Wahlfach Software-Qualität

GUI-Testing unter C#/.NET

Mit den Schwerpunkten WPF und IcuTest

Tobias Müller
08.02.2011

Inhaltsverzeichnis

Abbildungsverzeichnis

Listingverzeichnis

1 TestAutomationFX

1.1 Homepage
http://www.testautomationfx.com/

1.2 Lizenz
Freie Lizenz: Begrenzt auf eine bestimmte Anzahl von Aufnahmen und Wiedergaben

Verschiedene kommerzielle Lizenzen: Keine Einschränkungen, technischen Support, freie neue Versionsupdates.

1.3 Kurzbeschreibung
Bei TestAutomationFX handelt es sich um ein Werkzeug zur Generierung von Unit-Tests von GUI-Komponenten. Es ist vollständig in Visual Studio integriert. Es handelt sich um kein typisches Capture&Replay-Werkzeug, sondern um ein Werkzeug, welches anhand eigener Benutzerangaben Code generiert. Dieser simuliert daraufhin die vorgenommenen Benutzereingaben. Diese Benutzereingaben werden zu einem Test zusammengefasst und lassen und lassen sich mit Verifikationen ergänzen, welche jUnit-Asserts gleich kommen.

2 TestComplete

2.1 Homepage
http://www.automatedqa.com/products/testcomplete/

2.2 Lizenz
Frei und kommerziell erhältlich.

2.3 Kurzbeschreibung
Dies ist ein Tool zum automatisierten Testen von GUI-Applikationen. Es bietet Capture&Replay-Möglichkeiten an. Test-Szenarios lassen sich mit Scripts automatisieren. Ergebnisse können validiert und geloggt werden. Dieses Werkzeug ist nicht nur für C#/.NET, sondern auch für viele andere Technologien konzipiert.

3 Ranorex

3.1 Homepage
http://www.ranorex.com/

3.2 Lizenz
Freie 30-Tage Testversion und kommerzielle Version erhältlich

3.3 Kurzbeschreibung
Ranorex ermöglicht das Schreiben von Tests ohne Scriptsprache. Somit können GUI-Elemente direkt in C# erkannt und angesprochen werden. Zusätzlich besteht die Möglichkeit mittels Interaktion mit der GUI diesen Code generieren zu lassen. Ranorex besitzt eine eigene Entwicklungsumgebung.

Ranorex steht nicht nur für den .NET-Bereich, wie WPF, Windows Forms oder Silverlight, sondern für eine große Bandbreite an Technologien zur Verfügung.

4 White

4.1 Homepage
http://white.codeplex.com/

4.2 Lizenz
Apache License Version 2.0 vom Januar 2004

4.3 Letzter Untersuchungszeitpunkt
21.01.2011

4.4 Kurzbeschreibung
White ermöglicht das Ermitteln und Ansprechen von GUI-Elementen aus Anwendungen, wie zum Beispiel WPF-Applikationen. Dabei werden die Eigenheiten der Instanziierung einer WPF-Anwendung durch das Framework übernommen, sodass GUI-Elemente einfach instanziiert und angesprochen werden können. Interaktionen mit den GUI-Elementen ober Überprüfungen auf deren Inhalte sind ebenfalls möglich.

5 NunitForms

5.1 Homepage
http://nunitforms.sourceforge.net/

5.2 Lizenz
BSD-Lizenz

5.3 Kurzbeschreibung
NUnitForms ermöglicht das Testen von GUI-Anwendungen mit Unit Forms. Dabei sind typische Dinge, wie das Ansprechen und Interagieren mit GUI-Elementen möglich. Inhalte der GUI können ebenfalls überprüft werden.

6 SilkTest® 2010

6.1 Homepage
http://www.borland.com/us/products/silk/silktest/index.html

6.2 Lizenz
Kommerziell

6.3 Kurzbeschreibung
Ermöglicht das Testen von GUIs in verschiedenen Technologien. Unter anderem werden auch die .NET-Technologien Windows Forms und WPF von SilkTest unterstützt.

7 IcuTest

7.1 Homepage
Projektseite: http://icutest.com/icu/Default.aspx

Dokumentation: http://www.nxs7.net/icu/docs/

7.2 Lizenz
Freier Download nach Angabe der eigenen Email-Adresse möglich.

7.3 Untersuchte Version
unbekannt

7.4 Letzter Untersuchungszeitpunkt
21.01.2011

7.5 Kurzbeschreibung
Mit diesem Tool lassen sich GUI-Elemente der WPF einfach ansprechen. Dadurch können unter anderem Nutzereingaben simuliert werden. Das Werkzeug stellt jedoch ansonsten kein vollständiges Testwerkzeug dar, sondern lässt sich als Aufsatz auf MSTest, NUnit, XUnit oder MbUnit verwenden. Somit sind die Mechanismen von IcuTest in einem Unit-Test eines der genannten Frameworks nutzbar.

Zusätzlich verfügt dieses Tool über eine weitere nützliche Funktion, welche nicht ganz gewöhnlich ist. Diese wird im Folgenden erläutert.

Die werkzeuginternen Tests basieren auf Bitmap-Vergleichen. So lassen sich bestimmte visuelle Situationen der GUI als richtige Version abspeichern. Wenn nun ein Test diese Situation überprüft, wird der aktuelle Status der GUI auf Bitmap-Ebene mit der als richtig eingestuften Version vergleichen. Wenn keine Unterschiede festgestellt werden, besteht der Test den Durchlauf. Sollten Unterschiede festgestellt werden, so kann der Tester selber entscheiden, wie damit verfahren werden soll. Für diese Entscheidung stehen ihm drei Möglichkeiten zur Auswahl:

- Pass: Dies lässt die derzeitige Aufnahme als gültig zu und lässt den Test bestehen. Weitere Tests, die gleich ausfallen, werden ebenfalls als gültig erkannt. Dies kann bei kleinen Differenzen, die jedoch nicht die Funktionalität beeinflussen, nützlich sein. Lässt jedoch die alte als richtig erkannte Version im weiteren Verlauf fehlerhaft werden
- Skip: Überspringt den Test für diese Ausführung.
- Fail: Lässt den Test für diese Ausführung als nicht bestanden anzeigen. In diesem Fall ist etwas an der getesteten Software fehlerhaft und zu ändern, da die neue Anzeige der GUI vom Tester als nicht richtig eingestuft wurde.

7.6 Fazit
IcuTest scheint ein neues Werkzeug zu sein. Es deckt die für ein GUI-Testwerkzeug notwendigen Kriterien gut ab und ermöglicht zusätzlich das Vergleichen von Bildschirmaufnahmen der GUI.

Ein großer Kritikpunkt ist allerdings die Wartung des Projekts. Es stehen keine FAQs oder ein Forum zur Verfügung. Außerdem ist die Dokumentation nicht sehr umfangreich und ist auf der Projektseite schwer zu finden.

Bevor IcuTest verwendet wird, sollte recherchiert werden, ob sich diese Punkte eventuell seit dem Untersuchungszeitpunkt geändert haben.

7.7 Einsatzgebiete

IcuTest ist durch seine zusätzliche Funktion der Bitmap-Vergleiche hervorragend geeignet, um Gui-lastige Applikationen zu testen, bei welchen es wichtig ist, dass bestimmte Ausschnitte der GUI immer gleich aussehen sollen.

7.8 Einsatzumgebungen

Das Tool benötigt ein Unit-Test Framework und deren Testmethoden in denen es dann GUI-Elemente ansprechen und Bitmap-Vergleiche durchführen kann.

Folgende Testwerkzeuge werden dabei unterstützt:

MSTest, NUnit, XUnit und MbUnit

7.9 Installation

Um IcuTest in eigenen WPF-Projekten zugänglich zu machen, ist die heruntergeladene .exe-Datei zu starten und damit IcuTest an einem beliebigen Ort zu installieren.

Danach ist im Installationsverzeichnis neben einigen Visual Studio Beispielprojekten auch ein Ordner „bin“ zu finden. In diesem Ordner befinden sich neben anderen Dateien die beiden dll-Dateien „IcuTestCore.dll“ und „IcuTesting.WPFd.dll“.

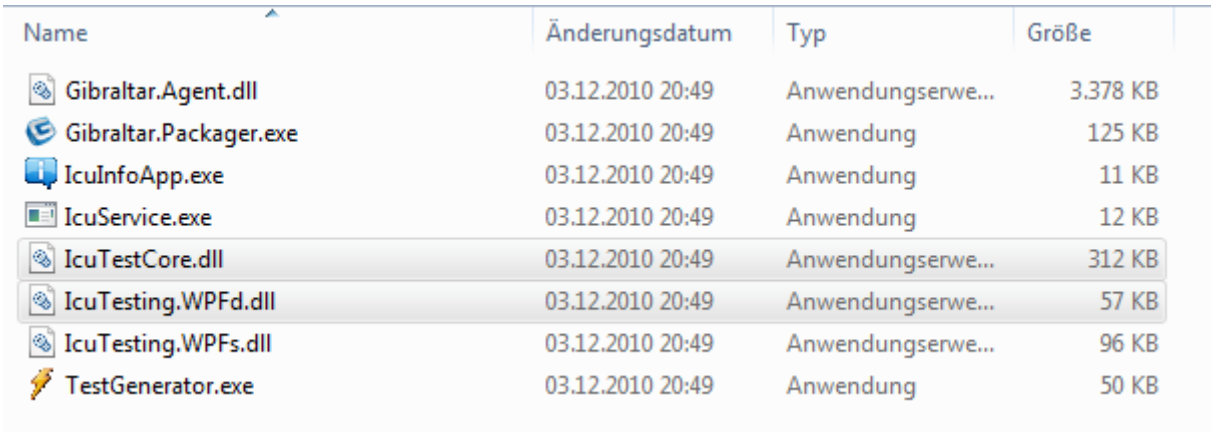

Abbildung 1: Notwendige .dll-Dateien im bin-Verzeichnis von IcuTest

Um IcuTest in eigenen Projekten nutzen zu können, müssen diese beiden dll-Dateien als Verweis hinzugefügt werden.

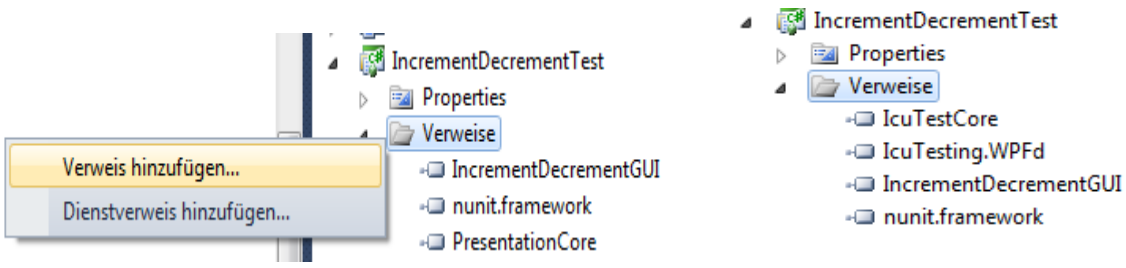

Abbildung 2: Nötige Verweise von IcuTest hinzufügen

7.10 Dokumentation

Der erste Einblick in die Funktionsweise des Werkzeugs ist durch zwei Videos auf der Projektseite gut gegeben.

Für einen Informatiker ohne QS-Erfahrungen[1] ist das Kapitel Quick Start jedoch nicht geeignet, da das Tool auf einem anderen in 7.8: „Einsatzumgebungen“ genannten Test-Werkzeug aufbaut und grundlegende Kenntnisse davon nötig sind.

Mit QS-Erfahrungen bietet die Seite jedoch einen geeigneten Einstieg.

Weitere Beispiele sind auf der Seite des Werkzeugs jedoch nur schwer zu finden. Die Dokumentation ist nicht intuitiv zu finden und bietet lediglich die gleichen Inhalte, die auf Homepage des Tools gezeigt sind und eine ergänzende API-Referenz.

Die installierte Version des Tools liefert jedoch einige Beispiele zur Nutzung von IcuTest mit jedem der genannten Unit-Test Werkzeuge.

7.11 Wartung der Projektseite

Die Seite des Werkzeugs macht einen schlecht gepflegten Eindruck. Es gibt keine News über Fortschritte oder Änderungen. Die Vorführvideos scheinen nicht ganz ausgereift in die Seite eingepflegt zu sein und weisen Fehlverhalten auf. Es wird aus der Projektseite nicht ersichtlich, wie lange es dieses Werkzeug schon gibt und ob es auch in Zukunft gewartet und verbessert wird.

7.12 Nutzergruppen und Support

Als Nutzer ist man sehr auf sich allein gestellt. Es gibt kein Forum und kein FAQ auf der Projektseite. Auch hat man nur eine Möglichkeit, an Support zu kommen. Diese ist mit der Support-Emailadresse der Firma[2] gegeben.

Ein angegebener Twitter-Account der Firma weißt zum Untersuchungszeitpunkt lediglich zwei Einträge auf.

7.13 Intuitive Nutzbarkeit

Die intuitive Nutzbarkeit des Werkzeugs ist gegeben. Trotz knapp gehaltener Einführungsbeispiele und wenig Dokumentationsmaterial kann sich ein erfahrener Entwickler sehr schnell in das Werkzeug einarbeiten.

Die genaue Funktionsweise des Abspeicherns der Bilder für die Bitmap-Vergleiche geht jedoch nicht aus der Dokumentation hervor und muss durch die Ausführung der gegebenen Beispiele oder eigener Beispiele ermittelt werden.

7.14 Automatisierung

Das Werkzeug weißt die gleichen Eigenschaften im Bezug auf Automatisierung auf, wie das mit ihm benutzte Test-Framework.

Die zusätzliche Testmöglichkeit der Bitmap-Vergleiche verläuft jedoch nur so lange automatisiert, bis ein Test fehlschlägt. An dieser Stelle ist eine Wertung dieses Fehlverhaltens durch den Tester notwendig. In Verbindung mit vielen weiteren Tests bremst dies den weiteren Testlauf aus. Sollten GUI-Tests von IcuTest in Kombination mit anderen Tests ausgeführt werden, so sollten die von IcuTest am Anfang oder am Ende ausgeführt und vom Tester beobachtet werden.

[1] Erfahrungen in der Qualitätssicherung
[2] support@nxs7.net

Es gibt auch eine Möglichkeit, die Wertung des Fehlverhaltens automatisiert geschehen zu lassen, sodass falsche Tests als falsch gewertet werden und das Analysieren dieses Fehlverhaltens zu einem späteren Zeitpunkt durchgeführt werden kann.

7.15 Einführendes Beispiel

In diesem Abschnitt befindet sich ein voll funktionsfähiges Beispiel der IcuTest-eigenen Möglichkeit, die GUI auf Korrektheit zu überprüfen[3].

WPF-Komponenten können nicht einfach von einem Unit-Test Werkzeug heraus angesprochen werden, da WPF eine Applikations-Instanz, einen eigenen GUI-Thread und die Synchronisation zwischen den verschiedenen Threads benötigt.

Genau diese Arbeitsschritte übernimmt die Klasse IcuTest, sodass der Tester von den WPF-Interna keine Kenntnisse benötigt.

Die Variable ICU vom Typ IcuTest in Listing 1 ist dabei das zentrale Element. Bei der Erschaffung einer Instanz steht dabei die Methode `IcuFromDir` zur Verfügung. Im Verzeichnis „C:\Testverzeichnis“ werden die Bilder der GUI abgelegt.

An der Variablen ICU kann nun in einem Test die Methode `Invoke` mit einem Delegate[4] aufgerufen werden, welche die in Listing 1 stehende Form aufweist. IcuTest übernimmt dabei die genannten WPF-Interna.

```
[TestFixture]
class IncrementDecrementGuiTests
{
     static IcuTest ICU = IcuTestStarter.IcuFromDir(@"C:\Testverzeichnis");

     [Test]
     public void RawWindowManualTest()
     {
          ICU.Invoke(() =>
          {
               var window = new Window();
               window.Show();
               // Interaktion mit "window"
               ICU.CheckView(window, "RawWindow");
               window.Close();
          });
     }
```

Listing 1: Ein einfaches IcuTest-Beispiel

In diesem Beispiel wird ein einfaches WPF-`Window` erzeugt, angezeigt, mit der `CheckView`-Methode überprüft und anschließend wieder geschlossen. `CheckView` ist das zentrale Prüfmittel von IcuTest, um Bitmap-Vergleiche durchzuführen. Die Benutzerinteraktionsmöglichkeiten nach einer fehlgeschlagenen `CheckView`-Methode sind Abbildung 3 dargestellt.

[3] Das Beispiel wurde in Verbindung mit dem NUnit-Testframework geschrieben

[4] „ICU.Invoke(() => { Methodenrumpf });“ ist gleichzusetzen mit „ICU.Invoke(methodenname);“, wobei die Methode bei zweiterem an anderer Stelle implementiert werden müsste. Die erste Schreibweise erzeugt somit eine Art „anonyme Methode“.

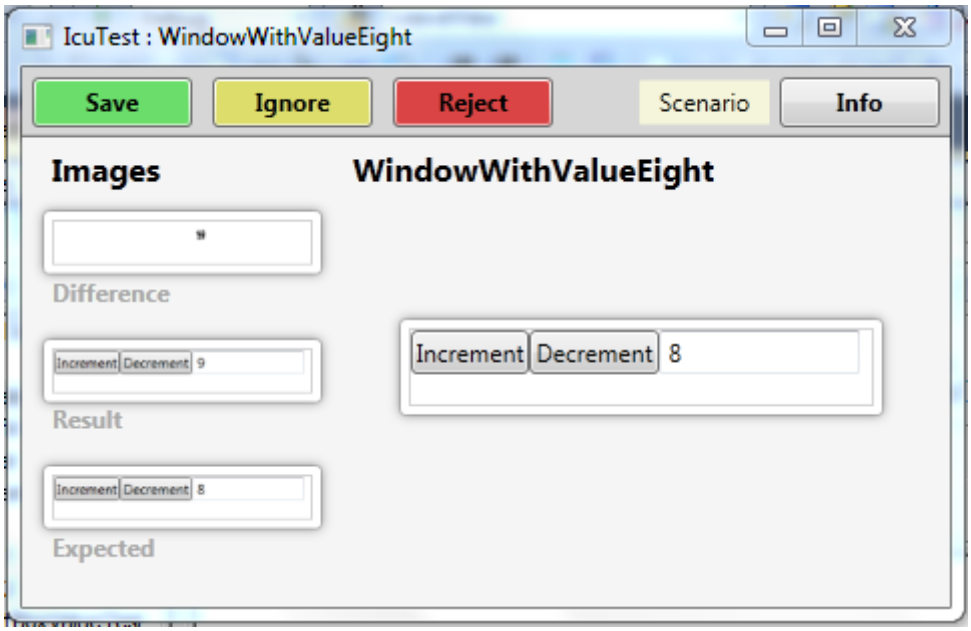

Abbildung 3: Interaktionsmöglichkeiten mit IcuTest nach einem fehlgeschlagenen Bitmap-Vergleich

7.16 Detaillierte Beschreibung

Die nachfolgenden vier Abschnitte behandeln wichtige Themen, welche notwendig sind, um mit IcuTest effektiv testen zu können.

7.16.1 Strukturiertes Testen

```
public class NUnitIcuBase{
     protected IcuTest icu;
     public virtual IcuTest ICU{
          get{
               if (icu == null){
                    icu = IcuTestStarter.IcuFromType(this.GetType());
               }
               return icu;
     }}
     [SetUp]
     public void TestInitialize(){
          ICU.BeginTest(null);
     }
     [TearDown]
     public void TestCleanup(){
          ICU.EndTest();
     }
}
```

Listing 2: Struktiertes Testen mit Basisklasse NUnitIcuBase

Um strukturierter testen zu können, kann es sinnvoll sein, eine Basisklasse für alle Tests zu schreiben, welche die `IcuTest`-Variable beinhaltet und für alle Tests geltende `Setup`- und `TearDown`-Methoden beinhaltet. Der Pfad der IcuTest-Ergebnisse wird hier auf „C:\Users\benutzername\Documents\IcuTests\this.GetType()-Ergebnis“ festgelegt.

7.16.2 Scenarios

Um sicher zu testen, ist es notwendig einen try/catch-Block zu verwenden und das Schließen des Fensters im finally-Block geschehen zu lassen (siehe Listing 3).

```
[Test]
public void RawWindowManualSafeTest(){
     ICU.Invoke(() =>
     {
          var window = new IncrementDecrementView();
          try{
               window.Show();
               ICU.CheckView(window, "RawWindow");
          }finally{
               window.Close();
          }
     });
}
```

Listing 3: Sicheres Testen mit finally-Block

Da dies und das Aufrufen von `window.Show()` und `window.Close()` jedoch bei jedem Test viel Overhead im Bezug auf den Programmieraufwand ergibt und sich leicht Fehler einschleichen können, gibt es die so genannten Scenarios, die dies für den Programmierer übernehmen.

In Listing 4 ist das vorige Beispiel mit Scenarios umgesetzt.

```
[Test]
public void RawWindowWithScenariosTest(){
     var context = new WindowScenario<IncrementDecrementView>();
     ICU.Given(context)
     .When(() =>
     {
          //Hier könen Vorbedingungen die vor dem Test stattfinden sollen stehen
     })
     .Then(() =>
     {
          var window = context.Window;
          ICU.CheckView(window, "RawWindow");
     })
     .Test();
}
```

Listing 4: Sicheres Testen mit Scenarios

7.16.2.1 Gui-Helper

Der Gui-Helper ist eine wichtige Klasse, um GUI-Elemente der WPF von einem `Window`-Objekt ausfindig machen und mit ihnen interagieren zu können. Zentral ist dabei die Methode `Find` der `GuiHelper`-Klasse. Mit ihr können Elemente des Root-Elements (erster Parameter) mit ihren Namen (zweiter Parameter) gefunden werden. Beim Namen ist damit die Name-Property[5] des WPF-Elements gemeint.

An der GuiHelper-Klasse stehen außerdem noch wichtige Methoden zur Interaktion mit den GUI-Elementen, wie zum Beispiel die Click- oder die SetText-Methode, zur Verfügung[6].

Diese Dinge können aus Gründen der Thread-Kommunikation nicht direkt an dem GUI-Element gesetzt werden.

[5] Für Properties siehe http://msdn.microsoft.com/en-us/library/x9fsa0sw%28v=vs.80%29.aspx
[6] Für weitere Methoden der Klasse GuiHelper siehe IcuTest-Dokumentation

```
[Test]
public void WindowTest(){
   GuiHelper guiHelper = new GuiHelper();
   var context = new WindowScenario<IncrementDecrementView>();
   ICU.Given(context)
   .When(() =>
   {
      Window window = context.Window;
      Button incrementButton = guiHelper.Find<Button>(window, "incrementButton");
      Button decrementButton = guiHelper.Find<Button>(window, "decrementButton");
      TextBox valueBox = GuiHelper.Find<TextBox>(window, "valueBox");
      guiHelper.SetText(valueBox, "7");
      guiHelper.Click(incrementButton);
      guiHelper.Click(incrementButton);
      guiHelper.Click(decrementButton);
   })
   .Then(() =>
   {
      Window window = context.Window;
      ICU.CheckView(window, "WindowWithValueEight");
   })
   .Test();
}
```

Listing 5: Ein Test mit der GuiHelper-Klasse

7.16.3 IcuConfig

IcuConfig ist eine Klasse, welche wichtig ist, um IcuTest zu automatisieren und andere wichtige Einstellungen vorzunehmen. Dies kann zum Beispiel in der SetUp-Methode vorgenommen werden und die Gestalt wie in Listing 6 besitzen.

- TestDirectory: Der Ort, an dem die Bilddaten abgespeichert werden
- IsEnabled: Diese Property legt fest, ob IcuTest-Checking aktiviert ist
- PassNewTests: Dieser Wert legt fest, ob neue Tests (d.h. noch nicht existierende Bilder-Pfade in der CheckView-Methode) automatisch akzeptiert werden sollen
- OnFailShowDialog: Legt fest, ob bei Differenzen zwischen Bildern ein Dialog erscheinen oder der Test direkt als ungültig markiert werden soll
- OnPassShowTest: Legt fest, ob ein Test angezeigt werden soll, auch wenn er gültig ist

```
ICU.Config.TestDirectory = @"c:\icu_test_verzeichnis";
ICU.Config.IsEnabled = true;
ICU.Config.PassNewTests = true;
ICU.Config.OnFailShowDialog = false;
ICU.Config.OnPassShowTest = true;
```

Listing 6: Tests ohne Benutzerintaktion. Diese Konfiguration gibt nur aus, ob ein Test korrekt oder nicht korrekt ausgeführt wurde.

Die hier gezeigten Einstellungen lassen alle Tests voll automatisiert und ohne nötige Benutzerinteraktion laufen, wenn die Einstellungen in der mit dem Attribut `SetUp` markierten Methode vorgenommen werden.

7.16.4 TestGenerator

Im bin-Ordner des Installationsverzeichnisses von IcuTest befindet sich die Datei „TestGenerator.exe“. Mit dieser lassen sich von einer kompilierten Assembly automatisch Test-Stubs generieren.

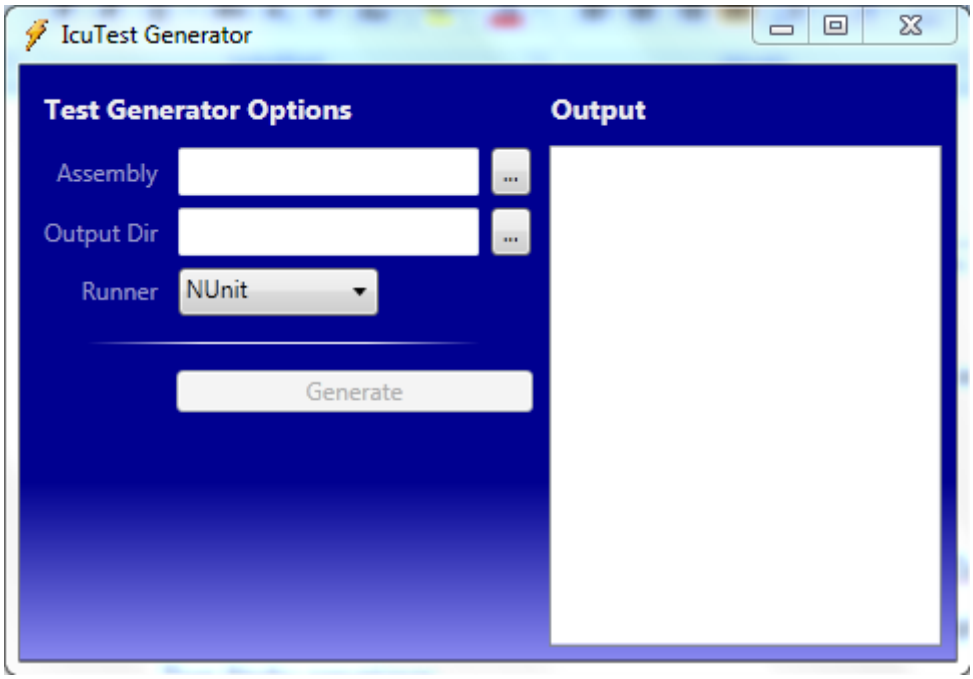

Abbildung 4: IcuTest Testgenerator

Erzeugt werden dabei immer drei Klassen. Im Fall NUnit werden die Klassen `NUnitIcuBase`, `ThatIncrementDecrementView` und `IcrementDecrementViewScenarios` erzeugt. Die Klasse `NUnitIcuBase` bildet die Basisklasse aller Tests mit den globalen Methoden `SetUp` und `TearDown` und der `IcuTest`-Instanz. `ThatIcrementDecrementView` erbt von dieser Klasse und bietet generierte Methoden für jedes GUI-Element der zu testenden Assembly an. Zusätzlich benutzt es die generierte Klasse `IcrementDecrementViewScenarios`, welche von `WindowScenario` erbt. Dieses generierte Scenario hat den Vorteil, dass es in jedem Test genutzt werden kann und dort die GUI-Elemente der Assembly bereits gesucht und in Membervariablen gespeichert sind. Somit muss das Element nicht in jedem Test per `GuiHelper` gesucht werden, sondern kann direkt im Scenario angesprochen werden, welches in jedem Test zugänglich ist.

7.17 Literatur

Für IcuTest existiert keine weitere Literatur.

8 Sonstige Quellen und hilfreiche Links

- http://miketwo.blogspot.com/2007/03/unit-testing-wpf-controls-with.html
  - Hier wird eine Möglichkeit vorgestellt, WPF-Elemente mittels Microsofts integriertem „UI Automation“ (http://msdn.microsoft.com/en-us/library/ms747327.aspx) in Testmethoden zugänglich und testfähig zu machen.
    Diese Herangehensweise ist umständlich und nicht sehr intuitiv.
- http://www.peterprovost.org/blog/post/NUnit-and-Multithreaded-Tests-CrossThreadTestRunner.aspx
  - WPF-Elemente können nur im Single Threaded Apartment (STA) Modell instanziiert werden. NUnit 2.1 l läuft normalerweise allerdings im Multithreaded Apartment (MTA) Modell. In diesem Beitrag wird ein Lösungsweg vorgeschlagen, Multithreaded Tests mit NUnit 2.1 durchzuführen.
- http://blogs.msdn.com/b/dancre/archive/2006/10/11/datamodel-view-viewmodel-pattern-series.aspx
- http://efreedom.com/Question/1-2421983/Testing-Gui-Heavy-WPF-Application
- http://social.msdn.microsoft.com/Forums/en/wpf/thread/418a71e6-b55c-4fb4-b510-4e06c2a812e3
- http://joshsmithonwpf.wordpress.com/2007/07/09/using-nunit-with-wpf/