Unit Tests

Testing

Even if you have never thought about testing, you have already done it

- Build, run
- Data input
- Checking the result

You manually worked through test scenarios

- Debugging

Manual code testing

Unreliable

- It's easy to forget to check something

Labor intensive

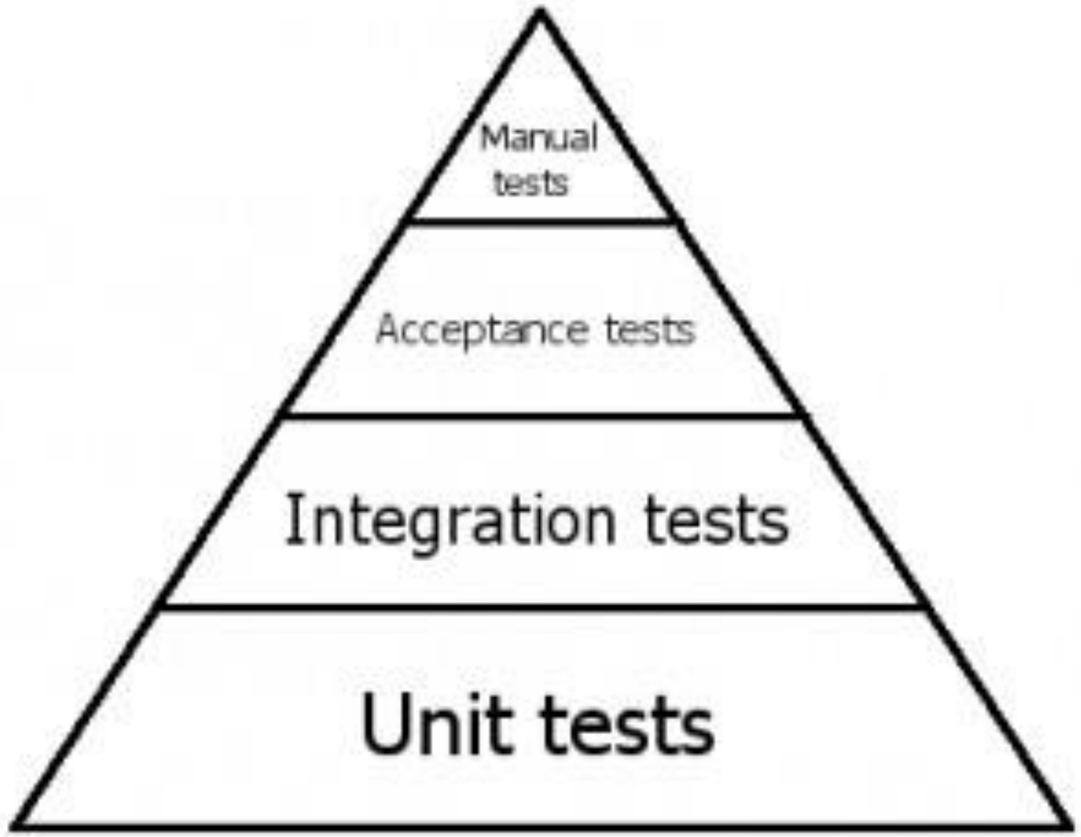
- With all changes, you need to check the performance
- Difficult to track down regression errors

So you need automated testing

Types of testing

By the level of isolation of components

- **Unit testing**
  - A single isolated component (method or class) is being tested
- **Integration testing**
  - The joint work of the components is being tested
- **System testing**
  - Testing the entire application as a black box (alpha, beta)



Unit testing

- First frontier in the fight against bugs
- Implemented by developers
- Checks isolated non-trivial modules
- Detects regression errors
- Less time performing functional tests
- Less coupled code

Test development

- Allows the project not to depend on people (see Bus factor)
- Allows you to get live documentation

- But it takes time and effort to develop and maintain

Code to be tested (requirements)

Functionality should be isolated

- I/O separate from logic/calculation
- One function in one method
- Global State Independence
- Separate functional modules into separate libraries

Test Requirements

- Must be fast
- Must not have access to real data
- Must be repeatable
- Must cover all code with non-trivial functionality
  - See Code coverage for tests.

Standard Tests

Microsoft.VisualStudio.TestTools.UnitTesting

Language Tools
- Unit test project
- Test classes `[TestClass]`
- Methods - Scripts `[TestMethod]`

Development environment tools
- Test Explorer
- Code coverage analysis

Writing Tests

Arrange – Act – Assert

- Test data initialization
- Performing the action under test
- Checking the result (postconditions)

Assert

Assert Classes
- Assert – to compare objects
- CollectionAssert – for comparison collection
- StringAssert – for string comparison

Attribute [ExpectedException]

Recommendations

- Tests should have "talking" names
  - Method_Scenario_ExpectedBehavior
    - GCD_108and42_Return12
- It will be better if you keep your unit tests in a separate project from your integration tests
- Try to write tests before implementing the functionality (see TDD, test first)
  - Interface -> tests -> implementation
    - Stub methods -> tests -> implementation
- Avoid multiple acts
- Validate private methods by unit testing public methods
- Avoid logic in tests
- Avoid magic strings

Test example

```
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace Tests
{
    public class ArithmeticTests
    {
        [Fact]
        public void GCD_108and48_Return12()
        {
            int v = Arithmetic.GCD(108, 48);
            Assert.AreEqual(v, 12);
        }
    }
}
```