The Sector Specialists

Unit Testing and Test Driven Development

Prepared by: Wojciech SICIŃSKI
Version: 1.0

Agenda

Agenda

What is Unit Test

- Unit

  Method or function

- Unit test

  a piece of code created for checking correctness another piece of code (unit)

- Pass, Error, Failure

Why to use Unit Test

- Easy bug locating

  Bugs in units may „overlap” eachother in later testing (integration, system)

- Thorough functionality testing

  every possible scenario should have unit test

- State of development

- Automation

  unit test should be repeatable, so it will be easy to test regression

When to use Unit Test

- When building functionality

  Bugs in units may „overlap” eachother in later testing (integration, system)

- When found a bug

  every possible scenario should have unit test

- Code coverage

  Test should „visit” every line of tested code, ideally (exceptions)

Agenda

- Unit Tests
- **Unit Tests in .NET**
- Code coverage
- Unit tests vs. Integration tests
- Test Driven Development

Unit tests in .NET

- MSUnit
- xUnit
- Nunit
  - Ported from java
  - Well known and supported

How to write UnitTest

```
public class Account
{
              private decimal balance;
              public void Deposit(decimal amount) {
                            balance += amount;
              }
              public void Withdraw(decimal amount) {
                            balance -= amount;
              }
              public void TransferFunds(Account destination, decimal amount) {
                            destination.Deposit(amount);
                            Withdraw(amount);
              }
              public decimal Balance {
                            get { return balance; }
              }
}
```

How to write UnitTest

- TestFixture
- Test
- SetUp
- TearDown
- Assertions
- Test method attributes
- TestFixtureSetUp
- TestFixtureTearDown

```
[TestFixture]
public class AccountTest
{
            [Test]
            public void TransferFunds()
            {
                        Account source = new Account();
                        source.Deposit(200m);

                        Account destination = new Account();
                        destination.Deposit(150m);

                        source.TransferFunds(destination, 100m);

                        Assert.AreEqual(250m, destination.Balance);
                        Assert.AreEqual(100m, source.Balance);
            }
}
```

Agenda

Unit Tests

Unit Tests in .NET

Code coverage

Unit tests vs. Integration tests

Test Driven Development

Code Coverage

```
a = CalculateA();
If(a || CalculateB())
{
            result = 1;
}
result = 0;
```

Coverages:

- Statement and Block
- Function and Function call
- Branch
- Condition/Decision

Agenda

- Unit Tests
- Unit Tests in .NET
- Code coverage
- **Unit tests vs. Integration tests**
- Test Driven Development

Types of tests

- Unit tests
- Integration tests
- System tests
- UAT
- V model
- Thin line between Unit Tests and IntegrationTests (points of failure)

Agenda

Unit Tests

Unit Tests in .NET

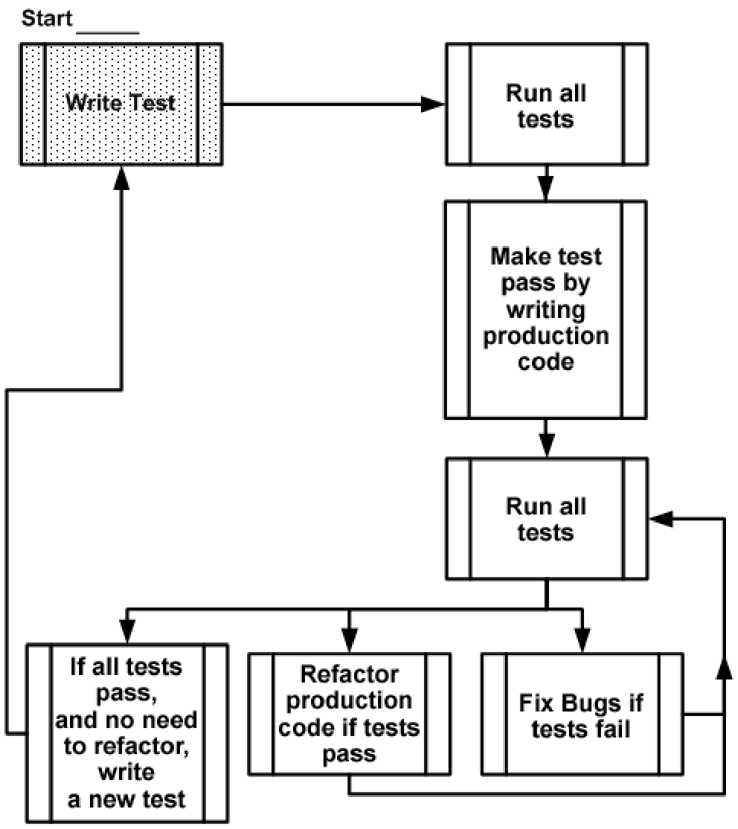
Code coverage

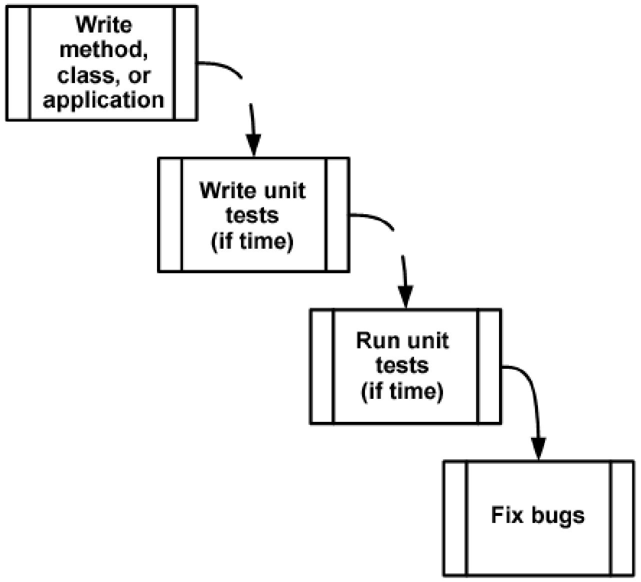
Unit tests vs. Integration tests

Test Driven Development

TDD

- Test Driven Development
- Red-Green-Refactor


Start
Write Test
Run all tests
Make test pass by writing production code
Run all tests
If all tests pass, and no need to refactor, write a new test
Refactor production code if tests pass
Fix Bugs if tests fail


Write method, class, or application
Write unit tests (if time)
Run unit tests (if time)
Fix bugs

Why to use TDD

- Gradient progress of software development
- Working on one thing at the time
- Less pain in debugging code
- Good enforcer to write tests and keep code coverage high
- TDD ≠ good tests

When to use TDD

ALWAYS ☺

- Always when functionality of unit is well defined
- Can be use in extending legacy system with new modules

If you prototyping it would be extra overhead caused by re-coding tests when using TDD

Naming convention

- Order in tests

References

- The Art of Unit Testing
- Nunit

Example Q & A

Q & A