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.
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.

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.
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.

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.

```
namespace UIStructure
{
    public partial class calculatstructure : Wndow
    {
        private const string _formName = "Main.\Screen1"
        public calculatstructure() { }
        test_automations() {
        }
        [FormControl("205_425_287_187")]
        public button TestButton()
        { return new Button("Main.\Screen1.TestButton") ; }
        [FormControl("205_505_287_313")]
        public button TextButton()
        { return new Button("Main.\Screen1.TextButton") ; }
        [FormControl("205_507_272_76")]
        public boolean TextIsPresent()
        { return new bool("Main.\Screen1.TextIsPresent") ; }
        [FormControl("205_425_287_194")]
        public boolean TextIsPresent()
        { return new bool("Main.\Screen1.TextIsPresent") ; }
        [FormControl("205_308_77_351")]
        public button TextButton()
        { return new Button("Main.\Screen1.TextButton") ; }
        [FormControl("205_308_77_351")]
        public button TextButton()
        { return new Button("Main.\Screen1.TextButton") ; }
    }
}
```

Fig 4. An example of UI structure class

```
namespace UIStructure
{
    public partial class calculatstructure : Wndow
    {
        private const string _formName = "Main.\Screen1"
        public calculatstructure() { }
        test_automations() {
        }
        [FormControl("205_425_287_187")]
        public button TestButton()
        { return new Button("Main.\Screen1.TestButton") ; }
        [FormControl("205_505_287_313")]
        public button TextButton()
        { return new Button("Main.\Screen1.TextButton") ; }
        [FormControl("205_507_272_76")]
        public boolean TextIsPresent()
        { return new bool("Main.\Screen1.TextIsPresent") ; }
        [FormControl("205_425_287_194")]
        public boolean TextIsPresent()
        { return new bool("Main.\Screen1.TextIsPresent") ; }
        [FormControl("205_308_77_351")]
        public button TextButton()
        { return new Button("Main.\Screen1.TextButton") ; }
        [FormControl("205_308_77_351")]
        public button TextButton()
        { return new Button("Main.\Screen1.TextButton") ; }
    }
}
```

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.
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.

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 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.

Fig 7. Test Case Editor tool

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.

Fig 9. An example of 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

[8] GUI Testing Made Easy