

Федеральное государственное автономное  
образовательное учреждение  
высшего образования  
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и информационных технологий

институт

Кафедра «Информатика»

кафедра

## ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ №5

Автоматизация функционального тестирования приложений с  
графическим интерфейсом пользователя

тема

Преподаватель

\_\_\_\_\_

подпись, дата

А. С. Кузнецов  
инициалы, фамилия

Студент КИ18-17/16 031830504

номер группы, зачетной книжки

\_\_\_\_\_

подпись, дата

Е.В. Железкин  
инициалы, фамилия

Красноярск 2021

## 1 Цель работы

Изучить методологию автоматизации функционального тестирования приложения с графическим пользовательским интерфейсом.

## 2 Задача работы

1. Научиться пользоваться фикстурами для автоматического заполнения компонентов Swing.
2. Научиться проводить функциональное тестирование при помощи тестовых заглушек.

### *Вариант 1*

Окно заказа пиццы

- a) текстовые поля:
  - a. Адрес доставки
  - b. Время доставки
- b) комбинированные списки
  - a. Название пиццы
  - b. Размер порции
  - c. Дополнительные напитки
  - d. Соусы

### 2.1 Инструкция по запуску

- Необходимо установить .Net Framework SDK:

Страница загрузки для Windows:

<https://dotnet.microsoft.com/download/dotnet-framework/net48>

- И Microsoft Build Tools 2019(выбрать из пакета установки Visual Studio):

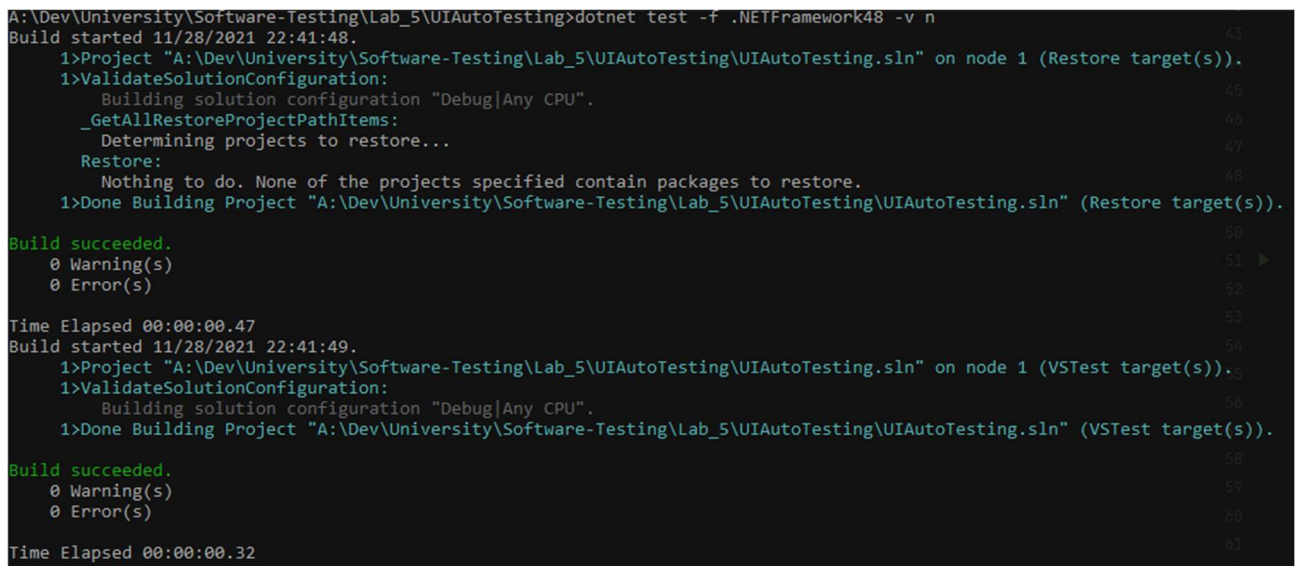
<https://visualstudio.microsoft.com/ru/downloads/>

Также необходимо установить пакет *TestStack.White* с помощью nuget.

Далее собрать и запустить проекты с помощью MSBuild (путь зависит от версии)

По каким-то неизвестным мне причинам стандартные средства запуска тестов не видят тесты при использовании .Net Framework:

*\$ dotnet test -f .NETFramework48 -v n*



```
A:\Dev\University\Software-Testing\Lab_5\UIAutoTesting>dotnet test -f .NETFramework48 -v n
Build started 11/28/2021 22:41:48.
1>Project "A:\Dev\University\Software-Testing\Lab_5\UIAutoTesting\UIAutoTesting.sln" on node 1 (Restore target(s)).
1>ValidateSolutionConfiguration:
    Building solution configuration "Debug|Any CPU".
    _GetAllRestoreProjectPathItems:
        Determining projects to restore...
    Restore:
        Nothing to do. None of the projects specified contain packages to restore.
1>Done Building Project "A:\Dev\University\Software-Testing\Lab_5\UIAutoTesting\UIAutoTesting.sln" (Restore target(s)).
Build succeeded.
    0 Warning(s)
    0 Error(s)
Time Elapsed 00:00:00.47
Build started 11/28/2021 22:41:49.
1>Project "A:\Dev\University\Software-Testing\Lab_5\UIAutoTesting\UIAutoTesting.sln" on node 1 (VSTest target(s)).
1>ValidateSolutionConfiguration:
    Building solution configuration "Debug|Any CPU".
1>Done Building Project "A:\Dev\University\Software-Testing\Lab_5\UIAutoTesting\UIAutoTesting.sln" (VSTest target(s)).
Build succeeded.
    0 Warning(s)
    0 Error(s)
Time Elapsed 00:00:00.32
```

Поэтому приложу ссылку на видео с работой программы:  
<https://disk.yandex.ru/i/YKpu-JfkmB8-bw>

### 3 Ход работы

В ходе данной практической работы было разработано WPF приложение в качестве UI для тестирования. (Рисунки 1-2)

PizzaDelivery

## Оформление заказа

Адрес:

Время:

Название пиццы:

Размер пиццы:

Напиток:

Соус:

```
Grid.Row="3" Text="Время:"  
" Text="{Binding Source=  
="300" Margin=" [lr] 0, [tb] 1
```

Рисунок 1 – Внешний вид приложения

```
1 <Window x:Class="UIAutoTesting.MainWindow"
2     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4     xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
5     xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
6     mc:Ignorable="d" FontSize="20"
7     Title="PizzaDelivery" Height="450" Width="525">
8     <Grid>
9         <Grid.ColumnDefinitions>
10             <ColumnDefinition Width="20" />
11             <ColumnDefinition Width="Auto" />
12             <ColumnDefinition Width="Auto" />
13             <ColumnDefinition Width="20" />
14         </Grid.ColumnDefinitions>
15         <Grid.RowDefinitions>
16             <RowDefinition Height="20" />
17             <RowDefinition Height="Auto" />
18             <RowDefinition Height="Auto" />
19             <RowDefinition Height="Auto" />
20             <RowDefinition Height="Auto" />
21             <RowDefinition Height="Auto" />
22             <RowDefinition Height="Auto" />
23             <RowDefinition Height="Auto" />
24             <RowDefinition Height="Auto" />
25             <RowDefinition Height="20" />
26         </Grid.RowDefinitions>
27
28         <TextBlock Grid.Column="1" Grid.Row="1" Grid.ColumnSpan="3" FontWeight="Bold" FontSize="30"
29             Text="Оформление заказа" />
30
31         <TextBlock Grid.Column="1" Grid.Row="2" Text="Адрес:" Margin=" [ln] 10, [tbc] 10" />
32         <TextBox x:Name="AddressTextBox" Text="{Binding Source={StaticResource ResourceKey= Order}, Path= (OrderStatic). Address}" Grid.Column="2"
33             Grid.Row="2" Width="300" Margin=" [ln] 0, [tbc] 10" />
34
35         <TextBlock Grid.Column="1" Grid.Row="3" Text="Время:" Margin=" [ln] 10, [tbc] 10" />
36         <TextBox x:Name="TimeTextBox" Text="{Binding Source={StaticResource ResourceKey= Order}, Path= (OrderStatic). Time}" Grid.Column="2"
37             Grid.Row="3" Width="300" Margin=" [ln] 0, [tbc] 10" />
38
39         <TextBlock Grid.Column="1" Grid.Row="4" Text="Название пиццы:" Margin=" [ln] 10, [tbc] 10" />
40         <ComboBox x:Name="PizzaNameComboBox" Text="{Binding Source={StaticResource ResourceKey= Order}, Path= (OrderStatic). PizzaName}"
41             Grid.Column="2" Grid.Row="4" Width="300" Margin=" [ln] 0, [tbc] 10" />
42
43         <TextBlock Grid.Column="1" Grid.Row="5" Text="Размер пиццы:" Margin=" [ln] 10, [tbc] 10" />
44         <ComboBox x:Name="PizzaSizeComboBox" Text="{Binding Source={StaticResource ResourceKey= Order}, Path= (OrderStatic). PizzaSize}"
45             Grid.Column="2" Grid.Row="5" Width="300" Margin=" [ln] 0, [tbc] 10" />
46
47         <TextBlock Grid.Column="1" Grid.Row="6" Text="Напиток:" Margin=" [ln] 10, [tbc] 10" />
48         <ComboBox x:Name="DrinksComboBox" Text="{Binding Source={StaticResource ResourceKey= Order}, Path= (OrderStatic). Drink}" Grid.Column="2"
49             Grid.Row="6" Width="300" Margin=" [ln] 0, [tbc] 10" />
50
51         <TextBlock Grid.Column="1" Grid.Row="7" Text="Соус:" Margin=" [ln] 10, [tbc] 10" />
52         <ComboBox x:Name="SaucesComboBox" Text="{Binding Source={StaticResource ResourceKey= Order}, Path= (OrderStatic). Sauce}" Grid.Column="2"
53             Grid.Row="7" Width="300" Margin=" [ln] 0, [tbc] 10" />
54
55         <Button x:Name="OrderSubmitButton" Grid.Column="1" Grid.Row="8" Grid.ColumnSpan="2" Content="Отправить" />
56     </Grid>
57 </Window>
```

Рисунок 2 – Файл разметки моего приложения

Далее представлена структура решения, состоящая из двух проектов. (Рисунок 3) Проект UIAutoTesting содержит данные для формы(зелёный), классы разметки и логики приложения(красный), сервис для обработки данных, полученных из формы,(жёлтый) и автоматизированные тесты(синий).

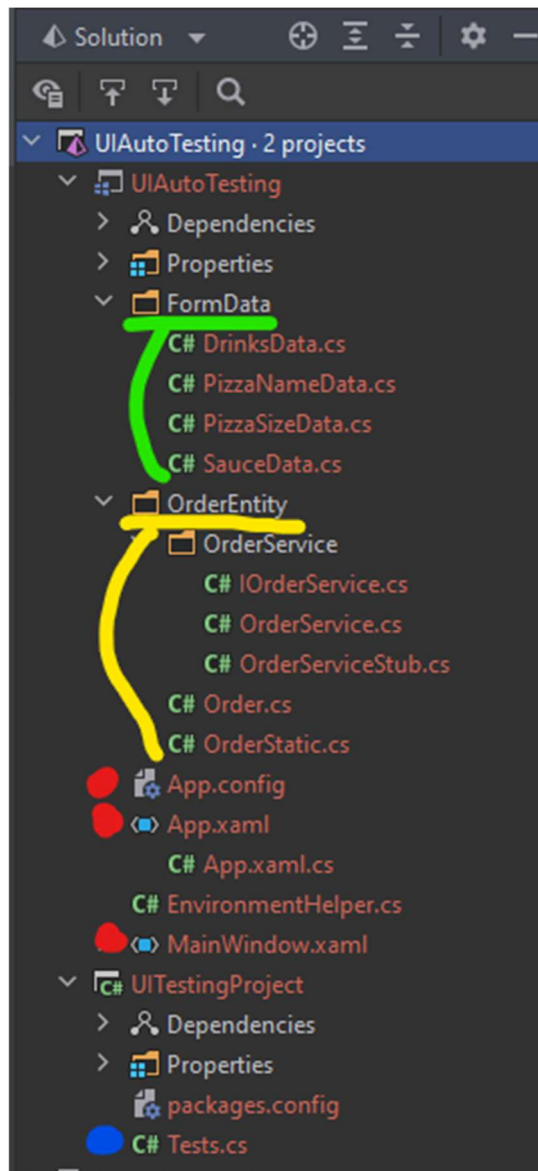


Рисунок 3 – Структура решения

Так же в проекте есть класс *EnvironmentHelper*. Проблема в том, что я не нашёл библиотеки для dotnet приложения, которая бы полностью имитировала его работу в фоновом режиме. Вместо этого использовался пакет *TestStack.White*, которые имитирует работу приложения, путём полного функционального повторения всех действий. Так как приложение запускается в штатном режиме и отдельном потоке (Рисунок 4), то сделать заглушку становится немного проблематично. Было принято решение использовать пародию(ручной вариант) на внедрение зависимостей. Для запуска режима тестирования (подстановки

заглушки вместо оригинального сервиса) приложение запускается с аргументом *Testing*. (Рисунки 5-6)

```
[TestFixture]
public class Tests
{
    private readonly string _applicationPath =
        Path.Combine(TestContext.CurrentContext.TestDirectory.Replace( oldValue: "UITestingProject", newValue: "UIAutoTesting"),
            "UIAutoTesting.exe");

    private Window _window;
    private Application _application;
    private UdpClient _client;
    private IPEndPoint _ip = null;

    /// <summary>
    /// Метод, инициализирующий приложение в начале каждого теста
    /// </summary>
    [SetUp]
    public void Init()
    {
        _application = Application.Launch(new ProcessStartInfo(fileName: _applicationPath, arguments: "Testing"));
        _window = _application.GetWindow(title: "PizzaDelivery", InitializeOption.NoCache);

        _client = new UdpClient(port: 54321);
    }

    /// <summary>
    /// Метод, завершающий приложение
    /// </summary>
    [TearDown]
    public void Cleanup()
    {
        _application.Close();
        _client.Close();
    }
}
```

Рисунок 4 – Конфигурация тестов

```
public static class EnvironmentHelper
{
    private static bool _isTesting;

    public static IOrderService GetOrderService => _isTesting ? new OrderServiceStub() : new OrderService();

    public static void SetMode()
    {
        foreach (var arg:string in Environment.GetCommandLineArgs())
        {
            if (arg.Equals("Testing"))
            {
                _isTesting = true;
            }
        }
    }
}
```

Рисунок 5 – Класс *Environment Helper*

```

public partial class MainWindow
{
    public MainWindow()
    {
        EnvironmentHelper.SetMode();

        InitializeComponent();

        PizzaNameComboBox.ItemsSource = PizzaNameData.GetNames;
        PizzaSizeComboBox.ItemsSource = PizzaSizeData.GetSizes;
        DrinksComboBox.ItemsSource = DrinksData.GetDrinks;
        SaucesComboBox.ItemsSource = SauceData.GetSauces;

        IOrderService service = EnvironmentHelper.GetOrderService;
        OrderSubmitButton.Click += service.UpdateOrder;
    }
}

```

Рисунок 6 – В зависимости от наличия аргумента возвращается нужный сервис

Пример работы приложения. (Рисунок 7)

The screenshot shows the 'PizzaDelivery' application window. The main window has a title bar with standard Windows controls. The content area is titled 'Оформление заказа' (Order Form). It contains several input fields and dropdown menus: 'Адрес:' (Address) with 'Street', 'Время:' (Time) with '11:15', 'Название пиццы:' (Pizza Name) with 'Сибирская', 'Размер пиццы:' (Pizza Size) with 'средняя', 'Напиток:' (Drink) with 'Чай', and 'Соус:' (Sauce) with 'Чесночный'. At the bottom is a button labeled 'Отправить' (Send). To the right, an 'Info' dialog box is open, displaying the order summary: 'Ваш заказ на Street(11:15): Пицца средняя Сибирская; Напиток: Чай; Соус: Чесночный'. The dialog has an 'OK' button.

Рисунок 7 – Пример работы



Теперь рассмотрим подробнее отличия в сервисах. Оригинальный сервис (рисунок 8) по задумке должен имитировать отправку данных на сервер.

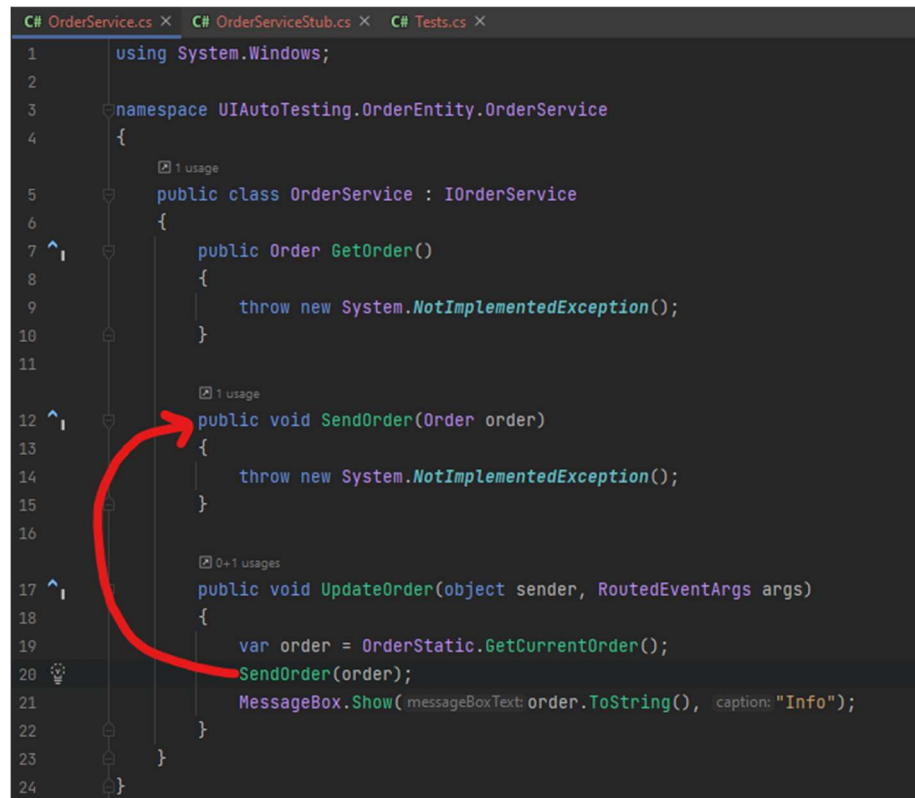


Рисунок 8 – Оригинальный сервис

Заглушка же вместо этого посылает из тестируемого приложения сигнал тестирующему, о завершении ввода данных.




```
C# OrderService.cs × C# OrderServiceStub.cs × C# Tests.cs ×
1  using System.Net.Sockets;
2  using System.Windows;
3
4  namespace UIAutoTesting.OrderEntity.OrderService
5  {
6       1 usage
7      public class OrderServiceStub : IOrderService
8      {
9          public Order GetOrder()
10         {
11             throw new System.NotImplementedException();
12         }
13
14         public void SendOrder(Order order)
15         {
16             throw new System.NotImplementedException();
17         }
18
19          0+1 usages
20         public void UpdateOrder(object sender, RoutedEventArgs args)
21         {
22             var order = OrderStatic.GetCurrentOrder();
23             MessageBox.Show(messageBoxText: order.ToString(), caption: "Info");
24             SendFree();
25         }
26
27          1 usage
28         private static void SendFree()
29         {
30             UdpClient client = new UdpClient();
31             byte[] data = { 1 };
32             client.Send(data, bytes: data.Length, hostname: "127.0.0.1", port: 54321);
33         }
34     }
35 }
```

Рисунок 9 – Сервис-заглушка

Так как тестируемое и тестирующее приложение никак не связаны (по крайней мере в документации к библиотеке *TestStack.White* я не нашёл такой возможности), то необходимо вручную сообщить о завершении ввода в тестируемое приложение. Форма отправляется по завершению ввода, то, при обработке клика по кнопке отправки (Рисунок 9), отправляется UDP-пакет на

порт 54321 с 1 байтом информации, символизирующий о завершении ввода. Тестирующее приложение ждёт ответ, пока не сработает и не выполнится обработчик кнопки отправления данных с рисунка 9. (Рисунок 10)

```
var submitButton = _window.Get<Button>(primaryIdentification: "OrderSubmitButton");
submitButton.Click();|

byte[] data = _client.Receive(ref _ip);
Assert.AreEqual(expected: new byte[] {1}, actual: data);

Window childWindow = _window.MessageBox("Info");
Assert.AreNotEqual(expected: childWindow, actual: null);
```

Рисунок 10 – Ожидание разблокировки

На рисунке 9 приведён сам метод автоматизированного функционального тестирования приложения, где создаётся имитация ввода данных для всех элементов управления на странице.

```
[Test]
public void TestUi()
{
    var addressTextBox = _window.Get<TextBox>(primaryIdentification: "AddressTextBox");
    addressTextBox.BulkText = "Main street";

    var timeTextBox = _window.Get<TextBox>(primaryIdentification: "TimeTextBox");
    timeTextBox.BulkText = "21:00";

    var pizzaNameComboBox = _window.Get<ComboBox>(primaryIdentification: "PizzaNameComboBox");
    pizzaNameComboBox.Select(index: 3);

    var pizzaSizeComboBox = _window.Get<ComboBox>(primaryIdentification: "PizzaSizeComboBox");
    pizzaSizeComboBox.Select(index: 2);

    var drinksComboBox = _window.Get<ComboBox>(primaryIdentification: "DrinksComboBox");
    drinksComboBox.Select(index: 1);

    var saucesComboBox = _window.Get<ComboBox>(primaryIdentification: "SaucesComboBox");
    saucesComboBox.Select(index: 0);

    var submitButton = _window.Get<Button>(primaryIdentification: "OrderSubmitButton");
    submitButton.Click();

    byte[] data = _client.Receive(ref _ip);
    Assert.AreEqual(expected: new byte[] {1}, actual: data);

    Window childWindow = _window.MessageBox("Info");
    Assert.AreNotEqual(expected: childWindow, actual: null);

    var messageBoxLabel = childWindow.Get<Label>();
    Assert.AreEqual(expected: messageBoxLabel.Text,
        actual: "Ваш заказ на Main street(21:00): Пицца большая Мексиканская; Напиток: Газировка; Соус: Майонезный");

    Thread.Sleep(millisecondsTimeout: 2000);
}
```

Рисунок 9 – Метод тестирования UI приложения

Ссылка на видео с тестированием: <https://disk.yandex.ru/i/YKpu-JfkmB8-bw>

#### **4 Вывод**

В ходе данной лабораторной работы была изучена методология автоматизации функционального тестирования приложения с графическим пользовательским интерфейсом.