An Automated Testing Tool Using UI Structure

Nutharat Harnvorawong, Taratip Suwannasart, *Member, IAENG*

Abstract—**Testers usually run a new version of software against existing test cases to validate that changes do not cause any unexpected results in legacy functionalities when the software is modified or enhanced. A solution that can reduce cost is automated testing. However regression testing and automated testing are resources consuming and high cost. In this paper we propose a framework called Neo Automation Framework (NAF) which allows testers to create and develop automated test cases easily and efficiently. The framework can generate UI structure of a given form inside application. The UI structure is a list of usable UI controls in hierarchical data structure in a class format of programming language. Automated test cases can be automatically generated from the UI structure. The framework also provides tools for simply modifying test case and analyzing usage of UI controls in the test cases. The analyzer tool can identify UI controls which are not used and so testers can be aware of area lacking of test in their test cases.**

Index Terms—**Automated Testing, GUI, Automation, Software Testing, Neo Automation Framework**

I. Introduction

Software testing is one of essential parts in software development life cycle to ensure software works correctly as expected [1]. Software testers need to understand software under test in order to design test cases properly and therefore defects in the software can be detected or discovered. When a new feature is developed, new test cases are created to validate functionalities and ensure if quality of the software is satisfied. Not only the new feature is tested but also existing features should be tested to ensure what was previously working still works correctly [2]. This is regression testing. Obviously number of test cases can be increasing rapidly and effort to execute test cases is numerous whereas resources are still being the same. It is a hard work of testers to validate new developed features and also validate all existing features every time a new version of software is released.

Nowadays automated testing is widely used in software testing [3] especially for regression testing as it is fast, reliable, and repeatable. It can reduce cost of testing significantly. However, there are some concerns when creating scripts to use in automated testing such as overhead for testers to learn an automation framework, overhead of maintaining automated test cases when software is changed, how much automated test cases are enough?, where is area lacking of test?, and etc. In this paper we propose an automation test framework which can help testers to automate test cases easily and help identify parts in software which lack of testing in automated test cases.

II. Related Works

There are many techniques used in automated testing. A technique which is easy for creating automated test cases called capture & playback [4] which was introduced in many testing tools in market such as Microsoft Visual Studio [5], and etc. An advantage of this approach is that automated test cases can be created fast but a disadvantage is that UI controls are only recorded if it is used in the automated test cases. If testers want to add more test cases for other UI controls, testers cannot reuse an existing list of UI controls and need to rerecord actions for new test cases specially.

Another technique that is frequently used is model based [6]. André M. P. Grilo and his team [7] have worked in GUI model for software testing. They have proposed to extract structural information of GUI and store it in XML format. From their work, it shows if UI controls in application are recorded in hierarchical data structure, i.e. XML, it could be easy to use and can be processed for other purposes easily.

Another interesting technique used when executing automated test has been proposed by Alex Ruiz and his team [8]. They have proposed an approach how to write automated test cases by using library with concept of "fluent interfaces” to simplify automated test cases in coding style. Their work shows that automated test cases written in code like programming language is readable and is also powerful as testers can use power of programming language to control flow of a test scenario.

III. Neo Automation Framework

Neo Automation Framework (NAF) is an automation test framework which provides tools to facilitate testers' works. It has been designed to support automated testing with Windows application. We chose Windows application developed by using .NET framework as a target of application under test because it is a well-known programming language and is widely used. NAF has a library of API for each supported UI control [9] which wraps UIA [10] commands for easy to use. The framework diagram is shown in **Fig 1**.

Manuscript received December 9, 2013; revised December 23, 2013.

N. H. Author is a student at the faculty of Engineering, Chulalongkorn University, Bangkok, Thailand (e-mail: Nutharat.H@student.chula.ac.th).

T. S. Author is with the faculty of Engineering, Chulalongkorn University, Bangkok, Thailand (e-mail: Taratip.S@chula.ac.th).

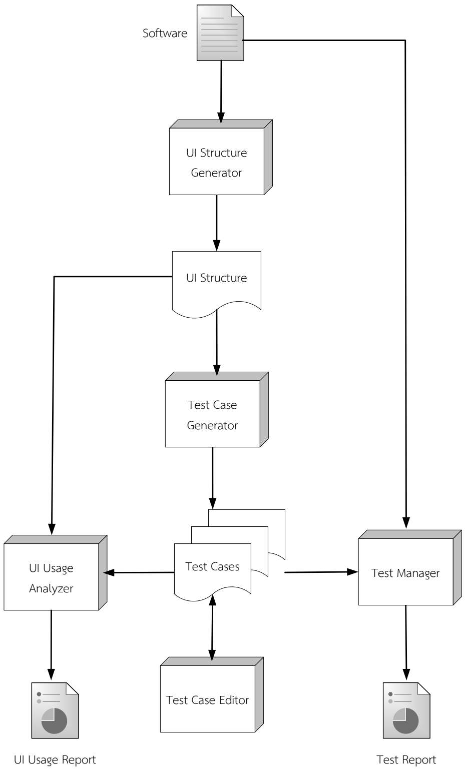


Fig 1. Neo Automation Framework

A. *Software*

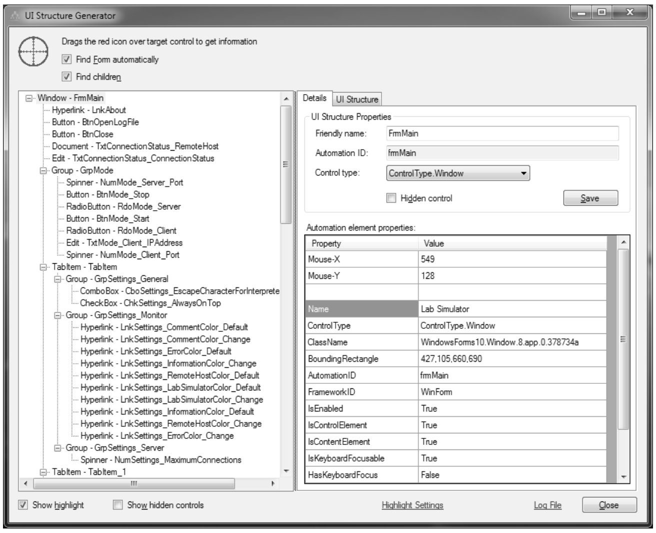
A Graphical User Interface (GUI) application consists of UI controls that can interact with users, for example button, radio button, and text box. Layout of these UI controls are designed and set since design phase by programmers. In case of many UI controls, they probably are grouped in a UI container control such as group box, tab, and etc., for ease of use.

A Windows application developed by using .NET framework and running on Windows operating system can be used in automated testing of this paper.

B. *UI Structure Generator*

UI Structure Generator is a tool that analyzes application under test and extracts information of UI controls such as automation ID, type of control, position of control, and etc., and generates a list of UI controls as a hierarchy and saves it into a format of class. Testers can use the generated UI structure in their test project and can access the UI control by accessing properties of UI structure class.

This tool has an ability to find a form which is expected to be a root in UI structure class. Testers just drag and drop cursor onto target form of application under test, then the tool will automatically extract UI control information of selected form and its children UI controls.



Drags the red icon over target control to get information

☑ Find Form automatically

☑ Find children

```
⊟ Window - FrmMain
    Hyperlink - LnkAbout
    Button - BtnOpenLogFile
    Button - BtnClose
    Document - TxtConnectionStatus_RemoteHost
    Edit - TxtConnectionStatus_ConnectionStatus
  ⊟ Group - GrpMode
      Spinner - NumMode_Server_Port
      Button - BtnMode_Stop
      RadioButton - RdoMode_Server
      Button - BtnMode_Start
      RadioButton - RdoMode_Client
      Edit - TxtMode_Client_IPAddress
      Spinner - NumMode_Client_Port
  ⊟ TabItem - TabItem
    ⊟ Group - GrpSettings_General
        ComboBox - CboSettings_EscapeCharacterForInterprete
        CheckBox - ChkSettings_AlwaysOnTop
    ⊟ Group - GrpSettings_Monitor
        Hyperlink - LnkSettings_CommentColor_Default
        Hyperlink - LnkSettings_CommentColor_Change
        Hyperlink - LnkSettings_ErrorColor_Default
        Hyperlink - LnkSettings_InformationColor_Change
        Hyperlink - LnkSettings_RemoteHostColor_Default
        Hyperlink - LnkSettings_LabSimulatorColor_Default
        Hyperlink - LnkSettings_LabSimulatorColor_Change
        Hyperlink - LnkSettings_InformationColor_Default
        Hyperlink - LnkSettings_RemoteHostColor_Change
        Hyperlink - LnkSettings_ErrorColor_Change
    ⊟ Group - GrpSettings_Server
        Spinner - NumSettings_MaximumConnections
  ⊟ TabItem - TabItem_1
```

Friendly name: FrmMain

Automation ID: frmMain

Control type: ControlType.Window

□ Hidden control

Automation element properties:

Property	Value
Mouse-X	549
Mouse-Y	128
Name	Lab Simulator
ControlType	ControlType.Window
ClassName	WindowsForms10.Window.8.app.0.378734a
BoundingRectangle	427,105,660,690
AutomationID	frmMain
FrameworkID	WinForm
IsEnabled	True
IsControlElement	True
IsContentElement	True
IsKeyboardFocusable	True
HasKeyboardFocus	False

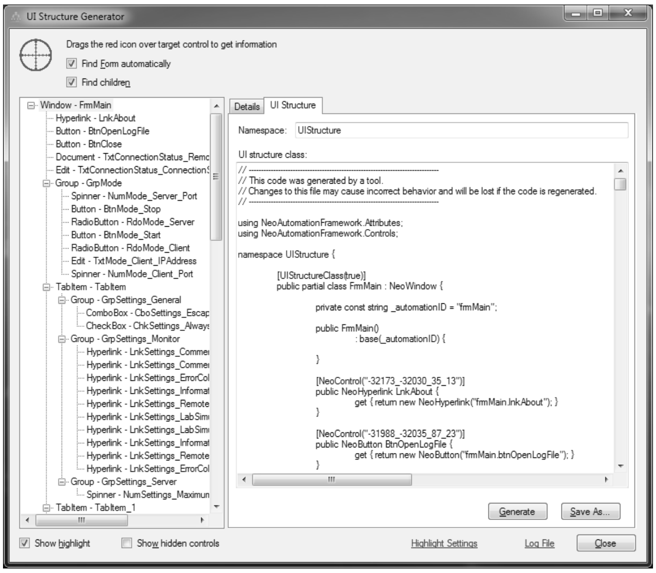
☑ Show highlight □ Show hidden controls

Fig 2. UI structure Generator tool with an example of UI structure

UI Controls that are usable in automated test will be displayed in a hierarchical tree as shown in **Fig 2**. Testers can select each UI control and see where the UI control is and also be able to change its name to be more understandable or descriptive. The original name of each UI control comes from automation ID of the control, therefore sometimes it is not a good name or is a code defined by programmer. If the UI control is not supposed in automated test such as label control, testers can mark it as hidden control so it will not be generated in a UI structure class.

One of useful data retrieved from UI control is automation ID. It is unique under the same parent control since it is used as an identifier. In some cases, UI controls do not have automation ID. NAF has a process to handle such cases. For example tab item control which is an individual tab in tab list control, by default it does not have automation ID, NAF uses name of tab item control instead when finding a target UI control, e.g. running automated test cases to access the tab item.

Once testers have completed reviewing the UI controls, they can simply generate a UI structure and the tool will show a generated UI structure in the preview area and testers are able to save it into a file, as shown in **Fig 3**.


UI Structure Generator
Drags the red icon over target control to get information
☑ Find Form automatically
☑ Find children
Window - FrmMain
Hyperlink - LnkAbout
Button - BtnOpenLogFile
Button - BtnClose
Document - TxtConnectionStatus_Remc
Edit - TxtConnectionStatus_Connection
Group - GrpMode
Spinner - NumMode_Server_Port
Button - BtnMode_Stop
RadioButton - RdoMode_Server
Button - BtnMode_Start
RadioButton - RdoMode_Client
Edit - TxtMode_Client_IPAddress
Spinner - NumMode_Client_Port
TabItem - TabItem
Group - GrpSettings_General
ComboBox - CboSettings_Escap
CheckBox - ChkSettings_Always
Group - GrpSettings_Monitor
Hyperlink - LnkSettings_Commer
Hyperlink - LnkSettings_Commer
Hyperlink - LnkSettings_ErrorCol
Hyperlink - LnkSettings_Informat
Hyperlink - LnkSettings_Remote
Hyperlink - LnkSettings_LabSim
Hyperlink - LnkSettings_LabSim
Hyperlink - LnkSettings_Informat
Hyperlink - LnkSettings_Remote
Hyperlink - LnkSettings_ErrorCol
Group - GrpSettings_Server
Spinner - NumSettings_Maximun
TabItem - TabItem_1
Details
UI Structure
Generate
Save As...
☑ Show highlight
□ Show hidden controls
Highlight Settings
Log File
Close

Namespace: UIStructure

UI structure class:

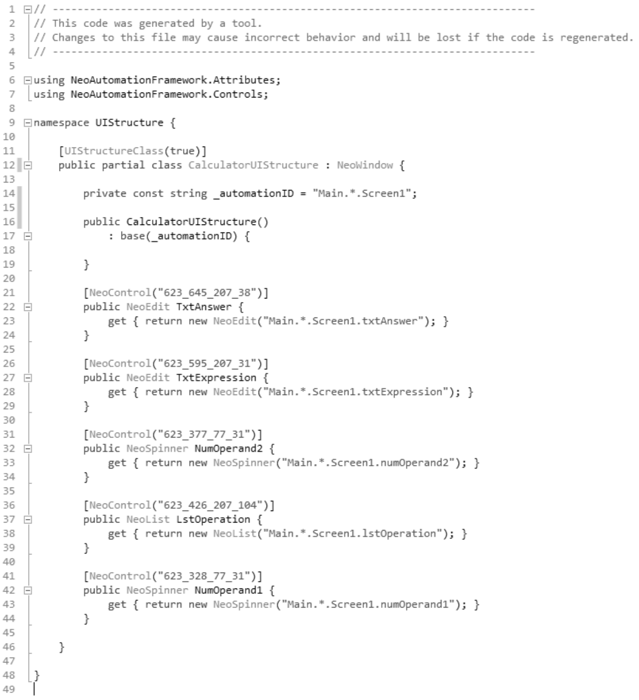
```
// ------------------------------------------------------------------
// This code was generated by a tool.
// Changes to this file may cause incorrect behavior and will be lost if the code is regenerated.
// ------------------------------------------------------------------

using NeoAutomationFramework.Attributes;
using NeoAutomationFramework.Controls;

namespace UIStructure {

        [UIStructureClass(true)]
        public partial class FrmMain : NeoWindow {

                private const string _automationID = "frmMain";

                public FrmMain()
                        : base(_automationID) {

                }

                [NeoControl("-32173_-32030_35_13")]
                public NeoHyperlink LnkAbout {
                        get { return new NeoHyperlink("frmMain.lnkAbout"); }
                }

                [NeoControl("-31988_-32035_87_23")]
                public NeoButton BtnOpenLogFile {
                        get { return new NeoButton("frmMain.btnOpenLogFile"); }
                }
```

Fig 3. UI structure Generator tool with an example of generated UI structure class

C. UI Structure

UI structure contains list of UI controls in hierarchical data structure recorded in format of C# language. Each form in application under test is extracted and recorded in a separate class.

A form will be generated as a class and its children controls will be generated as properties with appropriated type of control. The class has a variable storing automation ID value of the form. Also property members have automation ID values in a get method. An example of UI structure class is shown in **Fig 4**.



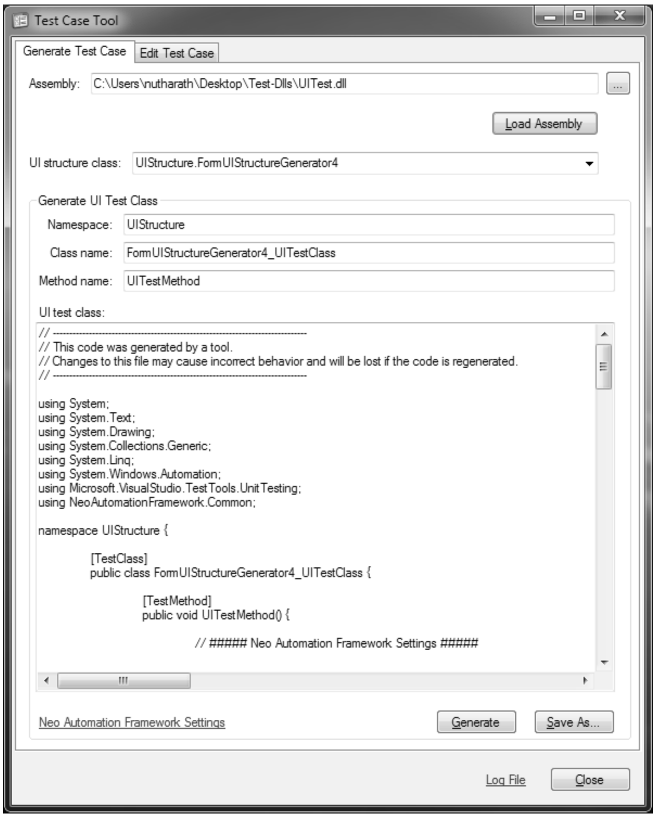
```
// ----------------------------------------------------------------------------
// This code was generated by a tool.
// Changes to this file may cause incorrect behavior and will be lost if the code is regenerated.
// ----------------------------------------------------------------------------

using NeoAutomationFramework.Attributes;
using NeoAutomationFramework.Controls;

namespace UIStructure {

    [UIStructureClass(true)]
    public partial class CalculatorUIStructure : NeoWindow {

        private const string _automationID = "Main.*.Screen1";

        public CalculatorUIStructure()
            : base(_automationID) {

        }

        [NeoControl("623_645_207_38")]
        public NeoEdit TxtAnswer {
            get { return new NeoEdit("Main.*.Screen1.txtAnswer"); }
        }

        [NeoControl("623_595_207_31")]
        public NeoEdit TxtExpression {
            get { return new NeoEdit("Main.*.Screen1.txtExpression"); }
        }

        [NeoControl("623_377_77_31")]
        public NeoSpinner NumOperand2 {
            get { return new NeoSpinner("Main.*.Screen1.numOperand2"); }
        }

        [NeoControl("623_426_207_104")]
        public NeoList LstOperation {
            get { return new NeoList("Main.*.Screen1.lstOperation"); }
        }

        [NeoControl("623_328_77_31")]
        public NeoSpinner NumOperand1 {
            get { return new NeoSpinner("Main.*.Screen1.numOperand1"); }
        }

    }

}
```

Fig 4. An example of UI structure class

D. Test Case Generator

Once testers have the UI structure class they should add it into the test project and compile it into an assembly file such as DLL.

Test Case Generator is a tool that analyzes the assembly which contains the UI structure class and automatically generates automated test cases based on the given UI structure class. All UI controls in the given UI structure class will be used at least once in a sequence of Top-Down and then Left to Right based on position of UI control. This is to ensure all UI controls have been accessed in order to ensure UI controls of application have been tested. However, the generated test cases by accessing UI controls in such sequence may not be valid test cases. Testers can modify test cases manually or use the editor tool which will be described below. **Fig 5** shows our Test Case Generator tool and an example of generated test case in the tool.



Assembly: C:\Users\nutharath\Desktop\Test-Dlls\UITest.dll

UI structure class: UIStructure.FormUIStructureGenerator4

Namespace: UIStructure

Class name: FormUIStructureGenerator4_UITestClass

Method name: UITestMethod

UI test class:

```
// ------------------------------------------------------------------
// This code was generated by a tool.
// Changes to this file may cause incorrect behavior and will be lost if the code is regenerated.
// ------------------------------------------------------------------

using System;
using System.Text;
using System.Drawing;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Automation;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using NeoAutomationFramework.Common;

namespace UIStructure {

        [TestClass]
        public class FormUIStructureGenerator4_UITestClass {

                [TestMethod]
                public void UITestMethod() {

                        // ##### Neo Automation Framework Settings #####
```

Fig 5. Test Case Generator tool and an example of generated test case

Test case generation consists of 3 parts.

1. Configuring NAF settings

This part contains commands to configure setting values of NAF such as Show highlight of target UI control, Use caching of UI control, and Duration to delay for searching UI control or after action performed.

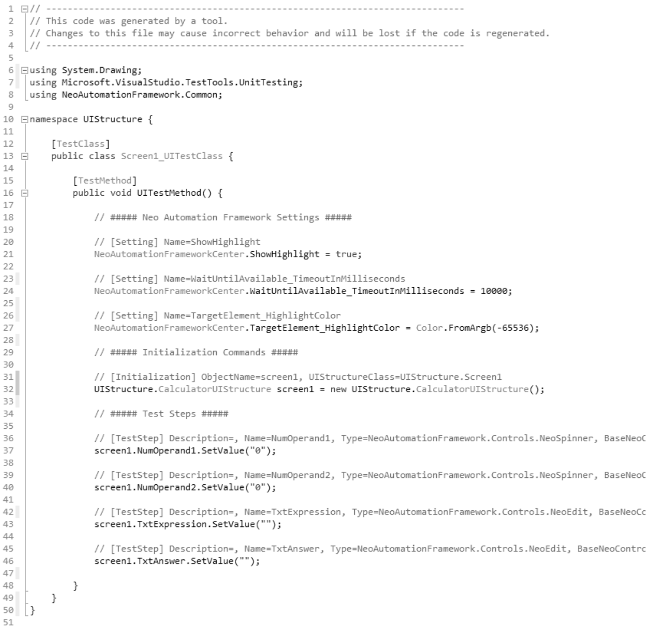
2. Initializing variable of target form

This part creates and instantiates a variable of target form so it can be called by later test steps.

3. Test steps

Actions to be performed on each UI control are included in the test steps. If the UI control has multiple actions, the commonly used action will be selected. For example, checkbox control has Toggle, GetState actions; the default action is Toggle.

After a test case is generated, it can be saved into a new file in C# format.



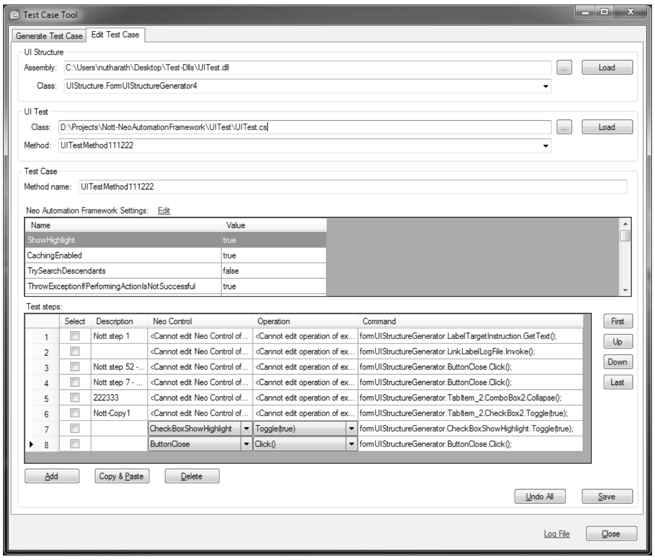
```
// ----------------------------------------------------------------------------
// This code was generated by a tool.
// Changes to this file may cause incorrect behavior and will be lost if the code is regenerated.
// ----------------------------------------------------------------------------

using System.Drawing;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using NeoAutomationFramework.Common;

namespace UIStructure {

    [TestClass]
    public class Screen1_UITestClass {

        [TestMethod]
        public void UITestMethod() {

            // ##### Neo Automation Framework Settings #####

            // [Setting] Name=ShowHighlight
            NeoAutomationFrameworkCenter.ShowHighlight = true;

            // [Setting] Name=WaitUntilAvailable_TimeoutInMilliseconds
            NeoAutomationFrameworkCenter.WaitUntilAvailable_TimeoutInMilliseconds = 10000;

            // [Setting] Name=TargetElement_HighlightColor
            NeoAutomationFrameworkCenter.TargetElement_HighlightColor = Color.FromArgb(-65536);

            // ##### Initialization Commands #####

            // [Initialization] ObjectName=screen1, UIStructureClass=UIStructure.Screen1
            UIStructure.CalculatorUIStructure screen1 = new UIStructure.CalculatorUIStructure();

            // ##### Test Steps #####

            // [TestStep] Description=, Name=NumOperand1, Type=NeoAutomationFramework.Controls.NeoSpinner, BaseNeoC
            screen1.NumOperand1.SetValue("0");

            // [TestStep] Description=, Name=NumOperand2, Type=NeoAutomationFramework.Controls.NeoSpinner, BaseNeoC
            screen1.NumOperand2.SetValue("0");

            // [TestStep] Description=, Name=TxtExpression, Type=NeoAutomationFramework.Controls.NeoEdit, BaseNeoCc
            screen1.TxtExpression.SetValue("");

            // [TestStep] Description=, Name=TxtAnswer, Type=NeoAutomationFramework.Controls.NeoEdit, BaseNeoContrc
            screen1.TxtAnswer.SetValue("");

        }
    }
}

```

Fig 6. An example of a generated test case

Fig 6 shows an example of a generated test case.

E. Test Case Editor

Test Case Editor is a tool that provides basic operations to modify test cases so testers can update test cases according to test design. Testers can add a new test step, remove a test step or change order of a test step as they want to follow a real user scenario. **Fig 7** shows the Test Case Editor tool and an example of a test case being edited.



UI Structure
Assembly: C:\Users\nutharath\Desktop\Test-Dlls\UITest.dll
Class: UIStructure.FormUIStructureGenerator4

UI Test
Class: D:\Projects\Nott-NeoAutomationFramework\UITest\UITest.cs
Method: UITestMethod111222

Test Case
Method name: UITestMethod111222

Neo Automation Framework Settings: Edit

Name	Value
ShowHighlight	true
CachingEnabled	true
TrySearchDescendants	false
ThrowExceptionIfPerformingActionIsNotSuccessful	true

Test steps:

	Select	Description	Neo Control	Operation	Command
1	□	Nott step 1	<Cannot edit Neo Control of...	<Cannot edit operation of ex...	formUIStructureGenerator.LabelTargetInstruction.GetText();
2	□		<Cannot edit Neo Control of...	<Cannot edit operation of ex...	formUIStructureGenerator.LinkLabelLogFile.Invoke();
3	□	Nott step 52 -...	<Cannot edit Neo Control of...	<Cannot edit operation of ex...	formUIStructureGenerator.ButtonClose.Click();
4	□	Nott step 7 - ...	<Cannot edit Neo Control of...	<Cannot edit operation of ex...	formUIStructureGenerator.ButtonClose.Click();
5	□	222333	<Cannot edit Neo Control of...	<Cannot edit operation of ex...	formUIStructureGenerator.TabItem_2.ComboBox2.Collapse();
6	□	Nott-Copy1	<Cannot edit Neo Control of...	<Cannot edit operation of ex...	formUIStructureGenerator.TabItem_2.CheckBox2.Toggle(true);
7	□		CheckBoxShowHighlight	Toggle(true)	formUIStructureGenerator.CheckBoxShowHighlight.Toggle(true);
▸ 8	□		ButtonClose	Click()	formUIStructureGenerator.ButtonClose.Click();

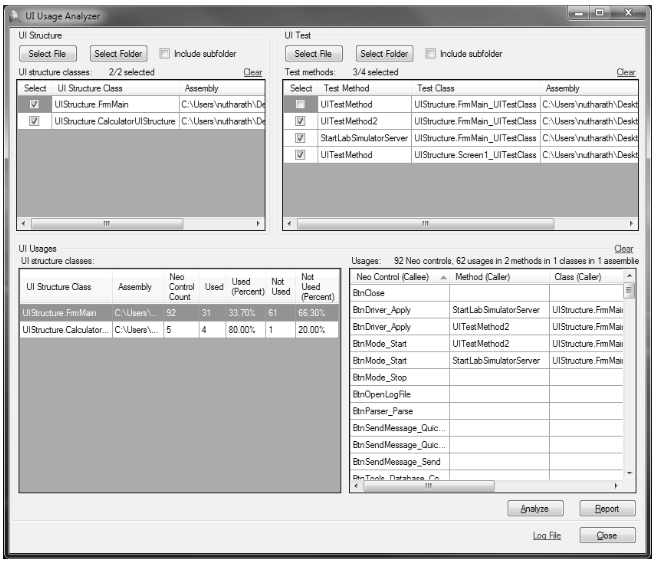
Fig 7. Test Case Editor tool

F. UI Usage Analyzer

UI Usage Analyzer is a tool that analyzes usage of UI controls in automated test cases. The tool requires 2 inputs in order to analyze UI usage: assembly of UI structure class and assembly of test cases.

A technique used to find usage of UI control in test cases is code refactoring. All UI controls in a specified UI structure will be listed out and then iterates a process of finding usage for each UI control one by one. UI control can be called by a test method directly or other shared method. For example, testers may create a common method for the login screen so each test method calls to the shared method instead of having steps perform in the login screen. Therefore, the result of UI control usage will be from traversing through all call chains in the application.

The result shows number of UI controls in the UI structure and number of UI controls which are used and not used in a table. Moreover, it can point out which UI controls are not used so testers can consider adding more tests to increase coverage in their testing. **Fig 8** shows the UI Usage Analyzer tool and the UI usage result from UI structures and test cases.



UI structure classes: 2/2 selected

Select	UI Structure Class	Assembly
☑	UIStructure.FrmMain	C:\Users\nutharath\De
☑	UIStructure.CalculatorUIStructure	C:\Users\nutharath\De

Test methods: 3/4 selected

Select	Test Method	Test Class	Assembly
□	UITestMethod	UIStructure.FrmMain_UITestClass	C:\Users\nutharath\Deskt
☑	UITestMethod2	UIStructure.FrmMain_UITestClass	C:\Users\nutharath\Deskt
☑	StartLabSimulatorServer	UIStructure.FrmMain_UITestClass	C:\Users\nutharath\Deskt
☑	UITestMethod	UIStructure.Screen1_UITestClass	C:\Users\nutharath\Deskt

UI Usages

UI structure classes:

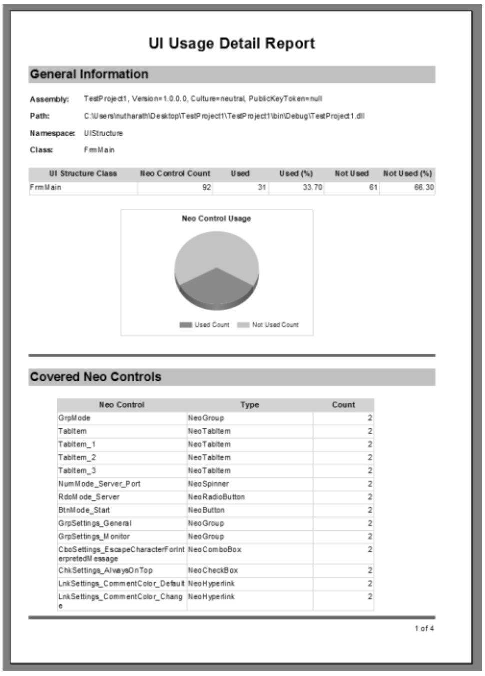
UI Structure Class	Assembly	Neo Control Count	Used	Used (Percent)	Not Used	Not Used (Percent)
UIStructure.FrmMain	C:\Users\...	92	31	33.70%	61	66.30%
UIStructure.Calculator...	C:\Users\...	5	4	80.00%	1	20.00%

Usages: 92 Neo controls, 62 usages in 2 methods in 1 classes in 1 assemblie

Neo Control (Callee)	Method (Caller)	Class (Caller)
BtnClose		
BtnDriver_Apply	StartLabSimulatorServer	UIStructure.FrmMai
BtnDriver_Apply	UITestMethod2	UIStructure.FrmMai
BtnMode_Start	UITestMethod2	UIStructure.FrmMai
BtnMode_Start	StartLabSimulatorServer	UIStructure.FrmMai
BtnMode_Stop		
BtnOpenLogFile		
BtnParser_Parse		
BtnSendMessage_Quic...		
BtnSendMessage_Quic...		
BtnSendMessage_Send		
BtnTools_Database_Co		

Fig 8. UI Usage Analyzer tool

The result can also be shown in a printable format as a usage report as shown in **Fig 9** and **Fig 10**.



UI Usage Detail Report

General Information

Assembly: TestProject1, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
Path: C:\Users\nutharath\Desktop\TestProject1\TestProject1\bin\Debug\TestProject1.dll
Namespace: UIStructure
Class: FrmMain

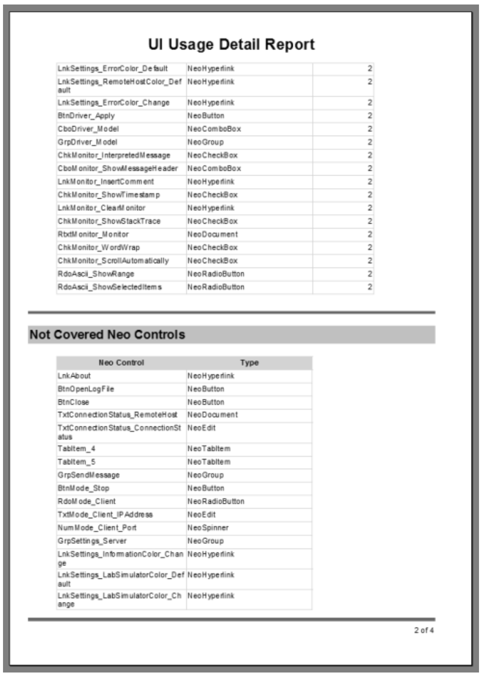
UI Structure Class	Neo Control Count	Used	Used (%)	Not Used	Not Used (%)
FrmMain	92	31	33.70	61	66.30

Covered Neo Controls

Neo Control	Type	Count
GrpMode	NeoGroup	2
TabItem	NeoTabItem	2
TabItem_1	NeoTabItem	2
TabItem_2	NeoTabItem	2
TabItem_3	NeoTabItem	2
NumMode_Server_Port	NeoSpinner	2
RdoMode_Server	NeoRadioButton	2
BtnMode_Start	NeoButton	2
GrpSettings_General	NeoGroup	2
GrpSettings_Monitor	NeoGroup	2
CboSettings_EscapeCharacterForInterpretedMessage	NeoComboBox	2
ChkSettings_AlwaysOnTop	NeoCheckBox	2
LnkSettings_CommentColor_Default	NeoHyperlink	2
LnkSettings_CommentColor_Change	NeoHyperlink	2

1 of 4

Fig 9. An example of UI usage report



UI Usage Detail Report

LnkSettings_ErrorColor_Default	NeoHyperlink	2
LnkSettings_RemoteHostColor_Default	NeoHyperlink	2
LnkSettings_ErrorColor_Change	NeoHyperlink	2
BtnDriver_Apply	NeoButton	2
CboDriver_Model	NeoComboBox	2
GrpDriver_Model	NeoGroup	2
ChkMonitor_InterpretedMessage	NeoCheckBox	2
CboMonitor_ShowMessageHeader	NeoComboBox	2
LnkMonitor_InsertComment	NeoHyperlink	2
ChkMonitor_ShowTimestamp	NeoCheckBox	2
LnkMonitor_ClearMonitor	NeoHyperlink	2
ChkMonitor_ShowStackTrace	NeoCheckBox	2
RtxtMonitor_Monitor	NeoDocument	2
ChkMonitor_WordWrap	NeoCheckBox	2
ChkMonitor_ScrollAutomatically	NeoCheckBox	2
RdoAscii_ShowRange	NeoRadioButton	2
RdoAscii_ShowSelectedItems	NeoRadioButton	2

Not Covered Neo Controls

Neo Control	Type
LnkAbout	NeoHyperlink
BtnOpenLogFile	NeoButton
BtnClose	NeoButton
TxtConnectionStatus_RemoteHost	NeoDocument
TxtConnectionStatus_ConnectionStatus	NeoEdit
TabItem_4	NeoTabItem
TabItem_5	NeoTabItem
GrpSendMessage	NeoGroup
BtnMode_Stop	NeoButton
RdoMode_Client	NeoRadioButton
TxtMode_Client_IPAddress	NeoEdit
NumMode_Client_Port	NeoSpinner
GrpSettings_Server	NeoGroup
LnkSettings_InformationColor_Change	NeoHyperlink
LnkSettings_LabSimulatorColor_Default	NeoHyperlink
LnkSettings_LabSimulatorColor_Change	NeoHyperlink

2 of 4

Fig 10. An example of unused UI controls in UI usage report

G. Test Manager

Automated test cases we have created in previous steps are in C# language. It can be run by using test command of .NET framework, i.e. mstest.exe, or can be easily run within Microsoft Visual Studio IDE which already provides functionalities of running test and test result report.

IV. Applications

We have tested NAF by creating a few automated test cases for example applications. The process of creating UI structure is done within few minutes and testers can finish writing automated test cases according to test design shortly. However, after testers have finished creating the automated test cases, we found some UI controls that are not used in the test cases. Obviously that UI Usage Analyzer tool could help identify what UI controls should be added into automated test cases to increase test coverage as much as possible.

V. Conclusion

We have presented an automation testing framework – Neo Automation Framework (NAF). NAF consists of three main parts: 1. generating UI structure of specified form in an application under test, 2. generating and modifying automated test cases which is kind of unit test in C# language, and 3. analyzing usage of UI control in automated test cases. Our approach is to keep information of UI controls in hierarchical data structure which can be used in other parts of framework easily. Testers can access UI controls directly via a class of UI structure which is stored in a class. Therefore, there is no extra work for preprocessing of use and it can also be used to verify the automated test cases to find unused UI controls.

Since we have a model of application under test in hierarchical data structure so for our future work, it is possible to validate the UI structure against the software. It could be a quick test to ensure that there is no change in UI. If the result of checking UI structure with the software is failed, then there might be new UI controls added into the software or existing UI controls do not exists anymore. In latter case, if it is not an intended change, then it is likely a bug in the software detected by UI structure.

References

[1] James A. Whittaker, “What Is Software Testing? And Why Is It So Hard?”, IEEE, 2000

[2] Wei Jin et al., "Automated Behavioral Regression Testing", in 3rd Int. Conf. on Software Testing, 2010, IEEE, pp. 137-146

[3] ZHU Xiaochun et al., "A Test Automation Solution on GUI Functional Test", pp. 1413-1418

[4] Pekka Aho; Nadja Menz; Tomi Räty; Ina Schieferdecker, "Automated Java GUI Modeling for Model-Based Testing Purposes", IEEE, 2011

[5] Microsoft, “Visual Studio Test Professional”, Available from: http://www.microsoft.com/visualstudio/eng#products/visual-studio-test-professional-2012+product-edition-testpro

[6] Automated GUI Test Coverage Analysis using GA

[7] Reverse Engineering of GUI Models for Testing

[8] GUI Testing Made Easy

[9] Microsoft, “Control Types and Their Supported Control Patterns”, Available from: http://msdn.microsoft.com/en-us/library/windows/desktop/ee671193(v=vs.85).aspx

[10] Microsoft, “Microsoft UI Automation”, Available from: http://msdn.microsoft.com/en-us/library/ms747327.aspx