International Journal of Scientific Engineering and Applied Science (IJSEAS) – Volume-2, Issue-4,April 2016
ISSN: 2395-3470
www.ijseas.com

Automation Testing Using Coded UI Test

[1]NIRANJANA S, [2]BALAMURUGAN A

[1] PG Scholar, Department of Computer Science and Engineering , Sri Krishna College of Technology Coimbatore ,Tamil Nadu
[2] Professor, Department of Computer Science and Engineering , Sri Krishna College of Technology Coimbatore ,Tamil Nadu

Abstract - Coded UI Test (CUIT) is a relatively new automation tool in the software market. It was made available as part of the Visual Studio 2010 update. The produce has undergone a lot of enhancements and its new version has been released as part of Visual Studio 2013. Software code can be easily reviewed and debugged in Visual Studio. It also has an IntelliSense code achievement feature, which helps in generating code faster. Coded UI mechanization tool is supported by high level programming languages such as C# and Visual Basic .NET.
Software testers respect the high level language maintain offered by Coded UI automation tool. It is a known fact that software testers prefer learning VB Script (used by QTP [presently known as UFT] and Test Complete automation tools), as it is easy compared to other languages such as Java, C# or VB.NET. Software testers who have knowledge of inscription code in object-oriented programming language (OOPL) especially like better using Coded UI automation tool.

Keywords – *QTP, UFT, OOPL, Coded UI test*

1. INTRODUCTION

Automated tests that drive your application through its user interface (UI) [1] are known as coded UI tests (CUITs). These tests include functional testing of the UI controls. They let you verify that the whole application, including its user interface, is functioning correctly. Coded UI Tests are particularly functional where there is validation or other logic in the user interface for example in a web page [2]. They are also commonly used to automate an breathing
Physical test.

A typical development experience might be one where, to begin with, you simply build your submission (F5) and click through the UI gearstick to verify that things are working correctly. You then might decide to create a coded test so that you don't need to continue to test the application manually. Depending on the particular functionality being tested in your application, you can put in writing code for either a purposeful test, or for an integration test that might or might not include trying at the UI level. If you simply want to directly access some business logic, you might code a unit test. However, under convinced circumstances, it can be beneficial to comprise testing of the various UI controls in your application. A coded UI test can automate the initial (F5) scenario, verifying that code churn does not impact the functionality of your application. Creating a coded UI test [5] is easy. You simply perform the test manually while the CUIT Test Builder runs in the background. You can also specify what values should appear in specific fields. The CUIT Test Builder records your actions and generates code from them. After the test is created, you can edit it in a specialized editor that lets you change the progression of actions.
Alternatively, a test case that can be recorded in Microsoft Test Manager (MTM), the code can be produced from MTM.

2. PURPOSE OF CODED UI TESTING

The specialized CUIT Test Builder and editor make it easy to create and edit coded UI tests even if your main skills are concentrated in testing slightly than coding. But if you are a developer and you want to make longer the test in a more advanced technique, the code is structured so that it is straightforward to copy and adapt.

2.1 Why Coded UI is a Smart Choice in Test Automation

The robust capabilities of Visual Studio and Team Foundation Server (TFS) have made them hot favourites among software developers. Developers utilize both these tools to produce better-quality software applications [10]. The collective usage of TFS, Visual Studio and its testing tools augments the process of agile development. Here are few reasons why Coded UI tool is a preferred choice for software testers:

1. Software testers and Developers can work using the same tools/language, which enables them to collaborate effectively.
2. The Coded UI automation tool chains both web and windows projects, as both C# and VB.NET are known for their robustness.
3. The element identification mechanism is a powerful feature in Coded UI.
4. Coded UI strongly supports Synchronization. The Playback Engine supports features such as 'Wait For Ready Level', 'Wait For Control Exist' etc., it makes the test implementation stop till UI Threads or All Threads are ready.
5. Automation tests can be run on remote machines with the help of 'Tests Agents'.
6. Coded UI supports AJAX controls.
7. Descriptive Programming is another impressive feature supported by Coded UI tool, which allows software testers to automate scenarios based on object properties. There's no need to wait for the user interface to record/assert scenarios.
8. Coded UI allows developing an extensive test suite and performing tests in local environments.
9. Using Coded UI with layered framework, automation teams can develop sophisticated tests.
10. Utilizing Log4net.dll, software testers can log the results and capture exceptions in an effective manner.

2.1.1 How to Begin with Coded UI Testing

Creating a coded UI test generates a UIMap object that is specific to your test and represents the windows, controls, parameters, and assertions that are in the UI or that you created during the test recording. You can then perform actions on these UI objects to automate your user interface. For example, you can have your test method click a hyperlink in a Web application, type a value in a text box, or branch off and take different testing actions based on a value in a field.

A coded UI test class is identified by a class. Each coded UI test is a test method in a coded UI test class. You can add multiple test methods to each coded UI test class and identify each coded UI test method by using the Test Method Attribute.

Your test method can also add validation code for a UI test control to obtain the value of a property of a UI test control. The test method can use an Assert statement to compare the actual value of the property to an expected value. The result of this comparison determines the outcome of the test. Every time that you run a coded UI test, you can analyze the test result and if the test fails, you can see or store the details of which assertion failed.

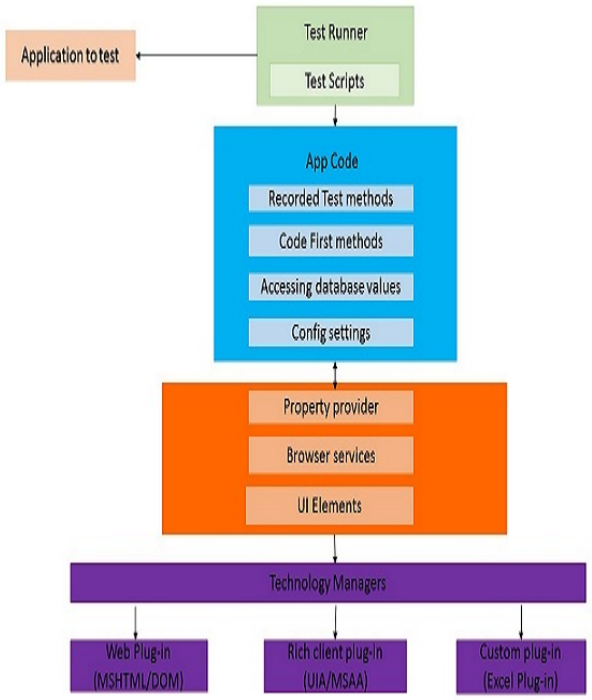
2.1.1.1 Software Requirements

Software testers can use either Visual Studio Premium or Visual Studio Ultimate to create Coded UI tests. Visit the MSDN website to obtain more information on supported configurations and platforms.

3. BENEFITS OF CODED UI

3.1. Coded UI Framework

The below illustration depicts the Coded UI Framework



Let's scrutinize the above framework illustration to interpret it effectively. As depicted in the above illustration, the App code folder contains all the reusable functions, which are used to write test scripts. Let's assume that a software tester is automating a Gmail application, the Login, Home page and Logout code are written in different functions such as Login(), Homepage() and Logout() under 'Gmail' class. While writing the test scripts, software testers can reuse the above functions. Here's the code to achieve the above functionality.

3.2. Improving Performance of Coded UI Test

Automated tests that drive your application through its user interface (UI) are known as coded UI tests (CUITs). These tests include functional testing of the UI controls. They let you verify that the whole application including its user interface is functioning correctly [6]. Coded UI Tests are particularly useful where there is validation or other logic in the user interface for example in a web page. They are also recurrently used to automate an existing manual test.

3.3. Advantages

- Everything seems very stable.
- User can test many different kinds of user interfaces, not just the web.
- Frequently recurring test steps can easily be re-used. It's up to user how fine-grained user wants to record user test steps. Calling a step (consisting of one or more actions) is just a matter of a single function call.
- Generates C# and XML code by default.
- Features fuzzy matching of UI elements. This seems to make the tests more robust when updating the user interface.

3.4. Disadvantages

- Even very simple tests generate a lot of code.
- Test steps are stored in an XML file (called UIMap) which in turn compiles to C# code. The UIMap is big and clunky and there currently is no editor or documentation for it. So if user wants to make some changes, things can get complicated unless user wants to re-record an entire test step.
- Creating very simple assertions (such as "look for the string "foo" on this web page") is a bit clumsy and requires too many mouse clicks. An IE accelerator would be great!
- No support for Safari, Opera or Chrome. On the bright side, Coded UI has an open interface, so anyone could implement it.
- While fuzzy matching works great in most cases, It can get in the way in others.
- Not quite as intuitive as Selenium at the beginning.
- Tests record and run quite a bit slower than with Selenium.

4. Coded UI Supported Technologies

Using Coded UI, one can easily test an application which has *UI* (User Interface). The application can be of Web based or Windows based, coded UI supports them.

4.1. UI technologies

The following are the **UI** technologies that supports Coded UI

1. Windows based Desktop Applications (Windows Forms etc)
2. WPF (Windows Presentation Foundation)
3. Web Applications (Html, Silver light, Jscript, HTML 5)
4. Web Services (SOAP, ASPX etc)
5. Windows Phone applications (Available from Visual Studio 2013 Update 2 or later***)***
6. Windows Store applications

4.2. Other tools

The following are the Comparison of Coded UI with other tools available for automation testing. It's very obvious here that,

1. Selenium does not support any Windows applications as well as WPF applications.
2. QTP does support Windows application, but requires .Net Plug-in to be purchased and installed separately
3. Test complete does support Windows application, but requires plug-in to be installed
4. No other tool, other than Visual Studio Coded UI support Windows phone application testing
5. No other tool, other than Visual Studio Coded UI support Windows store application testing

The comparison is just some of the most commonly used popular tools against Visual Studio Coded UI testing.

4.3 Coded UI Application Testing feature

Coded UI has two different ways feature to test your applications

1. *Coded UI Record and Playback*
2. *Coded UI Hand coding* (which will involve intense coding)

Coded UI record and playback feature allows to record the actions (i.e.) mouse click events in the UIMap Designer and plays the recorded steps

Coded UI hand coding allows to code the recording manually in C# language.

The way Coded UI automation is designed by Microsoft is to target both the *non-programmers* as well as *programmers* to work with Coded UI.

Microsoft Test Manager (**MTM**) uses Coded UI Record and Playback feature to record all the user action and save it as a test case, which can then be used to replay while executing the test case.

Coded UI Hand coding is used prevalently while designing custom frameworks while working with larger projects.

5 CONCLUSION

The sole purpose of Coded UI is to perform automatic functional testing that doesn't need human interactions. We can have a list of Test Cases managed in TFS and we can corroboration User Action while organization a Test Case using MTM (Microsoft Test Manager). These automated recorded actions can also be repeated any number of time after we find a change in build. Converting to a Coded UI test does not directly add the checks, but Coded UI allows automated checks to be added into the test. Having converted an MTM test to a Coded UI test, the cross-hairs tool of Coded UI (also called the declaration tool and similar terms) can be used to add assertions that values on the screen include the expected values.

6. REFERENCES

[1] Xun Yuan, Myra B. Cohen, Atif M. Memon, (2010) "GUI Interaction Testing: Incorporating Event Context", IEEE Transactions on Software Engineering, vol. 99.

[2] A. M. Memon, M. E. Pollack, and M. L. Soffa, (2001) "Hierarchical GUI test case generation using automated

planning", IEEE Transactions on Software Engineering, Vol. 27, no. 2, pp. 144- 155.

[3] X. Yuan and A. M. Memon, (2007) "Using GUI run-time state as feedback to generate test cases", in International Conference on Software Engineering (ICSE), pp. 396-405.

[4] X. Yuan, M. Cohen, and A. M. Memon, (2007) "Covering array sampling of input event sequences for automated GUI testing", in International Conference on Automated Software Engineering (ASE), pp. 405-408.

[5] X. Yuan, M. Cohen, and A. M. Memon, (2009) "Towards dynamic adaptive automated test generation for graphical user interfaces", in First International Workshop on TESTing Techniques & Experimentation Benchmarks for Event-Driven Software (TESTBEDS), pp. 1-4.

[6] Si Huang, Myra Cohen, and Atif M. Memon, (2010) "Repairing GUI Test Suites Using a Genetic Algorithm, "in Proceedings of the 3rd IEEE International Conference on Software Testing Verification and Validation (ICST).

[7] P. Brooks, B. Robinson, and A. M. Memon, (2009) "An initial characterization of industrial nd graphical user interface systems", in ICST 2009: Proceedings of the 2 IEEE International Conference on Software Testing, Verification and Validation, Washington, DC, USA: IEEE Computer Society.

[8] Q. Xie, and A.M. Memon (2006) "Model-based testing of community driven open-source GUI applications", in International Conference on Software Maintenance (ICSM), pp. 145-154.

[9] Q. Xie and A. M. Memon, (2005) "Rapid "crash testing" for continuously evolving GUI- based software applications", in International Conference on Software Maintenance (ICSM), pp. 473- 482.

[10] A. M. Memon and Q. Xie, (2005) "Studying the fault-detection effectiveness of GUI test cases for rapidly evolving software", IEEE Transactions on Software Engineering, vol. 31, no. 10, pp. 884- 896.