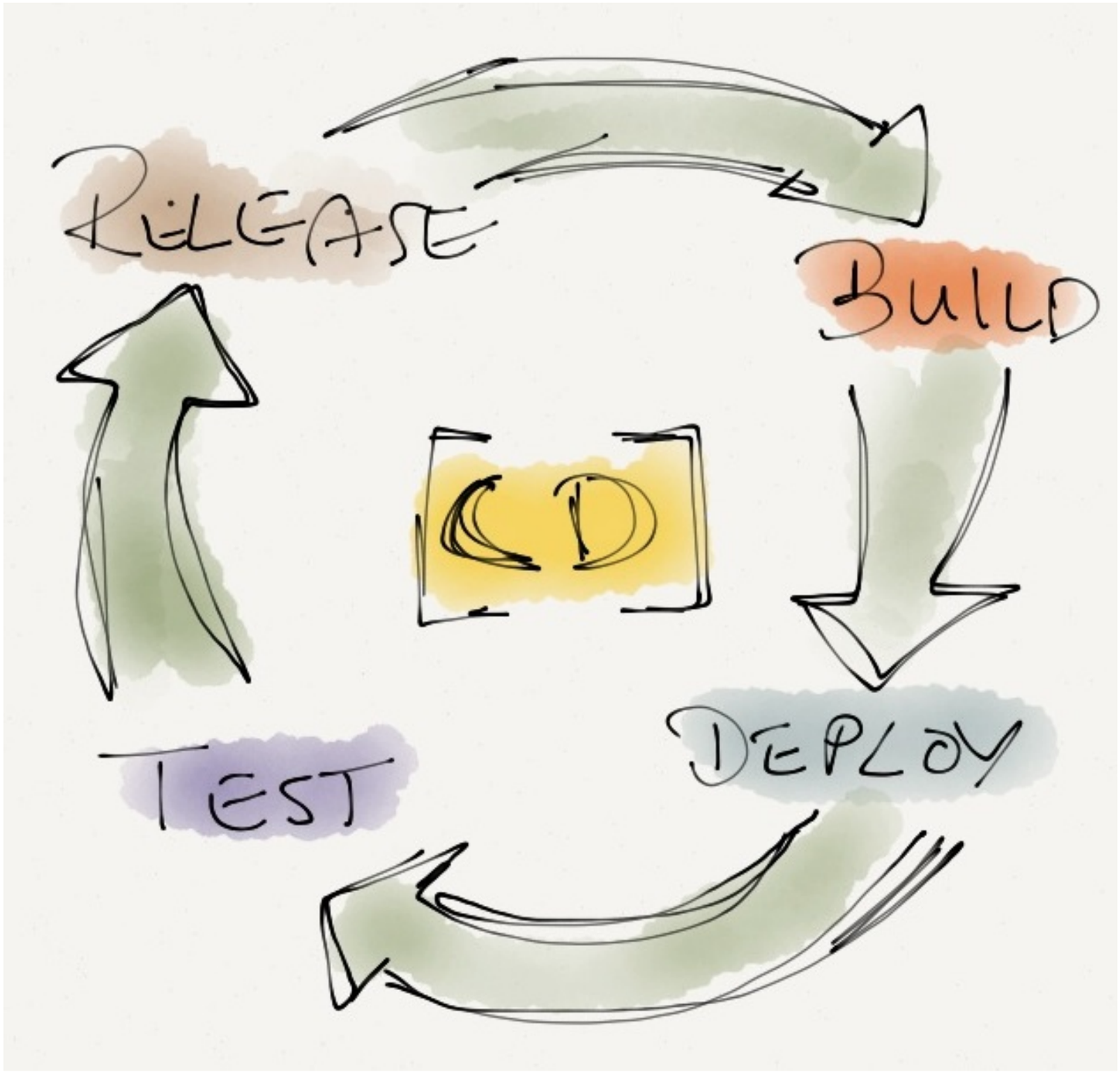
Continuous Integration

Strive for **fast feedback**

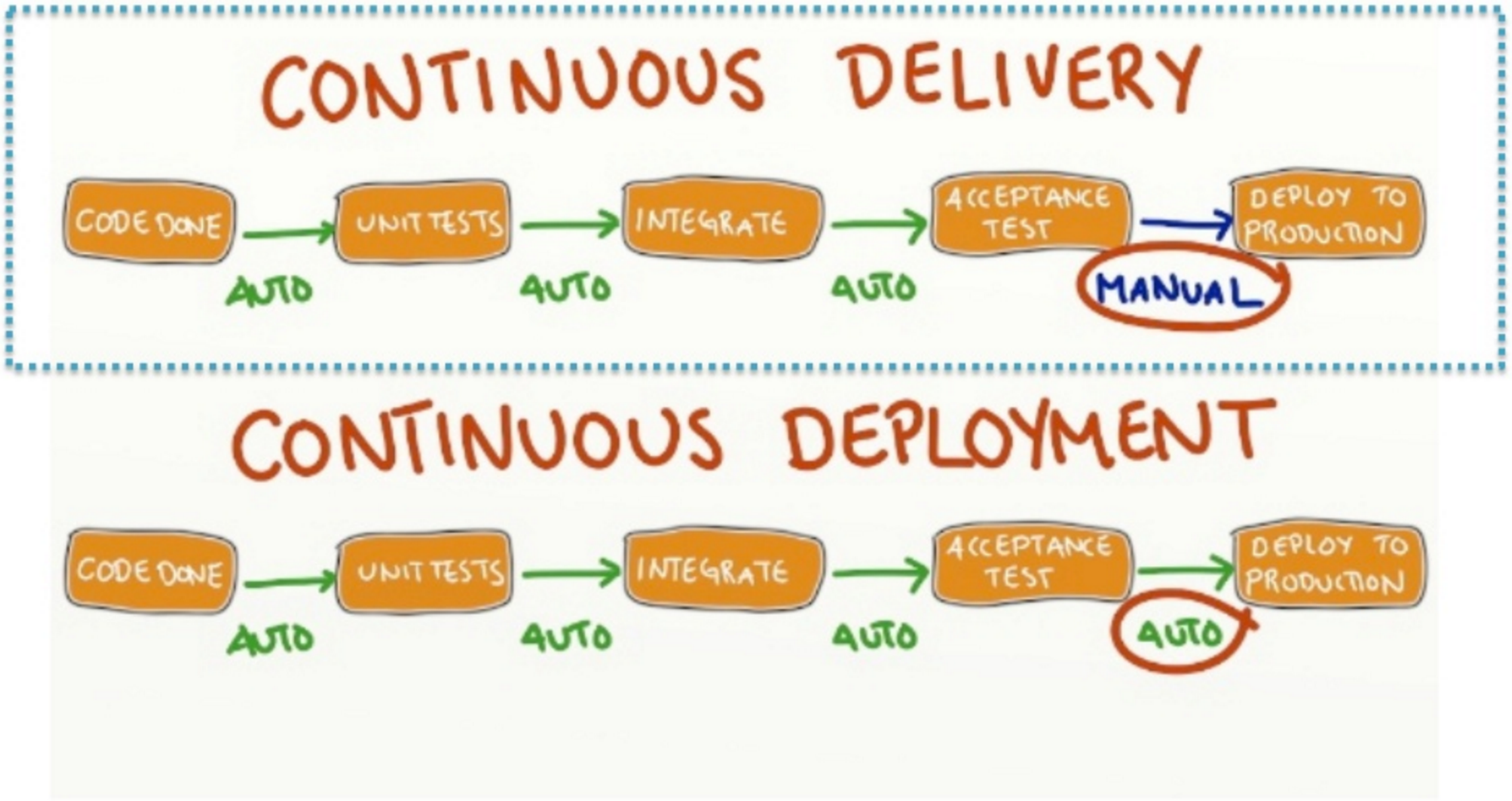
Continuous Integration



CD ?

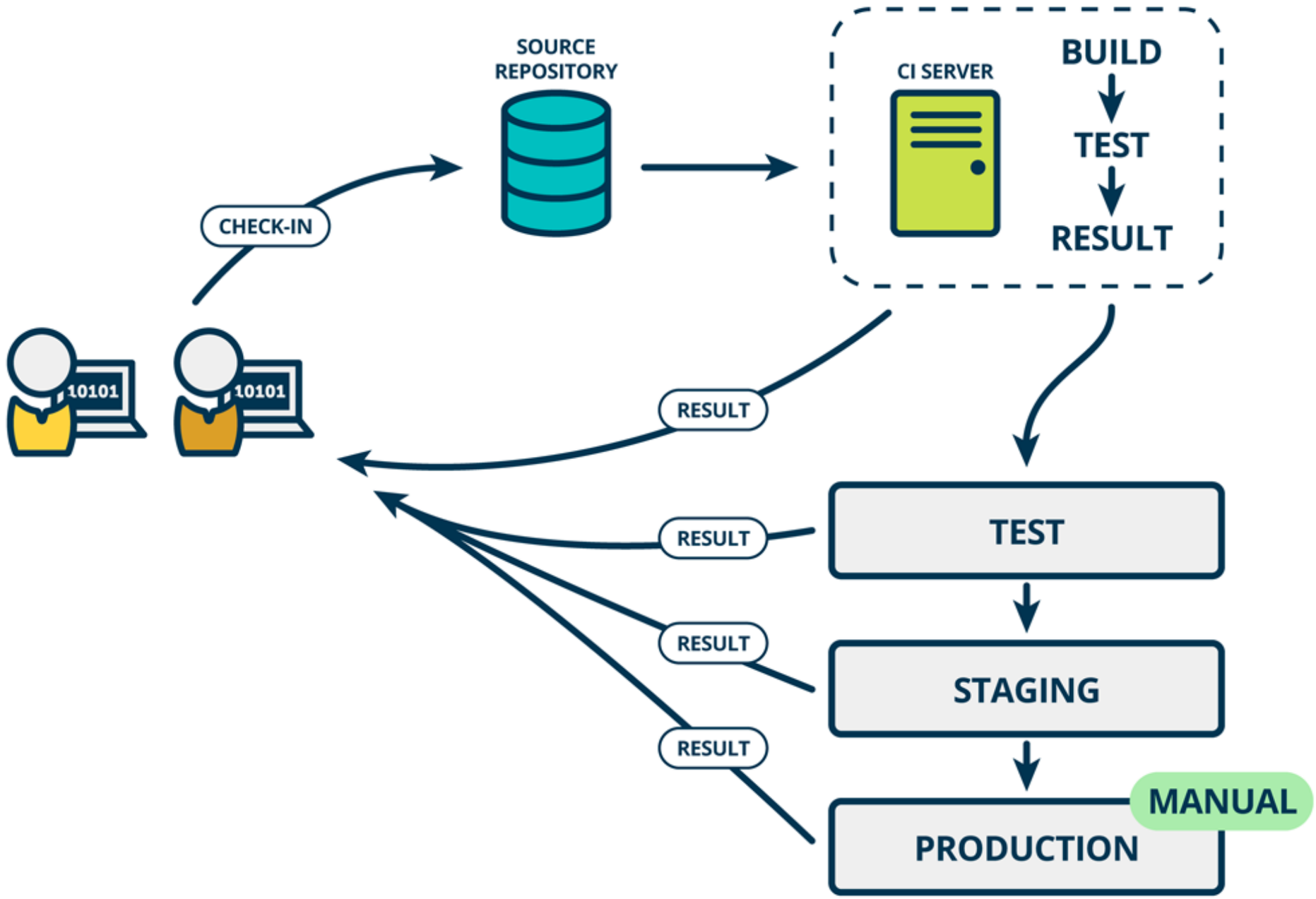


CD ?

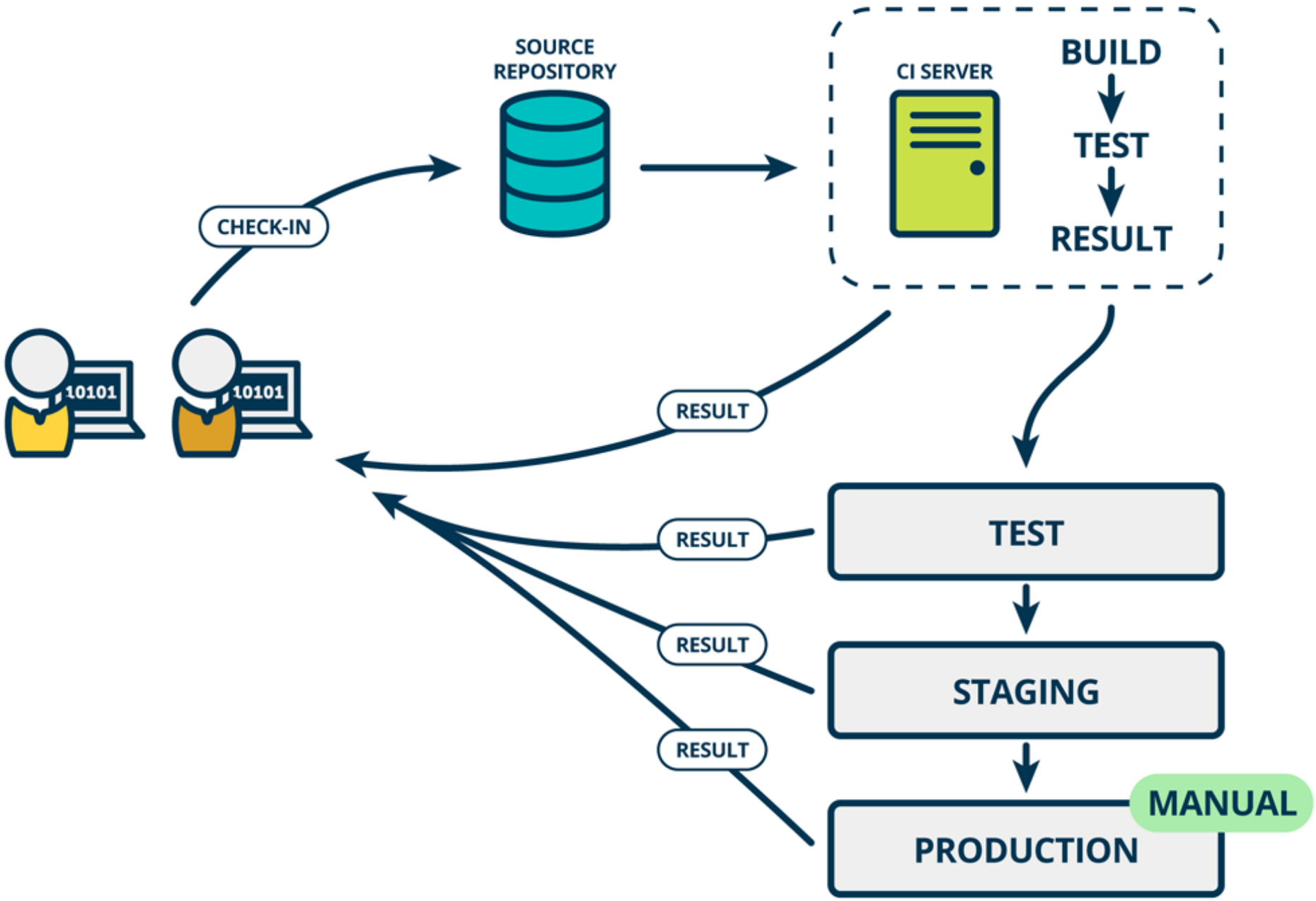


http://blog.crisp.se/2013/02/05/yassalsundman/continuous-delivery-vs-continuous-deployment

Continuous Delivery



Rise of DevOps



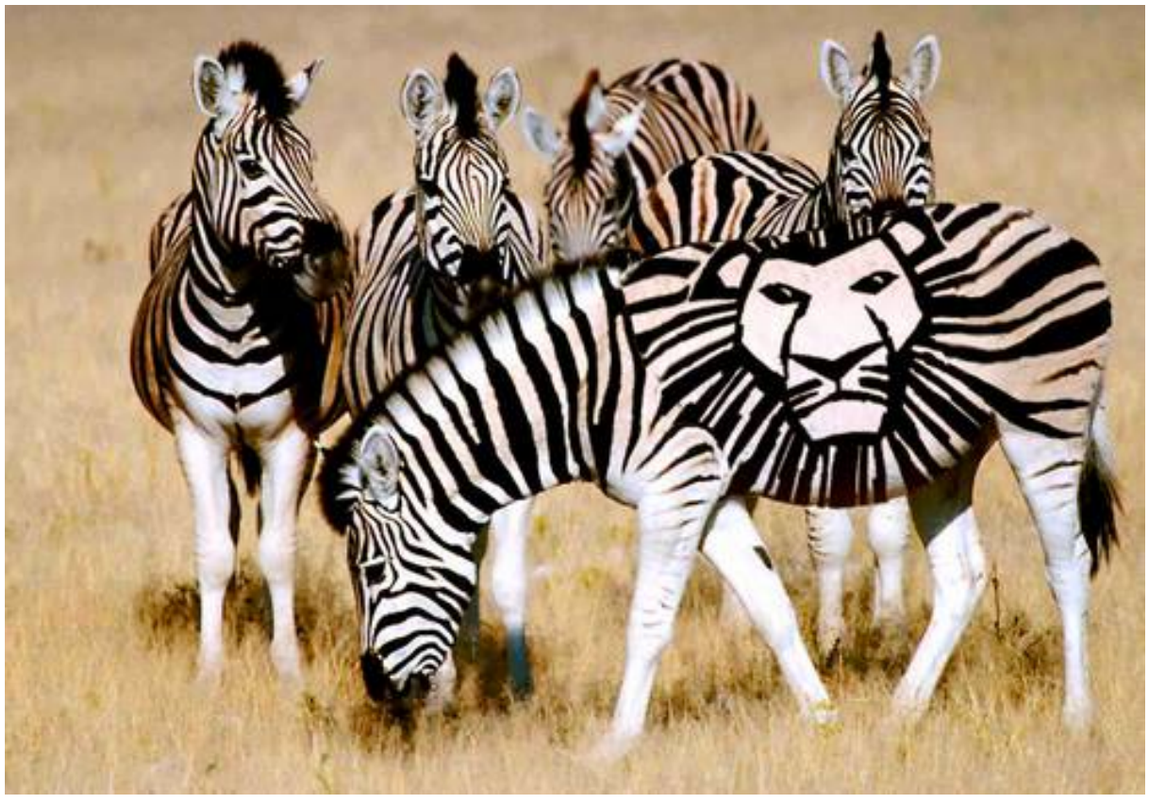
Continuous Integration

is a Software development **practices**

Practice 1

Maintain a single source repository

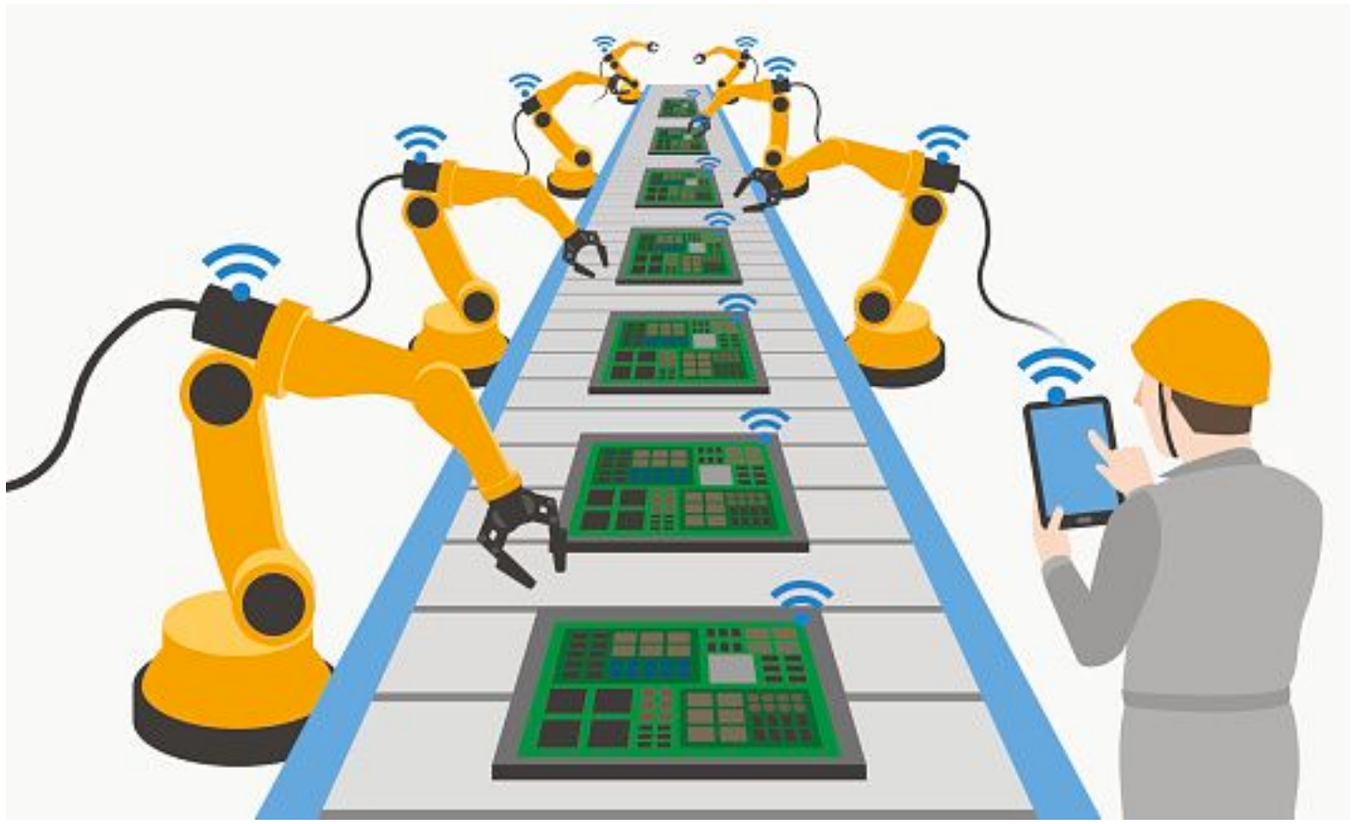
In general, you should store in source control everything you need to build anything



Practice 2

Automated the build

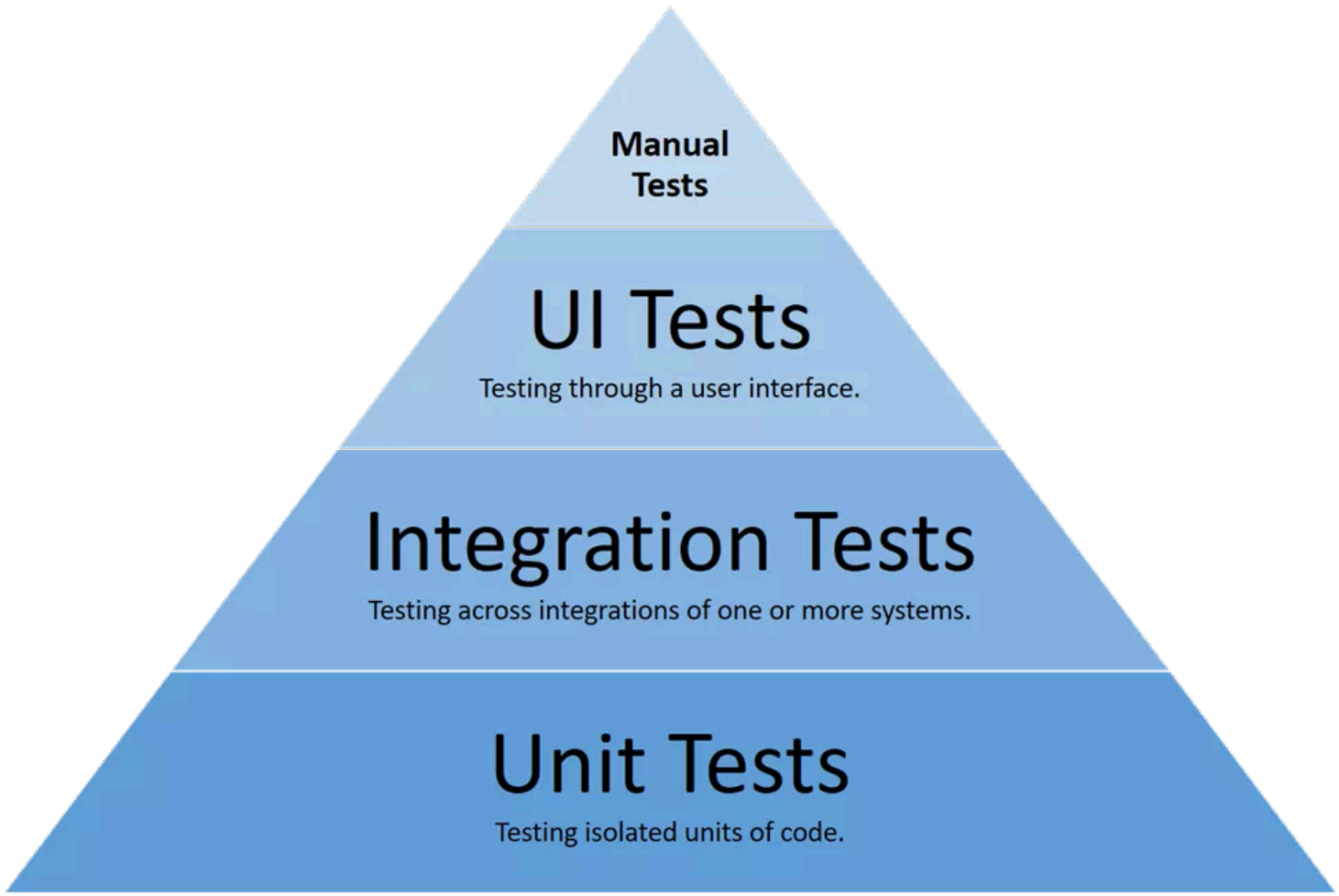
Automated environment for builds



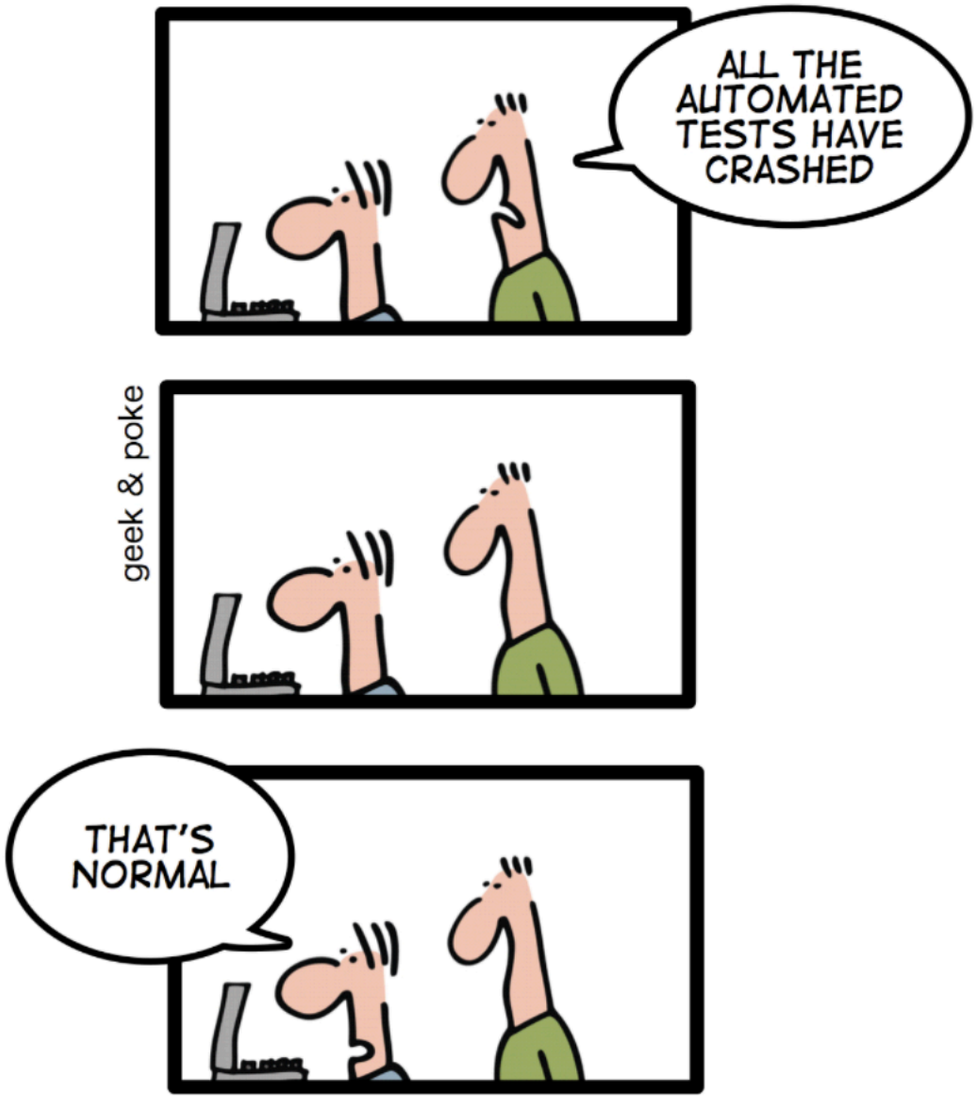
Practice 3

Make your build **self-testing**

Build process => compile, linking and **testing**



TODAY: CONTINUOUS INTEGRATION GIVES YOU THE COMFORTING FEELING TO KNOW THAT EVERYTHING IS NORMAL

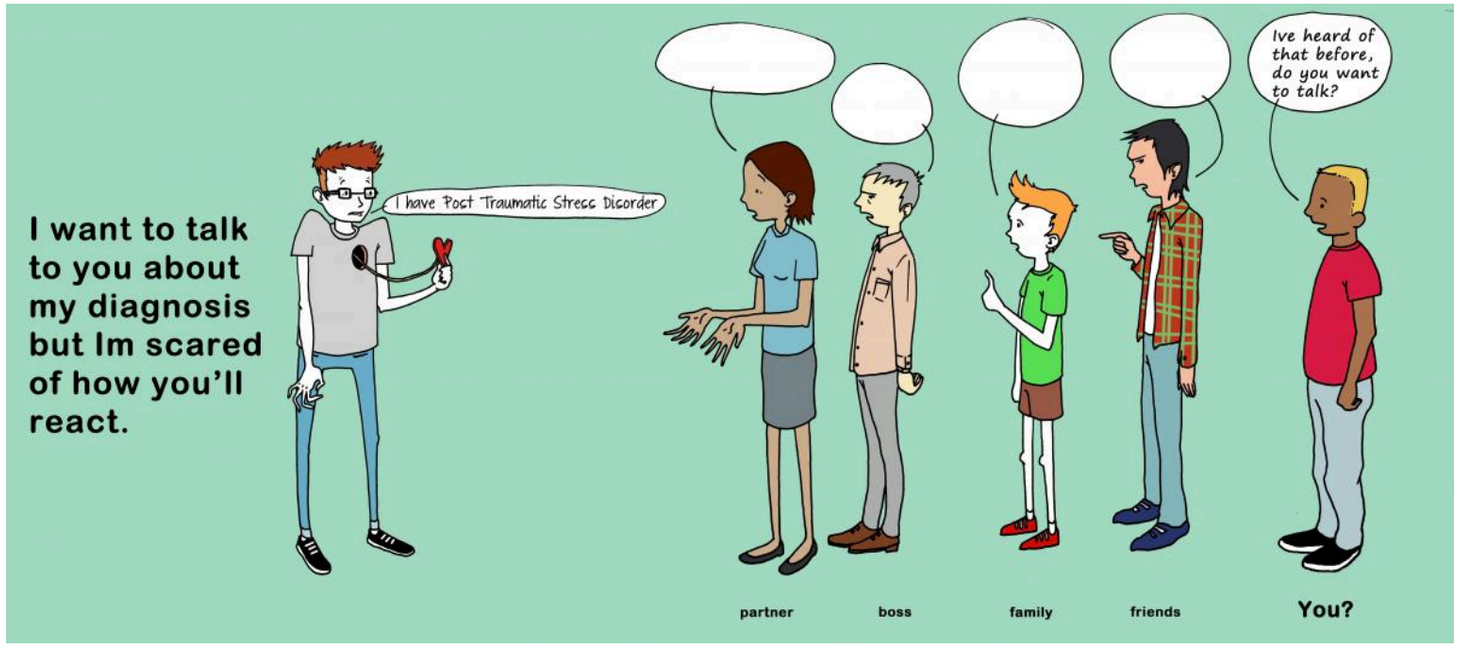


Practice 4

Everyone **commits** to the mainline everyday

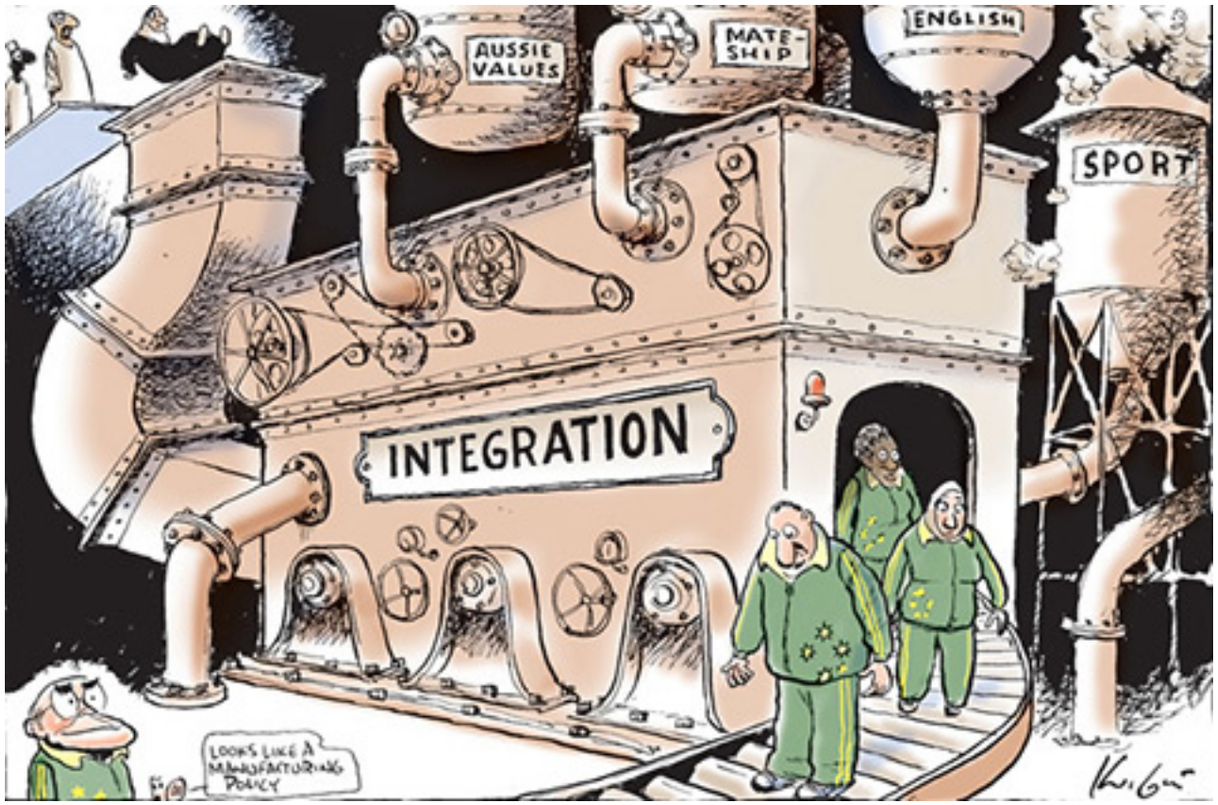
Integration is about **communication**

Integration allows developers to **tell** other developers



Practice 5

Every commits should build the mainline on an **Integration machine**



Nightly build is not enough for Continuous Integration

Practice 6

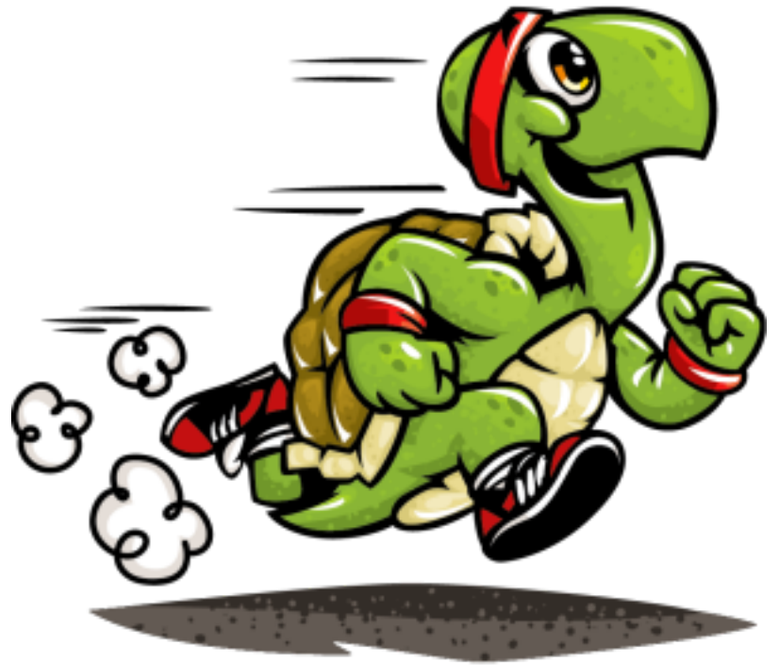
Fix broken builds **immediately**

"Nobody has a higher priority task than fixing the build"

Practice 7

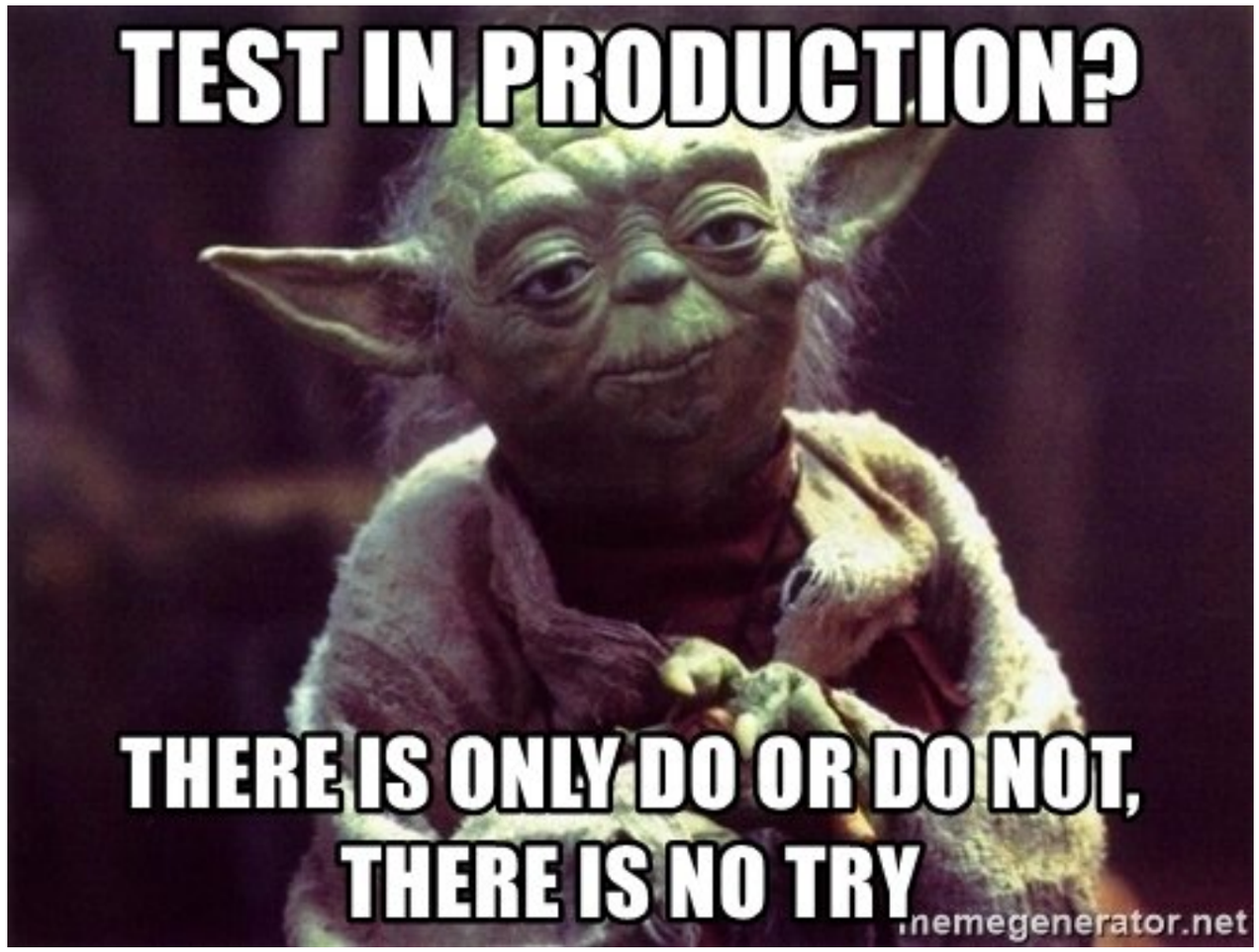
Keep the build **fast**

Continuous Integration is to provide rapid feedback



Practice 8

Test in clone of the **Production** environment



Practice 9

Make it easy for anyone to get
the latest executable

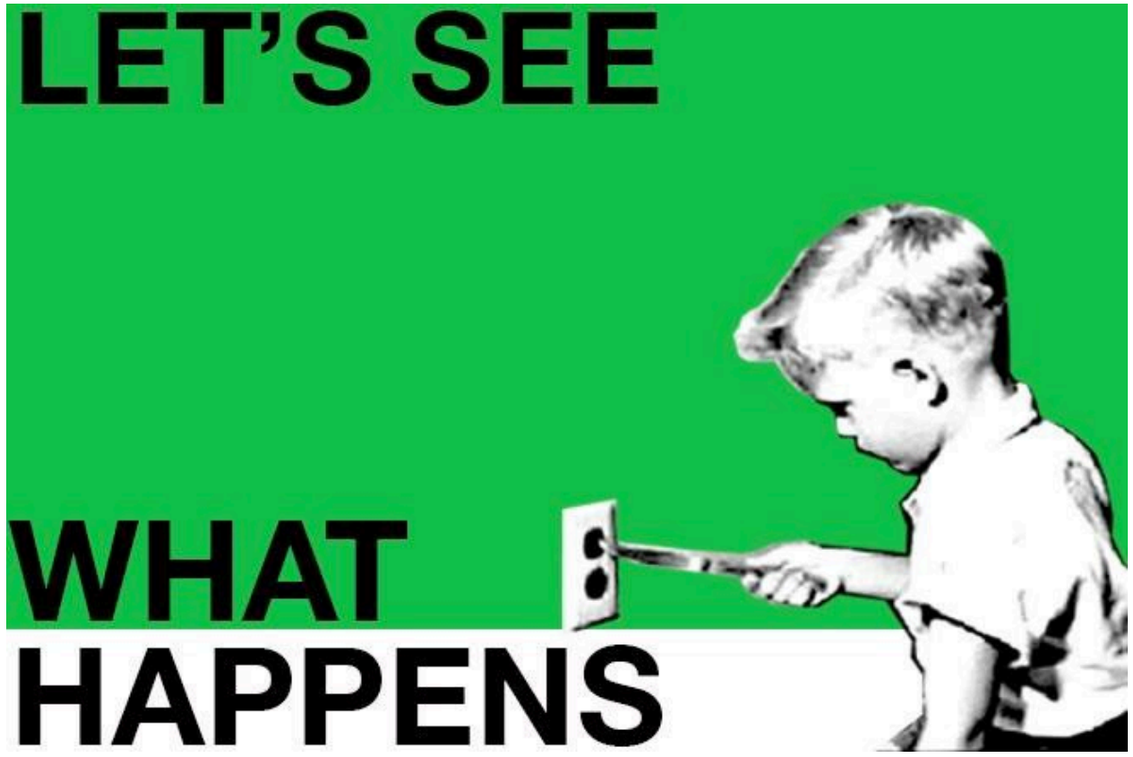
Make sure well known place where people can find

Practice 10

Everyone can see what's happening

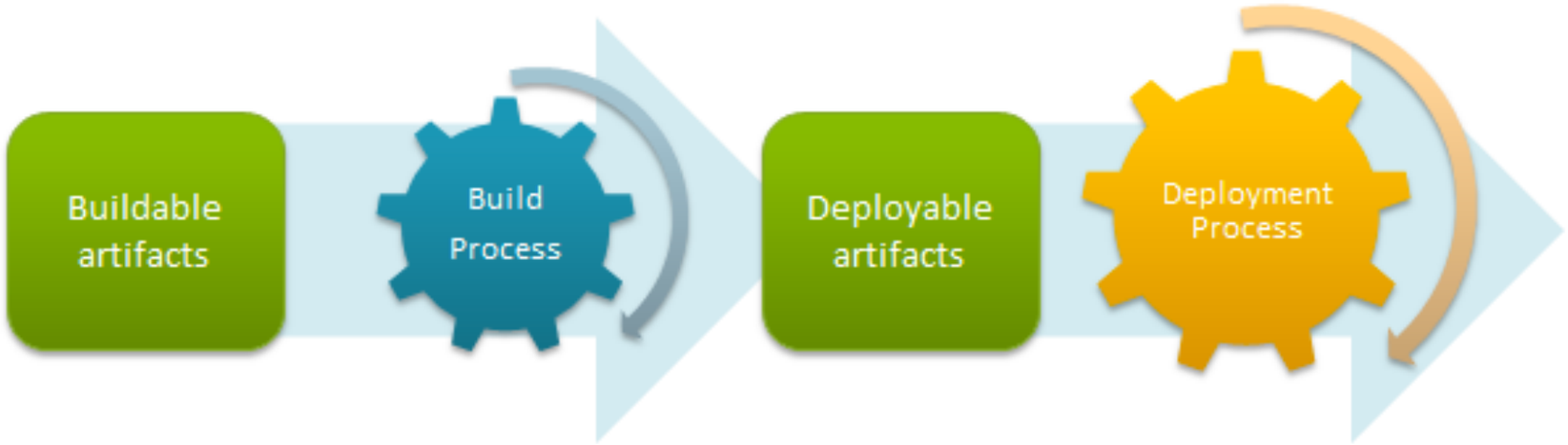
Easier to see the state of the system and changes

Show the good information



Practice 11

Automated deployment



Workshop