

Abschlussprüfung Sommer 2017

zum

**Mathematisch-technischer Softwareentwickler/-in (IHK)**

vor der IHK Aachen

**Entwicklung eines Softwaresystems**

**Thema:**

Carsharing Simulator

**Programmiersprache:**

C#

Prüfling: Heitbrock, Felix

Prüflingsnummer: 101 20505

Bearbeitungszeitraum: 15.05.2015-19.05.2015

Ausbildungsbetrieb: Werkzeugmaschinenlabor RWTH Aachen

Steinbachstraße 19

52074 Aachen

1. Inhaltsverzeichnis

[I Inhaltsverzeichnis i](#_Toc418841486)

[1 Benutzeranleitung 3](#_Toc418841487)

[1.1 Systemvoraussetzungen und Hinweise zum Aufruf 3](#_Toc418841488)

[1.2 Installation des Programms 3](#_Toc418841489)

[1.3 Programmstart 3](#_Toc418841490)

[2 Aufgabenanalyse 4](#_Toc418841491)

[2.1 Allgmeine Problemstellung 4](#_Toc418841492)

[2.2 Format der Eingabedatei 5](#_Toc418841493)

[2.3 Format der Ausgabedatei 6](#_Toc418841494)

[2.4 Algorithmusansatz 6](#_Toc418841495)

[3 Verbale Beschreibung des Verfahrens 7](#_Toc418841496)

[3.1 Einlesen der Eingabedatei 7](#_Toc418841497)

[3.2 Überführung der Eingabedaten in die Terminverteilung 9](#_Toc418841498)

[3.3 Berechnung der Bewertung aller Strategien 9](#_Toc418841499)

[4 Programmkonzeption 12](#_Toc418841500)

[4.1 UML Klassendiagramm 12](#_Toc418841501)

[4.2 Kompletter Programmablauf im Sequenzdiagramm 13](#_Toc418841502)

[4.3 Nassi-Shneiderman-Diagramme 14](#_Toc418841503)

[4.3.1 Main 14](#_Toc418841504)

[4.3.2 Einlesen einer Datei 15](#_Toc418841505)

[4.3.3 Erzeugung von Terminen auf Grundlage einer Strategie 16](#_Toc418841506)

[4.3.4 erzeugeAusgabe 16](#_Toc418841507)

[4.3.5 berechneBewertung 17](#_Toc418841508)

[4.3.6 berechneZeiten 18](#_Toc418841509)

[5 Abweichung von der handschriftlichen Ausarbeitung 19](#_Toc418841510)

[5.1 Interface „IEinlesen“ 19](#_Toc418841511)

[5.2 Interface „IAusgabe“ 19](#_Toc418841512)

[5.3 Klasse „Verarbeitung“ 19](#_Toc418841513)

[5.4 Klasse „BewertungStrategie“ 19](#_Toc418841514)

[5.5 Klasse „Terminverteilung“ 19](#_Toc418841515)

[6 Testfälle 20](#_Toc418841516)

[6.1 Normalfälle 20](#_Toc418841517)

[6.1.1 IHK\_Beispiel.txt 20](#_Toc418841518)

[6.1.2 Terminendung nach 12.txt 22](#_Toc418841519)

[6.1.3 Alle möglichen Normalfaelle.txt 23](#_Toc418841520)

[6.2 Sonderfälle 23](#_Toc418841521)

[6.2.1 Leere Zeile und eine Strategie.txt 23](#_Toc418841522)

[6.2.2 Führende Leerzeichen.txt 24](#_Toc418841523)

[6.2.3 Leerzeichen zwischen Strategiewerten.txt 25](#_Toc418841524)

[6.2.4 Strategien mit gleicher Gesamtbewertung.txt 26](#_Toc418841525)

[6.2.5 Kombination mehrerer Strategien mit Sonderfaellen.txt 28](#_Toc418841526)

[6.3 Fehlerfälle 30](#_Toc418841527)

[6.3.1 Leere Datei.txt 30](#_Toc418841528)

[6.3.2 Leerer Kommentar.txt 30](#_Toc418841529)

[6.3.3 Zu viele und zu wenige Zeiten.txt 31](#_Toc418841530)

[6.3.4 Falsch formatierte Zeiten.txt 32](#_Toc418841531)

[6.3.5 Ungültige Zahlenwerte.txt 34](#_Toc418841532)

[6.4 Kombination Sonder- und Fehlerfall.txt 34](#_Toc418841533)

[6.4.1 Eingabe 35](#_Toc418841534)

[6.4.2 Ausgabe in normaler Ausgabedatei 35](#_Toc418841535)

[6.4.3 Ausgabe in Fehlerdatei 35](#_Toc418841536)

[6.4.4 Diskussion 36](#_Toc418841537)

[7 Zusammenfassung und Ausblick 37](#_Toc418841538)

# Benutzeranleitung

## Systemvoraussetzungen und Hinweise zum Aufruf

Das Programm wurde unter Windows 10 (64-Bit) in der Programmiersprache C# entwickelt und getestet. Es wird eine Benutzung unter Windows 10 empfohlen, lauffähig sollte das Programm aber auch in anderen Versionen wie etwa Windows 7 sein. Benötigt wird außerdem das .NET Framework, welches mindestens in der Version 4.6 installiert sein muss.

## Installation

Zur Installation der Software muss die gelieferte ZIP-Datei entpackt werden. Alle entstehenden Verzeichnisse, bzw. Dateien sollten mit Schreib- und Leserechten für den ausführenden Benutzer versehen werden.

## Programmstart

Standardmäßig wird das Programm ausgeführt indem die Batch-Datei „run.bat“ im Hauptverzeichnis ausgeführt wird. Es werden dann anschließend alle Dateien im Ordner „Input“ an das Programm übergeben, welches daraufhin je eine Ausgabedatei im Ordner „Output“ generiert oder ggf. eine (Fehler-) Meldung in die „Error.log“ Datei schreibt.

Anzumerken ist, dass im Verzeichnis „TestCases“ noch weitere Verzeichnisse existieren, in denen schon fertige Testfälle vorhanden sind. Auf diese Fälle wird im achten Kapitel Bezug genommen.

Das Programm lässt sich auch manuell über die Kommandozeile mit folgendem Befehl ausführen:

ABLAGEVERZEICHNIS/CarsharingSimulator.exe [in-file] [out-file] [genauigkeit]

Dabei steht „[in-file]“ für den Pfad (inklusive Name) zur Datei die eingelesen werden soll und „[out-file]“ für den Pfad (inklusive Name) zu der Ausgabedatei die generiert werden soll. Die „[genauigkeit]“ muss nicht zwingend angegeben werden (standardmäßig 0.0001).

## Ausführung der Testfälle

Die Testfälle befinden sich im Verzeichnis „TestCases“. Sollen Diese ausgeführt werden müssen die jeweiligen Dateien in den Ordner Input kopiert werden. Ist das getan können sie mit der „run.bat“ ausgeführt werden.

# Aufgabenanalyse

## Problemstellung

Für die MATSE Firma soll eine Simulation für eine Carsharing Dienstleistung erstellt werden. Innerhalb einer Stadt können Autos geliehen und an anderer Stelle zurückgegeben werden. Nachfrage und Rückgabe der Autos wird über Funktionen, genauer Polynome des 4. Gerades, beschrieben. Es soll nun der Bedarf Ermittelt werden welcher sich aus der Nachfrage und Rückgabe von Autos ergibt. Der Bedarf gibt zu welchem Zeitpunkt zwischen 0 und 24 Stunden wie viele Autos Nachgefragt wurden bzw. abgestellt wurden sind. Es soll pro Quadrat, welches einen Teil einer Stadt darstellt, je eine Bedarfsfunktion ermittelt werden. Ist die Funktion des Bedarfs bekannt, so lässt sich daraus der Endzustand der Stadt ermitteln. Der Endzustand gibt an wie neue Verteilung der Autos nach 24 Stunden aussieht. Weiterhin soll der Maximale Bedarf an Autos ermittelt werden, also zu welchem Zeitpunkt die Bedarfsfunktion maximal wird.

## Ablauf

Das Programm liest die Größe der Simulation sowie Nachfrage und Angebots Funktionen aus einer Eingabedatei ein. Anhand der Daten wird dann Die Bedarfsfunktion berechnet und die End- und Maximalbedarfszustände ermittelt. Die Ergebnisse der Simulation werden dann in eine Ausgabedatei geschrieben. Das Ganze wird als Konsolenanwendung laufen und die Dateipfade zur Ein und Ausgabedatei werden als Parameter übergeben. Als dritter Parameter kann noch die Genauigkeit übergeben. Wird diese nicht mit übergeben wird standardmäßig der Wert 0.0001 angenommen.

## Format der Eingabedatei

Die Eingabe erfolgt über eine Datei, die folgendermaßen strukturiert ist:

# AHausen

# m

1

# Polynome Nachfrage

0 0.2023761 -0.0287711 0.0016925 -0.0000352

# Polynome Abstellungen

0.434782 0 0 0 0

Abbildung Formatierung einer gültigen Eingabe

Die Erste Kommentarzeile beschreibt die Eingaben. In diesem Fall handelt es sich um Daten zu der Stadt „AHausen“, weshalb der Entsprechende Kommentar eingefügt wurde. Als Beschreibung zählt immer nur die erste Zeile, die Anderen Kommentarzeilen werden ignoriert. In der ersten Nicht-Kommentarzeile wird das m angegeben. Es bestimmt wie viele Quadrate für die Stadt simuliert werden sollen. Es werden m² Quadrate simuliert. Damit eine Berechnung sinnvoll ist, muss ganzzahlig, positiv und größer als 0 sein. Durch die Angabe von m wird auch bestimmt wie viele Polynome angegeben werden müssen. Erforderlich sind m² Polynome für die Nachfrage und nochmal genauso viele für die Abstellungen. Die Polynome besitzen maximal den Grad 4. Es gibt also pro Polynom je 5 vorfaktoren (a+bx+cx²+dx³+ex4). Diese Vorfaktoren werden durch Leerzeichen getrennt angegeben und können negative oder positive gleitkommazahlen sein.

## Format der Ausgabedatei

Die Resultate werden in einer Ausgabedatei gespeichert, die den Folgenden Aufbau besitzt:

# AHausen

Abstellung in Q\_11 zu t=2,3

Nachfrage in Q\_11 zu t=3,76

Abstellung in Q\_11 zu t=4,6

Nachfrage in Q\_11 zu t=5,87

Abstellung in Q\_11 zu t=6,9

Nachfrage in Q\_11 zu t=7,86

Abstellung in Q\_11 zu t=9,2

Nachfrage in Q\_11 zu t=9,89

Abstellung in Q\_11 zu t=11,5

Nachfrage in Q\_11 zu t=11,96

Abstellung in Q\_11 zu t=13,8

Nachfrage in Q\_11 zu t=14,03

Nachfrage in Q\_11 zu t=16,06

Abstellung in Q\_11 zu t=16,1

Nachfrage in Q\_11 zu t=18,06

Abstellung in Q\_11 zu t=18,4

Nachfrage in Q\_11 zu t=20,17

Abstellung in Q\_11 zu t=20,7

Abstellung in Q\_11 zu t=23

Nachfrage in Q\_11 zu t=23,43

Endzustand des Tages:

0

Maximaler Bedarf:

1

Abbildung Formatierung der Ausgabe

In der Ausgabedatei wird zunächst die Beschreibung (also die erste Kommentarzeile) aus der Eingabedatei eingefügt. Daraufhin folgt der Simulationsverlauf. Im Simulationsverlauf wird abgebildet wann (t), wo (Q) und was (Nachfrage bzw. Nachfrage) eine Änderung auftritt. Eine Änderung bedeutet die Abstellung eines Autos oder Nachfrage nach einem Auto. Die Einträge in der Historie haben immer folgendes Format: „[Aktion] in Q\_[x][y] zu t=[t]“. Der Zeitpunkt der Änderung (also t) wird immer bis auf 2 Nachkommastellen gerundet. Nach dem Simulationsverlauf folgt die Zeile „Endzustand des Tages:“, nach der die Endzustände ausgegeben werden. Dabei wird zeilenweise mit einem Leerzeichen getrennt die Endergebnisse der Quadrate ausgegeben. Es gibt m² Ergebnisse die in m² Zeilen ausgegeben werden. Die Endzustände sind Ganzzahlig und können negative werte annehmen. Nach der Zeile „Maximaler Bedarf:“ folgt der Maximale Bedarf pro Quadrat. Diese werden genau wie die Endergebnisse zeilenweise ausgegeben mit m Werten pro Zeile und m Zeilen also insgesamt m² Werten. Der Maximale Bedarf kann nicht negativ werden da der Startwert für jeden Bedarf bei 0 liegt. Der maximale Bedarf ist also eine positive ganze Zahl größer oder gleich 0.

## Module

Um die geforderten Anforderungen erfüllen zu können werden mehrere Module im Programm benötigt. Es wird ein Modul benötigt um die Eingabedatei einzulesen und in eine Geeignete Datenstruktur zu überführen und die Ausgabedatei zu schreiben. Weiterhin wird ein Modul für die Berechnung der Bedarfsfunktion und eines für die Berechnung der Simulation bzw. des Maximalen Bedarfs und Endzustands benötigt.

## Genauigkeitsprobleme und Laufzeit

Das Programm rechnet mit Gleitkommazahlen. Es kann also sehr leicht zu rundungsfehlern kommen. Die Berechnungen bauen jedoch in diesem Anwendungsfall nicht aufeinander auf, es wird also nicht mit einem Ergebnis einer ungenauen Rechnung weiter gerechnet. Jede Nullstelle wird erneut berechnet, es kommt also nicht zu großen Problemen durch Ungenauigkeiten die sich im Verlaufe der Berechnung aufaddieren und das Ergebnis in großem Maße beeinflussen. Allerdings wird bei der Bestimmung der Nullstellen auf ein Numerisches Verfahren gesetzt, das Bisektionsverfahren. Es wird also in den meisten Fällen nur eine Nahrung der Nullstelle bestimmt. Wie genau diese Nahrung ausfallen soll lässt sich über den 3. Kommandozeilenparameter einstellen. Standardmäßig wird als Genauigkeit 0.0001 verwendet. Eine Genauigkeit von 0.001 bedeutet dass 18 Iterationen (da Start Intervall von 0 bis 24) des Bisektionsverfahrens angewendet werden und das Intervall in dem sich die Nullstelle befindet nun 0.0001 groß ist. Das Genauigkeitsproblem ist also nicht kritisch und lässt sich vernachlässigen.

Ein größeres Problem stellt die Größe des Problems dar. Mit der Größe der Stadt also der Anzahl an Quadraten (Abstellplätze für Autos), steigt auch die Anzahl an Bedarfsfunktionen die berechnet werden müssen. Ein weiteres Problem ist die Steigung der Nachfrage- und Angebotsfunktionen. Liegt eine starke Steigung vor müssen besonders viele Nullstellen berechnet werden. Bei der Funktion z. B. müssen schon 1382400 Nullstellen berechnet werden. Damit nimmt die Größe des Problems besonders schnell zu was bedeutet das deutlich mehr Rechenschritte benötigt werden und die Berechnung insgesamt sehr lange dauern kann.

## Sonderfälle und Fehlerfälle

# Verbale Beschreibung des Verfahrens

Wie schon in der Aufgabenanalyse beschrieben werden mehrere Module benötigt damit das Programm die Eingabedaten einlesen, die Simulation berechnen und das Ergebnis in eine Ausgabedatei schreiben kann. Diese sind:

1. Einlesen & Überführen in Datenstruktur
2. Berechnung von Bedarfsfunktion
3. Simulation bzw. Berechnung von Endzustand und Maximalem Bedarf
4. Ausgabe

Hinweis: Modul und Klassenstruktur sind meist nicht identisch

* Unabhängig von programmiersprache
* Begrife der OOP erlaubt (Klasse etc)
* Beschreibung wesentlicher Module
  + Überführung in eine geeignete Datenstruktur
* Kategorisierung Normal-, Sonder- und Fehlerfälle
* Anhand der verbalen Beschreibung soll ein Leser den Algorithmus „auf dem Papier“ ausführen können und die verschiedenen Sonder- und Fehlerfälle kategorisieren können.

## Einlesen der Eingabedatei

Das Programm wird mit Argumenten gestartet. Es enthält neben dem Verzeichnis, aus dem Eingabedateien eingelesen werden sollen, eine Dateiendung, die spezifiziert, welche Dateien aus diesem Verzeichnis gelesen werden sollen. Wird das Verzeichnis nicht gefunden, ist kein gültiges Verzeichnis vorhanden oder existiert der Pfad nicht, wird das Programm abgebrochen und eine Fehlermeldung auf der Konsole ausgegeben. Bei fehlerfreier Überprüfung wird für jede Datei in diesem Verzeichnis überprüft, ob die Dateiendung der übergebenen Endung entspricht. Ist dies nicht der Fall, wird die nächste Datei überprüft. Für jede Datei mit entsprechender Dateiendung wird dann zusätzlich die Lesbarkeit dieser Datei festgestellt. Kann die Datei nicht gelesen werden, wird die nächste Datei untersucht.

… bla bla blup …

## Verarbeitung

…

## Schreiben in Ausgabedatei

# Programmkonzeption

* Nennung der Programmiersprache
* Fehlermeldungskonzept (z. B. Eigene Exception Klasse)
* UML Klassendiagram (trennung Datenspeicherung, Anwendung und Präsentation)
  + Richtige verwendung von UML Pfeilen etc …
* Wesentliche Abläufe in Form eines UML-Sequenzdiagramms
* Ggf. Modulplan falls erforderlich
* Anhand der physikalischen Datenstruktur und der verbalen Beschreibung soll ein Leser (als ausgebildeter Programmierer) in der Lage sein, den Algorithmus in der gewählten Programmiersprache zu implementieren und im Wesentlichen zum gleichen Ergebnis zu kommen.

## UML Klassendiagramm

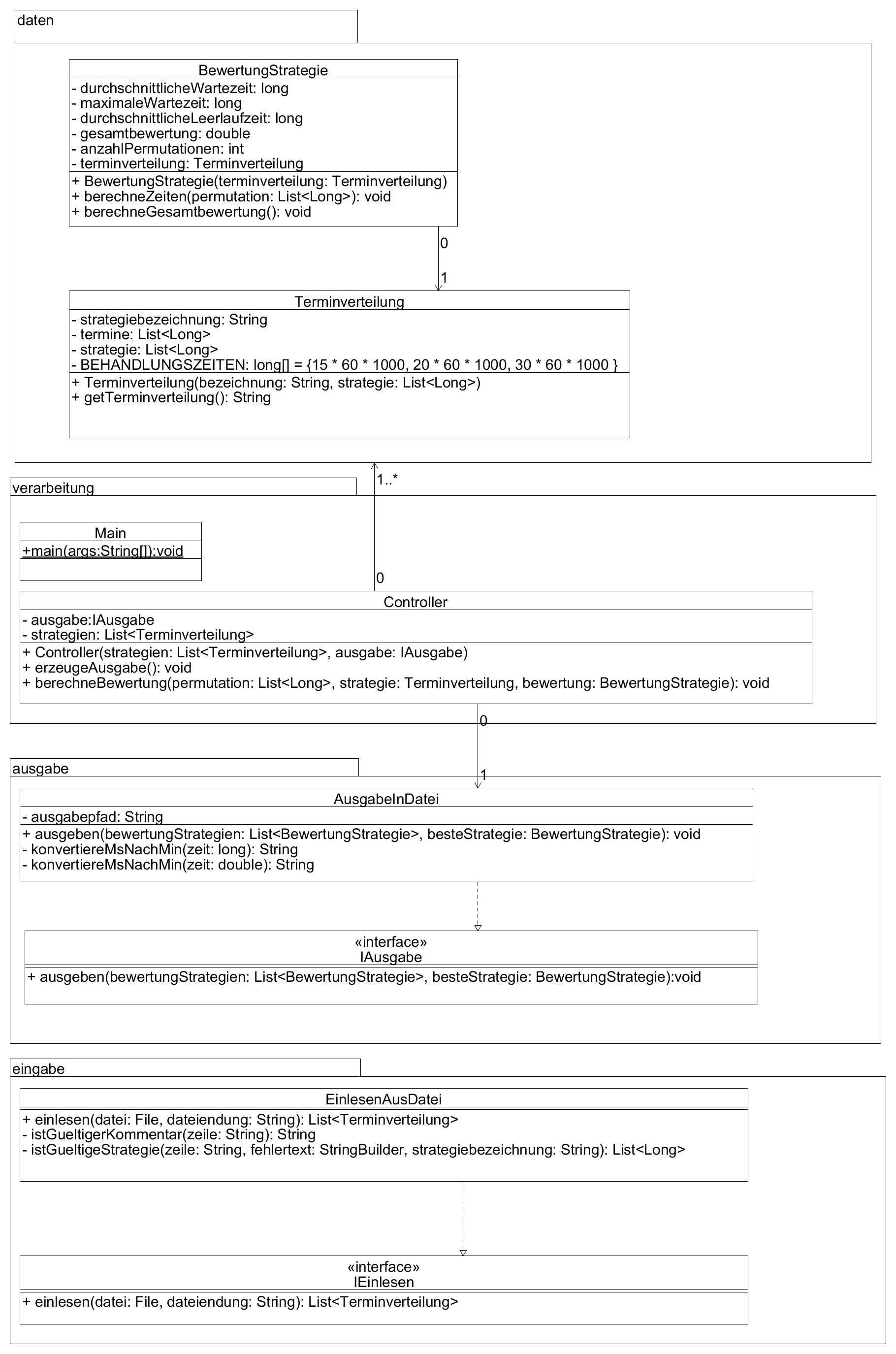
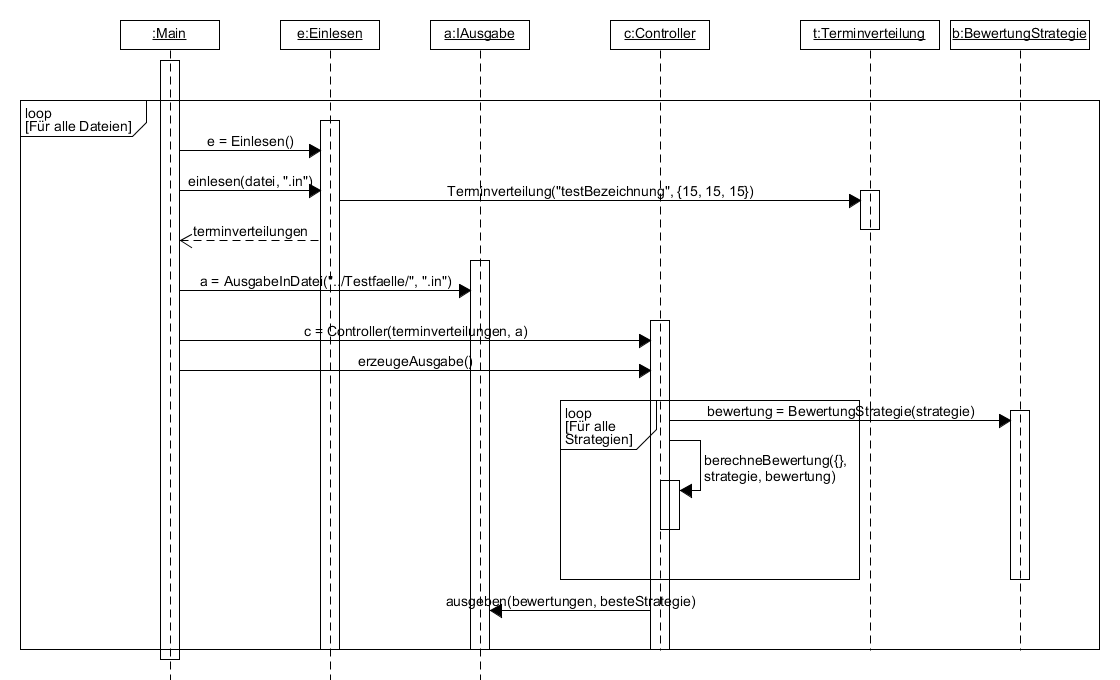


Abbildung Klassenstruktur des Programms und Beziehung untereinander

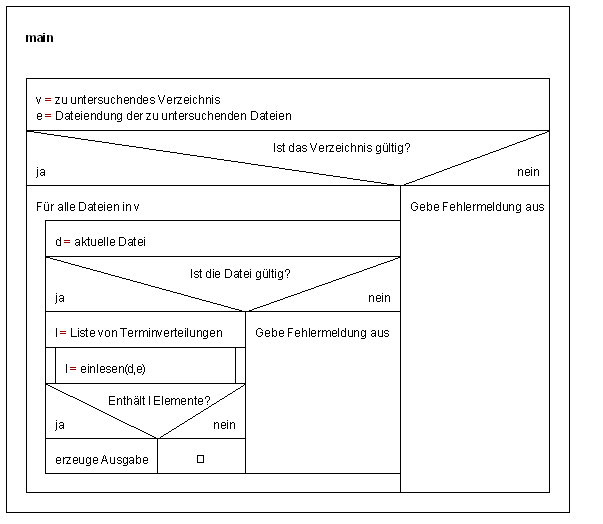
## Kompletter Programmablauf im Sequenzdiagramm



## Aktivitätsdiagramme

* Abbildung wesentlicher Funktionen als Diagramme
  + keine programmiersprachenspezifischen Funktionsaufrufe (höchstens als zusätzlicher Hinweis)
  + Klassenzugehörigkeit angeben
  + Abfragen sollten in „Umgangssprache“
  + Funktionsaufrufe sollten erkennbar
* Anhand der UML-Diagramme und formalen Beschreibung sollte man den Algorithmus implementieren. Von den hier im Konzept vorgegebenen Beschreibungen sollte in der Ausarbeitung möglichst wenig abgewichen werden bzw. Abweichungen nicht nur im Detail sollten dargestellt und begründet werden.

### Main



… etc.

# Abweichung von der handschriftlichen Ausarbeitung

Am ursprünglichen Konzept haben sich im Laufe der Bearbeitung einige Änderungen ergeben. Im Folgenden werden diese behandelt.

## Titel Änderung 1

## Titel Änderung 2 …

# Testfälle

* Einteilung in Kategorien (z. B. Whitebox & Blackbox oder Normal- und Sonderfälle etc.)
* Abdeckung aller in der Ausarbeitung definierter Normal-, Grenz- und Fehlerfälle
* Effiziente Auswahl von Testfällen
* Diskussion für Testfälle
  + Welchen Fall/welche Fälle deckt das Beispiel ab? Warum kommt das Ergebnis so zustande? Welche Zweige des Programms werden durchlaufen?

~~In diesem Kapitel werden Testeingaben nach dem Prinzip des Black-Box-Testings durchgeführt. Das bedeutet, dass nicht die Implementierung des Programms, sondern nur das Verhalten nach außen, also die Funktionsweise des Programms, untersucht wird. In diesem Fall werden die Ausgaben des Programms bei einer zahlreichen Anzahl an Eingabedaten überprüft und dann validiert, ob die Testergebnisse mit dem erwarteten Ergebnis übereinstimmen.~~

~~Zunächst werden die Beispiele aus der Aufgabenstellung untersucht. Im Anschluss daran werden Normal-, Sonder- und Fehlerfälle untersucht.~~

~~Als Normalfälle gelten diejenigen Fälle, die den Konventionen eines gültigen Eingabeformats entsprechen. Das heißt, dass eine gültige Eingabedatei mindestens aus zwei Strategien besteht, die miteinander verglichen werden können. Jede Zeile wechselt zwischen gültigem Kommentar und gültiger Strategie, wobei mit einem Kommentar begonnen wird. Die Gültigkeit einer Strategie und einer Strategiebezeichnung sind im Unterkapitel „Format der Eingabedatei“ im Kapitel „Aufgabenanalyse“ genau erklärt.~~

~~Als Sonderfälle gelten hingegen diejenigen Fälle, bei denen die Formatierung einer Kommentarzeile bzw. einer Strategie nicht gültig ist, aber durch die Behandlung des Programms dennoch zu einer gültigen Strategie umgewandelt und anschließend bewertet werden kann.~~

~~Fehlerfälle in Eingabedaten sind alle Fälle, die keine Sonder- und keine Normalfälle sind. Fehlerfälle können also nicht so umgewandelt werden, dass sie zur weiteren Nutzung durch das Programm verwendet werden können. Fehler werden in einem Fehlerprotokoll dokumentiert.~~

## Normalfälle

…

### IHK\_Beispiel.txt

#### Eingabe

…

#### Ausgabe

#### Diskussion

Durch Vergleich mit dem Beispiel ist festzustellen, dass die Ergebnisse fast identisch sind. Die einzigen Ausnahmen bilden die Gesamtbewertungen der Strategien Nr. 4 und Nr. 5. Dort ist jeweils die letzte Nachkommastelle um 0,0001 verschieden. Die Ursache liegt hier in der Berechnung der Zeiten in Millisekunden. Der Verlust der Genauigkeit dürfte dadurch geringer ausfallen, wodurch es zu diesem Ergebnis kommt..

## Sonderfälle

Nachfolgend werden Sonderfälle getestet. Dabei werden für jeden Sonderfall die vorherigen Sonderfälle mit in der Eingabedatei berücksichtigt.

### Leere Zeile und eine Strategie.txt

In diesem Test wird überprüft, ob Leerzeilen die Erkennung einer Strategie beeinflussen, was nicht der Fall sein sollte, da Leerzeilen ignoriert werden. Dazu wird zwischen einer gültigen Kommentarzeile und einer gültigen Strategie ein Leerzeichen eingefügt. Auch am Beginn der Datei steht eine Leerzeile. Zusätzlich wird der Sonderfall überprüft, ob nur eine Strategie angegeben ist. Dies überprüft, ob das Programm auch dann die beste Strategie bestimmen kann.

#### Eingabe

…

#### Ausgabe

…

#### Diskussion

Die Ausgabe zeigt, dass die Bewertung der Strategie erfolgt ist, obwohl eine Leerzeile die Strategiebezeichnung und die zugehörige Strategie voneinander trennt. Auch wurde validiert, dass es möglich ist nur eine Strategie anzugeben.

## Fehlerfälle

Fehlerfälle sind Fälle bei denen die Eingabedatei nicht korrekt verarbeitet werden konnte. Kommt es zu so einem Fall wird keine normale Ausgabedatei erzeugt sondern eine Fehlerdatei.

### Leere Datei.txt

Eine inhaltlich leere Datei wird als Fehler gewertet. Die folgende Eingabe besteht aus Leerzeilen und einer verschiedenen Anzahl an Leerzeichen in jeder Zeile (durch Markierung sichtbar machen). Es wird erwartet, dass dieser Fall als Fehler in der Protokolldatei erscheint.

#### Eingabe

#### Ausgabe in Fehlerdatei

ERROR: Die Datei Leere Datei.txt ist leer.

#### Diskussion

Der Erwartung entsprechend wurde die Fehlermeldung in das Fehlerprotokoll geschrieben. Eine Ausgabedatei wurde nicht geschrieben, da keine einzige gültige Strategie existiert.

## Kombination Sonder- und Fehlerfall.txt

Diese Datei befindet sich im Ordner „Testfaelle/eigene\_sonderfaelle“

~~Zusätzlich zu den drei Fällen muss noch ein weiterer Fall überprüft werden, der eine Kombination aus Normal- oder Sonderfall und Fehlerfall ist. In der folgenden Eingabedatei steht in der ersten Zeile ein gültiger Kommentar. Danach folgen eine Fehlerzeile, eine Leerzeile, eine weitere Fehlerzeile, ein ungültiger Kommentar und schließlich eine Strategie, die durch zu viele Leerzeichen in den Zwischenwerten einen Sonderfall darstellt. Die Erwartung ist, dass die erste Zeile die Strategiebezeichnung der letzten Zeile darstellen wird, da das Programm Fehlerzeilen und Leerzeilen ignoriert. Die Fehler sollten aber entsprechend in die Fehlerdatei geschrieben werden.~~

### Eingabe

…

### Ausgabe in normaler Ausgabedatei

…

# Zusammenfassung und Ausblick

* Zusammenfassung (ggf. auch von auftretenden Problemen)
* Mögliche Verbesserungen und Erweiterungen
  + Z. .B wenn die Grenzgrößen des Verfahrens erweitert werden

Im Rahmen des Prüfungsprodukts wurde ein Programm entwickelt, das …

# Entwicklungsumgebung

* Eingesetzte Hardware und Software
* Version d. Compilers, Sprache , Tools etc.

## Hardware

Entwickelt wurde auf einem ASUS ROG GL552VW mit folgender Ausstattung:



## Betriebssystem

Als Betriebssystem kam Windows 10 Home in der 64-Bit Version zum Einsatz.

## Entwicklungsumgebung

Als IDE (integrated development environment) wurde Microsoft Visual Studio Enterprise 2015 in der Version 14.0 Update 3 verwendet.

## Diagramme

Die UML Diagramme (Klassendiagramm, Ablaufdiagramm, etc.) wurden in Microsoft Visual Studio Enterprise 2015 in einem Modellierungsprojekt erstellt. (Aus dem UML Klassendiagramm kann man Code generieren, jedoch das Diagramm nicht updaten lassen wenn Änderungen im Code stattfinden.)

## Dokumentation

Zum Erstellen dieser Dokumentation wurde Microsoft Word 2016 verwendet.