Vorlesung Software Engineering

Foliensatz Nr. 12 (05.01.12)

Arbeitsgruppe Software Engineering
Prof. Elke Pulvermüller

Universität Osnabrück
Institut für Informatik, Fachbereich Mathematik / Informatik
Raum 31/318, Albrechtstr. 28, D-49069 Osnabrück

elke.pulvermueller@informatik.uni-osnabrueck.de

http://www.inf.uos.de/se

Sprechstunde: mittwochs 14 – 15 und n.V.

Inhalt

Inhalt

6.1 Softwareentwicklung in Phasen

- **Die Entwicklung von „großen“ Systemen geschieht nicht chaotisch, sondern nach einem**

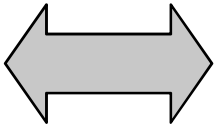
definierten Vorgehensplan (organisatorischer Rahmen)

$\Rightarrow$ **ingenieursmäßiges Vorgehen**

- **Alternative Bezeichnungen für Vorgehenspläne:**

  Phasenpläne, Entwicklungsprozess, Vorgehensmodell, Lebensläufe, Lebenszyklen

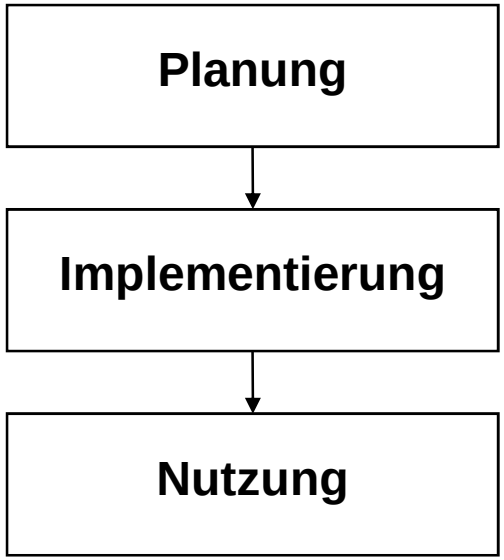
- **Entwicklung planbar**
  zeitliche Abläufe kontrollierbar und messbar
  Arbeitseinsätze berechenbar und koordinierbar (Teamarbeit)



eine starre Rasterung widerspricht der menschlichen Kreativität und auch der allgemeinen (ausprobierenden / chaotischen) menschlichen Arbeitsweise!
Organisatorischer Mehraufwand

6.1 Softwareentwicklung in Phasen

- **Beispiel**
  für einen sehr einfachen Phasenplan:



- **Herausforderungen:**

  - **Abgrenzungsproblem:** Phasen sind nur im Plan scharf getrennt, in der Realität verschwimmen Grenzen

  - **Abfolgeproblem:** Wie ist die Abfolge der Phasen (Prozessfolge)?

  - **Angemessenheitsproblem:** Ist das Modell auch tatsächlich dem angestrebten Projekt angemessen?

6.1 Softwareentwicklung in Phasen

Typische Phasen von Entwicklungsprozessen:

- **Anforderungsanalyse (Requirements Analysis / Analysis)**
  Bestimmung der Anforderungen an das zu entwickelnde System
- **Entwurf (Design)**
  Modellierung einer Lösung und Planung der Lösungserstellung
- **Implementierung (Implementation)**
  Bau des Systems
- **Wartung und Betrieb (Maintainance)**
- **Außerbetriebsetzung (Retirement)**

[PS94] B.-U. Pagel und H.-W. Six. *Software Engineering*, Band 1 Die Phasen der Softwareentwicklung, Addison-Wesley 1994

6.1 Softwareentwicklung in Phasen

Begriffe:

Process: (lat. procedere = fortschreiten)

(1) A sequence of steps performed for a given purpose; for example, the software development process.
(2) See also: task; job.
(3) To perform operations on data.

Software Development Process:

The process by which user needs are translated into a software product. The process involves translating user needs into software requirements, transforming the software requirements into design, implementing the design in code, testing the code, and sometimes, installing and checking out the software for operational use.
Note: These activities may overlap or be performed iteratively.

[IEEE Std. 610.12 (1990)]

6.1 Softwareentwicklung in Phasen

UNIVERSITÄT OSNABRÜCK

Begriffe:

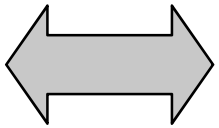
GEGENSTAND

Projektablauf

konkrete (real ausgeführte) Folge von Schritten zum Ergebnis

Projekt

Projektablauf + organisatorischer Rahmen



MODELL

Vorgehensmodell

explizite Vorstellung und Vorgabe, wie Projekte ablaufen sollen,
Abstrakte Folge von Schritten für beliebig viele Projekte (Modell, Muster)
Unabhängigkeit vom Produkt

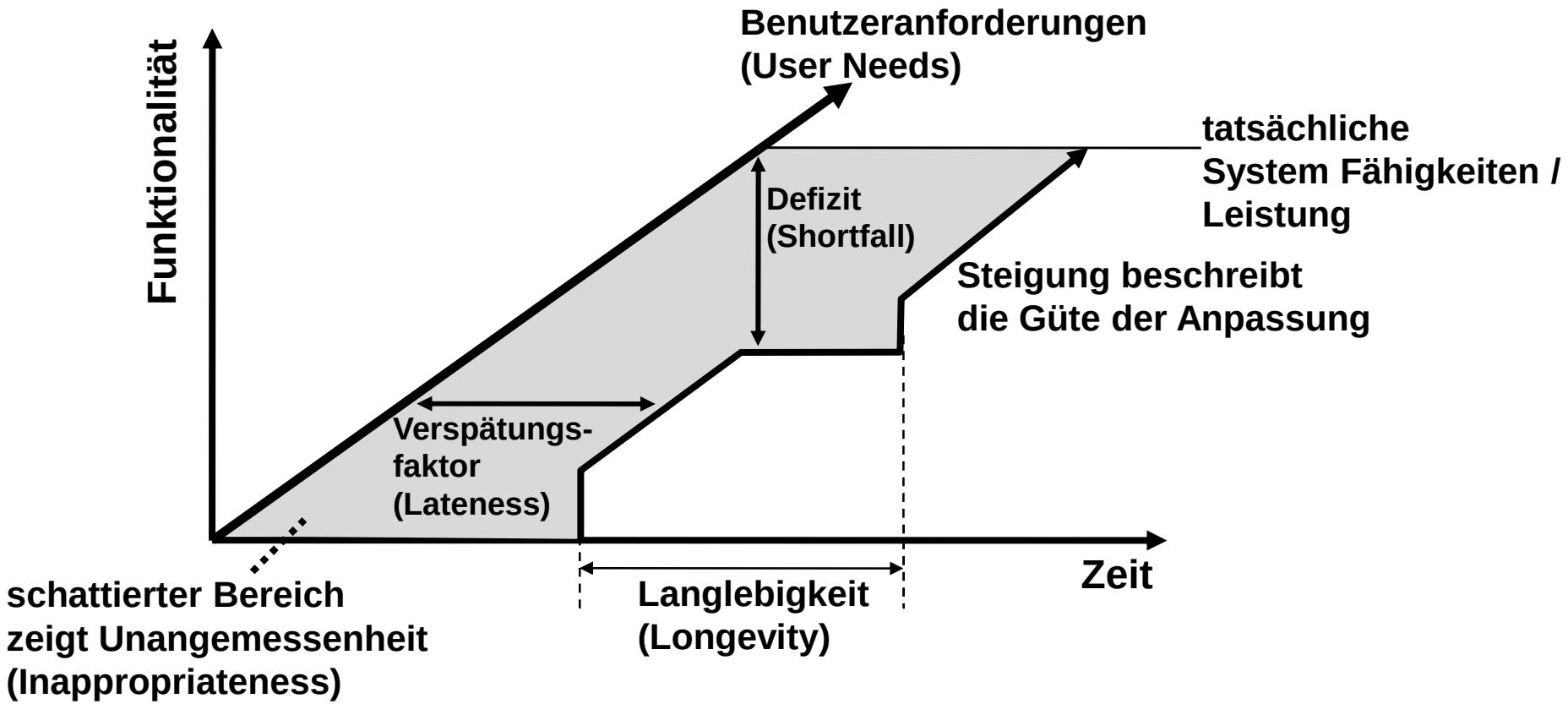
Prozeßmodell

Vorgehensmodell + Organisationsstruktur mit Vorgaben für Projektmanagement, Qualitätssicherung, Dokumentation, Konfigurationsverwaltung

[Unterscheidung nach Ludewig und Lichter: Software Engineering, dpunkt.verlag, 2007]

6.1 Softwareentwicklung in Phasen

Bewertungsmöglichkeiten für Software-Lebensläufe: [DBC88]

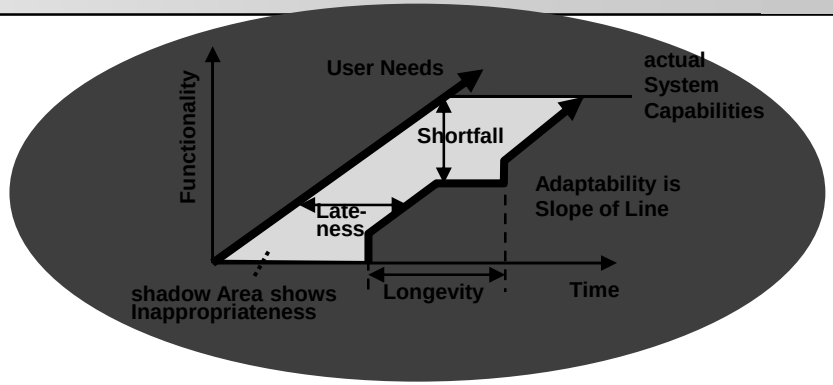


[DBC88] A. M. Davis, E. Bersoff und E. Comer. A Strategy for Comparing Alternative Software Development Life Cycles. *IEEE Transactions on Software Engineering*, 14(10):1453-1460, 1988.

- **Alternative Bewertungen wie z.B. SPICE = International Standard for Software Process Assessment** (http://www.sqi.gu.edu.au/spice)

6.1 Softwareentwicklung in Phasen

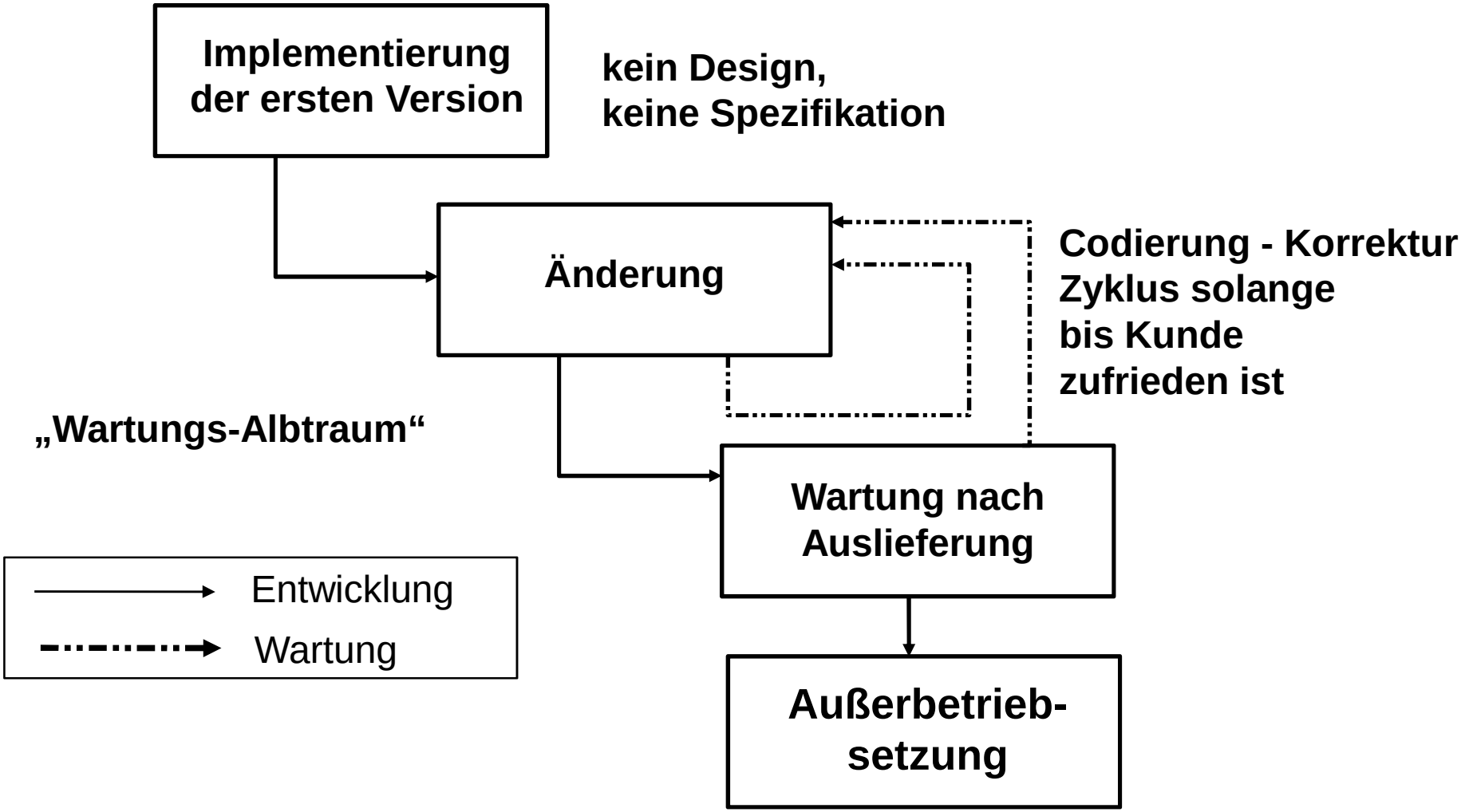
Bewertungsmöglichkeiten:



- **grauer Bereich: Unangemessenheit**
- **links neben dem schattierten Bereich:**
  die linear ansteigenden Benutzeranforderungen
  (*User Needs*); diese könnten aber auch unstetig oder expotentiell sein
- **rechts neben dem schattierten Bereich: die tatsächliche Leistung des Systems**
- **Verspätungsfaktor (*Lateness*) des implementierten Systems:**
  Vergleich von gewünschter und vorhandener Funktionalität in der horizontalen Achse
  Maß für die Zeit zwischen dem Erkennen einer neuen Anforderung und ihrer Implementierung im System
- **Defizit (*Shortfall*):**
  Vergleich auf der vertikalen Achse
  Maß dafür, wie weit die Funktionalität eines Systems zu einem Zeitpunkt t hinter den Anforderungen zurückbleibt
- **Langlebigkeit (*Longevity*):**
  Zeitspanne, in der das System anpassungsfähig ist; Zeitraum von der ersten Entwicklung über die Wartung bis zur Ablösung

6.2 Unsystematische “Modelle”

Code-and-Fix Modell



6.2 Unsystematische “Modelle”

Code-and-Fix Modell
Bewertung

- **Relativ einfach auszuführende Tätigkeiten**
- **Dem menschlichen Drang nach schnellem Ergebnis entsprechend mit schnell lauffähigem Ergebnis**
- **Einfachster und i.d.R. teuerster Software-Entwicklungsprozess** [McCo96]

“There is always an easy solution to every human problem – neat, plausible and wrong.” [nach H.L. Mencken (1880 – 1956) / LuLi07]

[McCo96] Steve McConnell, *Rapid Development.* Redmond, WA, USA: Microsoft Press, 1996

[LuLi07] Ludewig und Lichter: Software Engineering, dpunkt.verlag, 2007

6.2 Unsystematische “Modelle”

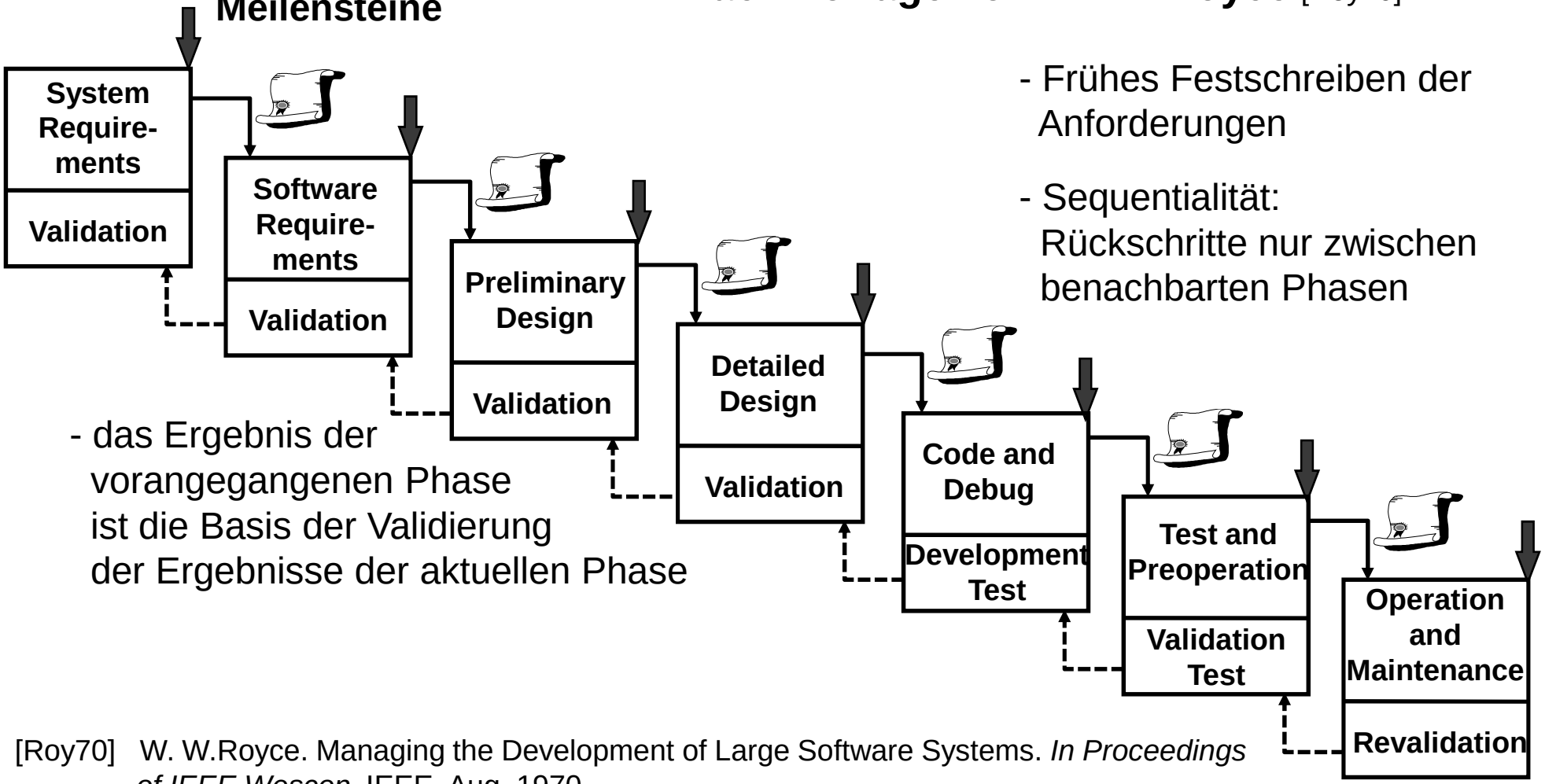
Code-and-Fix Modell
Bewertung

- Keine Planbarkeit, weil keine rationale Entscheidung, was in welcher Qualität hergestellt werden soll
- Keine Arbeitsaufteilung auf mehrere Personen
- Keine Anforderungserfüllung, da keine systematische Erhebung
- Prüfbarkeit eingeschränkt durch fehlende Soll-Vorgaben
- Schlechte Struktur des Ergebnisses, aufwendige Wartung
- Hoher Korrekturaufwand, da Mängel erst spät (im Einsatz) entdeckt werden
- Wichtige Konzepte und Entscheidungen sind nur in den Köpfen der Entwickler

6.3 Lineare, sequentielle Modelle

Wasserfallmodell von B. Boehm [Boe76]

nach Vorlage von W. W. Royce [Roy70]



- Frühes Festschreiben der Anforderungen
- Sequentialität: Rückschritte nur zwischen benachbarten Phasen
- das Ergebnis der vorangegangenen Phase ist die Basis der Validierung der Ergebnisse der aktuellen Phase

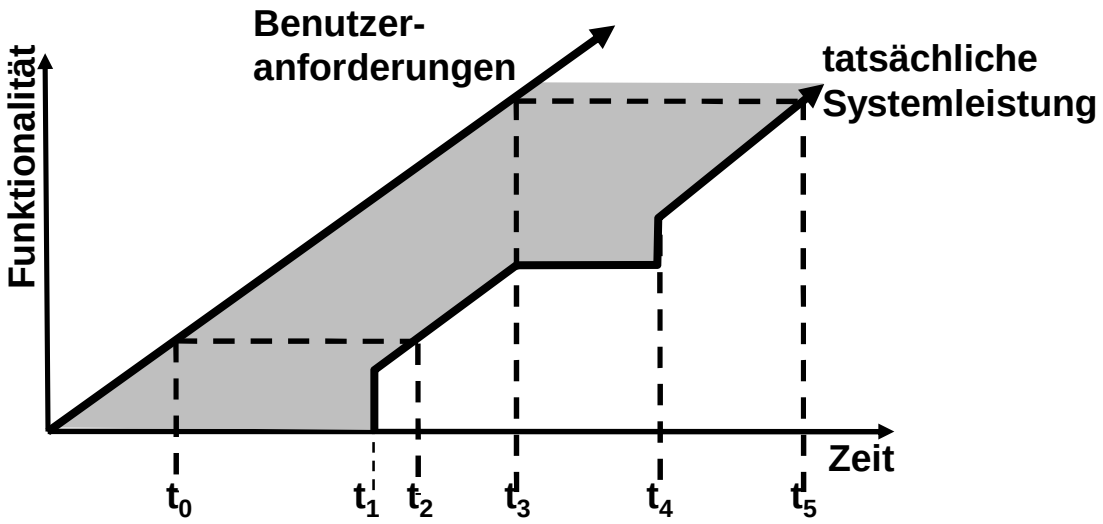
[Roy70] W. W.Royce. Managing the Development of Large Software Systems. *In Proceedings of IEEE Wescon*. IEEE, Aug. 1970.

[Boe76] B. W. Boehm. *Software Engineering*. IEEE Transactions on Computers, C-25:1226-1241, 1976.

6.3 Lineare, sequentielle Modelle

Wasserfallmodell Bewertung

Es wird ein optimaler Projektverlauf vorausgesetzt, d.h. z.B. keine schwerwiegenden Fehler in der Analyse



t0 : Start zur Entwicklung nach dem Wasserfallmodell

t1 : System fertig gestellt und schlagartig eingeführt („Big Bang“), es folgen inkrementelle Wartungen

t2 : erst jetzt werden die Anforderungen von t0 nach einigen Nachbesserungen erreicht

t3 : Einstellung der weiteren Wartungsarbeiten (von t1 bis t3)

t4 : Einführung eines neuen nach dem Wasserfallmodell entwickelten Systems

t5 : Erreichen der Anforderungen vom Zeitpunkt t3

Das Nachhinken der tatsächlichen Fähigkeiten des Systems ist bei jeder Vorgehensweise unausweichlich, hier jedoch sehr großer Abstand!

6.3 Lineare, sequentielle Modelle

Wasserfallmodell
Bewertung

- **Rückschritte sind nur zwischen den benachbarten Phasen möglich**
- **Frühes Festschreiben der Anforderungen erschwert nachträgliche Änderungen; Fehler in der Anforderungsanalyse werden erst spät erkannt**
- **Einführung des Systems erfolgt erst sehr spät und erfolgt auf einen Schlag (*Big Bang*)**

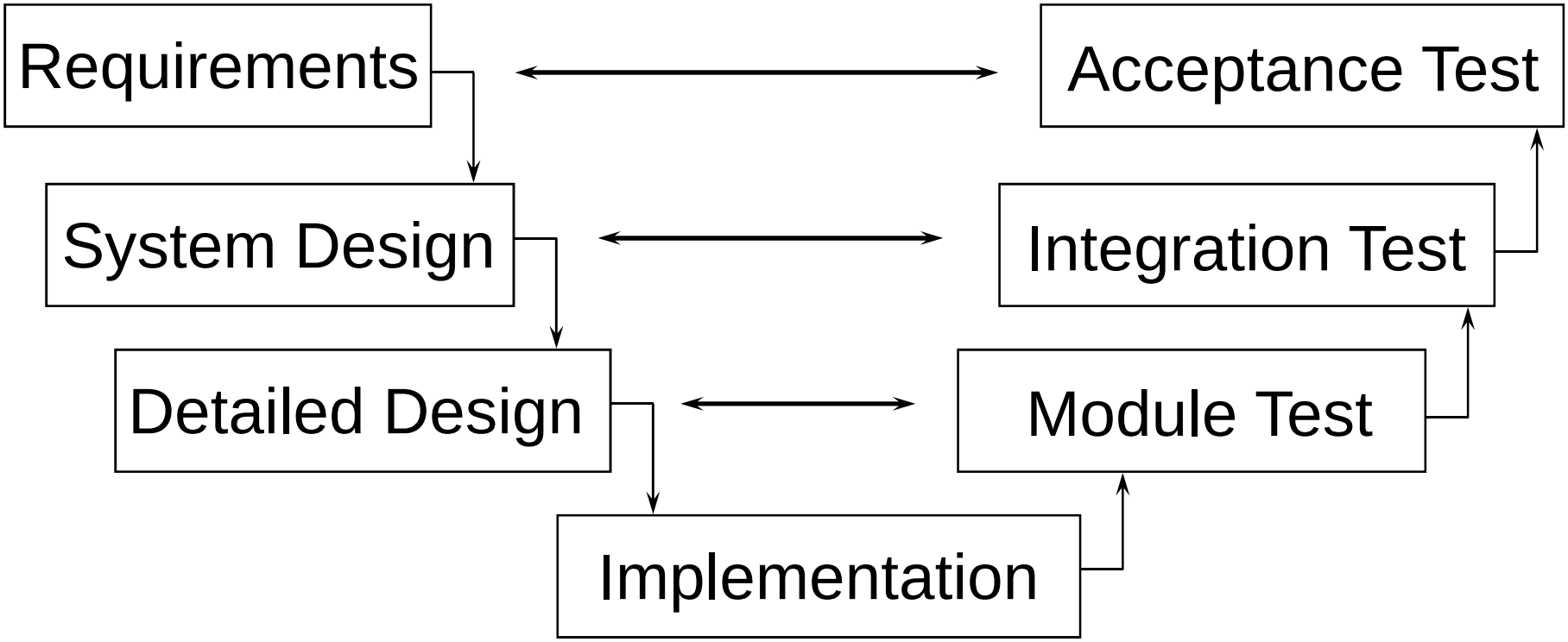
⇒ **Vorgehen nur bei relativ kleinen Projekten von Erfolg**

⇒ **Iterationen und Wiederholungen sind nur bedingt möglich.**
Die Anforderungen werden nicht mehrfach überprüft und praktisch erst bei der Einführung wirklich durch die Kunden validiert

⇒ **Das System wird schlagartig eingeführt und nicht in Teilschritten.**
Kapitalrückfluss (ROI = *Return of Investment*) erst sehr spät

V-Modell (“V” Life Cycle)

Phasen sind in V-Form gegenübergestellt



V-Modell (“V” Life Cycle)
Bewertung

- Betont die Wichtigkeit von Tests
  - Quality Assurance
- Was nicht wirklich verbessert wird:
  - Sequentialität
  - Feedback
  - Risk Management (während der Entwicklung)

6.4 Frühe Prototypen (Rapid Prototyping)

„Rapid Prototyping“: Prototypen-basiertes Entwicklungsvorgehen

- **Was ist ein Prototypen?** Eine partielle ausführbare Implementierung des Systems mit dem primären Ziel, schnell Problem und Lösung besser zu verstehen; dem geplanten System ähnlich; erfüllt aber wesentliche Anforderungen noch nicht

- **Zweck von Prototypen:** Überprüfen, der geplanten Arbeiten
  Unterstützung des Verständnisses
  Eine Lösung für IKIWISI (= „I'll Know It When I See It“)

- **Einsatz von Prototypen:**
  - Demonstration (v.a.) von Oberflächen / Schablonen (*Mock-up*)
  - Demonstration von Funktionalitätsabläufen (*behavioral Prototypes*)
  - Validierung von Problemlösungen durch Prototypen, z.B. Kommunikation, kritische, zeitliche Abläufe
  - Bezugspunkt für Anforderungsbeschreibungen , Kommunikationsbasis

6.4 Frühe Prototypen (Rapid Prototyping)

Begriffe

Prototype:

A preliminary type, form, or instance of a system that serves as a model for later stages or for the final, complete version of the system.

Prototyping:

A hardware and software development technique in which a preliminary version of part or all of the hardware or software is developed to permit user feedback, determine feasibility, or investigate timing or other issues in support of the development process.

Rapid Prototyping:

A type of prototyping in which emphasis is placed on developing prototypes early in the development process to permit early feedback and analysis in support of the development process.

[IEEE Std. 610.12 (1990)]

6.4 Frühe Prototypen (Rapid Prototyping)

Klassifikation von Prototypen:

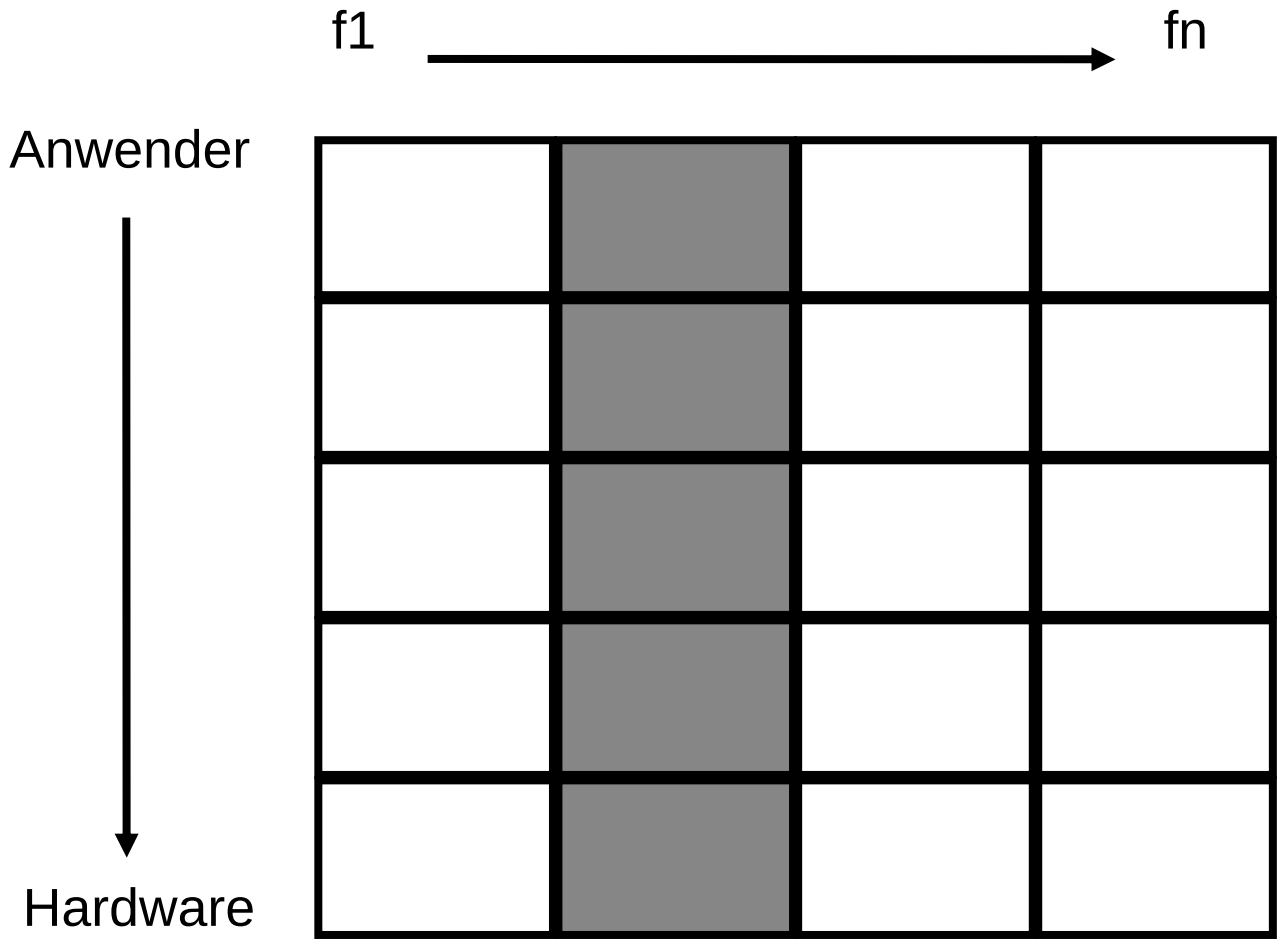
Demonstrationsprototyp
Funktionale Prototypen
Labormuster
Pilotsystem

- explorativ (*throw-away*) vs. evolutionär
- horizontal vs. vertikal

Keine scharfen Grenzen

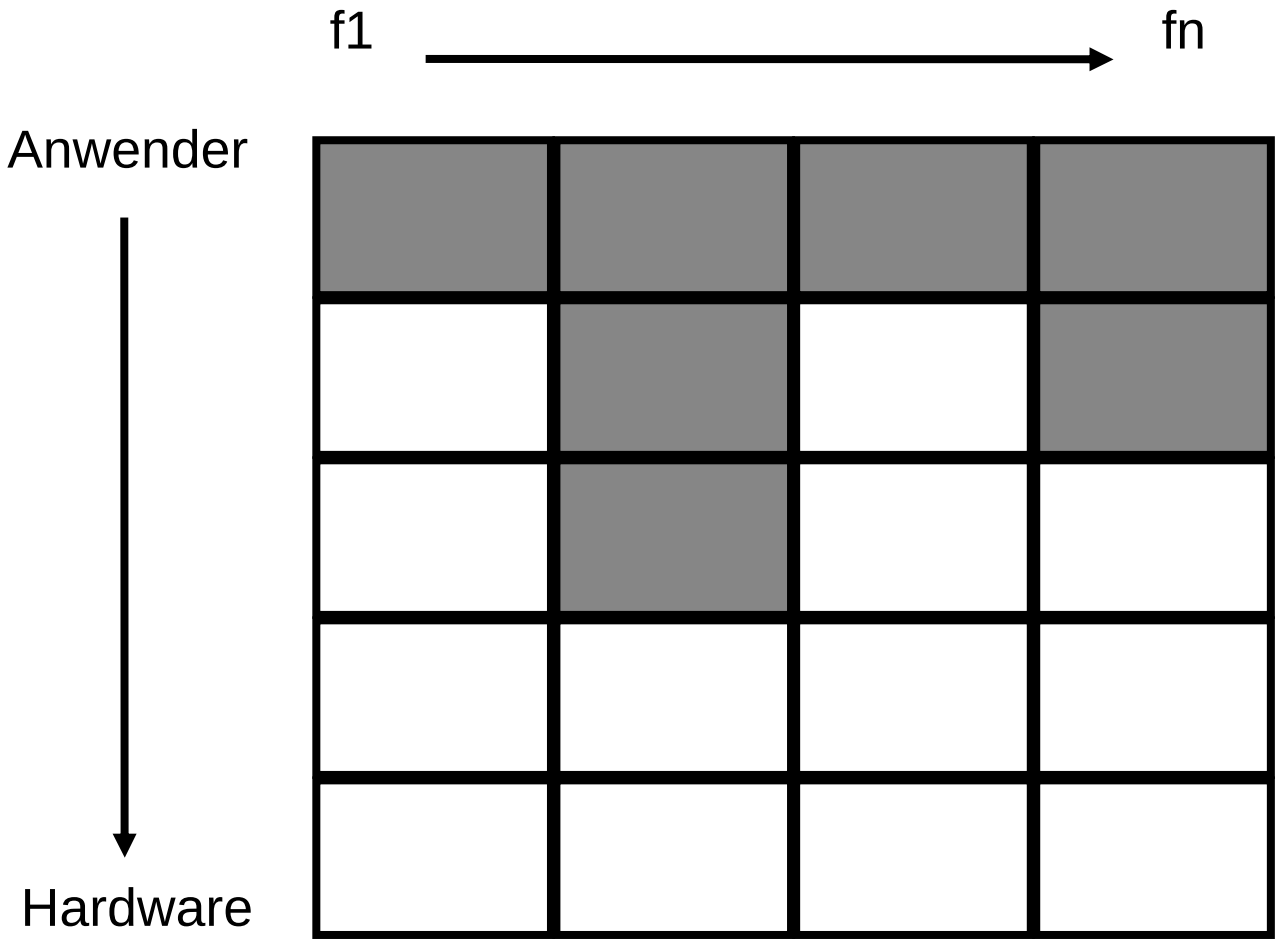
6.4 Frühe Prototypen (Rapid Prototyping)

Vertikale Prototypen



6.4 Frühe Prototypen (Rapid Prototyping)

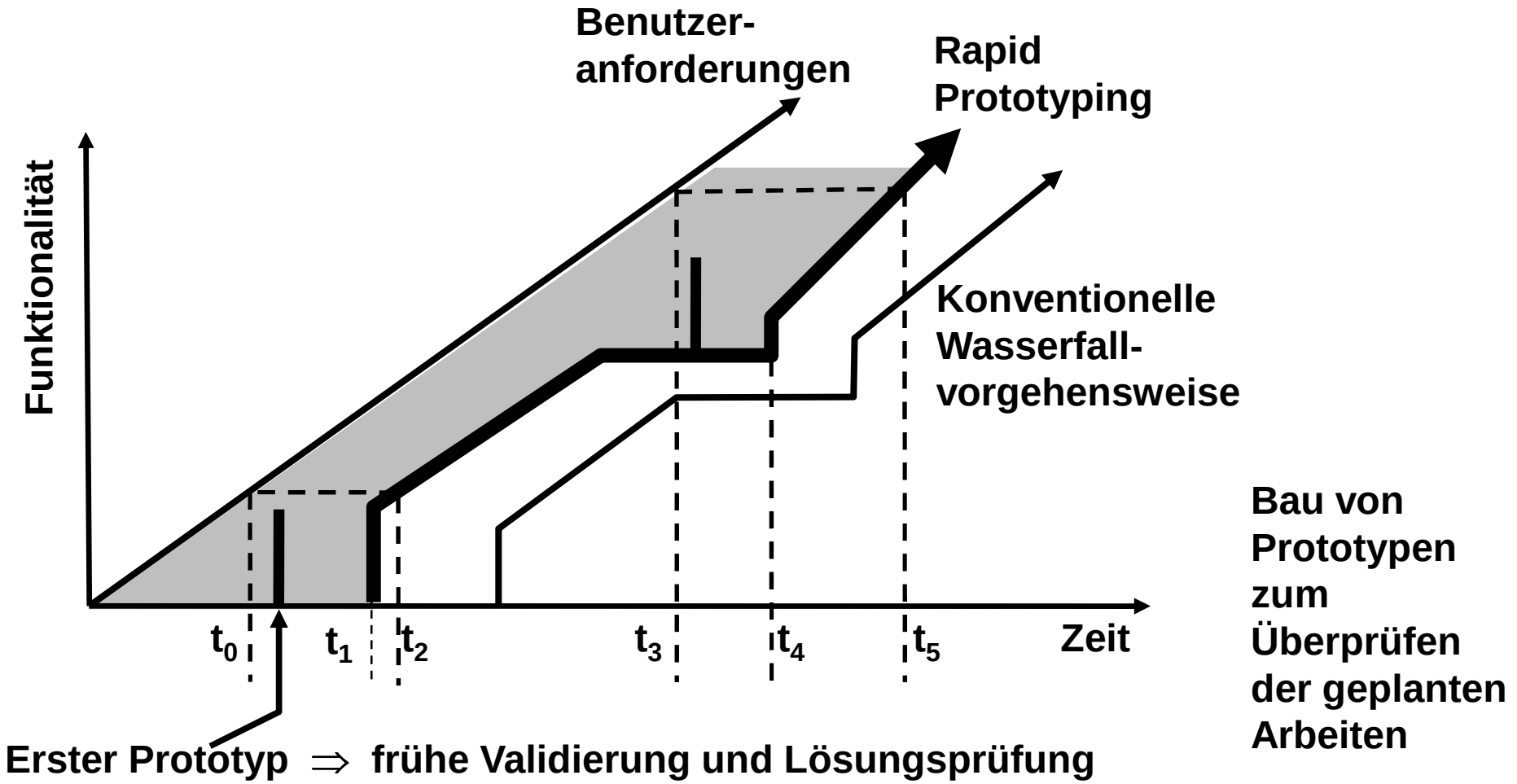
Horizontale Prototypen



6.4 Frühe Prototypen (Rapid Prototyping)

Bewertung

- In Kombination mit dem Wasserfallmodell (wegen fehlender organisatorischer Struktur des Rapid Prototyping)



6.4 Frühe Prototypen (Rapid Prototyping)

Bewertung

Wasserfallmodell mit Prototypen gegenüber dem „normalen“ Wassermodell.

Kurz nach Projektstart (t_0) entsteht ein erster Prototyp. Dieser deckt die wichtigsten Funktionalitäten ab. Durch die verbesserte Validierung kann das System schneller auf Basis der Validierungsergebnisse gebaut werden. Die Erfüllung der Systemanforderungen (zur Zeit t_0) kann schneller geschehen (t_2).

6.4 Frühe Prototypen (Rapid Prototyping)

Bewertung

- Verbesserung gegenüber dem reinen Wasserfallmodell, da durch Prototypen Anforderungen sehr früh validiert werden können, bzw. Prototypen die Anwendung von Lösungswegen früh geprüft werden.
- Weiterhin mangelnde Möglichkeiten zum Rücksprung analog dem Wasserfall
- Weiterhin *Big Bang*-Einführung von Systemen
- Besondere Problematik bei Prototypen zur Valdierung der Anwenderforderungen:
  Anwender (oder der Chef?) können durch schnell erstellten Prototypen über den tatsächlichen Aufwand der Entwicklung hinweggetäuscht werden.

$\Rightarrow$ Genaue Vorbereitung der Prototypenpräsentation ist notwendig

6.5 Evolutionäre, inkrementelle Modelle

Iterative Enhancement: „Grow, don't build software.“ (Fred Brooks)

- **Schritt 1: Prototyp**
  - Erzeuge ein einsetzbares aber kleines Teilsystem
  - Überprüfe daran die Benutzeranforderungen
  - Erzeuge einen groben Auszug des System Designs
  - “plan to throw one away, you have to do it anyway (Brooks 75)

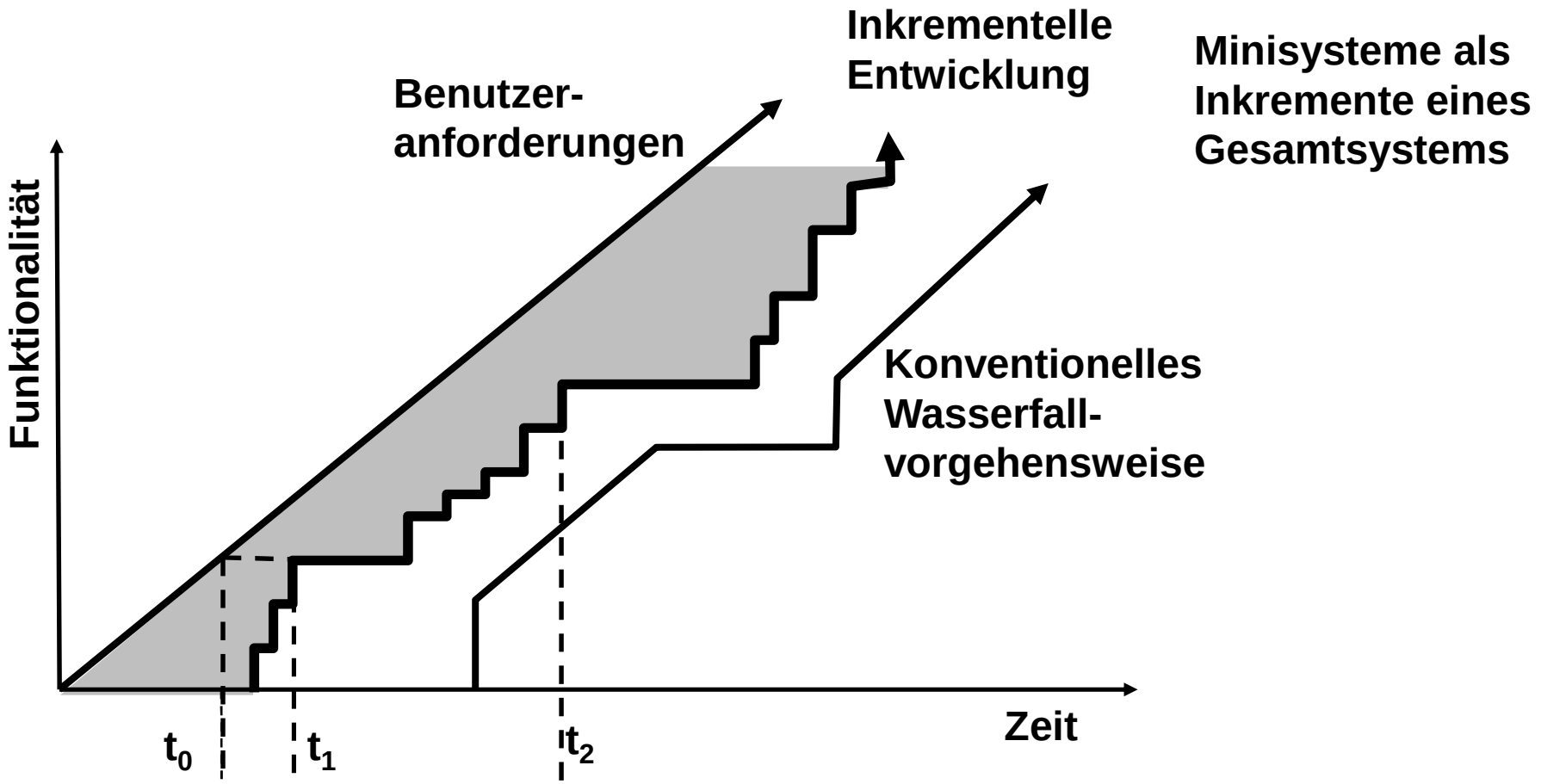
- **Schritt 2: Erweiterung**
  - Korrigiere Mängel und füge Funktionalität hinzu
  - Bestimme „Hot-Spots“

- **Schritt 3: Konsolidierung**
  - Korrigiere Entwurfsfehler
  - Führe neue Abstraktionen ein

Beispiele: Spiralmodell, STEPS-Methode

6.5 Evolutionäre, inkrementelle Modelle

Bewertung



6.5 Evolutionäre, inkrementelle Modelle

UNIVERSITÄT OSNABRÜCK

Wasserfallmodell mit inkrementellen Schritten Bewertung

- ***Big Bang*-Effekt kann verhindert werden;**
  durch das Setzen von vielen Meilensteinen ist der Projektverlauf besser überprüfbar (ebenso eine Bezahlung der Meilensteinleistung)

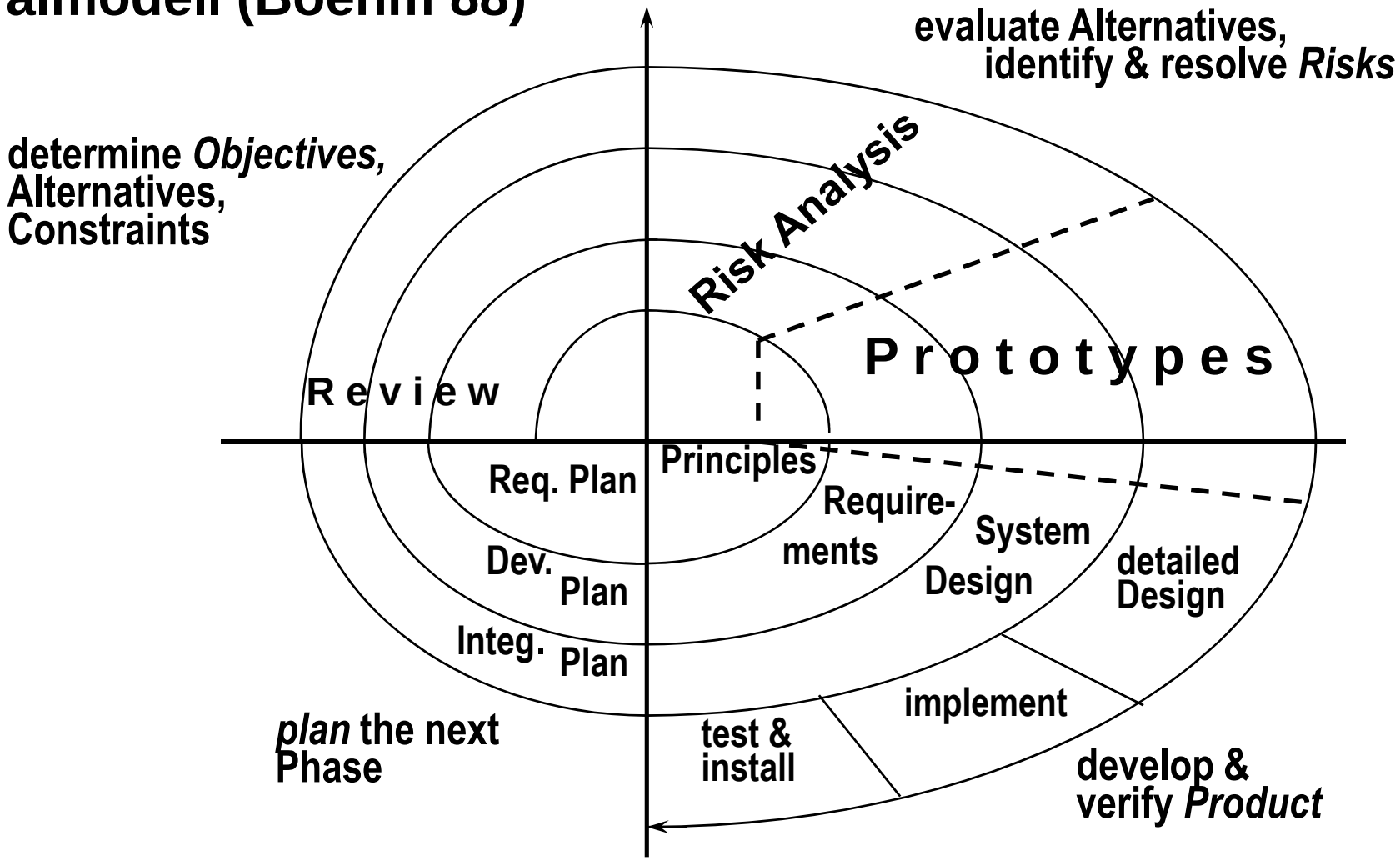
- **enge Rückkopplung mit Anwender ermöglicht bessere Umsetzung der Anforderungen**

- **weiterhin mangelnde Möglichkeiten zum Rücksprung (analog dem Wasserfall)**

- **Besondere Problematik ergibt sich durch die häufigen Inkremente:**
  - **größerer organisatorischer Aufwand zum Einbinden der Inkremente, d.h. wann wird welches Inkrement eingebunden**
  - **Ausrichten der Architektur auf den inkrementellen Bau**

- **Inkremente können auch als evolutionäre Prototypen interpretiert werden.**

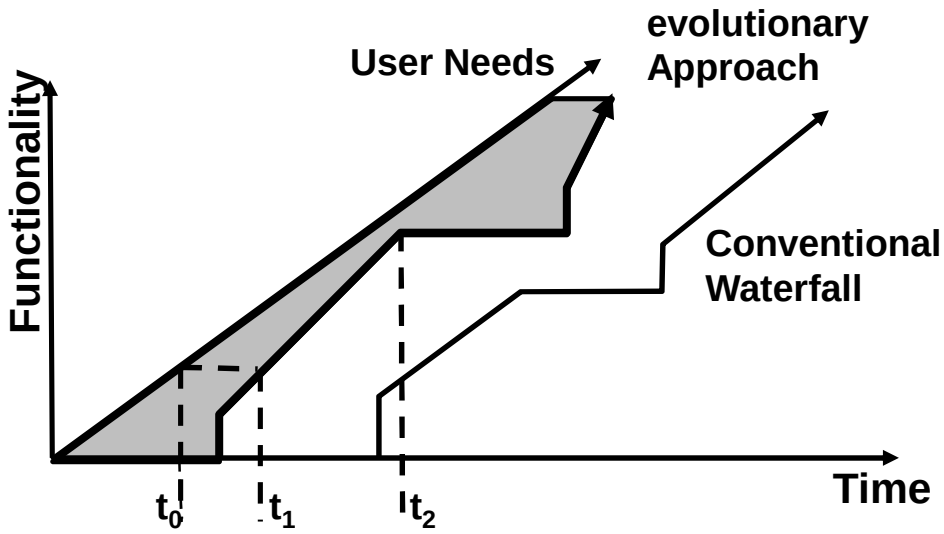
Spiralmodell (Boehm 88)



6.5 Evolutionäre, inkrementelle Modelle

Spiralmodell

evolutionäres Vorgehen entsprechend dem Spiralmodell



evolutionäres Vorgehen nach dem Spiralmodell gegenüber dem „normalen“ Wassermodell.

Ummittelbar nach Projektstart (t_0) entsteht ein erstes Minimalsystem.
Die Anforderungen zum Projektstart können sehr schnell im Zeitpunkt t_1 erfüllt werden. Der Abschluss des Systems (zum Zeitpunkt t_2) wird sehr schnell erreicht.

6.5 Evolutionäre, inkrementelle Modelle

UNIVERSITÄT OSNABRÜCK

Bewertung:

- ***Big Bang*-Effekt kann verhindert werden**
- **enge Rückkopplung mit Anwender ermöglicht bessere Umsetzung der Anforderungen**
- **im Gegensatz zum Wasserfallmodell sehr gute Möglichkeiten zum Rücksprung (Iterationen sind fest eingeplant)**
- **ABER: der Projektverlauf ist schwer überprüfbar; die Schwierigkeit liegt im Abschätzen der noch benötigten Iterationen**
- **Besondere Problematik:**
  - **größerer organisatorischer Aufwand bei der Planung der evolutionären Schritte, d.h. in welcher Iteration wird welche Funktionalität realisiert**
  - **Ausrichten der Architektur auf den evolutionären Bau**
  - **Durch Anpassung an die Änderungen der Anforderungen kann die ursprüngliche Architektur verloren gehen**
  - **Ein Projektende kann durch das Anpassen an neue Anforderungen nach hinten verschoben werden; ein dezidiertes Ende muss definiert werden**

6.6 Objektorientierte Modelle

Prozessmodelle finden sich in: [Boo94, RBP+91, JCJ92, CY91a, CY91b, You94].

[Boo94] G. Booch. *Object-Oriented Analysis and Design*, snd. Edition, Benjamin/Cummings, Redwood City, CA, 1994

[RBP+91] J. Rumbaugh, M. Blaha, W. Permerlani, F. Eddy, and W. Lorenson. Object-Oriented Modeling and Design. Prentice Hall, Englewood Cliffs, NJ, 1991.

[JCJ92] I. Jacobson, M. Christerson, and P. Jonsson. Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley, Wokingham, 1992.

[CY91a] P. Coad and E. Yourdon. Object-Oriented Analysis, Second Edition. Yourdon Press, Prentice Hall, Englewood Clis, NJ, 1991.
[CY91b] P. Coad and E. Yourdon. Object-Oriented Design. Yourdon Press,Prentice Hall, Englewood Clis, NJ, 1991.

[You94] E. Yourdon. Object-Oriented Systems Design. Yourdon Press, Prentice Hall, Englewood Clis, NJ, 1994.

6.6 Objektorientierte Modelle

Besonderheiten

- **Der Prozess der objektorientierten Software-Entwicklung wird in vielen objektorientierten Methoden als ein in Phasen eingeteiltes Vorgehen beschrieben** [Boo94, RBP+91, JCJ92, CY91a, CY91b, You94]**.**

- **Konvergenz der Methoden im Vorgehen:**
  Entwicklungsprozess immer in die Abschnitte
  Konzeptualisierung, Analyse, Design, Implementierung und Wartung
  unterteilt.
  (Teilweise haben die Abschnitte andere Namen.) [HSG96, Dod96]

- **Zusammenschluss der Methoden von G. Booch, J. Rumbaugh und I. Jacobson als Unified Method (UM) unter dem Dach von Rational Software Corporation deutlich [BR95]**

6.6 Objektorientierte Modelle

Alternative vereinigte Prozessmodelle werden vorgestellt von:

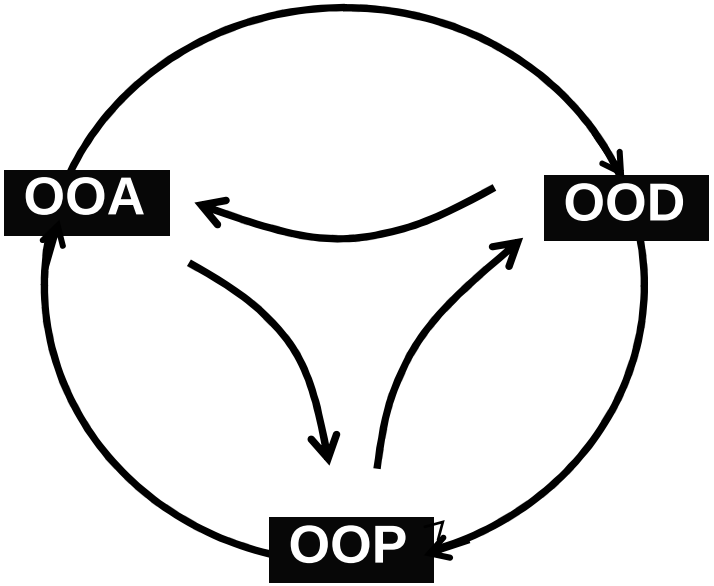
- **Grand Unified Method (GUM):**
  [Dod96] M. Dodani. Object-Oriented Methodologies in Practice: The"Big Picture". Journal of Object-Oriented Programming, 9(2):26 - 29, April 1996.

- **Open:**
  [HSG96] B. Henderson-Sellers and I. Graham. OPEN: toward method convergence. IEEE Computer, 28(4), April 1996.

OOA, OOD & OOP mit dem Baseball Prozessmodell

- Eingeführt von Coad, Yourdon, Nicola (1991)
- Fragenkataloge, Schichtung (Layers) mit fünf Detaillierungsgraden, eigene Notation
- OOA, OOD, OOP: Aktivitäten innerhalb der Methode mit speziell definierten Aufgaben und in nicht streng festgelegter Reihenfolge

Object-Oriented Analysis (Klassen und Objekte der realen Welt)



Object-Oriented Design (Mensch-Rechner-Interaktion, Taskmangement, Objekte und Klassen zur Datenverwaltung)

Object-Oriented Programming
(Strategien zur systematischen Implementierung von OOA/OOD Ergebnissen)

OOA, OOD & OOP mit dem Baseball Prozessmodell

- Interdisziplinäre Teams: keine Trennung von OOA, OOD, OOP

- Vorschlag des Teamaufbaus:
  - 1 – 2 Experten der realen Welt
  - 2 Analyseexperten
  - 1 GUI-Experte
  - 1 Datenbankspezialist
  - 1 Experte für Taskmanagement (nur bei Real-Time-Systemen)
  - 2 – 3 Programmierer (Experten f. Programmiersprache und Umgebung)

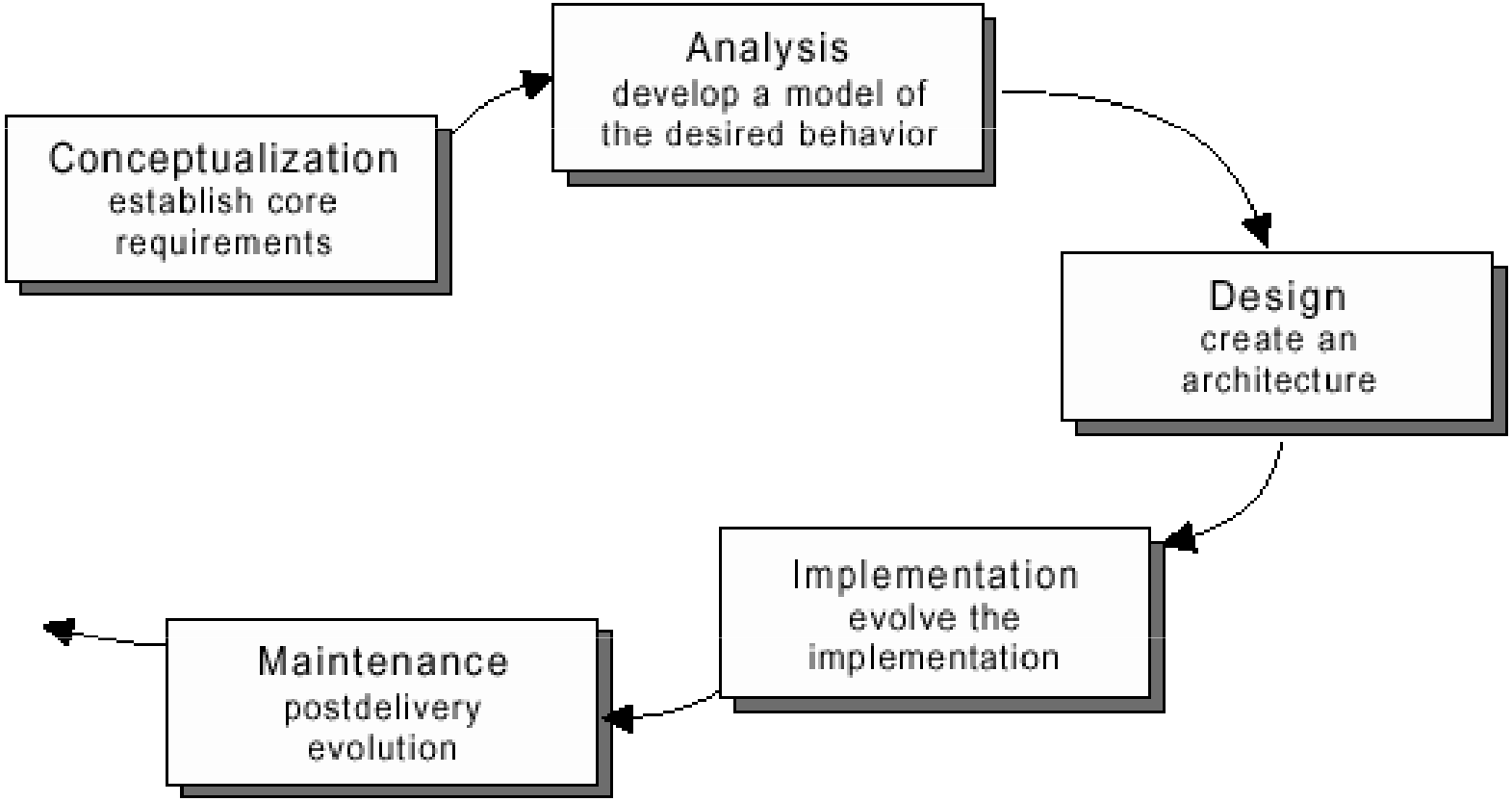
- Entwicklungstechnik basierend auf Fragenkataloge für die Aktivitäten:
  - Klassen und Objekte finden
  - Strukturen (Beziehungen) erkennen
  - Subjekte (Pakete) identifizieren
  - Attribute definieren
  - Services (Methoden) definieren

Prozessmodelle von G. Booch: Macro Process

- Eingeführt von Grady Booch (1994)
- Revolution im Ablauf (heute selbstverständlich): iterativ und inkrementell
- Großen Einfluß auf UML und RUP (Rational Unified Process)

[Boo94] G. Booch, Object-oriented Analysis and Design, Benjamin/Cummings, Redwood City, 1994

Prozessmodelle von G. Booch: Macro Process

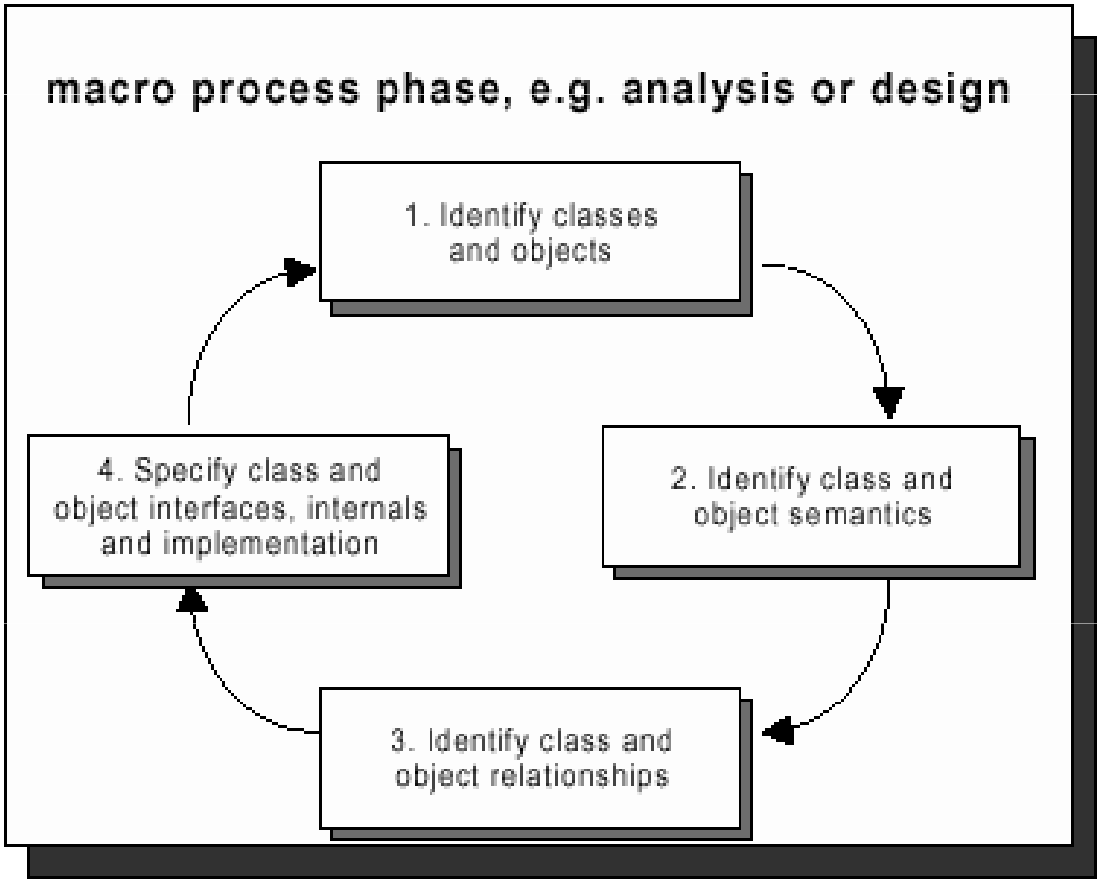


In späterer Version Beschränkung auf die Phasen:
Requirements Analysis, Domain Analysis, Design

[Boo94] G. Booch, Object-oriented Analysis and Design, Benjamin/Cummings, Redwood City, 1994

Prozessmodelle von G. Booch: Micro Process

Einzelabläufe innerhalb der Phasen des Makroprozesses, insbesondere Analyse und Design



Unified (Software Development) Process (UP)

[Jacobson, Booch, Rumbaugh 1999]

2 Ebenenmodell

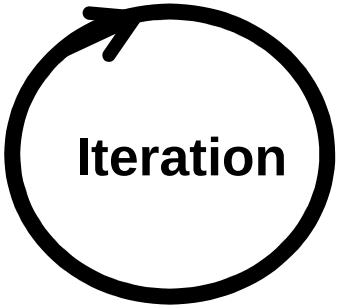
- **E1: Phasen**

  Inception (Projektsetup, Konzeptualisierung)
  Elaboration (Ausarbeitung)
  Construction (Durchführung)
  Transition (Umsetzung, Inbetriebnahme)

Zyklusende
⇒ Release

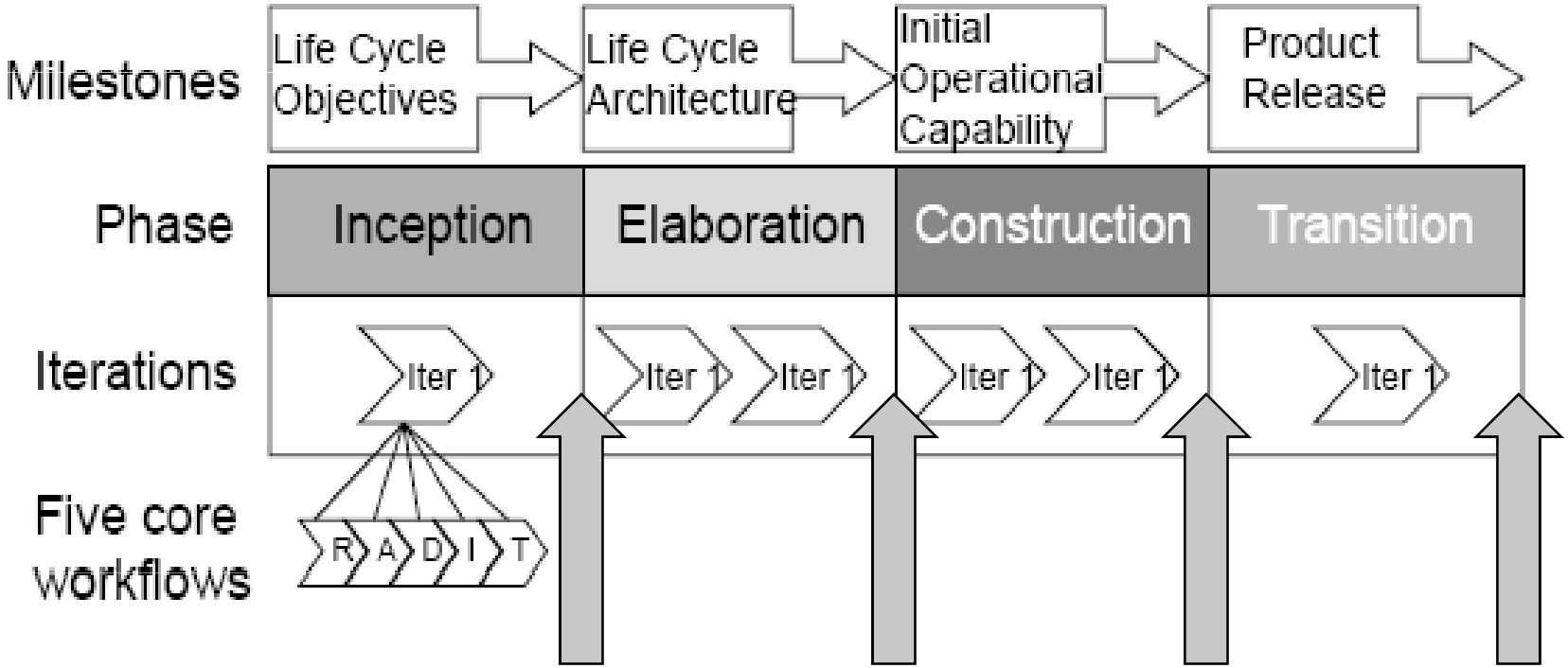
- **E2: Workflows** (Arbeitsablauf) (alle in jeder Phase)

  Requirements,
  Analysis,
  Design,
  Implementation,
  Test



- Konzentration auf Anwendungsfälle (Use Cases) und architekturzentriert

Unified (Software Development) Process (UP)



Management-sicht

1 bis n Iterationen pro Phase (technische Sicht)

pro Iteration: inkrementelle Weiter-entwicklung der für die Phase relevanten Artefakte

Meilensteine

(unterschiedliche Intensität der fünf Workflows je Phase)

[angelehnt an TU Graz, Institut für Softwaretechnology]

6.6 Objektorientierte Modelle

Begriffe in UP:

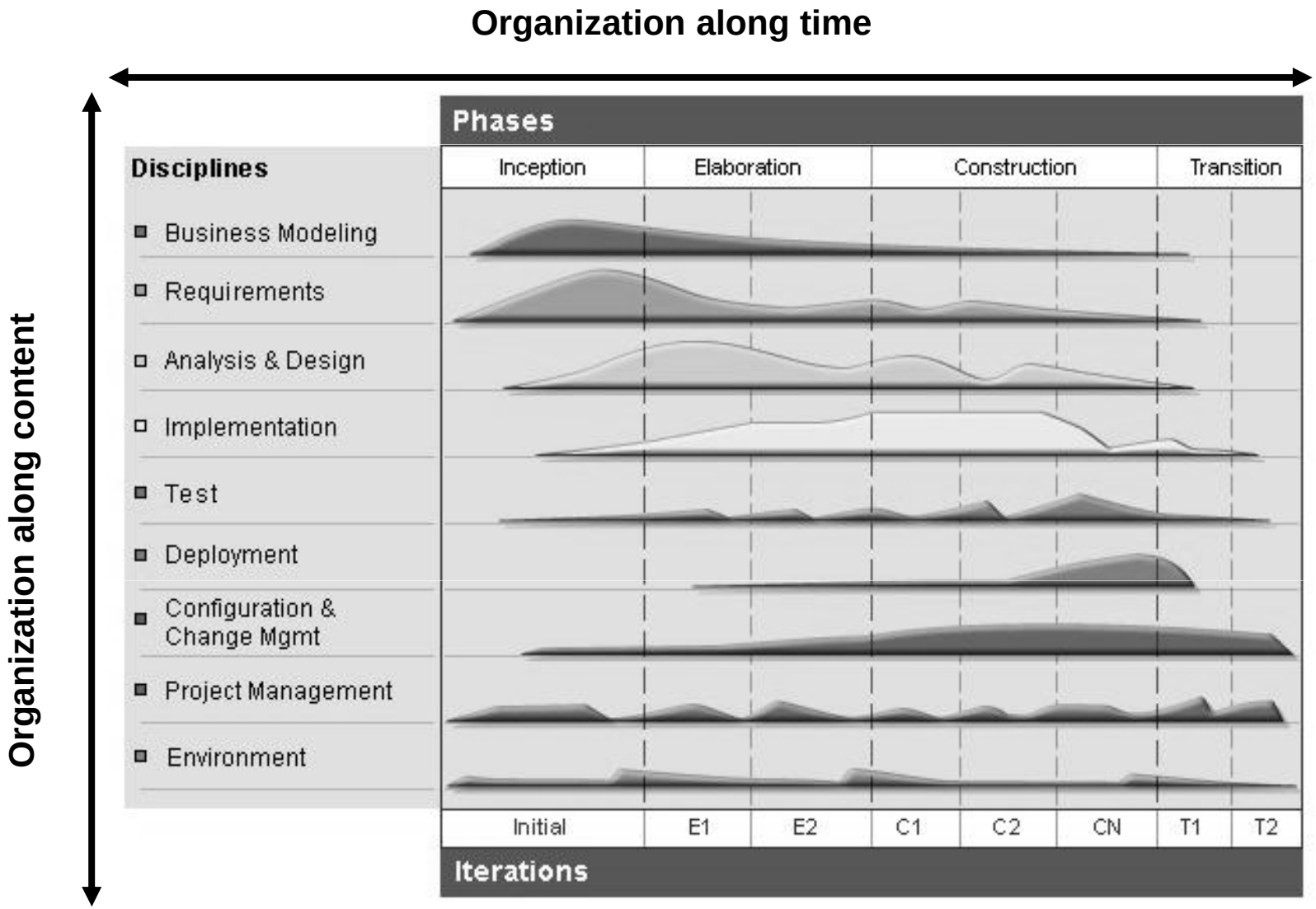
- **Worker / Rolle** = zusammengehörende Aufgaben, Verantwortlichkeiten und Fähigkeiten
- **Aktivität** = zusammenhängende Tätigkeit (Arbeitseinheit), die von einem Worker in einem Workflow ausgeführt wird
- **Artefakt** = explizit formulierte Information (Modell oder Dokument); werden von Worker erstellt, benutzt, modifiziert
- Beispiel: Worker „System Analyst“ hat die Aufgabe „Find Actors and Use Cases“ und verwendet das Artefakt „Business Model“, produziert oder erweitert die Artefakte „Use Case Model“ und „Glossary“
- Definition für Worker, Aktivitäten, Artefakte müssen für ein vollwertiges Prozessmodell angepasst und erweitert werden.

Rational Unified Process (RUP)

- Ausprägung des UP zu einem vollwertigen Prozeßmodell (Workers, Aktivitäten, Artefakte erweitert und konkretisiert, Core Workflows modifiziert und erweitert)
- Von IBM/Rational vermarktet (Anwendung ist lizenzpflichtig)
- Umfangreiche Dokumentation, kontinuierliche Weiterentwicklung (Vorlagen, Best Practices, fachspezifische Anleitungen)
- Achtung: Abweichungen von der UP Terminologie (z.B. Workflow $\rightarrow$ Discipline)

6.6 Objektorientierte Modelle

Intensität der Workflows in den RUP Phasen:



6.6 Objektorientierte Modelle

Beispiel einer Detaillierung in RUP:

- Detaillierung der Workflows mittels UML-Aktivitätsdiagramme zur Definition und Visualisierung der Reihenfolge und Abhängigkeiten der einzelnen Aktivitäten eines Workflows
- Aktivitätsdiagramme zur Detaillierung haben einen Ein- und Austrittspunkt, Aktionssymbole, Entscheidungssymbole mit Bedingungen, Synchronisationsbalken zur Modellierung von Parallelität
- Aktionssymbole fassen i.A. eine Gruppe von Aktionen zusammen; tiefere Detaillierungsebene (Workflow Detail) detaillieren diese Gruppen ebenfalls in einer grafischen Notation

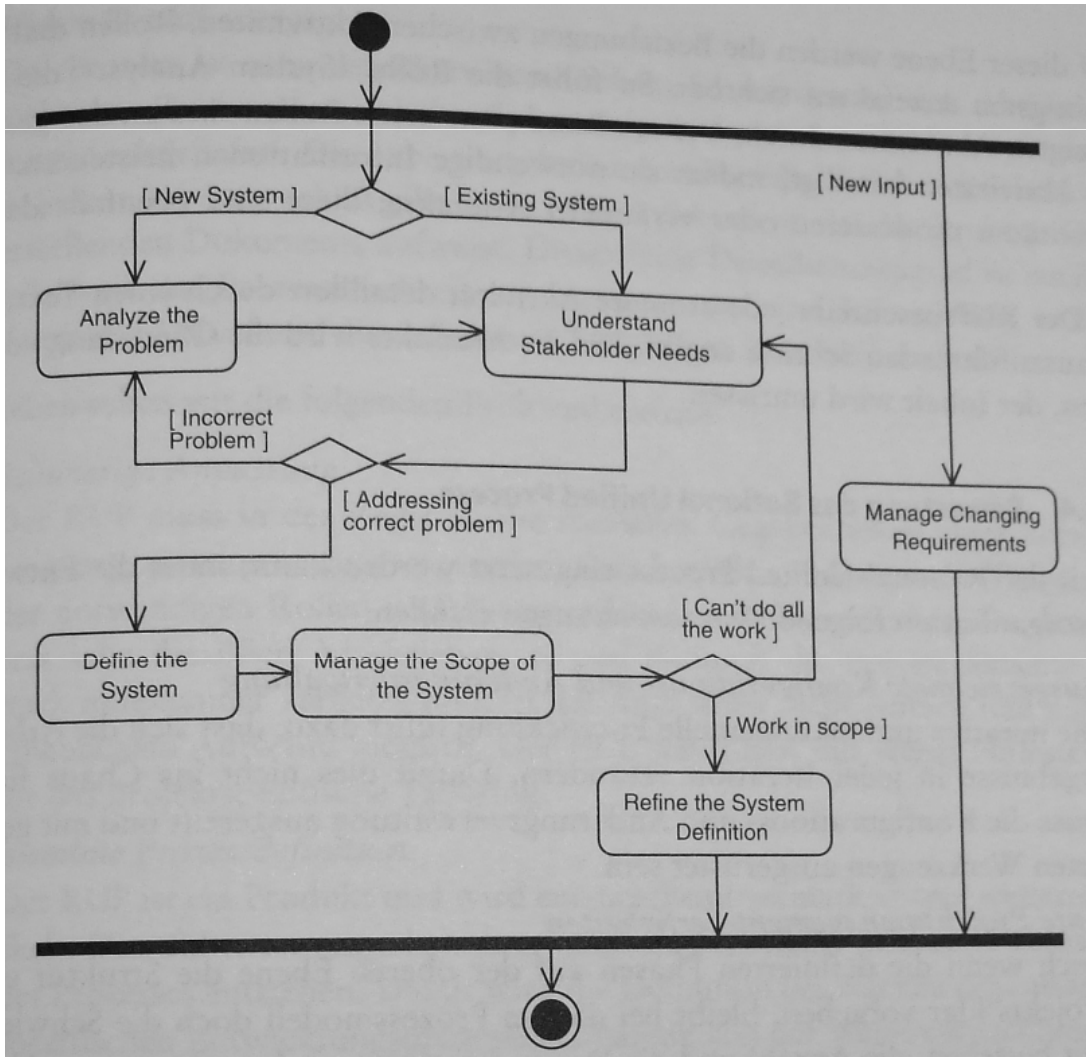
[Ludewig, Software Engineering, 2007]

6.6 Objektorientierte Modelle

Beispiel einer Detaillierung in RUP:

Eintrittspunkt

Aktivitätsdiagramm des Workflow „Requirements“



Synchronisations-balken (→ Parallelität)

Aktionssymbole

Entscheidungssymbole mit Bedingungen

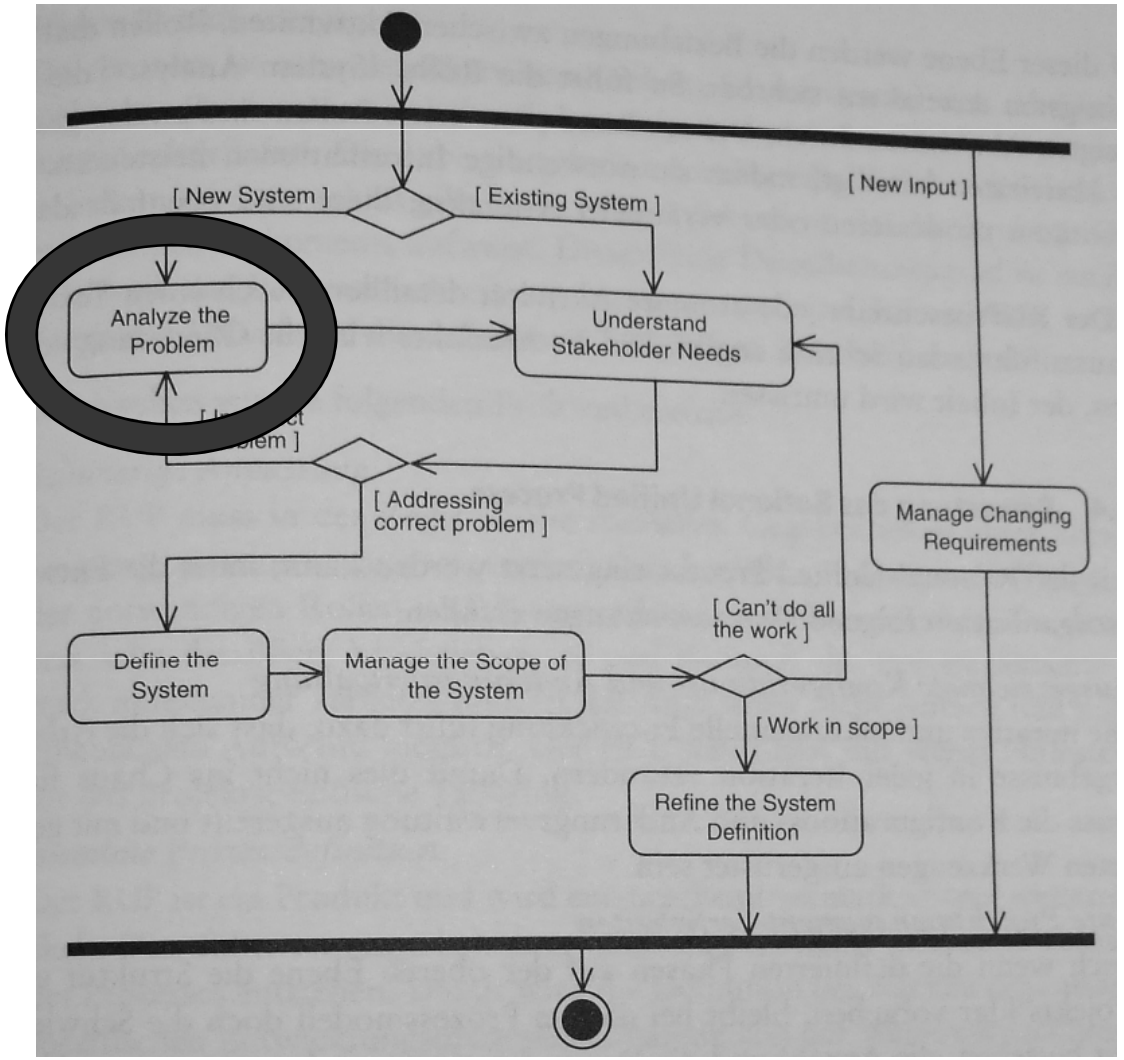
Austrittpunkt

[Ludewig, Software Engineering, 2007]

6.6 Objektorientierte Modelle

Beispiel einer Detaillierung in RUP:

Aktionsgruppe „Analyze the Problem“ (Workflow Detail)



[Ludewig, Software Engineering, 2007]

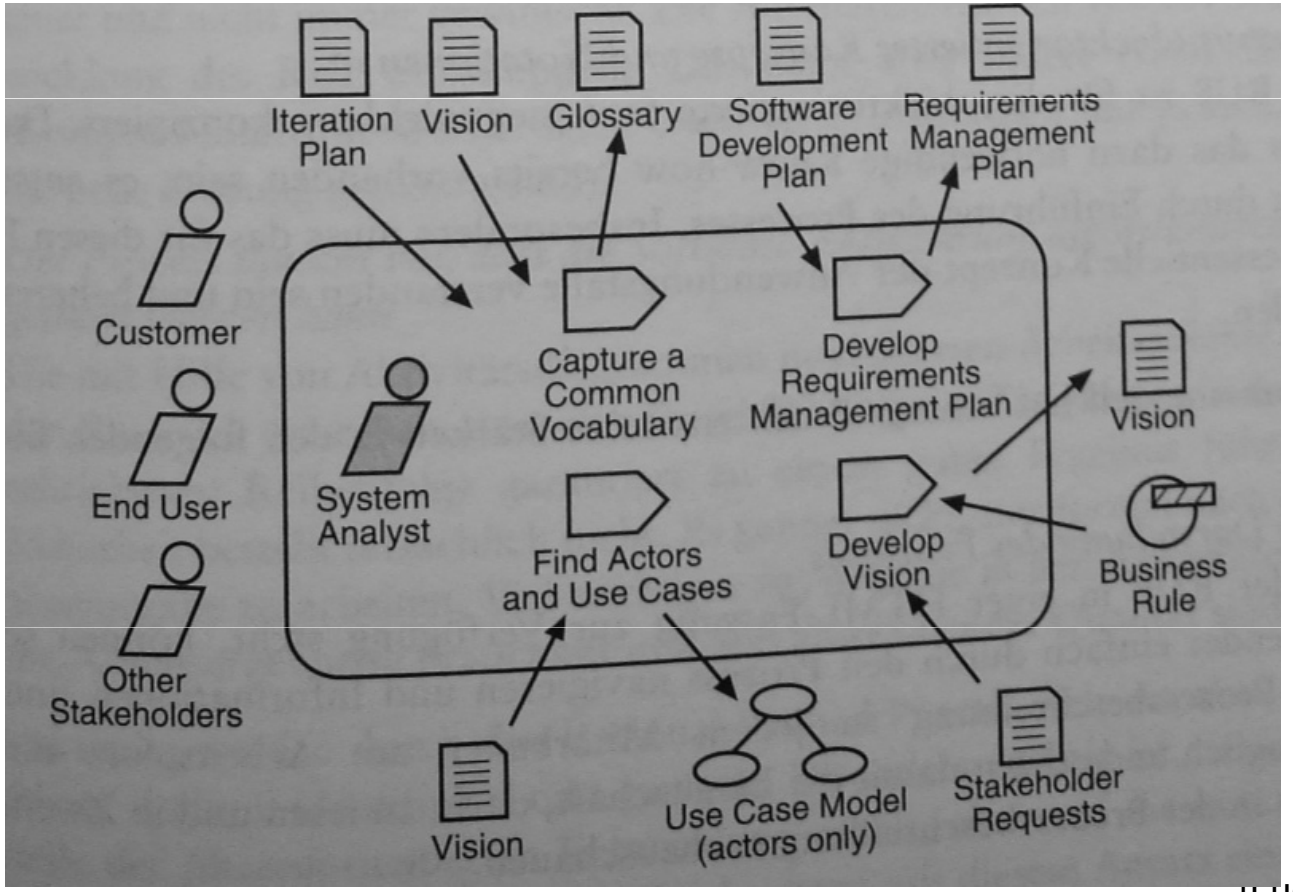
6.6 Objektorientierte Modelle

Beispiel einer Detaillierung in RUP:

Aktionsgruppe „Analyze the Problem“ (Workflow Detail)

Aktionen produzieren oder modifizieren Artefakte

An Aktionen beteiligte Worker



Artefakte: Gliederung + umrissener Inhalt

Atomare Aktion: textuell detailliert

[Ludewig, Software Engineering, 2007]

6.6 Objektorientierte Modelle

Tailoring des RUP:

- Wähle die Elemente des RUP Frameworks, die für das Projekt relevant sind
- Eliminiere unnötige RUP Elemente
- Ergänze das Vorgehensmodell um projektspezifische Elemente
- Passe, wo nötig, die ausgewählten Elemente an die Projekterfordernisse an

6.6 Objektorientierte Modelle

Bewertung von RUP:

- Sammlung von Best Practices und Vorlagen, häufig verwendet, kommerzielle Unterstützung, Werkzeuge, umfangreiche Dokumente mit HTML Navigation, genaue Arbeitsanleitung und Ergebnisstrukturierung (Musterdokumente), enge Anbindung an UML
- Herausforderung für Konfigurations- und Änderungsmanagement sowie für das Projektmanagement (wegen iterativer, inkrementeller Entwicklung)
- Kontinuierliche Prozessentwicklung (Aktualität aber auch Instabilität)

Bewertung von RUP:

- **Setzt OO und Anwendungsfallwissen voraus**
- **Schwergewichtig:**
  über 30 Rollen für über 130 Aktivitäten vorgesehen und es werden über 100 verschiedene Artefakttypen (Arbeitsergebnistypen) vorgeschlagen, die erzeugt, dokumentiert und verwaltet werden müssen.

  - RUP kann sehr weitgehend reduziert und angepasst werden ("Tailoring").
  - XP-Plugin für RUP (Rational und Object Mentor 2002) …
- Anpassung ist aufwendig
- Evtl. Pseudo- bzw. Überformalisierung von kreativen Aktivitäten

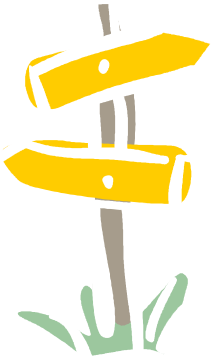
Softwareprozesse, -vorgehensmodelle, -lebensläufe

Zusammenfassung und Ausblick

1 Software-Krise und Software Engineering
2 Grundlagen des Software Engineering
3 Projektmanagement
4 Konfigurationsmanagement
5 Software-Modelle
6 Software-Entwicklungsphasen, -prozesse, -vorgehensmodelle
7 Qualität
8 … Fortgeschrittene Techniken

- 6.1 Softwareentwicklung in Phasen
- 6.2 Unsystematische “Modelle”
- 6.3 Lineare, sequentielle Modelle
- 6.4 Frühe Prototypen (Rapid Prototyping)
- 6.5 Evolutionäre, inkrementelle Modelle
- 6.6 Objektorientierte Modelle

Bekannte Vorgehensmodelle



→ Wege im Umgang mit der Software-Krise und Umsetzung der Grundlagen und Prinzipien:
Einsatz von Modellen: weitere bekannte Vorgehensmodelle für verschiedene Umgebungen