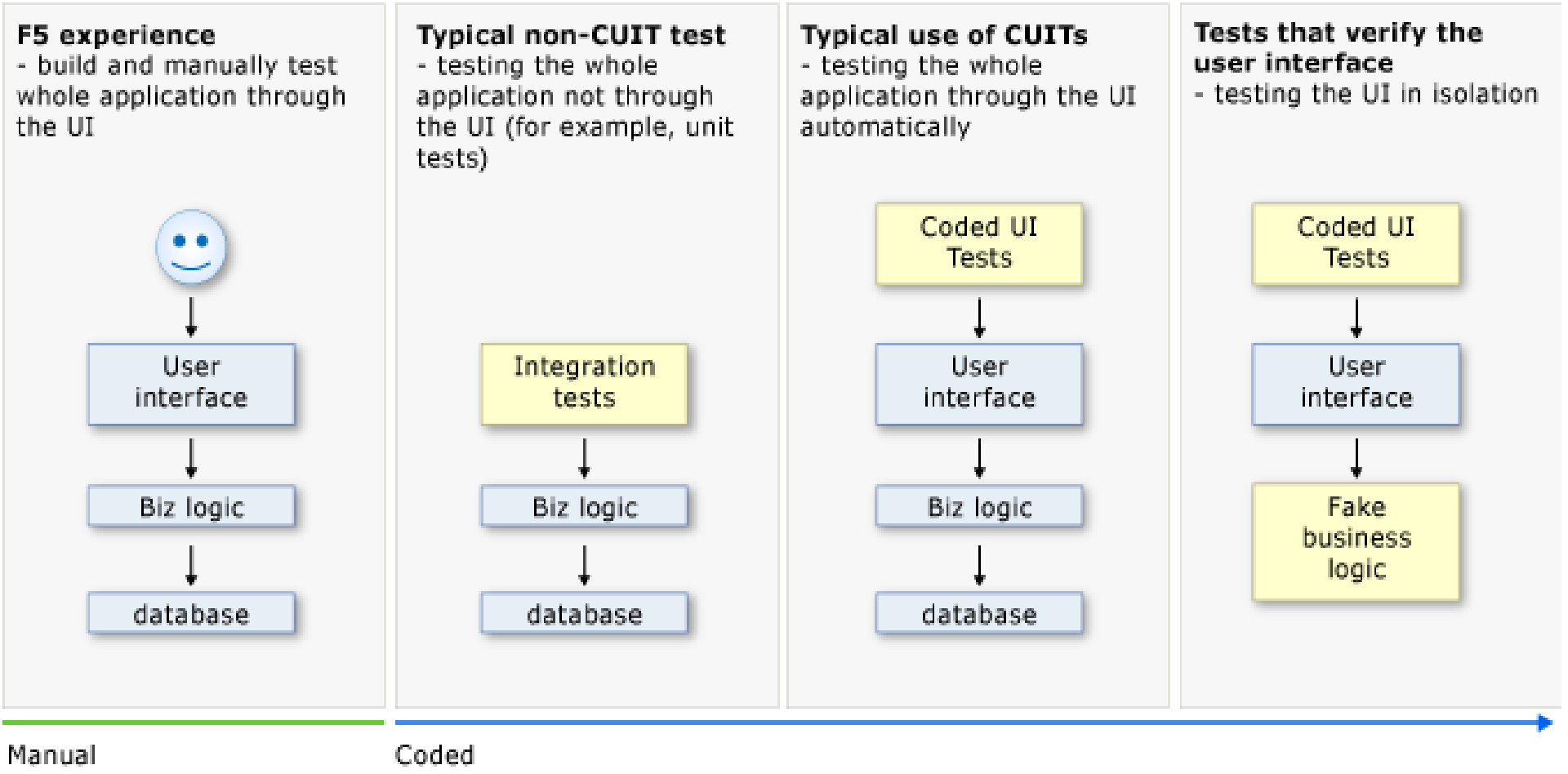
UI Tests

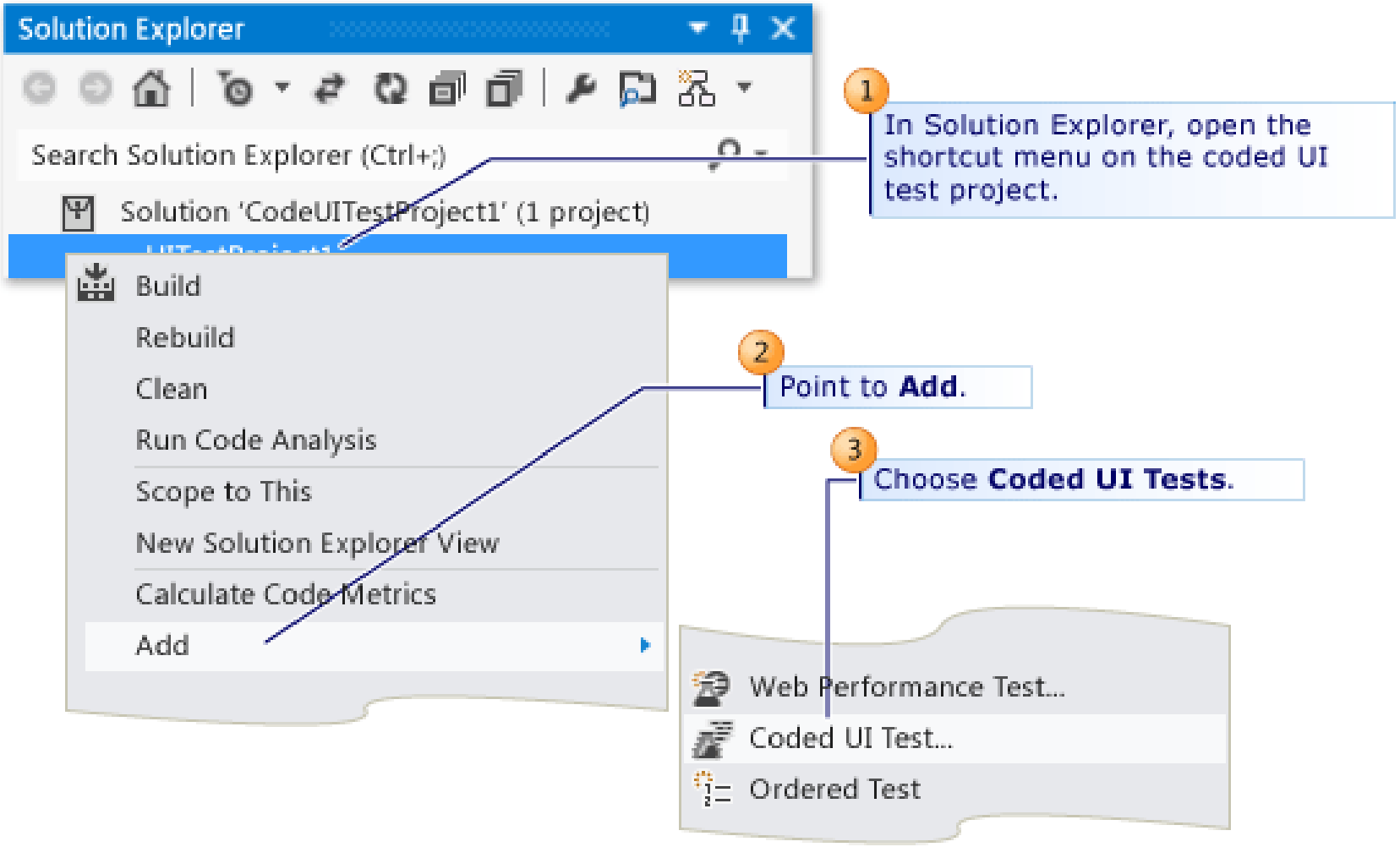
1) MS Coded UI Tests
2) pywinauto
3) TestStack.White

UI тестирование

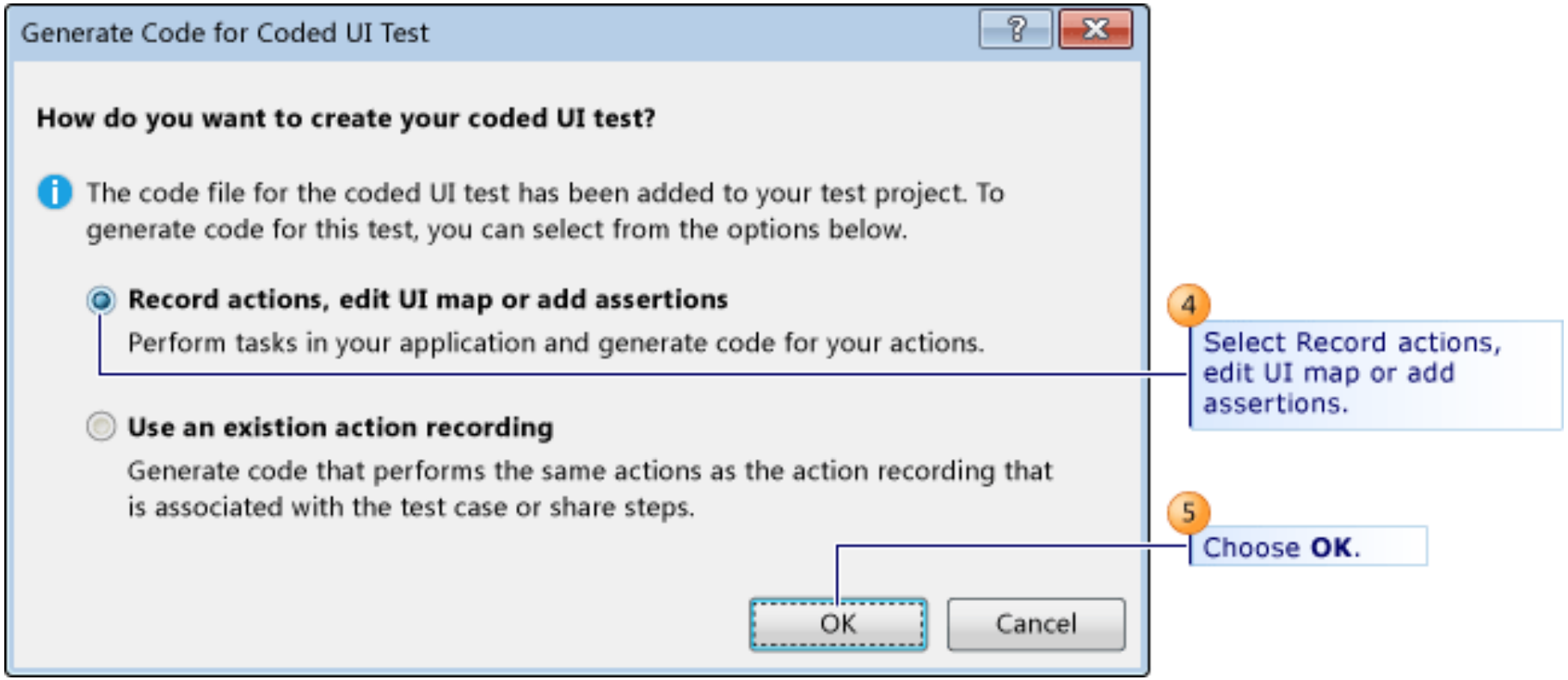


http://msdn.microsoft.com/en-us/library/dd286726.aspx

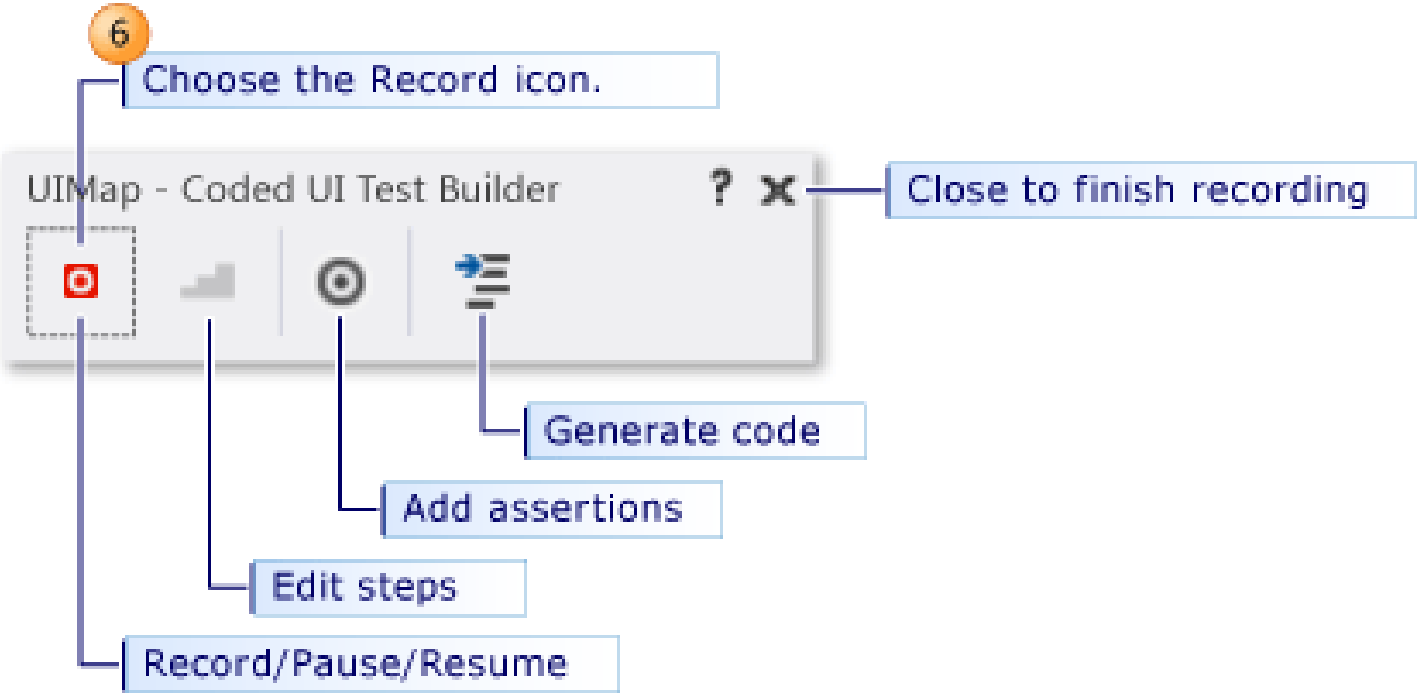
Добавление Coded UI Test в консольный проект



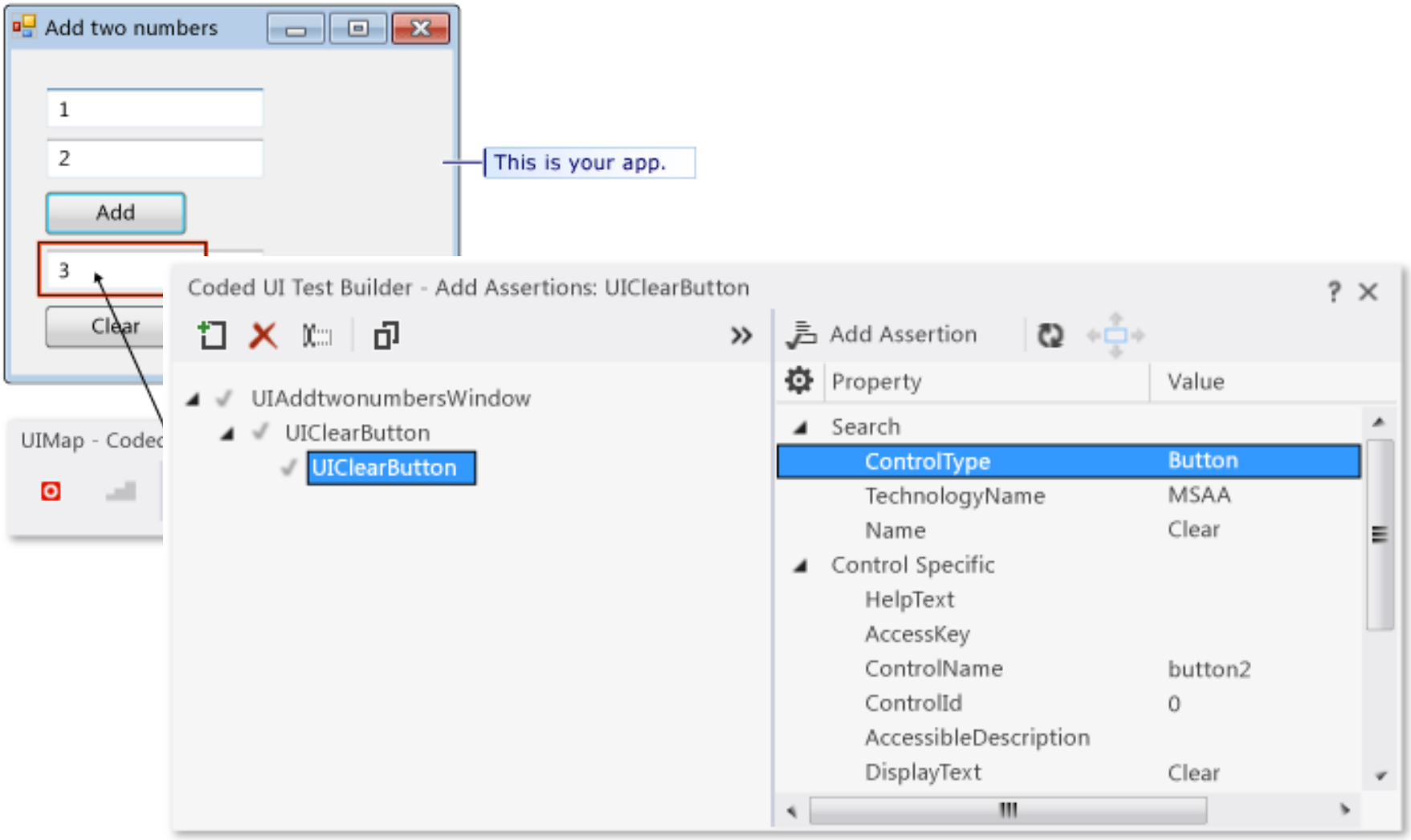
Запись событий и генерация кода тестов



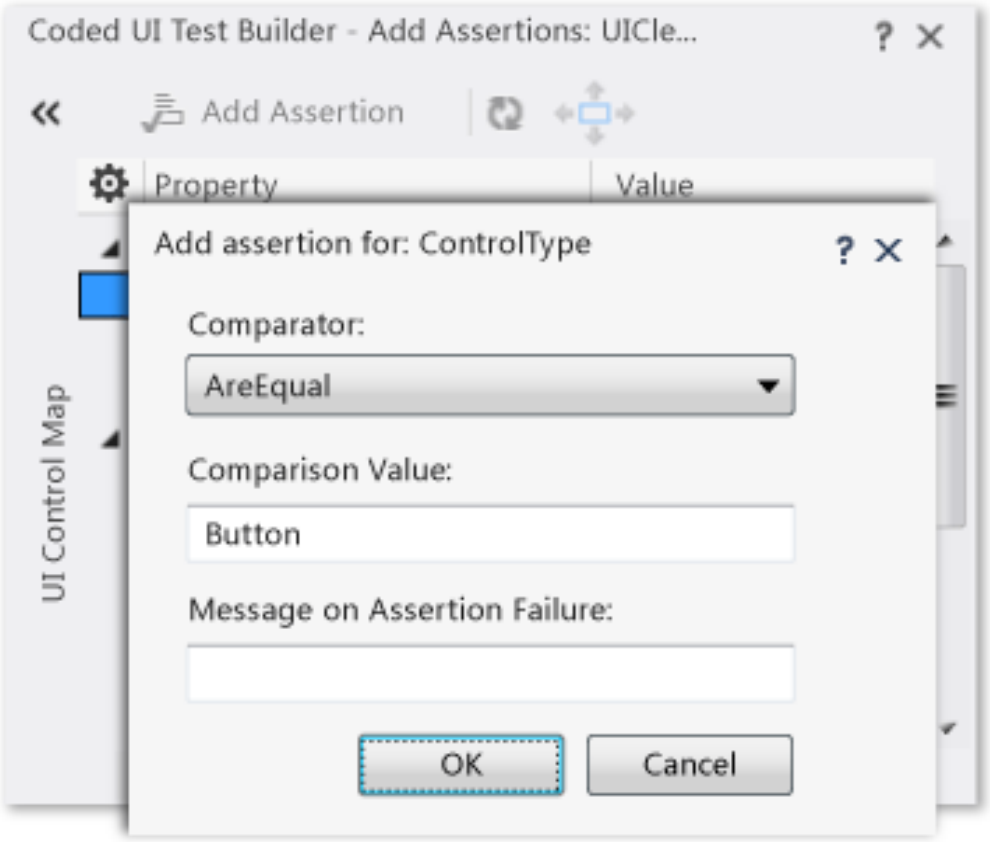
Coded UI Test Builder



Coded UI Test Builder



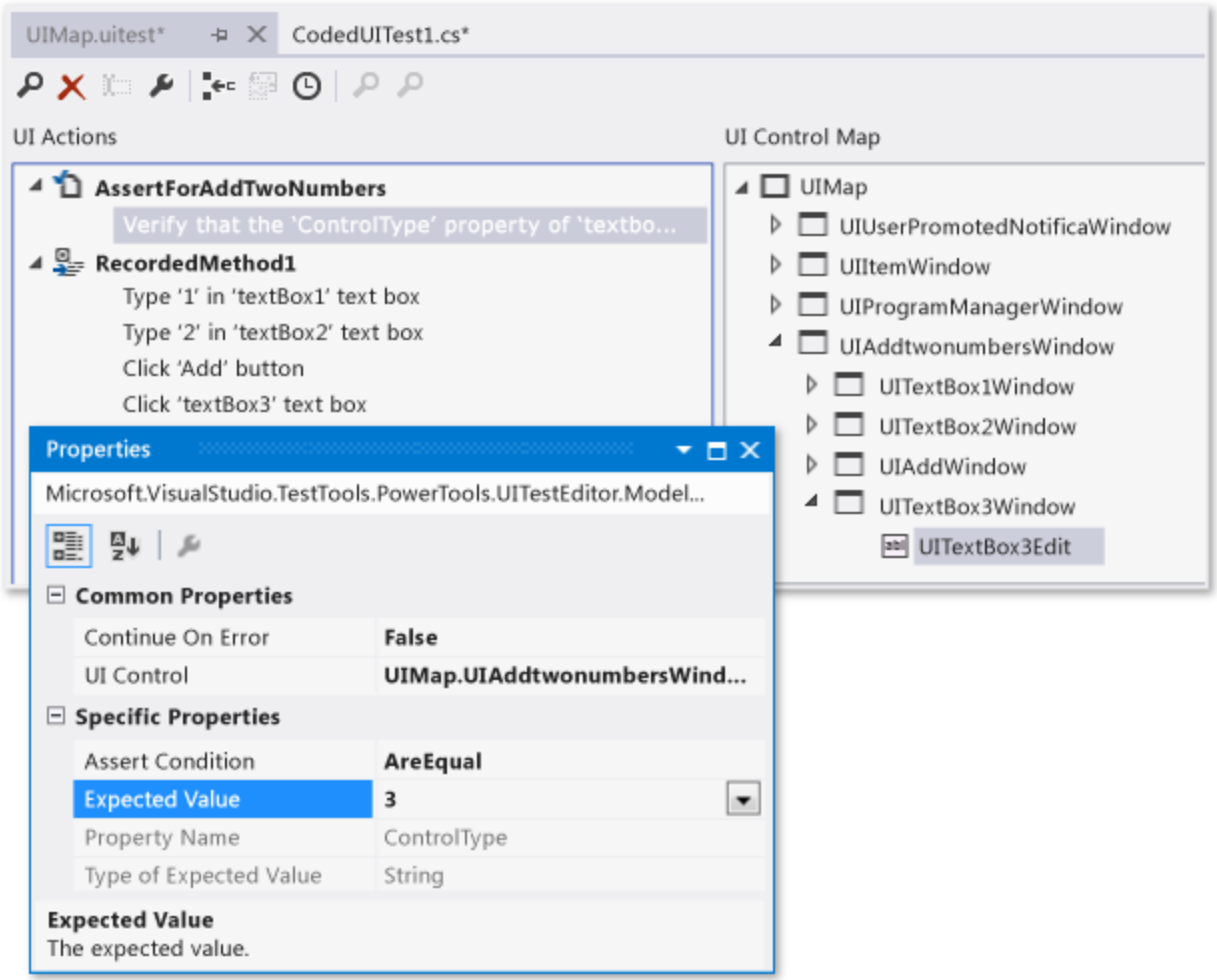
Coded UI Test Builder



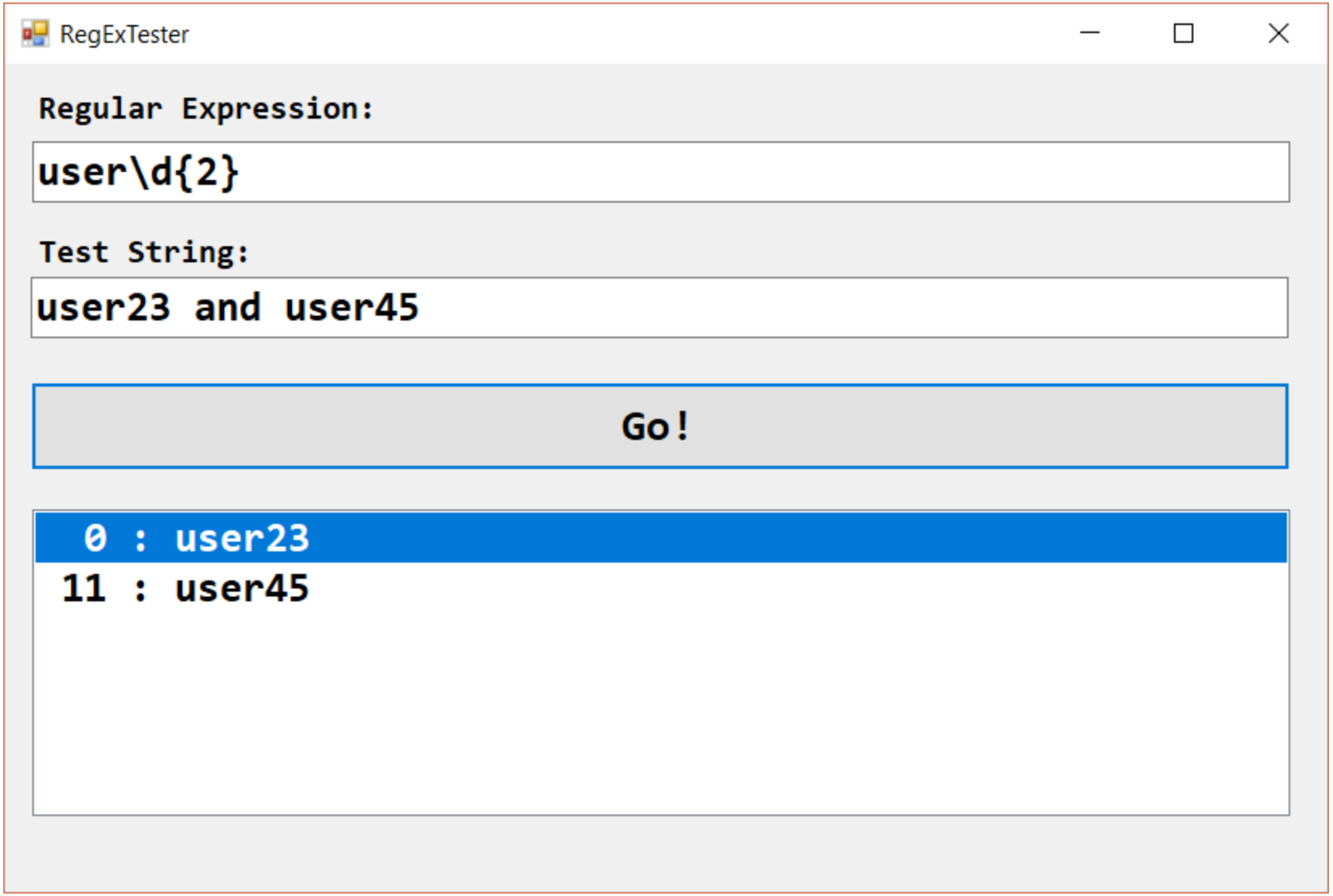
Coded UI Test Builder

```
[TestMethod]
public void CodedUITestMethod1()
{
        this.UIMap.AddTwoNumbers();
        this.UIMap.AssertForAddTwoNumbers();
}
```

Coded UI Test Builder



Ex.2 – App under Test



Coded UI Test Builder (Ex. 2)

```
[TestMethod]
public void CodedUITestMethod1()
{
    this.UIMap.StartApp();
    this.UIMap.EnterRegex( @"\d{2}" );
    this.UIMap.EnterTestString( "a23" );
    this.UIMap.buttonClick();
    this.UIMap.AssertCorrectMatch( "23" );

    this.UIMap.EnterRegex(@"\d{3}");
    this.UIMap.EnterTestString("a123");
    this.UIMap.buttonClick();
    this.UIMap.AssertCorrectMatch("123");

    this.UIMap.EnterRegex(@"\d{2}-\d{1}");
    this.UIMap.EnterTestString("a23-4");
    this.UIMap.buttonClick();
    this.UIMap.AssertCorrectMatch("23-4");
}
```

Coded UI Test Builder (Ex. 2)

```
    public void EnterRegex( string regex )
    {
        #region Variable Declarations
        WinClient uIRegExTesterClient = this.UIRegExTesterWindow.UIRegExTesterClient;
        WinEdit uITextBoxRegExEdit =
this.UIRegExTesterWindow.UITextBoxRegExWindow.UITextBoxRegExEdit;
        #endregion

        // Click 'RegExTester' client
        Mouse.Click(uIRegExTesterClient, new Point(137, 79));

        // Click 'RegExTester' client
        Mouse.Click(uIRegExTesterClient, new Point(130, 74));

        // Type '\d{2}' in 'textBoxRegEx' text box
        uITextBoxRegExEdit.Text = regex;// this.EnterRegexParams.UITextBoxRegExEditText;

        // Type '{Right}' in 'textBoxRegEx' text box
        Keyboard.SendKeys(uITextBoxRegExEdit, this.EnterRegexParams.UITextBoxRegExEditSendKeys,
ModifierKeys.None);
    }
```

Coded UI Test Builder (Ex. 2)

```
public void AssertCorrectMatch( string correct )
    {
        #region Variable Declarations
        WinListItem uIItem123ListItem =
this.UIRegExTesterWindow.UIListBoxResultsWindow.UIItem123ListItem;
        #endregion

        // Verify that '1 : 23' list item's property 'DisplayText' contains '23'
        StringAssert.Contains(uIItem123ListItem.DisplayText, correct);
            //this.AssertCorrectMatchExpectedValues.UIItem123ListItemDisplayText);
    }
```

Делай так…

o Each recorded method should act on a single page, form, or dialog box. Create a new test method for each new page, form or dialog box.

o Use the Coded UI Test Builder whenever possible.

o Explicitly set focus to the window on which the test Case is expected to input data.

o When possible, limit each recorded method to fewer than 10 actions. This modular approach makes it easier to replace a method if the UI changes.

o Create a separate UIMap file for each module in your application under test. For more information, see Testing a Large Application with Multiple UI Maps.

o If you are creating assertions by coding with the API, create a method for each assertion in the part of the UIMap class that is in the UIMap.cs file. Call this method from your test method to execute the assertion.

o Capture screen shots for failures. Helps in debugging.

Делай так…

o Leave the UI in a known state after a test Case is done executing. E.g.: Close the wizard even if the test Case is to test the first screen of a wizard by cancelling out of the wizard after the first screen.

o Log before you do something, not after.

o In code reviews for test cases, or new framework automation support review the log messages, is it clear what the test did from just the log?

o Have two assertion modes that are configurable: 1. Fail instantly. 2. Delay assertion of

failure until the test has completed. Steps might be mutually exclusive, you're losing coverage.

o Log non-fatal exceptions so as not to abort a test, bubble fatal exceptions up to the test and let the exception assert failure.

o Log every valid UI operation being performed in the test Case.

o Validate that the object being created using the UI with the one being stored in the backend system

Не делай так…

o Do not modify the UIMap.designer.cs file directly. If you do this, the changes to the file will be overwritten.

o Do not log multiple times.

o Do not sleep after every UI operation.

o Do not give unusually long timeouts for every major operation.

o Fail immediately after an unexpected screen.

o Do not re-launch the UI application for every test Case as it is a time consuming process.

o Test cases should never have hard-coded strings. This includes control ids, menu paths, etc. Even control IDs and menu paths change over time.

o Do not use sleep

pywinauto

https://habrahabr.ru/company/yandex/blog/336476/

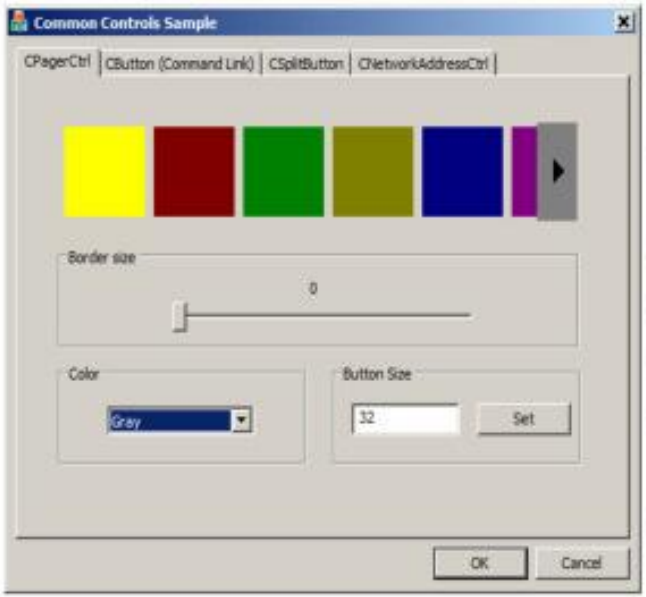
Accessibility технологии

Win32 API	MS UI Automation
MFC, VCL, частично WinForms (Spy++)	WinForms, WPF, Qt, браузеры, Store apps (Inspect.exe)
AT SPI (через Dbus)	**Apple Accessibility API**
Qt, GTK, wxWidgets, ...	Cocoa, ...

pywinauto

https://habrahabr.ru/company/yandex/blog/336476/

Пример скрипта на pywinauto




```
app = Application().start("sample.exe")

dlg = app.CommonControlsSample

dlg.ColorComboBox.Select("Green")
dlg.ButtonSizeEdit.SetText("64")
dlg.Set.Click()
dlg.OK.Click()

dlg.WaitNot("visible")
```

pywinauto

https://pywinauto.readthedocs.io/en/latest/getting_started.html

Once you have installed pywinauto - how do you get going? The very first necessary thing is to determine which accessibility technology (pywinauto's backend) could be used for your application.

The list of supported accessibility technologies on Windows:

- **Win32 API (`backend="win32"`) - a default backend for now**
  - MFC, VB6, VCL, simple WinForms controls and most of the old legacy apps
- **MS UI Automation (`backend="uia"`)**
  - WinForms, WPF, Store apps, Qt5, browsers

pywinauto

https://habrahabr.ru/company/yandex/blog/336476/

Объект Application

```
app = Application().start(“sample.exe”)
app = Application(backend='win32').connect(path=”sample.exe”)
app = Application(backend='uia').connect(title_re=”^.*Sample$”)
app.kill_()
# process-agnostic way
desktop = Desktop(backend=”uia”)
```

pywinauto

https://habrahabr.ru/company/yandex/blog/336476/

Как создать описание?

```
app.MainWindow.OK
app["Main Window"]["OK"]
app.window(best_match="Main Window") \
                .child_window(best_match="OK")

app.MainWindow.child_window(title="OK",
      работает быстрее  --->  class_name="Button")
```

pywinauto

https://habrahabr.ru/company/yandex/blog/336476/

Пример для explorer.exe

```
app = Application(backend="uia")
app.connect(path="explorer.exe", title="pywinauto")
app.pywinauto.set_focus()

list_view = app.pywinauto.ItemsView.wrapper_object()
appveyor = list_view.get_item('appveyor.yml')
appveyor.right_click_input()

app.ContextMenu.Properties.invoke()

dlg = Desktop(backend="uia")["appveyor.yml Properties"]
dlg.Cancel.click()
```

pywinauto

https://habrahabr.ru/company/yandex/blog/336476/

Где взять имена для доступа?

```
app.MainWindow.print_control_identifiers()

ComboBox - 'Gray'    (L789, T594, R906, B615)
        'Border sizeComboBox' 'ComboBox' ()
Button - 'Set'    (L1083, T592, R1157, B615)
        'Button' 'Button0' 'Button1' 'Set' 'SetButton' ()
Button - 'OK'    (L1048, T695, R1123, B718)
        'Button2' 'OK' 'OKButton' ()
Button - 'Cancel'    (L1129, T695, R1204, B718)
        'Button3' 'Cancel' 'CancelButton' ()
```

pywinauto

https://habrahabr.ru/company/yandex/blog/336476/

Способы делать магию

1. By title: app.Properties.OK.click()
2. By title+type: app.PropertiedDialog.OKButton.click()
3. By type+index: app.Properties.Button3.click()
4. By top-left label: app.SaveAs.FileNameEdit.set_text(“”)
5. By item text:

   app.Properties.TabControlSharing.select(“General”)

pywinauto

https://habrahabr.ru/company/yandex/blog/336476/

Обработка ожиданий

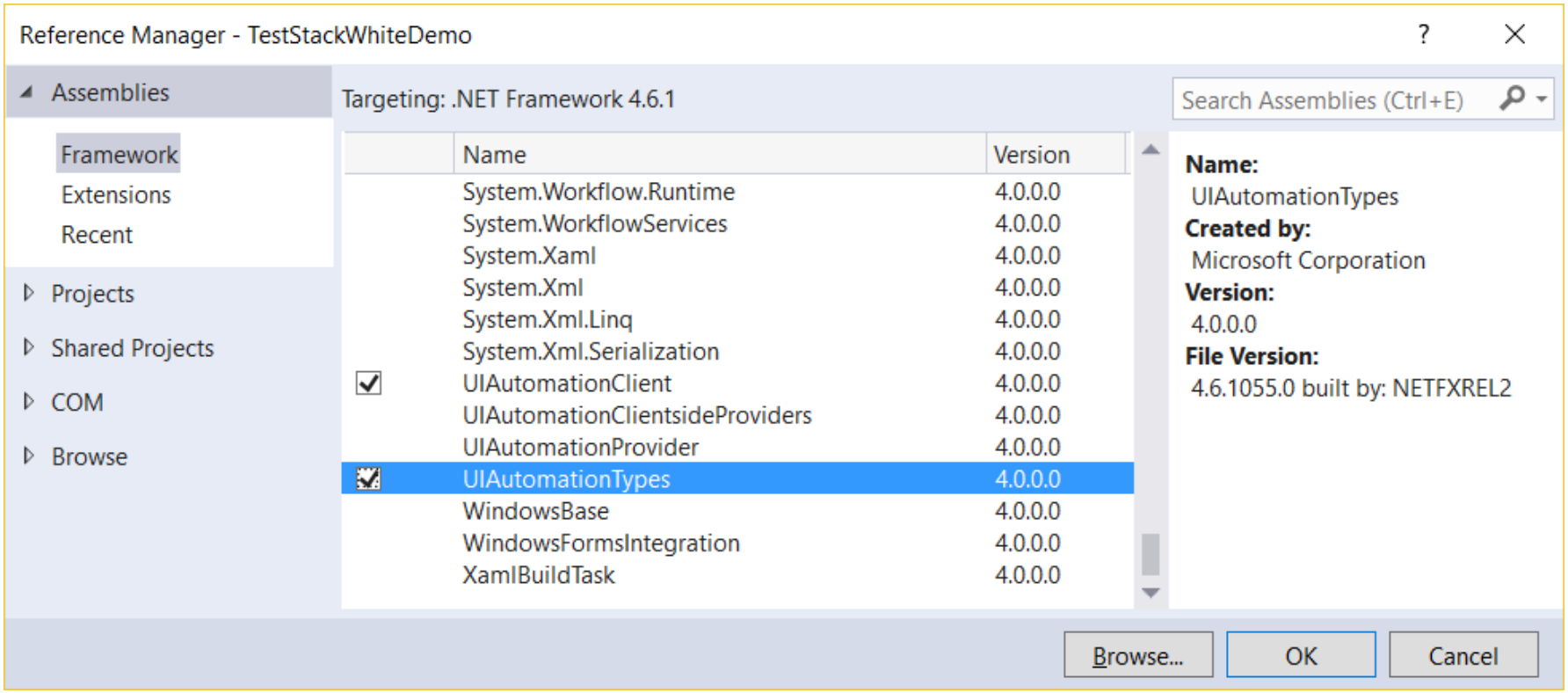
```
window = app.Window_(title="Main Window")
window.wait("active", timeout=20)
window.wait_not("visible enabled")

window.exists() / .visible() / .enabled() / .active()

# wait until CPU usage < 2%
app.wait_cpu_usage_lower(threshold=2, timeout=10)
```

TestStack.White

1) Download and install TestStack.White via NuGet
2) Add references (optional):



TestStack.White

```
[TestFixture]
public class TestClass
{
    // check http://teststackwhite.readthedocs.io/en/latest/ScreenObjects/ScreenRepository/
    // for examples of Screen Object pattern (similar to Page Object pattern)

    private Application _app;
    private Window _window;
    private TextBox _regex;
    private TextBox _testString;
    private Button _okButton;

    [OneTimeSetUp]
    public void Init()
    {
        // we'll work with the same instances of app and window for all tests
        // (it is safe, since NUnit tests run NOT in parallel by default)

        _app = Application.Launch(@"E:\Projects\Simple\Students\RegExTester\RegExTester\bin\Release\RegExTester.exe");
        _window = _app.GetWindow("RegExTester", InitializeOption.NoCache);

        var searchCriteria = SearchCriteria.ByControlType(ControlType.Edit).AndIndex(0);
        _regex = (TextBox)_window.Get(searchCriteria);

        searchCriteria = SearchCriteria.ByControlType(ControlType.Edit).AndIndex(1);
        _testString = (TextBox)_window.Get(searchCriteria);

        searchCriteria = SearchCriteria.ByText("Go!");
        _okButton = (Button)_window.Get(searchCriteria);
    }
```

TestStack.White

```
[Test]
public void TestNormalRegEx()
{
    _testString.Text = "user23 and user45";
    _regex.Text = @"user\d{2}";
    _okButton.Click();

    Assert.That(_window.Get<ListBox>("listBoxResults").Items.Count, Is.EqualTo(2));
}

[Test]
public void TestNoMatch()
{
    _testString.Text = "hello world!";
    _regex.Text = @"vas\d{2}";
    _okButton.Click();

    Assert.That(_window.Get<ListBox>("listBoxResults").Items.Count, Is.EqualTo(0));
}

[OneTimeTearDown]
public void Quit()
{
    _app.Close();
}
```