ВВЕДЕНИЕ В UNIT ТЕСТИРОВАНИЕ В C#

Author: Анжелика Кравчук

ДАЙТЕ ЧЕЛОВЕКУ РЫБУ, И ВЫ НАКОРМИТЕ ЕГО НА ОДИН ДЕНЬ. НАУЧИТЕ ЕГО ЛОВИТЬ РЫБУ, И ОН НИКОГДА НЕ БУДЕТ ГОЛОДЕН

Если пишешь код – пиши тесты

Ученик спросил мастера-программиста:

"Когда я могу перестать писать тесты?"

Мастер ответил:

"Когда ты перестаешь писать код"

Ученик спросил:

"Когда я перестаю писать код?"

Мастер ответил:

"Когда ты становишься менеджером"

Ученик задрожал и спросил:

"Когда я становлюсь менеджером?"

Мастер ответил:

"Когда ты перестаешь писать тесты"

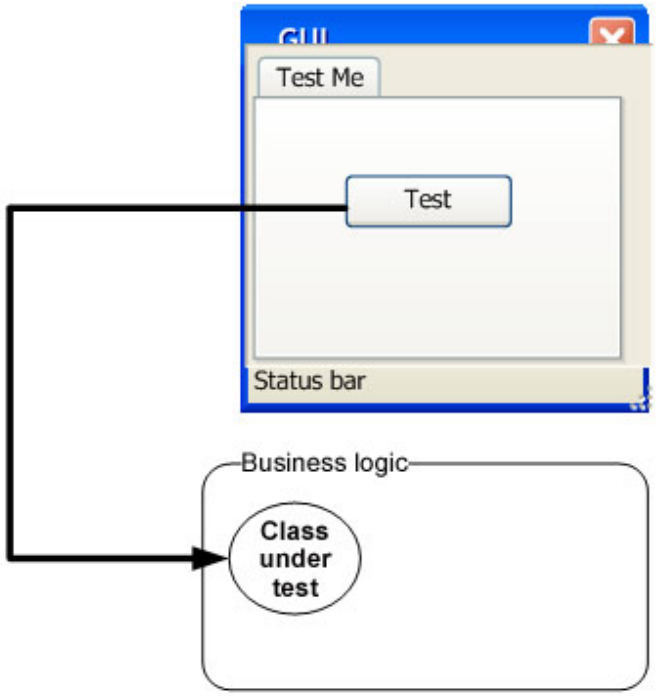
Ученик побежал писать тесты.

Остались только следы.

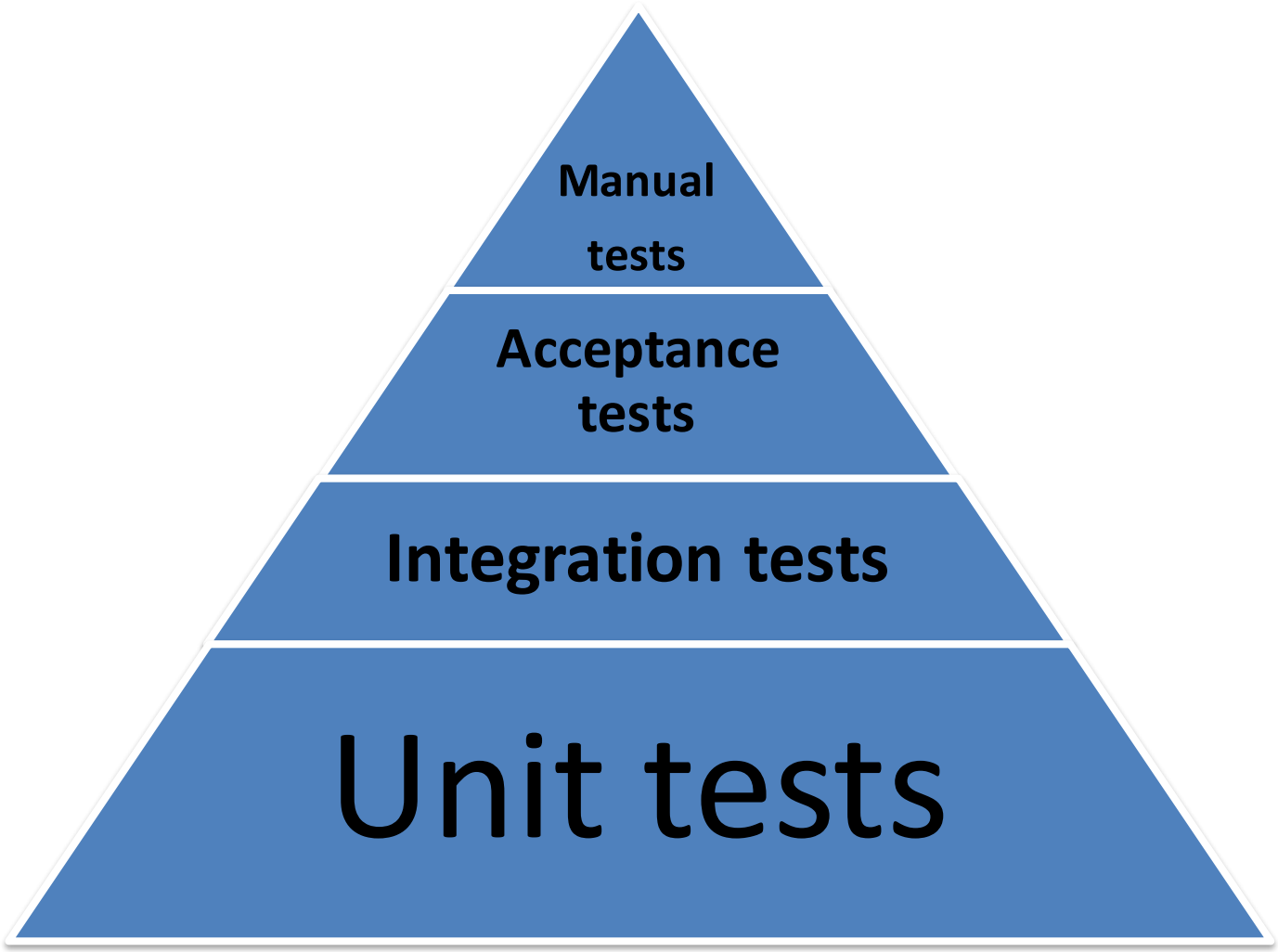
Если код заслуживает быть написанным, он заслуживает иметь тесты.

Что такое unit-тестирование

Важным этапом создания программного обеспечения является тестирование, в ходе которого выявляются ошибки, проверяется правильность функционирования программы и ее соответствие заявленным требованиям.



Что такое unit-тестирование



Что такое unit-тестирование

Модульное тестирование, или юнит-тестирование (англ. unit testing) – процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы.

Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

При модульном тестировании совместно с объектно-ориентированным проектированием, термин «модуль» соответствует понятию «класс», а термин «функция модуля» - понятию «метод класса»

Что такое unit-тестирование

Ученик спросил великого мастера программирования Летящего Пера:

"Что превращает тест в юнит-тест?"

Великий мастер программирования ответил:

"Если он обращается к базе, значит, он не юнит-тест.
Если он обращается к сети, значит, он не юнит-тест.
Если он обращается к файловой системе, значит, он не юнит-тест.
Если он не может выполняться одновременно с другими тестами, значит, он не юнит тест.
Если ты должен делать что-то с окружением, чтобы выполнить тест, значит, он не юнит тест".

Другой мастер-программист присоединился и начал возражать.
"*Извините, что я спросил*", — сказал ученик.
Позже ночью он получил записку от величайшего мастера-программиста.
Записка гласила:

"Ответ великого мастера Летящего Пера прекрасный ориентир.
Следуй ему, и в большинстве случаев не пожалеешь.
Но не стоит застревать на догме.
Пиши тест, который должен быть написан".

Ученик спал хорошо.
Мастера все еще продолжали спорить глубокой ночью.

Когда не нужно писать тесты

Разрабатывается простой сайт-визитка из 5 статических html-страниц и с одной формой отправки письма. На этом заказчик, скорее всего, успокоится, ничего большего ему не нужно. Здесь нет никакой особенной логики, быстрее просто все проверить «руками»

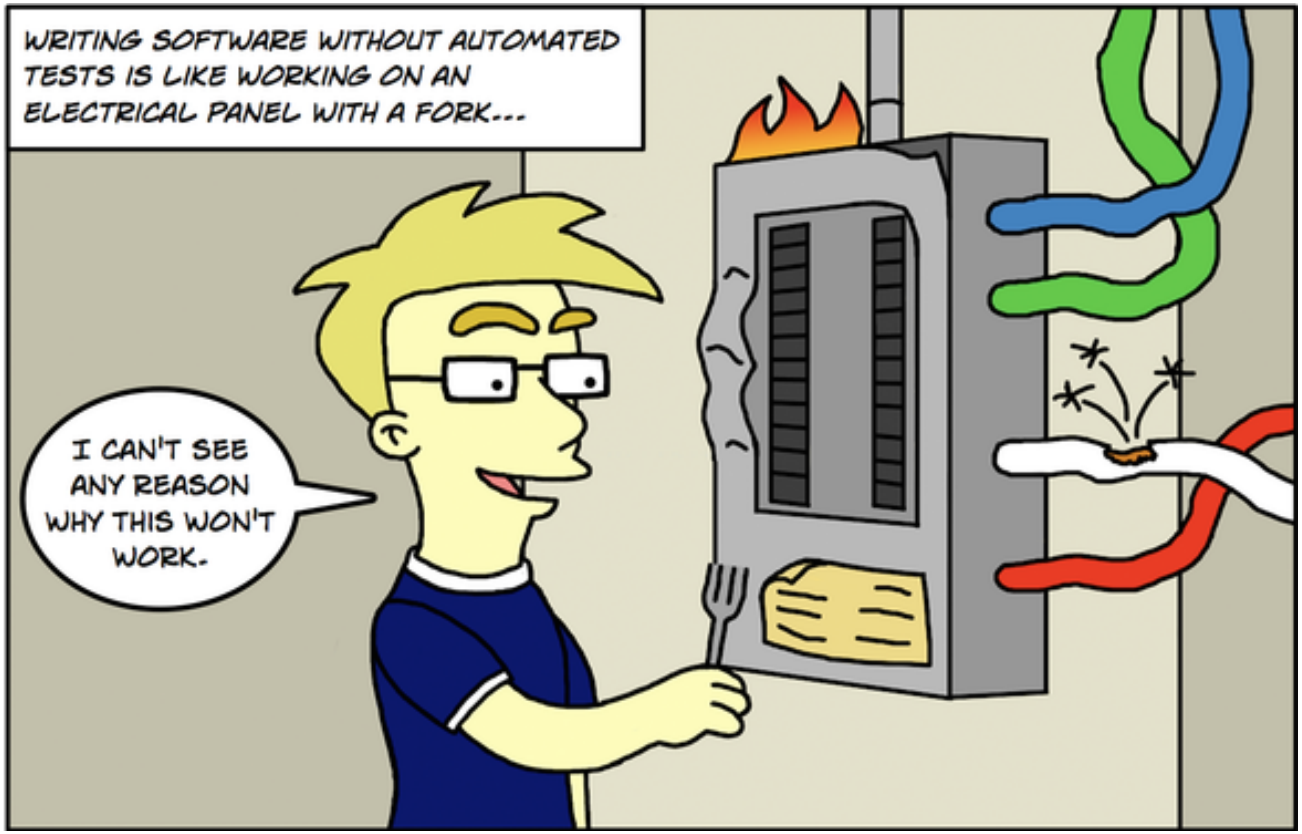
Разрабатывается рекламный сайт/простая флеш-игра или баннер – сложная верстка/анимация или большой объем статики. Никакой логики нет, только представление

Разрабатывается проект для выставки. Срок – от двух недель до месяца, система – комбинация железа и софта, в начале проекта не до конца известно, что именно должно получиться в конце. Софт будет работать 1-2 дня на выставке

Пишется идеальный код без ошибок, наделенный даром предвидения и способный изменить себя сам, вслед за требованиями клиента

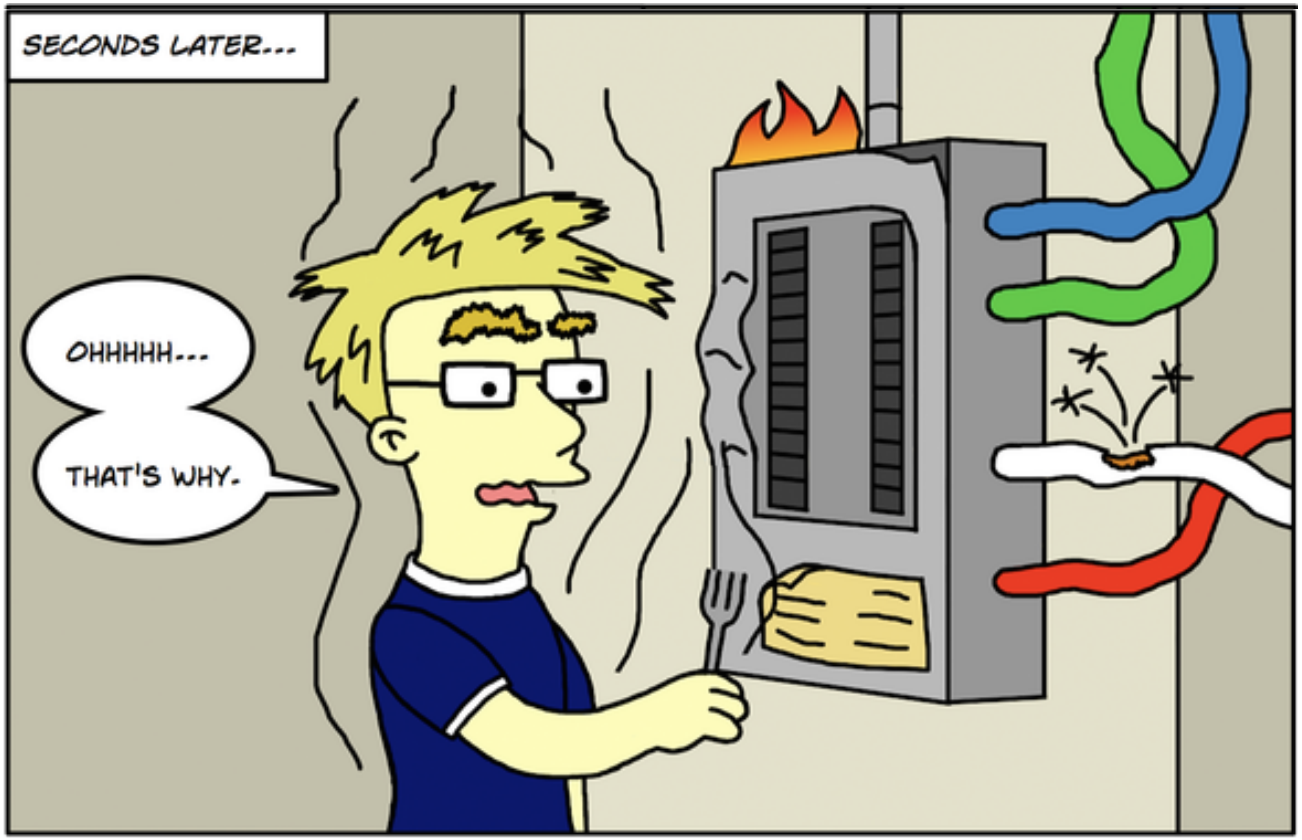
Когда нужно писать тесты

Любой долгосрочный проект без надлежащего покрытия тестами обречен рано или поздно …



Когда нужно писать тесты

...быть переписанным с нуля



Плюсы и минусы unit-тестирования

Плюсы

- **Обеспечивают мгновенную обратную связь**
- **Помогают документировать код и делать его понимание проще для других разработчиков Позволяют постоянно тестировать код, что сводит к минимуму появление новых ошибок**
- **Помогают уменьшить количество усилий, необходимых для повторного тестирования**
- **Поощряют написание слабосвязанного кода**

Минусы

- **Не проверяют взаимодействие объектов**
- **Не дают 100%-й гарантии**

Основные правила тестирования

Тесты должны

- **быть достоверными;**
- **не зависеть от окружения, на котором они выполняются;**
- **легко поддерживаться;**
- **легко читаться и быть простыми для понимания (новый разработчик должен понять что именно тестируется);**
- **соблюдать единую конвенцию именования;**
- **запускаться регулярно в автоматическом режиме.**

Логическое расположение тестов в системе контроля версий

Тесты должны быть частью контроля версий. В зависимости от типа решения, они могут быть организованы по-разному. Общая рекомендация: если приложение монолитное, следует положить все тесты в папку Tests; в случае множества разных компонентов, хранить тесты в папке каждого компонента

Соответствия между тестируемым и тестирующим кодами

Объект тестирования	Объект модульного теста
Проект	Создать проект (Class Library), содержащий тесты, с именем **`ProjectName.Tests`**
Класс	Для каждого класса, подлежащего тестированию, создать (по крайней мере) один тестирующий класс с именем **`ClassNameTests`**. Такие тестирующие классы называются наборами тестов (test fixtures)
Метод	Для каждого метода, подлежащего тестированию, создать (по крайней мере) один тестирующий public-метод (тест) с именем **`MethodName_Test Conditions_Expected Behavior`**

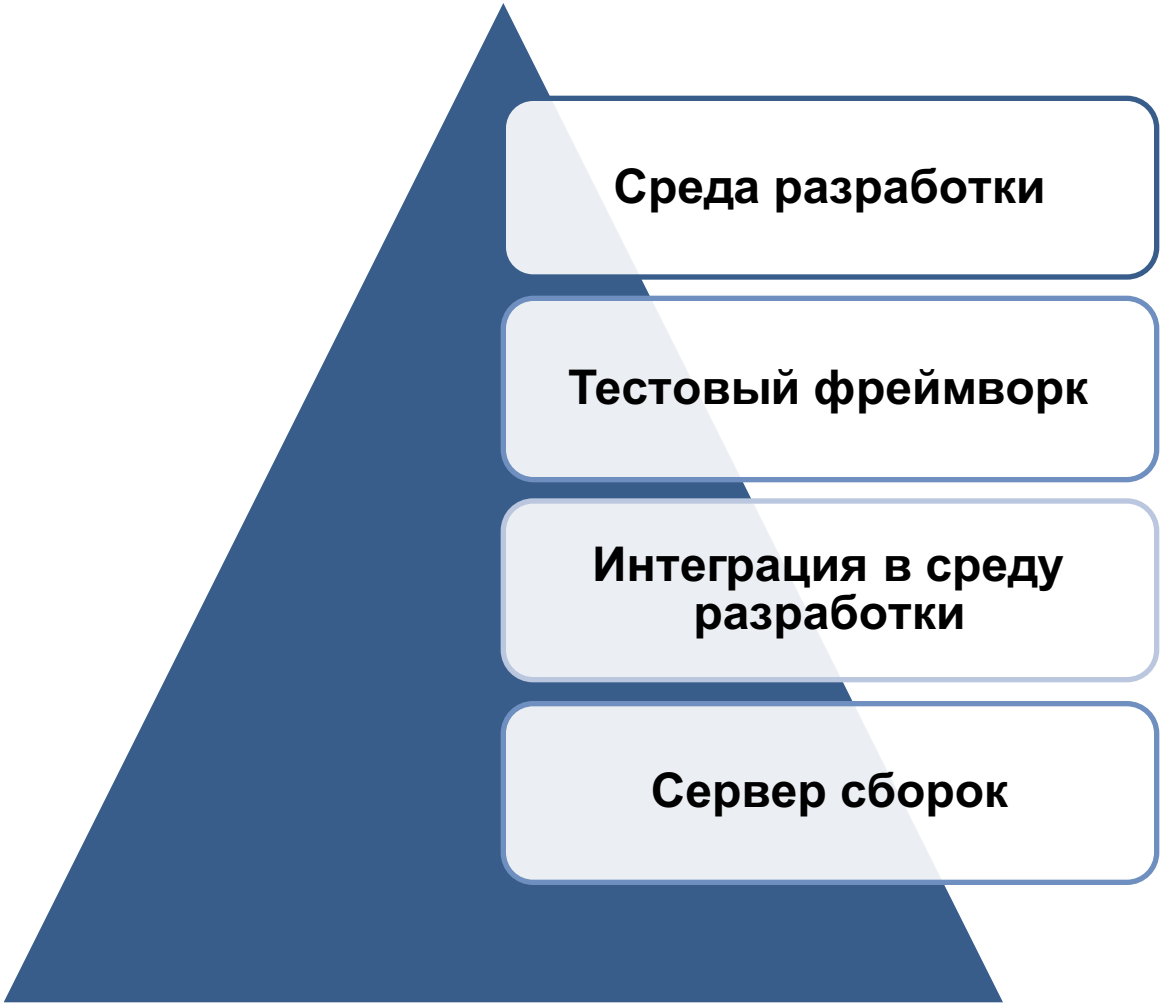
Соответствия между тестируемым и тестирующим кодами

<PROJECT_NAME>.Core <PROJECT_NAME>.Bl <PROJECT_NAME>.Web	<PROJECT_NAME>.Core.Tests <PROJECT_NAME>.Bl.Tests <PROJECT_NAME>.Web.Tests.
ProblemResolver	ProblemResolverTests

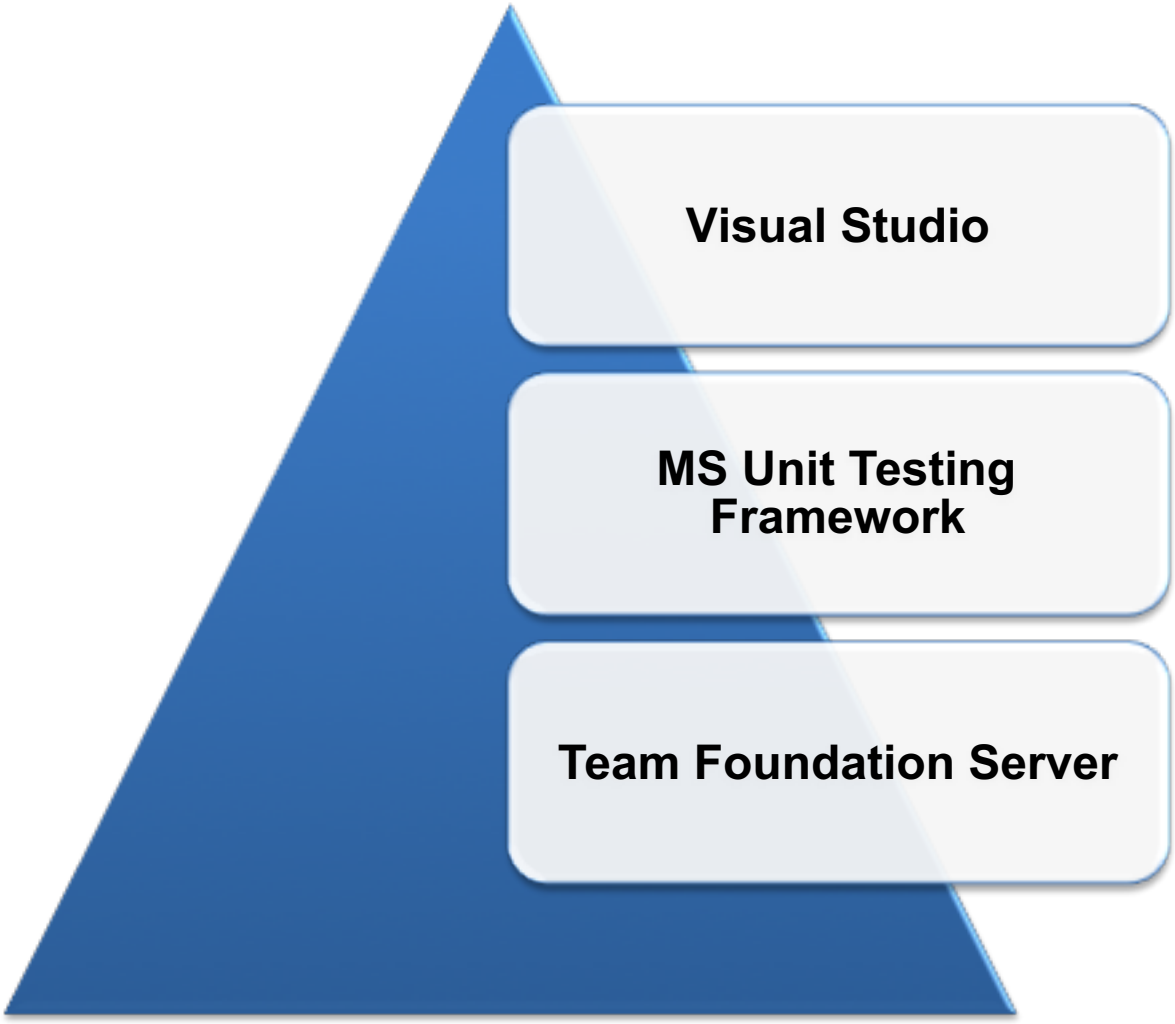
```
class Calculator
{
    public void Sum()
    {
        //TODO
    }
}
```

```
class CalculatorTests
{
    public void Sum_2Plus5_7Returned()
    {
        //TODO
    }
}
```

Инструменты тестирования



Инструменты тестирования. Microsoft



Сторонние инструменты тестирования

- **Среда: Visual Studio, Eclipse**
- **Фреймворк: NUnit, MbUnit, XUnit.NET, CsUnit (open source)**
- **Интеграция: TestDriven.Net (free/$), ReSharper (free/$)**
- **Сервер: CruiseControl (free), TeamCity (free/$)**

Фреймворки для unit-тестирования

Unit-testing Framework – базовый набор средств для написания тестов, предоставляющий следующие возможности

Библиотека

- **Разметка тестов**
- **Проверка различных условий**

Test Runner (специальное приложение)

- **Выполнение тестов**
- **Создание отчетов о выполненных тестах**

Примеры тестов с использованием MS Unit Testing Framework

- Библиотека Microsoft.VisualStudio.QualityTools.UnitTestFramework
- Разметка тестов с помощью атрибутов TestClass и TestMethod
- Проверка условий выполняется с помощью методов статического класса Assert

Что тестировать, а что – нет?

Не очень сложный код с зависимостями – связывает между собой разные компоненты, тесты важны, чтобы уточнить, как именно должно происходить взаимодействие

Сложный код с большим количеством зависимостей – вероятность сильной связанности, необходим рефакторинг

Простой код без зависимостей - можно не тестировать

Сложный код без зависимостей – некие алгоритмы или бизнес-логика, важные части системы – тестируем их

Практика написания модульных тестов. Шаблон Arrange-Act-Assert (AAA)

Шаблон для написания тестов – «Triple A» (Arrange-Act-Assert)

- **Arrange (Устанавить) – осуществить настройку входных данных для теста;**
- **Act (Выполнить) – выполнить действие, результаты которого тестируются;**
- **Assert (Проверить) – проверить результаты выполнения**

Практика написания модульных тестов. Шаблон Arrange-Act-Assert (AAA)

```
[TestClass]
public class ProgramTest
{
    [TestMethod]
    public void Sum_2Plus3_3Returned()
    {
        // Arrange
        var target = new ArithmeticUnit();
        target.OperandA = 2;
        target.OperandB = 3;
        //Act
        target.Add();
        //Assert
        Assert.AreEqual(5, target.Result);
     }
}
```

Тестовое покрытие

System.Int32	System.String
положительное число	null
Отрицательное число	Пустая строка -ß String.Empty или ""
ноль	Один или более пробелов
int.MaxValue или 2,147,483,647	Один или более символов табуляции
int.MinValue или -2,147,483,648	Новая строка или Environment.NewLine
	Допустимая строка
	неверная строка
	символы Unicode, например, китайский язык

Row tests или параметризированные тесты

NUnit testing framework

```
[TestCase(12,3,4)]
[TestCase(12,2,6)]
[TestCase(12,4,3)]
public void DivideTest(int n, int d, int q)
{
    Assert.AreEqual( q, n / d );
}

[TestCase(12,3, Result=4)]
[TestCase(12,2, Result=6)]
[TestCase(12,4, Result=3)]
public int DivideTest(int n, int d)
{
  return( n / d );
}
```

Row tests или параметризированные тесты

```
NUnit testing framework

[Test, TestCaseSource("DivideCases")]
public void DivideTest(int n, int d, int q)
{
    Assert.AreEqual( q, n / d );
}

static object[] DivideCases =
{
    new object[] { 12, 3, 4 },
    new object[] { 12, 2, 6 },
    new object[] { 12, 4, 3 }
};
```

Практика написания модульных тестов

Когда пишешь код, думай о тесте.
Когда пишешь тест, думай о коде.
Когда ты думаешь о коде и тесте как о едином,
тестирование просто, а код красив.

- **Unit-тесты автоматизированы**
- **Unit-тесты пишутся на том же языке, что и тестируемый код**
- **Unit-тесты – простые**
- **Unit-тесты – быстрые**
- **Unit-тесты – независимые**
- **Unit-тесты – надежные**
- **Unit-тесты – точные**