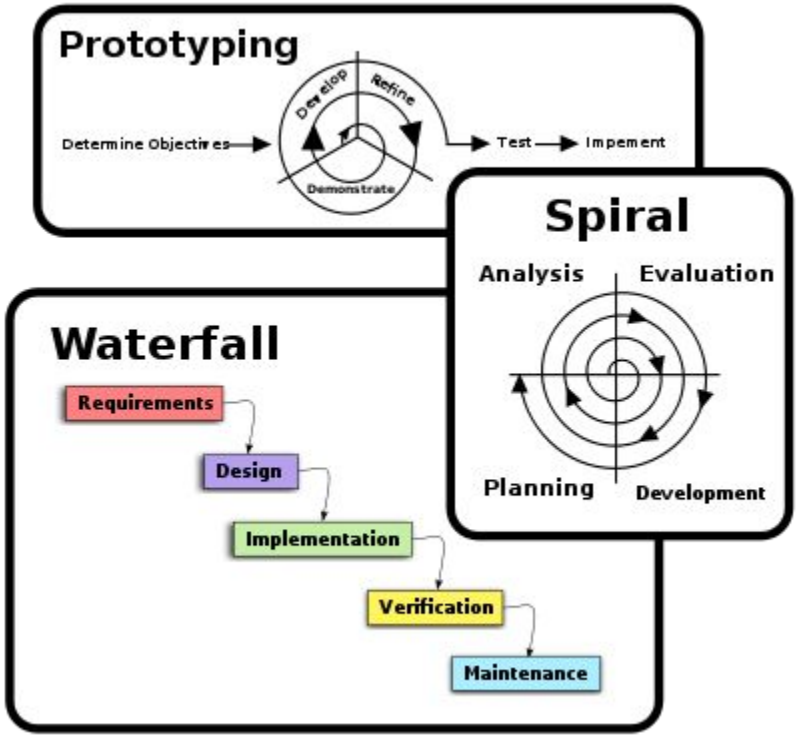
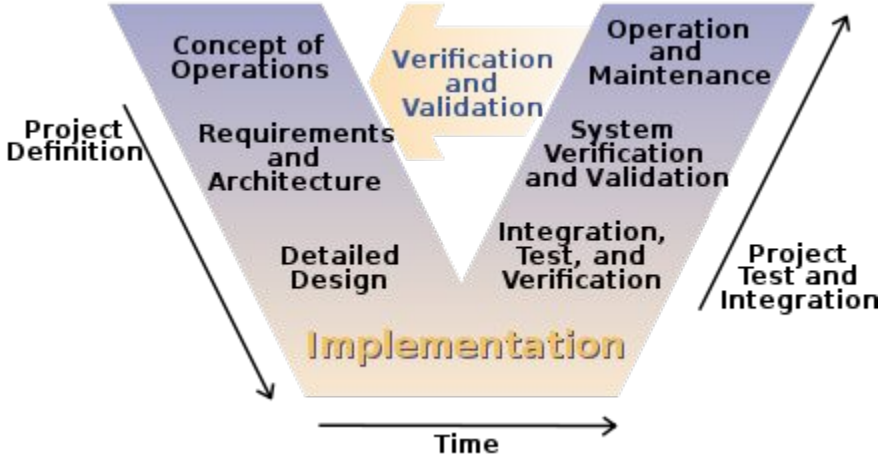
TDD - test-driven development

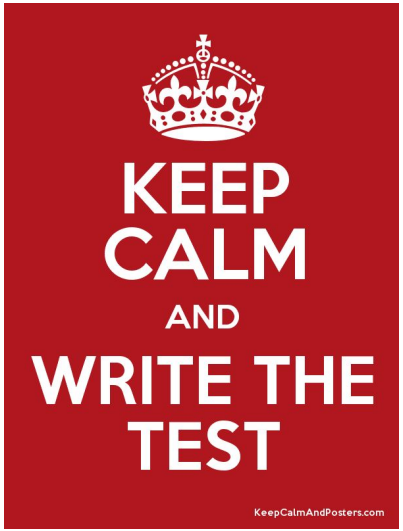
Student of group 555-vm
Dmitriy Aseev
dmitriy.aseev@it-devgroup.com

Software development methodologies

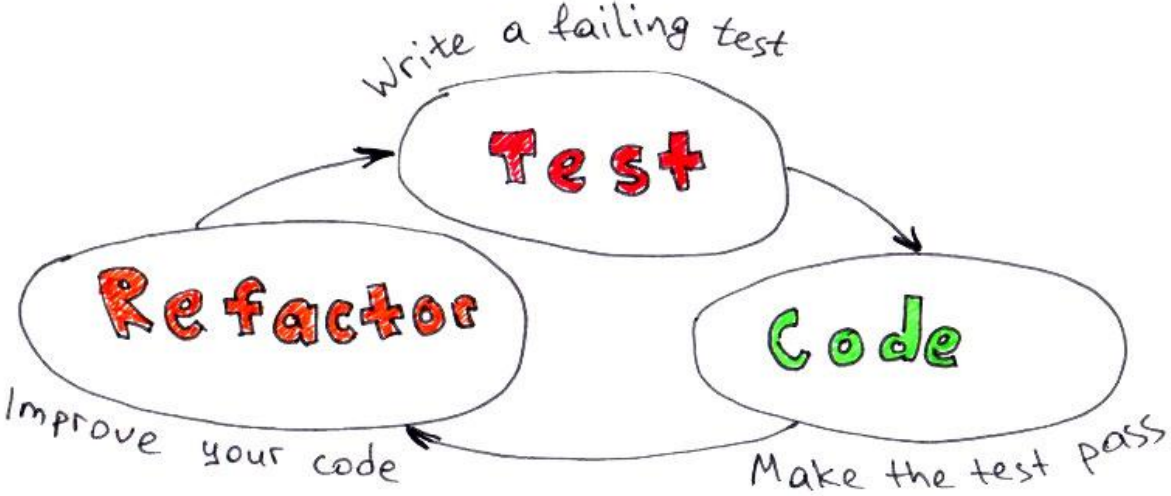




TDD

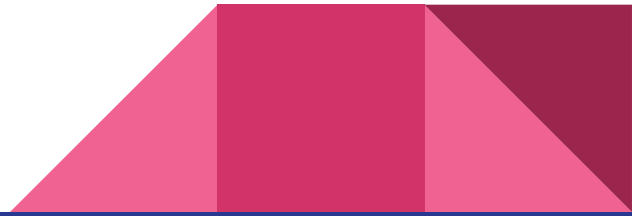


Test-driven development (TDD) is an evolutionary approach to development which combines test-first development where you write a test before you write just enough production code to fulfill that test and refactoring.



Example

```
interface ICalculable
{
    int Sum(int x, int y);

    int Sub(int x, int y);

    int Div(int x, int y);

    int Mult(int x, int y);
}

class Calculator : ICalculable
{
    ...
}
```



C# or PHP

```php
class CalculatorTest
{
    public function should_sum_two_and_five()
    {
        $calc = new Calculator();

        $res = $calc->sum(2,5);

        $this->assertEquels(7, $res);
    }
}
```

```csharp
class CalculatorTests
{
    public void should_sum_two_and_five()
    {
        var calc = new Calculator();

        var res = calc.Sum(2,5);

        Assert.AreEqual(7, res);
    }
}
```

```php
class CalculatorTest
{
    public function test_calc()
    {
        $calc = new Calculator();

        $res = $calc->sum(2,5);
        $this->assertEquels(7, $res);

        $res = $calc->sub(10,5);
        $this->assertEquels(5, $res);

        $res = $calc->div(10,5);
        $this->assertEquels(2, $res);
    }
}
```

?

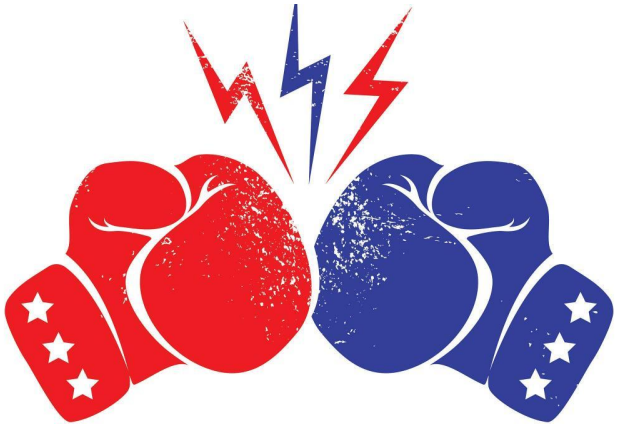
```php
class CalculatorTest
{
    public function test_sum_values()
    {
        $calc = new Calculator();
        $res = $calc->sum(2,5);
        $this->assertEquels(7, $res);
    }

    public function test_sub_values()
    {
        $calc = new Calculator();
        $res = $calc->sub(10,5);
        $this->assertEquels(5, $res);
    }

    public function test_div_values()
    {
        $calc = new Calculator();
        $res = $calc->div(10,5);
        $this->assertEquels(2, $res);
    }
}
```

```php
class CalculatorTest
{
    public function should_sum_values()
    {
        $calc = new Calculator();
        $res = $calc->sum(2,5);
        $this->assertEquels(7, $res);
    }

    public function should_sub_values()
    {
        $calc = new Calculator();
        $res = $calc->sub(10,5);
        $this->assertEquels(5, $res);
    }

    public function should_div_values()
    {
        $calc = new Calculator();
        $res = $calc->div(10,5);
        $this->assertEquels(2, $res);
    }
}
```

Behavior Driven Development

TDD

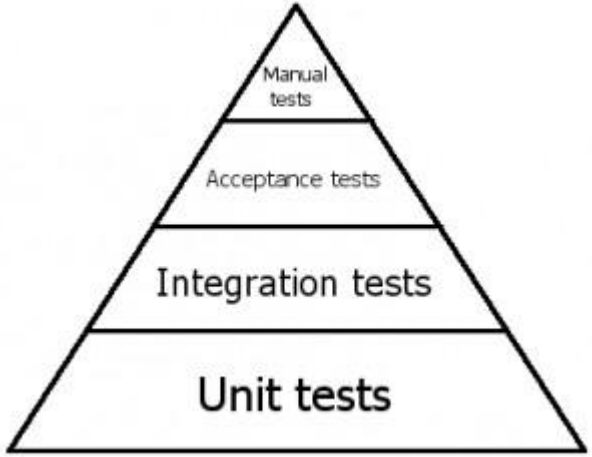
BDD



Tests

Integration testing is the phase in software testing in which individual software modules are combined and tested as a group

Acceptance testing - formal testing with respect to user needs, requirements, and business processes conducted to determine whether or not a system satisfies the acceptance criteria and to enable the user, customers or other authorized entity to determine whether or not to accept the system.



Не нужно писать тесты, если

Вы делаете простой **сайт-визитку** 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»

Вы занимаетесь рекламным сайтом/простыми флеш-играми или баннерами – сложная верстка/анимация 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**

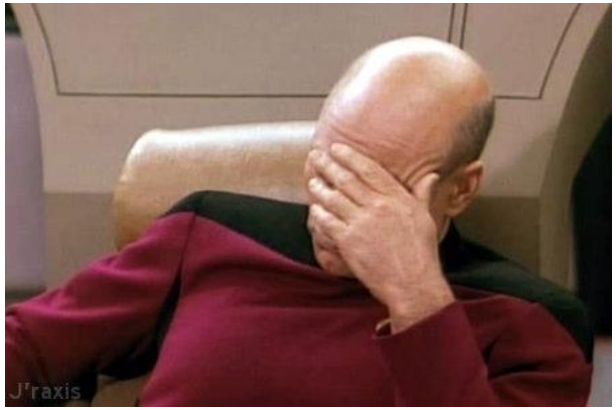
Вы всегда пишете код **без ошибок**, обладаете идеальной памятью и даром предвидения. Ваш **код настолько крут**, что изменяет себя сам, вслед за требованиями клиента. Иногда код объясняет клиенту, что его требования — не нужно реализовывать

Типы проектов

1. **Без покрытия тестами.**

2. **С тестами, которые никто не запускает и не поддерживает.**

3. **С серьезным покрытием. Все тесты проходят.**



Советы при написании тестов

Быть достоверными

Не зависеть от окружения, на котором они выполняются

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

Без велосипедов, пожалуйста

.NET:

MsTest (есть в студии)

NUnit

PHP:

PHPUnit

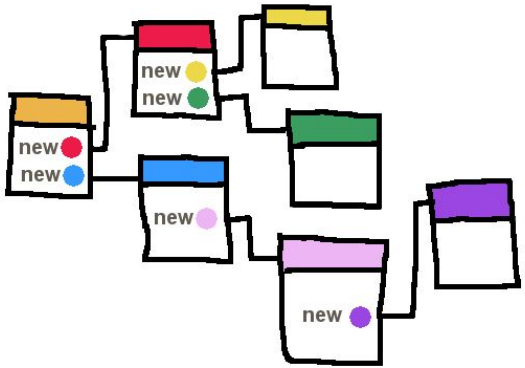
Codeception

Подход AAA (arrange, act, assert)

```
class CalculatorTests
{
        public void Sum_2Plus5_7Returned()
        {
                // arrange
                var calc = new Calculator();

                // act
                var res = calc.Sum(2,5);

                // assert
                Assert.AreEqual(7, res);
        }
}
```

```
class CalculatorTests
{
        public void Sum_2Plus5_7Returned()
        {
                Assert.AreEqual(7, new Calculator().sum(2,5));
        }
}
```

Борьба с зависимостями



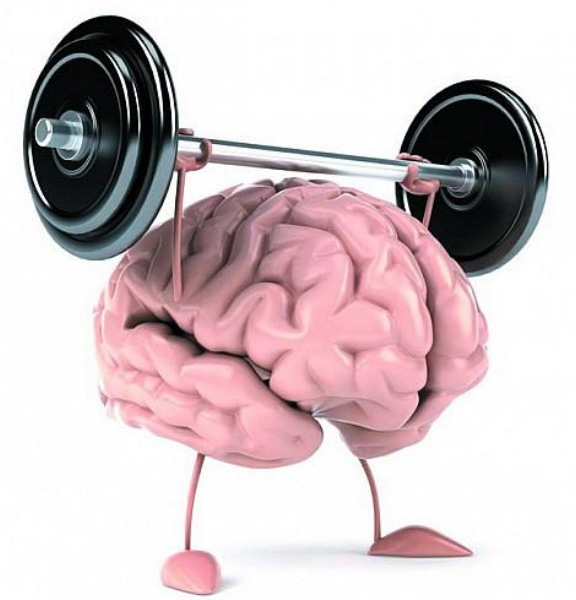
Выделяют два типа подделок: стабы (stubs) и моки (mock).

Часто эти понятия путают. Разница в том, что стаб ничего не проверяет, а лишь имитирует заданное состояние. А мок – это объект, у которого есть ожидания. Например, что данный метод класса должен быть вызван определенное число раз. Иными словами, ваш тест никогда не сломается из-за «стаба», а вот из-за мока может.

Как начать писать тесты?

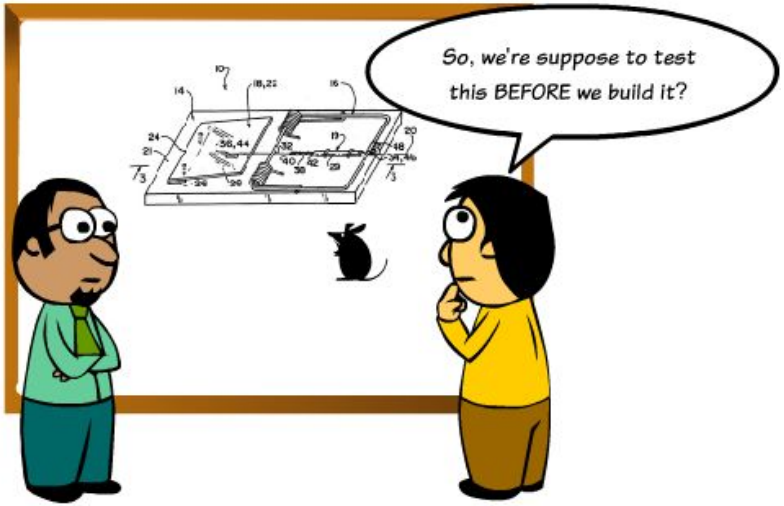
1. Лабораторные работы
2. Курсовые проекты
3. Дипломные работы

Саморазвитие :)

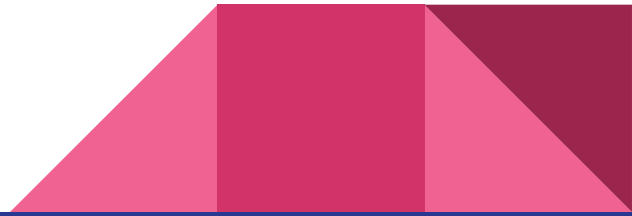


Выводы

Любой долгосрочный проект без надлежащего покрытия тестами обречен рано или поздно быть переписанным с нуля.



Вопросы в студию!



Практика

1. Сортировать массив из 10 элементов
2. Взаимодействовать с платежной системой (единоразовый платеж)
3. Чат (сервер и 2 клиента)
4. Транзакция (проверка целостности всех данных)
5. Мультиязычность (Content, Language, …)
6. Роли (Декан, Преподаватель, Студент)
7. Soft delete

Паттерны