universitätbonn ROOTS

Kapitel 12.
Softwareentwicklungsprozessmodelle

Stand: 27.1.2010

Hinweis: Dass in der Foliennummerierung „Sprünge“ auftreten, ist in Ordnung, das sind ausgeblendete Folien, die für Sie nicht relevant sind.

Einordnung

- Bisher (Kap. 3): **Womit** arbeiten wir?
  - UML
- Bisher (Kap. 2, 4-12): **Was** tun wir? → Aktivitäten
  - Anforderungen erfassen, entwerfen, implementieren, testen, ...
  - Versionsverwaltung, Qualitätssicherung, Projektmanagement, ...
- Nun (Kap. 13-14): **Wie** tun wir es?
  - Wie passt das alles zusammen?
  - Was ist ein Softwareentwicklungsprozessmodell?
  - Was ist „Agile Softwareentwicklung“?
- Ausblick (Kap. 15): Zurück zum „**Womit**?“
  - Aspektorientierte Programmierung

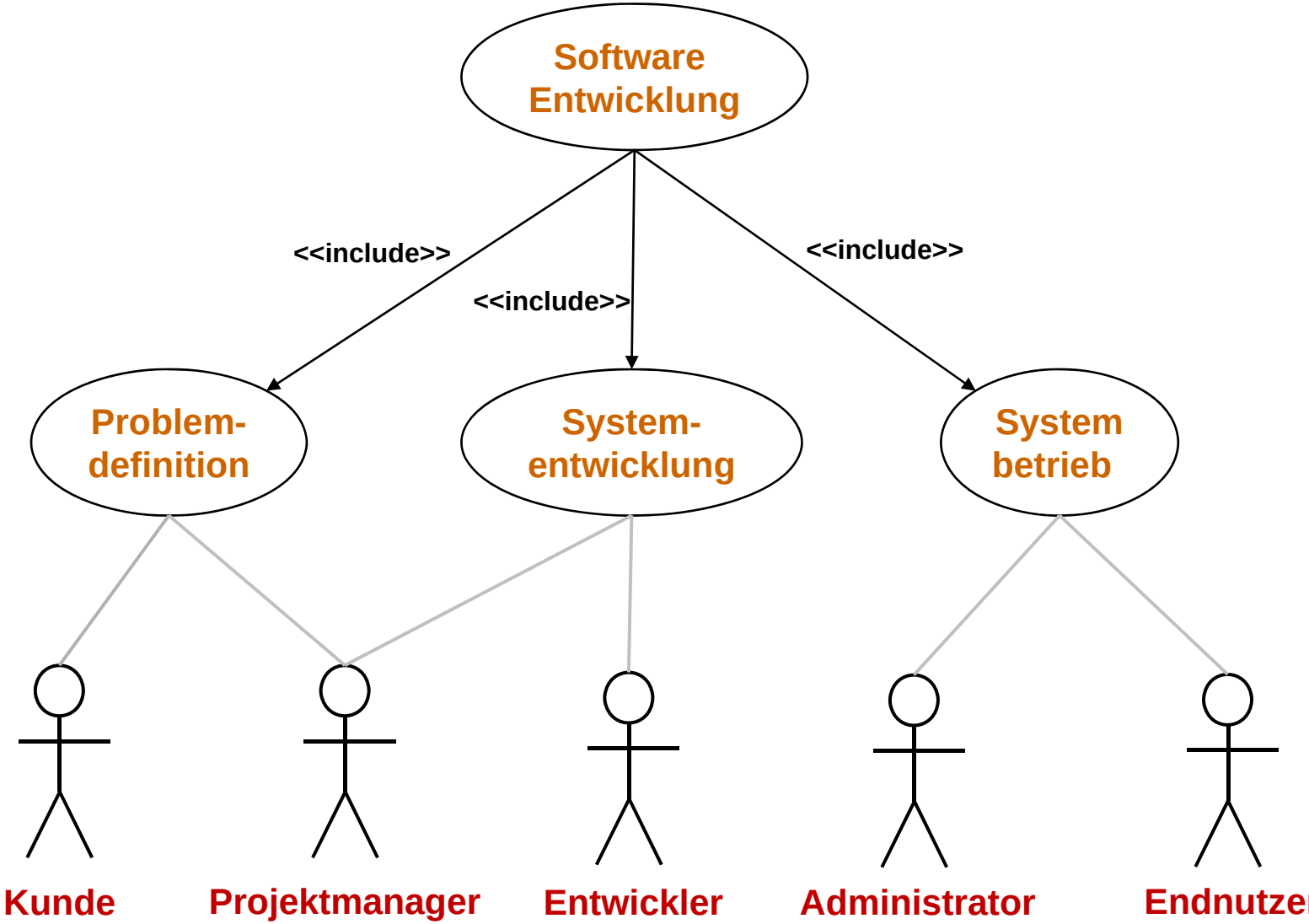
Typische Fragen zum Software-Lebenszyklus

- Welche **Aktivitäten** soll ich für das Softwareprojekt auswählen?
- Was sind die **Produkte** der Aktivitäten?
- Was sind die **Abhängigkeiten** zwischen den Aktivitäten?
  - Welche **Aktivitäten** hängen von welchen **Produkten** ab?
  - Hängt das Systemdesign von der Analyse ab?
  - Hängt die Analyse vom Design ab?
- Wie soll ich die Aktivitäten **planen**?
  - Soll die Analyse dem Design vorangehen?
  - Können Analyse und Design parallel ablaufen?
  - Sollten sie iterativ ablaufen?

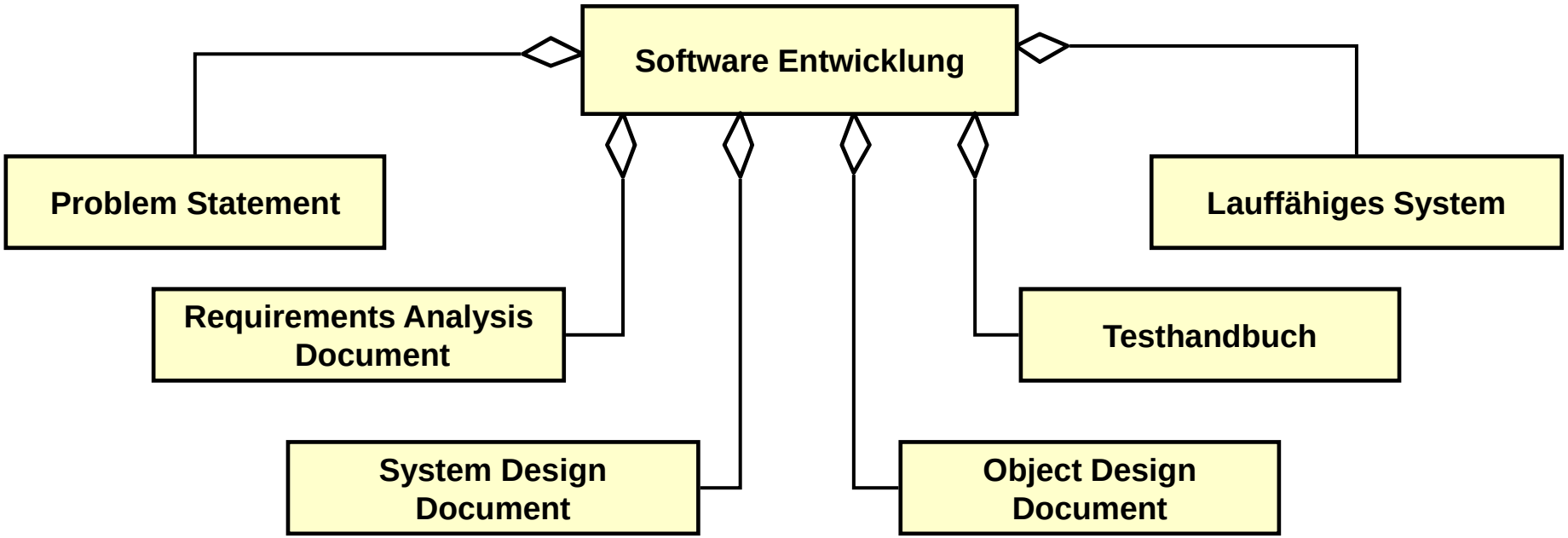
Aktivitäten (Beispiele)

Anforderungsanalyse	Was ist das Problem?
Systementwurf	Was ist die Lösung?
Programmentwurf	Welche Mechanismen liefern die beste Lösung?
Implementierung	Wie setzt sich die Lösung zusammen?
Tests	Ist das Problem gelöst?
Auslieferung	Kann der Benutzer die Lösung verwenden?
Wartung	Sind Verbesserungen notwendig?

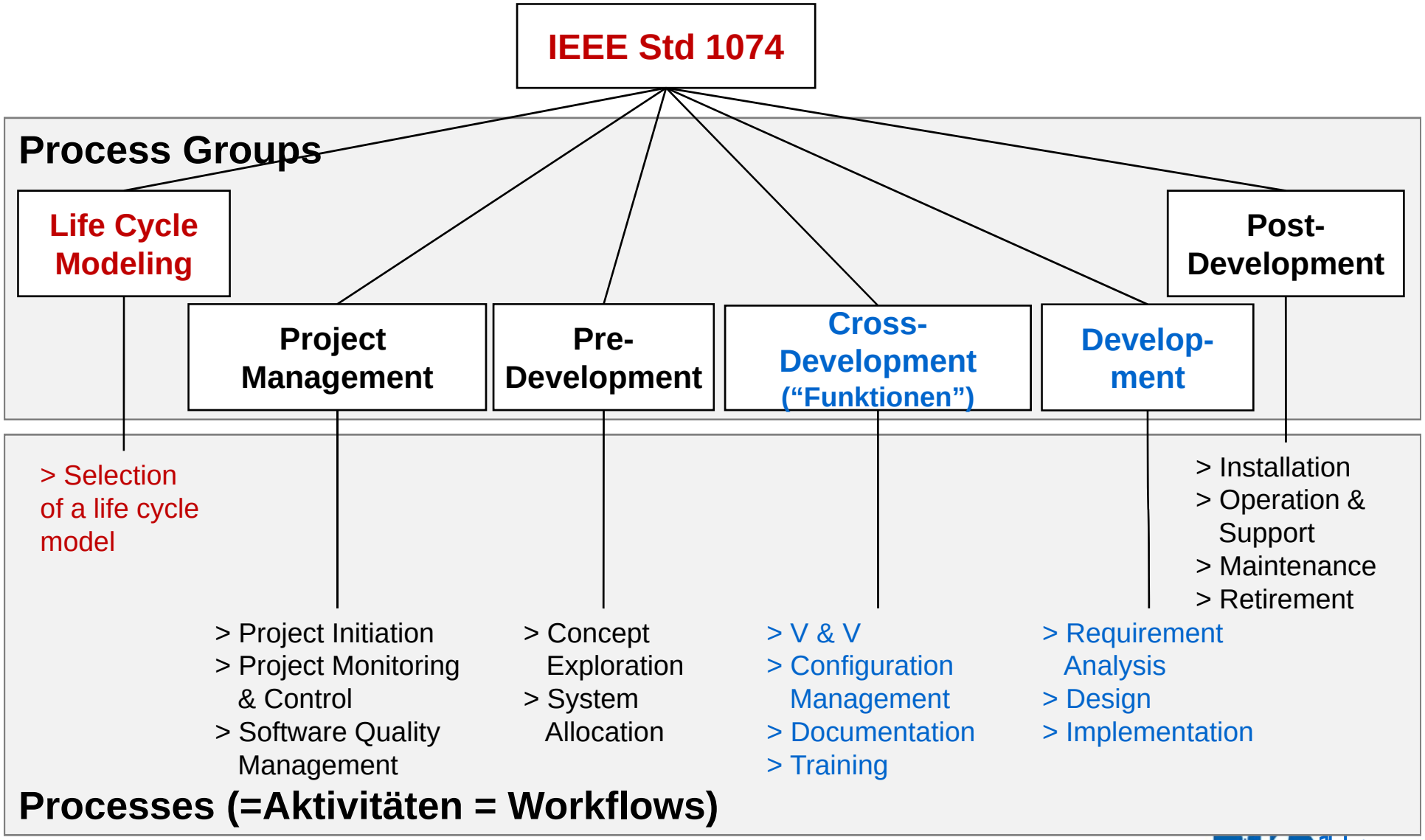
Rollen und Teilprozesse



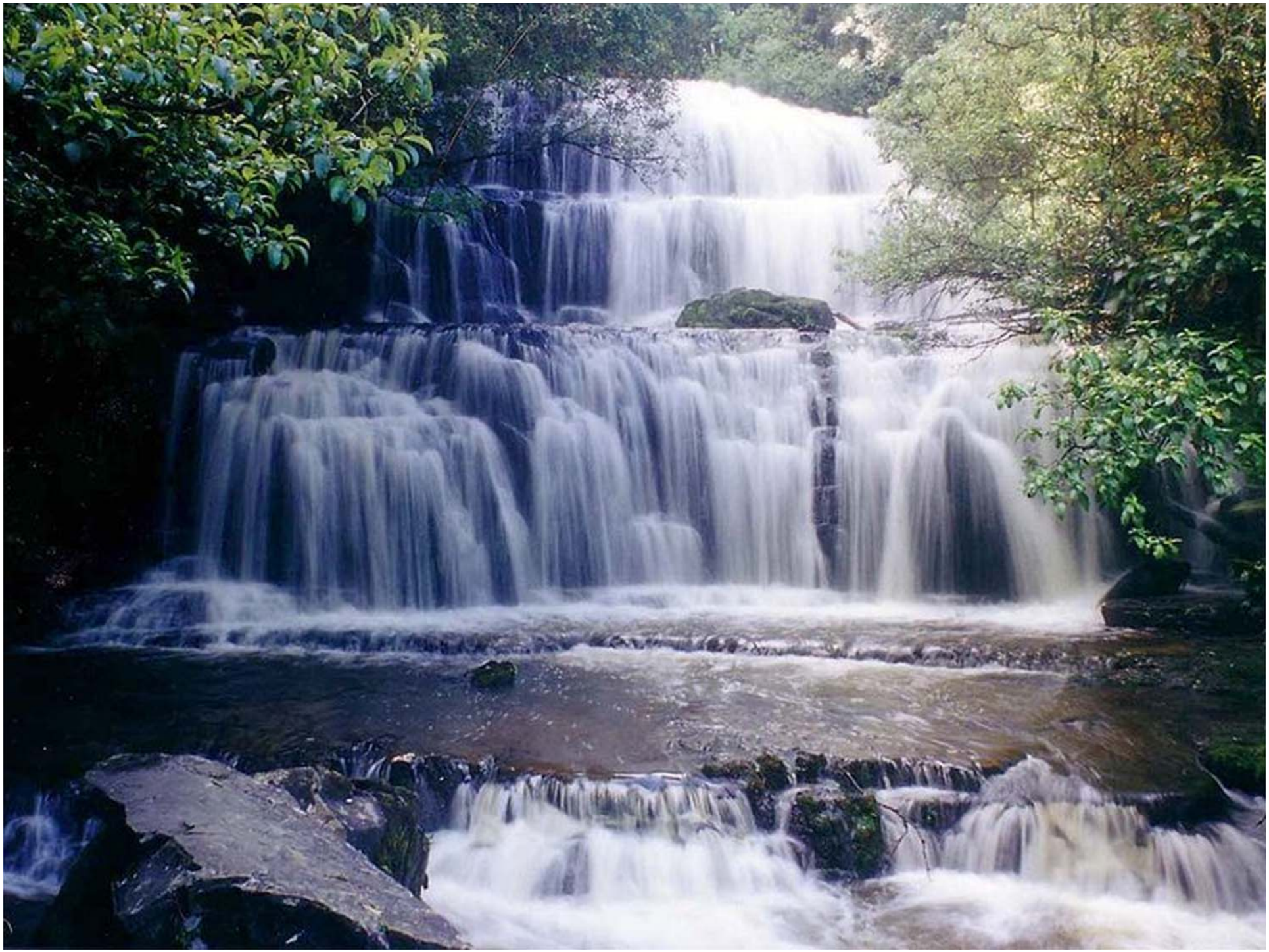
Produkte


Software Entwicklung
Problem Statement
Lauffähiges System
Requirements Analysis Document
Testhandbuch
System Design Document
Object Design Document

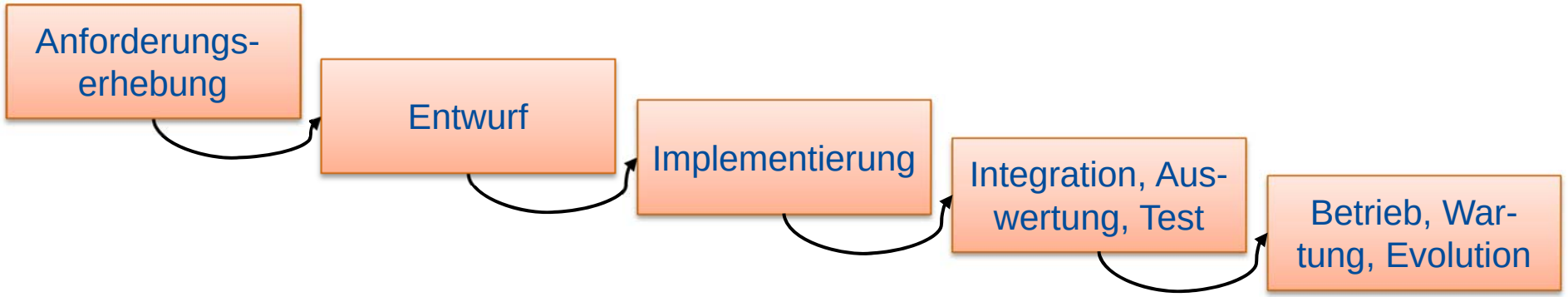
IEEE Standard 1074: Standard für den Software-Lebenszyklus



Wasserfallmodell



Wasserfallmodell (1/4)



- Prozess der die sequentielle Ausführung aller Aktivitäten einer Kategorie vor denen der nächsten Kategorie vorschreibt
  - Alle Anforderungen werden erhoben bevor mit dem Entwurf begonnen wird
  - Zwischenergebnisse: Dokumente (Z.B. “Pflichtenheft”).
- Man kann auch zur vorherigen Aktivitätskategorie zurückkehren, wenn man Änderungsbedarf feststellt
  - Das impliziert aber, dass man die Aktivitäten der aktuellen Kategorie ruhen lässt, bis alle vorherigen wieder vollständig aktualisiert sind → kostspielig
  - Vorhandene Probleme werden aufgeschoben, um sie “herum” programmiert


Das detaillierte Wasserfallmodell
Projekt-
initiierung
Machbarkeits-
analyse
Ressourcen-
zuordnung
Anforderungs-
analyse
Entwurf
Implementierung
Verifikation
& Validation
Installation
Betrieb &
Wartung

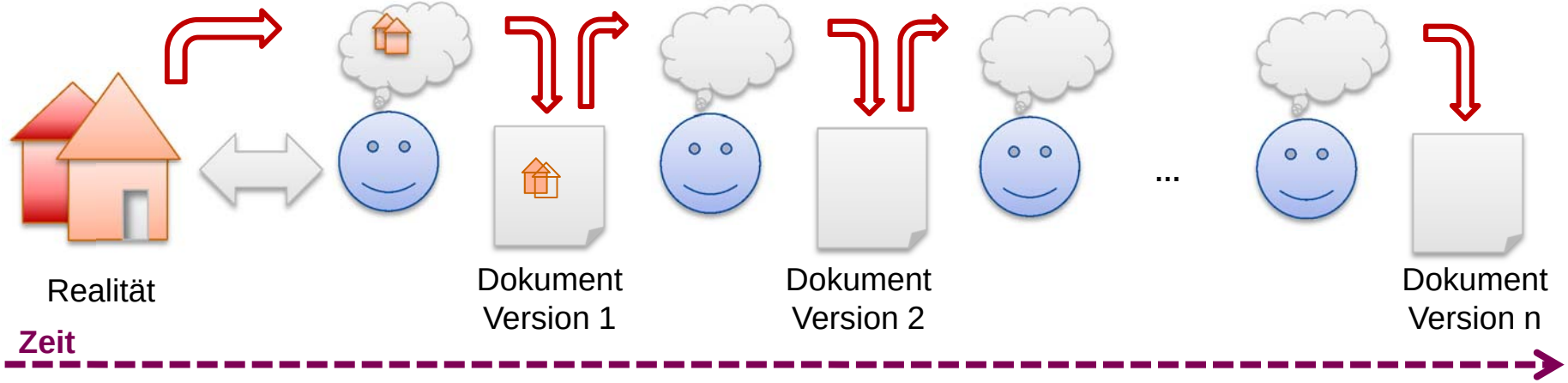
Wasserfallmodell (3/4)

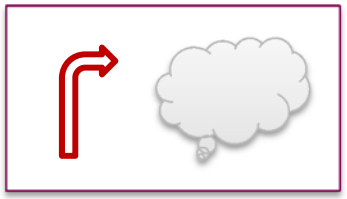
- Vorteile
  - Einfach zu verstehen
  - Stellt die “logische” (im Gegensatz zur temporalen) Reihenfolge der Gegenstände dar
  - Dokumentation wird in jeder Phase produziert
  - Optimal für große Projekte mit vielen (sich ändernden) Teams
- Nachteile
  - Inflexible Partitionierung des Projekts in eindeutige, unabhängige Abschnitte
  - Später Einsatz beinhaltet viele Risiken
    - ⇨ technologische, konzeptuelle, personalbezogen
  - (Zu) späte System- und Akzeptanztests
  - Prozessmodel ist nur dann geeignet wenn die Anforderungen richtig verstanden wurden
  - Anforderungen müssen ausgebessert werden noch bevor man mit dem Entwurf beginnt

Wasserfallmodell (4/4)

- Ist das Wasserfallmodell praktikabel?
  - Manche bewahren noch diese Illusion.
  - Praktiker berichten, dass es nie wie beschrieben funktioniert
  - Manchmal Sündenbock für Kritik, z.B.: http://waterfall2006.com/
- Wer hat's erfunden?
  - Ein sequentieller Ansatz wurde **beschrieben und verbessert** in: **Winston W. Royce**: "Managing the Development of Large Software Systems: Concepts and Techniques". In: *Technical Papers of Western Electronic Show and Convention* (WesCon) August 25-28, **1970**, Los Angeles, USA.
  - "*The **testing phase** which occurs at the end of the development cycle is the first event for which timing, storage, input/output transfers, etc., are **experienced as distinguished from analyzed**. […] Yet if these phenomena fail to satisfy the various external constraints, then invariably a **major redesign** is required*." (Royce, p. 329).

Informationen in Dokumenten verfallen





- Jede Person entwickelt ihr eigenes Verständnis, Missverständnisse sind möglich

- Jedes Mal wenn jemand sein eigenes Verständnis vermittelt, können Informationen verloren gehen
  - Löschung – Generalisierung – Verzerrung

- Korrekturen können natürlich auch stattfinden

Konsequenzen

- Innerhalb desselben (Wasserfall-) Projekts
  - Dokumente werden evtl. von Teammitgliedern, die für spätere Schritte verantwortlich sind, nicht verstanden
  - Wenn möglich, Anzahl der Schritte reduzieren!
  - Alle Teammitglieder sollen miteinander reden!
- Für die Evolution eines Produkts im Laufe der Zeit
  - Verständnis der ursprünglichen Anforderungen und Entwurfsziele kann mit der Zeit verloren gehen.
  - Rückverfolgbarkeit ermöglicht das Nachvollziehen der urspünglichen Anforderungen und Entwurfsvorhaben!
  - Formale Sprachen benutzen, um die Enstehung von Inkonsistenzen zu erschweren!
  - Natürliche Sprache benutzen, um Konzepte zu erklären!
- Für Ihre Recherche
  - Lesen Sie sowohl die “klassischen” Artikel als auch die aktuellen.
  - Untersuchen Sie das Problem selbst (wenigstens teilweise).
  - Bsp.: In Royces Artikel stecken mehr vernünftige Gedanken als man erwartet, angesichts des schlechten Rufs des Wasserfallmodells.

Für und Wider des Wasserfallmodells

- Manager lieben Wasserfallmodelle
  - Nette Meilensteine
  - Kein Rückblick nötig (lineares System), jederzeit nur eine Aktivität
  - Fortschritt leicht zu prüfen : 90% codiert, 20% getestet

Aber, ...

- Softwareentwicklung ist iterativ
  - Während des Designs werden Probleme mit den Anforderungen festgestellt
  - Während der Implementierung werden Design- und Anforderungsprobleme festgestellt
  - Während des Testens werden Implementierungs-, Design-, und Anforderungsfehler gefunden
  - → Spiralmodell
- Systementwicklung ist nicht-linear
  - → Problem-orientiertes Modell

Spiralmodell

Iterative Entwicklungsprozesse
Motivation

- Begreifen dass sich Anforderungen ändern
  - Caspers Jones' Forschung zu 8000 Projekten
    - ⇨ 40% der endgültigen Anforderungen kammen erst nach der Analysisphase, nachdem die Entwicklung bereits begonnen hatte
- Begünstigt vorzeitige Risikoverringerung
  - Aufspalten des Systems in Mini-Projekte
  - Schwerpunkt zuerst auf die riskanten Elemente legen
- Fördert die gemeinsame Beteiligung aller Teilnehmer, einschließlich Tester, Integrators und Technical-Writer
- Ermöglicht die Anpassung des Prozesses nach jeder Iteration, frühe Korrektur am Code und Verwirklichung der "Lektion" die man nach jeder Iteration lernt
- Fokus auf Komponentenarchitektur, anstatt auf endgültigen Einsatz

Spiralmodell (Boehm 1988)

- Als Alternative zum Wasserfallmodell
- Bietet Platz für häufige Änderungen während der Softwareentwicklung
- Inkrementiert den Grad der Definition und Implementation eines Systems und verringert gleichzeitig den Risikograd
- Basiert auf den Aktivitäten des Wasserfallmodells. Fügt mehrere (Sub-) Aktivitäts (-Zyklen) hinzu.
  - Objektives Einstellen
  - Risikoabschätzung und -reduzierung
  - Entwicklung und Bewertung
  - Planen der nächsten Phasen
- Übersicht
  - Prozess ist wie eine Spirale angeordnet
  - Jede Schleife repräsentiert eine Phase des Arbeitsablaufs (z.B. Konzept der Operationen, Anforderungen, Entwurf, Code, unit tests)
  - Risks are explicitly assessed and resolved throughout the process Risikien werden explizit abgeschätzt und während des Prozesses behoben

Example Spiral Model Project

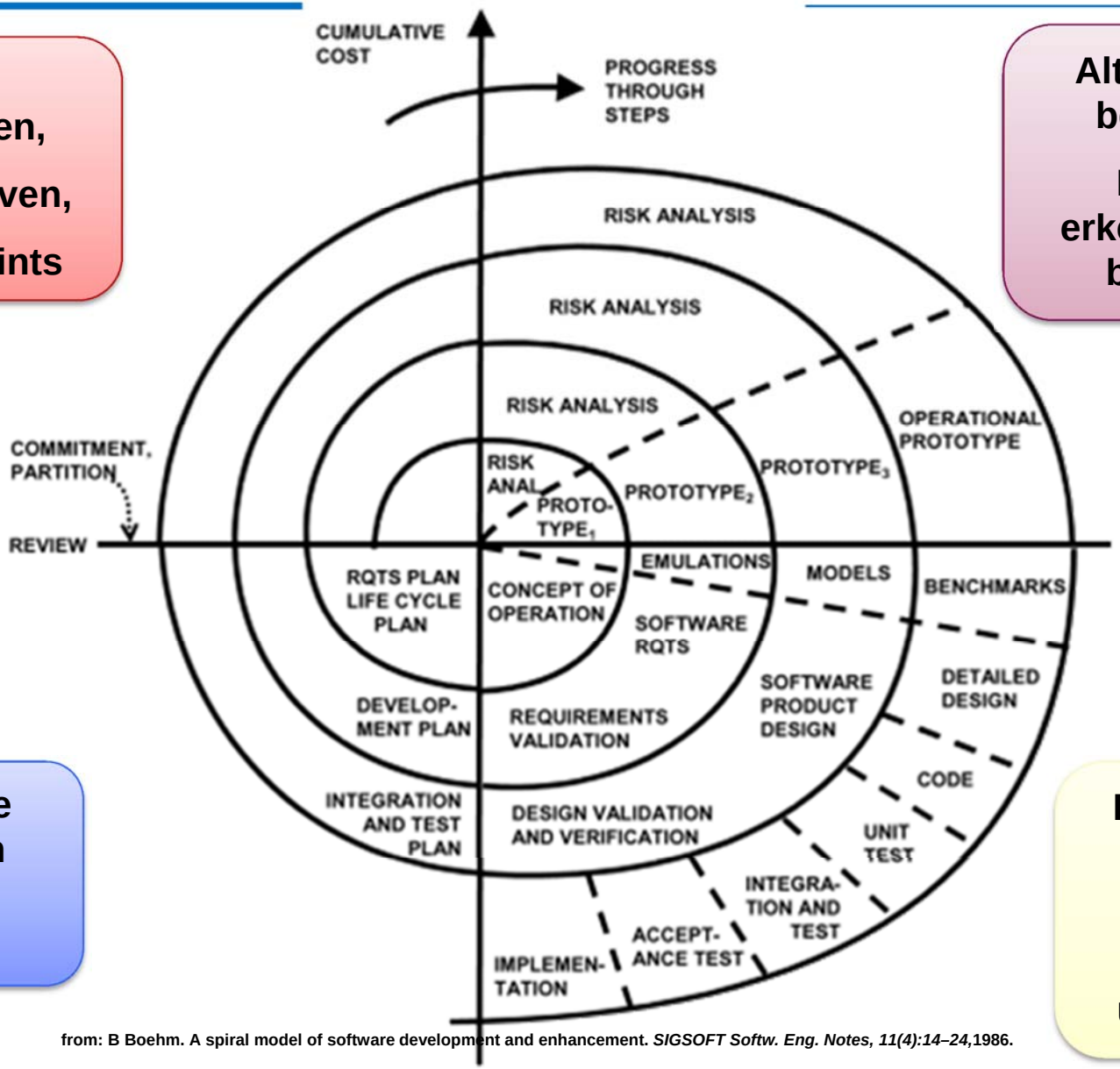
Ziele festlegen,

Alternativen,

Constraints

Alternativen bewerten,

Risiken erkennen und beheben



Nächste Phasen planen

Entwickeln,

Nächstes-Level-Produkt überprüfen

from: B Boehm. A spiral model of software development and enhancement. *SIGSOFT Softw. Eng. Notes, 11(4):14–24*,1986.

Typen von Prototypen beim Spiralmodell

- Illustrativer Prototyp
  - Entwickle das Userinterface mit einem Satz von Ablaufplänen
  - „Implementiere“ ein Mock-Up mit einem UI-Builder (Visual C++, QT-Designer, Papierserviette, Bierdeckel, ...)
  - Gut für den ersten Dialog mit dem Kunden
- Funktionaler Prototyp
  - Implementiere und liefere ein lauffähiges System mit minimaler Funktionalität
  - Dann füge Funktionalität hinzu
  - Das jeweilige Risiko bestimmt die Reihenfolge
- Explorations-Prototyp ("Hacken")
  - Implementiere einen Teil des Systems, um mehr über die Anforderungen zu lernen.
  - Gut für Brüche im Denkmuster

Typen des Prototyping (fortgesetzt)

- Revolutionäres Prototyping
  - Auch “specification prototyping” genannt
  - Ermittle die Praxis beim Kunden mit einer Wegwerf-Version um die Anforderungen richtig zu bestimmen, dann baue das ganze System
    - ⇨ Nachteil: Benutzer müssen einsehen das Funktionen des Prototypen teuer in der Implementierung sind
    - ⇨ Benutzer kann enttäuscht sein, weil ein Teil der Funktionalität des Prototyps in der späteren Implementierung nicht machbar ist

- Evolutionäres Prototyping
  - Der Prototyp ist Basis für die Implementierung des finalen Systems
  - Vorteil: Kurze Fertigstellungszeit
  - Nachteil: Kann nur benutzt werden, wenn das Zielsystem in der Sprache des Prototypen konstruiert werden kann

Das Spiralmodell (Boehm) befasst sich mit Iteration

- Identifiziere die Risiken
- Gib den Risiken Prioritäten
- Entwickle und teste eine Reihe von Prototypen für die einzelnen Risiken
- … in der Reihenfolge fallenden Risikos bzw. fallender Priorität
- Nutze das Wasserfallmodell zur Entwicklung jedes Prototyps (“Zyklus”)
- Wenn ein Risiko erfolgreich beseitigt wurde, bewerte das Ergebnis des Zyklus und plane die nächste Runde
- Wenn ein bestimmtes Risiko nicht beseitigt werden kann, beende das Projekt sofort

Aktivitäten (“Runden”) in Boehm’s Spiralmodell

- Die folgenden Aktivitäten verteilen sich über die Zyklen der Spirale
  - Rahmenbedingungen
  - Anforderungen
  - Systemdesign
  - Detailiertes Design
  - Implementierung
  - Modultest
  - Integration und Test
  - Akzeptanztest
- Frühere Aktivitäten geschehen in den inneren/initialen Zyklen, spätere in den äußeren Zyklen
  - Siehe Spirale auf der Seite zuvor
- Gehe in jedem Zyklus diese Schritte
  - Bestimme Ziele, Alternativen, Nebenbedingungen
  - Bewerte Alternativen, identifiziere und löse Risiken
  - Entwickle und verifiziere den Prototyp
  - Plane den nächsten Zyklus

Die Grenzen des Wasserfall- und Spiralmodells

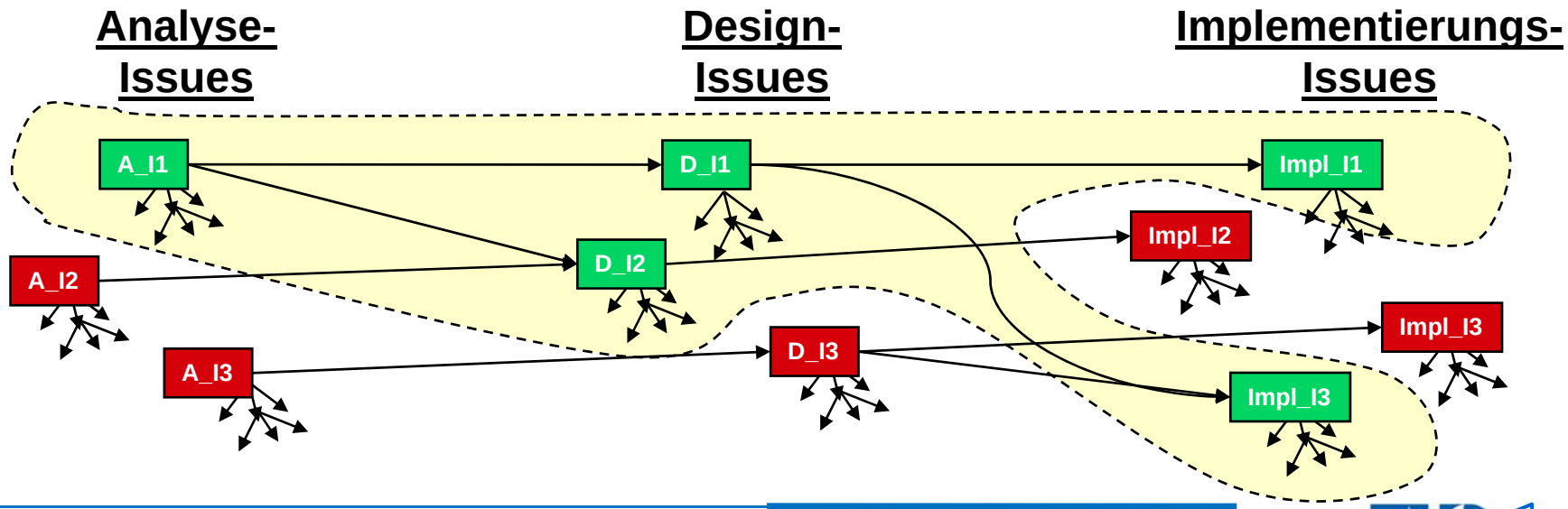
- Keines von beiden befasst sich mit häufigen Änderungen
  - Das Wasserfall unterstellt, dass nach einer Phase alle Probleme in dieser abgeschlossen sind und nicht wieder geöffnet werden können
  - Das Spiralmodell kann mit Änderungen zwischen den Phasen umgehen, aber nicht mit Änderungen während einer Phase

- Was tun, wenn Änderungen häufiger erfolgen? (“Das einzig konstante ist die Änderung”)

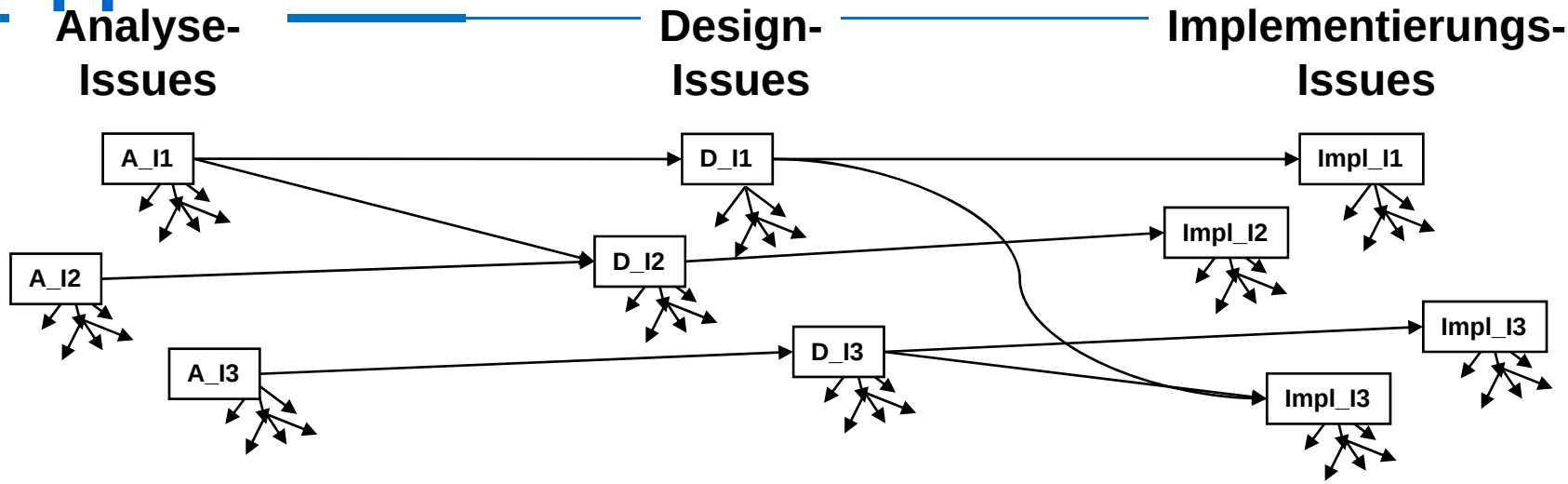
„Issue-Based Development“

Issue-Based Development

- Ein System wird durch eine Sammlung von Problemen beschrieben
  - Probleme sind offen A_I2 oder geschlossen A_I2
  - Geschlossene Probleme haben eine Lösung
  - Geschlossene Probleme können wieder geöffnet werden (Iteration!)
- Probleme haben Abhängigkeiten
- Die geschlossenen Probleme sind die Basis des Systemmodells

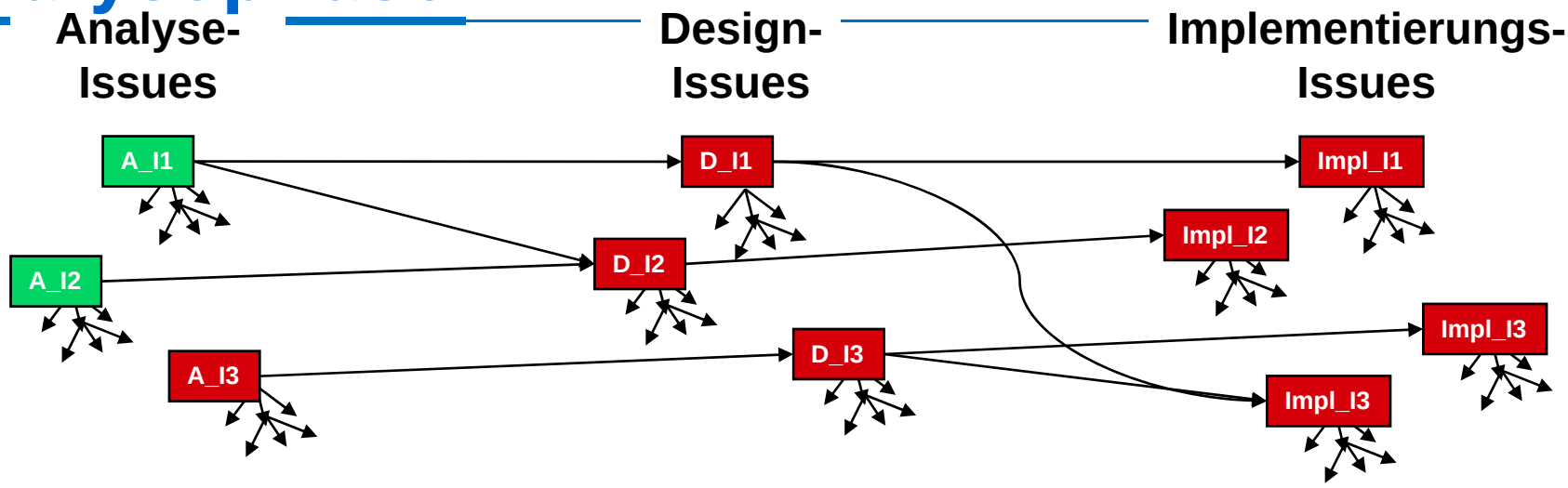


Beispiel: Projekt mit nach Aktivitäten gruppierten Issues



- Legende (auf folgenden Seiten)
  - A_I2 Rot = „offene issues“(noch zu bearbeiten)
  - A_I2 Grün = „geschlossene issues“(erledigt)

Issues im Wasserfallmodell: Analysephase

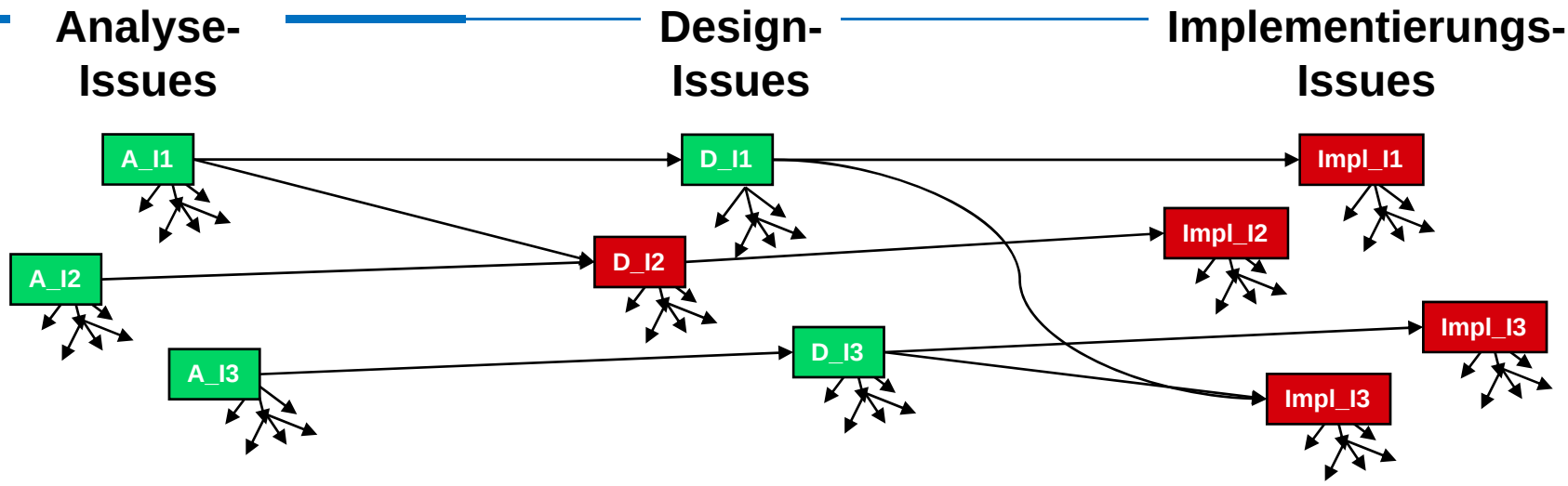


100-0% Offene Analyse-Issues

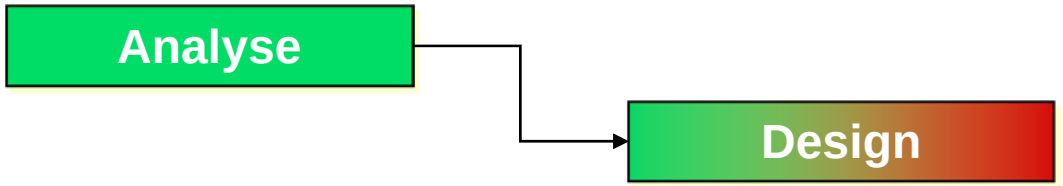
100% Offene Design-Issues

100% Offene Implem.-Issues

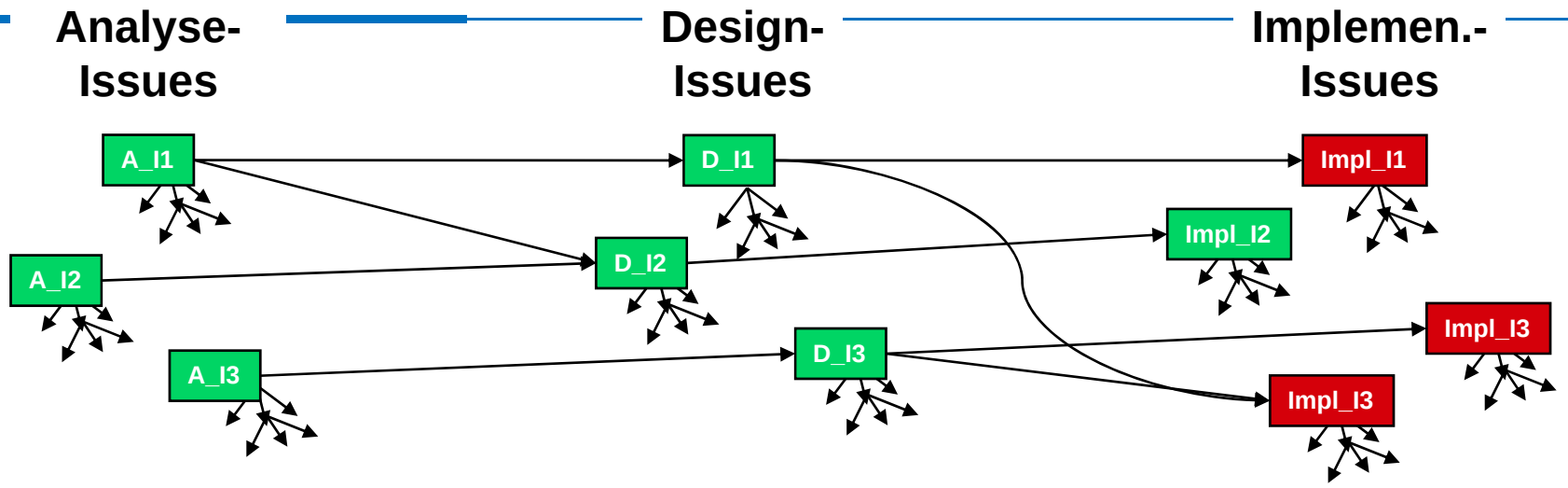
Issues im Wasserfallmodell: Designphase



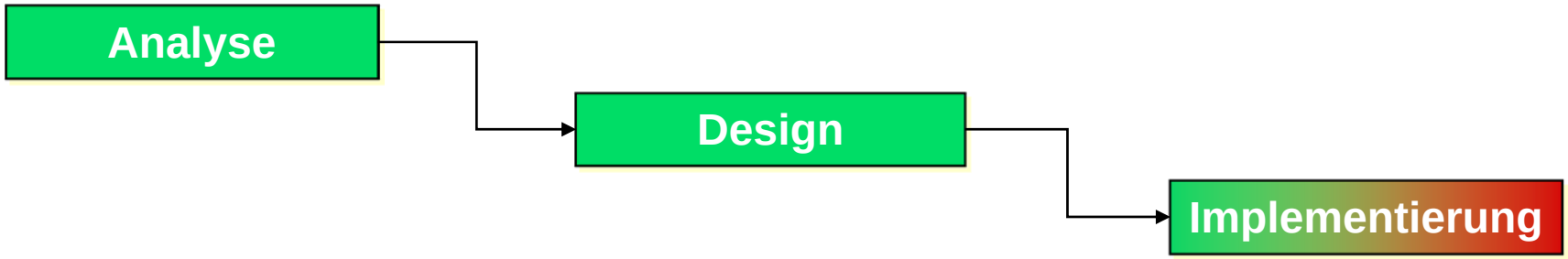
Wasserfall



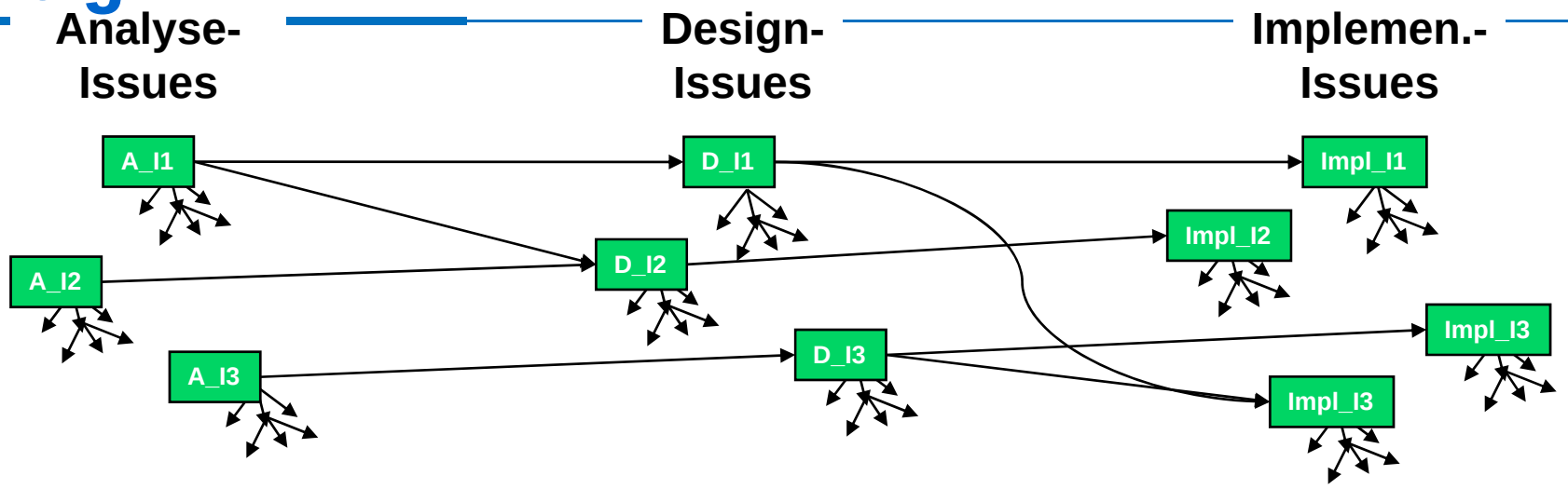
Issues im Wasserfallmodell: Designphase



Wasserfall



Issues im Wasserfallmodell: Projekt fertig!

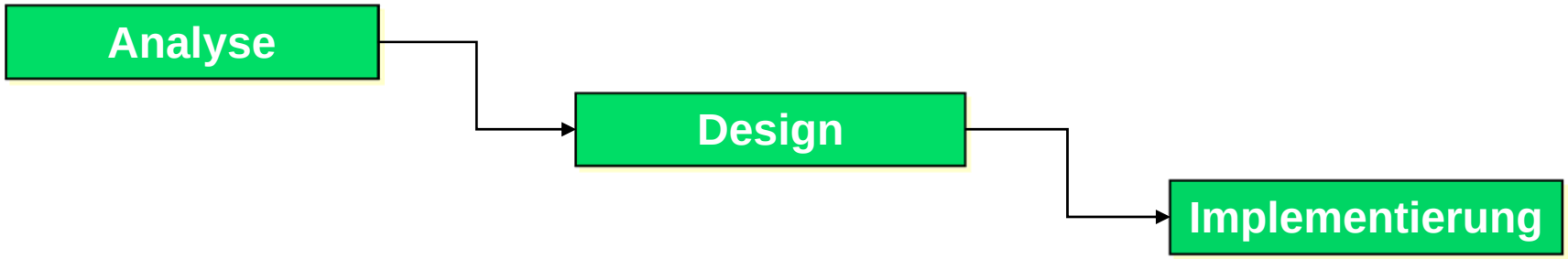


0% Offene Analyse-Issues

0% Offene Design-Issues

0% Offene Implem.-Issues

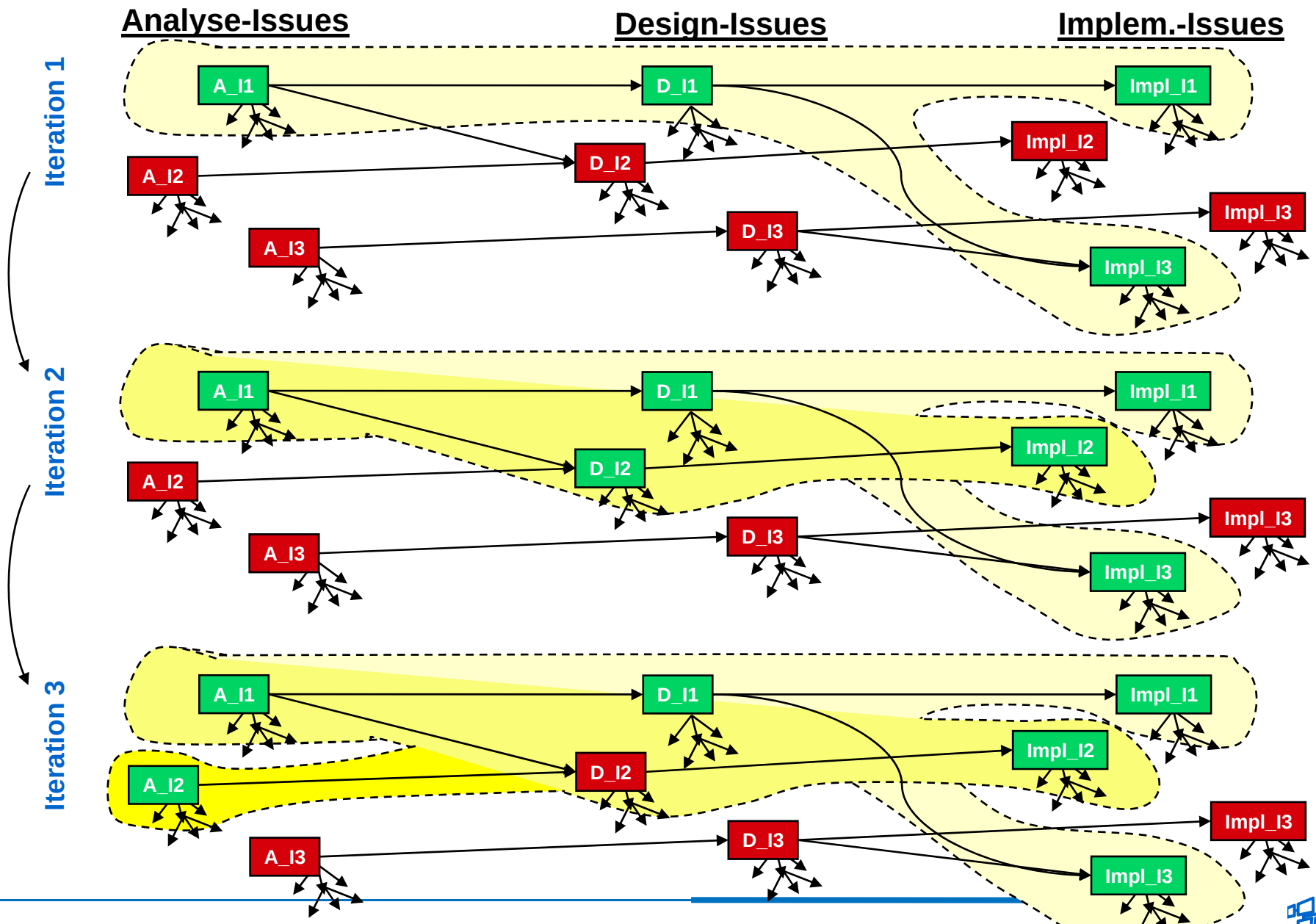
Wasserfall



← Bisher: Issue-basierte Illustration eines Wasserfalls

Nun: „Richtiges“ Issue-Based Development →

„Issue-Based Development“

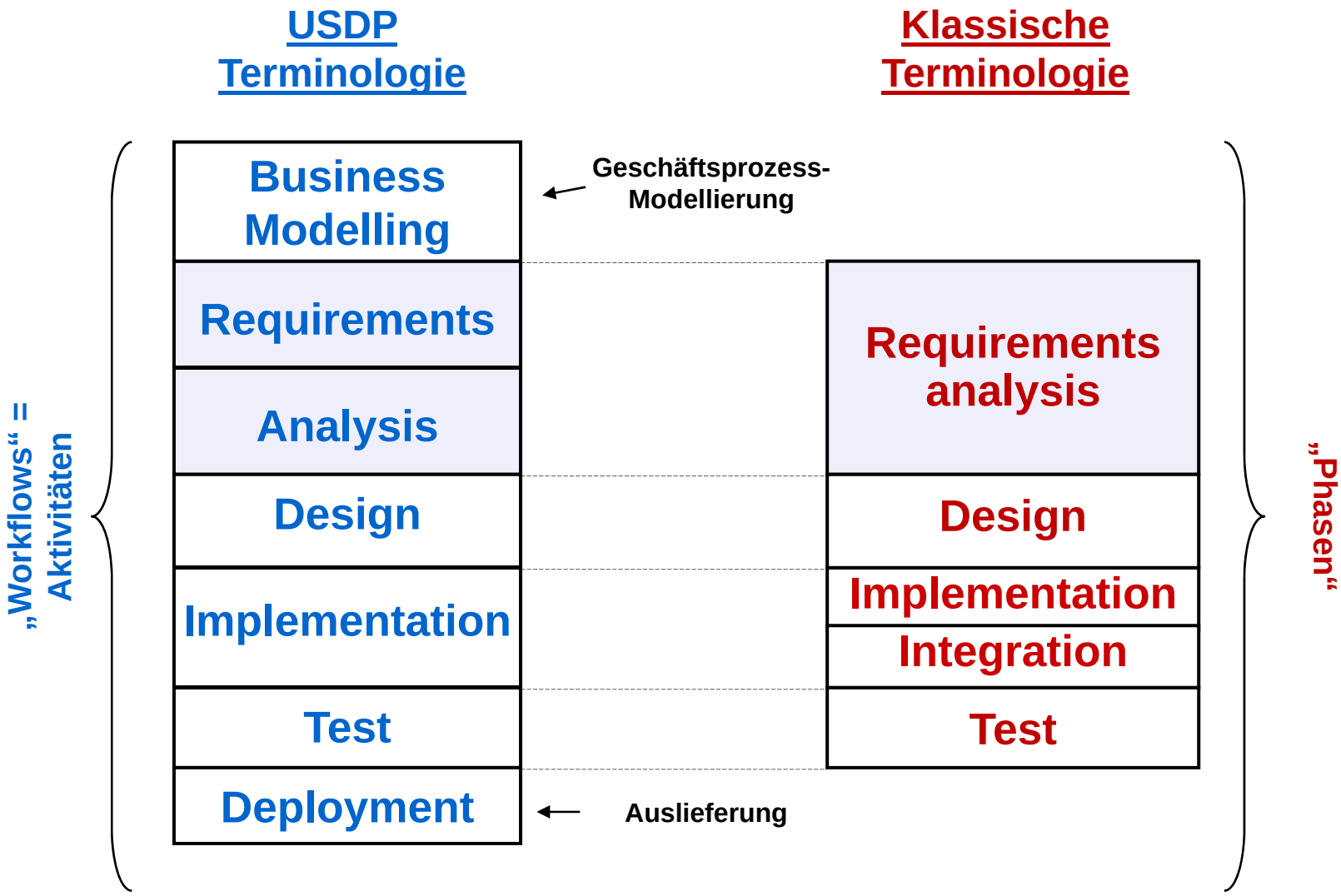


Wann welches Modell verwenden?

- Häufigkeit von Änderungen und Software-Lebenszyklus
  - PT = Projektzeit (project time)
  - MTBC = mittlere Zeit zwischen Änderungen (mean time between changes)
- Änderungen sehr selten (MTBC >> PT):
  - Wasserfallmodell
  - Alle Probleme einer Phase sind vor der nächsten geschlossen
- Änderungen manchmal (MTBC ≅ PT):
  - Boehm's Spiralmodell
  - Änderung während einer Phase kann zur Iteration einer früheren Phase oder der Beendigung des Projektes führen
- Ständige Änderungen (MTBC << PT):
  - Issue-based Development (Concurrent Development Model)
  - Phasen sind nie beendet, laufen alle parallel
    - ⇨ Entscheidung über den Abschluss eines Problems beim Management
    - ⇨ Menge abgeschlossener Probleme ist Basis für das zu entwickelnde System

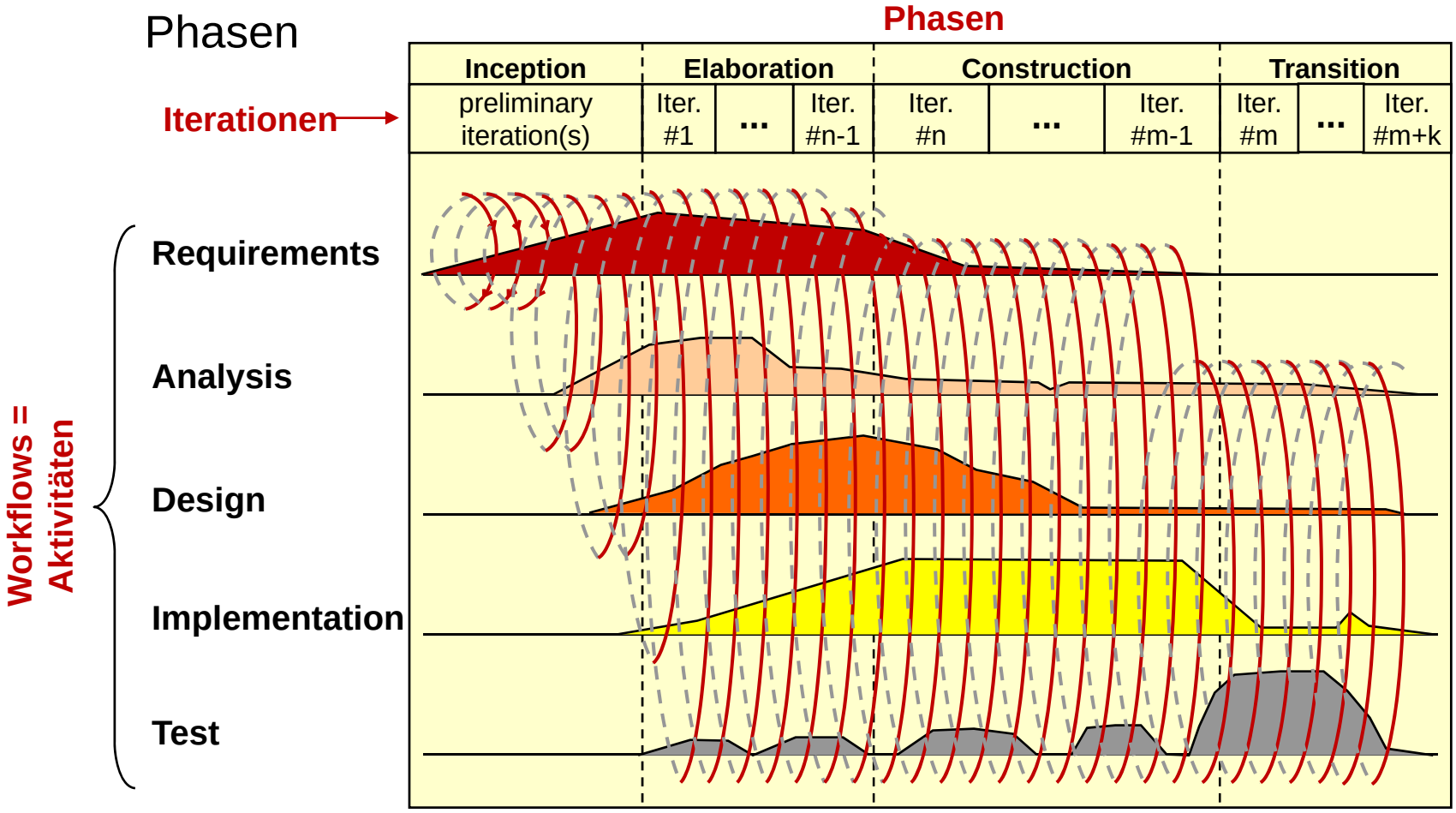
Der „Unified Software Process“

USDP vs. traditionelle Terminologie



Der Unified Software Development Process (USDP)

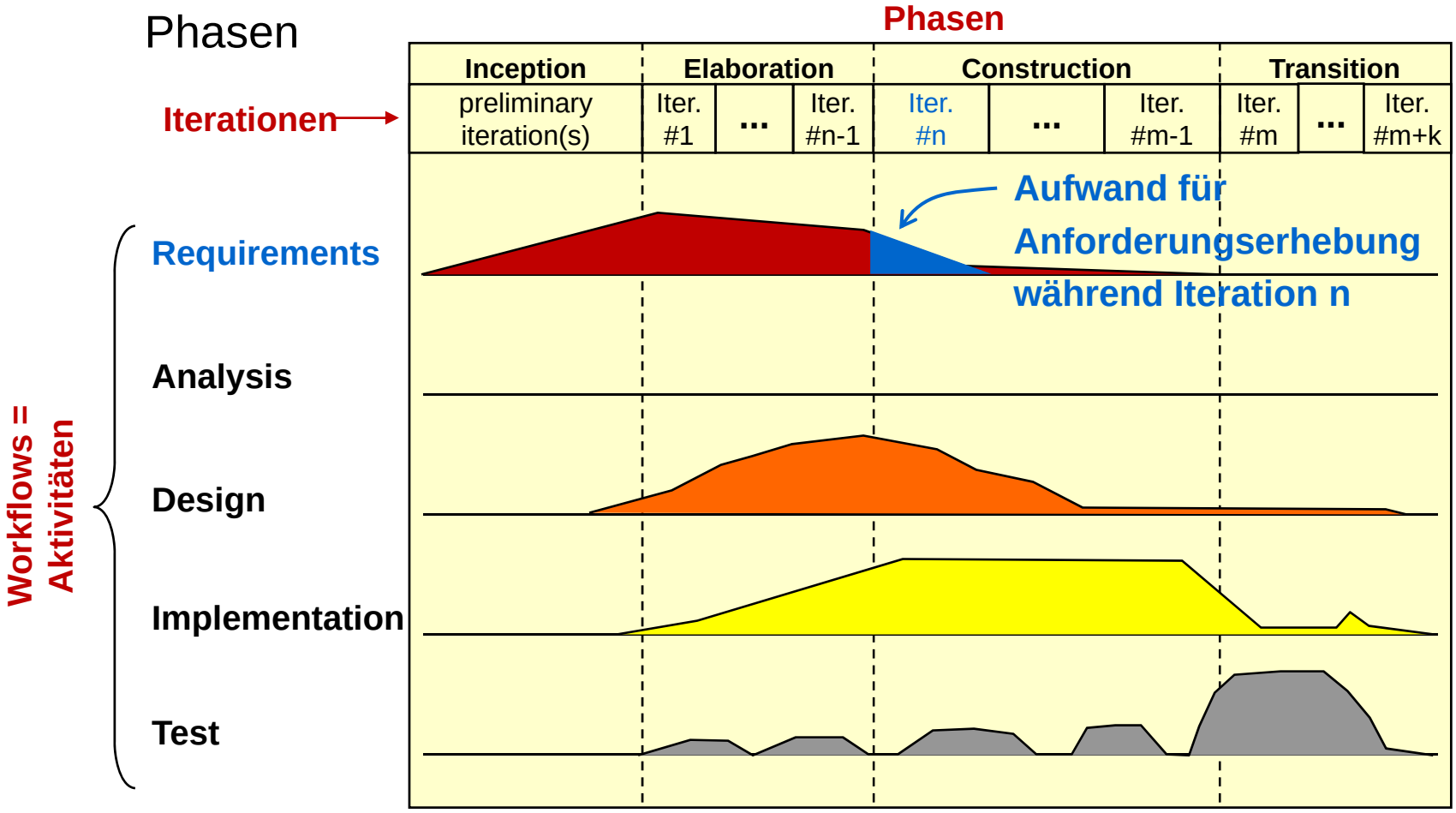
- Problemorientierte Iterationen der Aktivitäten in jeder Phase
- Anteil bestimter Aktivitäten unterschiedlich in den einzelnen Phasen



→ Ivar Jacobsen, Grady Booch, James Rumbaugh: „The Unified Software Development Process“, Addison-Wesley, 1999.

Der Unified Software Development Process (USDP)

- Problemorientierte Iterationen der Aktivitäten in jeder Phase
- Anteil bestimter Aktivitäten unterschiedlich in den einzelnen Phasen

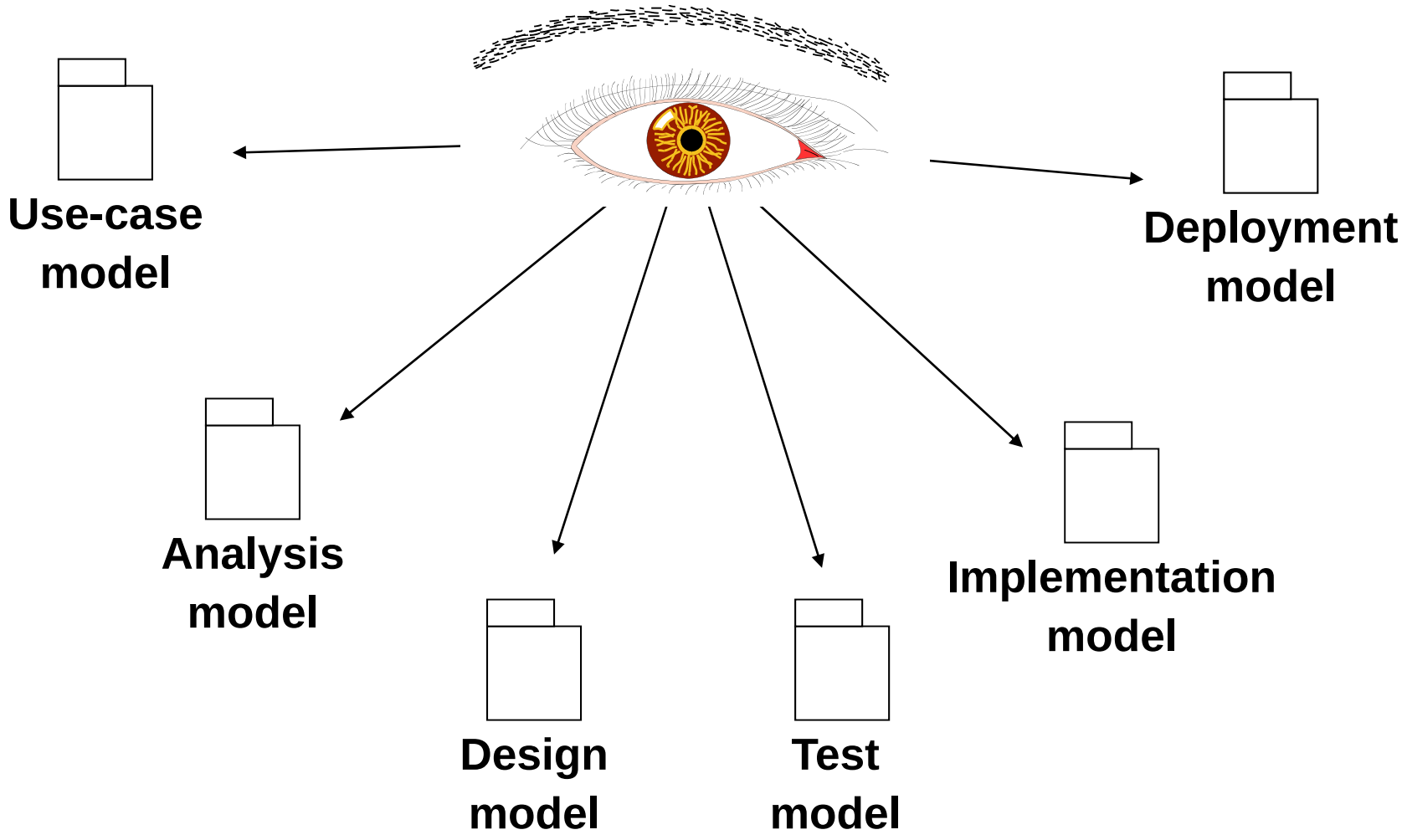


→ Ivar Jacobsen, Grady Booch, James Rumbaugh: „The Unified Software Development Process“, Addison-Wesley, 1999.

USDP Phasen

- Konzeption (‘Inception’)
  - Umfang des Produktes und dessen Eigenschaften festlegen
  - Machbarkeitsstudien aus wirtschaftlicher Sicht abschließen
  - Die größten Risiken ausschließen
- Ausarbeitung (‘Elaboration’)
  - Möglichst viele Anforderungen erfassen
  - Entwickeln des architektonischen Grundrisses
  - Weitere Risiken ausschließen
  - Abschließend: Kostenschätzung für die nächste Phase
- Konstruktion (‚Construction’)
  - Komplette Entwicklung des Systems
  - Fertig für die Auslieferung an den Kunden
- Inbetriebnahme (‘Transition’)
  - Sicherstellen, dass das Produkt an die User ausgeliefert werden kann
  - User lernen den Umgang mit dem Produkt

Die sechs USDP-Modelle (Sichten der Anwendung)



Zusammenfassung

- Softwareentwicklungsprozess
- Softwareentwicklungsprozess-Modell
  - Wasserfall
  - Spiral
  - Issue-Based
  - Unified

Weiter mit Foliensatz „Agile Softwareentwicklung“