Beispiel für einen Ablauf

1. Starten eines Servers auf 9000
2. Starten eines Servers auf Port 9001
3. Starten des Clients
4. Eingabe von Server im Client
5. Auswahl einer Assembly am Client.
6. Klicken des Start Buttons
7. Client sendet Assembly an alle Server
   1. Server laden die Assembly
8. Tests…

<…>

1. Unittest Framework

Im Projekt PCG3.TestFramework wurde ein eigenes Testframework implementiert. Die Verwendung von einem vorhandenen Framework, schien uns nicht sinnvoll, weil dadurch zusätzlicher Overhead durch das übermitteln von Referenzen und Co enstehen würde.

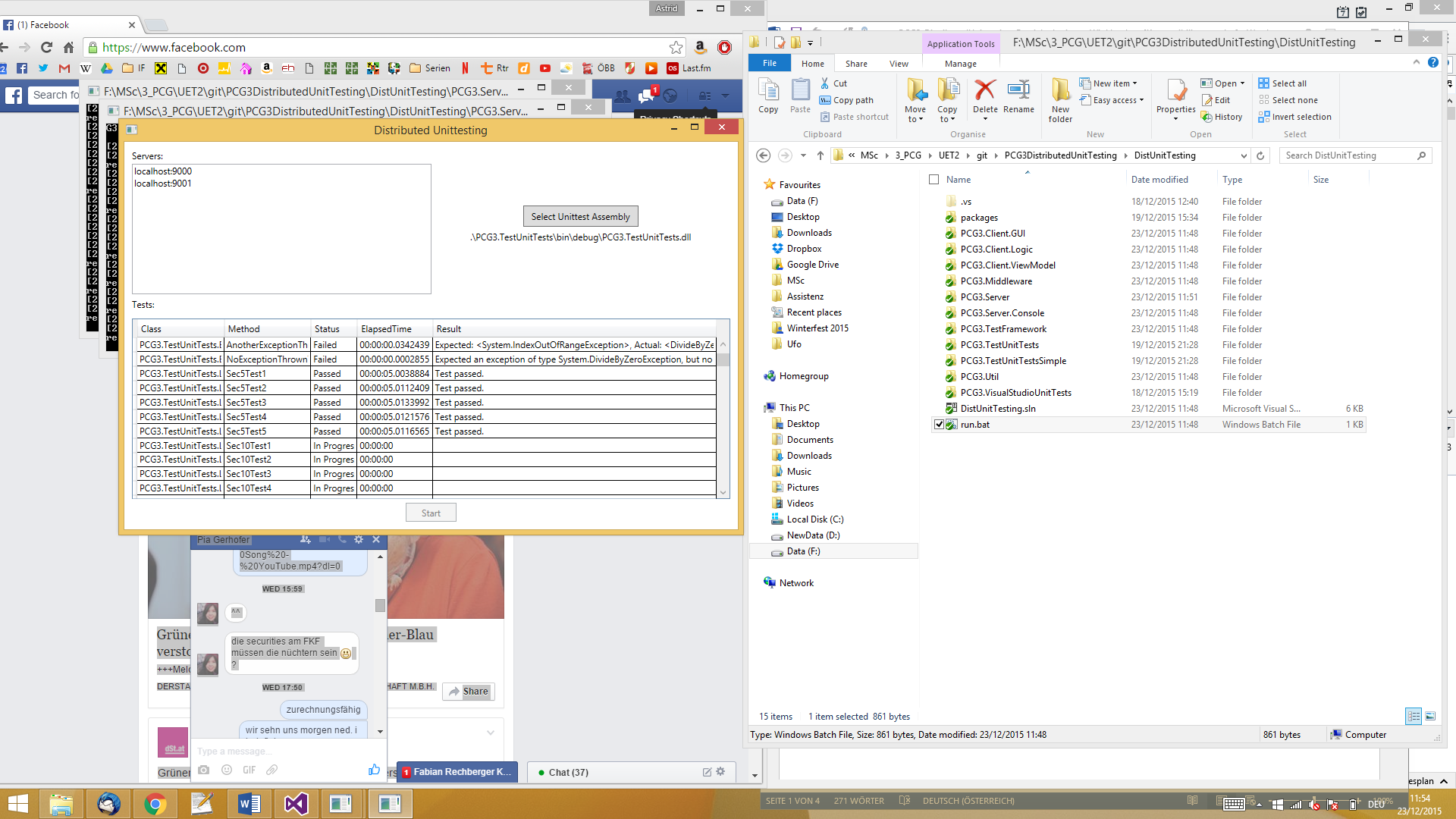
Das Framework stellt die wichtigsten Funktionen von anderen, üblichen Testframeworks zur Verfügung. Dazu zählen Assert, die Annotationen Test und ExceptedException. Die Klasse TestRunner ist für die Ausführung von Tests zuständig. Der Runner benötigt eine Instanz der Klasse Test.

Ein Objekt Test enthält zu Beginn den Methodennamen und die dazugehörige Klasse eines Tests (dies identifiziert einen Test eindeutig). Dann kann ein Test an den TestRunner gesendet werden. Die Rückgabe liefert mir diesen Test mit erweiterten Information: den Status, der vergangen Zeit und einer möglichen Message (z.B. bei einer Exception). Ein Test besitzt auch einen Status, welcher NONE, FAILED, PASSED oder WAITING sein kann, zu Beginn ist dieser Status NONE.

Das TestFramework liefert für die erstellten Tests dieselben Ergebnisse, als das TestTool vom Visual Studio.

Im Projekt PCG.TestUnitTests befinden sich 120 Unittests, welche sich zeitlich unterscheiden. Dies wurde mithilfe von einem einfachen Sleep gelöst.

1. Client
   1. WPF Anwendung



In der WPF Anwendung kann man zu Beginn nur eine Unittest Assembly hochgeladen werden. Lädt man diese hoch, sieht man die Klassen und Methoden, aber noch ohne Status oder Ergebnisse.

Rechts oben kann man die Server hinzufügen, welche in Array von Server Adressen gespeichert werden.

Wenn man auf Start klickt, werden zuerst die Assemblies auf jeden Server geladen. Im Anschluss werden die Tests an die Server verteilt.

* 1. Logik

Die Logik befindet sich im Projekt PCG3.Client.Logic. Dort werden die Aufrufe an den Server verarbeitet. Sowohl Client als auch Server müssen sich an den Contract halten, welcher sich im Projekt PCG3.Middleware befindet. Dort werden die Kommunikationsschnittstellen definiert.

Mithilfe des XcoAppSpace verbindet man sich je Serveradresse zu einem Worker. Da im ersten Schritt die Assemby auf die Server verteilt werden muss. Die Assembly selbst ist nicht zu einfach zu übertragen. Daher wird die Assembly als Byte-Array wie eine normale Datei eingelesen und gesendet. Serverseitig wird die Assembly geladen. Bei einem Fehler wird im Response die Fehlermeldung zurückgegeben. Darüber hinaus wird in jedem Fall ein boolescher Wert gesetzt, um am Client abprüfen zu können, ob dies funktioniert hat. Der

Tests

1. Server

Der Server befindet sich im Projekt PCG3.Server. Bei Ausführung des Servers kann mithilfe von Übergabeparametern, die Cores und den Port angeben. Ansonsten wird auf vier Cores gearbeitet und der Port 9000 verwendet.

Tests

1. Kommunikation
2. Resümee

Es gab Schwierigkeiten mit dem XcoAppSpace. In der Klasse Test befanden sich MethodInfo und Exception, welche jedoch beide wenn sie Daten enthalten (und erst dann) nicht Serialisierbar. Dies gibt jedoch leider keine Fehlermeldung, sondern führt dazu, dass diese dann einfach nicht gesendet werden.