Тестирование

Семинар #2

Что это?

Тестирование - процесс испытания программы для проверки соответствия между реальным поведением программы и ожидаемым.

Зачем?

- Чтобы быть уверенным в том, что ваш код готов к эксплуатации
- Чтобы другие разработчики могли рассчитывать на ваш код
- Чтобы при изменении кода чётко следить за контрактами и поведением
- Чтобы хвастаться своим покрытием кода перед друзьями :)

Как?

- Unit тестирование
- Функциональное тестирование
- Интеграционное тестирование
- Тестирование производительности
- И многое другое… (usability, security, localization)

Структура тестов

Каждый тест состоит из трёх блоков:

- Подготовка данных
- Выполнение действий
- Проверка результатов

Важно! Один тест проверяет одну ситуацию!

Если в тесте проверяется более одной ситуации - нужно разбить тест.

Как писать тесты?

- Создать обычный проект
- Подключить с помощью NuGet в проект NUnit Framework 3
- Создать класс, в котором будут находиться тесты
- Пометить его атрибутом **[TestFixture]**
- Создать метод, который будет тестировать определенную функциональность
- Пометить его атрибутом **[Test]**
- …
- **PROFIT**

Assert

Ключевой этап тестирования - проверка результатов.

Проверить результаты можно с помощью класса Assert.

Например:

```
[Test]
public void Example_Test()
{
    var actorName = "Ryan Gosling";

    var realHumanBean = DoSomethingCool(actorName);

    Assert.AreEqual(realHumanBean, "a real hero!");
}
```

Задание #1

Файл Task1_Tests.cs

Задание #1

- Откройте файл Task1_Tests.cs
- Напишите тесты на `Tasks.SumAbs` (используйте `this.Tasks` для тестирования)
- Запустите программу! (не тесты)
- Один Case - один тест

Те, кто всё сделал - можете почитать про атрибут **[TestCase]**

TestCase

Атрибут [TestCase] используется для уменьшения дублирования кода.
Когда много case проверяются одним кодом - можно обобщить это.
Но не стоит увлекаться и обобщать всё, надо использовать атрибут разумно!
Пример:

```
[TestCase("", "content")]
[TestCase("filename", "content")]
[TestCase("filename", "123456")]
[TestCase("4563587", "123456")]
[TestCase("4563587", "content")]
[TestCase("русский", "текст")]
[TestCase("filename", "текст")]
[TestCase("русский", "content")]
public void Save_WithValidParameters_ShouldReturnCorrectMeta(string filename, string content)
{
    var expected = new FileMeta(filename, 0, content.Length);

    var actual = this.Drive.Save(filename, content, 0);

    Assert.AreEqual(expected, actual);
}
```

Правило именования тестов

Существует множество разных вариантов именования тестов.

- ЧтоДолжноПроизойти_ПриКакихУсловиях
- ЧтоТестируем_ПриКакихУсловиях_ЧтоДолжноПроизойти
- …

Примеры:

```
[Test]
public void BeSumOfAbs_WhenFirstNegative()
[Test]
public void Load_MissingFilename_ShouldThrowFileNotFoundException()
```

Задание #2

Файл Task2_Tests.cs

Задание #2

- Откройте файл Task2_Tests.cs
- Напишите тесты на `Tasks.Move` (используйте `this.Tasks` для тестирования)
- Запустите программу! (не тесты)
- Один Case - один тест

Те, кто всё сделал - можете почитать про **CollectionAssert**

Сравнение сложных объектов

- Сравнивать все поля по отдельности **(плохо)**
- Переопределить Equals для тестирования **(плохо)**
- Воспользоваться Equals, если он уже есть
- Написать специальный Assert
- Использовать готовые специальные Assert, например, CollectionAssert
- Использовать библиотечки, которые умеют глубокое сравнение

Граничные условия

- Минимальное значение
- На один меньше минимального
- Максимальное значение
- На один больше максимального
- null
- 0
- ...

Задание #3

Файл Task3_Tests.cs

Задание #3

- Откройте файл Task3_Tests.cs
- Напишите тесты на `Tasks.Distinct` (используйте `this.Tasks` для тестирования)
- Запустите программу! (не тесты)
- Один Case - один тест

Те, кто всё сделал - можете почитать про проверку исключений в тестах.

Во втором и третьем задании можно написать тесты на поведение, которое будет если передать null в качестве одного из аргументов.

Обработка исключений

- Assert.DoesNotThrow(TestDelegate code);
- Assert.Throws<ExceptionType>(TestDelegate code);
- Assert.Catch<ExceptionType>(TestDelegate code);

Задание #4

Проект Drive

Задание #4

- Создайте проект Drive_Tests
- Напишите тесты на Drive.cs. При этом нужно попытаться учесть все возможные случаи и проверить все пороговые условия. А также исправить баги.
- Если осталось время, напишите тесты на методы, которые можно найти в файле DriveAdvanced.cs