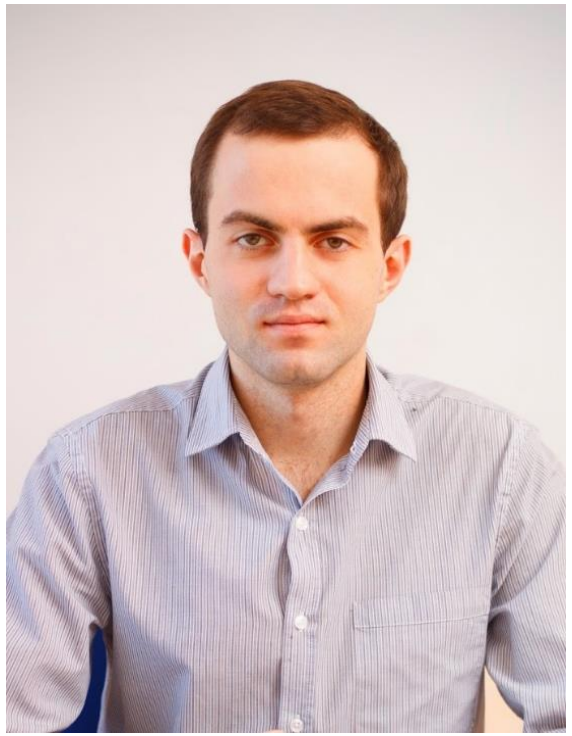
Unit тестирование в C#

Unit тестирование. Использование MSTest

Unit тестирование в C#

Introduction



Охрименко Дмитрий
MCT

MCID: 9210561

План курса Unit тестирование в C#

1 Урок – Unit тестирование. Использование MSTest
2 Урок – Использование stub объектов
3 Урок – Использование mock объектов. Работа с moq

Тема урока

Unit тестирование. Использование MSTest

Unit тестирование в C#

Содержание урока

Что такое Unit тестирование, типы тестирования
Верификация и валидация
Когда нужно писать unit тесты
Практики использования юнит тестов
Шаблон AAA
Assertion
Что такое TDD

Что такое UnitTest

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

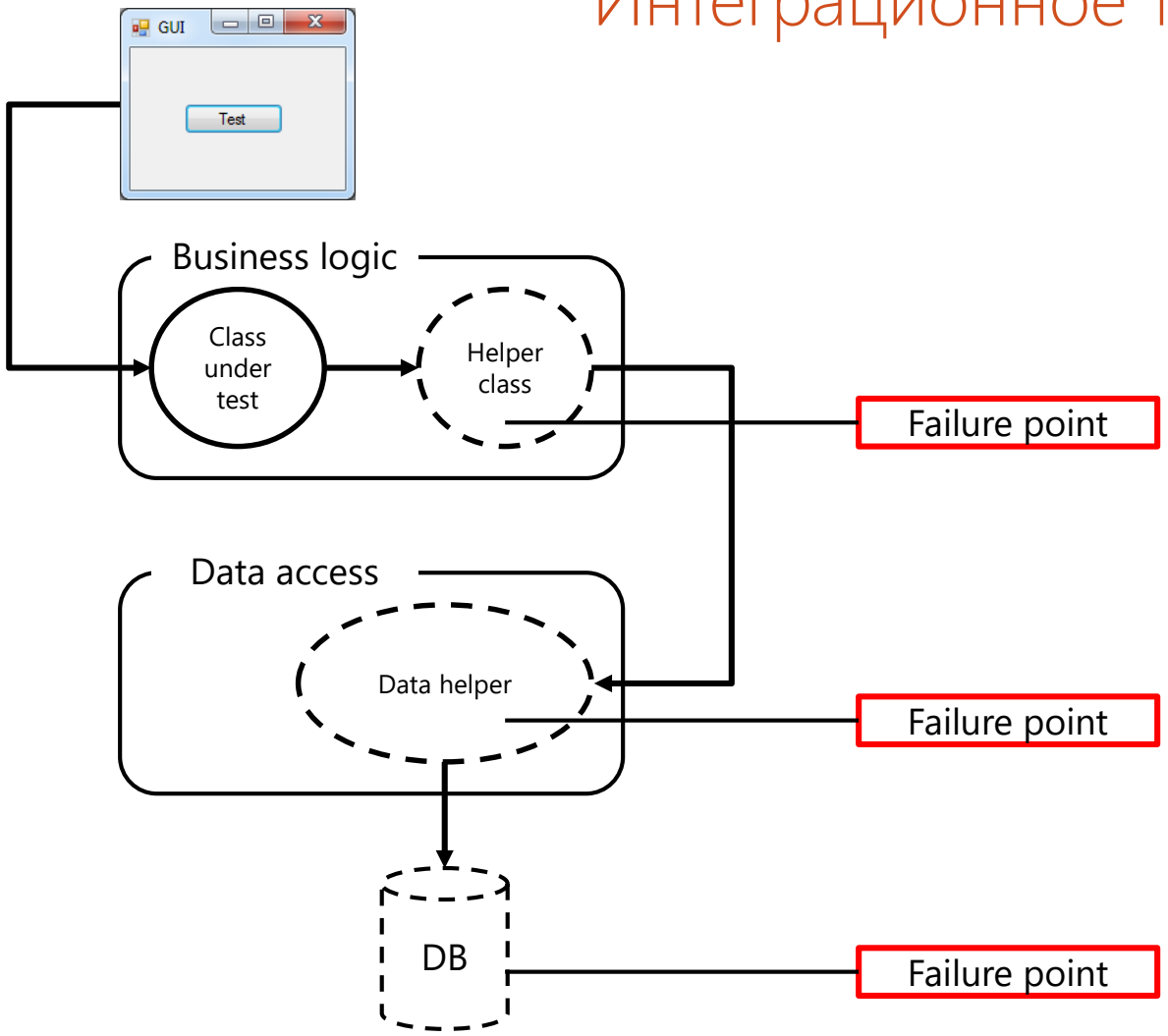
Типы тестирования

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.

Интеграционное тестирование



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

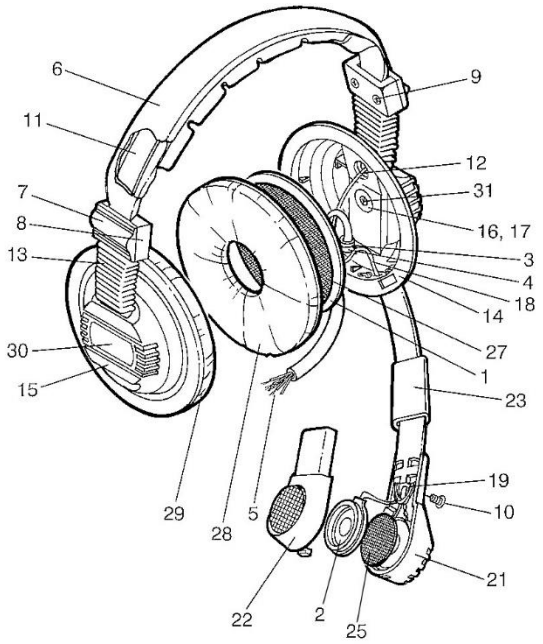
Верификация и валидация

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

Валидация и верификация



Верификация

Валидация

Best Practices

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.

Расположение тестов

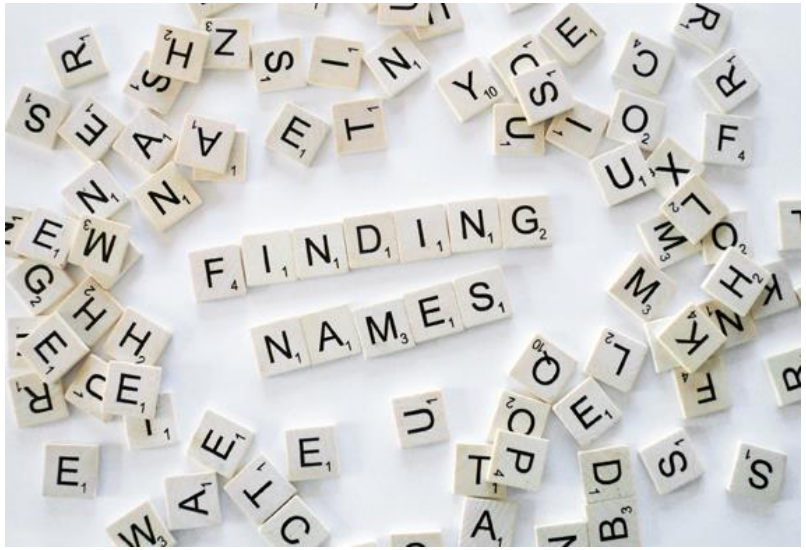
- Тесты должны быть частью контроля версий
- Если приложение монолитное – все тесты размещаются в директории tests
- Если приложение состоит из компонентов – тесты следует размещать в директории с тестируемым компонентом
- Выносите тесты в отдельный проект.

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



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

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:

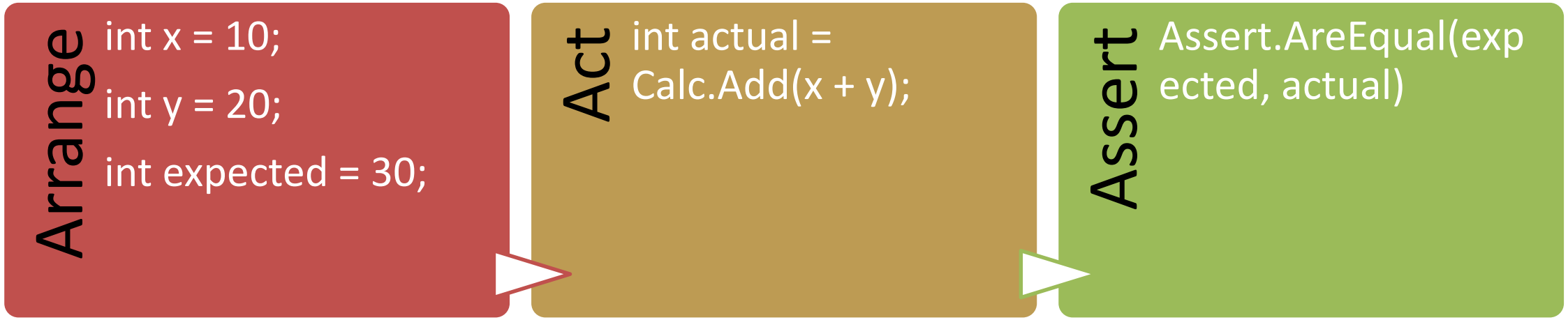
3) Сложный код без зависимостей – «запутанная» бизнес логика или сложные алгоритмы.
4) Не очень сложный код с зависимостями – код связывающий между собой разные компоненты.

Unit тестирование в C#

Unit Test Frameworks

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

Подход AAA


Arrange
int x = 10;
int y = 20;
int expected = 30;
Act
int actual =
Calc.Add(x + y);
Assert
Assert.AreEqual(exp
ected, actual)

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.

Assertion

Assert

- Сравнение двух входящих значений
- Много перегрузок для сравнения значений

CollectionAssert

- Сравнение двух коллекций
- Проверка элементов в коллекции

StringAssert

- Сравнение строк

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 ложи логической конструкции.

Как заставить себя писать юнит тесты

Я не пишу юнит тесты потому что...

Написание юнит тестов занимает слишком много времени
Мы привыкли что тестирование – это то, что делают в самом конце. Создавайте тесты по мере написания кода.

Как проверить что код работает? Запустить его вручную? Почему бы не потратить время для того чтобы написать юнит тест для проверки кода?

Запуск юнит тестов занимает слишком много времени
Должно быть несколько уровней тестирования. Юнит тесты должны запускаться быстро и часто. Интеграционные тесты с зависимостями (которые работают медленно), должны запускаться реже но все равно регулярно.

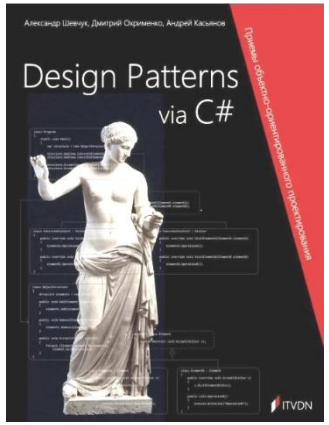
Я не пишу юнит тесты потому что...

Это не моя работа – тестировать код
Наша задача разрабатывать работающее ПО, а значит, должно быть основание заявлять, что код работает

Я не могу протестировать код, потому, что я точно не знаю, как он должен работать
Если вы точно не знаете, как должен работать код, как вы вообще могли начать его писать?

Польза юнит тестирования

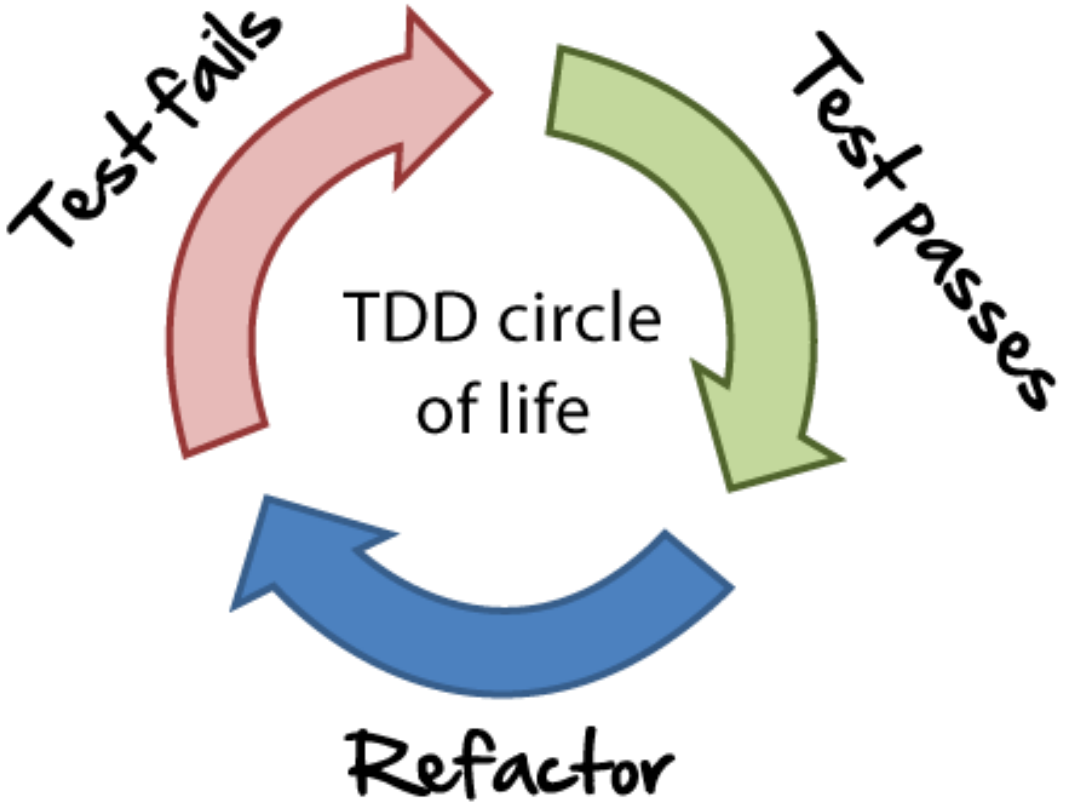
1. Инструмент борьбы с регрессией в коде.
2. Инвестиция в качественную архитектуру.
   - Изоляция зависимостей
   - Разработчик, понимающий, что его код будет использоваться в том числе и в модульных тестах, вынужден разрабатывать, пользуясь всеми преимуществами абстракций, и рефакторить при первых признаках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



http://itvdn.com/ru/patterns/DownloadBook

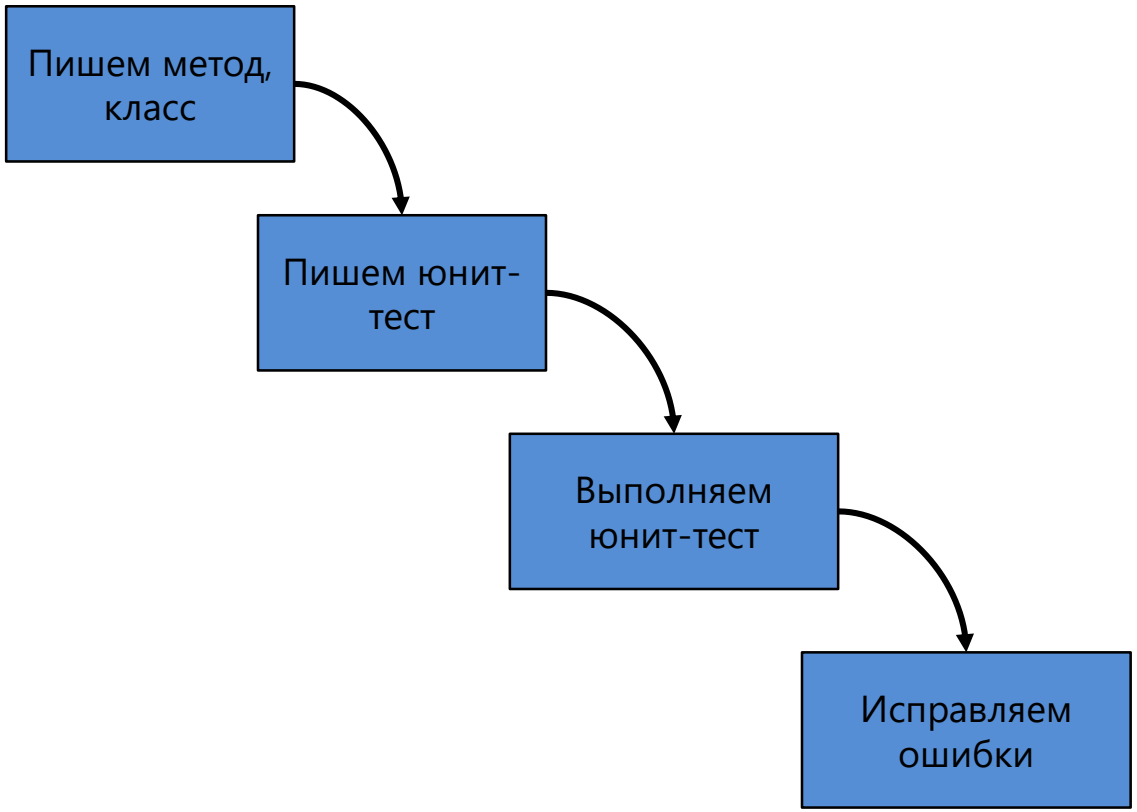
Test Driven Development

Test Driven Development

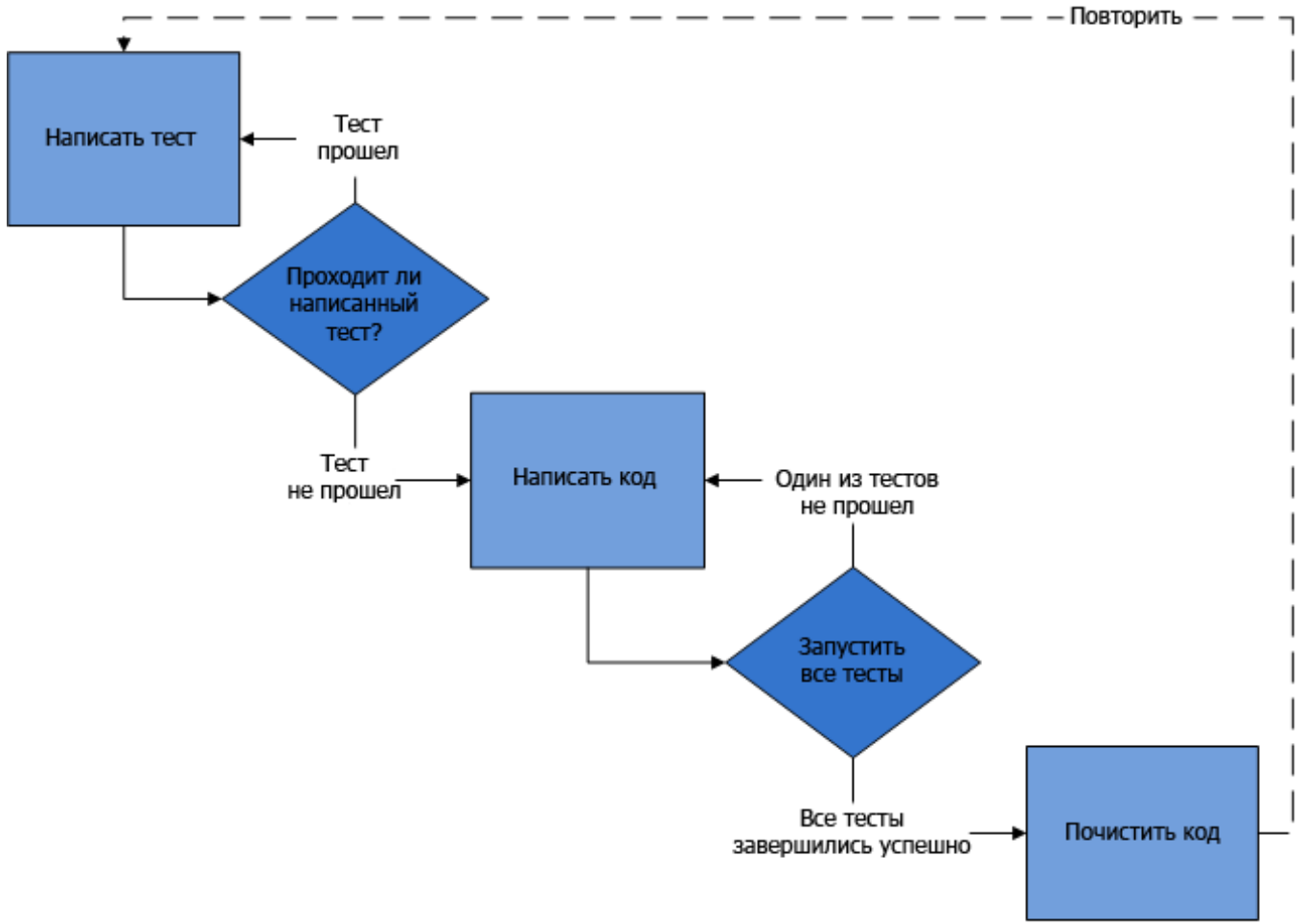


Test-Driven Development (TDD) – разработка через тестирование. Подход разработки ПО, который заключается в написании юнит-теста перед написанием самого кода.

Традиционный способ написания Unit тестов

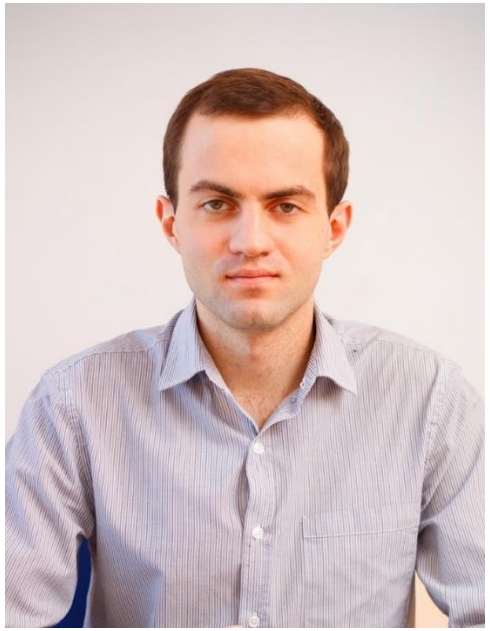


Test Driven Development



Unit тестирование в C#

Спасибо за внимание! До новых встреч!



Охрименко Дмитрий
MCT

MCID: 9210561

Информационный видеосервис для разработчиков программного обеспечения

ITVDN

IT VIDEO DEVELOPERS NETWORK

TestProvider PROMETRIC TEST CENTER CyberBionic systematics Microsoft Partner Silver Learning Windows Azure Cloud Partner EBA European Business Association