

- 3.1 Ziele und Historie**
- 3.2 Analyse**
- 3.3 Klassendiagramm**
- 3.4 Objektdiagramm**
- 3.5 Interaktionsdiagramm**
- 3.6 Zustandsdiagramm**
- 3.7 Entwurfsregeln**



Innensicht

Imperatives Programmieren

(Konstrukte aus Java)

- Praxis: Basistypen, Anweisungen, Verbunde, Felder, Schleifen, Rekursion
- Theorie: Syntaxbeschreibung, Induktion, Schleifeninvarianten
- Ausnahmebehandlung

Objektorientiertes Programmieren (Java)

- OO-Klassen
- Objekt und Zustände
- Methoden und Vererbung in Java
- Java: Einführung aller restlicher OO-Sprachkonstrukte

Außensicht

Dienste

- Funktionalität/Schnittstellen
- Qualitätsparameter: Aufwand, Zuverlässigkeit, Skalierbarkeit, Persistenz
- Algorithmenwahl

Objektorientierung

- Objekte, Zustände, Methoden
- Entwurf mittels UML
- Vererbung



Hohe Komplexität heutiger Softwaresysteme erfordert grundlegende **Analyse-** und **Designphase**.

- Z.B. komplexe Unternehmenssoftware ist mehr als ein paar Softwaremodule
- Skalierbarkeit, Sicherheit, Robustheit gefordert
- Klare Struktur (Software-Architektur), die Wartung ermöglicht

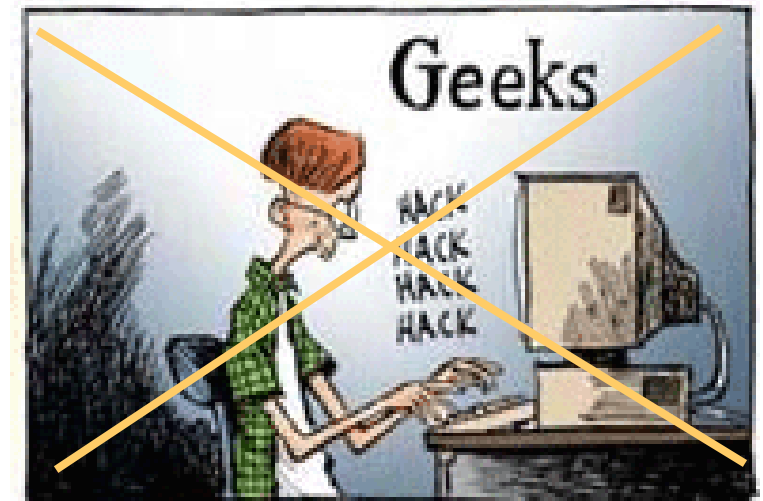
Gewünscht:

- Wiederverwendbarkeit von Code anstatt das Rad ständig neu zu erfinden

... nicht einfach drauf los programmieren!

→ Modellierung:

- Design einer Software-Anwendung
- Ergebnisse müssen übersichtlich und präzise ausdrückbar und darstellbar sein
 - nicht nur aus der Sicht von Informatikern!
- Modell hier vergleichbar mit Bauplan eines Hauses



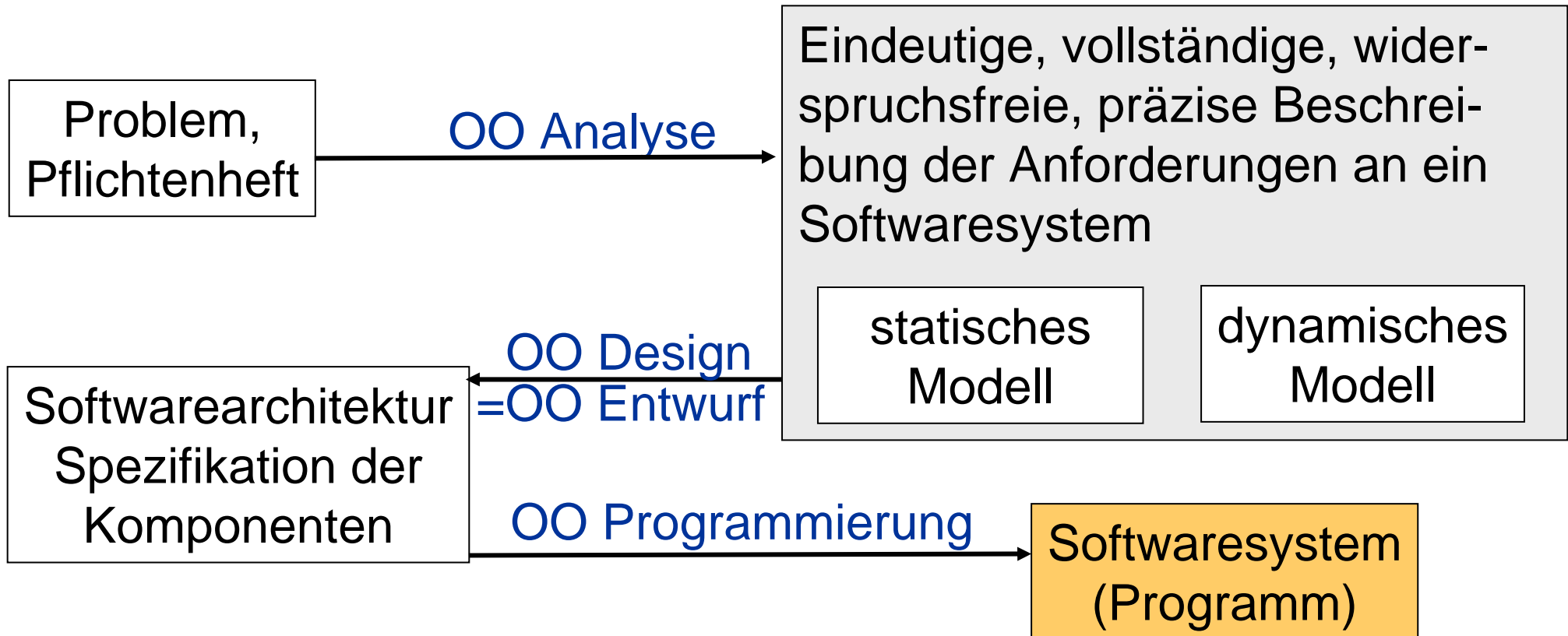
Beim Entwurf und der Planung eines Softwaresystems steht der Systemanalytiker an der Schnittstelle zwischen Auftraggeber und Entwickler:

- Systemanalytiker muss vor allem den Auftraggeber verstehen (und weniger umgekehrt).
- Beschreibung der Auftraggeberwünsche sollte unabhängig von technischen Möglichkeiten sein (perfekter Rechner).
- Auftraggeber sollte seinen Auftrag im eingesetzten objektorientierten Modell erkennen.

Gesucht wird folglich eine Repräsentation, mittels derer sich beide verständigen können.

- Sollte unabhängig von der zu verwendenden Programmiersprache und Entwicklungsumgebung sein.
- Also: Darstellung als **Grafik**.

Überblick

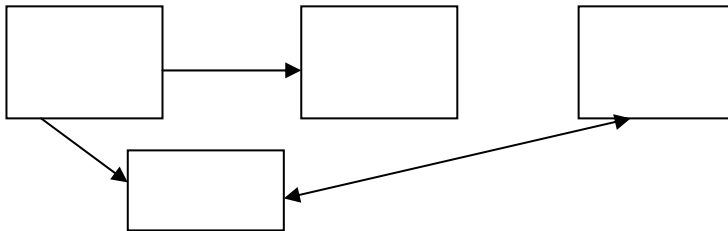


- Analyse, Design und Programmierung gibt es auch ohne Objektorientierung
- Objektorientierung
 - gleiche Denkweisen in allen Entwicklungsphasen

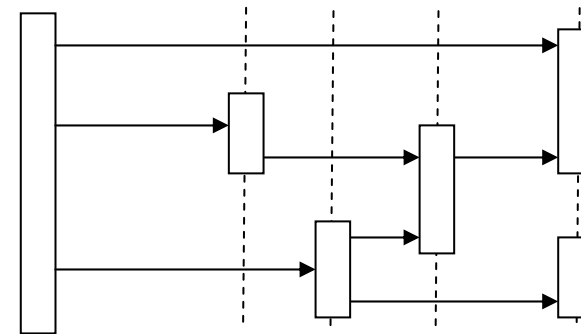
Duale Sichtweise auf die Elemente eines Systems:

- Objekt als Gegenstand der Anschauung
- Objekt als Dienstgeber und Dienstnehmer

Statisches Modell



Dynamisches Modell



Nur grob:

- | | |
|----------------------|------------------------------|
| ■ Statisches Modell | ~ Gegenstand der Anschauung |
| ■ Dynamisches Modell | ~ Dienstgeber / Dienstnehmer |

Typische Szene:

- Kunde braucht Softwaresystem zur Lösung bestimmter Probleme.
- Softwareentwickler hört zu und versucht Modell zu entwickeln.

Situation birgt die Gefahr, dass

- Sachverhalte (Klassen, Objekte, Methoden, Attribute) übersehen und falsch eingeschätzt werden
- Zusammenhänge (Relationen zwischen Klassen / Objekten) nicht erkannt werden
- unnötige Zusammenhänge gesehen werden, unnütze Klassendefinitionen u.ä. gemacht werden

Deshalb wünschenswert

Methodisches Vorgehen, Checklisten

Urahnen:

- ER-Analyse (1975), semantische Netze (KL-ONE, 1975)

Ende 80er, Anfang 90er: erste Bücher über Objektorientierte Analyse

- mehrere verschiedene grafische Notationen
- zunächst nur wenige Anwender

Von Booch (1991, 1994), Rumbaugh (1991) und andere

- Bestrebungen, Standards einzuführen;
- Werkzeuge zur Unterstützung der Analyse werden entwickelt.

1994:

- Booch und Rumbaugh einigen sich auf Unified Method.

1996:

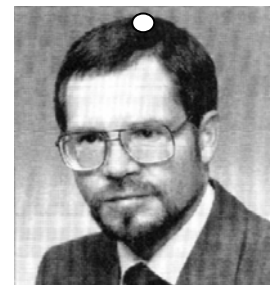
- Booch, Rumbaugh, Jacobson veröffentlichen die Unified Modeling Language (UML).



G. Booch

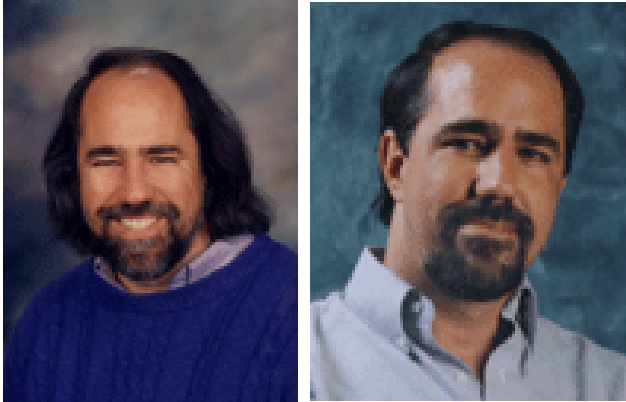


I. Jacobson



J. Rumbaugh

Grady Booch



- Chief Scientist bei IBM Rational Software Corporation
- Master an der UCSB
- Bachelor an der United States Air Force Academy (1977)

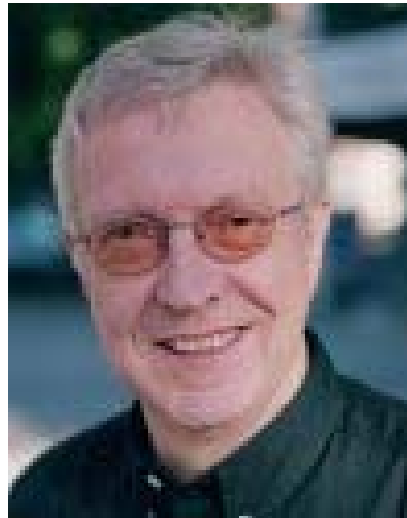
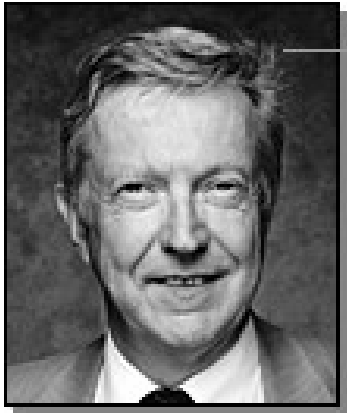


<http://www.wikipedia.org>

„I built my first computer when I was twelve in Amarillo, Texas. At the time I was into hardware because you couldn't really program things. No courses or books existed, and, as a twelve-year-old boy, I couldn't ask my peers for help. So, I pounded the doors at the local IBM sales office until a salesman took pity on me. After we chatted for a while, he handed me a Fortran [manual]. I'm sure he gave it to me thinking, "I'll never hear from this kid again." I returned the following week saying, "This is really cool. I've read the whole thing and have written a small program. Where can I find a computer?" The fellow, to my delight, found me programming time on an IBM 1130 on weekends and late-evening hours. That was my first programming experience, and I must thank that anonymous IBM salesman for launching my career. Thank you, IBM.“



Ivar Jacobson



- Vize-Präsident von IBM Rational Software Corporation
- Gastwissenschaftler am MIT
- Dr. am Royal Institute of Technology in Stockholm
- Master am Chalmers Institute of Technology in Göteborg
- Wie UML entstand und andere lustige Geschichten: <http://www.ivarjacobson.com>

Chat with Cyber Ivar



Cyber Ivar beantwortet Fragen zu UML unter

<http://www.jaczone.com/cyberivar/>

J. Rumbaugh



Die 3 Amigos (v.l.n.r: Booch, Jacobson, Rumbaugh)...

- IBM Rational Software Corporation
- Einer der "Three Amigos"
- Dr. am MIT
- Master (Astronomie) am Caltech
- Bachelor (Physik) am MIT
- Erfand den „UML-Song“:

<http://www.ivarjacobson.com/files/Jim's%20OOPSLA%2095%20Song.htm>



<http://www.wikipedia.org>



...und das Original

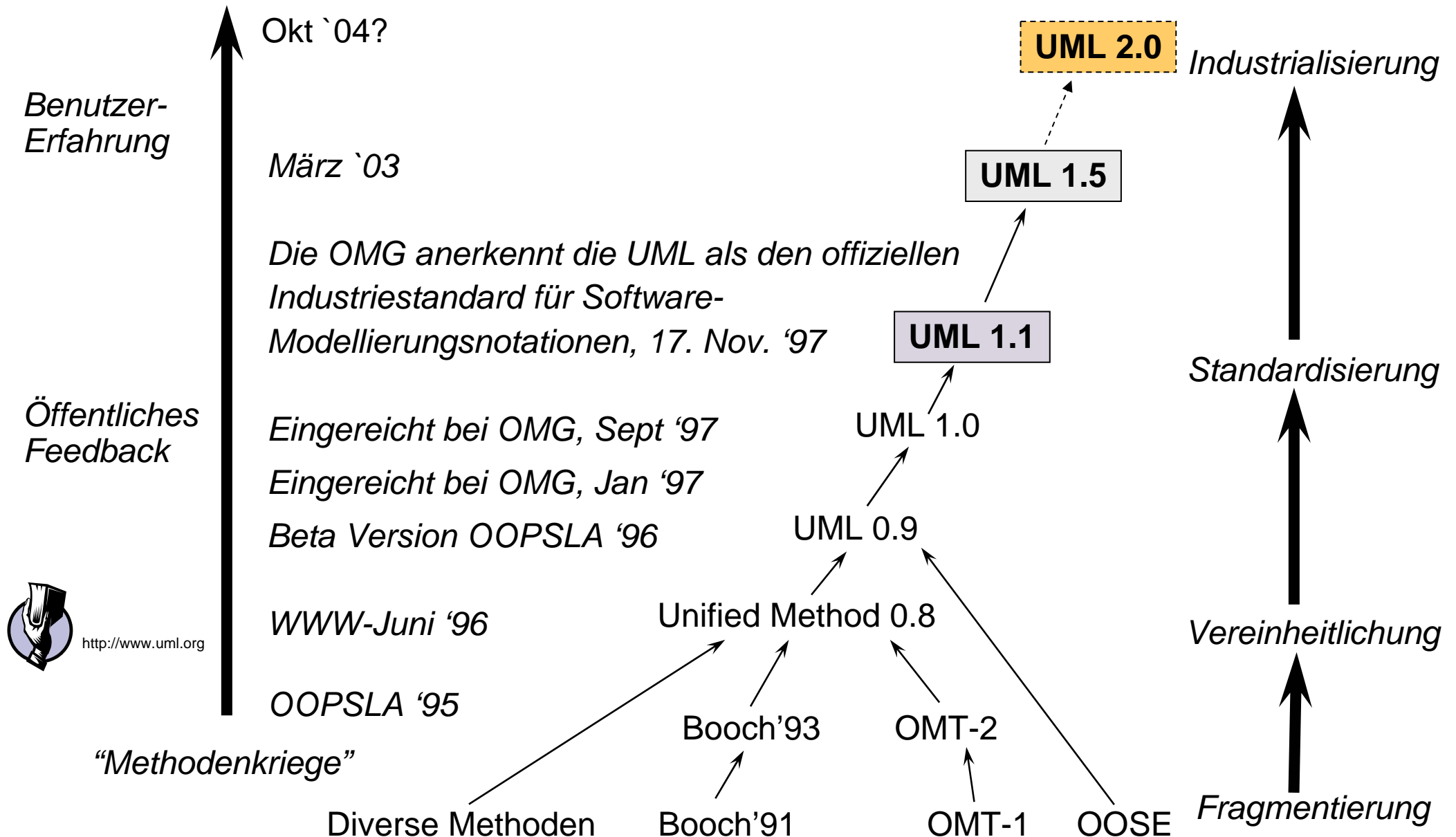
www.tm.uka.de



... als Ergänzung zu den Folien

ER-Analyse	= Entity-Relationship-Analyse
FTF	= Finalization Task Force
KL-ONE	= Knowledge Language One
KZK	= Klasse-Zugehörigkeit-Kooperation
OMA	= Object Management Architecture
OMG	= Object Management Group
OMT	= Object Modeling Technique
OO	= Object Oriented
OOPSLA	= Object Oriented Programming, Systems, Languages and Applications
OOSE	= Object Oriented Software Engineering
UML	= Unified Modeling Language
WWW	= World Wide Web

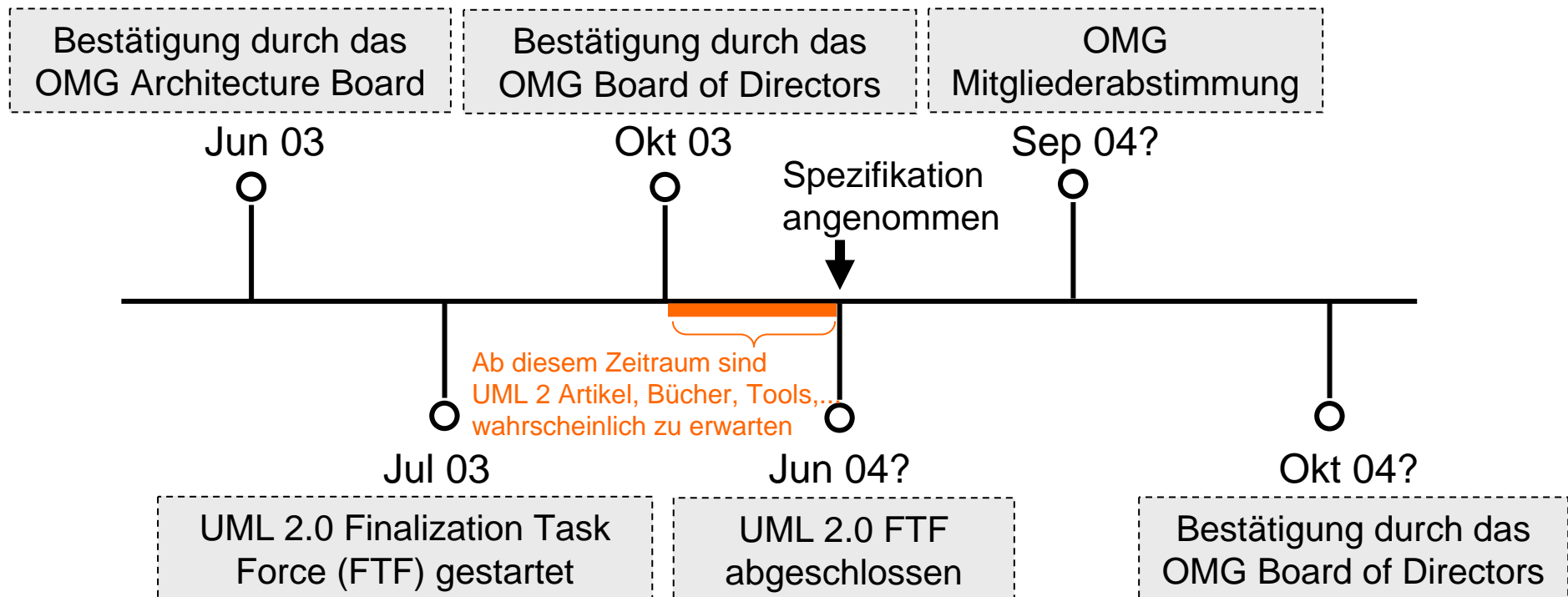




UML2

- Im Moment im Standardisierungs- und Abgleichungsprozess.

Zeitplan:



„UML 2 glasklar“, Mario Jeckle et al,
Hanser Fachbuchverlag, 2003



Die Unified Modeling Language (UML) ist eine grafische Sprache zur Visualisierung, Spezifikation, Konstruktion und Dokumentation der Bestandteile von Softwaresystemen.

Dieser Standard bietet ein breites Spektrum von Modellierungsmöglichkeiten, die von der konzeptuellen Ebene der Geschäftsprozesse bis zur konkreten Umsetzung einzelner Softwarekomponenten reichen.

Folgen der grafischen Sprache:

- Die Modellierung erfolgt in Form von Diagrammen.
- Sie entzieht sich einer rigorosen formalen Definition.
- Beschreibung erfolgt daher verbal und durch Beispiele.

- Keine Methode
- Keinen Entwicklungsprozess
- Kein Modellierungswerkzeug
- Keine Modellierungsrichtlinien
- Keine Programmiersprache



Struktur- diagramm

(1) **Anwendungsfalldiagramm**
(*use case diagram*)

Geschäftsvorfälle

(2) **Klassendiagramm**
(*class diagram*)

Gegenstand, statisch

(3) **Sequenzdiagramm**
(*sequence diagram*)

Dienste, dynamisch

(4) **Kollaborationsdiagramm**
(*collaboration diagram*)

**Interaktions-
diagramme**

(*interaction
diagrams*)

(5) **Zustandsdiagramm**
(*statechart diagram*)

Dienste, statisch

(6) **Aktivitätsdiagramm**
(*activity diagram*)

Gegenstand, dynamisch

Implementierungs- diagramme

(*implementation diagrams*)

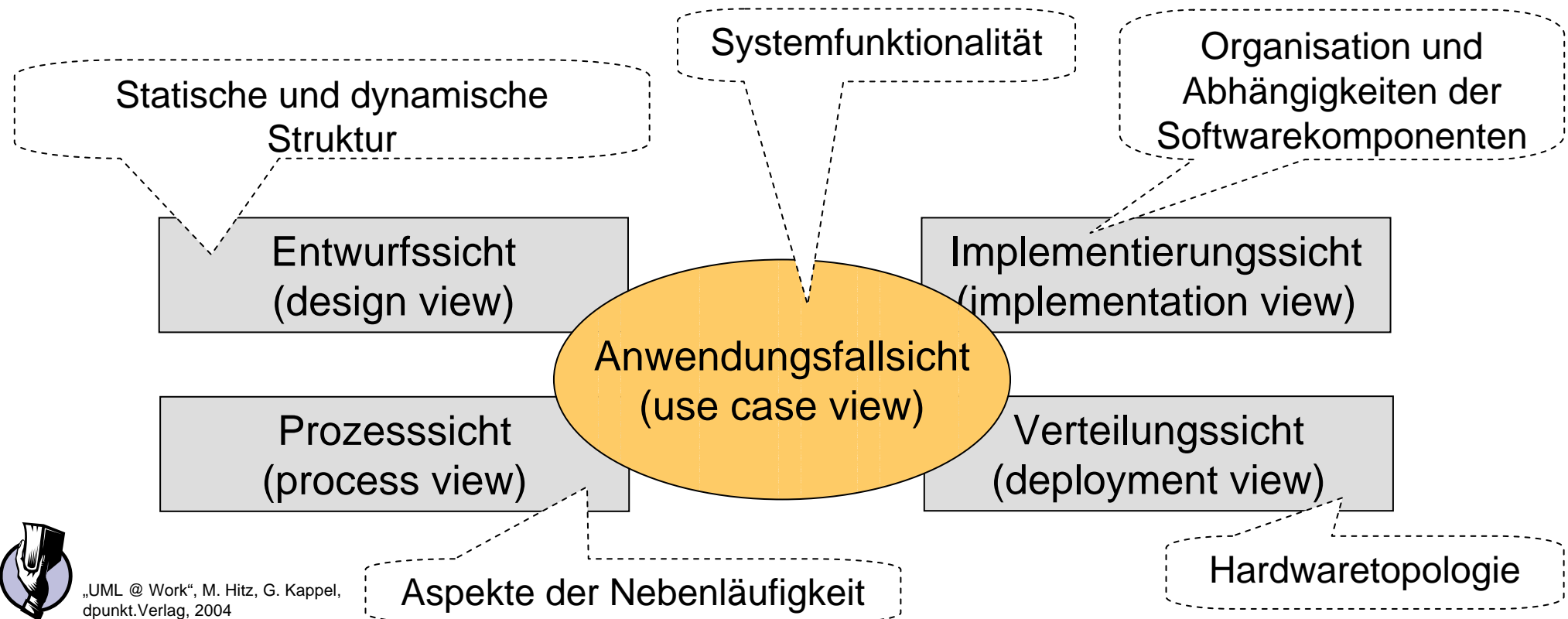
(7) **Komponentendiagramm**
(*component diagram*)

(8) **Verteilungsdiagramm**
(*deployment diagram*)

UML bietet mehrere Möglichkeiten die Diagramme zu Modellen eines Softwaresystems zu gruppieren.

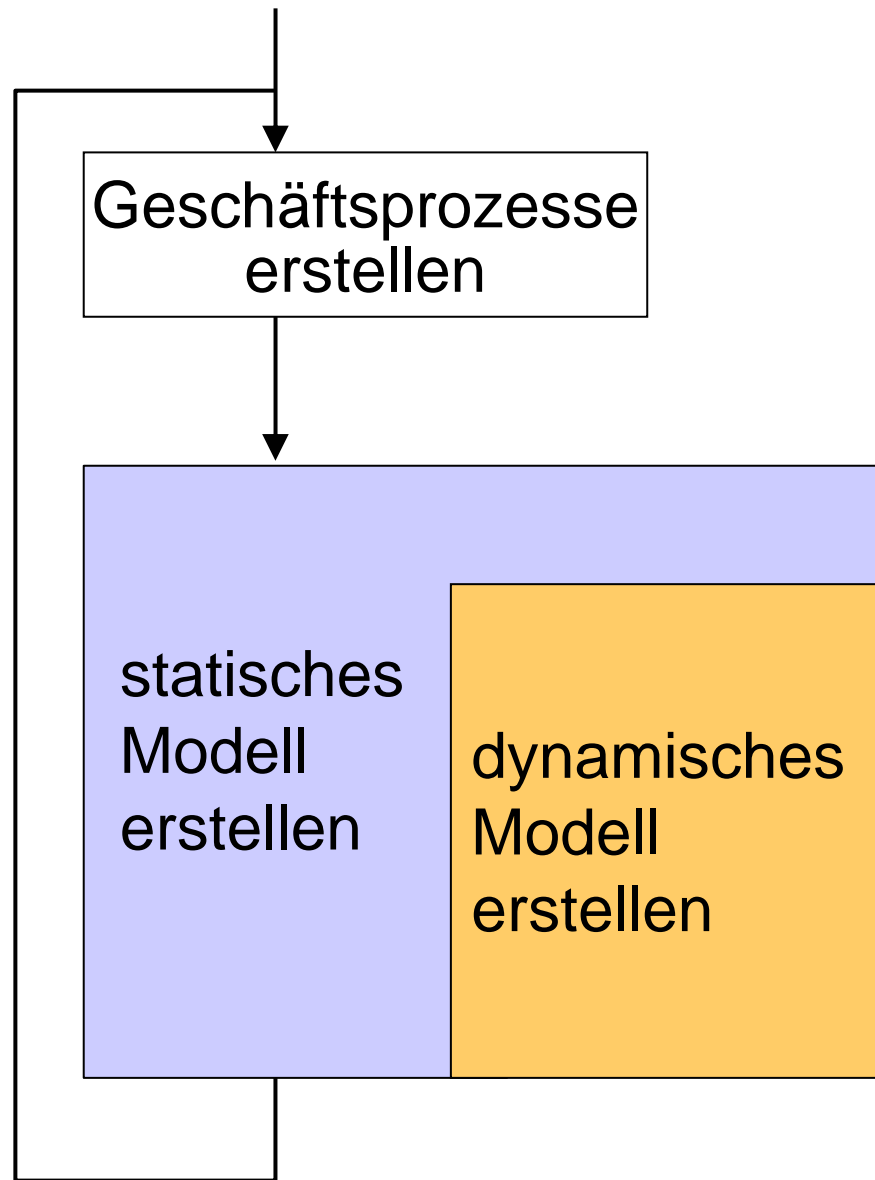
- Schwerpunkte können verschieden gelegt werden in Abhängigkeit der Art des zu entwickelnden Systems.

Beschreibung in Form von überlappenden Modellen (Sichten)



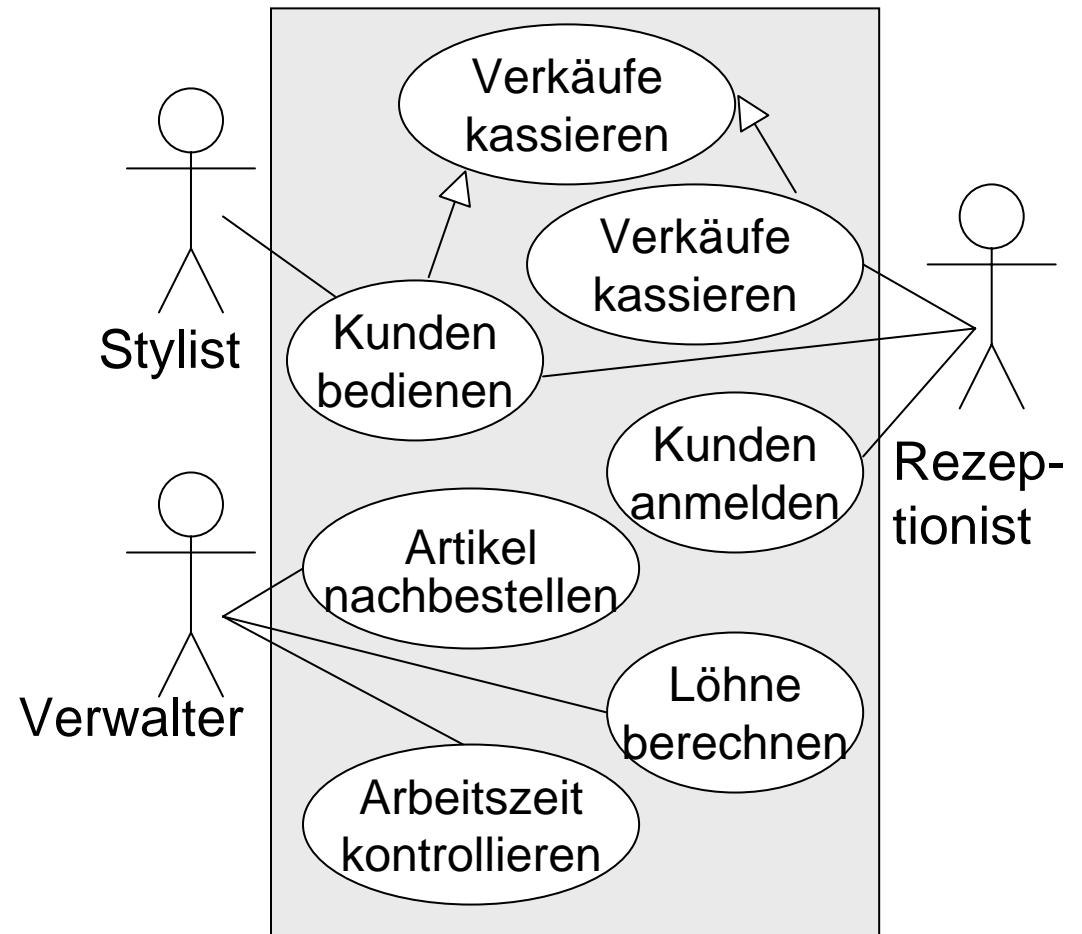
„UML @ Work“, M. Hitz, G. Kappel,
dpunkt.Verlag, 2004





Beispiel

- Geschäftsprozessdiagramm für Friseursalonverwaltung



Anwendungsfalldiagramm (use case diagram)

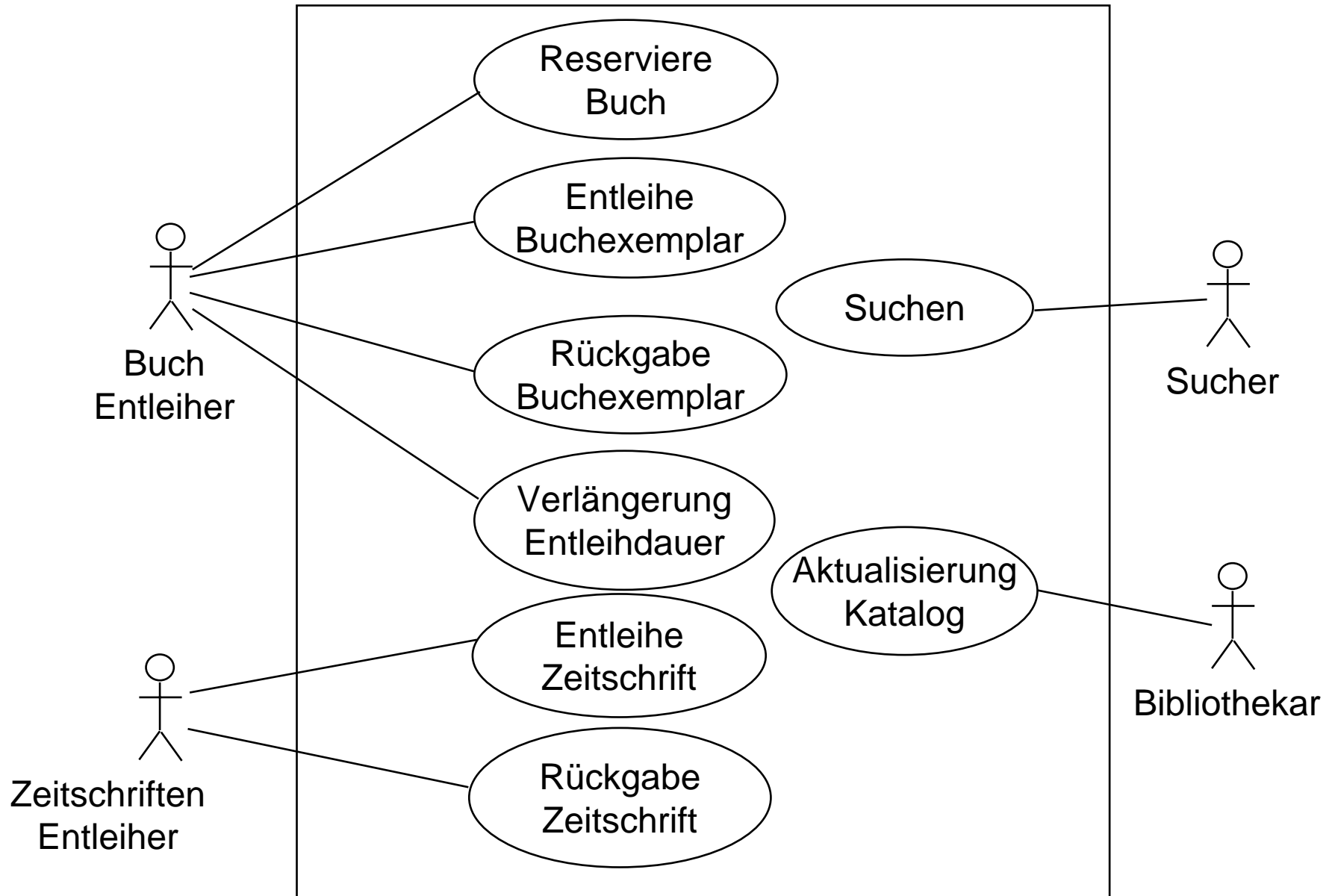
Analyse mit einem anwenderorientierten Standpunkt:

- **Akteur** (actor): Anwender (Mensch, Maschine, Programm) des zu entwerfenden Systems in einer bestimmten Rolle.
 - Tritt mit dem zu entwerfenden System in Interaktion.
 - Hat Anforderungen an dieses System.
- **Anwendungsfall** (use case): Aufgabe, die ein Akteur mit Hilfe des zu entwerfenden Systems lösen will oder muss.

Modellierungselemente eines Anwendungsfalldiagramms:

- Akteure als Strichfiguren
- Anwendungsfälle als Ovale
- Beteiligung eines Akteurs an einem Anwendungsfall als Linie

Beispiel einer Bibliothek (Anwendungsfalldiagramm)



Aus den in Umgangssprache beschriebenen Anwendungsfällen lassen sich durch Substantiv-Analyse die beteiligten Objekte bestimmen.

Aufgaben einer Bibliothek:

„In der Bibliothek gibt es Bücher und Zeitschriften. Von Büchern kann es mehrere Exemplare geben. Einige Bücher sind nur für Kurzzeitentleihe gedacht. Alle anderen Bücher können von jedem Bibliotheksmitglied für drei Wochen entliehen werden. Mitglieder der Bibliothek können normalerweise bis zu 6 Einheiten entleihen, während Mitarbeiter der Bibliothek bis zu 12 Einheiten zur gleichen Zeit entleihen können. Nur Mitarbeiter der Bibliothek dürfen Zeitschriften entleihen.“

Aus den in Umgangssprache beschriebenen Anwendungsfällen lassen sich durch Substantiv-Analyse die beteiligten Objekte bestimmen.

Aufgaben einer Bibliothek:

„In der Bibliothek gibt es Bücher und Zeitschriften. Von Büchern kann es mehrere Exemplare geben. Einige Bücher sind nur für Kurzzeitentleihe gedacht. Alle anderen Bücher können von jedem Bibliotheksmitglied für drei Wochen entliehen werden. Mitglieder der Bibliothek können normalerweise bis zu 6 Einheiten entleihen, während Mitarbeiter der Bibliothek bis zu 12 Einheiten zur gleichen Zeit entleihen können. Nur Mitarbeiter der Bibliothek dürfen Zeitschriften entleihen.“

außerhalb des Systems

Ereignis

=Bibliotheksmitglied

Zeiteinheit

zu vage

Aus den in Umgangssprache beschriebenen Anwendungsfällen lassen sich durch Substantiv-Analyse die beteiligten Objekte bestimmen.

Aufgaben einer Bibliothek:

„In der Bibliothek gibt es Bücher und Zeitschriften. Von Büchern kann es mehrere Exemplare geben. Einige Bücher sind nur für Kurzzeitentleihe gedacht. Alle anderen Bücher können von jedem Bibliotheksmitglied für drei Wochen entliehen werden. Mitglieder der Bibliothek können normalerweise bis zu 6 Einheiten entleihen, während Mitarbeiter der Bibliothek bis zu 12 Einheiten zur gleichen Zeit entleihen können. Nur Mitarbeiter der Bibliothek dürfen Zeitschriften entleihen.“

Ausgehend von den Anwendungsfällen erfolgt die Zuordnung der Funktionalitäten zu den ermittelten Objektklassen.

Entwurf nach Zuständigkeit:

- Untersuchung der Kooperationsaufgaben:
 - Jedes Objekt ist für bestimmte Aufgaben zuständig und besitzt entweder alle Fähigkeiten, um diese Aufgaben selbst zu lösen, oder es kooperiert dazu mit anderen Objekten.

Dokumentation der Ergebnisse dieses Entwurfsschritts mittels KZK-Karten (Klasse-Zuständigkeit-Kooperation):

Klassenname	
zuständig für	Zusammenarbeit mit

Bibliotheksmitglied	
zuständig für <ul style="list-style-type: none">• Verwalten der Daten über entliehene Exemplare• Bearbeiten der Anfragen nach Entleihe der Rückgabe	Zusammenarbeit mit Exemplar

Exemplar	
zuständig für <ul style="list-style-type: none">• Verwalten der Daten über ein bestimmtes Exemplar• Buch bei Entleihe oder Rückgabe informieren	Zusammenarbeit mit Buch

Buch	
zuständig für <ul style="list-style-type: none">• Verwalten der Daten über ein Buch• Wissen über Verfügbarkeit entleihbarer Exemplare	Zusammenarbeit mit

3.3 Klassendiagramm (class diagram)

Ein Klassendiagramm beschreibt die vorkommenden Objektklassen, deren innere Struktur und ihre Beziehungen untereinander.

Die strukturellen Eigenschaften einer Klasse werden durch Attribute (attribute, instance-scope attribute) angegeben.

Das Verhalten wird durch Operationen beschrieben.

Modellierungselemente in Klassendiagrammen:

- Klasse mit Struktur und Verhalten:
 - Rechteck mit drei Abschnitten für Name, Attribute und Operationen
- Beziehungen können als Assoziationen, Aggregationen, Generalisierungen auftreten
 - Kardinalitäten
 - Rollennamen und Navigationspfeile

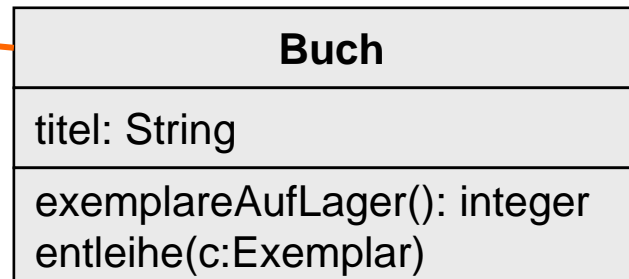
Antwort auf

- „Wie sind Daten und Verhalten meines Systems im Detail strukturiert?“

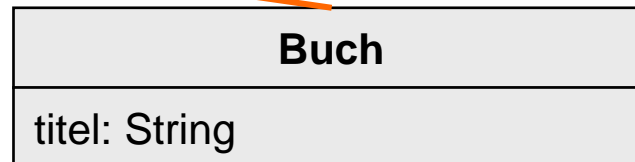


Zur besseren Übersichtlichkeit können Teile in der Darstellung ausgeblendet werden.

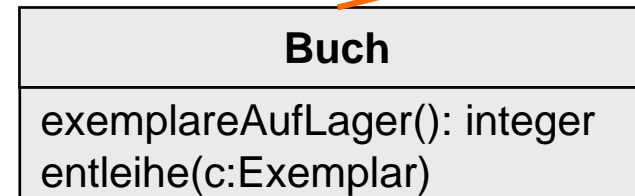
Vollständige Darstellung



Operationen ausgeblendet



Attribute ausgeblendet

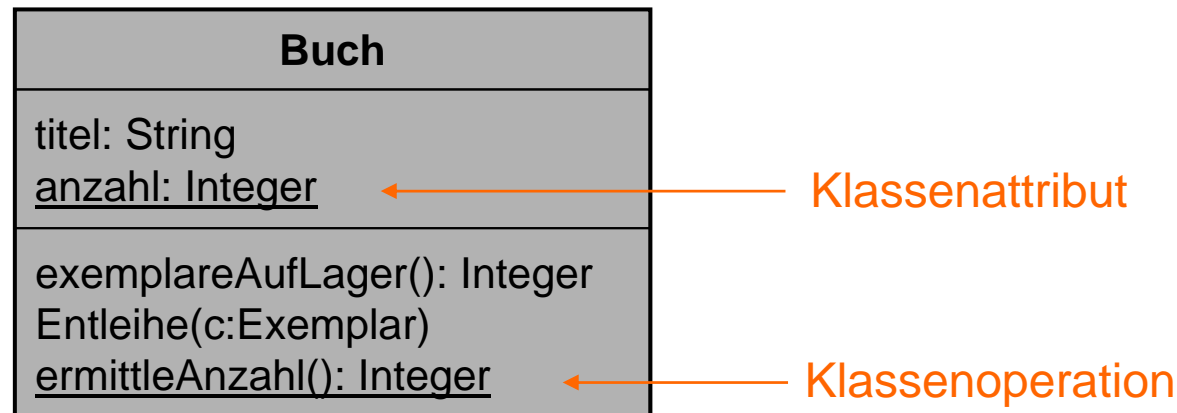


Darstellung in Minimalform



Klassenattribute (class-scope attribute) und **Klassenoperationen** (class-scope operation) sind nicht an ein einzelnes Objekt, sondern an die Klasse als Ganzes gebunden.

Sie werden unterstrichen dargestellt.



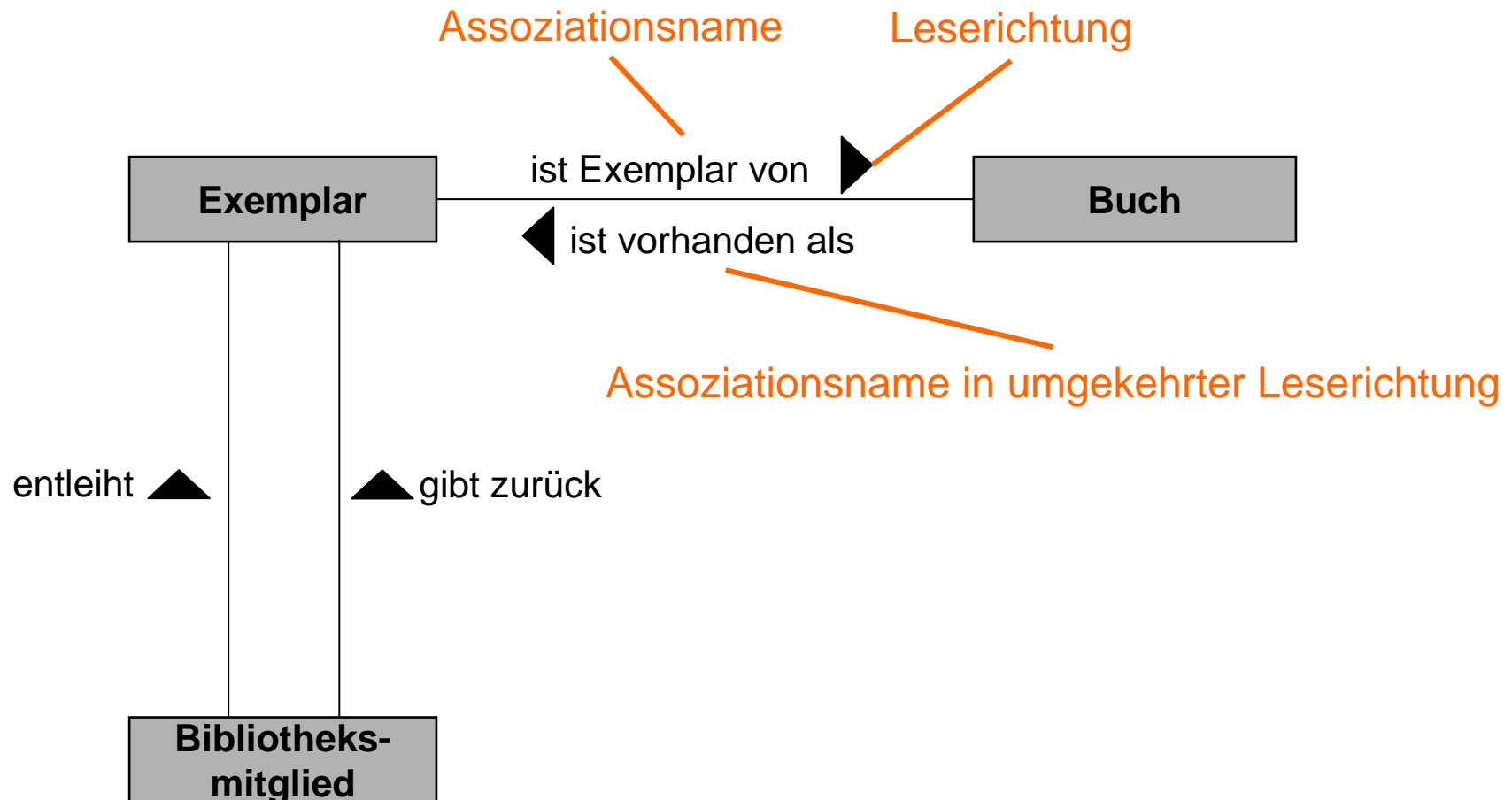
Zweistellige Assoziationen (binary association) beschreiben, wie Objekte von genau zwei Klassen untereinander in Beziehung stehen.

Eigenschaften von Assoziationen:

- **Kardinalität:**
 - Wie viele Objekte der Quell-/Zielklasse stehen mindestens/höchstens miteinander in Beziehung
- **Name:**
 - Welche Bedeutung hat die Assoziation
- **Rolle:**
 - Welche Bedeutung hat die Klasse hinsichtlich der Assoziation
- **Richtung:**
 - bidirektional: In beide Richtungen navigierbar
 - unidirektional: Nur in eine Richtung navigierbar
- **Restriktionen:**
 - Einzuhaltende Bedingungen

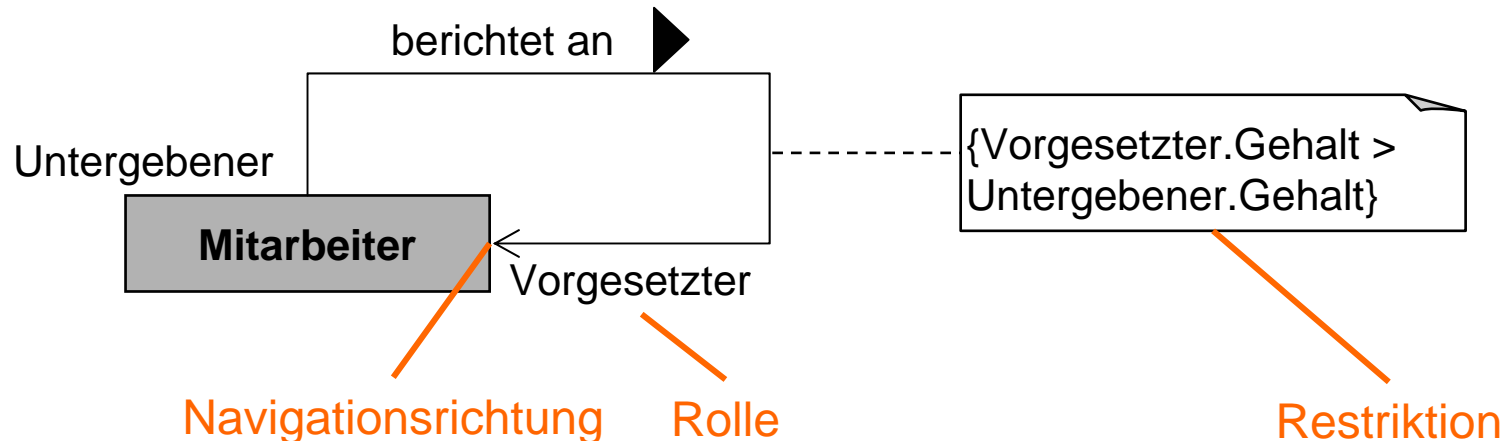
Notation für zweistellige Beziehungen:

- Linie zwischen den betreffenden Klassen
- Optional aber empfehlenswert:
 - Assoziation gemäß Leserichtung benennen und mit Pfeil neben Namen anzeigen



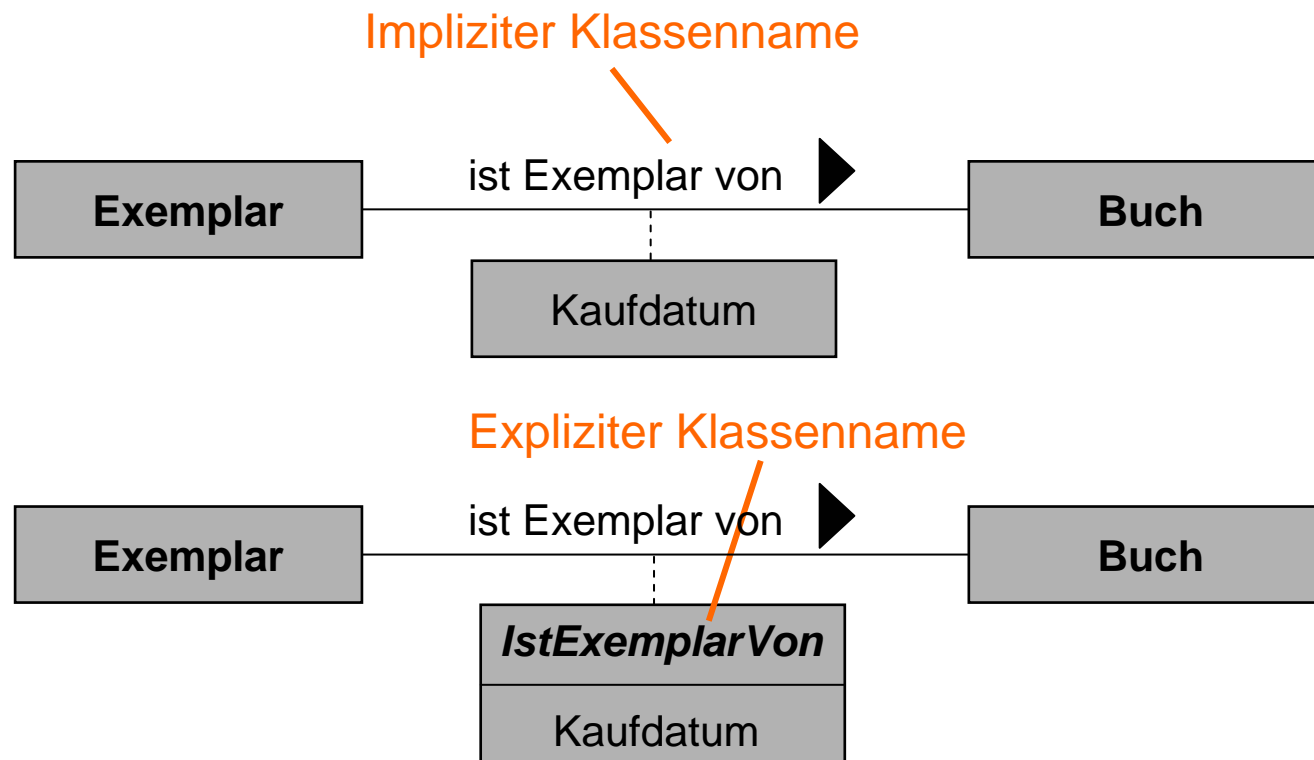
Notation:

- Pfeil an entsprechendem Linienende gibt Navigationsrichtung an
- Anschreiben der Rolle als Sichtweise eines Objektes durch das gegenüberliegende Objekt.
 - Natürliche Formulierung in Leserichtung.
 - Benötigt vor allem bei reflexiver Assoziation.
- Restriktionen werden in { ... } angegeben.



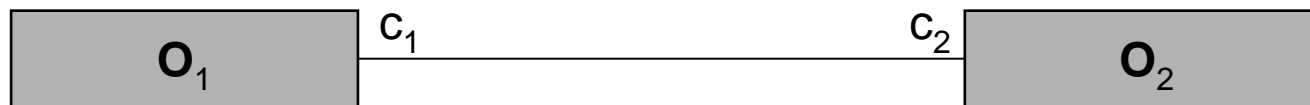
Mit einer **Assoziationsklasse** (association class) werden Klasseneigenschaften an eine Assoziation vergeben:

- Assoziationseigene Attribute und Operationen
- Teilnahme der Assoziation an Assoziationen

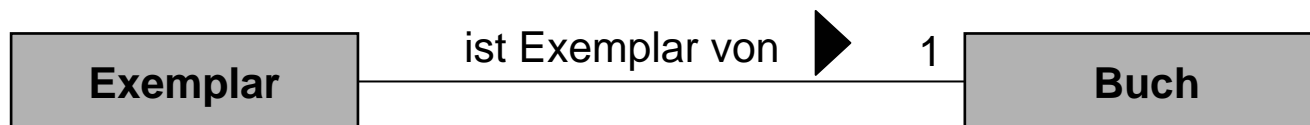


Kardinalitäten (multiplicity) geben an, wie viele Objekte der an der Assoziation beteiligten Klassen jeweils miteinander in Beziehung stehen.

Bei zweistelligen Beziehungen:



- Betrachte Assoziation zwischen Objektklassen O_1 und O_2 , wobei jedes Ende der Assoziation eine Kardinalität hat. Kardinalität c_2 drückt für jeden Zeitpunkt t die Anzahl jener O_2 -Objekte aus, die mit genau einem O_1 -Objekt in Beziehung stehen müssen/dürfen. Entsprechendes gilt analog für Kardinalität c_1 .
- Anschaulich:
 - Man lese die Kardinalitäten in Leserichtung.



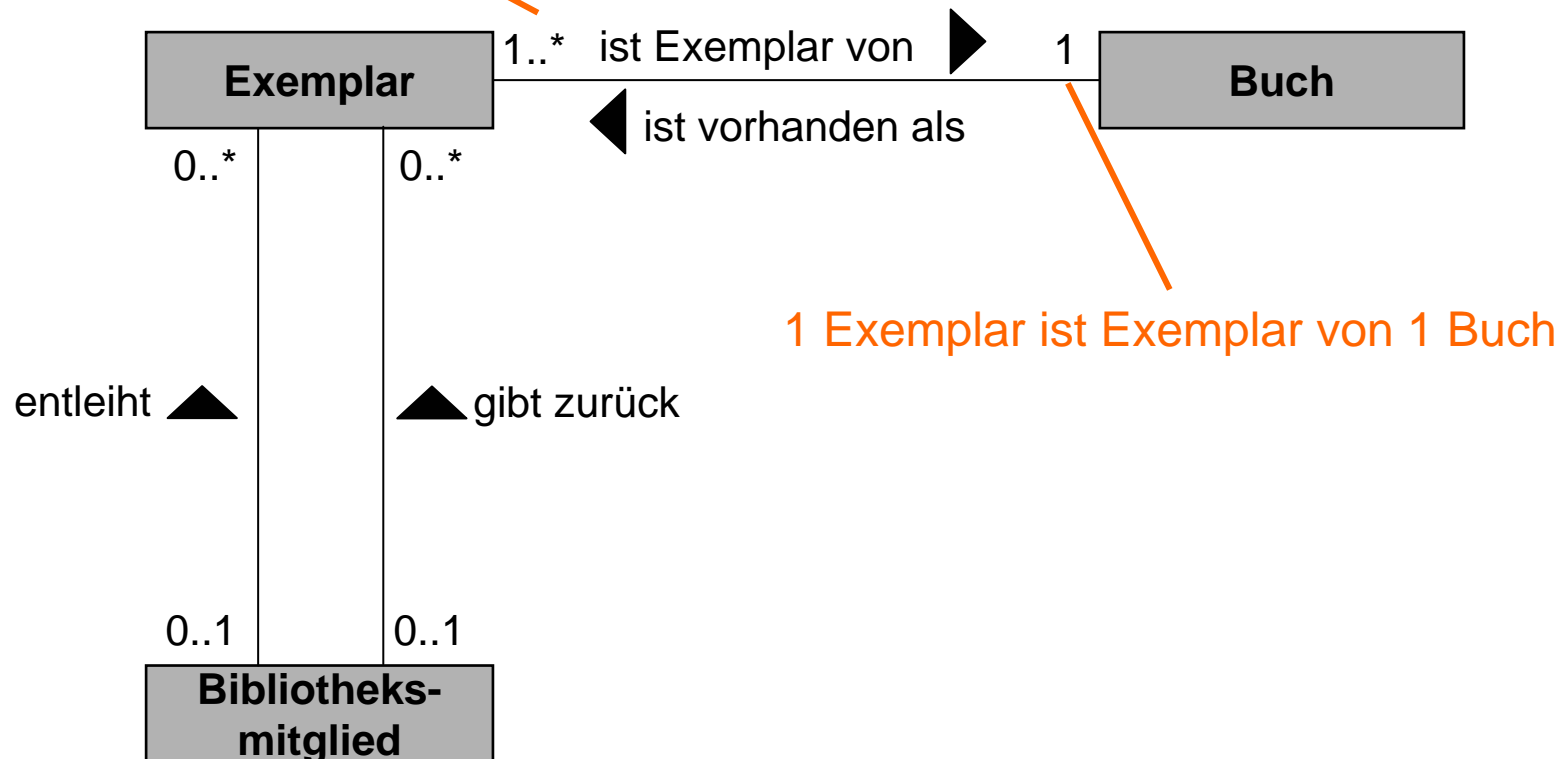
Es gibt keine Standardkardinalität, d.h. wenn keine angegeben ist, ist sie undefiniert.

Notation im Klassendiagramm:

- beliebige Anzahl wird gekennzeichnet durch „*“.
- ein Bereich wird spezifiziert durch „..“.
- Aufzählung möglicher Kardinalitäten durch Kommas getrennt.

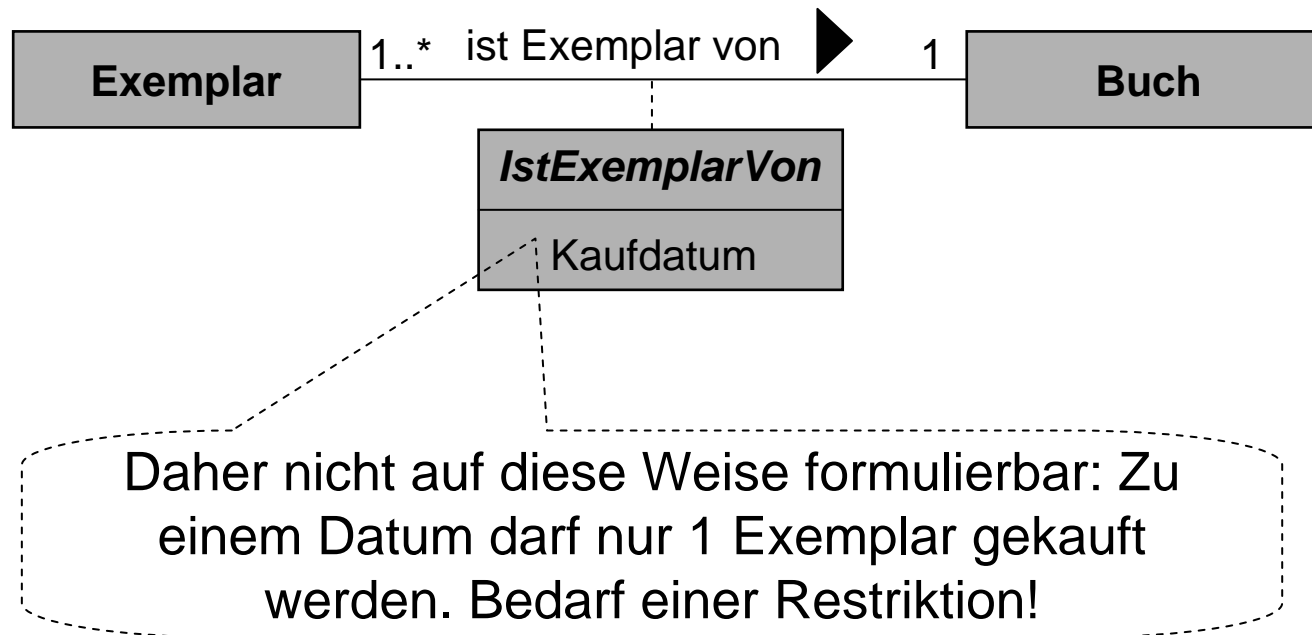
fixe Anzahl (z.B. 1):	1	
fixe Anzahl (z.B. 3):	3	
Bereich (z.B. 0 oder mehr):	*	(oder 0..*)
Bereich (z.B. 3 oder mehr):	3..*	
Bereich (z.B. 3-6):	3..6	
Aufzählung (z.B. 0 oder 1):	0,1	(oder 0..1)
Aufzählung (z.B. 3,6,7,8,9):	3, 6..9	

1 Buch ist vorhanden als 1 oder mehr Exemplare



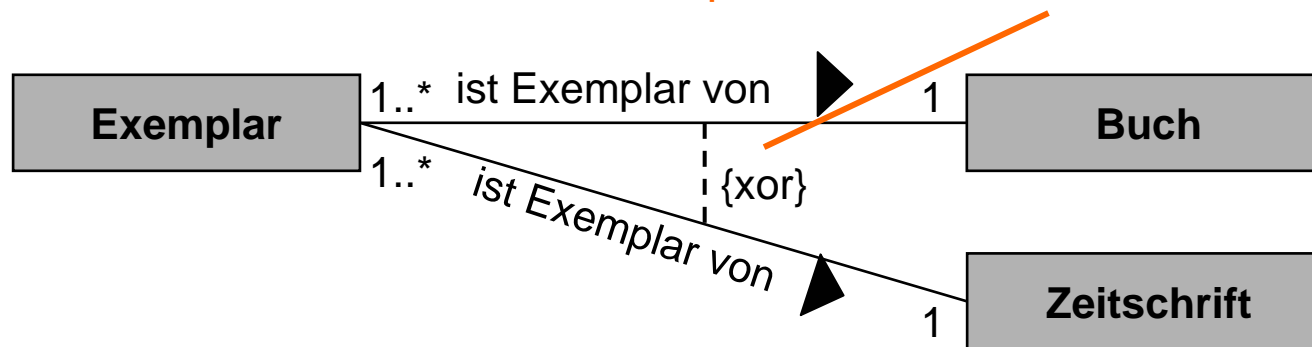
Kardinalitäten sind unabhängig davon, ob Assoziation als Klasse vereinbart wurde.

- Attribute der Assoziation haben daher keinen Einfluss.

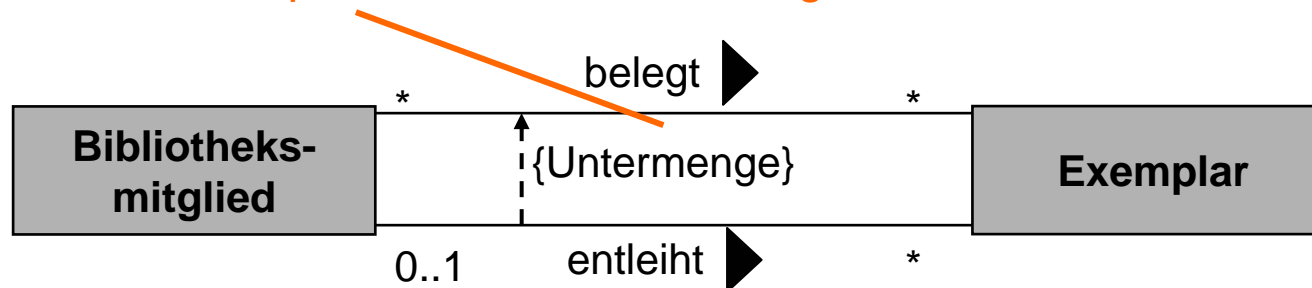


Es ist auch möglich, Assoziationen durch Restriktionen miteinander in Beziehung zu setzen:

Exemplar ist entweder Buch oder Zeitschrift

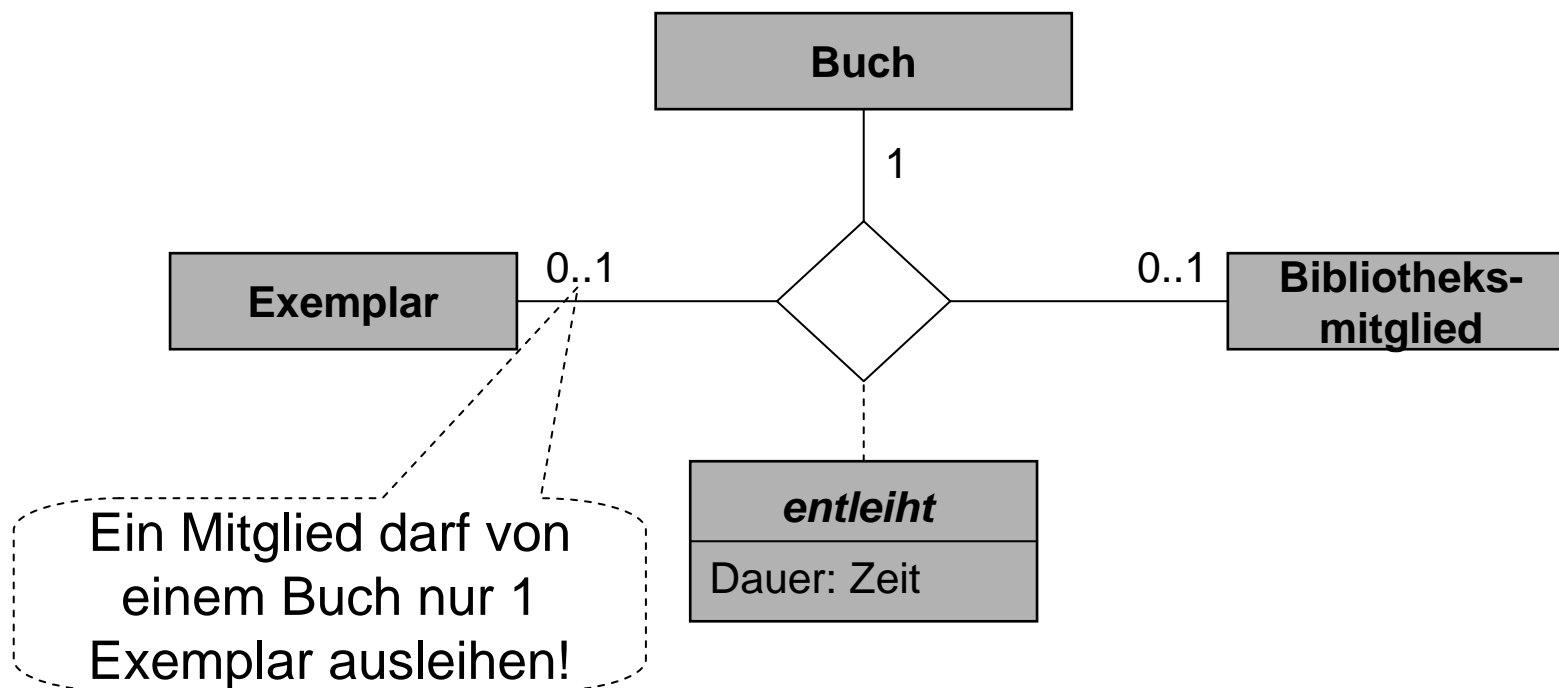


Entleihendes Exemplar ist immer auch belegt



Mehrstellige Assoziationen (n-ary association) setzen Objekte von mehr als zwei Klassen zueinander in Beziehung.

- Sie werden stets als Assoziationsklasse dargestellt.
- Kardinalität ist jetzt die Anzahl von Objekten einer Klasse, mit denen eine fixe Kombination von Objekten der übrigen Klassen eine Beziehung eingehen kann.



Eine **Aggregation** (aggregation) ist eine spezielle Form einer Assoziation, die eine ist-Teil-von-Beziehung zwischen einem Ganzen und dessen Bestandteilen ausgedrückt. Sie wird durch eine leere Raute beim Ganzen am Ende der Assoziation dargestellt.

Eine **Komposition** (composition, composite aggregation) ist eine Aggregation, bei der ein Bestandteil genau zu einem Ganzen gehört und nicht ohne das Ganze existieren kann. Dargestellt wird die Komposition durch eine gefüllte Raute.



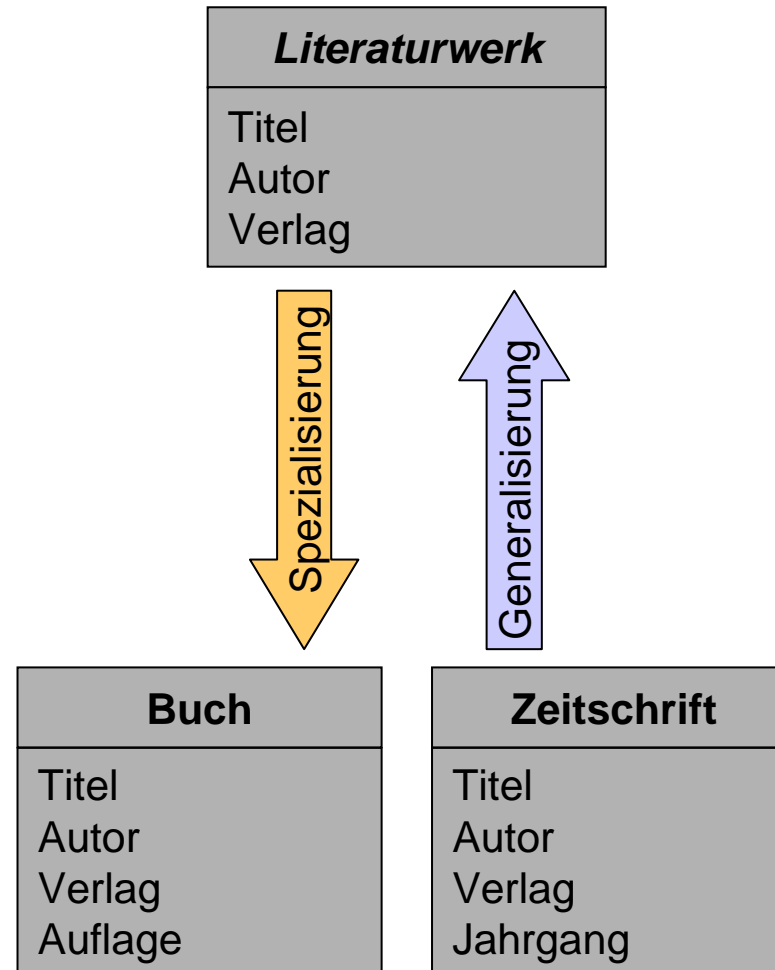
Generalisierung (generalization) ist eine taxonomische Beziehung zwischen einer spezialisierten Klasse (Unterklasse) und einer allgemeineren Klasse (Oberklasse), die Gemeinsamkeiten von Klassen in einer gemeinsamen Oberklasse zusammenfasst.

Generalisierung führt zu einer Vererbungsbeziehung (inheritance): Die Unterklasse hat alle Eigenschaften der Oberklasse und kann zusätzlich eigene haben.

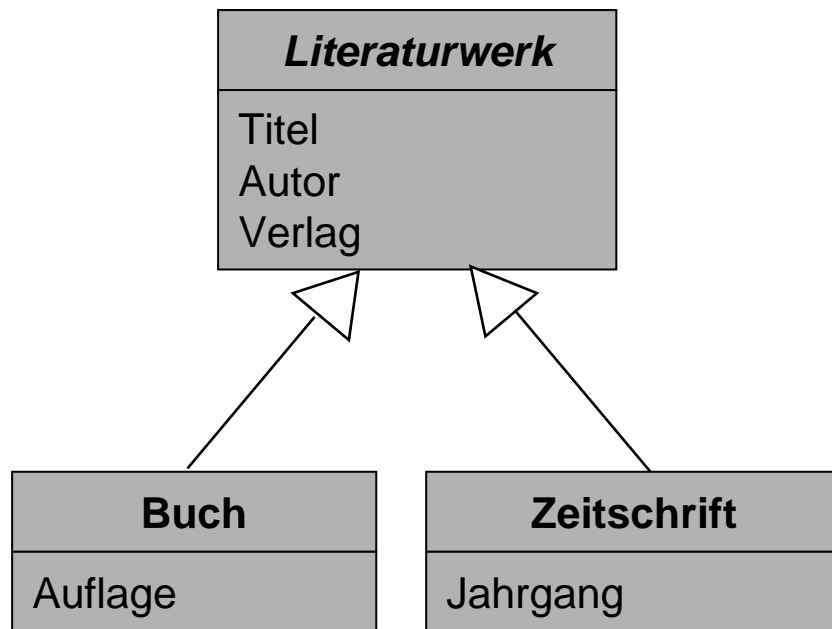
Ein Objekt der Unterklasse kann an jeder Stelle verwendet werden, an der ein Objekt der Oberklasse verwendet werden darf.

Mehrfachvererbung (mehrere Oberklassen) ist in UML erlaubt.

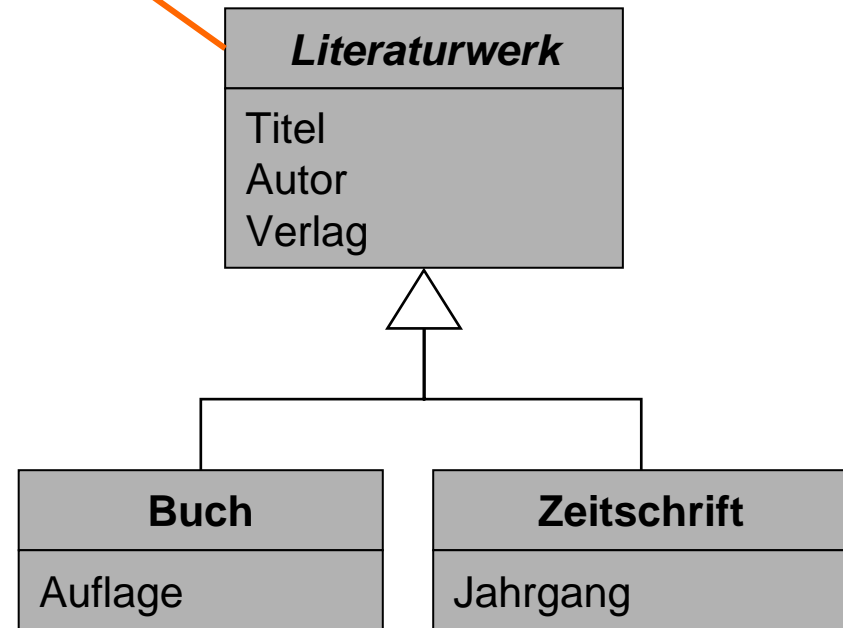
Kann es von der Oberklasse selbst keine Objekte geben, spricht man von einer abstrakten Klasse.



Namen von abstrakten Klassen werden kursiv dargestellt



Darstellungsmöglichkeit 1



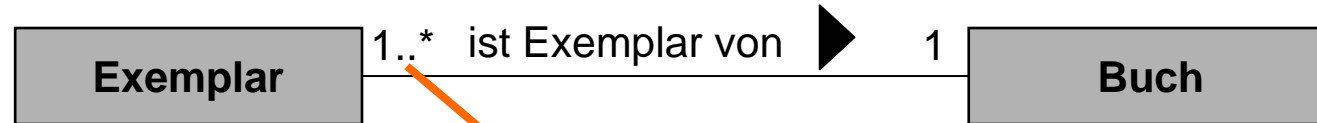
Darstellungsmöglichkeit 2

Ein **Objektdiagramm** stellt eine Menge von Objekten, deren Struktur und deren Beziehungen untereinander zu einem bestimmten Zeitpunkt dar.

Objektverbindungen (link) repräsentieren die Ausprägungen von Assoziationen.

Darstellungselement im Objektdiagramm:

- Objekte sind Rechtecke mit Namensfeld und Attributfeld
 - Das Namensfeld enthält den optionalen Namen des dargestellten Objekts und hiervon durch ":" getrennt dessen Objektklasse. Beides wird unterstrichen.
 - Die Darstellung der Attribute kann ausgeblendet werden.
- Objektverbindungen sind Linien zwischen den Objekten



Klassendiagramm

Objektdiagramm

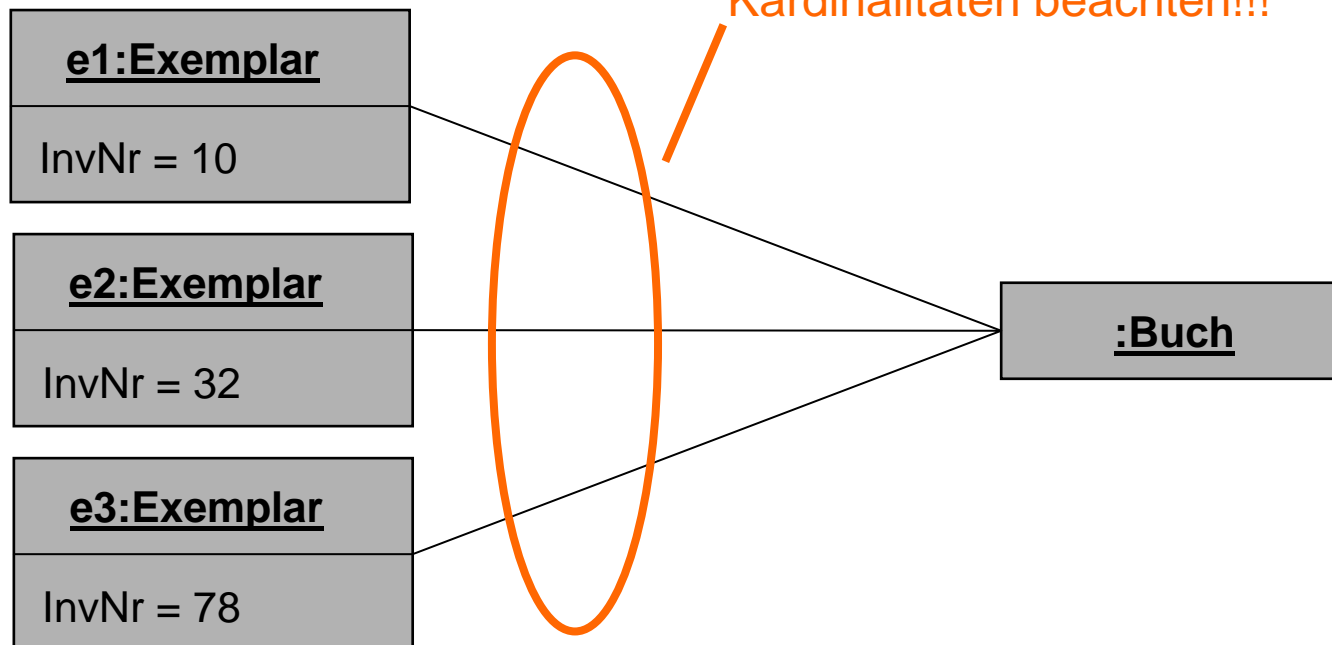


Schaubild von Objekten und Klassen ist nur eine statische Beschreibung \Rightarrow **statisches Modell**

- Geschäftsprozesse und Abläufe können so nicht dargestellt werden.

Zur vollständigen Systembeschreibung gehören noch Szenarien (\Rightarrow **dynamisches Modell**)

- Kollaborationsdiagramm
 - Schwerpunkt auf dem Zusammenwirken von Elementen ohne Berücksichtigung der zeitlichen Abhängigkeiten
- Sequenzdiagramm
 - Zum Hervorheben des dynamischen Verhaltens und der zeitlichen Abläufe des Zusammenwirkens

Szenarien beziehen sich stets auf Objekte

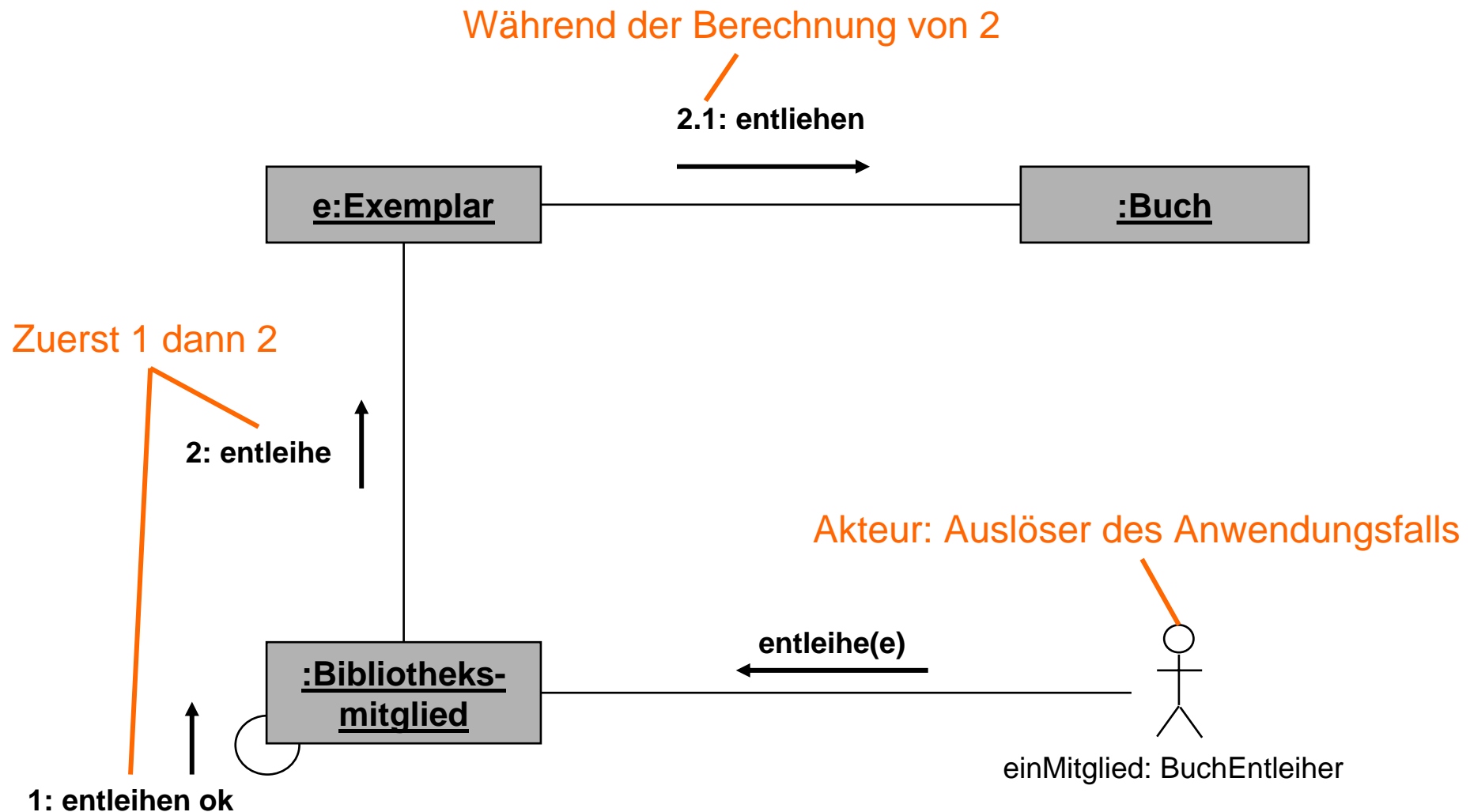
Eine **Kollaboration** beschreibt die Gesamtheit der Objekte, die bei der Ausführung einer bestimmten Aufgabe (eines konkreten Anwendungsfalles) interagieren, einschließlich der Verbindungen zwischen ihnen.

Eine Folge von Nachrichten zwischen verbundenen Objekten beschreibt eine **Interaktion** (interaction).

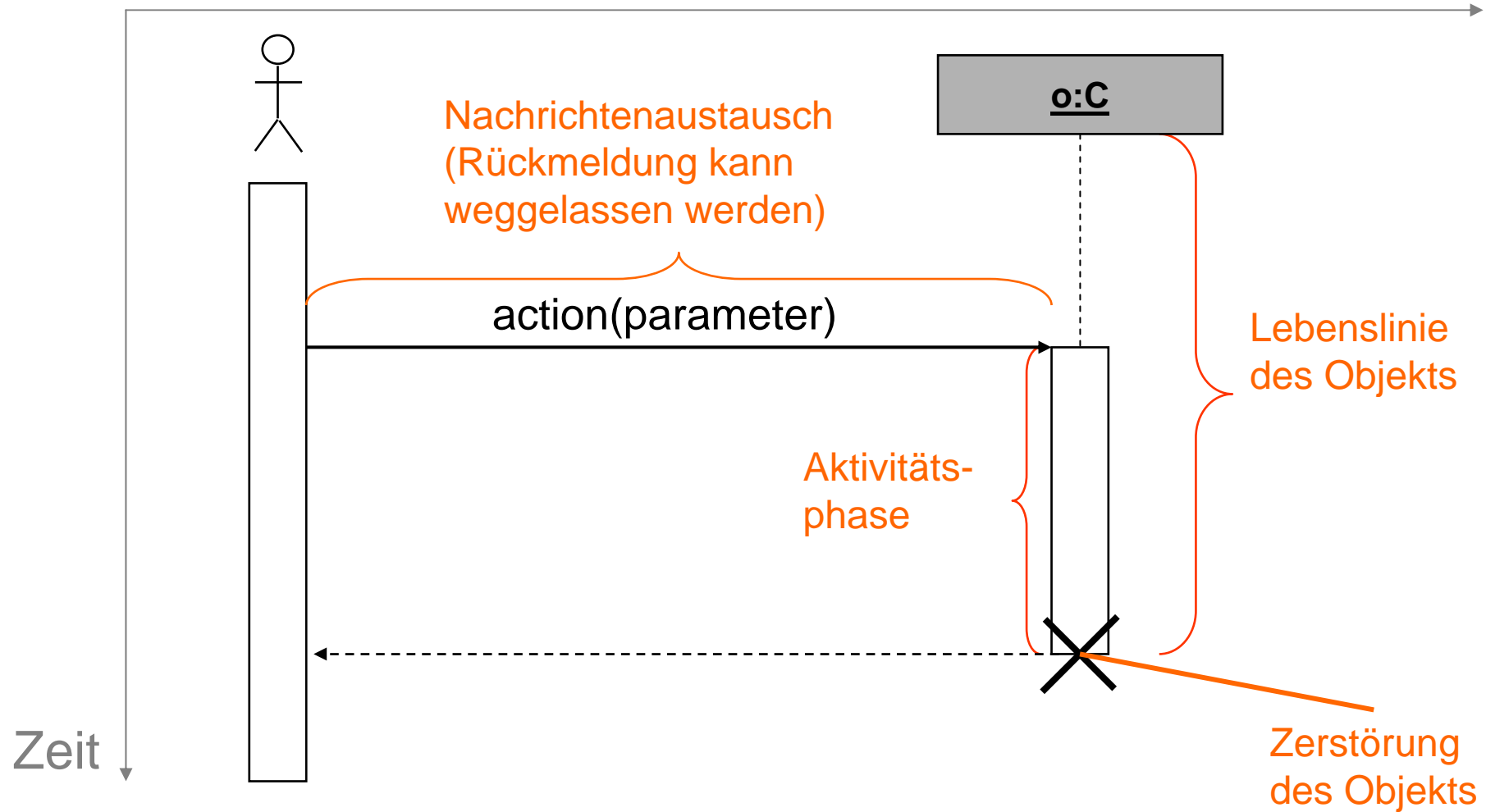
Notation:

- Objekte werden wie im Objektdiagramm dargestellt
- Linien zwischen Objekten zeigen den Austausch von Nachrichten an, wobei ein Name die Nachricht und eventuelle Parameter beschreibt und ein Pfeil die Richtung des Austausches angibt.
- Eine Nummerierung der Nachrichten zeigt deren Abfolge an.






Beispiel: Entleihen eines Exemplars



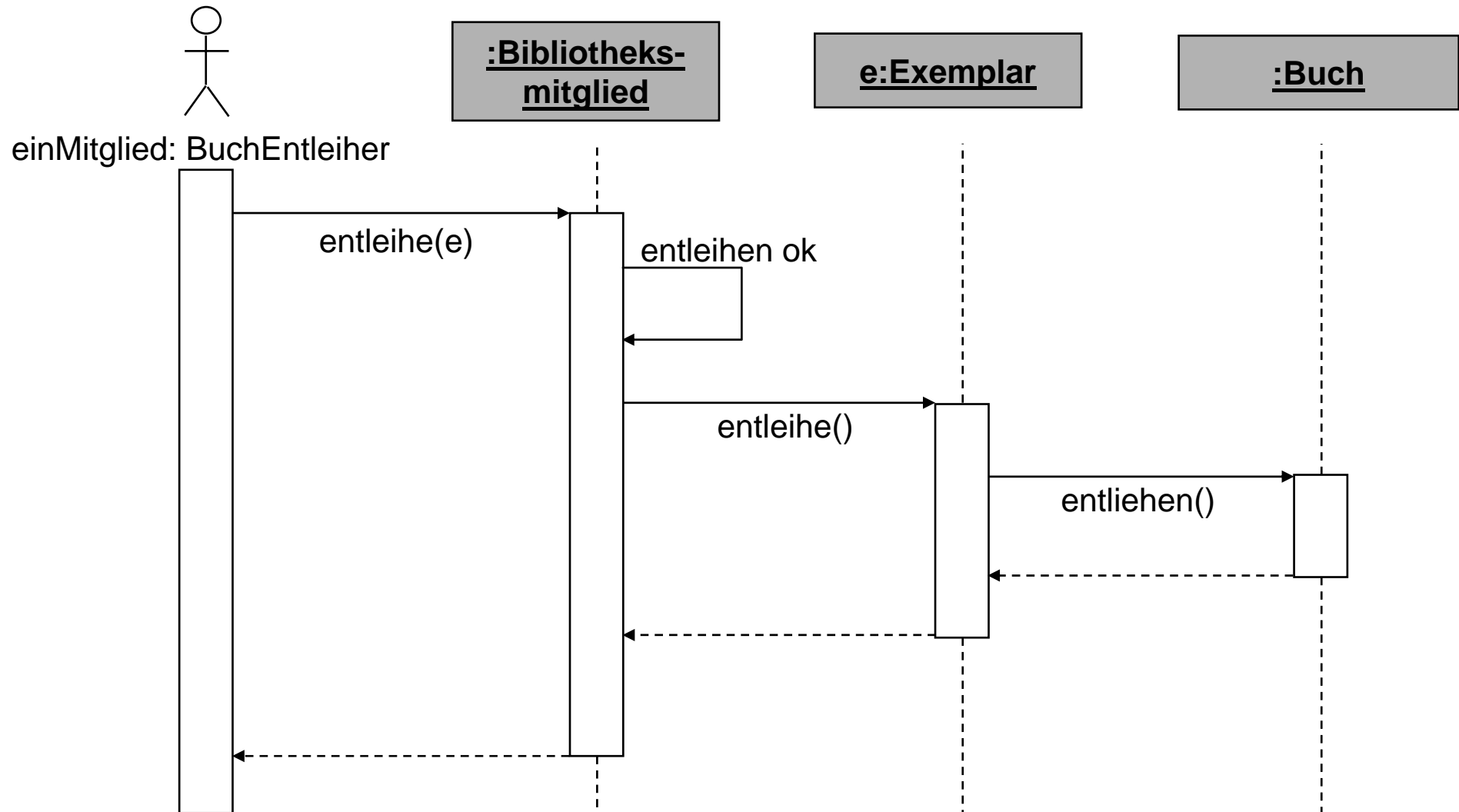
Instanzen



Pfeile geben den Typ des Kontrollflusses im Sequenzdiagramm an:

Offene Pfeilspitze		Synchron/asynchron nicht festgelegt
Gefüllte Pfeilspitze		Dito
Offene Pfeilspitze mit x		Synchroner Aufruf
Halbe Pfeilspitze		Asynchroner Aufruf
Gestrichelte, offener Pfeil		Rückkehr aus Methodenaufruf

Beispiel: Exemplar eines Buchs entleihen



3.6 Zustandsdiagramm (statechart diagram)

Zustandsdiagramme (genauer: Zustandsübergangsdiagramme) legen das **Intra-Objekt-Verhalten** fest.

Sie zeigen die mögliche „Lebensgeschichte“ von Objekten einer gegebenen Klasse auf:

- Zustände, in denen sich die Instanzen einer Klasse befinden können.
- Mögliche Zustandsübergänge von einem Zustand zum anderen.
- Ereignisse, die Zustandsübergänge verursachen.
- Operationen, die in Zuständen bzw. im Zuge von Übergängen ausgeführt werden.

Sie beschreiben somit das ganze oder auch nur Teile des möglichen Verhaltens von Objekten einer Klasse.



Der **Zustand** (state) eines Objektes wird beschrieben durch:

- Aktuelle Belegung seiner Attribute und Beziehungen.
- Einem Zustand wird eine Zeitdauer unterstellt.
- Im Diagramm:
 - Zustand als Abstraktion zulässiger Attributwerte und Beziehungen eines Objekts.

Ein **Ereignis** (event) ist ein Stimulus, der von einem Objekt auf ein anderes einwirkt.

- Ereignisse sind zeitlich punktuell, haben also keine messbare Dauer.

Ein **Zustandsübergang** (transition) ist ein durch ein Ereignis ausgelöster Zustandswechsel.

- Optional kann eine Überwachungsbedingung angegeben werden: Der Zustandsübergang kommt nur zustande, falls die Bedingung erfüllt ist.




Ein **Aktivität** ist die Ausführung eines Operators, der eine zeitliche Ausdehnung unterstellt wird.

- Stets mit einem Zustand verbunden. Sie beginnt bei Eintritt in den Zustand und terminiert entweder von selbst oder durch Verlassen des Zustandes.

Eine **Aktion** (action) ist eine atomare, nicht unterbrechbare Operatorausführung.

- Sie gilt als zeitlich punktuelle Ermittlung eines Funktionsergebnisses.
- Sie kann mit einem Zustand oder mit einem Ereignis verbunden sein.

Notation für Zustände:

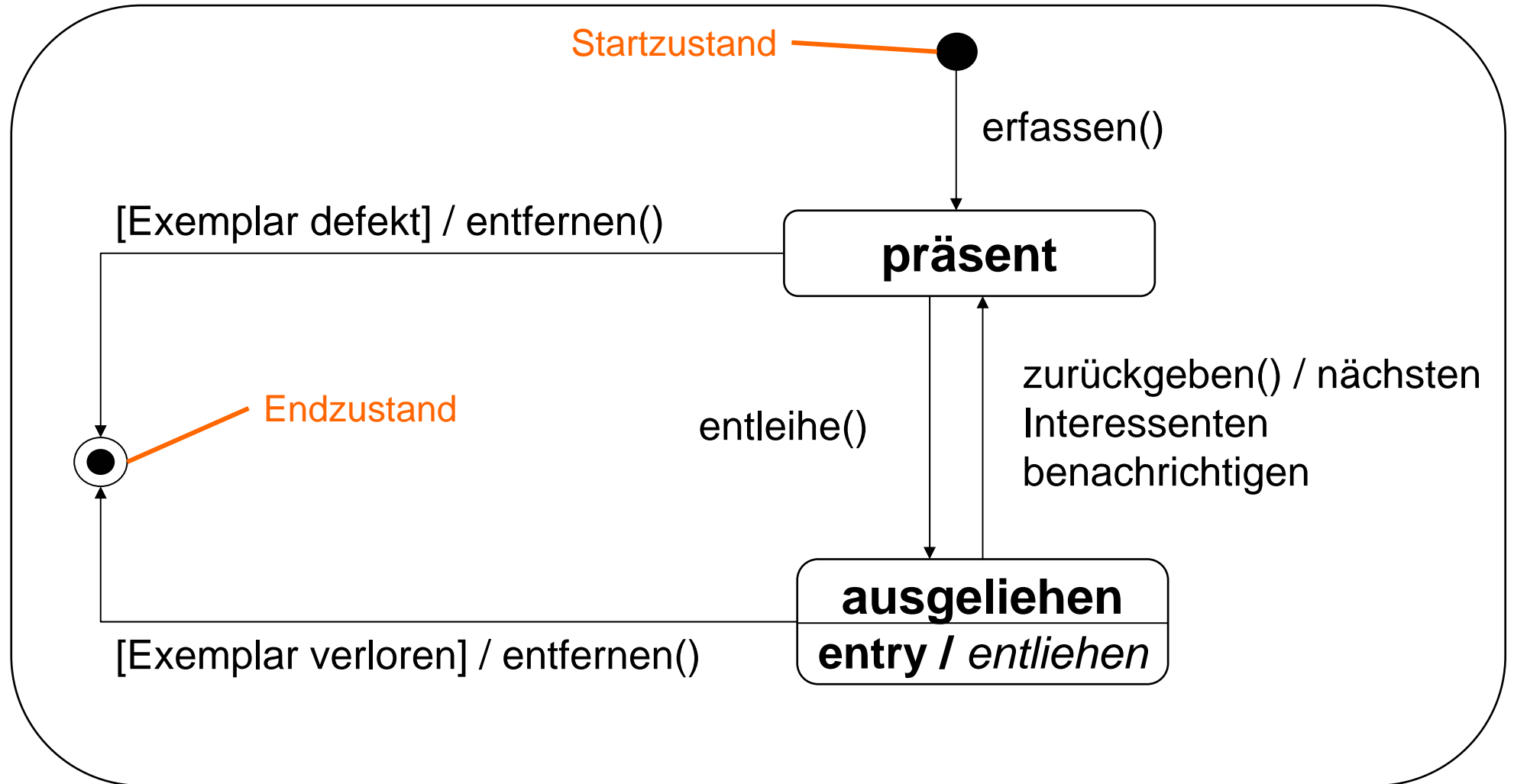
- Zustand 
- Finalzustand 
- Pseudozustände:
 - Initialzustand, Gabelung, Vereinigung 

Notation für Aktivitäten/Aktionen innerhalb eines Zustandes:

- do / aktivität Aktivität (Parameter sind erlaubt)
- entry / aktion Aktion bei Eintritt in den Zustand.
- exit / aktion Aktion bei Verlassen des Zustands.

Notation für Zustandsübergänge:

- Pfeile mit Namen des auslösenden Ereignistyps.
- Bei bedingten Übergängen folgt die Bedingung in eckigen Klammern.
- Eventuelle Aktionen folgen nach einem Schrägstrich.



Sechs Schritte zum statischen Modell

1. Klassen identifizieren
→ Klassendiagramme, Kurzbeschreibungen
2. Assoziationen identifizieren
→ Klassendiagramme vervollständigen
3. Attribute identifizieren
→ Klassendiagramme vervollständigen
4. Vererbungsstrukturen identifizieren
→ Klassendiagramme vervollständigen
5. Assoziationen vervollständigen
→ Klassendiagramme, Objektdiagramme
6. Attribute spezifizieren
→ Attributspezifikationen und Beschreibungen



Drei Schritte zum dynamischen Modell

1. Szenarios erstellen
→ Sequenzdiagramme, Kollaborationsdiagramme
2. Zustandsautomat erstellen
→ Zustandsdiagramm
3. Operationen beschreiben
→ Klassendiagramme vervollständigen,
fachliche Beschreibung der Operationen,
Aktivitätsdiagramme