

# Vorlesung "Software-Engineering"

---

Prof. Ralf Möller, TUHH, Arbeitsbereich STS

## ■ Vorige Vorlesungen

### ■ Planung

- | Lastenheft (requirements definition document)
- | Verfahren zur Aufwandsschätzung

## ■ Heute

### ■ Definition

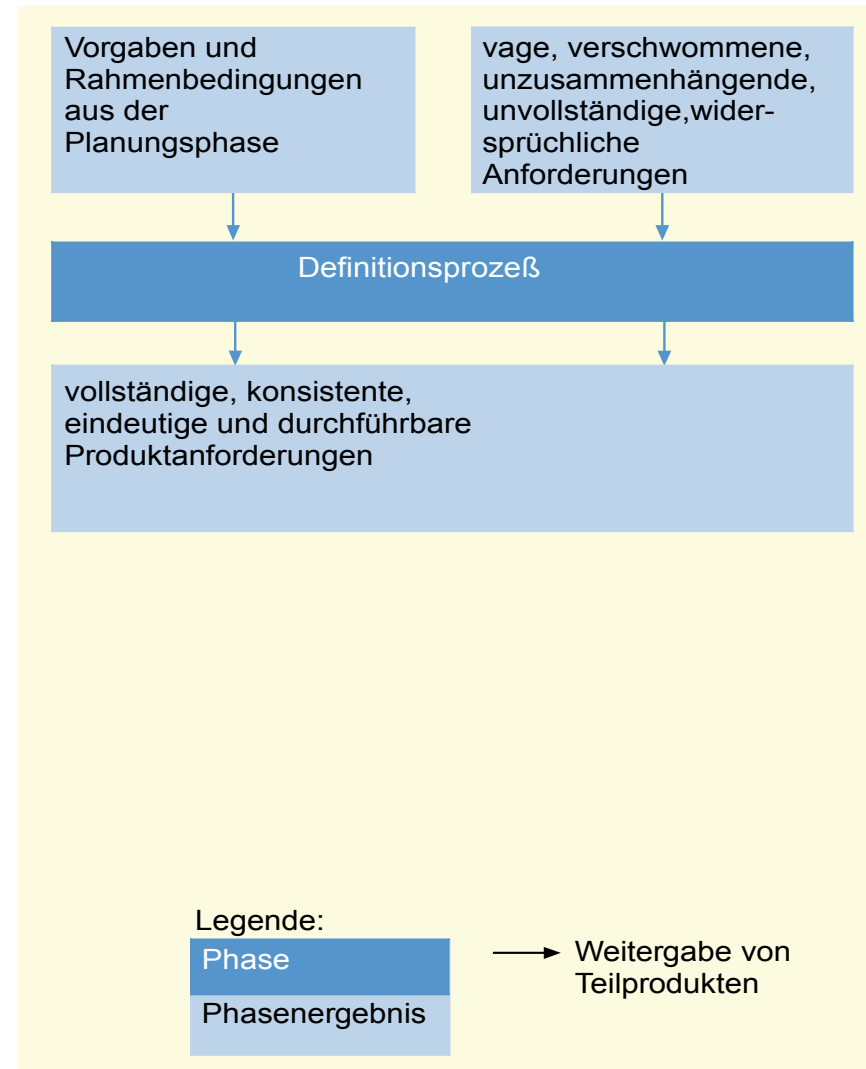
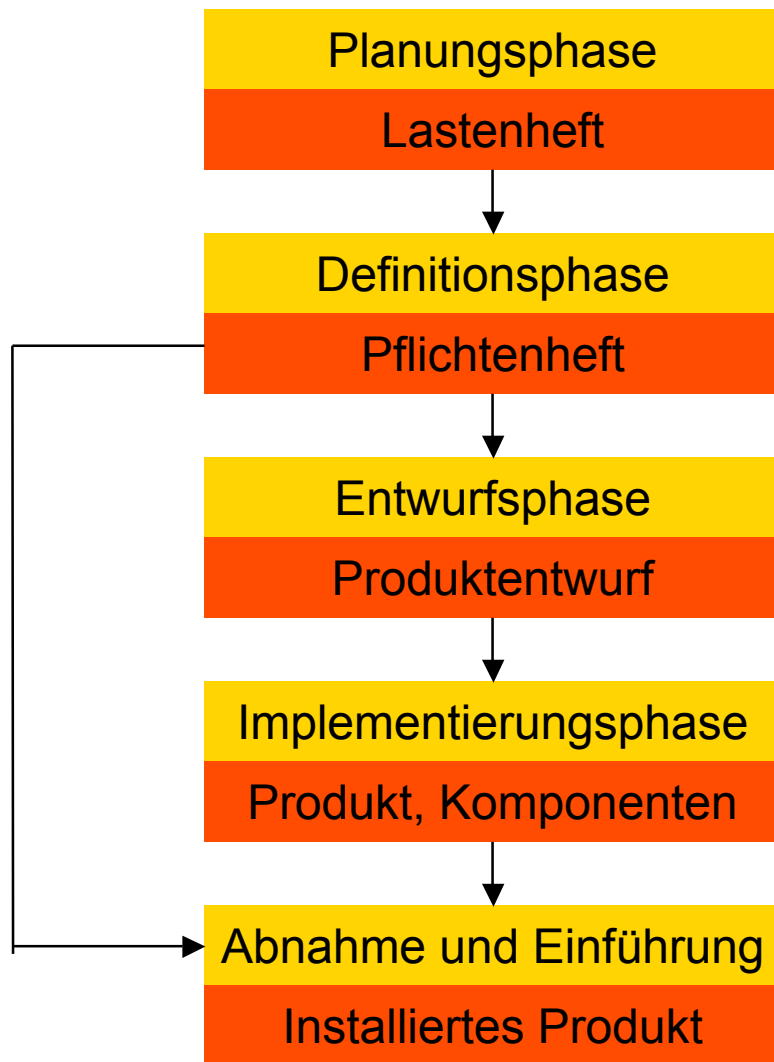
- | Pflichtenheft (requirements specification document)
- | Detaillierte Anforderungsanalyse (detailed requirements engineering)

# Gegenstand der Vorlesung

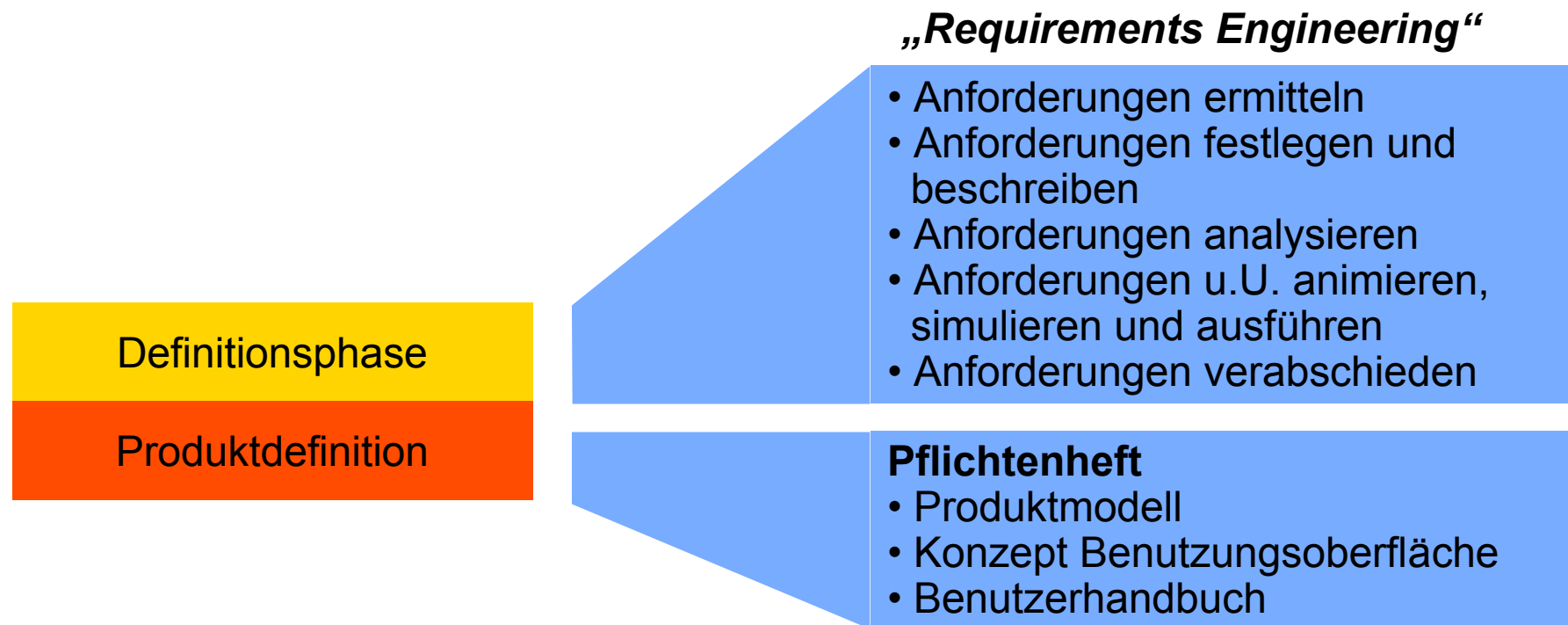
---

- Unter dem Namen SW-Engineering werden Methoden und Prinzipien zur Lösung von „böartigen Problemen“ (Rittel 73) diskutiert
- Probleme sind „böartig“,
  - wenn sie in so viele Einzelteile verstrickt sind, daß es keine endgültige Spezifikation des Problems gibt.
  - wenn sich das wahre Gesicht erst bei der Entwicklung der Lösung zeit
  - wenn sich Anforderungen erst bei bzw. nach der Implementierung ergeben

# Systemanalyse: Einordnung



# Systemanalyse: Aktivitäten und Ergebnisse



- **Anforderungen (*requirements*)** : Legen die qualitativen und quantitativen Eigenschaften eines Produkts aus der Sicht des Auftraggebers fest.
- **Systemanalyse (*requirements engineering*)**: Systematische Vorgehensweise, um die Anforderungen in einem iterativen Prozeß zu ermitteln.

# Produktdefinition

---

## ■ Ziel: Projektvertrag

- = Produktdefinition (Pflichtenheft, Produktmodell, Konzept für Benutzungsoberfläche, Benutzerhandbuch)
- + Zahlungsbedingungen
- + Garantie
- + Projektplan
- + ...

Juristischer Vertrag!

➔ Überprüfung der erbrachten Leistung

# Aufbau eines Pflichtenheftes

---

- **Aufgabe:**  
Zusammenfassung aller **fachlichen** Anforderungen
- **Adressaten:**  
Auftraggeber,  
Auftragnehmer (Manager, Entwickler, Tester, Warter)
- **Inhalt:**  
fachlicher Funktions-, Daten- und Leistungsumfang, Qualitätsanforderungen  
*Was, nicht Wie*  
Basis des juristischen Vertrages!
- **Form:**  
Standardisiertes, gegliedertes Schema,  
meistens: textuell  
**Eventuell:** Beschreibung durch Modelle
- **Sprache:**  
detaillierte verbale Beschreibung (falls textuell), für Auftraggeber lesbar!
- **Zeitpunkt:**  
direkt nach Planungsphase
- **Umfang:** ausführlich, vollständig

# Aufbau und Inhalt eines Pflichtenheftes (1)

---

## 1. Zielbestimmung

- Muß-Kriterien: Was muß das Produkt leisten?
- Wunschkriterien: Was soll das Produkt evtl. noch leisten?
- Abgrenzungskriterien: Was soll das Produkt **nicht** leisten?

## 2. Produkt-Einsatz

- definiert Anwendungsbereiche (wofür), Zielgruppen (für wen) und Betriebsbedingungen (z.B. physikalische Umgebung, Betriebszeiten, Bediener nötig etc.)

## 3. Produkt-Umgebung

- Software, Hardware, Orgware, Produkt-Schnittstellen (Betriebssystem, Rechner, Peripherie, organisatorische Voraussetzungen (z.B. Mail-Anschluß), Integration in bestehende Anwendungen)

# Aufbau und Inhalt eines Pflichtenheftes (2)

---

## 4. Produkt-Funktionen (funktionale Anforderungen)

- ! definiert abstrakte Funktionalität aus Benutzersicht
- ! textuelles, num. Gliderungsschema (z.B. /F10/ oder /F20/W für Wunschkriterium)

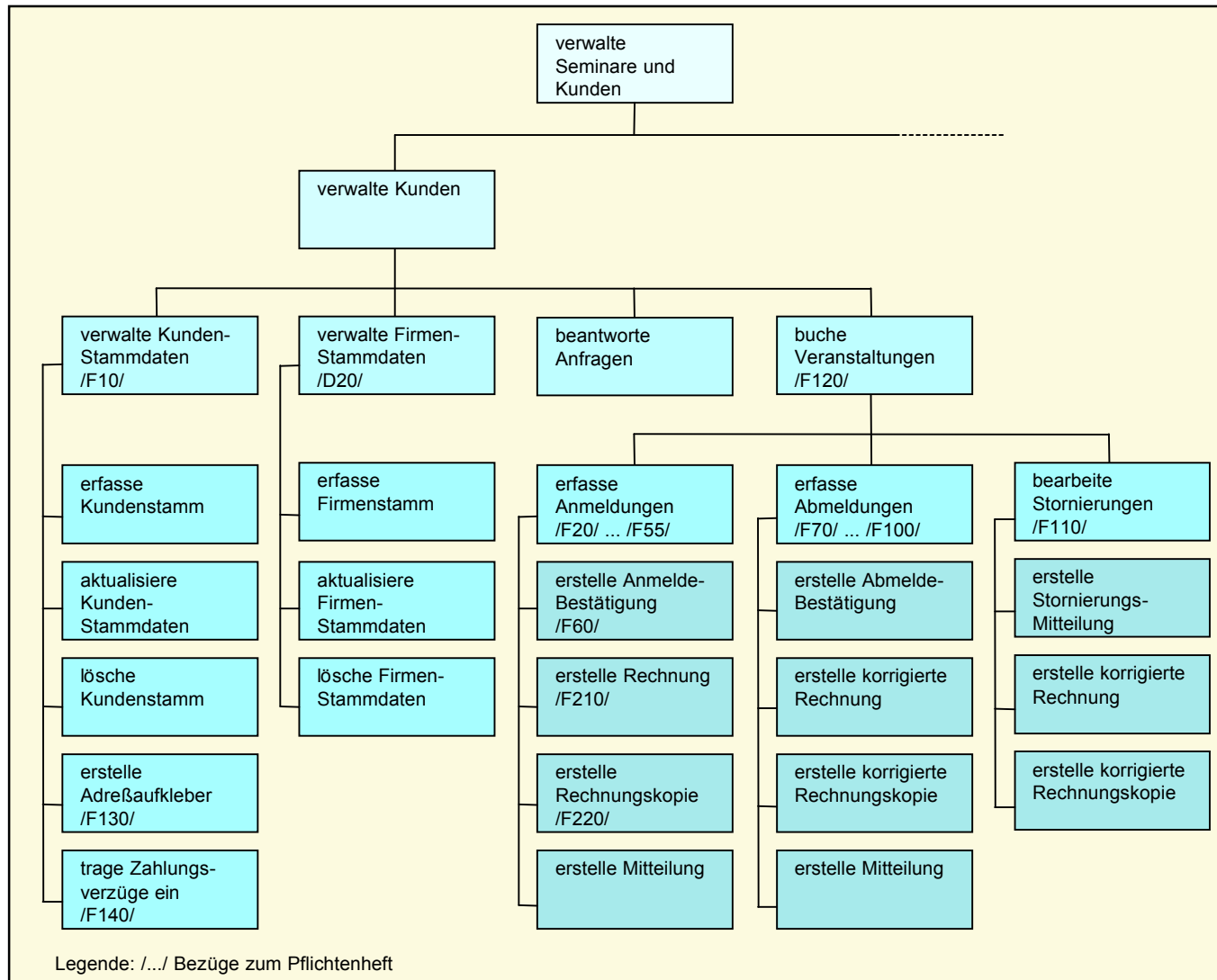
U.U.: Funktionsbäume, Zustandsautomaten, Petrinetze, Klassendiagramme, Interaktionsdiagramme

## 5. Produkt-Daten

- ! Beschreibung langfristig zu speichernder Daten aus Benutzersicht
- ! textuell, num. Gliderungsschema (z.B. /D10/)

U.U.: ER-Modelle, XML-Schemata, Klassendiagramme

# Verfeinerung der Funktionen (Beispiel)



# Aufbau und Inhalt eines Pflichtenheftes (3)

---

## **6. Produkt-Leistungen** (nicht-funktionale Anforderungen)

- Zeit, Umfang, Benutzerzahl, Genauigkeit, ...

## **7. Benutzungsoberfläche** (falls gewünscht)

## **8. Qualitäts-Zielbestimmung**

(nicht-funktionale Anforderungen)

## **9. Globale Testszenarien/Testfälle**

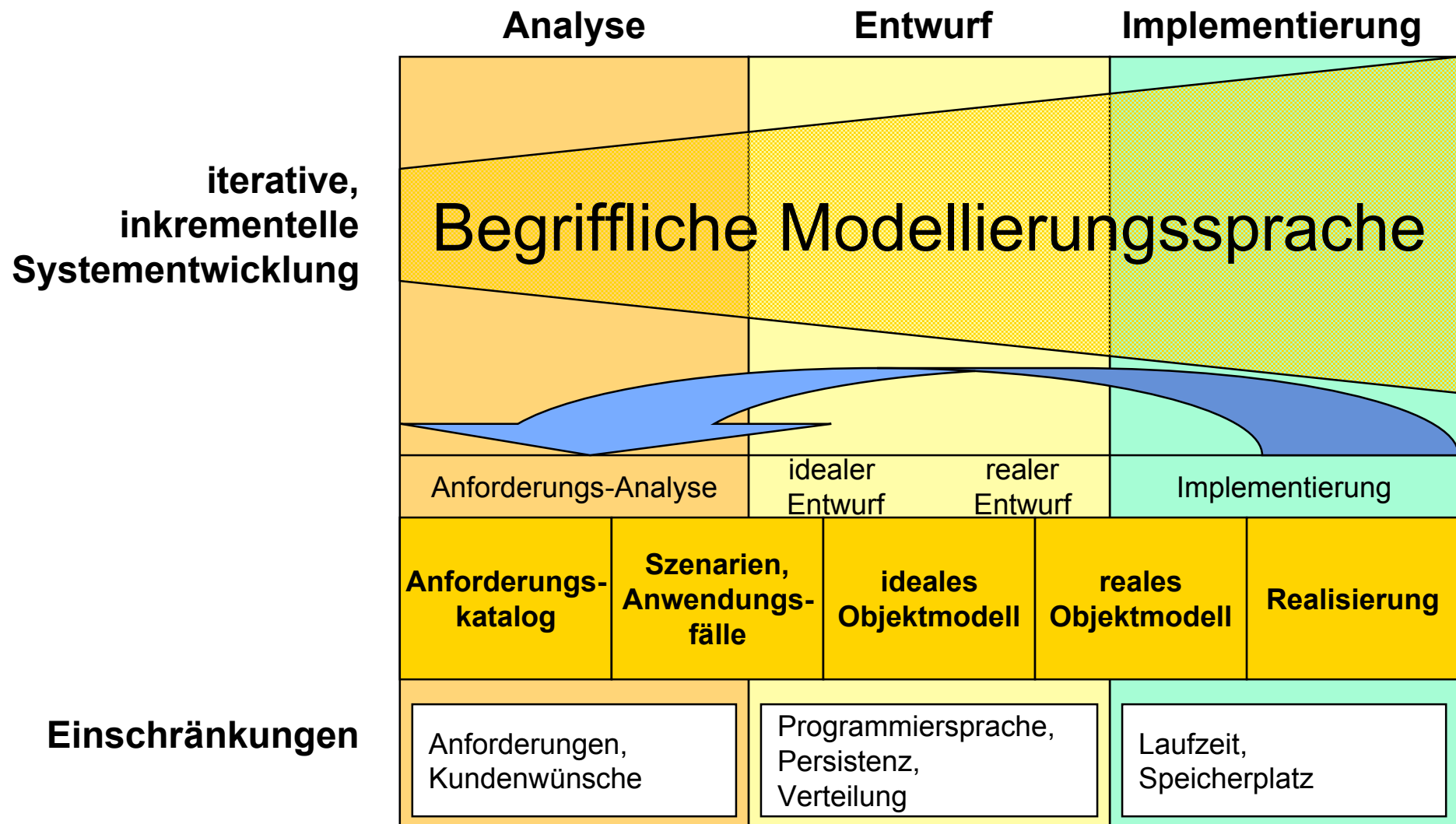
- Abnahmetest

## **10. Entwicklungsumgebung**

## **11. Ergänzungen**

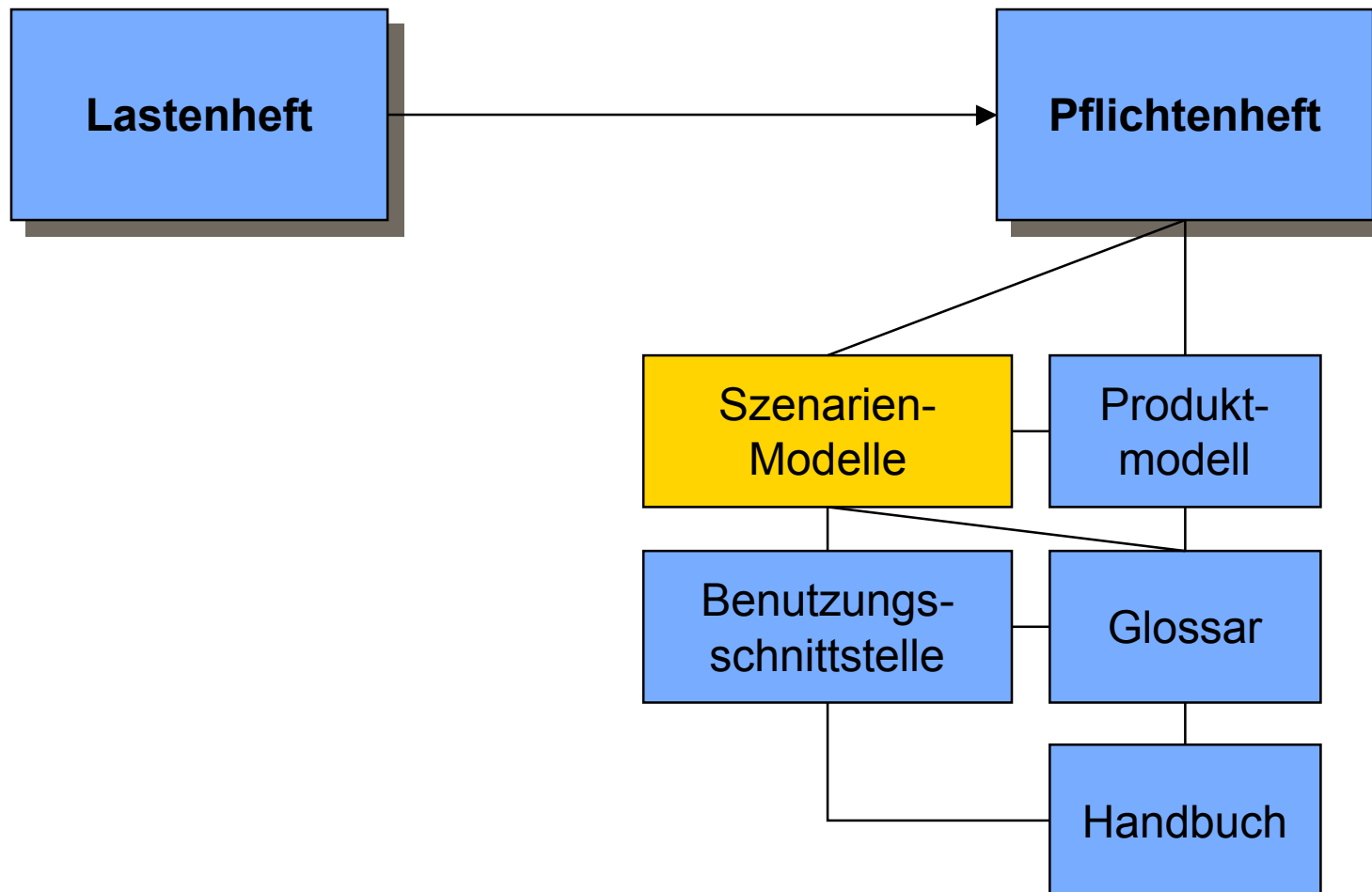
## **12. Glossar, Begriffslexikon**

# Der SW-Entwicklungsprozeß



# Systemanalyse: Einsatzszenarien

---



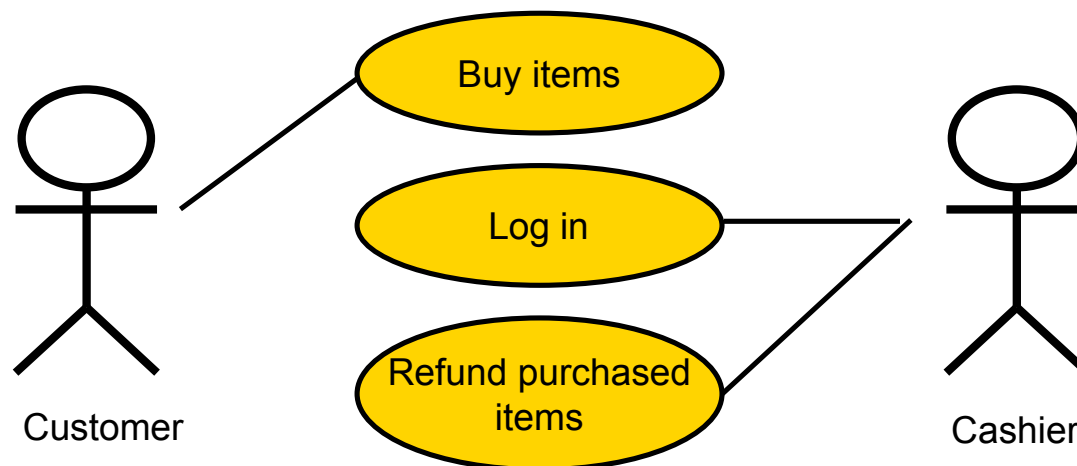
# Szenarien

---

- Ereignisszenarien:  
Beschreibung von Ereignissen, die eine gewisse Systemfunktionalität auslösen, und deren Interaktion
- Beschreibung von Anwendungsfällen (use cases):  
Beschreibung der Funktionalität aus der Benutzungsschnittstellensicht (für externe Akteure)
- Ethnographische Analyse: Einbettung eines Systems in den Arbeitskontext (vgl. Ist-Analyse)
  - Für Anforderungen, die sich daraus ergeben, wie Menschen wirklich arbeiten, und nicht wie sie arbeiten sollen nicht wie sie sagen wie sie arbeiten (sollen)
- ...

# Anwendungsfallbeispiel: *Point of Sale Terminal* (1)

- Ein *Point of Sale Terminal* (POST) ist ein computergestütztes System, um Verkäufe, Bezahlungen und Auszahlungen in einem Handelsgeschäft zu unterstützen.
- Es enthält Hardwarekomponenten (Computer, Anzeige, Balkencode-Lesegerät, ...) und Softwarekomponenten.



# Beispiel: *Point of Sale Terminal* (2)

## ■ Anwendungsfallbeschreibung

### Use Case: **Buy item**

1. This use case begins when a customer arrives at a POST checkout with items to purchase.
2. The cashier records the universal product code (UPC) from each item.
3. The System determines the item price and adds the item to the running sales transaction.
4. If there is more than one of the same item, the cashier can enter the quantity as well.
5. The description and the price of the current item are displayed.
6. ...

# Begriffliche Analyse

---

- In Funktionsbeschreibungen, Datenbeschreibungen oder Szenarien tauchen Begriffe und Beziehungen auf
- **Begriffe** beschreiben gleichstrukturierte Objekte einer Anwendung
  - Wir gehen von einer Grundmenge von unterscheidbaren Objekten aus
- Die Struktur von Objekten ergibt sich durch **Attribute** mit jeweiligen Wertebereichen
  - Wir gehen von verschiedenen Wertebereichen (Zahlen, Zeichenketten, usw.) zur Strukturbeschreibung aus
- **Assoziationen** beschreiben Beziehungen zwischen einzelnen Objekten (etwas ungenau wird auch zwischen Beziehungen zwischen Begriffen gesprochen)

# Methodisches Vorgehen: Finde Begriffe <sup>(1)</sup>

---

## 1. Identifiziere Begriffe über die „Substantivmethode“

- Relevante Dokumente: Lastenheft, Anwendungsfalldokumentation, Glossar, Formulare, technische Systembeschreibungen, Gesprächsnotizen, ...
- Identifiziere **Substantive** (Begriffe, Konzepte), *die für das Problemverständnis relevant sind*
  - Technik: Unterstreichen im Text, lieber zunächst zu viele Substantive als zu wenige
- Identifiziere **Synonyme, Akronyme**.
  - Zwei verschiedene Namen sind Synonyme, falls sie die gleiche Menge von Objekten bezeichnen
  - Ein Akronym (Initialwort) ist ein Name, der aus den Anfangsbuchstaben eines Namens gebildet wird
- Identifiziere **Homonyme**.
  - Ein Homonym ist ein *Name*, der zwei (verschiedene) Mengen von Objekten benennt
- Optional: Identifiziere **Oberbegriffe / Unterbegriffe**
  - Ein Begriff ist ein Oberbegriff (Unterbegriff), wenn er einen Obermenge (Untermenge) von Objekten bezeichnet

# Methodisches Vorgehen: Finde Begriffe (2)

---

## 2. Kategorisiere die Substantive

- Konkrete Objekte bzw. Dinge, z.B. Pkw
- Personen und deren Rollen, z.B. Kunde, Mitarbeiter, Dozent
- Informationen über Aktionen, z.B. Banküberweisung
- Orte, z.B. Wartezimmer
- Organisationen, z.B. Filiale
- Schnittstellen des Systems, z.B. Liste, Auswahlliste
- Beziehungen zwischen Begriffen, z.B. Mietvertrag zwischen Kunde und Mietobjekt
- Allgemeine und fachbezogene Informationen, z.B. Artikel, Seminartyp

# Methodisches Vorgehen: Finde Begriffe (3)

---

## 3. Streiche Substantive, die keine eigenständigen Begriffe bezeichnen

- | Das Ziel ist **nicht**, möglichst viele Begriffe oder Begriffe möglichst geringer Komplexität zu identifizieren.
- | Gestrichene Substantive können evtl. in einer **To-Do-Liste** gesammelt werden, da sie später bei der Identifikation relevanter Attribute, Operationen und Beziehungen hilfreich sind.
- | Streiche Namen für **Rollen**, die ein Begriff in einer Beziehung zu einem anderen Begriff spielt.
- | **Indizien** für zu streichende Substantive:
  - | Es lassen sich weder Attribute noch Funktionen identifizieren, die sich auf die Begriffe beziehen.
  - | Das Substantiv modelliert Implementierungsdetails.
  - | Die Begriffe ist nur durch Funktionen gekennzeichnet, die sich anderen Begriffen zuordnen lassen.

# Methodisches Vorgehen: Finde Begriffe (4)

---

## 4. Wähle aussagefähige und verbindliche Namen

- | Jeder Name für einen Begriff soll
  - | ein Substantiv im Singular sein
  - | so konkret wie möglich gewählt werden
  - | kein Homonym sein
  - | nur in seltenen Fällen ein Akronym sein

## 5. Dokumentiere jeden Begriff knapp

- | Ein definierender Satz, z.B.  
Eine *Seminarbuchung* ist ein Dokument, das die Anmeldung eines Kunden zu einer Seminarveranstaltung dokumentiert.
- | Liste der Synonyme, eventuell Abgrenzung zu anderen Begriffen  
**Synonyme:** Anmeldung, Reservierung

# Methodisches Vorgehen: Finde Attribute

---

## 1. Identifiziere Attribute im Anschluss an die Substantivmethode

- | Identifiziere für das Pflichtenheft relevante Attribute eines jeden Begriffes
- | Erfasse Attribute nur dann, wenn sie **fachlich** notwendig sind

## 2. Überprüfe den Attributnamen

- | Jeder Attributname soll
  - | ein Substantiv im Singular oder Plural sein
  - | so konkret wie möglich gewählt werden
  - | kein Homonym sein

## 3. Definiere den Typ der Attribute

- | Wähle einen der angenommenen Basiswertebereiche
- | Der Typ kann auch unspezifiziert oder nur vage spezifiziert bleiben
- | Komplex strukturierte Typen sollten u.U. durch eigenständige Begriffe ersetzt werden.

## Beispiel: *Point of Sale Terminal* (2)

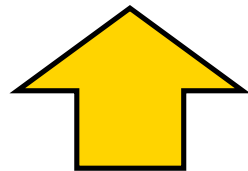
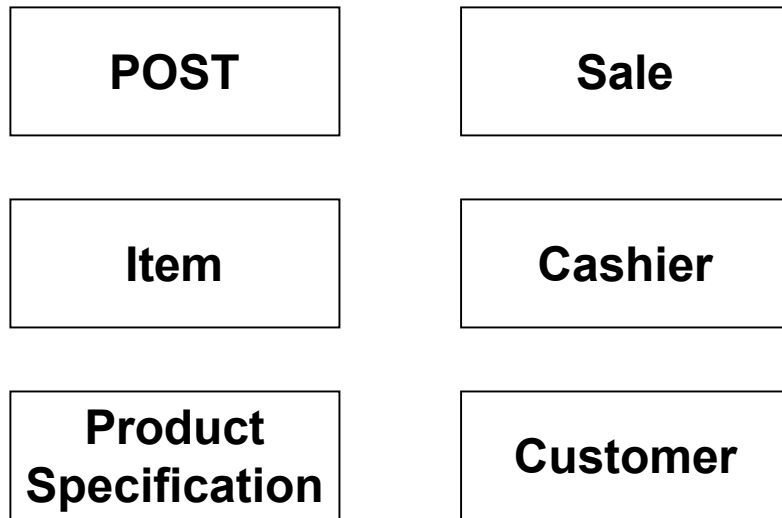
### ■ Anwendungsfallbeschreibung

#### Use Case: **Buy item**

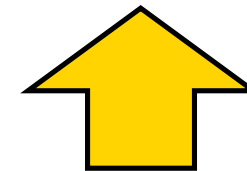
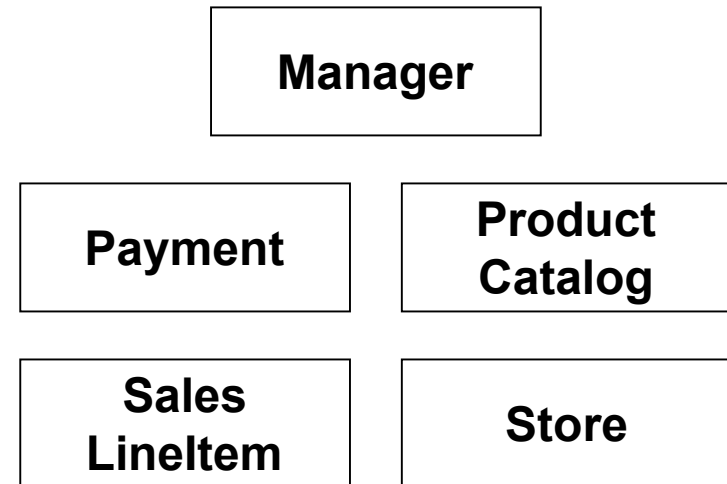
1. This use case begins when a customer arrives at a **POST checkout** with items to purchase.
2. The cashier records the universal product code (UPC) from each item.
3. The System determines the item price and adds the item to the running sales transaction.
4. If there is more than one of the same item, the cashier can enter the quantity as well.
5. The description and the price of the current item are displayed.
6. ...

# Beispiel: *Point of Sale Terminal* (3)

---



Anfänglich identifizierte Begriffe



Iterativ identifizierte Begriffe

# Assoziationen zwischen Begriffen

---

- Assoziationen beschreiben Beziehungen zwischen Elementen aus Objektmengen, die durch Begriffe beschrieben sind (assoziierte Begriffe)
  - Assoziierte Begriffe treten in verschiedenen Rollen auf
  - Beispiel: Vorlesung setzt Vorlesung voraus
    - Assoziation: Voraussetzung bzw. voraussetzen
    - Rollen: Vor, Nach
- Teil-Ganzes-Assoziationen
  - Aggregation
    - Teile existieren auch ohne das Ganze
    - Teile können in mehreren Teil-Ganzes-Beziehungen involviert sein
  - Komposition
    - Teile existieren nicht ohne das Ganze
    - Pro Teil nur ein Ganzes zugeordnet

# Methodisches Vorgehen: Finde Assoziationen (1)

---

## 1. Identifiziere mögliche Assoziationen zwischen Objekten

- Suche in den relevanten Dokumenten nach **Verben** und nach **Substantiven**, die Aktionen oder Prozesse in der Problembeschreibung identifizieren
- Identifiziere für jede Assoziation die beteiligten Begriffe
- Bevorzuge binäre Assoziationen (d.h. mit zwei beteiligten Begriffen)

## 2. Kategorisiere diese Assoziationen

- räumliche Nähe (in der Nähe von)
- Aktionen (fährt, bucht)
- Kommunikation (redet mit)
- Besitz (hat)
- allgemeine Beziehungen (ist abhängig von, ist verheiratet mit)

# Methodisches Vorgehen: Finde Assoziationen (2)

---

## 3. Streiche Assoziationen, die keine Assoziationen sind, die sich auf bestehende Begriffe beziehen

- Streiche nicht-permanente Beziehungen
- Streiche für das Pflichtenheft irrelevante Assoziationen
- Streiche implementierungsbezogene Assoziationen

## 4. Falls erforderlich, definiere Assoziations- und Rollennamen

- Assoziationen sind in der Mehrzahl der Fälle durch die partizipierenden Begriffe eindeutig bestimmt. In diesem Fall sind Assoziations- und Rollennamen nicht erforderlich.
- Ansonsten (z.B. bei rekursiven Assoziationen) definiere
  - einen **Namen** (ein Verb oder eine Verbalphrase) für die Assoziation
  - einen **Rollennamen** für jeden an der Assoziation beteiligte Begriff
  - einen **Satz**, der die Semantik der Assoziation beschreibt.

# Methodisches Vorgehen: Finde Assoziationen (3)

## 5. Bestimme die Kardinalität jeder Rolle jeder Assoziation

### ! Liegt eine **Muss-Beziehung** vor?

⇒ Kardinalität 1..

- | Sobald das Objekt erzeugt ist, muss auch die Beziehung zu dem anderen Objekt aufgebaut werden.

### ! Liegt eine **Kann-Beziehung** vor?

⇒ Kardinalität 0..

- | Die Beziehung kann zu einem beliebigen Zeitpunkt nach dem Erzeugen des Objekts erzeugt werden.

### ! Ist die Obergrenze fest oder variabel?

⇒ Kardinalität ..k

- | Ist eine Obergrenze vom Problem her **zwingend** vorgegeben?
- | Im Zweifelsfall immer mit **variablen** Obergrenzen arbeiten!

### ! Gelten besondere Bedingungen?

- | Gerade Anzahl, Untergrenze, ...

**Bemerkung:** Bei einer festen Kardinalität  $k$  (z.B. 2 als Unter- und Obergrenze) ist zu prüfen, ob nicht  $k$  separate Assoziationen zu verwenden sind.

# Methodisches Vorgehen: Finde Assoziationen (4)

---

## 5. Unterscheide Assoziation und Aggregation anhand der folgenden Kann-Kriterien:

- Lässt sich die Beziehung durch „besteht aus“ oder „ist Teil von“ beschreiben?  
(Kollektion, Behälter, Ganzes & Teile, Gruppe & Mitglied)
- Ist die Kardinalität auf einer Seite der Assoziation 1 oder 0..1 ?
- Ist die Assoziation transitiv und asymmetrisch?
- Gehören die Komponenten in ein Subsystem?
- Erfolgt der Zugriff auf die Teilobjekte ausschließlich über das Aggregat-Objekt?
- Ist die Lebensdauer der Komponente durch die Lebensdauer des Aggregats beschränkt?

# Methodisches Vorgehen: Finde Assoziationen (5)

---

## 6. Entscheide, ob ein Schnappschuss oder eine Historie zu modellieren ist

- **Schnappschuss:** Wer ist augenblicklich Chef der Abteilung?
- **Historie:** Wer war zum Zeitpunkt  $t$  Chef der Abteilung?

## 7. Dokumentiere Restriktionen auf der Assoziation

- Ordnung auf den Beziehungen {ordered}
- Abgeleitete Beziehungen {derived as ...}
- Konsistenzbeziehungen im Bezug auf andere Beziehungen

## Bemerkung

- In dieser Phase der Analyse werden **Generalisierungen** u.U. noch nicht beachtet.

# Modellierung von Homomorphismen

- **Homomorphismen** treten in der Praxis häufig auf, z.B.
  - Dienstleistung entspricht Dienstleistungsangebot

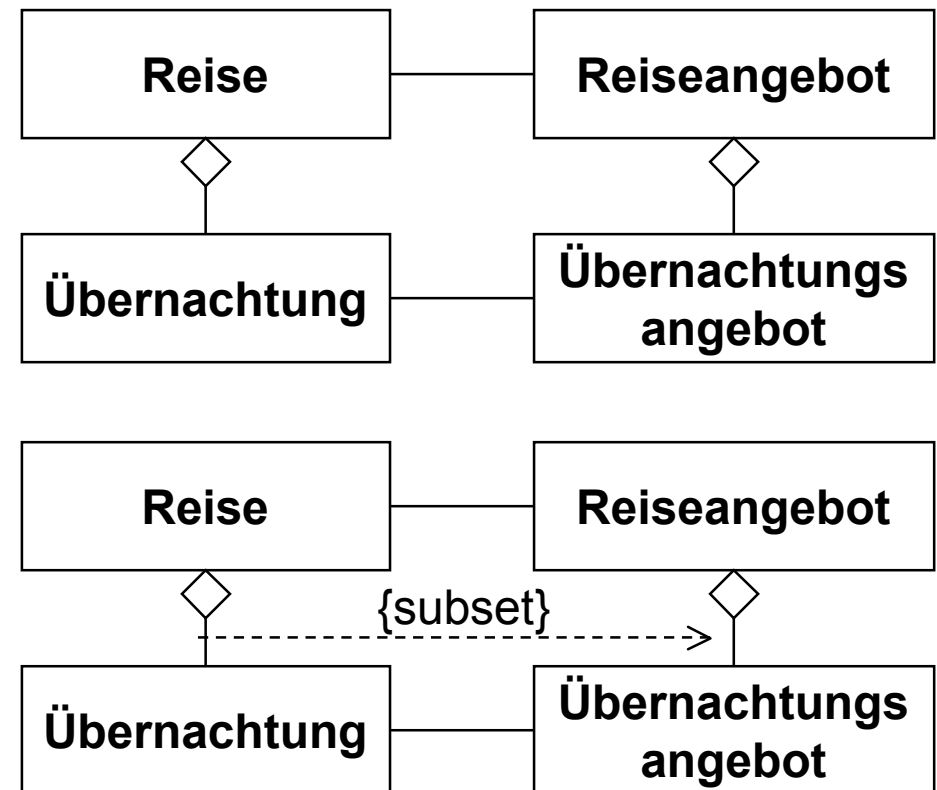
- **Modellierungsalternative A**

- Komplexe Restriktionen auf Begriffen

- {for all  $\ddot{u}$  in  $r$ .Übernachtung:  
some  $\ddot{u}a$  in  
 $r$ .Reiseangebot.Übernachtungsangebot  
( $\ddot{u}a = \ddot{u}$ .Übernachtungsangebot)}

- **Modellierungsalternative B**

- Beziehung zwischen Assoziationen mit zugehöriger Restriktion.



# Verwendung der Begriffe: Anforderungsformulierung

---

- Validierung von Anforderungen  
(Anforderungsreviews)
- Anforderungsmanagement
  - Anforderungen zwangsläufig unvollständig  
(„böartige Probleme“)
  - Dauerhafte und veränderliche Anforderungen
  - Planung des Anforderungsmanagements
    - Nachvollziehbarkeitsinformationen
  - Management von Anforderungsänderungen
    - Änderungsanalyse und Aufwandsschätzung

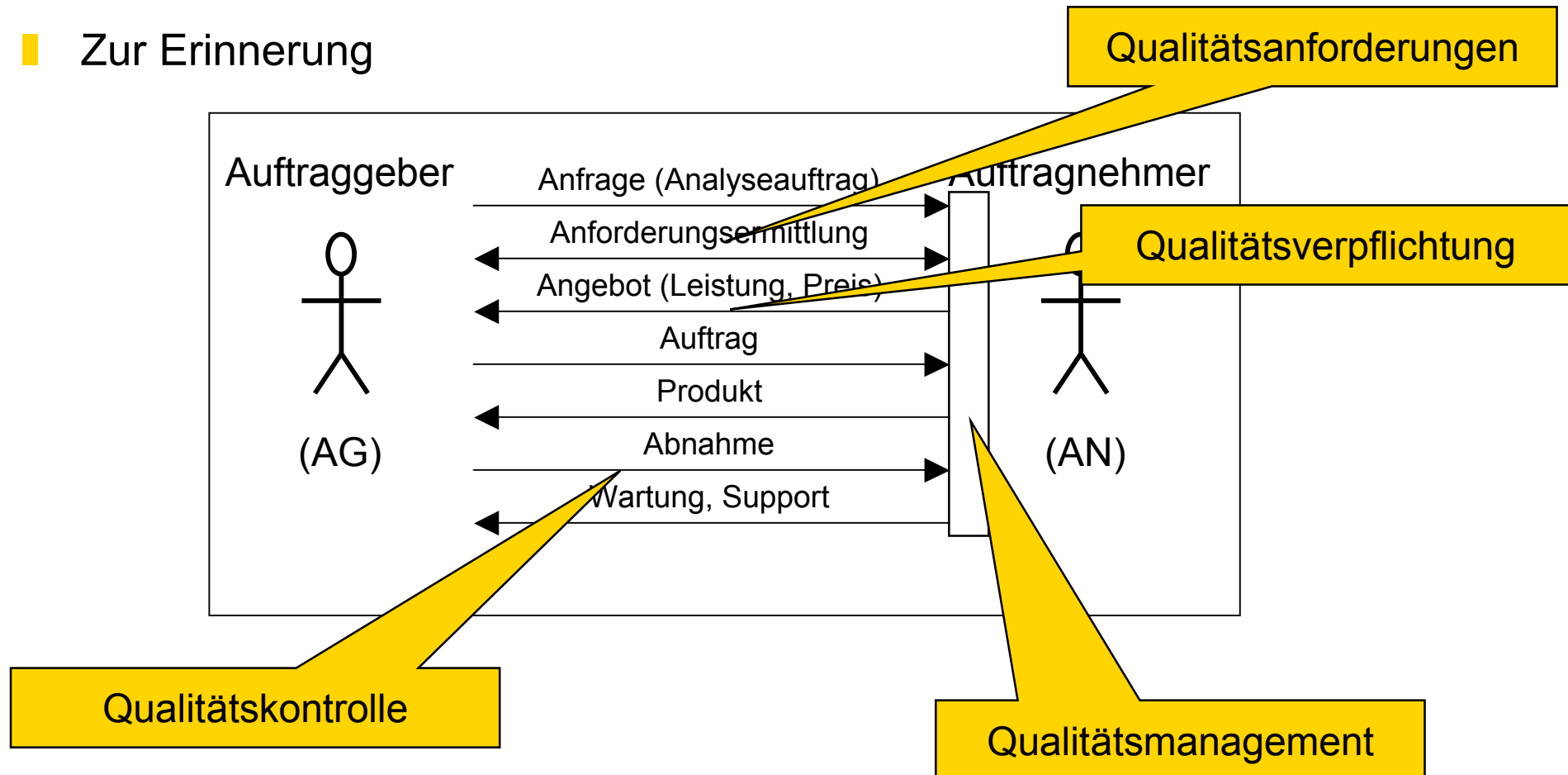
# Pflichtenheft

---

- Noch fehlend
  - Qualitätsangaben
  - Angaben zum Test (Abnahmekriterien)

# Qualität bei Individualsoftware

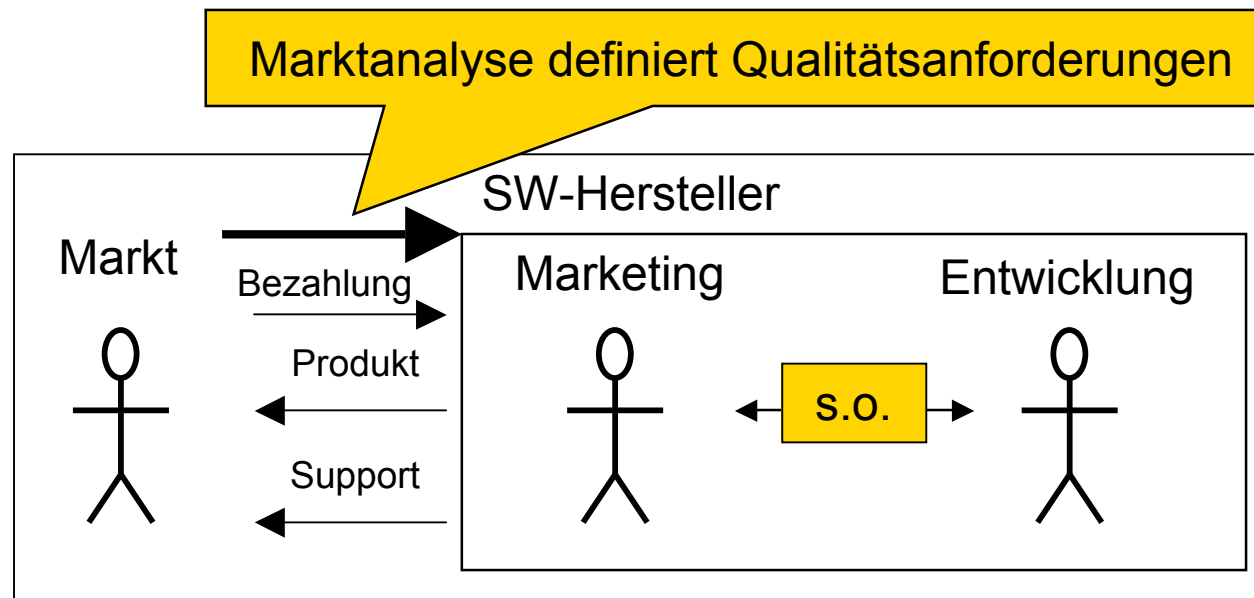
- Zur Erinnerung



- Ziel des **Qualitätsmanagements** beim Auftragnehmer ist die Erreichung von Qualitätsanforderungen in der gesamten Lebensdauer der Software

# Qualität bei Standardsoftware

## ■ Zur Erinnerung: Kapitel 2

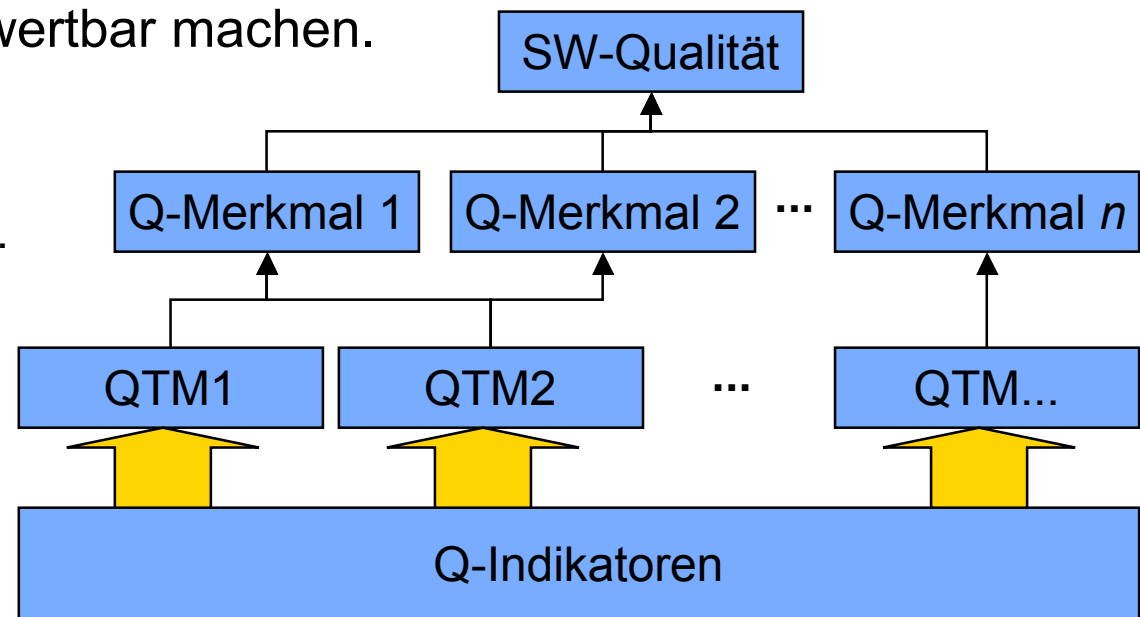


# Softwarequalität: Das FCM-Modell

## ■ Factors-Criteria-Metrics-Modell:

- **Quality Factors** beschreiben benutzerorientierte Qualitätsmerkmale (QMs) der Software.
- **Quality Criteria** sind Teilmerkmale (QTM), die Qualitätsmerkmale verfeinern.
- **Quality Metrics** sind Qualitätsindikatoren / Qualitätsmetriken, die Teilmerkmale meß- und bewertbar machen.

- Ein Teilmerkmal kann Einfluß auf mehrere Qualitätsmerkmale haben.
- Ein **Software-Qualitätsmaß** ist eine Metrik und eine Methode zur Berechnung der Softwarequalität aus den Indikatoren.

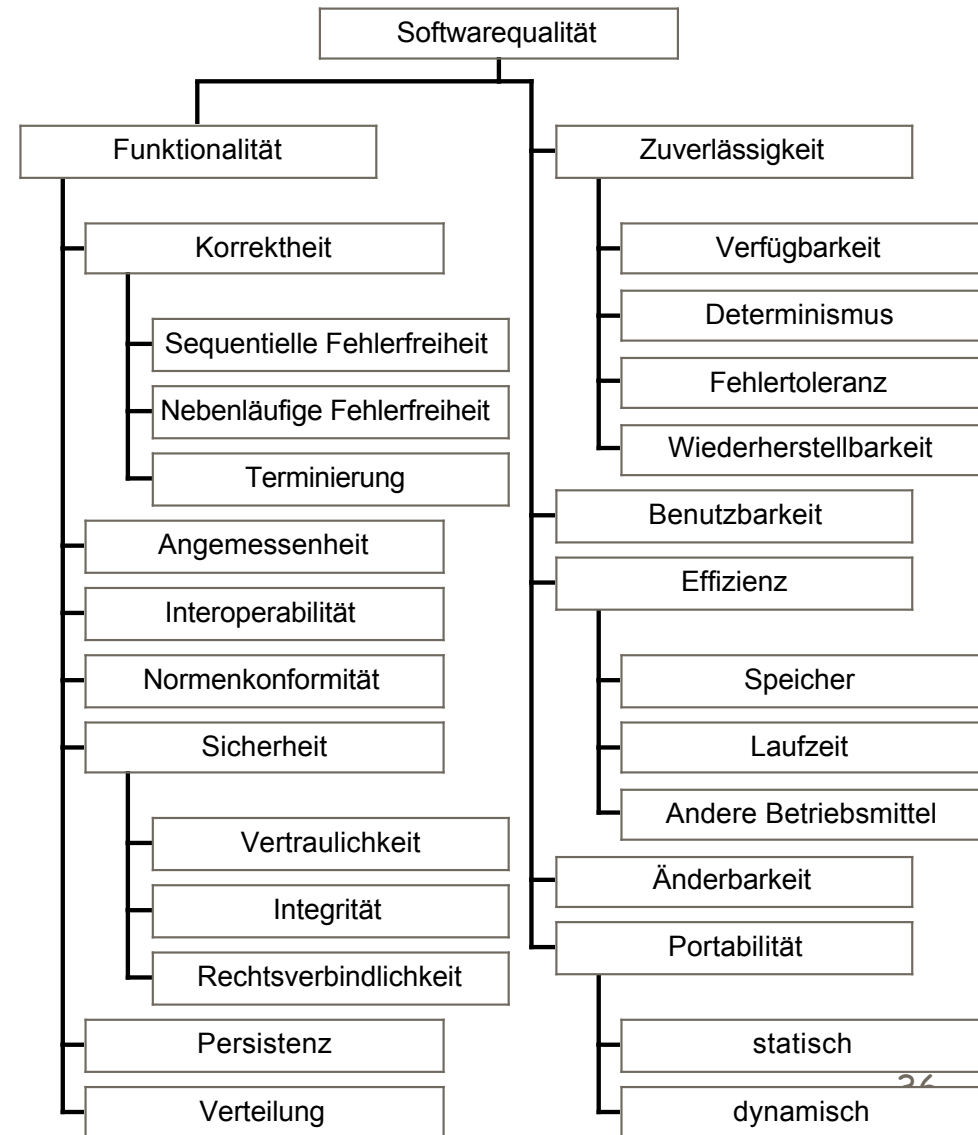


# Softwarequalitätsmerkmale nach DIN und ISO

- **DIN ISO 9126** definiert sechs Qualitätsmerkmale zur Beschreibung der Softwarequalität, die weitgehend disjunkte Teilmerkmale besitzen:

- Funktionalität
- Zuverlässigkeit
- Benutzbarkeit
- Effizienz
- Änderbarkeit
- Portabilität

- Funktionale Anforderungen und nicht-funktionale Anforderungen meist vermischt



# Metriken und ihre Eigenschaften (1)

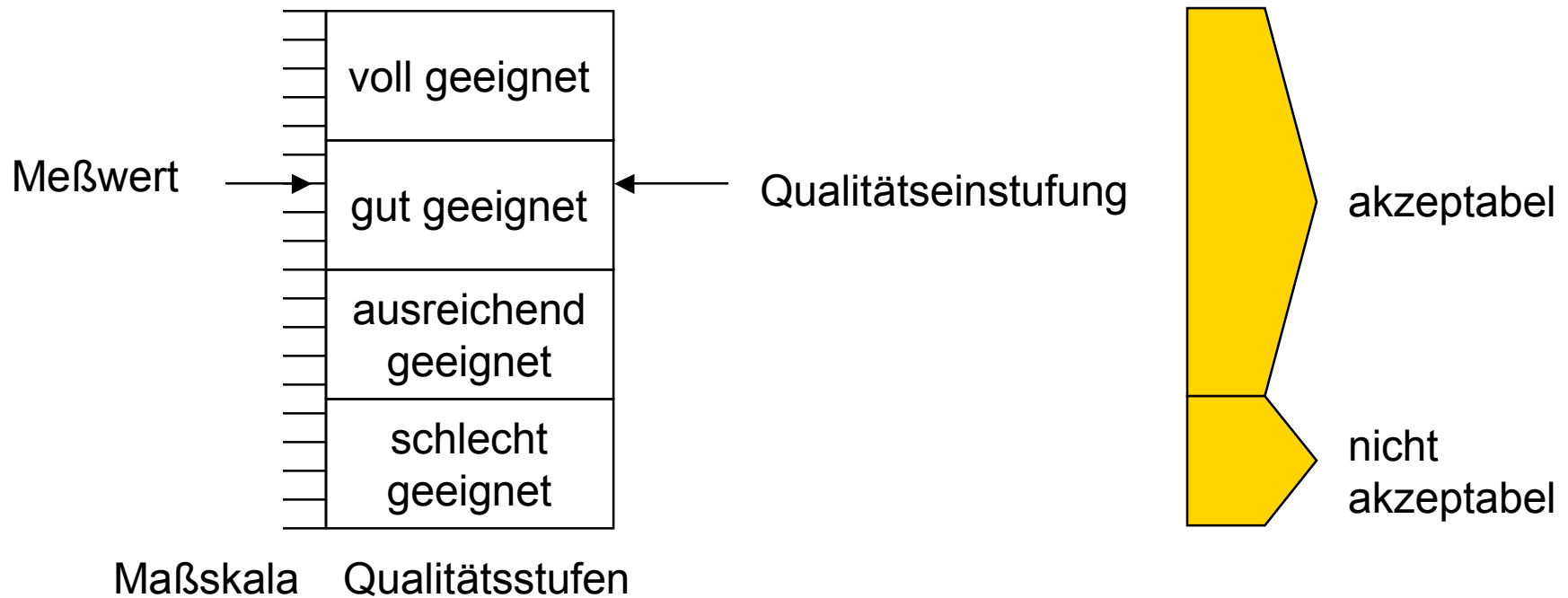
- Eine **Metrik** (ein Maßsystem) ist eine Algebra, die meßbare Eigenschaften durch **Zahlen** (Maßzahlen) ausdrückt.
- Metriken lassen sich anhand der für die Maßzahlen gültigen algebraischen Regeln (**Skalen**) klassifizieren:
  - Eine **Nominalskala** definiert eine endliche, ungeordnete Menge von diskreten Merkmalsausprägungen.  
*Beispiel:* Grundfarben.  
*Operationen:* Gleichheitstest
  - Eine **Ordinalskala** ist eine Nominalskala erweitert um eine vollständige Ordnung, und ist daher isomorph zu einer monoton steigenden Folge natürlicher Zahlen. *Beispiele:* Windstärken, Hubraumklassen.  
*Zusätzliche Operationen:* Ordnungstest, Median, Rang, Rangkorrelationskoeffizient.
- Eine **Intervallskala** ist eine Ordinalskala erweitert um ein Abstandsmaß.  
*Beispiele:* Schulnoten.  
*Zusätzliche Operationen:* Arithmetisches Mittel und Standardabweichung.

Minimalanforderung an SW-Qualitätsmaß

# Metriken und ihre Eigenschaften (2)

- Eine **Rationalskala** verwendet als Maßzahlen reelle Zahlen und eine Maßeinheit. Dabei kann ein absoluter oder natürlicher Nullpunkt vorhanden sein. *Beispiele:* Preise, Längen, Volumina, Zeiten.  
*Zusätzliche Operationen:* Quotientenbildung.
- Eine **Absolutskala** verwendet als Maßzahlen reelle Zahlen. Eine Skalenverschiebung ist nicht möglich.  
*Beispiele:* Häufigkeiten, Wahrscheinlichkeiten.
- Formal ist eine Metrik für eine Menge  $A$  eine **Distanzfunktion**  $d: A \times A \rightarrow \mathbb{R}$ , wobei für alle  $a, b \in A$  gilt:
  - $d(a, b) \geq 0$ ,  $d(a, a) = 0$
  - $d(a, b) = d(b, a)$
  - $d(a, b) \leq d(a, c) + d(c, b)$  für alle  $c \in A$  (Dreiecksungleichung)
  - $d(a, b) = 0 \Rightarrow a = b$

# Beispiel: Qualitätsstufen nach ISO 9126



# Qualitätsmanagement

---

- Das **Qualitätsmanagement** (QM) umfaßt alle Tätigkeiten der Gesamtführungsaufgabe, welche die Qualitätspolitik, Ziele und Verantwortungen festlegen sowie diese durch Mittel wie Qualitätsplanung, Qualitätslenkung, Qualitätssicherung und Qualitätsverbesserung im Rahmen des Qualitätsmanagements verwirklichen (DIN ISO 8402).
- Man unterscheidet **produktorientiertes QM** und **prozeßorientiertes QM**, (ISO9000) wobei sich letzteres ausschließlich auf den Softwareentwicklungsprozeß selbst bezieht (Meilensteine, Qualitätssicherungsmaßnahmen, Rollendefinition, ...).
- **Qualitätssicherung** (QS) sind alle geplanten und systematischen Tätigkeiten, die innerhalb des Qualitätsmanagement-Systems verwirklicht sind, um angemessenes Vertrauen zu schaffen, daß ein Produkt die Qualitätsforderung erfüllen wird (DIN ISO 8402).
- Die Qualitätssicherung umfaßt die **Validierung** und die **Verifikation**. Beide Tätigkeiten werden in der Praxis durch Tests unterstützt.

# Qualitätssicherung der Korrektheit

---

- Ein **Test** prüft die Qualität von Software, indem für einige, möglichst gut ausgewählte Testdaten die Übereinstimmung zwischen Anforderung und System untersucht wird
- Eine **Verifikation** prüft die Qualität von Software durch einen mathematischen Beweis, der die in der Spezifikation geforderten Anforderungen ausgehend von einem mathematisch exakten Modell des Systems beweist
- Ein **analysierendes Verfahren** stellt Maßzahlen oder Eigenschaften des Systems dar, die empirisch mit der Korrektheit von Software verknüpft sind.
  - Anzahl der Operatoren, Operanden, Länge der Implementierung, Anzahl der Knoten im Kontrollflußgraph, Anzahl uninitialisierter Variablen, Anzahl der Mehrfachzuweisungen ohne Lesezugriff, ...
  - Tiefe des Vererbungsbaums, Kopplung zwischen Klassen, Anzahl der Kinder pro Klasse, ...
  - Anzahl der Kommentare pro Programmeinheit, ...

# Qualitätssicherung durch Tests

---

- Ein Test ist prinzipiell nur geeignet, evtl. vorhandene Fehler eines Systems zu Tage treten zu lassen, nicht jedoch die Abwesenheit von Fehlern zu zeigen (vgl. Verifikation)
- Der **Überdeckungsgrad** ist ein Maß für den Grad der Vollständigkeit eines Tests
- Ein **Regressionstest** ist ein Testverfahren, bei dem eine Sammlung von Testfällen erstellt wird, die bei Änderungen am System (automatisch) erneut überprüft werden
- Bei einer Individualsoftware findet ein **Abnahmetest** statt
- Beim **Alpha-Test** für Standardsoftware wird das System in der Zielumgebung des Herstellers durch ausgewählte Anwender erprobt
- Beim **Beta-Test** für Standardsoftware wird das System bei ausgewählten Zielkunden in einer eigenen Umgebung zur Probenutzung zur Verfügung gestellt. Auftretende Probleme und Fehler werden protokolliert. Beta-Tester erhalten typischerweise einen Preisnachlaß auf das endgültige Produkt. Typischerweise werden Beta-Tests iterativ mit aufeinanderfolgenden "*release candidates*" (RC) durchgeführt.

# Zusammenfassung, Kernpunkte



- Definitionsphase
- Pflichtenheft
  - Inhalt
  - Struktur / Gliederungsschema
- Anforderungsanalyse
- Produktmodellierung
- Qualität / Test

# Was kommt beim nächsten Mal?



- Abnahme und Einführung
- Qualitätssicherungs- und Testverfahren