# 1 csUnit

## 1.1 Homepage
http://csunit.org

## 1.2 Lizenz
zlib-Lizenz (kompatibel zur „GNU General Public License“ )

## 1.3 Untersuchte Version
csUnit 2.6.3374

## 1.4 Letzter Untersuchungszeitpunkt
08.02.2011

## 1.5 Kurzbeschreibung
csUnit ist ein Unit-Testing-Framework. Mit ihm lassen sich Unit-Tests für alle .NET-kompatiblen Sprachen[1] schreiben und auswerten.

Das Tool ist zusätzlich in der Lage, Unit-Tests auszuwerten die mit anderen Frameworks kompiliert wurden[2].

## 1.6 Fazit
Die grafische Benutzeroberfläche ist im Vergleich zu der von NUnit anwendungsfreundlicher. Als Beispiel sei hier die Suchfunktion genannt, die das Finden von Tests und deren Ergebnisse bei großen Projekten vereinfacht. Hinzu kommt, dass es sich in Microsoft VS 2008 integrieren lässt.

Die Syntax des Frameworks ist fast identisch mit der von NUnit, bietet allerdings keinen so großen Funktionsumfang.

## 1.7 Einsatzgebiete
Das Werkzeug kann für .NET Programmiersprachen eingesetzt werden. Es bietet dem Entwickler die Möglichkeit seinen Quellcode mittels Unit-Tests zu prüfen.

## 1.8 Einsatzumgebungen
Das Programm besitzt eine grafische Oberfläche, kann aber auch als Kommandozeilen-Applikation genutzt werden. Eine weitere Möglichkeit besteht darin, es in Microsoft Visual Studio 2005/2008 zu integrieren.

## 1.9 Installation
Die benötigte Software kann mittels Microsoft Windows Installer[3] auf dem PC eingerichtet werden. Während der Installation sind keine besonderen Einstellungen vorzunehmen. Lediglich die AGBs sind zu bestätigen und der Speicherort zu wählen.

Um in einem C#-Projekt das csUnit-Framework verwenden zu können, muss das Projekt auf die „csunit.dll“-Bibliothek verweisen.

- Visual Studio – Im Projektbrowser, Rechtsklick auf den Ordner „Verweis“ und auf „Verweis hinzufügen...“ klicken. Nun den Pfad zur Datei „csunit.dll“ im Installationsverzeichnis auswählen.

[1] C#, VB.NET, Managed C++, …
[2] NUnit (version 2.4.7, .NET 2.0) und Microsoft Unit Testing (basic support, VS 2008)
[3] Dateieendung .msi

- Sharpdevelop - Im Projektbrowser, Rechtsklick auf den Ordner „Referenzen“ und auf „Referenz hinzufügen“ klicken. Nun den Pfad zur Datei „csunit.dll“ im Installationsverzeichnis auswählen.

## 1.10 Dokumentation

Die grafische Benutzeroberfläche kann eine Online-Dokumentation der Projektwebseite aufrufen. Dort findet man neben der Dokumentation ein FAQ und verschiedene Tutorials. Dabei befindet sich unter der Rubrik Support noch ein Troubleshooting Guide um csUnit in Visual Studio einzubinden.

## 1.11 Wartung der Projektseite

Die Seite ist übersichtlich aufgebaut und macht einen soliden Eindruck. Jedoch ist zu bemängeln, dass die Suchfunktion der Webseite nicht funktioniert und die Aktualisierung der News zu mangelhaft ist (letzter Eintrag vom April 2009).

## 1.12 Nutzergruppen und Support

Das Entwicklerteam bietet eine Diskussions-Gruppe an[4]. Nach Angaben des Entwicklerteams, werden die Fragen innerhalb von 1-2 Tagen beantwortet. Zusätzlich können Bug Reports eingeschickt werden.

## 1.13 Automatisierung

csUnit ist in der Lage, einem Test verschiedene Parameter zu übergeben. Dabei können die Parameterdaten aus Dateien im XML-Format, Datenbanktabellen oder statischen Methoden stammen[5]. Durch dieses Feature lassen sich Testfälle mit verschiedenen Werten ausführen, was bei großen Testdatenmengen von Vorteil ist.

Des Weiteren lassen sich Tests zu Gruppen zusammenfassen. Dabei lassen sich TestFixtures und Tests in Kategorien einteilen (analog zu NUnit)[6].

## 1.14 Einführendes Beispiel

An dieser Stelle befindet sich ein einführendes Beispiel für die Nutzung von csUnit.

### 1.14.1 Programmierung

Zur Nutzung von csUnit benötigt man ein TestFixture. Dieses wird als `[TestFixture]`-Attribut vor die Testklasse geschrieben.

Anschließend können eine oder mehrere Testmethoden erstellt werden. Diese müssen mit dem Attribut `[Test]` gekennzeichnet werden.

```
[TestFixture]
public class MitarbeiterTest{ // es soll die Klasse Mitarbeiter getestet werden
     [Test]
     public void MitarbeiterTest(){ //Test Methode
           /* ... */
     }
}
```

Listing 1: TestFixture- und TestAttribute in csUnit

[4] http://groups.google.com/group/csunit
[5] http://www.csunit.org/documentation/2.4/parameterizedTests.html
[6] http://www.csunit.org/documentation/2.4/categories.html

Es können vor und nach jedem Test Methoden ausgeführt werden, die zum Beispiel vor jedem Test das gleiche Objekt erzeugen, damit alle Test-Methoden ein Objekt zu gleichen Bedingungen erhalten. Dabei handelt es sich um die Attribute `[SetUp]` und `[TearDown]`. Letzteres sorgt dafür, dass nach jeder Testmethode, diese Funktion ausgeführt wird.

Falls es nötig sein sollte einen Test nicht ausführen zu wollen, weil er zum Beispiel noch nicht komplett ausprogrammiert ist, dann kann er mit dem Attribut `[Ignore]` übersprungen werden. Auf diese Weise lassen sich auch TextFixtures überspringen. Der optionale Kommentar wird dann beim Testen angezeigt.

```
[TestFixture]
public class MitarbeiterTest {
     [SetUp] public void Init()
     { /* ... */ }

     [TearDown] public void Cleanup()
     { /* ... */ }

     [Test] public void Gehalt()
     { /* ... */ }

     [Test]
     [Ignore("Diesen Test Ignorieren")]
     public void TestIgnorien()
     { /* ...*/ }

     [Test]
     [ExpectedException(typeof(Exception)
     ,ExpectedMessage="Maximal 3 Fachgebiete")]
     public void FachgebietException()
     {
          mitarbeiter.AddFachgebiet(Fachgebiet.DESIGN); //Exception wird geworfen
     }
}
```

Listing 2: Ignore- und ExceptedExceptionAttribute in csUnit

Wenn davon auszugehen ist, beziehungsweise erwartet wird, dass eine Testmethode (zum Beispiel GehaltException()) eine Exception auslöst, dann muss diese Methode vorher als solche gekennzeichnet werden, ansonsten gilt der Test als nicht bestanden. Ist die Exception nicht vom geforderten Typ, so gilt dies auch als nicht bestanden. Über das Attribut `[ExpectedException]` und dem Typ der Exception ist dies anzugeben.

Um nun einzelne Konditionen / Bedingungen zu prüfen, benötigt man die Assert-Klasse.

Mit Hilfe dieser kann zum Beispiel geprüft werden, ob das Gehalt eines Mitarbeiters größer, kleiner oder genau gleich einem bestimmten Betrag ist.

Bei fehlgeschlagener Assertion kommt es zu einem Fehler, der als fehlgeschlagener Test zu werten ist. Nach dieser Assertion führt die Test Methode keine weiteren Assertions mehr aus. Daher ist es zu empfehlen, pro Test nur eine Assertion durchzuführen.

```
[Test(Categories="MitarbeiterGehalt") //Anlegen einer Kategorie
public void Gehalt(){
     Assert.Greater(mitarbeiter.Gehalt, 20000); //i.o. Wenn Betrag>20.000
     Assert.Less(mitarbeiter.Gehalt, 100000); //i.o. Wenn Betrag<100.000
     Assert.AreEqual(mitarbeiter.Gehalt, 36000, "Verdient 36.000 euro");
}
```

Listing 3: Katergorien in csUnit

In Listing 3 werden `Greater`, `Less` und `Equal` als Constraints bezeichnet. Alle Asserts, Attribute und Constraints befinden sich im Framework-namespace von csUnit.

Weitere Beispiele befinden sich in den folgenden Quelldateien:

*Mitarbeiter.cs, MitarbeiterTest.cs, Projekt.cs, ProjektTest.cs*

### 1.14.2 Die grafische Benutzeroberfläche von csUnit

Auf Grundlage des Projektes „Mitarbeiterverwaltung“[7] befindet sich nachfolgend eine kleine Demonstration von csUnit.

Zu Anfang muss die Assembly-Datei des kompilierten Projektes in die Anwendung geladen werden. Das Programm ist auch in der Lage mehrere Assemblies einzubinden.

Über die „Einfügen“ Taste oder den Menüpfad „Assembly“→ „Add...“ ist die gewünschte Assembly-Datei auszuwählen. Im nachfolgenden Beispiel handelt es sich um die „Mitarbeiterverwaltung.dll“ aus dem Ordner „..\Mitarbeiterverwaltung\bin\Debug\“.

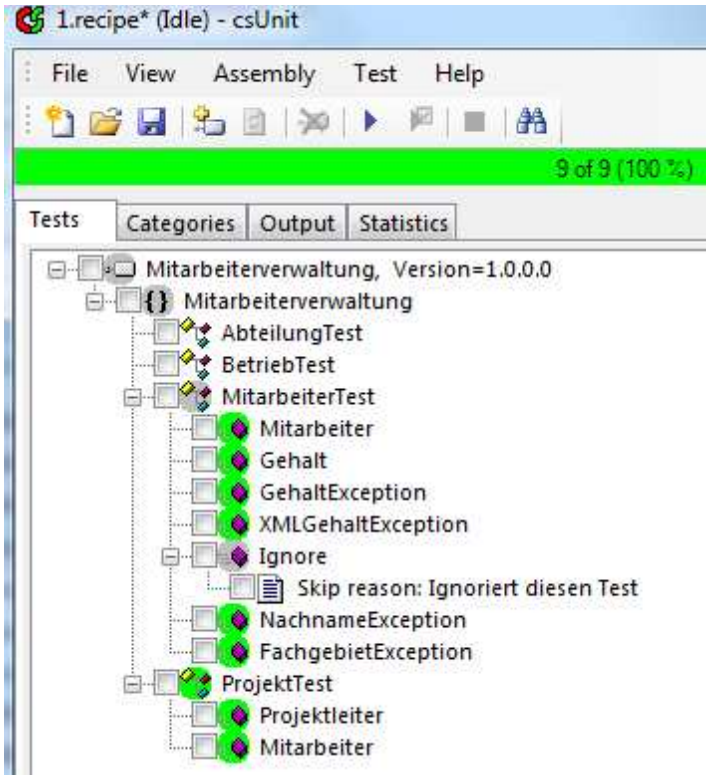


Abbildung 1: Ergebnisse der Assebly „Mitarbeiterverwaltung.dll“

Nach dem Einbinden der Datei kann entweder der gesamte Test oder aber auch nur einzelne Test-Kategorien (`CategoryAttribut`) ausgewählt werden.

Um Kategorien in einen Test einzubinden, ist es notwendig zur Ansicht „Categories“ zu wechseln. Dort kann wie in Abbildung 1 zu sehen über die Buttons (+,-,0) entschieden werden, welcher Test einzubinden ist.

In der Baumansicht, lassen sich auch gezielt Tests auswählen (Check Boxes) und anschließend über den Button „Run Checked Tests“ oder die Taste F8 ausführen.

Ein weiterer schneller Weg ist es, einen Rechtsklick auf den gewünschten Test ausführen und „Run This“ auszuwählen.

[7] Siehe Anhang

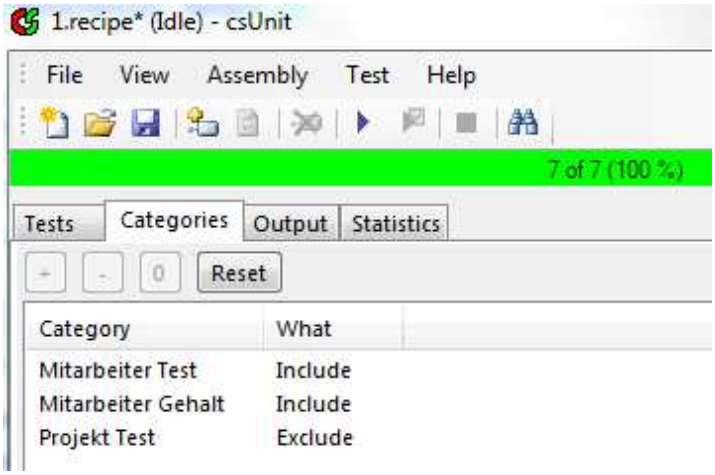


Abbildung 2: Gruppieren von Tests mit Hilfe von Kategorien

Der Fortschrittsbalken (siehe Abbildung 1) besitzt dabei zwei Farbcodierungen: Grün für erfolgreiche und rot für nicht erfolgreiche Tests. Zusätzlich sind die Icons in der Baumansicht entsprechend eingefärbt.

Unter dem Tab „Statistic“ (siehe Abbildung 3) befindet sich explizit aufgelistet, welcher Test wieviel Millisekunden zur Durchführung benötigte. Die Prozentangaben beziehen sich hierbei auf den prozentualen Anteil der gesamt Laufzeit. Die letzte Angabe soll

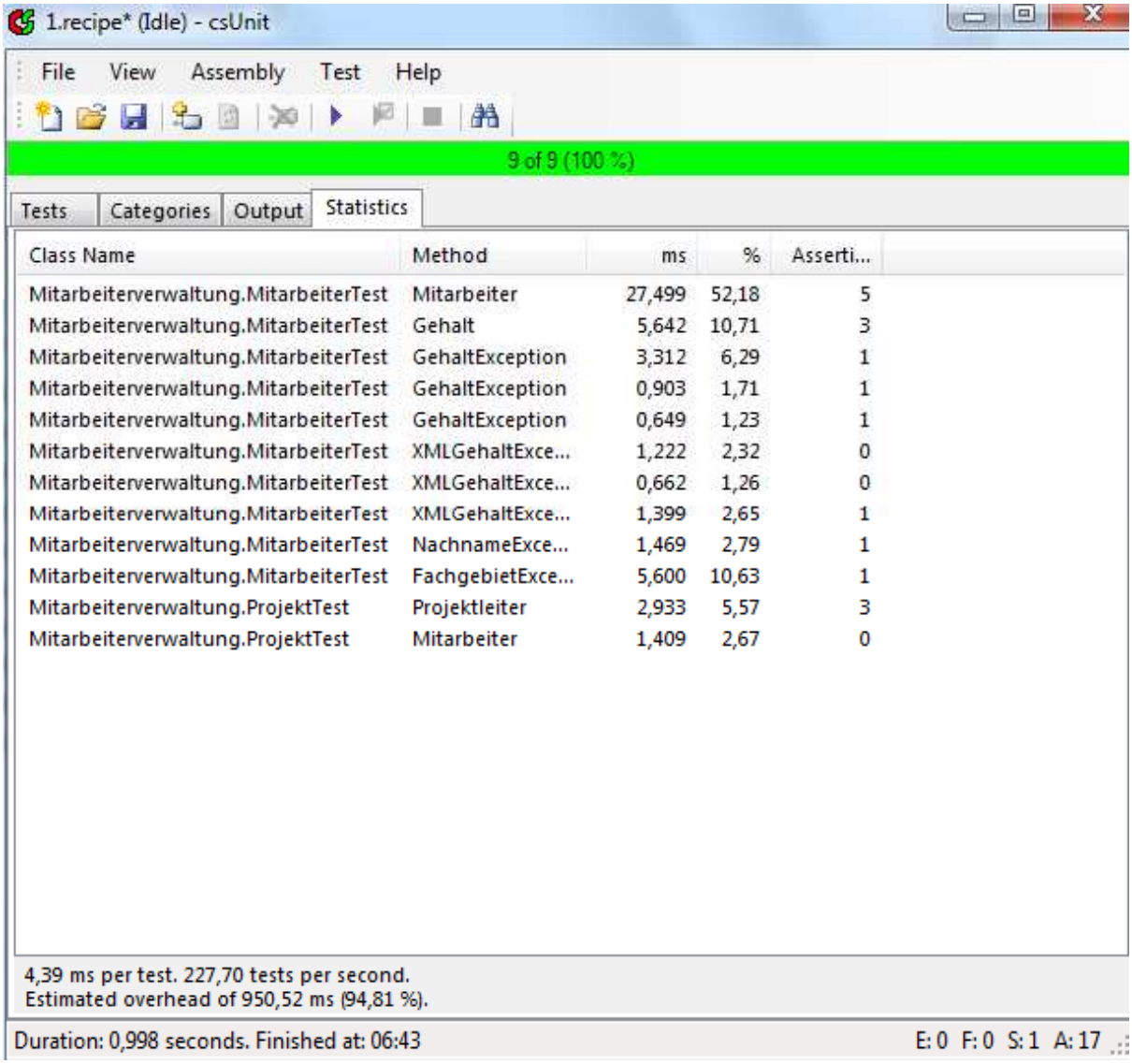


Abbildung 3: Statistische Ergebnisse der Testklassen MitarbeiterTest.cs und ProjektTest.cs

wiedergeben, wie viele Assertions dabei erfüllt wurden. Dabei ist gut zu erkennen, dass die parametergesteuerten Methoden „GehaltException“ und „XMLGehaltException“ pro Datensatz einmal ausgeführt wurden. Allerdings ist nicht ganz klar, wieso die beiden Tests unterschiedliche Anzahlen an Assertions liefern.

Für den gezielten und schnellen Zugriff auf einen Test oder seine Ergebnisse bietet csUnit eine hilfreiche Suchfunktion an. Sie kann über das „Fernglas“-Schaltsymbol oder die Tastenkombination „Strg+F“ aufgerufen werden.

Um zu überprüfen, ob es eine neue Version von csUnit gibt, kann mit einer bestehenden Internetverbindung die Version geprüft werden („Help“ → „Check for Updates“).

Damit nicht alle Einstellungen verloren gehen, können diese gespeicher werden (File→ …). Die Resultate der Tests lassen sich ebenfalls speichern (im XML-Format).

## 1.15 Detaillierte Beschreibung

### 1.15.1 Parameterübergabe

Um das Problem zu umgehen, dass der Entwickler für jeden zu testenden Wert eine eigene Testmethode schreiben muss, kann er eine Testmethode von außen mit Testwerten versorgen.

Es stehen hierbei die folgenden vier Konzepte zur Parameterübergabe zur Verfügung.

#### *1.15.1.1 DataRowAttribute - Inline Anweisung*

Das Attribut DataRow erhält die benötigten Parameter für die Test Methode in einer Anweisung. Die Testmethode wird pro DataRow einmal aufgerufen. Falls es zu Exceptions kommt, muss auf die DataRow das Attribut ExpectedException folgen.

Das Objekt „Jens“ ist vom Typ Mitarbeiter, der maximal ein Gehalt von 100000 erhalten darf. Daher wird beim Versuch über die Set-Methode das Gehalt auf 100001 zu erhöhen, eine Exception geworfen.

```
[Test(Categories="Mitarbeiter Gehalt")]
[DataRow(99999)] //[DataRow(5,2)] für mehrere Werte MyTest(int x,int y)
[DataRow(100000)]
[DataRow(100001 ,ExpectedException=typeof(ArgumentOutOfRangeException))]
public void GehaltException(int money){
     jens.Gehalt=money; //Set-Methode von Gehalt
}
```

Listing 4: DataRow-Attribut in csUnit

#### *1.15.1.2 DataSourceAttribute - Statische Methoden und Properties*

Möchte der Entwickler die Testdaten mehrfach nutzen, so muss er für diese Tests nicht die gleichen DataRow-Attribute schreiben, sondern kann sich mit folgender Lösung behelfen.

```
[Test]
[DataSource(typeof(StaticDataProvider))]
public void GehaltException(int money){
     jens.Gehalt=money; //Set-Methode von Gehalt
}
```

Listing 5: DataSource-Attribut in csUnit

Dafür muss die Klasse StaticDataProvider angelegt werden. Diese sorgt für die Parameterdaten, die für den Test notwendig sind.

```
public class StaticDataProvider
{
     public static DataRow[] GetTestData(){
          return new DataRow[] {
               new DataRow(99999),
               new DataRow(100000),
               new DataRow(100001
                    ,ExpectedException=typeof(ArgumentOutOfRangeException))
          };
     }
}
```

Listing 6: StaticDataProvider-Beispiel

#### *1.15.1.3 XML-Dateien*

Eine weitere Möglichkeit Parameter an einen Test zu übergeben, besteht darin, die Werte aus einer XML-Datei zu lesen. Dazu muss wie im folgenden Beispiel verfahren werden.

```
[Test(Categories="Mitarbeiter Gehalt")]
[DataSource("money.xml")] //Angabe des Dateipfades
public void XMLGehaltException(int money) { //erwartet integer Werte
     jens.Gehalt=money; //Set-Methode von Gehalt
}
```

Listing 7: XML-DataSource in csUnit

Hier sollen nun die gleichen Werte, wie bei dem Test mit DataRow verwendet werden.

Wichtig ist hier es, darauf zu achten, dass die ExpectedException Anweisung nun in der XML-Datei stehen muss[8].

```
<dataTable>
     <dataRow>
        <value>99999</value>
     </dataRow>
     <dataRow>
        <value>100000</value>
     </dataRow>
     <dataRow>
        <value>100001</value>
        <expectedException>System.ArgumentOutOfRangeException</expectedException>
     </dataRow>
</dataTable>
```

Listing 8: Inhalt der XML-Datei „money.xml“

#### *1.15.1.4 Tabellen einer Datenbank*

Für die Unit-Tests können auch Parameter aus einer Datenbanktabelle geladen werden. Angenommen ein Test benötigt 3 Parameter, so wäre eine Tabelle mit 3 Spalten nötig. Dies bedeutet, dass eine Zeile der Tabelle einen Test mit Parameterdaten versorgt.

Der Verbindungsaufbau setzt sich aus den Komponenten .NET Data Provider, Konfiguration der Verbindung und der Tabellenname der Datenbank zusammen.

[8] Die XML-Datei befindet sich im Verzeichnis „..\Mitarbeiterverwaltung\bin\Debug\“

Das Folgende Beispiel stammt aus der Dokumentation von csUnit.

```
[Test]
[DataSource("System.Data.SqlClient",
"Data Source=.\\SQLEXPRESS;AttachDbFilename=" +
"\"C:\\data\\csUnitTestData.mdf\";" +
"Integrated Security=True;Connect Timeout=30;User Instance=True",
"DivisionTests")]

public void Division(int nominator, int denominator, int expectedResult) {
     int actualResult = nominator / denominator;
     Assert.Equals(expectedResult, actualResult);
}
```

Listing 9: Datenbank-DataSource in csUnit