SimbirSoft

Курс C#

Лекция 7

Николай Артамонов
Разработчик .NET

31.05.2018

Последняя лекция будет ЗАВТРА (1.06.2018)

Немного о себе

Артамонов Николай

Окончил УлГТУ по специальности ИСТд

Опыт работы на коммерческих проектах более 4 лет.

Принимал активное участие более чем в 8 проектах.

Что будем рассматривать

- Тестирование – что это и зачем?
- Модульные (Unit) тесты
- TDD (разработка через тесты)
- CI(Непрерывная интеграция)
- DI(Внедрение зависимости)
- Практика. Библиотеки для юнит тестов.

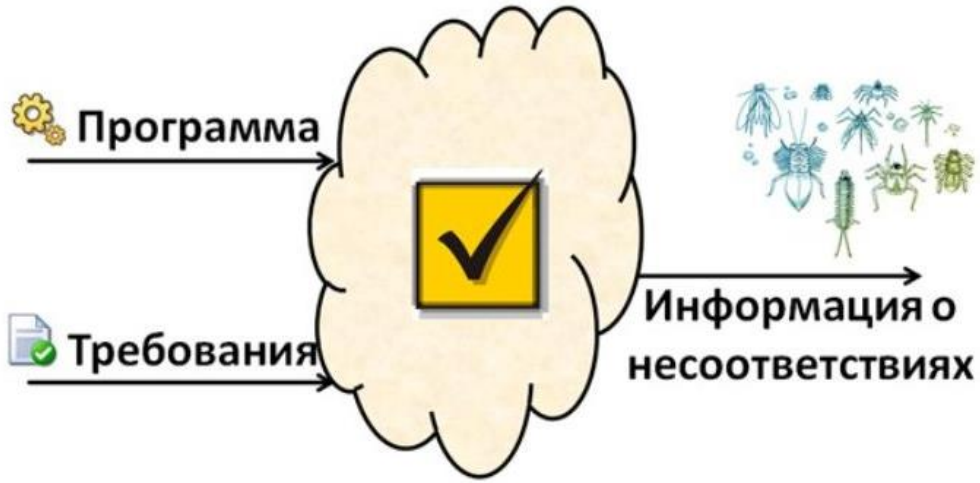
Тестирование – что это и зачем?

Простыми словами – это то, чем мы занимаемся ежедневно. Этот процесс стал просто неотъемлемой частью нашей жизни, что порой мы даже не замечаем его и он проходит на автомате.
Мы просто каждый раз проверяем что результат значимых для нас действий совпадает с тем, что мы ожидали от этого действия и данный результат не приведет к каким-либо нежелательным последствиям.

Тестирование – что это и зачем?

Тестирование ПО — процесс исследования, испытания программного продукта, имеющий две различные цели:

- продемонстрировать разработчикам и заказчикам, что программа соответствует поставленым требованиям;
- выявить ситуации, в которых поведение программы является неправильным, нежелательным или не соответствующим спецификации.

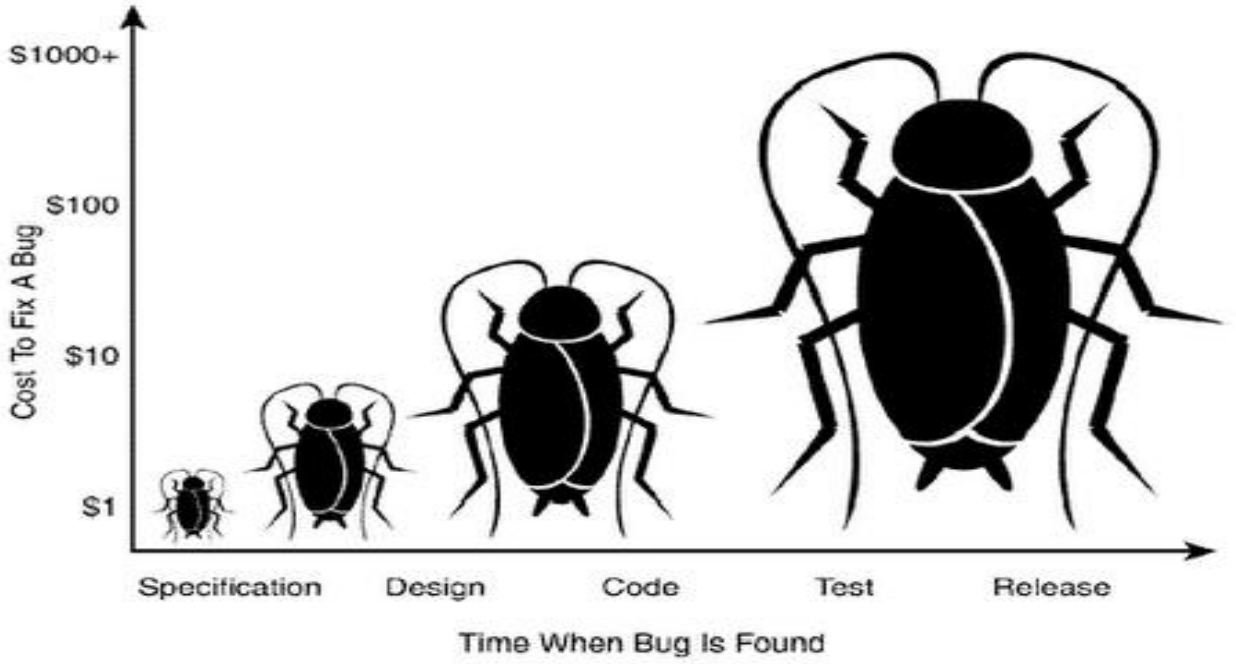


Тестирование – что это и зачем?

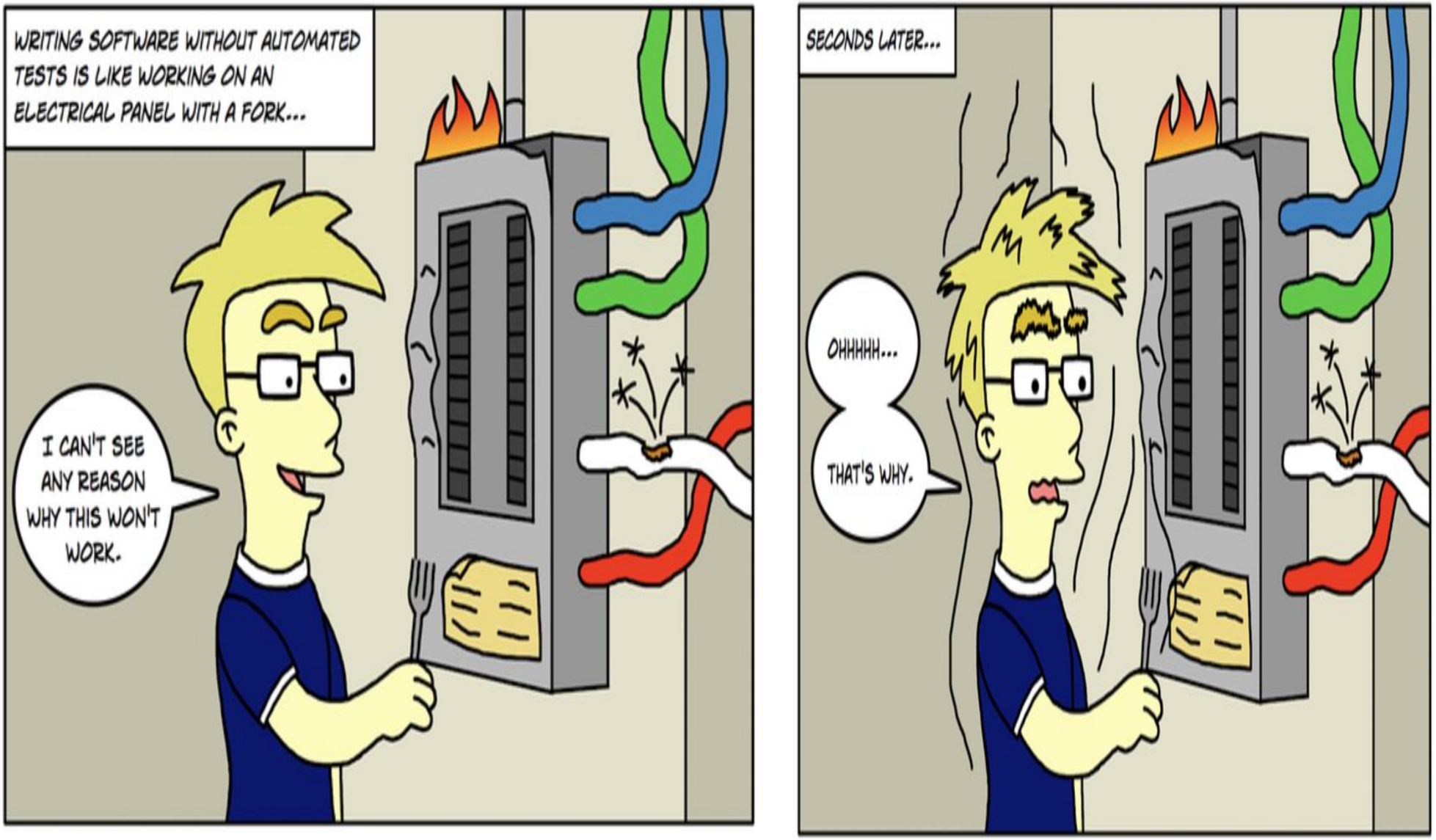
Основная задача тестирования **–** не “найти как можно больше ошибок”, а **пропустить как можно меньше критических неисправностей.**

Тестирование – что это и зачем?

Настоящий проект имеет свойство разрастаться. Серьезные баги начинают эволюционировать, и это приводит к тому, что с каждым днём такой баг исправить всё сложнее и дороже.



Тестирование – что это и зачем?



Тестирование – что это и зачем?

Тесты НЕ нужно писать, если

● Вы занимаетесь рекламным сайтом/простыми флеш-анимациями или баннерами – несложная верстка или большой объем статики. Никакой логики нет, только представление

● Вы делаете проект для выставки. Срок – от двух недель до месяца, ваша 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

● Вы всегда пишете код без ошибок, обладаете идеальной памятью и даром предвидения. Ваш код настолько крут, что изменяет себя сам, вслед за требованиями клиента. Иногда код объясняет клиенту, что его требования — полный бред и их не нужно реализовывать.

Тестирование – что это и зачем?

Последний случай рассмотрим отдельно. Я знаю только одного такого человека, и если вы не узнали себя на фото ниже, то у меня для вас плохие новости.

Тестирование – что это и зачем?

Проекты без покрытия тестами

- Массовое зарастание «спагетти-кодом»
- Практически никто не знает как именно все это работает
- И как правило, что оно должно в конечном итоге делать – тоже
- Постоянно всё ломается, в т.ч. то, что «ну вот совсем не трогали»

Тестирование – что это и зачем?

Проекты с серьезным покрытием тестами. Все тесты проходят

- Если тесты в проекте действительно запускаются, то их много.

- Каждый из них – атомарный: один тест проверяет только одну вещь.

- Тест является спецификацией метода класса, контрактом: какие входные параметры ожидает этот метод, и что остальные компоненты системы ждут от него на выходе.

- Текста немного: обычно пара страниц, с описанием основных фич, схем серверов и getting started guide'ом.

- Система надежно протестирована и сама рассказывает о себе путем тестов.

Тестирование – что это и зачем?

Проекты с тестами, которые никто не запускает и не поддерживает.

● Тесты в системе есть, но что они тестируют, и какой от них ожидается результат, неизвестно

● Присутствует какая-никакая архитектура, можно отыскать очень много документации на самые основные процессы.

● Скорее всего, над проектом еще трудится главный разработчик системы, который держит в голове особенности и хитросплетения кода.

Почему так?

● Бездумное написание тестов не только не помогает, но вредит проекту.

● Если раньше у вас был один некачественный продукт, то написав тесты, не разобравшись в этой теме, вы получите два. И удвоенное время на сопровождение и поддержку.

● Необходимо найти золотую середину между ситуацией, когда тесты отсутствуют и моментом, когда 20% времени уходит на написание нового функционала и 80% - на то чтобы поправить тесты.

Тестирование – что это и зачем?

Проекты с тестами, которые никто не запускает и не поддерживает.

Для того чтобы темная сторона кода не взяла верх, нужно придерживаться следующих основных правил.

В идеальном случае ваши тесты должны:

● Быть достоверными (+ проверять реакцию не только на положительные, но и отрицательные ситуации)

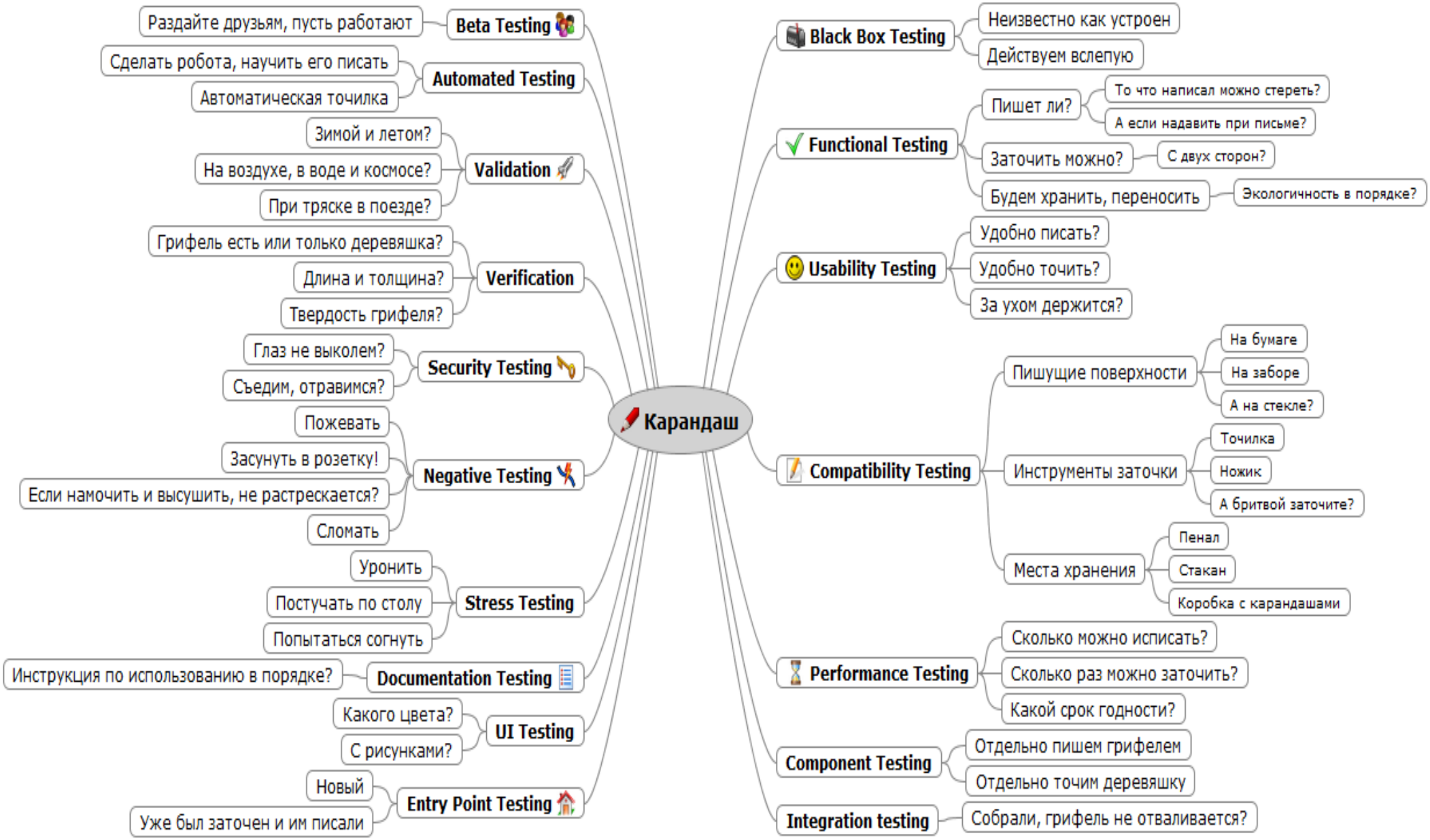
● Легко поддерживаться

● Легко читаться и быть простыми для понимания (даже новый разработчик должен понять что именно тестируется)

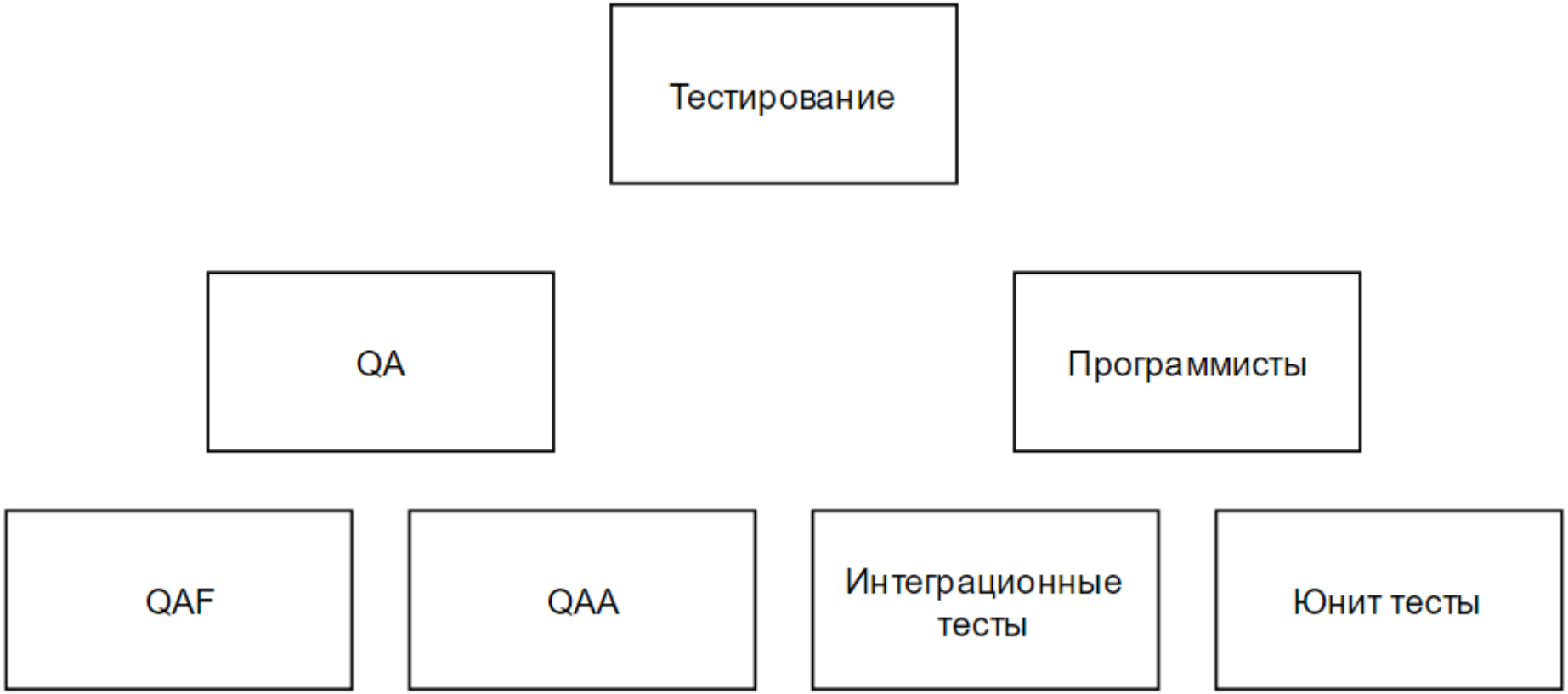
● Соблюдать единую конвенцию именования

● Запускаться регулярно в автоматическом режиме

В реальности качество ПО складывается из огромного списка критериев:



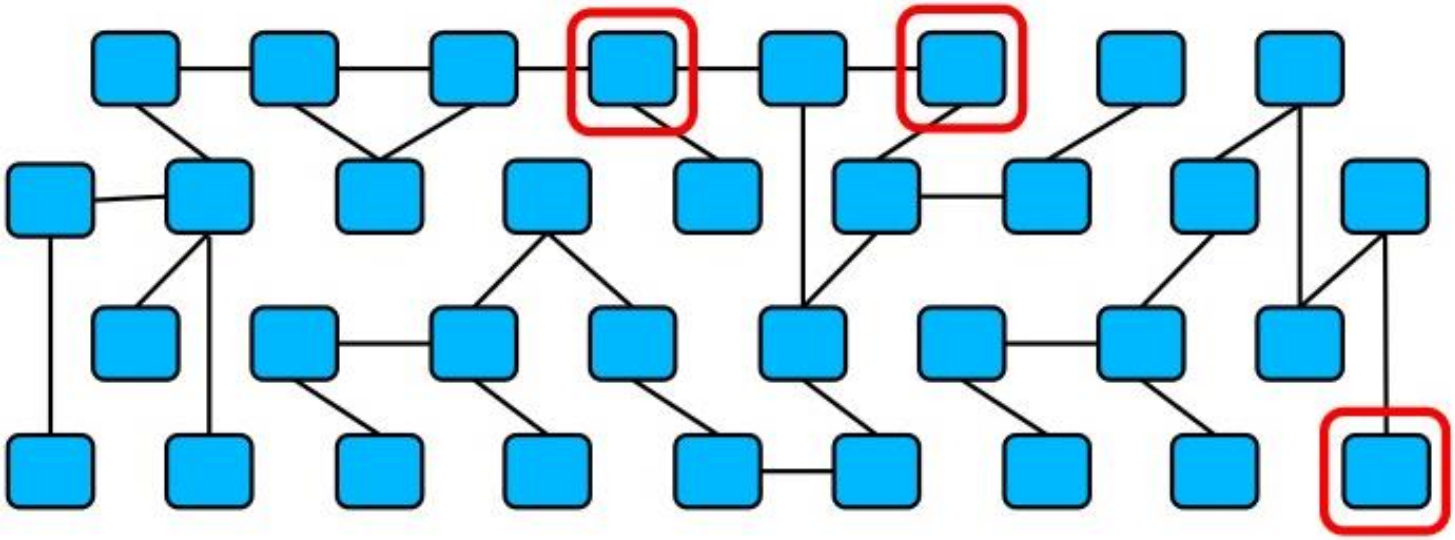
Тестирование – что это и зачем?



Тестирование – что это и зачем?

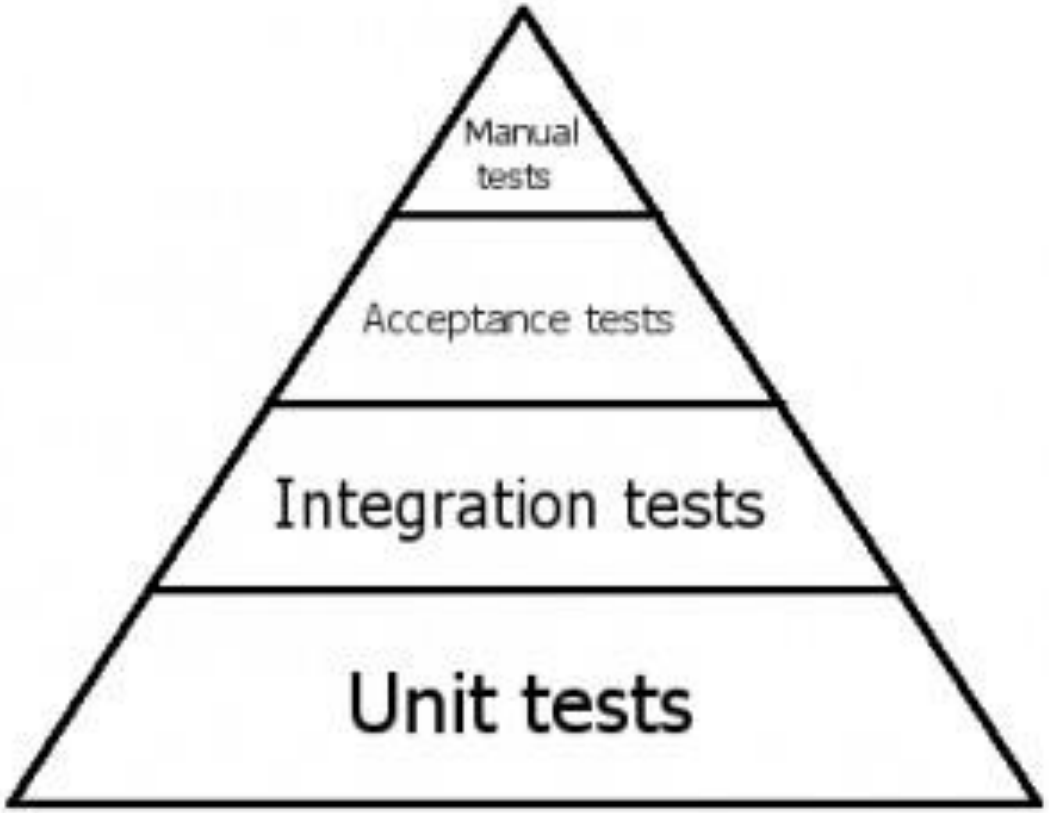
Модульное тестирование

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



Тестирование – что это и зачем?

Модульное тестирование



Таким образом, юнит-тестирование – это первый бастион на борьбе с багами. За ним еще интеграционное, приемочное и, наконец, ручное тестирование, в том числе «свободный поиск».

Dependency injection (DI)

Внедрение зависимости (англ. Dependency injection, DI) — процесс предоставления внешней зависимости программному компоненту.
При внедрении зависимости объект пассивен и не предпринимает вообще никаких шагов для выяснения зависимостей, а предоставляет для этого сеттеры и/или принимает своим конструктором аргументы, посредством которых внедряются зависимости.

Dependency injection (DI)

```
public class Printer
{
    private readonly IDBProvider dbprovider;
    public string lastTextWrittenInConsole;

    3 references
    public Printer(IDBProvider dbprovider)
    {
        this.dbprovider = dbprovider;
    }

    3 references
    public void PrintUserInfoById(int userId)
    {
        if (dbprovider != null)
        {
            var user = dbprovider.GetUserById(userId);
            var text = $"UserId: {user.UserId}, User name: {user.FirstName} {user.SecondName}";
            Console.WriteLine(text);
            lastTextWrittenInConsole = text;
        }
    }
}
```

Dependency injection (DI)

```
public class DBProvider : IDBProvider
{
    1 reference
    public int GetUserId()
    {
        //...connect to db and get user id;
        //return 1;
        throw new NotImplementedException();
    }

    4 references
    public User GetUserById(int userId)
    {
        //...connect to db and get user;
        //return user;
        throw new NotImplementedException();
    }
}
```

Dependency injection (DI)

```
public class DBProviderFake : IDBProvider
{

    4 references
    public User GetUserById(int userId)
    {
        //var user = new User() { UserId = 10, FirstName = "FirstName", SecondName = "SecondName" };
        //var user = new Fixture().Create<User>();
        var user = new Fixture().Build<User>()
                                .With(u => u.FirstName, "FirstName")
                                .With(u => u.SecondName, "SecondName")
                                .Create();

        return user;
    }
}
```

Основные виды Unit-тестов

• **Тесты состояния (state-based tests)** — тесты, проверяющие что вызываемый метод объекта отработал корректно, проверяя состояние тестируемого объекта после вызова метода. Базируются на Stub-объектах.
• **Stub-объекты (стабы) –** это типичные заглушки. Они ничего полезного не делают и умеют лишь возвращать определенные данные в ответ на вызовы своих методов.

```
class DBProvider : IDBProvider
{
    public string GetUserNameById(int userId)
    {
        //...connect to db and get user name;
        var realName = "Real User Name"; // db.Users.Find(user => user.Id == userId).Name;
        return realName;
    }
}

class DBProviderStub : IDBProvider
{
    public string GetUserNameById(int userId)
    {
        var fakeName = "Fake User Name";
        return fakeName;
    }
}
```

Основные виды Unit-тестов

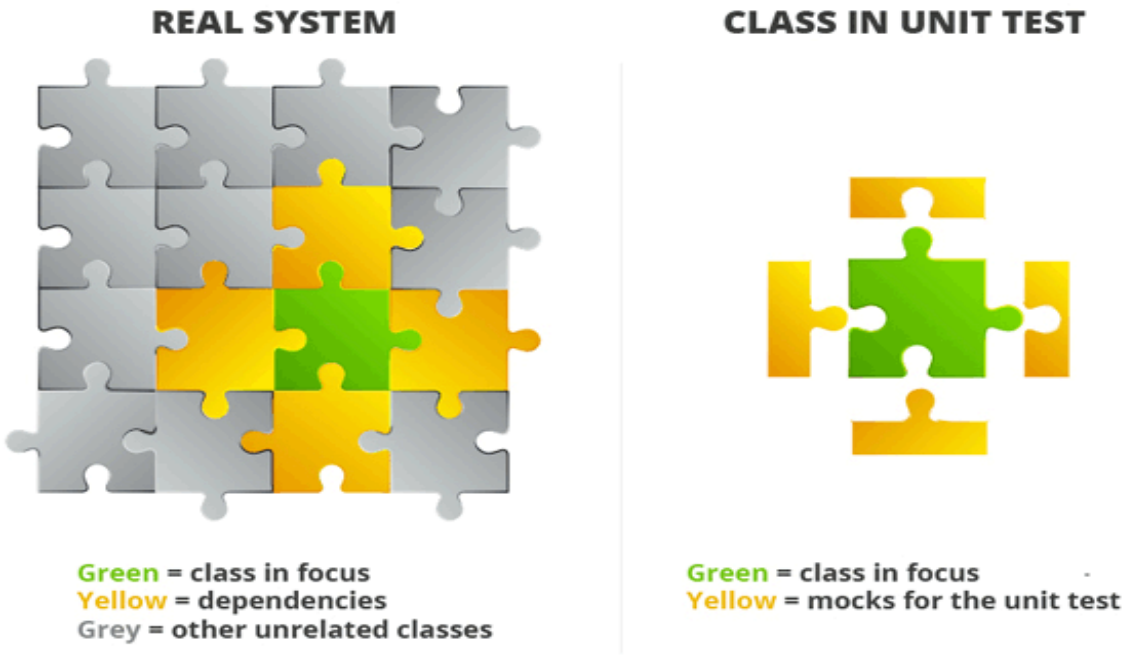
```
class DBProvider : IDBProvider
{
    public string GetUserNameById(int userId)
    {
        //...connect to db and get user name;
        var realName = "Real User Name"; // db.Users.Find(user => user.Id == userId).Name;
        return realName;
    }
}

class DBProviderStub : IDBProvider
{
    public string GetUserNameById(int userId)
    {
        var fakeName = "Fake User Name";
        return fakeName;
    }
}
```

Основные виды Unit-тестов

• **Тесты взаимодействия (interaction tests)** — это тесты, в которых тестируемый объект производит манипуляции с другими объектами. Применяются, когда требуется удостовериться, что тестируемый объект корректно взаимодействует с другими объектами.

• Базируются на **Mock** объектах, которые являются заменой реальных объектов системы и позволяют проверить вызовы своих членов тестируемым классом



Основные виды Unit-тестов

```
[TestMethod()]
0 references
public void PrintUserInfoByIdTestByMock()
{
    // Arrange
    var mock = new Mock<IDBProvider>();
    var printer = new Printer(mock.Object);

    // old school
    User user = new User() { UserId = 10, FirstName = "FirstName", SecondName = "SecondName" };
    // AutoFixture
    user = new Fixture().Create<User>();

    mock.Setup(x => x.GetUserById(It.IsAny<int>())).Returns(user);
    var userId = new Fixture().Create<int>();

    // Act
    printer.PrintUserInfoById(userId);

    //Assert
    Assert.IsNotNull(printer.lastTextWrittenInConsole);
}
```

Unit tests

Расположение тестов. Именование проектов, классов.

- Тесты должны быть частью контроля версий (находиться в репозитории проекта)
- Каждый тестирующий класс должен тестировать только одну сущность (класс).
- 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
- Добавляйте к каждому проекту его собственный тестовый проект

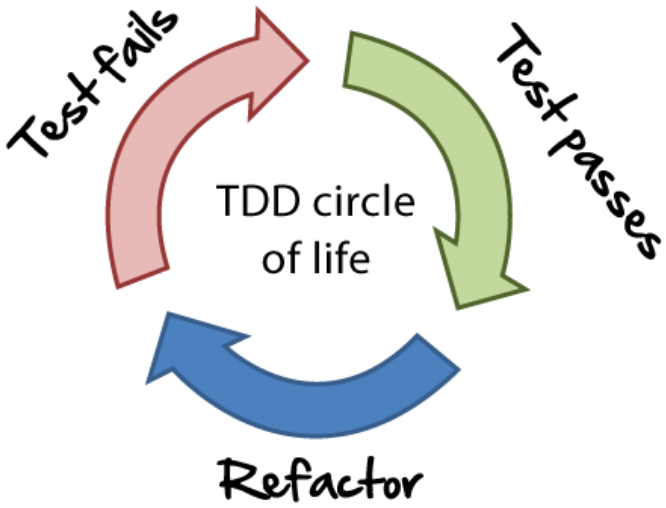
Проекты:	Проекты с тестами:
<PROJECT_NAME>.Core	<PROJECT_NAME>.Core.**Tests**
<PROJECT_NAME>.BL	<PROJECT_NAME>.BL.**Tests**
<PROJECT_NAME>.Web	<PROJECT_NAME>.Web.**Tests**

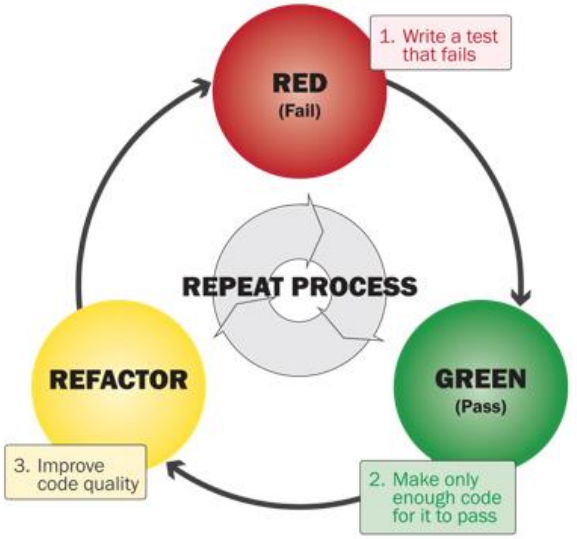
Классы и методы:	Тестовые классы и методы:
ClassName	ClassName**Tests**
ClassName.Method	ClassNameTests.**Method_Scenario_ExpectedResult**
	Sum_5plus3_8returned

Test Driven Development (TDD)

Разработка через тестирование (TDD) процесс применения юнит-тестов, при котором сначала пишутся тесты, а потом уже программный код, достаточный для выполнения этих тестов.

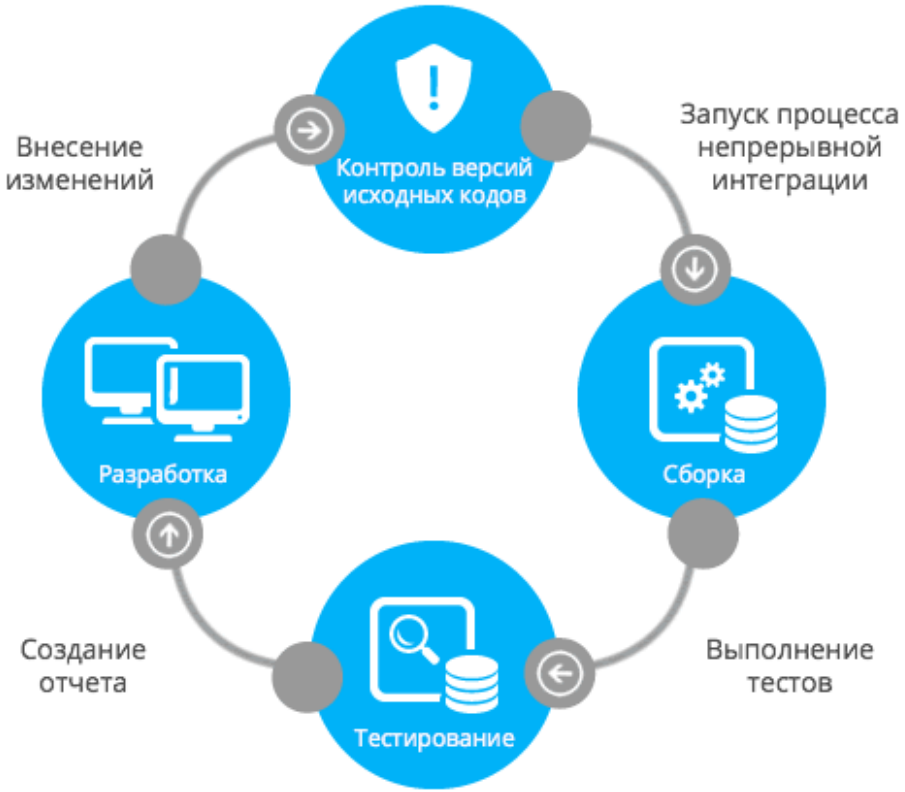
Использование TDD позволяет снизить количество потенциальных багов в приложении. Создавая тесты перед написанием кода, мы тем самым описываем способ поведения будущих компонентов, не связывая себя при этом с конкретной реализацией этих тестируемых компонентов (тем более что реализация на момент создания теста еще не существует).





Continuous Integration(CI)

Непрерывная интеграция — это среда, которая следит за вашим репозиторием, и при появлении там изменений автоматически стягивает их, билдит, выполняет тесты (конечно, если их пишут) и возможно выкладывает готовую версию на тестовый стенд. В случае же неудачи она дает об этом знать всем заинтересованным лицам, в первую очередь – последнему коммитеру.



Continuous Integration(CI)

Что нам это дает?

Во первых – в любой момент времени мы имеем достоверную информацию о состоянии исходников в системе. Если последний билд был неудачным («упал»), значит брать свежую версию из сорц-контрола нельзя – он может даже не компилится, а если зелёненький, удачный – значит все отлично.

Во-вторых – очень просто найти виновника «торжества» — скорее всего, это последний коммитер – он то и будет отвечать за «ремонт». В подобной среде задачей высочайшего приоритета является исправление билда.

#	Results	Artifacts	Changes	Started	Duration
#2120	Tests passed: 2	None	Maxim Podkolzine (1)	15 Feb 12 15:27	17m:26s
#2118	Tests failed: 2 (2 new), passed: 6360, ignored: 127	None	Changes (2)	15 Feb 12 15:19	2h:25m
#2116	Tests failed: 2 (2 new), passed: 7098, ignored: 35	None	Changes (2)	15 Feb 12 13:05	2h:33m
#2115	Tests passed: 7618, ignored: 35	None	Changes (7)	14 Feb 12 23:51	2h:49m
#2114	Tests passed: 2655, ignored: 24	None	Changes (4)	14 Feb 12 22:19	1h:31m
#2112	Tests failed: 2 (2 new), passed: 7098, ignored: 35	None	Changes (4)	14 Feb 12 21:19	2h:33m
#2110	Tests passed: 8181, ignored: 35	None	Changes (2)	14 Feb 12 19:30	2h:48m
#2109	Tests failed: 1 (1 new), passed: 8180, ignored: 35	None	Eugene Petrenko (1)	14 Feb 12 19:14	6h:51m
#2108	Tests passed: 7098, ignored: 35	None	Changes (3)	14 Feb 12 18:50	2h:28m
#2107	Tests passed: 7102, ignored: 35	None	Changes (4)	14 Feb 12 17:18	2h:43m
#2106	Tests failed: 1 (1 new), passed: 7096, ignored: 35	None	nikita.skvortsov (1)	14 Feb 12 16:37	2h:46m
#2105	Tests passed: 2594, ignored: 24	None	Changes (3)	14 Feb 12 15:16	1h:19m
#2102	Tests passed: 2594, ignored: 24	None	Changes (2)	14 Feb 12 13:45	1h:18m

Continuous Integration(CI)

#	Results	Artifacts	Changes	Started	Duration
#2120	Tests passed: 2	None	Maxim Podkolzine (1)	15 Feb 12 15:27	17m:26s
#2118	Tests failed: 2 (2 new), passed: 6360, ignored: 127	None	Changes (2)	15 Feb 12 15:19	2h:25m
#2116	Tests failed: 2 (2 new), passed: 7098, ignored: 35	None	Changes (2)	15 Feb 12 13:05	2h:33m
#2115	Tests passed: 7618, ignored: 35	None	Changes (7)	14 Feb 12 23:51	2h:49m
#2114	Tests passed: 2655, ignored: 24	None	Changes (4)	14 Feb 12 22:19	1h:31m
#2112	Tests failed: 2 (2 new), passed: 7098, ignored: 35	None	Changes (4)	14 Feb 12 21:19	2h:33m
#2110	Tests passed: 8181, ignored: 35	None	Changes (2)	14 Feb 12 19:30	2h:48m
#2109	Tests failed: 1 (1 new), passed: 8180, ignored: 35	None	Eugene Petrenko (1)	14 Feb 12 19:14	6h:51m
#2108	Tests passed: 7098, ignored: 35	None	Changes (3)	14 Feb 12 18:50	2h:28m
#2107	Tests passed: 7102, ignored: 35	None	Changes (4)	14 Feb 12 17:18	2h:43m
#2106	Tests failed: 1 (1 new), passed: 7096, ignored: 35	None	nikita.skvortsov (1)	14 Feb 12 16:37	2h:46m
#2105	Tests passed: 2594, ignored: 24	None	Changes (3)	14 Feb 12 15:16	1h:19m
#2102	Tests passed: 2594, ignored: 24	None	Changes (2)	14 Feb 12 13:45	1h:18m

Домашнее задание

Доработка существующего приложения

1. Добавить тестовый проект для вашего решения

2. Максимально возможно покрыть тестами весь основной функционал вашего приложения. Проявите фантазию!

Практика

Создать простое приложение с несколькими тестами

1. Создать простое приложение (консоль)
2. Реализовать простой поставщик данных (Можно без коннекта к БД)
3. Реализовать класс “Принтер” который будет принимать поставщик данных (через интерфейс)
4. Создать класс юнит тестов и реализовать фейковый поставщик данных
5. Написать тесты (1-2)

Спасибо за внимание!

Артамонов Николай (Skype – art732009)
Разработчик .NET

+7 (8422) 44-66-91
+7 (495) 133-90-01
www.simbirsoft.com