

Vorlesung "Software-Engineering"

Prof. Ralf Möller, TUHH, Arbeitsbereich STS

■ Vorige Vorlesung

- Einführung in die durch Software-Engineering gelösten Probleme
- Charakterisierung von Software-Qualität

■ Heute

- Produkte und Leistungen
- Projektphasen und Vorgehensmodelle
- Lastenheft

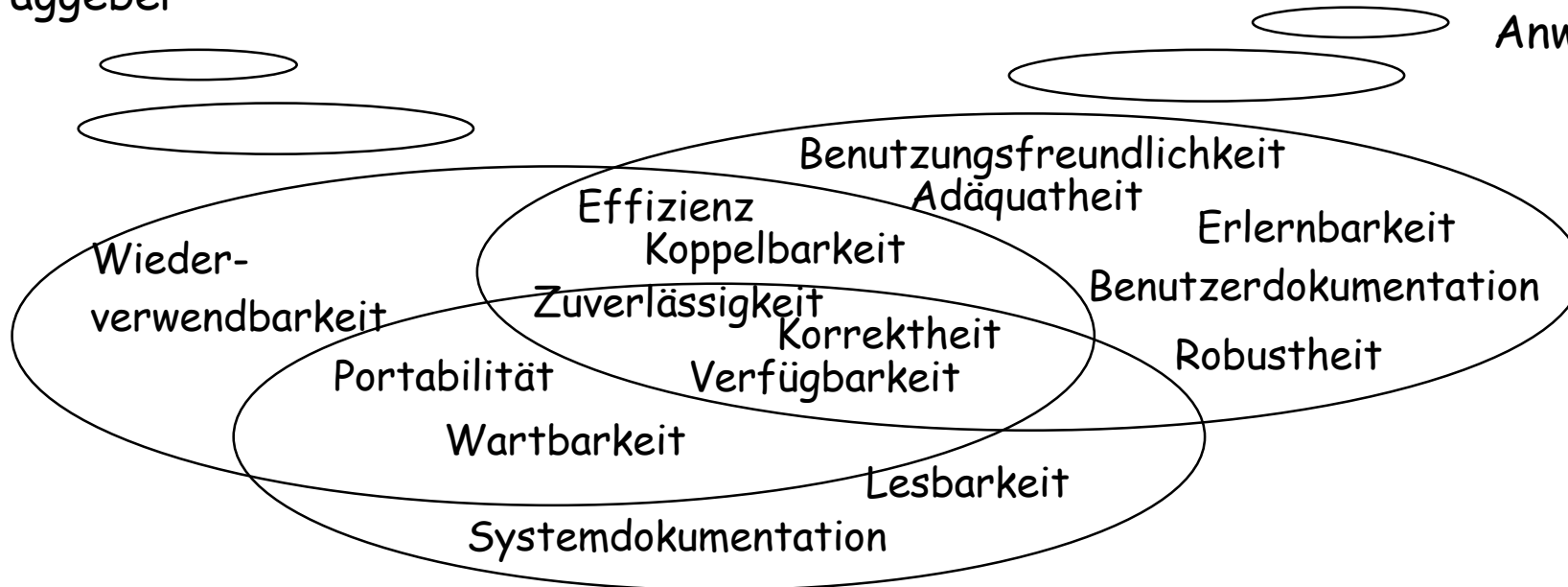
Software-Qualität: Perspektiven



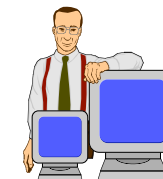
Auftraggeber



Anwender



Software-
Betreuer



Projektmanagement

■ Aufgaben und Phasen der Softwareentwicklung

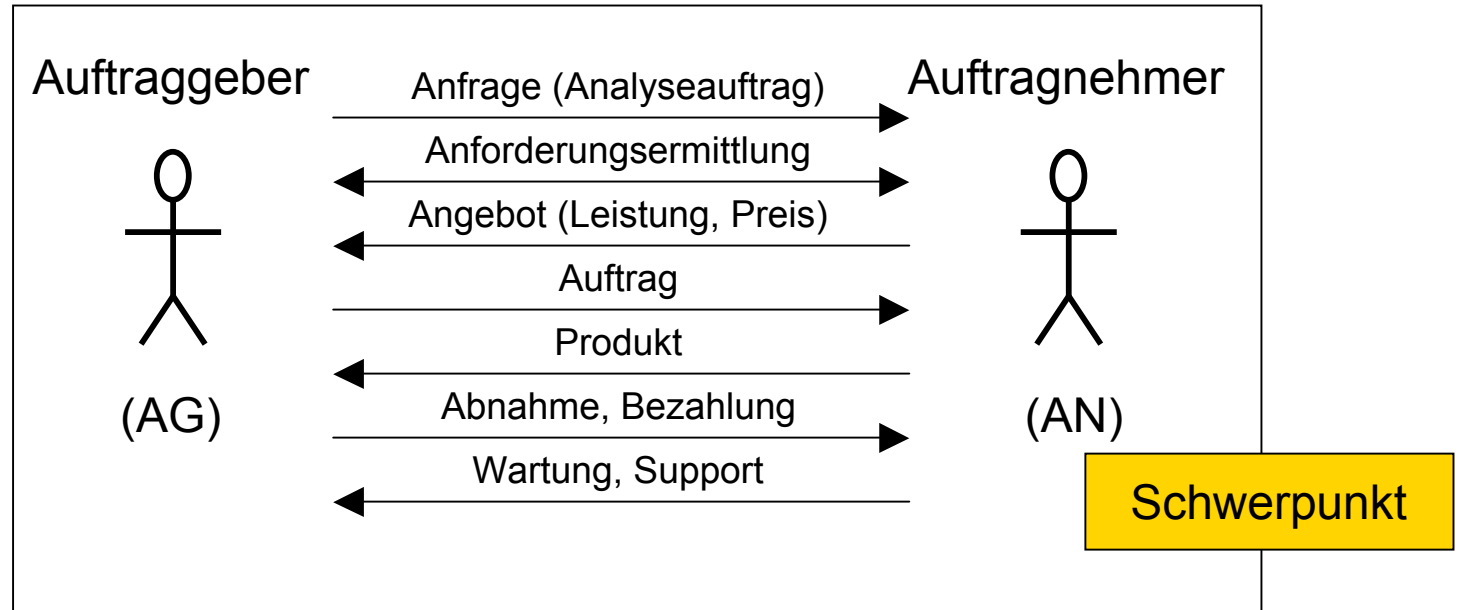
- Projektplan, Meilenstein
- Prozeßmodelle (auch Vorgehensmodelle genannt)
Wasserfall-, V-, Prototypen-, Evolutionäres-,
Inkrementelles-, Spiralmodell

■ Lernziele

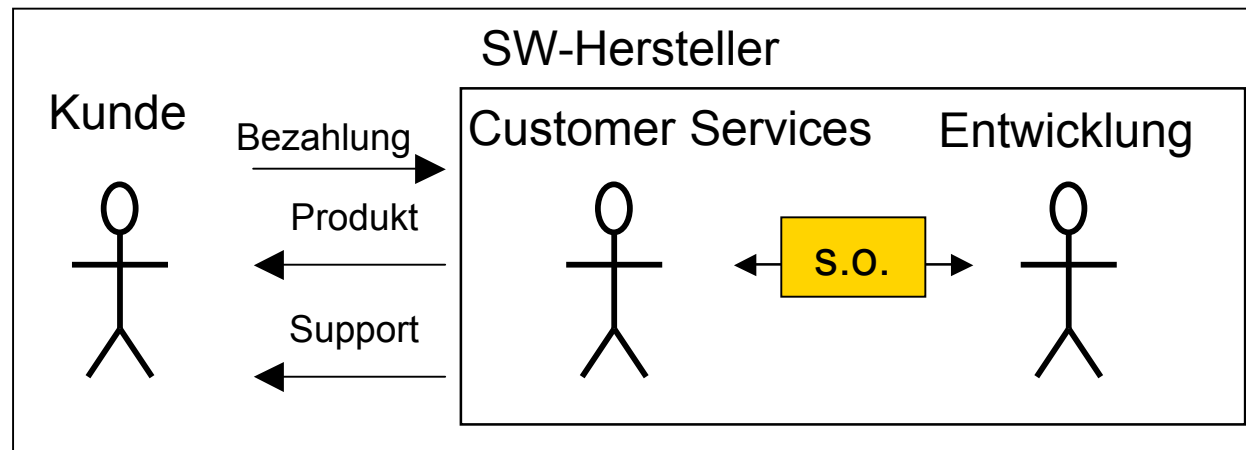
- Prozeßnotation, -modell und -plan unterscheiden können.
- Hauptaufgaben beim Prozeßmanagement wiedergeben können.
- Prozeßmodelle wiedergeben können.

Projektablauf

Individualsoftware



Standardsoftware



SW-Engineering als kooperative Aktivität (1)

Arbeitsaufteilung größerer Software-Entwicklungsteams in verschiedene **Ebenen**:

- | **Programmierung**: Programmierer, Entwickler, Kodierer, Datenbank-Administrator (DBA), Mediendesigner, ...
 - | Implementierung und Anpassung von Komponenten

technische
Kompetenz

- | **Softwarearchitektur**: Software- / System-Architekt
 - | Analyse und Design
 - | Definition von Komponenten und Protokollen

Abstraktions- und
Kommunikations-
fähigkeiten

- | **Projektmanagement**: Projekt-, Gruppen- und Abteilungsleiter
 - | Anforderungsermittlung
 - | Kostenplanung, Ressourcenverteilung
 - | Projektplanung und Controlling
 - | Gruppenkommunikation und Führung

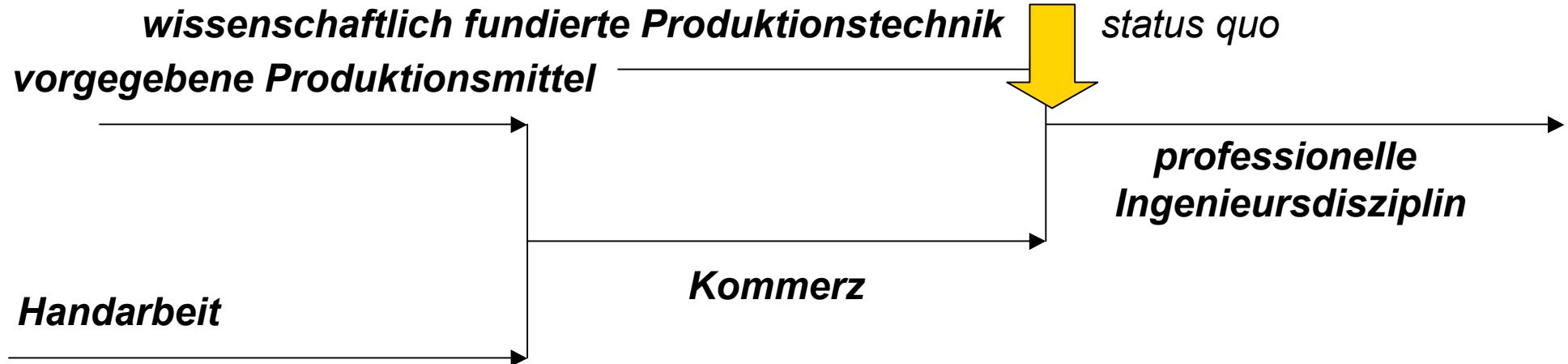
betriebswirtschaftliche
und soziale Kompetenz

Aufgaben überlappen sich!

Von Handarbeit zur Ingenieursdisziplin

- **Historisch:** implizite, informelle, anonyme, “zufällige” SW-Architekturen
 - Projekt- und Produktgetrieben
- **Ziel: Erhöhung der Produktivität und Planungssicherheit großer Softwareprojekte** durch explizite, formale, benannte und geprüfte Vorgehensweisen:
 - Verbesserte Kommunikation im Projekt-Team
 - Erhöhtes Wissen am Ende der Software-Engineering-Ausbildung
 - Verfügbarkeit eines Katalogs von Vorgehensmodellen (Handbuchartiges Wissen; vgl. Knuth / Sedgewick bei Algorithmen)
 - (Formale Modelle zum Testen, Verifizieren, Nutzen und Messen von Modellen).
- **Hindernisse**
 - Altsysteme („*legacy*“), bestehendes (veraltetes) Wissen, Personal, Organisation, ...
 - Schneller Fortschritt der Technik und Anwendungen
- **Status:** Software-Engineering als sich entwickelnde extrem flexible Informatik-Disziplin, die sich sogar dem Reifegrad der Standard-Ingenieursdisziplinen annähert.

Evolution einer Ingenieursdisziplin

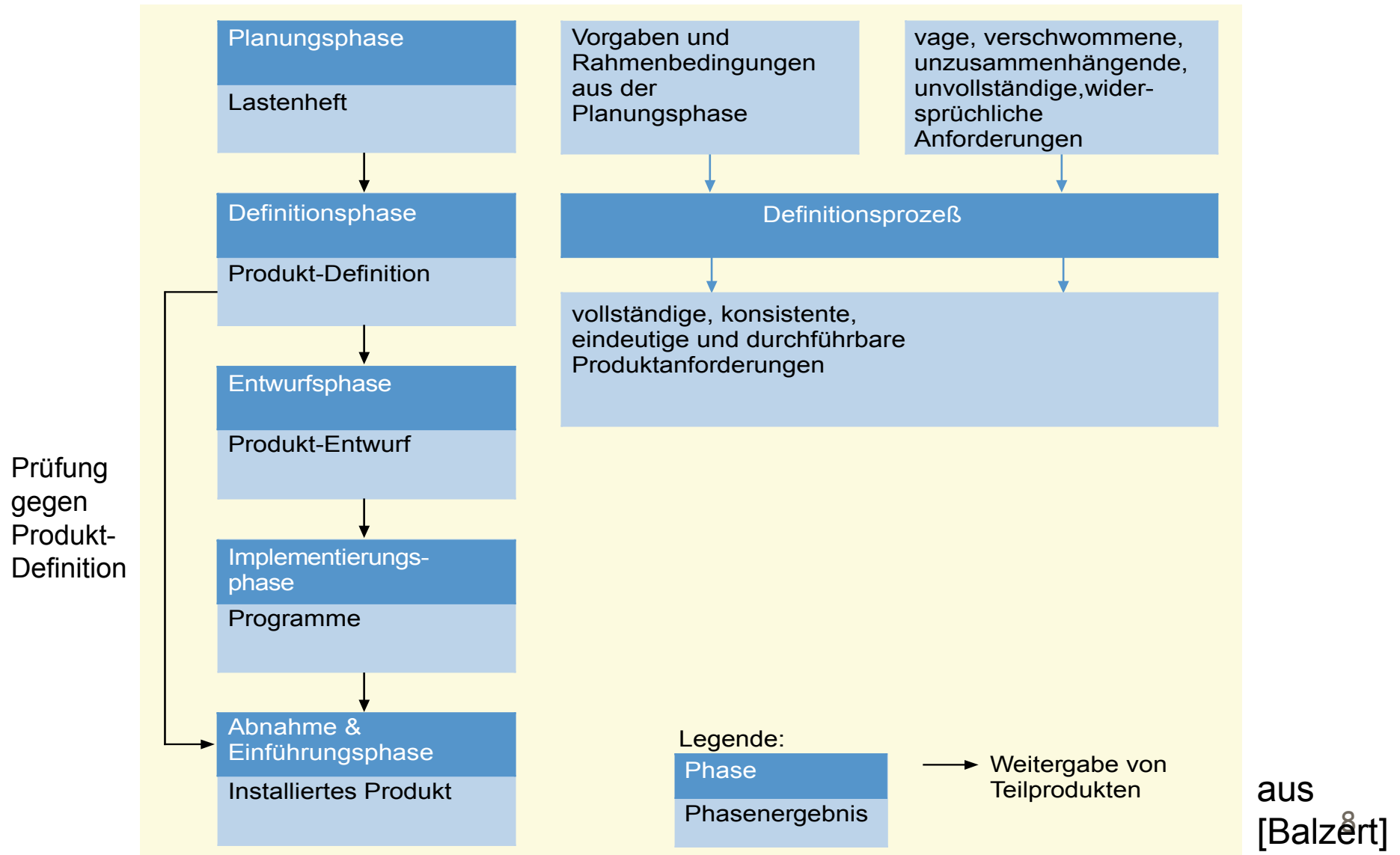


- Virtuosen und talentierte Amateure
- Intuition und "brute force"
- Zufälliger Fortschritt
- Fallweiser Austausch
- Extravagante Benutzung vorhandener Materialien
- Handarbeit für Benutzung statt Verkauf

- Erfahrene Handwerker
- Etablierte Verfahren
- Pragmatische Verbesserungen
- Ökonomische Aspekte: Kosten und Materialien
- Handarbeit für den Verkauf

- Ausgebildete Profis
- Analyse und Theorie
- Fortschritt basiert auf Wissenschaft
- neue Anwendungen durch Analyse
- Markt-Segmentierung und Produktvielfalt

Phasen der Softwareentwicklung



Aufgaben beim Software-Projektmanagement

- | Erstellung eines Projektplans
- | Auswahl einer Prozeßnotation
- | Auswahl eines Prozeßmodells

Planung

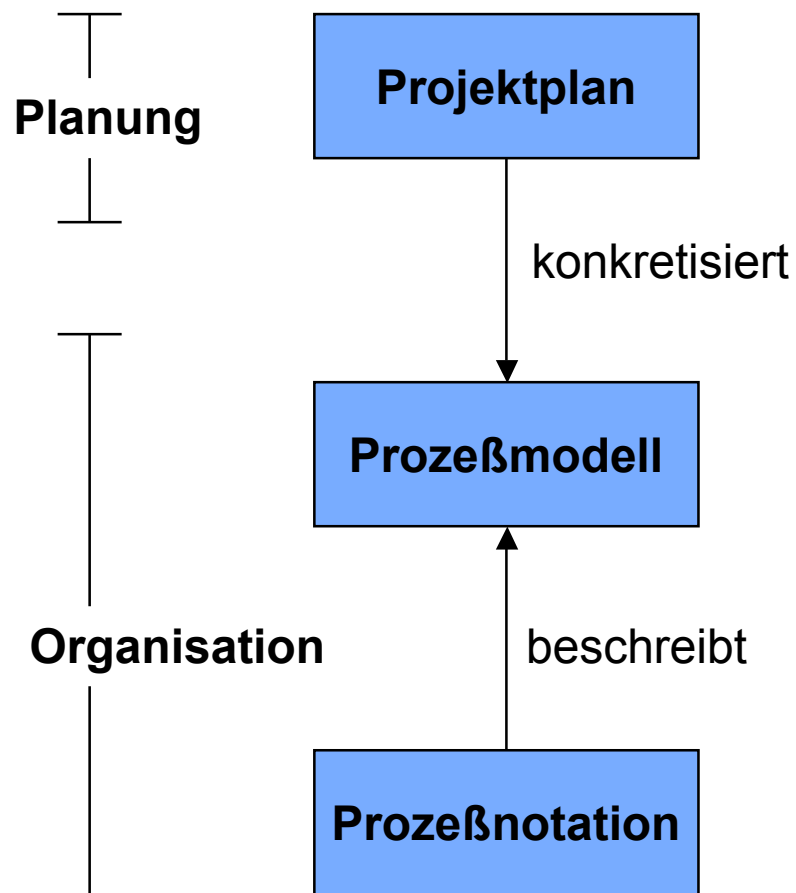
Organisation

■ Definitionen

- | **Software-Entwicklungsprozeß:** Aktivitäten, Methoden und Verfahren zur Entwicklung und Überprüfung von Software.
- | **Planung:** „Planung ist Entscheiden im voraus, was zu tun ist, wie es zu tun ist, wann es zu tun ist und wer es zu tun hat.“ [~ Koontz, O'Donnell 72]

Begriffe der Prozeßmodellierung

3 Abstraktionsebenen



Wird für jedes konkrete Software-Projekt erstellt (Projektleiter).
Beispiel: „Projektkalender“

Generelles Vorgehen (z.B. einer Firma) zum Entwickeln eines Software-Produkts.
Auch: **Vorgehensmodell**.
Beispiel: Wasserfall-Modell, Evol. Modell

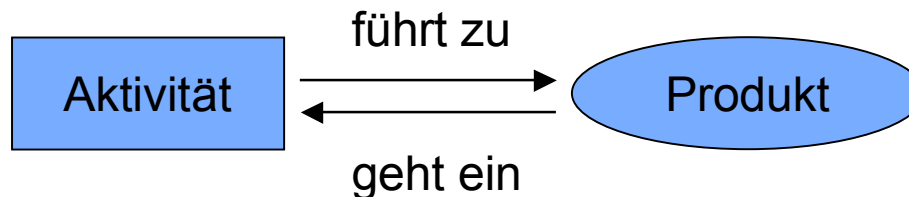
Sprache zur Spezifikation des Ablaufs von Software-Entwicklungen.
Beispiel: UML

Ausgewählte Prozeßmodelle

■ Prozeßmodell definiert:

- durchzuführende Aktivitäten
- Definition der Teilprodukte
- Fertigstellungskriterien
- Mitarbeiterqualifikationen
- Verantwortlichkeiten und Kompetenzen
- Standards, Richtlinien, Methoden und Werkzeuge

■ Hier verwendete Notation: „Boxes and Arrows“

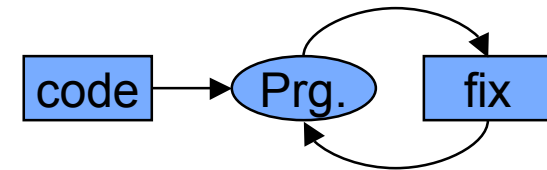


- häufig auch ohne Produkte (Dokumente) dargestellt

„Naives“ SWT-Grundmodell: Code & Fix

■ Grundmodell aus den Anfängen der Softwaretechnik: *Code & Fix*

- ❶ Schreibe ein Programm.
- ❷ Finde und behebe die Fehler im Programm.



■ Nachteile

- Fehlerbehebung strukturiert Programm so um, daß weitere Fehlerbehebungen und die Weiterentwicklung immer teurer werden.
➔ **Entwurfsphase** wird nötig.
- Selbst gut entworfene Software wird von den Benutzern oft nicht akzeptiert.
➔ **Definitionsphase** vor dem Entwurf wird nötig.
- Fehler sind schwer zu finden, da Tests schlecht vorbereitet und Änderungen unzureichend durchgeführt wurden.
➔ Separate **Testphase** wird nötig.

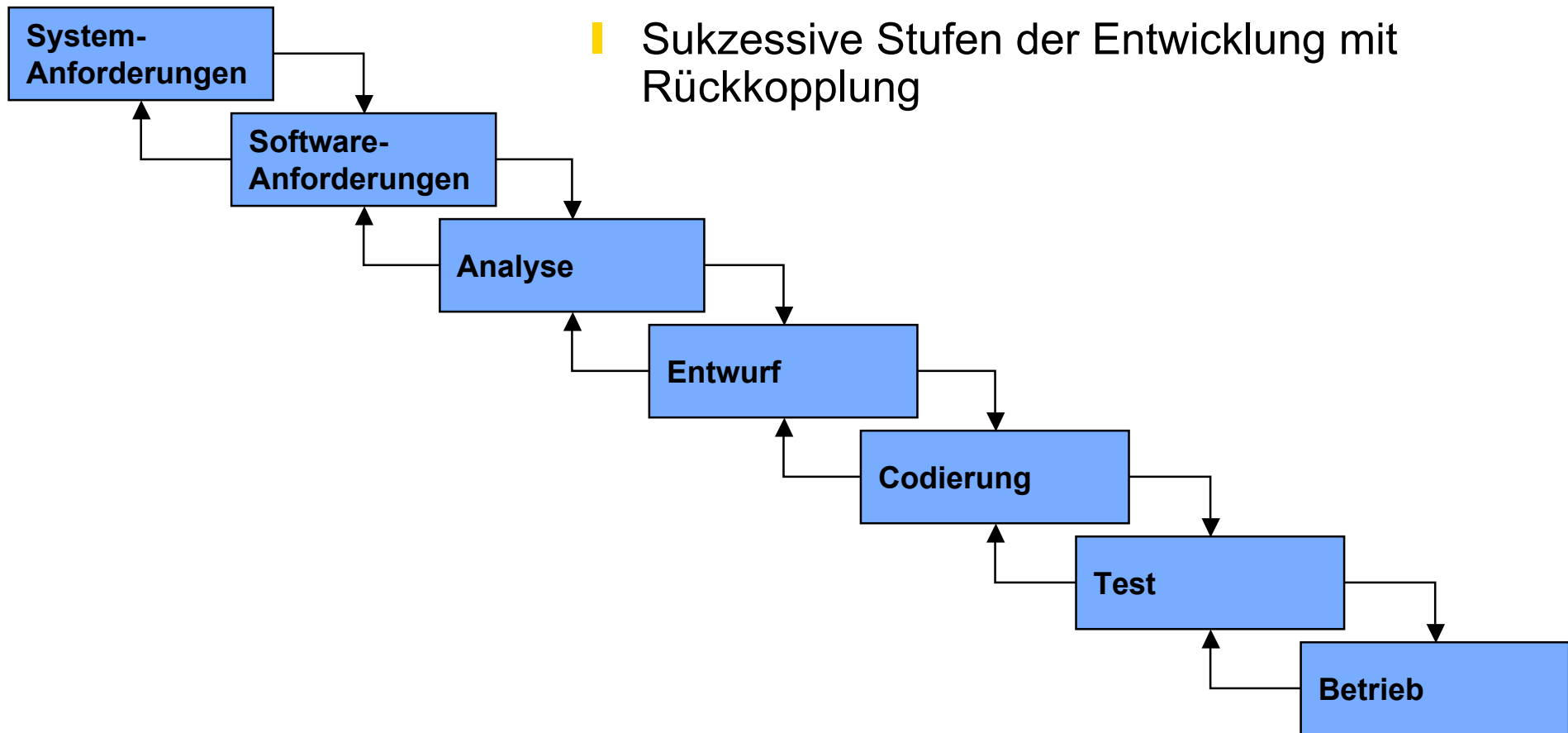
■ Folge: Entwicklung einer Reihe von besseren Modellen.

Vorgehensmodelle im Überblick

- Wasserfallmodell
- V-Modell
- Prototypmodell
- Evolutionsmodell
- Spiralmodell
- (Rational) Unified Process

Das Wasserfallmodell (1)

- Weiterentwicklung des stufenorientierten Modells
- Sukzessive Stufen der Entwicklung mit Rückkopplung



Das Wasserfallmodell (2)

■ Charakteristika

- Aktivitäten sind in der richtigen Reihenfolge und vollen Breite durchzuführen
- Am Ende jeder Aktivität steht ein Dokument (**dokumentgetriebenes Modell**)
- Entwicklungsablauf ist **sequentiell**, vorhergehende Aktivität muß beendet werden, bevor die nächste beginnt
- Orientiert am **Top-down**-Vorgehen
- Einfach, verständlich, wenig Managementaufwand
- Benutzerbeteiligung nur in der Definitionsphase

■ Nachteile

- Notwendige „Kurskorrekturen“ nicht frühzeitig erkennbar
- Sequentialität nicht immer nötig
- Gefahr, daß Dokumente wichtiger als das System werden
- Risikofaktoren werden u.U. zu wenig berücksichtigt

Das V-Modell (1)

- Erweiterung des Wasserfall-Modells, das **Qualitätssicherung** integriert
- **Verifikation** und **Validation** werden Bestandteile des Modells

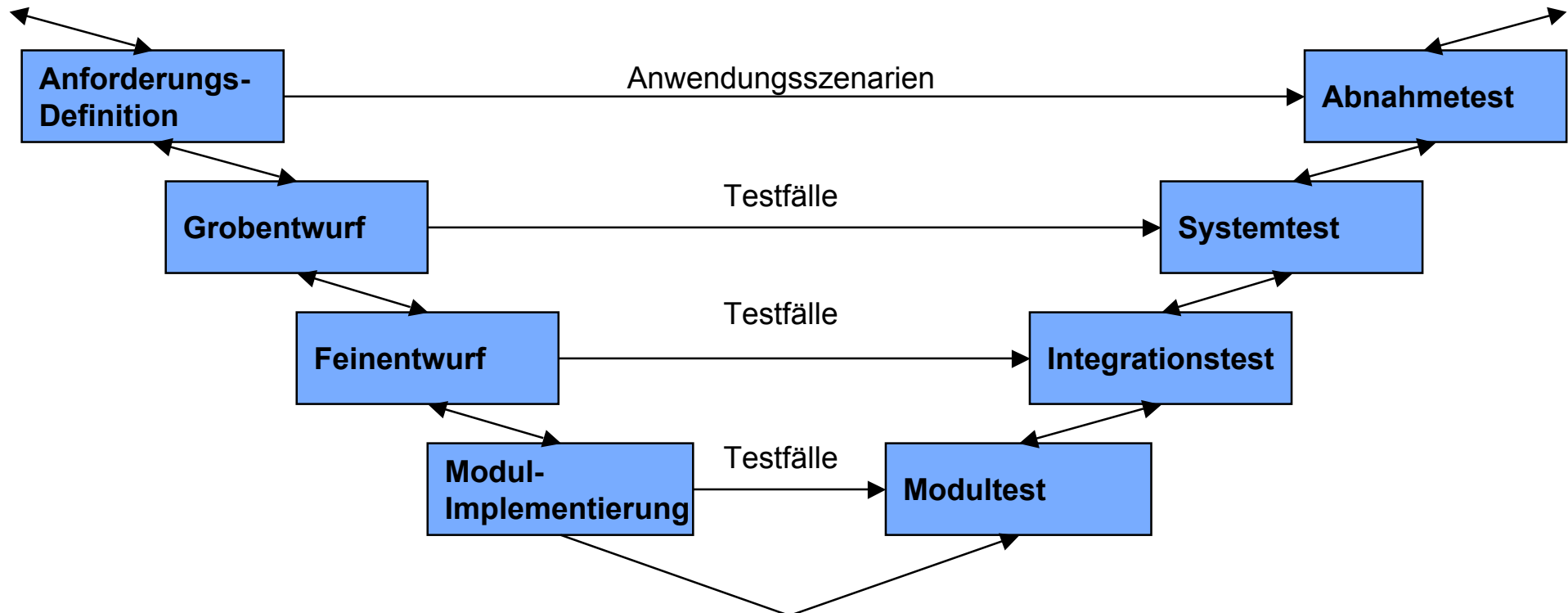
„Are we building the product right?“

- **Verifikation:** Überprüfung der Übereinstimmung zwischen Software-Produkt und seiner Spezifikation

„Are we building the right product?“

- **Validation:** Eignung bzw. Wert eines Produkts bezogen auf seine Einsatzzweck

Das V-Modell (2)



- Entwickelt ab ~1990 für Bundeswehr und später für weitere Behörden (Bundesverwaltung).
- Submodelle für Systemerstellung (SE), Qualitätssicherung (QS), Konfigurationsmanagement (KM) und Projektmanagement (PM).
- Ursprünglich für eingebettete Systeme entwickelt.

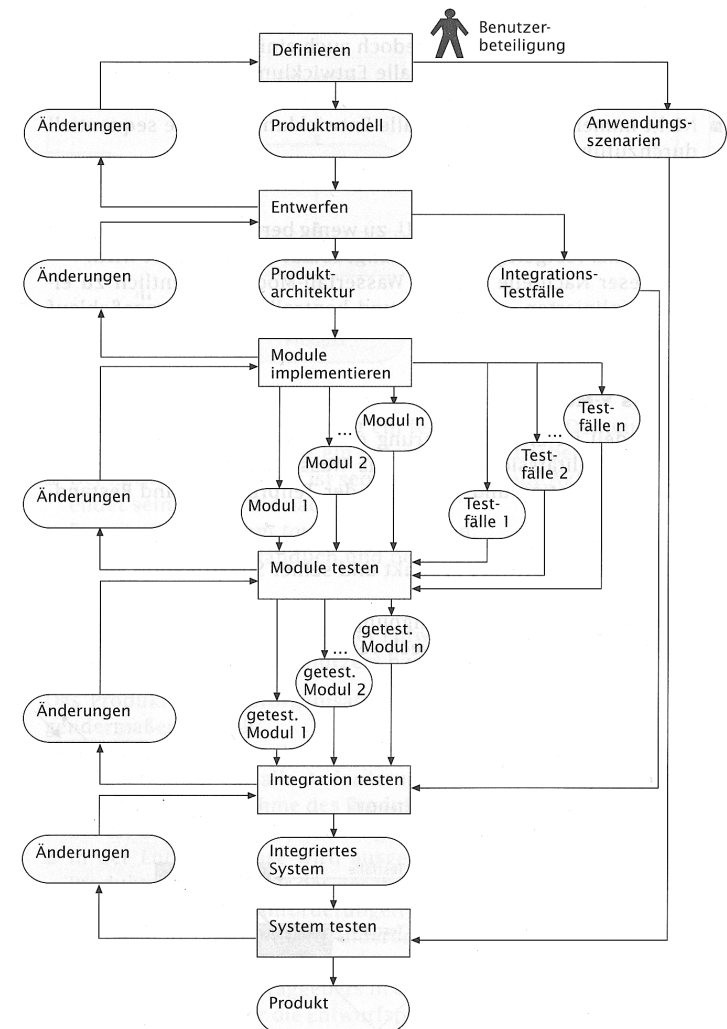
Das V-Modell: Bewertung (3)

■ Vorteile

- Integrierte, detaillierte Beschreibung von Systemerstellung, Qualitätssicherung, Konfigurationsmanagement und Projektmanagement
- Generisches Vorgehensmodell
- Gut geeignet für große Projekte

■ Nachteile

- Unkritische Übernahme der Konzepte, die für eingebettete Systeme entwickelt wurden, für andere Anwendungstypen
- Software-Bürokratie bei kleinen & mittleren Projekten
- Ohne CASE-Unterstützung nicht handhabbar



Das Prototypen-Modell (1)

■ Probleme traditioneller Modelle:

- Auftraggeber / Endbenutzer können oft **Anforderungen nicht vollständig / explizit formulieren**. Dies ist aber in klassischen Definitionsphasen nötig!
- **Kooperation zwischen Anwendern und Entwicklern endet mit der Definitionsphase**: Entwicklungsabteilungen ziehen sich nach Definitionsphase zurück und präsentieren erst nach Fertigstellung das Ergebnis; wünschenswerte Koordination zum Lernen von den jeweils anderen unterbleibt
- Oft existieren **unterschiedliche Lösungswege**, die besser experimentell erprobt werden und mit dem Auftraggeber diskutiert werden können.
- Manche **Anforderungen lassen sich theoretisch nicht garantieren** (z.B. Echtzeitanforderungen). Vor dem Abschluß der Definitionsphase muß also ggf. einiges **ausprobiert** werden.
- Das **Überzeugen des Auftraggebers** von der prinzipiellen Durchführbarkeit oder Handhabung einer Idee während der Akquisitionsphase wird nicht unterstützt (Folge für Verantwortungsteilung, Mittelfluss, etc).

Das Prototypen-Modell (2)

■ **Begriffsbestimmung Software-Prototyp:**

(im Gegensatz zum Begriff in anderen Ingenieursdisziplinen)

Ein Software-Prototyp...

- ... ist nicht das erste Muster einer großen Serie
(beliebig kopierbar, Massenfertigung)
- ... ist keine Simulation, sondern zeigt ausgewählte Eigenschaften des Zielprodukts im praktischen Einsatz (vgl. z.B. Windkanal oder Architekturmodell)
- ... dient zum Klären von relevanten Anforderungen oder Entwicklungsproblemen.
- ... dient als Diskussionsbasis für Entscheidungen.
- ... dient zu experimentellen Zwecken und Sammeln von praktischen Erfahrungen.

Vorgehensweise:

„prototyping“

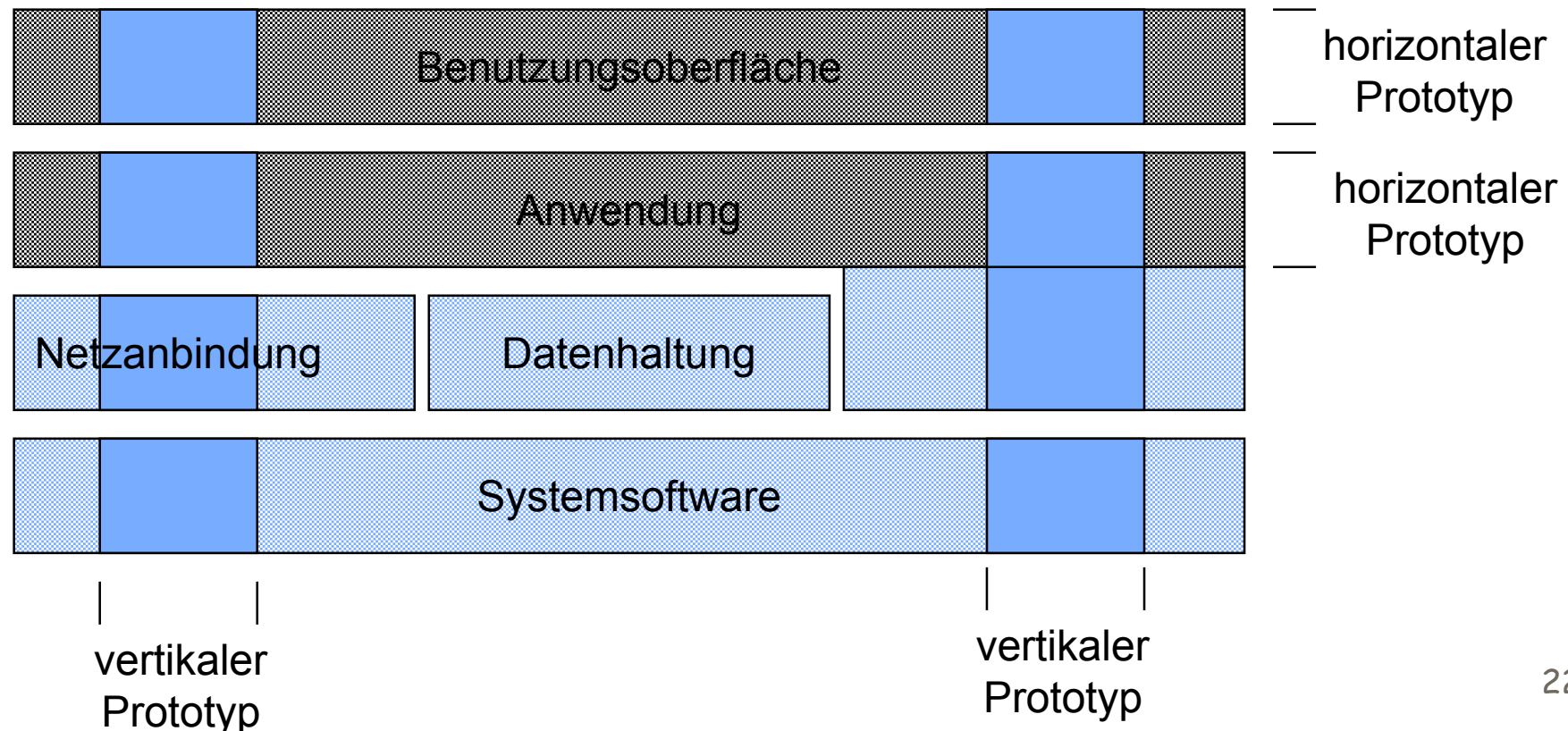
Das Prototypen-Modell (3)

■ Arten von Software-Prototypen:

- **Demonstrationsprototyp:** Dient zur Auftragsakquisition; verschafft Eindruck, wie das Produkt aussehen kann. Wichtig: Wird später weggeworfen!
- **Prototyp im engeren Sinne:** Wird parallel zur Modellierung des Anwendungsbereiches erstellt, um Aspekte der Benutzungsschnittstelle oder Teile der Funktionalität zu veranschaulichen. Dient zur Analyse.
(Exploratives Prototyping)
- **Labormuster:** Dient zur Beantwortung konstruktionsbezogener Fragen und Alternativen.
(Experimentelles Prototyping)
- **Pilotsystem:** Dient nicht nur zur experimentelle Erprobung oder Veranschaulichung, sondern ist schon Kern des Produkts. Unterscheidung zwischen Prototyp und Produkt verschwindet später. Die Weiterentwicklung erfolgt in Zyklen unter Beteiligung der Benutzer. Es ist ein wesentlich sorgfältigerer Entwurf nötig, da der Prototyp später weiterbenutzt wird!
(Evolutionäres Prototyping)
Prototyp ⇒ Pilot ⇒ Produkt

Das Prototypen-Modell (4)

- Ein fertiges Software-Produkt besteht aus vielen Komponenten und Ebenen.
- Unterscheidung zwischen **horizontalen** und **vertikalen Prototypen**:



Das Prototypen-Modell: Bewertung

■ Vorteile:

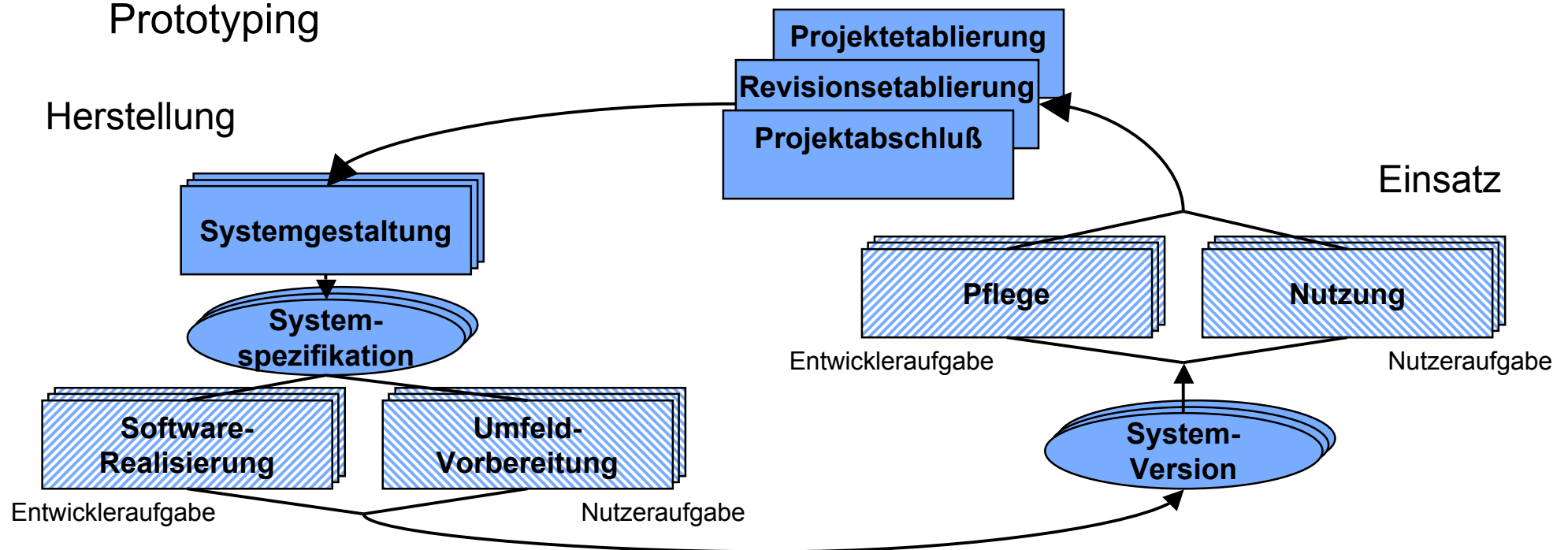
- Reduktion des Entwicklungsrisikos durch frühzeitige/stärkere Rückkopplung.
- Sinnvoll in andere Prozeßmodelle integrierbar.
- Prototypen sind durch geeignete Werkzeuge schnell erstellbar.
→ „**Rapid Prototyping**“

■ Nachteile

- Höherer Entwicklungsaufwand.
- Gefahr, daß ein „Wegwerf“-Prototyp nicht weggeworfen wird.
- Prototypen werden oft als Ersatz für Dokumentation angesehen.

Das evolutionäre/inkrementelle Modell

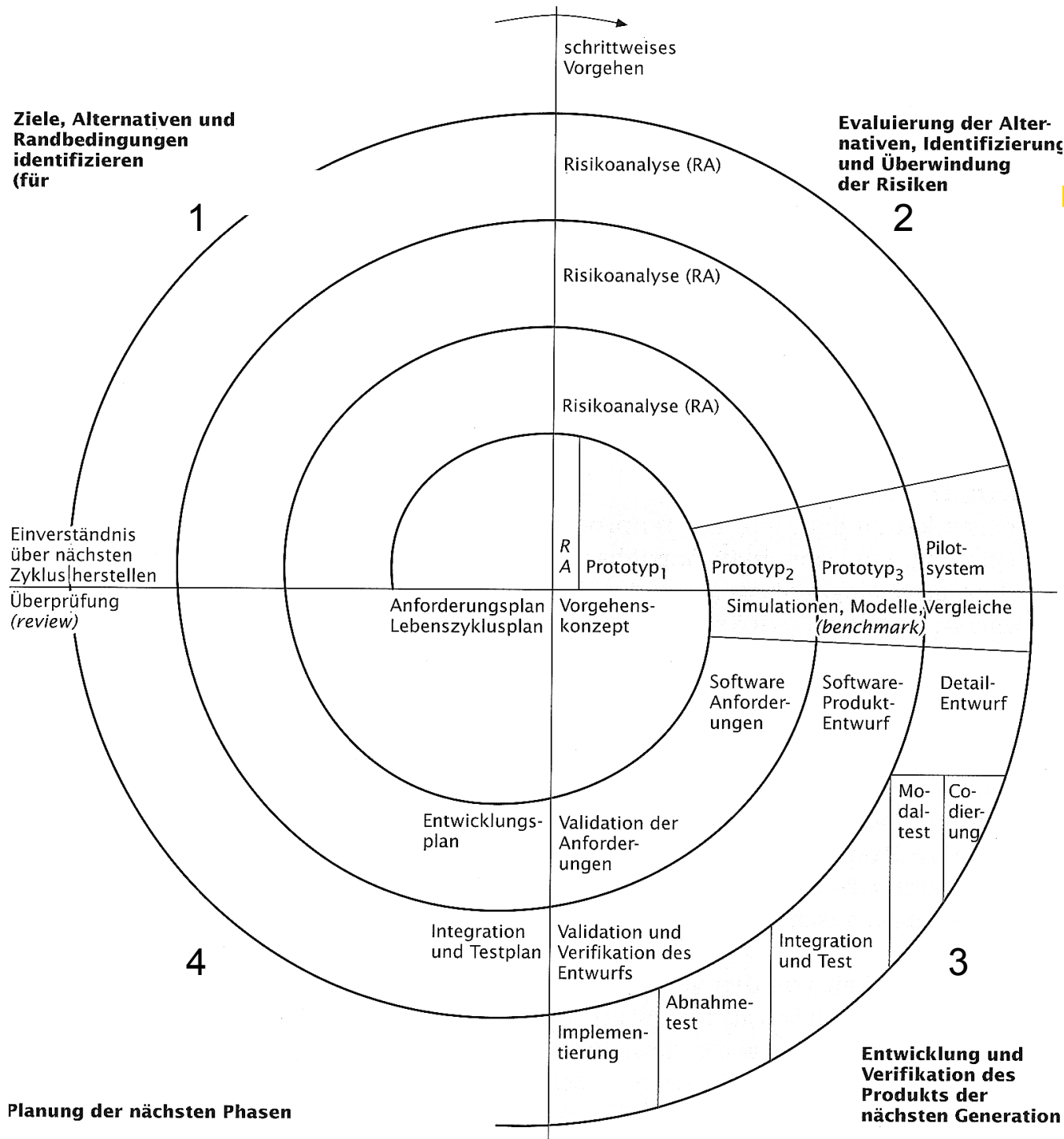
- **Beobachtung:** Software-(Weiter) Entwicklung unterliegt Änderungen
 - Lernen zwischen Entwicklern und Anwendern nötig, da
 - Veränderungen im technischen und Einsatzkontext stattfinden
 - sich durch den Einsatz des Systems neue Anforderungen ergeben
- Systementwicklung in Ausbaustufen, inkrementelle Entwicklung, Prototyping



Erweiterung: Das Spiralmodell ⁽¹⁾

- Das Spiralmodell ist eigentlich ein **Modell höherer Ordnung**
- Für jedes (Teil-)Produkt sind zyklisch vier Schritte zu durchlaufen:
- **Schritt 1:**
 - Identifizierung der Ziele des Teilprodukts (Leistung, Funktionalität, Anpaßbarkeit, ...)
 - Alternative Möglichkeiten zur Realisierung des Teilprodukts finden.
 - Randbedingungen bei verschiedenen Alternativen finden
- **Schritt 2:**
 - Evaluierung der Alternativen unter Berücksichtigung aller Alternativen
 - Identifizieren und ggf. Überwinden von Risiken (durch Prototypen, Simulation, ...)
- **Schritt 3:**
 - Abhängig vom Risiko wird ein Prozeßmodell festgelegt (oder eine Kombination).
 - Anwendung des Modells
- **Schritt 4:**
 - Planung des nächsten Zyklus, Überprüfung der nächsten 3 Schritte im nächsten Zyklus, Einverständnis mit Beteiligten sichern.

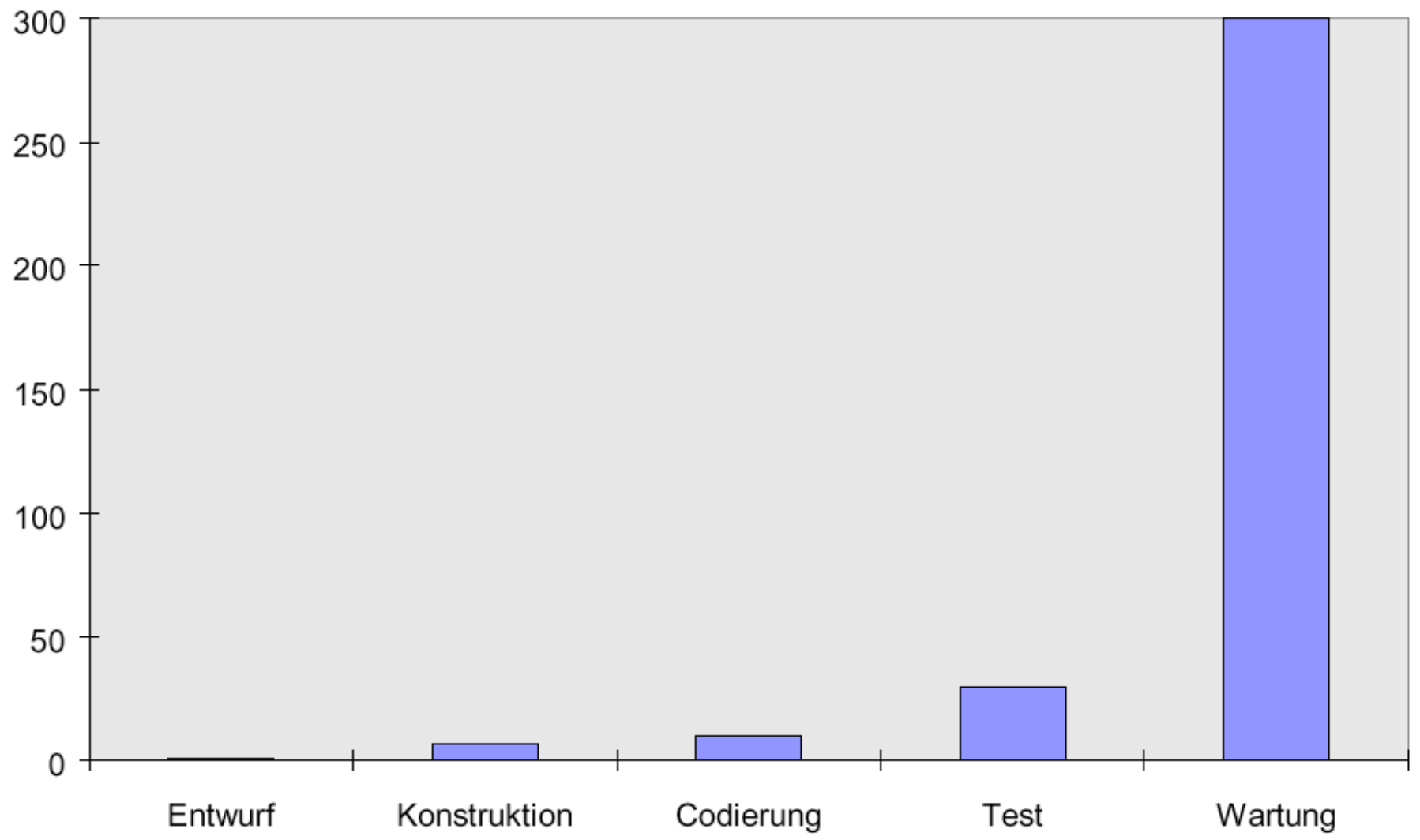
Das Spiralmodell (3)



Boehm, Barry: A Spiral Model of Software Development and Enhancement. ACM SIGSOFT Software Engineering Notes, August 1986;

Boehm, Barry: A Spiral Model of Software Development and Enhancement. IEEE Computer, Vol.21, Ausg. 5, Mai 1988, pp 61-72.

Wdhlg.: Zeitaufwand je nach Entwicklungsphase



Konsequenz

- Erfahrungsgemäß hohe Kosten bei Änderungen in späten Phasen rechtfertigen hohen Aufwand in frühen Phasen zur Vermeidung von späteren Änderungen
- Einfluß auf vorgeschlagene „klassische“ Vorgehensmodelle
- Aber: Kostenreduktion durch aufwendiges Vorgehen in frühen Phasen umstritten
- These: Änderungen sind kaum vermeidbar
- Durch neue Vorgehensweisen soll Änderungsflexibilität erhalten bleiben

Neuere Entwicklungen

- Unified Process (im Kontext von UML)
- Extreme Programming
- Agile Modeling
- Software Product Lines
- Component-Oriented Software Engineering
- Model-Driven Architecture
- ...

Wir greifen diese Themen etwas später wieder auf.