Outline

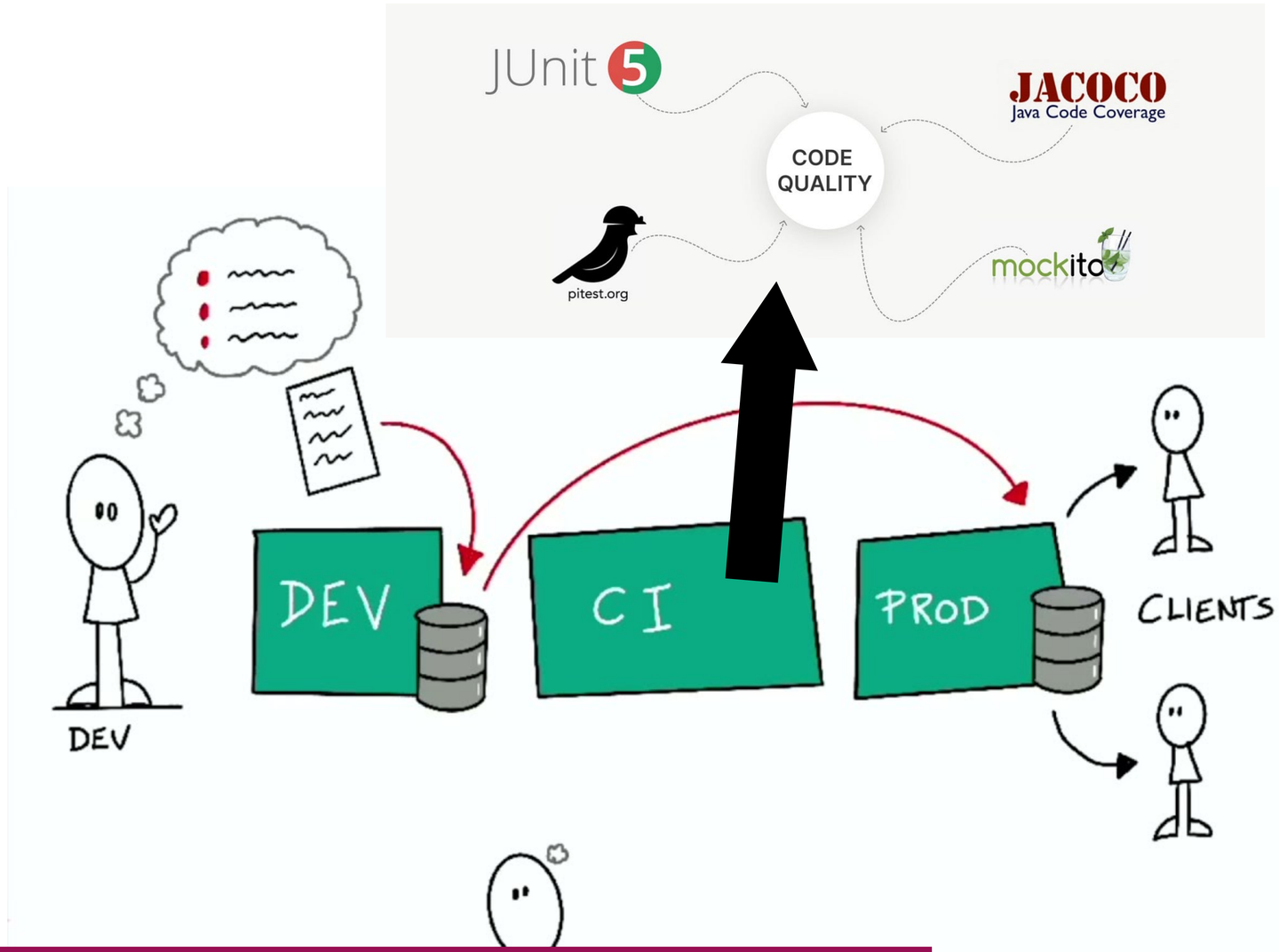
Planned topics for this lesson:

- Test Automation / Unit Testing

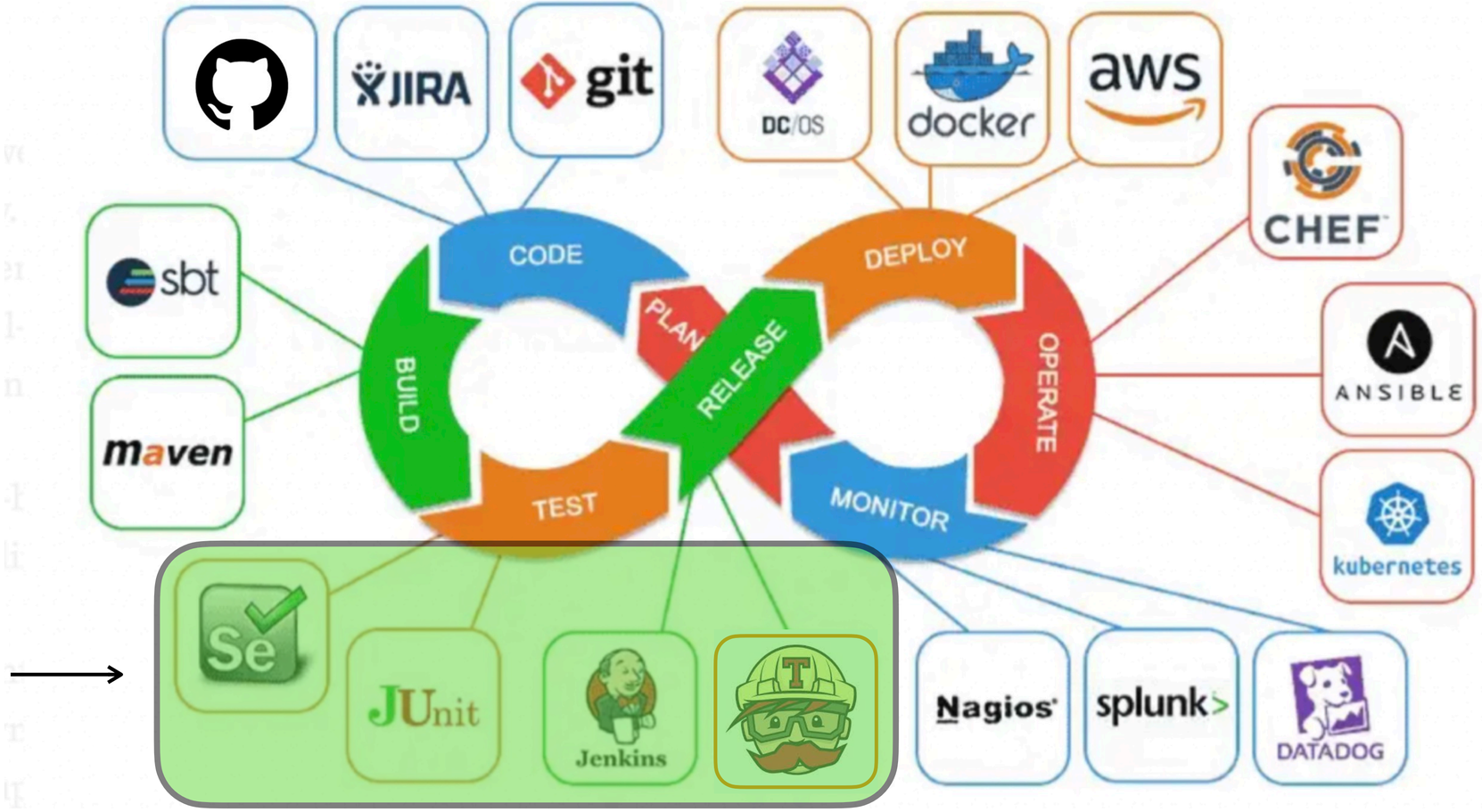
  What is it?
  via J-UNIT
  How to write?

- Test-Driven Development (TDD)

  What is it?
  Benefits?



What is Unit Testing?



This week : unit test →

What is Unit Testing?

- Unit testing is all about testing individual units of source code ← SMALLEST TESTABLE PARTS OF THE CODEBASE
- Use to check that your code is working as expected — It's called unit testing because you breakdown the functionality of your program into discrete testable behaviours that you can test as individual units
- Unit testing is a key feature of the test-driven development (TDD) approach to software development
- Developer writes a set of automated unit tests and ensures that they fail initially. Next the developer implements the bare minimum amount of code required to pass the test cases.

Key Benefit of Unit Testing

Early Bug Detection:

- Catching issues in small, isolated pieces of code makes it easier to fix bugs before they become more complicated.

Improved Code Quality:

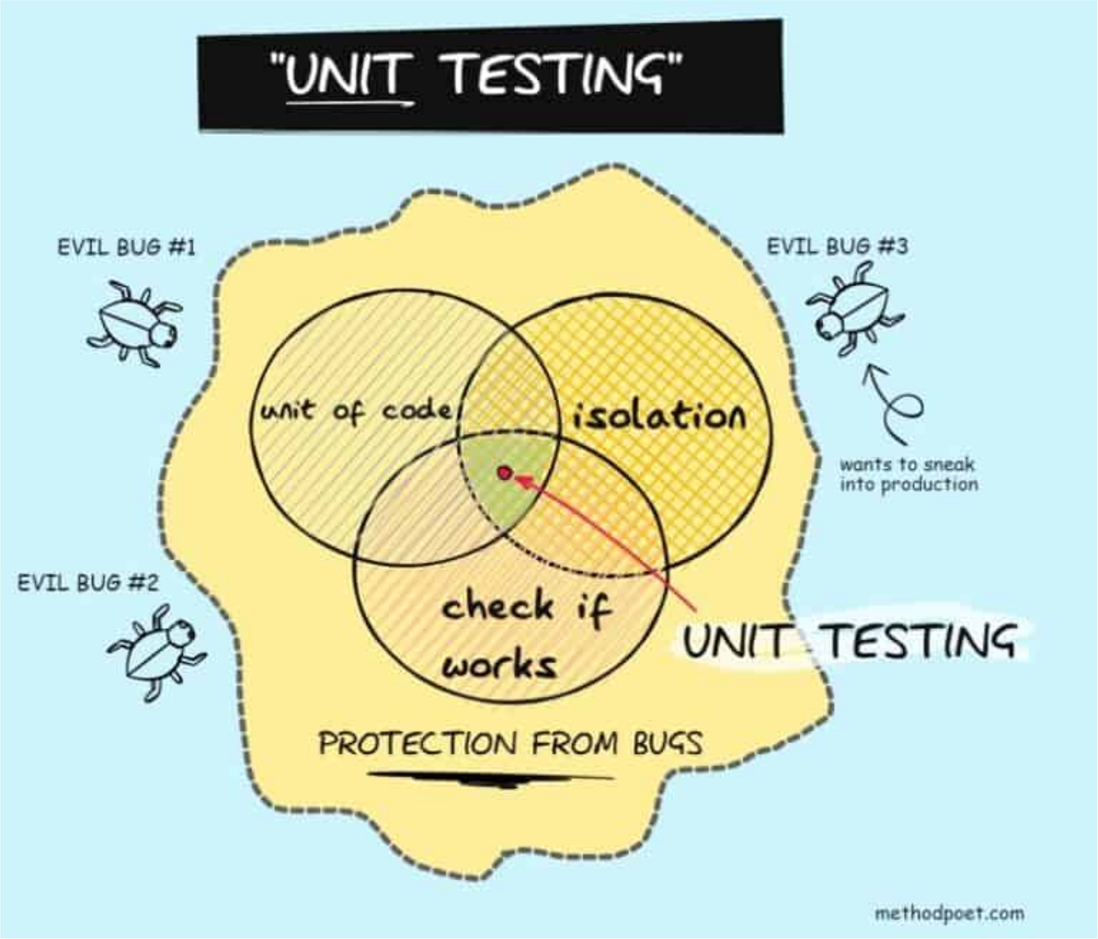
- Unit tests enforce modular code design and clean coding practices since units are developed to be testable.

Facilitates Refactoring:

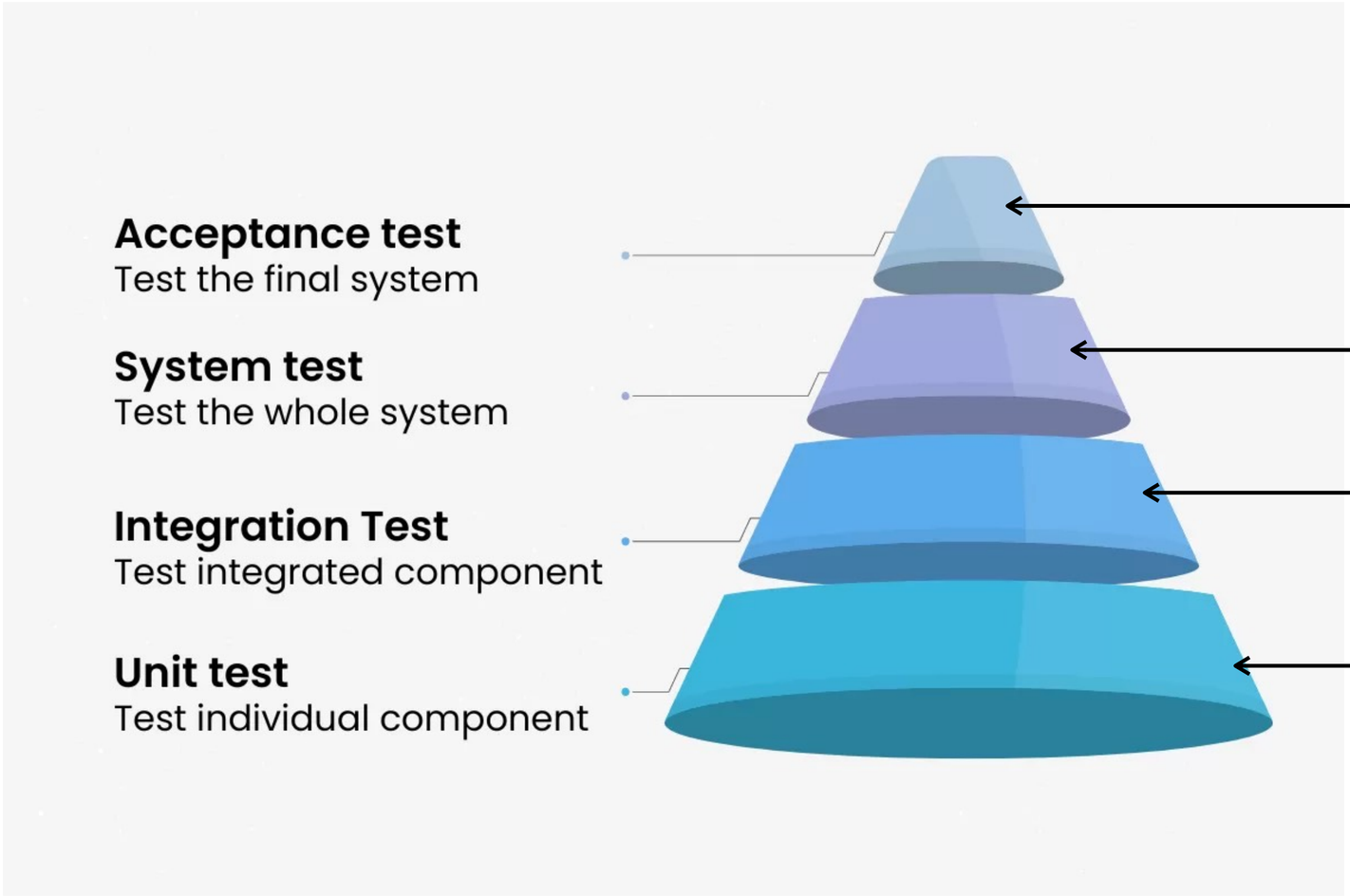
- As you refactor code, unit tests ensure that the original functionality remains intact.

Faster Development Cycle:

- Once tests are in place, they can be rerun quickly, ensuring that newly introduced code doesn't break existing features.



Types of Testing



ENSURES S/W MEETS USER REQUIREMENT

VERIFIES THE ENTIRE SYSTEMS'S FUNCTIONALITY

TESTS HOW DIFFERENT MODULES / SERVICES WORK TOGETHER

TESTING INDIVIDUAL FUNCTIONS OR METHODS

Principles of Good Unit Testing

Isolate the Unit:

- A unit test should only test one function or class without involving dependencies like databases or external APIs. Use mocks and stubs for dependencies.

Repeatability:

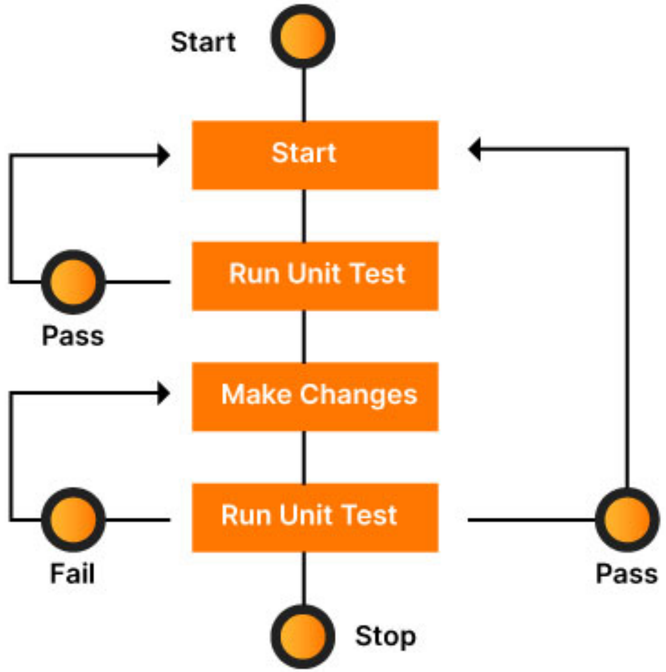
- Unit tests should produce the same results each time they run, no matter the environment or order.

Fast Execution:

- Unit tests should execute quickly to allow for continuous feedback during development.

Independent Tests:

- Each unit test should be independent, meaning tests should not rely on the order of execution or shared state.



Anatomy of a Unit Test

```
[TestMethod]
public void GetSumTest()
{
    /// Arrange
    DemoClass demoClass = new DemoClass();
    int firstValue = 5;
    int secondValue = 6;
    int expectedSumResult = 11;
    int actualSumResult = default(int);

    /// Act
    actualSumResult   = demoClass.GetSum(firstValue, secondValue);

    /// Assert
    Assert.AreEqual(expectedSumResult, actualSumResult);
}
```

Set up the context and inputs

Perform the action being tested (e.g., call the method)

Check if he result matches the expected outcome

Best Practices in Unit Testing

- **Keep Tests Small and Focused:**

  A single test should only validate one behaviour or scenario.

- **Use Descriptive Names:**

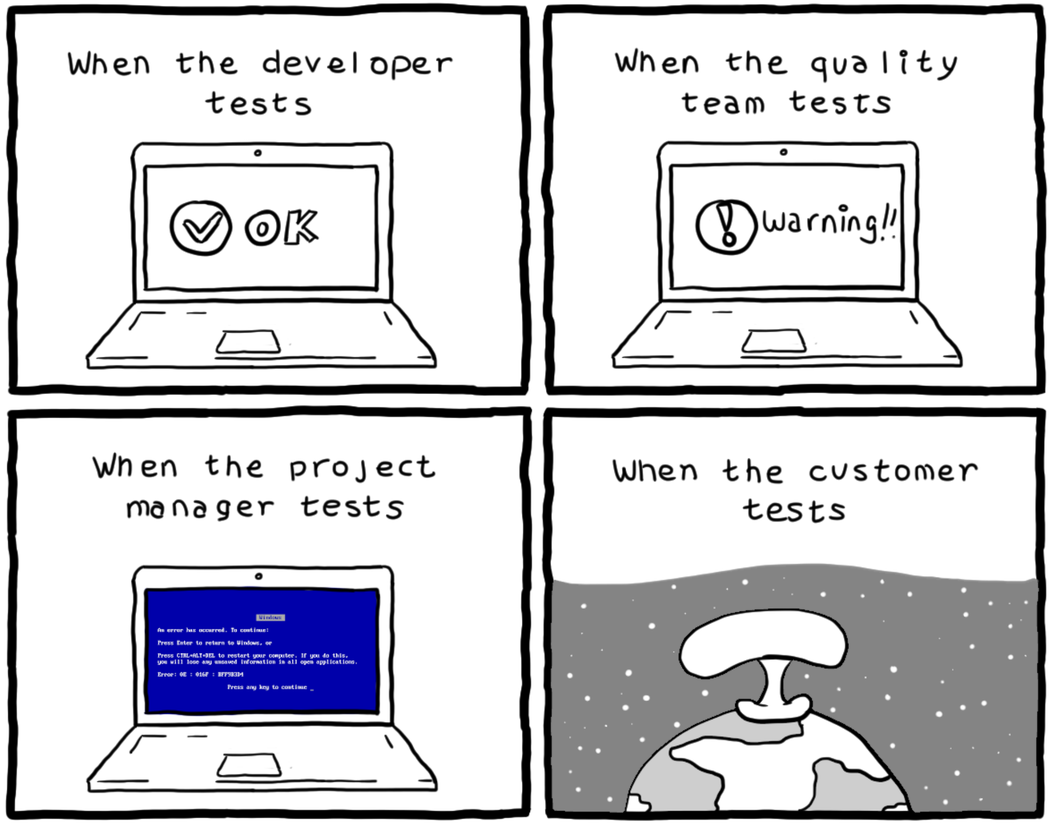
  Test names should explain exactly what behaviour is being verified.

- **Avoid Over-Mocking:**

  While mocks are useful for isolating the unit, overusing them can lead to brittle tests.

- **Test Edge Cases:**

  Don’t just test the "happy path" but also consider boundary values and error scenarios.



Unit Testing in CI/CD

Automation in CI/CD Pipelines:

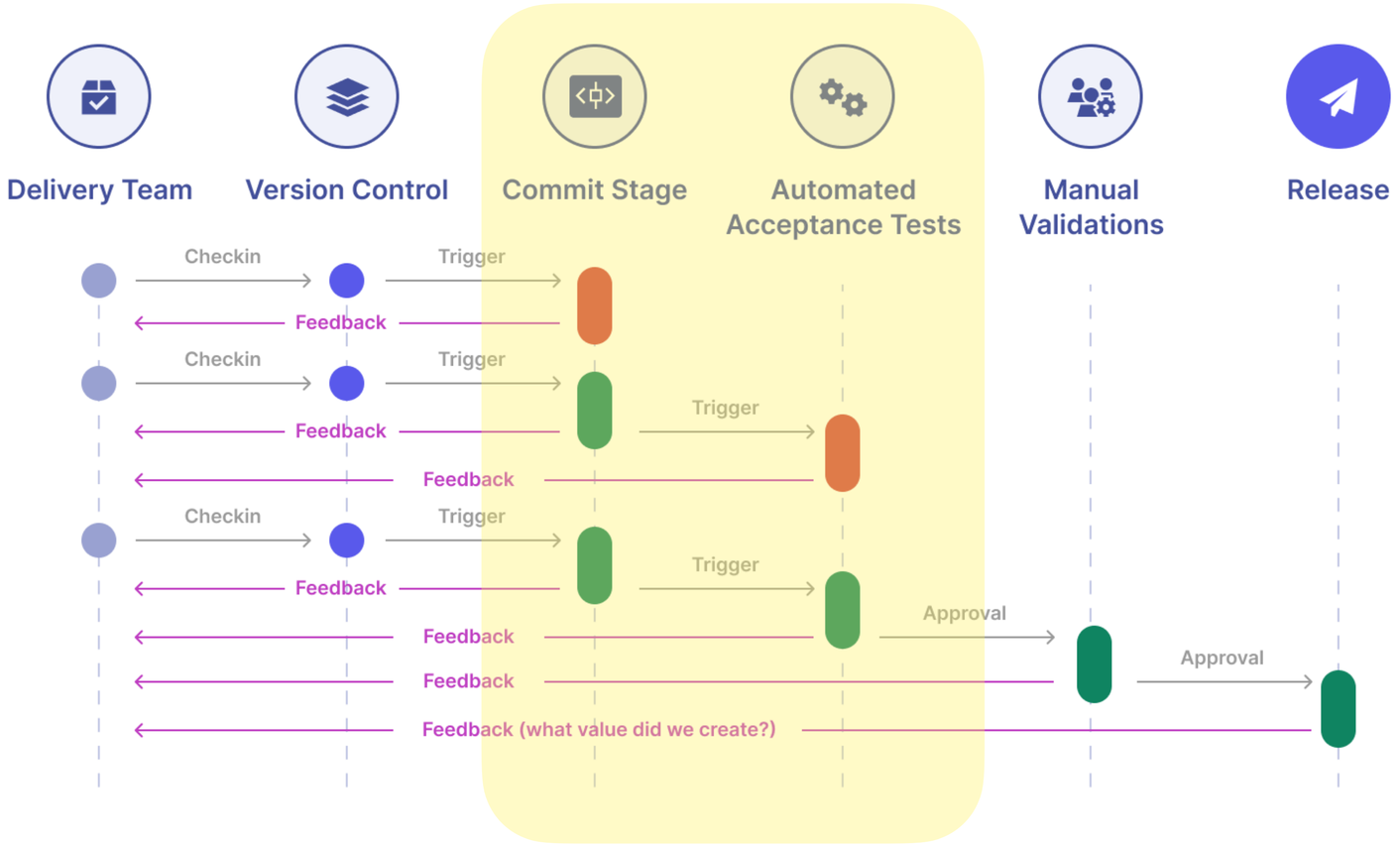
- Unit tests are automatically executed as part of the Continuous Integration (CI) process to ensure new code does not introduce regressions.

Fast Feedback Loop:

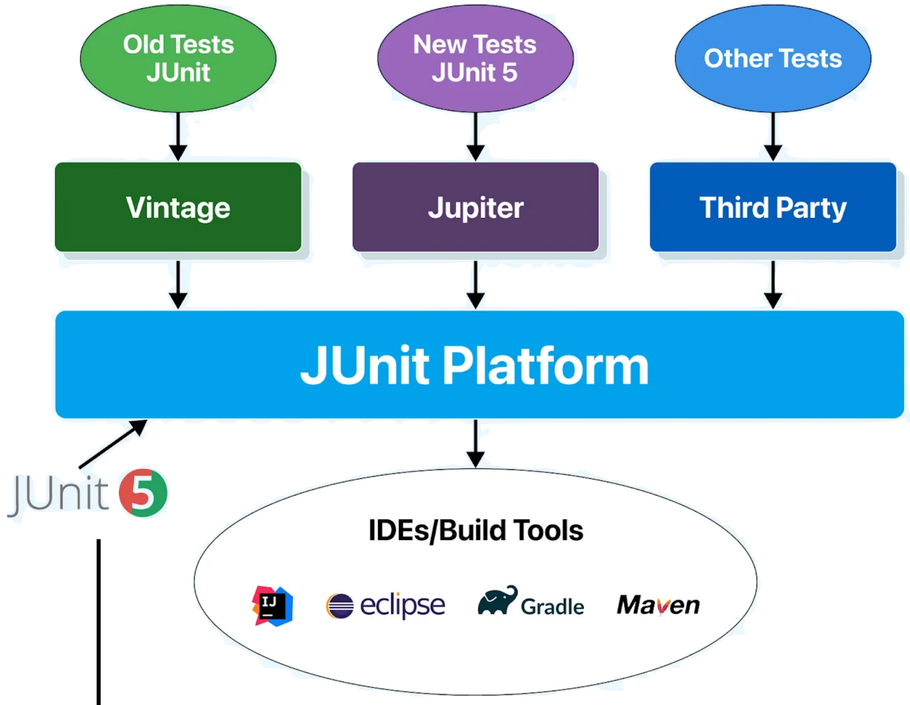
- When developers push code, unit tests immediately verify whether the change breaks any functionality, allowing for quick fixes.

Key Metrics:

- **Code Coverage:** Measures how much of the code is covered by unit tests.
- High coverage improves confidence, but coverage is not the only measure of quality.
- **Flakiness:** Unit tests should be reliable.
- Flaky tests that randomly fail can create confusion.



Unit Testing Tools



MOST POPULAR TESTING FRAMEWORK FOR JAVA APPS

FOR PYTHON, C# AND JS

pytest

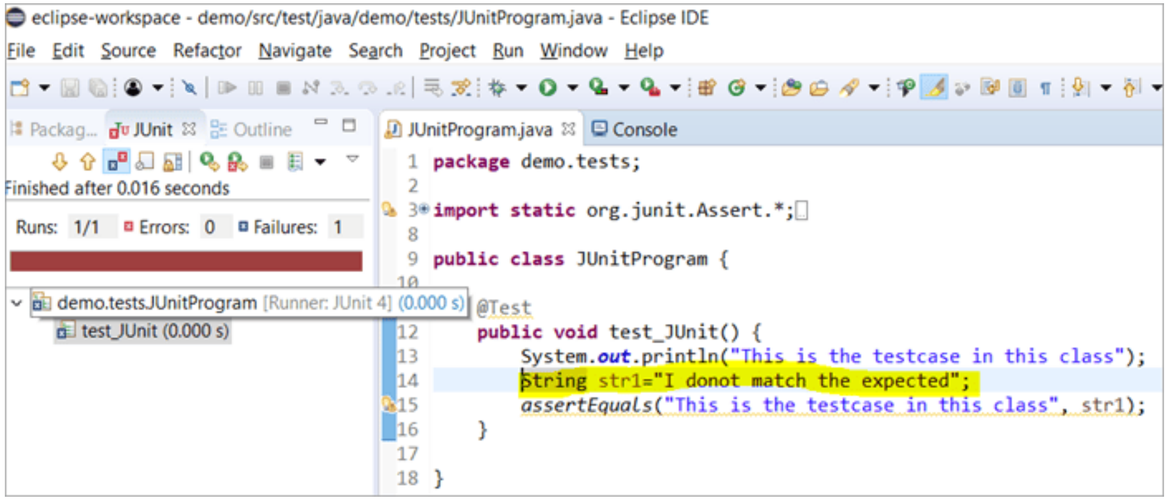
Jest

mockito

A MOCKING FRAMEWORK USED TO MOCK DEPENDENCIES AND ISOLATE UNITS DURING TESTING

via J-Unit

JUnit



- In Java, you can write a unit test to check the behaviour of a number of methods of a single class, however, it should remain within one class (i.e., multiple test cases applied to a single class)
- Test code can be run and evaluated automatically (test automation)
- Similar tools exist for C, C#, js, and other languages

via J-Unit

Example

```
public Class Calculator {
   public int evaluate(String expression) {
      int sum = 0;
      for (String summand: expression.split("\\+"))
         sum += Integer.valueOf(summand);
      return sum;
   }
}
```

```
import static org.junit.Assert.*;
import org.junit.Test;

public class CalculatorTest {
   @Test
   public void evaluatesExpression() {
      Calculator calculator = new Calculator();
      int sum = calculator.evaluate("1+2+3");
      assertEquals(6, sum);
   }
}
```

- JUnit resources:
  - https://github.com/junit-team
  - https://www.vogella.com/tutorials/JUnit/article.html
  - https://www.baeldung.com/junit-5
  - https://mvnrepository.com/artifact/junit/junit