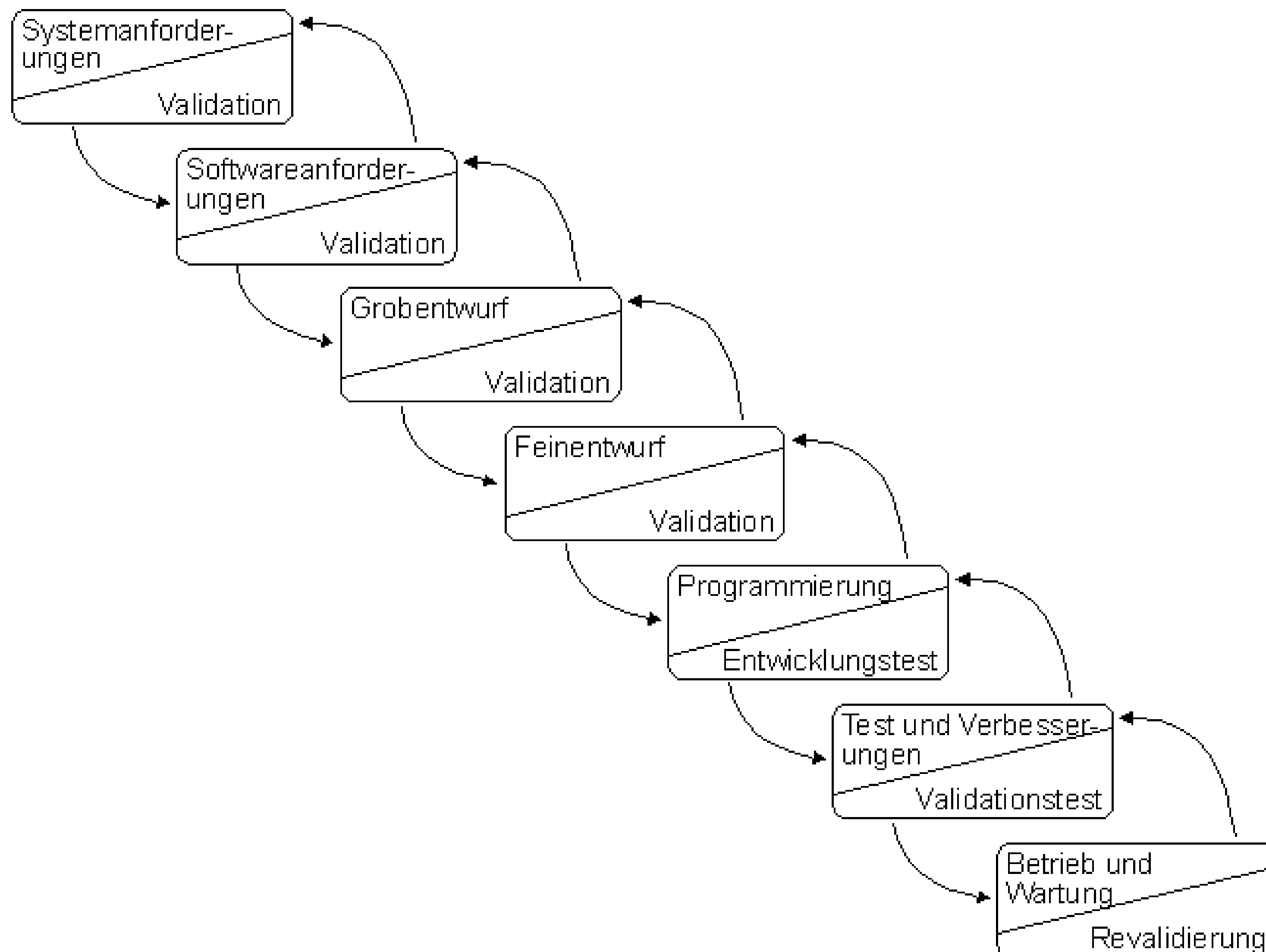
Software-Engineering

Sebastian Iwanowski
FH Wedel

Kapitel 6: Projektmanagement

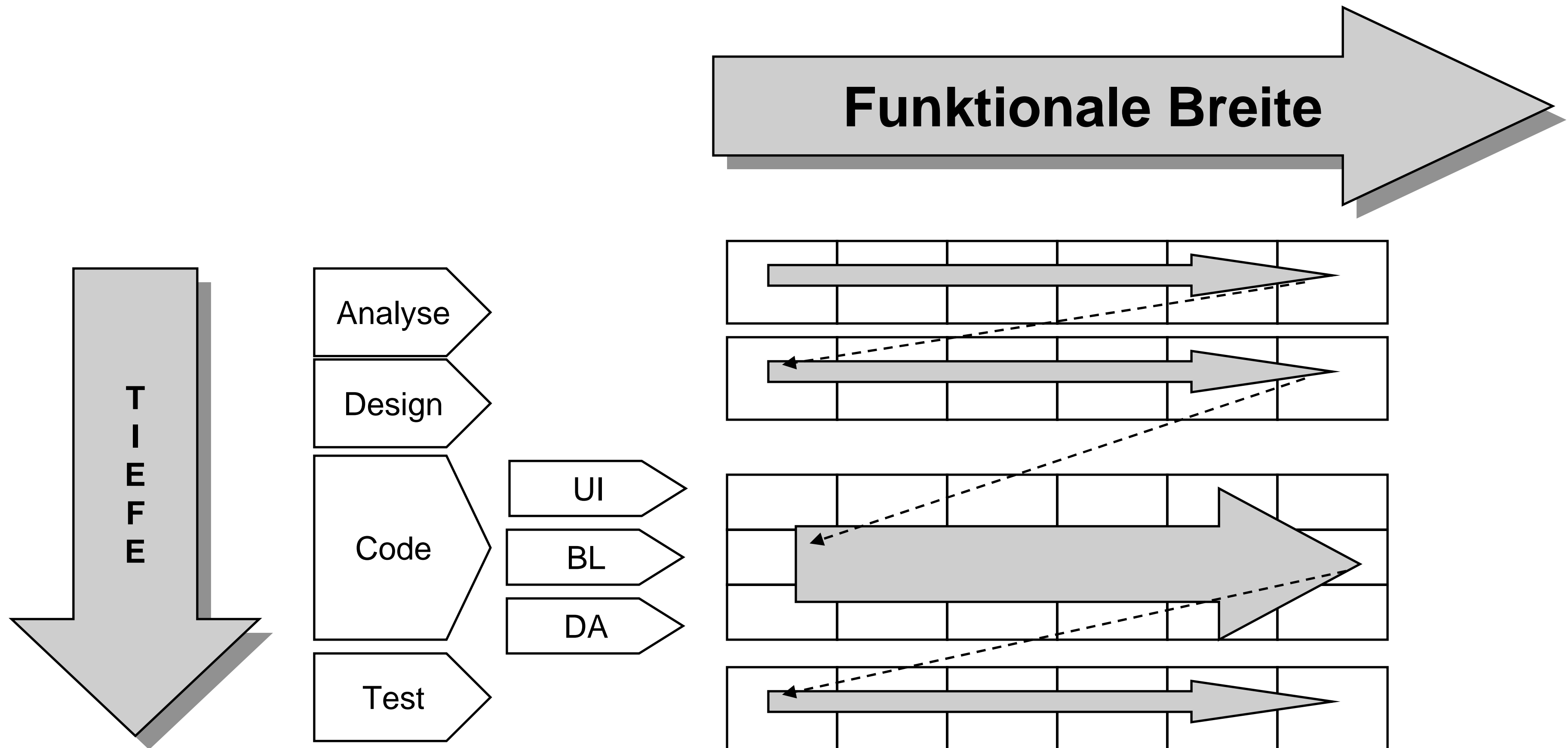
Wasserfallmodell



© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Wasserfallmodell

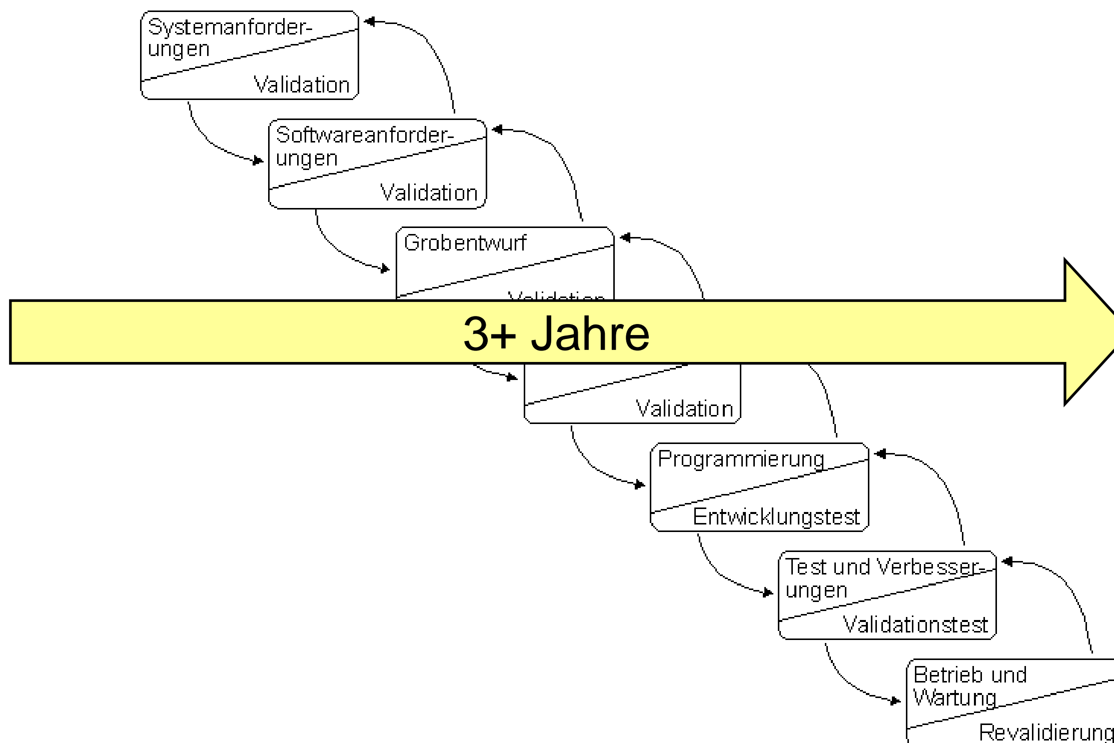
Zeitliche Einordnung in Breiten-/Tiefenraster



Vorlesung „Software Engineering für große, betriebliche Informationssysteme“,
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Wasserfallmodell

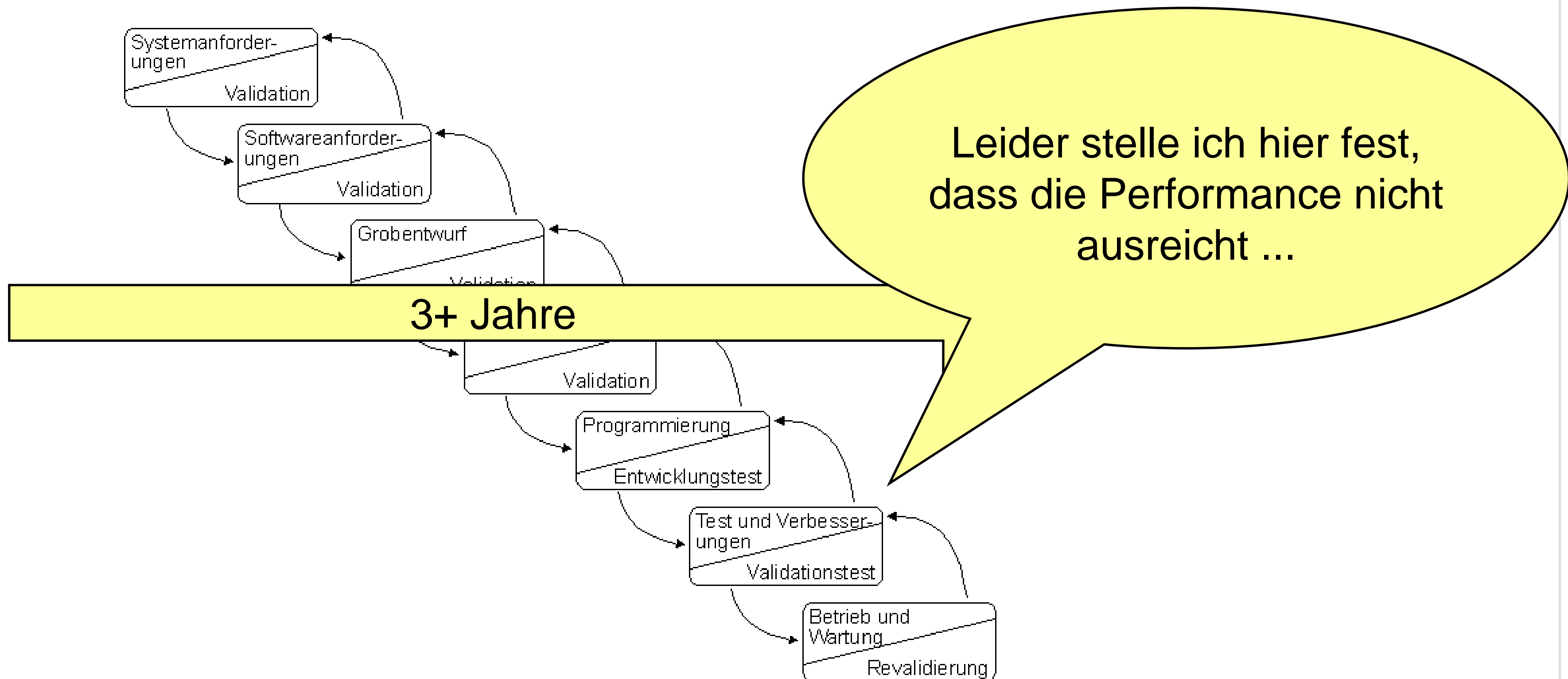
… ist für langfristige Produktentwicklungen gedacht



Vorlesung „Software Engineering für große, betriebliche Informationssysteme“,
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Wasserfallmodell

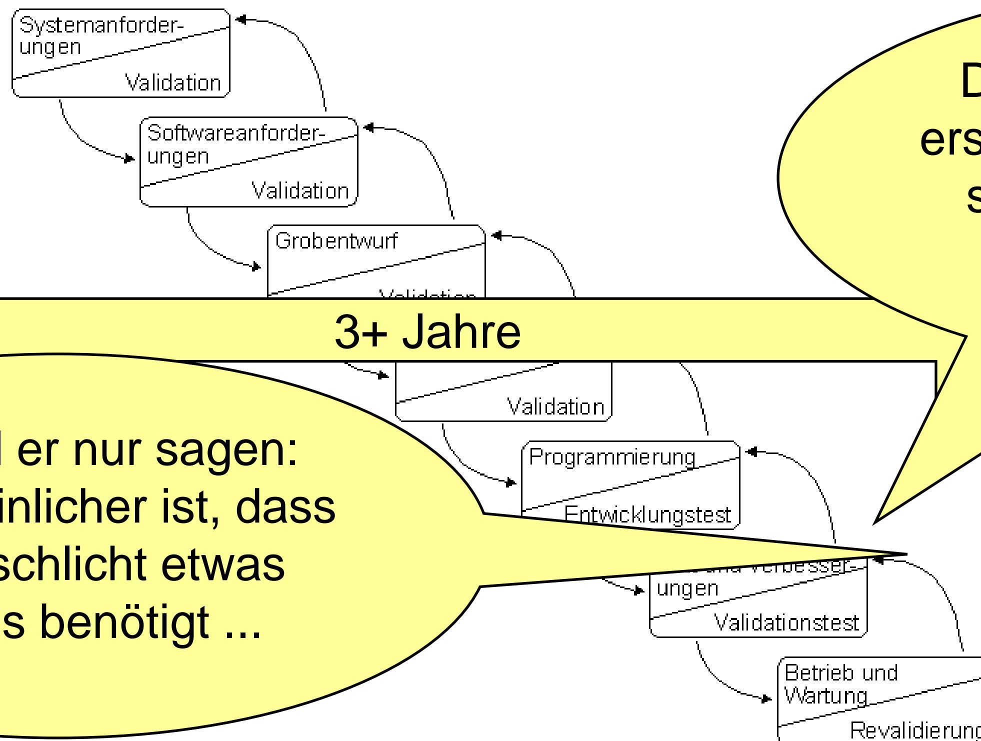
Langfristig heißt: Mit späten Konsequenzen …



Vorlesung „Software Engineering für große, betriebliche Informationssysteme“,
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Wasserfallmodell

Für die Softwareentwicklung: Mit <u>zu</u> späten Konsequenzen …



3+ Jahre

Der Benutzer sieht zum ersten Mal das System und stellt fest, dass er es ja damals ganz anders gemeint hat

Das wird er nur sagen: Wahrscheinlicher ist, dass er jetzt schlicht etwas anderes benötigt ...

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Wasserfallmodell

Vorteile

- Tätigkeiten in logischer Reihenfolge → leicht nachvollziehbar
- Tätigkeiten und Dokumente leicht standardisierbar → leicht zu managen
  → auch für große Teams geeignet
- Einfache Qualitätssicherung durch Validierung und Freigabe der einzelnen Phasen vor Start der nächsten

Vorteile für Softwareprojekte nicht ausschlaggebend

Wasserfallmodell

Nachteile

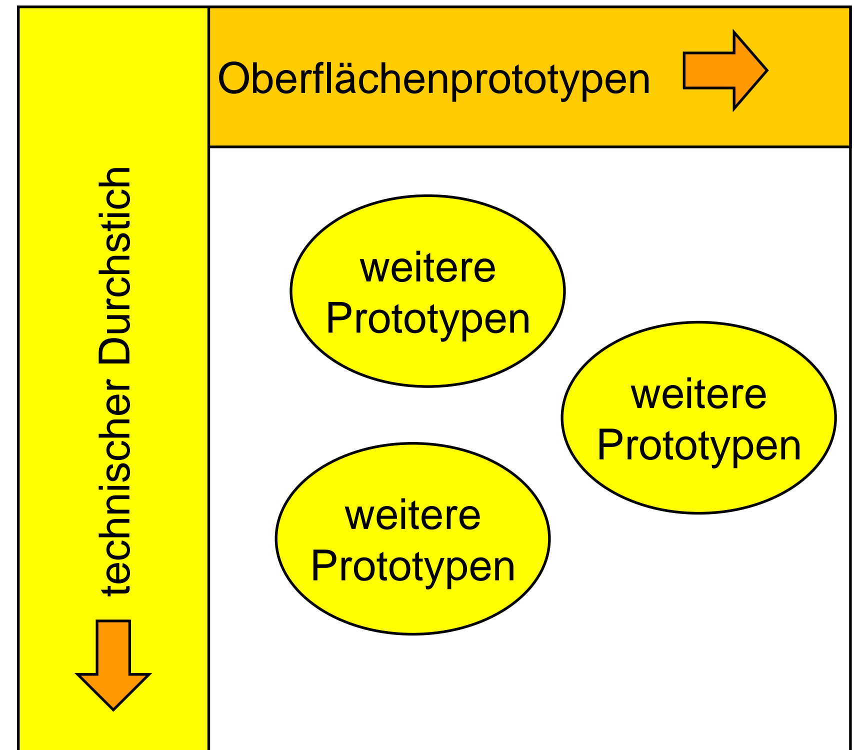
- Kunde sieht Produkt viel zu spät
- Kunden bezahlen für Software, nicht für die Spezifikationen
- Gesamtlaufzeit des Projekts so lange, dass Änderungen erforderlich sind. Änderungskosten um so größer, je später die Änderung erforderlich ist (leider wird sie dann immer wahrscheinlicher)
- Bisherige Dokumentation durch Änderungen wertlos
  → viel Arbeit für den Papierkorb

Nachteile sind KO-Kriterien für Softwareprojekte !

Prototyping

Fokussierung auf verschiedene Aspekte

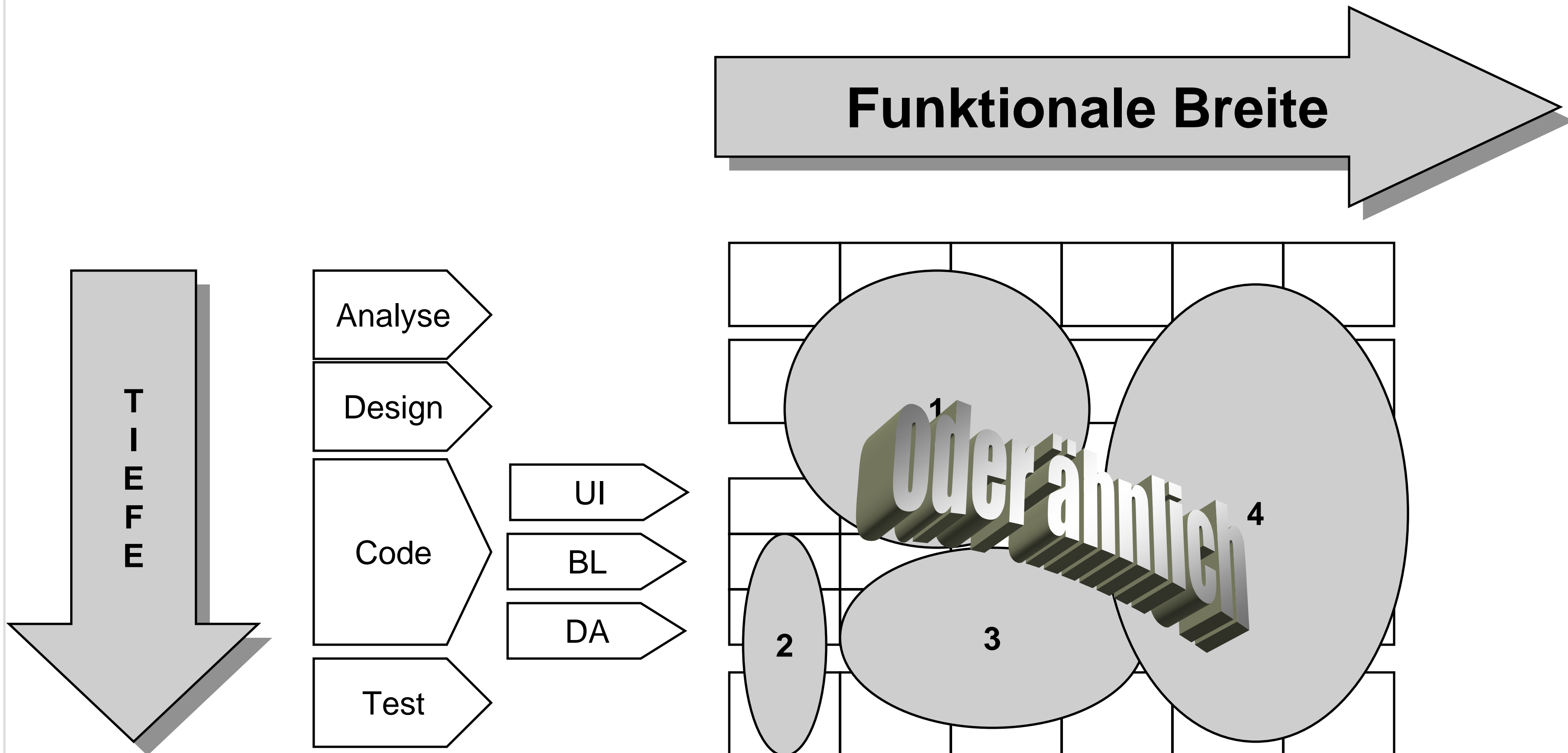
- Technische (vertikale) Prototypen zur Vermeidung von Performance-Überraschungen
- Oberflächenprototypen bietet frühe Kundeneinbindung
- Strategie: „hardest things first“



Vorlesung „Software Engineering für große, betriebliche Informationssysteme“,
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Prototyping

Zeitliche Einordnung in Breiten-/Tiefenraster



Vorlesung „Software Engineering für große, betriebliche Informationssysteme“,
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Prototyping

Vorteile

- Minimierung von Risiken
- Frühzeitig Rückmeldung vom Endbenutzer
- Lieferung von guten Basisdaten für Aufwandsschätzungen für das Gesamtsystem

Nachteile

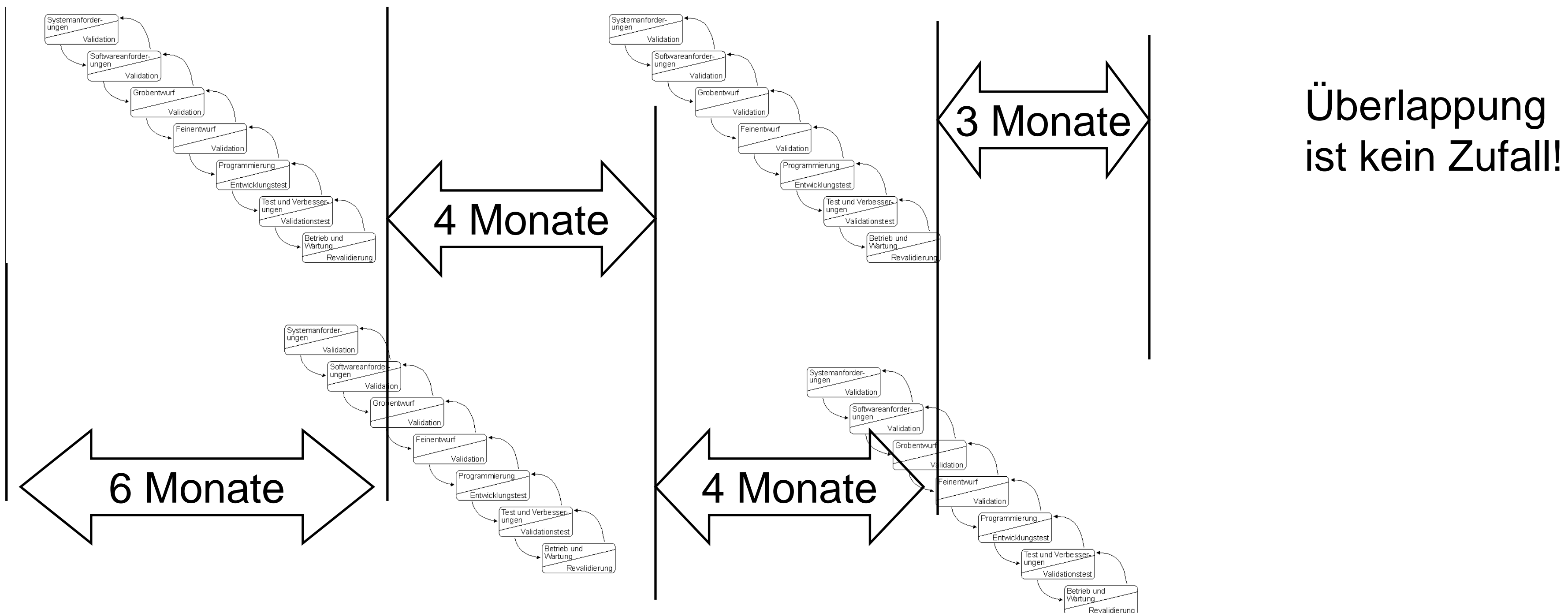
- Höherer Entwicklungsaufwand,
  weil Prototypen nicht 1:1 übernommen werden.
- Management / Kunde könnte Wegwerfprototypen
  als das fertige Produkt sehen.
- und damit läuft man Gefahr,
  beim Management / Kunde unter Druck zu geraten ...

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Spiralmodell

Wesentliches Problem des Wasserfallmodells: Moving Targets

Neue Idee: Aufteilen der Entwicklung in viele kurze Teilaufgaben

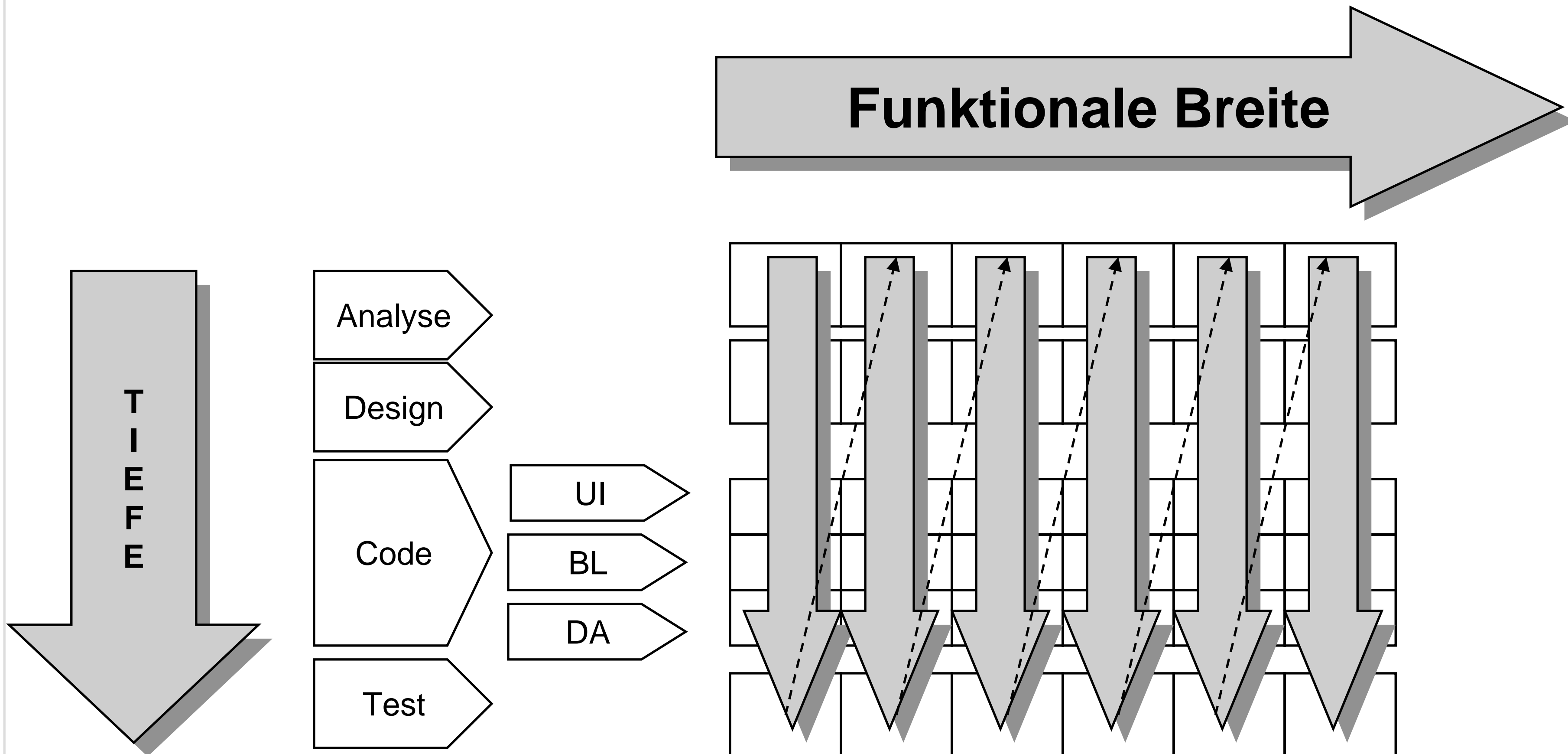


Überlappung
ist kein Zufall!

Vorlesung „Software Engineering für große, betriebliche Informationssysteme“,
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Spiralmodell

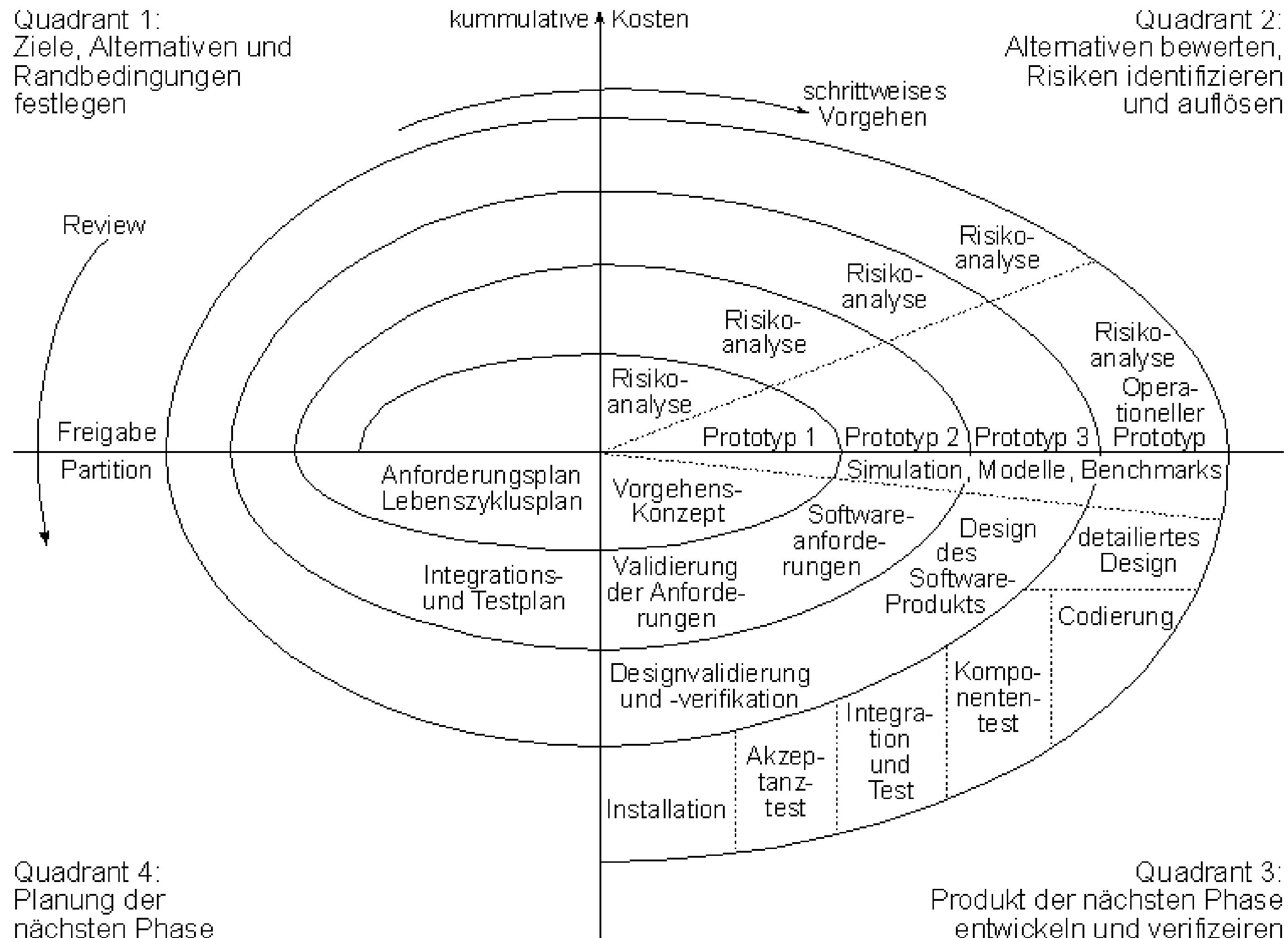
Zeitliche Einordnung in Breiten-/Tiefenraster



Vorlesung „Software Engineering für große, betriebliche Informationssysteme“,
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Spiralmodell

und so sieht es dann im Detail aus:



Vorlesung „Software Engineering für große, betriebliche Informationssysteme“,
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Spiralmodell

Vorteile

- Schnelles Feedback durch kurze Entwicklungszeiten möglich
  – auf Moving Targets kann reagiert werden.
- Pro Zyklus jeweils anderes Prozess- und Teammodell wählbar
- Fehler relativ schnell erkennbar
- bessere Eingriffsmöglichkeiten als beim Wasserfallmodell
- bewirkt automatisch eine gute top-down-Strukturierung der Software

Nachteile

- Managementaufwand sehr groß (für kleine Projekte unverhältnismäßig)
- Probleme, wenn grundlegende Eigenschaften geändert werden sollen (Gesamtlaufzeit wird ja nicht kürzer)

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Projektmanagement

Zeitliche Organisation des Projektablaufs

- Wasserfallmodell
- Prototyping
- Spiralmodell
- RUP: Rational Unified Process
- XP: eXtreme Programming

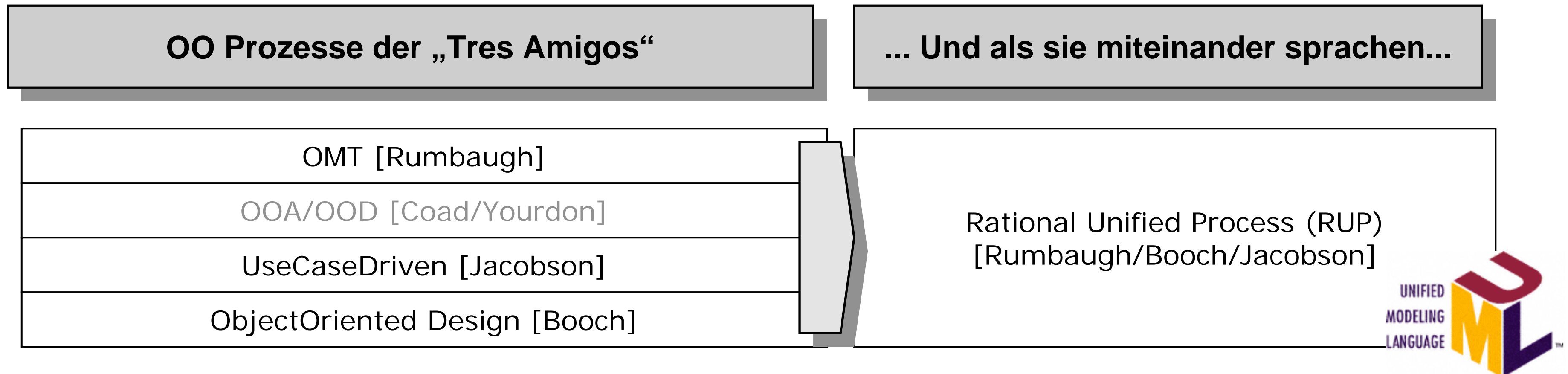
Grobe Zeitplanung für das Gesamtprojekt (Wasserfallmodell, Prototyping, Spiralmodell)

Detaillierte Zeitplanung für Teilphasen (RUP, XP)

mit Forderungen an Personalorganisation

RUP: Rational Unified Process

Historie

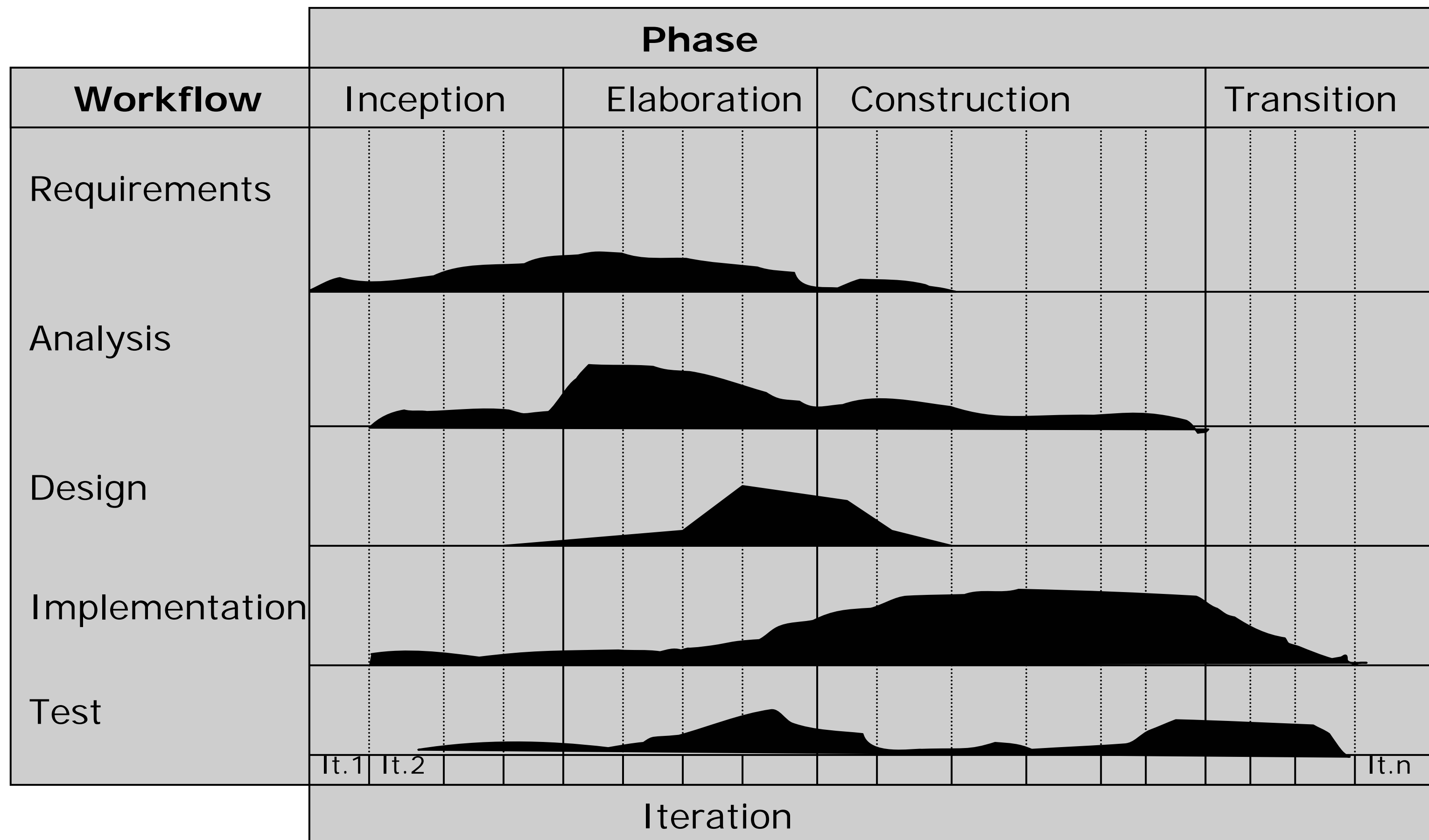


**RUP ist eine Anleitung,
wie man UML in der Projektorganisation bestmöglich einsetzt.**

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

RUP: Rational Unified Process

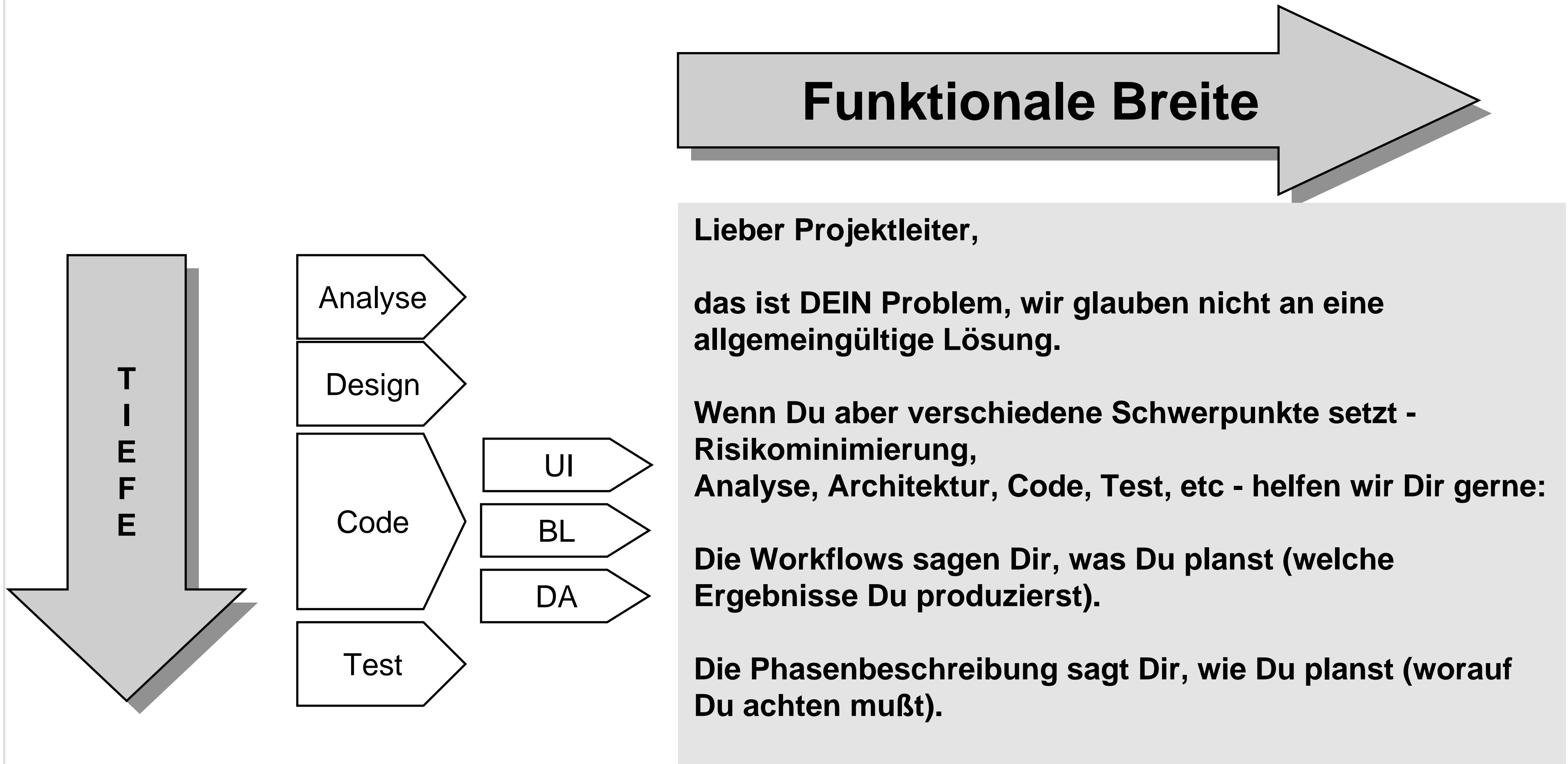
Aufteilung der Arbeitsschritte in bestimmte Phasen



Vorlesung „Software Engineering für große, betriebliche Informationssysteme“,
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

RUP: Rational Unified Process

Zeitliche Einordnung in Breiten-/Tiefenraster



© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

RUP: Rational Unified Process

Wesentliche Bausteine und Prinzipien

Use-Case-getrieben

- Use cases bilden die Basis für alle Phasen incl. Test
- Die Planung erfolgt nach Use-Case-Paketen und Fertigungstiefe

Inkrementell

- Das System wird in Iterationen errichtet
- Jede Iteration kann etwas mehr als die Vorgängeriteration

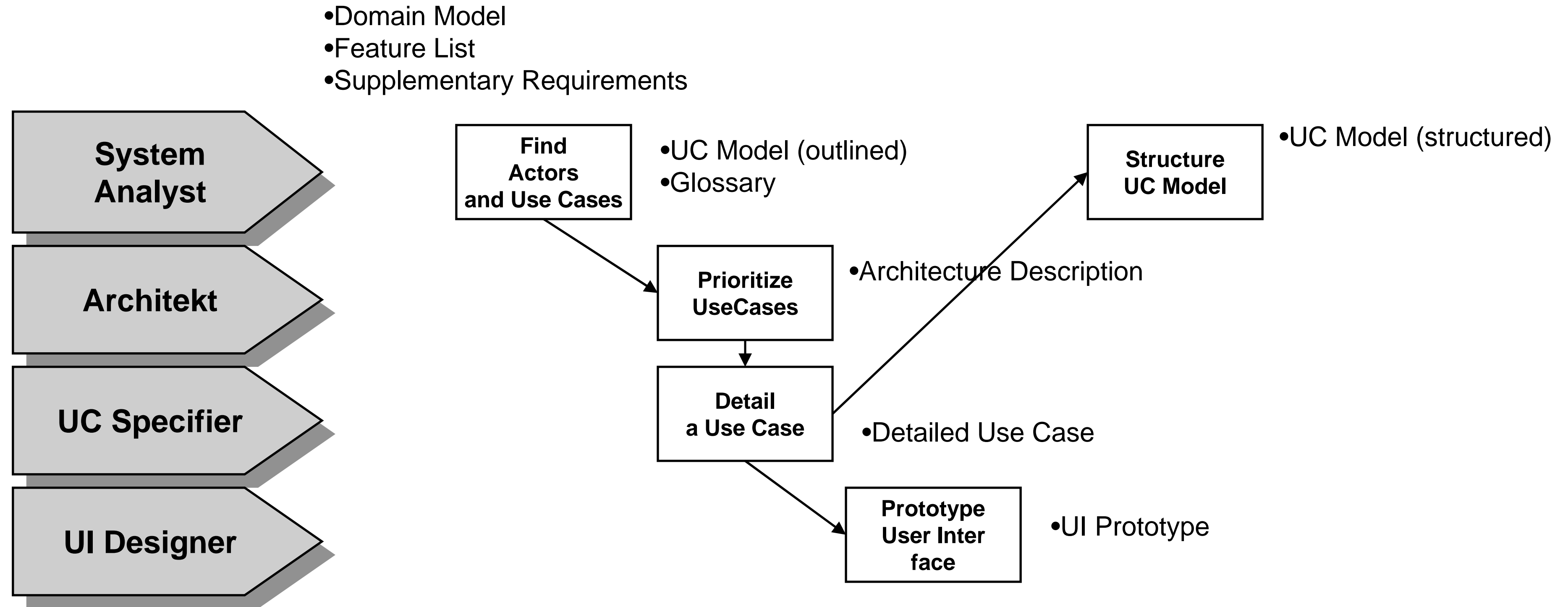
Architektur Basiert

- Für kritische Use Cases wird eine Musterlösung erstellt
- Diese Lösung dient als „Schema F“ für die weiteren Use Cases
- Diese Lösung heißt „Architektur“

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

RUP: Rational Unified Process

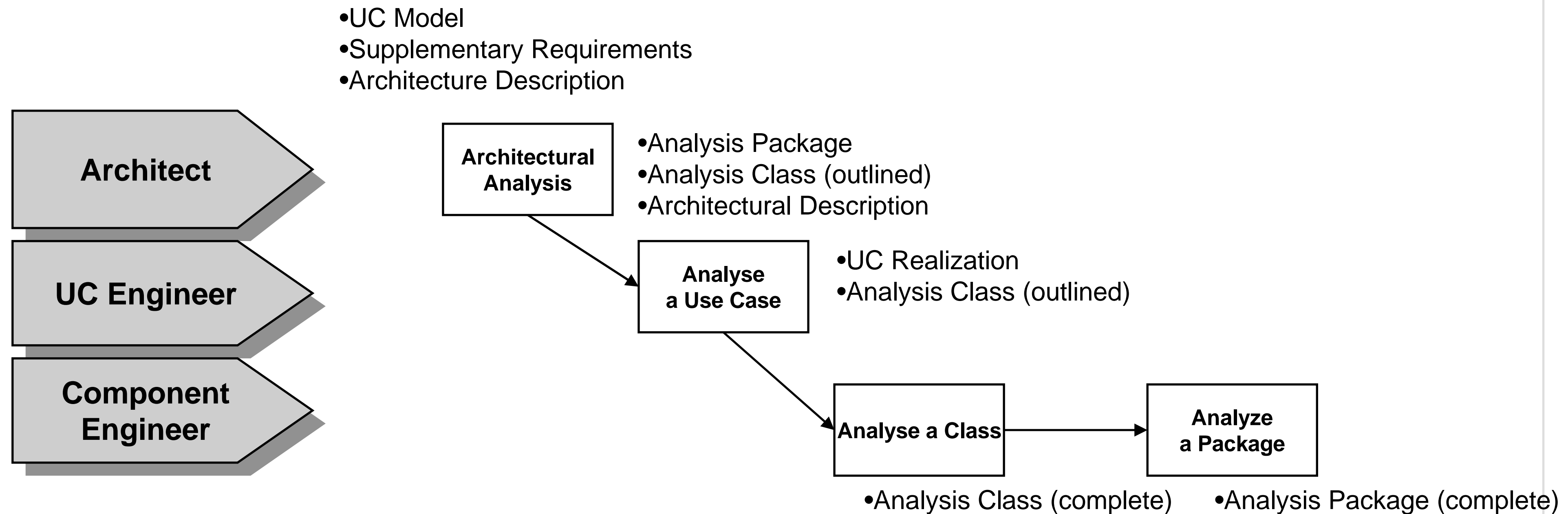
Requirements Workflow



Vorlesung „Software Engineering für große, betriebliche Informationssysteme“,
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

RUP: Rational Unified Process

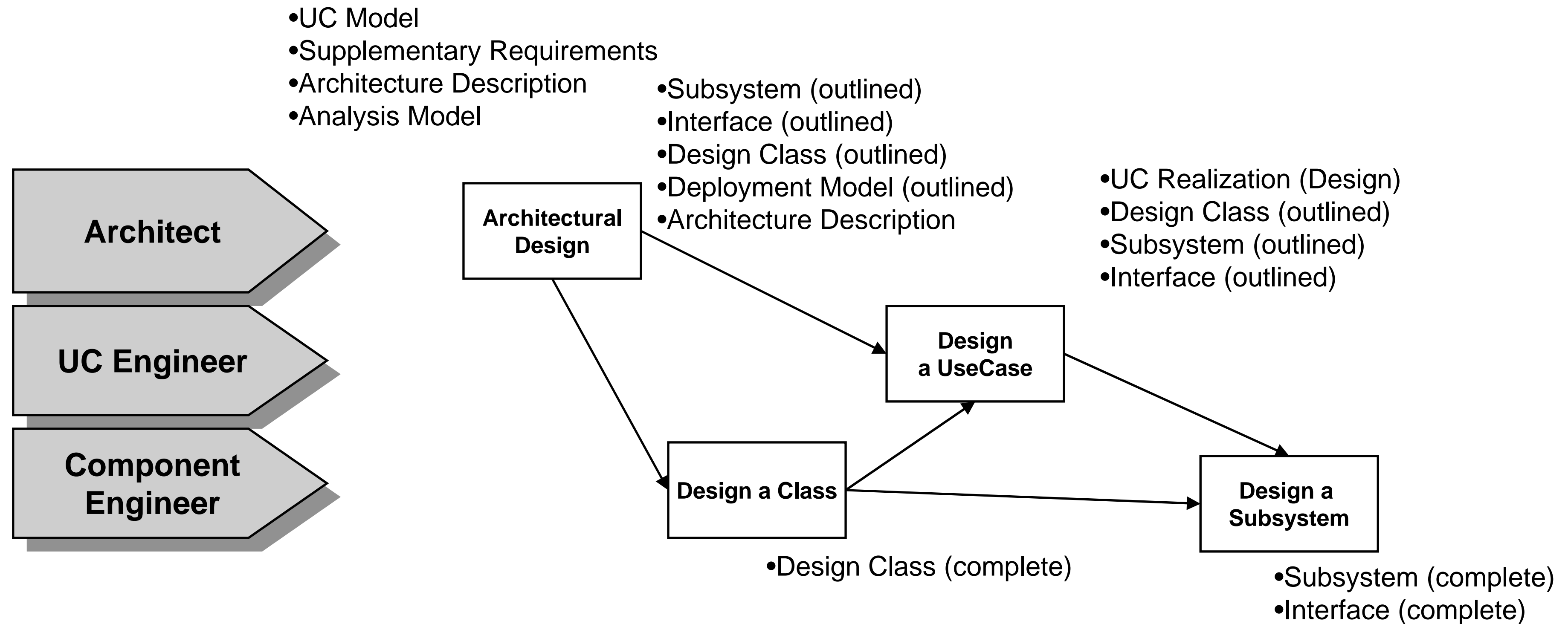
Analysis Workflow



© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

RUP: Rational Unified Process

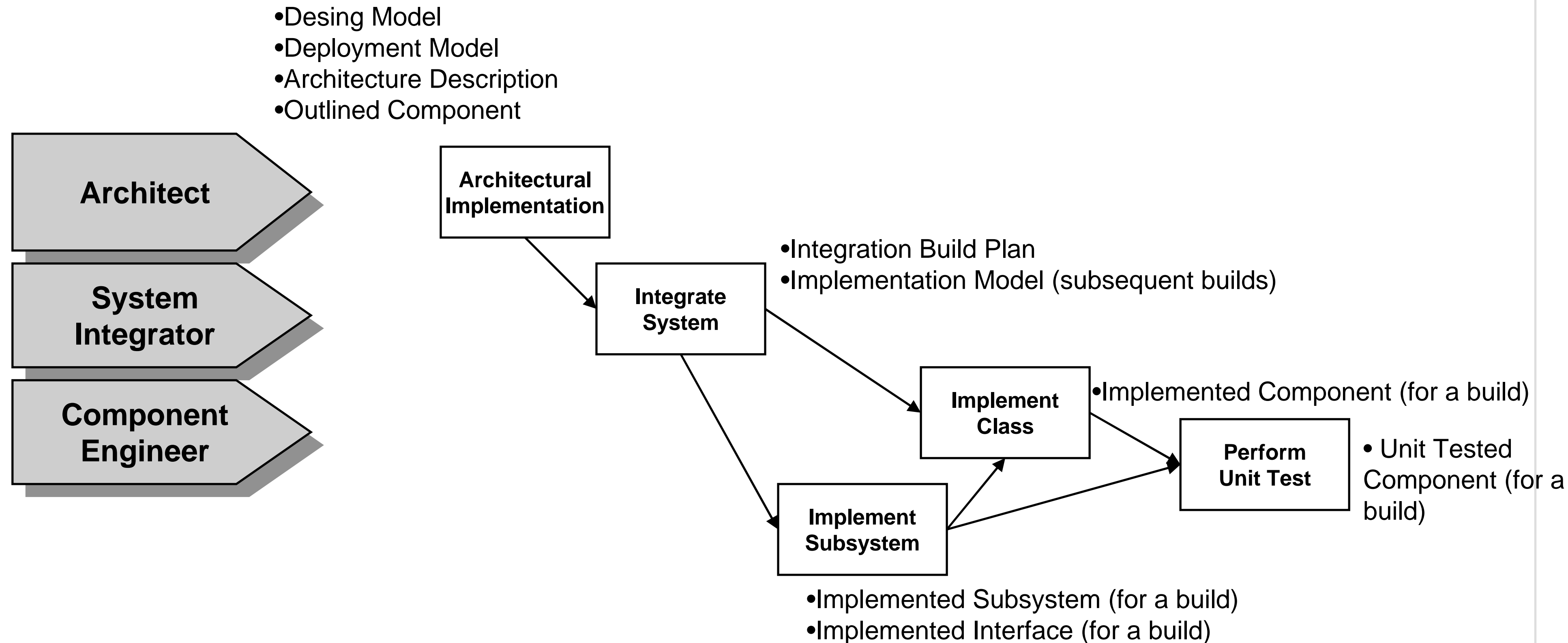
Design Workflow



Vorlesung „Software Engineering für große, betriebliche Informationssysteme“,
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

RUP: Rational Unified Process

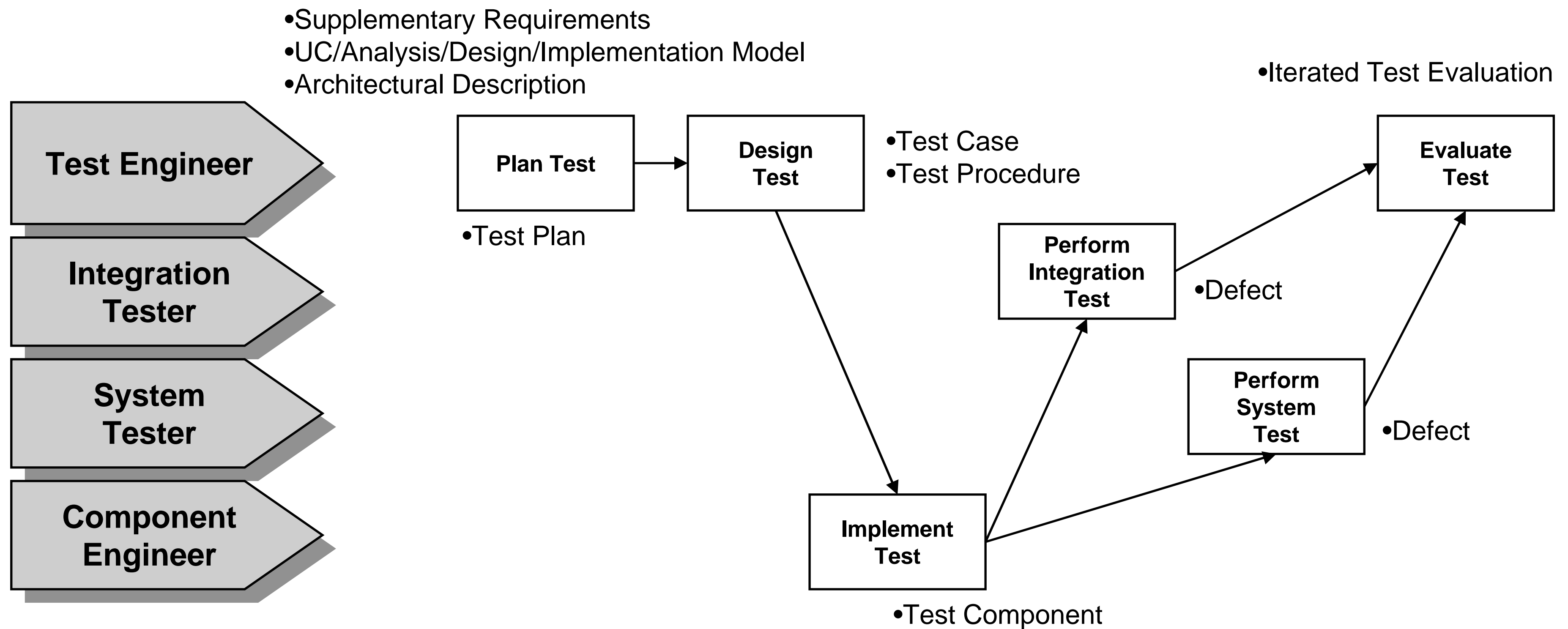
Implementation Workflow



Vorlesung „Software Engineering für große, betriebliche Informationssysteme“,
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

RUP: Rational Unified Process

Test Workflow



Vorlesung „Software Engineering für große, betriebliche Informationssysteme“,
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

XP: eXtreme Programming

Historie

- propagiert durch Kent Beck (Smalltalk-Softwareentwickler) 1999
- weitere Weggefährten: Martin Fowler, Erich Gamma, Ralph Johnson

Wesentliche Prinzipien

- XP stellt die **Programmierung** in den Vordergrund – der Code ist das, wofür bezahlt wird
- XP nimmt Moving Targets als den Normalfall an. Änderungen werden nicht bekämpft, sondern übernommen: **embrace change**
- Bei XP steht im Verhältnis Auftraggeber und Auftragnehmer **Vertrauen** im Vordergrund ...

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

XP: eXtreme Programming

Bestandteile von XP

Code Reviews sind gut:

- Also werden Sie dauernd gemacht.
- **Pair Programming**
- Es gibt informelle **Coding Standards**

Permanentes Testen ist gut:

- Also werden die Testfälle ständig auf dem Stand gehalten und ausgeführt.
- Use Cases dienen auch als Testscripts für **Unit Testing**.
- **Testfälle werden gebaut, bevor codiert wird.**

Integrationstests sind wichtig:

- Deshalb wird permanent integriert und wieder getestet: **Continuous Integration**

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

XP: eXtreme Programming

Bestandteile von XP

Einfachheit ist gut:

- **„Do the simplest thing that might possibly work“**

Design ist wichtig:

- Wird deshalb während des Codierens durch **Refactoring** ständig verbessert

Kurze Iterationen sind wichtig:

- Also werden die Iterationen extrem kurz gemacht
- 10 Tage, dann wieder „Planning Game“

Feedback durch den Kunden ist wichtig:

- Also ist der Kunden im selben Raum: **On site customer**

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

XP: eXtreme Programming

Beispiel für Refactoring

Replace Magic Number with Symbolic Constant

You have a literal number with a particular meaning.

Create a constant, name it after the meaning, and replace the number with it.

lace Magic
mber with
Symbolic
Constant

```
double potentialEnergy(double mass, double height) {
    return mass * 9.81 * height;
}
```

```
double potentialEnergy(double mass, double height) {
    return mass * GRAVITATIONAL_CONSTANT * height;
}
static final double GRAVITATIONAL_CONSTANT = 9.81;
```

Quelle: Martin Fowler

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

XP: eXtreme Programming

Voraussetzungen für XP:

- kleines Entwicklungsteam (2-10)
- gute Leute, die nicht anfangen zu hacken
- geringe Änderungskosten: Grundlage Refactoring
- eine Software-Entwicklungsumgebung, die permanente Tests erlaubt
- einfaches Design – keine gigantischen Frameworks

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

XP: eXtreme Programming

Achtung !

Manche benutzen XP als Ausrede um nicht zu dokumentieren ..

- Im Chrysler C3 Projekt wurde nicht dokumentiert
- Das hat später durch Wechsel von Teammitgliedern zu Problemen geführt.
- Außerdem wurde das Projekt inzwischen abgebrochen - angeblich aus „politschen Gründen".

Manche sagen, sie machen XP:

- Nur leider ist der Kunden nicht greifbar
- Damit kann es nicht mehr funktionieren und bleibt dann lediglich eine Ausrede für schlechte Architektur und mangelnde Dokumentation

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

RUP ↔ XP

RUP versus XP oder RUP kombiniert mit XP ?

siehe white paper von Barchfeld et al.

Einiges spricht für die Kombination:

- Use cases helfen der Verständigung mit dem Kunden und erzeugen das notwendige fachliche Verständnis für die Programmierer.
- Kleine Inkremente erhöhen die Planungssicherheit und schaffen Möglichkeiten zu einer adaptiven Vorgehensweise.
- Code-Unit-Tests sind das Wesentliche, um feststellen zu können, ob Inkremente wirklich Inkremente sind, d.h. nachweisbare Resultate aufweisen.

auch hier: Dogmatik schadet !

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Personalorganisation

Am Anfang eines Projekts:

Zusammenstellung von Teams

Bereitstellung von benötigten Ressourcen

Aufbau einer Organisationsstruktur

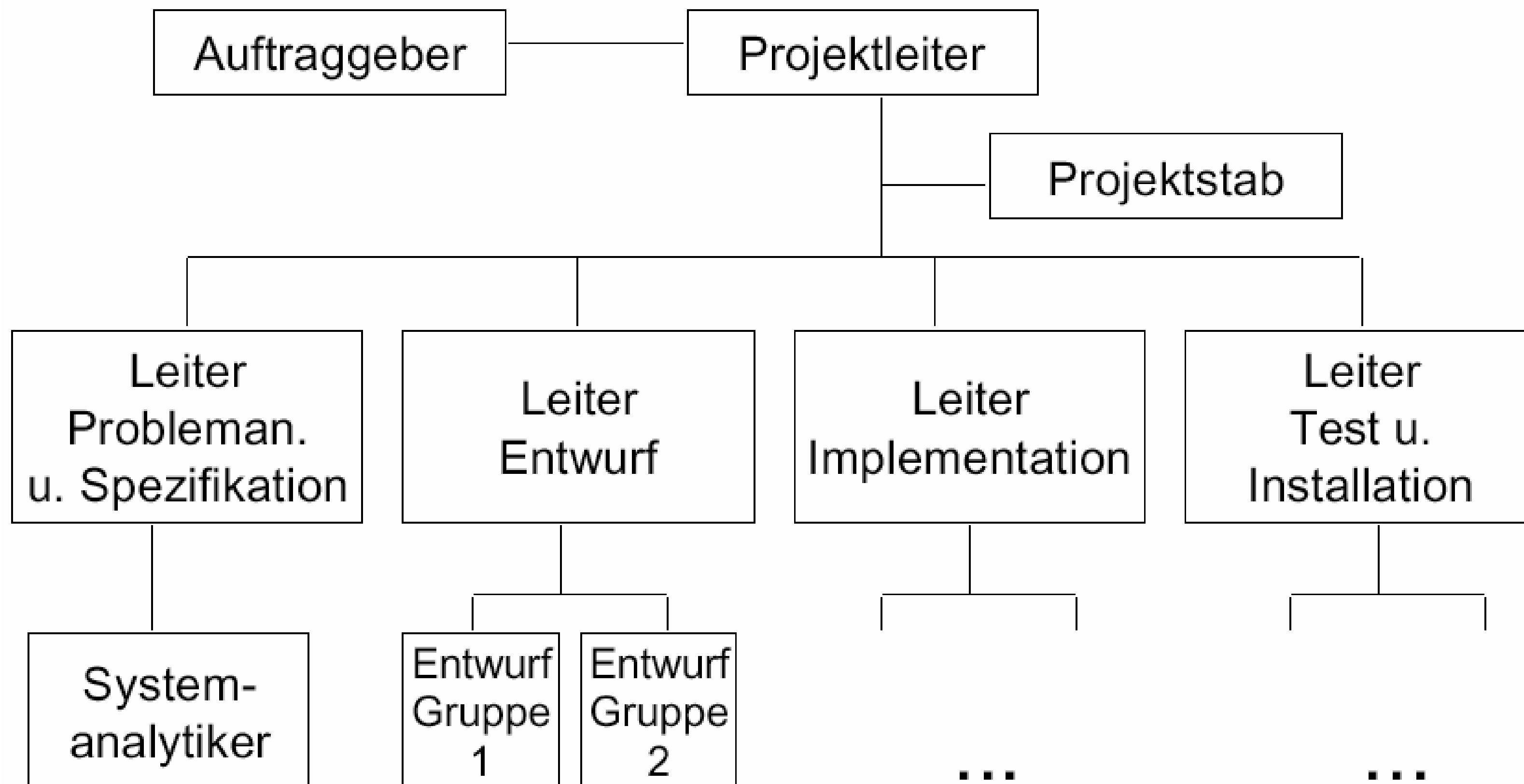
- Zuordnung von Teilaufgaben/Phasen zu Teams oder einzelnen Personen
- Genaue Festlegung der Aufgaben, Rechte und Pflichten aller am Projekt beteiligten Personen (Zuständigkeiten)

Typen von Organisationsstrukturen

- Klassische hierarchische Struktur
- Chef-Programmierer-System

Personalorganisation

Klassische hierarchische Struktur



Personalorganisation

Klassische hierarchische Struktur

Probleme:

- Projektleiter zu weit von Programmierung entfernt
- Mehrstufigkeit behindert Kommunikation
- Aufstieg in Hierarchie bis zur Inkompetenz

Personalorganisation

Chef-Programmierer-System

Verzicht auf Projektleiter, der nicht an Systementwicklung beteiligt ist

Einsatz von sehr guten Spezialisten, die mit hoher Eigenverantwortung arbeiten

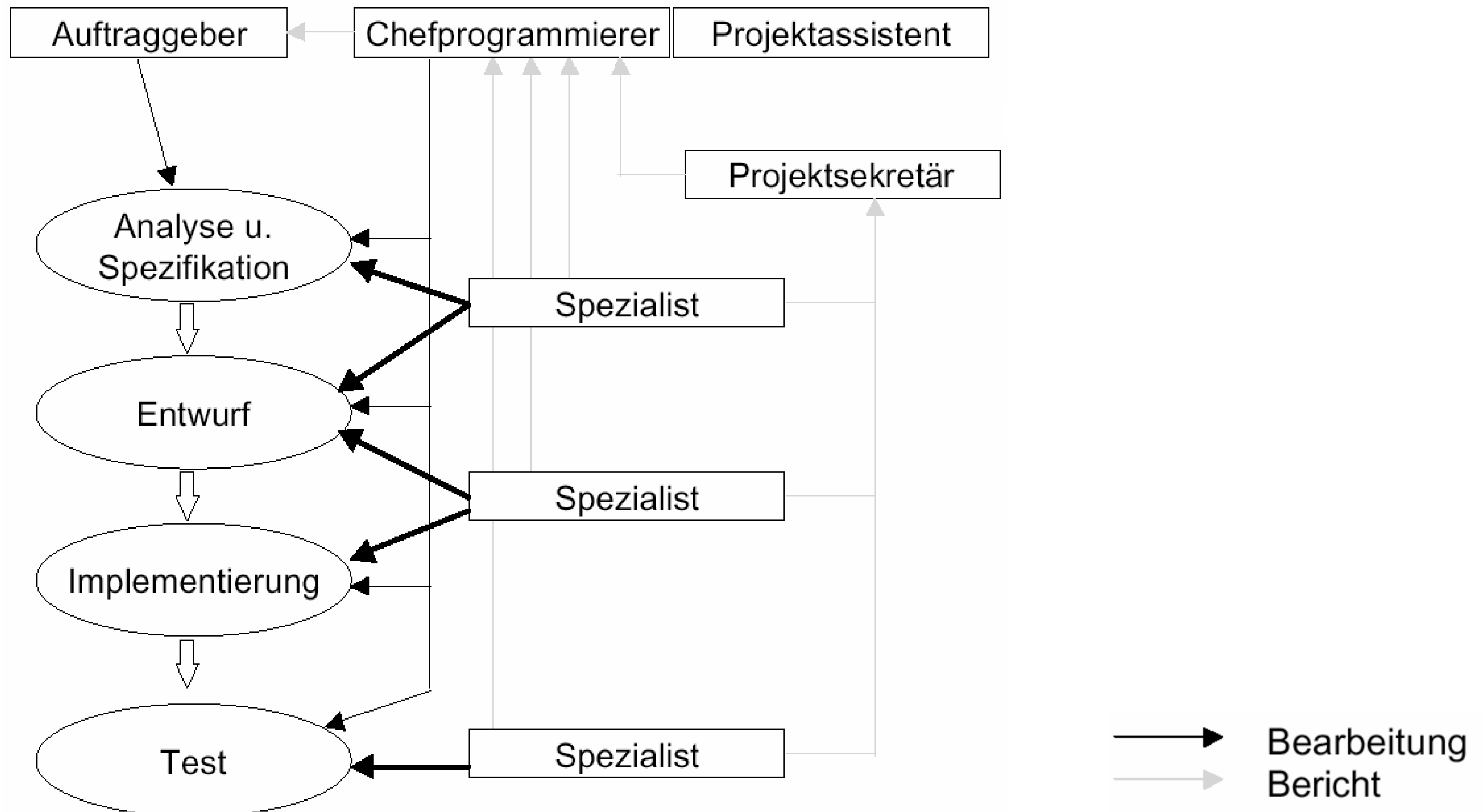
Beschränkung der Teamgröße

Zusammensetzung der Teams:

- Chef-Programmierer
- Projektassistent
- Projektsekretär
- Spezialisten (Systemanalytiker, Programmierer, Testspezialisten)

Personalorganisation

Chef-Programmierer-System



Personalorganisation

Chef-Programmierer-System

Vorteile

- Chefprogrammierer kann durch direkte Einbindung Kontrollfunktion besser wahrnehmen
- Geringere Kommunikationsschwierigkeiten
- Kleinere (Spezialisten-)Teams sind produktiver

Nachteile

- Beschränkung auf kleine Teams
- Anforderungen an Chefprogrammierer nahezu unerfüllbar
- Stellung des Projektsekretärs problematisch