Objektorientierte Programmierung

09 - Analyse

Alexander Stuckenholz

Version 2022-05-04

Inhalt

Softwaredesign

Bislang haben wir uns ausschließlich mit der Programmierung beschäftigt.

- Bevor wir aber Programmieren können, müssen einige Vorbedingungen erfüllt sein.
- Wir benötigen eine Vorstellung davon, was in welcher Weise umgesetzt werden soll.

Das Ergebnis dieser Vorüberlegungen ist das **Design** bzw. der **Entwurf**.

- Das Design beschreibt, aus welchen Teilen die Software im Inneren besteht und wie diese Teile zusammenwirken.

Das Softwaredesign ist immer vorhanden!

- Es entsteht auch dann, wenn man sich darüber keine Gedanken macht.

Die Alternative zu gutem Design ist schlechtes Design, nicht überhaupt kein Design ([1]).

- Softwaresysteme, bei denen das Design rein zufällig entstanden ist, werden auch als *big ball of mud* bezeichnet.

Echte **Softwareentwicklung** ist weit mehr, als nur Programmieren!

- Das Design des Softwaresystems sollte vor der Programmierung explizit festgelegt werden!
- Dies ist aber nicht der einzige wichtige Schritt zu qualitativ hochwertiger Software.

Wie man in einem geordneten Prozess von den Anforderungen bis zum lauffähigen Softwaresystem gelangen kann, wird durch die Disziplin der **Softwaretechnik** (*engl. software engineering*) beantwortet.

Die Softwaretechnik hat das Ziel, Prinzipien, Methoden und Werkzeuge bereitzustellen und zielorientiert anzuwenden, so dass eine arbeitsteilige Entwicklung von umfangreichen Softwaresystemen möglich wird, siehe [2], [4], [3].

Kernprozesse der Softwaretechnik

Die Softwareentwicklung umfasst fünf elementare Kernprozesse:

1. **Planung**: Es werden Anforderungen erhoben (Lasten- bzw. Pflichtenheft).
2. **Analyse**: Es wird ein Informationsmodell erstellt und die elementare Verarbeitungslogik definiert.
3. **Entwurf**: Die sog. Softwarearchitektur wird festgelegt.
4. **Programmierung**: Der Programmcode wird umgesetzt.
5. **Qualitätssicherung**: Die Qualität der Umsetzung wird verifiziert und validiert.

Vorgehensmodelle

Wie und in welcher Reihenfolge diese Kernprozesse der Softwareentwicklung abgearbeitet werden, legt ein **Vorgehensmodell** fest.

- Das einfachste dieser Vorgehensmodelle wird als **Wasserfallmodell** bezeichnet.
- Darin wird jeder Prozess erst vollständig abgearbeitet, bevor zum nächsten Prozess fortgeschritten werden kann.

In der Praxis hat sich aber gezeigt, dass es nicht sinnvoll ist, z.B. erst alle Anforderungen komplett aufzuschreiben, bevor die Entwicklung beginnen kann.

- Es haben sich eher agile Verfahrensmodelle etabliert, z.B. Scrum oder Kanban.

Darin werden für kleinere Produktteile alle Prozesse bearbeitet.

- Dies sorgt dafür, dass schneller entsprechende Ergebnisse sichtbar werden.

Planung

HOCHSCHULE HAMM-LIPPSTADT

Was genau umgesetzt werden soll, wird in der **Planungsphase** festgelegt.

- Das Ergebnis sind genaue funktionale und nicht funktionale Anforderung an das Produkt.

Funktionale Anforderungen beschreiben, was ein Produkt leisten soll.

- Nicht funktionale Anforderungen beschreiben Qualitätsmerkmale, z.B. wie schnell eine Reaktion erfolgen muss usw.

Die Anforderungen können z.B. in Form von Anwendungsfällen (*engl. use cases*), User Stories oder Prozessabläufen dargestellt werden.

- Sie werden in Lasten-, Pflichtenheften oder Backlogs gesammelt.

Anforderungen so zu erheben und zu verwalten, dass sie eine gute Vorstellung vom finalen Produkt liefern, ist eine Kunst für sich (*engl. requirements engineering*).

- Damit wollen wir uns hier aber nicht weiter befassen.

Analysephase und UML

HOCHSCHULE HAMM-LIPPSTADT

An die Planungsphase schließt sich die **Analysephase** an.

- Aus den Anforderungen wird hier ein erstes Informationsmodell bzw. ein Modell der elementaren Verarbeitungslogik abgeleitet.
- Ein solches Modell ist dabei eine erste Vorstellung, eine Idee, ein grober Plan.

Modelle, z.B. der Bauplan von einem Gebäude, werden meist grafisch dargestellt.

- Auch in der Softwareentwicklung hat sich die grafische Modellierung durchgesetzt.
- Für die Notation dieser grafischen Darstellung haben sich gewisse Standards etabliert.

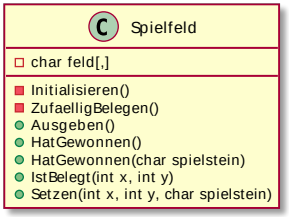
Die **Unified Modeling Language (UML)** ist eine normierte Modellierungssprache, siehe [5].

- Die UML definiert unterschiedliche Diagrammarten, um die Struktur oder das Verhalten eines Systems zu beschreiben.

Klassen in der UML

Das Klassendiagramm ist sicherlich das wichtigste Diagramm der UML, um die Struktur eines Systems zu beschreiben.

- Klassen werden im Klassendiagramm als Rechteck dargestellt.
- Im oberen Teil findet sich der Klassenname, darunter die Variablen und die Methoden.



Variablen und Methoden können zudem mit ihrer Sichtbarkeit dargestellt werden.

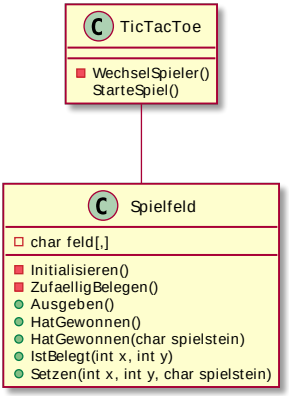
- Die rot markierten Elemente sind privat, die grün markierten öffentlich.
- Mitunter werden auch andere Symbole für die Sichtbarkeit benutzt (+, -, #).

Assoziationen

HOCHSCHULE HAMM-LIPPSTADT

Eine Klasse bildet den Bauplan für konkrete Objekte.

- Eine **Assoziation** stellt den Bauplan für konkrete Objektbeziehungen dar.
- Nur wenn zwischen Klassen eine Assoziation vorhanden ist, können die Objekte dieser Klassen Beziehungen ausprägen.



Assoziationen werden im Klassendiagramm als Linie zwischen Klassen dargestellt.

- Diese Linie ist zunächst ungerichtet.
- Das bedeutet, dass die Objekte auf beiden Seiten der Beziehung einander kennen.

Klassendiagramme auf Basis von Klassen und Assoziatonen erlauben die Modellierung beliebig komplexer Informationsmodelle.

Multiplizität

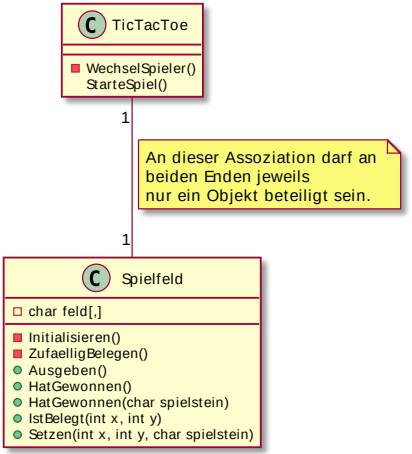
Eine Assoziation wird durch **Multiplizitäten** weiter spezifiziert.

- An jedem Ende der Assoziation findet sich eine solche Multiplizität.
- Sie legt fest, wie viele Objekte zur Laufzeit an der Beziehung beteiligt sein müssen.

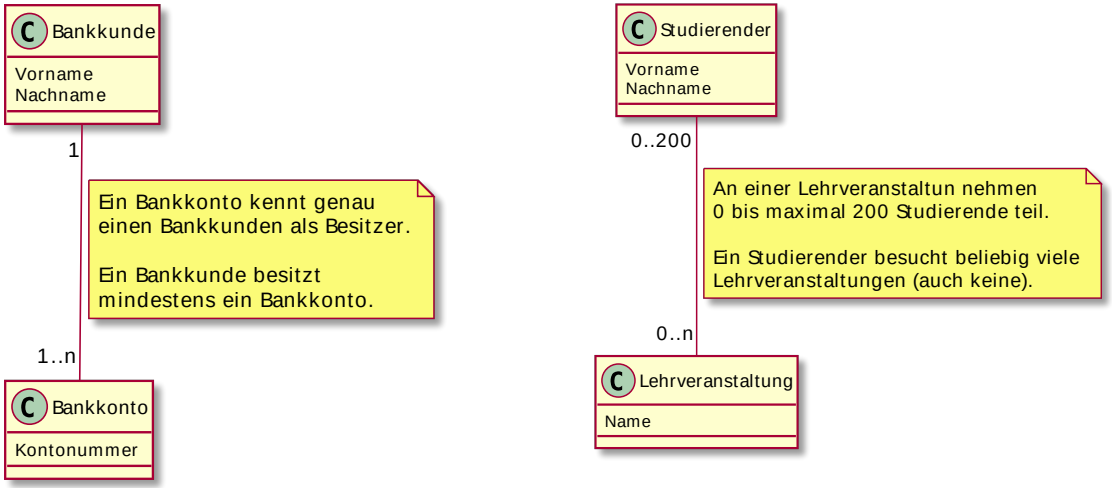
Bei der Assoziation zwischen der Klasse *TicTacToe* und dem *Spielfeld* muss an beiden Enden der Assoziation jeweils ein Objekt beteiligt sein.

- Ein Objekt der Klasse TicTacToe muss also ein Objekt der Klasse Spielfeld kennen (und anders herum).
- Ansonsten ist das System in keinem erlaubten Zustand.

Multiplizitäten machen also wichtige Vorgaben für das System.



Beispiele für Multiplizitäten im Klassendiagramm



Optionale Beziehung

Die Multiplizitäten machen im Modell wichtige Vorgaben für gültige Systemzustände.

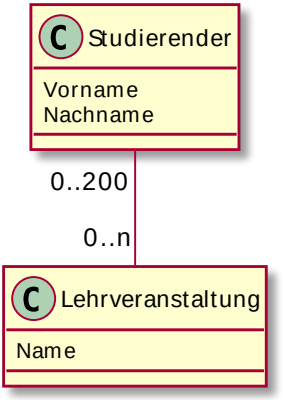
- Ein System muss sich zur Laufzeit an diese Modellvorgaben halten.

Eine Assoziation ist optional, wenn zur Laufzeit einem Objekt ein anderes Objekt zugeordnet werden kann, aber nicht muss.

- Dies wird durch die Untergrenze 0 in der Multiplizität ausgedrückt.

Im Beispiel kann einem Objekt der Klasse Studierender zur Laufzeit eine Menge von Lehrveranstaltungen zugeordnet werden.

- Aber auch keine Lehrveranstaltung zuzuordnen wäre eine gültige Ausprägung des Modells.



Verpflichtende Beziehung

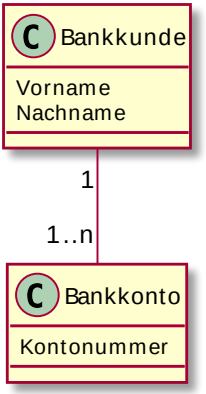
Eine Assoziation ist verpflichtend, wenn zur Laufzeit einem Objekt ein anderes zugeordnet werden muss.

- Dies wird durch die Untergrenze $n > 0$ in der Multiplizität ausgedrückt.

Im Beispiel muss dem Bankkunden zur Laufzeit mindestens ein Bankkonto bekannt sein.

- Ebenso muss ein Objekt der Klasse Bankkonto genau einen Bankkunden kennen.
- Sind diese Bedingungen nicht erfüllt, ist das System in keinem erlaubten Zustand.

Der Programmierer, der dieses Modell umsetzt, muss dafür Sorge tragen, dass diese Regeln jederzeit eingehalten werden.



Navigierbarkeit

Bislang waren alle Assoziationen ungerichtet.

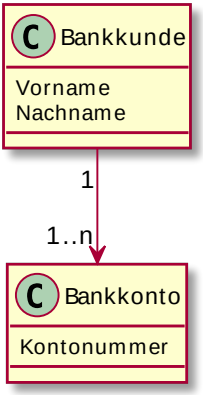
- Jede Seite kennt dann die Objekte der jeweils anderen Seite.

In der Modellierungsphase ist das sicher auch in Ordnung.

- Für die Programmierung erhöht dies aber den Aufwand.
- Die Objekte in den Beziehungen müssen auf beiden Seiten jeweils synchron gehalten werden.

Spätestens vor der Programmierung wird man daher die **Navigierbarkeit** der Assoziation einschränken.

- Man kann dann nur noch von genau einem Objekt zur anderen Seite der Beziehung navigieren.
- Eine solche Einschränkung äußert sich bei der Assoziation durch einen kleinen Pfeil.



Aggregation und Komposition

HOCHSCHULE HAMM-LIPPSTADT

Zwei speziellere Arten der Assoziation sind die **Aggregation** und die **Komposition**.

- Beide beschreiben eine Teile/Ganzes-Beziehung, die eine baumartige Hierarchie darstellt.
- Für das Ganze (die Wurzel der Hierarchie) wird oft die Bezeichnung Aggregat benutzt.
- Die Einzelteile (die Kindknoten) werden als Komponenten bezeichnet.

Folgende Beispiele werden durch eine Aggregation bzw. eine Komposition abgebildet:

- Eine Mannschaft besteht aus Spielern.
- Ein Haus besteht aus Zimmern.
- Ein Auto besteht aus einem Motor, einem Chassis und Rädern.

Ob die Komponenten unabhängig von dem Aggregat existieren können, entscheidet darüber, ob es sich um eine Aggregation oder um eine Komposition handelt.

Aggregation

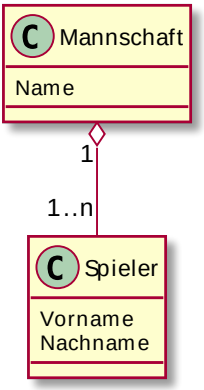
Bei der Aggregation können die Komponenten unabhängig vom Aggregat existieren.

- Die Komponenten können auch Teil eines anderen Aggregats sein.

Ein Beispiel für eine Aggregation sind die Spieler einer Mannschaft.

- Die Spieler sind Teil einer Mannschaft.
- Sie können aber auch ohne die Mannschaft existieren und auch Teil einer anderen Mannschaft sein.

Im Klassendiagramm wird eine Aggregation durch eine ungefüllte Raute an der Aggregatklasse darstellt.



Komposition

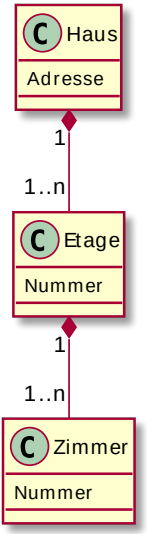
Bei der Komposition können die Komponenten nicht unabhängig vom Aggregat existieren.

- Die Komponenten können auch nur Teil eines einzigen Aggregats sein.
- Wird das Aggregat zerstört, werden auch die Komponenten zerstört.

Ein Beispiel für eine Komposition sind die Etagen eines Hauses.

- Das Haus besteht aus Etagen.
- Die Etagen können aber nicht ohne das Haus existieren.
- Auch kann eine Etage nicht Teil eines weiteren Hauses sein.

Im Klassendiagramm wird eine Komposition durch eine gefüllte Raute an der Aggregatklasse darstellt.



Vererbung

Auch die Vererbung kann im Klassendiagramm abgebildet werden.

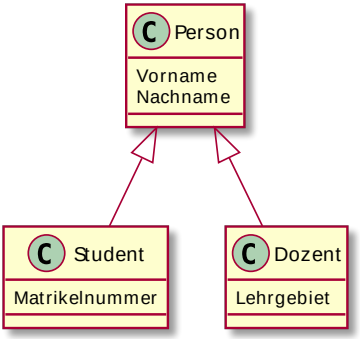
- Leitet eine Kindklasse von einer Basisklasse ab, zeigt ein geschlossener Pfeil von der Kindklasse zur Basisklasse.

Multiplizitäten existieren bei der Vererbung nicht.

- Entsprechend dürfen auch keine an den Vererbungspfeil angetragen werden.

Ebensowenig müssen die geerbten Variablen und Methoden in den Kindklassen aufgeführt werden.

- Durch die Vererbung sind sie automatisch in den Kindklassen vorhanden.



Modellbildung

In der Analysephase wird ein erstes Modell des Systems entwickelt.

- Das Klassendiagramm ist dafür ein wichtiges Werkzeug.

Dabei muss die Frage beantwortet werden, welche Klassen und Assoziationen die Anforderungen am besten abdecken.

- Wie viele Objekte sind zur Laufzeit an den Beziehungen beteiligt (Multiplizitäten)?
- Sind Teile/Ganzes-Beziehungen vorhanden (Aggregation oder Komposition)?
- Gibt es Ähnlichkeiten zwischen den Klassen (Vererbung)?

Dieser Prozess wird auch als **Modellbildung**, bzw. **Modellierung** bezeichnet.

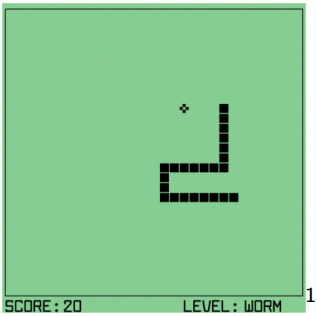
- Dabei geht es zunächst gar nicht um Technologie, wie z.B. Datenbanken, Oberflächen usw.
- In der Analysephase geht es lediglich um die **Fachlichkeit** des Systems.

Beispiel

HOCHSCHULE HAMM-LIPPSTADT

Wir wollen das bekannte Computerspiel **Snake** umsetzen.

- Bei diesem Spiel wird eine Schlange über den Bildschirm gesteuert.
- Die Schlange soll zufällig platzierte Äpfel fressen, um Punkte zu erhalten.
- Stößt die Schlange mit dem Kopf gegen eine Wand oder gegen den eigenen Körper, ist das Spiel verloren.



[1]

Die Schlange bewegt sich in eine von vier Richtungen über den Bildschirm.

- Die Schlange besteht aus einzelnen Blöcken, die nach und nach über den Bildschirm ziehen.
- Frisst die Schlange einen Apfel, wird sie dadurch länger.

[1] Bildquelle: `https://www.chip.de/downloads/Snake_18757550.html`

Vorgehen

HOCHSCHULE HAMM-LIPPSTADT

Unsere Aufgabe ist es nun, die Fachlichkeit des Spiels in einem Klassendiagramm abzubilden.

- Es müssen geeignete Klassen, Eigenschaften, Assoziationen (...) gefunden werden.
- Wir müssen uns fragen, welche Objekte zur Laufzeit existieren?
- Welche Aufgabe haben sie und wie interagieren sie miteinander?

Begriffe, die in den Anforderungen vorkommen, müssen sich im Modell wiederfinden.

- Im Beispiel ist von Schlange, Körper, Kopf, Punkte usw. die Rede.
- Bei jedem Begriff muss entschieden werden, ob es sich um eine Klasse, um die Eigenschaft einer Klasse, oder um eine Beziehung handelt.
- Zudem kann man damit beginnen, zu überlegen, welche Fähigkeiten (Methoden) die Objekte besitzen müssen.

Bei dieser Modellbildung kann es durchaus zu unterschiedlichen Varianten kommen.

- Es bedarf einiges an Erfahrung, um zu guten Modellen zu kommen.

Modell von Snake

Ein Objekt der Klasse *Spiel* verwaltet die aktuelle Punktezahl.

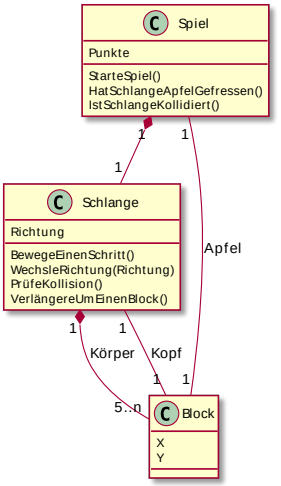
- Es muss das Spiel starten können und zwischenzeitlich prüfen, ob die Schlange einen Apfel gefressen oder mit einer Wand kollidiert ist.

Auch die Schlange wird als Klasse umgesetzt.

- Sie hat eine Richtung und besteht aus einer Anzahl von Blöcken.
- Einer dieser Blöcke ist der Kopf der Schlange.
- Die Schlange muss einige Fähigkeiten besitzen, z.B. bewegen, oder Richtung wechseln.
- Alle diese Fähigkeiten werden als Methoden umgesetzt.

Das Spiel kennt auch einen Apfel, der als Block umgesetzt ist.

- Ein einzelner Block besitzt eine Position auf dem Bildschirm.



Technische Umsetzung

HOCHSCHULE HAMM-LIPPSTADT

Ein Klassendiagramm ist ein Bauplan, der bei der Programmierung umgesetzt werden muss.

- Klassen aus dem Modell können 1:1 in C# umgesetzt werden.
- Die Assoziationen werden in Form von Objektreferenzen abgebildet.

Als Beispiel können wir die Klasse *Schlange* umsetzen.

- Die Assoziation auf die Blöcke (Körper) wird als `List<Block>` umgesetzt.
- Der Kopf ist der erste Block im Körper der Schlange.

Lediglich einige technische Details sind noch unklar.

- Wir brauchen eine Idee, wie breit und hoch das Spielfeld (ein Fenster) sein wird.
- Dann müssen wir auch in der Lage sein, Blöcke zu zeichnen.
- Das ist aber nicht Teil der Analyse, sondern des technischen Entwurfs.

Programmcode der Klasse Schlange

HOCHSCHULE HAMM-LIPPSTADT

```
1  public class Schlange
2  {
3      private int richtung = 0;
4      private List<Block> koerper = new List<Block>();
5
6      public Block Kopf
7      {
8          get { return koerper[0]; }
9      }
10
11     public void WechsleRichtung(int richtung)
12     {
13         this.richtung = richtung;
14     }
15
16     // Die restlichen Methoden können erst implementiert werden,
17     // wenn auch die technische Details (z.B. Breite/Höhe des Spielfelds) klar sind.
18 }
```

Zusammenfassung

HOCHSCHULE HAMM-LIPPSTADT

Wir haben heute gelernt, ...

- dass wir uns explizit um ein gutes Design von Software bemühen müssen.
- dass ein erstes Modell der statischen Struktur und der elementaren Verarbeitungslogik in der Analysephase entsteht.
- dass wir das Klassendiagramm der UML dazu benutzen, um die statische Struktur des Systems zu beschreiben.
- dass ein Klassendiagramm neben Klassen auch Assoziationen und Vererbung beinhaltet.
- dass Assoziationen der Bauplan für konkrete Objektbeziehungen sind.
- dass Assoziationen durch Multiplizitäten weiter beschrieben werden.
- dass Aggregation und Komposition spezielle Arten der Assoziationen sind.
- wie man aus den Anforderungen ein erstes Modell erzeugt.
- wie man dieses Modell dann technisch auf C# abbildet.

Aufgabe 1

HOCHSCHULE HAMM-LIPPSTADT

Eine Firma zur Finanzierung von Autokäufen will eine neue Online-Anwendung aufbauen.

Folgende Anforderungen wurden dazu aufgeschrieben:

- Ein Besitzer wird durch einen Namen beschrieben.
- Ein Besitzer besitzt ein oder mehrere Autos welches durch Baujahr und Modell beschrieben werden.
- Eine Bank (Standort) erteilt für ein Auto einen Kredit mit einer Kontonummer, Zinssatz und Anfangsdatum.
- Zu Krediten können Tilgungszahlungen (Datum und Höhe) existieren.
- Ein Besitzer kann eine Person oder eine Firma sein.
- Eine Person hat ein Geschlecht, eine Firma eine Umsatzsteuernummer.

Erstellen Sie das Modell der Anwendung in Form eines UML-Klassendiagramms.

- Bilden Sie jeden Sachverhalt der Anforderungen im Modell ab.

Aufgabe 2

HOCHSCHULE HAMM-LIPPSTADT

Ein Automobilhersteller will ein neues Navigationsgerät für seine Fahrzeugflotte realisieren.

- Sie sollen das Modell der Anwendung erstellen.

Folgende Anforderungen wurden dazu aufgeschrieben:

- Ein Ort besitzt einen Namen, einen Längen- und einen Breitengrad.
- Eine Verbindung hat eine Länge und eine Durchschnittsgeschwindigkeit.
- Eine Verbindung führt von einem Start-Ort zu einem Ziel-Ort (unterscheidbar!).
- Eine Region besteht aus vielen Orten und Verbindungen, eine Karte besteht aus Regionen.
- Ein Ort kann eine Stadt, eine Sehenswürdigkeit oder eine Tankstelle sein.
- Eine Route besteht aus einer Menge von Verbindungen.

Die Klasse Routenplaner soll die Funktion *BerechneSchnellsteRoute()* besitzen.

- Welche Parameter muss die Funktion bekommen? Welches Ergebnis wird sie liefern?

Aufgabe 3

HOCHSCHULE HAMM-LIPPSTADT

Erstellen Sie das Modell eines beliebigen Brettspiels, z.B. Mensch-ärgere-dich-nicht.

- Notieren Sie zunächst einige Anforderungen die darlegen, welche Elemente im Spiel überhaupt existieren und wie diese in Beziehung miteinander stehen.

Erstellen Sie dann dazu ein passendes UML-Klassendiagramm.

- Welche Fähigkeiten müssen die Objekte der Klassen zur Laufzeit besitzen?
- Gibt es alternative Modellierungen?
- Warum haben Sie sich für dieses Modell entschieden?

Quellen I

[1] Douglas Martin. *Book Design: A Practical Introduction*. 1. Dez. 1990.

[2] Helmut Balzert u. a. *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering*. 3. Aufl. 2009 Edition. Heidelberg: Spektrum Akademischer Verlag, 17. Sep. 2009. 642 S. ISBN: 978-3-8274-1705-3.

[3] Helmut Balzert. *Lehrbuch der Softwaretechnik: Entwurf, Implementierung, Installation und Betrieb*. 3. Aufl. 2012 Edition. Heidelberg: Spektrum Akademischer Verlag, 13. Sep. 2011. 614 S. ISBN: 978-3-8274-1706-0.

[4] Helmut Balzert und Heide Balzert. *Lehrbuch der Objektmodellierung: Analyse und Entwurf mit der UML 2*. 2. Aufl. 2004 Edition. Heidelberg: Spektrum Akademischer Verlag, 26. Okt. 2011. 592 S. ISBN: 978-3-8274-2903-2.

[5] Bernhard Rumpe. *Modellierung mit UML: Sprache, Konzepte und Methodik*. 2. Aufl. 2011 Edition. Berlin: Springer, 19. Aug. 2011. 304 S. ISBN: 978-3-642-22412-6.