Objektorientierte Modellierung

1 Objektorientierte Software-Entwicklung

Aufgaben bei der Software-Entwicklung

Modellierung

Objektorientierte Software-Entwicklung

Übersicht und Ausblick

UML

Software-Entwicklung

- Ziele der Software-Entwicklung
  - Lösung einer **Problemstellung** in einer **realen Welt**
  - Typisch: **Geschäftsprozesse** in der realen Welt sollen durch Software und Rechnersysteme unterstützt werden
    - Die Software reagiert auf Benutzer-Eingaben und verarbeitet von Benutzern eingegeben Daten und „Befehle“ (**interaktives System**)
    - Beispiel: Geschäftsprozesse in einer Bibliothek
      - Die Benutzer der Bibliothek (Leser) verwalten
      - Die Bücher und andere Medien der Bibliothek verwalten
      - Das Ausleihen, Vormerken, Zurückgeben etc. von Büchern organisieren
  - Typisch: **technische Prozesse** sollen mit Hilfe von Software und Rechnersystemen überwacht und gesteuert werden
    - Die Software reagiert auf Signale der technischen Umgebung (**reaktives System**)
    - Beispiel: Steuerung von technischen Prozessen im Fahrzeug
      - Anti-Blockier-System für Bremsen (ABS)
      - Elektronische Fensterheber

Aufgaben bei der SW-Entwicklung

Nach [Ludewig, Lichter 10]

- Analyse
  - Ziel: das **Problem** durchdringen und verstehen
  - Fragestellungen:
    - Inwieweit kann SW eingesetzt werden, um das Problem zu lösen?
    - Welche Aufgaben werden von der zu entwickelnden SW übernommen?
- Spezifikation der Anforderungen
  - Die in der Analyse festgestellten Anforderungen
    - ordnen
    - dokumentieren
    - prüfen
    - ergänzen
    - korrigieren

Aufgaben bei der SW-Entwicklung

- Architekturentwurf und Spezifikation der Module
  - Software ist nicht monolithisch, sondern besteht aus **Modulen** oder **Komponenten**, die
    - miteinander in Beziehung stehen
    - miteinander die Gesamtfunktionalität des Systems bilden
  - **Software-Architektur**: die Gesamtstruktur der Module
  - Festlegung und Beschreibung der **Schnittstellen** der einzelnen Module, so dass
    - eine unabhängige und parallele Entwicklung der einzelnen Module möglich ist
    - die fertigen Module problemlos zusammengebaut („integriert“) werden können

Aufgaben bei der SW-Entwicklung

- Codierung und Modultest
  - Implementierung der einzelnen Module
    - Codierung in einer Programmiersprache
  - Test der implementierten Module gemäß ihrer Spezifikation
  - Korrektur der beim Test entdeckten Fehler
- Integration, Test, Abnahme
  - Zusammenbau der fertigen Module (Integration)
  - Test des fertigen Systems
  - Speziell: Abnahmetest durch den Kunden

Aufgaben bei der SW-Entwicklung

- Betrieb und Wartung
  - Installation beim Auftraggeber
  - Inbetriebnahme beim Auftraggeber (inkl. Schulung des Bedienpersonals)
  - Während des Betriebs treten sehr wahrscheinlich Fehler auf
    - Diese Fehler sind zu korrigieren
  - Während des Betriebs entstehen neue Anforderungen
    - Diese neuen Anforderungen sind im System umgesetzt werden

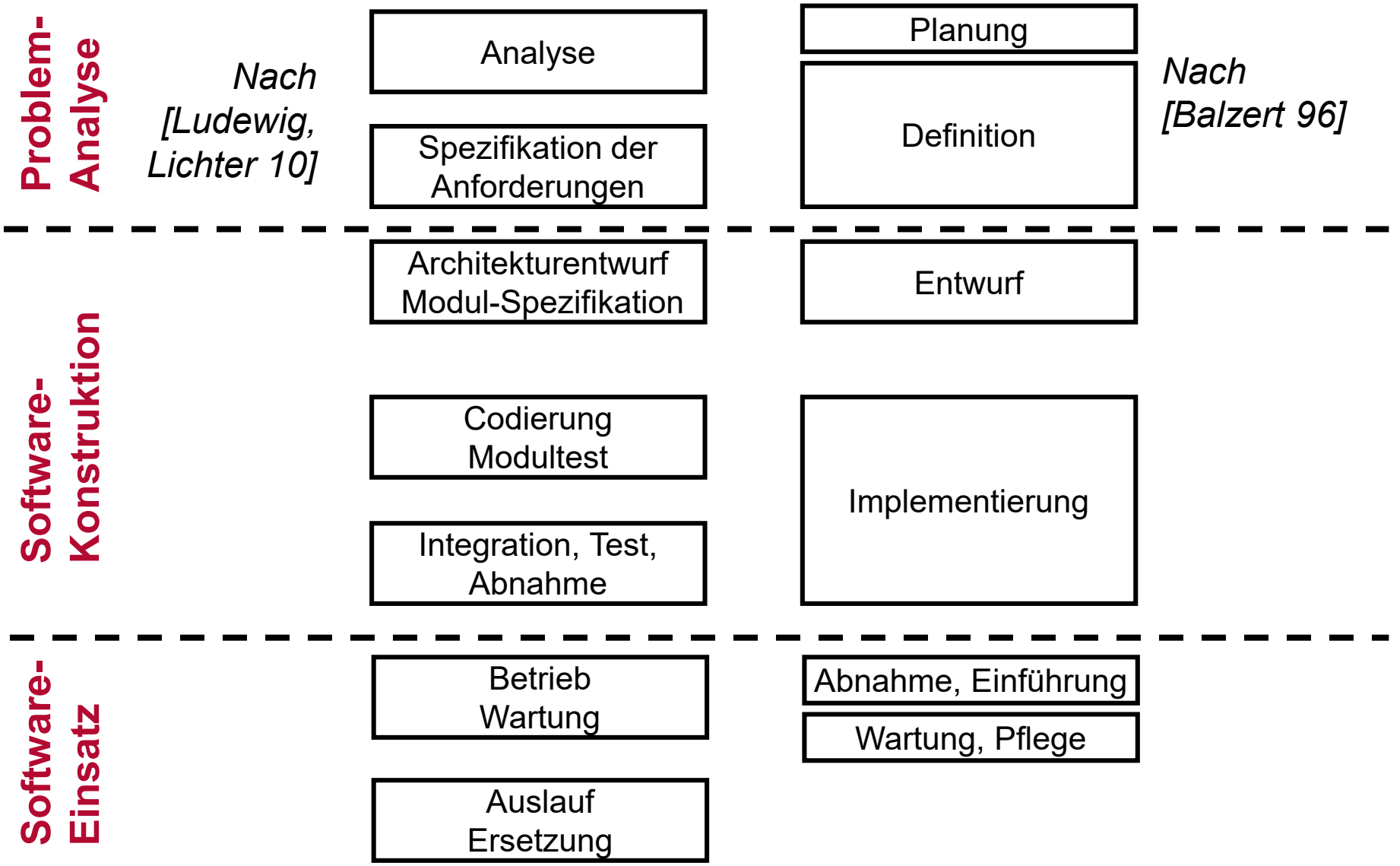
    → **Wartungsprojekte**
- Auslauf und Ersetzung
  - Außerbetriebnahme des Systems, weil
    - altes System nicht mehr wartbar
    - neue Anforderungen nicht mehr integrierbar
  - Präzise Planung erforderlich!

Aufgaben bei der SW-Entwicklung

- Phasen der SW-Entwicklung nach [Balzert 96]
  - Planung
  - Definition (Analyse)
  - Entwurf (Design)
  - Implementierung (Codierung)
  - Abnahme und Einführung
  - Wartung und Pflege

Aufgaben bei der SW-Entwicklung

- Phasen der SW-Entwicklung



Aufgaben bei der SW-Entwicklung

- **Die** zentralen Phasen der Software-Entwicklung:
  - **Analyse** (Definition, Spezifikation der Anforderungen)
  - **Entwurf** (engl.: *Desing,* Architektur- und Modul-Entwurf)
  - **Implementierung** (Codierung und Test der Module, Integration der Module)
- Tätigkeiten parallel zur gesamten Entwicklung
  - Planung und Management
  - Qualitätssicherung
- Integrierter Bestandteil aller Tätigkeiten
  - Dokumentation
- *Mehr zum Thema Software Engineering im 4. Semester!*

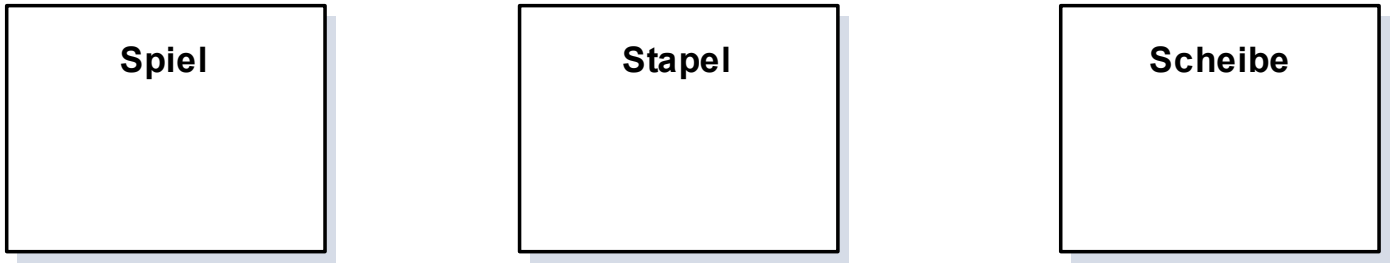
Modellierung

- In allen Phasen der Software-Entwicklung, besonders in Analyse, Entwurf und Implementierung werden **Modelle** eingesetzt
- **Modell**
  - Ein Modell ist eine **Abbildung** eines realen, fiktiven, oder geplanten Originals
    - Beispiele: Die Fotografie einer Person, der Plan eines (noch zu bauenden Hauses)
  - Ein Modell verkürzt die Merkmale des Originals
    - Es gibt Merkmale des Originals, die nicht im Modell dargestellt sind
      - Beispiel: Gewicht, Name, Blutgruppe der dargestellten Person
    - Das Modell kann Merkmale enthalten, die es nicht im Original gibt
      - Beispiel: Qualität und Format des Fotopapiers
  - Bezüglich bestimmter Fragestellungen kann das Modell das Original ersetzen
    - Ein Unfall kann aufgrund eines Fotos vom Unfallort beurteilt werden
    - Ein Fingerabdruck erlaubt die Identifizierung einer Person
  - Welche Merkmale des Originals im Modell dargestellt werden, ist durch den Einsatzzweck bestimmt

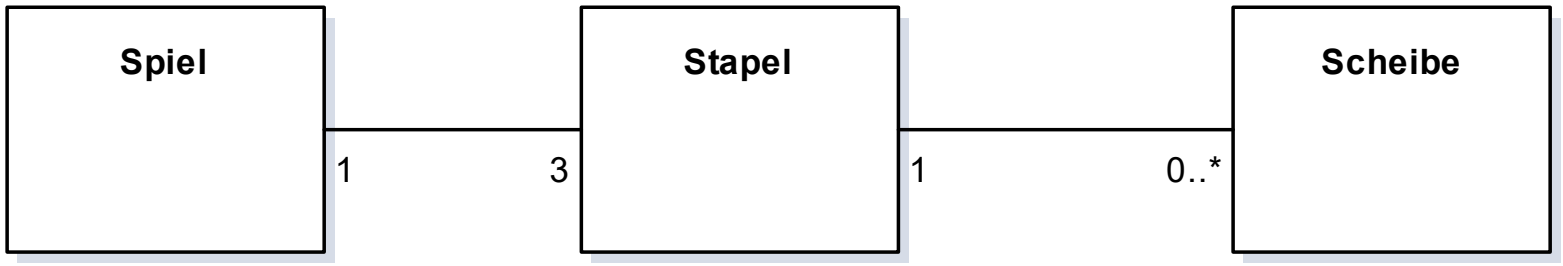
Modellierung

- Modelle in der Softwareentwicklung
  - Analysemodell
    - Das Original ist ein Ausschnitt aus der realen Welt
    - Das Modell beschreibt die Merkmale der realen Welt, die für die Problemstellung relevant sind
  - Entwurfsmodelle
    - Das Original ist das geplante Software-System
    - Das Modell beschreibt die Strukturen („Architektur“) der zukünftigen Software
  - Code
    - Der Code ist ein Modell des ausführbaren Programms auf dem Rechner
  - In den letzten 20 Jahren haben sich **objektorientierte** Software-Entwicklungsmethoden durchgesetzt und bewährt
    - **Objektorientierte Modellierung** in Analyse und Entwurf
      - Standard-Modellierungssprache: UML (*Unified Modeling Language*)
    - Implementierung der Modelle in objektorientierten Programmiersprachen
      - Java, C++,. C#,…

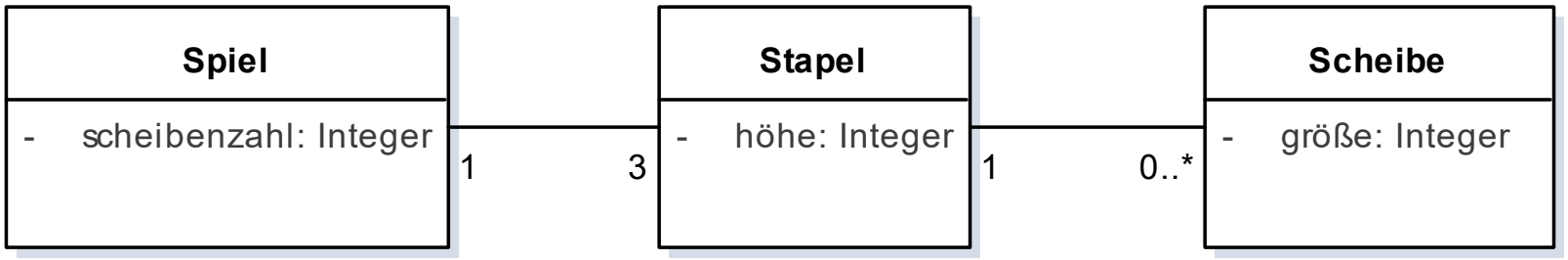
- **Objektorientierte Analyse – Statische Analyse (1)**
  - *Beispiel:* Computerspiel *Türme von Hanoi*
  - Was sind die relevanten „**Objekte**“ (konkrete Gegenstände, Konzepte, Begriffe,…) des Anwendungsbereichs?
    - Objekte: *Spiel(-brett), 3 Stapel, 5 Scheiben*
  - Fasse diese Objekte in **Klassen** zusammen
    →Objekte sind Elemente dieser Klasse
    - Klassen: *Spiel, Stapel, Scheibe*
  - Stelle die Klassen in **UML** grafisch dar
    - Verwende dazu ein geeignetes CASE-Tool (z.B. Enterprise Architect)



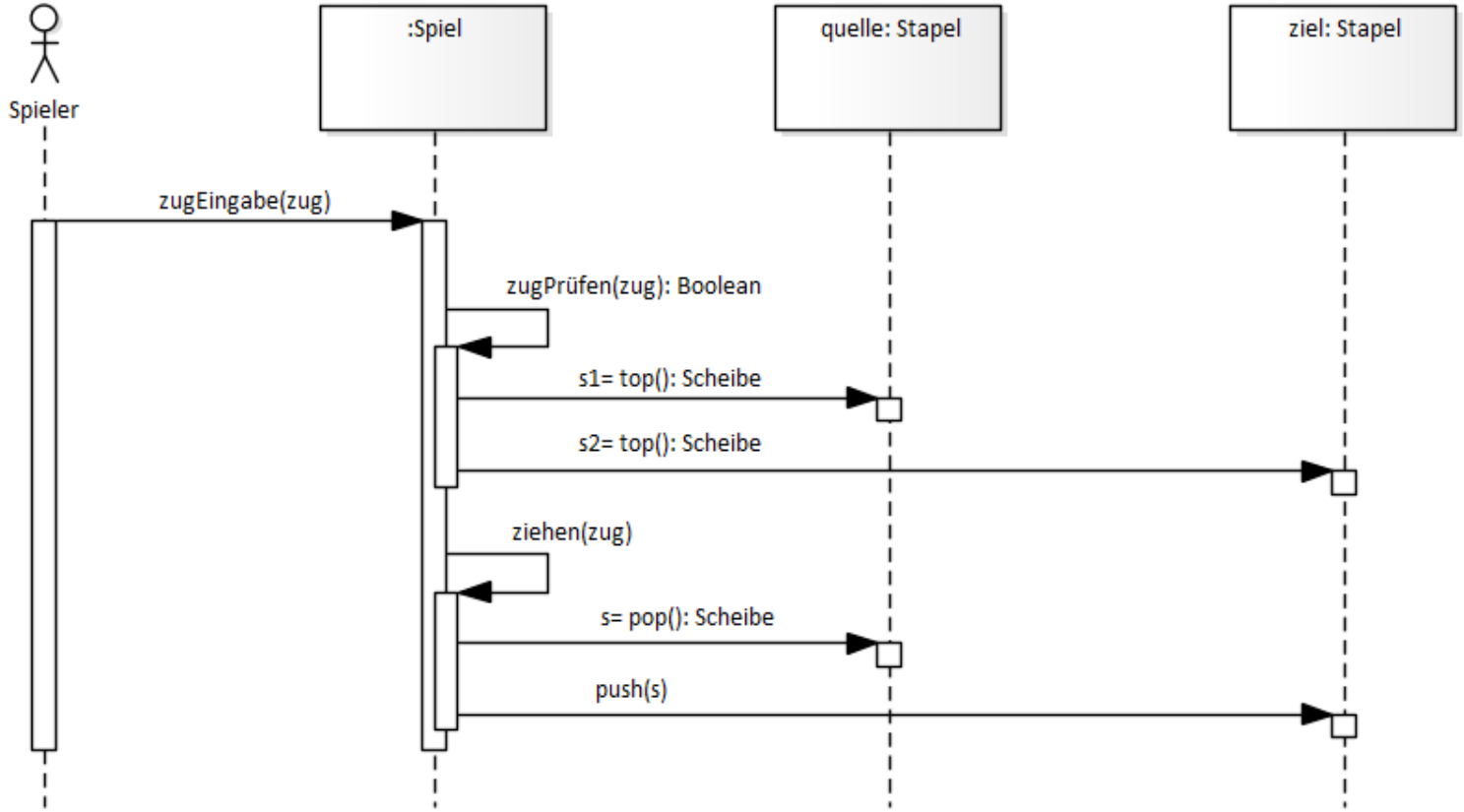
- **Objektorientierte Analyse – Statische Analyse (2)**
  - Welche **Beziehungen** bestehen zwischen den Objekten
    - Scheiben sitzen auf Stapeln
    - Die Stapel stehen auf dem Spielbrett
  - Stelle die Beziehungen im Modell als **Assoziationen** dar



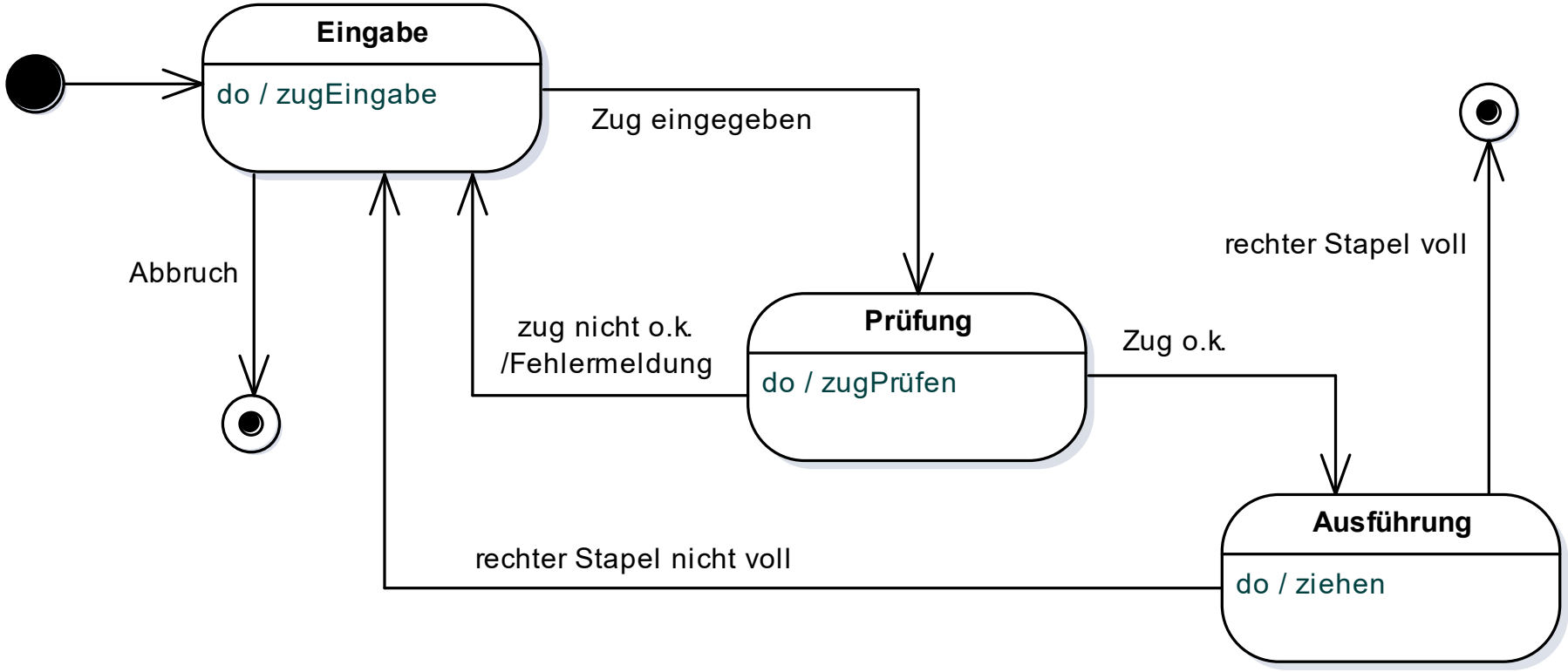
- **Objektorientierte Analyse – Statische Analyse (3)**
  - Was sind die relevanten **Eigenschaften** der Objekte einer Klasse
    - Das Spiel hat eine **Anzahl von Scheiben**
    - Stapel haben eine **Höhe**
    - Scheiben haben eine **Größe**
  - Stelle diese Eigenschaften im Modell als **Attribute** von Klassen dar



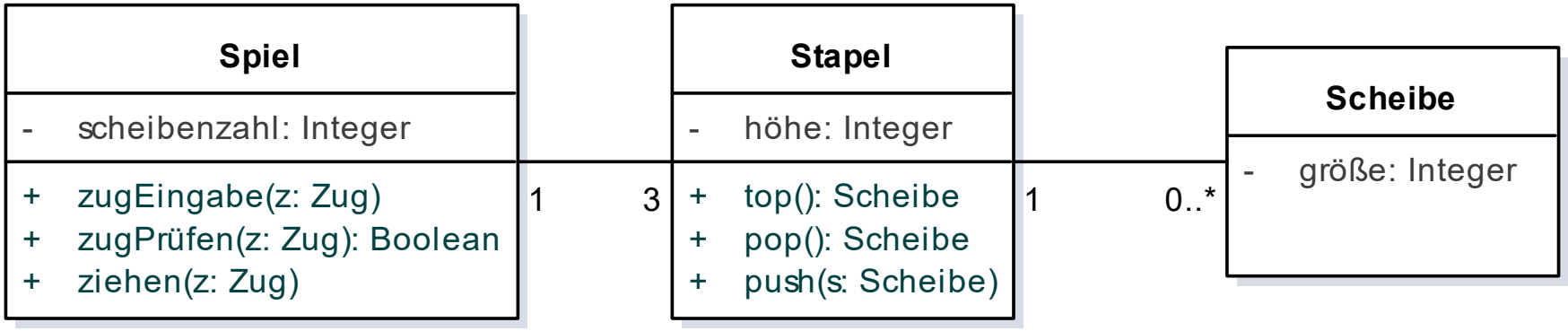
- **Objektorientierte Analyse – dynamische Analyse (1)**
  - Welche Abläufe gibt es im System?
  - Wie verhält sich das System (zur Laufzeit)
    - *Beispiel (1):* Verarbeitung eines Zuges als Sequenzdiagramm:



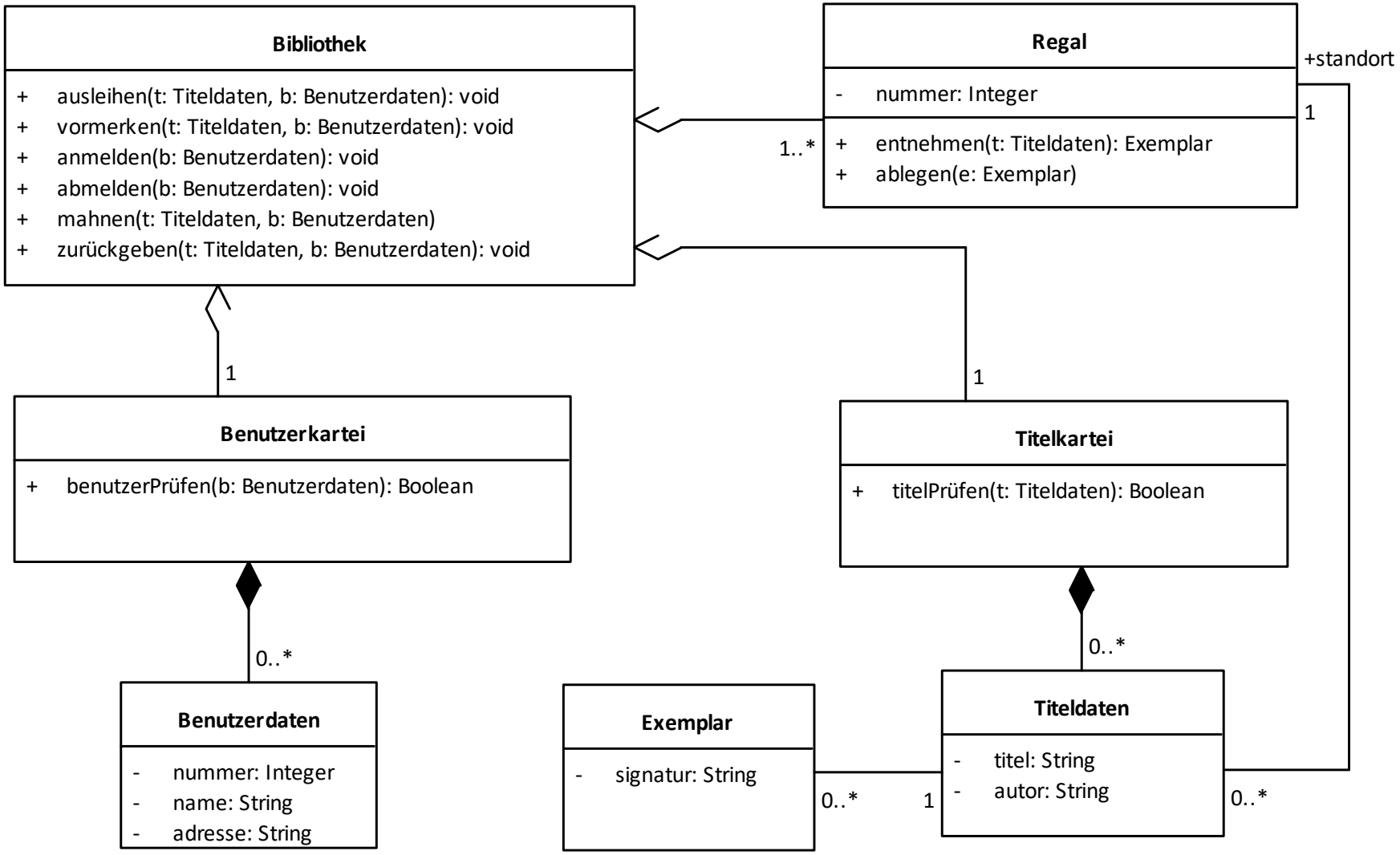
- **Objektorientierte Analyse – dynamische Analyse (2)**
  - Welche Abläufe gibt es im System?
  - Wie verhält sich das System (zur Laufzeit)
    - *Beispiel (2):* „Lebenszyklus“ eines Spiels als Zustandsautomat



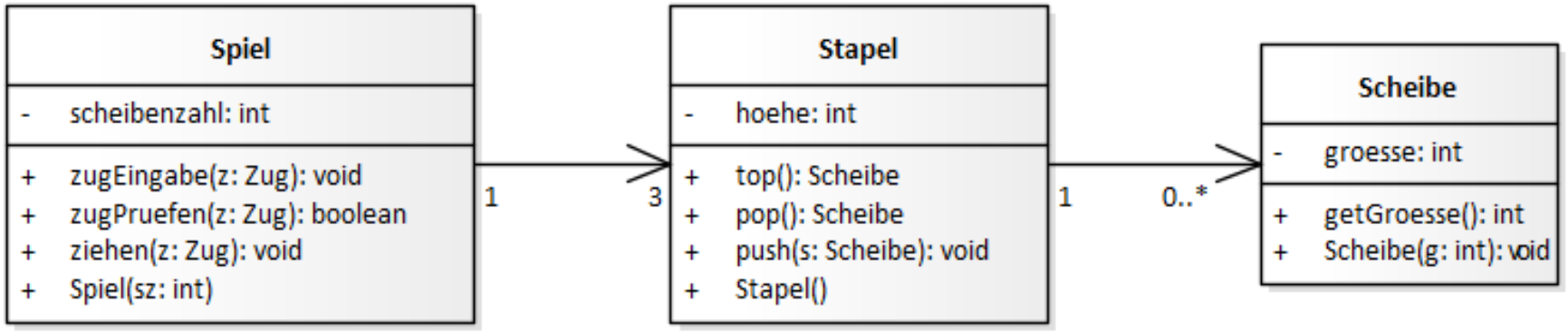
- **Objektorientierte Analyse – dynamische Analyse (3)**
  - Die Operationen aus den dynamischen Modellen geeigneten Klassen zuordnen



- **Beispiel:** Objektorientiertes Analysemodell einer Bibliothek

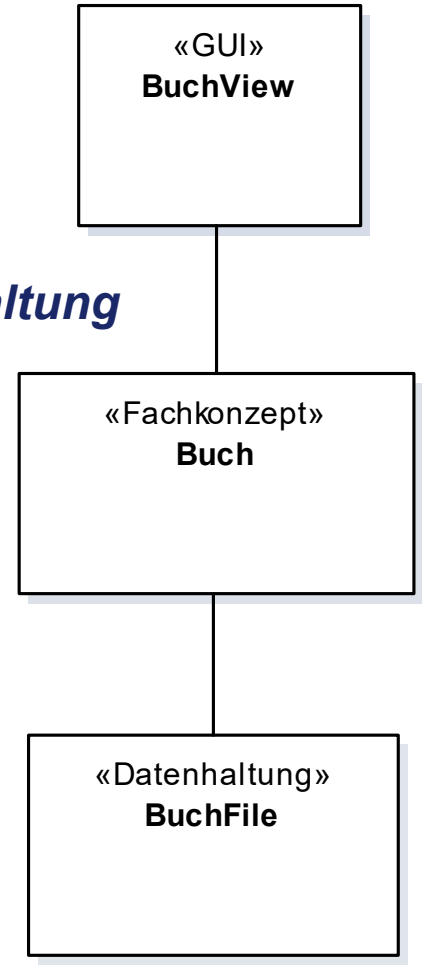


- **Objektorientierter Entwurf (1)**
  - Das „Reale-Welt-Modell“ in ein Modell des Programms transformieren, u.a.
    - Modell an die Programmiersprache anpassen
      - geeignete Datentypen
      - konforme Bezeichner
    - Fehlende Operationen ergänzen
      - Konstruktoren
      - set-/get-Methoden
    - Navigierbarkeit von Assoziationen festlegen



- **Objektorientierter Entwurf (2)**
  - Modellierung der **SW-Architektur**
  - Entwurfsziel
    - Weitgehende Entkopplung von
      Benutzungsoberfläche – Fachkonzept – Datenhaltung
      ggfs. auch Netzwerk-Kommunikation
  - Realisierung durch
    - Drei-Schichten-Architektur, Vier-Schichtenarchitektur
    - allgemein: Mehr-Schichten-Architekturen
  - Einfluss durch ...
    - verwendetes GUI *(Graphical User Interface)*
      - Bestimmt Aussehen der Benutzungsoberfläche
    - verwendete Form der Datenhaltung
      - Relationale Datenbank
      - Objektorientierte Datenbank
      - Flache Dateien

→ *Dazu mehr in „Software Engineering“ (IN4)*



- **Objektorientierte Implementierung**
  - Abbildung des statischen Entwurfsmodells in eine Programmiersprache
    - Üblich: objektorientierte Programmiersprache (C++, Java, C#,…)
      - Immer noch dieselben Konzepte wie in Analyse und Entwurf
        - » Objekte
        - » Klassen
        - » Vererbung
      - 1:1 Abbildungen von Modell-Klassen in Programmiersprachen-Klassen
      - Systematische Abbildung von Assoziationen
        - » Aus Assoziationen werden Objekt-Zeiger/-Referenzen
      - Das Programm spiegelt 1:1 das statische Entwurfsmodell
    - Möglich: systematische Abbildung in eine prozedurale Programmiersprache (z.B. C)
      - Häufig erforderlich, falls die Zielsysteme Mikrokontroller sind
  - Automatische Code-Generierung aus Entwurfsmodellen möglich
    - Auch bei Hand-Codierung ist das OOD-Modell eine exzellente Vorlage!

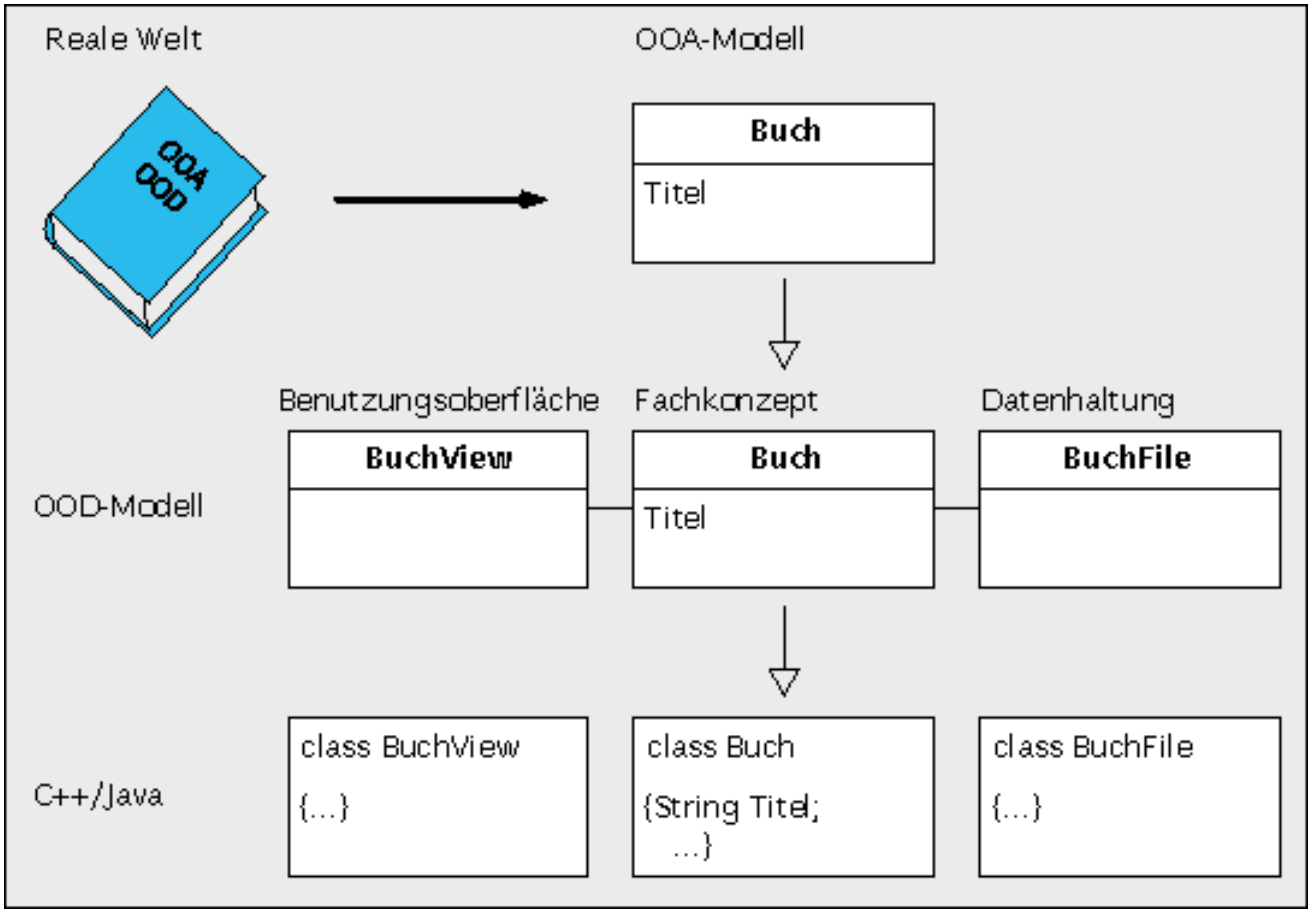
- Vorteile in der Objektorientierung
  - Systematische Übergänge von der „Realen Welt“ bis zum „Programm“
    - Reale-Welt-Objekte werden in Software-Objekte abgebildet

    → Hohe Verständlichkeit
  - dieselben Konzepte in allen Phasen der Entwicklung
  - eine einzige Notation in Analyse und Entwurf
  - kein Strukturbruch

    → Leichtere Nachvollziehbarkeit der Übergänge

Objektorientierter Entwurf

- Drei Schichten-Architektur



Quelle: [Balzert 05], Abb. 1.3-1

Objektorientierte SW-Entwicklung

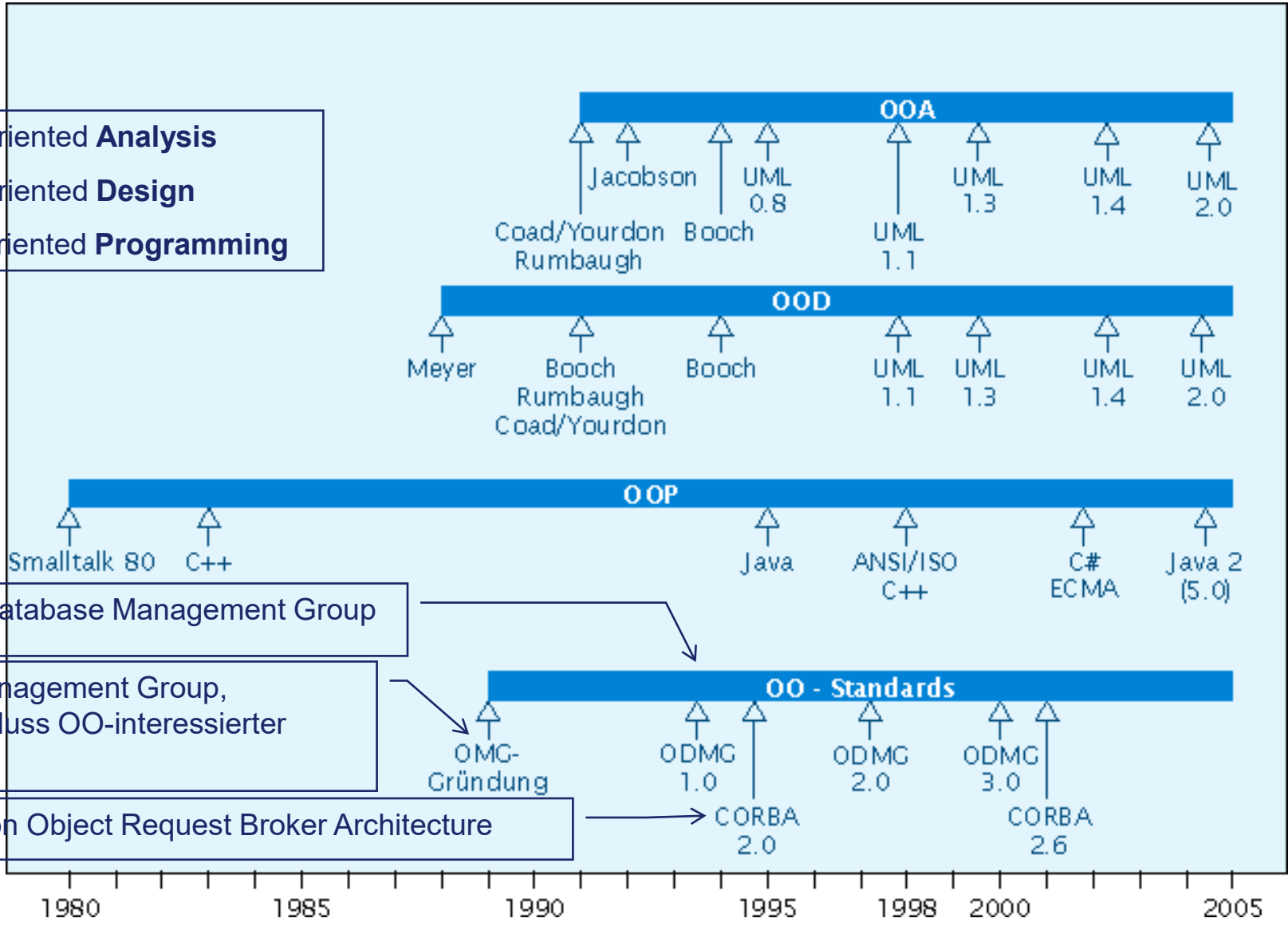
Geschichte der Objektorientierung

Quelle: [Balzert 05], Abb. 1.1-2

OOA: Object Oriented **Analysis**

OOD: Object Oriented **Design**

OOP: Object Oriented **Programming**



ODMG: Object Database Management Group

OMG: Object Management Group, Zusammenschluss OO-interessierter Firmen

CORBA: Common Object Request Broker Architecture

Übersicht und Ausblick

- Lernziele der Vorlesung
  - **Techniken** der
    - objektorientierten Analyse,
    - des objektorientierten Entwurfs und
    - der Transformation eines objektorientierten Entwurfs in eine Implementierung

    mit Hilfe der
    - Unified Modeling Language (UML),
    - der Programmiersprache C++ und
    - geeigneten Modellierungs- und Entwicklungswerkzeugen

    anwenden können.
- Voraussetzungen
  - Strukturiertes Programmieren
  - Objektorientiertes Programmieren (nicht notwendig C++)

Übersicht und Ausblick

- Inhalt der Vorlesung

1. Objektorientierte Softwareentwicklung ✓
2. Anforderungsanalyse mit UML
   - Anwendungsfalldiagramme
3. Statische Modellierung mit UML
   - Klassendiagramme
     Objekte und Klassen, Assoziationen, Vererbung
   - Paketdiagramme
4. Der Analyseprozess und Analysemuster
5. Dynamische Modellierung mit UML
   - Interaktionsdiagramme (Sequenz- und Kollaborationsdiagramme)
   - Aktivitätsdiagramme
   - Zustandsautomaten
6. Entwurf mit UML
7. Implementierung in C++

Übersicht und Ausblick

- Übungsaufgaben
  - Regelmäßige Übungsaufgaben auf den Folien
  - Gelegentlich ein extra-Übungsblatt
  - Übungsaufgaben sollten bearbeitet werden!
    - Auch für das Modellieren ist wie schon beim Programmieren ein gewisses Training erforderlich!
  - Lösungen werden teilweise in der Vorlesung besprochen
- Werkzeuge
  - Zum Modellieren: ***Enterprise Architect*** (SparxSystems)
    - Kann auch zu Hause installiert werden
    - Arbeiten zu Hause bei bestehender VPN-Verbindung zum Hochschul-Netz möglich
  - Zum Programmieren: was immer man möchte
    - z.B. ***Visual Studio*** (Microsoft)
    - z.B. ***DevC++*** (GNU GPL)

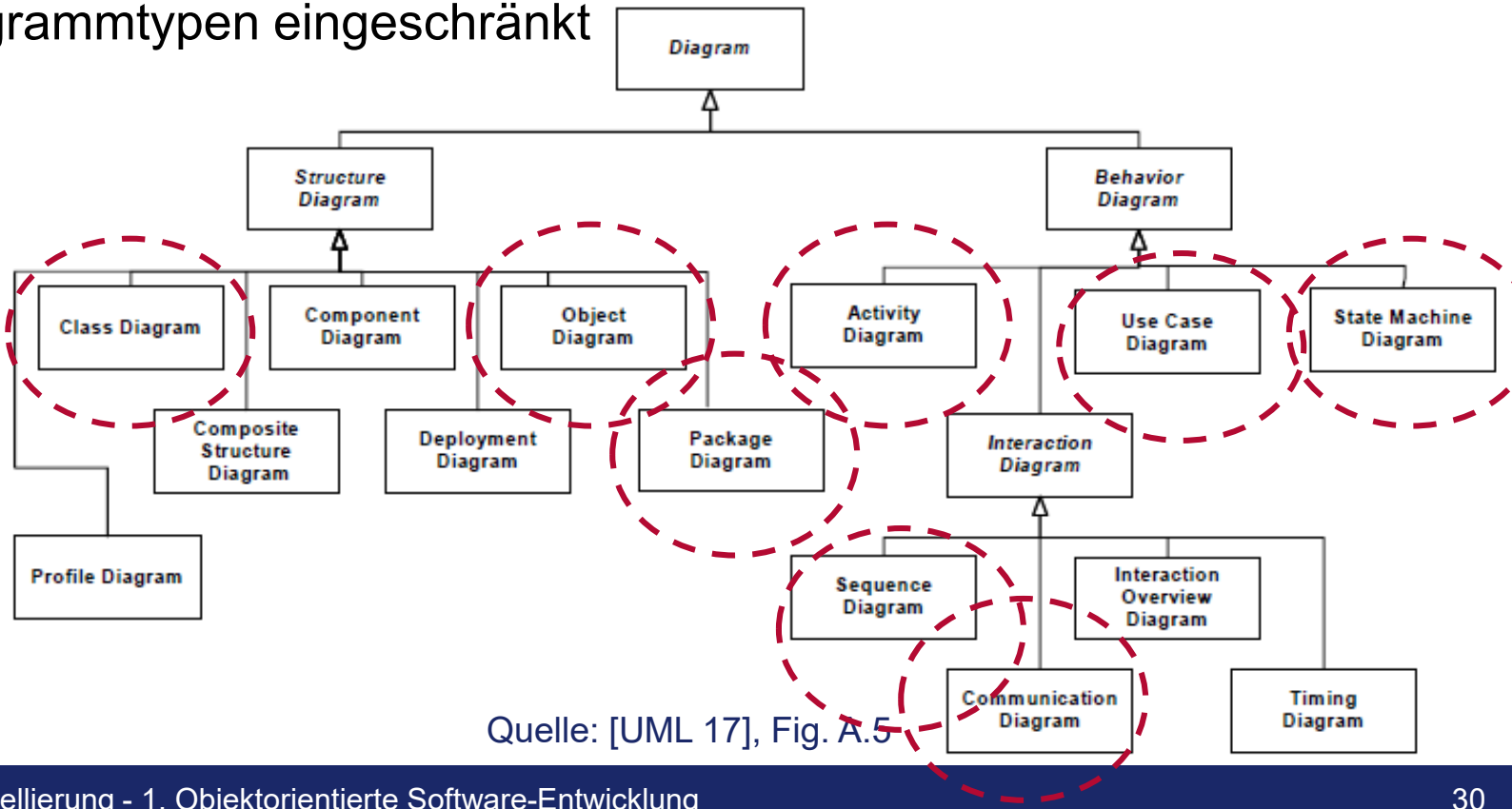
Übersicht und Ausblick

- **Praktikum** Objektorientierte Modellierung
  - 5 Termine
    - Im Wechsel mit Praktikum „Datenbanksysteme“ (*Grambow, 3 Termine*)
    - Beginn ca. Anfang November (WS) bzw. Anfang April (SS)
    - Die genauen Termine werden hier bekannt gegeben.
  - Inhalt:
    - Modellieren mit dem *Enterprise Architect*
    - Implementieren von Modellen in C++
- Prüfung
  - Klausur
    - **Zulassungsvoraussetzung:** bestandenes Praktikum
    - erforderlich: aktive Teilnahme
      - begründetes Fehlen möglich (→ **vorher** entschuldigen!)
  - **Zur Vorbereitung dringend empfohlen**:
    - Übungsaufgaben bearbeiten!

Übersicht und Ausblick

- Ausblick: Was kommt noch?
  - Die folgenden Lehrveranstaltungen verwenden die in OOM gelernten Techniken:
    - Software Engineering (IN4 – alle Schwerpunkte)
      - Mehr und detaillierter über den SW-Entwicklungsprozess
      - Erstellen von GUIs
      - Objektrelationale Abbildung: Abbildung eines OO-Fachkonzepts auf eine relationale Datenbank
      - Entwurfsmuster
    - Alle spezifischen Veranstaltungen des Schwerpunkts SE
      - Software Quality
      - Software Architecture
      - Cloud and Distributed Computing
    - OOM-Techniken können hilfreich sein in allen Lehrveranstaltungen, in den Softwareentwicklung eine Rolle spielt

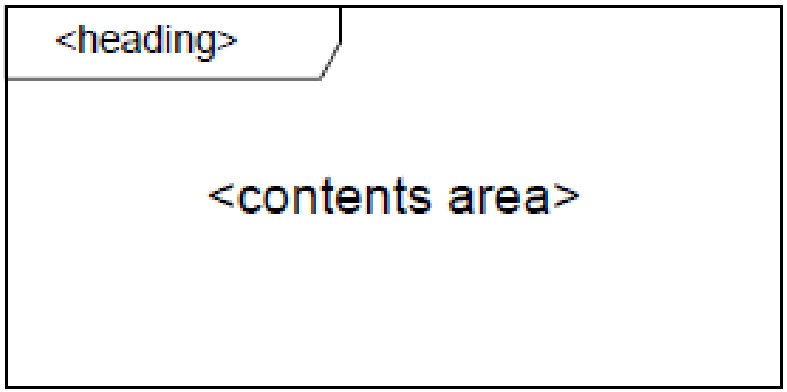
- Diagrammtypen
  - Die UML gruppiert Modellelemente in **Strukturelemente** und **Verhaltenselemente**
  - Es werden 13 unterschiedliche Diagrammtypen unterschieden
  - Die Verwendung eines Modellelements ist nicht auf bestimmte Diagrammtypen eingeschränkt



Quelle: [UML 17], Fig. A.5

- Diagrammrahmen
  - Ein „Diagramm“ ist keine UML-Modellelement
    - UML-Werkzeuge bieten in der Regel „Zeichenflächen“ an, auf denen man grafische Darstellungen von UML-Modellelementen platzieren kann.
  - Für manche Diagrammtypen definiert die UML **Diagrammrahmen,** bestehend aus
    - Kopf

      [<typ>] name [(parameter)]
      - Der Typ wird meist als Kürzel angegeben
    - Inhalt
      - beliebige Modellelemente

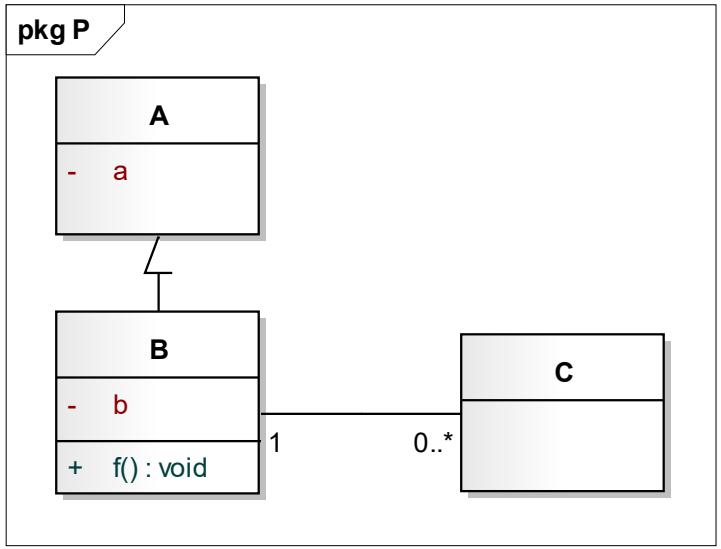


Quelle: [UML 17], Fig. A.1

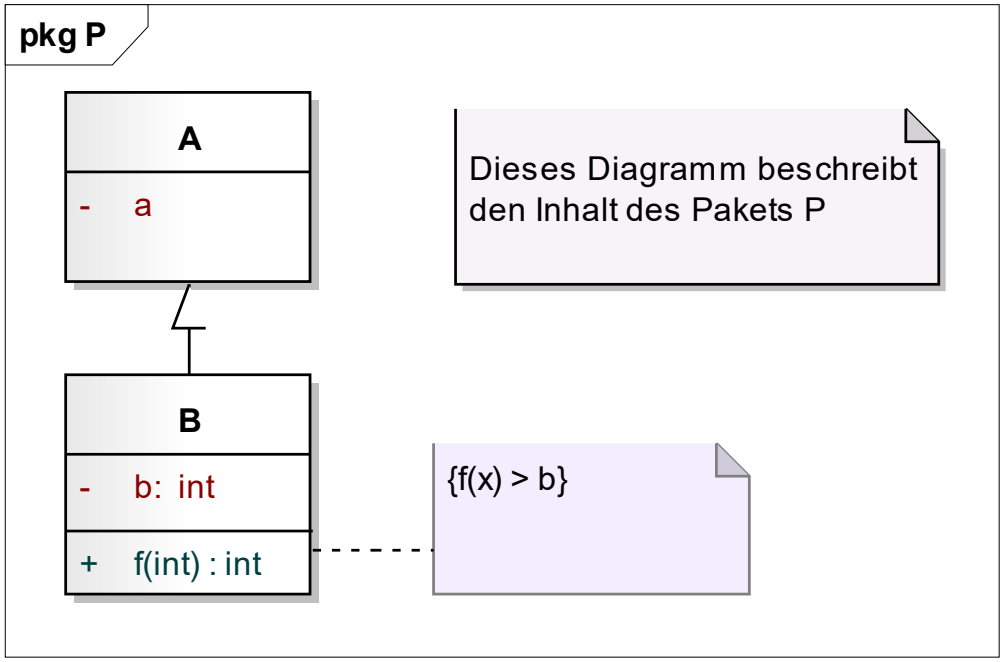
  - Zweck: das in einem Diagrammrahmen dargestellte Modell detailliert ein anderes Modellelement.
  - Der Rahmentyp drückt aus, was beschrieben wird

- Diagrammrahmen
  - **Rahmentypen** und deren **Kürzel**
    - *class (ohne Kürzel)* – Klasse
    - *package (**pkg**)* – Paket
    - *component (**cmp**)* – Komponente
    - *use case (**uc**)* – Anwendungsfall
    - *activity (**act**)* – Aktivität
    - *state machine (**stm**)* - Zustandsautomat
    - *interaction (**sd**)* - Interaktion
  - **Beispiel:** ein Paket und die Darstellung seines Inhalts als Klassendiagramm mit Diagrammrahmen vom Typ ***pkg***

*Ein **Paket** mit Namen **P***



- **Kommentare** (Notizen)
  - Freie Texte, die auf einem „Notizzettel“ in ein Diagramm platziert werden
  - Notizen können mit Modellelementen verbunden werden
- ***Constraints*** (Einschränkungen, Bedingungen)
  - Boole‘sche Ausdrücke, die über Eigenschaften eines Modellelements formuliert werden
  - Können in einer Notiz dargestellt werden



- **Stereotyp (*stereotype*)**
  - Gibt Elementen (z.B. Klassen, Attributen, Operationen) eines Modells eine besondere Bedeutung oder Charakteristik
  - UML enthält vordefinierte Stereotypen
  - Es können weitere Stereotypen definiert werden
  - Angabe in französischen Anführungszeichen *(<<guillemets>>)* mit Spitzen nach außen
  - Beispiele: Stereotypen für Anwendungsfälle und Klassen

Literatur

[Balzert 96] H. Balzert: *Lehrbuch der Softwaretechnik: Softwareentwicklung.* Spektrum Akademischer Verlag, 1996.

[Balzert 09] H. Balzert: *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering,* Spektrum Akademischer Verlag, 2009. (Überarbeitete Neuauflage von 1996, als e-book verfügbar)

[Ludewig, Lichter 10] J. Ludewig, H. Lichter: *Software Engineering. Grundlagen, Menschen, Prozesse, Techniken.* dpunkt.verlag 2010

[Balzert 05] H. Balzert: *Lehrbuch der Objektmodellierung*, Spektrum Akademischer Verlag, 2005.

[Rupp et al. 07] Ch. Rupp et. al.: *UML 2 glasklar*. Hanser Verlag, 2007.

[UML 17] *OMG Unified Modeling Language (OMG UML), Version 2.5.1.* https://www.omg.org/spec/UML/2.5.1/PDF . Dezember 2017