

Аппрувал- тестирование в .NET

Константин Финагин

Аппрувал- тестирование в .NET

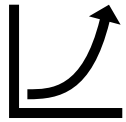
Или как подружить тесты, таблицы и DIFF



Константин Финагин

Ст. разработчик в Kaspersky

konstantin.finagin@gmail.com



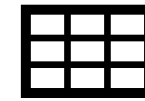
Юнит-тесты и рост их сложности

Как растет сложность тестов
и что с этим можно сделать



Аппрувал-тесты

Как один из способов
ускорить и упростить
тестирование



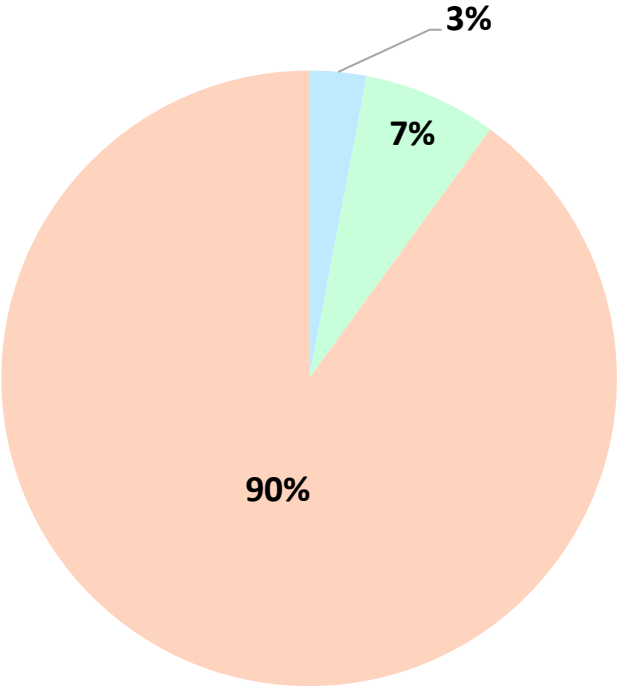
Таблицы

И чем они могут помочь при
использовании аппрувал-
тестирования

+ бонус: Чем могут помочь F# и excel

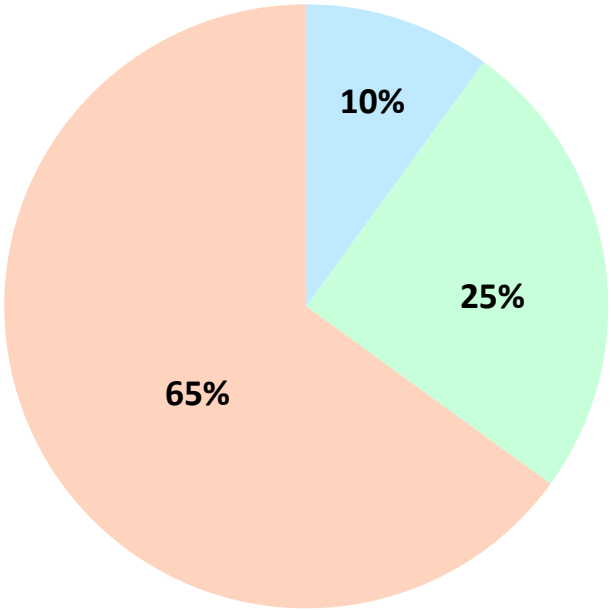


Персональная



Общая

(оценка по GitHub, SO, конференциям)



- Слышали и применяют
- Слышали
- Не слышали

Что мы знаем о юнит тестах?

Гарантируют стабильность кода
И снижают технический долг

Dependency Injection

Неразрывно связаны с DI через
моки

Пишутся обычно в трех случаях:

- Покрыть старую логику
- TDD
- Нашелся баг

Структура теста из трех секций

Arrange

Вывод моков и их реакции на входные данные

Act

Вызов метода. Самая короткая.

Assert

Проверяются поля результата, вызовы и данные методов моков

```
[Fact]
public void Test()
{
    // Arrange
    _service1Mock.When(...).Do(...);
    _service2Mock.When(...).Returns(...);

    // Act
    var outputData = _testedService.DoSomething(inputData);

    // Assert
    Assert.Equal("ExpectedValue", outputData.Field1);
    _service1Mock.Received(1).DoSomethingElse();
}
```

Когда тесты начинают усложняться

Простые тесты

Не вызывают проблем

Если тест сложно писать

То надо еще раз посмотреть на архитектуру

Но! Метод может быть сложен сам по себе

Акт остается коротким

Но Arrange и Assert растут

ACTUAL CODE



@THEJENKINScomic

UNIT TESTS



Часть 1. Рост сложности секции Assert

В ней мы знакомимся с необходимостью
использования Approvals

Сценарий 1. Много ассертов.

**Метод возвращает сложный
объект**

Много полей, иерархия

Секция Assert растёт

С ростом количества полей
объекта

```
[Fact]
public void Test()
{
    // Arrange
    var dataReceivedByMock1 = new Data1();
    _service1Mock.When(...).Do(_ => dataReceivedByMock1 = ...);
    _service2Mock.When(...).Return();

    // Act
    var outputData = _testedService.DoSomething(inputData);

    // Assert
    Assert.Equal("ExpectedValue1", outputData.Field1);
    Assert.Equal("ExpectedValue2", outputData.Field2);
    Assert.Equal("ExpectedValue51", outputData.Field5.Child1);
    Assert.Equal("ExpectedValue52", outputData.Field6.Child2);
    Assert.Equal("DataReceivedByMock1", dataReceivedByMock1.Field1);
    Assert.Equal("DataReceivedByMock2", dataReceivedByMock1.Field2);
}
```

```
public class BasePositionGetResponse
{
    public string CurrencyCode { get; set; }
    public decimal? Funded { get; set; }
    public decimal? Unfunded { get; set; }
    public decimal? Committed { get; set; }
    public decimal? CashOnCash { get; set; }
    public decimal? FundedPIK { get; set; }
    //...
}
public class ExtendedPositionGetResponse : BasePositionGetResponse
{
    public string DimensionId { get; set; }
    public string DimensionName { get; set; }
    public List<ExtendedPositionGetResponse> ChildPositionsResponses { get; set; }
    public Dictionary<int, BasePositionGetResponse> UnitrancheClassPositions { get; set; }
    //...
}
...
await _positionService.GetPositionsAsync(request); // один вызов! нужен тест
```

Рост сложности теста

Много ассертов

Рост количества строк кода, из-за чего трудно воспринять тест целиком

Изменение логики

Добавление полей и пропуск ассертов

Проверка логирования и

других сайд-эффектов

Mocks, ILogger

В итоге тест может не отражать реальное поведение

Несмотря на 100% покрытие

Как проверить
все данные
без «ада»
ассертов?

Сериализовать в строку?

Да, но...

**Поймаем только первое
расхождение**

Остальные ошибки придется
ловить следующими запусками

**Должны иметь ожидаемый
результат, эталон**

И заранее создать его вручную

Approval тесты (aka Snapshot)

Автоматизация сравнения строк

С использованием Diff-инструмента

Два прогона теста

1: файл .received (красный)

2: файл .approved / .verified (зеленый)

Approve - одобрение результата

Нужно просмотреть и одобрить текстовое представление результата

Если логика снова меняется?

.received \neq .approved,
тест снова красный

<https://github.com/approvals/ApprovalTests.Net> - зрелая базовая библиотека

<https://github.com/VerifyTests/Verify> - более современная альтернатива


```
[UseReporter(typeof(DiffReporter))]  
public class TestWithApprovals  
{  
    [Fact]  
    public void Test()  
    {  
        // Arrange  
        var processor = new Processor();  
        var input = new InputData();  
  
        // Act  
        var result = processor.Process(input);  
  
        // Assert  
        ApprovalTests.Approvals.Verify(ToJson(result));  
    }  
  
    private string ToJson(object data)  
    {  
        return JsonSerializer.Serialize(data,  
            new JsonSerializerOptions { WriteIndented = true });  
    }  
}
```

```
public class TestWithVerify
{
    [Fact]
    public async Task Test()
    {
        // Arrange
        var processor = new Processor();
        var input = new InputData();

        // Act
        var result = processor.Process(input);

        // Assert
        await Verifier.Verify(result);
    }
}
```

Demo 1

Пример 1 – Базовые тесты

Запуск в CI/CD пайплайне

Не нужно запускать DIFF tool

Но по-прежнему нужно
сравнивать результаты

**Необходимо определять, где
запускается тест**

И выключать запуск Diff-
инструмента на тех средах, где это
не требуется

```
if(Environment.GetEnvironmentVariable("CI")?.ToLowerInvariant() == "true")  
{  
    _settings.DisableDiff();  
}
```

Настройка запуска в CI/CD пайплайне – ApprovalTests

```
ApprovalTests.Approvals.SetFrontLoadedReporter(  
    Environment.GetEnvironmentVariable("CI")?.ToLowerInvariant() == "true"  
    ? new QuietReporter()  
    : new DiffReporter());
```

```
#if DEBUG  
    [UseReporter(typeof(DiffReporter))]  
#else  
    [UseReporter(typeof(QuietReporter))]  
#endif
```

Частые ситуации

Частичное сравнение

Не нужна вся информация
объекта

Динамические данные

DateTime, GUID, числовые
значения с определенным
допуском

Меняющиеся списки и строки

Значение в динамическом списке
или строке, Contains/StartsWith

```
var partial = new
{
    result.ExternalInfo.Category,
    result.Metadata.CharFrequency,
    result.IsPalindrome,
};

await Verifier.Verify(partial);

_settings.IgnoreMember<ExternalAnalysis>(x => x.Tags);
_settings.IgnoreMember<OutputData>(x => x.RawProcessingLog);
_settings.ScrubInlineDateTimes("s");

...
await Verifier.Verify(result, _settings);
```

Динамические данные

Необходимо привести к статичным
DI, “врапперы”, “скрабберы”
(убираем из результата
динамические поля)

Универсальное форматирование
Убрать зависимость от локали
(особенно DateTime) – **важно для**
CI/CD

Смотреть по ситуации
Может быть именно в этом случае
аппрувалы и не нужны




```
public static class DateTimeWrapper // минус подхода – статика, непараллельность
{
    private static Func<DateTime> _utcNowProvider = () => DateTime.UtcNow;
    public static DateTime UtcNow => _utcNowProvider();

    public static void Set(Func<DateTime> customProvider)
    {
        _utcNowProvider = customProvider
            ?? throw new ArgumentNullException(nameof(customProvider));
    }

    public static void Reset()
    {
        _utcNowProvider = () => DateTime.UtcNow;
    }
}

...

var date = DateTimeWrapper.UtcNow();
```

DO YOU TEST YOUR CODE?

YES

**YESTERDAY I DELETED A TEST THAT WAS
BLOCKING THE CI**

Сценарий 2. Табличные данные на выходе

Непосредственно табличные
DataTable, CSV,
двумерные массивы

Сводимые к табличным
Списки и иерархии объектов

JSON не подходит

Так как становится сложным для
восприятия

```
[Fact]
public class ApprovalTest
{
    // Arrange
    DataTable savedData = null;

    _repositoryMock
        .When(r => r.SaveDataTable(Arg.Is<DataTable>()))
        .Do(saved => savedData = saved);

    // Act
    _service.DoSomethingAndSave();

    // Assert
    var json = JsonHelper.ToJson(savedData);
    Approvals.Approve(json);
    var table = TableFormatter.ToAsciiTable(savedData);
    Approvals.Approve(table);
}
```

	Committed	Funded	Funded PIK	Unfunded
	9700000.00	2700000.00	21917.80	6978082.20
CustomerId	Committed	Funded	Funded PIK	Unfunded
2	6790000.00	1890000.00	15342.47	4884657.53
3	2910000.00	810000.00	6575.34	2093424.66

← 2 Таблицы

Иерархия в плоском виде



Level	Customer	ParticipationId	Expected	Actual
Lender	2	null	16.66	16.66
Grantor	2	1011	6.25	6.25
Participant	11	1011	3.12	3.12
Participant	12	1011	1.88	1.88
Participant	13	1011	1.25	1.25
Lender	3	null	16.67	16.67
Lender	4	null	16.67	16.67

Польза табличного формата

Удобный аппрув

Табличная структура упрощает просмотр полей при аппруве

Подсветка изменений в DIFF

При красном тесте и запуске DIFF можно увидеть конкретное расположение изменившегося поля

Demo 2

Пример 2 – Таблицы

Часть 2. Рост сложности секции Arrange

Упрощаем входные данные для теста при помощи F#

Сценарий 3. Данные от бизнес-аналитика

Формулы и данные в Excel

Формулы – для
программирования логики
Данные – для проверки

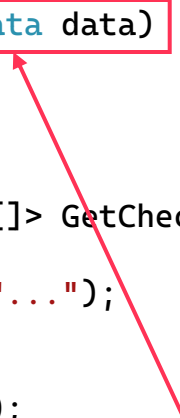
Стандартный подход для множественных данных (в Xunit)

Theory: InlineData, MemberData и
др.

```
[Theory]
[InlineData(16.66, 6.25, ...)]
[InlineData(12.66, 5.45, ...)]
public void TestTableCase1(double value1, double value2, ..., double result)
{
    ...
}
```

```
[Theory]
[MemberData(nameof(GetCheckTableData))]
public void TestTableCaseN(InputData data)
{
    ...
}

private static IEnumerable<object[]> GetCheckTableData()
{
    var lines = GetLinesFromFile("...");
    foreach (var line in lines)
    {
        var data = ParseLine(line);
        yield return new object[] { InputData = data };
    }
}
```



Как получить данные из Excel?

Сохранить как csv и использовать парсер?

Неплохой вариант, но нужно переводить в другой формат, парсить, использовать сторонние библиотеки

Нам может помочь F#

В нем есть готовый механизм для доступа к структурированным данным

Провайдеры типа в F#

Что это?

Мощный и удобный механизм типобезопасного доступа к структурированным данным

Есть провайдеры типа для разных источников

Excel: FSharp.ExcelProvider

CSV: встроен в FSharp.Data

А также баз данных

Автоматическая генерация типа

На основе файла во время компиляции

F# проект с провайдером типа можно подключить к тестам

И таким образом сократить секцию Arrange

```
module ExcelDataHelper =  
  
    open FSharp.Interop.Excel  
    open SpbDotNet.Approvals.Model  
  
    [    let private samplePath = __SOURCE_DIRECTORY__ + "/DataSamples/InputData.xlsx"  
  
    type InputDataExcel = ExcelFile<samplePath>  
  
    let private inputDataRows (sheet: InputDataExcel) =  
        sheet.Data  
        |> Seq.map (fun row -> InputData((int)row.UserId, row.RawInput))  
  
    let GetInputDataItems (path: string) =  
        let file = new InputDataExcel(path)  
        inputDataRows file |> Seq.toArray
```

Метод для использования в тесте



Demo 3

Пример 3 – Использование F# Type Providers

Сценарий 4.

Баг с прода

С прода пришел баг
Какой-то параметр принимает
ошибочное значение

**Есть: готовый ввод (request),
готовый вывод (response)**
А также логика в сервисах

**Необходимо: починить и
воспроизвести в тесте**



Гибридный тест – сумма подходов

Уже не юнит

Позволяет воспроизвести всю цепочку построения ответа

Еще не интеграционный

Некоторые сервисы могут быть замоканы

Текст запроса

Готовые данные для Arrange / Act

Текст ответа

Может быть использован для апрувала

```
public async Task GetPositionsForMultipleContracts()
{
```

Реальные данные запроса и ответа API

Частичный мок с реальными данными

Реальная логика вычислений

}

[illegible]



made with mematic

JAKE-CLARK.TUMBLR

Минусы аппрувалов

Для детерминированных данные
Плюс обращать внимание на
локализацию

Немного медленнее ассертов
А также растет количество файлов
в репозитории

**Требуют внимательности при
начальной проверке**
Особенно при объемных
результатах

Стремление делать ими всё
Даже в самых простых случаях

Когда использовать Approvals

Сценарий	Approvals	«Обычные» тесты
Проверка части объекта	<input checked="" type="radio"/>	<input checked="" type="radio"/>
Проверка точных значений	<input checked="" type="radio"/>	<input checked="" type="radio"/>
Проверка больших объектов (JSON, таблицы, логи)	<input checked="" type="radio"/>	<input type="radio"/>
Детерминированные сложные данные	<input checked="" type="radio"/>	<input type="radio"/>
Динамические значения	<input type="radio"/>	<input checked="" type="radio"/>
Вхождения строк, StartsWith, нужные байты итд	<input type="radio"/>	<input checked="" type="radio"/>

Спасибо за внимание!

Константин Финагин. Доклад «Approval-тестирование в .NET»
<https://github.com/KonstantinFinagin/SpbDotNet/tree/master/CodeSample>