VU Qualitätssicherung in der Softwareentwicklung
LV-Nr. 716.066, SS 2018

# Aufgabenblatt 1

ao. Univ.-Prof. Dr. Bernhard Aichernig
Christoph Rehbichler, `c.rehbichler@student.tugraz.at`
Stefan Kuhs, `kuhs@student.tugraz.at`

15.03.2018, Graz

## Gruppenanmeldung

**Frist:** bis Di 20.03.2018
**Art der Anmeldung:** Online-System des IST

Ein beliebiges Mitglied der Gruppe muss dabei die gesamte Gruppe auf einmal eintragen. Es werden nur Gruppen zu jeweils 3 Studierenden akzeptiert. Zur Gruppenfindung kann die Newsgroup `news.tugraz.at/tu-graz.lv.qs` verwendet werden. Jeder Gruppe wird nach der Anmeldefrist eine Gruppennummer zugewiesen, welche per E-Mail bekanntgegeben wird. Die Gruppe bleibt für alle 3 Aufgabenblätter bestehen. Außerdem wird für jede Gruppe automatisch ein SVN-Repository angelegt, welches uns als Medium für die Abgabe dienen wird.

**Achtung:** Wer am 20.03.2018 noch keiner Gruppe zugeteilt ist, kann nicht an der Übung teilnehmen und erhält somit auch keine Beurteilung in diesem Fach.

## 1 Codeabdeckung

Ziel der ersten Aufgabe ist es, sich mit der Entwicklungsumgebung und der Anwendungsdomäne vertraut zu machen. Dazu sollen Unittests geschrieben werden, welche eine Codeabdeckung von 100 % erzielen. Die Bewertungsgrundlage ist die Codeabdeckungsanalyse von Visual Studio für das Projekt `MessageBoard` im Ordner MessageBoardCodeCoverage. Weiters ist zu beachten, dass die Tests sinnvolle Asserts beinhalten müssen (nicht sinnvoll ist zum Beispiel Assert.IsTrue(true)). Das zu testende Framework finden Sie auf der Lehrveranstaltungswebseite (`http://www.ist.tugraz.at/qs.html`). Eine Beschreibung der Domäne folgt in Abschnitt 5.

**Tipp:** Um 100 % Codeabdeckung zu erreichen, ist es hilfreich einzelne Klassen für Testzwecke abzuleiten. (ähnlich dem bereits vorhandenem `TestClient`)

### 1.1 Bewertungsgrundlage

Als Bewertungsgrundlage dient das kompilierte Unittestprojekt `MessageBoardTest` im Ordner MessageBoardCodeCoverage, das Ihre Unittests beinhalten soll. Beachten Sie, dass alle Änderungen an der Implementierung im Projekt `MessageBoard` verworfen und für die Bewertung nicht berücksichtigt werden. Des Weiteren ist zu beachten, dass, obwohl Binärdateien im Regelfall nicht abzugeben sind, für diese Abgabe auch die dll-Datei des Unittestprojekts abgegeben werden soll. Die volle Punktezahl erhält eine Abgabe genau dann, wenn

- eine CodeCoverage von 100 % erreicht wird,
- alle Tests sinnvolle Asserts beinhalten und
- alle Tests positiv abschließen.

## 2 Randwerttesten

Diese Aufgabe beschäftigt sich mit dem Testen einer Funktion, welche die soziale Aktivät eines Benutzers berechnet. Folgende Eingabeparameter nimmt die Funktion entgegen:

**Follows:** bezeichnet die Anzahl an anderen Benutzern, die ein Benutzer pro Woche neu folgt. Die Untergrenze ist 0 und die Obergrenze beträgt 20 Benutzer inklusive.

**Likes:** bezeichnet die Anzahl an Likes, die ein Benutzer pro Woche vergibt. Der Wert liegt zwischen 0 und 200 Likes inklusive.

**Posts:** bezeichnet die Anzahl an Posts, die ein Benutzer pro Woche verfasst. Der Wert liegt zwischen 0 und 13 Posts inklusive.

Die soziale Aktivität soll dabei den Wert von 100 nicht übersteigen. Die Funktion soll für valide Eingabeparameter eine Zahl zwischen 0 und 100, einschließlich der Grenzen, berechnen und zurückgeben. Für nicht erlaubte Eingabeparameter soll die Funktion eine `ArgumentException`-Exception werfen.

### 2.1 Aufgabe

Erstellen Sie Robustheits-Randwerttests, um diese Funktion zu testen. Nehmen Sie für diese Tests die "single fault assumption" an, gehen Sie also davon aus, dass die Parameter voneinander unabhängig sind. Um die volle Punkteanzahl zu erreichen, muss eine Abgabe alle benötigten, aber keine laut "single fault assumption" unnötigen Tests beinhalten.
Beantworten Sie zusätzlich in der bereits vorhandenen Datei `solution.txt` folgende Frage: Gibt es eine Kombination von Parametern, die durch die "single fault assumption" ausgeschlossen wird und zu einer fehlerhaften Berechnung führen würde? Falls ja, geben Sie diese Parameterkombination an. Begründen Sie Ihre Antwort.

### 2.2 Hilfestellung

Es wird ein Test-Projekt zur Verfügung gestellt, welches bereits eine Test-Klasse `SocialActivityTest` beinhaltet. Wenn die Tests durchgeführt wurden, erstellt die `teardown()`-Methode dieser Klasse ein formatiertes Log-File, welches die Eingabe-Parameter der Funktionsaufrufe beinhaltet. Dieses Log-File kann mit Hilfe des zur Verfügung gestellten Programms `RobustnessTester` ausgewertet werden. Das

Programm erstellt basierend auf dem Log-File-Inhalt drei PNG-Dateien, welche die abgedeckten Eingabebereiche grün und die noch abzudeckenden Bereiche rot kennzeichnet. Die drei Dateien beziehen sich jeweils auf zwei Eingabeparameter, die Datei `FollowsLikes.png` zeigt Bereiche für Follows und Likes, die Datei `LikesPosts.png` zeigt Likes und Posts und die Datei `PostsFollows.png` zeigt Posts und Follows.

### 2.3 Bewertungsgrundlage

Als Bewertungsgrundlage dient das kompilierte Unittestprojekt `SocialActivityTest`, das Ihre Randwerttests beinhalten soll. Beachten Sie, dass alle Änderungen an der Implementierung im Projekt `SocialActivity` verworfen und für die Bewertung nicht berücksichtigt werden. Weiters ist zu beachten, dass, obwohl Binärdateien im Regelfall nicht abzugeben sind, für diese Abgabe auch die dll-Datei des Unittestprojekts abgegeben werden soll. Die volle Punktezahl erhält eine Abgabe genau dann, wenn

- alle relevanten Bereiche durch Tests abgedeckt werden,
- kein Bereich durch mehr als einen Test abgedeckt wird und
- die theoretische Frage richtig beantwortet wurde.

## 3 Punkteverteilung 16 Punkte

- **10 Punkte:** Codeabdeckung
- **6 Punkte:** Randwerttesten
  - **5 Punkte:** Soziale Aktivität
  - **1 Punkte:** Theoriefrage

## 4 Entwicklungsumgebung

Um die Möglichkeiten von C# bestmöglich zu nutzen, wird als Entwicklungsumgebung **Visual Studio Enterprise 2015** verwendet. Wir empfehlen diese Version zu verwenden, da in den nächsten Aufgabenblättern weitere Plugins und Features von Visual Studio 2015 verwendet werden. Visual Studio (und falls notwendig Windows) bekommen Sie über Microsoft Imagine.

## 5 Systembeschreibung

Die Anwendungsdomäne wird in diesem Punkt überblicksartig beschrieben. Eine genauere Beschreibung der Klassen und Methoden finden Sie als Dokumentationskommentare in den Quellcodedateien. Die dazugehörige Implementierung kann als korrekt angenommen werden.
Das zu testende System ist ein einfaches Messageboard, welches als (simuliertes) Aktorensystem realisiert ist. Das Projekt ist in zwei Teile gegliedert und zwar in jenen Teil, der das Aktorensystem implementiert und in jenen Teil, der die Messageboard-Funktionalität implementiert. Im Code wird diese Aufteilung mittels Namespaces umgesetzt.

**Aktorensystem** Das simulierte Aktorensystem, das verwendet wird, bietet Funktionalitäten an, die durch das *actor model* (siehe beispielsweise `http://en.wikipedia.org/wiki/Actor_model`) inspiriert sind. Grundsätzlich bestehen solche Systeme aus parallel agierenden Aktoren, welche durch den Austausch von Nachrichten miteinander kommunizieren.
In der zu testenden Implementierung werden Aktoren durch Objekte der Klasse `SimulatedActor` repräsentiert. Die wichtigste Methode von Aktoren ist `Receive`, welche die Logik für das Verhalten von Aktoren beinhaltet. Die Aktoren werden von einer Instanz der Klasse `SimulatedActorSystem` gemanagt, welche das Verstreichen von Zeit simuliert. Dazu wird die Methode `SimulatedActor.Tick()` aufgerufen, um Aktoren das Verstreichen einer Zeiteinheit zu signalisieren.
Zeit hat in unserem Aktorensystem zwei verschiedene Auswirkungen. Zum einen braucht eine Nachricht nach dem Senden eine bestimmte Anzahl an Zeiteinheiten, um einen Aktor zu erreichen. Das wird durch die Klasse `CommunicationChannel` simuliert. Zum anderen benötigen Aktoren Zeit, um Nachrichten zu verarbeiten. Diese Zeit wird durch die Methode `Message.Duration()` definiert. `Message` ist ein Interface, welches von allen Nachrichten implementiert werden muss.

**Messageboard** Das Messageboard bietet Clients die Möglichkeit, kurze Nachrichten zu posten, Nachrichten anderer zu "liken" und "disliken", und Nachrichten eines bestimmten Autors zu erhalten. Das System ist auf vier verschiedene Aktoren-Klassen aufgeteilt. Es gibt einen `Dispatcher`-Aktor, mehrere `Worker`-Aktoren, einen `MessageStore`-Aktor, welcher für Persistenz zuständig ist und mehrere `WorkerHelper`-Aktoren. Kommunikationen zwischen einem Client und dem System finden folgendem Muster entsprechend statt:

- Client sendet eine `InitCommunication`-Nachricht an Dispatcher
- Dispatcher wählt einen Worker aus und leitet die Nachricht weiter
- Worker acknowledgt den Kommunikationsstart
- Client sendet an Worker beliebig viele Nachrichten der Typen `Publish`, um etwas zu posten, `RetrieveMessages`, um Nachrichten zu erhalten, oder `Like`/`Dislike`, um eine Nachricht zu "liken"/"disliken".
- Client sendet eine `FinishCommunication`-Nachricht an Worker
- Worker acknowledgt diese Nachricht und die Kommunikation gilt als beendet

Abhängig von den Nachrichten, die der Client sendet, werden von den Workern verschiedene Checks durchgeführt und über einen Hilfsaktor des Typs `WorkerHelper` mit dem `MessageStore`-Aktor kommuniziert. Im Übungsframework finden Sie bereits einen rudimentären Test, der einen Kommunikationaufbau und -abbau zwischen einem Client und dem System beinhaltet. Die Codeabdeckung, die dieser Test erreicht, zählt allerdings nicht zu den in der ersten Teilaufgabe erreichbaren Punkte dazu.

# 6 Abgabedetails

**Frist:** bis Di 10.04.2018, **10:30 MEZ** (somit kurz vor Vorlesungsbeginn)
**Art der Abgabe:** Per SVN, welches vom OnlineSystem erzeugt wird und die folgende Struktur haben soll. Es sollen keine weiteren Files, insbesondere Binär-Files und Test-Resultate, eingecheckt werden. Korrigiert wird die letzte innerhalb der Frist eingecheckte Version.

**Anmerkungen:** In der Datei `readme.txt` können Sie für die Korrektur relevante Informationen angeben. Im Normalfall kann diese Datei leer bleiben. In der Datei `participation.txt` soll die prozentuelle Arbeitsaufteilung auf die Gruppenmitglieder angegeben werden. Signifikante Unterschiede zwischen Mitgliedern führen zu Punkteabzügen für die Gruppenmitglieder, die weniger zur Abgabe beigetragen haben. Die Höhe der Abzüge richtet sich nach der Differenz der Arbeitsleistung zur Durchschnittsarbeitsleistung.
**Wichtig:** Die Projekte **müssen kompilieren** und die Tests **müssen sich ausführen lassen können**. Die korrekte Implementation darf nicht als fehlerhaft oder inconclusive erkannt werden. Die Quellcode-Dateien der Implementation dürfen nicht verändert werden. Sollte dies nicht beachtet werden wird dieser Übungsteil mit `0 Punkten` bewertet.

## 6.1 SVN Ordnerstruktur

- task1
  - participation.txt
  - readme.txt
  - solution.txt
  - MessageBoardCodeCoverage
    - MessageBoard
      - alle Source-Files
      - MessageBoard.csproj
      - Properties/AssemblyInfo.cs
    - MessageBoardTest
      - alle Source-Files
      - Properties/AssemblyInfo.cs
      - MessageBoardTest.csproj
      - bin\Debug\MessageBoardTest.dll Kompiliertes Unittestprojekt
    - MessageBoard.sln Haupt-Projektdatei
  - BoundaryValueAnalysis
    - SocialActivity
      - ArgumentLogger.cs
      - SocialActivityFunction.cs
      - SocialActivity.csproj
      - Properties/AssemblyInfo.cs
    - SocialActivityTest
      - SocialActivityFunctionTest.cs
      - Properties/AssemblyInfo.cs
      - SocialActivityTest.csproj
      - bin\Debug\SocialActivityTest.dll Kompiliertes Unittestprojekt
    - SocialActivity.sln Haupt-Projektdatei