Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Факультет Программной инженерии и компьютерной техники

**Лабораторная работа №4**

по дисциплине

«Тестирование программного обеспечения»

**Выполнил**

Ореховский А.,

группа P3317

**Преподаватель**

Исаев И. В.

Санкт-Петербург

2020

Оглавление

[Цель 3](#_Toc54025556)

[Задачи 3](#_Toc54025557)

[Описание TDD 3](#_Toc54025558)

[Требования 3](#_Toc54025559)

[Листинг кода 4](#_Toc54025560)

[Services 4](#_Toc54025561)

[Layouts 6](#_Toc54025562)

[Models 14](#_Toc54025563)

[Commons 16](#_Toc54025564)

[Unit tests 17](#_Toc54025565)

[UI Tests 20](#_Toc54025566)

[Выводы 31](#_Toc54025567)

# Цель

Изучить метод разработки через тестирование (TDD).

# Задачи

* Сформировать требования к приложению (не менее 20);
* Необходимо полностью покрыть тестами данные требования используя разработку через тестирование;
* Реализовать продукт 100% покрытый тестами.

# Описание TDD

TDD — это техника разработки программного обеспечения, которая основывается на повторении очень коротких циклов разработки: сначала пишется тест, покрывающий желаемое изменение, затем пишется код, который позволит пройти тест, и под конец проводится рефакторинг нового кода к соответствующим стандартам.

Смысл написания сначала теста и только затем кода состоит в уменьшении вероятности появления ошибки на более поздних этапах разработки. Придерживаясь TDD, разработчики могут представить архитектуру и дизайн приложения гораздо раньше, в сравнении со стандартным подходом.

Написание тестов до кода также помогает «протестировать тесты». Другими словами, это помогает исключить появление ложноположительных тестов, ведь мы не хотим успешное выполнение теста, который покрывает еще не существующий код.

# Требования

1. Приложение должно поддерживать регистрацию при помощи логина и пароля
2. Приложение должно поддерживать вход пользователя при помощи логина и пароля
3. Пользователь не должен иметь возможность зарегистрироваться при помощи логина, который уже зарезервирован
4. Пароль должен храниться в хэшированном виде
5. Длина логина не должна превышать 15 символов или быть меньше 6 символов
6. Длина пароля не должна превышать 15 символов или быть меньше 6 символов
7. При регистрации необходимо указывать имя и фамилию пользователя
8. Длина имени пользователя не должна превышать 30 символов или быть нулевой
9. Длина фамилии пользователя не должна превышать 30 символов или быть нулевой
10. Имя и Фамилия должны соответствовать регулярному выражению вида

/^[a-zA-Zа-яА-Я]+(([,. -][a-zA-Zа-яА-Я ])?[a-zA-Zа-яА-Я]\*)\*$/

1. Пользователь должен иметь возможность отправлять сообщения
2. Длина сообщения пользователя не должна превышать 256 символов или быть нулевой
3. При отправке сообщения необходимо сохранять его с текущим временем на сервере.
4. Каждый пользователь должен иметь доступ ко всем сообщениям
5. Каждое сообщение должно иметь формат:
   1. Логин отправителя
   2. Время отправления
   3. Содержание сообщения
6. Пользователь не должен иметь возможность отправить сообщение, содержание которого совпадает с предыдущим сообщением этого же пользователя
7. После отправки сообщения пользователем необходим обновить список сообщений у данного пользователя
8. Список сообщений должен обновляться автоматически при отправке сообщения другим пользователем
9. Пользователь должен иметь возможность выйти из чата

# Листинг кода

## Services

*DataBaseService.cs*

public static class DataBaseService

{

private static ChatContext db = new ChatContext();

private static readonly Regex regex = new Regex("[a-zA-Zа-яА-Я]+(([,. -][a-zA-Zа-яА-Я ])?[a-zA-Zа-яА-Я]\*)\*");

public static Result AddUser(string login, string password, string firstname, string lastname)

{

if (login.Length > 15) return new Result(false, Errors.LongLoginError());

if (login.Length < 6) return new Result(false, Errors.ShortLoginError());

if (!regex.IsMatch(login)) return new Result(false, Errors.InvalidLoginError());

if (password.Length > 15) return new Result(false, Errors.LongPasswordError());

if (password.Length < 6) return new Result(false, Errors.ShortPasswordError());

if (firstname.Length > 30) return new Result(false, Errors.LongFirstnameError());

if (firstname.Length == 0) return new Result(false, Errors.EmptyFirstnameError());

if (!regex.IsMatch(firstname)) return new Result(false, Errors.InvalidFirstnameError());

if (lastname.Length > 30) return new Result(false, Errors.LongLastnameError());

if (lastname.Length == 0) return new Result(false, Errors.EmptyLastnameError());

if (!regex.IsMatch(lastname)) return new Result(false, Errors.InvalidLastnameError());

if (db.User.Any(u => u.Login == login)) return new Result(false, Errors.DuplicateLoginError());

try

{

var user = new User(login, password, firstname, lastname);

db.User.Add(user);

db.SaveChanges();

var id = db.User.FirstOrDefault(u => u.Login == login).Id;

return new Result(true, id);

}

catch (Exception e)

{

var result = new Result(false, e.Message);

return result;

}

}

public static Result DeleteUserById(int id)

{

try

{

db.Remove(db.User.Find(id));

db.SaveChanges();

return new Result(true);

}

catch (Exception e)

{

return new Result(false, e.Message);

}

}

public static Result GetUser(string login, string password)

{

var user = db.User.FirstOrDefault(u => u.Login == login && u.Password == Hasher.GetHash(password));

if (user == default)

{

return new Result(false, Errors.AuthError());

}

else

{

return new Result(true, user);

}

}

public static Result UserExists(string login)

{

var user = db.User.FirstOrDefault(u => u.Login == login);

if (user == default)

{

return new Result(false);

}

else

{

return new Result(true, user.Id);

}

}

public static Result AddMessage(string content, string login)

{

if (!content.Any()) return new Result(false, Errors.EmptyMessageError());

if (content.Length > 256) return new Result(false, Errors.LongMessageError());

var previousMessage = db.Messages.Where(m => m.Login == login).ToList().LastOrDefault();

if (previousMessage != default && previousMessage.Content == content) return new Result(false, Errors.DuplicateMessageError());

try

{

var message = new Messages(content, login);

db.Messages.Add(message);

db.SaveChanges();

var sentMessage = db.Messages.Where(m => m.Login == login).ToList().Last().Id;

if (sentMessage == default)

{

return new Result(true, "Couldn't get an id");

}

else

{

return new Result(true, sentMessage);

}

}

catch (Exception e)

{

return new Result(false, e.Message);

}

}

public static Result DeleteMessage(int id)

{

try

{

db.Remove(db.Messages.Find(id));

db.SaveChanges();

return new Result(true);

}

catch (Exception e)

{

return new Result(false, e.Message);

}

}

public static Result GetAllMessages()

{

try

{

var massages = db.Messages.ToList();

return new Result(true, massages);

}

catch (Exception e)

{

return new Result(false, e.Message);

}

}

}

*UserService.cs*

namespace ChatApp.Services

{

public static class UserService

{

private static string \_login;

public static string GetLogin() => \_login;

public static void SetLogin(string login) => \_login = login;

}

}

## Layouts

*MainWindow.xaml*

<Window x:Class="ChatApp.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:ChatApp"

mc:Ignorable="d"

Title="MainWindow" Height="450" Width="800" ResizeMode="NoResize">

<Grid>

<Grid.RowDefinitions>

<RowDefinition></RowDefinition>

<RowDefinition Height="50"></RowDefinition>

<RowDefinition Height="50"></RowDefinition>

<RowDefinition></RowDefinition>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition></ColumnDefinition>

<ColumnDefinition></ColumnDefinition>

</Grid.ColumnDefinitions>

<TextBlock

AutomationProperties.AutomationId="MainWindowTitle"

Grid.Row="0"

Grid.Column="0"

FontSize="40"

HorizontalAlignment="Center"

VerticalAlignment="Bottom"

Grid.ColumnSpan="2"

Text="Log in"/>

<StackPanel

Grid.Row="1"

Grid.Column="0"

Grid.ColumnSpan="2"

Orientation="Horizontal"

HorizontalAlignment="Center" >

<TextBlock

Width="150"

FontSize="30"

Text="Login"/>

<TextBox

x:Name="Login"

AutomationProperties.AutomationId="MainWindowLoginTextBox"

FontSize="30"

Height="40"

Width="400"

VerticalAlignment="Center"/>

</StackPanel>

<StackPanel

Grid.Row="2"

Grid.Column="0"

Grid.ColumnSpan="2"

Orientation="Horizontal"

HorizontalAlignment="Center" >

<TextBlock

Width="150"

FontSize="30"

Text="Password"/>

<PasswordBox

x:Name="Password"

AutomationProperties.AutomationId="MainWindowPasswordPasswordBox"

FontSize="30"

Height="40"

Width="400"

VerticalAlignment="Center"/>

</StackPanel>

<Button

AutomationProperties.AutomationId="MainWindowLoginButton"

Grid.Row="3"

Grid.Column="1"

Width="150"

Height=" 50"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="90 20"

FontSize="30"

Content="login"

Click="LogIn">

</Button>

<Button

AutomationProperties.AutomationId="MainWindowRegisterButton"

Grid.Row="3"

Grid.Column="0"

Width="150"

Height=" 50"

VerticalAlignment="Top"

HorizontalAlignment="Right"

Margin="0 20 90 0"

FontSize="30"

Content="register"

Click="SwitchToRegister">

</Button>

</Grid>

</Window>

*MainWindow.xaml.cs*

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

WindowStartupLocation = System.Windows.WindowStartupLocation.CenterScreen;

}

private void LogIn(object sender, RoutedEventArgs e)

{

var login = Login.Text;

var password = Password.Password;

var loginResult = DataBaseService.GetUser(login, password);

if (loginResult.Success)

{

SwitchToChat(login);

}

else

{

var text = (string)loginResult.Value;

var caption = "Login fault";

var button = MessageBoxButton.OK;

MessageBox.Show(text, caption, button);

}

}

private void SwitchToRegister(object sender, RoutedEventArgs e)

{

var register = new Register();

register.Show();

this.Close();

}

private void SwitchToChat(string login)

{

UserService.SetLogin(login);

var chat = new Chat();

chat.Show();

this.Close();

}

}

*Register.xaml.cs*

<Window x:Class="ChatApp.Register"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:ChatApp"

mc:Ignorable="d"

Title="Register" Height="450" Width="800" ResizeMode="NoResize">

<Grid>

<Grid.RowDefinitions>

<RowDefinition></RowDefinition>

<RowDefinition Height="50"></RowDefinition>

<RowDefinition Height="50"></RowDefinition>

<RowDefinition Height="50"></RowDefinition>

<RowDefinition Height="50"></RowDefinition>

<RowDefinition></RowDefinition>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition></ColumnDefinition>

<ColumnDefinition></ColumnDefinition>

</Grid.ColumnDefinitions>

<!--Title-->

<TextBlock

AutomationProperties.AutomationId="RegisterWindowTitle"

Grid.Row="0"

Grid.Column="0"

FontSize="40"

HorizontalAlignment="Center"

VerticalAlignment="Bottom"

Grid.ColumnSpan="2"

Text="Register"/>

<!--Login-->

<StackPanel

Grid.Row="1"

Grid.Column="0"

Grid.ColumnSpan="2"

Orientation="Horizontal"

HorizontalAlignment="Center" >

<TextBlock

Width="150"

FontSize="30"

Text="Login"/>

<TextBox

x:Name="Login"

AutomationProperties.AutomationId="RegisterWindowLoginTextBox"

FontSize="30"

Height="40"

Width="400"

VerticalAlignment="Center"/>

</StackPanel>

<!--Password-->

<StackPanel

Grid.Row="2"

Grid.Column="0"

Grid.ColumnSpan="2"

Orientation="Horizontal"

HorizontalAlignment="Center" >

<TextBlock

Width="150"

FontSize="30"

Text="Password"/>

<PasswordBox

x:Name="Password"

AutomationProperties.AutomationId="RegisterWindowPasswordPasswordBox"

FontSize="30"

Height="40"

Width="400"

VerticalAlignment="Center"/>

</StackPanel>

<!--Firstname-->

<StackPanel

Grid.Row="3"

Grid.Column="0"

Grid.ColumnSpan="2"

Orientation="Horizontal"

HorizontalAlignment="Center" >

<TextBlock

Width="150"

FontSize="30"

Text="Firstname"/>

<TextBox

x:Name="Firstname"

AutomationProperties.AutomationId="RegisterWindowFirstnameTextBox"

FontSize="30"

Height="40"

Width="400"

VerticalAlignment="Center"/>

</StackPanel>

<!--Lastname-->

<StackPanel

Grid.Row="4"

Grid.Column="0"

Grid.ColumnSpan="2"

Orientation="Horizontal"

HorizontalAlignment="Center" >

<TextBlock

Width="150"

FontSize="30"

Text="Lastname"/>

<TextBox

x:Name="Lastname"

AutomationProperties.AutomationId="RegisterWindowLastnameTextBox"

FontSize="30"

Height="40"

Width="400"

VerticalAlignment="Center"/>

</StackPanel>

<Button

AutomationProperties.AutomationId="RegisterWindowRegisterButton"

Grid.Row="5"

Grid.Column="1"

Width="150"

Height=" 50"

VerticalAlignment="Top"

HorizontalAlignment="Left"

Margin="90 20"

FontSize="30"

Content="register"

Click="Signup">

</Button>

<Button

AutomationProperties.AutomationId="RegisterWindowLoginButton"

Grid.Row="5"

Grid.Column="0"

Width="150"

Height=" 50"

VerticalAlignment="Top"

HorizontalAlignment="Right"

Margin="0 20 90 0"

FontSize="30"

Content="login"

Click="SwitchToLogin">

</Button>

</Grid>

</Window>

*Chat.xaml*

Window x:Class="ChatApp.Chat"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:ChatApp"

mc:Ignorable="d"

Title="Chat" Height="450" Width="800" ResizeMode="NoResize">

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="35"></RowDefinition>

<RowDefinition></RowDefinition>

<RowDefinition Height="70"></RowDefinition>

</Grid.RowDefinitions>

<Button

AutomationProperties.AutomationId="ChatWindowRefreshButton"

Grid.Row="0"

HorizontalAlignment="Left"

Width="50"

Margin="5"

Content="Refresh"

Click="Refresh">

</Button>

<!--Login info-->

<StackPanel

Grid.Row="0"

Orientation="Horizontal"

HorizontalAlignment="Right">

<TextBlock

FontSize="20"

Text="You are logged in as"/>

<TextBlock

x:Name="CurrentLogin"

AutomationProperties.AutomationId="ChatWindowCurrentLoginTextBlock"

FontSize="20"

Margin="10 0"

Foreground="DodgerBlue"/>

</StackPanel>

<ScrollViewer

x:Name="ScrollViewer"

AutomationProperties.AutomationId="Wrapper"

Grid.Row="1"

CanContentScroll="True">

<ListBox

x:Name="Messages"

AutomationProperties.AutomationId="ChatWindowMessagesListBox">

<ListBox.ItemTemplate>

<DataTemplate>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="2\*"></ColumnDefinition>

<ColumnDefinition Width="10\*"></ColumnDefinition>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition></RowDefinition>

<RowDefinition></RowDefinition>

</Grid.RowDefinitions>

<TextBlock

Grid.Row="0"

Grid.Column="0"

FontSize="10"

VerticalAlignment="Bottom"

Text="{Binding Path=Login}"/>

<TextBlock

Grid.Row="1"

Grid.Column="0"

Foreground="Gray"

FontSize="10"

Text="{Binding Path=TimeSend}"/>

<TextBlock

Grid.Row="0"

Grid.Column="1"

Grid.RowSpan="2"

FontSize="25"

VerticalAlignment="Top"

Margin="5"

Text="{Binding Path=Content}"/>

</Grid>

</DataTemplate>

</ListBox.ItemTemplate>

</ListBox>

</ScrollViewer>

<!--Messages-->

<Grid Grid.Row="2">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="10\*"></ColumnDefinition>

<ColumnDefinition Width="2\*"></ColumnDefinition>

</Grid.ColumnDefinitions>

<TextBox

AutomationProperties.AutomationId="ChatWindowMessageTextBox"

x:Name="Message"

Grid.Column="0"

FontSize="15"/>

<Button

AutomationProperties.AutomationId="ChatWindowSendMessageButton"

Grid.Column="1"

FontSize="30"

Content="Send"

Click="SendMessage"/>

</Grid>

</Grid>

</Window>

*Chat.xaml.cs*

namespace ChatApp

{

/// <summary>

/// Логика взаимодействия для Chat.xaml

/// </summary>

public partial class Chat : Window

{

private ObservableCollection<Messages> \_messages;

public Chat()

{

InitializeComponent();

WindowStartupLocation = System.Windows.WindowStartupLocation.CenterScreen;

CurrentLogin.Text = UserService.GetLogin();

ScrollViewer.ScrollToEnd();

\_messages = new ObservableCollection<Messages>();

Messages.ItemsSource = \_messages;

UpdateMessages();

}

private void SendMessage(object sender, RoutedEventArgs e)

{

var content = Message.Text;

var login = UserService.GetLogin();

var sendMessageResult = DataBaseService.AddMessage(content, login);

if (sendMessageResult.Success)

{

Message.Text = "";

UpdateMessages();

}

else

{

var text = (string) sendMessageResult.Value;

var caption = "Send message fault";

var button = MessageBoxButton.OK;

MessageBox.Show(text, caption, button);

}

}

private void Refresh(object sender, RoutedEventArgs e)

{

UpdateMessages();

}

private void UpdateMessages()

{

\_messages.Clear();

var messagesList = (List<Messages>)DataBaseService.GetAllMessages().Value;

foreach (var message in messagesList)

{

\_messages.Add(message);

}

}

}

}

## Models

*Messages.cs*

public partial class Messages

{

public Messages(string content, string login)

{

Login = login;

Content = content;

TimeSend = DateTime.Now;

}

public int Id { get; set; }

public string Login { get; set; }

public string Content { get; set; }

public DateTime TimeSend { get; set; }

}

*User.cs*

public partial class User

{

public User(string login, string password, string firstname, string lastname)

{

Login = login;

Password = Hasher.GetHash(password);

Firstname = firstname;

Lastname = lastname;

}

public int Id { get; set; }

public string Login { get; set; }

public string Password { get; set; }

public string Firstname { get; set; }

public string Lastname { get; set; }

}

*chatContext.cs*

public partial class ChatContext : DbContext

{

public ChatContext()

{

}

public ChatContext(DbContextOptions<ChatContext> options)

: base(options)

{

}

public virtual DbSet<Messages> Messages { get; set; }

public virtual DbSet<User> User { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

if (!optionsBuilder.IsConfigured)

{

optionsBuilder.UseSqlServer("Server=(LocalDb)\\MSSQLLocalDB;Database=chat;Trusted\_Connection=True;");

}

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Messages>(entity =>

{

entity.Property(e => e.Id).HasColumnName("id");

entity.Property(e => e.Content)

.IsRequired()

.HasColumnName("content")

.HasColumnType("text");

entity.Property(e => e.Login)

.IsRequired()

.HasColumnName("login")

.HasMaxLength(15);

entity.Property(e => e.TimeSend)

.HasColumnName("timeSend")

.HasColumnType("date");

});

modelBuilder.Entity<User>(entity =>

{

entity.HasIndex(e => e.Login)

.HasName("UQ\_\_User\_\_7838F2722050921F")

.IsUnique();

entity.Property(e => e.Id).HasColumnName("id");

entity.Property(e => e.Firstname)

.IsRequired()

.HasColumnName("firstname")

.HasMaxLength(30);

entity.Property(e => e.Lastname)

.IsRequired()

.HasColumnName("lastname")

.HasMaxLength(30);

entity.Property(e => e.Login)

.IsRequired()

.HasColumnName("login")

.HasMaxLength(15);

entity.Property(e => e.Password)

.IsRequired()

.HasColumnName("password")

.HasMaxLength(32);

});

OnModelCreatingPartial(modelBuilder);

}

partial void OnModelCreatingPartial(ModelBuilder modelBuilder);

## Commons

*Errors.cs*

public struct Errors

{

// Login errors

public static string DuplicateLoginError() => "User with this login already exists.";

public static string LongLoginError() => "Login length can not be longer than 15 characters.";

public static string ShortLoginError() => "Login length can not be shorter than 6 characters.";

public static string InvalidLoginError() => "Login does not match the pattern.";

// Password errors

public static string LongPasswordError() => "Password length can not be longer than 15 characters.";

public static string ShortPasswordError() => "Password length can not be shorter than 6 characters.";

//Name errors

public static string LongFirstnameError() => "Firstname length can not be longer than 30 characters.";

public static string EmptyFirstnameError() => "Firstname can not be empty.";

public static string InvalidFirstnameError() => "Firstname does not match the pattern.";

public static string LongLastnameError() => "Lastname length can not be longer than 30 characters.";

public static string EmptyLastnameError() => "Lastname can not be empty.";

public static string InvalidLastnameError() => "Login does not match the pattern.";

// Auth errors

public static string AuthError() => "Authorization failed. Please check the login and password.";

//Chat errors

public static string LongMessageError() => "Message length can not be greater than 256 characters.";

public static string EmptyMessageError() => "Message can not be empty.";

public static string DuplicateMessageError() => "Message should be different from your previous message.";

}

*Hasher.cs*

public class Hasher

{

public static string GetHash(string input)

{

using (MD5 md5Hash = MD5.Create())

{

byte[] data = md5Hash.ComputeHash(Encoding.UTF8.GetBytes(input));

StringBuilder sBuilder = new StringBuilder();

for (int i = 0; i < data.Length; i++)

{

sBuilder.Append(data[i].ToString("x2"));

}

return sBuilder.ToString();

}

}

public static bool VerifyHash(string input, string hash)

{

string hashOfInput = GetHash(input);

StringComparer comparer = StringComparer.OrdinalIgnoreCase;

if (0 == comparer.Compare(hashOfInput, hash))

{

return true;

}

else

{

return false;

}

}

}

*Result.cs*

public struct Result

{

public bool Success { get; }

public object Value { get; }

public Result(bool success)

{

Success = success;

Value = null;

}

public Result(bool success, object value)

{

Success = success;

Value = value;

}

}

## Unit tests

public static class DataBaseServiceTests

{

private const string Login = "loginTest";

private const string Password = "passwordTest";

private const string Firstname = "test";

private const string Lastname = "test";

private const string Content = "test";

[SetUp]

public static void Setup()

{

var userExists = UserExists(Login);

if (!userExists.Success) return;

var userId = (int)userExists.Value;

DeleteUserById(userId);

}

#region Auth Part

[Test]

public static void ShouldAddUser()

{

var result = AddUser(Login, Password, Firstname, Lastname);

Assert.IsTrue(result.Success);

}

[Test]

public static void ShouldKeepPasswordHashed()

{

GetUser(Login, Password);

var hashedPassword = Hasher.GetHash(Password);

Assert.IsTrue(Hasher.VerifyHash(Password, hashedPassword));

}

[Test]

public static void ShouldNotAddUserWithDuplicateLogin()

{

var result = AddUser(Login, Password, Firstname, Lastname);

Assert.IsTrue(result.Success);

result = AddUser(Login, Password, Firstname, Lastname);

Assert.IsFalse(result.Success);

Assert.AreEqual(Errors.DuplicateLoginError(), (string)result.Value);

}

[Test]

public static void ShouldNotAddUserWithLongLogin()

{

var login = new string('a', 16);

var result = AddUser(login, Password, Firstname, Lastname);

Assert.IsFalse(result.Success);

Assert.AreEqual(Errors.LongLoginError(), (string)result.Value);

}

[Test]

public static void ShouldNotAddUserWithShortLogin()

{

var login = new string('a', 5);

var result = AddUser(login, Password, Firstname, Lastname);

Assert.IsFalse(result.Success);

Assert.AreEqual(Errors.ShortLoginError(), (string)result.Value);

}

[Test]

public static void ShouldNotAddUserWithInvalidLogin()

{

var login = "123456";

var result = AddUser(login, Password, Firstname, Lastname);

Assert.IsFalse(result.Success);

Assert.AreEqual(Errors.InvalidLoginError(), (string)result.Value);

}

[Test]

public static void ShouldNotAddUserWithLongPassword()

{

var password = new string('a', 16);

var result = AddUser(Login, password, Firstname, Lastname);

Assert.IsFalse(result.Success);

Assert.AreEqual(Errors.LongPasswordError(), (string)result.Value);

}

[Test]

public static void ShouldNotAddUserWithShortPassword()

{

var password = new string('a', 5);

var result = AddUser(Login, password, Firstname, Lastname);

Assert.IsFalse(result.Success);

Assert.AreEqual(Errors.ShortPasswordError(), (string)result.Value);

}

[Test]

public static void ShouldNotAddUserWithLongFirstname()

{

var firstname = new string('a', 31);

var result = AddUser(Login, Password, firstname, Lastname);

Assert.IsFalse(result.Success);

Assert.AreEqual(Errors.LongFirstnameError(), (string)result.Value);

}

[Test]

public static void ShouldNotAddUserWithEmptyFirstname()

{

var result = AddUser(Login, Password, "", Lastname);

Assert.IsFalse(result.Success);

Assert.AreEqual(Errors.EmptyFirstnameError(), (string)result.Value);

}

[Test]

public static void ShouldNotAddUserWithInvalidFirstname()

{

var firstname = "123456";

var result = AddUser(Login, Password, firstname, Lastname);

Assert.IsFalse(result.Success);

Assert.AreEqual(Errors.InvalidFirstnameError(), (string)result.Value);

}

[Test]

public static void ShouldNotAddUserWithLongLastname()

{

var lastname = new string('a', 31);

var result = AddUser(Login, Password, Firstname, lastname);

Assert.IsFalse(result.Success);

Assert.AreEqual(Errors.LongLastnameError(), (string)result.Value);

}

[Test]

public static void ShouldNotAddUserWithEmptyLastname()

{

var result = AddUser(Login, Password, Firstname, "");

Assert.IsFalse(result.Success);

Assert.AreEqual(Errors.EmptyLastnameError(), (string)result.Value);

}

[Test]

public static void ShouldNotAddUserWithInvalidLastname()

{

var lastname = "123456";

var result = AddUser(Login, Password, Firstname, lastname);

Assert.IsFalse(result.Success);

Assert.AreEqual(Errors.InvalidLastnameError(), (string)result.Value);

}

[Test]

public static void ShouldAuthorizeExistingUser()

{

AddUser(Login, Password, Firstname, Lastname);

var result = GetUser(Login, Password);

Assert.IsTrue(result.Success);

Assert.IsInstanceOf(typeof(User), result.Value);

}

[Test]

public static void ShouldNotAuthorizeNonExistingUser()

{

var result = GetUser(Login, Password);

Assert.IsFalse(result.Success);

Assert.AreEqual(Errors.AuthError(), (string)result.Value);

}

#endregion

#region Chat Part

[Test]

public static void ShouldAddMessage()

{

var result = AddMessage(Content, Login);

Assert.IsTrue(result.Success);

var id = (int)result.Value;

DeleteMessage(id);

}

[Test]

public static void ShouldNotAddEmptyMessage()

{

var result = AddMessage("", Login);

Assert.IsFalse(result.Success);

Assert.AreEqual(Errors.EmptyMessageError(), (string)result.Value);

}

[Test]

public static void ShouldNotAddLongMessage()

{

var content = new string('a', 257);

var result = AddMessage(content, Login);

Assert.IsFalse(result.Success);

Assert.AreEqual(Errors.LongMessageError(), (string)result.Value);

}

[Test]

public static void ShouldNotAddDuplicateMessage()

{

var result = AddMessage(Content, Login);

Assert.IsTrue(result.Success);

var id = (int)result.Value;

result = AddMessage(Content, Login);

Assert.IsFalse(result.Success);

Assert.AreEqual(Errors.DuplicateMessageError(), (string)result.Value);

DeleteMessage(id);

}

[Test]

public static void ShouldGetAllMessages()

{

var result = GetAllMessages();

Assert.IsTrue(result.Success);

Assert.IsInstanceOf(typeof(List<Messages>), result.Value);

}

#endregion

}

## UI Tests

*ChatAppSession.cs*

public class ChatAppSession

{

private const string WindowsApplicationDriverUrl = "http://127.0.0.1:4723";

private const string app = "C:/Users/anton/source/repos/ChatApp/ChatApp/bin/Debug/netcoreapp3.1/ChatApp.exe";

protected static WindowsDriver<WindowsElement> session;

protected static RemoteTouchScreen touchScreen;

protected static WindowsDriver<WindowsElement> chatapp;

protected static WindowsDriver<WindowsElement> mainwindowsession;

public static void Setup(TestContext context)

{

if (session == null || touchScreen == null)

{

TearDown();

DesiredCapabilities appCapabilities = new DesiredCapabilities();

appCapabilities.SetCapability("app", "Root");

session = new WindowsDriver<WindowsElement>(new Uri(WindowsApplicationDriverUrl), appCapabilities);

Assert.IsNotNull(session);

Assert.IsNotNull(session.SessionId);

session.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(1.5);

touchScreen = new RemoteTouchScreen(session);

Assert.IsNotNull(touchScreen);

}

}

public static void TearDown()

{

touchScreen = null;

if (session != null)

{

session.Close();

session.Quit();

session = null;

}

if (chatapp != null)

{

chatapp.Quit();

chatapp = null;

}

if (mainwindowsession != null)

{

mainwindowsession.Quit();

mainwindowsession = null;

}

}

[TestInitialize]

public virtual void TestInit()

{

WindowsElement title = null;

try

{

DesiredCapabilities appCapabilities = new DesiredCapabilities();

appCapabilities.SetCapability("app", app);

chatapp = new WindowsDriver<WindowsElement>(new Uri(WindowsApplicationDriverUrl), appCapabilities);

var mainWindow = session.FindElementByName("MainWindow");

var mainWindowHandle = mainWindow.GetAttribute("NativeWindowHandle");

mainWindowHandle = (int.Parse(mainWindowHandle)).ToString("x");

DesiredCapabilities mainCapabilities = new DesiredCapabilities();

mainCapabilities.SetCapability("appTopLevelWindow", mainWindowHandle);

mainwindowsession = new WindowsDriver<WindowsElement>(new Uri(WindowsApplicationDriverUrl), mainCapabilities);

title = mainwindowsession.FindElementByAccessibilityId("MainWindowTitle");

}

catch

{

throw new Exception();

}

Assert.IsNotNull(title);

Assert.IsTrue(title.Displayed);

}

}

*ScenarioMainWindow.cs*

[TestClass]

public class ScenarioMainWindow : ChatAppSession

{

private const string Login = "testuser";

private const string Password = "testuser";

private const string WrongPassword = "wrongpassword";

private const string WindowsApplicationDriverUrl = "http://127.0.0.1:4723";

private static WindowsDriver<WindowsElement> chat;

private static WindowsDriver<WindowsElement> register;

[TestMethod]

public void ShouldLogin()

{

mainwindowsession.FindElementByAccessibilityId("MainWindowLoginTextBox").Clear();

mainwindowsession.FindElementByAccessibilityId("MainWindowLoginTextBox").SendKeys(Login);

mainwindowsession.FindElementByAccessibilityId("MainWindowPasswordPasswordBox").Clear();

mainwindowsession.FindElementByAccessibilityId("MainWindowPasswordPasswordBox").SendKeys(Password);

mainwindowsession.FindElementByAccessibilityId("MainWindowLoginButton").Click();

Thread.Sleep(1000);

ChatSetUp();

var currentLogin = chat.FindElementByAccessibilityId("ChatWindowCurrentLoginTextBlock");

Assert.IsNotNull(currentLogin);

Assert.AreEqual(Login, currentLogin.Text);

}

[TestMethod]

public void ShouldNotLogin()

{

mainwindowsession.FindElementByAccessibilityId("MainWindowLoginTextBox").Clear();

mainwindowsession.FindElementByAccessibilityId("MainWindowLoginTextBox").SendKeys(Login);

mainwindowsession.FindElementByAccessibilityId("MainWindowPasswordPasswordBox").Clear();

mainwindowsession.FindElementByAccessibilityId("MainWindowPasswordPasswordBox").SendKeys(WrongPassword);

mainwindowsession.FindElementByAccessibilityId("MainWindowLoginButton").Click();

Thread.Sleep(2000);

var messagebox = mainwindowsession.FindElementByName("Login fault");

Assert.IsNotNull(messagebox);

var text = messagebox.FindElementByName("Authorization failed. Please check the login and password.");

Assert.IsNotNull(text);

messagebox.FindElementByName("Закрыть").Click();

}

[TestMethod]

public void ShouldSwitchToRegister()

{

mainwindowsession.FindElementByAccessibilityId("MainWindowRegisterButton").Click();

RegisterSetUp();

var title = register.FindElementByAccessibilityId("RegisterWindowTitle");

Assert.IsNotNull(title);

}

#region Test automating things

private static void RegisterSetUp()

{

var window = session.FindElementByName("Register");

var chatWindowHandle = window.GetAttribute("NativeWindowHandle");

chatWindowHandle = (int.Parse(chatWindowHandle)).ToString("x");

DesiredCapabilities capabilities = new DesiredCapabilities();

capabilities.SetCapability("appTopLevelWindow", chatWindowHandle);

register = new WindowsDriver<WindowsElement>(new Uri(WindowsApplicationDriverUrl), capabilities);

}

private static void ChatSetUp()

{

var chatWindow = session.FindElementByName("Chat");

var chatWindowHandle = chatWindow.GetAttribute("NativeWindowHandle");

chatWindowHandle = (int.Parse(chatWindowHandle)).ToString("x");

DesiredCapabilities capabilities = new DesiredCapabilities();

capabilities.SetCapability("appTopLevelWindow", chatWindowHandle);

chat = new WindowsDriver<WindowsElement>(new Uri(WindowsApplicationDriverUrl), capabilities);

}

[ClassInitialize]

public static void ClassInitialize(TestContext context)

{

Setup(context);

}

[ClassCleanup]

public static void ClassCleanup()

{

try

{

mainwindowsession.Close();

mainwindowsession.Quit();

mainwindowsession.Close();

} catch { }

try

{

chatapp.Close();

chatapp.Quit();

chatapp = null;

} catch { }

try

{

register.Close();

register.Quit();

register = null;

} catch { }

try

{

chat.Close();

chat.Quit();

chat = null;

} catch { }

TearDown();

}

[TestInitialize]

public override void TestInit()

{

base.TestInit();

}

#endregion

}

*ScenarioRegister.cs*

[TestClass]

public class ScenarioRegister : ChatAppSession

{

private const string WindowsApplicationDriverUrl = "http://127.0.0.1:4723";

private const string Login = "testuser";

private const string Password = "testuser";

private const string Firstname = "testuser";

private const string Lastname = "testuser";

private static WindowsDriver<WindowsElement> register;

[TestMethod]

public void ShouldNotRegisterWithLongLogin()

{

register.FindElementByAccessibilityId("RegisterWindowLoginTextBox").SendKeys(new string('a', 16));

register.FindElementByAccessibilityId("RegisterWindowRegisterButton").Click();

}

[TestMethod]

public void ShouldNotRegisterWithShortLogin()

{

register.FindElementByAccessibilityId("RegisterWindowLoginTextBox").SendKeys(new string('a', 5));

register.FindElementByAccessibilityId("RegisterWindowRegisterButton").Click();

}

[TestMethod]

public void ShouldNotRegisterWitInvalidLogin()

{

register.FindElementByAccessibilityId("RegisterWindowLoginTextBox").SendKeys("123456");

register.FindElementByAccessibilityId("RegisterWindowRegisterButton").Click();

}

[TestMethod]

public void ShouldNotRegisterWitLongPassword()

{

register.FindElementByAccessibilityId("RegisterWindowLoginTextBox").SendKeys(Login);

register.FindElementByAccessibilityId("RegisterWindowPasswordPasswordBox").SendKeys(new string('a', 16));

register.FindElementByAccessibilityId("RegisterWindowRegisterButton").Click();

}

[TestMethod]

public void ShouldNotRegisterWitShortPassword()

{

register.FindElementByAccessibilityId("RegisterWindowLoginTextBox").SendKeys(Login);

register.FindElementByAccessibilityId("RegisterWindowPasswordPasswordBox").SendKeys(new string('a', 5));

register.FindElementByAccessibilityId("RegisterWindowRegisterButton").Click();

}

[TestMethod]

public void ShouldNotRegisterWitLongFirstname()

{

register.FindElementByAccessibilityId("RegisterWindowLoginTextBox").SendKeys(Login);

register.FindElementByAccessibilityId("RegisterWindowPasswordPasswordBox").SendKeys(Password);

register.FindElementByAccessibilityId("RegisterWindowFirstnameTextBox").SendKeys(new string('a', 31));

register.FindElementByAccessibilityId("RegisterWindowRegisterButton").Click();

}

[TestMethod]

public void ShouldNotRegisterWitEmptyFirstname()

{

register.FindElementByAccessibilityId("RegisterWindowLoginTextBox").SendKeys(Login);

register.FindElementByAccessibilityId("RegisterWindowPasswordPasswordBox").SendKeys(Password);

register.FindElementByAccessibilityId("RegisterWindowRegisterButton").Click();

}

[TestMethod]

public void ShouldNotRegisterWitLongLastname()

{

register.FindElementByAccessibilityId("RegisterWindowLoginTextBox").SendKeys(Login);

register.FindElementByAccessibilityId("RegisterWindowPasswordPasswordBox").SendKeys(Password);

register.FindElementByAccessibilityId("RegisterWindowFirstnameTextBox").SendKeys(Firstname);

register.FindElementByAccessibilityId("RegisterWindowLastnameTextBox").SendKeys(new string('a', 31));

register.FindElementByAccessibilityId("RegisterWindowRegisterButton").Click();

}

[TestMethod]

public void ShouldNotRegisterWitEmptyLastname()

{

register.FindElementByAccessibilityId("RegisterWindowLoginTextBox").SendKeys(Login);

register.FindElementByAccessibilityId("RegisterWindowPasswordPasswordBox").SendKeys(Password);

register.FindElementByAccessibilityId("RegisterWindowFirstnameTextBox").SendKeys(Firstname);

register.FindElementByAccessibilityId("RegisterWindowRegisterButton").Click();

}

[TestMethod] public void ShouldNotRegisterWitDuplicateLogin()

{

register.FindElementByAccessibilityId("RegisterWindowLoginTextBox").SendKeys(Login);

register.FindElementByAccessibilityId("RegisterWindowPasswordPasswordBox").SendKeys(Password);

register.FindElementByAccessibilityId("RegisterWindowLastnameTextBox").SendKeys(Lastname);

register.FindElementByAccessibilityId("RegisterWindowFirstnameTextBox").SendKeys(Firstname);

register.FindElementByAccessibilityId("RegisterWindowRegisterButton").Click();

}

#region Automation things

[ClassInitialize]

public static void ClassInitialize(TestContext context)

{

Setup(context);

}

[TestCleanup]

public void TestCleanup()

{

try

{

register.FindElementByName("Registration fault").FindElementByName("Закрыть").Click();

register.Close();

}

catch

{

}

}

[ClassCleanup]

public static void ClassCleanup()

{

try

{

register.Close();

register.Quit();

register = null;

}

catch

{

}

}

[TestInitialize]

public override void TestInit()

{

base.TestInit();

mainwindowsession.FindElementByAccessibilityId("MainWindowRegisterButton").Click();

var window = session.FindElementByName("Register");

var chatWindowHandle = window.GetAttribute("NativeWindowHandle");

chatWindowHandle = (int.Parse(chatWindowHandle)).ToString("x");

DesiredCapabilities capabilities = new DesiredCapabilities();

capabilities.SetCapability("appTopLevelWindow", chatWindowHandle);

register = new WindowsDriver<WindowsElement>(new Uri(WindowsApplicationDriverUrl), capabilities);

}

#endregion

}

*ScenarioChat.cs*

[TestClass]

public class ScenarioChat : ChatAppSession

{

private const string WindowsApplicationDriverUrl = "http://127.0.0.1:4723";

private const string app = "C:/Users/anton/source/repos/ChatApp/ChatApp/bin/Debug/netcoreapp3.1/ChatApp.exe";

private const string Login = "testuser";

private const string Password = "testuser";

private static WindowsDriver<WindowsElement> chat;

[TestMethod]

public void ShouldNotSendEmptyMessage()

{

chat.FindElementByAccessibilityId("ChatWindowSendMessageButton").Click();

var messagebox = chat.FindElementByName("Send message fault");

Assert.IsNotNull(messagebox);

var text = messagebox.FindElementByName("Message can not be empty.");

Assert.IsNotNull(text);

messagebox.FindElementByName("Закрыть").Click();

}

[TestMethod]

public void ShouldNotSendLongMessage()

{

chat.FindElementByAccessibilityId("ChatWindowMessageTextBox").SendKeys(new string('a', 257));

chat.FindElementByAccessibilityId("ChatWindowSendMessageButton").Click();

var messagebox = chat.FindElementByName("Send message fault");

Assert.IsNotNull(messagebox);

var text = messagebox.FindElementByName("Message length can not be greater than 256 characters.");

Assert.IsNotNull(text);

messagebox.FindElementByName("Закрыть").Click();

}

[TestMethod]

public void ShouldNotSendDuplicateMessage()

{

chat.FindElementByAccessibilityId("ChatWindowMessageTextBox").SendKeys("test");

chat.FindElementByAccessibilityId("ChatWindowSendMessageButton").Click();

try

{

chat.FindElementByAccessibilityId("ChatWindowMessageTextBox").SendKeys("test");

chat.FindElementByAccessibilityId("ChatWindowSendMessageButton").Click();

}

catch { }

var messagebox = chat.FindElementByName("Send message fault");

Assert.IsNotNull(messagebox);

var text = messagebox.FindElementByName("Message should be different from your previous message.");

Assert.IsNotNull(text);

messagebox.FindElementByName("Закрыть").Click();

}

[TestMethod]

public void ShouldUpdateMessageListOnMessageSend()

{

var messages = chat.FindElementByAccessibilityId("ScrollViewer")

.FindElementByAccessibilityId("ChatWindowMessagesListBox");

var messagessCount = messages.FindElements(By.Name("ChatApp.Models.Messages")).Count;

var message = $"Current time is {DateTime.Now}";

chat.FindElementByAccessibilityId("ChatWindowMessageTextBox").SendKeys(message);

chat.FindElementByAccessibilityId("ChatWindowSendMessageButton").Click();

messages = chat.FindElementByAccessibilityId("ScrollViewer")

.FindElementByAccessibilityId("ChatWindowMessagesListBox");

var newMessagesCount = messages.FindElements(By.Name("ChatApp.Models.Messages")).Count;

Assert.AreEqual(messagessCount + 1, newMessagesCount);

}

[TestMethod]

public void ShouldUpdateMessagesOnlyOnRefreshButtonClick()

{

var messages = chat.FindElementByAccessibilityId("ScrollViewer")

.FindElementByAccessibilityId("ChatWindowMessagesListBox");

var messagesCount = messages.FindElements(By.Name("ChatApp.Models.Messages")).Count;

#region New chat user

// new chat user

DesiredCapabilities appCapabilities = new DesiredCapabilities();

appCapabilities.SetCapability("app", app);

var newchat = new WindowsDriver<WindowsElement>(new Uri(WindowsApplicationDriverUrl), appCapabilities);

var mainWindow = session.FindElementByName("MainWindow");

var mainWindowHandle = mainWindow.GetAttribute("NativeWindowHandle");

mainWindowHandle = (int.Parse(mainWindowHandle)).ToString("x");

DesiredCapabilities mainCapabilities = new DesiredCapabilities();

mainCapabilities.SetCapability("appTopLevelWindow", mainWindowHandle);

var newmainwindowsession = new WindowsDriver<WindowsElement>(new Uri(WindowsApplicationDriverUrl), mainCapabilities);

// new user login process

newmainwindowsession.FindElementByAccessibilityId("MainWindowLoginTextBox").SendKeys(Login);

newmainwindowsession.FindElementByAccessibilityId("MainWindowPasswordPasswordBox").SendKeys(Password);

newmainwindowsession.FindElementByAccessibilityId("MainWindowLoginButton").Click();

var window = session.FindElementByName("Chat");

var chatWindowHandle = window.GetAttribute("NativeWindowHandle");

chatWindowHandle = (int.Parse(chatWindowHandle)).ToString("x");

DesiredCapabilities capabilities = new DesiredCapabilities();

capabilities.SetCapability("appTopLevelWindow", chatWindowHandle);

var newchatuser = new WindowsDriver<WindowsElement>(new Uri(WindowsApplicationDriverUrl), capabilities);

#endregion

var message = $"Current time is {DateTime.Now}";

newchatuser.FindElementByAccessibilityId("ChatWindowMessageTextBox").SendKeys(message);

newchatuser.FindElementByAccessibilityId("ChatWindowSendMessageButton").Click();

messages = chat.FindElementByAccessibilityId("ScrollViewer")

.FindElementByAccessibilityId("ChatWindowMessagesListBox");

var messagesAfterSendCount = messages.FindElements(By.Name("ChatApp.Models.Messages")).Count;

Assert.AreEqual(messagesCount, messagesAfterSendCount);

chat.FindElementByAccessibilityId("ChatWindowRefreshButton").Click();

messages = chat.FindElementByAccessibilityId("ScrollViewer")

.FindElementByAccessibilityId("ChatWindowMessagesListBox");

var messagesAfterRefresh = messages.FindElements(By.Name("ChatApp.Models.Messages")).Count;

Assert.AreEqual(messagesCount + 1, messagesAfterRefresh);

}

#region Automation things

[ClassInitialize]

public static void ClassInitialize(TestContext context)

{

Setup(context);

}

[TestCleanup]

public void TestCleanup()

{

try

{

}

catch

{

}

}

[ClassCleanup]

public static void ClassCleanup()

{

try

{

chat.Close();

chat.Quit();

chat = null;

}

catch { }

}

[TestInitialize]

public override void TestInit()

{

base.TestInit();

mainwindowsession.FindElementByAccessibilityId("MainWindowLoginTextBox").SendKeys(Login);

mainwindowsession.FindElementByAccessibilityId("MainWindowPasswordPasswordBox").SendKeys(Password);

mainwindowsession.FindElementByAccessibilityId("MainWindowLoginButton").Click();

var window = session.FindElementByName("Chat");

var chatWindowHandle = window.GetAttribute("NativeWindowHandle");

chatWindowHandle = (int.Parse(chatWindowHandle)).ToString("x");

DesiredCapabilities capabilities = new DesiredCapabilities();

capabilities.SetCapability("appTopLevelWindow", chatWindowHandle);

chat = new WindowsDriver<WindowsElement>(new Uri(WindowsApplicationDriverUrl), capabilities);

}

#endregion

}

# Выводы

В ходе данной работы я познакомился с подходом TDD. Как утверждает один из пользователей Habr-а, у новичков в TDD создание проекта занимает на 25-30% больше времени. Несмотря на то, что эти данные не имеют основания и являются сугубо субъективными, я тоже почувствовал, что разработка прошла гораздо медленней.

Обычные unit тесты для меня оказались не очень сложными, пришлось просто посидеть и подумать. UI тесты же оказались испытанием для меня. Во-первых, сам подход написания UI тестов состоит в использовании существующих элементов через их id и т. п. Во-вторых, сам подход тестирования десктоп приложений так же оказался не простой задачей. Несмотря на то, что я использовал знакомый мне Appium, возникло немало проблем. Одна из них заключалась в том, что после смены окон в wpf, новое окно было вне зоны видимости драйвера, поэтому пришлось указывать драйвер на каждое отдельное окно.

По моему мнению подход TDD является наиболее оправданным в разработке больших приложений, ведь там ошибка на позднем этапе разработки обойдется в большую затрату. В моем, относительно маленьком, приложении эффект TDD был значительно меньше.