

TDD with Java





Somkiat Puisungnoen

Somkiat Puisungnoen

Update Info 1 View Activity Log 10+ ...

Timeline About Friends 3,138 Photos More

When did you work at Opendream? X

... 22 Pending Items

Intro

Software Craftsmanship

Software Practitioner at สยามชัมนาภิกิจ พ.ศ. 2556

Agile Practitioner and Technical at SPRINT3r

Post Photo/Video Live Video Life Event

What's on your mind?

Public Post

Somkiat Puisungnoen 15 mins · Bangkok · ⚙️

Java and Bigdata





Page

Messages

Notifications 3

Insights

Publishing Tools

Settings

Help ▾



somkiat.cc

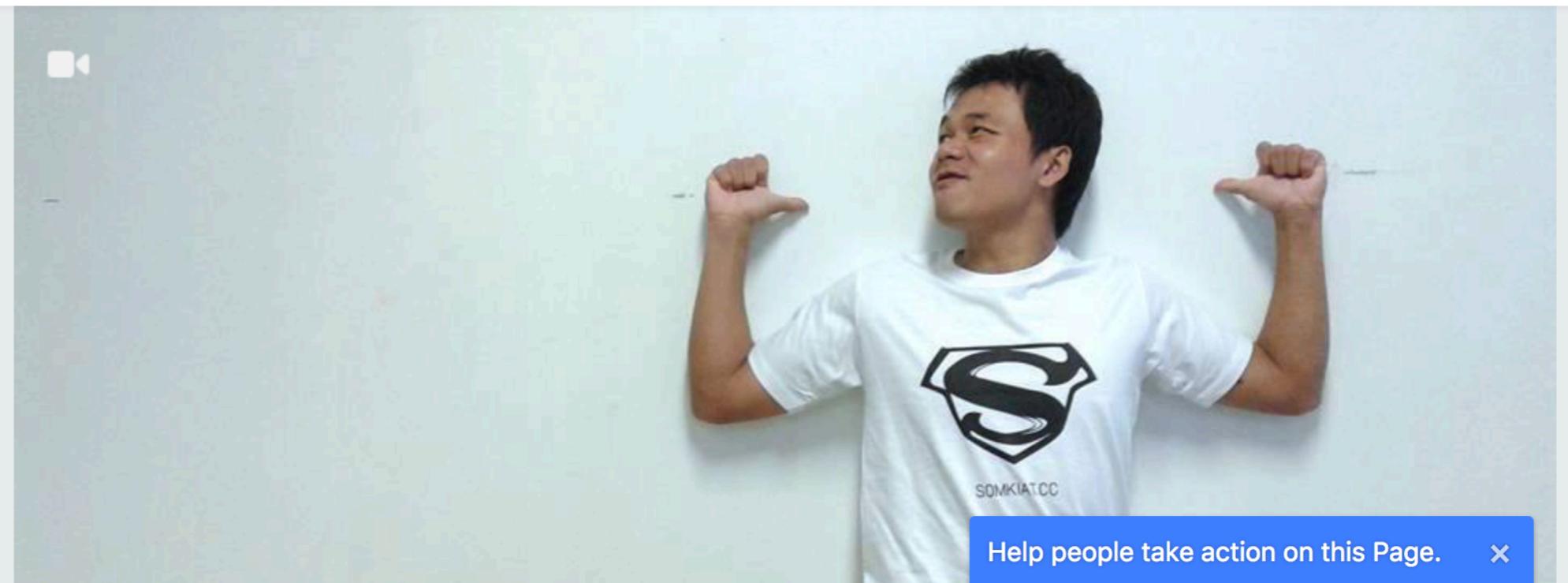
@somkiat.cc

Home

Posts

Videos

Photos



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 ?

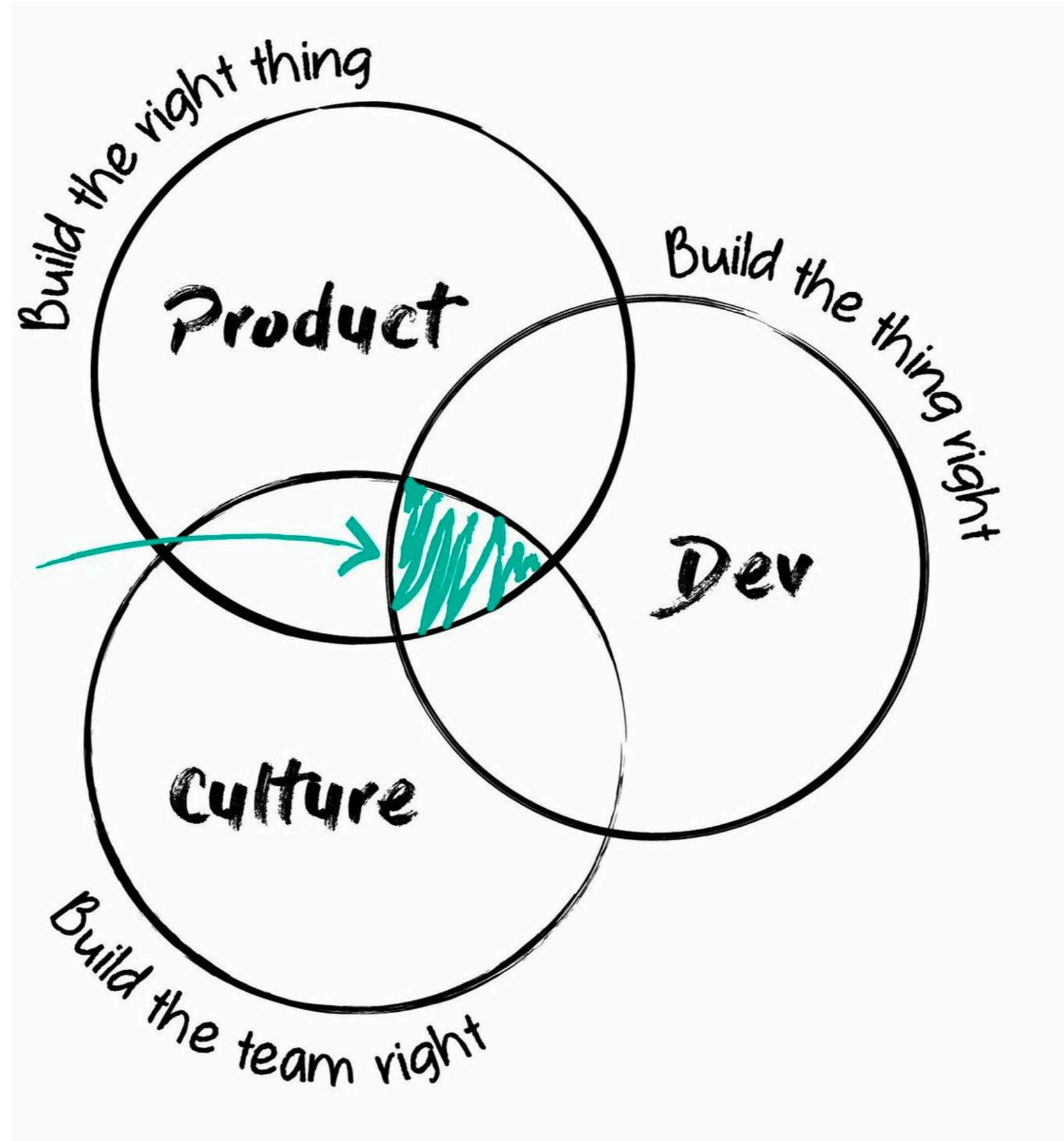
Good design principle

BDD (Behavior-Driven Development)

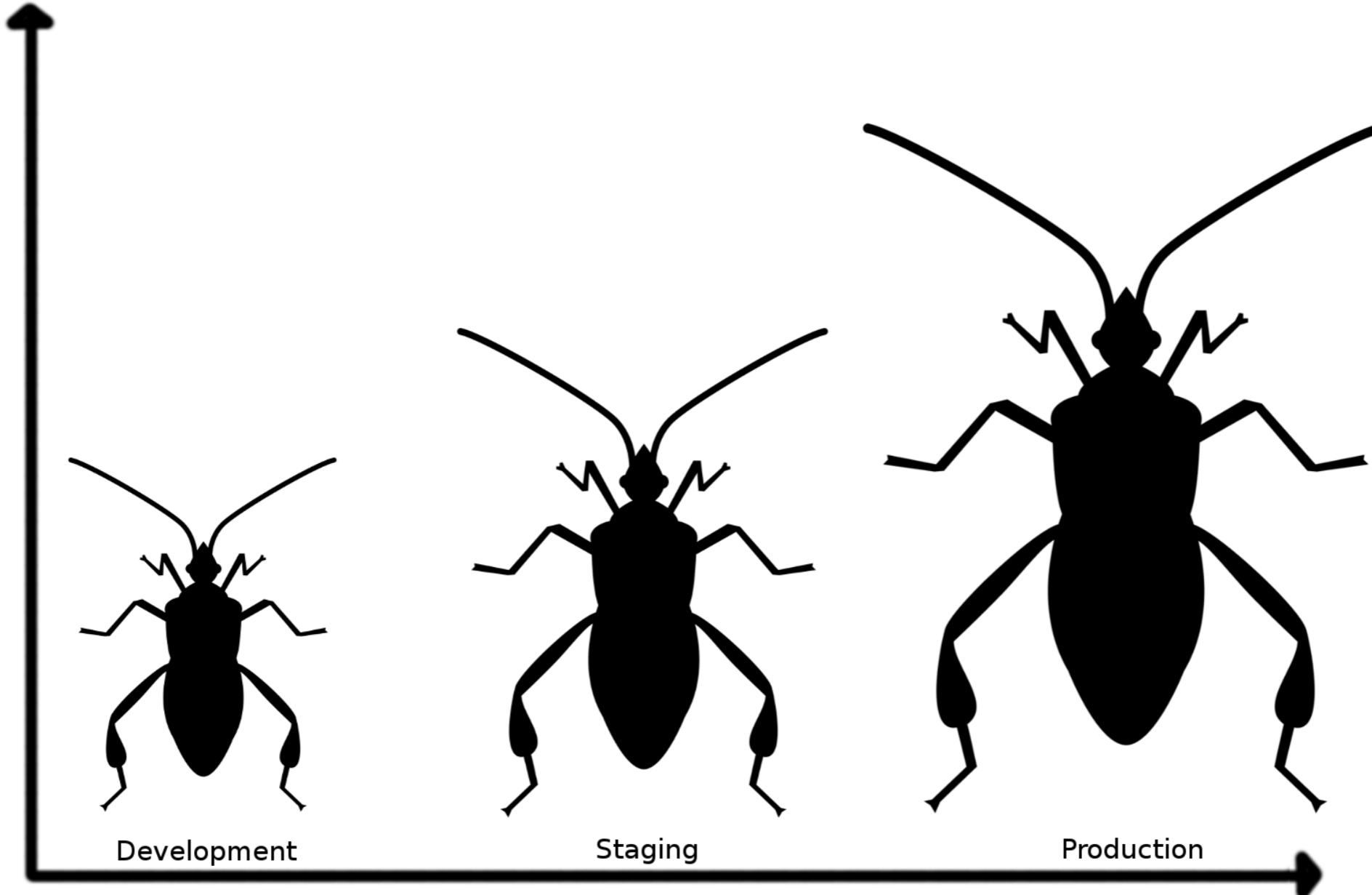
Working with Legacy code

Workshop





Cost to fix a bug



Stage when a bug is found





Fixing Bugs



Bug



Find Code



Write Test



Fix Test





SPECIFICATION BY EXAMPLE



CONTINUOUS
INTEGRATION



CONTINUOUS DELIVERY



TEST AUTOMATION



THINKING ABOUT TESTING



TECHNICAL
EXCELLENCE



DEVELOPMENT
TEST-DRIVEN DEVELOPMENT



ARCHITECTURE
& DESIGN



ACCEPTANCE
TESTING



CLEAN CODE



UNIT TESTING

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





SPECIFICATION BY EXAMPLE



CONTINUOUS
INTEGRATION



CONTINUOUS DELIVERY



TEST AUTOMATION



THINKING ABOUT TESTING



TECHNICAL
EXCELLENCE



ARCHITECTURE
& DESIGN



ACCEPTANCE
TESTING

CODE
CLEAN CODE



DEVELOPMENT
TEST-DRIVEN DEVELOPMENT



UNIT TESTING

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





SPECIFICATION BY EXAMPLE



CONTINUOUS
INTEGRATION



CONTINUOUS DELIVERY



TEST AUTOMATION



THINKING ABOUT TESTING



TECHNICAL
EXCELLENCE



TEST-DRIVEN DEVELOPMENT



ARCHITECTURE
& DESIGN



ACCEPTANCE
TESTING



CLEAN CODE



UNIT TESTING

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





TDD

**ALL CODE IS GUILTY
UNTIL PROVEN INNOCENT**

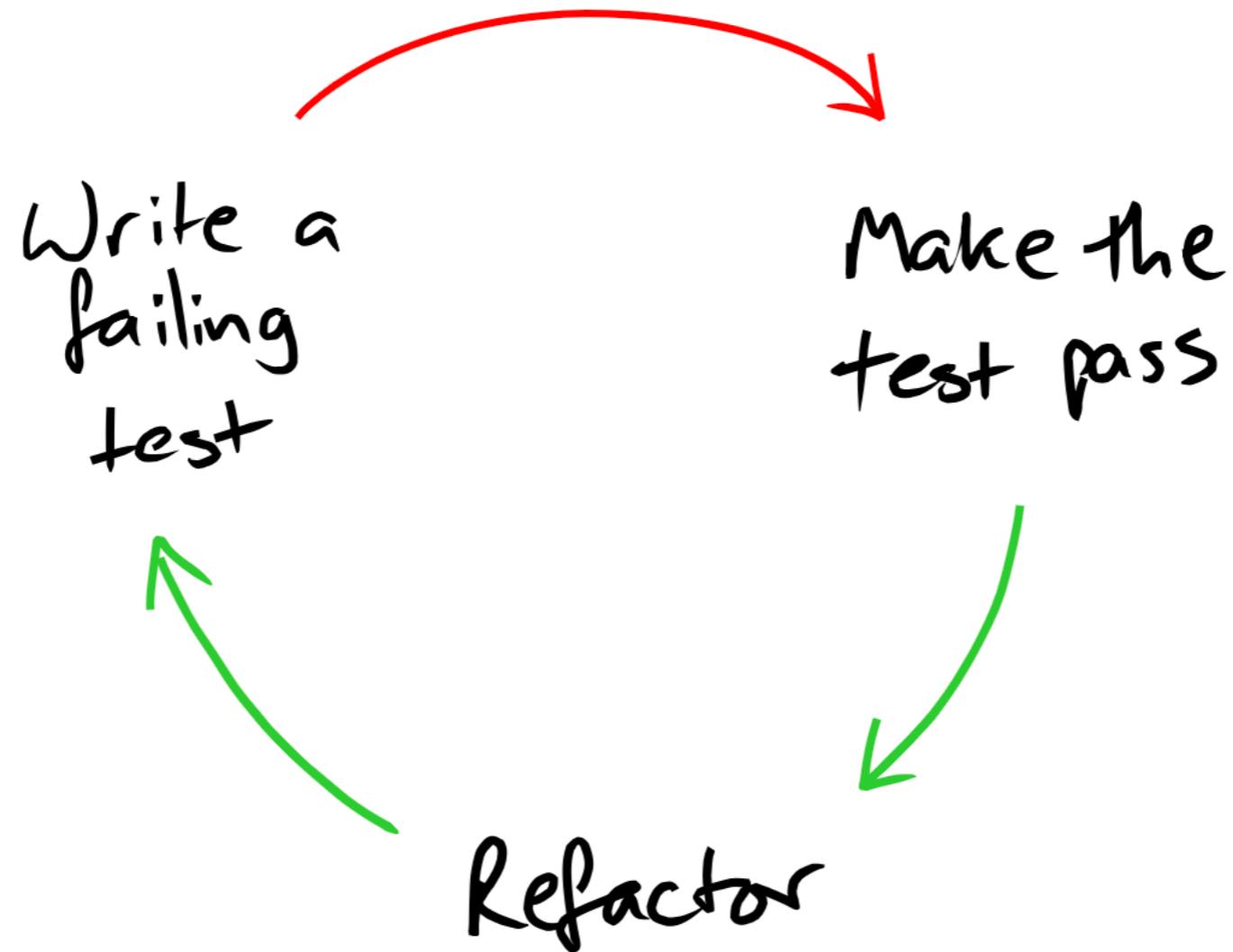


TDD

Test-Driven-Development Test-Driven Design



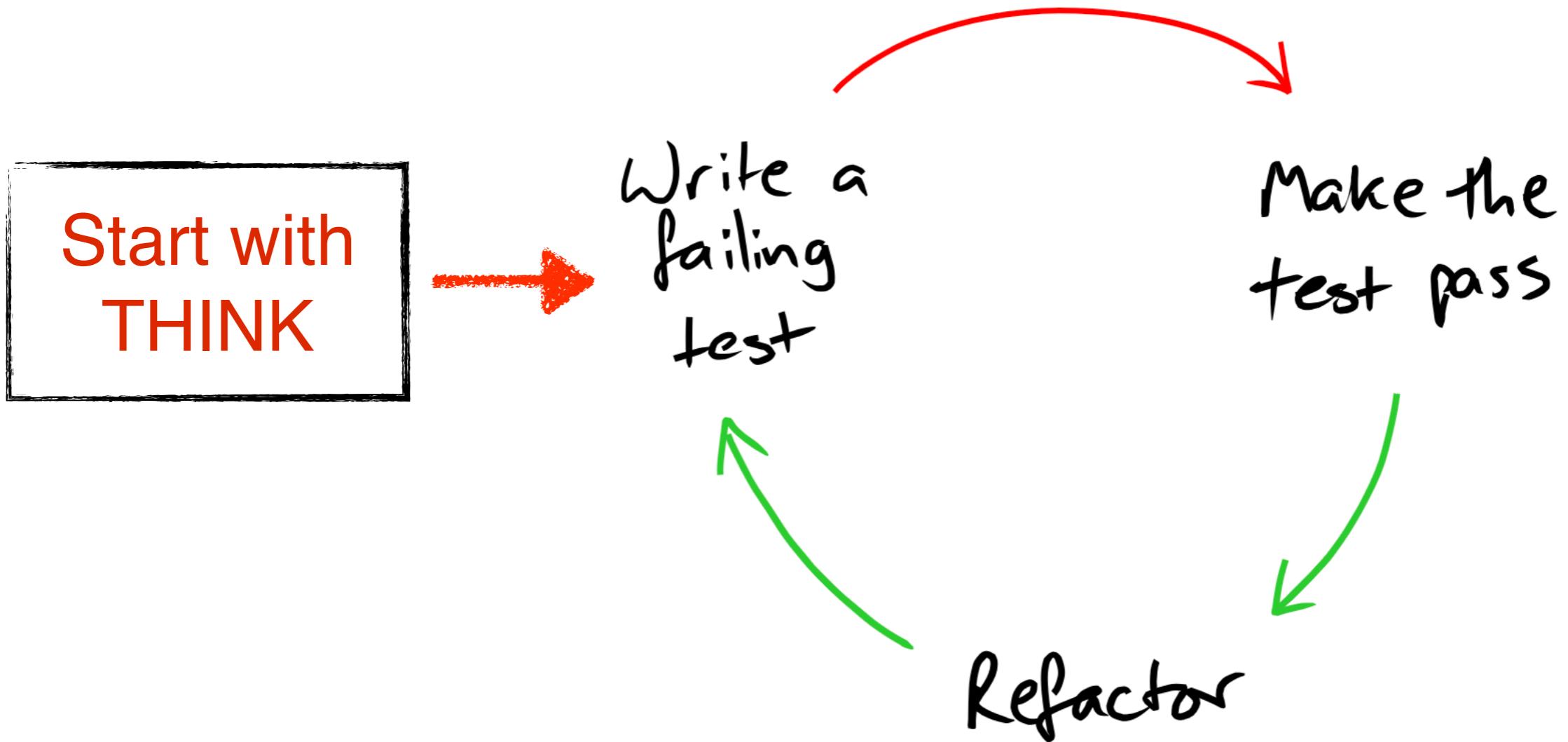
TDD Cycle



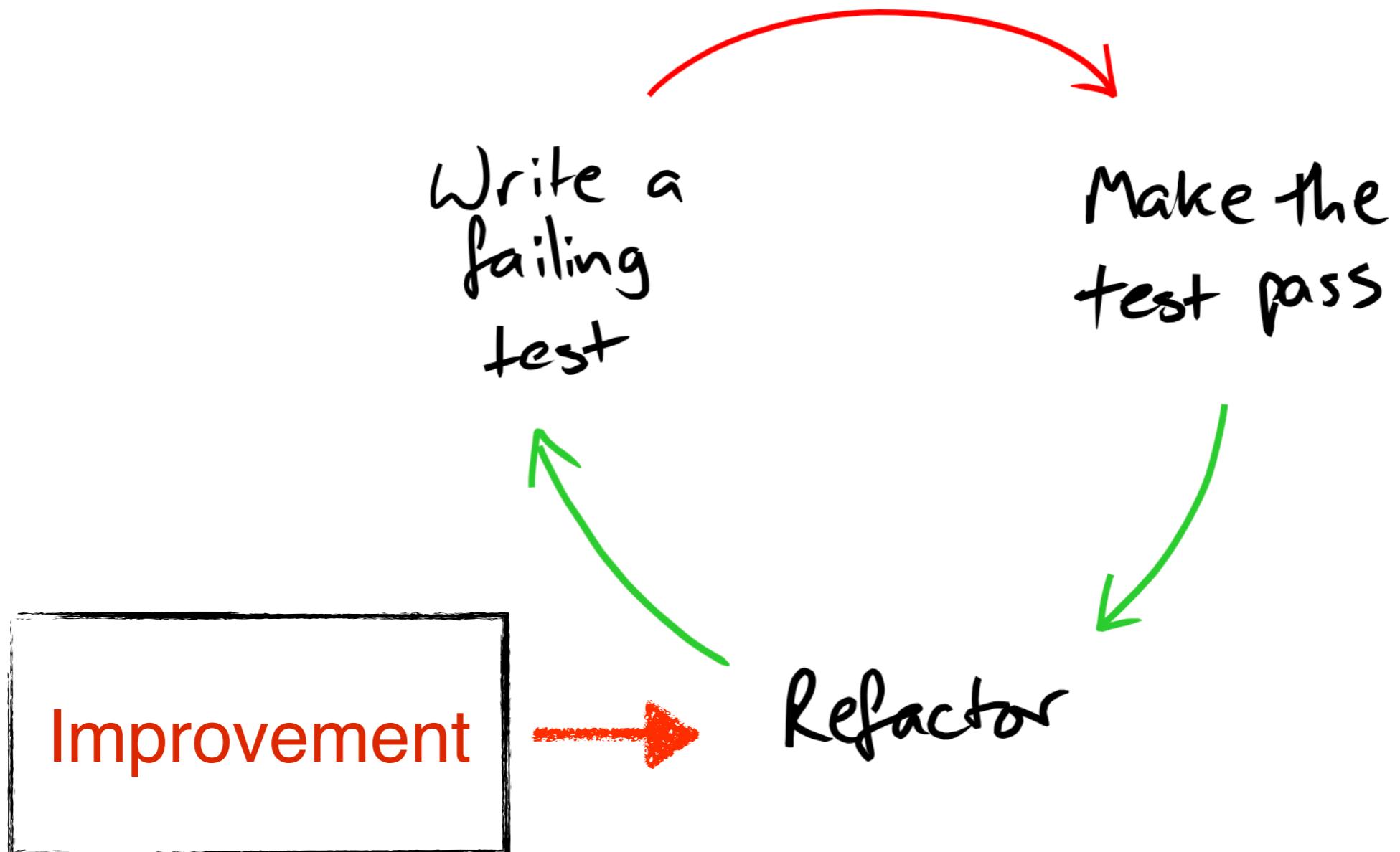
Red Green Refactor



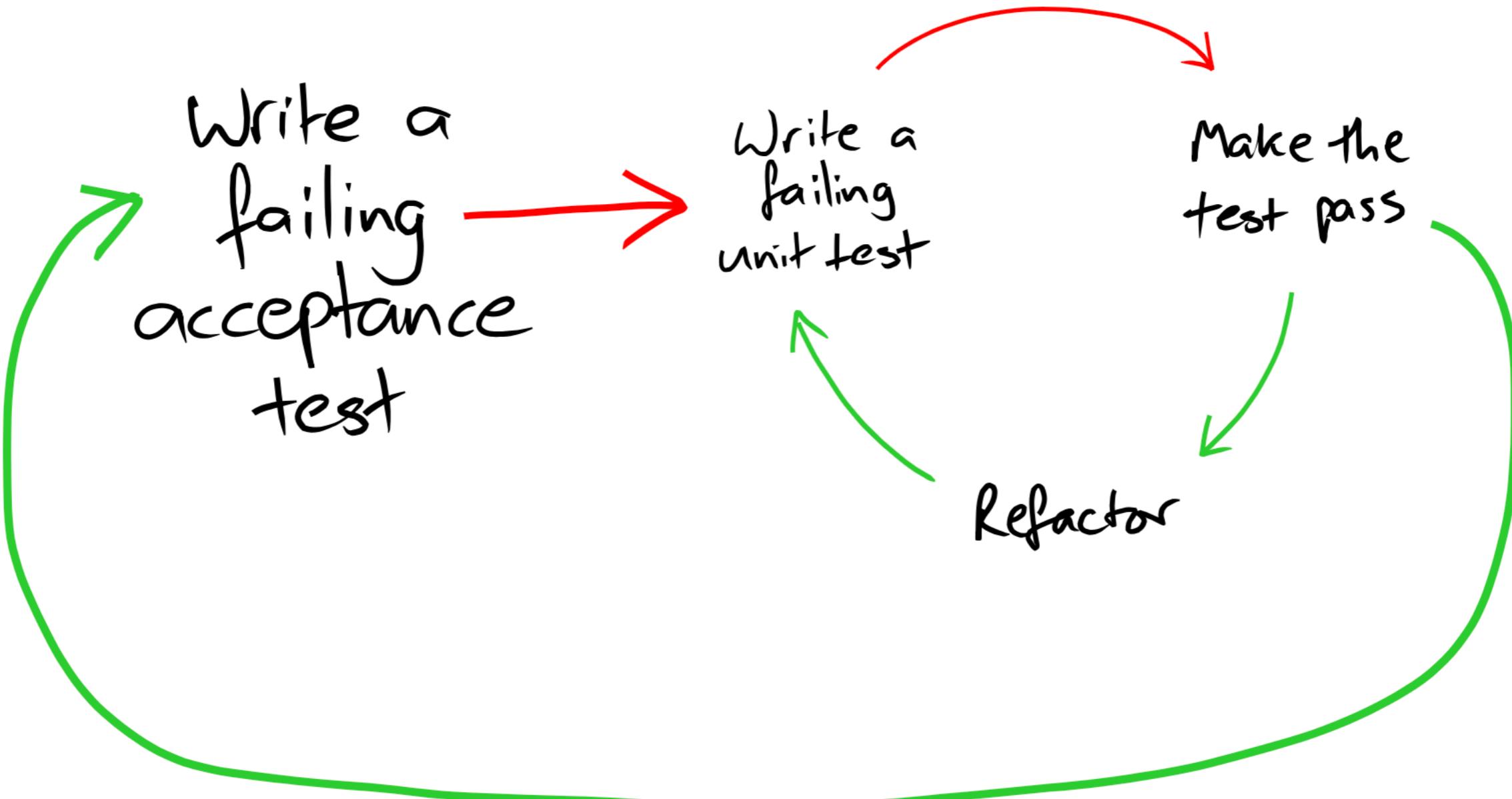
Improve TDD Cycle



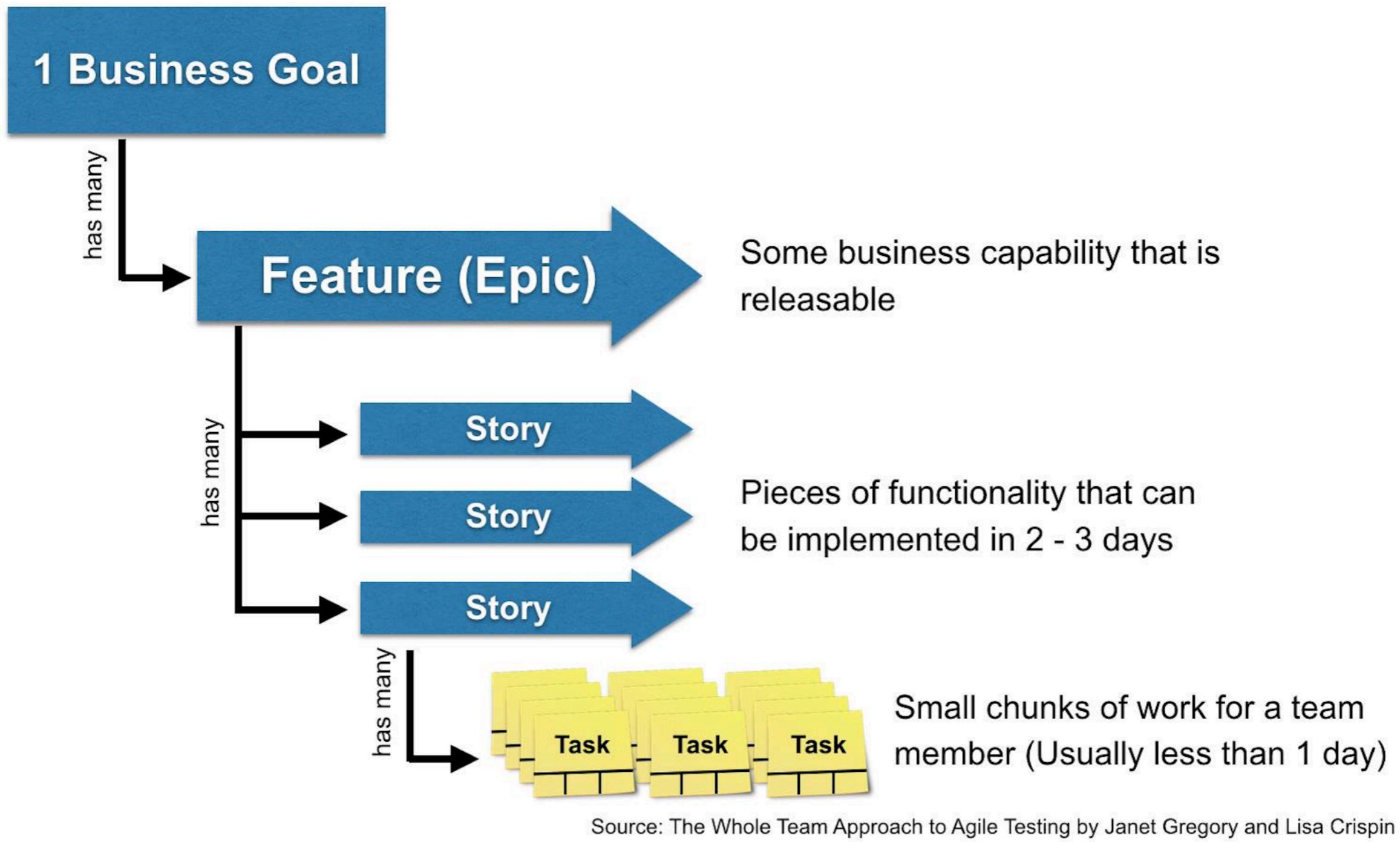
Improve TDD Cycle



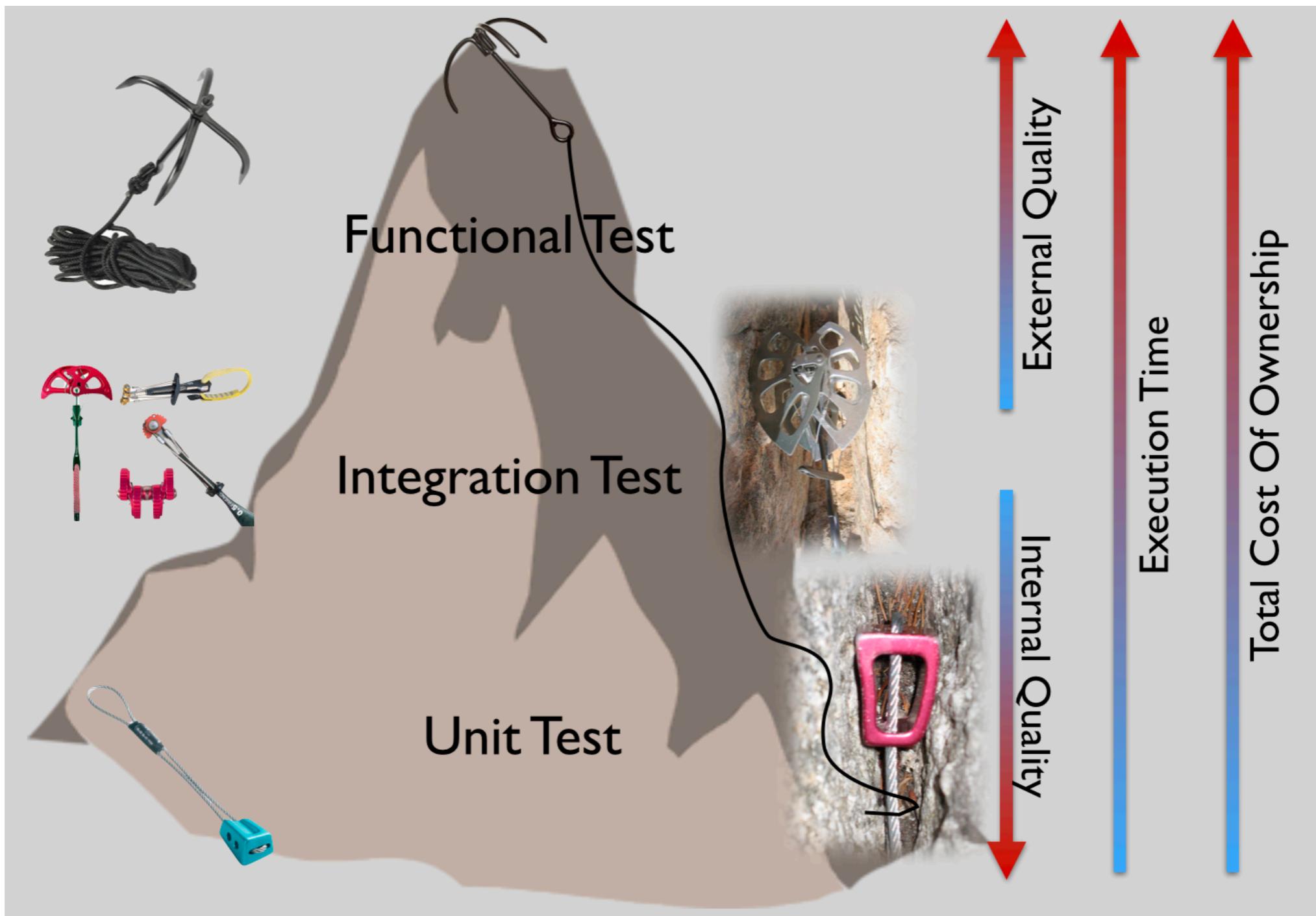
Acceptance tests



Start with Business Goal to Tasks



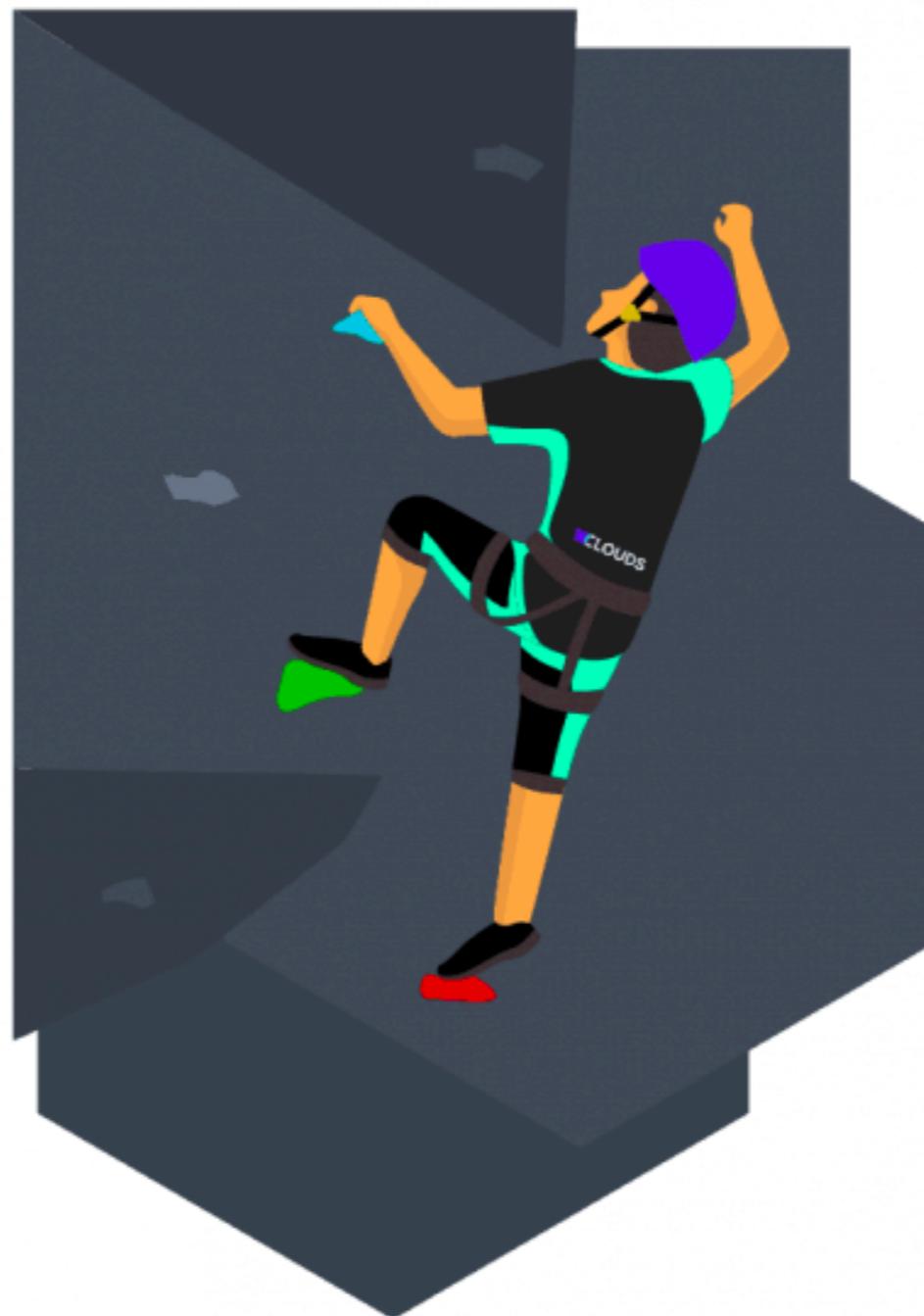
The Golden rule of TDD



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



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/>



Make the test pass !!

Fake or hard code
Triangulation
Professional code



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

Fast
Independent/Isolate
Repeat
Self-validate
Thorough/Timely



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



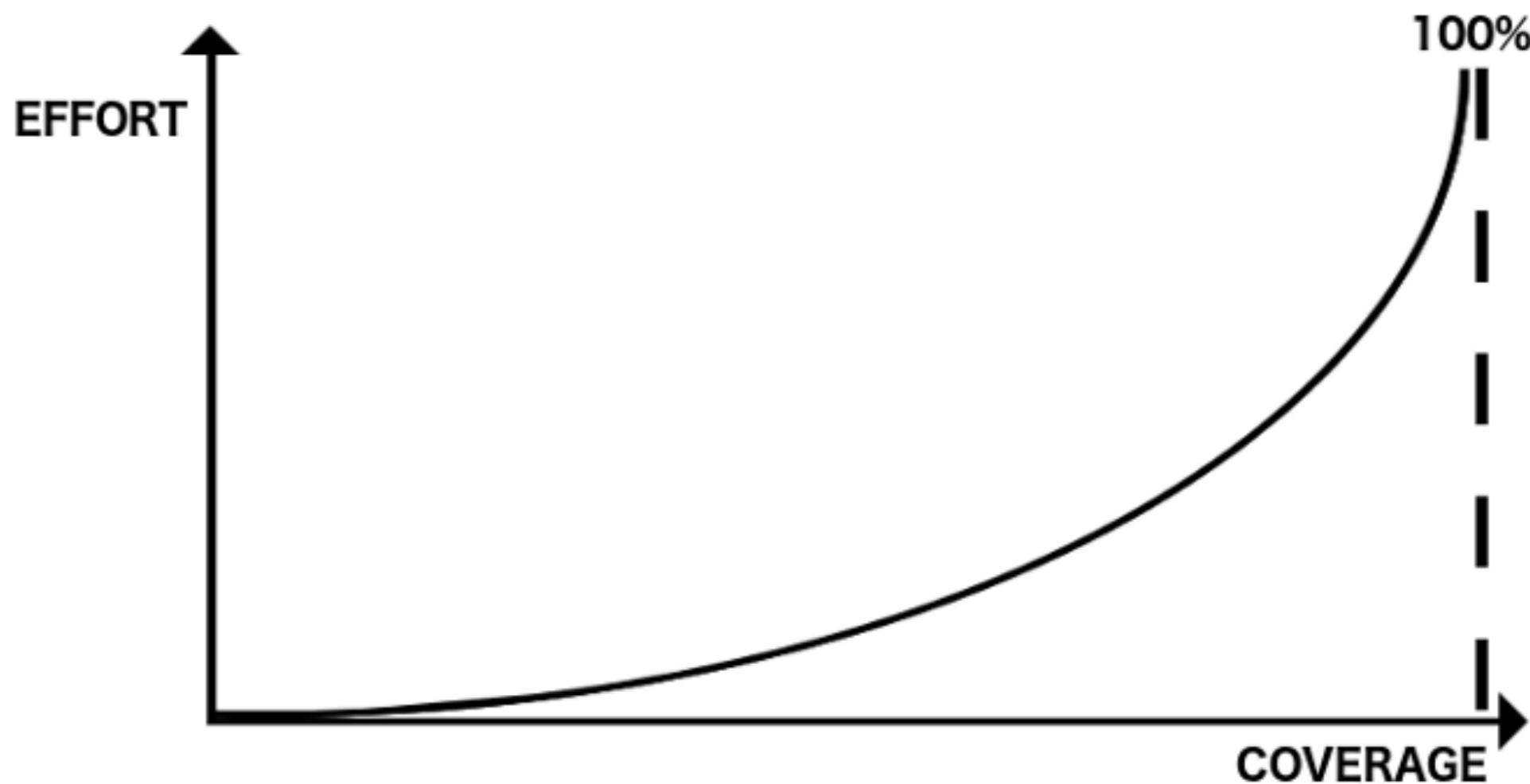
Code coverage

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

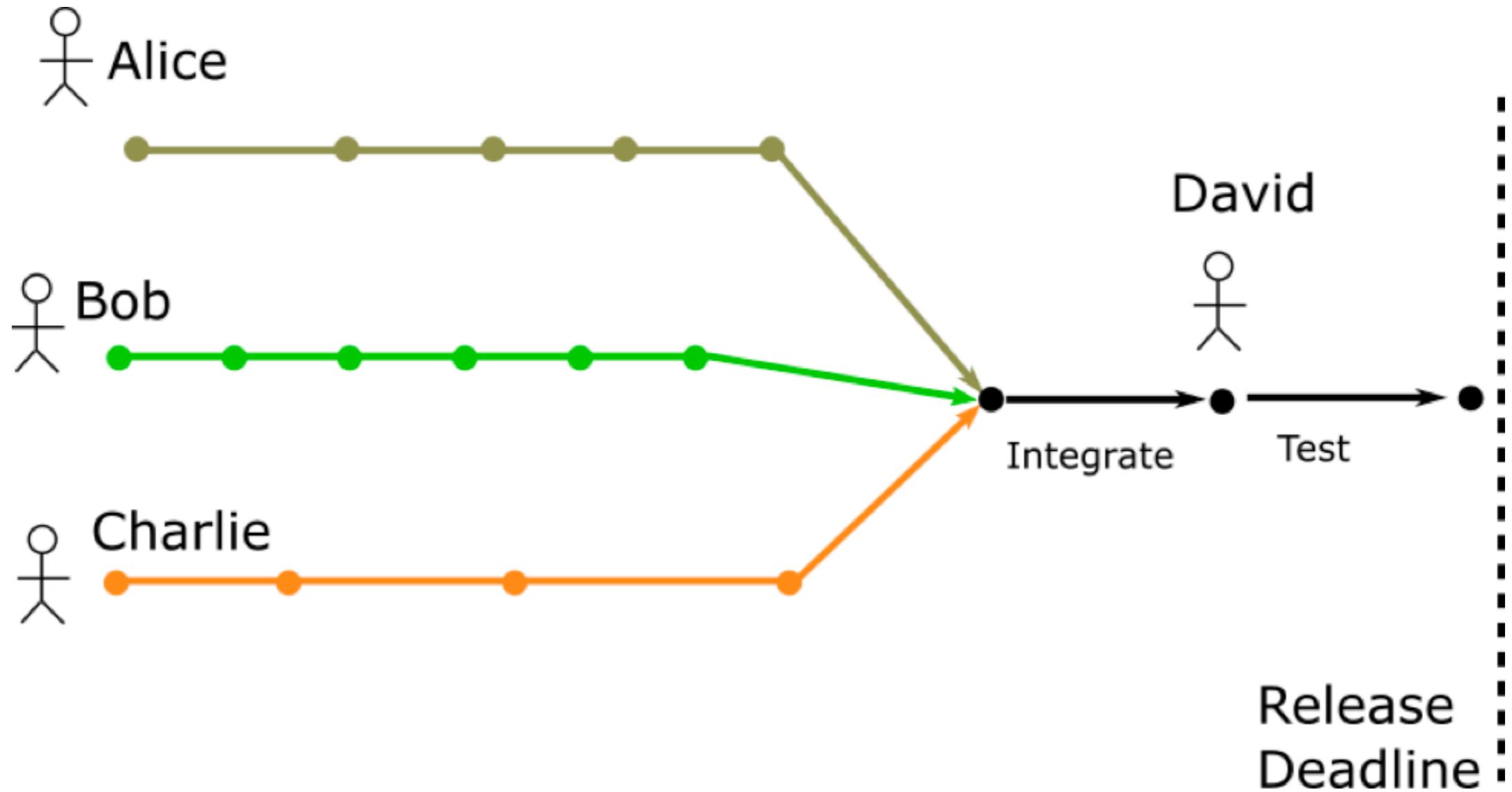
Iteration cycle 2-4 weeks

Continuous integration cycle
(multiple times a day)

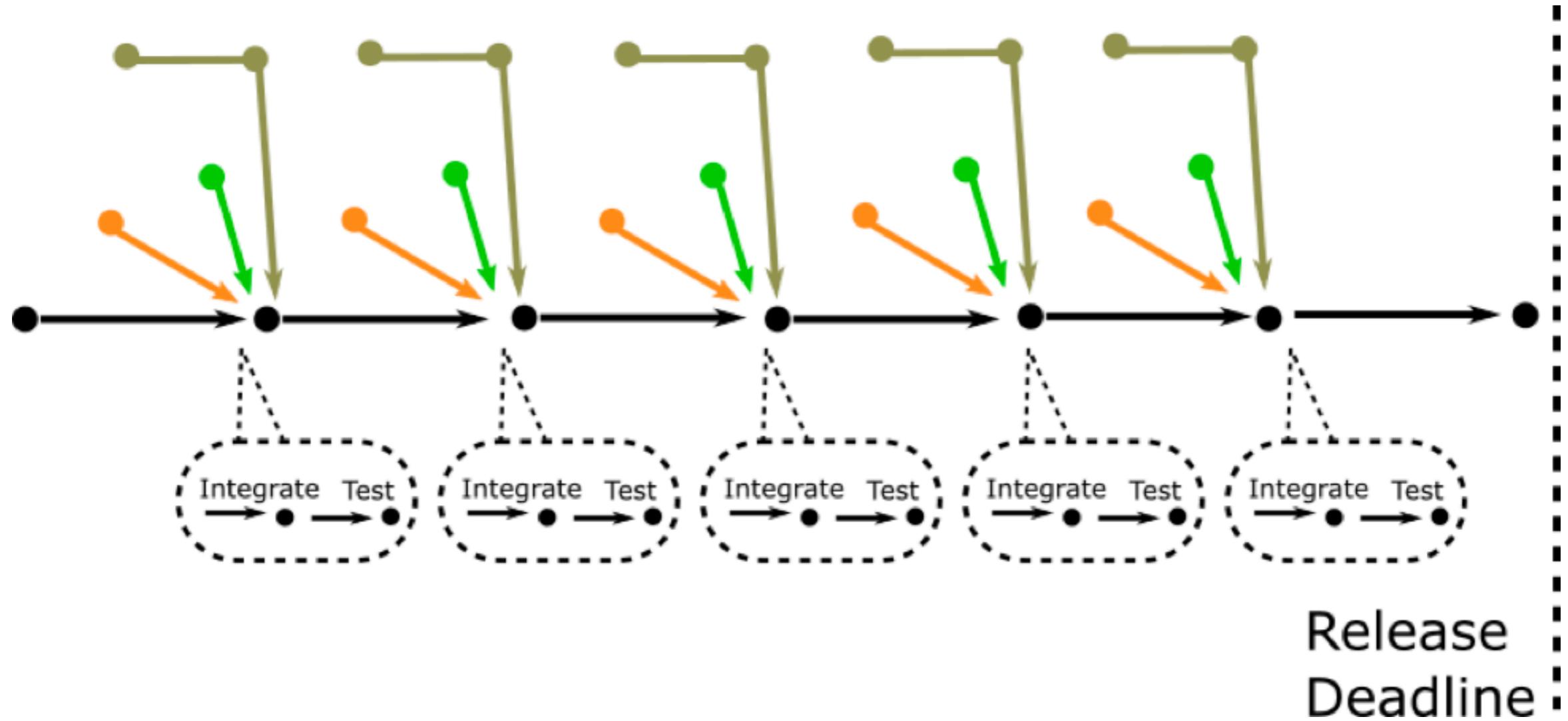
TDD cycle in a few minutes (3-10 minutes)



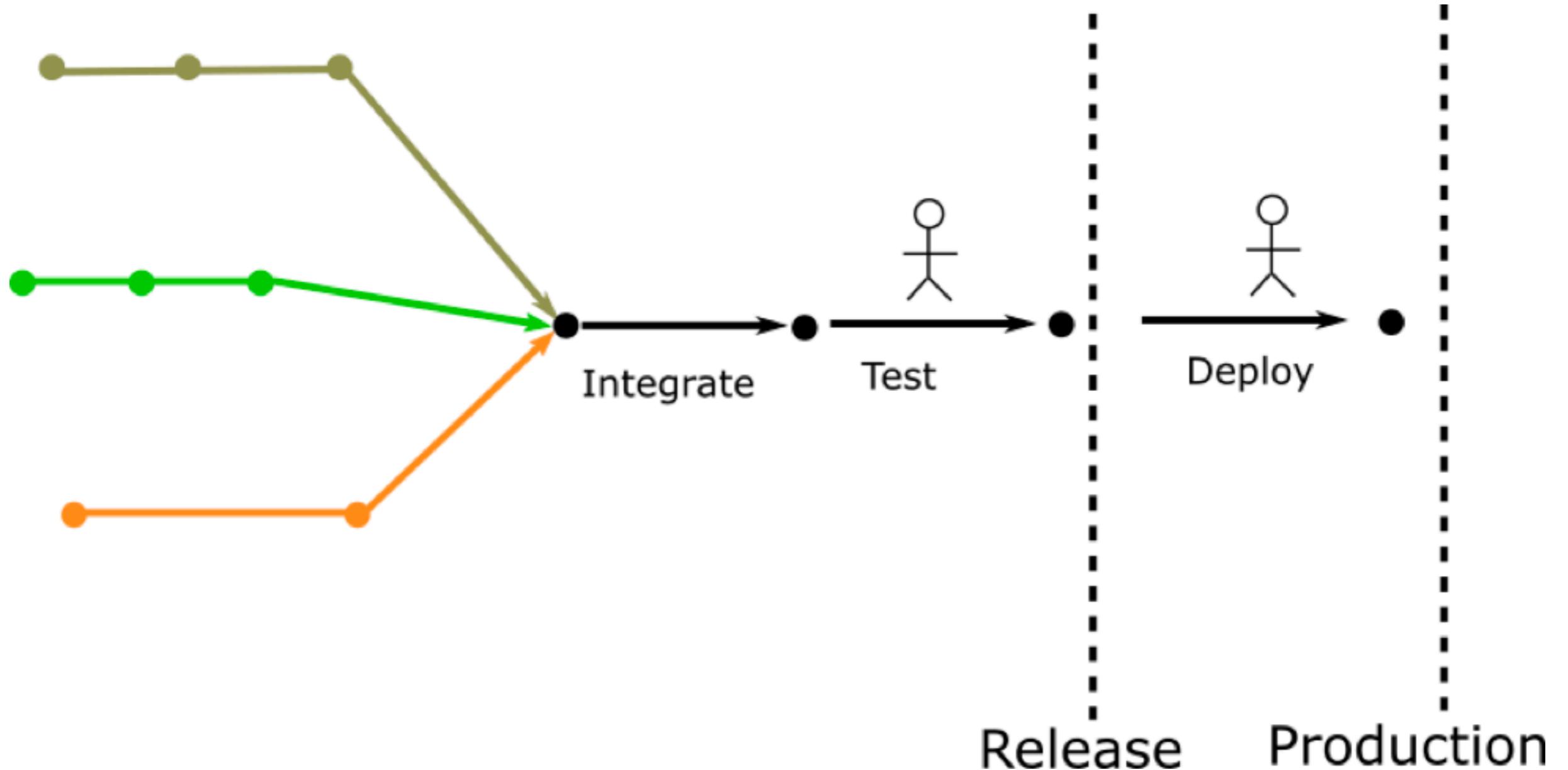
TDD in Iteration development



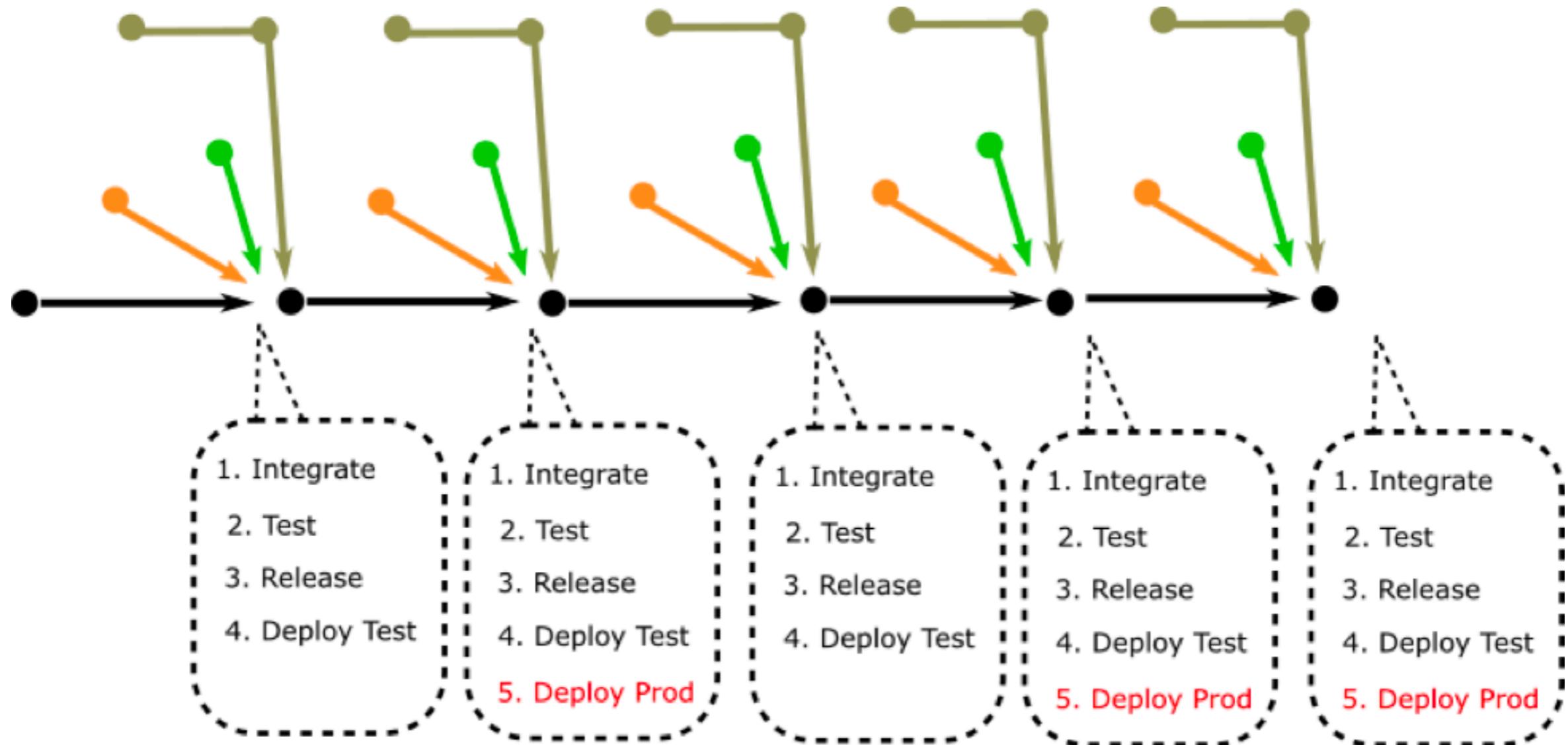
TDD in Iteration development



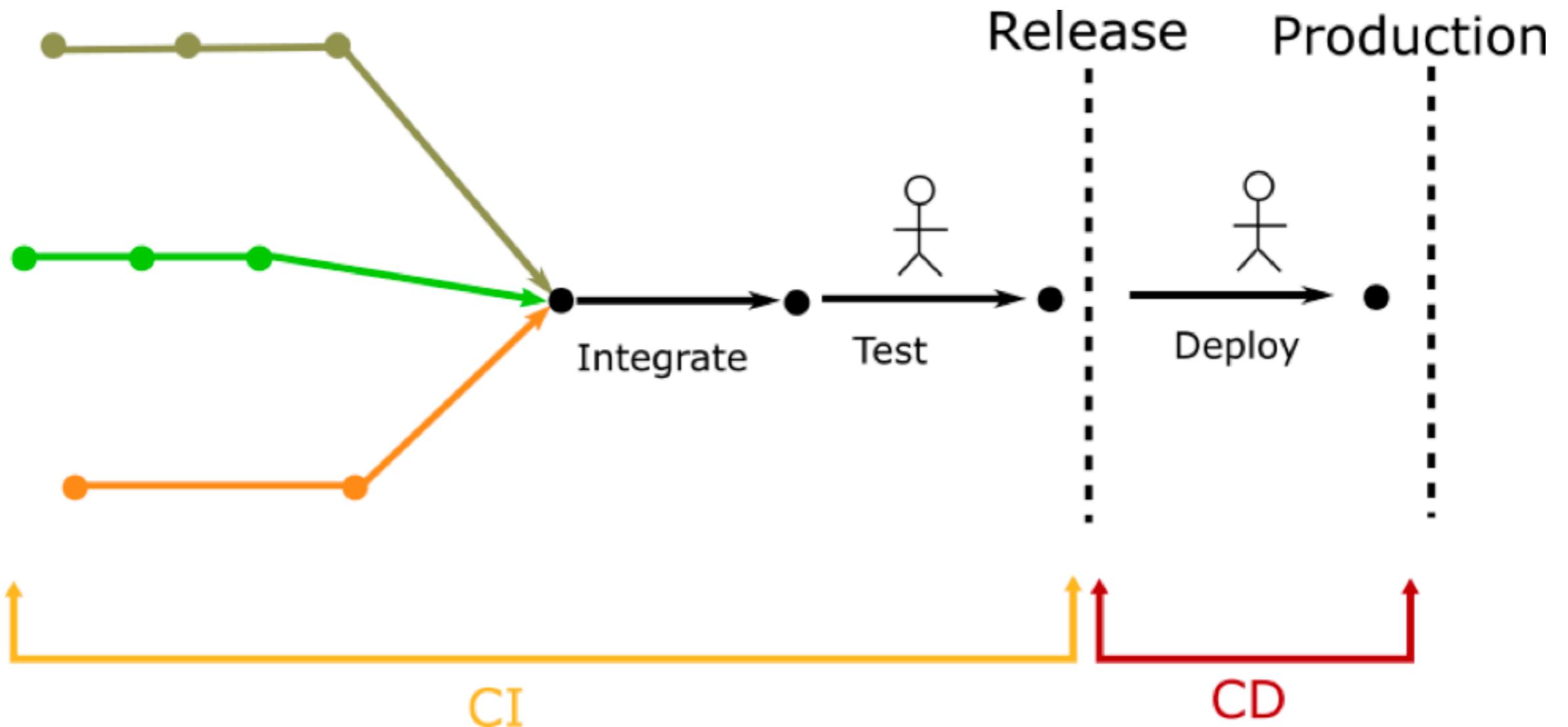
TDD in Iteration development



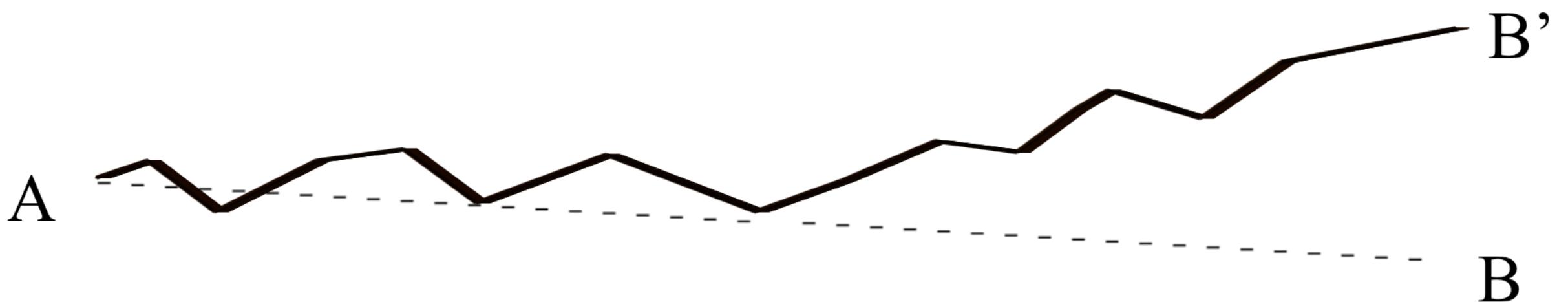
TDD in Iteration development



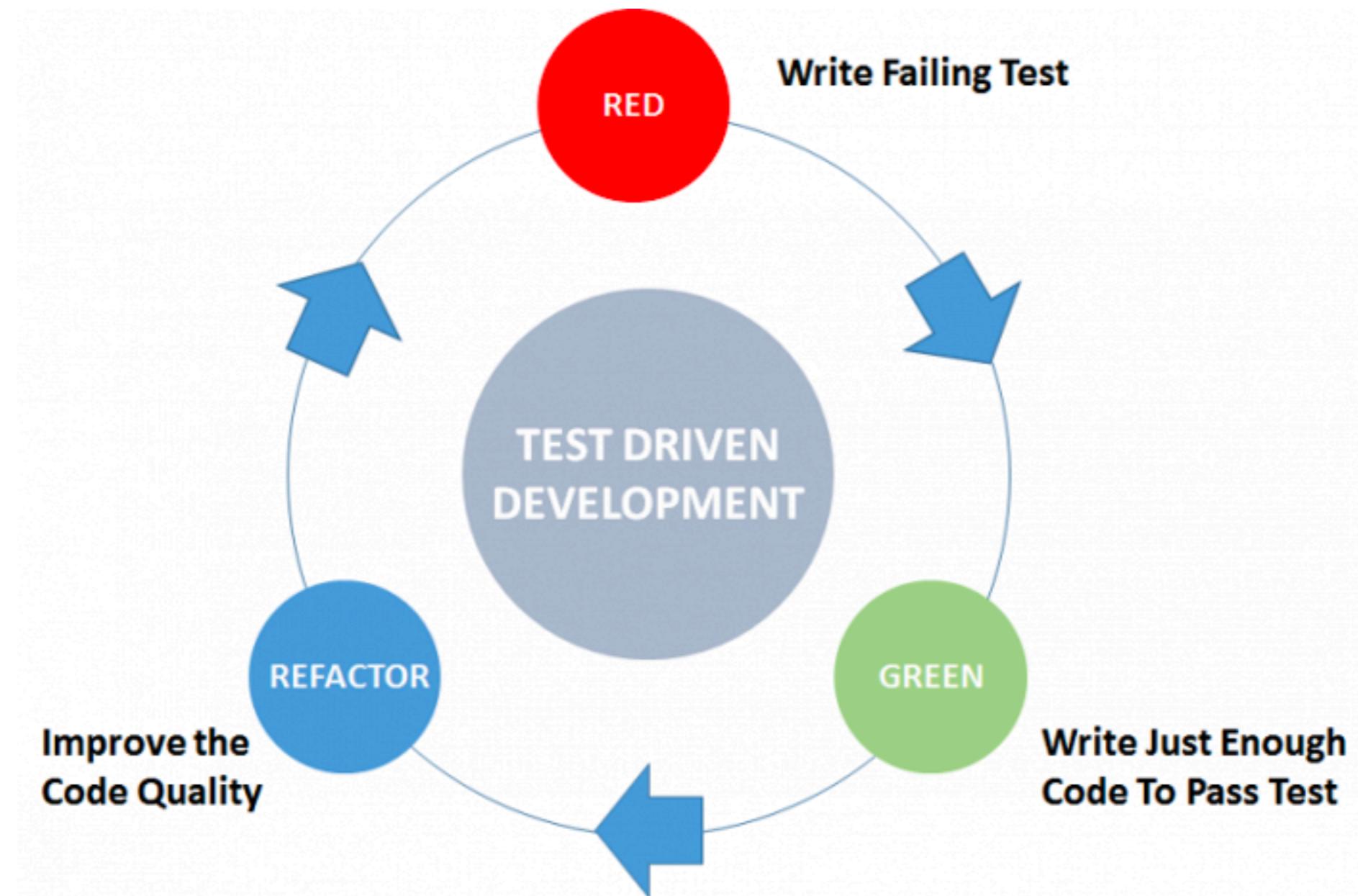
TDD in Iteration development



Small steps



TDD = ทำดีๆ



Types of Test





THE TESTING Manifesto

we value:

Testing throughout
OVER
testing at the end

Preventing bugs
OVER
finding bugs

Testing understanding
OVER
checking functionality

Building the best system
OVER
breaking the system

Team responsibility for quality
OVER
tester responsibility

www.GrowingAgile.co.za

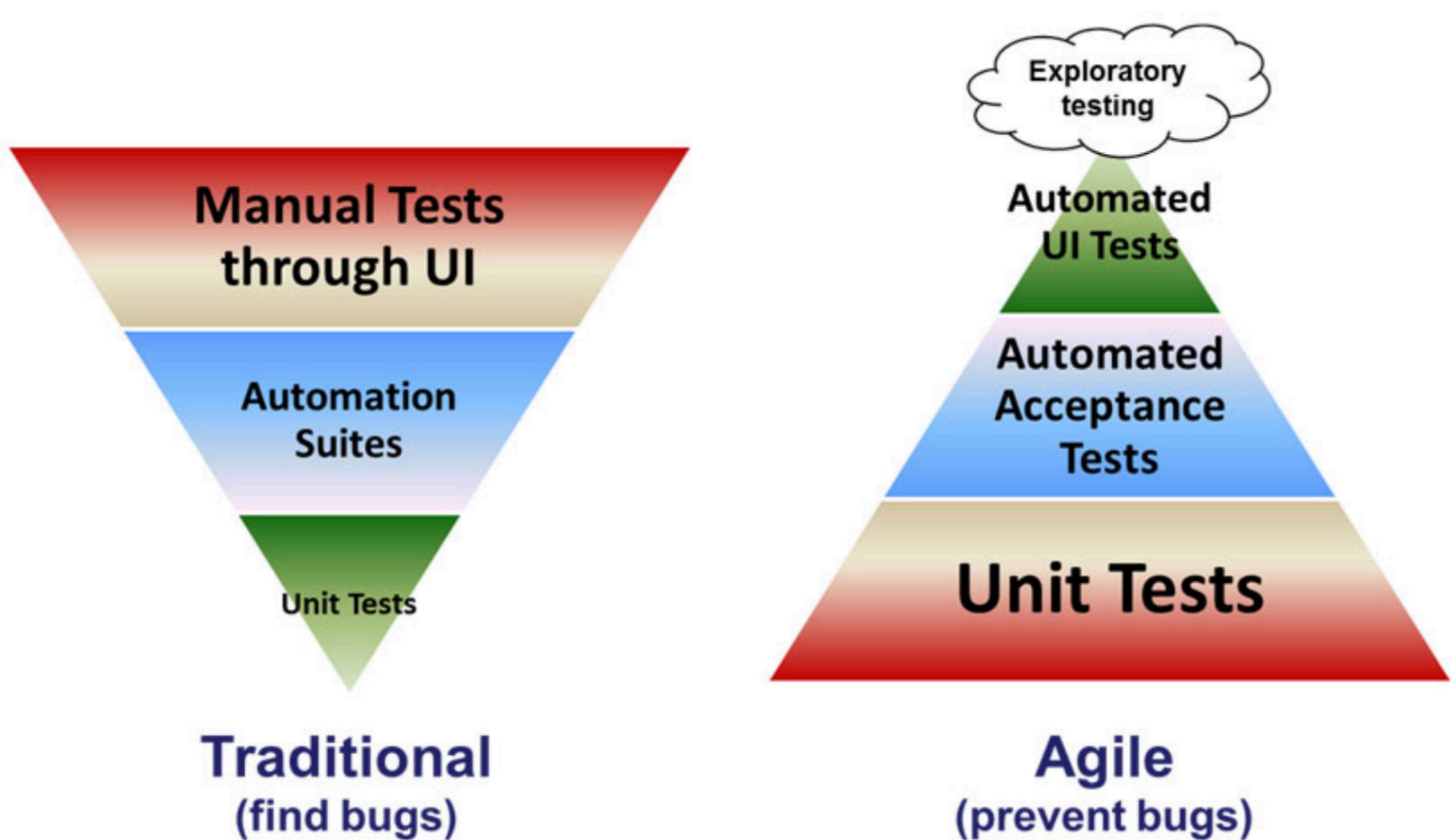
@growingAgile

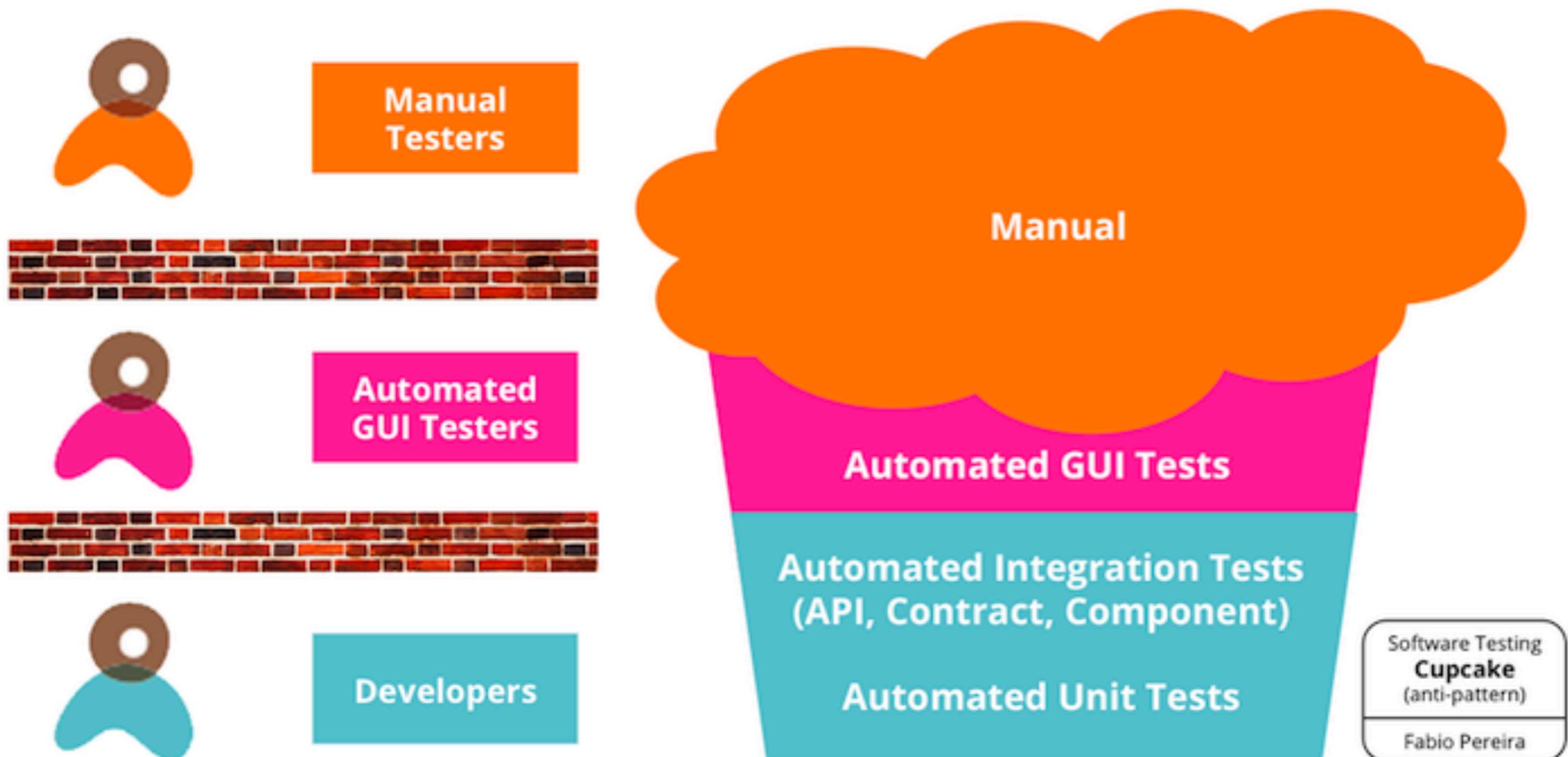




Testing is an ACTIVITY!







<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.

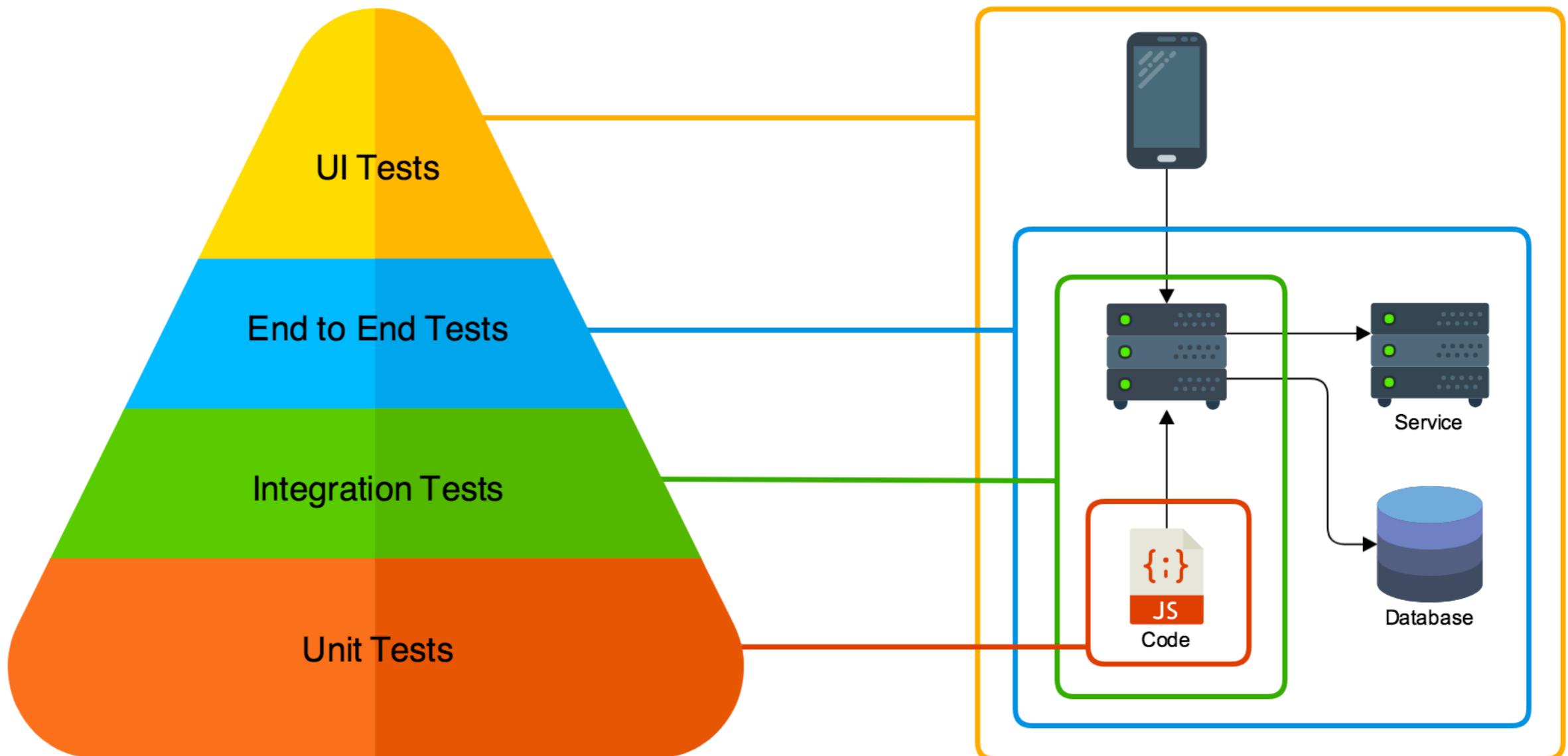
Static

Catch typos and type errors as you write the code.

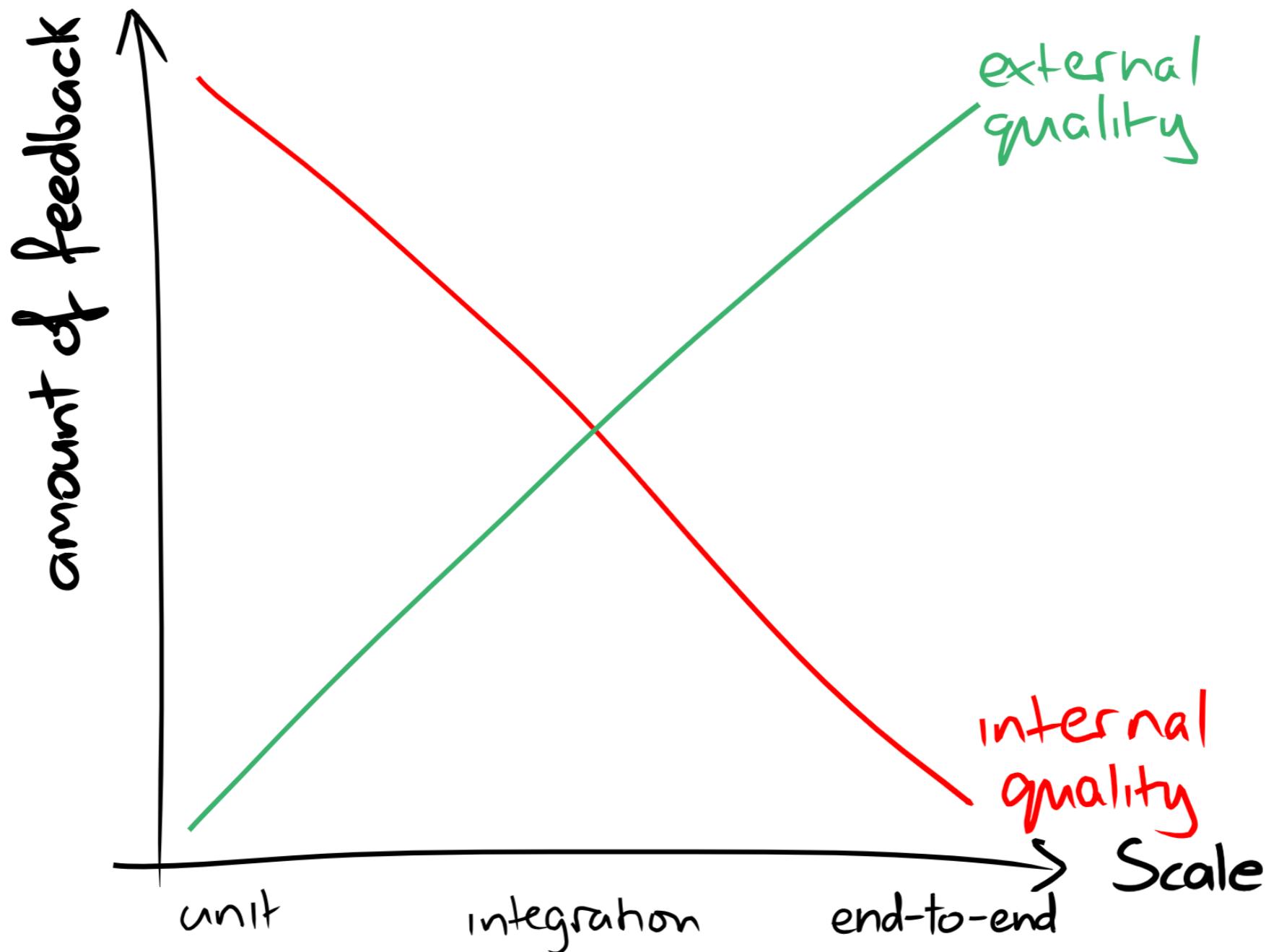


<https://testingjavascript.com/>

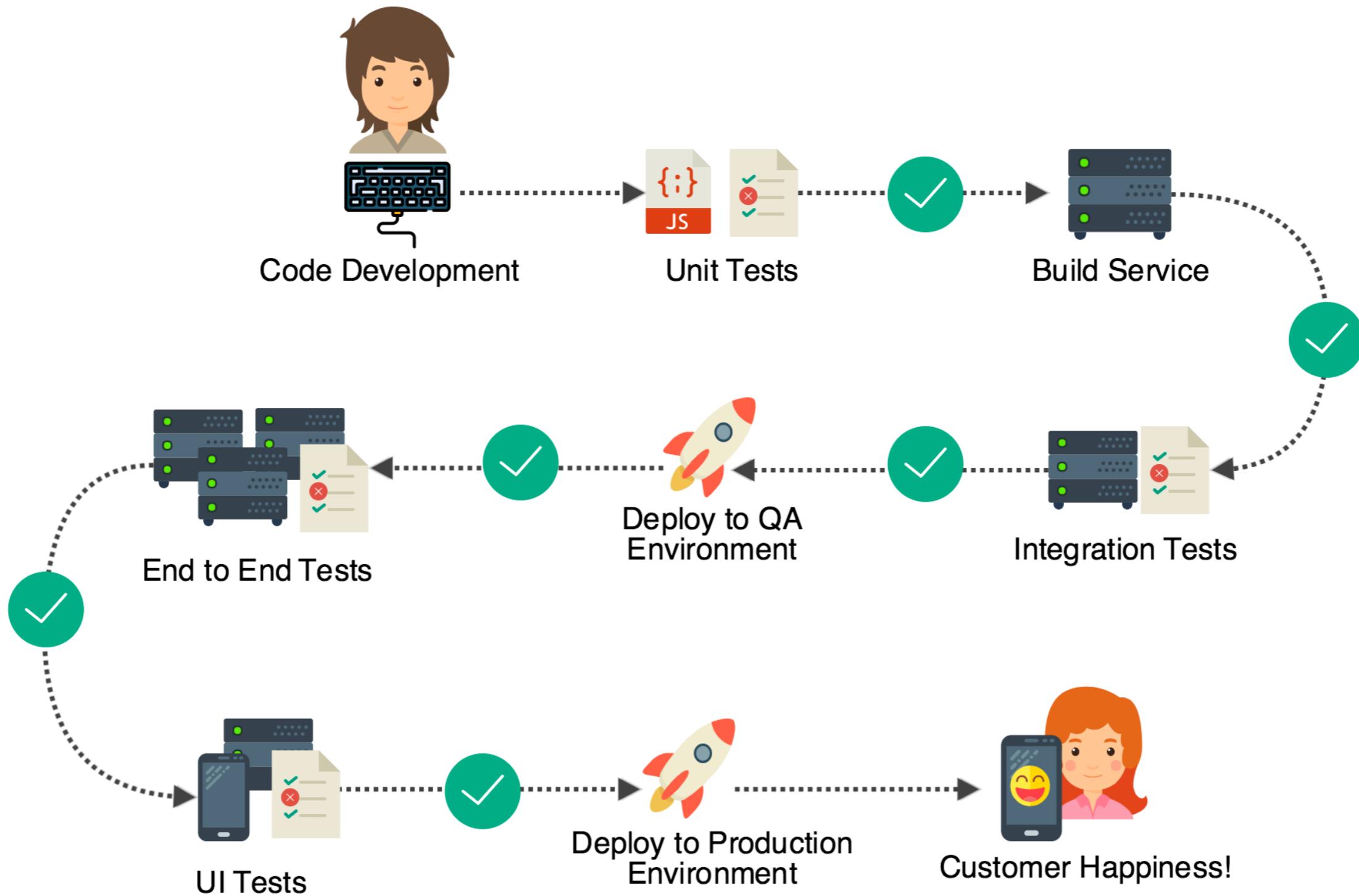




Internal and External quality



Delivery process



More with Microservices Testing

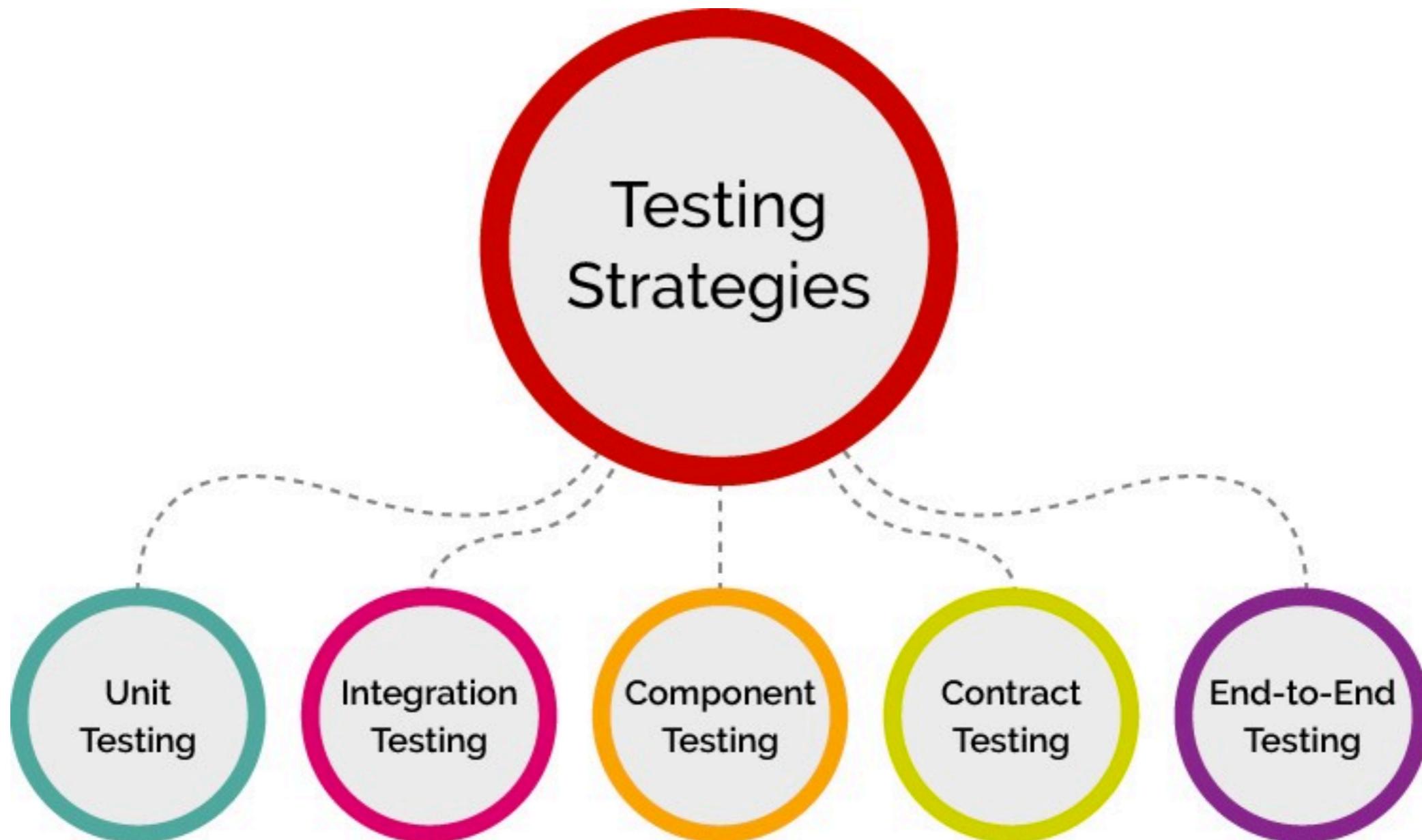
<https://martinfowler.com/articles/microservice-testing/>



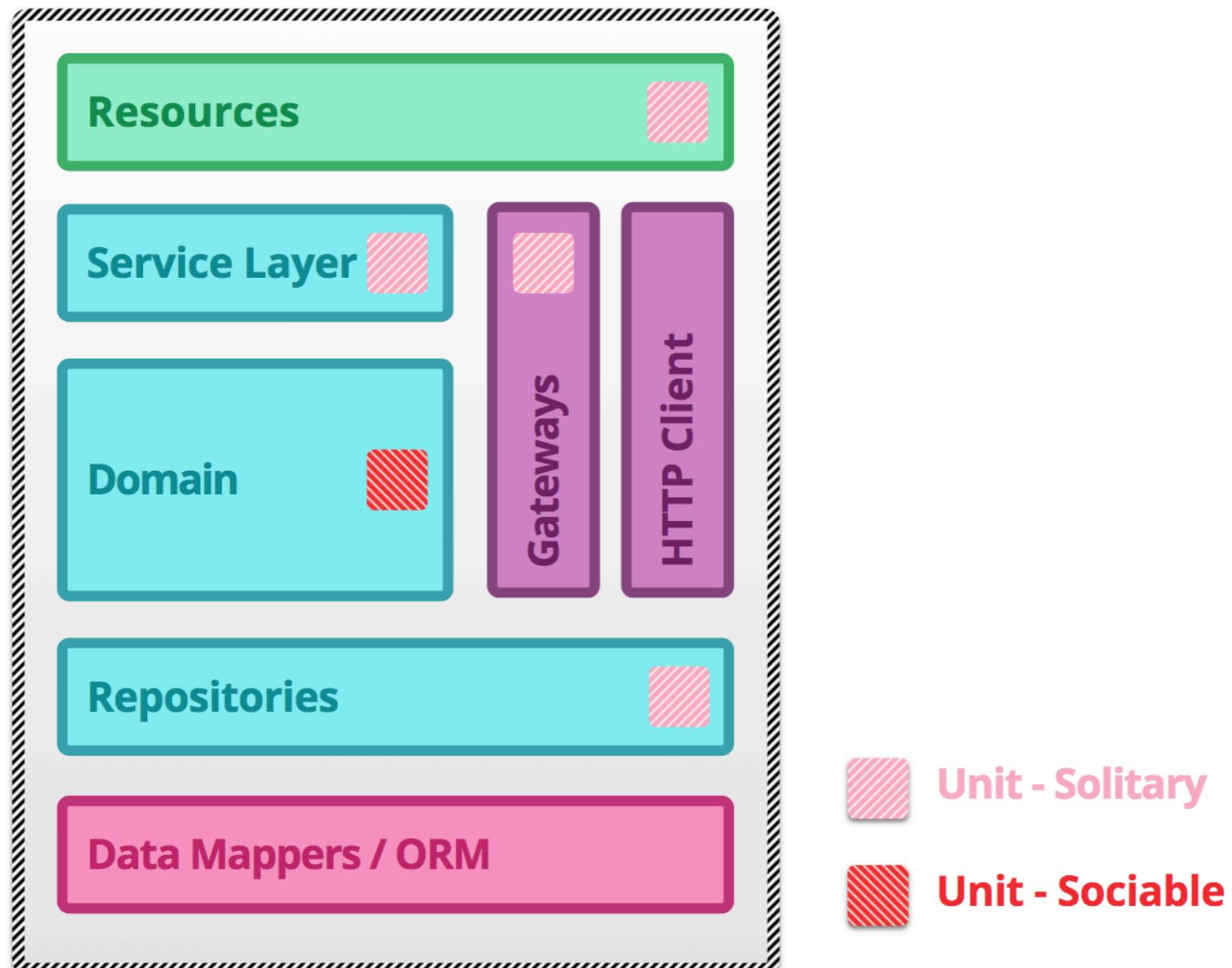
Testing strategies

Unit
Integration
Component
Contract
End-to-End

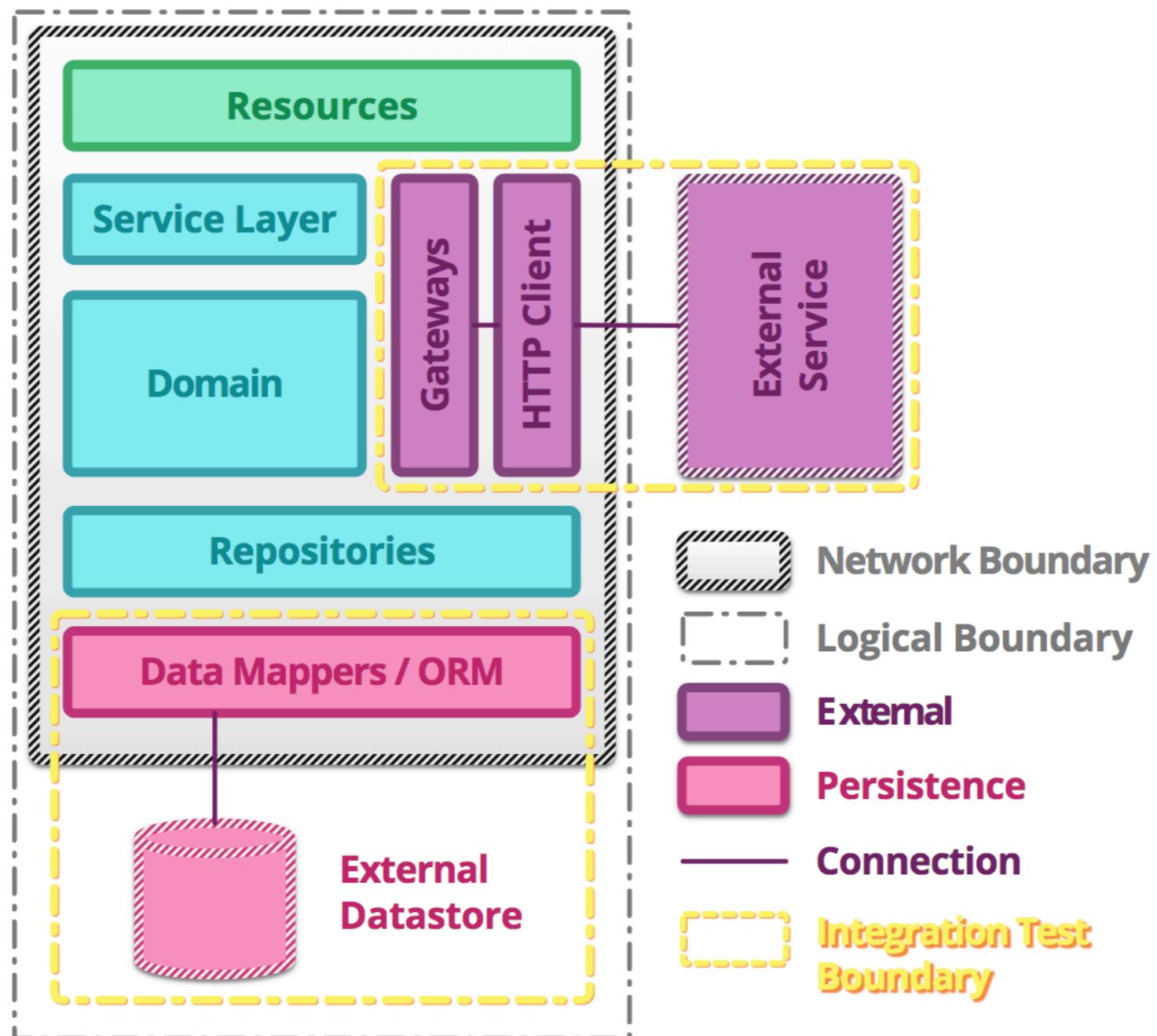




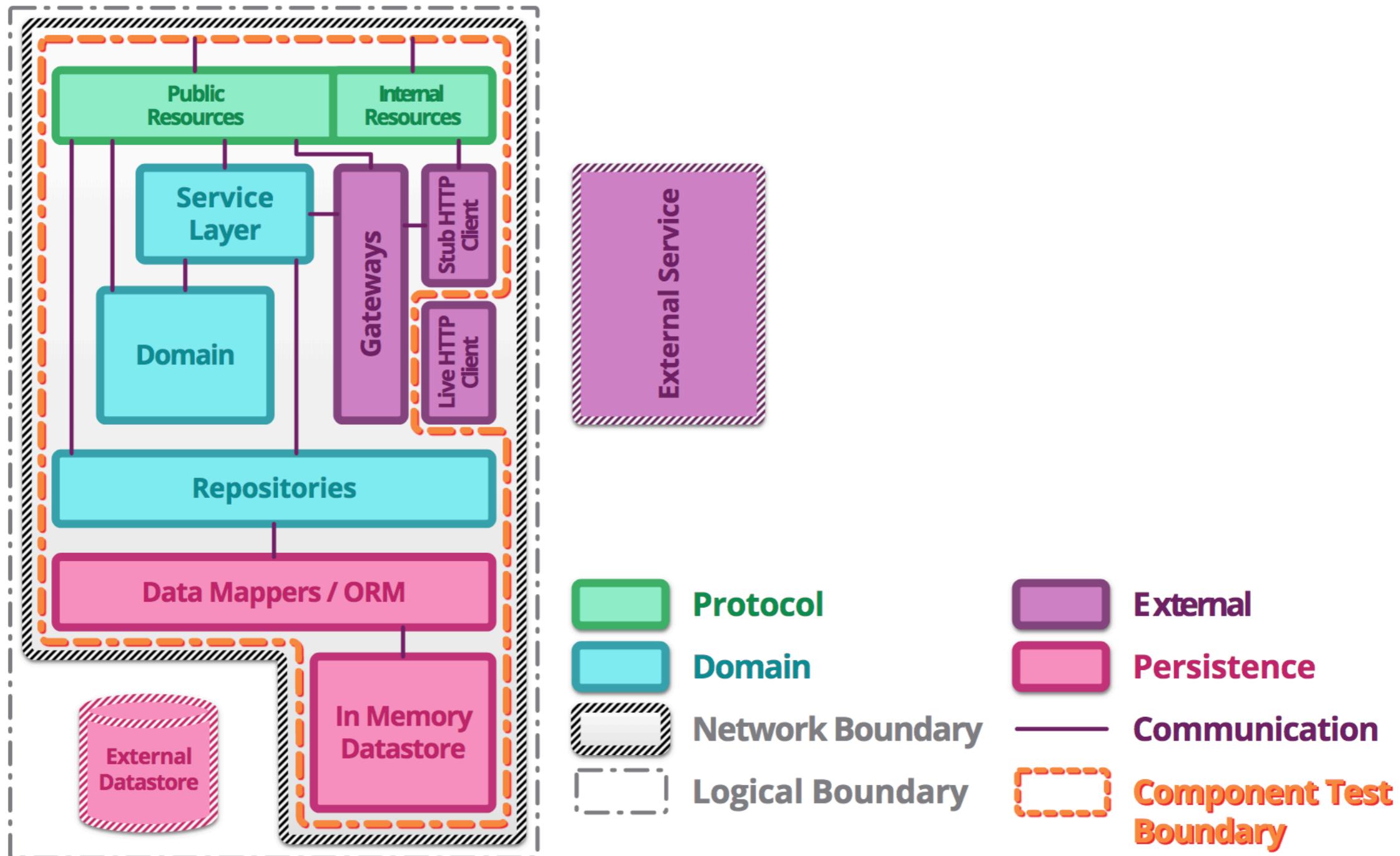
Unit testing



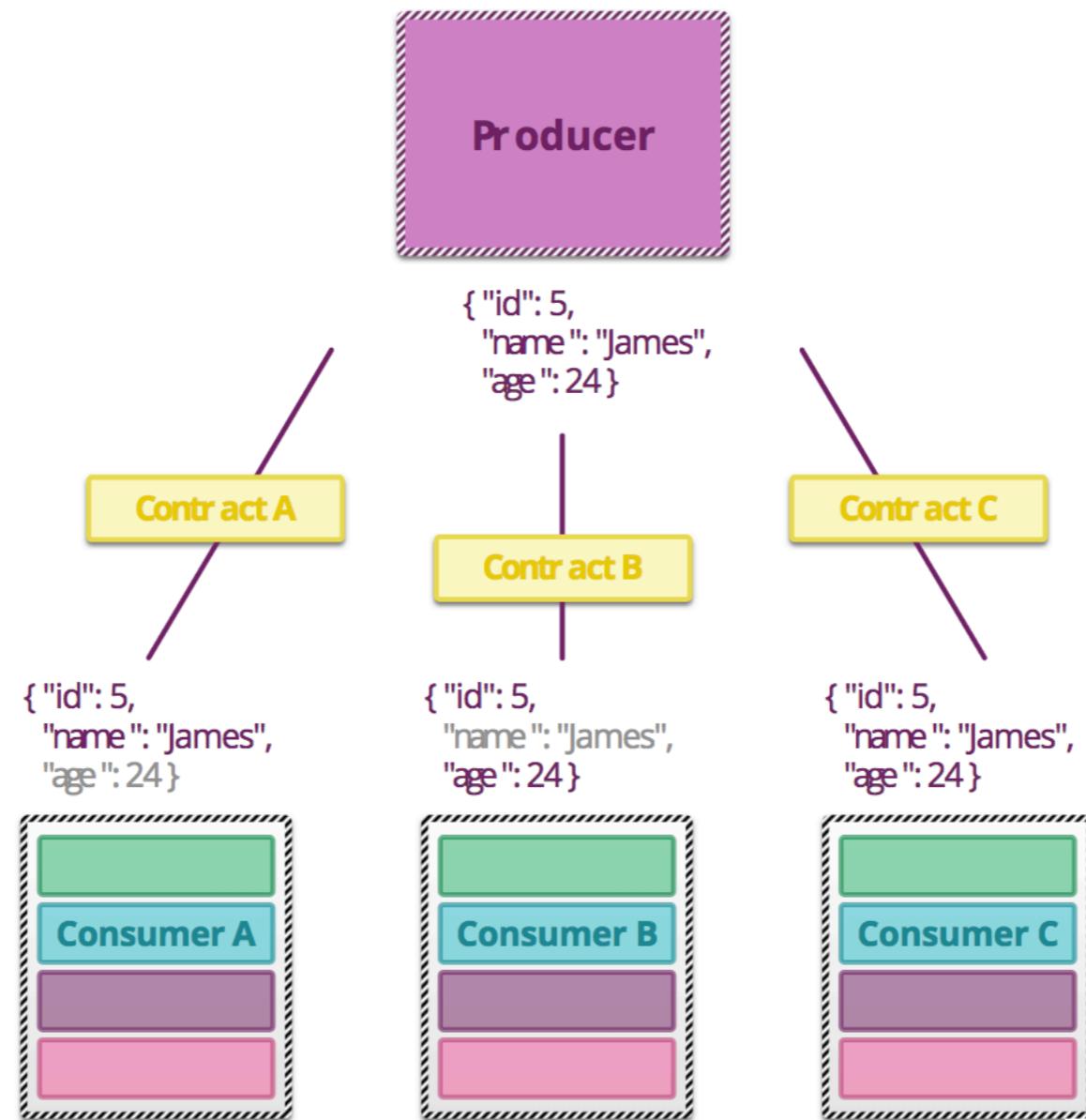
Integration testing



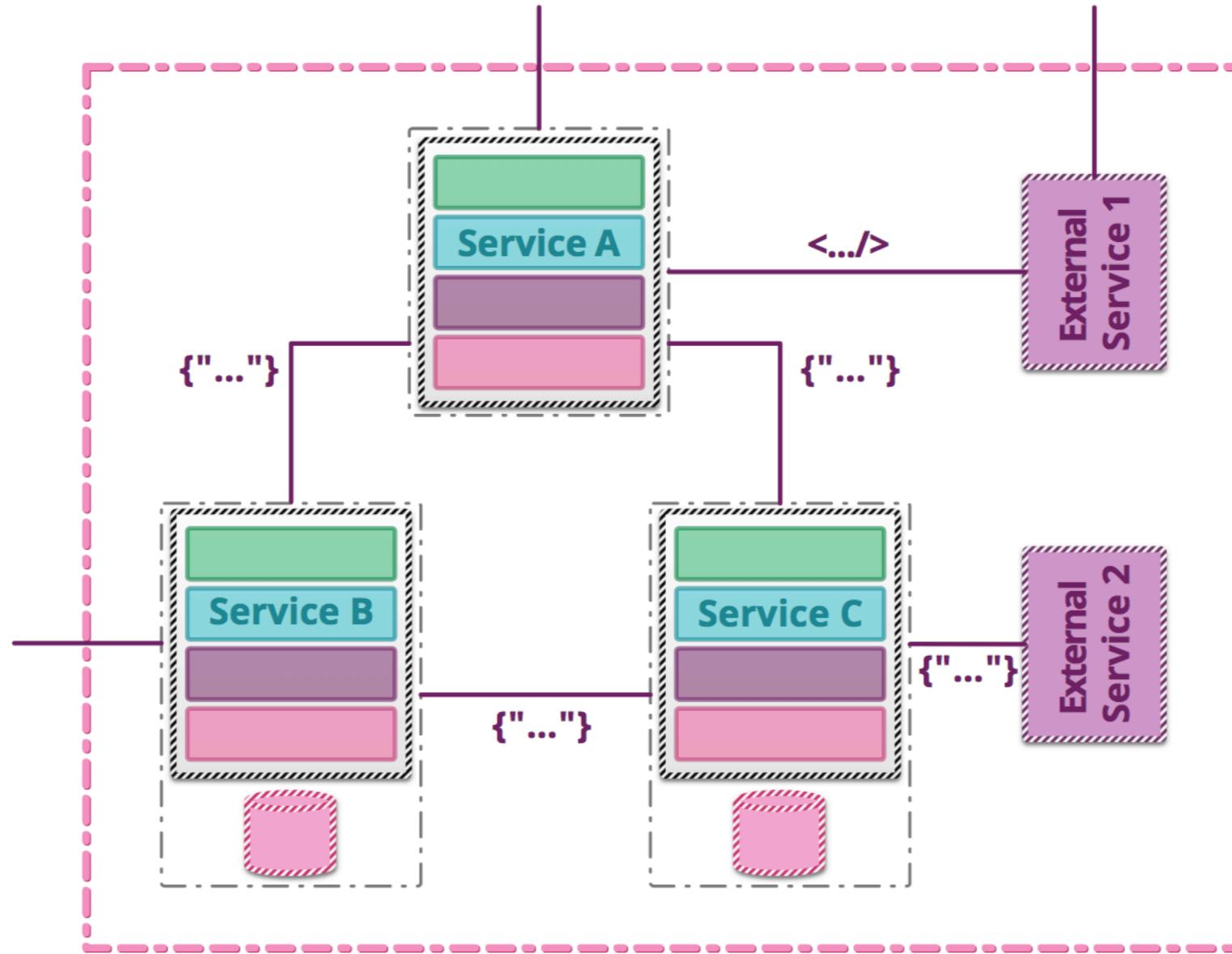
Component testing



Contract testing



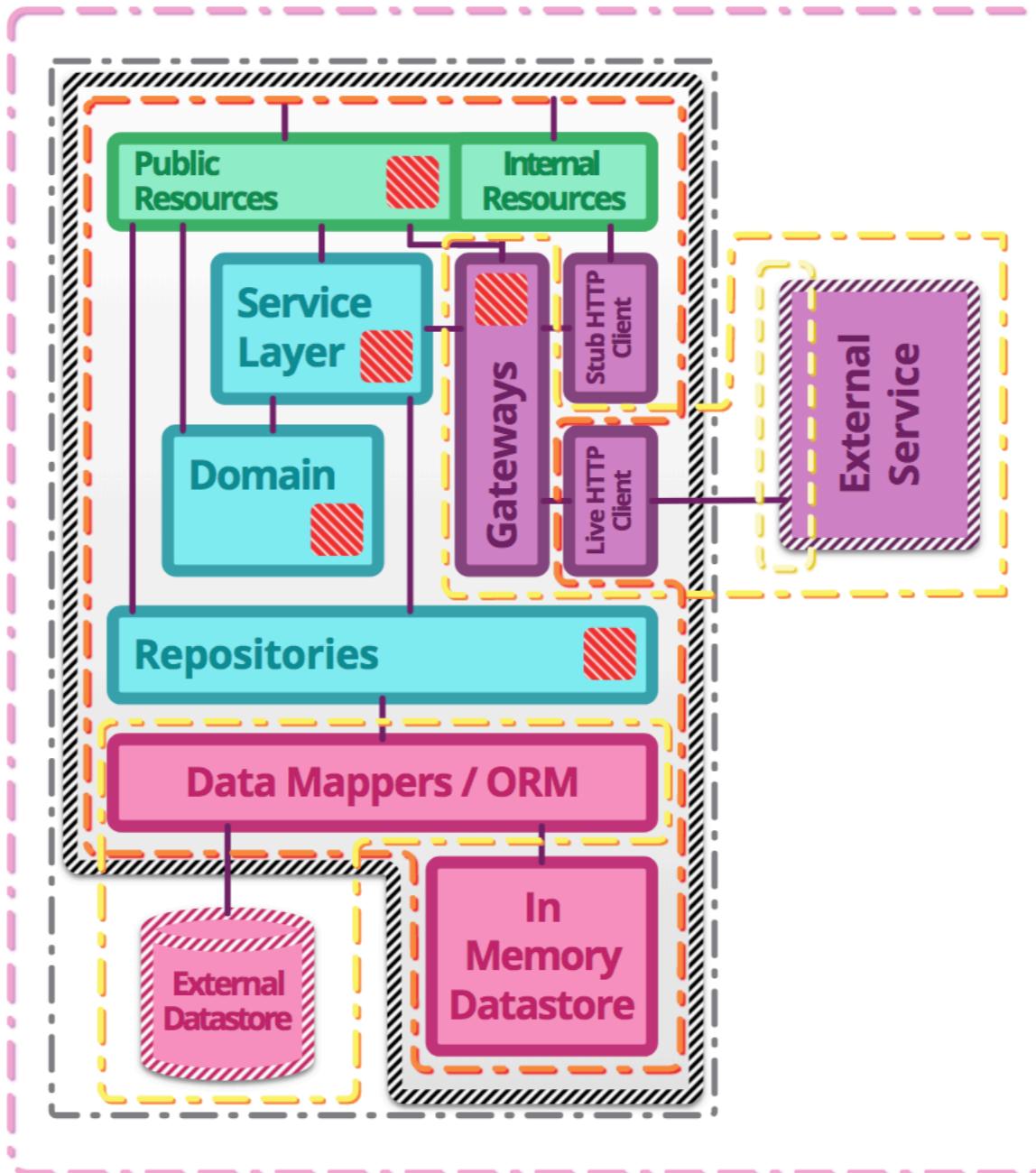
End-to-End testing



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.



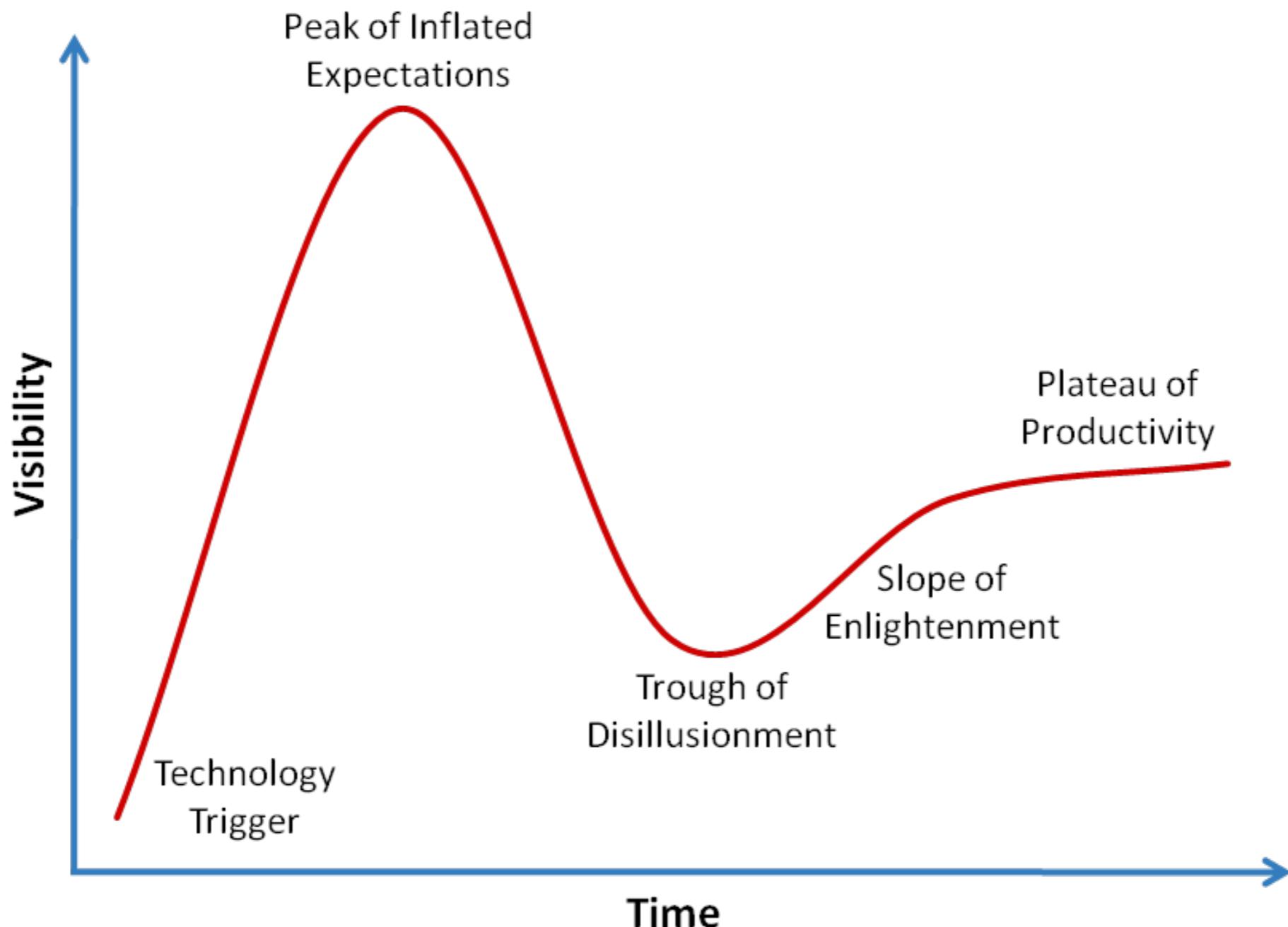
 **Component tests** : limit the scope of the exercised software to a portion of the system under test, manipulating the system through internal code interfaces and using test doubles to isolate the code under test from other components.

 **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/>



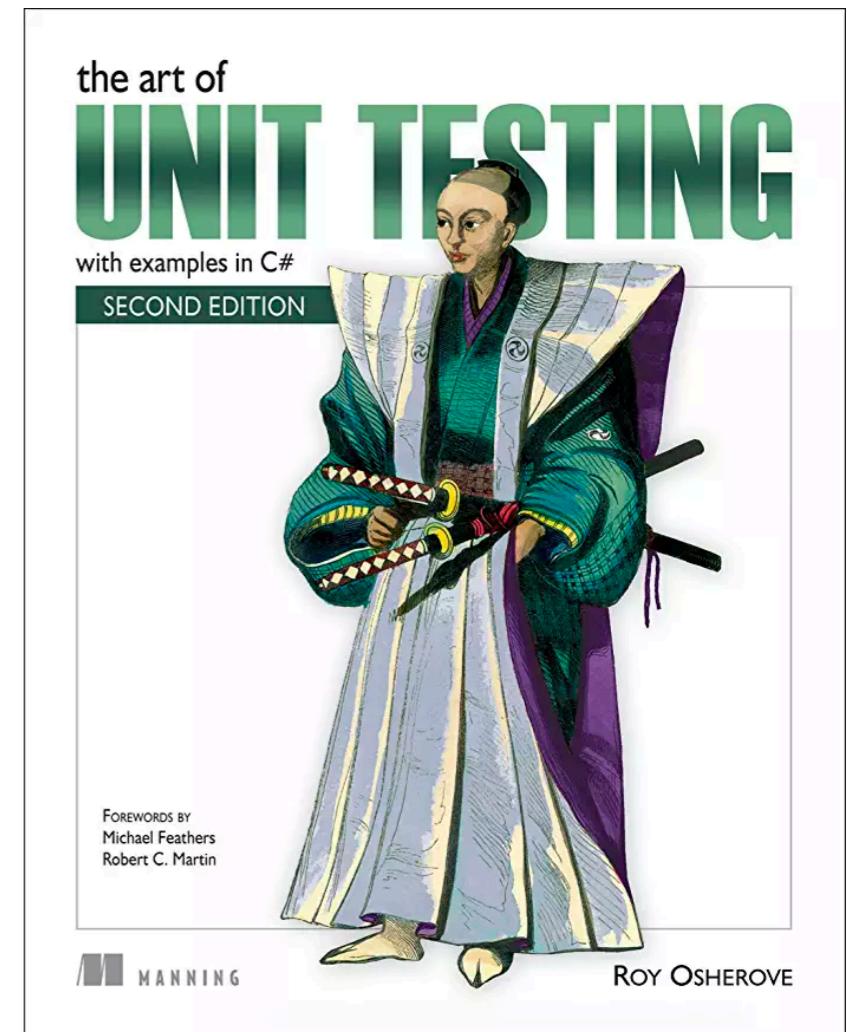
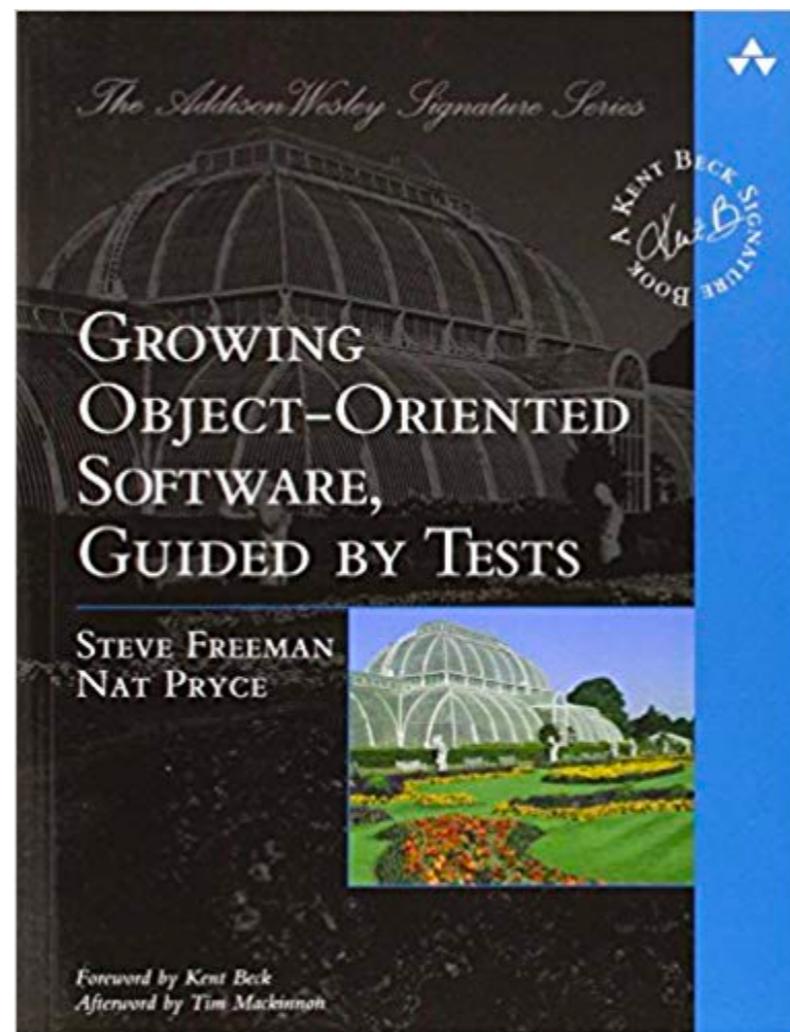
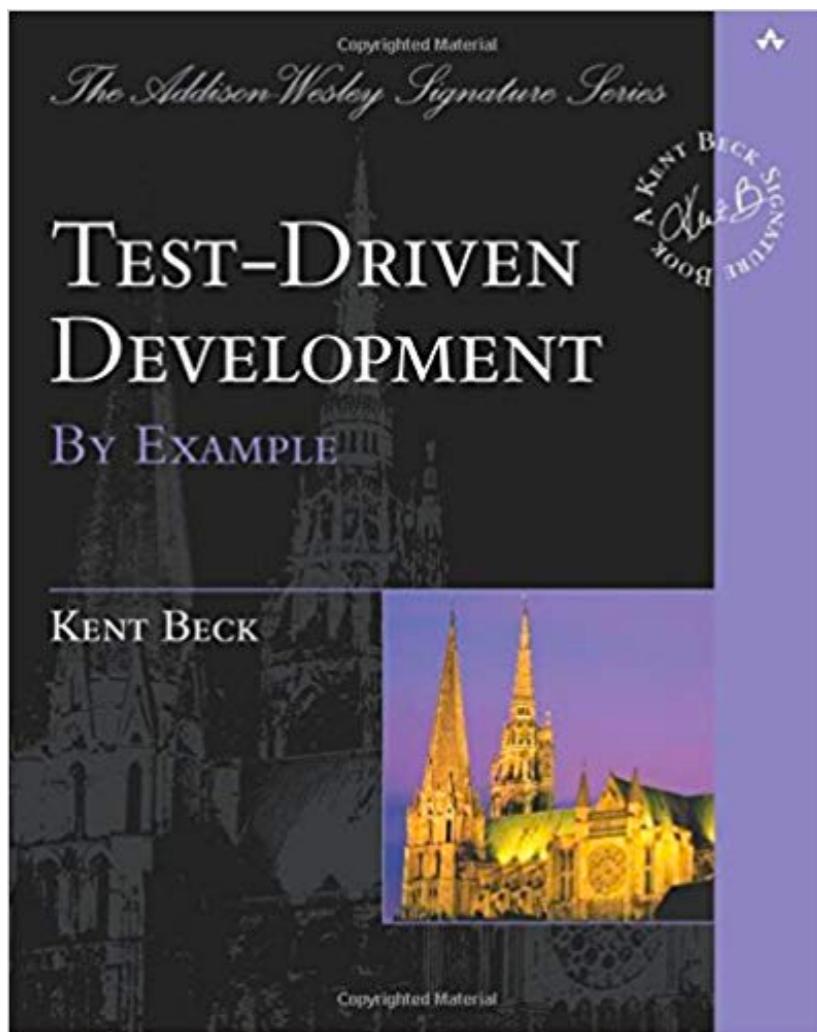
Don't forgot Refactoring



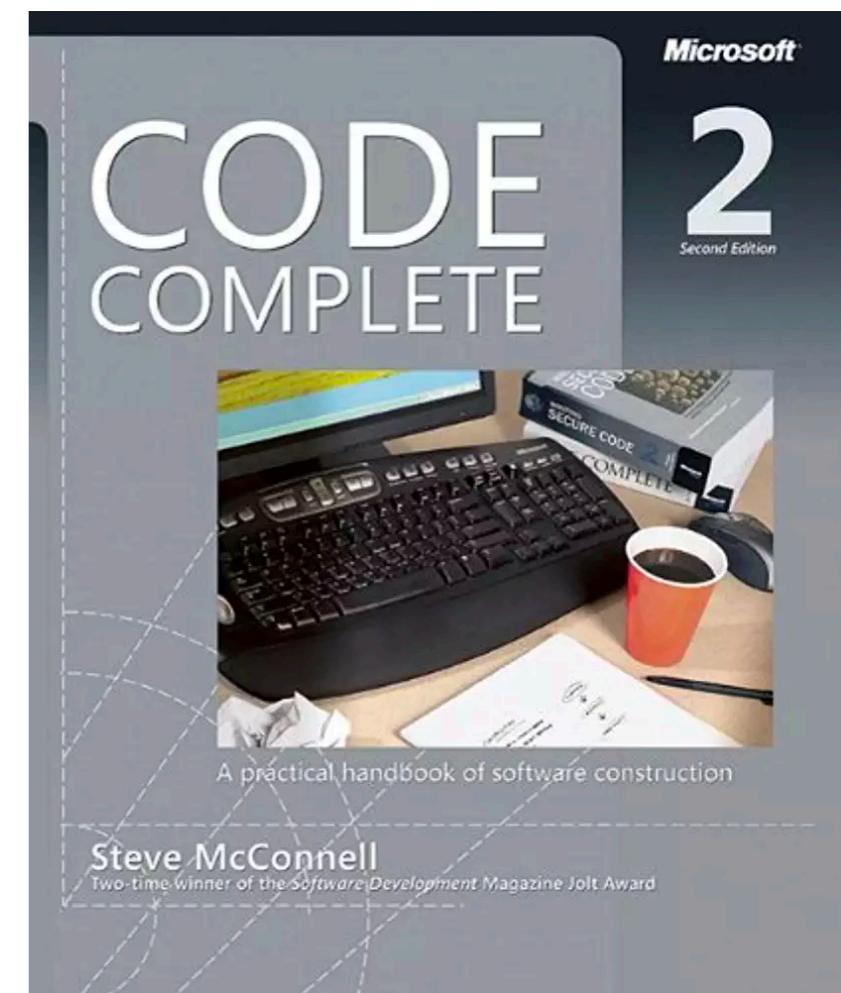
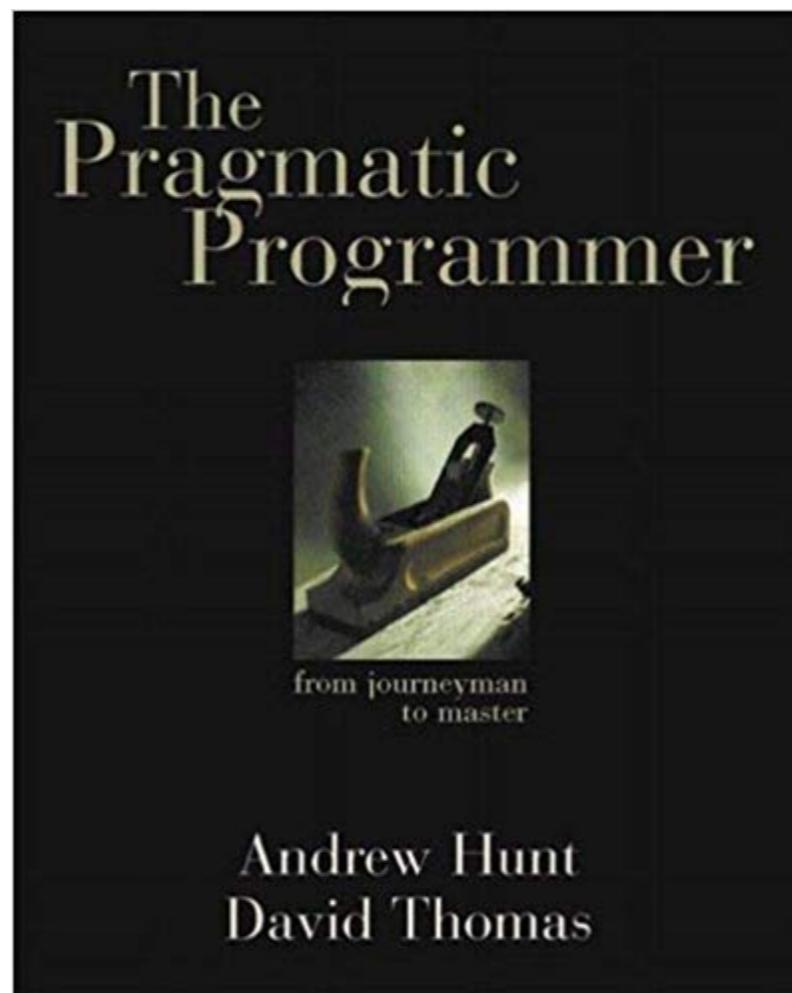
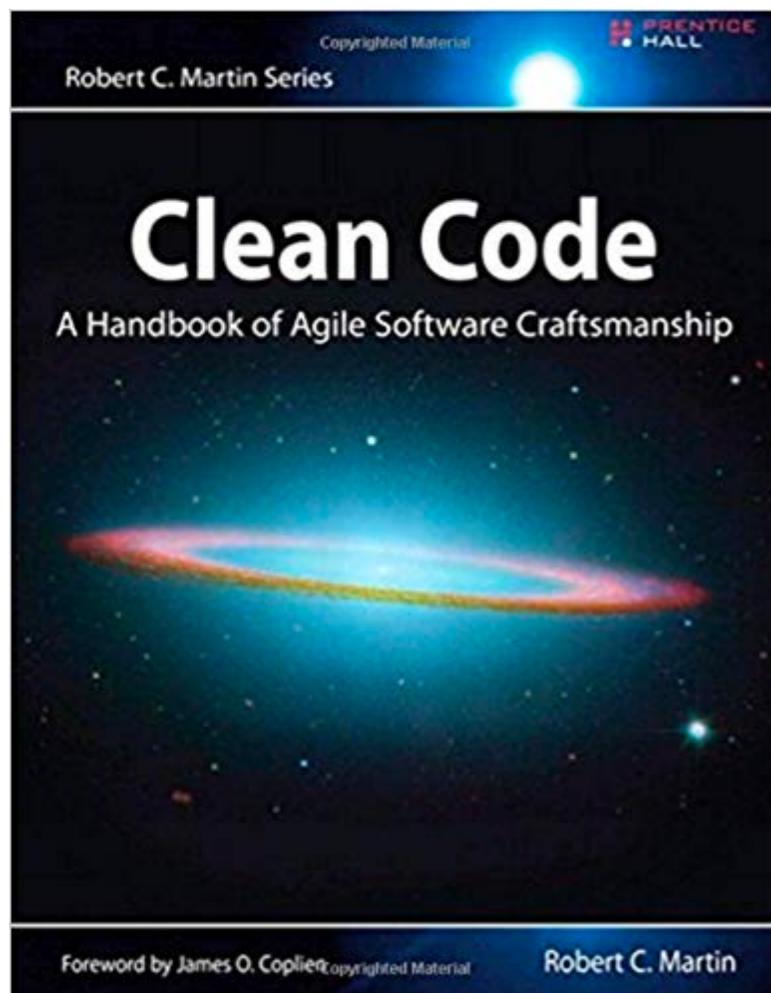
Code Smell



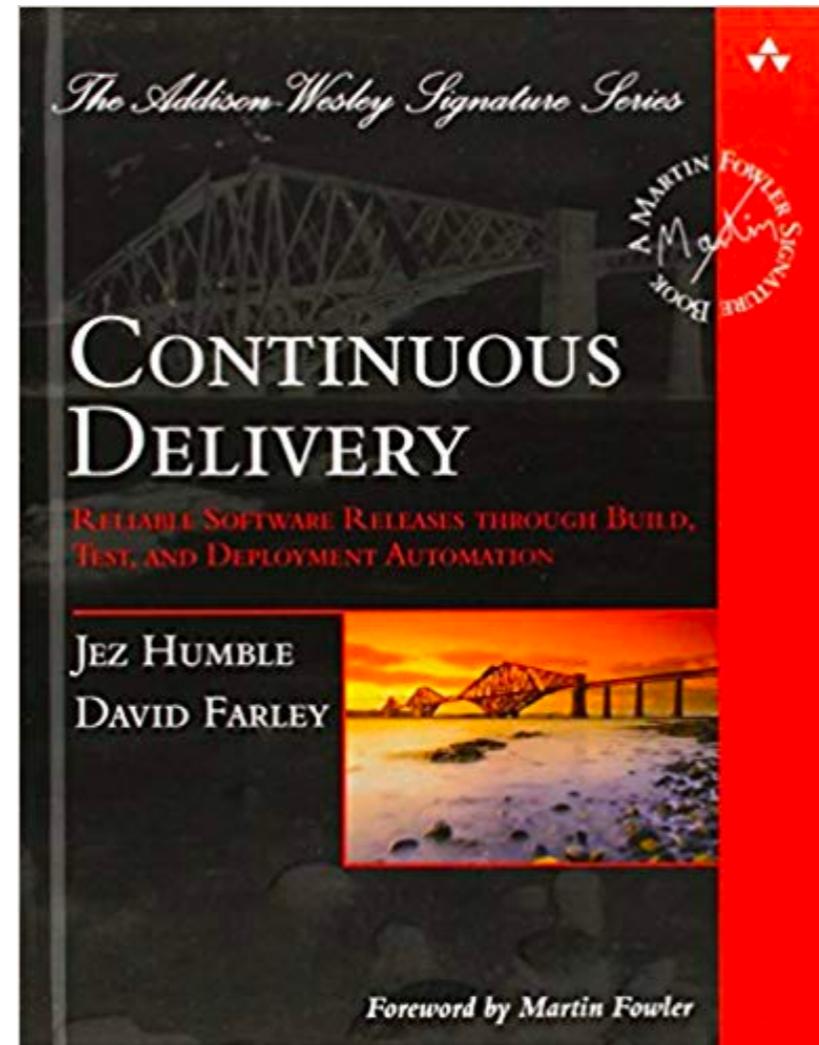
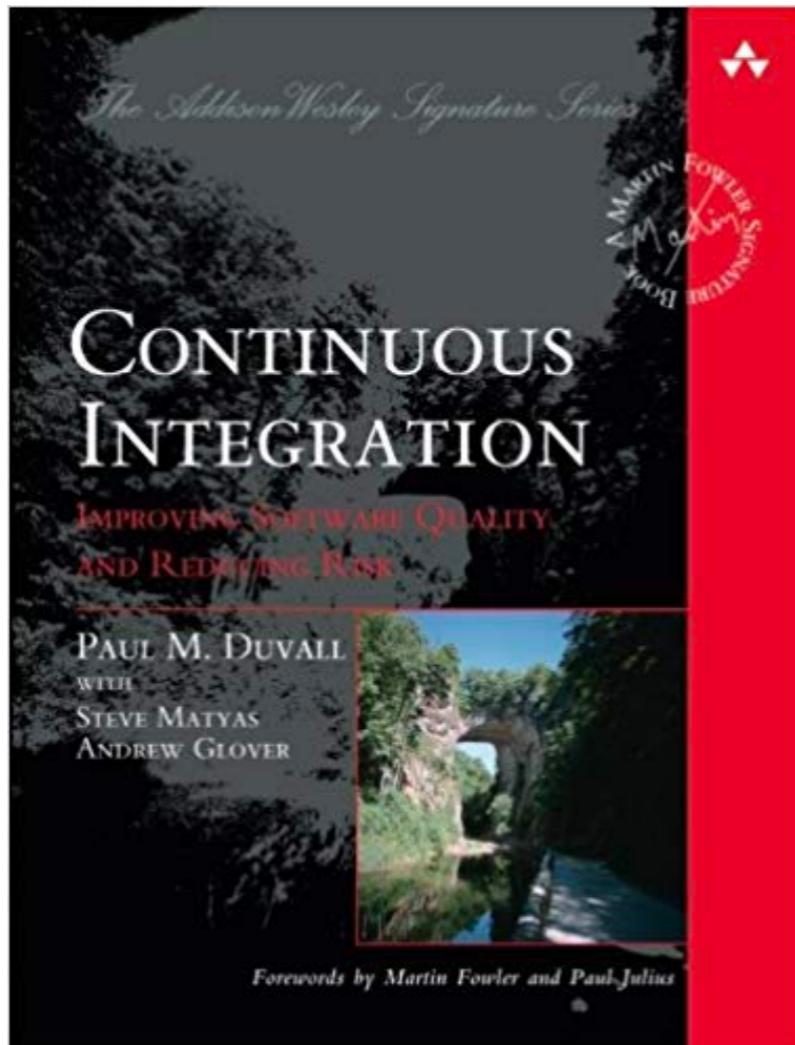
Books



Books



Books





SPECIFICATION BY EXAMPLE



CONTINUOUS
INTEGRATION



CONTINUOUS DELIVERY



TEST AUTOMATION



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>





SPECIFICATION BY EXAMPLE



TEST AUTOMATION



THINKING ABOUT TESTING



CONTINUOUS
INTEGRATION



TECHNICAL
EXCELLENCE



TEST-DRIVEN DEVELOPMENT



CONTINUOUS DELIVERY



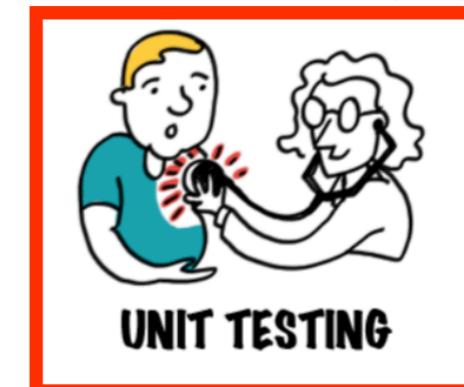
ARCHITECTURE
& DESIGN



ACCEPTANCE
TESTING



CLEAN CODE



UNIT TESTING

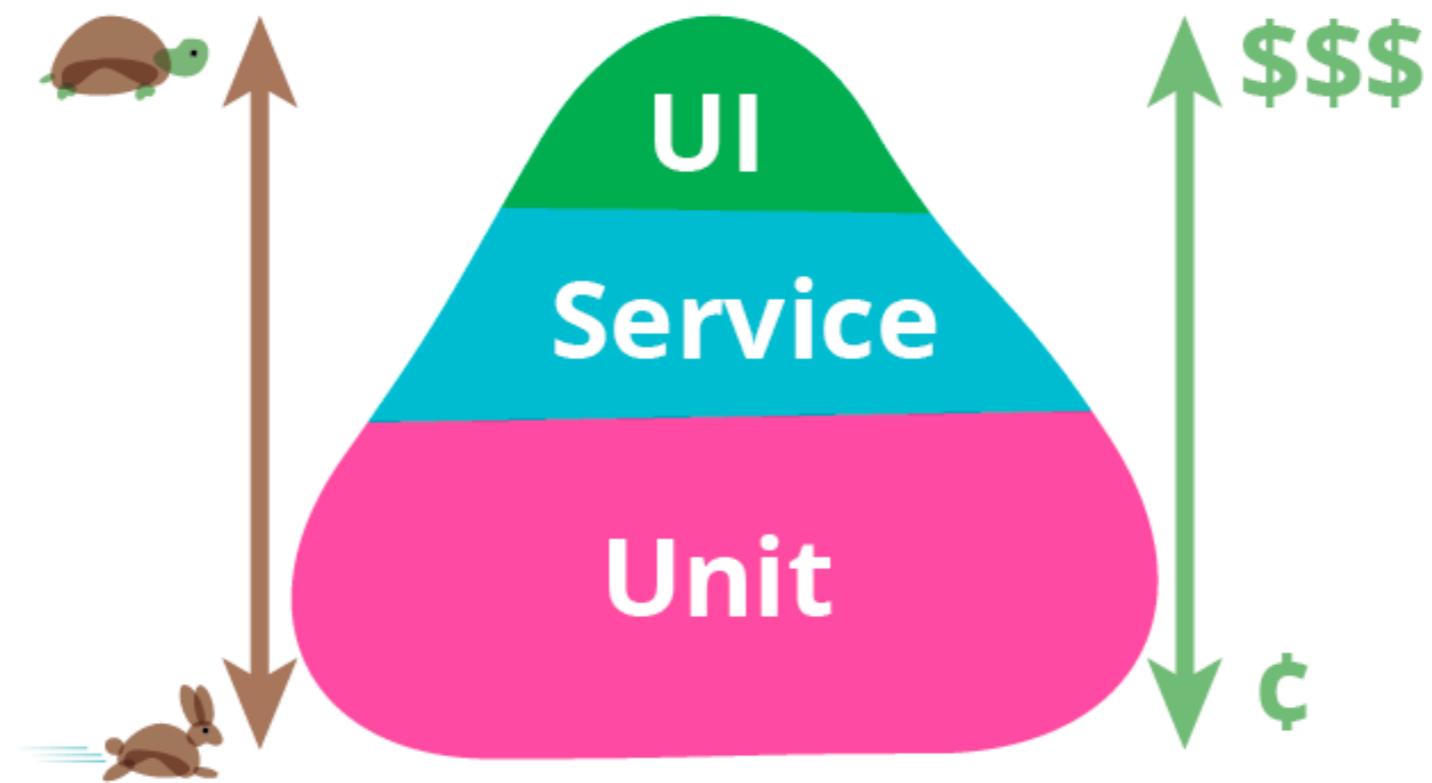
<https://less.works/less/technical-excellence/index.html>



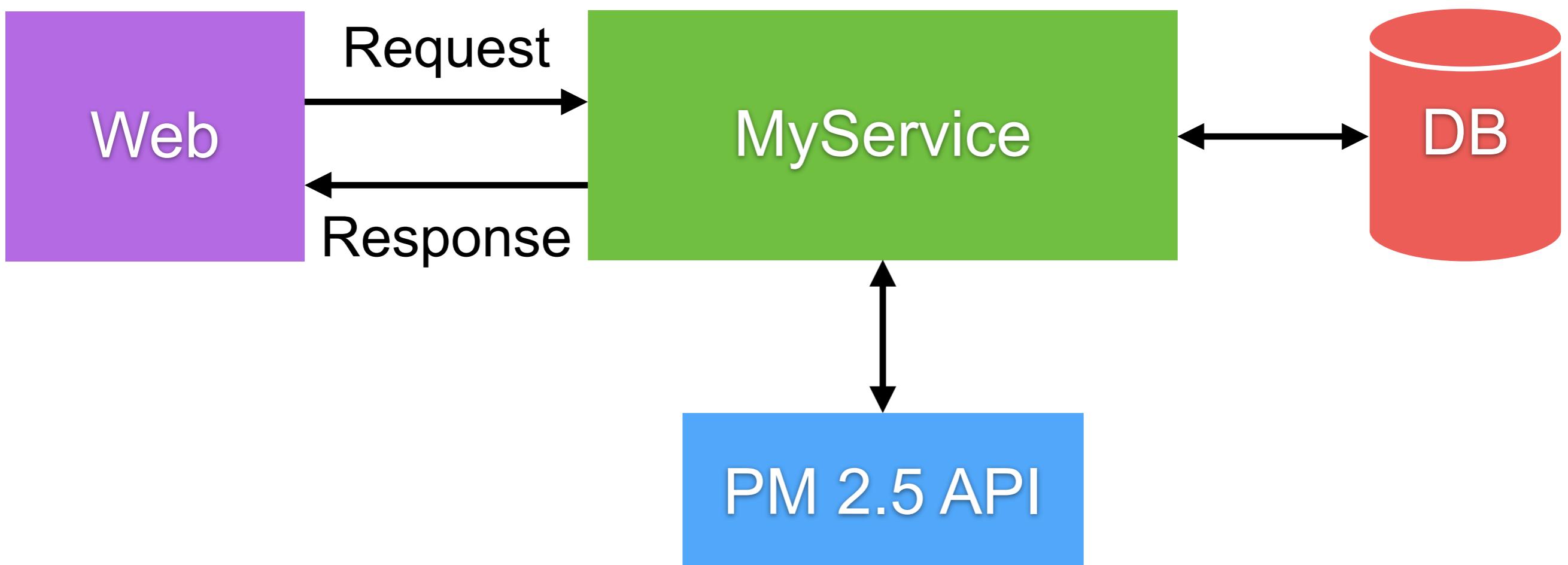
Workshop with Testing



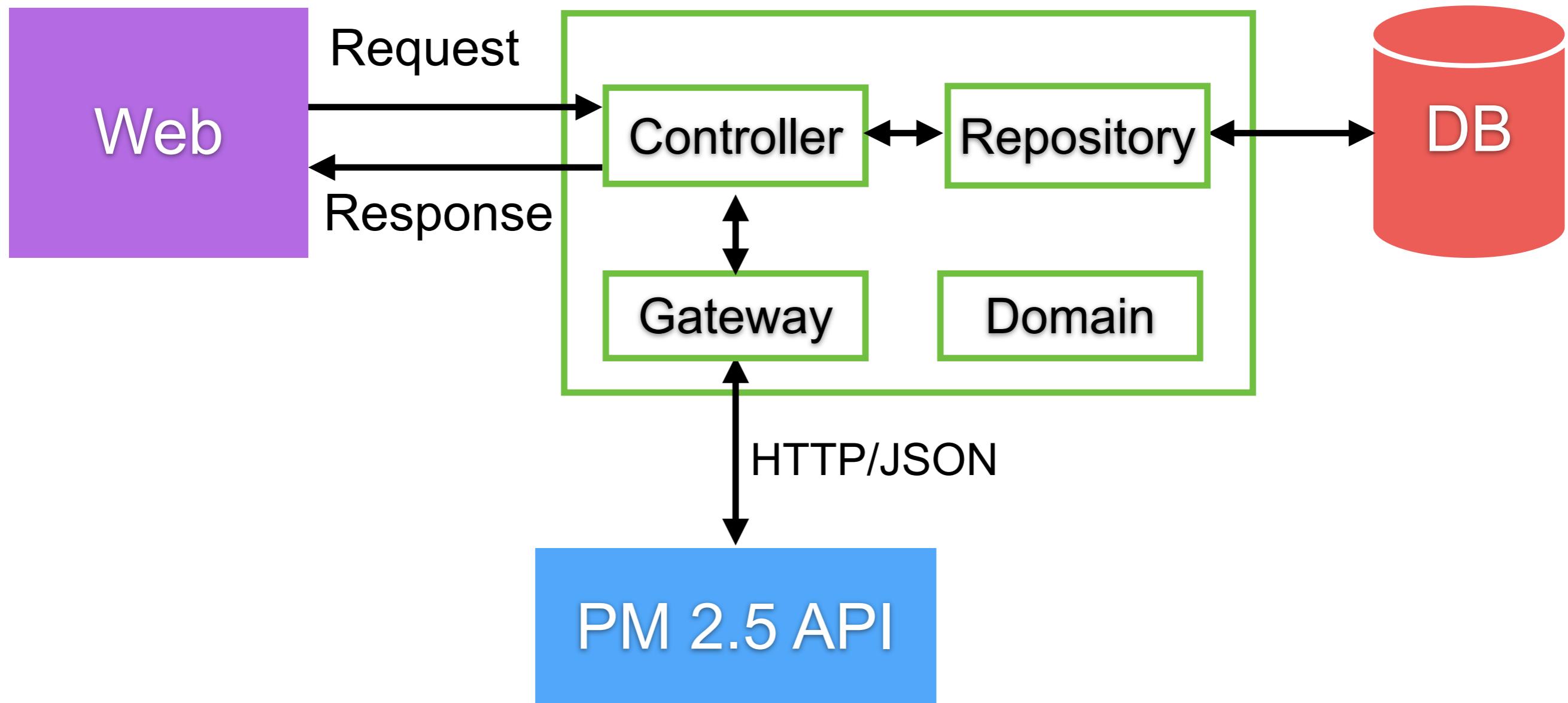
Test Pyramid



Workshop



Structure of service

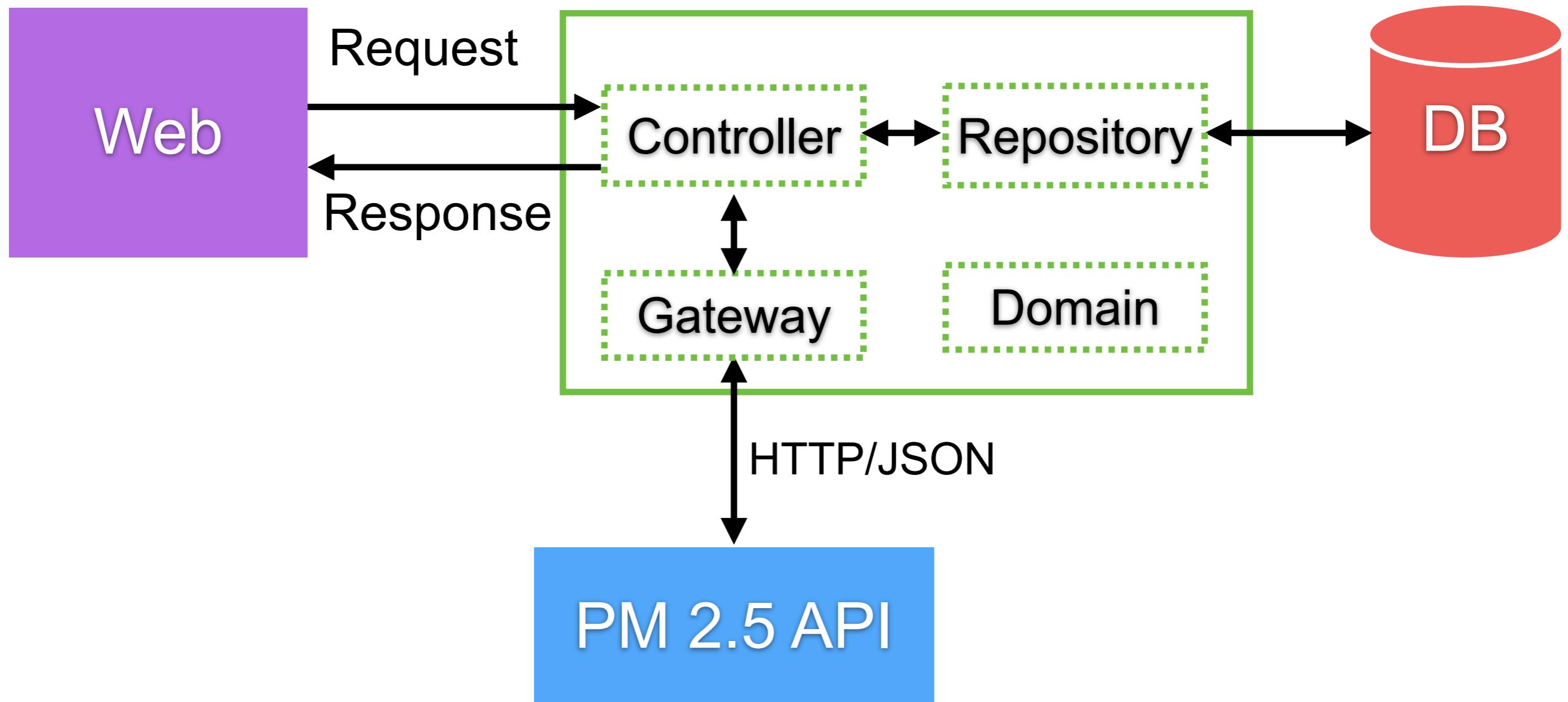




Unit testing



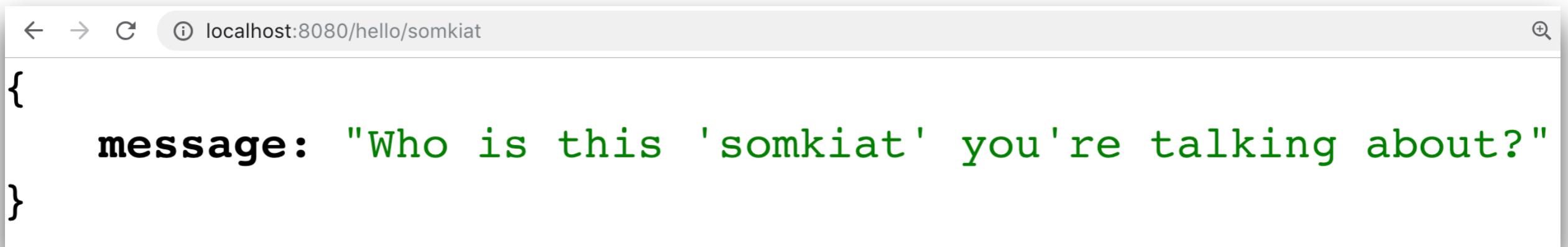
Unit testing



Controller testing ?

\$mvnw spring-boot:run

http://localhost:8080/hello/somkiat



A screenshot of a web browser window. The address bar shows "localhost:8080/hello/somkiat". The page content is a JSON object:

```
{  
  message: "Who is this 'somkiat' you're talking about?"  
}
```



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"));
}
```



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)

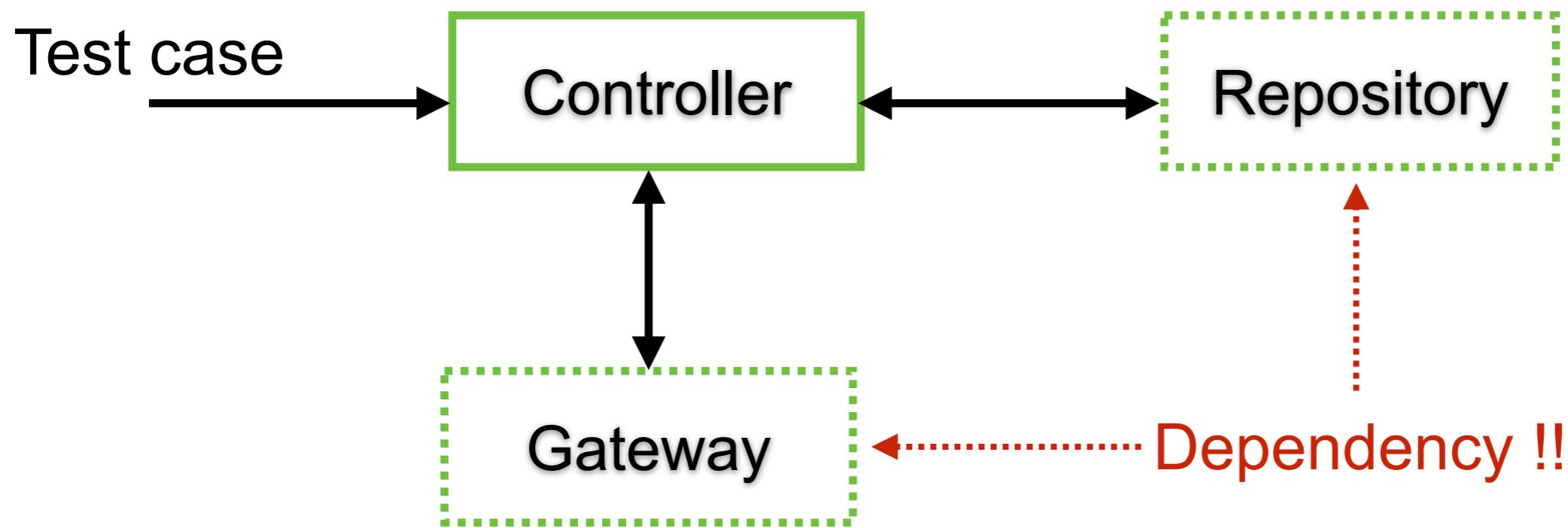
Given When Then from BDD style

<https://www.martinfowler.com/bliki/GivenWhenThen.html>

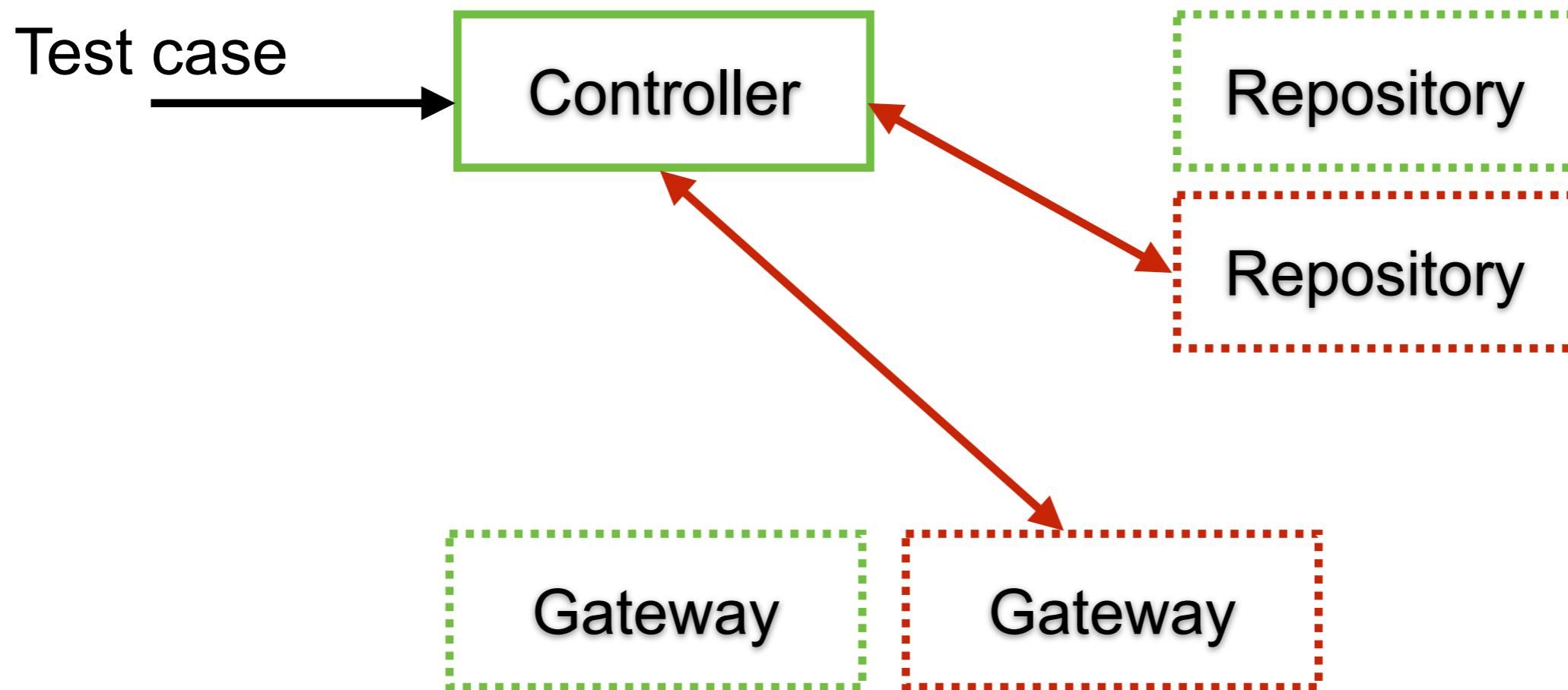
<https://xp123.com/articles/3a-arrange-act-assert/>



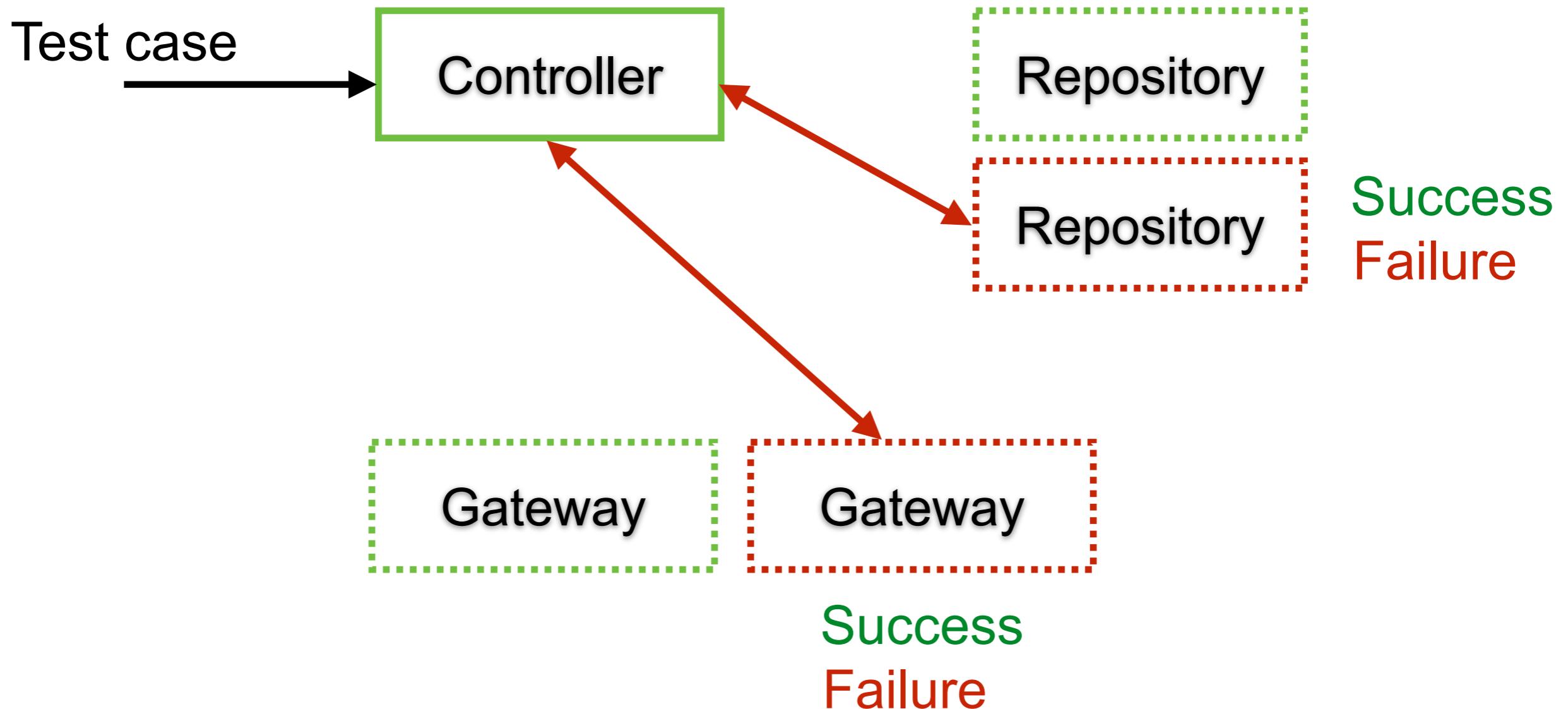
How to test Controller ?



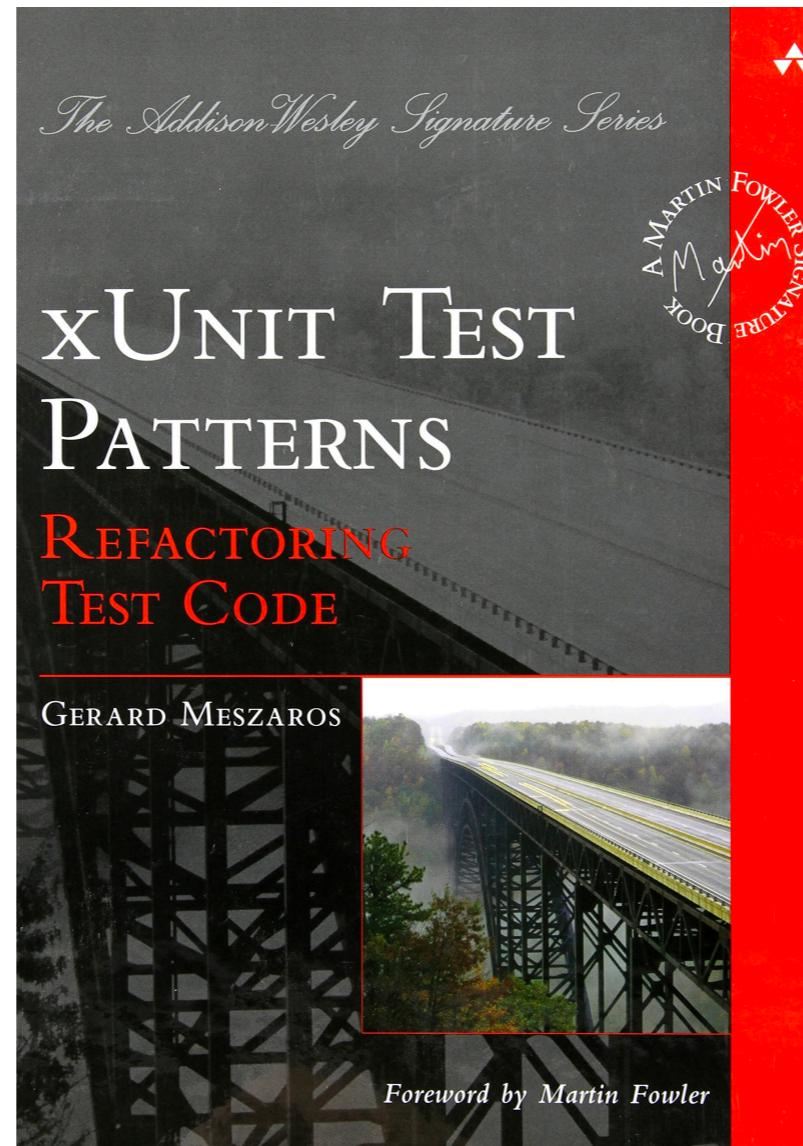
Working with Test Double



Working with Test Double



Test Double ?



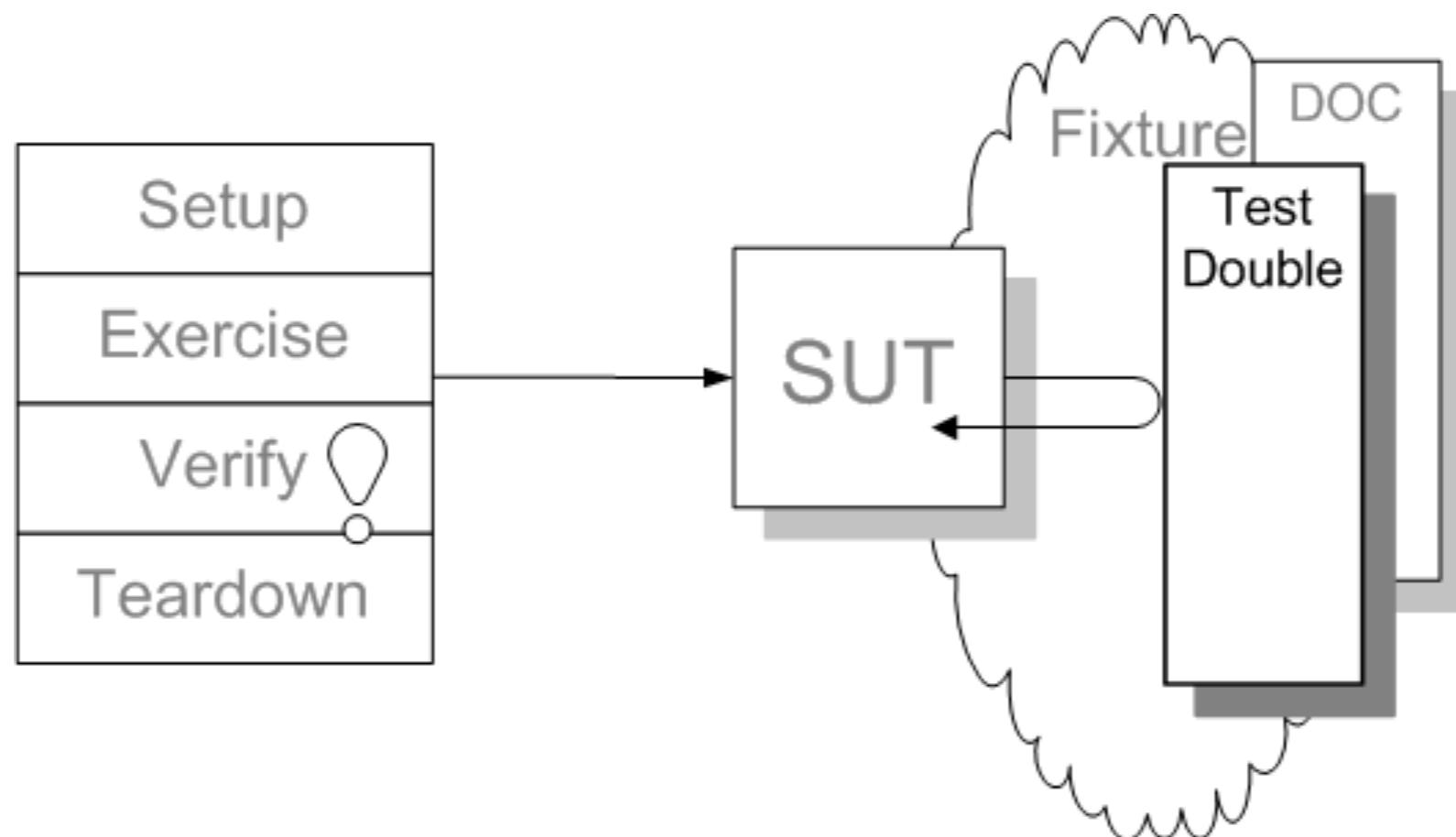
<http://xunitpatterns.com>



TDD with Java

© 2017 - 2018 Siam Chamnankit Company Limited. All rights reserved.

Test Double ?



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



Test Double ?

Dummy
object

Stub

Spy

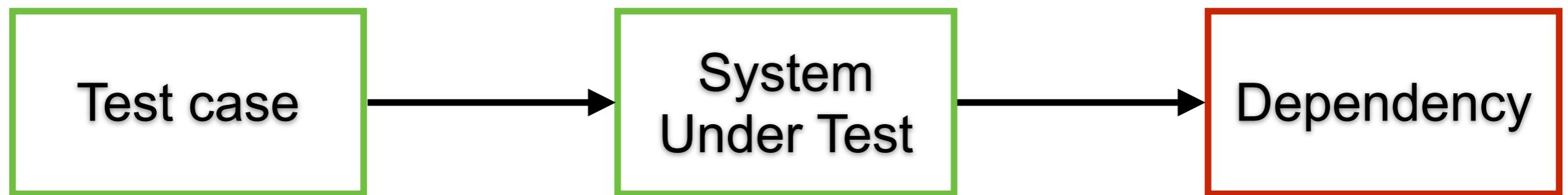
Mock object

Fake object

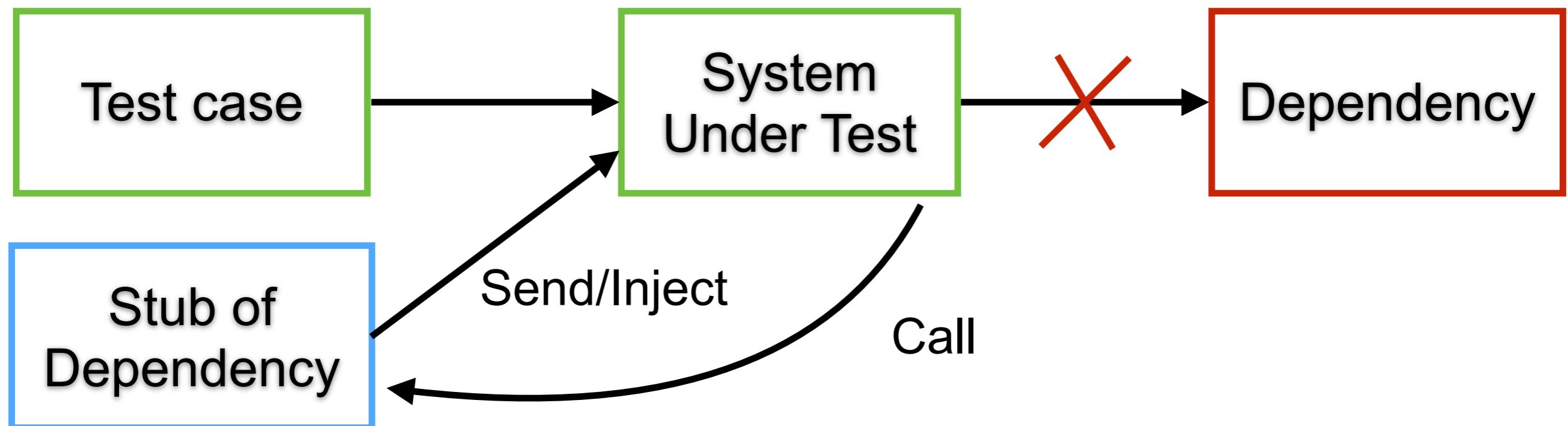
<http://xunitpatterns.com/Test%20Double.html>



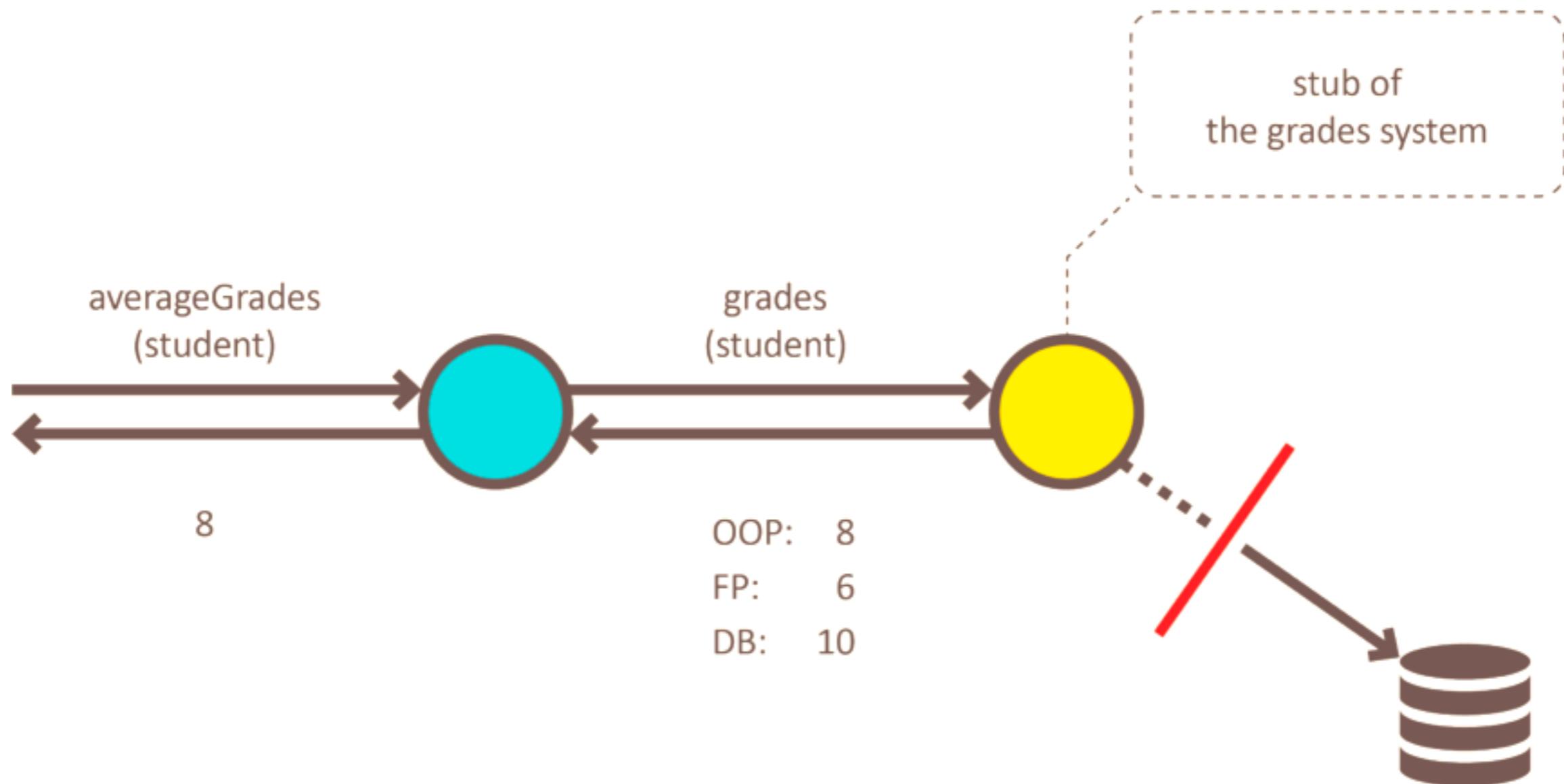
Stub



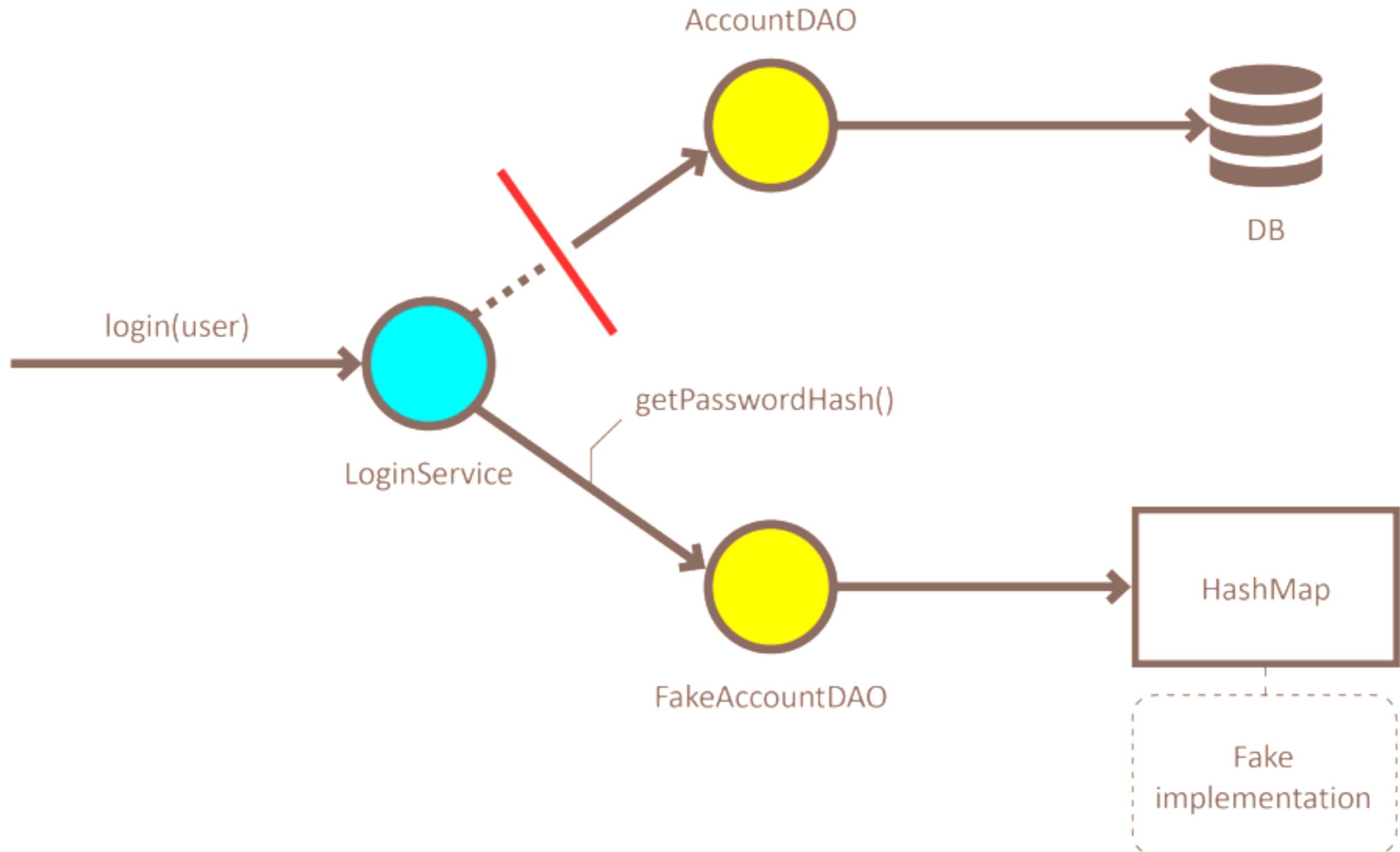
Create and Send Stub



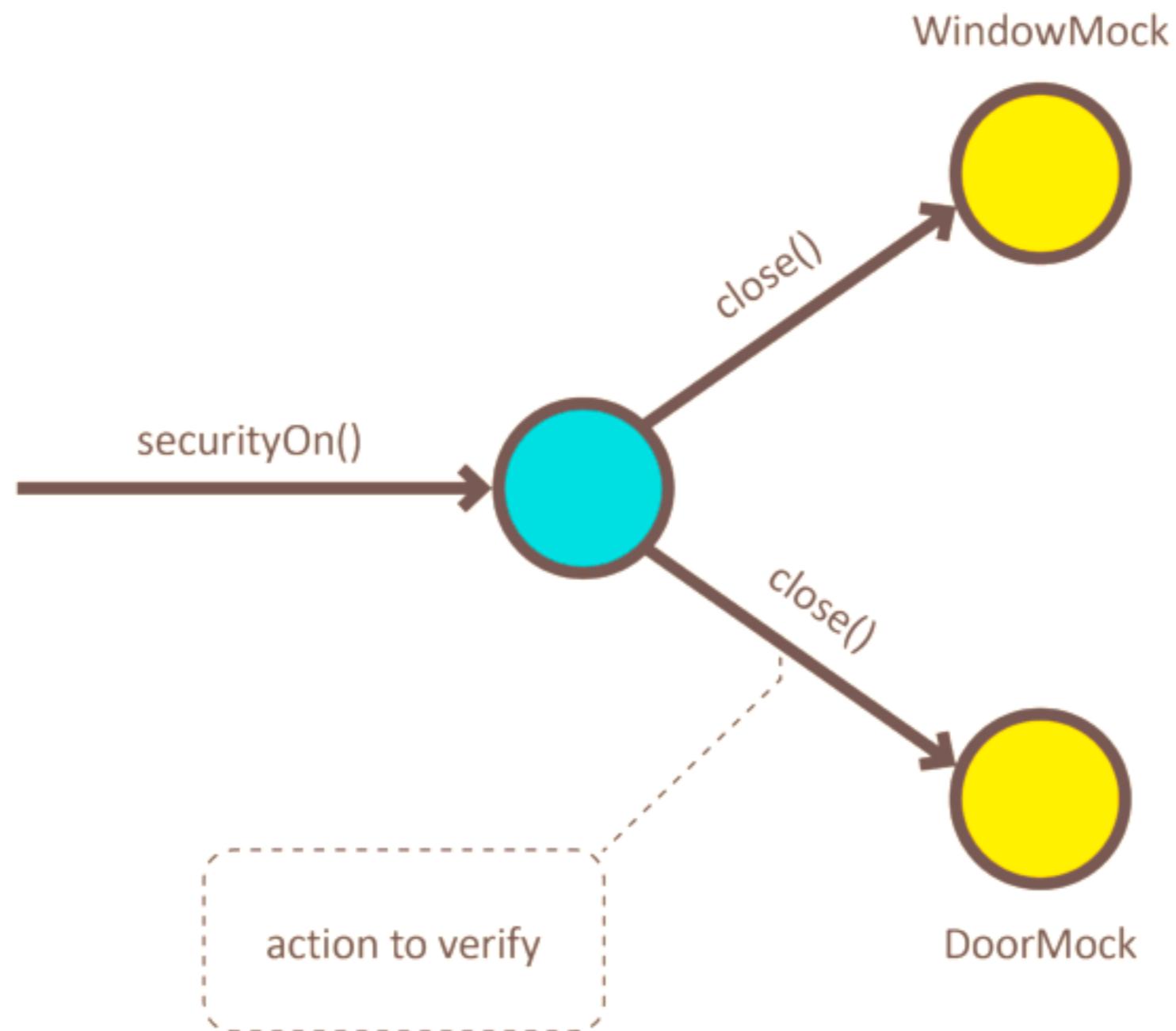
Stub



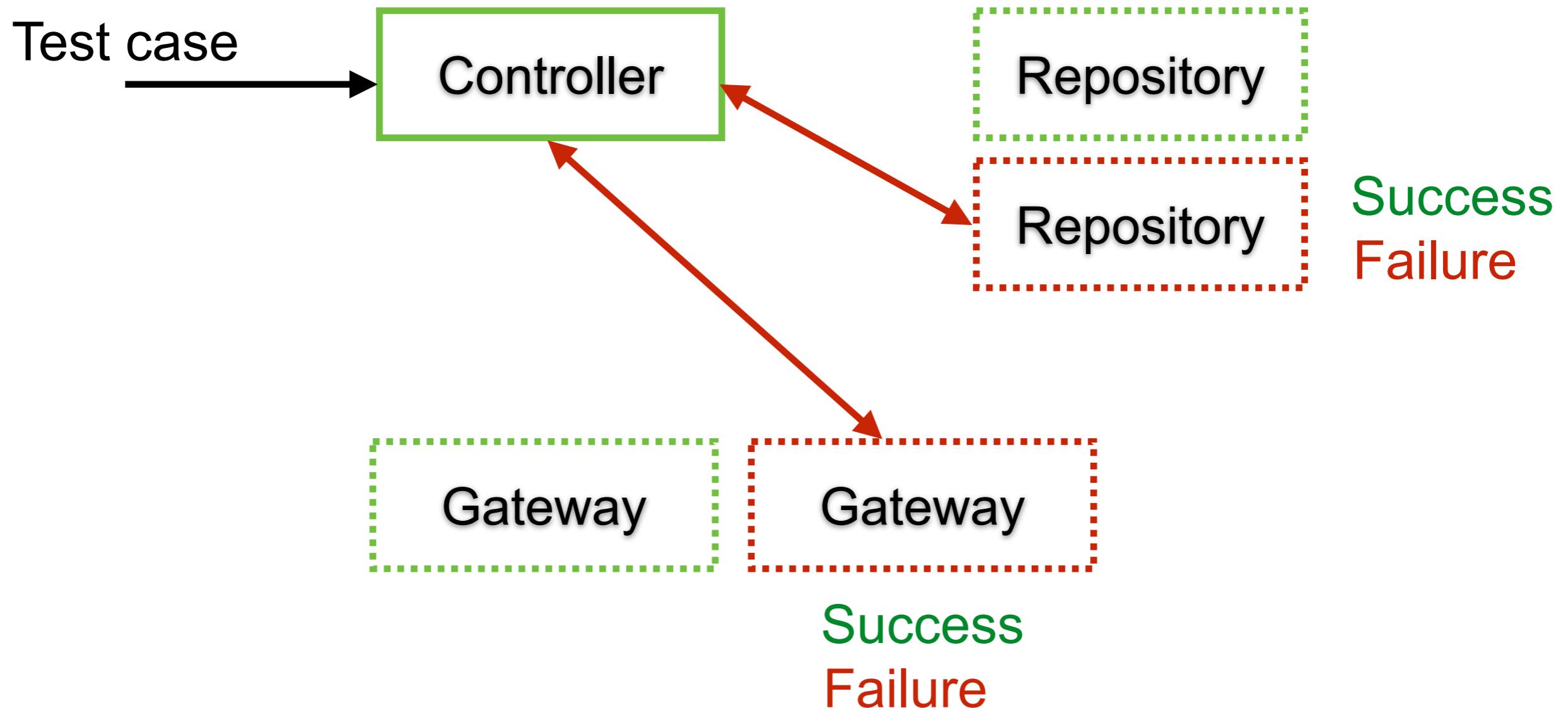
Fake



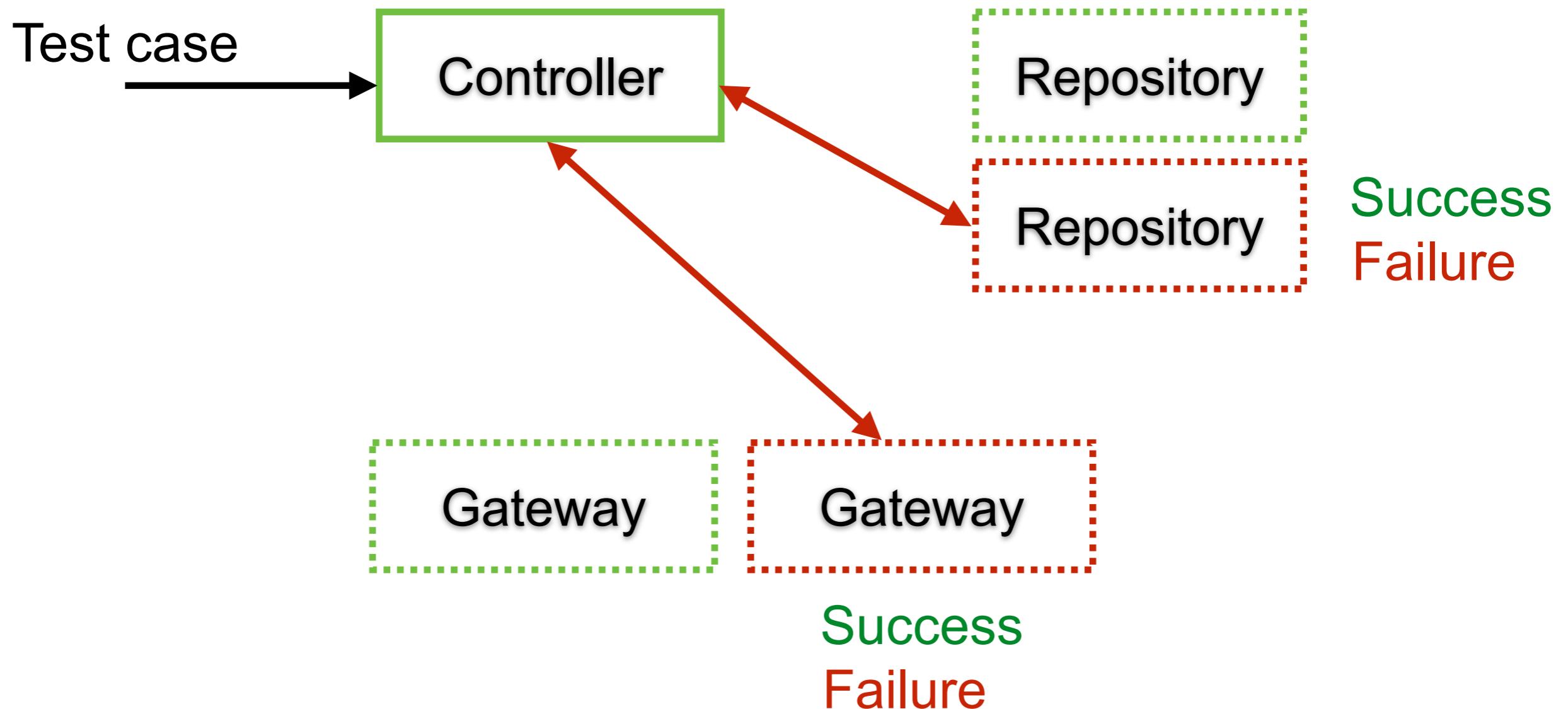
Mock



Working with Test Double ?



Stub



How to create Stub object ?

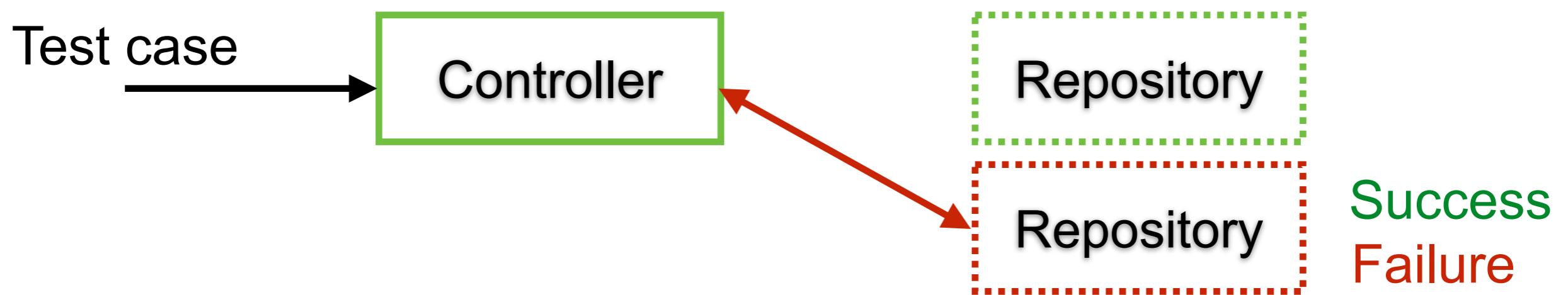
Manual
Library



<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.findByName("Pui"))  
        .willReturn(Optional.of(somkiat));  
  
    // Act  
    HelloResponse greeting = subject.hello("Pui");  
  
    // Assert  
    assertThat(greeting.getMessage(), is("Hello Somkiat Pui!"));  
}
```



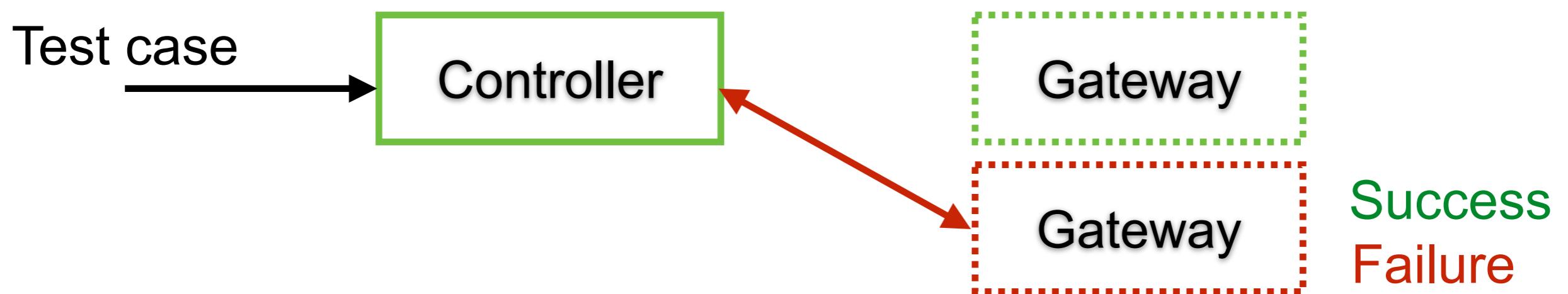
Unit test for Controller #3

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

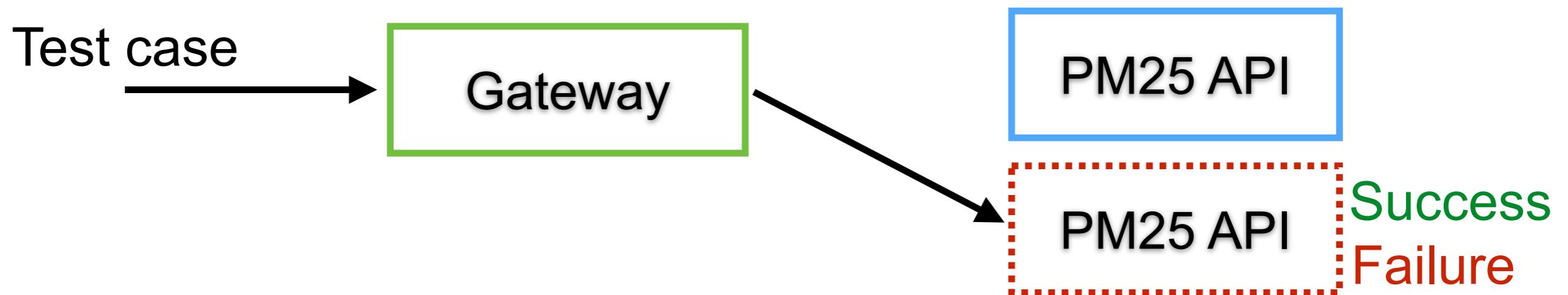
```
@Test  
public void shouldReturnErrorMessageIfPM25IsUnavailable() throws  
Exception {  
  
    // Create stub  
    given(pm25Gateway.fetchData())  
        .willReturn(Optional.empty());  
  
    // Act  
    String pm25 = subject.getDataPM25();  
  
    // Assert  
    assertThat(pm25, is("Can 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

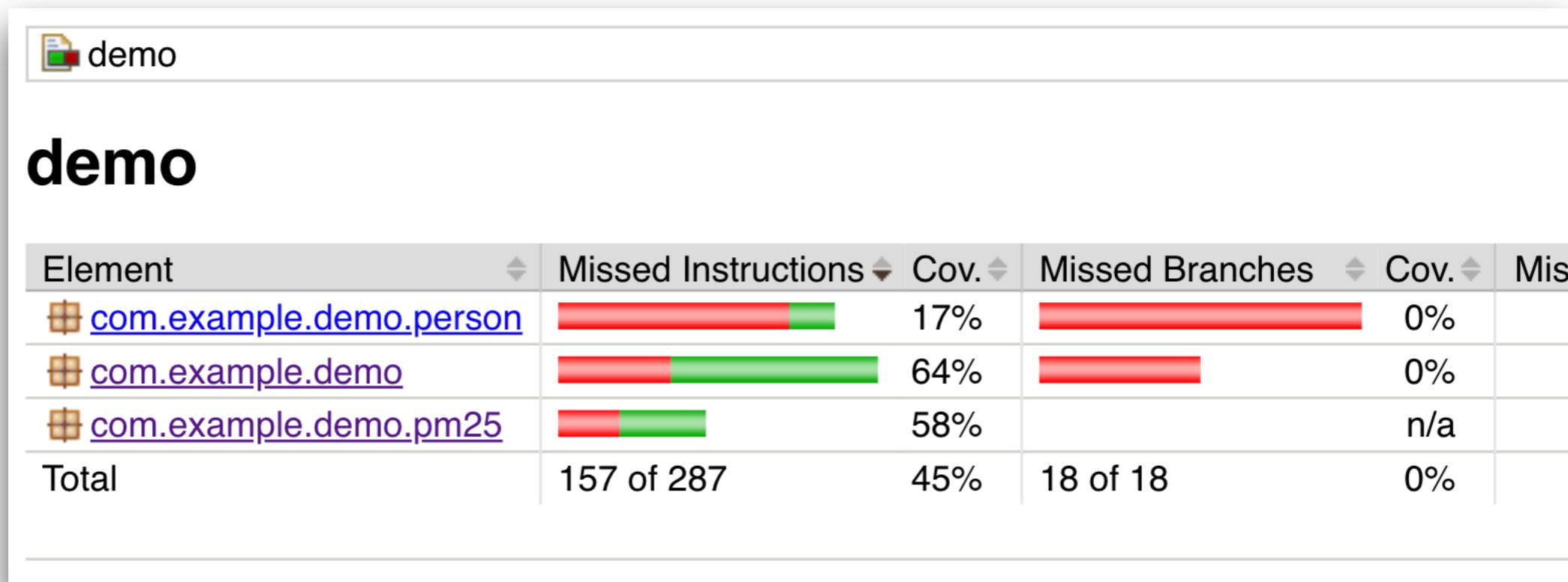
```
@Test  
public void shouldReturnEmptyOptionalIfPM25APIIsUnavailable() {  
    // Create Stub  
    given(restTemplate.getForObject(  
        "http://localhost:8089/feed/thailand/", PM25Response.class))  
        .willThrow(new RestClientException("something went wrong"));  
  
    // Act  
    Optional<PM25Response> actualResponse = subject.fetchData();  
  
    // Assert  
    assertThat(actualResponse, is(optional.empty()));  
}
```



Run all test + coverage

\$mvnw clean package

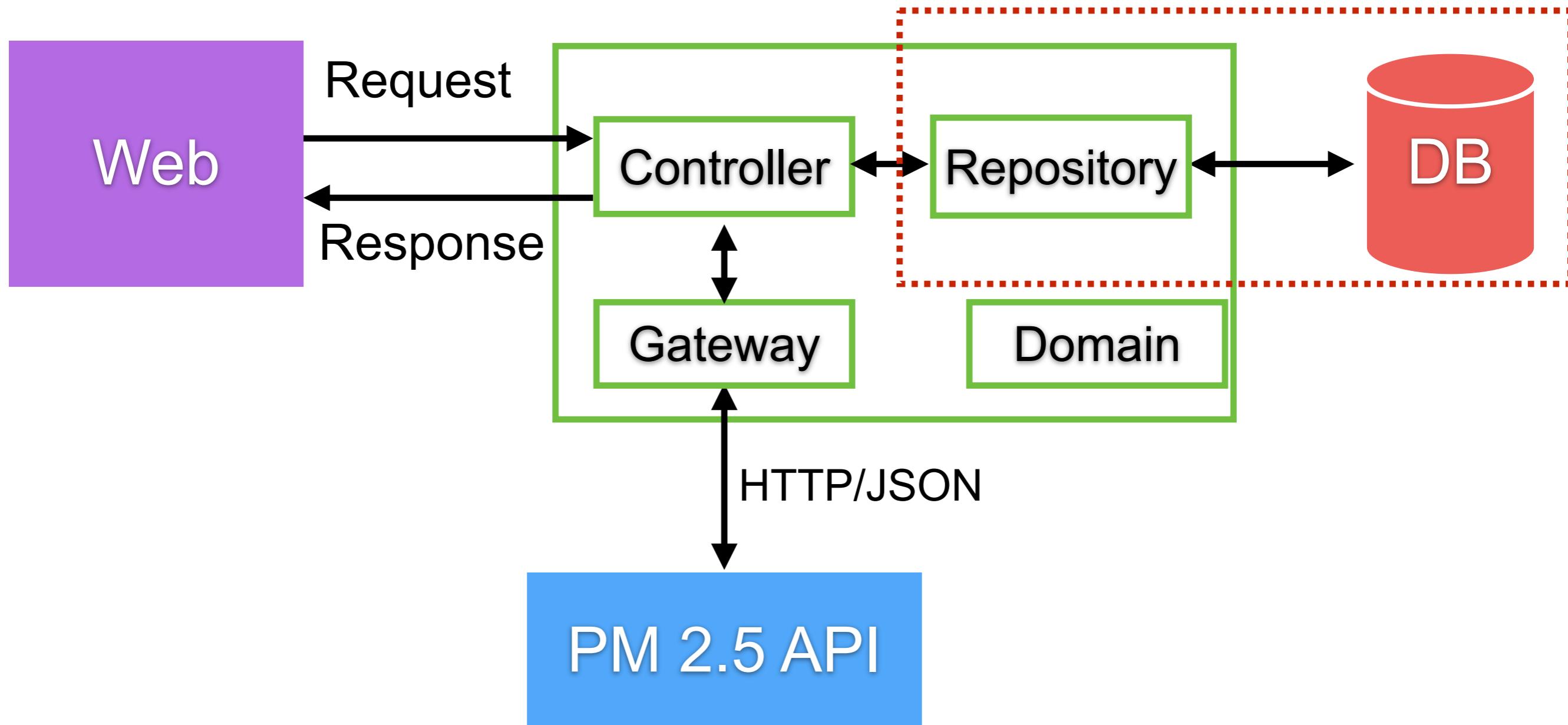
Open file /target/site/jacoco/index.html



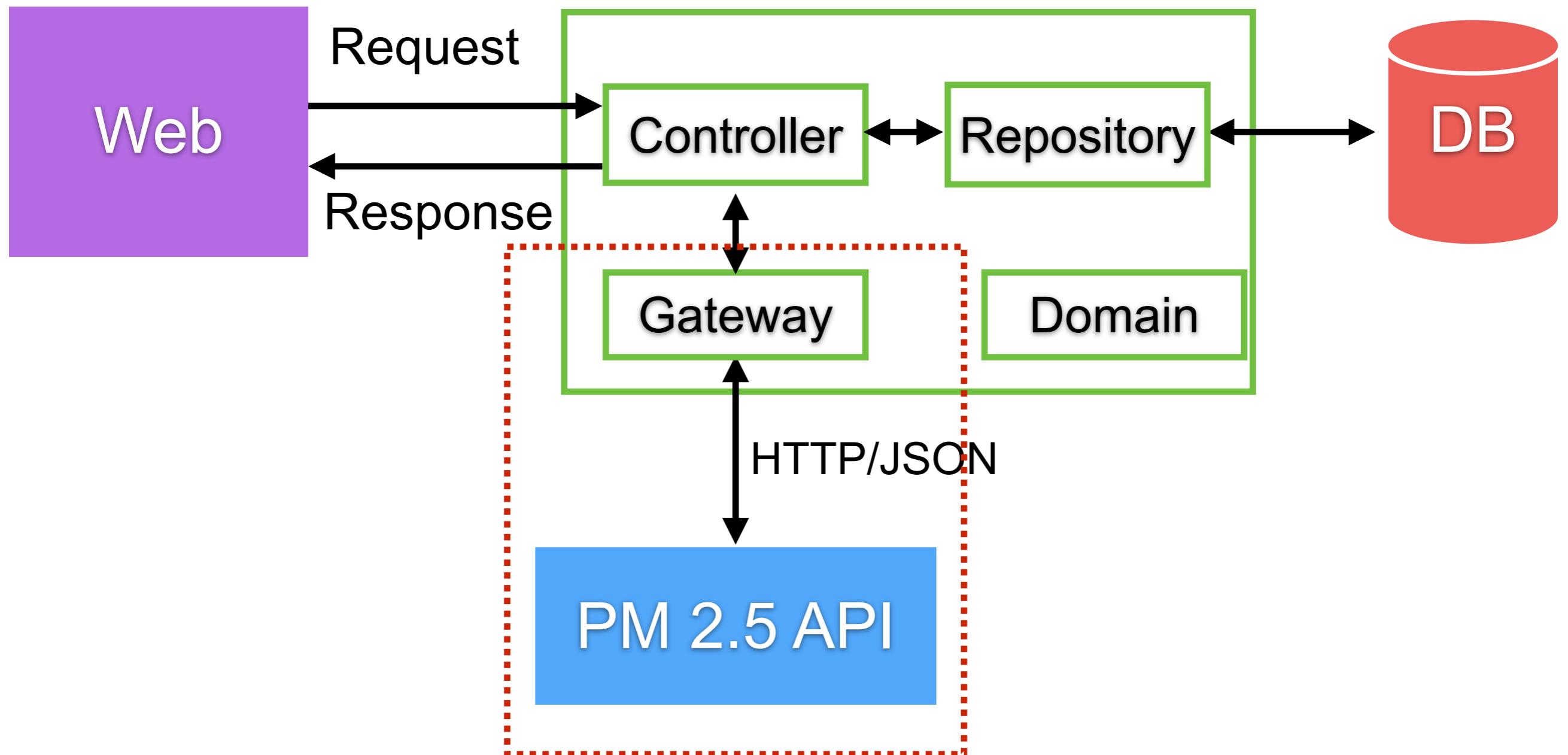
Integration testing



Integration testing

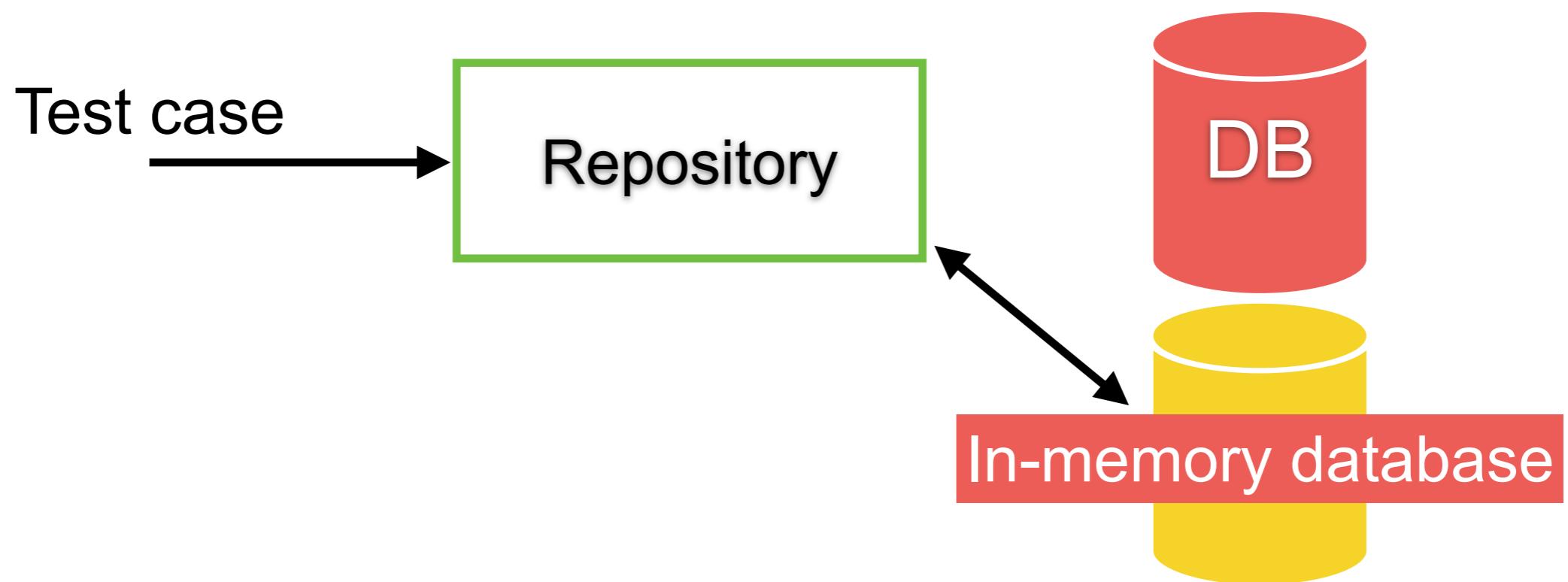


Integration testing



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

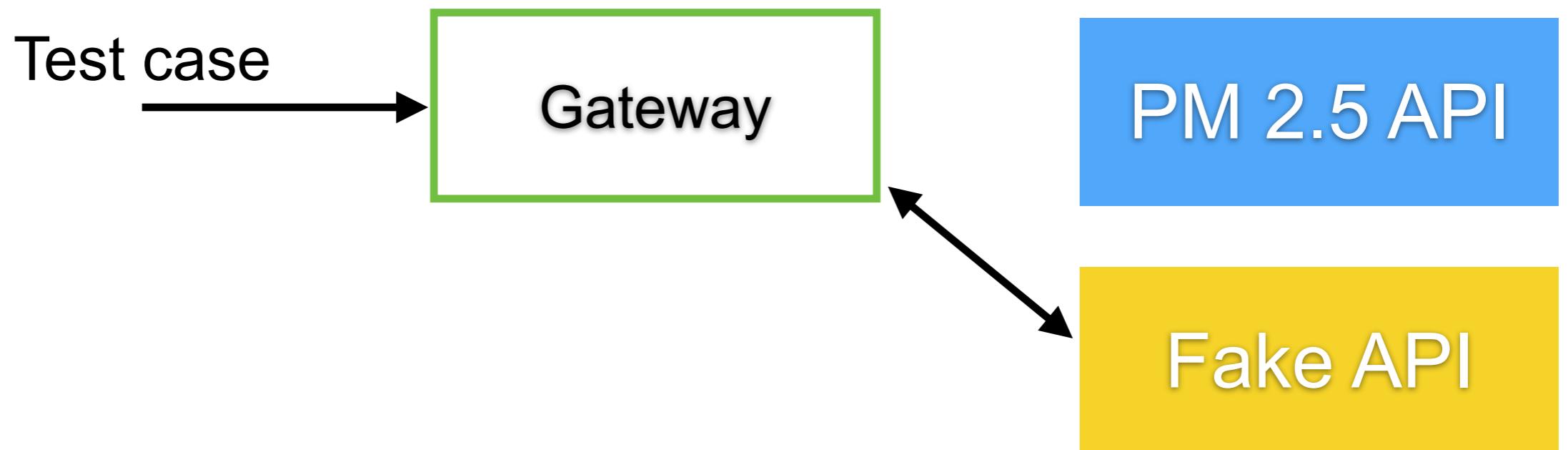
File /src/test/java/PersonRepositoryIntegrationTest.java

```
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

The screenshot shows the official WireMock website. At the top, there's a dark header bar with the WM logo, navigation links for Docs, About, GitHub, Mailing List, External Resources, and MockLab, and a search bar. Below the header, the main content area has a large title "WireMock" and a subtitle "Mock your APIs for fast, robust and comprehensive testing". It describes WireMock as a simulator for HTTP-based APIs, comparing it to a service virtualization tool or mock server. It highlights its productivity benefits and support for edge cases and failure modes. To the right, a modal window displays a JSON configuration file for a mock API endpoint.

```
{  
  "request": {  
    "method": "GET",  
    "urlPathMatching": "/path(\\*)/match",  
    "queryParameters": {  
      "search": {  
        "contains": "WireMock"  
      }  
    }  
  },  
  "response": {  
    "status": 200,  
    "headers": {  
      "Content-Type": "application/json"  
    },  
    "jsonBody": {  
      "search_results": []  
    }  
  }  
}
```

<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));
}
```



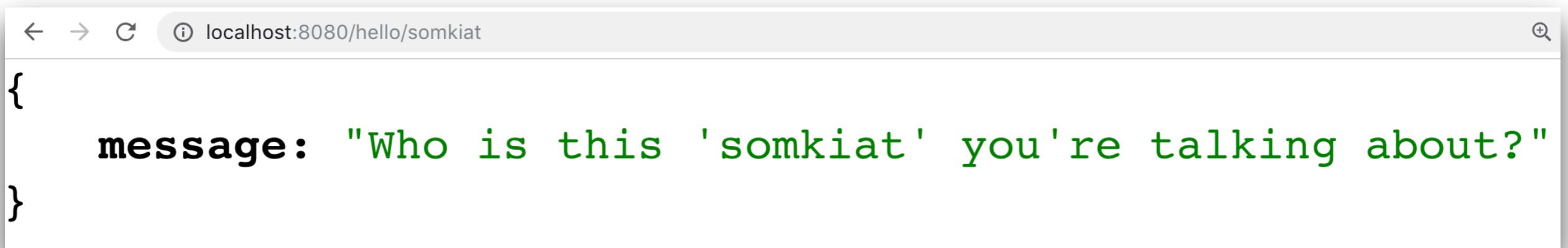
UI testing



UI Testing

Open your browser

<http://localhost:8080/hello/somkiat>



```
{  
  message: "Who is this 'somkiat' you're talking about?"  
}
```



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!"));  
}
```



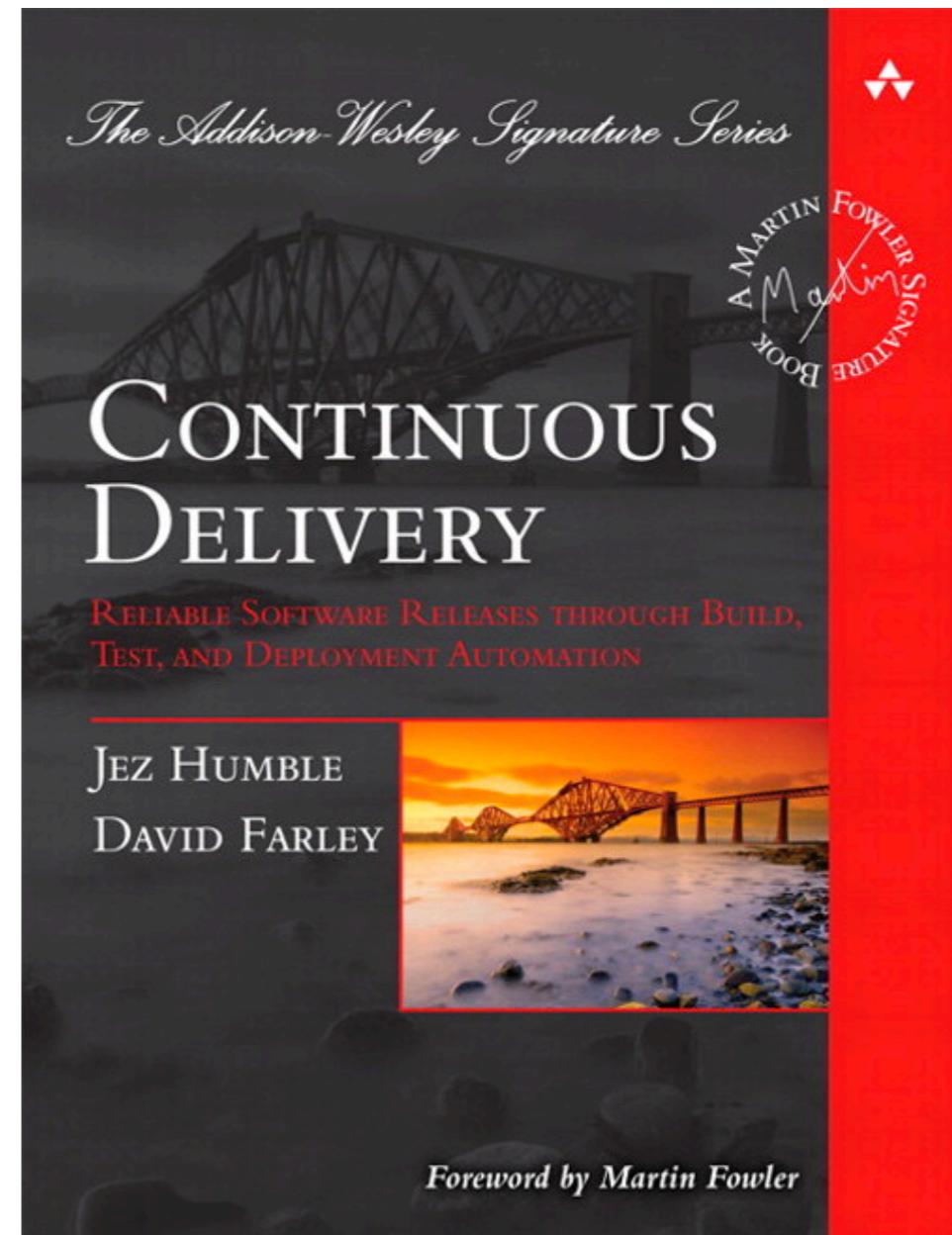
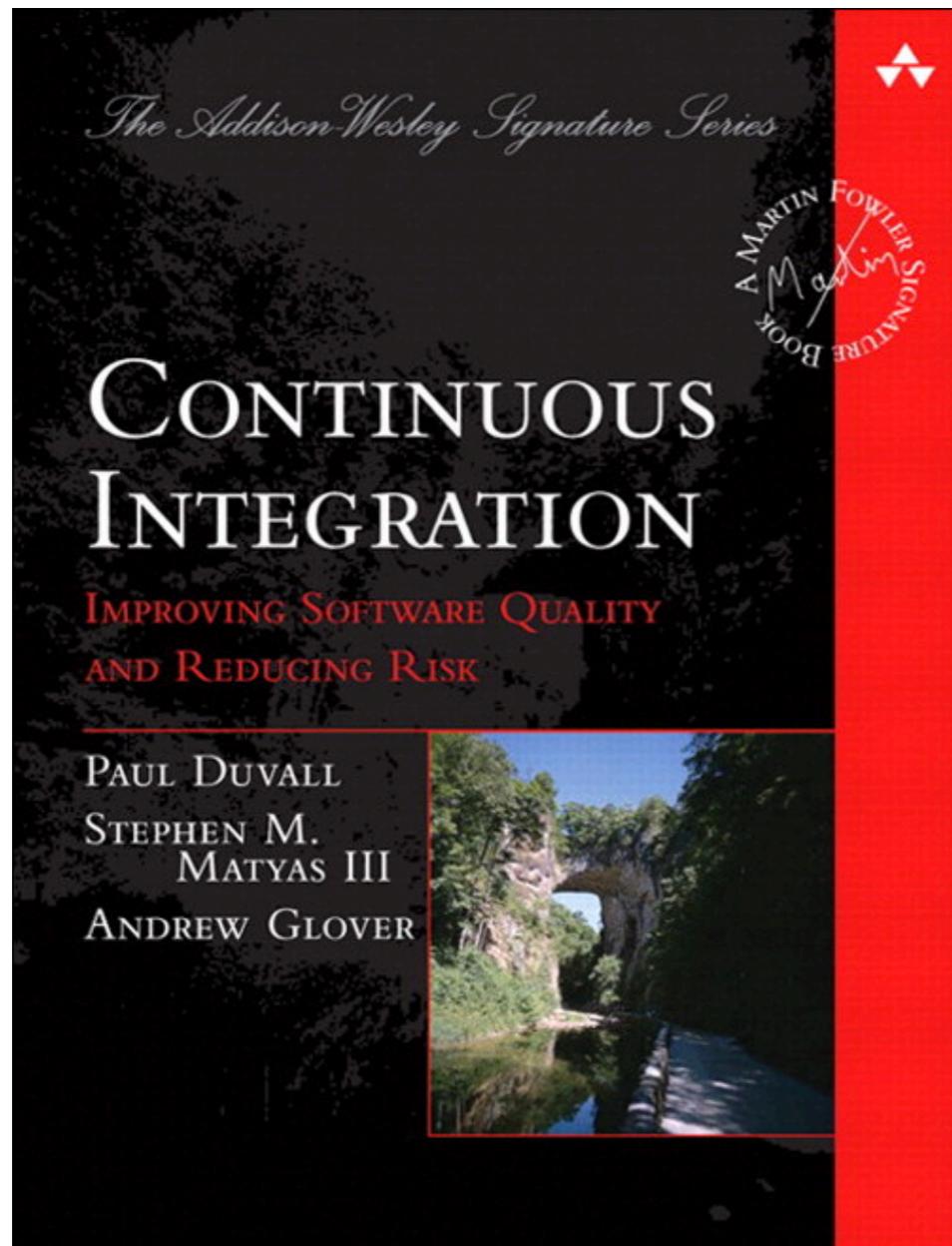
Continuous Integration



**“Behind every successful agile
project, there is a
Continuous Integration System”**



Improve quality and reduce risk

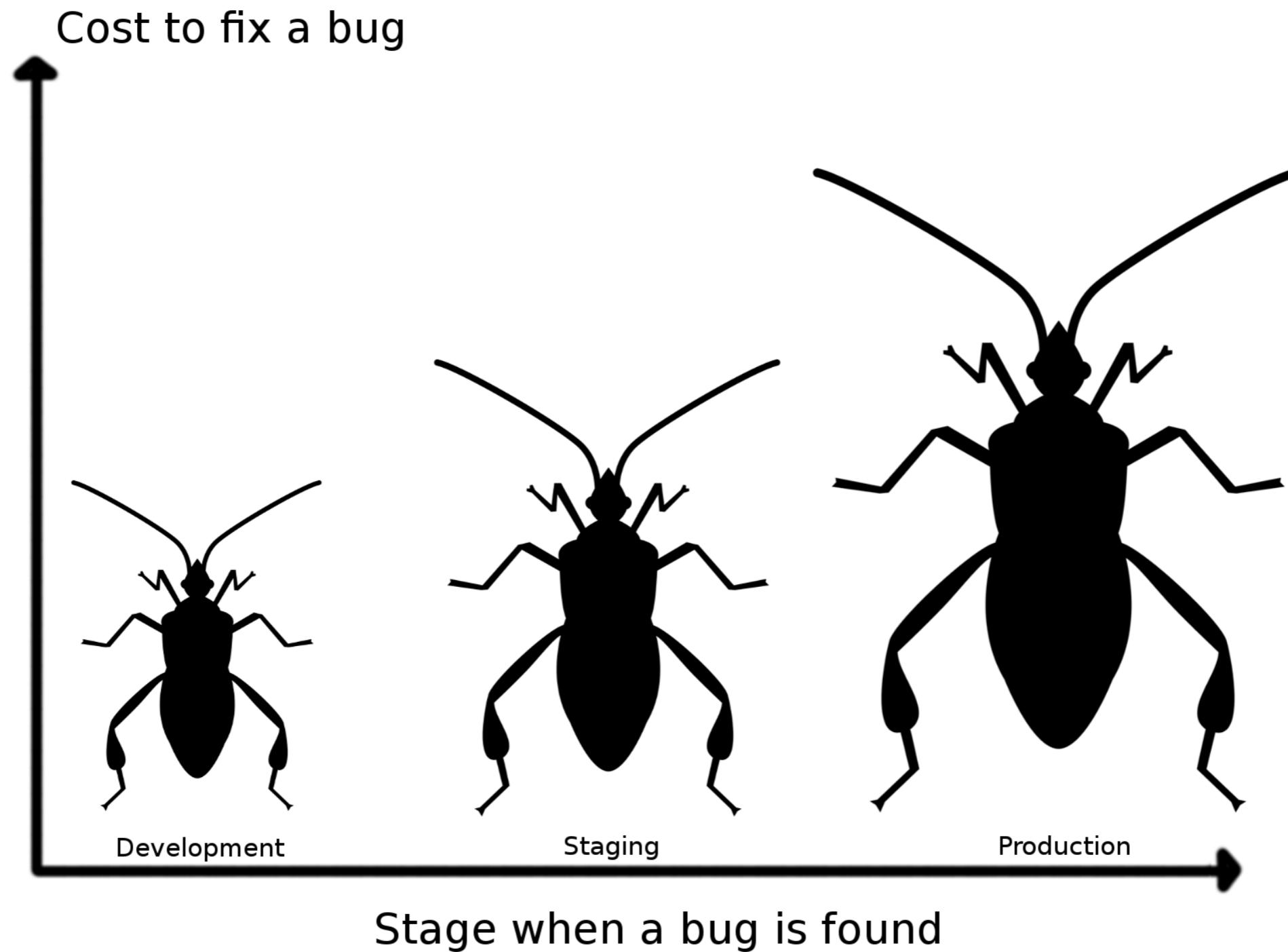


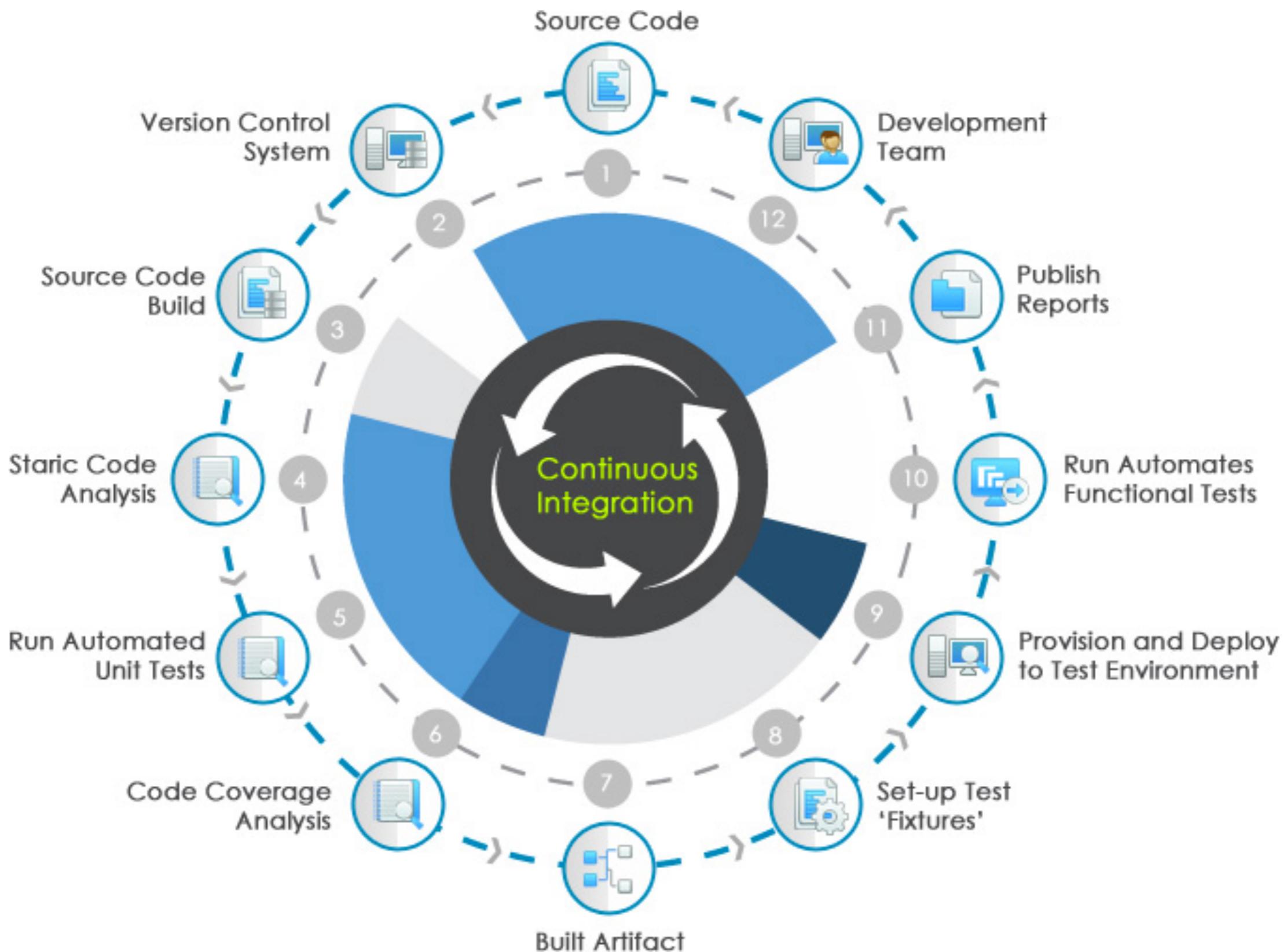
The cost of integration

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

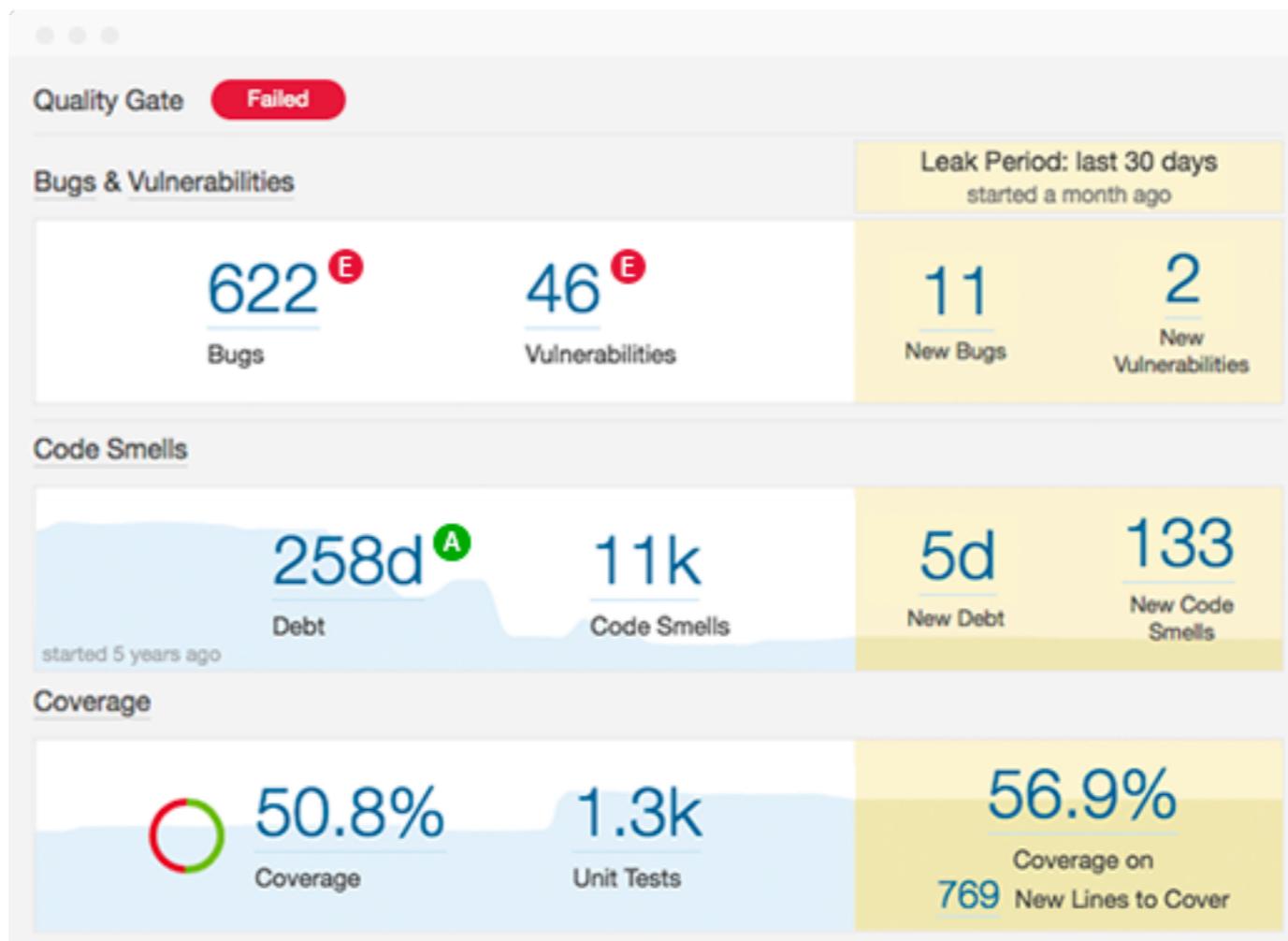


The cost of integration





Code Quality



<https://www.sonarqube.org/>





Jenkins

Bamboo



TeamCity

> goTM



Hudson





Jenkins

Bamboo

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



Hudson



Continuous Integration

Discipline to integrate frequently



Continuous Integration

Strive to make **small change**

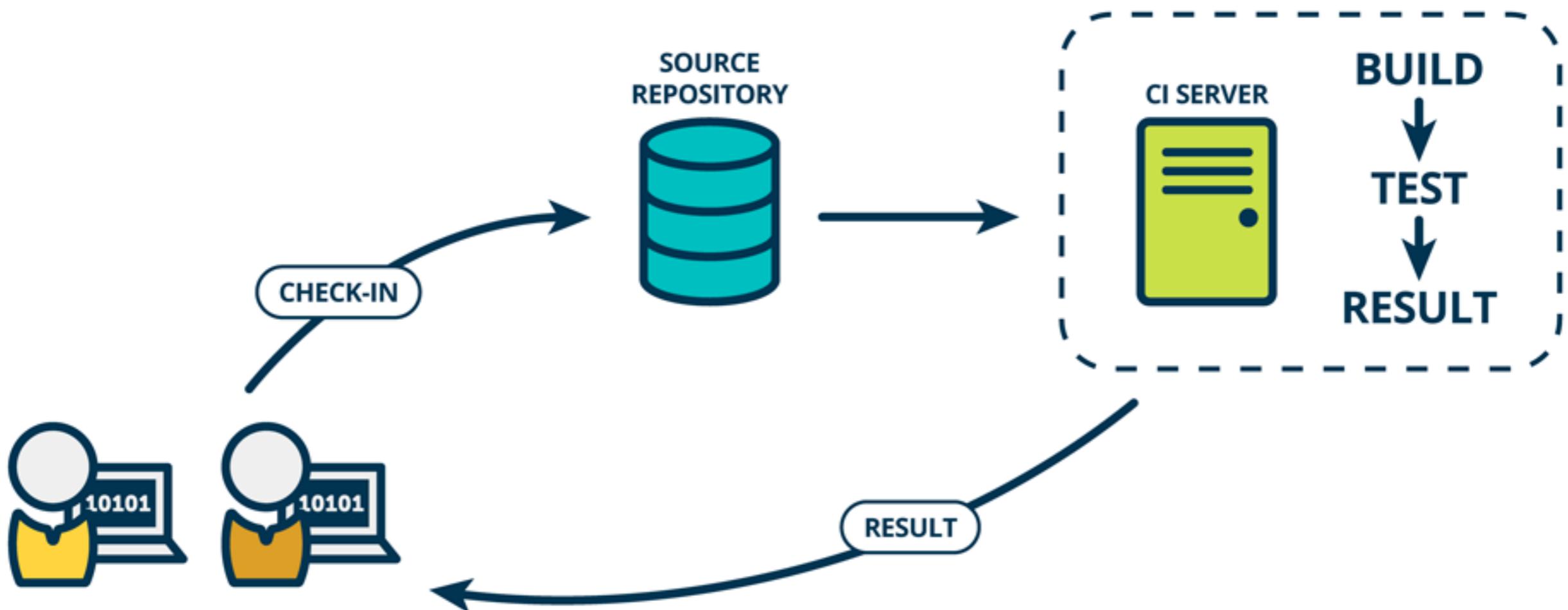


Continuous Integration

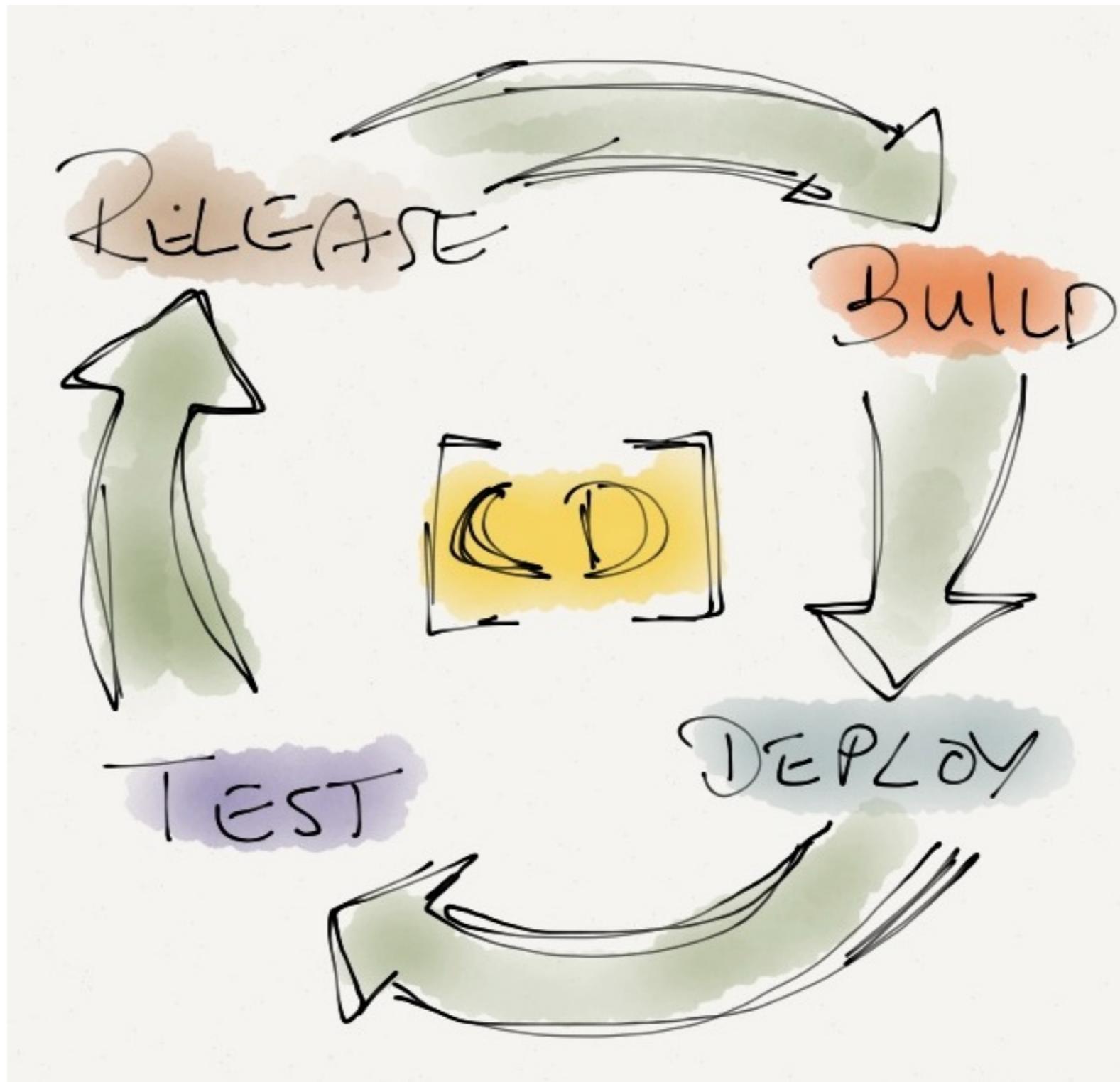
Strive for **fast feedback**



Continuous Integration



CD ?



CD ?

CONTINUOUS DELIVERY



CONTINUOUS DEPLOYMENT



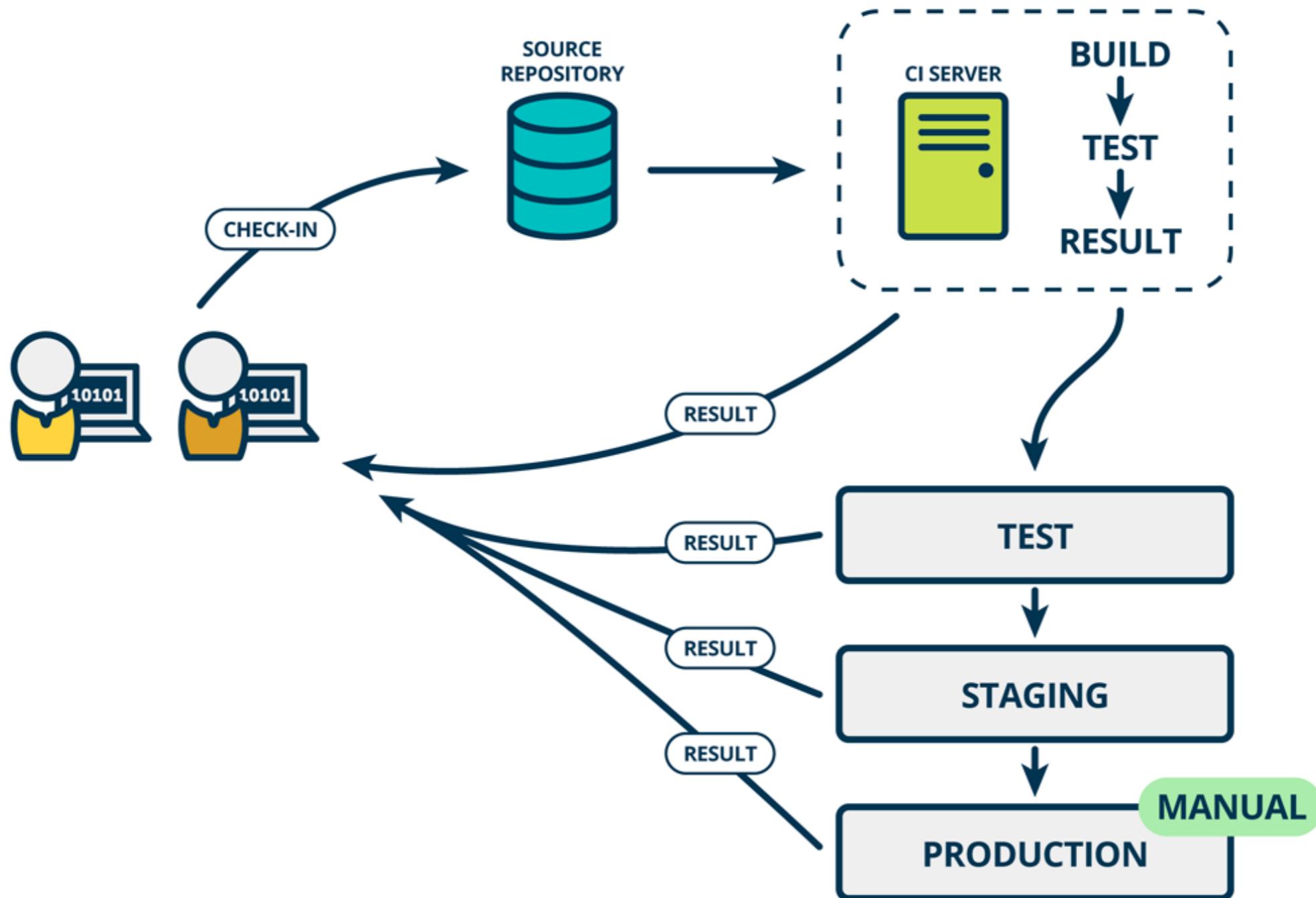
<http://blog.crisp.se/2013/02/05/yassalsundman/continuous-delivery-vs-continuous-deployment>



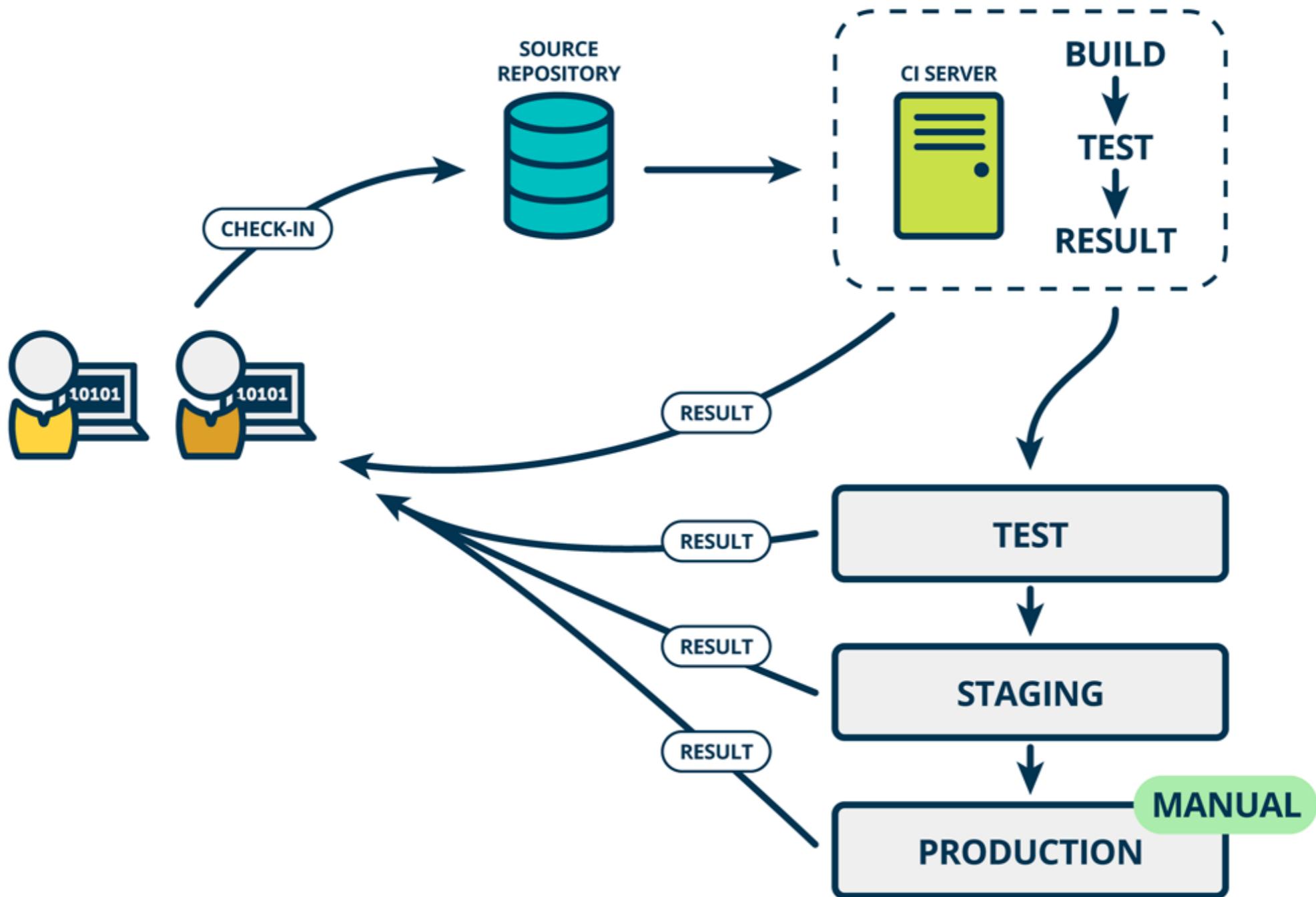
Microservices

© 2017 - 2018 Siam Chamnankit Company Limited. All rights reserved.

Continuous Delivery



Rise of DevOps



Continuous Integration

is a Software development practices



Practice 1

Maintain a single source repository

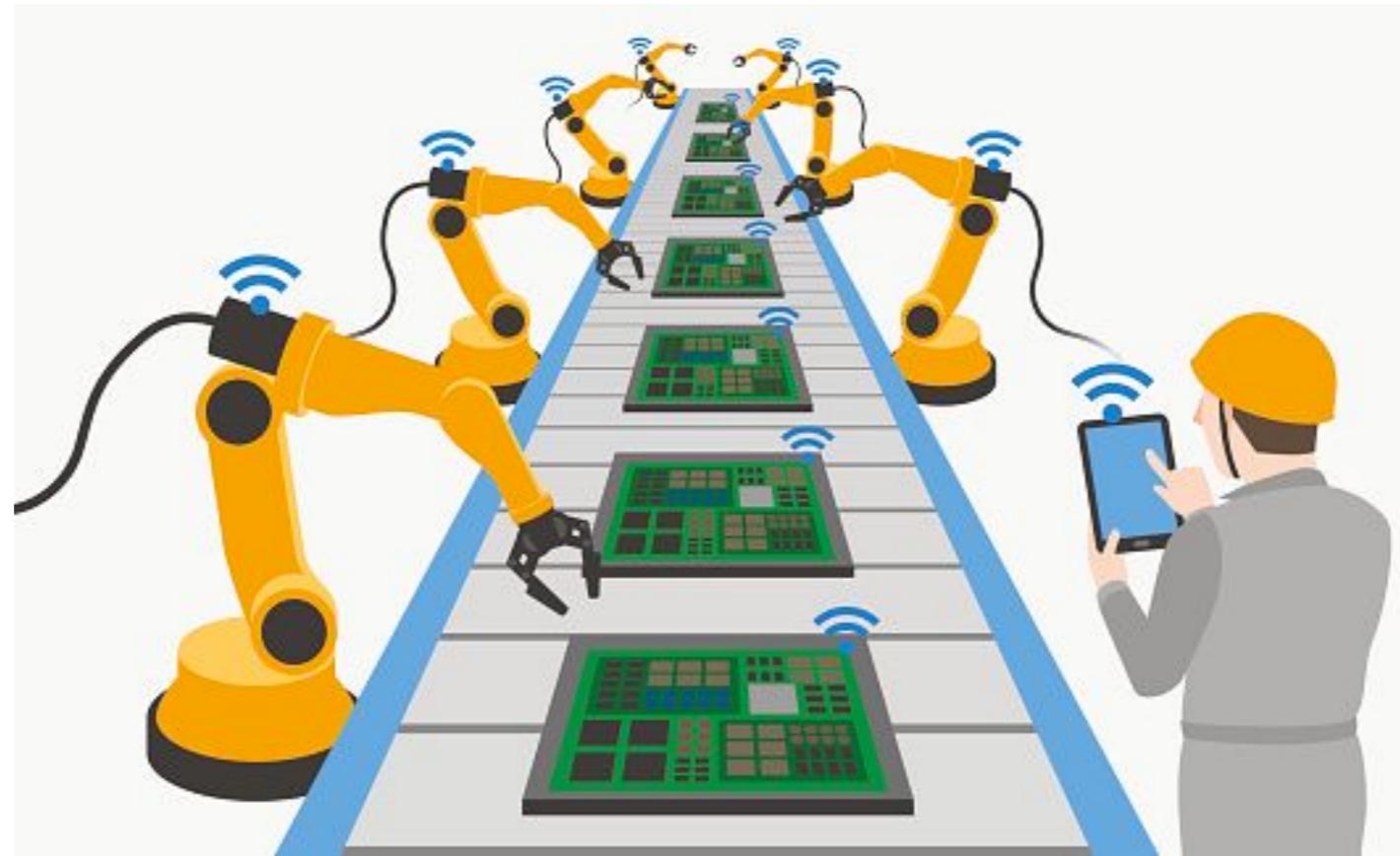
In general, you should store in source control
everything you need to build anything



Practice 2

Automated the build

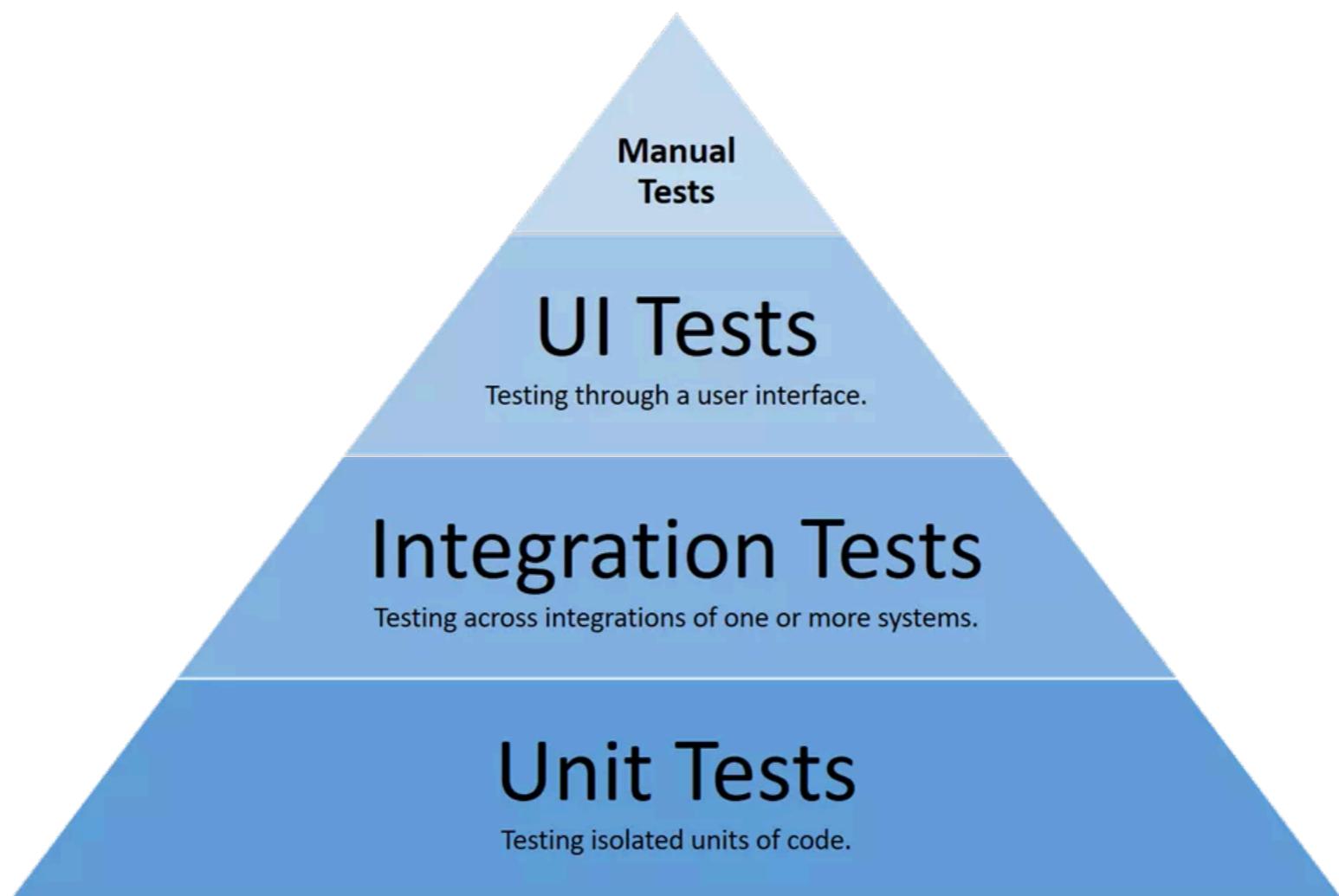
Automated environment for builds



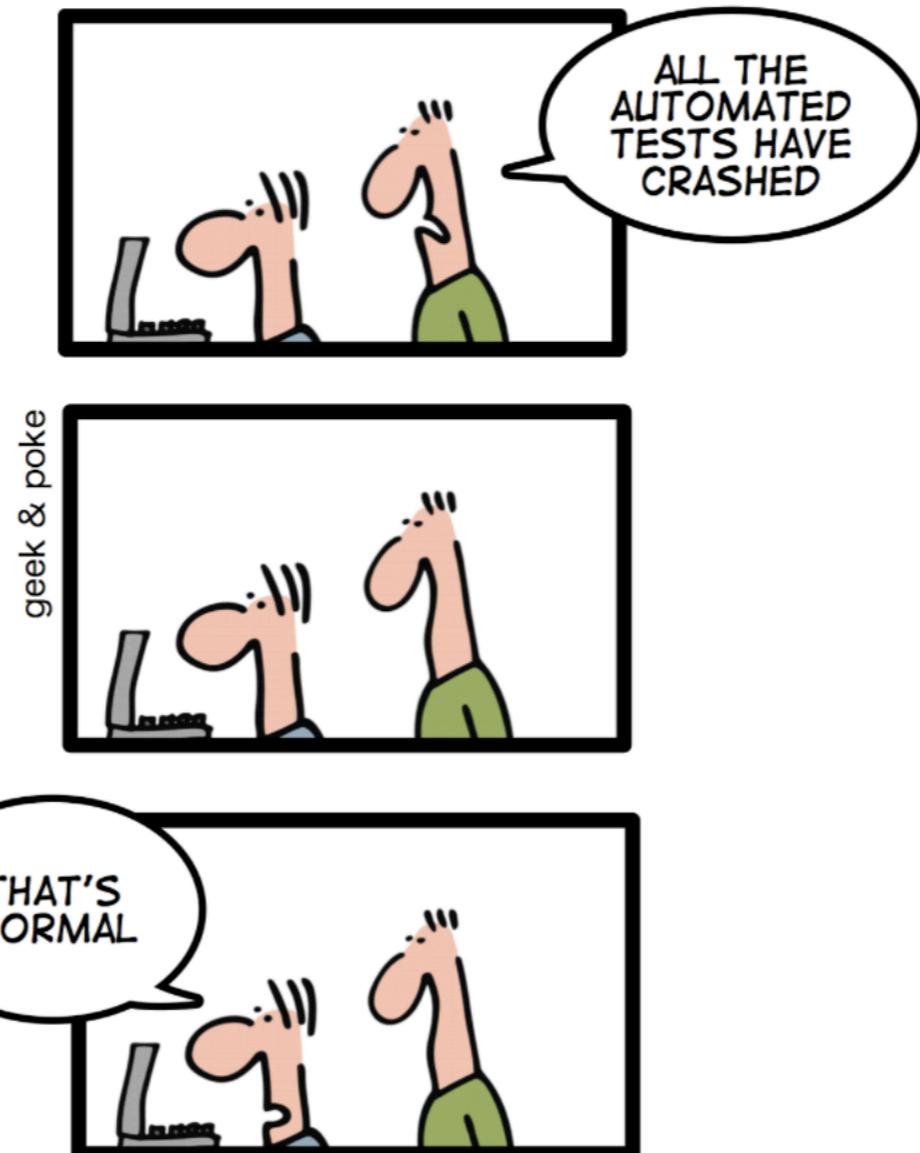
Practice 3

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*

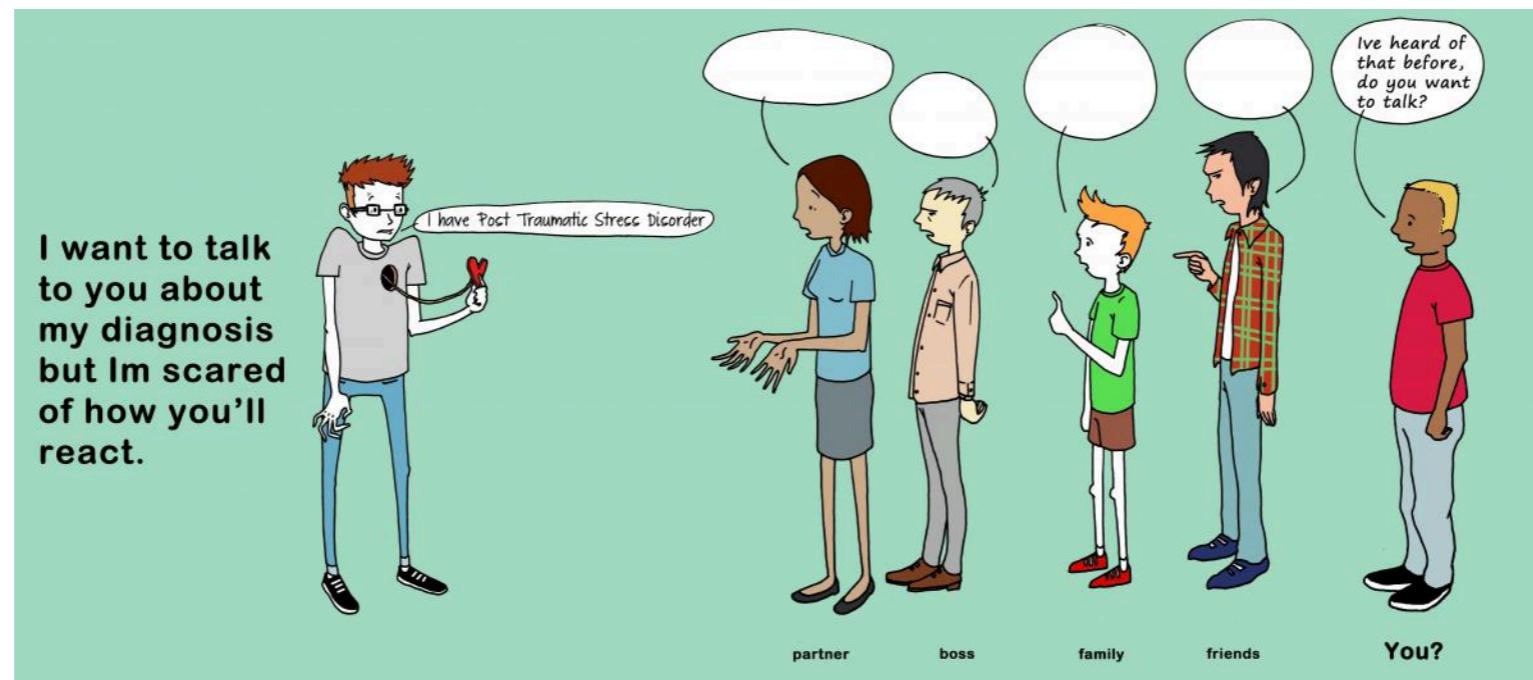


Practice 4

Everyone commits to the mainline everyday

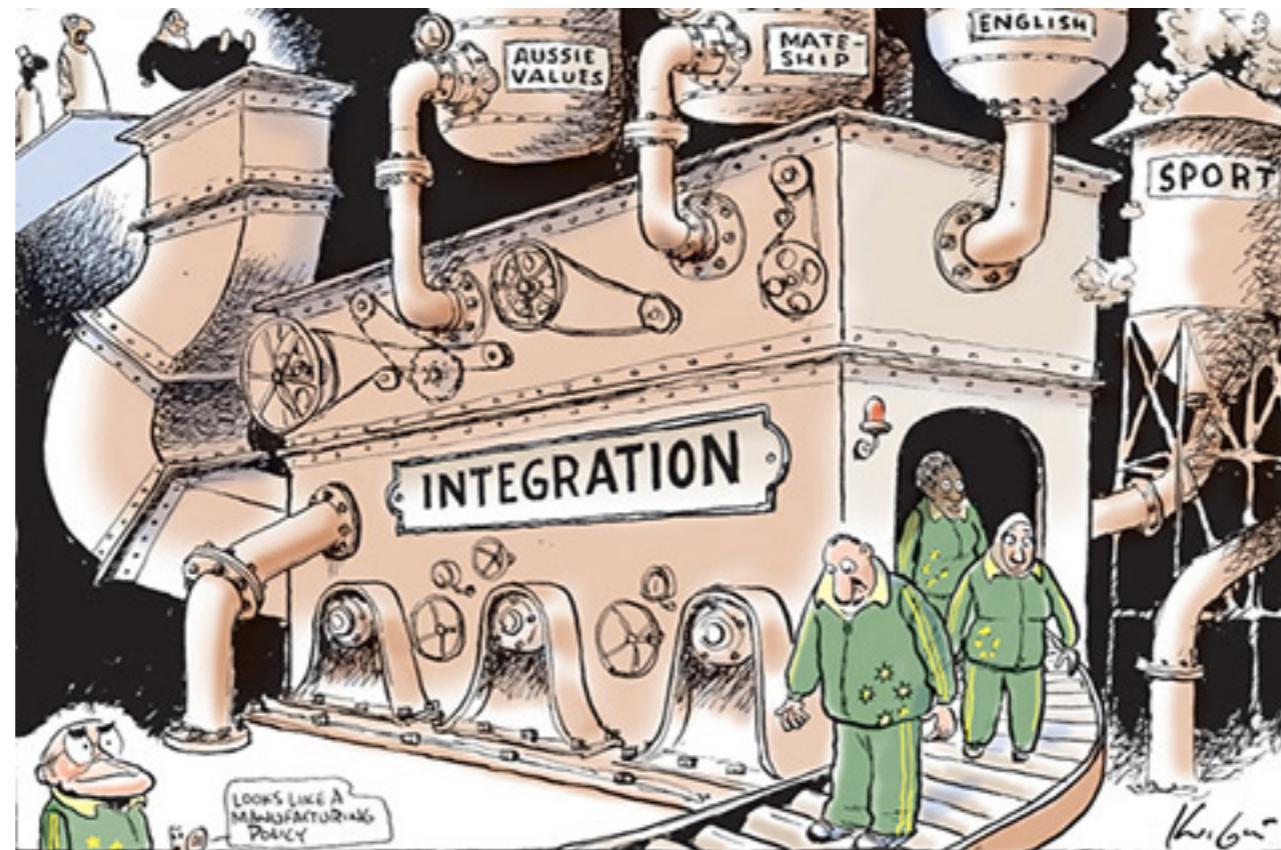
Integration is about communication

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

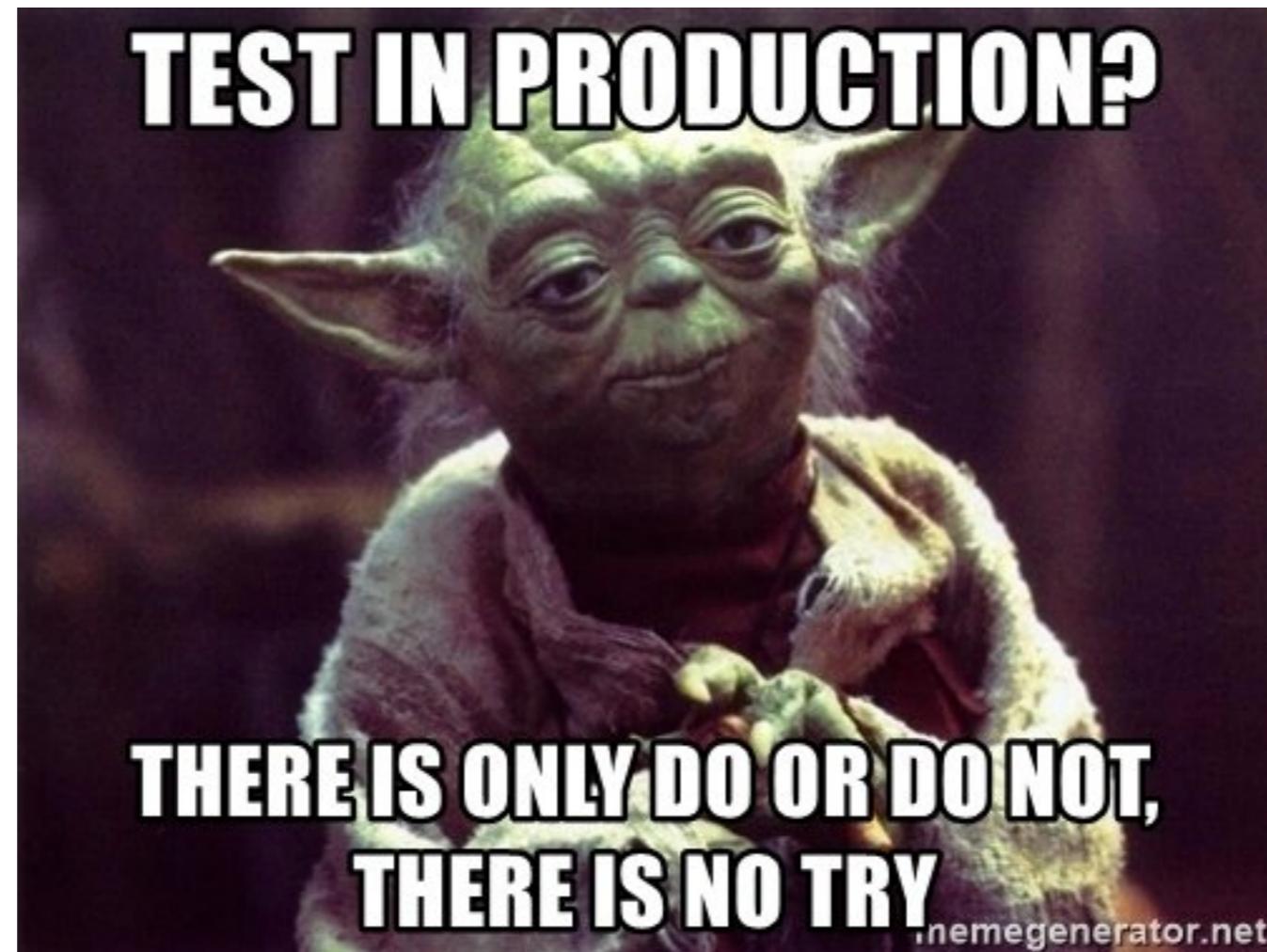
Keep the build **fast**

Continuous Integration is to provide rapid feedback



Practice 8

Test in clone of the **Production** environment



Practice 9

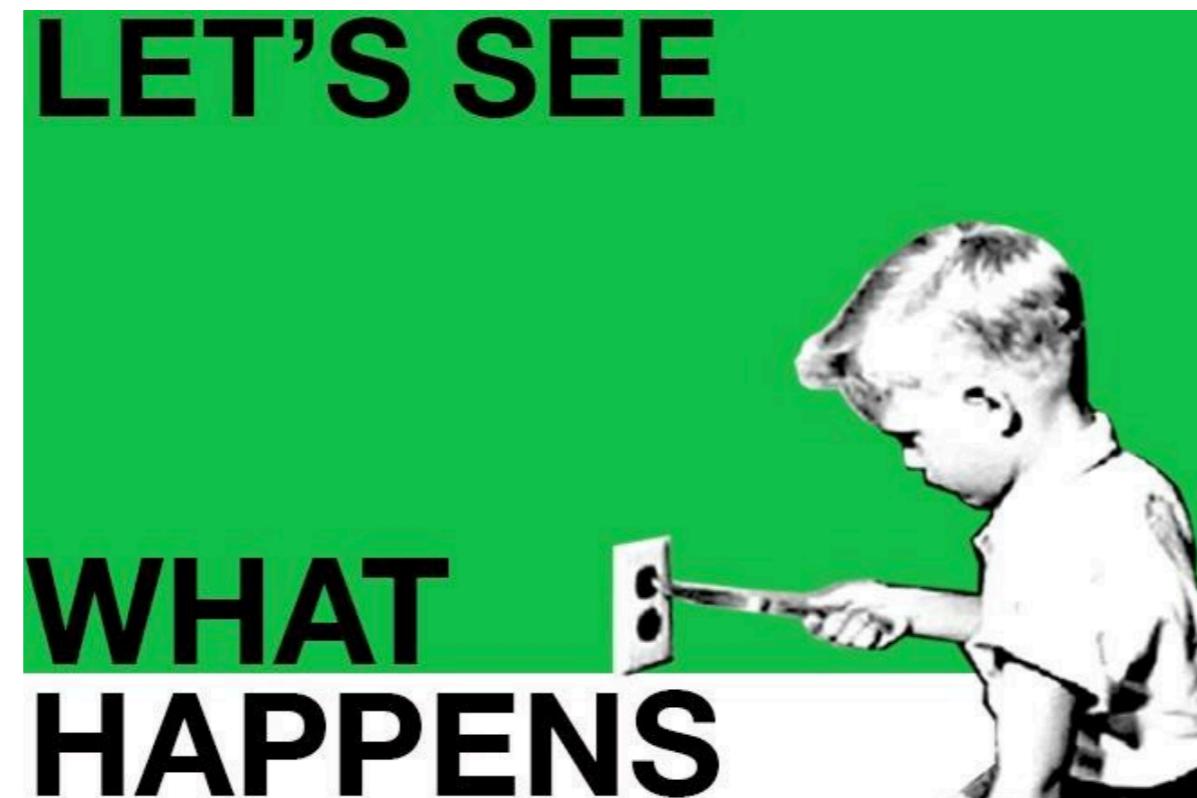
Make it easy for anyone to get
the latest executable

Make sure well known place where people can find



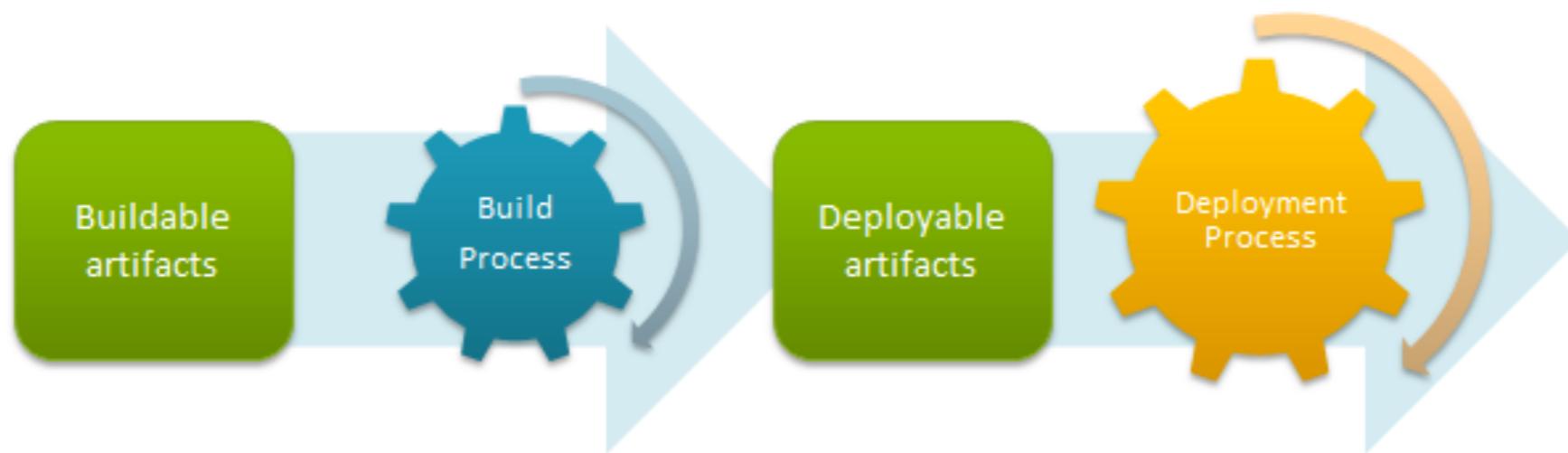
Practice 10

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



Goal

