Software Engineering & PM Vorlesung

Block 4 „Modellierung von Anwendungsszenarien”

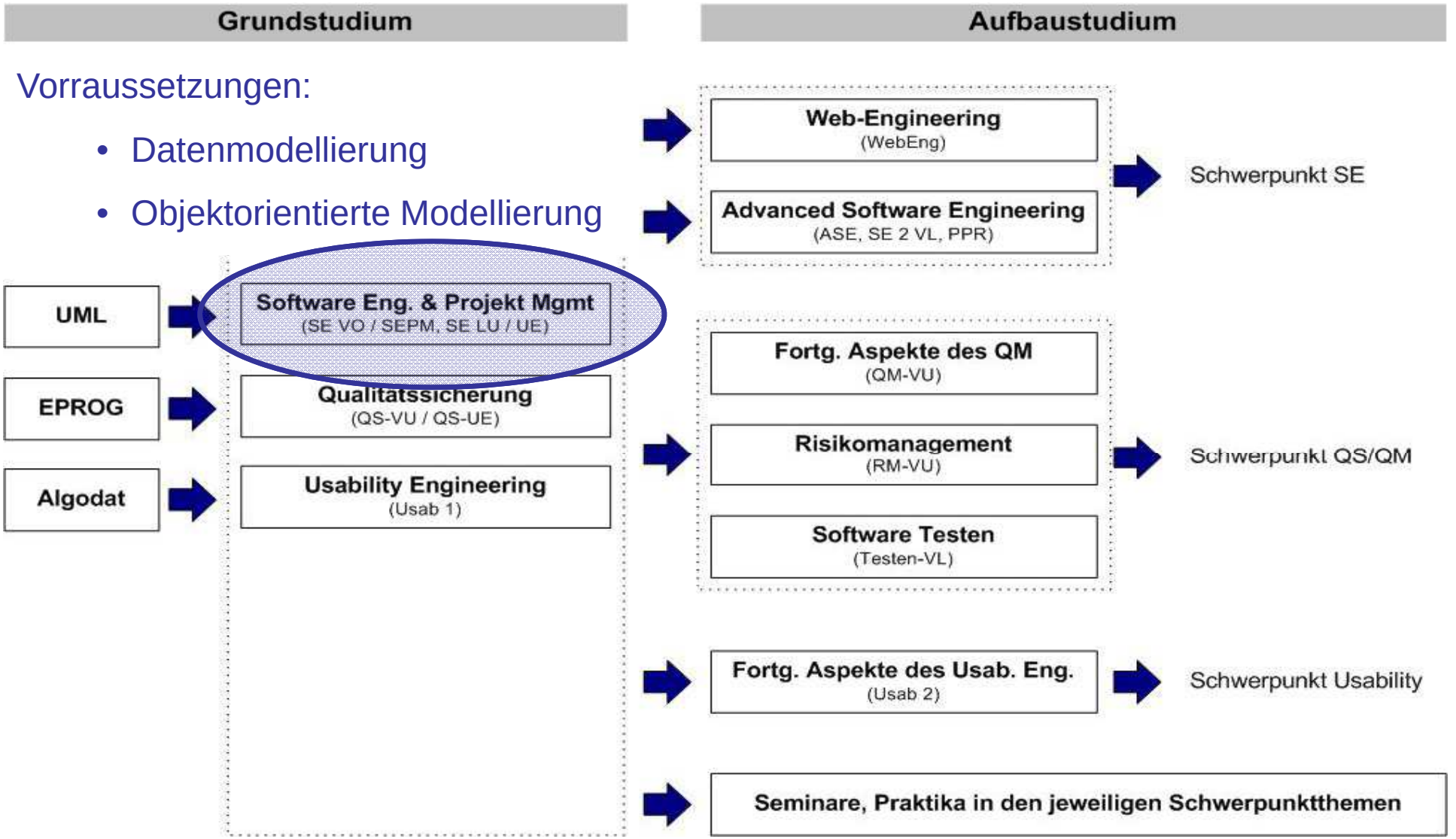
Institut für Softwaretechnik und Interaktive Systeme

Stefan Biffl Erik Gostischa-Franta

Inhalt:

- Grundlagen der Modellierung
- Bedarf an Modellierung in SEPM
- SEPM LU: Kernmodelle und deren Anwendung
  - Datenmodell (Relationenmodell in UML)
  - Daten- und Kontrollfluss eines Teilsystems (UML, IDEFØ)
- SEPM VO: Prüfungsbeispiel „Restaurant“

Software Engineering im Studium



Überblick Block 4: Modellierung von Anwendungszenarien

Block 4-1 “Modellierung in SEPM-Projekten” (20’)

- Grundlagen, Begriffe
- Typische Modelle und deren Verwendung (UML 2.1, IDEFØ)
- Ausblick: Verifikation und Validierung, Test-Driven Development

Block 4-2 “Datenmodellierung” (Struktur) (50’)

- Entity Relationship Diagramm (UML 2.1 Notation)
- Anwendung von Datenmodellen in der Verifikation und Validierung
- Anwendungsbeispiel “Restaurant”

Block 4-3 “Objektorientierte Modellierung” (Verhalten) (50’)

- Aktivitätsdiagramm: Daten- und Kontrollfluss (UML 2.1, IDEFØ)
- Anwendungsbeispiel “Restaurant”
- Zustandsdiagramm (State Chart)
- *Anwendung der Modelle in der Verifikation und Validierung*

Modellierung in SEPM-Projekten

- Grundlagen, Begriffe
  - Konzeptionelle Modellierung
    - Abstrakte Repräsentation
    - Zeigt Struktur und Verhalten des Problembereichs sowie des zu entwickelnden Softwaresystems
  - Modelle, Diagramme und Werkzeuge
    - UML Modell vs. UML Diagramm
    - UML: statische und dynamische Modelle
    - IDEFØ: Daten- und Kontrollfluss
- Modellierung in der Softwareentwicklung
  - Verfeinerung von Modellen (IDEFØ, UML)
- *Ausblick: Test-Driven Development, Verifikation und Validierung*
  - Modelle als Basis für die Implementierung und QS

Motivation - Ziel

- Software-Entwicklung erfordert die Herstellung konsistenter Sichten auf Anforderungen und Entwurf eines Systems.
- Modelle helfen den Überblick zu bekommen und zu behalten.
  - Grundlage für effektives und effizientes Arbeiten im Team.
  - Gemeinsame Notation mit konsistenter Bedeutung.
- Prinzipien der Abstraktion; Information Hiding und Vererbung.
- Herausforderung: Unterschiedliche Modelle flexibel und konsistent zu verwenden.
  - Systemstruktur: Subsysteme, Komponenten, Schnittstellen.
  - Verhalten von Komponenten; Interaktion zwischen Komponenten.
- Anforderungen -> Modelle -> Daten/Komponenten/Testspezifikationen.

Aus: OO Modellierung, BIG, 2006

Modellierungssprachen

Eine Brille zur Betrachtung des Problembereichs

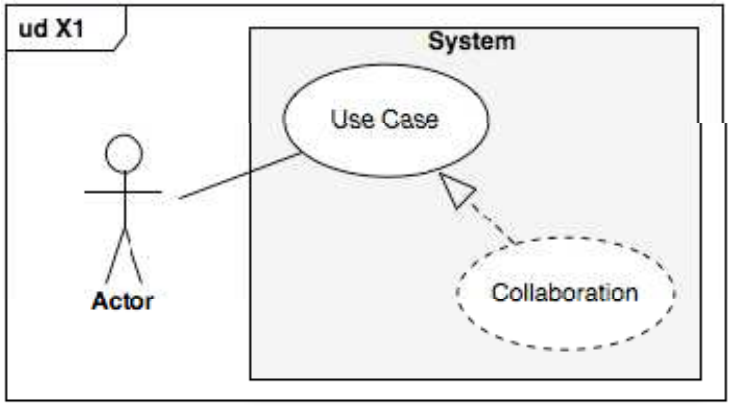
- beeinflusst die Wahrnehmung
- lässt Wesentliches sichtbar werden (ausser sie passt nicht – dann macht sie blind)

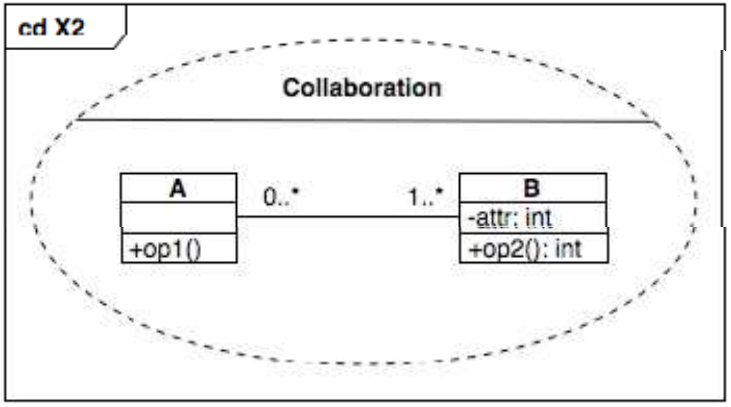
Ein Werkzeugkasten um neue Modelle zu bauen

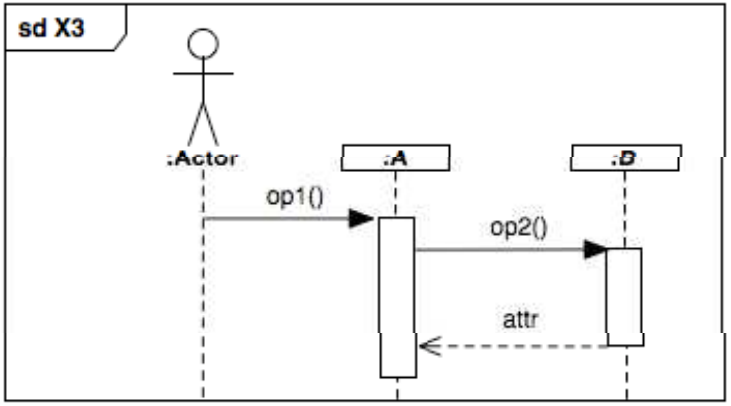
- gibt Möglichkeiten und Richtlinien vor
- schränkt allerdings auch ein

Aus: OO Modellierung, BIG, 2006

Modelle und Diagramme







UML **Modell** vs. UML **Diagramm**

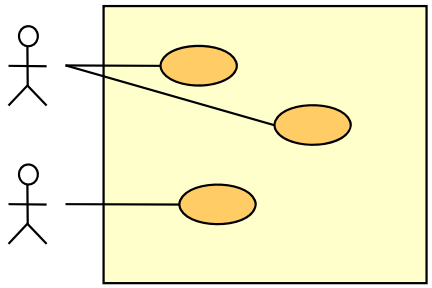
- Ein UML-Modell ist eine Menge von Modellelementen
  - Klassen, Attribute, Interaktionen, Anwendungsfälle, etc.
- Ein UML-Diagramm ist eine **Sicht** auf ein Modell)
  - Use Case Diagram (ud X1)
  - Class Diagram (cd X2)
  - Sequence Diagram (sd X3)
- Elemente in unterschiedlichen Diagrammen können sich auf ein und dasselbe Modellelement beziehen.

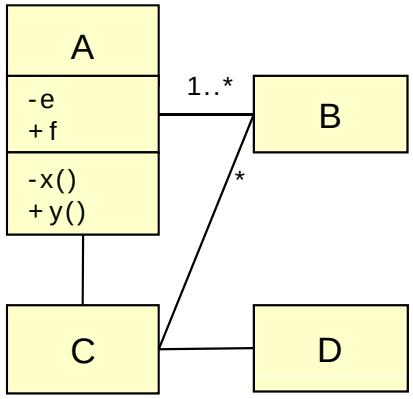
Werkzeuge

- **Modellierungswerkzeuge** implementieren die UML
  - Vorteile
    - Vorgabe der Modellstruktur, z.B. Konsistenzprüfer
    - Sourcecode-Generierung
  - Nachteil
    - oft komplex, schnelle Erstellung von Prototypen schwierig

- **Zeichenwerkzeuge** ermöglichen es nur, Diagramme zu zeichnen (z.B., auch ein Whiteboard ist ein Zeichenwerkzeug)
  - Vorteil
    - Freiheit in der Anwendung, evtl. besser Präsentationstauglich
  - Nachteil
    - unhandlich bei grösseren Modellen, z.B. mehrere Diagramme für ein Modell

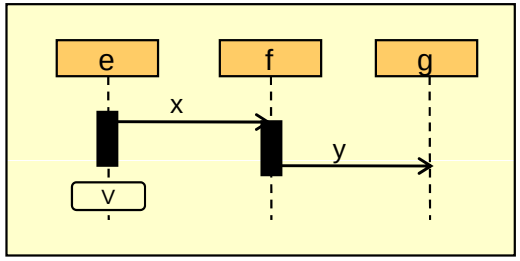
Schemata für statische Aspekte UML (“immer”)

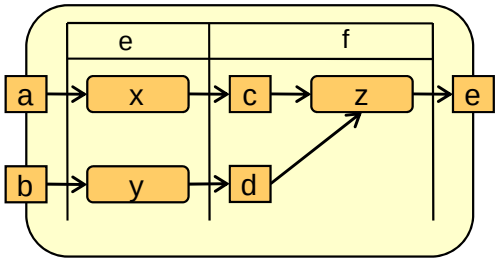


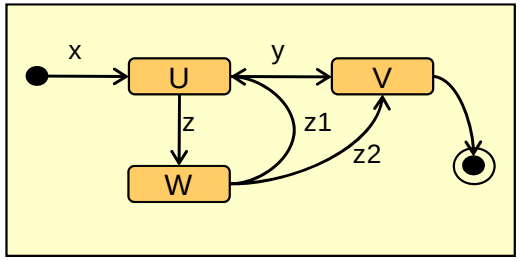


- **Anwendungsfälle**
  - die Benutzer und Nützlichkeit eines Systems
  - definiert durch Akteure und Hauptszenario
- **Klassen**
  - Menge gleichartiger Objekte
  - definiert durch Attribute, Operationen, und Beziehungen zu anderen Klassen
  - auch ein Datenbankschema kann in UML Klassendiagrammnotation modelliert werden.

Dynamische Aspekte mit UML ("wenn – dann")



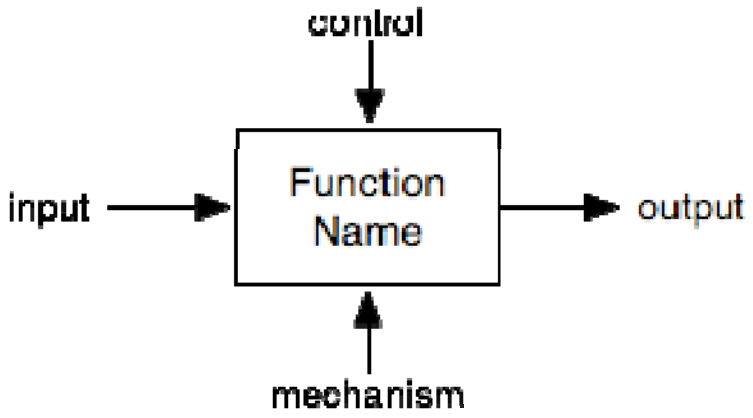


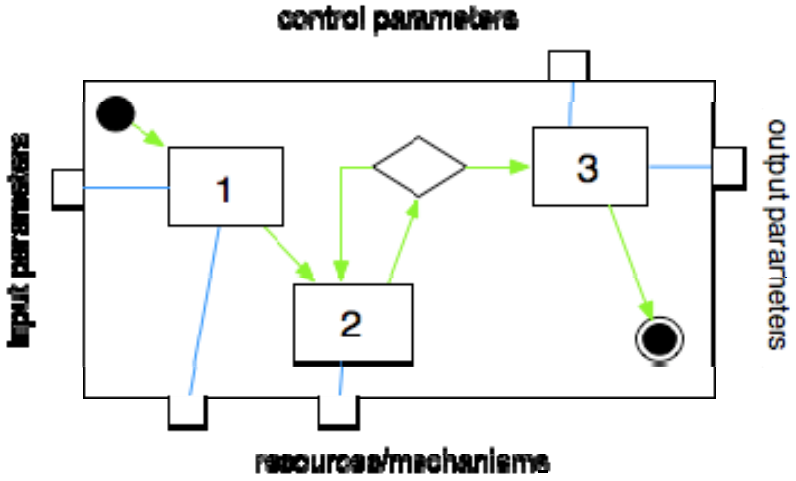


- **Sequenz**
  - ge- bzw. verbotene Interaktionsszenarien
  - definiert durch Folgen von Nachrichtenaustauschen zwischen Interaktionspartnern
- **Aktivität**
  - in sich geschlossenes, pro-aktives Verhalten
  - definiert durch Aktionen, Kontroll- und Datenfluss sowie evtl. Zuständigkeiten
- **Zustandsautomat**
  - reaktives Verhalten
  - definiert durch Zustände, Ereignisse und Transitionen

Aus: OO Modellierung, BIG, 2006

Prozess und Datenfluss Analyse (IDEFØ)





- Top-Down Modell
- Modellierung des Problembereichs
- Hilft neue Aufgabenbereiche zu erfassen und zu strukturieren

- Datenfluss (Daten-Token)
- Kontrollfluss (Kontroll-Token)
  - ergänzt um Start- und Endpunkt

Modellierung in der Softwareentwicklung

- **Konzeption**
  - Problemstellung, Geschäftsmodell
  - Big Picture – inkomplett, informal, vage
- **Anforderungen**
  - Problemstellung aufklären und ausarbeiten
  - Begriffe der Kundensprache (Fachbereich, Anwendung)
- **Entwurf**
  - Entwurfsalternativen explorieren und auswählen
  - Generische Begriffe der Softwaretechnik (Architektur, Komponenten, Algorithmen & Datenstrukturen)
- **Implementierung**
  - Implementierungsentscheidungen treffen, z.B., welche Datenbank, wie werden Assoziationen implementiert, etc.
  - Bestimmte Programmiersprachen, Bibliotheken, APIs

Modellierung

Aus: OO Modellierung, BIG, 2006

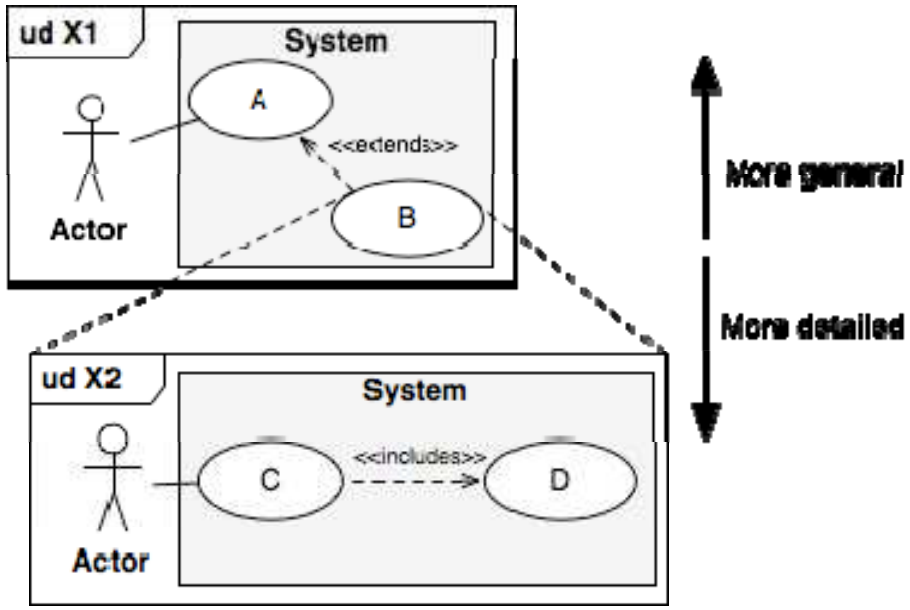
Ablauf SE: Wo werden Modelle eingesetzt

- Anforderungen ➔Modelle ➔Datenbank, Business Logic, GUI
  - Datenbanken: Datenmodellierung, Entity-Relationship Diagramm
  - Business Logic: Teilsysteme, Daten- und Kontrollfluss
  - Schnittstellen zwischen Teilsystemen und Benutzern

- Anforderungen ➔Qualitätsdefinition
- Modelle ➔Qualitätssicherung: Review, Testen
  - Anforderungen beschreiben Funktionen und Qualitäten
  - Zur operativen Definition sind die Qualitäten testbar zu beschreiben.

- Fokus: Test-Driven Development (TDD) Testfälle
  - Herstellen von ablauffähigen Testfällen **vor** der Implementierung
  - TDD überprüft die Spezifikation vor dem detaillierten Entwurf

Ablauf SE: Wie werden Modelle eingesetzt



- Anwendungsfalldiagramme sowie Daten- und Kontrollflussdiagramme werden Top-Down verfeinert
- Jede Phase im Entwicklungsprozess erfordert eine andere Abstraktionsebene in den Modellen
- System- und Modultests können anhand von Top-Down Modellen erstellt werden

Typische Fehler im Software Engineering SEPM LU Projekte

- Datenmodellierung, Reports
  - Schlüsselwerte nicht eindeutig, Fehlende Attribute
  - Fehler bei Beziehungen, z.B. 1:1 Relation im ER sehr selten
- Entwurf: Ergebniswerte von Methoden
  - Typkonversion falsch (z.B. Integer/Real)
- Initialisierung von Variablen
  - Fehlende Zuweisung: nicht eindeutiger Wert
  - Keyword static falsch verwendet: Klasse/Objekt (UML Rollen)
- Bedingungen
  - Logische Fehler; falsche Ausdrücke (und/oder)
- Schleifen
  - Anzahl Durchläufe nicht korrekt; z.B. Endlosschleifen
- Testfälle
  - Unzureichende Überdeckung (Auslassen von Use Cases, Programmteilen, GUI-Elementen, Datenzuständen)
  - Überflüssige Testfälle (gleiche Äquivalenzklasse)

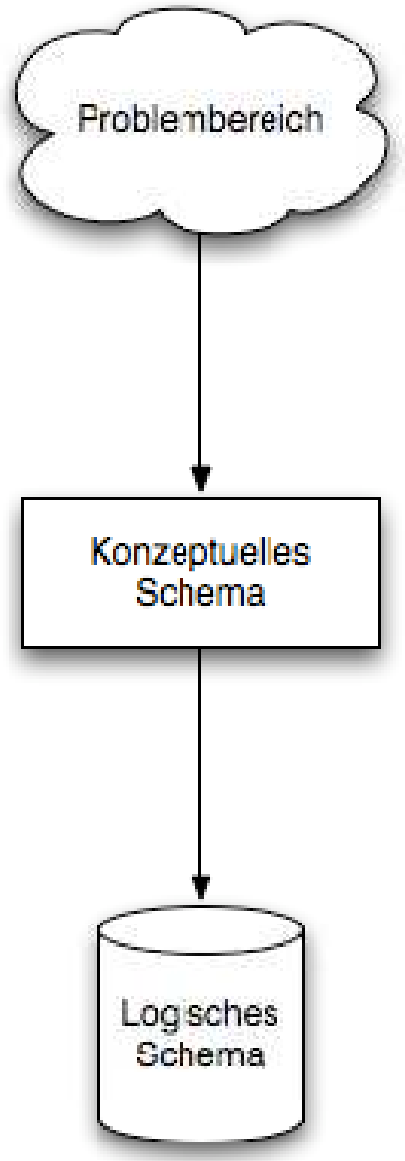
Ausblick: Test-Driven Development

- Ablauffähige Testfälle anhand eines Interfaces erstellen
- Assertions für „korrekte“ Durchführung des Testfalls ableiten (= erwartetes Ergebnis)
  - Normalfall soll nicht fehlschlagen
  - „Korrekter“ Fehlerfall muss auch wirklich fehlschlagen
    -> Testen mit Exceptions
- Brauchbare Modellierung unterstützt das Herstellen effektiver und effizienter Testfälle
  - Datenmodellierung (Datenbank)
  - Datenfluss und Kontrollfluss (Business Logic)
  - Zustandsübergangsmodelle (GUI)

Datenmodellierung (Struktur)

- Entity Relationship Diagramm in UML
  - Konzeptuelles und logisches Schema einer Datenbank
  - Entitäten, Attribute, Schlüssel, Relationen
  - Konsistenz- bzw. Integritätsbedingungen
  - Relationenmodell
- Anwendung von Datenmodellen in der Verifikation und Validierung
  - Datenorientierte Testfälle (Black Box)
  - Datenbank: SQLUnit
  - Unit Tests eines DAO Pattern via Mocking
- Anwendungsbeispiel “Restaurant”

Datenmodellierung Überblick



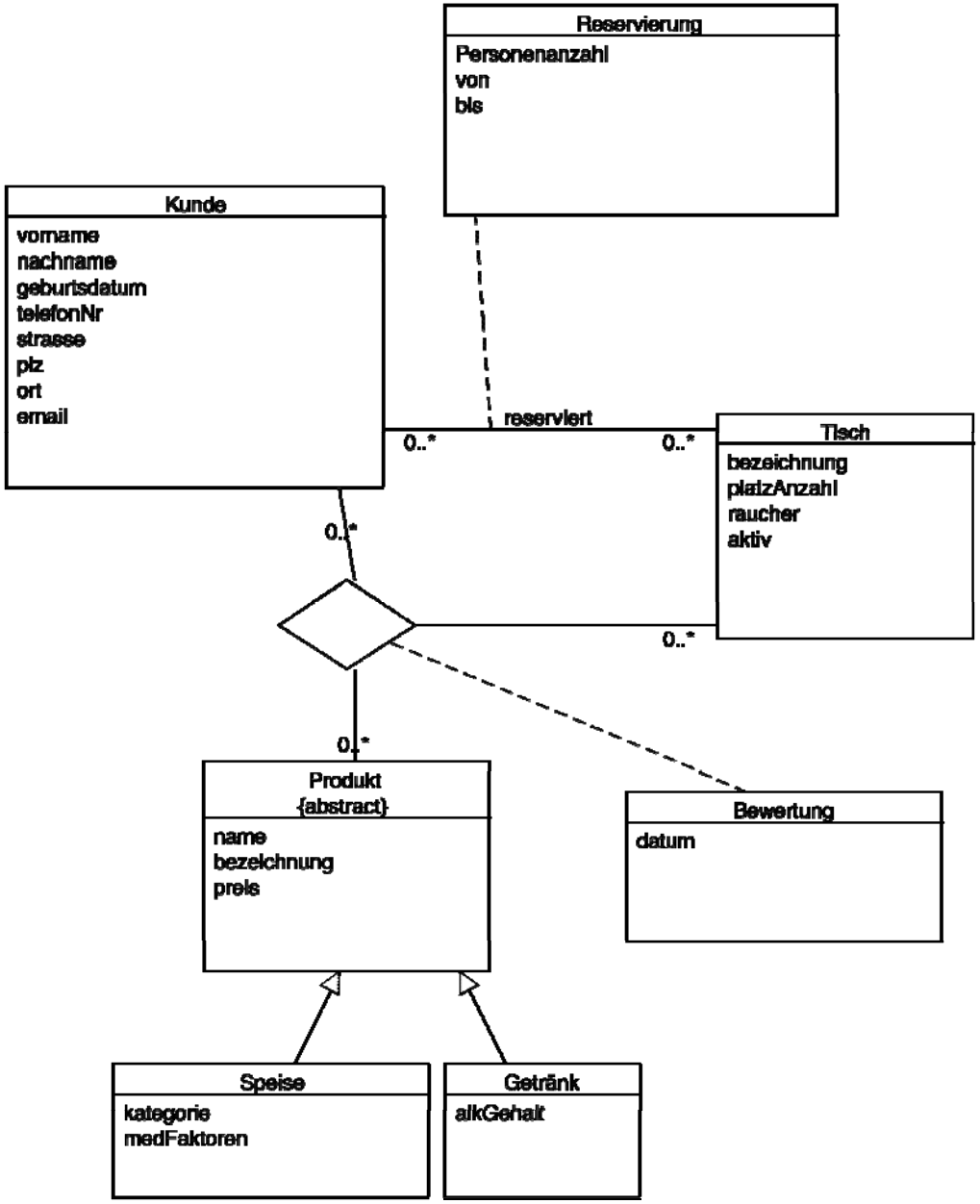
- Ausschnitt der Realen Welt
  - Beschrieben mit Text als Spezifikation oder Anforderungen.
  - Text Beschreibung Enthält Akteure, Entitäten, Aktivitäten und Operationen.
- Modellierung des Problembereichs durch Top-Down Diagramme wie Anwendungsfälle (UML) und Daten-/Kontrollfluss (IDEFØ)
  - Domänenmodelle, Klassendiagramme und Sequenzdiagramme (UML) führen zur Beschreibung des Softwaresystems
  - Datenmodelle (Entity Relationship) beschreiben die Persistenten Daten
  - Logisches Schema in dritter Normalform

Beispiel für Entity Relationship
Bewertung von Speisen und Getränken

Restaurant Datenbank:

- Kontext: Gehobenes Restaurant
- Ziel: Vorlieben der Stammgäste erheben
- Bewertung bei jedem Besuch

- Elemente:
  - Entitäten
  - Attribute
  - Beziehungen
    - Vielfachheiten

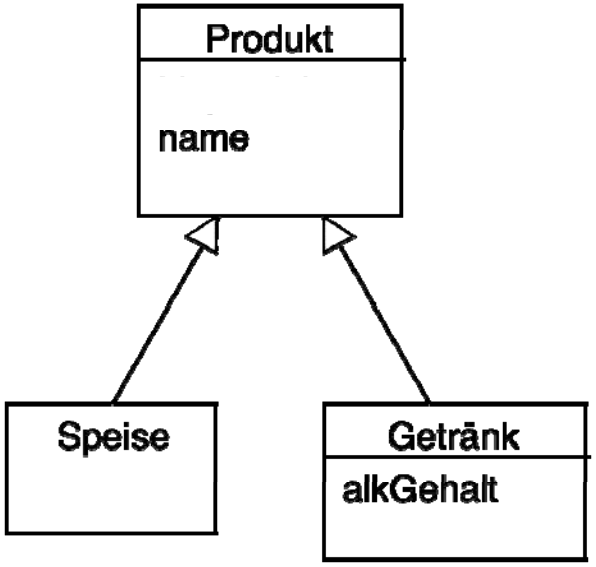


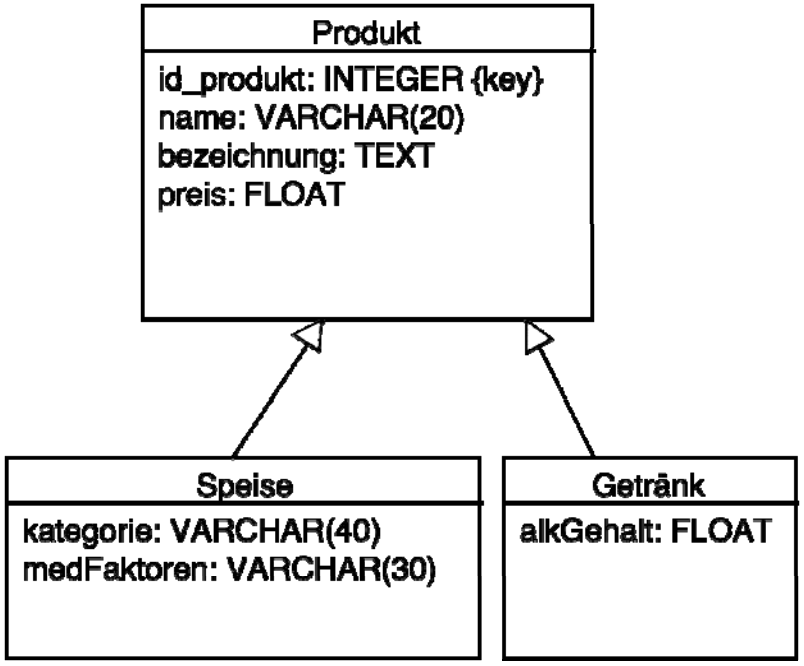
Beispiel „Restaurant“

- Typisches Prüfungsbeispiel
  - Siehe auch Prüfungsordner im TUWEL
- Auszug aus Anforderungsbeschreibung
  - Tip: Bei unklaren Anforderungen: Annahmen festhalten.
- Input für die Erstellung von diversen Modellen
- Einführung

Beispiel für Entity Relationship
Bewertung von Speisen und Getränken

- Bottom-Up Analyse: Entitäten ergänzen
  - Attribute, Typen und Schlüssel hinzufügen





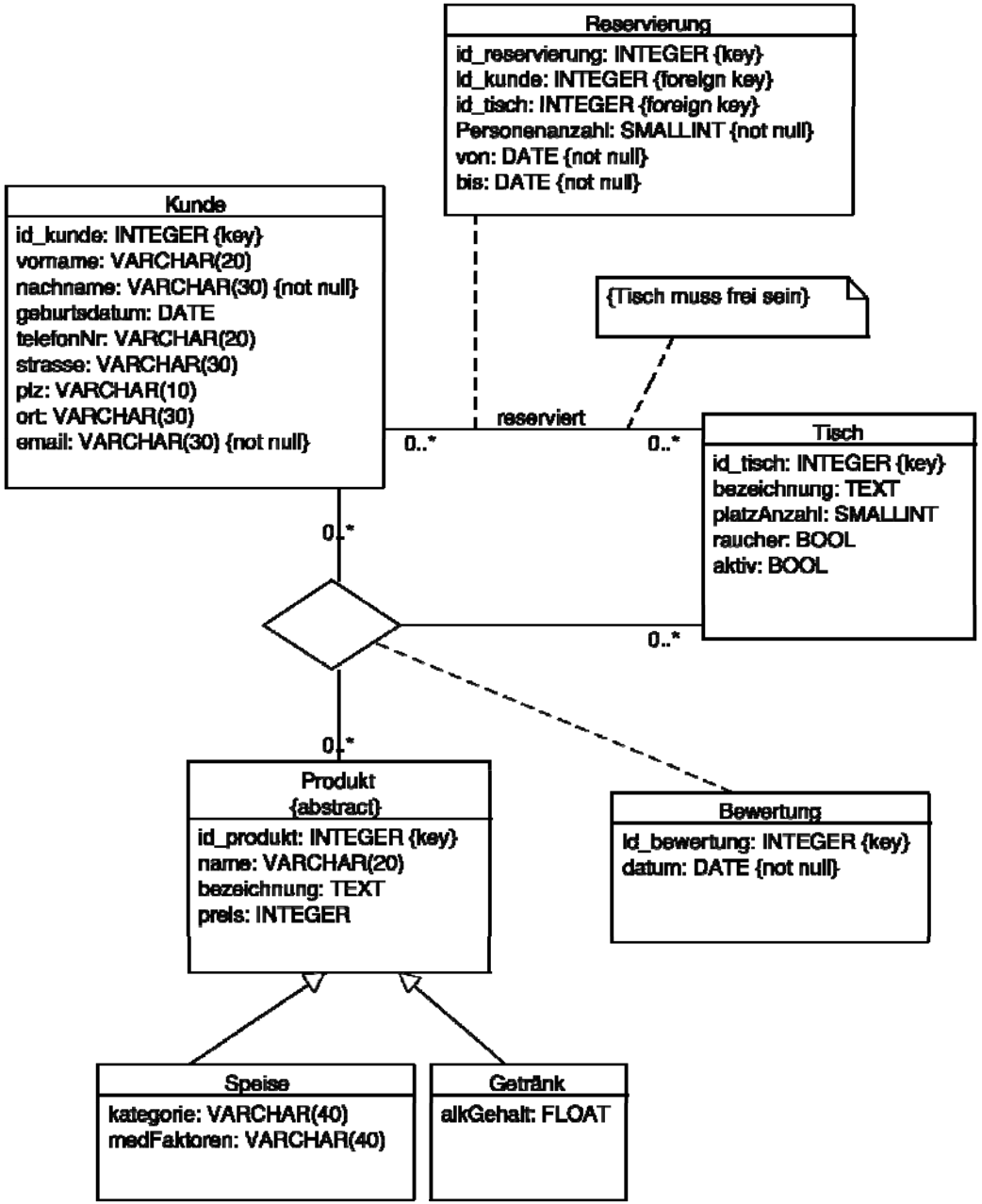
Entity Relationship: Verfeinerung

- Kandidaten für Entitäten identifizieren
  - Attribute zu Typen hinzufügen
  - Kandidaten für Schlüsselattribute markieren
- Kandidaten für Beziehungen zwischen Entitäten identifizieren
  - Vielfachheiten der beteiligten Entitäten überprüfen
    - Typisch: 1, 1..*, 0..1, 0..*
  - Beziehungen auf Attribute überprüfen
- Integritätsbedingungen
  - Pro Attribut (z.B. Zahl muss positiv sein)
  - Über mehrere Attribute (ev. auch Beziehungen) hinweg (z.B. eine Zahl muss kleiner als eine andere sein)

Beispiel für Entity Relationship
Bewertung von Speisen und Getränken

Verfeinerung:

- Entitäten
  - Eine Tabelle pro Entität
- Attribute
  - Schlüssel
    - Eigenschlüssel
    - Fremdschlüssel
  - Typen
- Integritätsbedingungen
- Beziehungen
  - Vielfachheiten
- Beschreibung als UML Notiz statt Tabelle



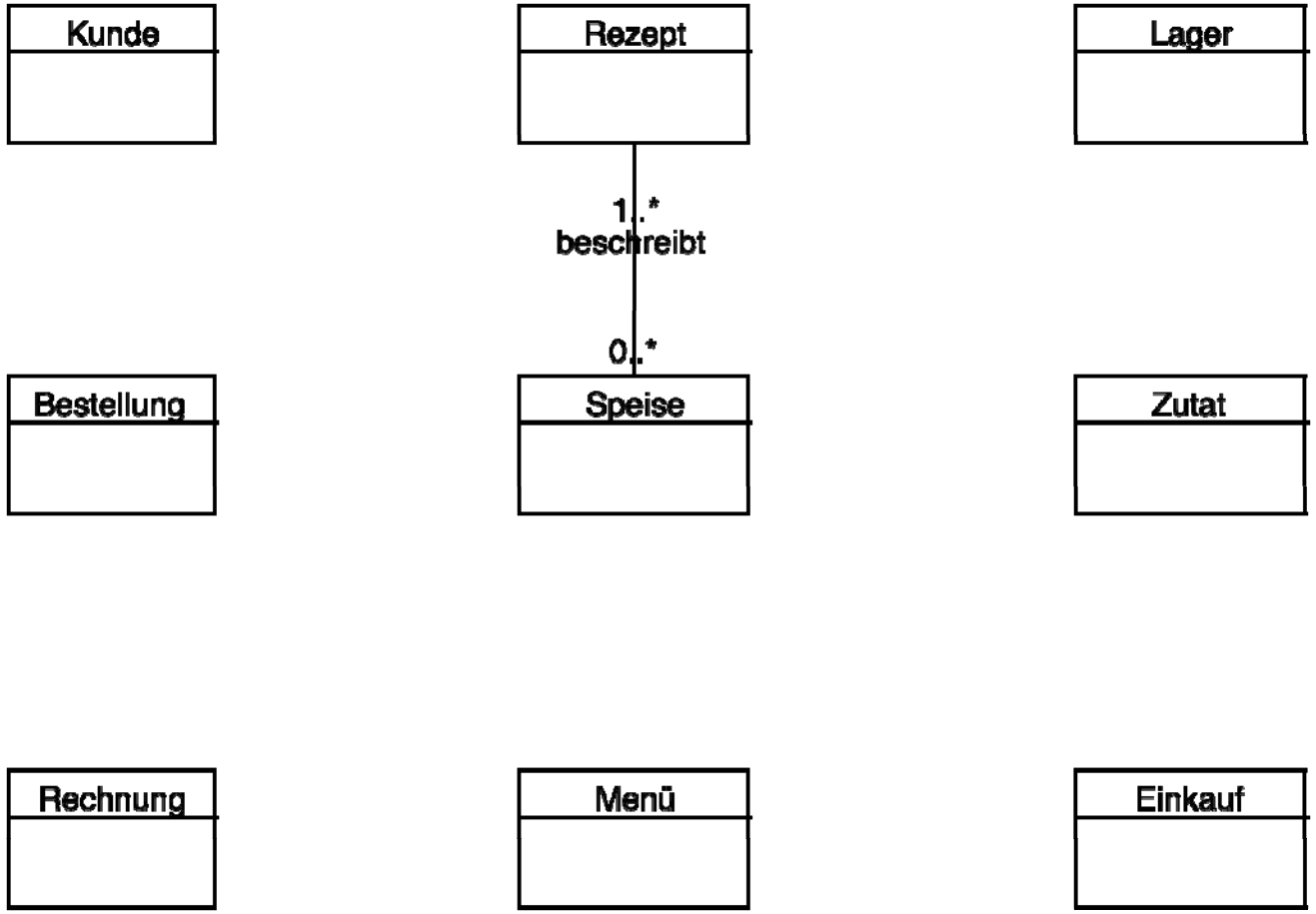
Zusammenfassung Datenmodellierung

- Ableitung von Datenbankrelationen
  - Jede Entität als Relation abbilden
  - Jede m:n Beziehung als neue Relation abbilden
  - Jede 1:n Beziehung einer bestehenden Relation hinzufügen.

- Datenbankrelationen: Attribute spezifizieren
  - Typen (VARCHAR, INTEGER, BOOL, FLOAT etc.)
  - Schlüssel überprüfen (müssen immer eindeutig sein)
    - Eigenschlüssel und Fremdschlüssel

Übung:
Restaurant – Sammlung von Entitäten

- Entitäten um Attribute ergänzen
- Beziehungen und Schlüssel hinzufügen

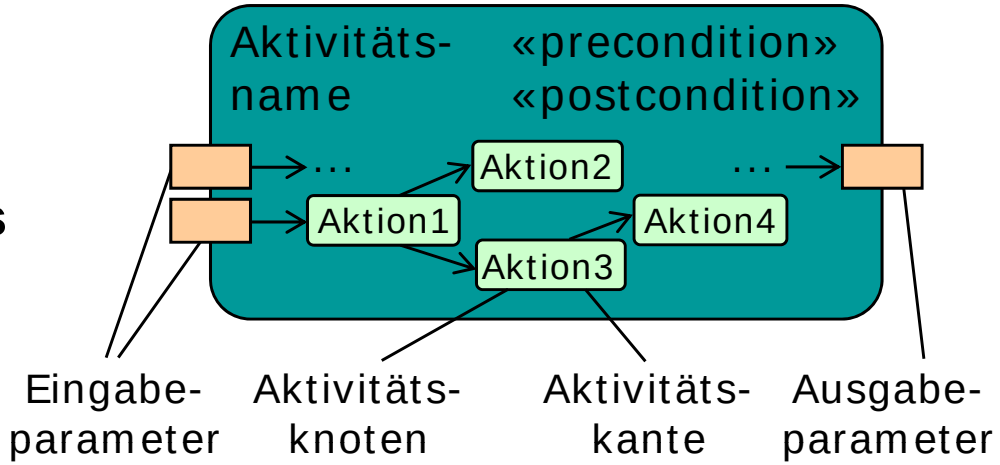


Objektorientierte Modellierung

- Aktivitätsdiagramm: Daten- und Kontrollfluss
  - Anwendungsfall: Typisches Verwendungsszenario für System
  - Aktivitätsdiagramm: Datenfluss zwischen Prozessabläufen
  - Zustandsdiagramm (State Chart): Falls bei Abläufen die Beschreibung von Zuständen und erlaubten Übergängen wichtig ist (GUI)
  - Sequenzdiagramm: Kommunikationsdetails zwischen Objekten.
- Anwendung der Modelle in der Verifikation und Validierung
  - Test-Driven Development: Testspezifikation, Testautomatisierung
  - Ableitung aus Modellen: Datenorientiert, Ablauforientiert
- Anwendungsbeispiel “Restaurant”

Aktivitätsdiagramm

Basiskonzepte – Bestandteile einer Aktivität



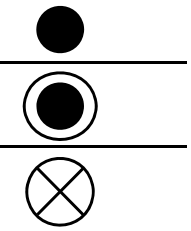
- Aktivität
  - Spezifiziert **benutzerdefiniertes Verhalten** auf unterschiedlichen Granularitätsebenen
  - Ist ein **gerichteter Graph**, bestehend aus **Knoten** und **Kanten**
  - Umfasst **Aktionen**, **Kontroll**- und **Datenflüsse**
  - Kann einem **Classifier zugeordnet** werden – direkt oder indirekt über Methode – oder aber »**autonom**« existieren
  - Wird durch **Operationsaufruf** oder **Classifier-Instanziierung** ausgeführt
  - Kann **Parameter** sowie Vor- und Nachbedingungen aufweisen und definiert einen Namensraum
- Aktion
  - Ist atomar
- Kontroll- und Datenflüsse legen potenzielle »Abläufe« fest

Aus: OO Modellierung, BIG, 2006

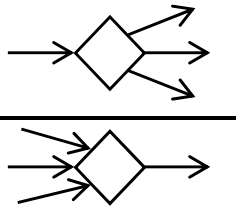
Aktivitätsdiagramm

Basiskonzepte – Knoten

- **Aktionsknoten** repräsentieren vordefinierte Aktionen
- **Objektknoten** dienen als formale Ein- und Ausgabeparameter sowie als zentrale Puffer
- **Kontrollknoten** steuern Aktivitätsabläufe
  - **Start und Ende von Abläufen**
    - Initialknoten
    - Aktivitätsendknoten
    - Ablaufendknoten



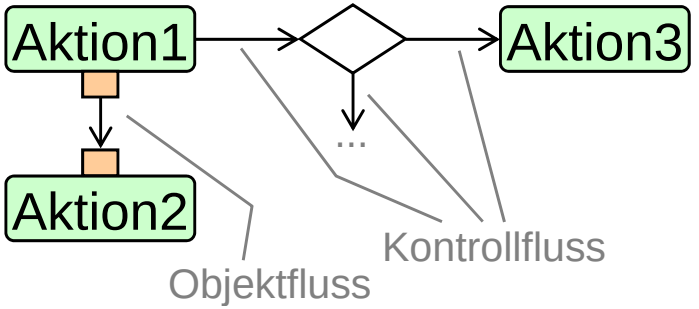
  - **Alternative Abläufe**
    - Entscheidungsknoten
    - Vereinigungsknoten



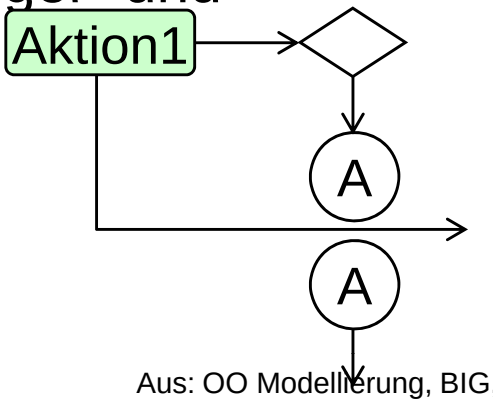
Aus: OO Modellierung, BIG, 2006

Aktivitätsdiagramm

Basiskonzepte – Kanten



- Kanten verbinden Knoten und legen **mögliche Abläufe** einer Aktivität fest
- **Kontrollflusskanten**
  - Drücken eine **reine Kontrollabhängigkeit** zwischen Vorgänger- und Nachfolgerknoten aus
- **Objektflusskanten**
  - Transportieren **zusätzlich Daten** und drücken dadurch auch eine **Datenabhängigkeit** zwischen Vorgänger- und Nachfolgerknoten aus
- **Überwachungsbedingung (guard)**
  - Bestimmt, ob Kontroll- und Datenfluss weiterläuft oder nicht
- **Fortsetzungsmarken (connectors)**

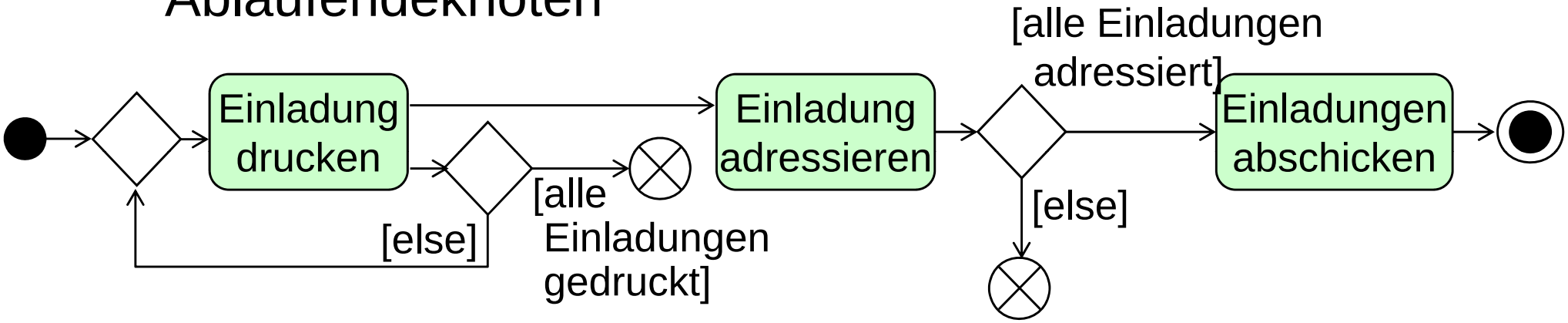


Aus: OO Modellierung, BIG, 2006

Aktivitätsdiagramm

Basiskonzepte – Start und Ende von Aktivitäten und Abläufen 2/2

- Beispiel mit Initial-, Aktivitätsende- und Ablaufendeknoten

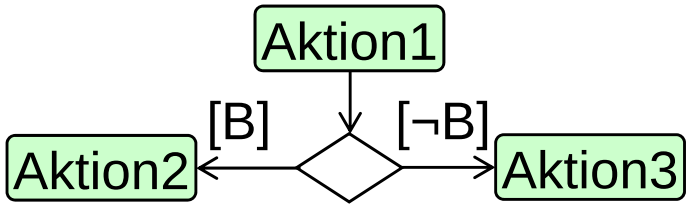


Aus: OO Modellierung, BIG, 2006

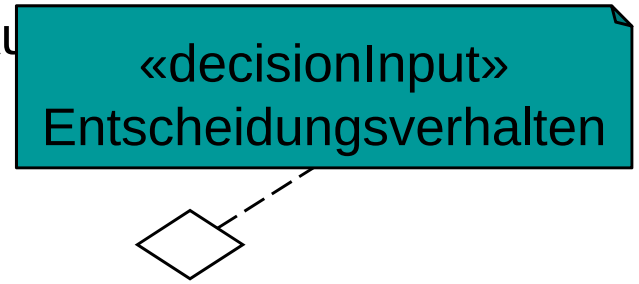
Aktivitätsdiagramm

Basiskonzepte – Alternative Abläufe 1/2

- **Entscheidungsknoten**
  - Definiert alternative Zweige und repräsentiert eine »Weiche« für den Tokenfluss
    - Verwendung auch zur Modellierung von Schleifen



  - **Überwachungsbedingungen**
    - Wählen den Zweig aus
    - Müssen wechselseitig ausschließend sein, [else] ist vordefiniert
  - **Entscheidungsverhalten**
    - Ermöglicht detailliertere Spezifikation der Au an zentraler Stelle
    - Ankunft von Token startet das Entscheidungsverhalten – Datentoken fungieren als Parameter

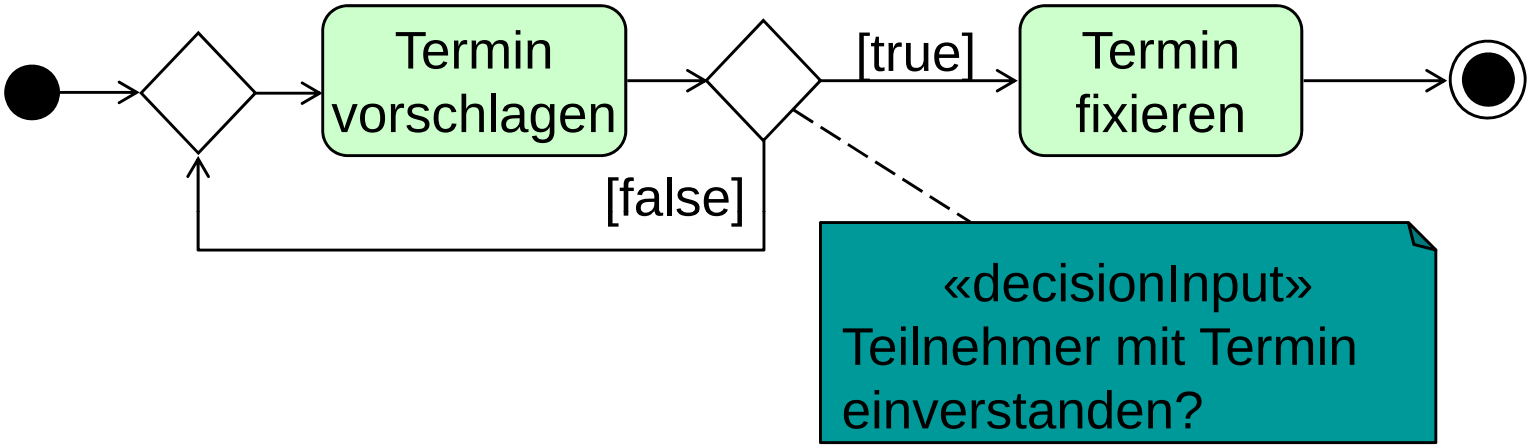


Aus: OO Modellierung, BIG, 2006

Aktivitätsdiagramm

Basiskonzepte – Alternative Abläufe 2/2

- Beispiel mit Entscheidungsknoten und -Verhalten

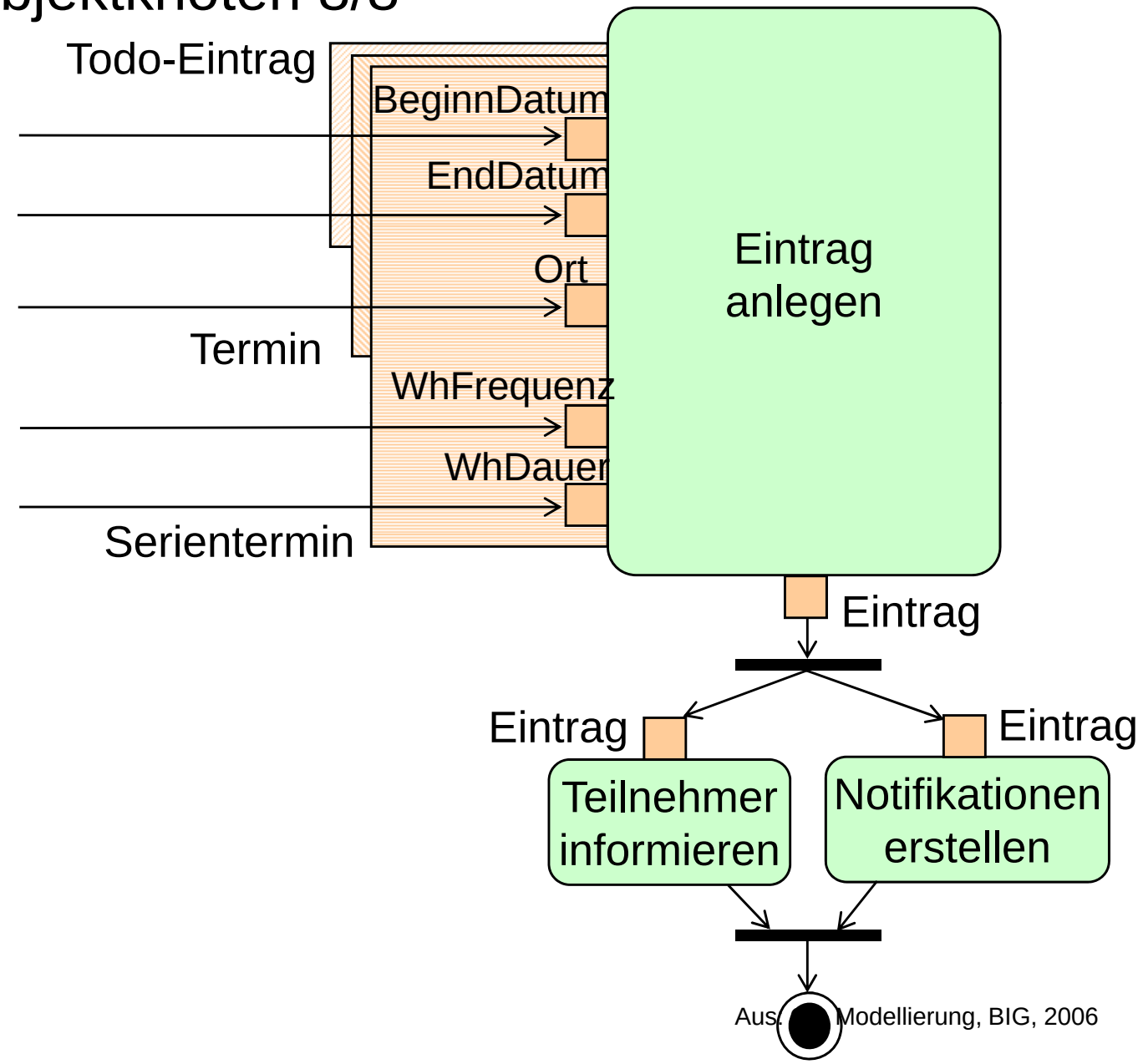


Aus: OO Modellierung, BIG, 2006

Aktivitätsdiagramm

Basiskonzepte – Objektknoten 8/8

- Beispiel für Parametersätze



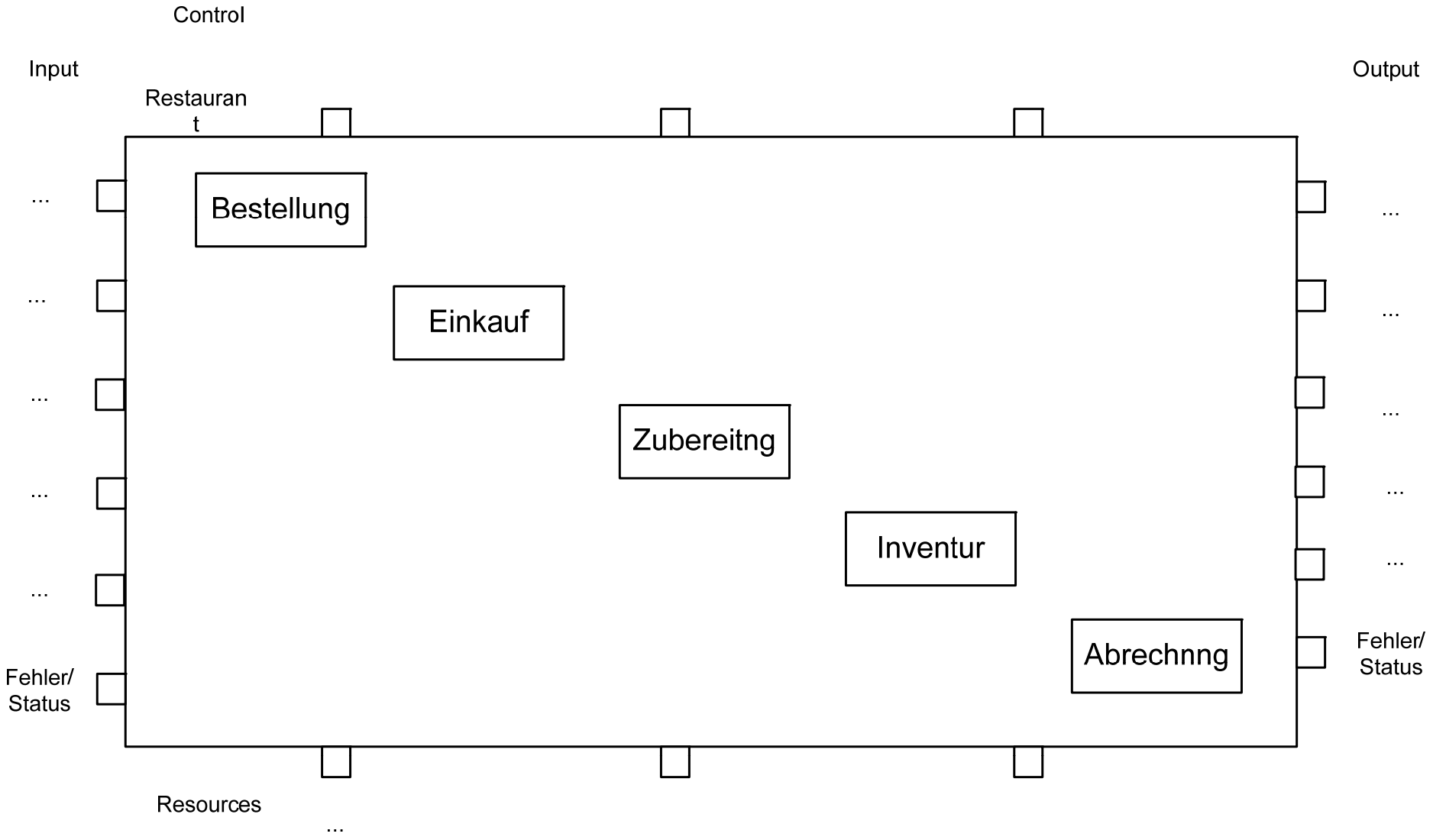
Aus: Modellierung, BIG, 2006

Aktivitätsdiagramm (Business Logic)
Modellieren von Daten- und Kontrollflüssen

- Basierend auf Anwendungsszenario
  - Rollen und Aktivitäten
- Detailgrad von Prozessen
  - System und Teilsysteme: Schnittstellen
  - Teilsystem und Aktionen: Detaillierter Kontrollfluss
- Ableitung von Prozessen
  - Datenfluss: Ein- und Ausgabedaten
  - Kontrollfluss: Logische Darstellung eines Ablaufes
    - Entscheidungen und Schleifen

Beispiel Datenfluss – organisatorische Untereinheiten

- Links: Eingabeparameter
- Rechts: Ausgabeparameter
- Oben: Steuerparameter
- Unten: Ressourcen



Ablauf Aktivitätsdiagramm - Kontrollfluss

- Fokus auf Transformation der Eingangsparameter in Ausgangsparameter
  - Startpunkt(e) identifizieren
  - Aktionen bzw. Sub-Aktivitäten beschreiben
  - Entscheidungspunkte ergänzen
  - Endpunkt(e) identifizieren

  - Optional: Parallele Kontrollflüsse
    (siehe fortgeschrittene Konzepte der OO-Modellierung)

Ablauf Aktivitätsdiagramm - Kontrollfluss

- Überprüfung
  - An Endpunkten müssen Ausgangsparameter gültige Werte haben (entsprechend der Spezifikation der Aktivität)
  - Testfälle entwickeln, die abdecken:
    - Alle Alternativen des Anwendungsszenarios (Use Case) zur Aktivität sind abgebildet
    - Alle Knoten und Kanten
    - Wesentliche Pfade (ev. mehrfaches Besuchen von Knoten notwendig)
    - Variationen der Eingabedaten (Äquivalenzklassen)
    - Variationen der Ausgabedaten (Fälle aus Ergebniskandidaten ableiten)

Beispiel für Testfallbeschreibung
Tische verwalten für Reservierungen

- Testfälle mit erwartetem Ergebnis.

Komponente: Tische
Vorbedingung: Programm gestartet, in die Tischübersicht gewechselt

Nr.	Typ	Kurzbeschreibung	Eingaben	Ergebnis	erw. Ergebnis	Fehlerfallbeschreibung.	ok / nok
14.	FF	Tische; Tisch anlegen	Tischnummer, Tischname	fehlerhaft	Eingabemaske: Tischdaten werden erfaßt und gespeichert	Anlegen von bereits vergebenen Tischnummern.	
15.	NF	Aktualisieren der Tischdaten	Tischdaten werden geändert	fehlerhaft	Infomeldung erfolgreiche Änderung		
16.	NF	Tisch löschen	Tisch auswählen, Löschvorgang starten	Bestätigung / Hinweis	Tisch wird aus der DBgelöscht		
17.	FF	Tisch löschen	unmögliche / doppelte Tischdaten	Bestätigung / Hinweis	Fehlermeldung		
18.	FF	Tisch anlegen	Doppelte / unmögliche Tischdaten	wird angelegt	Fehlermeldung		

Ablauf Testfälle bestimmen

- Ziel aus Testplan bestimmen
  - Z.B. Normal-, Sonder- und Fehlerfälle in einem Anwendungsszenario abdecken
  - Z.B. Alle Knoten und Kanten in einem Kontrollflussgraphen abdecken
- Fälle bestimmen
  - Eingabeparameter
  - Entscheidungen im Ablauf (Wahl der Entscheidung festhalten)
- Erwartetes Ergebnis bestimmen
- Erreichung des Testziels mit Menge der spezifizierten Testfälle überprüfen

Beispiel Testfälle Restaurant.Bestellung

- I. Prüfung Bestellwunsch
  - Äquivalenzklassen der Parameter
  - Mögliche Ergebnisse
- II. Berechnung des Bestellwerts
- III. Prüfung Anzahlung für Großbestellung
  - Prüfung Anzahlung in Ordnung
- Test-Driven Development
  - Ableitung von Testfällen
  - Assertions für Unit Test
    - Programmstück, das Korrektheit der Logik überprüft; Fehlermeldung, falls eine logische Bedingung verletzt wird.
    - Hilft Fehler zu lokalisieren.

Beispiel für Testfallbeschreibung
Tische verwalten für Reservierungen

- Testfälle mit erwartetem und tatsächlichem Ergebnis.

Komponente: Tische
Vorbedingung: Programm gestartet, in die Tischübersicht gewechselt

Nr.	Typ	Kurzbeschreibung	Eingaben	Ergebnis	erw. Ergebnis	Fehlerfallbeschreibung.	ok / nok
14.	FF	Tische; Tisch anlegen	Tischnummer, Tischname	Fehler mit Hinweis	Eingabemaske: Tischdaten werden erfaßt und gespeichert	Anlegen von bereits vergebenen Tischnummern.	NOK
15.	NF	Aktualisieren der Tischdaten	Tischdaten werden geändert	fehlerhaft	Infomeldung erfolgreiche Änderung	TischNr* und Sitzplatzzahl sind vertauscht	N OK
16.	NF	Tisch löschen	Tisch auswählen, Button „löschen“	Bestätigung / Hinweis	Tisch wird aus der DBgelöscht		OK
17.	FF	Tisch löschen	unmögliche / doppelte Tischdaten	Bestätigung / Hinweis	Fehlermeldung	Begründung falls Tisch nicht gelöscht werden kann; zB Reservierung aktiv.	NOK
18.	FF	Tisch anlegen	Doppelte / unmögliche Tischdaten	wird angelegt	Fehlermeldung	Tischnummer darf nicht mehrfach vergeben werden	NOK

Zustandsdiagramm
Einführung

- Ein Zustandsdiagramm (State Machine Diagram) beschreibt die möglichen **Folgen von Zuständen** eines **Modell-elements**, i.A. eines Objekts einer bestimmten Klasse
  - Während seines **Lebenslaufs** (Erzeugung bis Destruktion)
  - Während der **Ausführung** einer **Operation** oder **Interaktion**

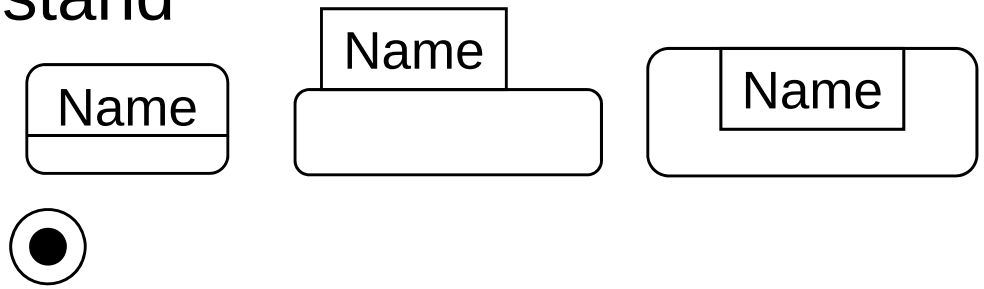
- Modelliert werden
  - Die **Zustände**, in denen sich die Objekte einer Klasse befinden können
  - Die möglichen **Zustandsübergänge** (Transitionen) von einem Zustand zum anderen
  - Die **Ereignisse**, die Transitionen auslösen
  - **Aktivitäten**, die in Zuständen bzw. im Zuge von Transitionen ausgeführt werden

Aus: OO Modellierung, BIG, 2006

Zustandsdiagramm

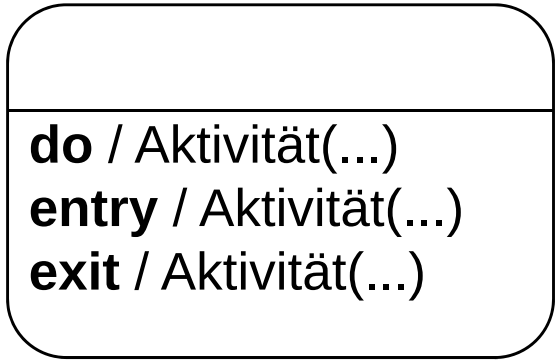
Basiskonzepte – Zustand

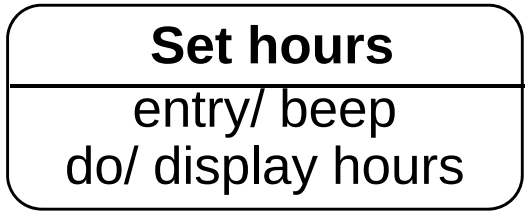
- Zustand (state)
  - Zustand i.e.S.
  - Endzustand



- Pseudozustände (weil transient)
  - Initialzustand
  - History-Zustand, Synch-Zustand, Gabelung, Vereinigung, etc.

- **Aktivität** innerhalb eines Zustands
  - **entry** / *aktivität*
    Aktivität wird beim Eingang in den Zustand ausgeführt
  - **exit** / *aktivität*
    Aktivität wird beim Verlassen des Zustands ausgeführt
  - **do** / *aktivität*
    Aktivität wird ausgeführt, Parameter sind erlaubt
  - ***event*** / *aktivität*
    Aktivität behandelt Ereignis innerhalb des Zustands





Aus: OO Modellierung, BIG, 2006

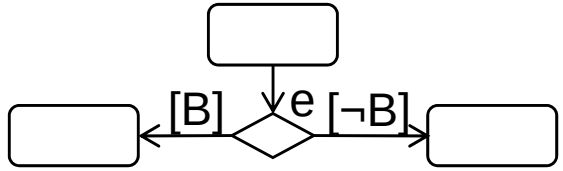
Zustandsdiagramm

Basiskonzepte – Zustandsübergang

- Ein Zustandsübergang (state transition) erfolgt, wenn
  - das **Ereignis** eintritt – eine evt. noch andauernde **Aktivität** im Vorzustand wird **unterbrochen**!
  - und die **Bedingung** (guard) erfüllt ist – bei Nicht-Erfüllung geht das nicht »konsumierte« Ereignis **verloren**
- Durch entsprechende Bedingungen können **Entscheidungsbäume** modelliert werden
- Standardannahmen
  - **Fehlendes Ereignis** entspricht dem Ereignis »Aktivität ist abgeschlossen«
  - **Fehlende Bedingung** entspricht der Bedingung [true]
- **Aktionen** auf Zustandsübergängen möglich
  - Spezielle Aktion: Nachricht an anderes Objekt senden

    send empfänger.nachricht()
  - Beispiel:

    right-mouse-down (loc) [loc in window]
    / obj:= pick-obj (loc); send obj.highlight()

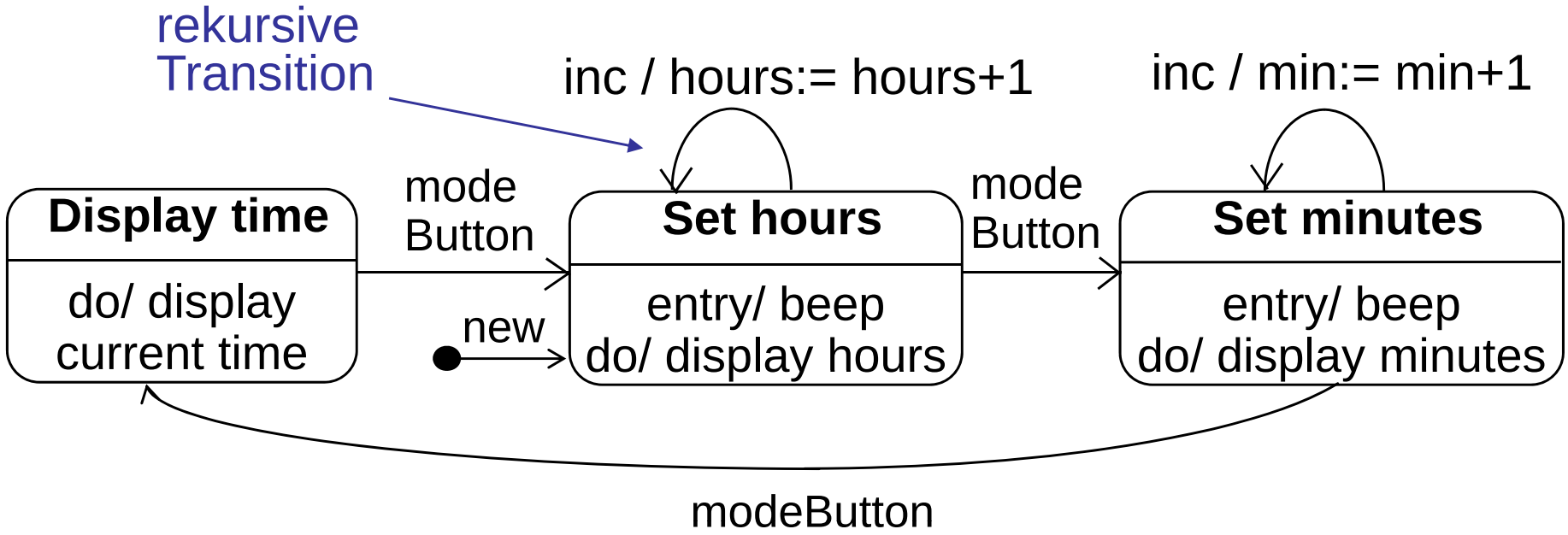


Aus: OO Modellierung, BIG, 2006

Zustandsdiagramm

Basiskonzepte – Beispiel: Klasse DigitalWatch

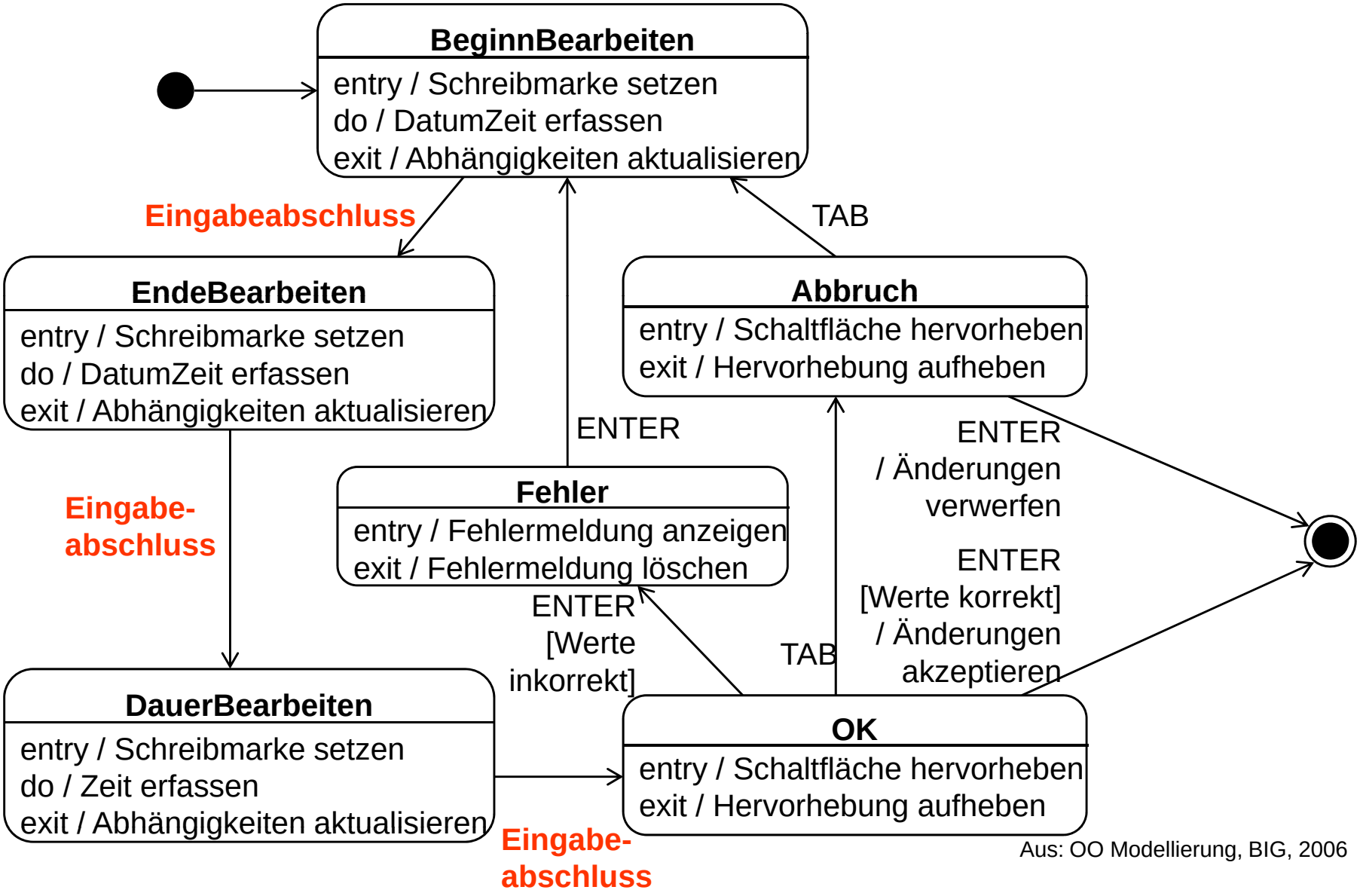
DigitalWatch
min : Integer = 0 hours : Integer = 0
modeButton() inc()



Aus: OO Modellierung, BIG, 2006

Zustandsdiagramm

Basiskonzepte – Beispiel: Lebenslauf einer Termin-Eingabemaske 3/3

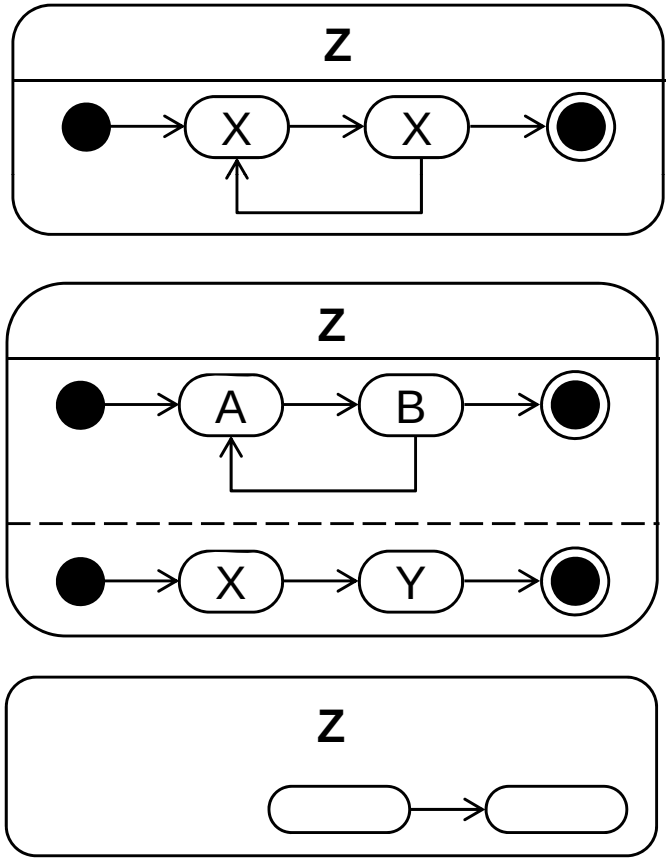


Aus: OO Modellierung, BIG, 2006

Zustandsdiagramm

Strukturierung – ODER- vs. UND-Verfeinerung

- Verfeinerung eines **komplexen Zustands (composite state)** — geschachteltes Zustandsdiagramm
- **ODER-Verfeinerung**
  - Disjunkte Sub-Zustände, d.h. **genau ein Subzustand ist aktiv**, wenn der komplexe Zustand aktiv ist
- **UND-Verfeinerung**
  - Nebenläufige Sub-Zustände, d.h. **alle Subzustände sind aktiv**, wenn der Superzustand aktiv ist
  - Die Subzustände werden i.A. ihrerseits Oder-verfeinert
- Hinweis auf **ausgeblendete Verfeinerung**
  - Diese kann an anderer Stelle dargestellt werden

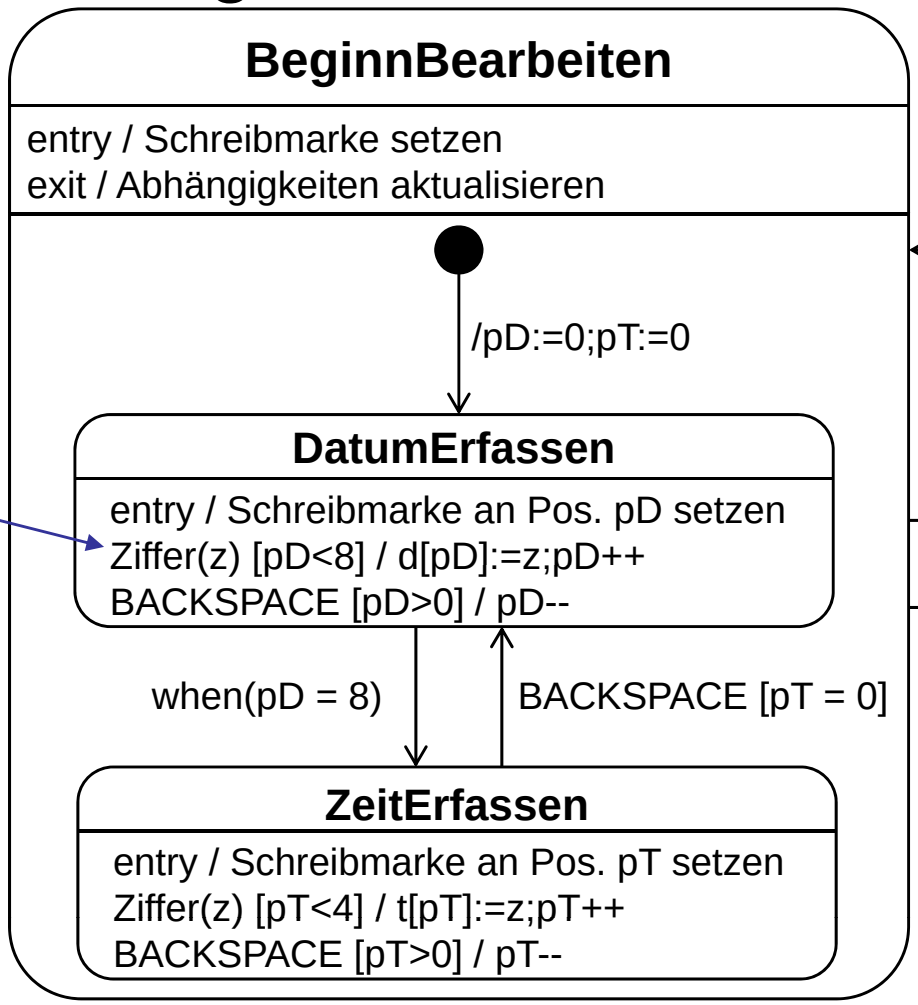


Aus: OO Modellierung, BIG, 2006

Zustandsdiagramm

Strukturierung – Beispiel 1/6

- Lebenslauf einer Termin-Eingabemaske
- **ODER-Verfeinerung** der Aktivität »DatumZeit erfassen«



Parametrisiertes Ereignis

Gruppen-Transitionen (nur ausgehende!) werden an alle Subzustände »distribuiert«

Aus: OO Modellierung, BIG, 2006

Ablauf Zustandsübergangsdiagramm

- Entitäten und deren Zustände identifizieren
- Parallele Zustände aufteilen

- Für jede Entität
  - Initialzustand bestimmen
- Zustandsübergänge einzeichnen
    - Ereignis(se) bestimmen, die den Übergang anstoßen
  - Wo sinnvoll: Bedingungen für Übergänge bestimmen

- Überprüfung
  - Abläufe des Anwendungsszenarios durchspielen
  - Testfälle durchspielen
  - Überdeckung: Alle Knoten und Kanten abdecken.

Zusammenfassung

- Test-Driven Development als Basis für zielorientierte Software-Entwicklung (Implementierung und QS)
- Brauchbare Modellierung unterstützt das Herstellen effektiver und effizienter Testfälle
  - Datenmodellierung (Datenbank)
  - Datenfluss und Kontrollfluss (Business Logic)
  - Zustandsübergangsmodelle (z.B. GUI)
- Nächste Einheit zu diesem Themenbereich: SE Prozesse im Überblick

Referenzen

Bücher und Skripten

- „Datenmodellierung“; Unterlagen zur Vorlesung; TU Wien, Inst. f. DB & AI, 2006.
- Kemper A., Eickler A.; „Datenbanksysteme“; 6. Auflage, Oldenbourg, 2007.
- „Objektorientierte Modellierung“; Unterlagen zur Vorlesung; Business Informatics Group (BIG), TU Wien, 2006.
- Sommerville I.; „Software Engineering“; 8. Auflage; Addison Wesley, 2007 (v.a. Kapitel 2, 3, 8, 14, 16, 19, 22).
- Booch, Rumbaugh, Jacobson: "The UML User Guide", 2. Auflage, Addison Wesley, 2005

Web resources

- SEPM Tuwel: https://tuwel.tuwien.ac.at/course/view.php?id=33
- Sommerville book companion website: www.pearsoned.co.uk/sommerville
- Integrated DEFinition Methods: http://www.idef.com/idef0.html

Welche Studienrichtung studieren Sie?

Was könnten mögliche Nutzen von Modellen im Software Engineering sein?

Michael List

Grobe Orientierung vom Fortschritt des Projekts

Dennis Gurnhofer

Genau kommunikation mit anderen

Anton Hößl

Kosten reduzieren

Simon Maximilian Fraiss

Einfach lesbare Spezifikation von Problemlösungen

Peter Kittenberger

Bessere Verständlichkeit von Abläufen

Werner Schober

Besserer Überblick über den Projektfortschritt

Markus Lehr

Einfacher Überblick über komplexe und / oder große Projekte.

Lukas Felician Krasel

Anhalte- & Orientierungspunkt während der Implementiertung

Simon Reisinger

Programmiersprachenunabhängig

Andreas Wiesinger

Ermoeglicht einfaches Verstaendnis des gesamten Projekts und der Zusammenhaenge der einzelnen Komponenten

Michael Pointner

Auch Software-Entwickler spielen gerne mit Flieger-Modellen.

Paul Stelzhammer

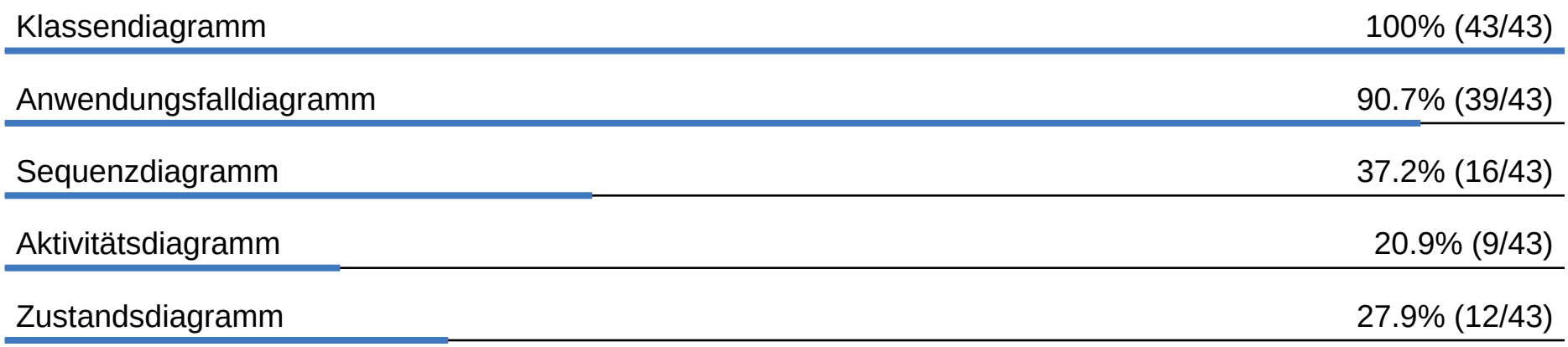
Darstellung der Annahmen und Abstrahierung

Martin Robl

Ein bild sagt mehr als 1000 worte

Welche der nachfolgenden UML Diagrammtypen können Sie lesen und verstehen?

Haben sie die nachfolgenden UML Diagrammtypen schon angewendet um selbst Software zu entwickeln?



Welche weiteren Fehler kennen sie aus eigener Erfahrung?

Lukas Felician Krasel — Votes: + 18 / - 1

Vorschnelles Implementieren, "Drauf los Programmieren"

Markus Lehr — Votes: + 16 / - 1

Gruppenmitglieder haben unterschiedliche Vorstellungen vom Resultat und kommunizieren diese nicht.

Markus Buchta — Votes: + 8 / - 0

Exceptions ignorieren (sieht der benutzer eh nicht)

Simon Maximilian Fraiss — Votes: + 7 / - 0

Vergessen auf Null zu prüfen -> Exception

Martin Robl — Votes: + 6 / - 0

Unpräzise spezifikation, schlechtes zeitmanagement,

Bruno Tiefengraber — Votes: + 6 / - 0

Keine einheitlichen Namenkonventionen/Coding Style (lesbarkeit)
"Gewachsenes" Design

Andreas Wiesinger — Votes: + 5 / - 0

Project nicht configurable implementiern: Aenderungen ziehen sich durch das gesamte Projekt und sind sehr aufwaendig.
Variablen hardcoden, statt sinnvolle dynamische Datenstrukturen wie Hashmaps

Alfred Mincinoiu — Votes: + 13 / - 8

Tests werden am Ende geschrieben #YOLO

Robert Glanz — Votes: + 3 / - 0

Falsche Testfälle

Wolfgang Gundacker — Votes: + 3 / - 0

falsche variable verwenden und sich wundern, dass es nicht passt

Viktor Vidovic — Votes: + 4 / - 1

Viel zu wenig refactoring

Timo Hinterleitner — Votes: + 2 / - 0

Anforderungen und Environmentdetails werden erst nach und nach bekannt

Michael Pointner — Votes: + 2 / - 0

Deadlines falsch gesetzt

Simon Reisinger — Votes: + 2 / - 0

Es wird direkt mit der Codierung angefangen.

Niklas Roth — Votes: + 2 / - 0

Keine Architekturübersciht am Anfang und dann sehr zersplitterter Code und unsaubere Programmierung

Michael Lang — Votes: + 1 / - 0

Architektur nicht im vorhinein festgelegt

Peter Kittenberger — Votes: + 1 / - 3

Tools nicht ausgenutzt (git, Intellij)
Zu spät begonnen/Zu wenig Zeit freigehalten
Immer wieder fehlende Javadocs

Ergänzen sie die Attribute der Entitäten.

Bitte antworten Sie mit [Entität]: [Attribute] (z.B. Kunde: Name,...)

Michael Krejci — Votes: + 5 / - 0

Rechnung: Endsumme

Simon Reisinger — Votes: + 4 / - 1

Rezept: notwendiges Werkzeug

Simon Maximilian Fraiss — Votes: + 3 / - 0

Speise: Bezeichnung

Lukas Naske — Votes: + 3 / - 0

Rezept:Zeitaufwand

Markus Buchta — Votes: + 4 / - 1

Rechnung: bezahlt

David Banyasz — Votes: + 2 / - 0

Lager: verdorbene Menge

Anton Hößl — Votes: + 2 / - 0

Rezept: RezeptID

Niklas Roth — Votes: + 2 / - 0

Bestellung: Bestellnummer

Thomas Hader — Votes: + 2 / - 0

Bestellung: mitNehmen

Maximilian Irlinger — Votes: + 4 / - 2

Speise: Verkaufspreis

Dennis Gurnhofer Votes: + 2 / - 0

Bestellung: Datum

Alfred Mincinoiu Votes: + 2 / - 0

Kunde: Kontostand

Andreas Wiesinger Votes: + 0 / - 0

Zutat: Menge

Fabian Wurm Votes: + 0 / - 0

Bestellung: Anzahlung

Michael List Votes: + 0 / - 0

Bestellung: istAnzahlungGeleistet

Peter Holzer Votes: + 0 / - 0

Speisen: bestellbar

Alexander Schwarz Votes: + 0 / - 1

Rechnung: Koch

Ievgenii Gruzdev Votes: + 0 / - 1

Kunde: Adresse

Rainer Laschober Votes: + 0 / - 2

Menü: Anzahl Gänge

Sarah El-Sherbiny Votes: + 0 / - 4

Rechnung: Endsumme, Koch, Kunde

Werner Schober — Votes: + 0 / - 5

Kunde: Kontostand

Timo Hinterleitner — Votes: + 0 / - 8

Rezept: Anzahl der Portionen
Rezept: Relation zu Rezept
Rezept: Relation zu Zutat mit Attribut Menge

Markus Lehr — Votes: + 0 / - 9

Speise: _Bezeichnung_, Preis, bestellbar
Menü: Preis, _Bezeichnung_, Von, Bis

Robert Glanz — Votes: + 0 / - 10

Speise: Verkaufspreis

Peter Kittenberger — Votes: + 1 / - 12

Rezept: RezeptID, Zeitaufwand, Portionen, Werkzeuge

Ergänzen sie die Input- und Outputparameter.

Bitte antworten Sie mit [Input/Output]: [Parameter] (z.B. Input: Zutaten,...)

Thomas Hader — Votes: + 7 / - 0

Input: Tagesplan

Niklas Roth — Votes: + 6 / - 0

Input: Kundenbestellung

Viktor Vidovic — Votes: + 5 / - 0

Output: Rechnung verschicken

Alfred Mincinoiu — Votes: + 3 / - 0

Input: Einnahmen

Timo Hinterleitner — Votes: + 3 / - 1

Input: Zutaten
Output: Speise

Werner Schober — Votes: + 2 / - 0

Input: Einkaufsliste

David Banyasz — Votes: + 1 / - 0

Output: Journal updaten

Andreas Wiesinger — Votes: + 3 / - 3

Input: Geld Output: Abfall

Michael List — Votes: + 0 / - 0

Output: Bestellnummer

Rainer Laschober — Votes: + 3 / - 4

Output: Super Essen

Dominique Grieshofer — Votes: + 0 / - 2

Input: Bestellungen + Kundendaten Output: Rechnung + Bilanz

Sarah El-Sherbiny — Votes: + 0 / - 4

Abrechnung:
Input: Belege
Output: Endsumme