# Тестирование программного обеспечения

# Виды тестирования

# Терминология

**Ошибка (error, mistake) –** ошибка в рассуждениях программиста, его понимании свойств программы.

**Дефект, баг (defect)** - самое общее нарушение каких-либо требований или ожиданий, не обязательно проявляющееся вовне.

**Неисправность (failure)** - наблюдаемое нарушение требований, проявляющееся при каком-то реальном сценарии работы ПО

**Проблема (problem)** – важная для конечного пользователя неисправность.

# Составляющие дефекта

- **Заголовок (Summary)** – короткое отражение сути проблемы. Одно емкое и информативное предложение (обычно длиной 100-120 символов).
- **Описание (Description)** – детальное описание проблемы: что, где и как происходит.
- **Шаги воспроизведения (Steps to reproduce)** – пошаговая инструкция по доведению программы до состояния, когда дефект виден.
- **Ожидаемый результат (Expected result)** – ожидаемое правильное поведение программы в результате выполнения шагов воспр.
- **Наблюдаемый результат (Actual result)** – наблюдаемое поведение программы в результате выполнения шагов воспр.

*Разница между ожидаемым и наблюдаемым результатом и есть дефект*

- **Приложения (Attachments) –** артефакты, помогающие воспроизведению и пониманию дефекта: файлы логов, скриншоты экрана, тестовые утилиты и т.д.

# Составляющие дефекта

- **Заголовок:**

  *При выполнении операции извлечения квадратного корня из отрицательных чисел возникает ошибка «System crash»*

- **Описание:**

  *Дефект является регрессионным относительно сборки №837. Ошибка проявляется только в том случае, если вид калькулятора – обычный.*

- **Шаги воспроизведения:**

  *1. Открыть калькулятор;*

  *2. Нажать кнопку ‘1’;*

  *3. Нажать кнопку ‘+-’;*

  *4. Нажать кнопку ‘√’.*

- **Ожидаемый результат:** *Сообщение «Недопустимый ввод» на дисплее калькулятора.*

- **Наблюдаемый результат:** *Диалоговое окно с текстом «System crash».*

# Виды дефектов

Существует 2 основных признака классификации:

**Серьезность (Severity)** – степень влияния дефекта на продукт.

- фатальная (fatal)
- серьезная (serious)
- ошибка неудобства (inconvenient)
- косметическая (cosmetic)
- предложение по улучшению (suggestion for improvement, feature request)

**Приоритет (Priority)** – степень важности / срочности исправления дефекта.

- высокий (high)
- нормальный (medium)
- низкий (low)

# Тестовый набор, тестовый сценарий

- **Тестовый набор**
  - Набор тестов, реализующих бизнес-задачу, выполняемую тестируемой системой
  - Тестовый набор включает кроме тестовых сценариев еще и тестовые данные или правила их генерации

# Формирование тестовых наборов

**Покрытие операторов**. Этот критерий предполагает выбор такого тестового набора данных, который вызывает выполнение каждого оператора в программе хотя бы 1 раз (критерий очень слабый).

**Покрытие узлов ветвления (покрытие решений).** Предполагает разработку такого количества тестов, чтобы в каждом узле ветвления был обеспечен переход по веткам «истина» и «ложь» хотя бы 1 раз.

# Формирование тестовых наборов

**Покрытие условий**. Если узел ветвления содержит более 1 условия, тогда нужно разрабатывать число тестов, достаточное для того, чтобы возможные результаты каждого условия в решении выполнялись, по крайней мере 1 раз. (case … of)

**Комбинаторное покрытие условий.** Для отслеживания таких ошибок используют комбинаторное покрытие условий. Этот критерий требует создания такого числа тестов, чтобы все возможные комбинации результатов условия в каждом решении и все точки входа выполнялись по крайней мере 1 раз.

# Формирование тестовых наборов

**Метод эквивалентного разбиения.** Построение тестов этим методом осуществляется в 2 этапа:

- выделение классов эквивалентности;

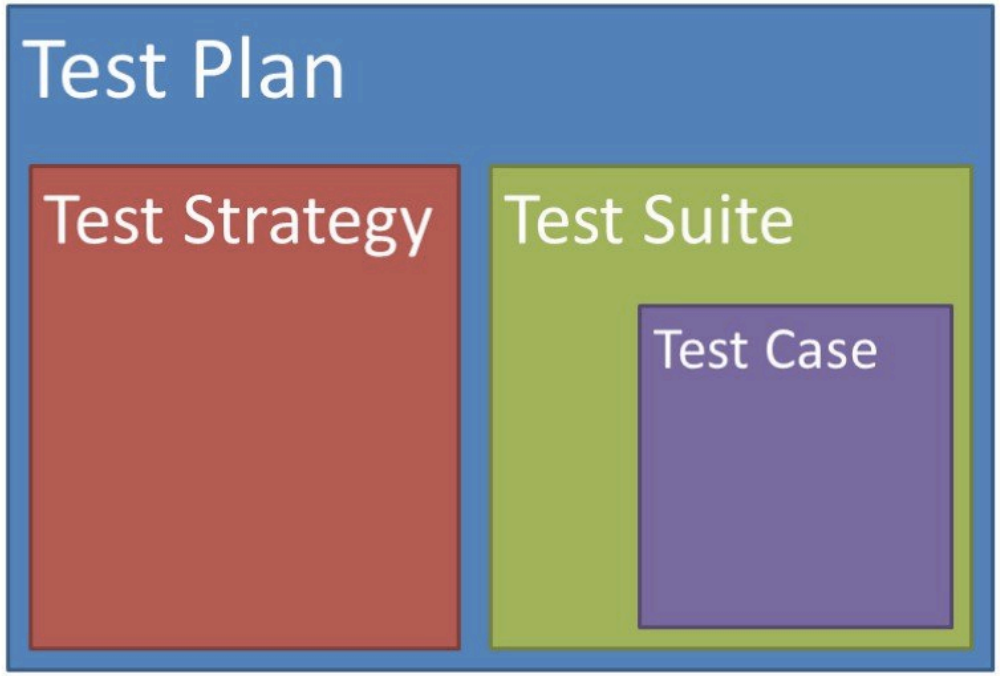
- построение тестов.

**Классом эквивалентности** называют множество входных значений, каждое из которых имеет одинаковую вероятность обнаружения конкретного типа ошибок.

Классы эквивалентности выделяются путем анализа входного условия и разбиением его на 2 или более группы. Для каждого условия существует правильный и неправильный класс эквивалентности.

# Документация

Тестовая документация



# Unit  тестирование в C#

Unit – тестирование
Валидация и верификация
Когда нужно писать Unit – тесты
Шаблон AAA - ?
Assertion
Техники TDD (Test Driven Development

# Unit  тестирование в C#

Unit тест – блок кода (обычно метод), который вызывает тестируемый блок кода и проверяет правильность его работы. Если результат юнит-теста не совпадает с ожидаемым результатом, тест считается не пройденным

# Unit тестирование в C#

## Типы тестирования

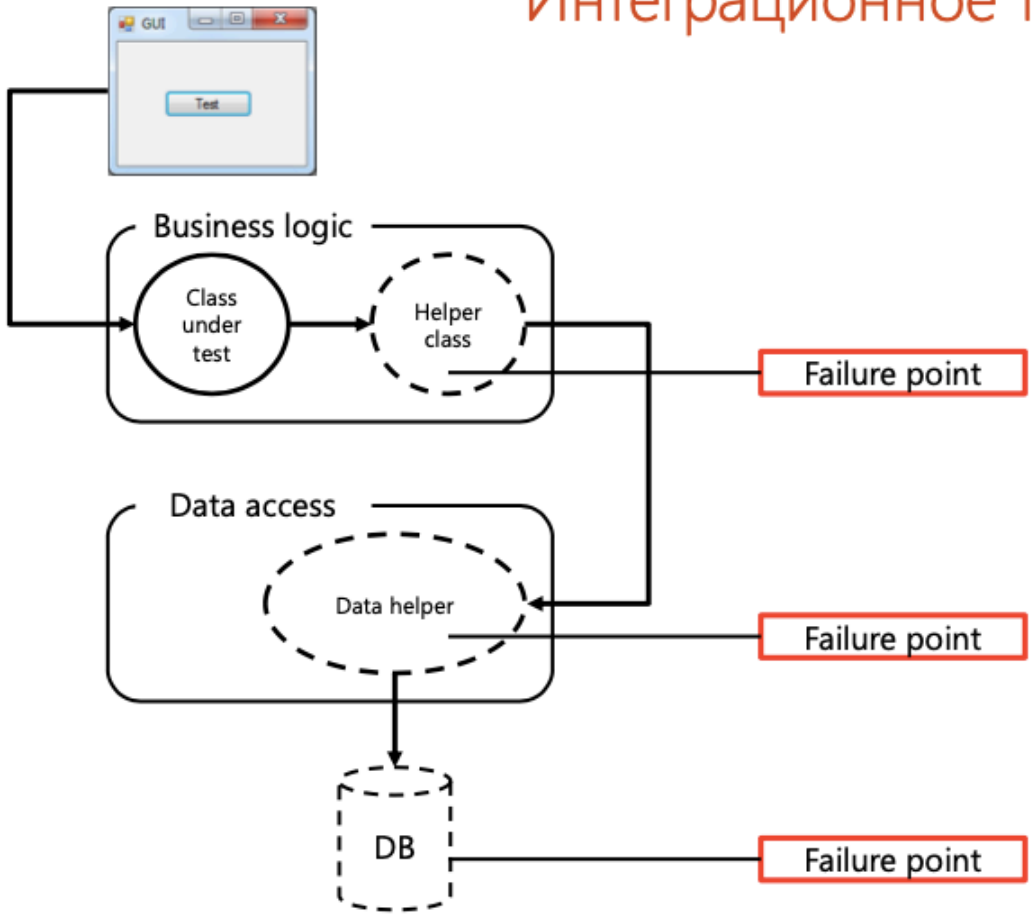
**Модульное тестирование (Unit testing)** – тестирование каждой атомарной функции приложения отдельно, с использованием объектов искусственно смоделированной среды.

**Интеграционное тестирование** – вид тестирования, при котором на соответствие требований проверяется интеграция модулей, их взаимодействие между собой, а также интеграция подсистем в одну общую систему.

**Системное тестирование** – это тестирование программного обеспечения выполняемое на полной, интегрированной системе, с целью проверки соответствия системы исходным требованиям, как функциональным, так и не функциональным.

# Unit тестирование в C#

## Интеграционное тестирование




При интеграционном тестировании существует много критических точек, в которых приложение может дать сбой, что делает поиск ошибок сложнее.

# Unit тестирование в C#

## Верификация и валидация

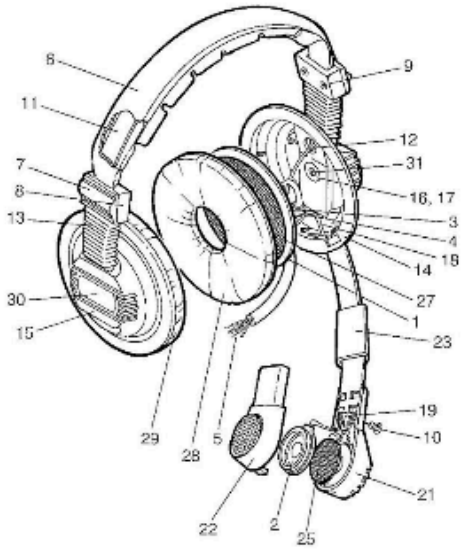
**Верификация (verification)** - это процесс оценки системы или её компонентов с целью определения того, удовлетворяют ли результаты текущего этапа разработки условиям, сформированным в начале этого этапа.


**Валидация (validation)** – это определение соответствия разрабатываемого ПО ожиданиям и потребностям пользователя, требованиям к системе.

| Verification | Validation |
|---|---|
| Делаем ли мы продукт ***правильно*** | Делаем ли мы ***правильный*** продукт |
| Реализована ли ***вся*** функциональность | ***Правильно ли*** реализована функциональность |
| Производиться ***разработчиками*** | Производиться ***тестировщиками*** |
| Инспектирование кода, сравнение требований | Выполнение программы |
| Объективная оценка реализованных функций | Субъективная оценка приложения |

# Unit тестирование в C#

## Валидация и верификация




**Верификация**

**Валидация**

# Unit тестирование в C#

## Свойства хорошего Unit теста

**Unit тест должен быть**:

- Автоматизированным и повторяемым;
- Простым в реализации;
- После написания он должен остаться для последующего использования;
- Кто угодно в команде должен иметь возможность запустить Unit тест;
- Должен запускаться одним нажатием кнопки;
- Должен выполнятся быстро.

# Unit тестирование в C#

## Расположение тестов

- Тесты должны быть частью контроля версий
- Если приложение монолитное – все тесты размещаются в директории tests
- Если приложение состоит из компонентов – тесты следует размещать в директории с тестируемым компонентом
- Выносите тесты в отдельный проект.

# Unit тестирование в C#

## Именование проектов

Добавляйте к каждому проекту его собственный тестовый проект.

Проекты:
<PROJECT_NAME>.Core
<PROJECT_NAME>.BI
<PROJECT_NAME>.Web

Проекты с тестами:
<PROJECT_NAME>.Core.Tests
<PROJECT_NAME>.BI.Tests
<PROJECT_NAME>.Web.Tests

# Unit тестирование в C#

## Именование методов и классов

***Именование классов:***

Класс UserManager тестирующий класс для него - UserManager**Tests.**
Каждый тестирующий класс должен тестировать только одну сущность.

***Именование unit тестов (методов):***

Принцип именования [Тестирующийся метод]_[Сценарий]_[Ожидаемое поведение]
Примеры:
Sum_10plus20_30returned
GetPasswordStrength_AllCahrs_5Points

# Unit тестирование в C#

## Какой код тестировать

Когда не нужно создавать юнит тесты:

**1) Простой код без зависимостей**
**2) Сложный код с большим количеством зависимостей -** скорее всего, для такого кода следует провести рефакторинг. Нет смысла писать тесты для методов, сигнатуры которых будут меняться. Для такого кода лучше создавать *приемочные тесты*.

Когда нужно создавать юнит тесты:

**3) Сложный код без зависимостей** – «запутанная» бизнес логика или сложные алгоритмы.
**4) Не очень сложный код с зависимостями** – код связывающий между собой разные компоненты.

# Unit тестирование в C#

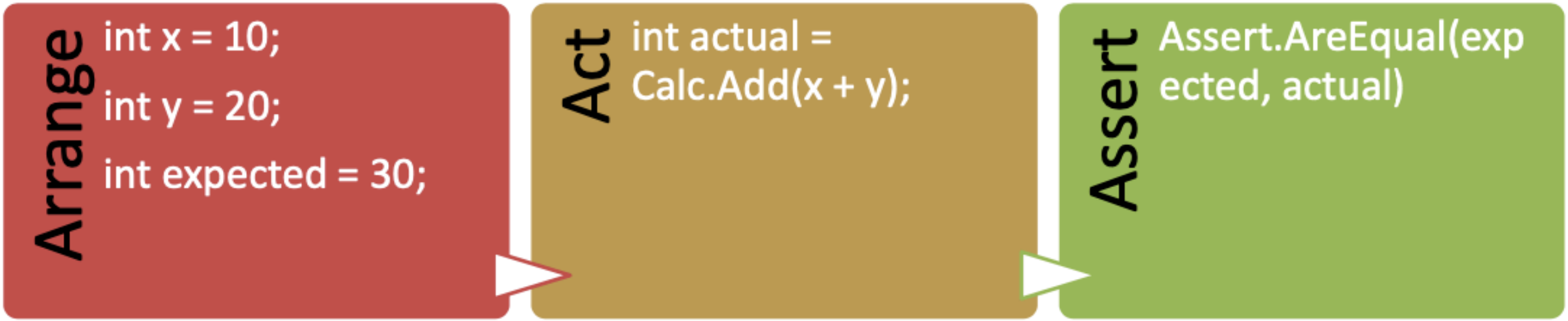
## Unit Test Frameworks



NUnit www.nunit.org
MS Test http://bit.ly/10TLj4L
xUnit.Net https://github.com/xunit/xunit

# Unit тестирование в C#

## Подход AAA



# Unit тестирование в C#

## Атрибуты

**TestClass** – Тестирующий класс
**TestMethod** – Тестирующий метод

**TestInitialize** – Метод для инициализации. Вызывается перед каждым тестирующим методом.
**TestCleanup** – Метод для освобождения ресурсов. Вызывается после каждого тестирующего метода

**ClassInitiazlie** – Вызывается один раз для тестирующего класса, перед запуском тестирующего метода.
**ClassCleanup** – Вызывается один раз для тестирующего класса, после завершения работы тестирующих методов

**AssemblyInitialize** – вызывается перед тем как начнут работать тестирующие методы в сборке
**AssemblyCleanup** – вызывается после завершения работы тестирующих методов в сборке.

# Unit тестирование в C#

## Assertion

### Assert

- Сравнение двух входящих значений
- Много перегрузок для сравнения значений

### CollectionAssert

- Сравнение двух коллекций
- Проверка элементов в коллекции

### StringAssert

- Сравнение строк

# Unit тестирование в C#

## Основные методы класса Assert

**Assert.AreEqual()**
Проверка двух аргументов на равенство

**Assert.AreSame()**
проверяет, ссылаются ли переменные на одну и ту же область памяти.

**Assert.InstanceOfType()**
Метод для проверки типов объектов.

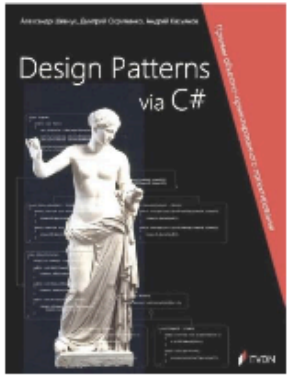
**Assert.IsTrue()**
проверка истинности логической конструкции.

**Assert.IsFalse()**
проверка лжи логической конструкции.

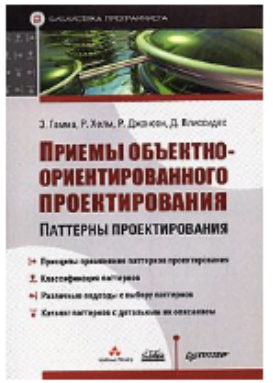
# Unit тестирование в C#

## Польза юнит тестирования

1. Инструмент борьбы с регрессией в коде.
2. Инвестиция в качественную архитектуру.
   - Изоляция зависимостей
   - Разработчик, понимающий, что его код будет использоваться в том числе и в модульных тестах, вынужден разрабатывать, пользуясь всеми преимуществами абстракций, и рефакторить при первых признаках появления высокой связанности.

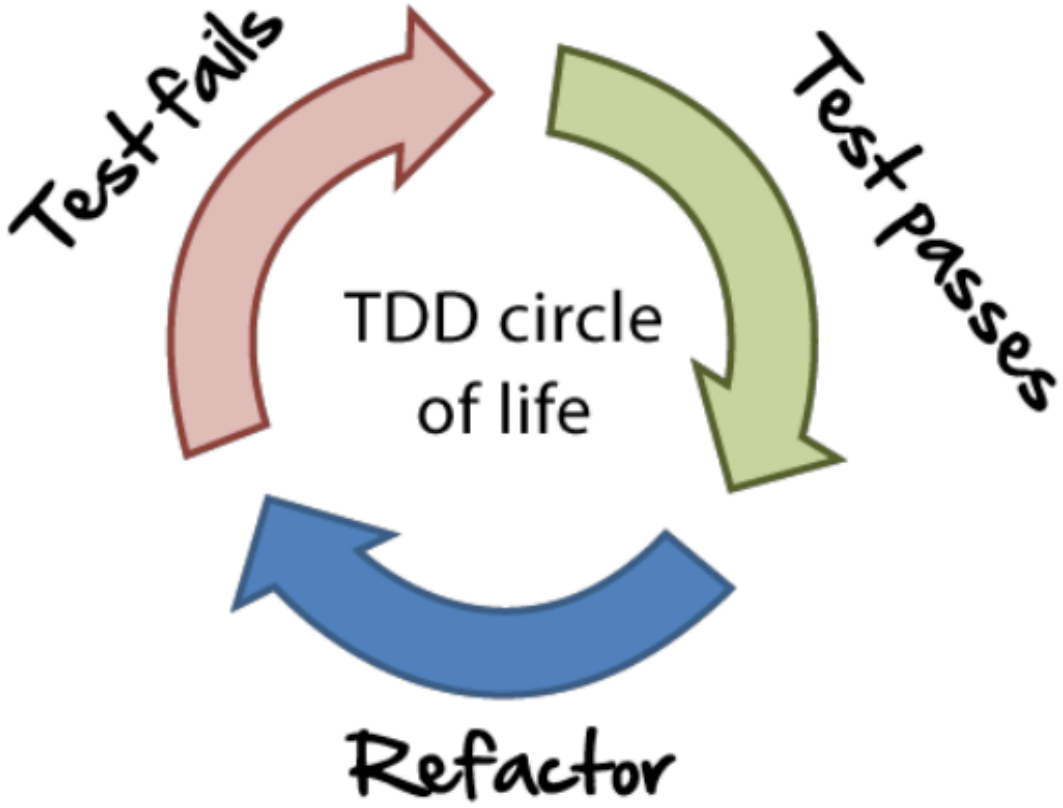







# Unit тестирование в C#

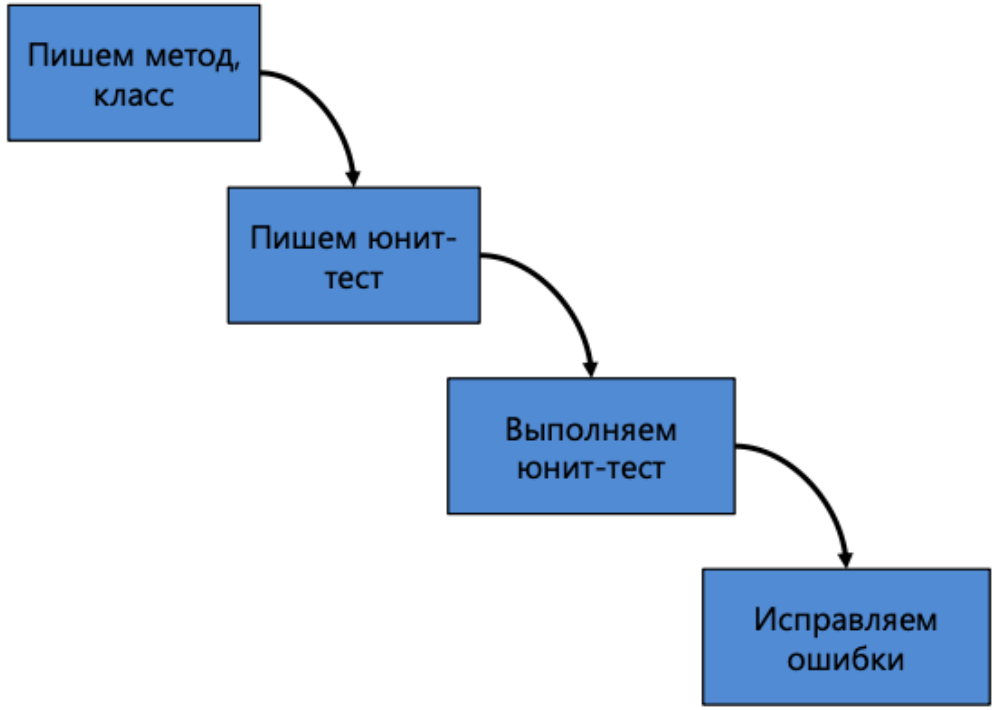
## Test Driven Development




**Test-Driven Development** (TDD) – разработка через тестирование. Подход разработки ПО, который заключается в написании юнит-теста перед написанием самого кода.

# Unit тестирование в C#

## Традиционный способ написания Unit тестов



# Unit тестирование в C#