1 Einleitung

Die *Unified Modeling Language* (UML) bildet eine Vereinigung der Best Practices von Modellierungstechniken, die sich über Jahre hinweg etablieren konnten. UML erlaubt es, die unterschiedlichsten Aspekte eines Softwaresystems (z. B. Anforderungen, Datenstrukturen, Daten- und Informationsflüsse) innerhalb eines Rahmenwerks auf einheitliche Weise mittels objektorientierter Konzepte darzustellen. Bevor wir uns in die Tiefen von UML vorwagen, erläutern wir in diesem Kapitel zunächst kurz, warum Modellieren aus der Softwareentwicklung nicht mehr wegzudenken ist. Dafür gehen wir der Frage nach, was ein Modell ist und wofür Modelle gebraucht werden. Wir wiederholen kurz grundlegende Konzepte der Objektorientierung, bevor wir einen Überblick über den weiteren Aufbau des Buchs geben.

Unified Modeling Language (UML)

1.1 Motivation

Stellen Sie sich vor, Sie möchten ein Softwaresystem entwickeln, das ein Kunde bei Ihnen in Auftrag gegeben hat. Eine der ersten Herausforderungen, mit der Sie konfrontiert werden, ist das Klären der Fragen, was der Kunde eigentlich genau will und ob Sie die genauen Anforderungen des Kunden an das zukünftige System korrekt erfasst haben. Schon von diesem ersten Schritt hängt das Gelingen oder das Scheitern Ihres Projekts ab. Sofort stellt sich die Frage, wie Sie mit Ihrem Kunden kommunizieren. Natürliche Sprache stellt hierbei nicht unbedingt die erste Wahl dar, da diese unpräzise und mehrdeutig ist. Missverständnisse können sehr leicht auftreten, und die Gefahr ist sehr groß, dass Leute mit unterschiedlichen Hintergründen (z. B. Informatiker und Betriebswirt) aneinander vorbeireden, was fatale Folgen haben kann.

Was Sie jetzt benötigen, ist die Möglichkeit, ein Modell für Ihre Software zu erstellen, das die wesentlichen Aspekte in einer möglichst einfachen und eindeutigen Notation hervorhebt, aber von irrelevanten Details abstrahiert, genauso wie es bei Modellen in

der Architektur, wie z. B. den Bauplänen, passiert. Zum Beispiel enthält ein Bauplan für ein Gebäude zwar Informationen, wie der Grundriss des zu bauenden Hauses auszusehen hat, die zu verwendenden Baustoffe werden hier aber nicht angegeben, da sie im Moment keinerlei Bedeutung haben und den Plan unnötig aufblasen würden. Genauso finden wir hier keinerlei Informationen, wie die elektrischen Leitungen zu verlegen sind. Hierfür wird ein eigener Plan erstellt, um nicht zu viele Informationen auf einmal darzustellen. Auch in der Informatik ist es wie in der Architektur wichtig, dass Leute mit unterschiedlichen Hintergründen (z. B. Architekt und Bauherr) das Modell lesen, interpretieren und umsetzen können.

Modellierungssprache

Genau für solche Szenarien wurden *Modellierungssprachen* entwickelt, die klar definierte Regeln zur strukturierten Beschreibung eines Systems aufweisen. Solche Sprachen können *textuell* (z. B. eine Programmiersprache wie Java) oder *visuell* sein (z. B. eine Sprache, die miteinander verbindbare Symbole für Transistoren, Dioden etc. zur Verfügung stellt). Modellierungssprachen können speziell für eine bestimmte Domäne, z. B. für die Beschreibung von Webanwendungen, bestimmt sein. Diese *domänenspezifischen Modellierungssprachen* geben Möglichkeiten und Richtlinien zur Problemlösung in einem bestimmten Bereich vor, können dafür aber auch stark einschränkend sein. Oder Modellierungssprachen sind universell einsetzbar. Die Sprache UML, die wir in diesem Buch betrachten, gehört der Kategorie der universell einsetzbaren Sprachen an. Anhand von UML lernen wir die wichtigsten Konzepte der objektorientierten Modellierung kennen.

Domänenspezifische Modellierungssprache

Universelle Modellierungssprache

Objektorientierte Modellierung

Objektorientierte Modellierung ist eine Form der Modellierung, die dem objektorientierten Paradigma gehorcht. In den nachfolgenden zwei Unterkapiteln gehen wir auf den Modellbegriff und die wesentlichen objektorientierten Konzepte kurz ein, um uns dann der objektorientierten Modellierung mit UML widmen zu können.

1.2 Modell

System

Modelle erlauben eine effiziente und elegante Beschreibung von Systemen. Ein *System* ist eine Gesamtheit von Elementen, die so aufeinander bezogen sind und in einer Weise wechselseitig wirken, dass sie als eine aufgaben-, sinn- oder zweckgebundene Einheit angesehen werden können und sich in dieser Hinsicht gegenüber der sie umgebenden Umwelt abgrenzen [Wik11]. Beispiele für Sys-

teme sind materielle Dinge wie Autos oder Flugzeuge, ökologische Umgebungen wie Seen und Wälder, aber auch Organisationseinheiten wie eine Universität. Im Bereich der Informatik sind wir speziell an Softwaresystemen und damit einhergehend an Modellen interessiert, die Softwaresysteme beschreiben.

Softwaresystem

Softwaresysteme selbst basieren auf *Abstraktionen*, die eine maschinenverarbeitbare Form von Sachverhalten der Realität darstellen. Unter Abstraktion versteht man in diesem Zusammenhang eine Verallgemeinerung, das Absehen vom Besonderen und Einzelnen, das Loslösen vom Dinglichen. Abstraktion ist das Gegenteil von Konkretisierung. Unter Abstrahieren versteht man dementsprechend das Abgehen vom Konkreten, das Herausheben des Wesentlichen aus dem Zufälligen, das Erkennen gleicher Merkmale [Bal09].

Abstraktion

Bei der Erstellung von Softwaresystemen ist die Wahl geeigneter Abstraktionsmittel einerseits für die Umsetzung, andererseits aber auch für deren spätere Benutzung extrem wichtig. Wählt man die richtigen Abstraktionsmittel, so geht das Programmieren leicht von der Hand. Die einzelnen Teile haben dann einfache und kleine Schnittstellen. Neue Funktionalität kann ohne größere Reorganisation eingeführt werden. Wählt man die falschen Abstraktionsmittel, so ergeben sich bei der Umsetzung eine Vielzahl von bösen Überraschungen: Die Schnittstellen werden schwerfällig und Änderungen sind nur mühsam einzupflegen. Nur durch geeignete Abstraktionsmittel ist es möglich, mit der ständig steigenden Komplexität von modernen Softwaresystemen umzugehen [Jac11]. Gerade die Modellierung kann hier wertvolle Dienste leisten.

Wahl von Abstraktionsmittel

Um ein besseres Verständnis für den Modellbegriff zu entwickeln, werden nachfolgend weitverbreitete und allgemein anerkannte Definitionen des Modellbegriffs sowie die Eigenschaften, die ein gutes Modell aufweisen soll, vorgestellt.

Der Begriff *Modell* spielt nicht nur in der Informatik, sondern auch in vielen anderen wissenschaftlichen Disziplinen (Mathematik, Philosophie, Psychologie, Wirtschaftswissenschaften etc.) eine bedeutende Rolle. Abgeleitet von dem lateinischen Wort »modulus«, das einen Maßstab in der Architektur bezeichnet, wurde in Italien während der Renaissance das Wort »modello« für ein Anschauungsobjekt verwendet, anhand dessen Auftraggeber Form und Gestaltung eines geplanten Gebäudes vorgeführt bekamen sowie konstruktive und architektonische Fragestellungen geklärt wurden [FHR08]. Über die nachfolgenden Jahrhunderte fand der Begriff »Modell« Einzug in diverse Wissenschaftszweige, und zwar

Modellbegriff

zur vereinfachenden Beschreibung von komplexen Sachverhalten aus der Realität.

Definition von H. Stachowiak

1973 schlug Herbert Stachowiak einen Modellbegriff vor, der durch drei Merkmale gekennzeichnet ist [Sta73]:

1. *Abbildung.* Ein Modell ist immer ein Abbild von etwas, eine Repräsentation natürlicher oder künstlicher Originale, die selbst wieder Modelle sein können.
2. *Verkürzung.* Ein Modell erfasst nicht alle Attribute des Originals, sondern nur diejenigen, die dem Modellierer bzw. dem Benutzer des Modells relevant erscheinen.
3. *Pragmatismus.* Pragmatismus bedeutet so viel wie Orientierung am Nützlichen. Die Zuordnung eines Modells zum Original wird durch die Fragen *Für wen?*, *Warum?* und *Wozu?* getroffen. Ein Modell wird vom Modellierer bzw. von seinem Benutzer innerhalb einer bestimmten Zeitspanne und zu einem bestimmten Zweck anstelle eines Originals eingesetzt. Das Modell wird somit interpretiert.

Modelle unterstützen eine auf das Wesentliche reduzierte Darstellung eines Systems, um seine Komplexität auf handhabbare Teilaspekte zu verringern. Ein System wird üblicherweise nicht durch eine einzige Sicht beschrieben, sondern durch eine Vielzahl von Sichten, die zusammen ein einheitliches Gesamtbild ergeben. So kann sich eine Sicht auf die Beschreibung der involvierten Objekte und ihre Beziehungen zueinander beziehen, eine andere Sicht hingegen kann die Beschreibung des Verhaltens einer Gruppe von Objekten beinhalten oder die Interaktionen zwischen unterschiedlichen Objekten darstellen.

Eigenschaften von Modellen

Modelle müssen sorgfältig und wohlbedacht erstellt werden. Nach Bran Selic [Sel03] bestimmen folgende fünf Charakteristika die Qualität eines Modells:

- *Abstraktion.* Ein Modell ist immer eine reduzierte Darstellung des Systems, das es abbildet. Indem für eine Sicht irrelevante Details versteckt oder entfernt werden, ist es für den Benutzer leichter, die Essenz des Ganzen zu erfassen.
- *Verständlichkeit.* Es ist unzureichend, sich zur Erhöhung der Verständlichkeit auf das Weglassen irrelevanter Details zu beschränken. Vielmehr noch ist es wichtig, dass die verbleibenden Elemente in einer möglichst intuitiven Form, z. B. in einer grafischen Notation, dargestellt sind. Die Verständlichkeit ergibt sich direkt aus der Ausdrucksstärke der Modellierungssprache. Ausdrucksstärke kann als die Fähigkeit de-

finiert werden, einen komplexen Sachverhalt mit so wenig Information wie möglich darzustellen. Auf diese Weise reduziert ein gutes Modell den intellektuellen Aufwand, der notwendig ist, um den abgebildeten Sachverhalt zu verstehen. Zum Beispiel sind typische Programmiersprachen nicht besonders ausdrucksstark für den menschlichen Leser, da sehr viel Aufwand notwendig ist, um den Inhalt des Programms zu verstehen.

- *Genauigkeit.* Ein Modell muss eine möglichst realitätsgetreue Abbildung der relevanten, d. h. der nicht wegabstrahierten Eigenschaften des modellierten Systems ermöglichen.
- *Vorhersagewert.* Durch ein Modell muss es möglich sein, interessante, aber nicht offensichtliche Eigenschaften des modellierten Systems vorherzusagen. Dies kann entweder durch Simulation oder durch Analyse formaler Eigenschaften passieren.
- *Verhältnismäßigkeit.* Die Erstellung des Modells muss billiger sein als die Erstellung des modellierten Systems.

Modelle können für unterschiedliche Zwecke eingesetzt werden. So unterscheiden wir zwischen *deskriptiven* und *präskriptiven* Modellen [FHR08]. *Deskriptive Modelle* beschreiben das Element der Wirklichkeit, um einen gewissen Aspekt leichter verständlich zu machen. Zum Beispiel beschreibt ein Stadtplan eine Stadt derart, dass eine ortsunkundige Person Routen innerhalb dieser Stadt leichter finden kann. *Präskriptive Modelle* hingegen dienen dazu, eine Bauanleitung zur Erstellung eines zu entwickelnden Systems zu bieten.

Deskriptives Modell

Präskriptives Modell

In diesem Buch betrachten wir, wie die unterschiedlichen Aspekte eines Softwaresystems mithilfe einer Modellierungssprache, der Unified Modeling Language, modelliert werden können, sodass daraus ausführbarer Code manuell oder (teil-)automatisch abgeleitet bzw. eine leicht verständliche Dokumentation erstellt werden kann. Übrigens stellt auch der ausführbare Code, der in einer beliebigen Programmiersprache wie z. B. Java entwickelt wurde, wiederum ein Modell dar. Dieses Modell repräsentiert die zu lösende Problemstellung und ist für die Ausführung auf Computer optimiert. Zusammengefasst gibt es für Modelle folgende drei Verwendungsarten [Fow03b]:

Ausführbarer Code als Modell

- Modelle als Skizze
- Modelle als Blaupause
- Modelle als ausführbare Programme

Modelle als Skizze

Modelle werden als *Skizze* verwendet, um gewisse Aspekte eines Systems auf einfache Weise zu kommunizieren. Hierbei geht es nicht um eine vollständige Abbildung des Systems. Vielmehr zeichnen sich Skizzen durch ihre Selektivität aus, indem sie auf das für die Lösung eines Problems Wesentliche reduziert sind. Oft werden durch die Skizze alternative Lösungsansätze sichtbar, die dann im Entwicklerteam diskutiert werden. Modelle dienen also auch als Diskussionsgrundlage für Mitglieder eines Teams.

Modelle als Blaupause

Im Gegensatz zum Einsatz von Modellen als Skizze spielt Vollständigkeit eine große Rolle, wenn Modelle als *Blaupause* eingesetzt werden. Diese müssen genügend Details enthalten, sodass Entwickler daraus lauffähige Systeme erstellen können, ohne Entwurfsentscheidungen treffen zu müssen. Modelle als Blaupausen spezifizieren oft nicht das gesamte System, sondern nur bestimmte Teile. Zum Beispiel werden die Schnittstellendefinitionen zwischen Subsystemen festgelegt, wobei die internen Implementierungsdetails den Entwicklern überlassen werden. Handelt es sich bei den Modellen um Verhaltensbeschreibungen, so können diese auch simuliert und getestet und damit Fehler im Vorfeld erkannt werden.

Forward Engineering
Backward Engineering

Modelle als Skizzen und als Blaupausen können sowohl für *Forward Engineering* als auch für *Backward Engineering* eingesetzt werden. Beim Forward Engineering dient das Modell als Grundlage für die Erstellung von Code, während beim Backward Engineering das Modell aus dem Code generiert wird, um diesen auf leicht verständliche und übersichtliche Weise zu dokumentieren.

Modelle als ausführbare Programme

Schließlich können Modelle als *ausführbare Programme* genutzt werden. Dies bedeutet, dass Modelle so genau spezifiziert werden, dass daraus automatisch Code generiert werden kann. Im Kontext von UML wurde die modellgetriebene Softwareentwicklung in den letzten Jahren extrem populär, die ein Verfahren bietet, UML als Programmiersprache zu verwenden. Diese werden wir kurz in Kapitel 9 dieses Buchs kennenlernen, nachdem wir uns die Grundlagen von UML angeeignet haben. In manchen Praxisbereichen wie z. B. bei der Entwicklung von eingebetteten Systemen werden Modelle bereits anstelle von traditionellen Programmiersprachen eingesetzt. In anderen Bereichen findet rege Forschung statt, um die Entwicklung von Softwaresystemen auf eine neue und damit leichter wartbare und weniger fehleranfällige Abstraktionsebene zu heben.

1.3 Objektorientierung

Wenn wir objektorientiert modellieren wollen, müssen wir zuerst klären, was unter *Objektorientierung* zu verstehen ist. Die Motivation für die erstmalige Einführung der Objektorientierung in den 60er-Jahren in Form der Simulationssprache SIMULA [Hol94] war ein für den Menschen möglichst natürliches Paradigma zu verwenden, um die Welt zu beschreiben. Die objektorientierte Betrachtungsweise entspricht unserem Denken über die reale Welt. Es handelt sich dabei um ein Denken über eine Gesellschaft von autonomen Individuen, die einen festen Platz in dieser Gesellschaft einnehmen und dabei vordefinierte Pflichten erfüllen müssen.

Objektorientierung

Es gibt nicht eine festgelegte Definition für den Begriff Objektorientierung. Es gibt allerdings einen allgemein anerkannten Konsens, welche Eigenschaften Objektorientierung charakterisieren. Natürlich spielen bei objektorientierten Ansätzen Objekte eine zentrale Rolle. Vereinfacht gesehen sind Objekte Elemente im System, deren Daten und Operationen beschrieben werden und die miteinander interagieren und kommunizieren können. Im Allgemeinen erwartet man von einem objektorientierten Ansatz folgende Konzepte.

Klasse. In objektorientierten Ansätzen muss es möglich sein, *Klassen* zu definieren, die die Attribute und das Verhalten einer Menge von Objekten, den Instanzen einer Klasse, abstrakt beschreiben und so Gemeinsamkeiten von Objekten zusammenfassen. Zum Beispiel haben Personen einen Namen, eine Adresse und eine Sozialversicherungsnummer. Lehrveranstaltungen haben eine eindeutige Nummer, einen Titel und eine Beschreibung. Hörsäle haben eine Bezeichnung sowie einen Standort usw. Zusätzlich definiert eine Klasse eine Menge von zulässigen Operationen, die auf ihre Instanzen angewendet werden dürfen. Zum Beispiel kann man einen Hörsaal für einen gewissen Termin reservieren, ein Student kann sich für eine Prüfung anmelden etc. Auf diese Weise wird das Verhalten von Objekten beschrieben.

Klasse

Objekt. Die Instanzen einer Klasse werden als ihre *Objekte* bezeichnet. Beispielsweise ist »Hörsaal 1 der TU Wien« eine konkrete Ausprägung der Klasse Hörsaal. Ein Objekt zeichnet sich speziell dadurch aus, dass es eine eigene Identität aufweist, d. h., unterschiedliche Instanzen einer Klasse sind eindeutig identifizierbar. Zum Beispiel handelt es sich bei dem Beamer in Hörsaal 1 um ein anderes Objekt als bei dem Beamer in Hörsaal 2, auch wenn es

Objekt

baugleiche Geräte sind. Wir sprechen hier von *gleichen* Geräten, nicht aber vom *selben* Gerät. Anders verhält es sich bei konkreten Werten von Datentypen: Die Zahl »1«, die eine konkrete Ausprägung des Datentyps Integer darstellt, besitzt keine eigene Identität.

Ein Objekt befindet sich immer in einem bestimmten Zustand. Ein Zustand wird ausgedrückt durch die Werte seiner Attribute. Ein Hörsaal kann sich z. B. in dem Zustand »belegt« oder »frei« befinden. Außerdem weist ein Objekt Verhalten auf. Das Verhalten eines Objekts wird durch die Menge seiner Operationen beschrieben. Operationen werden durch das Senden einer Nachricht aktiviert.

Kapselung

Kapselung. Die *Kapselung* bezeichnet den Schutz vor unerlaubtem Zugriff auf den internen Zustand eines Objekts durch eine eindeutig definierte Schnittstelle. Verschiedene Ebenen der Sichtbarkeit der Schnittstellen helfen, verschiedene Zugriffsberechtigungen zu definieren. So gibt es in Java z. B. die Sichtbarkeiten `public`, `private` und `protected`, die den Zugriff für alle, nur innerhalb des Objekts und nur für Angehörige derselben Klasse bzw. aller Unterklassen zulassen.

Nachricht

Nachricht. Objekte kommunizieren miteinander durch *Nachrichten*. Eine Nachricht an ein Objekt stellt eine Aufforderung dar, eine Operation auszuführen. Das Objekt selbst entscheidet, ob und wie es diese Operation ausführt. Die Operation wird nur dann ausgeführt, wenn der Sender zum Aufruf der Operation berechtigt ist, was in Form von Sichtbarkeiten (siehe vorhergehender Absatz) geregelt werden kann, und wenn eine geeignete Implementierung vorhanden ist. Meist wird das Konzept des *Überladens* unterstützt, das es ermöglicht, eine Operation für verschiedene Typen von Parametern unterschiedlich zu definieren. Zum Beispiel verhält sich der Operator »+« für die Addition von ganzen Zahlen anders (z. B. `1 + 1 = 2`) als für die Konkatenation von Zeichenketten (z. B. `'a' + 'b' = 'ab'`).

Überladen

Vererbung

Vererbung. Das Konzept der *Vererbung* ist ein Mechanismus, neue Klassen aus existierenden Klassen abzuleiten. Eine aus einer bestehenden Klasse (= Oberklasse) abgeleitete Unterklasse erbt alle Attribute und Operationen (Spezifikation und Implementierung) der Oberklasse. Eine Unterklasse kann

- neue Attribute und/oder Operationen definieren,
- die Implementierung geerbter Operationen überschreiben und
- geerbte Operationen durch eigenen Code ergänzen.

Die Vererbung ermöglicht eweiterbare Klassen und in weiterer Folge den Aufbau von *Klassenhierarchien* als Basis objektorientierter Systementwicklung. Eine Hierarchie besteht aus Klassen mit ähnlichen Eigenschaften (z. B. Person ← Mitarbeiter ← Professor ← ...). *Klassenhierarchie*

Bei richtigem Einsatz bietet Vererbung eine Vielzahl von Vorteilen: Wiederverwendung von Programm- bzw. Modellteilen, wodurch Redundanzen und Fehler vermieden werden, konsistente Definition von Schnittstellen, Modellierungshilfe durch eine natürliche Kategorisierung der auftretenden Elemente und Unterstützung von inkrementeller Entwicklung, d. h. schrittweise Verfeinerung von allgemeinen Konzepten zu speziellen Konzepten.

Polymorphismus. Im Allgemeinen versteht man unter *Polymorphismus* die Fähigkeit, verschiedene Gestalt anzunehmen. Ein polymorphes Attribut kann im Laufe der Ausführung eines Programms Referenzen auf Objekte von verschiedenen Klassen haben. Bei der Deklaration dieses Attributs wird statisch zur Compilezeit eine Klasse (z. B. Person) zugeordnet. Zur Laufzeit kann dieses Attribut dynamisch auch an eine Unterklasse gebunden sein (z. B. Mitarbeiter oder Student). *Polymorphismus*

Eine polymorphe Operation kann auf Objekte verschiedener Klassen ausgeführt werden und jeweils eine andere Semantik haben. Dies kann auf mehrere Arten realisiert werden: (i) Bei *parametrisiertem Polymorphismus*, besser bekannt als Generizität, werden den Operationen die konkreten Klassen als Argumente übergeben, (ii) beim *Inklusionspolymorphismus* sind Operationen auf Klassen und auf ihre direkten und indirekten Unterklassen anwendbar, (iii) durch Umsetzung mittels *Überladen von Operationen* und (iv) mittels *Coercion*, also der Umwandlung von Typen. Bei den ersten zwei Realisierungsarten spricht man vom *universellen Polymorphismus*, die anderen beiden Arten werden als *Adhoc-Polymorphismus* bezeichnet [CW85].

UML basiert durchgängig auf objektorientierten Konzepten. Dies fällt besonders beim Klassendiagramm auf, das sehr einfach in eine objektorientierte Programmiersprache übersetzt werden kann. Das Klassendiagramm und mögliche Übersetzungen nach Java lernen wir in Kapitel 4 kennen.

1.4 Aufbau des Buchs

Nachdem wir uns in Kapitel 2 einen kurzen Überblick über UML verschafft haben, indem wir dessen Entstehungsgeschichte näher beleuchten und die 14 unterschiedlichen Diagramme kurz betrachtet haben, werden in Kapitel 3 die Konzepte des Anwendungsfalldiagramms eingeführt. Damit sind wir in der Lage, die Anforderungen, die an ein zu entwickelndes System gestellt werden, zu beschreiben. In Kapitel 4 wird das Klassendiagramm vorgestellt, das uns erlaubt, die Struktur eines Systems zu beschreiben. Um nun zusätzlich auch das Verhalten des Systems abzubilden, werden in Kapitel 5 das Zustandsdiagramm, in Kapitel 6 das Sequenzdiagramm und in Kapitel 7 das Aktivitätsdiagramm eingeführt. Das Zusammenspiel der unterschiedlichen Diagrammarten wird in Kapitel 8 anhand mehrerer Beispiele erläutert. Weiterführende Konzepte, die für den praktischen Einsatz von UML von maßgeblicher Bedeutung sind, werden abschließend in Kapitel 9 kurz betrachtet.

Da der UML-Standard auf Englisch verfasst wurde und auch die Modellierung – genauso wie viele andere Gebiete der Informatik – Englisch als Arbeitssprache einsetzt, werden die Bezeichnungen der Sprachelemente sowohl in Deutsch als auch in Englisch angegeben. Anhang A fasst alle englischen Originalbegriffe und die verwendeten deutschen Begriffe zusammen.

Die Konzepte werden durchgängig anhand von Beispielen erklärt. Alle Beispiele stammen aus dem universitären Bereich. Sie stellen meistens stark vereinfachte Szenarien dar. Wir wollen in diesem Buch kein durchgängiges System modellieren, da die Gefahr sehr hoch ist, dass wir uns in einer Vielzahl von technischen Details verlieren. Wir haben die Beispiele vielmehr entsprechend ihrem didaktischen Nutzen und entsprechend ihrer illustrativen Ausdrucksstärke gewählt. In vielen Fällen wurden daher Annahmen getroffen, die aus didaktischen Gründen auf vereinfachten Darstellungen der Wirklichkeit beruhen.