```
1  namespace Core.Tests;
2
3  using Xunit;
4  using Microsoft.VisualStudio.TestTools.UnitTesting;
5  using Core;
6  using Orders;
7
   0 references | Run All Tests | Debug All Tests
8  public class AddNewOrderTest
9  {
10     [TestMethod]
11     [Priority(3)]
12     [Owner("Karol Borecki")]
13     [TestCategory("AddNewOrder")]
14     [Description("AddNewOrder should add new order to the shop data")]
       0 references | Run Test | Debug Test
15     public void AddNewOrder_ForValidNewOrderData_ResultsInNewOrderBeingAdded()
16     {
17         // arrange
18         var shopData = new ShopData();
19         var newOrder = new Order("order id", "client id", new Product("product id", "product name", 10, 10), OrderState.New);
20
21         // act
22         shopData.AddNewOrder(newOrder);
23
24         // assert
25         var addedOrder = shopData.GetOrder("order id");
26         Assert.Equal(newOrder.GetID(), addedOrder.GetID());
27         Assert.Equal(newOrder.GetClientID(), addedOrder.GetClientID());
28         Assert.Equal(newOrder.GetProduct().GetID(), addedOrder.GetProduct().GetID());
29         Assert.Equal(OrderState.Approved, addedOrder.GetState());
30     }
31 }
```

1. Stworzenie testu jednostkowego

tests > Core.Tests > C# PlaceNewOrderTest.cs > PlaceNewOrderTest

```
namespace Core.Tests;

using Xunit;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Core;

0 references | Run All Tests | Debug All Tests
public class PlaceNewOrderTest
{
    [TestMethod]
    [Priority(3)]
    [Owner("Karol Borecki")]
    [TestCategory("PlaceNewOrder")]
    [Description("PlaceNewOrder should add new order to the shop data")]
    0 references | Run Test | Debug Test
    public void PlaceNewOrder_ForValidData_ResultsInOrderAdding()
    {
        // arrange
        var shopDataMock = new Mock<IShopData>();
        shopDataMock.Setup(x => x.GetProduct(It.IsAny<string>())).Returns(new Product("productID", "productName", 1, 1));
        shopDataMock.Setup(x => x.GetClient(It.IsAny<string>())).Returns(new Client("clientID", "clientName", "clientAddress"));
        var shop = new Shop(shopDataMock.Object);

        // act
        shop.PlaceNewOrder("clientID", "productID", 1);

        // assert
        shopDataMock.Verify(x => x.AddOrder(It.IsAny<IClient>(), It.IsAny<IProduct>(), It.IsAny<int>()), Times.Once);
        shop.Verify(x => x.UpdateProduct(It.IsAny<IProduct>(), It.IsAny<int>()), Times.Once);
        shop.GetOrders().Count.Should().Be(1);
    }

    [TestMethod]
    [Priority(2)]
    [Owner("Karol Borecki")]
    [TestCategory("PlaceNewOrder")]
    [Description("PlaceNewOrder should throw exception if product quantity is not enough")]
    [ExpectedException(typeof(ArgumentException))]
    0 references | Run Test | Debug Test
    public void PlaceNewOrder_ForFaultyProductID_ResultsInException()
    {
        // arrange
        var shopDataMock = new Mock<IShopData>();
        shopDataMock.Setup(x => x.GetProduct(It.IsAny<string>())).Returns(null);
        var shop = new Shop(shopDataMock.Object);

        // act
        shop.PlaceNewOrder("clientID", "productID", 1);

        // assert
    }

    [TestMethod]
    [Priority(2)]
    [Owner("Karol Borecki")]
    [TestCategory("PlaceNewOrder")]
    [Description("PlaceNewOrder should throw exception if product quantity is not enough")]
    [ExpectedException(typeof(ArgumentException))]
    0 references | Run Test | Debug Test
    public void PlaceNewOrder_ForFaultyClientID_ResultsInException()
    {
        // arrange
        var shopDataMock = new Mock<IShopData>();
        shopDataMock.Setup(x => x.GetClient(It.IsAny<string>())).Returns(null);
        var shop = new Shop(shopDataMock.Object);

        // act
        shop.PlaceNewOrder("clientID", "productID", 1);

        // assert
    }
}
```

6. Mokowanie obiektów

3. Testowanie sytuacji gdy rzucany jest błąd

11. Tworzenie scenariusza pozytywnego i negatywnego

```
1  namespace Core.Tests;
2
3  using Xunit;
4  using Microsoft.VisualStudio.TestTools.UnitTesting;
5  using Core;
6
   0 references | Run All Tests | Debug All Tests
7  public class UpdateOrderStatusTest
8  {
9      [TestMethod]
10     [Priority(3)]
11     [Owner("Karol Borecki")]
12     [TestCategory("UpdateOrderStatus")]
13     [Description("UpdateOrderStatus should update order in the shop data")]
14     [InlineData(OrderState.Pending)]
15     [InlineData(OrderState.Completed)]
16     [InlineData(OrderState.Cancelled)]
       0 references | Run Test | Debug Test
17     public void UpadteOrderStatus_ForValidOrerID_ResultsInUpdatingAnOrder(OrderState state)
18     {
19         // arrange
20         var shopDataMock = new Mock<IShopData>();
21         shopDataMock
22         .Setup(x => x.GetOrder(It.IsAny<string>()))
23         .Returns(
24             new Order("orderID", new Client("clientID", "clientName", "clientAddress"),
25             new Product("productID", "productName", 1, 1),
26             1,
27             OrderState.New));
28         shopDataMock
29         .Setup(x => x.UpdateOrder(It.IsAny<IOrder>(), It.IsAny<OrderState>()))
30         .Callback((IOrder order) => {
31             order.SetState(state);
32         }).Verifiable();;
33
34         var shop = new Shop(shopDataMock.Object);
35
36         // act
37         shop.UpdateOrderStatus("orderID", state);
38
39         // assert
40         shop.GetOrder("orderID").GetState().Should().Be(state);
41     }
42
43     [TestMethod]
44     [Priority(2)]
45     [Owner("Karol Borecki")]
46     [TestCategory("UpdateOrderStatus")]
47     [Description("UpdateOrderStatus to other than approved should throw exception if order is new")]
48     [ExpectedException(typeof(ArgumentException))]
49     [InlineData(OrderState.Pending)]
50     [InlineData(OrderState.Completed)]
51     [InlineData(OrderState.Cancelled)]
       0 references | Run Test | Debug Test
52     public void UpadteOrderStatus_ForNotAcceptedOrder_ToPendding_ResultsInException(OrderState state)
53     {
54         // arrange
55         var shopDataMock = new Mock<IShopData>();
56         shopDataMock
57         .Setup(x => x.GetOrder(It.IsAny<string>()))
58         .Returns(
59             new Order("orderID", new Client("clientID", "clientName", "clientAddress"),
60             new Product("productID", "productName", 1, 1),
61             1,
62             OrderState.New));
63
64         var shop = new Shop(shopDataMock.Object);
65
66         // act
67         shop.UpdateOrderStatus("orderID", state);
68
69         // assert
70     }
71 }
```

2. Prosta parametryzacja testów

7. Mokowanie metody nie zwracającej danych

5. Polepszenie czytelności testu

~/Desktop/MetodyWytwarzaniaOprogramowania/LAB2/tests • Contains emphasized items

tests > Core.Tests > C# UpdateOrderTest.cs > UpdateOrderStatusTest > UpadteOrder_ForGivenOrderData_ResultsInCorrectOrderUpdate

```
1  namespace Core.Tests;
2
3  using Xunit;
4  using Microsoft.VisualStudio.TestTools.UnitTesting;
5  using Core;
6
   0 references | Run All Tests | Debug All Tests
7  public class UpdateOrderStatusTest
8  {
       1 reference
9      public TestContext TestContext { get; set; }
10
       1 reference
11     public static IEnumerable<object[]> Orders
12     {
13         get
14         {
15             yield return new object[] { new Order("order1", "client1", new Product("product1", "product1", 10, 20), OrderState.New) };
16             yield return new object[] { new Order("order2", "client2", new Product("product2", "product2", 2, 54), OrderState.New) };
17         }
18     }
19
20     [TestMethod]
21     [Priority(3)]
22     [Owner("Karol Borecki")]
23     [TestCategory("UpdateOrder")]
24     [Description("UpdateOrder should update order in the shop data")]
25     [MemberData(nameof(Orders))]
       0 references | Run Test | Debug Test
26     public void UpadteOrder_ForGivenOrderData_ResultsInCorrectOrderUpdate(Order order)
27     {
28         // arrange
29         TestContext.WriteLine("Order: {0}", order.GetID());
30         var shopDataMock = new Mock<IShopData>();
31         shopDataMock.Setup(x => x.ContainsOrder(It.IsAny<string>())).Returns(true);
32
33         // act
34         shopDataMock.UpdateOrder(order);
35
36         // assert
37         shopDataMock.GetOrder("orderID").GetState().Should().Be(order.GetState());
38         shopDataMock.GetOrder("orderID").GetQuantity().Should().Be(order.GetQuantity());
39         shopDataMock.GetOrder("orderID").GetProduct().GetID().Should().Be(order.GetProduct().GetID());
40     }
41 }
```

8. Parametryzacja testów złożonymi obiektami

4. Testowanie złożonych obiektów

10. Logowanie podczas testowania

```
You, 3 days ago | 1 author (You)
1  using Xunit;
2  using Math.Services;
3
4  namespace Math.UnitTests.Services
5  {
       0 references | Run All Tests | Debug All Tests | You, 3 days ago | 1 author (You)
6      public class MathService_IsPrimeShould
7      {
8          [Fact]
           0 references | Run Test | Debug Test
9          public void IsPrime_InputIs1_ReturnFalse()
10         {
11             // arrange
12             var mathService = new MathService();
13
14             // act
15             bool result = mathService.IsPrime(1);
16
17             // assert
18             Assert.False(result, "1 should not be prime");
19         }
20
21         [Theory]
22         [InlineData(1, false)]
23         [InlineData(2, true)]
             0 references | Run Test | Debug Test
24         public void IsDivideableBy2_ForGivenNumber_ReturnsCorrect(int number, bool correctResult)
25         {
26             // arrange
27             var mathService = new MathService();
28
29             // act
30             bool result = mathService.IsDivideableBy2(number);
31
32             // assert
33             Assert.Equal(correctResult, result);
34         }
35     }
36 }      You, 3 days ago • Added files from lecture
```

Plik json do podpunktu 9

```json
{
    "order":
    [
        {
            "id": "order1",
            "clientID": "Order 1",
            "product": {
                "id": "product1",
                "name": "Product 1",
                "price": 100,
                "quantity": 5
            },
            "quantity": 2
        },
        {
            "id": "order2",
            "clientID": "Order 2",
            "product": {
                "id": "product2",
                "name": "Product 2",
                "price": 63,
                "quantity": 35
            },
            "quantity": 6
        },
        {
            "id": "order3",
            "clientID": "Order 3",
            "product": {
                "id": "product3",
                "name": "Product 3",
                "price": 100,
                "quantity": 6352
            },
            "quantity": 14
        }
    ]
}
```

```
1  namespace Orders.Tests;
2
3  using Xunit;
4  using Microsoft.VisualStudio.TestTools.UnitTesting;
5  using Orders;
6
   0 references | Run All Tests | Debug All Tests
7  public class ConstructorTest
8  {
       1 reference
9      private const string order_json_path = "Data/OrderData.json";
10
       1 reference
11     public static IEnumerable<object[]> ValidOrderTestData()
12     {
13         var filePath = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, order_json_path);
14         var json = File.ReadAllText(filePath);
15         var jobject = JObject.Parse(json);
16         var orders = jobject["order"]?.ToObject<IEnumerable<Order>>();
17
18         foreach (var order in orders ?? Enumerable.Empty<Order>())
19         {
20             yield return new[] { user };
21         }
22     }
23
24     [TestMethod]
25     [Priority(1)]
26     [Owner("Karol Borecki")]
27     [TestCategory("Order")]
28     [Description("Order should be constructed with valid data")]
29     [MemberData(nameof(ValidOrderTestData))]
       0 references | Run Test | Debug Test
30     public void ConstructingOrder_ForValidDataWithoutStateParameter_ResultsInNewOrderBeeingConstructed(Order order)
31     {
32         // arrange
33         string id = order.GetID();
34         string clientID = order.GetClientID();
35         IProduct product = order.GetProduct();
36
37         // act
38         var newOrder = new Order(id, clientID, product);
39
40         //assert
41         Assert.Equal(id, newOrder.GetID());
42         Assert.Equal(clientID, newOrder.GetClientID());
43         Assert.Equal(product.GetID(), newOrder.GetProduct().GetID());
44         Assert.Equal(state, OrderState.New);
45     }
46 }
```

9. Parametryzacja testu danymi z pliku

Kod dostępny na Github: https://github.com/KarolBorecki/MetodyWytwarzaniaOprogramowania