

# Introduction to Unit Tests

My Reaction



When someone says testing is easy

# Agenda

---

- Motivation
- Case study: NUnit
- Testing
  - Basic parts of a unit tests
  - What does a unit test look like?

# Motivation: Manually test

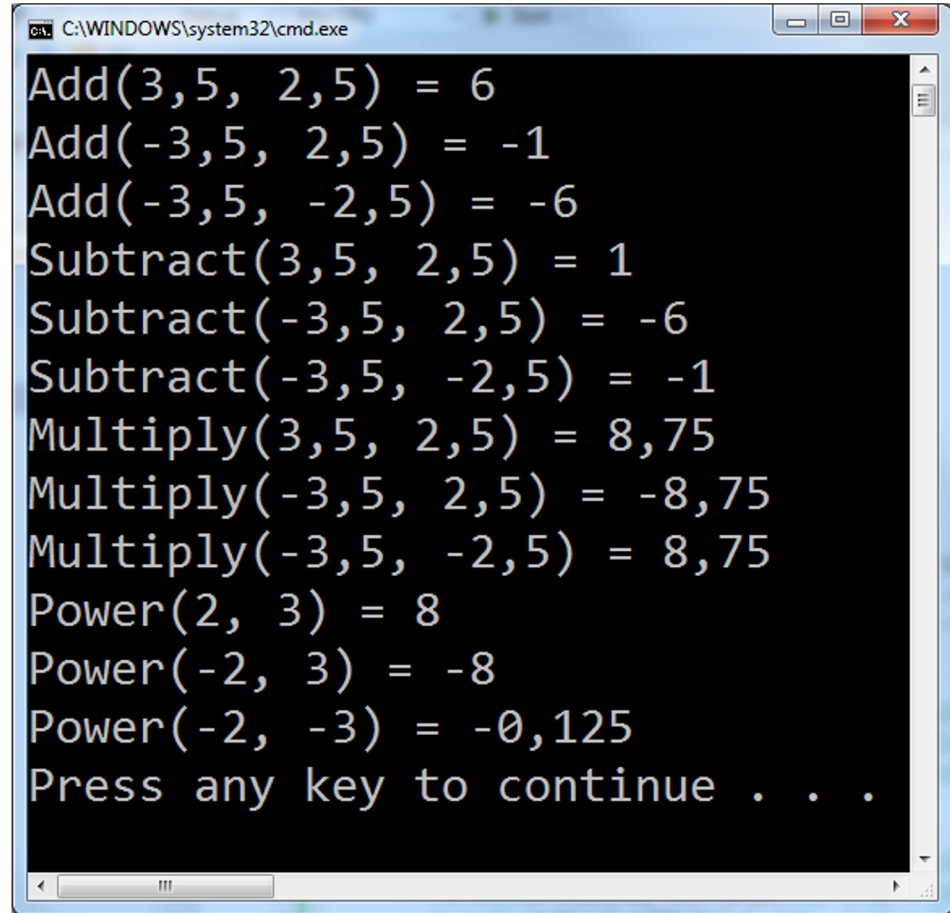
```
public class Calculator {  
    public double Add(double a, double b) {  
        return a + b;  
    }  
  
    public double Subtract(double a, double b) {  
        return a - b;  
    }  
  
    public double Multiply(double a, double b) {  
        return a * b;  
    }  
  
    public double Power(double a, double b) {  
        return Math.Pow(a,b);  
    }  
}
```

```
class Program  
{  
    static void Main(string[] args)  
    {  
        // Declare the unit-under-test  
        var uut = new Calculator();  
  
        // Test Add()  
        Console.WriteLine("Add({0}, {1}) = {2}", 3.5, 2.5, uut.Add(3.5, 2.5));  
        Console.WriteLine("Add({0}, {1}) = {2}", -3.5, 2.5, uut.Add(-3.5, 2.5));  
        Console.WriteLine("Add({0}, {1}) = {2}", -3.5, -2.5, uut.Add(-3.5, -2.5));  
  
        // Test Subtract()  
        Console.WriteLine("Subtract({0}, {1}) = {2}", 3.5, 2.5, uut.Subtract(3.5, 2.5));  
        Console.WriteLine("Subtract({0}, {1}) = {2}", -3.5, 2.5, uut.Subtract(-3.5, 2.5));  
        Console.WriteLine("Subtract({0}, {1}) = {2}", -3.5, -2.5, uut.Subtract(-3.5, -2.5));  
  
        // Test Multiply()  
        Console.WriteLine("Multiply({0}, {1}) = {2}", 3.5, 2.5, uut.Multiply(3.5, 2.5));  
        Console.WriteLine("Multiply({0}, {1}) = {2}", -3.5, 2.5, uut.Multiply(-3.5, 2.5));  
        Console.WriteLine("Multiply({0}, {1}) = {2}", -3.5, -2.5, uut.Multiply(-3.5, -2.5));  
  
        // Test Power()  
        Console.WriteLine("Power({0}, {1}) = {2}", 2.0, 3.0, uut.Power(2.0, 3.0));  
        Console.WriteLine("Power({0}, {1}) = {2}", -2.0, 3.0, uut.Power(-2.0, 3.0));  
        Console.WriteLine("Power({0}, {1}) = {2}", -2.0, -3.0, uut.Power(-2.0, -3.0));  
    }  
}
```



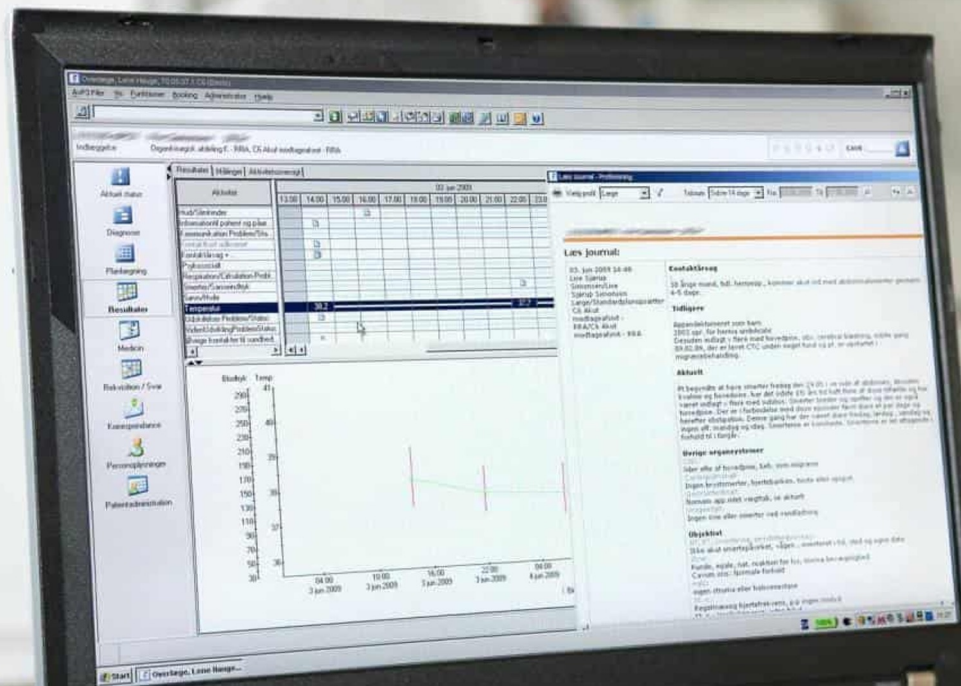
# Validate result?

---



A screenshot of a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The window has a black background with white text. It displays a series of mathematical calculations, each on a new line. The calculations are:  $\text{Add}(3, 5, 2, 5) = 6$ ,  $\text{Add}(-3, 5, 2, 5) = -1$ ,  $\text{Add}(-3, 5, -2, 5) = -6$ ,  $\text{Subtract}(3, 5, 2, 5) = 1$ ,  $\text{Subtract}(-3, 5, 2, 5) = -6$ ,  $\text{Subtract}(-3, 5, -2, 5) = -1$ ,  $\text{Multiply}(3, 5, 2, 5) = 8,75$ ,  $\text{Multiply}(-3, 5, 2, 5) = -8,75$ ,  $\text{Multiply}(-3, 5, -2, 5) = 8,75$ ,  $\text{Power}(2, 3) = 8$ ,  $\text{Power}(-2, 3) = -8$ , and  $\text{Power}(-2, -3) = -0,125$ . The last line of the output is "Press any key to continue . . .". The window has standard Windows window controls (minimize, maximize, close) in the top right corner and a scrollbar on the right side.

```
C:\WINDOWS\system32\cmd.exe
Add(3,5, 2,5) = 6
Add(-3,5, 2,5) = -1
Add(-3,5, -2,5) = -6
Subtract(3,5, 2,5) = 1
Subtract(-3,5, 2,5) = -6
Subtract(-3,5, -2,5) = -1
Multiply(3,5, 2,5) = 8,75
Multiply(-3,5, 2,5) = -8,75
Multiply(-3,5, -2,5) = 8,75
Power(2, 3) = 8
Power(-2, 3) = -8
Power(-2, -3) = -0,125
Press any key to continue . . .
```



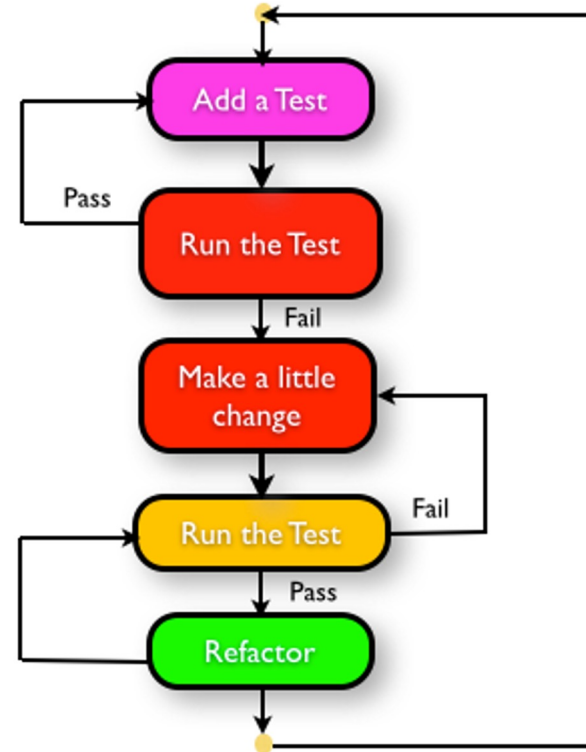
# Test also gives...

---

- specifications built into the program
- confidence in code
- early error finding
- decoupled system
- better design

# How to plan and execute test

1. Define a scenario
2. Write the [TestCase]
3. Run the test
4. Implement the code
5. Repeat



# Demo: A cash register

---

- We will implement and test a class CashRegister

| CashRegister             |
|--------------------------|
| + AddItem(price: double) |
| + GetNItems(): int       |
| + GetTotal(): double     |



- Your turn: What test cases do we need for each of the class' methods?
  - What is the scenario?
  - What is the test input?
  - What is the expected result?



# Your turn

---

For around 30 minutes do

1. Do **'Exercise 1: Plan your tests'**
2. Continue to **'Exercise 2: Prepare workspace'** when you are done with 1.



—

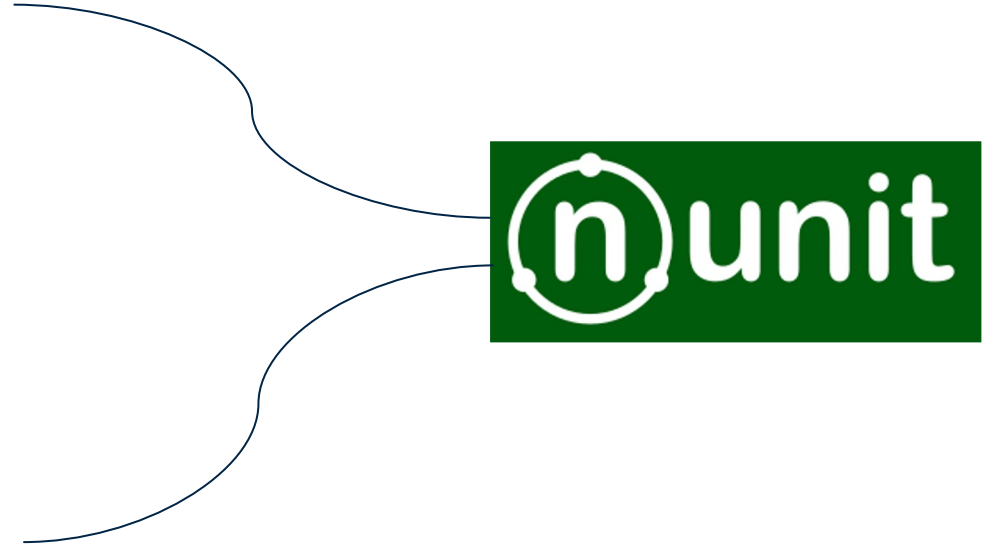
# NUnit

# Test framework

---

Gives:

- Support for automation
- Easy setup and removal
- Good assertion constructs
- Detailed test reports
- Nice IDE integration
- Testing styles



# Automation / test report

The screenshot displays the Visual Studio Test Explorer on the left and the Code Editor on the right. The Test Explorer shows a list of tests under 'MultiTest (26)'. The tests are categorized by status: failed (red X), warning (yellow triangle), and passed (green checkmark). The 'TestMethodN11' test is highlighted with a red arrow. The Code Editor shows the source code for 'MSUnitTestBase'.

**Test Results:**

| Test Name           | Duration | Status  |
|---------------------|----------|---------|
| TestMethodMSOutput1 | 19 ms    | Failed  |
| TestMethodMs4       |          | Warning |
| TestMethodMs5       |          | Warning |
| AsyncTestMS1        | 1 ms     | Passed  |
| BaseTestMethod1     | < 1 ms   | Passed  |
| DerivedTestMethod   | < 1 ms   | Passed  |
| NameTest            | 3 ms     | Passed  |
| TestAddInNet4       | 2 ms     | Passed  |
| TestMethodMs1       | 2 sec    | Passed  |
| TestMethodMs11      | 201 ms   | Passed  |
| TestMethodMs12      | < 1 ms   | Passed  |
| TestMethodMs2       | 2 sec    | Passed  |
| TestMethodMs3       | 3 sec    | Passed  |
| TestMethodMs6       | < 1 ms   | Passed  |
| TestMethodMSOutput2 | < 1 ms   | Passed  |
| TestMethodN1        |          | Passed  |
| TestMethodN11       |          | Passed  |
| TestMethodN12       |          | Passed  |
| TestMethodN2        |          | Passed  |
| TestMethodN3        |          | Passed  |
| TestMethodX1        |          | Passed  |

**Source Code:**

```
[TestClass]
1 reference
abstract public class MSUnitTestBase
{
    [TestMethod]
    0 references
    public void BaseTestMethod1()
    {
        Assert.IsTrue(true);
    }
}
```

**Output:**

Show output from: Tests

----- Run test started -----  
===== Run test finished: 18 run (0:00:07,9559915) =====

# Nunit example

---

```
[TestFixture]
public class UnitTest1 {
    Calculator uut;
    [Setup]
    public void Setup() {
        uut = new Calculator();
    }

    [Test]
    public void Test_AddMethod() {
        double res = uut.Add(3.5, 2.5);
        Assert.AreEqual(res, 6);
    }
    ...
}
```



# NUnit assertion - constraints

- Constraint based assert model

```
Assert.That(uut.Count, Is.EqualTo(10));
```

Actual state

Constraint with  
expected state

- Other constraints examples - many more exists

```
Assert.That(uut.Count, Is.GreaterThan(3));  
Assert.That(myString, Is.EqualTo("Hello"));  
Assert.That(array, Has.Exactly(3).LessThan(100));
```

More constraints <https://nunit.org/docs/2.5/constraintModel.html>

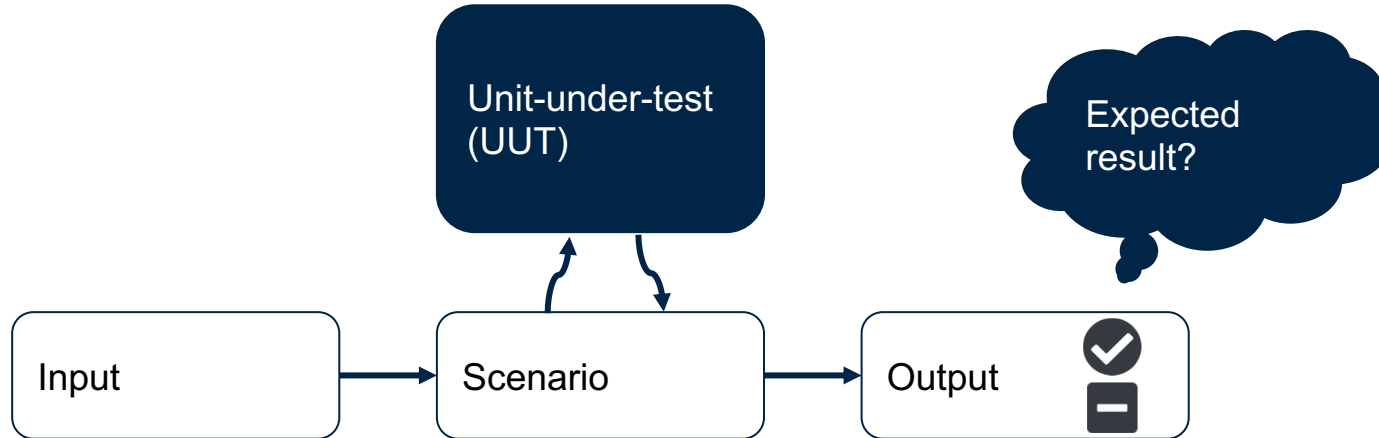
# Test framework

---

- **Assertions** Different ways to compare expected and actual test results in a readable way
- **Test case** Each test case - tests one specific, isolated aspect of the unit-under-test
- **Test fixture** Collects test cases, helps with setup, teardown, etc.
- **Test runner** Runs the tests and reports the result
- **Test reports** Result of the tests run

# A unit tests

---



Icons: FrICONiX.com



# Demo: A cash register

---

- Let us implement the CashRegister

| CashRegister   |
|--|
| + AddItem(price: double)<br>+ GetNItems(): int<br>+ GetTotal(): double |



# Your turn

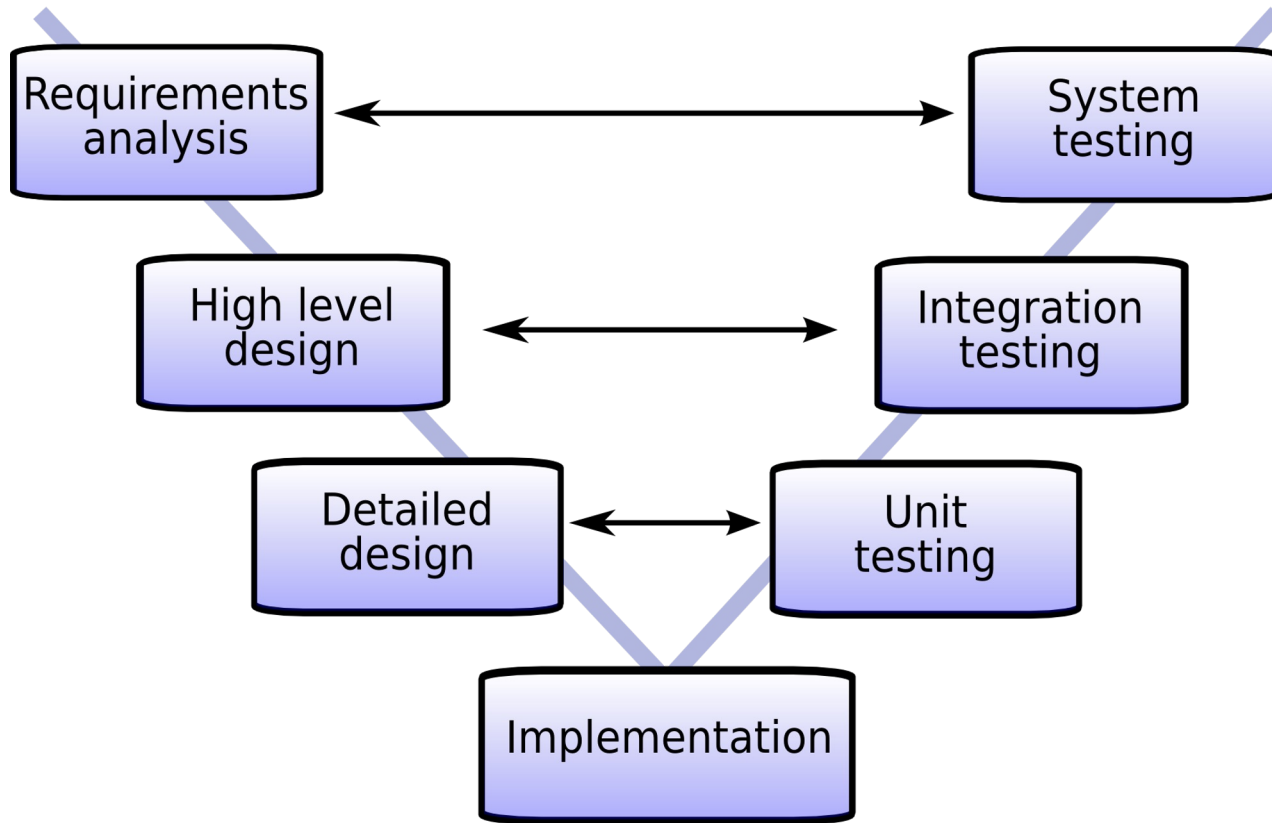
---

Continue with exercises

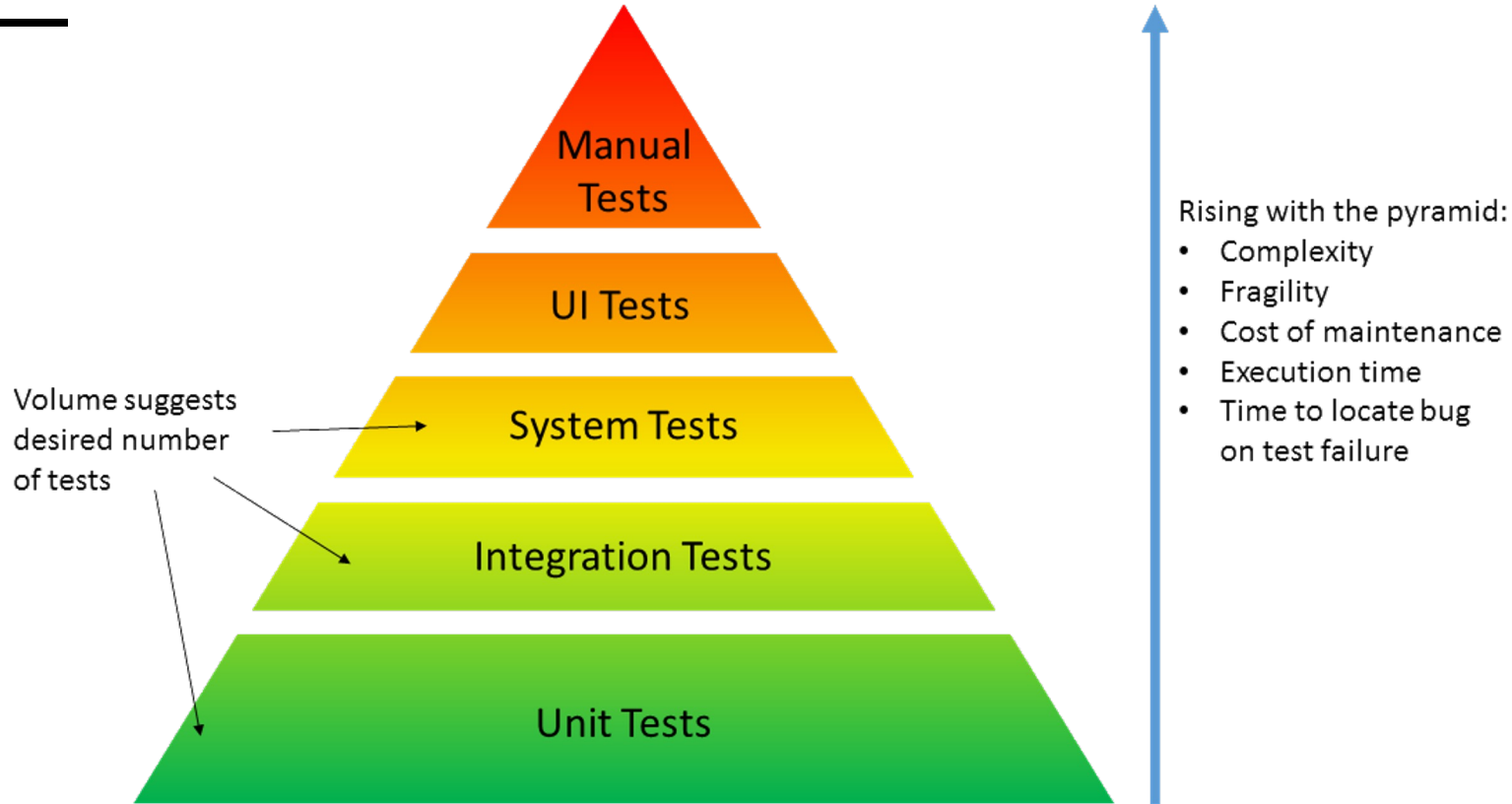


---

# Testing and Pitfalls

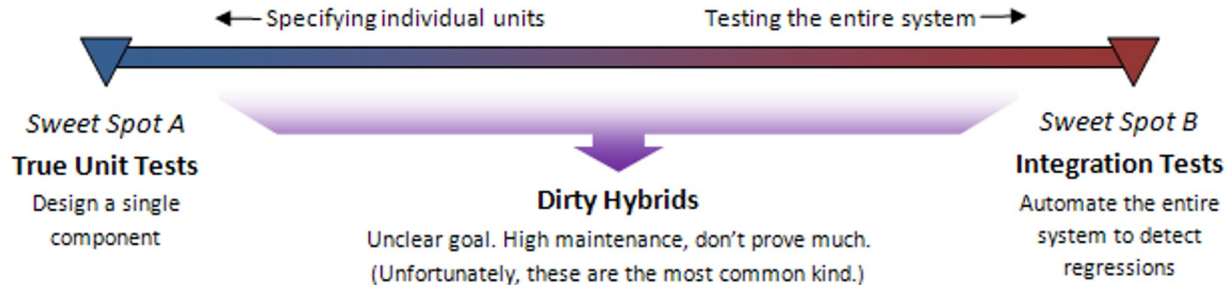


# Why test? Why unit test?



# Pitfalls

- True unit test contains information about design and behaviour of UUT (Unit-under-test)
  - **Do not** make any assumption about other parts
- Integration test **do not** tell anything about how code base is broken down into units
  - Make assumptions about the whole system behaviour
- In between
  - Small changes breaks unrelated test
  - Tests breaks - but system works as “expected”





# Reference

---

TDD: [agilefaqs.com/services/training/test-driven-development](https://agilefaqs.com/services/training/test-driven-development)

Dancing man: <https://giphy.com>