TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Bauingenieurwesen Institut für Bauinformatik

Digitalisierung im Bauwesen: Konzepte zum Softwareentwurf

Towards Digital Construction: Advanced Concepts for Software Design

Bachelorarbeit

Fabian Collin, 4679289

Dresden, den 16. Oktober 2020

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Bauingenieurwesen Institut für Bauinformatik

Aufgabenstellung für die Bachelorarbeit

Name: Fabian Collin Matrikelnummer: 4679289

Thema: Digitalisierung im Bauwesen: Konzepte zum Softwareentwurf

(Towards Digital Construction: Advanced Concepts for Software Design)

Zielsetzung:

Die ganzheitliche Umsetzung von Konzepten der Digitalisierung im Bauwesen stellt immer noch eine Herausforderung dar. Um in einer interdisziplinären Zusammenarbeit mit Informatikern Konzepte der Digitalisierung erfolgreich in Ingenieurbüros und Bauunternehmen umsetzen zu können, brauchen Bauingenieure ein umfassendes, anwendungsbereites Grundwissen im Bereich des Software Engineering. Insbesondere das Verständnis von grundlegenden Methoden und Techniken zum Entwurf und zum Betrieb von komplexen Softwaresystemen (z.B. BIM, Analysesysteme oder Planungs- und Abrechnungssoftware) ist eine essentielle Voraussetzung.

Das Erwerben von Wissen und Fertigkeiten in einer scheinbar fachfremden Disziplin stellt Lehrende und Lernende vor besondere Herausforderungen. Auf unterschiedlichste Vorkenntnisse der Lernenden muss eingegangen werden. Verschiedene Lehr-Lernszenarien sollten unterstützt werden.

Im Rahmen dieser Arbeit soll ein flexibles Lehr-Lernkonzept für den Erwerb von Grundlagenwissen im Bereich des Software Engineering und komplementärer Fertigkeiten im Bereich des Programmierens mit einer objektorientierten Programmiersprache für Bauingenieur entwickelt werden. Besondere Beachtung ist auf die ausgewogene Vermittlung von theoretischen Grundlagen zum Softwareentwurf und der praktischen Umsetzung der Konzepte zu legen.

Innovative Lehr-Lernformen, wie z.B. ‚*eLearning*' oder ‚*blended learning*' sind zu analysieren und in den zu erarbeitenden Lösungsvorschlag einzubeziehen. Lehr-Lernangebote sind sowohl für Direktstudenten als auch für Formen des lebenslangen Lernens (z.B. Fernstudium) darzustellen.

Arbeitsumfang:
Im Rahmen der Ausarbeitung sollen die folgenden Punkte bearbeitet werden:

1. Bestandsanalyse eines universitären Ausbildungsprogrammes „Grundlagen der Bauinformatik", vorzugsweise an der TU Dresden.
2. Schwachstellenanalyse (z.B. unter Nutzung des SWOT-Ansatzes – strength, weaknesses, opportunities, threats).
3. Erarbeitung eines integrierten Ausbildungskonzeptes im Bereich Grundlagen des Software Engineering für Bauingenieure mit den Schwerpunkten Datenstrukturen und Algorithmen.
4. Erarbeitung von unterstützenden Lehrmaterial zur Vermittlung von theoretischem Wissen in den o.g. Lehrgebieten.
5. Erarbeitung von unterstützendem Lehrmaterial zur Vermittlung praktischer Programmierfertigkeiten.

Hinweise zur Umsetzung:
Vorhandenes Lehrmaterial und geeignete Fachbücher sind einzubeziehen. Es sind vorzugsweise die Programmiersprachen Java zu benutzen. Unterlagen sind in MS Word und MS Powerpoint zu erstellen. Bei der Umsetzung ist auf die an der TU Dresden übliche Lehr-Lernumgebung OPAL zurückzugreifen.

Wissenschaftlicher Betreuer:
Dipl.-Ing. Adrian Schubert
Institut für Bauinformatik
Technische Universität Dresden

Zweitprüfer:
Prof. Dr.-Ing. Raimar Scherer
Seniorprofessor
Institut für Bauinformatik
Technische Universität Dresden

Erstprüfer:
Prof. Dr.-Ing. habil. Karsten Menzel
Institut für Bauinformatik
Technische Universität Dresden

Ausgehändigt am: 21.08.2020

Einzureichen am: 27.11.2020

Prof. Dr.-Ing. habil. Karsten Menzel
Verantwortlicher Hochschullehrer

Kurzfassung

Mit Fortschreiten der Digitalisierung müssen auch Bauingenieure sich Grundwissen im Bereich des Software Engineering aneignen. Die vorliegende Arbeit beschreibt ein flexibles Lehr-Lernkonzept, welches Studenten des Bauingenieurwesens mit Begriffen und Konzepten der Softwareentwicklung vertraut macht und das Verständnis für komplexe Softwaresysteme fördert. Auf Grundlage didaktischer Handlungsempfehlungen von renommierten Pädagogen und in Kombination mit innovativen Lehr-Lernformen werden anhand praxisnaher Beispiele die Konzepte des strukturierten und des objektorientierten Programmierens vermittelt.

In Verbindung mit dieser Arbeit wird umfangreiches Lehrmaterial in Form von Präsentationsfolien, Aufgabenstellungen, Erklärvideos und Quellcode zur Verfügung gestellt.

Abstract

As digitalization progresses even civil engineers need basic knowledge about software engineering. This paper describes a flexible teaching and learning concept that brings students of civil engineering in contact with terms and concepts of software development and supports the understanding of advanced software systems. Based on didactic methods published by well-known educators and in combination with innovative teaching concepts the concepts of structured and object-oriented programming are taught using practical examples.

As part of this paper comprehensive teaching material like slides, assignments, video tutorials and source code is included.

Hinweis:

Aus Gründen der besseren Lesbarkeit werden in der vorliegenden Arbeit soweit möglich geschlechtsneutrale Formulierungen eingesetzt. Wo dies nicht der Fall ist, wird weitestgehend die grammatisch maskuline Form verallgemeinernd verwendet. Diese Formulierung umfasst gleichermaßen weibliche wie männliche Personen, die damit selbstverständlich gleichberechtigt angesprochen sind.

Inhaltsverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
AVL	Adelson-Velski und Landis
BIM	Building Information Modelling
BIW	Bauingenieurwesen
BST	Binary Search Tree
CSV	Comma-separated Values
GUI	Graphical User Interface
IFC	Industry Foundation Classes
OPAL	Online-Plattform für Akademisches Lehren und Lernen
SWS	Semesterwochenstunden
UML	Unified Modeling Language
ZEW	Leibniz-Zentrum für Europäische Wirtschaftsforschung

1 Einleitung

Einleitend werden die Beweggründe sowie die Notwendigkeit dieser Arbeit herausgestellt. Nachdem die Ziele erläutert wurden, findet eine Bestandsaufnahme statt. Um einen Überblick über diese Arbeit zu bekommen, wird abschließend der Aufbau erklärt.

1.1 Motivation

Das Baugewerbe ist einer der wichtigsten Wirtschaftssektoren in Deutschland. Mit Fortschreiten der Digitalisierung finden auch in der Baubranche moderne Informations- und Kommunikationstechnologien Einzug in Arbeitsabläufe, Prozesse und Wertschöpfungsketten (vgl. Bertschek et al. 2019: S. 9). Eine Publikation des ZEW aus dem Jahr 2019 beschäftigte sich unter anderem mit der Frage, wo das deutsche Baugewerbe im Angesicht der Digitalisierung stehe. Laut den WissenschaftlerInnen befinde sich Deutschland im europäischen Vergleich im hinteren Mittelfeld. Eine mögliche Erklärung seien die überdurchschnittlich vielen Kleinunternehmen, die in der Branche tätig sind (vgl. Bertschek et al. 2019: S. 1). Generell gelte die deutsche Bauindustrie als konservativ und traditionell (vgl. Oesterreich/Teuteberg 2016; WEF/BCG 2016, zitiert nach Bertschek et al. 2019: S. 12). Dringend benötigte Investitionen in Digitalisierungsprojekte blieben aus, obwohl viele Unternehmen angaben, die Digitalisierung als Chance wahrzunehmen. Abschließend gaben die WissenschaftlerInnen einige Handlungsempfehlungen. Unter anderem: „Unternehmen über die Potenziale der Digitalisierung und die Möglichkeiten entsprechender Vorhaben informieren und für deren Relevanz sensibilisieren." (Bertschek et al. 2019: S. 4)

An diesem Punkt knüpft die vorliegende Arbeit an. Schon früh im Studium sollten die Studierenden mit Fragestellungen der Bauinformatik in Kontakt kommen und den Stellenwert der Digitalisierung im Bauwesen begreifen. Ein umfangreiches Grundwissen im Bereich des Software Engineering kann dazu beitragen, Vorbehalte gegenüber digitaler Lösungsansätze abzubauen. Daneben sind die gewonnenen Kompetenzen Voraussetzung für eine interdisziplinäre Zusammenarbeit zwischen Informatikern und Bauingenieuren.

1.2 Zielstellung der Arbeit

Da die Informatik ein riesiges Themengebiet ist, muss bei der Erstellung des Konzeptes abgewogen werden, welche Lehrinhalte den Spagat zwischen Verständlichkeit und inhaltlicher Tiefe bzw. Komplexität zulassen. Ziel dieser Arbeit ist, ein flexibles Lehr-Lernkonzept zu entwickeln, welches das Ineinandergreifen von Algorithmen und Datenstrukturen vermittelt. Ein schlüssiger Aufbau ist ebenso wichtig wie die Korrektheit der Inhalte und das Geben von Anreizen, sich vertiefend mit der Thematik zu beschäftigen. Während des gesamten Lernprozesses sollen die Inhalte auf möglichst vielen Eingangskanälen an die Studierenden herangetragen werden, um den individuellen Bedürfnissen der gängigen Lerntypen gerecht zu werden. Klassische Lehrmethoden wie Frontalunterricht sollen durch innovative Lehr-Lernformen wie *E-Learning* ergänzt und zum *Blended Learning* ausgebaut werden. Ferner soll das Konzept sowohl für Direktstudenten als auch für Formen des lebenslangen Lernens geeignet sein. Alles in allem soll das Verständnis gefördert werden, wie die besprochenen Algorithmen und Datenstrukturen bei Fragestellungen des Bauingenieurwesens zur Anwendung kommen könnten. Das Erlernen einer Programmiersprache (in diesem Fall Java) ist dabei nicht das oberste Lernziel.

Für einen Bauingenieur ist es essenziell, die Grundbegriffe der Programmierung zu kennen. Weiterhin ist es wichtig, Programmierparadigmen wie das strukturierte und objektorientierte Programmieren anwenden zu können. Im Zusammenhang dazu steht die Kompetenz, vielfältige Algorithmen und Datenstrukturen zu dokumentieren und zu implementieren. Auch das Einlesen und Ausgeben von Dateien spielt im Alltag eines Ingenieurs eine zentrale Rolle. Neben den Informatikkenntnissen sollen auch Einblicke in Werkzeuge der Bauinformatik ermöglicht werden. So werden beispielsweise der Aufbau der Industry Foundation Classes besprochen sowie die Möglichkeiten des Building Information Modelling aufgezeigt. Allgemein soll eine erste Grundlage zum Entwurf und Betrieb von komplexen Softwaresystemen gelegt werden.

1.3 Bestandsaufnahme

Das Lehrkonzept wird für das Modul „BIW1-07 Grundlagen der Bauinformatik“ des Studiengangs Bauingenieurwesen der Technischen Universität Dresden entwickelt. Studierende im Direktstudium belegen dieses Modul seit der Reform der Studienordnung im Jahr 2020 im ersten Fachsemester. Für Fernstudierende im Bachelorstudiengang (Teilzeit) ist das Modul im sechsten Fachsemester vorgesehen. Um im Einklang mit der Studienordnung zu stehen, werden die Inhalte und Aufgaben so ausgewählt und entworfen, dass sich ein Arbeitsaufwand von insgesamt 150 Stunden für die Studierenden ergibt. Als Lehrveranstaltungen stehen pro Woche zwei SWS (Semesterwochenstunden) Vorlesung sowie vier SWS Übungen zur Verfügung.

An dieser Stelle sei erwähnt, dass sich neben diesem Modul auch noch zwei weitere Module („BIW2-09 Informationsmanagement“ sowie „BIW3-13 Weiterführende Bauinformatik“) während des Grundfachstudiums mit Problemstellungen der Bauinformatik befassen.

Im Folgenden findet eine Schwachstellenanalyse statt. Nach kurzer Beschreibung der Lehrinhalte werden unter Nutzung des SWOT-Ansatzes Stärken (strength), Schwächen (weaknesses), Chancen (opportunities) und Risiken (threats) herausgestellt.

Derzeitiger Ablauf der Lehrveranstaltungen: Im Laufe von elf Veranstaltungsblöcken (Vorlesung, Seminar, Tutorium) sollten grundlegende Algorithmen, Datenstrukturen und Programmiertechniken für Ingenieure vermittelt werden. In den folgenden Passagen wird der Fokus der Beschreibungen auf die damals durchgeführten Seminare und Tutorien gelegt.

Nach der Einführung in die Entwicklerumgebung folgten im zweiten Seminar zwei Beispiele zur Verwendung von Kontrollstrukturen. Anfangs wurde die Gaußsche Summenformel mit Hilfe von Schleifen programmiert. Anschließend wurde das klassische „Reiskörner auf Schachbrett“-Problem vorgestellt. Nach Lösung der Probleme wurden die Ergebnisse auf der Konsole ausgegeben. Im Tutorium sollten die Studierenden einen Algorithmus schreiben, der römische Zahlen in arabische Zahlen umwandelt.

Das dritte Seminar behandelte das Thema ArrayList. Es sollte ein Programm geschrieben werden, welches eine Datei mit Daten mehrerer Examen einliest, verarbeitet und die Durchschnittsnote ausgibt. Dabei musste eine bestimmte Gewichtung der Noten berücksichtigt werden. Im Tutorium bestand die Aufgabe darin, eine Datei mit Zahlen einzulesen und Minimum, Maximum, Durchschnitt und

Summe in einer Datei auszugeben. Das vierte Seminar führte in die objektorientierte Programmierung ein. Es wurden das Konzept der Vererbung vorgestellt und die ersten Objekte erstellt. Als praxisnahes Beispiel wurde die Vererbungshierarchie der IFC (Industry Foundation Classes) stark vereinfacht nachgebildet. Anschließend wurden Objekte der Klassen Wall, Slab, Door und Window erstellt. Daraufhin folgte das Sortieren dieser Objekte anhand ihrer Attribute und die Ausgabe der Daten in einer Datei. Das fünfte Seminar diente zur Einführung in den ersten Teil des Belegs, in dem es um objektorientierte Datenstrukturen, Sortieralgorithmen und Graphen ging. Außerdem mussten die Studierenden Kenntnisse über die Dokumentation von Algorithmen und Datenstrukturen nachweisen.

Nach den Weihnachtsferien wurde in die Graphentheorie eingeführt und der Dijkstra-Algorithmus auf einen Graphen angewendet. Als Beispiel sollte der kürzeste Weg in einem Straßennetz gesucht werden. Im ersten Teil der Prüfungsvorleistung sollten die Studierenden Fluchtweglängen eines Gebäudes berechnen und zu lange Wege auf der Konsole ausgeben. Die Seminare 7 bis 10 behandelten Bäume. Es wurde die rekursive Umsetzung binärer Suchbäume besprochen, anschließend minimale Spannbäume mit Hilfe des Prim-Algorithmus berechnet und darauffolgend die iterative Implementierung der binären Suchbäume gezeigt. Abschließend wurde sich mit dem Splay-Baum beschäftigt. Das letzte Seminar behandelte das Themengebiet Backtracking. Als Beispiel wurde gezeigt, wie ein Backtracking-Algorithmus zum einen rekursiv und zum anderen iterativ den Ausweg aus einem Labyrinth finden kann.

Stärken: Grundsätzlich haben die Studierenden einen vielfältigen Einblick in Algorithmen und Datenstrukturen bekommen. Mit Blick auf die Digitalisierung des Bauwesens wurden digitale Lösungsansätze vorgestellt und den Studierenden die Möglichkeit gegeben, eigene Erfahrungen zu sammeln. Vor allem Studierende mit fortgeschrittenen Programmierkenntnissen konnten ihr vorhandenes Wissen anwenden und ausbauen.

Schwächen: Aus didaktischer Perspektive war das Lehrkonzept nicht sonderlich ausgereift. Viele Studierende beschwerten sich, dass das Tempo, in dem neuer Lehrstoff eingeführt wurde, viel zu hoch gewesen sei. Des Weiteren boten die Präsentationsfolien anfangs nicht viel mehr als bloßen Text der Aufgabenstellung und Quellcode der Lösung. Im Laufe der Seminare kamen für die Algorithmen umfangreiche, dadurch aber unübersichtliche, Activity-Diagramme und Struktogramme zum Einsatz. Die Sinnhaftigkeit einiger Aufgabenstellungen kann auch infrage gestellt werden. Sie wirken sehr konstruiert und benötigen eine gewisse Fantasie, um den Bezug zum Bauingenieurwesen herzustellen. Wird die Chronologie

der Themenauswahl betrachtet, sind einige weitere Schwachstellen zu entdecken. So wurde nie eine gesamte Veranstaltung der Rekursion gewidmet, die Grundlage für Bäume und Backtracking ist. Die iterative Umsetzung von binären Suchbäumen erfüllte vorwiegend den Zweck, zu zeigen, warum diese Datenstruktur in den meisten Fällen rekursiv implementiert wird.

Chancen: Das analysierte Lehrkonzept ist inhaltlich breit aufgestellt. Die Studierenden erhalten Einblicke in zukunftsweisende Technologien des Baugewerbes und lernen erste Grundlagen im Umgang mit einer objektorientierten Programmiersprache.

Risiken: Das hohe Tempo, in dem neue Lehrinhalte eingeführt werden, sowie die unabgestimmte Reihenfolge der Themen bereiten vor allem Studierenden mit geringen Informatikkenntnissen Schwierigkeiten. Zusätzlich erschweren wenig auf Lerntypen angepasste Präsentationsfolien das Lernen. Unausgereifte und fehlerhafte Aufgabenstellungen tragen zur Frustration einiger Studierenden bei.

Fazit: Alles in allem bietet das Lehrkonzept erste Einblicke in Algorithmen, Datenstrukturen und Fragestellungen der Bauinformatik. In dieser Arbeit werden viele Inhalte aufgegriffen, in eine geeignete Reihenfolge gebracht und methodisch-didaktisch aufgearbeitet.

1.4 Aufbau der Arbeit

Die vorliegende Arbeit ist in drei Hauptkapitel unterteilt. Kapitel 2 erläutert die methodisch-didaktische Grundlage, auf der das Lehr-Lernkonzept erstellt wurde. Handlungsempfehlungen renommierter Pädagogen werden aufgegriffen und unter Berücksichtigung der Rahmenbedingungen angewendet.

Kapitel 3 beschäftigt sich mit den theoretischen Grundlagen. Dieses Kapitel ist als eine Art Werkzeugkasten zu verstehen. Thematisch abgetrennt werden die Inhalte des Lehrkonzeptes allgemeingültig erklärt. Nachdem ein Überblick über das Themengebiet geschaffen wurde, wird auf jeweils zwei Komponenten vertiefend eingegangen.

Kapitel 4 bedient sich aus diesem Werkzeugkoffer und zeigt die Umsetzung in der objektorientierten Programmiersprache Java. Jedes Unterkapitel beginnt mit einer Nennung der Lernziele. Im Anschluss folgen dann konkrete Ausführungen.

Mit der Schlussbetrachtung endet diese Arbeit.

An dieser Stelle sei auf den umfangreichen digitalen Anhang hingewiesen, der Präsentationsfolien, Aufgabenstellungen, Erklärvideos und Quellcode beinhaltet.

2 Didaktik und Methodik

Eine gängige Sichtweise ist, dass die Didaktik die Inhaltsfrage und die Methodik die Vermittlungsfrage beantwortet (vgl. Jank 2019: S. 14). Die renommierten Pädagogen Werner Jank und Hilbert Meyer zählen zur Didaktik neben der Was-Frage außerdem noch die Beantwortung weiterer Fragen. Diese werden im Unterkapitel 2.1 gestellt und Stellung dazu bezogen. Die Wie-Frage wird in den darauffolgenden Unterkapiteln beantwortet. Nachdem auf Hilbert Meyers *Merkmale guten Unterrichts* eingegangen wurde, folgen Erläuterungen zu den angewendeten Lehr-Lern-Formen und den vier Lerntypen nach Frederic Vester.

2.1 Fragen der Didaktik

Das Buch „Didaktische Modelle“ von Werner Jank und Hilbert Meyer geht in der ersten Lektion der Frage nach: Was ist Didaktik? Die beiden Pädagogen definieren die Didaktik als „Theorie und Praxis des Lernens und Lehrens.“ (Jank 2019: S. 14) Diese weite Definition spiegele wider, dass es nicht nur um Inhalte und Methoden gehe, sondern auch um Ziele und die beteiligten Personen, so Jank und Meyer.

Im Folgenden werden drei *W-Fragen der Didaktik* gestellt und mit Blick auf das in dieser Arbeit vorgestellte Lehr-Lernkonzept beantwortet:

Wer soll lernen? Als Hauptzielgruppe sind die Studierenden im ersten Fachsemester des Diplomstudiengangs Bauingenieurwesen zu nennen. Die größtenteils jungen Erwachsenen haben kaum bis keine Erfahrung mit universitären Abläufen und es ist zu erwarten, dass deren Denkweise noch stark von der Schule geprägt ist. Zudem starten die Studierenden mit unterschiedlichen Voraussetzungen in das Studium. Je nach Bundesland und Fächerwahl in der Oberstufe verfügen sie zwischen wenig bis hin zu umfangreichen Informatikkenntnissen.

In Hinblick auf das lebenslange Lernen richtet sich das Konzept auch an Studierende, die nebenbei berufstätig sind und auf diesem Weg höhere Qualifikationen in ihrem Berufsfeld anstreben. Diese sind vermehrt im Bachelorstudiengang zu finden, welches als Teilzeit-Fernstudium angeboten wird. In Gesprächen wurde des Öfteren von Seiten der Fernstudierenden angemerkt, dass ihr Schulabschluss schon viele Jahre zurückläge und deswegen das Lernen schwerfiele.

Das Spektrum der Lernenden ist demnach sehr vielfältig. Es reicht vom frisch gebackenen Abiturienten bis hin zum ausgebildeten technischen Zeichner mit jahrelanger Berufserfahrung.

Was soll gelernt werden? Die Frage nach den Lehrinhalten beinhaltet drei weitere Fragen: Woher kommen die Inhalte? Nach welchen Kriterien wird ausgewählt? Wer trifft die Entscheidung?

In der Studienordnung sind die Lehrinhalte des Moduls aufgeführt. Dennoch besteht ein Spielraum in der Abfolge sowie in der Tiefgründigkeit der einzelnen Themengebiete. Vom Hochschullehrer sind Literaturhinweise gegeben und das in dieser Arbeit vorgestellte Konzept greift größtenteils auf Literatur der Autoren Mark Allen Weiss und Sebastian Dörn zurück.

Als Kriterien der Auswahl, Gewichtung und Abfolge stehen ein umfangreicher Überblick über Fragestellungen der Bauinformatik und ein nachvollziehbarer *roter Faden* im Vordergrund. Wird sich auf Fachwissen aus anderen Gebieten des Bauingenieurwesen bezogen, wird dieses nur oberflächlich eingesetzt. Eine ausführliche Einführung und Erklärung dieses Fachwissens sind den entsprechenden Instituten vorbehalten.

Wozu soll gelernt werden? Diese Frage wurde im Unterkapitel 1.2 ausführlich beantwortet.

2.2 Merkmale guten Unterrichts

Das in dieser Arbeit vorgestellte Lehr-Lernkonzept orientiert sich in der Frage, wie das Wissen vermittelt werden kann, an den zehn von Hilbert Meyer zusammengetragenen empirisch abgesicherten Gütekriterien für Unterricht. Da der Kriterienkatalog für Schulunterricht entworfen wurde und die Studienordnung die Rahmenbedingungen des Moduls festlegt, mussten an einigen Stellen Abstriche und Anpassungen vorgenommen werden. Besonders hervorzuhebende Merkmale werden im Folgenden kurz beschrieben:

Das erste Merkmal ist die **klare Strukturierung des Unterrichts**. Damit ist gemeint, dass sich ein klar erkennbarer *roter Faden* durch den Unterricht ziehen sollte. Die Stimmigkeit von Zielen, Inhalten und Methoden ist genauso wichtig wie die Folgerichtigkeit des methodischen Gangs. Der methodische Gang beschreibt die aufeinander abgestimmte Reihenfolge der Unterrichtsschritte. Des Weiteren sollten Aufgaben-, Regel- und Rollenklarheit bestehen (vgl. Meyer 2011: S. 26ff.).

Ein weiteres Merkmal guten Unterrichts ist die **inhaltliche Klarheit**. „[Sie] liegt dann vor, wenn die Aufgabenstellung verständlich, der thematische Gang plausibel und die Ergebnissicherung klar und verbindlich gestaltet worden sind.“ (Meyer 2011: S. 55) Der thematische Gang beschreibt, wie Themen aufeinander aufbauen und Zugang zu ihnen geschaffen werden kann (vgl. Meyer 2011: S. 57).

Sinnstiftendes Kommunizieren bezieht sich auf den Austausch zwischen Lehrendem und Lernendem. Unter anderem durch Rückmeldung zum Lernfortschritt sowie zu Lernschwierigkeiten kann die Qualität auf beiden Seiten erhöht werden. Auch weiterführende Fragen tragen dazu bei, dem Lehr-Lern-Prozess und seinen Ergebnissen eine persönliche Bedeutung zu geben (vgl. Meyer 2011: S. 67ff.).

Übungsphasen haben nach Hilbert Meyer ein eigenes didaktisches Gewicht. Sie dienen „der Automatisierung des zuvor gelernten (Festigung, Routinisierung), der Qualitätssteigerung (Vertiefung) und dem Transfer (Anwendung in neuen Wissens- und Könnensbereichen)." (Meyer 2011: S. 104) Übungsphasen sollten ausreichend oft und passgenau eingeplant werden. Des Weiteren sollten gezielte Hilfestellungen angeboten werden (vgl. Meyer 2011: S. 104f.).

Die weiteren Merkmale sind: Hoher Anteil echter Lernzeit, Lernförderliches Klima, Methodenvielfalt, Individuelles Fördern, Transparente Leistungserwartungen und Vorbereitete Umgebung (vgl. Meyer 2011: S. 17f.).

2.3 Lehr- und Lernformen

Um den Studierenden nachhaltig Wissen zu vermitteln, wird der Lehrstoff auf verschiedene Sozialformen des Unterrichts aufgeteilt (vgl. Meyer 2011: S. 76). Der methodische Gang der Lehrveranstaltungen lässt sich im Wochenrhythmus als Lenkungslinie beschreiben. Dabei wird mit einer hohen Lehrerdominanz in ein Themengebiet eingeführt und mit einer entsprechend hohen Schülerdominanz abgeschlossen. Da es nach Hilbert Meyer nicht den *einen methodischen Gang* gibt, sondern sich in jeder Unterrichtsstunde mehrere ineinander verwobene Linien finden lassen, spricht er von der *methodischen Linienführung* (vgl. Meyer 2011: S. 27). Werden die unten aufgeführten Lehrveranstaltungen und Unterrichtsformen als Ganzes betrachtet, finden sich Vertrautheitslinie, Abstraktionslinie sowie Komplexitätslinie wieder. Inhaltlich wird vom Vertrauten ins Fremde, vom Abstrakten zum Konkreten und vom Einfachen zum Komplizierten gegangen (vgl. Meyer 2011: S. 28). Jede Unterrichtsform definiert feste Rollen für Studierende und Lehrpersonal. Somit ergibt sich eine klare Strukturierung.

Die Wissensvermittlung und Festigung des Gelernten werden auf folgende fünf Lehr-Lernformen aufgeteilt:

Vorlesung: Ziel ist es, den Studierenden einen Überblick über das Themengebiet zu verschaffen. Neue Lehrinhalte werden vom Hochschullehrer überwiegend allgemeingültig beschrieben und an einzelnen Beispielen erklärt. Des Weiteren sollte von

aktuellen Forschungsprojekten berichtet werden und Ausblicke in die Zukunft gewagt werden. Über Diskussionen kann angeregt werden, akademisch zu denken.

Seminar: In den Seminaren werden die wichtigsten Inhalte der Vorlesung vom Übungsleiter kurz zusammengefasst und wiederholt (Erkenntnissicherung). Darüber hinaus wird gezeigt, wie die besprochenen Inhalte in Java umgesetzt werden. Bezüglich der thematischen Gangart folgt entweder eins aus dem anderen (linearer Gang) oder mehrere kleine Teile ergeben ein großes Gesamtkonstrukt (vernetzter Gang mit Zusammenführung) (vgl. Meyer 2011: S. 58). Anhand umfangreicher Beispiele sollen die Studierenden die Möglichkeit bekommen, Themen zu verstehen und nicht-triviale Algorithmen Schritt für Schritt nachzuvollziehen. Ein Beispiel für das schrittweise Nachvollziehen eines Algorithmus ist im Seminar 9 zu finden, in dem der Dijkstra-Algorithmus erklärt wird. Des Weiteren findet am Ende des Seminars eine Einführung in das nächste Tutorium statt. Zum einen werden die zu lösenden Aufgaben erklärt. Zum anderen kommt es des Öfteren vor, dass die Aufgaben des Tutoriums in einem vorbereiteten Projekt gelöst werden müssen. In diesem Fall werden die im Projekt gegebenen Algorithmen und Datenstrukturen ausführlich besprochen.

Tutorium: Spätestens an dieser Stelle werden die Studierenden aktiv. Die im Seminar vorgestellten Aufgaben sollen nun am Computer bearbeitet werden. Als Hilfestellung können die Tutoren vor Ort und Erklärvideos online genutzt werden. Dass die Studierenden von Kommilitonen höherer Semester beim Lernen begleitet werden, wird auch als Peer Assisted Learning bezeichnet. Das Ziel ist, dass die Studierenden Inhalte aus dem Seminar auf die Aufgaben des Tutoriums übertragen und anwenden können. Die gestellten Aufgaben stellen dabei Probleme des Bauingenieurwesens und der Informatik in den Vordergrund.

Um die Verständlichkeit der Aufgabenstellungen zu verbessern, wurde auf einen einheitlichen Sprachgebrauch geachtet und auf die Verwendung von Synonymen verzichtet. Gemäß Erkenntnissen von Ernst Fürntratt-Kloep wurden die Aufgaben in ihrer Schwierigkeit von leicht nach schwer angeordnet. Der Psychologe empfiehlt diese Strukturierung, um einen höchstmöglichen Lernerfolg zu erzielen und eine größere Freude am Lernen zu bieten (vgl. Fürntratt 1978, zitiert nach Wellenreuther 2018: S. 126). Um auf die unterschiedlichen Lernvoraussetzungen der Studierenden einzugehen, wurden Aufgaben, die über das erwartete Niveau hinausgehen, als Zusatzaufgaben gekennzeichnet.

E-Learning: Freiwillige Selbsttest geben den Studierenden eine Rückmeldung über ihren Wissensstand. Auf der Online-Lernplattform Opal können zu jedem Themengebiet Multiple-Choice-Tests (zum Teil mit Drag-and-Drop-Elementen) bearbeitet

werden. Sollte eine eingegebene Lösung fehlerhaft sein, klärt ein Hinweistext über den gemachten Fehler auf und verweist auf die entsprechende Lehrveranstaltung, in der das Thema behandelt wurde. Als Anreiz zur Bearbeitung der Selbsttests kann die Vergabe von Bonuspunkten in der Klausur in Erwägung gezogen werden.

In einer Studie von 1992 wurde die Wirkung der Wiederholbarkeit von kleinen Zwischentests untersucht. Die Forscher fanden heraus, dass die Möglichkeit zum Wiederholen eines Zwischentests das Ergebnis im Abschlusstest positiv beeinflusste (vgl. Martinez/ Martinez 1992, zitiert nach Wellenreuther 2018: S. 124). Dem zur folge dürfen die Studierenden einen online Selbsttest bis zu dreimal absolvieren.

Übungsaufgabe: Die als Prüfungsvorleistung vorgesehene Übungsaufgabe wird in drei Teile aufgeteilt. Ziel ist es, eine erste Aussage treffen zu können, ob die Studierenden die Lehrinhalte verstanden haben und anwenden können.

Früh im Semester wird die Aufgabenstellung des ersten Teils der Übungsaufgabe ausgegeben. Da zu vermuten ist, dass die Programmierkenntnisse der Studierenden zu diesem Zeitpunkt eine hohe Streuweite aufweisen werden, überprüfen viele kleine Programmieraufgaben einzelne Konzepte des strukturierten Programmierens. Das Lösen der Aufgaben hat als Ziel zu motivieren, statt zu frustrieren. Die Lösungen sind einerseits online hochzuladen, andererseits aber auch in gedruckter Form abzugeben. Besonders wichtig ist, dass die Studierenden in ihrem frühen Stadium aussagekräftiges Feedback für ihre Leistung bekommen. Ein einfacher Vermerk, ob dieser Teil bestanden oder nicht bestanden wurde, ist dabei nicht zielführend. Diese Aussagen decken sich mit den Ergebnissen einer Studie von Emily Fyfe und Bethany Rittle-Johnson aus dem Jahr 2016. Die Wissenschaftlerinnen kommen zu dem Schluss, dass Feedback bei geringem Vorwissen und zu den ersten Anwendungsaufgaben besonders wirksam sei (vgl. Fyfe/Rittle-Johnson 2016, zitiert nach Wellenreuther 2018: S. 121). Des Weiteren untersuchten mehrere WissenschaftlerInnen die Wirkungen der verschiedenen Formen des Feedbacks. „Wenn nur die Information gegeben wird, ob die gegebene Antwort richtig oder falsch ist, dann geht von der Rückmeldung sogar ein negativer Effekt aus. Positive Effekte treten erst ein, wenn Informationen über die richtige Antwort oder über den Weg zur richtigen Antwort gegeben werden.“ (vgl. Wellenreuther 2018: S. 123)

Der zweite Teil wird im Rahmen des Tutoriums bearbeitet. Nach der Bearbeitung der Aufgaben und Abgabe der Lösung folgt eine zehnminütige mündliche Projektvorstellung. Als Abschluss der Übungsaufgabe wird ein großer Multiple-Choice-Test durchgeführt. Dieser Teil der Übungsaufgabe dient gleichzeitig als Vorbereitung auf die Klausur.

Grundsätzlich sollte das Lehrpersonal häufig gemachte Fehler erkennen und am Anfang des nächsten Seminars darauf eingehen (vgl. Wellenreuther 2018: S. 144). „Feedback bezieht sich nicht nur auf Informationen, die der Lehrer seinen Schülern über die Güte ihrer Leistungen gibt. Es bezieht sich auch auf Informationen, die Schüler dem Lehrer über die Verständlichkeit seiner Erklärungen geben." (Wellenreuther 2018: S. 122)

Eine Übersicht der Gliederung des Lehrkonzeptes ist in Anlage 1 zu finden.

2.4 Vier Lerntypen

Jeder Mensch ist unterschiedlich und jeder Studierende lernt unterschiedlich. In seinem Buch „Denken, Lernen, Vergessen" beschreibt der Biochemiker Frederic Vester, dass die Denk- und Auffassungsweise eines Menschen schon früh in seiner Kindheit gefestigt werden (vgl. Vester 2011: S. 44ff.). Obwohl jeder Mensch sein eigenes Denkmuster entwickelt, können daraus Grundmuster abgeleitet werden. „Je nach Grundmuster sind also die Eingangskanäle wie Sehen, Hören, Fühlen und alle damit zusammenhängenden Empfindungen recht verschieden ausgebildet und […] die Nervenleitungen […] gänzlich anders verknüpft." (Vester 2011: S. 125) Ein Zusammenhang zwischen Grundmuster und Intelligenz bestehe nicht. In der Kommunikation sei entscheidend, wie gut zwei fremde Muster miteinander harmonieren (vgl. Vester 2011: S. 48f.). Frederic Vester spricht hierbei von Resonanz, also, „dass beide Muster gleiche Schwingungen aufweisen. Das können sie aber nur, wenn sie in ihrer Struktur ähnlich sind." (Vester 2011: S. 49) Zusammenfassend schreibt er: „Lernerfolg und gute Schulleistungen liegen also nicht nur in der absoluten Intelligenz des Einzelnen (der Fähigkeit zu behalten, zu kombinieren, Zusammenhänge zu erkennen), sondern oft an der relativen Übereinstimmung zweier Muster, an der Möglichkeit oder Unmöglichkeit einer Resonanz." (Vester 2011: S. 50) Damit ein Lehrkonzept möglichst wirksam ist, empfiehlt er die Entfaltung der unterschiedlichen Lerntypen zu erlauben und den Lernenden die Möglichkeit zu geben, herauszufinden, welcher Lerntyp sie sind (vgl. Vester 2011: S. 134ff.).

Im Folgenden werden die vier Lerntypen kurz erläutert und beschrieben, wie mit einzelnen didaktischen Werkzeugen auf deren Bedürfnisse eingegangen wird bzw. eingegangen werden kann:

Der **auditive Lerntyp** lernt über das Gehörte und Gesprochene. Umgangssprachliche Erklärungen sowie der Austausch mit Kommilitonen helfen ihm, das Gelernte zu verstehen (vgl. Vester 2011: S. 51). Für diesen Lerntyp ist Frontal-

unterricht besonders geeignet (vgl. MIQR). Während der Vorlesung und des Seminars können von ihm schon viele Informationen aufgenommen werden. Zu dem kann er sich im Tutorium mit seinen Kommilitonen austauschen. Menschen, die zum auditiven Lerntyp zählen, lernen gerne mit Videos (vgl. MIQR). Idealerweise werden die Lehrveranstaltungen aufgezeichnet, sodass sich zu Hause der Mitschnitt erneut angesehen werden kann. Die für die Tutorien angefertigten Erklärvideos unterstützen diesen Lerntyp besonders. Zusätzlich wird während der Seminare gelegentlich auf externe Videos hingewiesen, die ein Themengebiet besonders gut oder weiterführend beleuchten. Videoaufzeichnungen und Erklärvideos sind in weiterer Hinsicht nützlich, da sich in ihnen vor- und zurückspulen lässt.

Der **visuelle Lerntyp** lernt über die Augen. Beobachtungen und Experimente helfen ihm beim Verstehen (vgl. Vester 2011: S. 51). Die Arbeit mit bildbasierten und anschaulichen Medien spielt eine große Rolle, da sich dieser Lerntyp visuell aufbereitete Informationen besonders gut merken kann (vgl. MIQR). Um auf dessen Bedürfnisse einzugehen wurden praxisnahe Beispiele verwendet und in den Präsentationsfolien darauf geachtet, komplexe Zusammenhänge grafisch aufzuarbeiten. Es wurde auch auf einen einheitlichen Umgang mit Farben, Formen und Mustern geachtet. Eine große Hilfe für den visuellen Lerntyp sind die Turtle-Grafiken. Die Turtle kann einfache Zeichenanweisungen umsetzen. Bewegt sie sich nach vorne, zieht sie einen Strich hinter sich her. Über die Drehen-Anweisung dreht sich die Turtle um einen gewünschten Winkel. Mittels dieser zwei simplen Funktionen lassen sich ganze Algorithmen visualisieren. Zusätzlich verfügt die Turtle über Sonderfunktionen. So können ganze Formen gezeichnet und die Farbe sowie Strichstärke angepasst werden. Des Weiteren kann sich die Turtle frei auf der Zeichenebene bewegen.

Der **haptische Lerntyp** möchte Dinge anfassen und fühlen (vgl. Vester 2011: S. 51). Damit ist auch die praktische Ausführung des Gelernten gemeint – frei nach dem Motto „Learning by doing“ (vgl. MIQR). In den Tutorien und der Übungsaufgabe kann die Theorie in die Praxis umgesetzt werden. Der haptische Lerntyp kann sich so ausprobieren und Erfahrungen sammeln.

„Der **intellektuelle Lerntyp**, auch abstrakt-verbaler Lerntyp genannt, versteht und speichert Informationen ab, indem er über die Informationen nachdenkt und sich kritisch mit diesen auseinandersetzt. Diesem Lerntyp kann eine Information, bei Vester eine mathematische Formel, gegeben werden und er speichert diese einfach ab.“ (smarter-learning.de) Um die Bedürfnisse dieses Lerntyps zu befriedigen, werden Begriffe definiert und auf weiterführende Literatur hingewiesen.

3 Theoretische Grundlagen

In diesem Kapitel werden die theoretischen Grundlagen erklärt, die benötigt werden, um mit einer objektorientierten Programmiersprache wie Java sicher umgehen zu können. Anfangs wird der Fokus auf die Algorithmen gelegt. Nachdem vier unterschiedliche Arten des Ablaufs von Algorithmen vorgestellt wurden, werden vier Algorithmenmuster erläutert.

Das zweite Unterkapitel behandelt Programmierparadigmen. Dabei wird ausführlich auf zwei Unterformen eingegangen: die strukturierte und die objektorientierte Programmierung.

Abschließend wird der Fokus auf die Datenstrukturen gelegt. Nachdem die grundlegendsten Datenstrukturen kurz erklärt wurden, folgen im Anschluss ausführliche Erläuterungen zu zwei im Bauwesen besonders häufig eingesetzten Datenstrukturen: Bäume und Graphen.

3.1 Algorithmen

Ein Algorithmus ist eine Folge elementarer Anweisungen (z. B. Grundrechenarten, logische Operationen, Standardfunktionen, Zuweisungen, etc.), die nach endlich vielen Schritten eine Lösung des gestellten Problems liefern. Um von einem Computer interpretiert werden zu können, muss ein Algorithmus folgende Eigenschaften erfüllen: Allgemeingültigkeit, Ausführbarkeit, Endlichkeit, Eindeutigkeit, Korrektheit und Effizienz (vgl. Dörn 2016: S. 4f.; Von Rimscha 2008: S. 3).

In der Literatur finden sich verschiedene Formen der Klassifizierung von Algorithmen. Sebastian Dörn (2016) stellt in dem Kapitel „Entwurfsmuster von Algorithmen“ verschiedene Arten vor. Der Begriff „Entwurfsmuster“ (engl. Design Pattern) ist allerdings in der Software-Architektur eng mit einem häufig wiederkehrenden Entwurfsproblem und seiner bewährten generischen Lösung verbunden (vgl. Balzert 2001: S. 844; Weiss 2010: S. 96). Des Weiteren findet bei ihm keine Unterteilung zwischen prinzipieller Implementierung und charakteristischen Verhaltensweisen statt.

Im Folgenden wird eine aus Sicht des Autors sinnvolle Unterteilung genutzt. Diese orientiert sich zum einen an Markus von Rimschas Gliederung des Kapitels „Arten von Algorithmen“, in dem nach Art des Ablaufs unterteilt wird. Zum anderen werden die „Algorithm Design Techniques“ (dt. Algorithmenmuster) von Mark Allen Weiss (2012) größtenteils übernommen.

3.1.1 Arten des Ablaufs von Algorithmen

Wiederholend: Iterative Algorithmen benutzen Schleifen, um einen Startwert schrittweise in die Lösung zu überführen. Anhand einer festen Handlungsabfolge wird sich mit jeder Iteration der Lösung genähert. Der Algorithmus wird beendet, wenn das Problem gelöst ist oder das Ergebnis der Lösung ausreichend nah gekommen ist (vgl. Von Rimscha 2008: S. 6). Die Implementierung iterativer Algorithmen wird im Vergleich zu anderen Arten als einfach wahrgenommen, da diese Algorithmen eine einfache Struktur aufweisen und leichter nachzuvollziehen sind. Allerdings gehen die angesprochenen Vorteile oft mit Effizienzdefiziten einher (vgl. Dörn 2016: S. 235).

Geschachtelt: Rekursive Algorithmen benutzen Selbstaufrufe, um ein Problem zu lösen. Mit jeder Inkarnation wird ein Problem auf eine einfachere Version von sich selbst zurückgeführt. Dies geschieht solange, bis ein unmittelbar lösbarer Anfang erreicht ist (vgl. Lamm 2014: S. 15). Die Rekursion ermöglicht es, Probleme elegant und mit vergleichsweise wenigen Zeilen Quellcode zu lösen. Sie hat allerdings einen hohen Speicherbedarf und die Implementierung ist vor allem für Anfänger herausfordernd (vgl. Von Rimscha 2008: S. 34).

Zufällig/Vermutend: Randomisierte Algorithmen (auch probabilistische Algorithmen) liefern lediglich eine näherungsweise oder wahrscheinlich korrekte Lösung für ein Problem. Sie kommen zum Einsatz, wenn es für ein Problem keinen geeigneten Algorithmus gibt oder eine Lösung zu lange dauern würde. Teilweise wird dabei gänzlich auf den Zufall vertraut (vgl. Von Rimscha 2008: S. 47ff.). Andere Verfahren werten Indizien aus und treffen aus ihnen heraus Entscheidungen. Dabei wird zwischen zwei Verfahren unterschieden: Las-Vegas-Algorithmen können versagen, liefern aber keine falschen Ergebnisse. Monte-Carlo-Algorithmen liefern mit einer gewissen Wahrscheinlichkeit auch fehlerhafte Resultate (vgl. Von Rimscha 2008: S. 61).

Evolutionär: Genetische Algorithmen liefern lediglich hinreichend gute Lösungen. Wie schon bei den randomisierten Algorithmen kommen sie zum Einsatz, wenn es für ein Problem keinen geeigneten Algorithmus gibt oder eine Lösung zu lange dauern würde. Anfänglich wird eine Generation mit zufälligen Genen erzeugt. Jede weitere Generation entwickelt sich aus ihren Vorgängern. Dabei wird anhand einer Fitness-Funktion die Leistungsfähigkeit bestimmt und anschließend selektiert. Die Gene des Kindes ergeben sich aus einer Kombination der Gene von Mutter und Vater. Mit einer bestimmten Wahrscheinlichkeit treten Mutationen auf. Letztendlich beendet ein Abbruchkriterium die Evolution (vgl. Von Rimscha 2008: S. 53).

3.1.2 Algorithmenmuster

Gierig: Greedy-Algorithmen starten mit einer leeren Lösung. Schrittweise wird die Teillösung erweitert. Sollten einmal mehrere Möglichkeiten zur Verfügung stehen, wird die scheinbar beste Möglichkeit ausgewählt. Es wird gehofft, dass ein lokales Optimum zu einem globalen Optimum führt (vgl. Dörn 2016: S. 248ff.).

In der Praxis wird beispielsweise der Dijkstra-Algorithmus benutzt, um den kürzesten Weg in einem Straßennetz zu finden. Der Dijkstra-Algorithmus kann auf Graphen angewendet werden, die gewichtete Kanten haben. Ein negatives Kantengewicht ist dabei nicht zulässig. Die Gier des Dijkstra-Algorithmus zeigt sich in seiner Entscheidung, in jedem Durchlauf nur den aktuell kürzesten Pfad weiterzuverfolgen. Anfangs werden alle Knoten, bis auf den Startknoten, als unbekannt eingestuft. Unbekannte Knoten liegen vorübergehend unendlich weit weg vom Startknoten. Mit jedem Durchlauf wird ein bekannter Knoten abgearbeitet. Zum Abarbeiten gehört, alle Nachbarknoten zu besuchen. Wird dabei ein neuer Knoten entdeckt, wird ein Pfad vom Startknoten zum neuentdeckten Knoten gespeichert. Wurden alle Nachbarknoten besucht, startet ein neuer Durchlauf. Der Algorithmus terminiert, wenn der Zielknoten zum ersten Mal besucht wurde oder alle Knoten abgearbeitet sind (vgl. Weiss 2014: S. 546; Weiss 2012: S. 372).

Aufteilend: Teile-und-herrsche-Algorithmen (engl. Divide and conquer) sind eine Problemlösungsstrategie, bei der ein Problem solange in Teilprobleme aufgeteilt wird, bis die einzelnen Teilprobleme einfach zu lösen sind. Anschließend wird aus den Teillösungen die Gesamtlösung zusammengesetzt. Die Umsetzung erfolgt meistens rekursiv (vgl. Dörn 2016: S. 260).

Klassischer Vertreter dieses Algorithmenmusters ist der Mergesort-Algorithmus. Hierbei wird eine Liste mit unsortierten Zahlen rekursiv solange halbiert, bis jede Zahl für sich steht. Anschließend werden in jeder Inkarnation beide Teile der Liste in sortierter Reihenfolge zusammengeführt. Der Mergesort-Algorithmus ist mit einer Laufzeit von O(N log N) ein relativ schneller Sortieralgorithmus (vgl. Weiss 2014: S. 363).

Systematisch: „Backtracking (dt. Rücksetzverfahren) geht nach dem ‚Versuch-und-Irrtum'-Prinzip vor. Es wird hierbei versucht, eine erreichte Teillösung zu einer Gesamtlösung auszubauen. Wenn absehbar ist, dass eine Teillösung nicht zu einer endgültigen Lösung führen kann, werden der letzte Schritt oder die letzten Schritte zurückgenommen und stattdessen ein alternativer Weg probiert. Auf diese Weise wird sichergestellt, dass alle infrage kommenden Lösungswege ausprobiert werden können." (Dörn 2016: S. 274) Die Umsetzung erfolgt in der Regel rekursiv. Da der

Rechenaufwand in vielen Fällen mit zunehmender Menge exponentiell steigt, wird oft nach Auffinden der ersten Lösung abgebrochen oder eine maximale Rekursionstiefe festgelegt (vgl. Dörn 2016: S. 275).

Zur Anwendung kommen Backtracking-Algorithmen häufig bei Strategiespielen. So auch beim N-Queen-Problem: Ziel ist es n Damen auf einem n x n Schachbrett so zu platzieren, dass keine Dame eine andere Dame schlagen kann. Der Algorithmus platziert zeilenweise eine Dame und berechnet, wo in der nächsten Zeile eine weitere Dame platziert werden könnte. Kann keine Dame in der nächsten Zeile platziert werden, wird die Position der zuvor platzierten Dame geändert. Der Algorithmus terminiert, wenn in der letzten Zeile eine Dame platziert werden konnte. Da auf diese Weise offensichtlich unsinnige Lösungen gar nicht erst ausprobiert werden, ist der Backtracking-Algorithmus deutlich effizienter als ein Brute-Force-Algorithmus (vgl. Dörn 2016: S. 275ff.).

Zwischenspeichernd: Dynamisches Programmieren orientiert sich an der Vorgehensweise der Teile-und-herrsche-Algorithmen. Große Probleme werden in kleinere Teilprobleme aufgeteilt. Entscheidender Unterschied ist aber, dass die Teilprobleme nicht immer gleicher Natur sind. Charakterisierend ist, dass Lösungen für Teilprobleme in einer Tabelle zwischengespeichert werden und nicht erneut berechnet werden müssen.

Werden die Fibonacci-Zahlen rekursiv berechnet, wird eine Flut von rekursiven Aufrufen getätigt. Die meisten Aufrufe lösen dann ein Problem, das etliche Male zuvor schon gelöst wurde. Würden die Fibonacci-Zahlen nach der ersten Berechnung in einer Liste gespeichert werden, könnte die exponentiell wachsende Anzahl an rekursiven Aufrufen vermieden werden (vgl. Dörn 2016: S. 286; Weiss 2012: S. 474).

3.2 Paradigmen

Ein Paradigma ist ein grundsätzliches Denkmuster. Laut dem Inhaber des Lehrstuhls für Software-Technik der Ruhe-Universität Bochum, Helmut Balzert, werde in der Software-Technik zwischen drei Paradigmen unterschieden: strukturiertes, objekt-orientiertes und wissensbasiertes Paradigma (vgl. Balzert 2001: S. 40).

Je nach Problemstellung, muss entschieden werden, welches Paradigma sich am besten eignet. In der Praxis kommen das strukturierte und das objektorientierte Paradigma besonders häufig zur Anwendung. Diese beiden Paradigmen werden im Folgenden genauer erläutert. Das wissensbasierte Paradigma spielt in der Software-Technik eine untergeordnete Rolle und wird deswegen an dieser Stelle nicht vertieft (vgl. Balzert 2001: S. 40).

3.2.1 Strukturierte Programmierung

Als Wegbereiter des strukturierten Programmierens gilt unter anderem Edsger Wybe Dijkstra, der 1968 in seinem Aufsatz „Go To Statement Considered Harmful" den übermäßigen Gebrauch des Go-To-Statements kritisierte. Zu dieser Zeit befand sich die Branche in der „Softwarekrise" und Dijkstra sah in der Sprunganweisung die Ursache für nichtmehr-nachvollziehbare Programme (vgl. Dijkstra 1968: 147f.; Balzert 2001: S. 266).

Die strukturierte Programmierung arbeitet nur mit linearen **Kontrollstrukturen**. Sie „dienen dazu, den Ablauf eines Algorithmus zu steuern [und] [...] geben an, ob bzw. wie oft Anweisungen ausgeführt werden sollen." (Balzert 2001: S. 260) Es wird in Sequenz, Auswahl, Wiederholung und Aufruf unterschieden (vgl. Heinisch et al. 2011: S. 8; Balzert 2001: S. 261ff.).

Sequenz: Abarbeitung mehrerer Anweisungen hintereinander.

Auswahl: Ausführen von Anweisungen in Abhängigkeit bestimmter Bedingungen.

Wiederholung: Mehrfaches Ausführen von Anweisungen in Abhängigkeit einer festen Wiederholungsanzahl oder -bedingung.

Aufruf: Anwendung eines anderen Algorithmus.

Kontrollstrukturen können ineinander geschachtelt werden. Außerdem haben alle vier linearen Kontrollstrukturen gemeinsam, dass sie jeweils genau einen Eingang und einen Ausgang besitzen. „Zwischen dem Eingang und dem Ausgang gilt das Lokalitätsprinzip, d.h., der Kontrollfluss verlässt den durch Eingang und Ausgang definierten Kontrollflussbereich nicht." (Balzert 2001: S. 265f.)

Struktogramme werden eingesetzt, um Kontrollstrukturen grafisch darzustellen. Die Darstellungen beruhen auf Vorschlägen der Informatiker Isaac Nassi und Ben Shneidermann, weswegen diese Notation auch Nassi-Shneidermann-Diagramme heißen. Sie ist in DIN 66261 genormt (vgl. Balzert 2001: S. 260).

Aktivitätsdiagramme der UML (Unified Modeling Language) sind eine moderne grafische Darstellungsform von Programmen. Die UML-Notation gilt als Industrie-Standard und löst früher verwendete Arten der Darstellung wie Flussdiagramme (auch Programmablaufpläne) ab (vgl. Balzert 2009: S. 228).

3.2.2 Objektorientierte Programmierung

Als erste rein objektorientierte Programmiersprache gilt Smalltalk-80. Sie wurde in den Jahren 1970 bis 1980 entwickelt und beruht auf Klassenkonzepten der Programmiersprache SIMULA 67 (vgl. Balzert 2001: S. 152).

Zentraler Bestandteil des objektorientierten Programmierens sind **Objekte**. Mit ihnen wird versucht, reale Gegenstände softwaretechnisch zu modellieren. Ein Objekt ist einfach gesagt ein Ding (Gegenstand, Person, Messwert, Bestellung, usw.). Es verfügt über drei wichtige Eigenschaften:

- Identität: Ein Objekt hat ein Merkmal, durch das es sich eindeutig von anderen Objekten unterscheidet.
- Zustand: Attribute beschreiben die Eigenschaften eines Objektes und stellen durch Attributwerte den Zustand dar.
- Verhalten: Methoden ermöglichen dem Objekt, seinen Zustand zu verändern oder wiederzugeben (vgl. Dörn 2016: S. 307f.; Balzert 2001: S. 156).

Klassen sind Schablonen für gleichartige Objekte. „In der objektorientierten Welt spezifiziert eine Klasse die Gemeinsamkeiten einer Menge von Objekten mit denselben Eigenschaften [...], demselben Verhalten und denselben Beziehungen. Eine Klasse besitzt einen Mechanismus, um Objekte zu erzeugen [...]. Jedes erzeugte Objekt gehört zu genau einer Klasse." (Balzert 2001: S. 161)

Sebastian Dörn (2016) beschreibt in seinem Buch fünf Grundprinzipien des objektorientierten Programmierens. Diese werden im Folgenden kurz erläutert.

Abstraktion: Beim Übertragen der Objekte aus der realen Welt in die Softwarewelt sollte darauf geachtet werden, sich auf das Wesentliche zu beschränken. Gemeinsamkeiten können zusammengefasst werden und unwichtige Einzelheiten sollten vernachlässigt werden.

Vererbung: Klassen können in einer hierarchischen Beziehung zueinander stehen. Die Vererbung spiegelt eine „IS A"-Beziehung wider. Dabei werden Attribute und Methoden einer generalisierten Eltern-Klasse in eine spezialisierte Kind-Klasse übernommen. Die neue Kind-Klasse kann zusätzlich neue Methoden und Attribute ergänzen und bereits bestehende Methoden überschreiben. Was die Kind-Klasse allerdings nicht kann, ist, vererbte Methoden und Attribute entfernen.

Assoziation: Ein Objekt kann in Verbindung mit anderen Objekten stehen. Eine Assoziation modelliert dabei die Beziehung zwischen Objekten gleichrangiger Klassen. Spezialfälle treten auf, wenn ein Objekt Teil eines anderen ist. Ist die Existenz beider Objekte unabhängig voneinander, wird von einer Aggregation gesprochen. Hängt die Existenz des Teilobjektes von der Existenz des zusammengesetzten Objektes ab, wird von einer Komposition gesprochen.

Kapselung: Das sogenannte Geheimhaltungsprinzip besagt, dass ein Objekt seine Daten eigenständig verwaltet. Attributwerte und Verbindungen sind außerhalb des Objektes nicht sichtbar. Der Zustand eines Objektes kann nur über speziell definierte Schnittstellen abgefragt oder manipuliert werden.

Polymorphie: Objekte aus Unterklassen haben die Möglichkeit, anders auf eine Anweisung zu reagieren, als Objekte der Oberklasse es tun würden. Innerhalb einer Klassenhierarchie können vererbte Methoden überschrieben werden. Besonderheiten einer Unterklasse werden so berücksichtigt und Schnittstellen vereinheitlicht.

(vgl. Dörn 2016: S. 307, Weiss 2010: S. 113, Balzert 2001: S. 156ff.)

Typischerweise wird der Aufbau eines objektorientierten Programms in einem **UML-Klassendiagramm** festgehalten. Neben den einzelnen Klassen, inklusive der Attribute und Methoden, werden außerdem die Beziehungen und Vererbungshierarchien vermerkt.

UML-Sequenzdiagramme dienen zur schematischen Visualisierung von zeitbasierten Vorgängen. In das Diagramm werden chronologisch die Methodenaufrufe von Objekten eingetragen, um deren Kommunikation untereinander zu verdeutlichen (vgl. Balzert 2001: S. 211).

3.3 Datenstrukturen

Datenstrukturen speichern und organisieren gleichartige Daten (vgl. Dörn 2016: S. 61). Algorithmen manipulieren diese Daten. Somit stehen Algorithmen und Datenstrukturen in ständiger Wechselwirkung zueinander. Für jeden Algorithmus muss eine passende Datenstruktur ausgewählt werden. Moderne Programmiersprachen stellen in Klassenbibliotheken ein umfangreiches Sortiment zur Verfügung. Die Nutzung, der von Spezialisten entwickelten Klassen, senkt den eigenen Programmieraufwand. Zusätzlich zeugen die bereits implementierten Klassen von einer hohen Qualität und Flexibilität. Allerdings besteht möglicherweise ein hoher Einarbeitungsaufwand (vgl. Balzert 2001: S. 840f.).

Im Folgenden werden die grundlegendsten Datenstrukturen kurz erläutert. Auf Graphen und Bäume wird in separaten Unterkapiteln eingegangen.

Listen verknüpfen Daten miteinander. Dabei verweist jedes Glied auf seinen Nachfolger (einfach verkettete Liste) oder gegebenenfalls sogar auf seinen Vorgänger (doppelt verkettete Liste) (vgl. Weiss 2010: S. 251ff.).

Stapel sind Listen, bei denen neue Elemente immer an das vordere Ende der Liste eingefügt werden. Getreu dem LIFO-Prinzip (last in first out) werden Elemente ausschließlich von dort wieder entnommen (vgl. Dörn 2016: S. 83).

Schlangen sind Listen, bei denen neue Elemente immer an das hintere Ende der Liste eingefügt werden und ausschließlich vom vorderen Ende wieder entnommen werden. Eine Warteschlange arbeitet somit nach dem FIFO-Prinzip (first in first out) (vgl. Von Rimscha 2008: S. 95).

Felder (auch Array) fassen Variablen des gleichen Datentyps unter einem Namen zusammen. Mittels eines Index kann auf den gespeicherten Wert zugegriffen werden. Werden Felder mehrdimensional angelegt, steigt die Anzahl Indizes mit jeder weiteren Dimension (vgl. Dörn 2016: S. 64ff.).

Graphen sind eine Menge von Knoten, die über Kanten verbunden sein können (siehe Unterkapitel 3.3.1).

Bäume sind eine hierarchische Datenstruktur, bei der Eltern-Knoten auf mehrere Kind-Knoten verweisen (siehe Unterkapitel 3.3.2).

Zuordnungen ermöglichen einen effizienten Zugriff auf gespeicherte Werte über einen Schlüssel (vgl. Von Rimscha 2008: S. 98).

3.3.1 Graphen

Aufbau: Graphen bestehen aus Knoten, die über Kanten miteinander verbunden sind. Mittels Graphen lassen sich zum Beispiel Verkehrsnetze, Computernetze und Schaltnetzte modellieren. Mit Hilfe dieser Modelle kann anschließend unter anderem der kürzeste Weg berechnet oder ein Versorgungsnetz geplant werden (vgl. Dörn 2017: S. 119f.).

Bei den Kanten eines Graphen wird in gerichtet und ungerichtet sowie in gewichtet und ungewichtet unterschieden. Gerichtete Kanten können nur in eine Richtung benutzt werden. Ungerichtete Kanten ermöglichen die Benutzung in beide Richtungen. Das Gewicht einer Kante wird auch Kosten genannt. Dabei kann es sich um finanzielle Abgaben handeln, genauso gut sind damit aber auch Distanzen oder Zeitkosten gemeint (vgl. Weiss 2012: S. 359).

Ein Pfad ist die Reihenfolge von Knoten, die vom Start bis zum Ziel besucht werden. Die Länge eines Pfades entspricht bei ungewichteten Kanten der Anzahl der Kanten eines Pfades und bei gewichteten Kanten der Summe der Kosten der Kanten eines Pfades. Besonderes Augenmerk sollte auf Knoten gerichtet werden, die eine Verbindung zu sich selbst haben. In diesen Fällen wird von einem „Loop“ gesprochen. Gleiches Fehlerpotential bieten Kreisläufe bei gerichteten Graphen. Werden diese Besonderheiten nicht beachtet, kann ein Algorithmus schnell falsche Ergebnisse liefern (vgl. Weiss 2012: S. 359).

Adjazenzmatrix: Um den Aufbau eines Graphen zu speichern, kann eine Adjazenzmatrix benutzt werden. Jede Zeile und Spalte repräsentiert dabei einen Knoten. Sind zwei Knoten miteinander verbunden, wird in dem entsprechenden Feld eine Eins eingetragen. Handelt es sich um eine gewichtete Kante, werden an dieser Stelle die Kosten der Kante vermerkt. Sind zwei Knoten nicht miteinander verbunden, können die Kosten auf Unendlich gesetzt werden (unter Umständen auch auf null) (vgl. Weiss 2012: S. 361). Des Weiteren gilt: Adjazenzmatrizen ungerichteter Graphen sind symmetrisch.

Adjazenzliste: In der Regel ist die Benutzung einer Adjazenzmatrix nicht die optimale Lösung, um einen Graphen zu speichern. Da sie einen Speicherbedarf hat, der mit zunehmender Menge an Knoten quadratisch steigt, sollten Adjazenzmatrizen nur bei Graphen mit wenig Knoten und vielen Kanten verwendet werden. Als Alternative kann eine Adjazenzliste benutzt werden. Diese speichert pro Zeile alle Nachbarn eines Knotens. Für gewichtete Graphen muss zusätzlich eine Kantenliste erstellt werden (vgl. Weiss 2012: S. 361).

3.3.2 Binäre Bäume

In Unterkapitel 3.3 wurde die verkettete Liste als eine Datenstruktur kurz vorgestellt. Grundprinzip war, dass jedes Glied der Kette auf seinen Nachfolger verweist. Da die Rechenzeit für einen Zugriff auf ein Element bei zunehmender Menge linear ansteigt, ist eine verkettete Liste für große Datenmengen nicht gut geeignet. Eine Abwandlung dieser Datenstruktur sind die Bäume. Hier verweist ein Knoten auf mehrere Nachfolger. Die Rechenzeit nimmt in diesem Fall lediglich logarithmisch zu. Deswegen wird diese Datenstruktur zum effizienten Abspeichern und Suchen eingesetzt (vgl. Dörn 2017: S. 114; Weiss 2012: S. 101).

Aufbau: In der Informatik besteht ein Baum aus einer endlichen Anzahl von Knoten, die durch gerichtete Kanten verbunden sind. Der Ursprung eines Baums wird Wurzel genannt. Dieser Knoten ist der einzige Knoten, der keinen Vorgänger besitzt. Ein Knoten, der keinen Nachfolger hat, heißt Blatt. Eine Kante stellt die Beziehung zwischen Eltern- und Kind-Knoten dar. Demzufolge werden Knoten, die dasselbe Elternteil haben, Geschwister genannt (vgl. Dörn 2017: S. 114). Das gleiche Prinzip kann für Großeltern und Groß-Großeltern eines Knotens angewendet werden. Ebenso kann ein Baum rekursiv definiert werden. Dann spricht man von einem Wurzel-Knoten, der mittels Kanten auf null oder mehrere Teilbäume verweist. Ein Teilbaum besteht aus einer Wurzel, die wiederum mit weiteren Teilbäumen verbunden ist (vgl. Weiss 2012: S. 101f.).

Wird der ganze Baum betrachtet, ergeben sich weitere Begriffe, die definiert werden müssen: Ein Pfad ist die Reihenfolge von Knoten, die vom Start bis zum Ziel besucht werden. Die Länge eines Pfades entspricht der Anzahl der Kanten eines Pfades (vgl. Weiss 2012: S. 102). „Die Ebene eines Knotens ist die Länge des Pfades von diesem Knoten bis zur Wurzel. [Die Wurzel befindet sich somit auf Ebene 0.] Die Tiefe eines Baumes ist die maximale Ebene, auf der sich Knoten befinden. Der Verzweigungsgrad eines Knotens ist die Anzahl seiner Kinder." (Dörn 2017: S. 115)

Anwendungsgebiete: Neben der Speicherung großer Datenmengen kommen Bäume an vielen Stellen der Informatik zur Anwendung. So zum Beispiel zur Organisation von Dateisystemen, Darstellung der Aufrufstruktur rekursiver Algorithmen und mathematischer Ausdrücke sowie zur hierarchischen Unterteilung von Objekten (vgl. Dörn 2017: S. 115).

Binäre Bäume: Ein Baum heißt binär, wenn alle Knoten nicht mehr als zwei Kinder haben (Weiss 2012: S. 107). Vor allem für Suchbäume ist diese Variante der Baumstruktur geeignet. Ein Knoten beinhaltet das zu speichernde Element und verweist zusätzlich auf einen linken und rechten Nachfolger.

Operationen: Soll ein Element dem binären Baum hinzugefügt werden, wird das neue Element mit dem im Knoten gespeicherten Element verglichen und entweder nach links oder rechts weitergeleitet. Das beschriebene Schema wird solange ausgeführt, bis es links bzw. rechts des Knotens keine weiteren Nachfolger gibt. Dann wird das neue Element an diese Stelle eingefügt (vgl. Weiss 2012: S. 116f.). Der Einfachheit halber wird im Folgenden davon ausgegangen, dass nur Ganzzahlen in dem Baum gespeichert werden. Demnach befinden sich aus Sicht eines Knotens im linken Teilbaum ausschließlich kleinere Werte und im rechten Teilbaum ausschließlich größere Werte (vgl. Weiss 2012: S. 112).

Soll ein Element in dem Baum gefunden werden, wird nach dem gleichen Prinzip vorgegangen. Der gesuchte Wert wird mit dem Wert des Knotens verglichen. Stellt sich bei dem Vergleich heraus, dass es sich nicht um den gesuchten Wert handelt, wird nach links bzw. rechts abgestiegen (vgl. Weiss 2012: S. 113).

Soll ein Knoten entfernt werden, muss betrachtet werden, wie viele Kinder der Knoten hat. Handelt es sich um ein Blatt-Knoten, kann dieser ohne weitere Schritte entfernt werden. Hat der Knoten ein Kind, muss vor dem Löschen die Referenz des Eltern-Knoten vom zu löschenden Knoten auf das eine Kind geändert werden. Komplizierter ist es, wenn der Knoten zwei Kinder hat. Dann muss der Knoten den gespeicherten Wert mit seinem direkten Vorgänger oder Nachfolger tauschen. Der direkte Vorgänger/Nachfolger ist per Definition ein Blatt-Knoten und kann dementsprechend mühelos entfernt werden (vgl. Weiss 2012: S. 118ff.).

AVL-Bäume: Wie am Anfang des Abschnittes erwähnt, steigt der Rechenaufwand mit zunehmender Menge im Durchschnitt logarithmisch an. Voraussetzung ist, dass der Baum sich im Gleichgewicht befindet. Im schlimmsten Fall ist mit einem linearen Wachstum zu rechnen, wenn es sich praktisch um eine verkettete Liste handelt. Ein Konzept, einen binären Baum nach Hinzufügen von Knoten zurück ins Gleichgewicht zu bringen, bietet ein AVL-Baum.

Ein AVL-Baum, entwickelt von Adelson-Velski und Landis, ist ein binärer Baum mit einer Gleichgewichtsbedingung. Diese besagt, dass sich die Höhe der beiden Teilbäume um nicht mehr als Zwei unterscheiden darf. Sollte diese Bedingung verletzt werden, muss rotiert werden (vgl. Weiss 2012: S. 123). Je nachdem in welchem Teilbaum das Ungleichgewicht verursacht wurde, muss einmal bzw. zweimal rotiert werden. Das Resultat ist ein balancierter binärer Suchbaum.

4 Umsetzung in einer objektorientierten Programmiersprache

Die ingenieurgemäße Nutzung, Anpassung und Weiterentwicklung von Informationstechnologien sind Bestandteil der täglichen Arbeit von Bauingenieuren. Die Umsetzung größerer Programmierprojekte übersteigt den limitierten Rahmen dieser Arbeit. Allerdings muss jeder Bauingenieur in der Lage sein, die Ergebnisse von Problemanalysen für IT-Dienstleister beschreiben und an Softwareentwicklungen als Fachspezialist substanziell mitwirken zu können. Programmierkenntnisse sind gefragt, wenn es um die Anpassung und Integration von Softwaresystemen in laufenden Projekten geht.

Die zurückliegenden Kapitel beschrieben die theoretischen Grundlagen sowie die Methodik und Didaktik, wie die Lehrinhalte vermittelt werden. In diesem Kapitel wird gezeigt, wie diese Konzepte mit Hilfe der objektorientierten Programmiersprache Java umgesetzt werden.

4.1 Ablauf und Organisation

Wochenstruktur: Jede Woche wird beginnend mit der Vorlesung eine neue Kaskade an Lehrveranstaltungen angestoßen. Der *methodische Gang* der Lehrveranstaltungen lässt sich als Lenkungslinie beschreiben. Dabei wird mit einer hohen Lehrerdominanz in ein Themengebiet eingeführt und mit einer entsprechend hohen Schülerdominanz abgeschlossen. Innerhalb einer Woche finden eine Vorlesung, ein Seminar und pro Übungsgruppe ein Tutorium statt. In der Vorlesung erhalten die Studierenden einen Überblick über das Stoffgebiet. Im Seminar werden die wichtigsten Inhalte wiederholt und gezeigt, wie die Thematik in Java umgesetzt werden kann. Am Ende des Seminars findet eine Einführung in das nächste Tutorium statt. Je nachdem in welche Übungsgruppe ein Studierender sich eingeschrieben hat, erscheint dieser zum angegebenen Zeitraum. Während des Tutoriums sollten die Studierenden größtenteils eigenständig programmieren. Als Hilfestellung kann sich mit Kommilitonen ausgetauscht werden oder bei Fragen können die Tutoren hinzugezogen werden. Zusätzlich stehen für jedes Tutorium Erklärvideos online zur Verfügung. Sollten 90 Minuten nicht zum Lösen der Aufgaben ausgereicht haben, wird den Studierenden empfohlen, die Aufgaben im Selbststudium zu beenden. Über die Lehrveranstaltungen hinaus können die Studierenden online Multiple-Choice-Tests bearbeiten. So erhalten sie die Möglichkeit, ihr Wissen zu festigen und ein Feedback zu ihrem aktuellen Kenntnisstand zu bekommen.

Semesterstruktur: Im Laufe der Lehrveranstaltungen finden sich drei verschiedene *methodische Linienführungen* wieder: Vertrautheitslinie, Abstraktionslinie sowie Komplexitätslinie. Inhaltlich wird vom Vertrauten ins Fremde, vom Abstrakten zum Konkreten und vom Einfachen zum Komplizierten gegangen.

Thematisch wird bei der strukturierten Programmierung begonnen. Nachdem erste Algorithmen in Kombination mit einfachen Datenstrukturen behandelt wurden, wird mit Hilfe der objektorientierten Programmierung eine komplexe Datenstruktur erstellt. Am Beispielprojekt „little BIM“ soll die Möglichkeit geschaffen werden, ein stark vereinfachtes digitales Gebäudemodell zu speichern. Da zum Building Information Modelling (BIM) nicht nur das digitale Gebäudemodell gehört, wird exemplarisch an der Berechnung von Fluchtwegen gezeigt, welche weiteren Möglichkeiten BIM bietet. Es werden die Grundlagen der Graphentheorie erklärt und anschließend der Dijkstra-Algorithmus angewendet. Dieser Algorithmus aus der Klasse der Greedy-Algorithmen bestimmt in einem Graph den kürzesten Weg. Die Ausgabe des Weges erfolgt rekursiv, womit zum nächsten großen Themengebiet übergeleitet wird: Rekursion. Als Vertreter einer rekursiven Datenstruktur werden binäre Bäume behandelt. Backtracking wird abschließend als ein rekursives Algorithmenmuster behandelt.

Eine Übersicht der Gliederung des Lehrkonzeptes ist in Anlage 1 zu finden.

Organisatorisches: Das Lehrkonzept wurde so entwickelt, dass das Lehrpersonal von Semester zu Semester kaum Anpassungen vornehmen muss. Die Aufgabenverteilung sieht wie folgt aus: Der Hochschullehrer hält die Vorlesung. Ein Übungsleiter ist für die praktische Anwendung des Wissens zuständig. Dazu zählt das Halten der Seminare und Organisieren der Tutorien. Die Tutorien werden von Tutoren betreut. Pro Übungsgruppe sollten zwei Tutoren anwesend sein. Im Vorfeld der Tutorien werden die Tutoren vom Übungsleiter eingewiesen und erhalten die Lösung der Aufgaben. Des Weiteren wird das zurückliegende Tutorium besprochen und mögliche Probleme der Studierenden beim Bearbeiten der Aufgaben festgehalten. Die Ausgabe und Bewertung der Übungsaufgabe liegt in der Verantwortung des Übungsleiters. Einzelne Aufgaben können unter Aufsicht und Anleitung an Tutoren übertragen werden. Die Übungsaufgabe wurde so gestaltet, dass sich der Korrekturaufwand und die Bürokratie im Rahmen hält. Die Klausur wird in enger Zusammenarbeit zwischen dem Hochschullehrer und dem Übungsleiter erstellt.

Über das Semester pflegt der Übungsleiter die Inhalte des OPAL-Kurses. Über den E-Mail-Verteiler können alle Studierenden kontaktiert werden. Zusätzlich sollten alle relevanten E-Mails über das Mitteilungs-Tool auf OPAL abrufbar sein.

4.2 Einführung in die strukturierte Programmierung

Lernziele: Das erste Themengebiet befasst sich mit dem strukturierten Programmieren. Neben der Vorlesung vermitteln zwei Seminare und Tutorien den Studierenden folgende Kompetenzen: Grundgedanken des strukturierten Programmierens, Erstellen von Variablen sowie der Umgang mit primitiven Datentypen, Operatoren, Kontrollstrukturen, Methoden und Arrays.

Einführung: Einführend werden die in Unterkapitel 3.2.1 behandelten theoretischen Grundlagen wiederholt. Anschließend folgen die ersten Schritte in der Entwicklungsumgebung IntelliJ. Neben dem Erstellen eines neuen Projektes wird auch gezeigt, wie eine Klasse erstellt und die main-Methode eingefügt wird. Die main-Methode muss zwingend in einem Programm vorhanden sein, da sie beim Start ausgeführt wird. Optional können über ein String-Array Argumente als Parameter mitgegeben werden. Diese Funktion wird allerdings in nur sehr wenigen Fällen benötigt.

Variablen und Datentypen: Das Arbeiten mit Variablen ist eine Kernkompetenz, die jeder Studierende beherrschen sollte. Um eine Variable benutzen zu können, muss sie am Anfang deklariert und vor der ersten Benutzung initialisiert werden. Bei der Deklaration müssen der Datentyp sowie der Name der Variable festgelegt werden. Die Initialisierung weist der Variable ein Startwert zu. Grundsätzlich unterscheidet Java in primitive und komplexe Datentypen. Primitive Datentypen speichern den Wert der Variable an sich. Wird der gespeicherte Wert einer anderen Variable zugewiesen, wird eine Kopie des Wertes gespeichert. Dieses Verhalten wird Wertsemantik genannt. Komplexe Datentypen speichern lediglich die Speicheradresse als Referenz auf ein Objekt. Die Referenzsemantik beschreibt, dass beim Zuweisen eines Objektes zu einer anderen Variable nur die Referenz auf das Objekt und nicht das Objekt an sich kopiert wird. Komplexe Datentypen spielen bei der objektorientierten Programmierung eine zentrale Rolle (Weiss 2010: S. 27; Wachtler.de).

Java stellt acht primitive Datentypen zur Verfügung: boolean speichert Wahrheitswerte; char speichert Schriftzeichen; byte, short, int, long speichern Ganze Zahlen; float und double speichern Fließkommazahlen. Je nach Datentyp nimmt die Variable zwischen einem und acht Byte Speicher des Rechners ein. Der in Java schon vordefinierte komplexe Datentyp String, wird zum Speichern von Zeichenketten genutzt (vgl. Dörn 2016: S. 18).

Operatoren: Nach den Variablen werden die Operatoren vorgestellt. Operatoren werden genutzt, um Ausdrücke zu erzeugen. Ein Ausdruck ist eine Kombination von

Variablen, Konstanten oder Funktionen mittels Operatoren. Mehrere Ausdrücke können zu einem komplexen Ausdruck verknüpft werden, wobei immer auf die Lesbarkeit und Nachvollziehbarkeit des Quellcodes geachtet werden sollte. Die Operatoren können in vier Kategorien eingeteilt werden: Zuweisungsoperatoren, arithmetische Operatoren, relationale Operatoren und logische Operatoren (vgl. Dörn 2016: S. 23ff.). Eine Auflistung der wichtigsten Operatoren ist in Anlage 6 zu finden.

Kontrollstrukturen: Um Einfluss auf den linearen Programmfluss zu nehmen, gibt es Kontrollstrukturen. Es wird zwischen Auswahlanweisungen und Schleifen unterschieden. Beide Arten prüfen eine Bedingung. Auswahlanweisungen verzweigen den Programmfluss. Das if-Statement wählt zwischen zwei Anweisungsblöcken aus (Einfachfallunterscheidung), während das switch-Statement mehrere Optionen zur Auswahl hat (Mehrfachfallunterscheidung). Schleifen gliedern sich in Zählschleifen (for-Schleife) und Bedingungsschleifen, wobei letztere noch einmal in abweisende Bedingungsschleifen (while-Schleife) und nicht abweisende Bedingungsschleifen (do-while-Schleife) unterteilt werden. Schleifen wiederholen einen Anweisungsblock solange, bis die Bedingung nicht mehr erfüllt ist. Um eine Kollektion vollständig zu durchlaufen ist in Java eine for-each-Schleife implementiert (vgl. Kölling 2010: S. 225ff.; Dörn 2016: S. 35ff.). Zusätzlich zur Schreibweise in Java werden für die Kontrollstrukturen die jeweiligen Abbildungen als Nassi-Shneiderman-Diagramm (Struktogramm) und UML-Aktivitätsdiagramm vorgestellt.

Erstes Programmierbeispiel: In einem ersten Beispiel wird gezeigt, wie alle ganzen Zahlen zwischen 1 und 500 mittels einer for-Schleife aufaddiert werden können. Zur Überprüfung des Resultats kann die Gaußsche Summenformel benutzt werden. Als zweite Aufgabe soll auf der Konsole ausgegeben werden, wann die Summe einen Wert über 100.000 erreicht. Hier kommt das Schlüsselwort „break“ zur Anwendung. Es beendet Schleifen sofort.

Zweites Programmierbeispiel: Am Ende des ersten Seminars wird ein Blick auf die numerische Umsetzung der Iteration geworfen. Als Beispiel wird die iterative Berechnung der Quadratwurzel x einer Zahl z mittels des Newton-Verfahrens gezeigt. Wird ausgehend von einem Startwert $x_{start} = z$ die Formel $x_{neu} = \frac{x_{alt} + z/x_{alt}}{2}$ benutzt, nähert sich das Ergebnis mit jedem Iterationsschritt der exakten Lösung an. Ein Abbruchkriterium bestimmt die Genauigkeit der Lösung. Ist beispielsweise die Differenz zwischen dem alten und dem neuen Wert kleiner als 10^{-8}, soll die Iteration abgebrochen werden (vgl. Von Rimscha 2018: S. 5). Im Laufe des Studiums ist die iterative Berechnung einer Unbekannten keine Seltenheit. Im Wasserbau wird beispielsweise bei der Bemessung des Tosbeckens eines Wehrs die Fließtiefe und

im Spannbetonbau die Lage des Blockierungspunktes iterativ berechnet. Umgesetzt in Java kommt eine while-Schleife zum Einsatz. Diese wird solange durchlaufen, bis das Abbruchkriterium erfüllt ist. Eine Auslagerung dieses Algorithmus in eine eigene Methode dient als Überleitung zum nächsten Seminar.

Methoden: Eine Methode fasst eine Anweisungsfolge unter einem Namen zusammen. Sie kann beim Aufruf Parameter empfangen und am Ende ein Ergebnis zurückgeben (vgl. Dörn 2016: S. 53). Der Aufbau einer Methode gliedert sich in den Methodenkopf und Methodenrumpf. Im Methodenkopf werden Angaben zur Sichtbarkeit, dem Datentyp des Rückgabewerts, dem Namen sowie den erwarteten Parametern gemacht. Im Methodenrumpf befindet sich der Quellcode, der ausgeführt wird, wenn die Methode aufgerufen wird (vgl. Dörn 2016: S. 57).

Da zu diesem Zeitpunkt Methoden im nicht-objektorientierten Kontext behandelt werden, sind lediglich der Rückgabetyp, Name und die Parameterliste von Bedeutung (vgl. Weiss 2010: S. 18). Darüber hinaus tragen in der strukturierten Programmierung alle Methoden das Schlüsselwort „static“. Der Rückgabetyp gibt an, von welchem Datentyp der Rückgabewert ist. Steht an dieser Stelle ein „void“ bedeutet dies, dass die Methode keinen Wert als Ergebnis liefert. Der Name einer Methode kann frei gewählt werden. Als Konvention hat sich ein Kleinbuchstabe am Anfang des Methodennamens etabliert (vgl. Dörn 2016: S. 57). Da Methoden Tätigkeiten darstellen, werden häufig Verben als Methodenname verwendet. Die Parameterliste beinhaltet null oder mehr Parameter. Dabei handelt es sich um Variablen, die beim Aufruf der Methode übergeben werden müssen (vgl. Dörn 2016: S. 57).

Überladen von Methoden: Mehrere Methoden können den gleichen Namen haben. Einzige Voraussetzung ist, dass die Parameterliste unterschiedlich ist. Je nachdem wie viele Parameter beim Aufruf der Methode übergeben werden, wählt der Compiler die richtige Variante der Methode aus. Dieses Vorgehen heißt Überladen einer Methode (vgl. Weiss 2010: S. 19).

Turtle-Grafik: Um den Studierenden eine bessere Vorstellung zu vermitteln, wie der Computer Anweisungen abarbeitet und Kontrollstrukturen Einfluss auf den Programmablauf nehmen, folgen im Anschluss einige Beispiele mit Verwendung der Turtle-Grafik. Mittels der Laufe- und Drehen-Anweisung kann die Turtle gesteuert werden. Bewegt sie sich, zieht sie einen Strich hinter sich her. Läuft beispielsweise die Turtle eine gewisse Strecke, dreht sich anschließend um 120 Grad und wiederholt den Ablauf noch zweimal, wurde ein regelmäßiges Dreieck gezeichnet. Unter Verwendung von Schleifen und Auswahlanweisungen entstehen regelrecht Kunstwerke. Einzelne Tätigkeiten, zum Beispiel das Zeichnen eines Dreiecks, können

dabei in eine eigene Methode ausgelagert werden. Mit Hilfe eines Parameters kann die Größe des Dreiecks bestimmt werden.

Zufallszahlen: Zufallszahlen spielen in der Informatik eine wichtige Rolle. Sie werden beispielsweise in der Kryptografie oder zur Simulation von Systemen benötigt (vgl. Weiss 2010: S. 393). Java bietet mehrere Möglichkeiten, Zufallszahlen zu erzeugen. Zum einen kann die statische Methode „random()“ der Klasse „Math“ aufgerufen werden. Diese liefert einen Wert zwischen 0,0 und 1,0. Um größere Zahlen zu erhalten, muss der Rückgabewert mit einem Faktor multipliziert werden. Zum anderen kann ein Zufallszahlengenerator-Objekt erzeugt werden. Dieser verfügt über zusätzliche Funktionen. So kann über ein Seed ein Zufallsexperiment (z. B. eine Windsimulation) mit den gleichen Zufallszahlen wiederholt werden (vgl. Weitz 2020).

Arrays: Um mehrere Werte unter einem Variablennamen zu speichern werden Arrays benutzt. Genau genommen werden Arrays wie auch schon der komplexe Datentyp String in Java als Objekte implementiert. Da nicht-objektorientierte Programmiersprachen allerdings ähnliche Datentypen verwenden, wurden diese zwei Ausnahmen trotzdem dem strukturierten Programmieren zugeordnet. Beim Erzeugen eines Arrays muss der Datentyp, Name und die Anzahl Felder angegeben werden. Anschließend kann mittels eines Index auf das entsprechende Feld zugegriffen werden. Als Beispiel wird gezeigt, wie einzelne Felder des Arrays mit einer Zufallszahl belegt werden können.

4.3 Grundlegende Datenstrukturen der Java API

Was ist das Java API? Um nicht immer wieder häufig genutzte Datenstrukturen programmieren zu müssen, bietet Java eine umfangreiche Klassenbibliothek mit einer großen Auswahl an Datenstrukturen und einigen Algorithmen. API steht für Application Programming Interface und bedeutet so viel wie Programmier-schnittstelle. Über diese Schnittstelle kann der vorhandene Quellcode genutzt werden. Dabei findet eine klare Abgrenzung zwischen dem eigens geschriebenen Quellcode und dem Java API Quellcode statt. Die Kommunikation zwischen den Programmteilen findet ausschließlich über die vorgesehenen Methoden statt (vgl. Luber/Augsten 2017).

Lernziele: Dieser Abschnitt soll den Studierenden eine Übersicht, der in der Java API implementierten Datenstrukturen, geben und deren Verwendung zeigen.

Einführung: Die meisten Datenstrukturen ähneln sich in ihrem Aufbau. Jede von ihnen speichert eine Sammlung an Elementen und stellt Methoden zum Hinzufügen,

Entfernen und Manipulieren zur Verfügung. Deswegen basiert die Implementierung zu einem großen Teil auf dem Konzept der Vererbung, welches an dieser Stelle allerdings nicht vertieft wird. Stattdessen werden die in Unterkapitel 3.3 vorgestellten grundlegenden Datenstrukturen mit ihrer Repräsentation in Java vorgestellt:

ArrayList: Der größte Nachteil des klassischen Arrays ist, dass die Größe bereits bei der Deklaration festgelegt werden muss. Sollte einmal nicht klar sein, wie viele Felder benötigt werden, ist es schwierig, die richtige Menge Speicherplatz zu reservieren. Eine Lösung bietet die in der Java API enthaltene ArrayList. Die *add*-Methode vergrößert das Array und fügt den gewünschten Wert in das neue Feld ein. Um Zugriff auf einen Wert zu bekommen, wird die *get*-Methode unter Angabe des Index benutzt. Mit der *set*-Methode kann der Wert eines bestimmten Feldes bearbeitet werden. Des Weiteren kann die Länge des Arrays mithilfe der *size*-Methode abgefragt werden. Neben diesen Standartfunktionen kann beispielsweise abgefragt werden, an welcher Stelle sich ein Element befindet oder ob die Liste leer ist. Entfernt werden Felder mit der *remove*-Methode (vgl. Weiss 2010: S. 40ff.).

LinkedList: Ein wenig anders funktioniert die LinkedList. Hier werden die Elemente in Knoten gespeichert, welche jeweils auf ihren Vorgänger und Nachfolger verweisen. Eine LinkedList ist einer ArrayList vorzuziehen, wenn viele Elemente der Liste hinzugefügt oder aus der Liste entfernt werden. Der Aufwand, um auf ein einzelnes Element zuzugreifen, ist bei einer ArrayList geringer (vgl. Weiss 2010: S. 251f.).

Stack und Queue: Stacks und Queues sind Datenstrukturen mit eingeschränktem Zugriff auf die gespeicherten Elemente. Stacks erlauben den Zugriff auf das zuletzt hinzugefügte Element. Mit der push-Methode wird ein Element in den Stapel gedrückt. Die *peek*-Methode erlaubt einen Blick auf das oberste Element des Stapels und die *pop*-Methode holt dieses Element aus dem Stapel heraus. Ähnlich funktioniert eine Queue. Allerdings werden die Elemente mit der *add*-Methode am Ende hinzugefügt und mit der *poll*-Methode vom Anfang der Schlange entnommen. Die *peek*-Methode erlaubt wieder einen Blick auf das nächste zu entnehmende Element (vgl. Weiss 2010: S. 258ff.). Eine PriorityQueue vergleicht Elemente untereinander und verschiebt das Element mit der höchsten Priorität an den Anfang der Schlange (vgl. Weiss 2010: S. 275).

HashMap: Um mehrere Elemente unter einem Namen speichern zu können, ohne dabei Einschränkungen bei den Zugriffsmöglichkeiten in Kauf nehmen zu müssen, stehen bis langen nur die ArrayList und LinkedList zur Verfügung. Allerdings muss bei diesen Datenstrukturen bekannt sein, an welcher Position sich das gesuchte

Element befindet, um Zugriff darauf zu bekommen. HashMaps erlauben den Zugriff auf gespeicherte Elemente über Schlüssel. Beim Hinzufügen eines Elements über die *put*-Methode wird anhand des Schlüssels ein Hash-Code generiert. Dieser dient als Grundlage zu entscheiden, an welchem Platz das Element gespeichert werden soll. Wird die *get*-Methode aufgerufen, muss der Schlüssel angegeben werden. Anschließend wird mit dem übergebenen Schlüssel der Speicherplatz berechnet, an dem sich das gesuchte Element befindet (vgl. Ramachandran 2015).

4.4 Sortieralgorithmen

Einführung: Sortieralgorithmen stecken in nahezu jedem Softwaresystem. Egal, ob Dateien in einem Ordner oder Artikel in einem Onlineshop, um einfacher auf ein Element zugreifen zu können, müssen diese sortiert werden (vgl. Weiss 2010: S. 352).

Lernziele: Die Studierenden lernen im Folgenden mehrere Sortieralgorithmen kennen. Des Weiteren wird darauf eingegangen, anhand welcher Kriterien Sortieralgorithmen verglichen werden.

Bubblesort: Ein leicht nachvollziehbarer, jedoch sehr ineffizienter Sortieralgorithmus ist der Bubblesort-Algorithmus. So wie große Luftblasen schneller an die Wasseroberfläche steigen als kleinere, werden große Zahlen nach und nach ans Ende einer unsortierten Liste geschwemmt. Eine Schleife durchläuft die gesamte Liste und vertauscht zwei benachbarte Elemente, wenn diese nicht in der richtigen Reihenfolge stehen. Somit befindet sich nach dem ersten Durchlauf das größte Element am Ende der Liste. Nach Abschluss des zweiten Durchlaufs ist das zweitgrößte Element auf dem vorletzten Platz der Liste zu finden. Dieser Vorgang wird so lange wiederholt, bis die gesamte Liste sortiert ist (vgl. Dörn 2016: S. 236). In Java ist dieser Algorithmus mittels zwei Schleifen und einer Auswahlanweisung leicht implementiert. Die äußere Schleife ist dafür verantwortlich, dass die Liste (n-1)-mal durchlaufen wird. Die innere Schleife ermöglicht die Iteration durch die Liste. Mit zunehmender Anzahl von Durchläufen muss auf weniger Felder zugegriffen werden, da die letzten Felder bereits als sortiert gelten. Die Auswahlanweisung vergleicht zwei benachbarte Felder und entscheidet, ob die Elemente sich bereits in der richtigen Reihenfolge befinden oder ihren Platz tauschen müssen.

Alphabetisches Sortieren: Um Wörter alphabetisch zu sortieren, muss die *compareTo*-Methode benutzt werden. Befindet sich ein Wort alphabetisch vor dem anderen Wort, gibt die Methode einen negativen Wert zurück. Befindet es sich hinter dem anderen Wort, ist der Rückgabewert positiv.

Bewertung von Sortieralgorithmen: Sortieralgorithmen werden nach ihrem Laufzeitverhalten bewertet. Eine weitere wichtige Information ist, ob ein Sortieralgorithmus stabil ist. Das ist der Fall, wenn Datensätze mit gleichem Sortierschlüssel in der Reihenfolge bleiben, wie sie eingegeben wurden (vgl. Weiss 2010: S. 385). Je nach Einsatzgebiet ist ein Sortieralgorithmus mehr oder weniger gut für die Anwendung geeignet. Weitere iterative Sortieralgorithmen sind beispielsweise der Selectionsort- oder der Insertionsort-Algorithmus. Alle weisen jedoch ein quadratisches Laufzeitverhalten auf. Besser schneiden rekursive Sortieralgorithmen ab. Der Mergesort-Algorithmus wurde bereits in Unterkapitel 3.1.2 beschrieben.

4.5 Objektorientiertes Programmieren am Beispiel „little BIM“

Lernziele: Die theoretischen Grundlagen des objektorientierten Programmierens wurden im Unterkapitel 3.2.2 bereits beschrieben. Dieser Abschnitt behandelt die praktische Umsetzung in Java. Die Studierenden lernen den Umgang mit folgenden Konzepten: Objekt, Datenkapselung, Vererbung und Überschreiben von Methoden. Zusätzlich werden grafische Benutzeroberflächen sowie das Einlesen und Ausgeben von Dateien behandelt.

Objekte: Zentraler Bestandteil des Konzeptes sind die Objekte. Um ein Objekt zu erzeugen, muss der Konstruktor aufgerufen werden. Dieser instanziiert das Objekt und liefert eine Referenz auf das Objekt. In ihm werden die Attribute des Objektes mit Anfangswerten belegt. Konstruktoren müssen immer genau so heißen, wie der Klassenname. Ist kein Konstruktor definiert, wird automatisch ein Standardkonstruktor erstellt. Konstruktoren können überladen werden. Das ermöglicht es, mehrere Konstruktoren zu verketten. So kann der parameterlose Standardkonstruktor über das Schlüsselwort „this“ einen weiteren Konstruktor aufrufen und Standardwerte als Parameter übergeben. Unnötige Duplikate werden durch geschicktes Wiederverwenden des Quellcodes vermieden (vgl. Dörn 2016: S. 313ff.).

Datenkapselung: Da die Informationen eines Objektes in ihm gekapselt werden, wird der Zugriffsmodifizierer der Attribute auf „private“ gesetzt. Dadurch kann nur das Objekt auf diese zugreifen. Über spezielle Methoden, deren Sichtbarkeit auf „public“ gestellt ist, wird eine kontrollierte Schnittstelle zu den Attributen geschaffen. Getter-Methoden geben den Wert eines Attributes zurück. Setter-Methoden können einem Attribut einen neuen Wert zuweisen. Dabei kann überprüft werden, ob die Änderung des Attributwertes zulässig ist (vgl. Dörn 2016: S.317ff.).

Vererbung: „Mithilfe der Vererbung können neue Klassen auf der Basis bereits vorhandener Klassen definiert werden. Die neuen, abgeleiteten Klassen erben

sämtliche Elemente (mit Ausnahme der Konstruktoren) der nach dem Schlüsselwort extends angegebenen Basisklasse. […] Die abgeleitete Klasse kann geerbte Methoden überschreiben, indem sie eine eigene Methode definiert, die den gleichen Namen wie eine geerbte Methode trägt.“ (Dörn 2016: S. 325) So wie das Schlüsselwort „this“ eine Referenz auf das Objekt selbst ist, ist das Schlüsselwort „super“ eine Referenz auf die Elternklasse. Mit Hilfe von „super“ lässt sich beispielsweise der Konstruktor der Elternklasse aufrufen oder eine Methode partiell überschreiben (vgl. Dörn 2016: S. 327f.).

Beispielprojekt „little BIM“: Diese beschriebenen Konzepte werden im Projekt „little BIM“ angewendet. Ziel ist es, ein Programm zu entwickeln, welches Bauteildaten Einlesen, Verarbeiten und Ausgeben kann. Inspiriert am Building Information Modeling (BIM) und auf Grundlage des offenen Standards IFC, der zur Beschreibung von digitalen Gebäudemodellen genutzt wird, sollen Bauteile (Wände, Fenster und Türen) in verschiedenen Objekten gespeichert werden. Abschließend soll das Programm mittels einer grafischen Benutzeroberfläche bedient werden.

Erstellen der Klassenstruktur: Anfangs wird eine Klassenstruktur aufgebaut. Alle Klassen erben das Attribut „globalID“ von der Klasse Root. Anschließend teilt sich der Vererbungsbaum in BuildingElement und BuildingStorey. Bei der Klasse BuildingElement handelt es sich immer noch um eine abstrakte Klasse, von der keine Objekte erzeugt werden können. Sie erfüllt den Zweck, allen weiteren Klassen Attribute für Geometriedaten zu vererben. Um zu konkretisieren, worum es sich bei „einbaubaren Elementen“ handelt, gibt es die Klassen Wall, Window und Door.

Überschreiben von Methoden: Nachdem die Grundstruktur geschaffen ist, erhalten die drei Klassen eine Methode, um auf der Zeichenebene dargestellt werden zu können. Ein erster Zwischenstand kann beobachtet werden, nachdem die ersten Wände erstellt und visualisiert wurden. In der Klasse BuildingElement ist dafür eine Methode definiert, die aus den Geometriedaten ein Rechteck erzeugt. Um Wände, Fenster und Türen voneinander unterscheiden zu können, wird das Überschreiben von Methoden genutzt. Die Farbe des Rechtecks wird von der getElementColor-Methode bestimmt. Die Klassen Wall, Window und Door überschreiben diese Methode. Beim Visualisieren des Bauteils gibt somit ein Objekt der Klasse Wall die Farbe Grau, ein Objekt der Klasse Window die Farbe Blau und ein Objekt der Klasse Door die Farbe Rot als Rückgabewert zurück. Partielles Überschreiben findet in der Klasse Wall statt. Zur Visualisierung einer Wand gehören auch die Fenster und Türen dazu. Deswegen wird über das Schlüsselwort „super“ die Methode zum Visualisieren der Wand aufgerufen und anschließend für alle in der Wand gespeicherten Fenster und Türen deren Visualisierungsmethode aufgerufen.

Weitere Funktionen: So wie eine Wand speichert, welche Fenster und Türen in ihr eingebaut werden, speichert ein Stockwerk, welche Wände zu ihm gehören. Mittels eigener Methoden können Bauteile einem anderen hinzugefügt werden oder abgefragt werden, welche Bauteile in einem anderen gespeichert sind.

Programmsteuerung: Bis zu diesem Zeitpunkt wird das Programm durch Änderungen im Quelltext gesteuert. Das heißt, soll eine neue Wand erstellt werden, muss der Konstruktor aufgerufen werden und die entsprechenden Parameter übergeben werden. Benutzer ohne Java-Kenntnisse könnten solch ein Programm nicht bedienen. Eine weitere Möglichkeit der Programmsteuerung ist es, Bauteildaten über die Konsole einzugeben. Diese Variante ist allerdings sehr aufwändig, unübersichtlich und wenig flexibel. Nahezu jede kommerzielle Software benutzt eine grafische Benutzeroberfläche (Graphical User Interface, GUI) zur Steuerung des Programms. Deswegen sollen die Studierenden einen Einblick bekommen, wie eine GUI erstellt wird und wie ein Knopfdruck bestimmte Aktionen auslösen kann.

GUI: Eine Möglichkeit, eine GUI in Java zu erstellen, ist die Nutzung des Swing-Toolkit. Unzählige Bausteine können zu einer grafischen Benutzeroberfläche zusammengesetzt werden. Das Jframe stellt das Fenster dar, in dem das Programm läuft. Inhalte werden auf Flächen namens Jpanel angeordnet. Layout Manager ordnen und skalieren die Bestandteile automatisch, sollte die Größe des Fensters verändert werden. Weitere Bestandteile sind beispielsweise Jlabel (Beschriftungsfelder), Jbutton (Knöpfe) und JtextField/JtextArea (Textfelder) (vgl. Weiss 2010: S. 931). Benutzt ein User die Maus oder Tastatur wird ein Event erzeugt. Ein Listener überprüft, ob beispielsweise ein Knopf gedrückt wurde. Ist das der Fall, wird die actionPerformed-Methode ausgeführt. An dieser Stelle hat der Programmierer hinterlegt, was passieren soll, wenn dieser bestimmte Knopf gedrückt wird (vgl. Weiss 2010: S. 947).

Einlesen und Ausgeben von Dateien: Eine weitere Kernkompetenz ist es, Dateien einzulesen und auszugeben. Wie genau das Scanner-Objekt in Kombination mit dem FileChooser und FileReader eine Datei einliest, ist dabei nicht von Bedeutung. Wichtiger ist es, dass die Studierenden den gegebenen Quellcode einarbeiten und mit der zurückgegebenen Datenstruktur sicher umgehen können. Die Eingabedateien orientieren sich an dem Aufbau einer csv-Datei. Tabellenfelder werden durch ein Semikolon getrennt und Zeilen durch einen Absatz. In Java werden solche Dateien in eine ArrayList umgewandelt, das String-Arrays speichert. Anhand der eingelesenen Daten sollen dann Objekte erzeugt werden. Ähnlich verhält es sich mit dem Erstellen einer Textdatei. Primär geht es darum, gegebenen Quellcode an den richtigen Stellen anzupassen und zu implementieren.

4.6 Anwendung eines Greedy-Algorithmus zur Fluchtwegberechnung

Einführung: Nachdem die Studierenden mit dem Projekt „little BIM“ einen Einblick bekommen haben, wie die Datenstruktur eines digitalen Gebäudemodells sehr stark vereinfacht aufgebaut ist, werden im Anschluss einige Einsatzmöglichkeiten von BIM während und nach dem Bauprozess vorgestellt.

Lernziele: Die Studierenden erhalten eine Einführung in die Graphentheorie und benutzen anschließend einen Greedy-Algorithmus, um den kürzesten Weg in einem Graphen zu berechnen.

Einsatzmöglichkeiten von BIM: In der Tragwerksplanung kann das BIM-Modell zum Erzeugen eines Analysemodells genutzt werden. Dabei werden tragende Bauteile aus dem Gesamtmodell herausgefiltert, da nur diese für die statische Berechnung bzw. numerische Simulation benötigt werden (vgl. Van Treeck et al.: S. 50). Des Weiteren ist das geometrische Modell prädestiniert, um eine Mengenermittlung für die Kostenschätzung durchzuführen. Da jedes Bauteil in einer Datenbank hinterlegt ist, können Raumbuch, Kostengruppen und Leistungsverzeichnis automatisiert erstellt werden (vgl. Van Treeck et al.: S. 53). Weiterhin kann das Modell zur Termin- und Ablaufplanung, Planung der technischen Gebäudeausrüstung und Brandschutzplanung genutzt werden. Auch nachdem das Gebäude fertiggestellt ist, kann BIM für ein computergestütztes Facility Management und für die Gebäudeautomatisierung genutzt werden (vgl. Van Treeck et al.: S. 43ff.). Der Aspekt der Brandschutzplanung soll im Folgenden konkretisiert werden. Wird anhand des Gebäudemodells ein Graph erzeugt, lässt sich überprüfen, ob gesetzliche Vorgaben für Fluchtwege eingehalten werden.

Aufbau der Datenstruktur: Wie in Unterkapitel 3.3.1 beschrieben, bestehen Graphen aus Knoten und Kanten. Die in diesem Lehrkonzept verwendete Datenstruktur und Implementierung des Dijkstra-Algorithmus orientiert sich stark an Mark Allen Weiss Umsetzung. In Java wird zum einen die Klasse Vertex erstellt. Ein Knoten speichert seinen Namen und die Kanten, die von ihm ausgehen. Zum anderen wird die Klasse Edge erstellt, welche speichert, wohin die Kante führt und wie hoch die Kosten sind. Die Klasse Graph speichert in einer HashMap, aus welchen Knoten der Graph aufgebaut ist. Die Benutzung einer HashMap bietet sich an, da über einen Schlüssel auf das gewünschte Knotenobjekt zugegriffen werden kann. Würden die Knoten in einer ArrayList gespeichert werden, müsste hinterlegt werden, an welchem Index sich ein bestimmter Knoten befindet. Der Schlüssel, um in der HashMap Zugriff auf den Knoten zu bekommen, ist lediglich der Name des Knotens. Die Klasse Graph beinhaltet des Weiteren Methoden, um Knoten und

Kanten zu erstellen sowie die Referenz auf Knoten zurückzugeben. Mithilfe einer eingelesenen Adjazenzmatrix kann ein Graph erstellt werden.

Dijkstra-Algorithmus: Um den Dijkstra-Algorithmus zur Berechnung des kürzesten Weges anwenden zu können, müssen einige Anpassungen an der Datenstruktur vorgenommen werden. Ein Knoten muss neben seinem Namen und den ausgehenden Kanten die aktuell kürzeste Distanz zum Startknoten speichern. Wurde ein Knoten noch nicht entdeckt, ist dieser Wert unendlich groß. Zusätzlich wird gespeichert, welcher Knoten auf dem aktuell kürzesten Weg der Vorgänger ist. Abschließend kennzeichnet eine boolesche Variable, ob ein Knoten als abgearbeitet gilt. Des Weiteren wird die Klasse Path erstellt. Ein Pfad speichert, wohin er führt und wie lang er ist. Um Objekte der Klasse Path vergleichen zu können, wird das Comparable-Interface implementiert. Anhand der Länge können Pfade verglichen werden. Diese Vergleichbarkeit wird genutzt, damit der „gierige“ Dijkstra-Algorithmus den besten Pfad weiterverfolgen kann. Die Implementierung und genaue Vorgehensweise des Dijkstra-Algorithmus werden im Seminar 9 ausführlich beschrieben und werden deswegen an dieser Stelle nicht weiter thematisiert. Die Ausgabe des endgültigen Lösungspfades erfolgt rekursiv, da jeder Knoten seinen Vorgänger auf dem kürzesten Weg speichert. Der Aspekt, dass die Ausgabe rekursiv erfolgt, ist für das Verständnis des Algorithmus nicht von Bedeutung. Vielmehr leitet dieser Kunstgriff fließend in das nächste Themengebiet über.

Fluchtwegberechnung: Im Tutorium sollen mit Hilfe des Dijkstra-Algorithmus mögliche Fluchtwege ermittelt werden. Da bei einfachen Bürogebäuden die Fluchtwegführung trivial ist, wurde das Parkhaus im 4. Obergeschoss der Centrum Galerie Dresden ausgewählt. Durch viele miteinander verbundenen Wege und mehreren Notausgängen ist die Fluchtwegführung nicht offensichtlich. Die Studierenden bekommen einen vorbereiteten Graphen, der die sicheren Bewegungsflächen repräsentiert. Anschließend sollen die nach sächsischer Garagen- und Stellplatzordnung zulässigen Notausgänge ausgewählt und der kürzeste Weg zu ihnen ermittelt werden.

Werden die Vorschläge des Algorithmus mit der Realität verglichen, stellen sich Abweichungen heraus. Der Fachplaner für vorbeugenden Brandschutz Bastian Funcke vom Institut für Bauklimatik der TU Dresden erklärte in einer Konsultation, dass es sich bei der Centrum Galerie Dresden um einen Sonderbau handeln würde. Demnach müsse ein Brandschutzkonzept vorliegen, das eventuelle Kompensationsmaßnahmen vorschlägt, wenn von Anforderungen der jeweiligen Verordnung abgewichen werde. Außerdem würden Autos nicht als aufgehende Bauteile gesehen werden, weswegen die gesetzliche Fluchtweglänge als Luftlinie zu bemessen ist.

Kernbotschaft dieses Tutoriums ist, dass die Rechengeschwindigkeit von Computern besonders bei unübersichtlichen Graphen viele Vorteile mit sich bringt. Änderungen an der Position oder Anzahl der Notausgänge können unkompliziert berechnet werden. Des Weiteren werden Fehlplanungen entdeckt, die in der Masse wahrscheinlich untergegangen wären. Es werden allerdings auch Grenzen aufgezeigt. So ist eine Adjazenzmatrix nicht die geeignetste Form, um einen Graphen mit so vielen Knoten zu speichern. Von 5.776 Elementen speichern 5.538 den Wert Null (ca. 96%).

4.7 Anwendung rekursiver Algorithmen und Datenstrukturen

Lernziele: Nachdem mit der Ausgabe des kürzesten Weges „unbewusst" ein rekursiver Algorithmus angewendet wurde, behandelt das letzte Themengebiet ausführlich die Rekursion.

Einführung: Ein gut nachvollziehbares Standardbeispiel ist die Berechnung der Fakultät. Um 5! Berechnen zu können, muss 4! berechnet werden usw. Der wiederholte Selbstaufruf einer Methode, um ein Problem auf eine einfachere Version von sich selbst zurückzuführen, wird rekursiver Abstieg genannt. Dies geschieht solange, bis ein unmittelbar lösbarer Anfang erreicht wurde. Dieser „Base Case" wird auch als Terminierungs- bzw. Abbruchbedingung am Rekursionsanfang bezeichnet. Von dort aus folgt der rekursive Aufstieg, bei dem die Rückgabewerte der abgeschlossenen Inkarnationen zur Berechnung der Gesamtlösung benutzt werden (vgl. Lamm 2014; Weiss 2012: S. 12). Wird die Terminierungsbedingung fehlerhaft implementiert, ist „Stack Overflow" eine typische Fehlermeldung im Zusammenhang mit Rekursion. Mit jedem rekursiven Aufruf wird eine weitere Methode auf den Stapel mit wartenden Methoden gelegt. Erfolgt die Rekursion unaufhaltsam, läuft der Stapel nach kurzer Zeit über.

Die Umsetzung in Java unterscheidet sich nicht wesentlich von anderen Methodenaufrufen. Typischerweise befindet sich am Anfang des Methodenrumpfes die Abbruchbedingung. Im Anschluss folgt der rekursive Aufruf, der als Parameter eine einfachere Version des Problems an die nächste Rekursionsebene übergibt.

Arten der Rekursion: Die Berechnung der Fakultät ist ein Beispiel für eine lineare Rekursion. Ein rekursiver Aufruf löst maximal einen weiteren rekursiven Aufruf aus. Bei der verzweigten Rekursion löst ein rekursiver Aufruf zwei oder mehr rekursive Aufrufe aus. Als Beispiel kann die Berechnung der Fibonacci-Zahlen herangezogen werden. Die dritte und rechenintensivste Art heißt verschachtelte Rekursion. „Das Argument für den rekursiven Aufruf ruft selbst wieder die rekursive Methode auf.

Das Resultat ist eine extrem hohe Anzahl von Selbstaufrufen.“ (Dörn 2016: S. 241) Als Beispiel dafür kann die Berechnung der Ackermann-Funktion genannt werden. Diese Art von Funktionen werden unter anderem für Belastungstests von Computern eingesetzt (vgl. Dörn 2016: S. 242).

Lindenmayer-Systeme: Eine faszinierende Anwendung findet die Rekursion in den Lindenmayer-Systemen. Benannt nach dem ungarischen Biologen Aristid Lindenmayer ist ein L-System ein mathematisch formalisiertes Ersetzungssystem, welches zur Modellierung von Pflanzen genutzt wird (vgl. Prusinkiewicz/Lindenmayer 1996: S. vi). Ausgehend von einem einfachen geometrischen Objekt (Initiator) werden anhand von Produktionsregeln (Generator) einzelne Teile des Objektes ersetzt. Diese Regeln können theoretisch beliebig oft angewendet werden. Mit zunehmender Anzahl an Ersetzungen steigt der Detailgrad der Abbildung (vgl. Prusinkiewicz/Lindenmayer 1996: S. 1f.). Ein Einblick in erzeugte Muster und pflanzliche Darstellungen ist am Ende des Seminars 10 zu finden.

Binäre Suchbäume: Binäre Suchbäume wurden im Unterkapitel 3.3.2 bereits ausführlich vorgestellt. In Java werden sie vorzugsweise rekursiv umgesetzt. Auch für diese Datenstruktur wurde sich an Mark Allen Weiss‘ Umsetzung orientiert.

Aufbau und Operationen: Die Klasse Node speichert einen Wert und zusätzlich Referenzen auf die Wurzelknoten des linken und rechten Teilbaums. Die Klasse BST (Binary Search Tree) speichert lediglich eine Referenz auf den Wurzelknoten des gesamten Baums. Um einen Knoten hinzuzufügen, wird rekursiv im Baum soweit abgestiegen, bis ein Knoten auf der entsprechenden Seite keinen Nachfolger mehr hat. Eine Referenz auf das einzufügende Element wird dann an dieser Stelle gespeichert. Um einen Wert abzufragen, wird nach dem gleichen Schema vorgegangen. Kleine Änderung ist, dass, wenn der Wert gefunden wurde, „true“ zurückgegeben wird, beziehungsweise „false“, sollte der Wert nicht enthalten sein.

Traversierung: Es gibt drei gängige Varianten, einen binären Baum vollständig zu durchlaufen. Die Pre-order Traversierung gibt erst den Wert des Knotens aus und steigt danach in den linken und anschließend in den rechten Teilbaum ab. Die In-order Traversierung besucht erst den linken Teilbaum, gibt dann den Wert des Knotens aus und besucht abschließend den rechten Teilbaum. Diese Art der Traversierung arbeitet die Knoten in geordneter Reihenfolge von klein nach groß ab. Die Post-order Traversierung besucht erst beide Teilbäume und gibt zum Schluss den Wert des Knotens aus (vgl. Weiss 2010: S. 668).

Weitere Operationen: Das Entfernen und Rotieren von Knoten werden im Seminar 11 bebildert gezeigt und erklärt. Abschließend sind die Studierenden in der Lage,

balancierte binäre Suchbäume (AVL-Baum) händisch/manuell sowie mittels Java zu erstellen.

Backtracking: Abschließend wird sich dem Backtracking gewidmet. Eingeleitet wird das Themengebiet mit einem sehr intuitiven Beispiel, wie systematisch der Ausgang eines Labyrinths gefunden werden kann. Nach Auffinden einer Sackgasse wird soweit zurückgegangen, bis ein neuer Weg eingeschlagen werden kann. Dies geschieht so lange, bis der Ausgang gefunden wird.

Platzieren von Turmdrehkranen: In Unterkapitel 3.1.2 wurde das N-Queen-Problem als typisches Beispiel für einen Backtracking-Algorithmus genannt. Dieses Prinzip wird im letzten Tutorium auf das Bauingenieurwesen übertragen. Aufgabe ist es, eine bestimmte Anzahl Turmdrehkrane auf einer Baustelle so zu platzieren, dass der gesamte Grundriss des Bauwerks überschwenkt wird. Dabei sollen folgende Bedingungen eingehalten werden: Ein Kran ist so aufzustellen, dass kein Turm eines anderen Krans in seinem Schwenkbereich steht; ein Kran kann nur in der direkten Umgebung des Bauwerks aufgestellt werden; jeder Kran hat dieselbe Auslegerlänge. Das Vorgehen orientiert sich stark an dem N-Queen-Problem. Es werden alle Krane platziert und sollte das Bauwerk in der gewählten Konstellation nicht vollständig überschwenk sein, wird der zuletzt platzierte Kran verschoben. Dies geschieht solange, bis eine akzeptable Lösung gefunden wurde.

Um dieses letzte Tutorium erfolgreich bearbeiten zu können, müssen die Studierenden viele der zuvor erlernten Fähigkeiten und Kompetenzen anwenden. Zusätzlich ist der Ingenieurverstand gefragt, denn zum einen muss sich in dem umfangreichen Projekt zurechtgefunden werden. Zum anderen muss die Lösung eines ähnlichen Problems auf die gegebene Fragestellung übertragen werden. Abschließend wird den Studierenden verdeutlicht, dass kleinste Ungenauigkeiten in der Implementierung eines Algorithmus exorbitante Auswirkungen auf die Laufzeit eines Programms haben können. Wird in diesem Beispiel beim rekursiven Abstieg nicht mitgegeben, welche Standorte bereits untersucht wurden, vertausendfacht sich die benötigte Zeit des Computers, um eine Lösung auszugeben.

5 Schlussbetrachtung

Abgeschlossen wird diese Arbeit mit einer Diskussion, in der der Stellenwert, die Auswahl der Lehrinhalte sowie die Methodik und Durchführung kritisch hinterfragt werden. Als eine Art Ausblick kann das Unterkapitel zur Anwendbarkeit des „Flipped Classroom“ gesehen werden. Zusammenfassend wird am Schluss ein Fazit gezogen.

5.1 Diskussion

Stellenwert der Arbeit

Um die Digitalisierung im deutschen Baugewerbe voran zu treiben, müssen zukünftige Bauingenieure bereits im Studium über die zahlreichen Möglichkeiten der neuen Technologien informiert werden. Diese Arbeit leistet einen ersten Beitrag, Studierende an Fragestellungen der Bauinformatik heranzuführen. Darüber hinaus verfügen die Studierenden nach Abschluss des Moduls über umfassendes, anwendungsbereites Grundwissen im Bereich Software Engineering.

Das erstellte Lehr-Lernkonzept ist auf den Bachelor- sowie Diplomstudiengang Bauingenieurwesen der Technischen Universität Dresden zurechtgeschnitten. Die Verwendung an anderen Universitäten erfordert gegebenenfalls einige Anpassungen.

In Anbetracht der Globalisierung und der damit verbundenen Internationalisierung ist ein berechtigter Kritikpunkt, dass die Lehrangebote nicht in englischer Sprache entwickelt wurden. Da viele Begriffe der Informatik ursprünglich aus dem Englischen kommen und teilweise keine geeigneten Übersetzungen existieren, ist die Entscheidung, Deutsch als Lehrsprache zu wählen, mit einem Mehraufwand verbunden. Nicht selten treten dabei Schwierigkeiten auf, englische Begriffe sinnvoll in einen deutschen, grammatikalisch korrekten Satz einzubauen. Dennoch wurde sich gegen Englisch als Lehrsprache entschieden. Hauptsächlich aus einem Grund: Die Studierenden werden in diesem Modul mit Lehrinhalten einer anderen Fachdisziplin konfrontiert. Das Tempo, in dem diese neuen Inhalte vermittelt werden, kann als zügig eingestuft werden. Kommt dann auch noch dazu, dass die Erklärungen nicht in ihrer Muttersprache erfolgen, ist die Wahrscheinlichkeit sehr hoch, dass ein Großteil der Studierenden den Anschluss verliert.

Auswahl der Lehrinhalte

Dass ein Modul nicht die gesamte Bauinformatik abbilden kann, ist offensichtlich. Trotzdem sollte kritisch hinterfragt werden, ob die vermittelten Lehrinhalte für das Erreichen der Qualifikationsziele geeignet sind.

Das Beherrschen der Grundbegriffe der Programmierung sowie das Verständnis verschiedener Programmierparadigmen sind für die Zukunft unerlässlich. Des Weiteren ist die Fähigkeit, ein eigenes Programm zu schreiben, für einen Ingenieur sehr vorteilhaft. Das erstellte Lehrkonzept geht auf die Grundbegriffe ein und erklärt diese im Kontext des strukturierten und objektorientierten Programmierens. Parallel werden die wesentlichen Phasen des Softwareentwurfs abgedeckt. Neben dem Einlesen und Ausgeben von Dateien wird den Studierenden zusätzlich gezeigt, wie eine grafische Benutzeroberfläche zur Programmsteuerung erstellt werden kann. Zusätzlich erhalten die Studierenden ein Überblick, wie vielfältig Algorithmen und Datenstrukturen sein können.

Methodik und Durchführung

Da die Studienordnung gewisse Rahmenbedingungen festlegt, besteht in Bezug auf den Umfang und die Anzahl von Lehrveranstaltungen wenig Flexibilität. Bei der Gestaltung und Planung der Lehrveranstaltungen diente Literatur von renommierten Pädagogen und Psychologen als Handlungshinweise, wie die Lehrinhalte idealerweise vermittelt werden sollten. Kritisch sollte betrachtet werden, dass überwiegend Literatur von Hilbert Meyer benutzt wurde. Seine Lehrmethoden basieren größtenteils auf Frontalunterricht, welcher als veraltete Lehrform gilt. Deswegen wird im folgenden Unterkapitel die Anwendbarkeit des „Flipped Classroom“ diskutiert.

Ob das entworfene Lehrkonzept auch in der Praxis standhält, konnte noch nicht ermittelt werden. Ein erster Prototyp wurde im Studienjahr 2019/2020 unterrichtet und lieferte hoffnungsvolle Ergebnisse. Aus Sicht der Lehrenden verliefen die beiden Semester im Vergleich zum Vorjahr harmonisch und abgestimmt. Insgesamt ließ sich bei den Studierenden ein höheres Interesse an dem Stoffgebiet beobachten. Nichtsdestotrotz wird davon ausgegangen, dass im Laufe der Jahre kleine Anpassungen an diesem Lehrkonzept vorgenommen werden müssen.

5.2 Anwendbarkeit des „Flipped Classroom“

Im aktuellen Zustand wird mit einer hohen Lehrerdominanz in ein Themengebiet eingeführt. Im Laufe der Lehrveranstaltungskaskade verschiebt sich das Verhältnis, sodass mit einer hohen Schülerdominanz abgeschlossen wird (vgl. Unterkapitel 4.1). Dieses Verhältnis kann unter bestimmten Umständen umgedreht werden. In der Pädagogik wird dabei vom Konzept des „Flipped Classroom“ gesprochen. Die Studierenden arbeiten sich im Selbststudium in ein Themengebiet ein. Als unterstützendes Material werden beispielsweise Aufzeichnungen von Vorlesungen und Seminaren, Erklärvideos oder Literatur zur Verfügung gestellt. Nachdem die Studierenden ein Grundverständnis für die Thematik entwickelt haben und sich beim Lösen erster Aufgaben Fragen und Probleme ergeben, kommt das Lehrpersonal ins Spiel (vgl. Spannagel 2015).

Christian Spannagel ist Professor für Mathematik und wendet das Konzept des „Flipped Classroom“ in einigen seiner Lehrveranstaltungen an. Er hat seine Vorlesungen aufgezeichnet und stellt diese den Studierenden für einen bestimmten Zeitraum zur Verfügung. Nachdem die Studierenden ein Themengebiet im Selbststudium erarbeitet haben, nehmen sie an Seminaren teil, bei den unter Hilfestellung von Tutoren die ersten Aufgaben bearbeitet werden. Erst danach findet eine Lehrveranstaltung mit dem Professor im Hörsaal statt. Spannagel bezeichnet diese Veranstaltung als Plenumsveranstaltung, in der die „kniffeligen“ Fragen, auf die selbst die Tutoren keine Antwort hatten, besprochen werden. In einem Interview erklärt er, welche Fehler zum Versagen des „Flipped Classroom“ führen. Unteranderem dürfe in den Plenumsveranstaltungen keinesfalls eine Wiederholung stattfinden. Selbst wenn eine Mehrzahl der Studierenden unvorbereitet sind, müsse konsequent geblieben werden (vgl. Spannagel 2015).

Die Anwendung dieses Konzeptes ist prinzipiell auch mit dem entwickelten Lehrkonzept denkbar. Allerdings sollten erst einige Durchläufe stattgefunden haben, um Kinderkrankheiten des Lehrkonzeptes zu finden und auszubessern. Des Weiteren müsste das Material für den Zweck des ausschließlichen Selbststudiums angepasst werden.

5.3 Fazit

Das in der Arbeit vorgestellte Lehr-Lernkonzept leistet einen ersten Beitrag, Studierende an Fragestellungen der Bauinformatik heranzuführen. Mit Durchlaufen der Lehrveranstaltungen entwickeln die Studierenden ein Verständnis über grundlegende Methoden und Techniken zum Entwurf und Betrieb von komplexen Softwaresystemen. Sie werden darauf vorbereitet Informationstechnologien zu nutzen, anzupassen und weiterzuentwickeln. Darüber hinaus sind die Studierenden in der Lage, Ergebnisse von Problemanalysen für IT-Dienstleister zu beschreiben und als Fachspezialist an einer Lösung mitzuwirken.

Im Rahmen dieser Arbeit wurde ein flexibles Lehr-Lernkonzept für den Erwerb von Grundlagenwissen im Bereich des Software Engineering und komplementären Fertigkeiten im Bereich des Programmierens mit einer objektorientierten Programmiersprache für Bauingenieure entwickelt. Es wurde besondere Beachtung auf die ausgewogene Vermittlung von theoretischen Grundlagen zum Softwareentwurf und der praktischen Umsetzung der Konzepte gelegt.

Innovative Lehr-Lehrformen wurden analysiert und in den Lösungsvorschlag eingearbeitet. Inhalte werden an die Studierenden über vielfältige Eingangskanäle herangetragen und die Bedürfnisse der unterschiedlichen Lerntypen werden beachtet. Des Weiteren ist das Lehr-Lernangebot sowohl für Direktstudenten als auch für Formen des lebenslangen Lernens geeignet.

Literaturverzeichnis

Monographien und Aufsätze in Fachzeitschriften

Balzert 2009	Balzert, Helmut (2009): Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering, 3. Auflage, Heidelberg, Deutschland: Spektrum Akademischer Verlag.
Balzert 2001	Balzert, Helmut (2001): Lehrbuch der Software-Technik: Software-Entwicklung, 2. Auflage, Heidelberg, Deutschland: Spektrum Akademischer Verlag.
Bertschek et al. 2019	Bertschek, Irene / Thomas Niebel / Jörg Ohnemus (2019): Zukunft Bau: Beitrag der Digitalisierung zur Produktivität in der Baubranche, Endbericht, Mannheim, Deutschland: ZEW.
Dijkstra 1968	Dijkstra, Edsger (1968): Go To Statement Considered Harmful. In: Communications of the ACM, Volume 11, Number 3.
Dörn 2016	Dörn, Sebastian (2016): Programmieren für Ingenieure und Naturwissenschaftler: Grundlagen, Berlin, Deutschland: Springer Vieweg.
Dörn 2017	Dörn, Sebastian (2017): Programmieren für Ingenieure und Naturwissenschaftler: Algorithmen und Programmier-techniken, Berlin, Deutschland: Springer Vieweg.
Heinisch et al. 2011	Heinisch, Cornelia / Frank Müller-Hofmann / Joachim Goll (2011): Java als erste Programmiersprache: Vom Einsteiger zum Profi, 6. Überarbeitete Auflage, Wiesbaden, Deutschland: Vieweg+Teubner.
Jank 2019	Jank, Werner / Hilber Meyer (2019): Didaktische Modelle, 13. Auflage, Berlin, Deutschland: Cornelsen.
Kölling 2010	Kölling, Michael (2010): Einführung in Java mit Greenfoot: Spielerische Programmierung mit Java, München, Deutschland: Pearson Schule.
Meyer 2011	Meyer, Hilbert (2011): Was ist guter Unterricht? 8. Auflage, Berlin, Deutschland: Cornelsen.

Prusinkiewicz/ Lindenmayer 1996	Prusinkiewicz, Przemyslaw / Aristid Lindenmayer (1996): The Algorithmic Beauty of Plants, New York, USA: Springer-Verlag.
Van Treeck et al. 2016	Van Treek, Christoph / Robert Elixmann / Klaus Rudat / Sven Hiller / Sebastian Herkel / Markus Berger (2016): Building Information Modelling, in: Claus Holst-Gydesen (Hrsg.), Gebäude. Technik. Digital. Building Information Modelling, Berlin, Heidelberg, Deutschland: Springer Vieweg.
Vester 2011	Vester, Frederic (2011): Denken, Lernen, Vergessen: Was geht in unserem Kopf vor, wie lernt das Gehirn, und wann läßt es uns im Stich? 34. Auflage, München, Deutschland: Deutscher Taschenbuch Verlag.
Von Rimscha 2008	Von Rimscha, Markus (2008): Algorithmen kompakt und verständlich: Lösungsstrategien am Computer, Wiesbaden, Deutschland: Vieweg+Teubner.
Weiss 2010	Weiss, Mark Allen (2010): Data Structures & Problem Solving Using Java, 4th Edition, Boston, USA: Pearson Education.
Weiss 2012	Weiss, Mark Allen (2012): Data Structures and Algorithm Analysis in Java, 3rd Edition, Boston, USA: Pearson Education.
Wellenreuther 2018	Wellenreuther, Martin (2018): Lehren und Lernen – aber wie? Ein Studienbuch für das Lehramtsstudium, 9., vollständig überarbeitete und erweiterte Auflage, Baltmannsweiler, Deutschland: Schneider Verlag Hohengehren.

Internet

Lamm 2014	Lamm, Wigburg (2014): Programmieren in Logik: 3. Rekursion, in SlidePlayer.org, https://slideplayer.org/slide/1626/ [06.09.2020].
Luber/Augsten 2017	Luber, Stefan / Stephan Augsten (2017): Definition „Programmierschnittstelle“: Was ist eine API? In dev-insider.de, https://www.dev-insider.de/was-ist-eine-api-a-583923/ [21.09.2020].
MIQR	4 Lerntypen nach Vester (Übersicht) – mit Lerntypentest, in mitteldeutsches-institut.de, https://mitteldeutsches-institut.de/lerntypen/ [28.09.2020].
Ramachandran 2015	Ramachandran, Ranjith (2015): How HashMap works in Java? With Animation!! whats new in java8 tutorial, in youtube.com, https://youtu.be/c3RVW3KGIIE [21.09.2020].
Spannagel 2015	Spannagel, Christian (2015): Flipped Classroom: Was tun, wenn viele Studierende unvorbereitet sind? In youtube.com, https://youtu.be/z9AZDBE7gF8 [04.10.2020].
smarter-learning.de	Klassische Lerntypen nach Vester, in smarter-learning.de, https://smarter-learning.de/lerntypen/klassische-lerntypen-nach-vester/ [28.09.2020].
wachtler.de	3.1.1 Referenzsemantik, Wertsemantik, in wachtler.de, http://www.wachtler.de/informatik_2/node13.html [20.09.2020].
Weitz 2020	Weitz, Edmund (2020): Wie kommt der Zufall in den Computer? (Teil 1 von 2), in youtube.com, https://youtu.be/Q4J0-uvAuPI [20.09.2020].

Konsultationsverzeichnis

Bastian Funcke	Wissenschaftlicher Mitarbeiter am Institut für Bauklimatik der TU Dresden und Fachplaner „Vorbeugender Brandschutz“, Dresden, +49 351 463-36135, 31. Januar 2020.

Anlagenverzeichnis

Anlage 1	Übersichtstabelle Lehrveranstaltungen

Digitale Anlagen:

Anlage 2	Ordner „BIW1-07_Bauinformatik_Grundlagen“
	Quellcode aller Beispiele aus den Seminaren und Aufgaben der Tutorien
Anlage 3	Ordner „Dokumentation“
	Javadoc-Dokumentation des Quellcodes
Anlage 4	Ordner „Seminar“
	Präsentationsfolien aller 12 Seminare
Anlage 5	Ordner „Tutorium“
	Aufgabenstellungen und Erklärvideos (inkl. Skript) aller 12 Tutorien
Anlage 6	Übersicht Programmieren mit Java

Anlage 1: Übersichtstabelle Lehrveranstaltungen

Nr.	Vorlesung	Seminar	Tutorium	Übungsaufgabe
	Hören/Sehen	*Verstehen*	*Anwenden*	*Festigen*
1	Introduction into Programming	Strukturiertes Prog. Teil 1	Einführung in Java	Übungsaufgabe Teil 1
2	Array and ArrayList	Strukturiertes Prog. Teil 2	Vertiefende Prog.-Beispiele	
3	Introduction to Bauinformatik & OO-Programming	Datenstrukturen der Java API	ArrayList	
4	Sorting Algorithms	Sortieralgorithmen und objekt-orientiertes Prog. Teil 1	Sortieralgorithmen & Objekte	
5	OO-Programming	objektorientiertes Prog. Teil 2	Übungsaufgabe Teil 2: „little BIM"	
6	OO-Programming and Inheritance	objektorientiertes Prog. Teil 3		
7	GUI and Libraries	Grafische Benutzeroberflächen		
8	Graphs	Graphentheorie	Graphentheorie	
9	Greedy-Algorithms	Dijkstra-Algorithmus	Fluchtwege berechnen	
10	Recursive Algorithms	Rekursion	Rekursion	
11	Binary Trees	Binärer Baum & AVL-Baum	Binärer Baum & AVL-Baum	
12	Backtracking	Backtracking	Backtracking	Übungsaufgabe Teil 3

Erklärung zur Anfertigung der Bachelorarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form einer anderen Prüfungsbehörde vorgelegt und ist auch noch nicht veröffentlicht worden.

Dresden, den 16. Oktober 2020

Fabian Collin