Einführung in die Informatik II

SS 2012

1 Software-Entwicklung

1 Software-Entwicklung ...

Lernziele

- Grundsätzliche Vorgehensweise beim Software-Entwurf
- Traditionelle Methoden zur Programmentwicklung
- Idee der objektorientierten Software-Entwicklung

Literatur

- **[GS02], Kapitel 12**
- [Ba05], LE 1
- [Oe05], Kapitel 2.1 und 2.2
- Wikipedia: http://de.wikipedia.org

1.1 Motivation

Kleine Projekte

- Beispiel aus dem Alltag: Bau einer Hundehütte
  - Keine formale Vorgehensweise (Projektorganisation) nötig
  - Kaum Planung erforderlich
  - Ausführung durch eine einzige Person möglich
  - Einfache Werkzeuge ausreichend
    - Säge, Hammer, ...
- In der SW-Entwicklung: (SW = Software)
  - Projekte bis max. 10000 Codezeilen / 2 Personenjahre

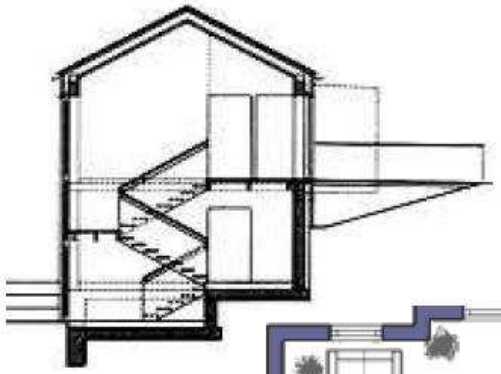
1.1 Motivation ...

Große Projekte

- Beispiel aus dem Alltag: Bau eines Einfamilienhauses
  - Wohldefinierte Vorgehensweise:
    - erst Plan, dann Rohbau,
  - Installation, Innenausbau, ...
  - Genaue Planung erforderlich
  - Ausführung durch ein Team
  - Mächtige Werkzeuge nötig:
    - Kran, Bagger, ...
- In der SW-Entwicklung:
  - Projekte bis etliche Mio. Codezeilen / ≥ 200 Personenjahre

Entwurfshilfsmittel: Beispiel Architektur

- Unterschiedliche Pläne und Modelle für unterschiedliche Zwecke und Aspekte

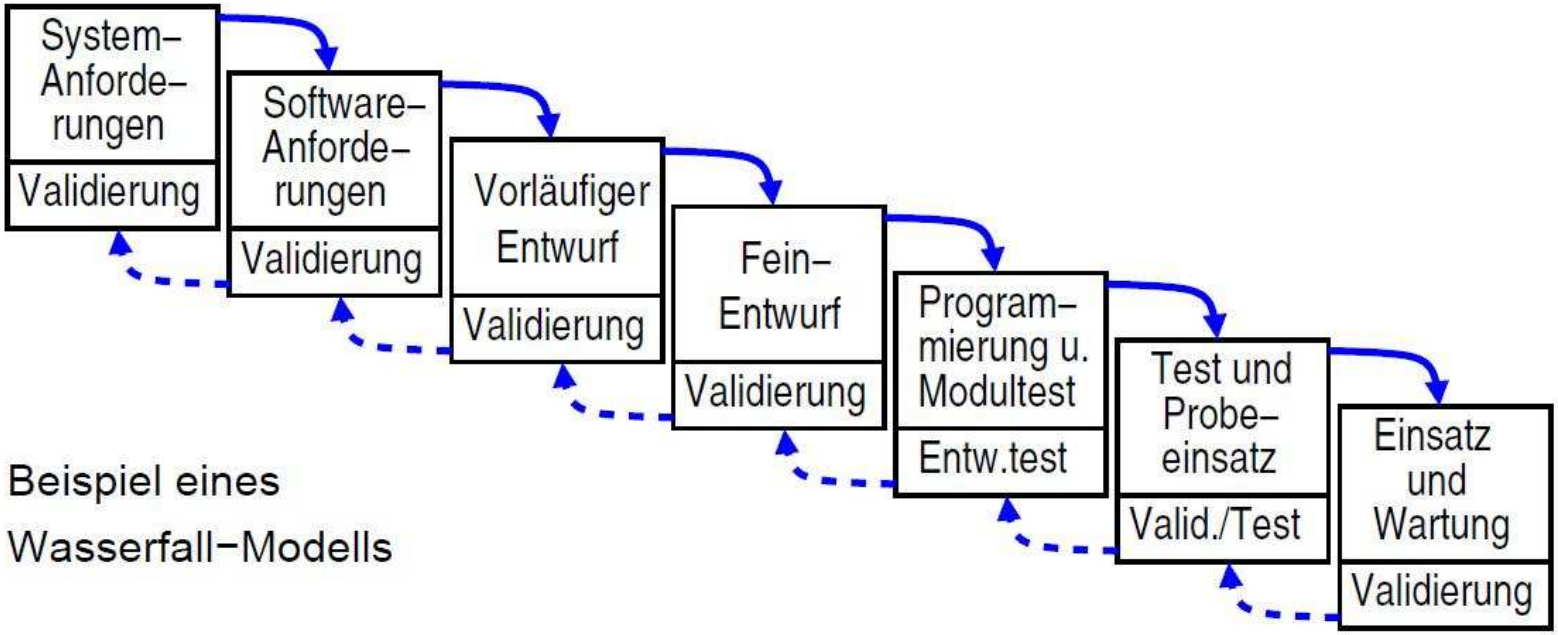


1.2 Vorgehensweise beim Software-Entwurf

- Zentrale Frage der Software-Technik:
  - was ist das beste Vorgehen bei der Durchführung eines Software-Projekts?
  - in erster Linie geht es also um Projektplanung, nicht um Programmiertechniken
- **Vorgehensmodelle** beschreiben erprobte Vorgehensweisen
- Wir betrachten hier exemplarisch Beispiele, u.a.:
  - das Wasserfall-Modell als grundlegendstes Modell
  - das Spiralmodell als Beispiel für ein nicht mehr strikt sequentielles Vorgehen
- Weitere Details: Vorlesungen ”Softwaretechnik I - III“

1.2.1 Wasserfall-Modelle

- Eine der ersten systematischen Vorgehensmodelle
  - Grundidee: Entwicklung als feste Abfolge von Phasen



Beispiel eines
Wasserfall–Modells

1.2.1 Wasserfall-Modelle ...

- Projektabschnitte in typischen Phasenmodellen:
  - Problemanalyse und Anforderungsdefinition
  - Fachlicher Entwurf
  - Software-technischer Entwurf
  - Programmierung und Modultest
  - System-Integration und Systemtest
  - Installation, Betrieb und Weiterentwicklung

- Anmerkung zur Verteilung des Aufwands:
  - Analyse, Entwurf und Programmierung machen nur etwa 20% der Kosten aus
  - 80% der Kosten entfallen auf Wartung und Test!

1.2.1 Wasserfall-Modelle ...

Problemanalyse und Anforderungsdefinition

- Analyse und Abgrenzung des Gegenstandsbereichs des Projekts
  - z.B. Kontenführung einer Bank
- Ermitteln der **Anforderungen** des Auftraggebers an das Software-System
  - *was* und *wofür*?
  - festgehalten in schriftlicher Form: **Lastenheft**
- Projekterfolg hängt oft von sauberer Anforderungsanalyse ab
  - eigener Zweig der Software-Technik: *Requirements Engineering*

1.2.1 Wasserfall-Modelle ...

Fachlicher Entwurf

- Spezifikation der **Funktionen** des Systems aus **fachlicher** Sicht
- Grundlage ist i.d.R. ein Daten- bzw. Objektmodell des Gegenstandsbereichs
  - Objekte (z.B. Konten, Kunden, Überweisungen, ...)
  - Strukturierung der Objekte
  - Beziehungen zwischen Objekten
- Anwendungsmodell = Datenmodell + Beschreibung der Funktionen und Abläufe
- **Pflichtenheft** beschreibt geplante Leistung des Software- Systems als *Black Box*
  - präzise und vollständig, aber ohne Implementierungsaspekte

1.2.1 Wasserfall-Modelle ...

Software-technischer Entwurf

- Festlegung der **Struktur** der zu entwickelnden Software
- Gliederung des Systems in Teilsysteme (Komponenten, Pakete)
  - weitere Unterteilung in möglichst unabhängige Module
  - Module interagieren über Operationen oder gemeinsame Daten
- Wesentlich: exakte, formalisierte Spezifikation aller Schnittstellen im System

1.2.1 Wasserfall-Modelle ...

Programmierung und Modultest

- Codierung (= Umsetzung der Spezifikation) macht nur einen kleinen Teil der Entwicklung aus
- Wichtig: systematischer Test der Module
  - Testfälle mit ausgesuchten Testdaten
  - Wiederholung der Tests nach jeder Programmänderung

System-Integration und Systemtest

- Zusammenfügen der Module zu Subsystemen und zum Gesamtsystem
- Test der Subsysteme und des Gesamtsystems

1.2.1 Wasserfall-Modelle ...

Diskussion des Wasserfall-Modells

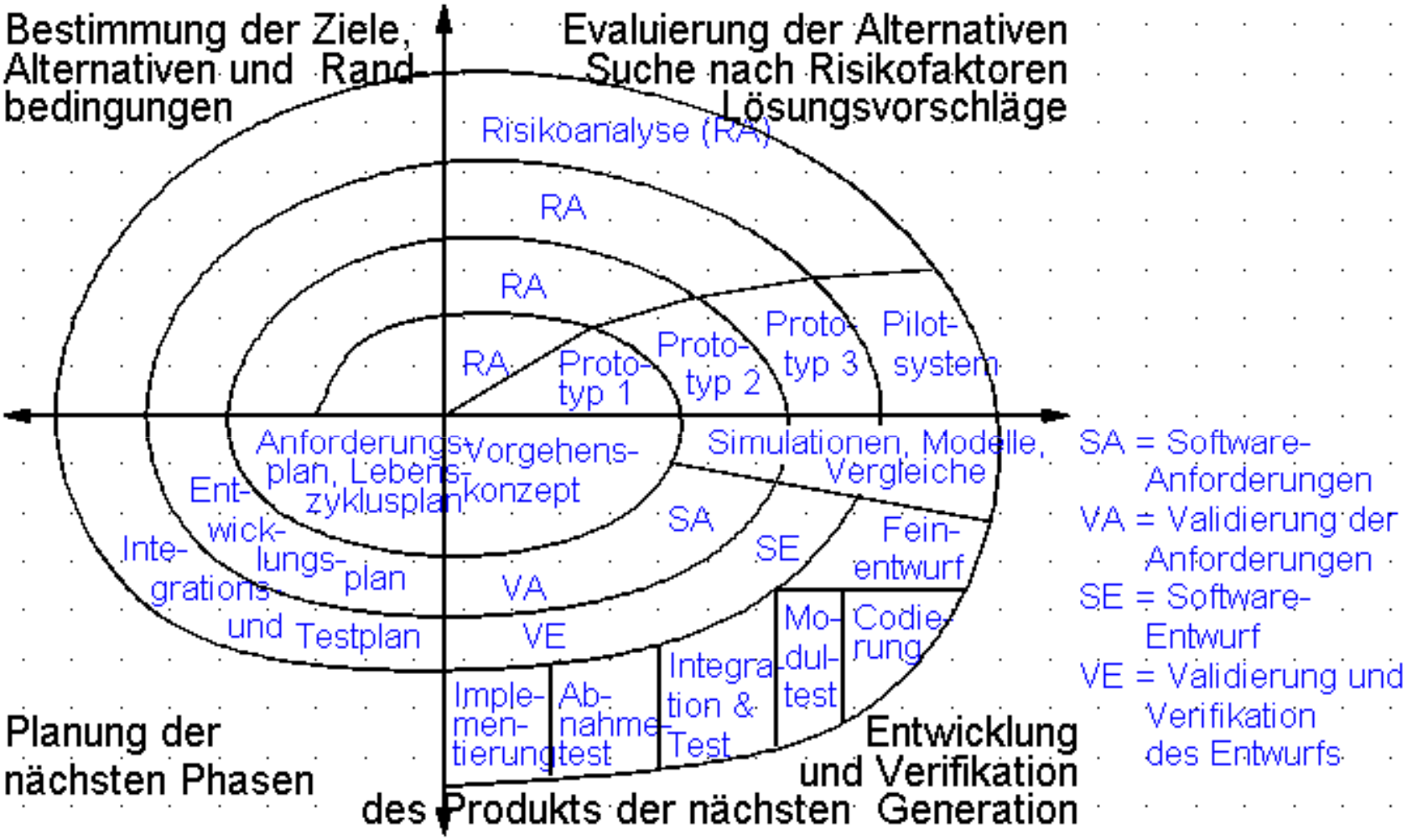
- Klare Abgrenzung der Phasen unrealistisch
  - Übergänge sind oft fließend
- Festlegung aller Anforderungen zu Projektbeginn sehr problematisch
  - Anforderungen werden oft erst im Projektverlauf klar
    - z.B. bei erstem Einsatz des Systems
- Keine Änderung der Anforderungen im Projektverlauf möglich
- Lauffähiges System erst sehr spät verf ügbar
- Vorgehensweise ist zu starr

- Heute daher: nicht-sequentielle Vorgehensmodelle

1.2.2 Das Spiralmodell

- Inkrementelles und iteratives Vorgehensmodell
- Ziel: Minimierung des Projektrisikos
- Zyklisches Durchlaufen von vier Aktivitäten:
  - Festlegung der Ziele, Alternativen und Randbedingungen
  - Evaluierung der Alternativen und Risikoanalyse
    - mit Hilfe von Prototypen
  - Entwicklung und Verifikation des Produkts der nächsten Generation
    - unter Verwendung eines geeigneten Vorgehensmodells
  - Planung der nächsten Phasen
- Spiralmodell ist eigentlich ein Meta-Modell

1.2.2 Das Spiralmodell ...



1.2.3 Das V-Modell

- Das V-Modell ist ein Vorgehensmodell zum Softwareentwicklungsprozess, also eine Richtschnur für die Organisation und Durchführung von IT-Vorhaben. Das "V-Modell 97", auch EstdIT (Entwicklungsstandard für IT-Systeme des Bundes) genannt, bzw. der Nachfolger "V-Modell XT" ist bei vielen zivilen und militärischen Vorhaben des Bundes verbindlich vorgeschrieben. Es ist ein wenig mit dem Wasserfallmodell vergleichbar, aber frühe Phasen werden mit späten über Testdaten verbunden (V-förmig)

1.2.3 Das V-Modell ...

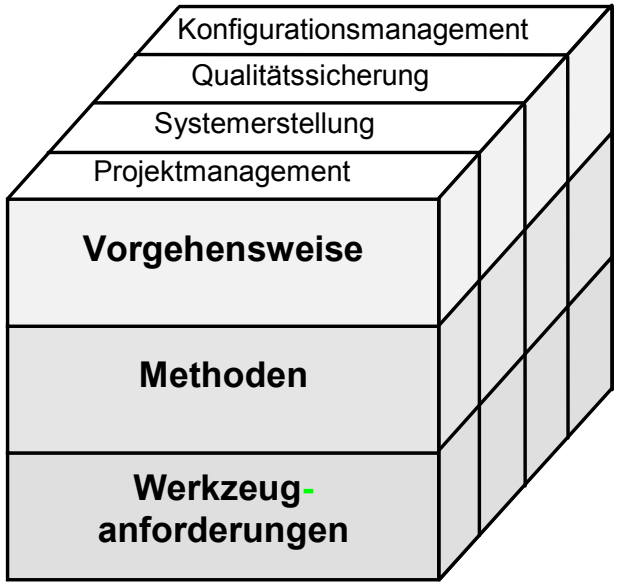
- Was ist das V-Modell ?

Der Entwicklungsstandard für IT-Systeme des Bundes besteht aus drei Teilen:

Vorgehensmodell (**Was** ist zu tun?)

Methodenzuordnung (**Wie** ist etwas zu tun?)

Funktionale Werkzeuganforderungen (**Womit** ist etwas zu tun?)



1.2.3 Das V-Modell ...

Zu jeder Aktivität existiert eine **Aktivitätenbeschreibung** als Arbeitsanleitung. Im zugehörigen **Produktfluss** wird angegeben

- welche Produkte als Eingangsprodukte benötigt werden,
- wo sie zuletzt bearbeitet wurden,
- welche Produkte erzeugt oder modifiziert werden und
- in welcher Folgeaktivität die erzeugten/modifizierten Produkte verwendet werden.

Dadurch wird der logische Ablauf des Vorgehens eindeutig festgelegt. Die Inhalte der Produkte werden in den **Produktmustern** festgelegt.

Der gesamte Prozess ist in Tätigkeitsbereiche untergliedert. Im V-Modell werden diese als **Submodelle** beschrieben:

- Die **Systemerstellung (SE)** erstellt das System bzw. die Softwareeinheiten.
- Das **Projektmanagement (PM)** plant, initiiert und kontrolliert den Prozess und informiert die Ausführenden der übrigen Submodelle.
- Die **Qualitätssicherung (QS)** gibt Qualitätsanforderungen, Prüffälle und Kriterien vor und unterstützt die Produkte bzw. den Prozess hinsichtlich der Einhaltung von Qualitätsanforderungen und Standard.
- Das **Konfigurationsmanagement (KM)** verwaltet die Produkte. Es stellt sicher, dass die Produkte eindeutig identifizierbar sind und Produktänderungen nur kontrolliert durchgeführt werden.

1.2.3 Das V-Modell ...

Elemente des V-Modell ‘97

Submodelle

sind charakterisiert durch Vorgehensweisen, Methoden und Werkzeuge. Das V-Modell unterscheidet die Submodelle Projektmanagement, Qualitätssicherung, Konfigurationsmanagement und Systemerstellung

Aktivitäten

sind Aufgaben, die hinsichtlich ihrer Ergebnisse und Abwicklung genau spezifiziert sind. Aufgaben von Aktivitäten sind Erzeugen, Modifizieren (Zustandsänderung) und Manipulieren (Veränderung) von Produkten

Produkte

sind das Ergebnis einer Aktivität. Produkte können sein Code, Entwicklungsdokumente, begleitende Dokumente (Pläne) etc.

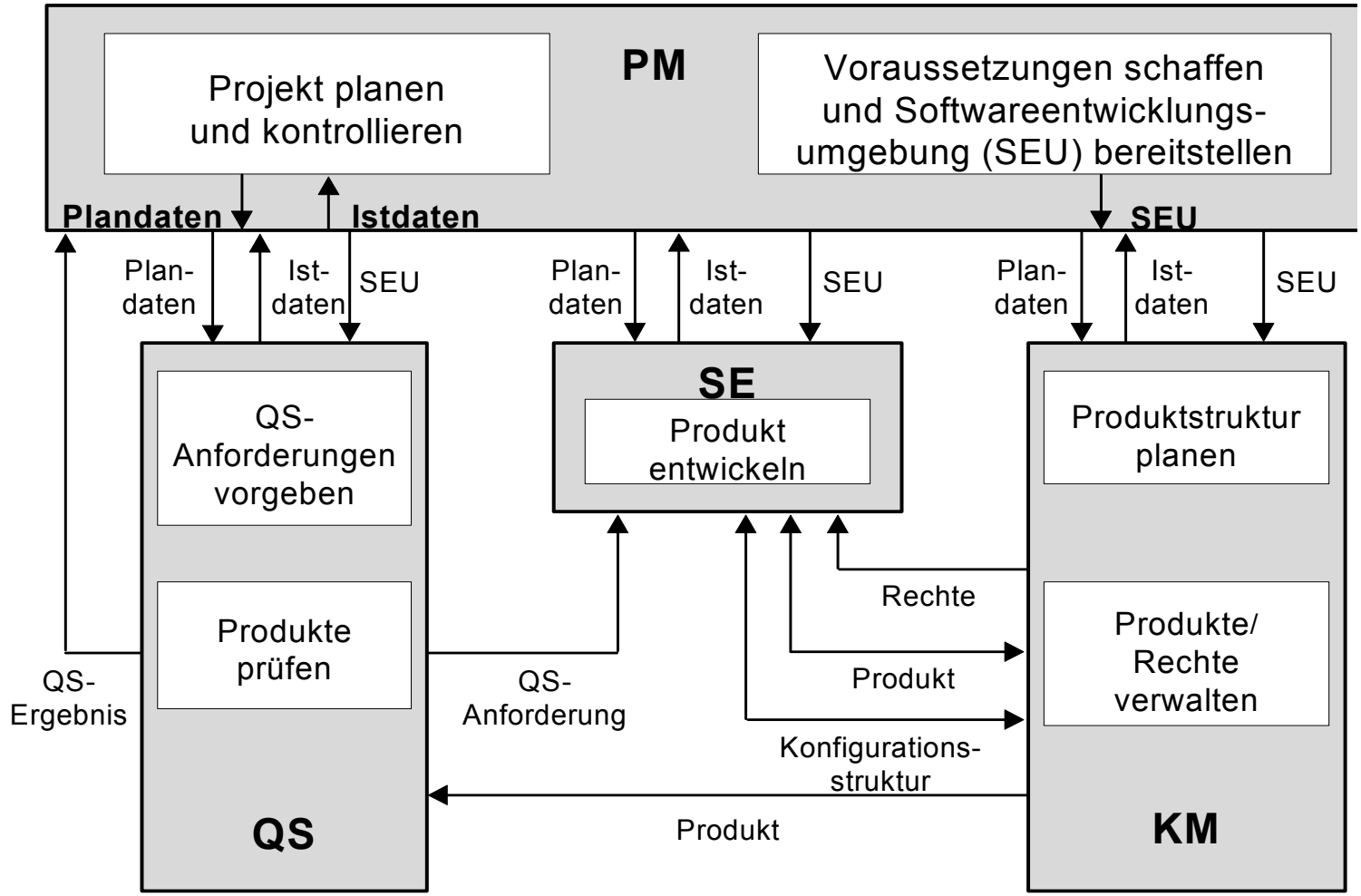
Produkte können die Zustände geplant, in Bearbeitung, vorgelegt und akzeptiert annehmen

Beschreibungsmuster

stehen als Templates für Produkte und Listen der Aktivitäten zur Verfügung

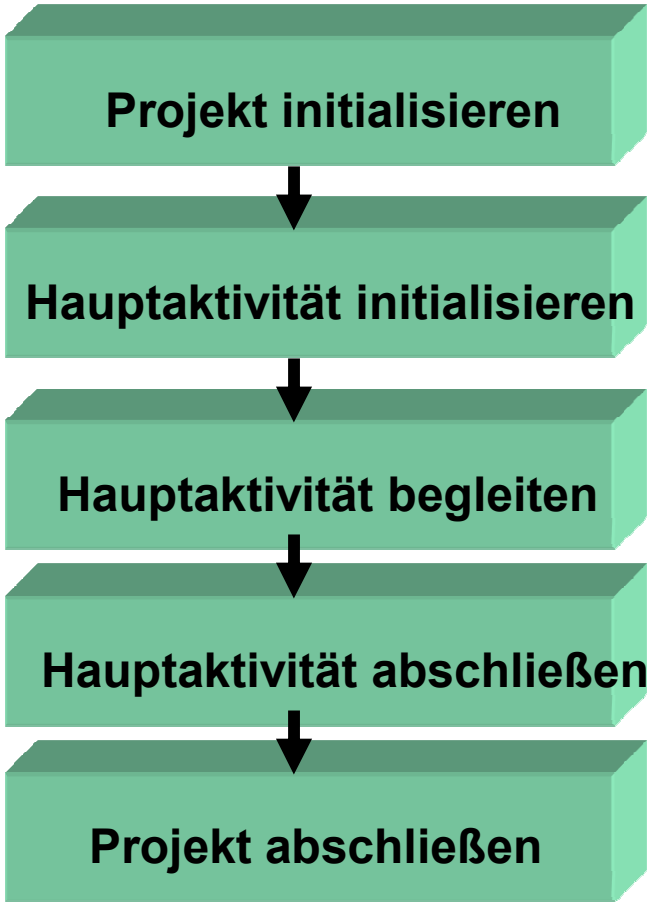
1.2.3 Das V-Modell ...

Zusammenspiel der Submodelle



1.2.3 Das V-Modell ...

Submodell Projektmanagement



1.2.3 Das V-Modell ...

Hauptaktivitäten des Subsystems PM

Projekt initialisieren:

- Regelt den organisatorischen Rahmen für das gesamte Projekt im *Projektplan* und *Projekthandbuch.*
- Legt Modalitäten für die projektinterne Zusammenarbeit und die Schnittstelle zum Auftraggeber fest.
- Erfordert Anpassung des Vorgehensmodells (*Tailoring)*.

Projekt begleiten mit den Phasen

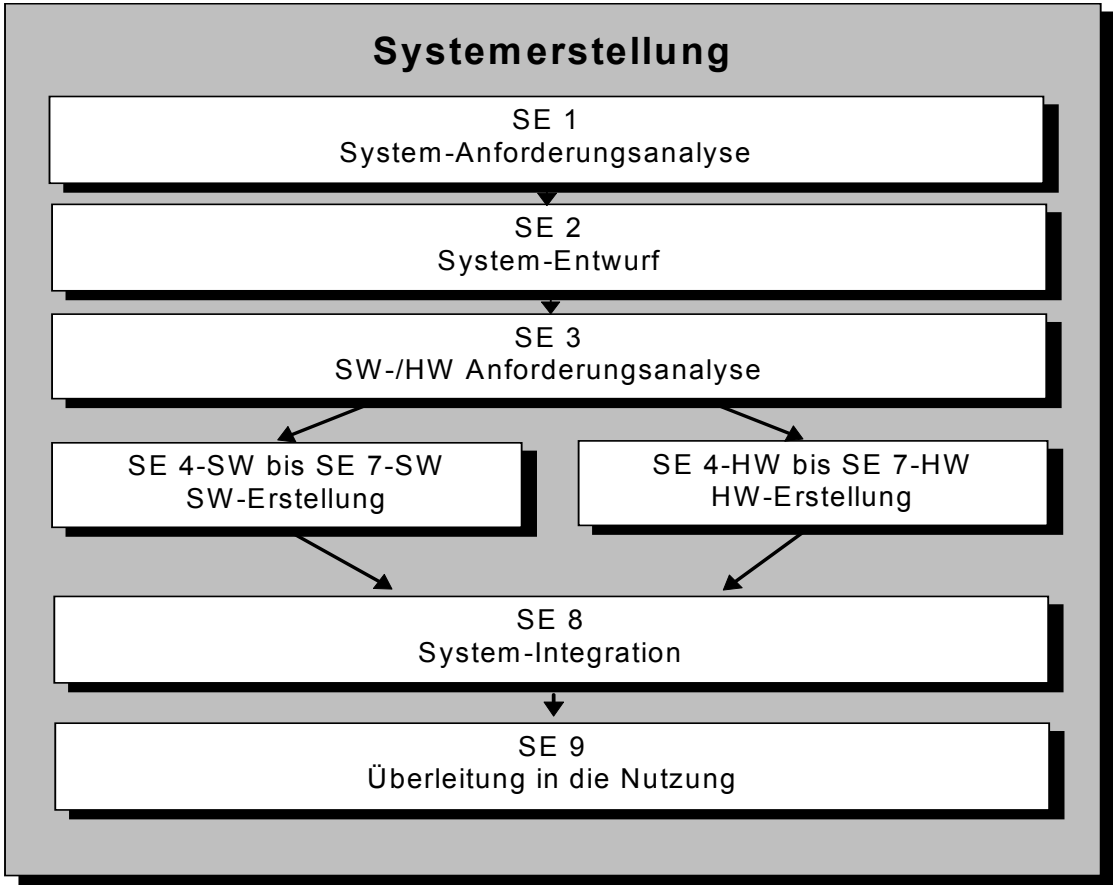
- *Initialisieren,*
- *Begleiten*
- und *Abschließen* der einzelnen Aktivitäten im Projekt.

Projekt abschließen:

- Aufbereitung der Ergebnisse,
- Soll-Ist-Vergleich mit Verbesserungsvorschlägen.

1.2.3 Das V-Modell ...

Submodell Systemerstellung SE



1.2.3 Das V-Modell ...

Hauptaktivitäten des Subsystems SE -1

- **System-Anforderungsanalyse (SE 1)**

  Beschreibung der Anforderungen an das zu erstellende System und seine technische und organisatorische Umgebung; Durchführung einer Bedrohungs- und Risikoanalyse; Erarbeiten eines fachlichen Modells für Funktionen/Daten/ Objekte.

- **System-Entwurf (SE 2)**

  Zerlegung des Systems in Segmente sowie SW- (Software-) und HW- (Hardware-) Einheiten.

- **SW-/HW Anforderungsanalyse (SE 3)**

  Die technischen Anforderungen an die SW- und ggf. HW-Einheiten werden präzisiert. Von hier ab spaltet sich der weitere Fortgang in die SW-Entwicklung und ggf. in die HW-Entwicklung

  SE4-SW bis SE7-SW
  SE4-HW bis SE9-HW

1.2.3 Das V-Modell ...

Hauptaktivitäten des Subsystems SE -2

- **Software-Entwicklung (SE4-SW bis SE7-SW)**

  Die Software-Entwicklung hat nach einem dem Projekt adäquaten Prozessmodell zu erfolgen.

- **System-Integration (SE 8)**

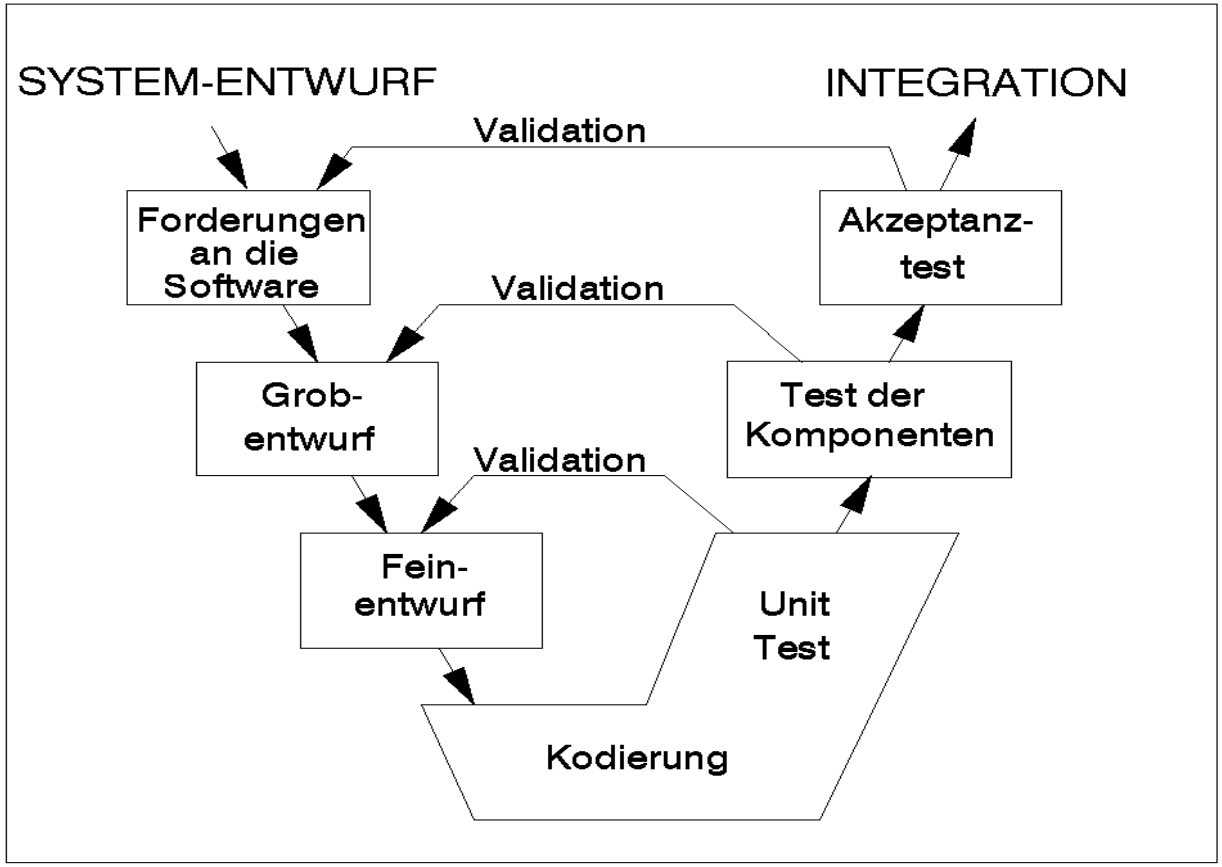
  Integration der verschiedenen Software- und Hardwareeinheiten zu einem Segment und Integration der Segmente (falls vorhanden) zum System.

- **Überleitung in die Nutzung (SE 9)**

  Beschreibung aller Tätigkeiten, die notwendig sind, um ein fertiggestelltes System an der vorgesehenen Einsatzstelle zu installieren und in Betrieb zu nehmen.

1.2.3 Das V-Modell ...

Teil-Submodell Softwareentwicklung



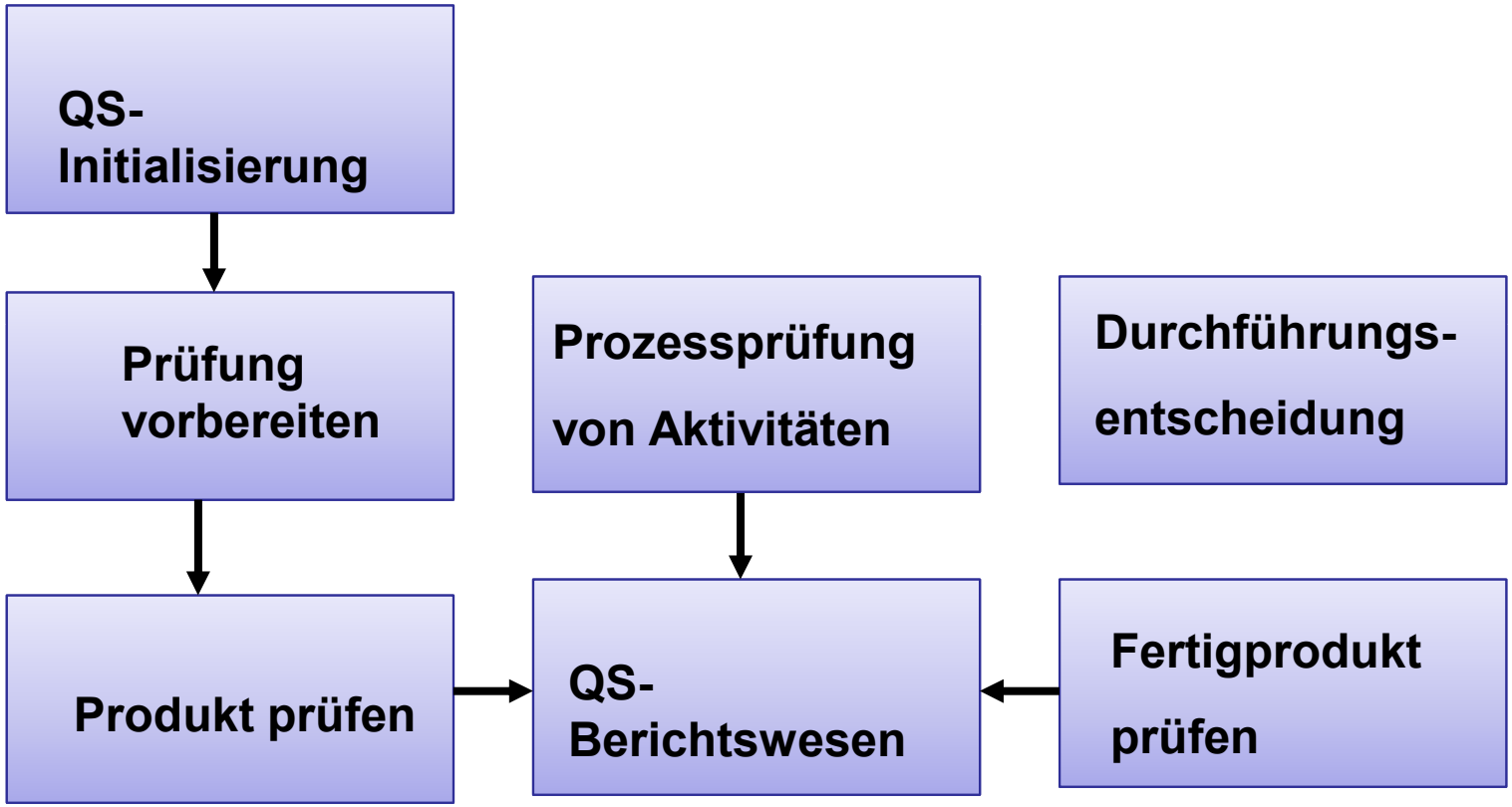
1.2.3 Das V-Modell ...

Kernaktivitäten und -Produkte des Submodells Softwareentwicklung

Aktivität	Produkt
Systemanforderungsanalyse	Anwenderforderungen
Systementwurf	Systemarchitektur
Softwareanforderungsanalyse	Technische Anforderungen
Softwaregrobentwurf	Softwarearchitektur
Softwarefeinentwurf	Softwareentwurf
Softwareintegration	Module und Datenbanken
Softwareimplementierung	Softwareeinheiten
Softwareintegration	System
Systemintegration	Installiertes System
Nutzerfreigabe	

1.2.3 Das V-Modell ...

Submodell Qualitätssicherung QS



Hauptaktivitäten des Subsystems QS -1

QS-Initialisierung (QS 1)

Die QS-Initialisierung legt den organisatorischen und abwicklungstechnischen Rahmen im **QS-Plan** und in Prüfplänen fest.

Prüfungsvorbereitung (QS 2)

Zur Prüfungsvorbereitung gehören die Erstellung von **Prüfspezifikation und -prozedur** und die Vervollständigung des **Prüfplans** sowie Anforderungen an die **Prüfumgebung**. Die Prüfkriterien müssen so festgelegt werden, dass der Erfolg oder Misserfolg einer Prüfung eindeutig und nachvollziehbar entschieden werden kann.(Es kann nicht Qualität sondern nur die Erfüllung von Qualitätskriterien geprüft werden)

Prozessprüfung von Aktivitäten (QS 3)

Bei der Prozessprüfung (nach DIN 55350) wird festgestellt, ob *vorgegebene Vorgehensweisen* bei der Durchführung bestimmter Aktivitäten eingehalten werden.

Hauptaktivitäten des Subsystems QS -2

Produktprüfung (QS 4)

Die Produktprüfung erfolgt in zwei Stufen: Prüfung der *formalen Kriterien* und *inhaltliche Prüfung* des Produktes. Für die Prüfungen sind die Prüfspezikationen zu verwenden. Das Ergebnis wird in einem **Prüfprotokoll** festgehalten.

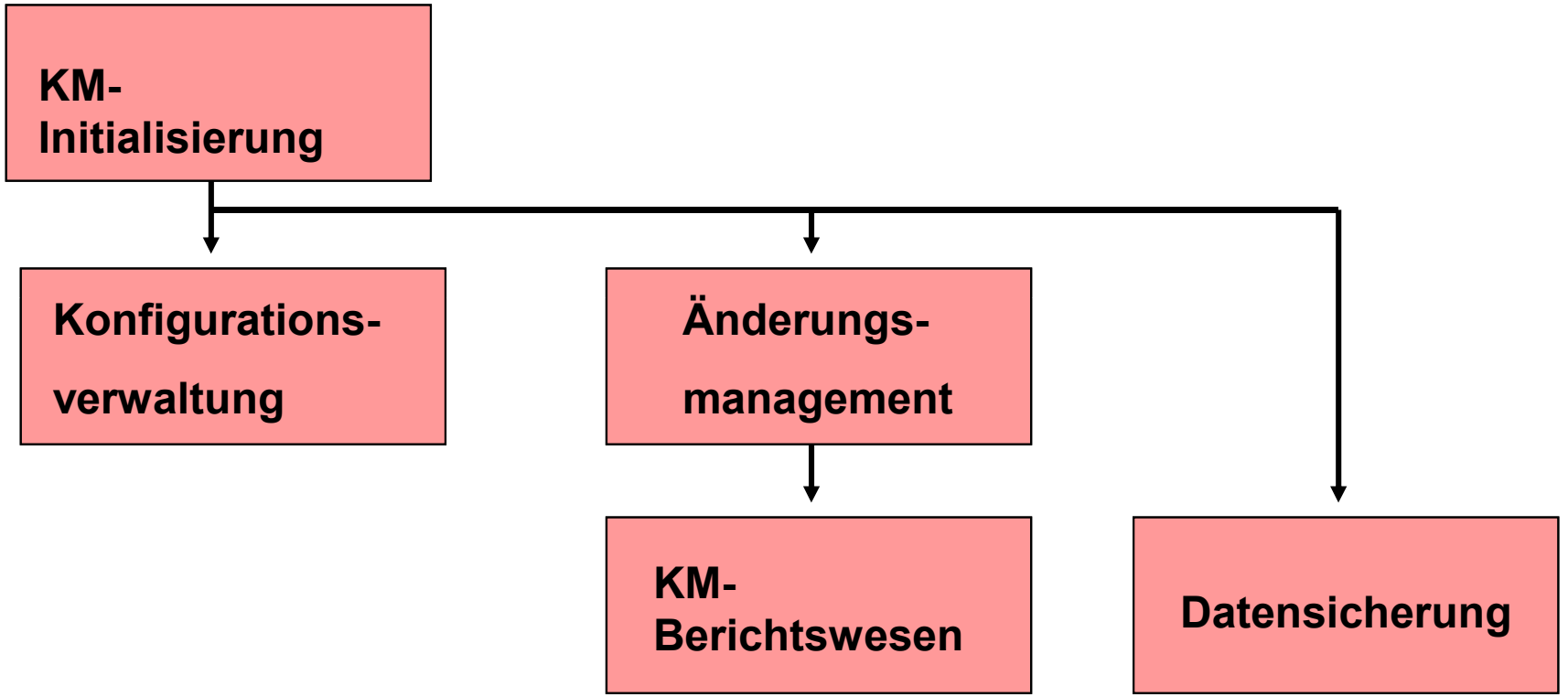
QS-Berichtswesen (QS 5)

Hier sind in regelmäßigen Abständen die Prüfprotokolle nach vorgegebenen Kriterien auszuwerten und die Ergebnisse dem

Projektmanagement vorzulegen.

1.2.3 Das V-Modell ...

Submodell Konfigurationsmanagement KM



1.2.3 Das V-Modell ...

Hauptaktivitäten des Subsystems KM

KM-Initialisierung (KM 1)

Die KM-Initialisierung regelt den organisatorischen und abwicklungstech-nischen Rahmen im **KM-Plan.** Des weiteren sind die Einsatzmittel (Produktbibliothek, Werkzeuge) bereitzustellen.

Produkt- und Konfigurationsverwaltung (KM 2)

Die Produkt- und Konfigurationsverwaltung umfasst das Verwalten von Produkten, Konfigurationen und Rechten.

Änderungsmanagement (KM 3)

Über das Änderungsmanagement werden eingehende *Fehlermeldungen, Problemmeldungen, Verbesserungsvorschläge* usw. erfasst und über die im KM-Plan festgeschriebenen Änderungsprozeduren einer kontrollierten Bearbeitung zugeführt.

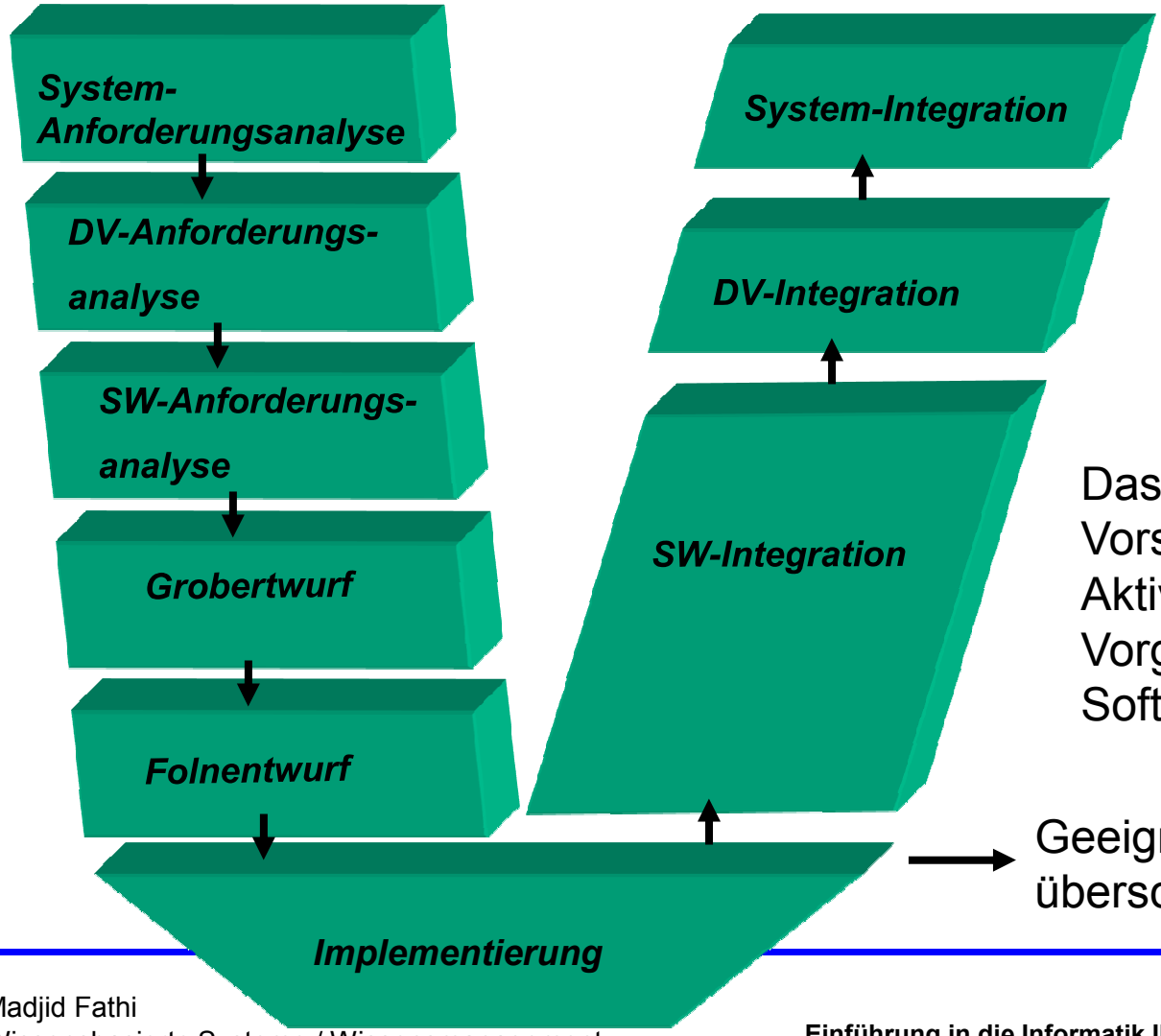
KM-Dienste (KM 4)

Unter KM-Dienste werden allgemeine Serviceleistungen zusammengefasst.

Zwei Beispiele sind angegeben.

1.2.3 Das V-Modell ...

Beispiel 1 - Wasserfallmodell im V-Modell



Das V-Modell enthält Vorschläge für alle Aktivitäten des Standard-Vorgehens bei der Softwareentwicklung.

Geeignet für leicht überschaubare Projekte

1.2.3 Das V-Modell ...

Beispiel 2 Iterative-Inkrementelle Vorgehensmodelle

Annahmen:

- Anforderungen sind unvollständig
- wichtige Erkenntnisse werden erst im Laufe des Projektes gewonnen

Vorteile:

- Evolutionäre SW-Entwicklung (Iterationsende: Programm)
- Reaktion auf Änderungen und Unvorhergesehenes einfacher
- Feinere Steuerung möglich

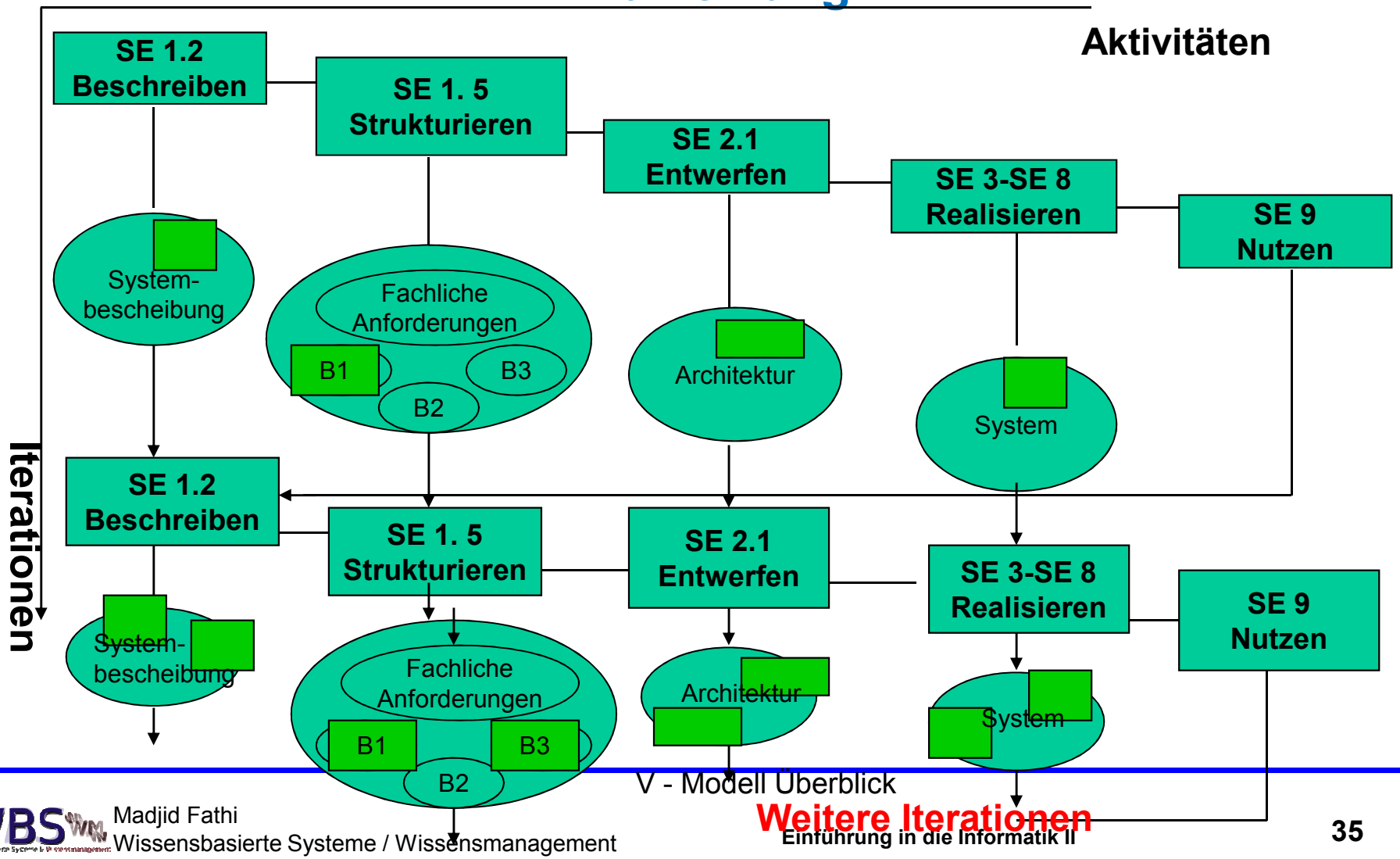
Nachteile:

- scheinbar mehr Aufwand
- Schwieriger Umzusetzen

→ Geeignet für Projekte mit Unwägbarkeiten

1.2.3 Das V-Modell ...

Ablauf der SE-Aktivitäten bei einer iterativ-inkrementellen Entwicklung



1.2.3 Das V-Modell ...

KOMPASS als V-Modell der Deutschen Post

Deutsche Post

Zielsetzung

- Der KOMPASS regelt die Entwicklung sowie die Pflege und Änderung von Systemen, deren Aufgabenerfüllung vorwiegend durch den Einsatz von Informationstechnik (IT) realisiert wird. Diese Systeme werden Applikationen genannt bzw. IT-Systeme, falls mehrere Applikationen in einem System zusammenarbeiten.

- Der KOMPASS soll den Entwicklungsprozess von Applikationen und IT-Systemen im Unternehmensbereich BRIEF standardisieren, Produktivität und Qualität verbessern und die Wahrnehmung der Verantwortung aller Beteiligten gewährleisten. Im einzelnen wird zum Erreichen folgender Ziele beigetragen:
  siehe folgende Folien

1.2.3 Das V-Modell ...

KOMPASS als V-Modell der Deutschen Post

Zielsetzung

➢**Verbesserung der Qualität**

- ➢Durch standardisiertes Vorgehen wird die Vollständigkeit der zu liefernden Ergebnisse am ehesten gewährleistet.
- ➢Definierte Zwischenergebnisse ermöglichen frühzeitige Prüfmaßnahmen.
- ➢Einheitliche Produktinhalte erleichtern die Lesbarkeit der Produkte und die Prüfmaßnahmen.
- ➢Mit der Nachvollziehbarkeit der Entwicklung und ihrer Ergebnisse leistet der KOMPASS einen entscheidenden Beitrag für das Qualitätsmanagement im Unternehmensbereich BRIEF.

1.2.3 Das V-Modell ...

KOMPASS als V-Modell der Deutschen Post

Zielsetzung

➢**Verringerung der Entwicklungs- und Wartungskosten**

- ➢Das standardisierte Vorgehen macht die Kalkulation des Aufwands transparenter. Kostenrisiken werden besser erkannt.
- ➢Einheitliche Regelungen reduzieren Reibungsverluste zwischen den Projektbeteiligten.
- ➢Bei standardisiertem Vorgehen werden universelle Lösungsansätze erkennbar und damit mehrfach verwendbar.
- ➢Fehlentwicklungen werden früher erkannt.
- ➢Das standardisierte Vorgehen und die konsequente Erstellung der Entwicklungsdokumentation reduzieren den Pflegeaufwand.

1.2.3 Das V-Modell ...

KOMPASS als V-Modell der Deutschen Post

Zielsetzung

➢**Verbesserung der Kommunikation zwischen allen Beteiligten**

- ➢Die Verwendung definierter Begriffe und einer gemeinsamen Terminologie reduziert Reibungsverluste zwischen den Projektbeteiligten (z. B. Fachseite, IT-Seite, externe Auftragnehmer).
- ➢Die Koordination von Projektpartnern sowie externer Auftragnehmer wird verbessert, da standardisierte Vorgaben gemacht werden können.
- ➢Die Zwischenergebnisse/Ergebnisse sind soweit standardisiert, dass sich andere Beteiligte - falls erforderlich - mit vertretbarem Aufwand einarbeiten können.

1.2.3 Das V-Modell ...

KOMPASS als V-Modell der Deutschen Post

Zielsetzung

➢**Verbesserung der Rahmenbedingungen für die Projektarbeit**

➢Der Projekt- und Teamgedanke wird durch die Festlegung von Projektrollen und durch die Beteiligung der Rolleninhaber an den Projektaufgaben über die Geschäftsbereiche bzw. Organisationseinheiten hinweg gefördert.

1.2.3 Das V-Modell ...

KOMPASS als V-Modell der Deutschen Post

- … gibt eine Übersicht aller gültigen Standards
- … ist der Best-Practice der Systemerstellung in IT BRIEF
- … zeichnet sich durch Allgemeingültigkeit und Verbindlichkeit aus
- … sorgt für einheitliche Projekt-Vorgehensweise
- … enthält die Werkzeugkiste der Systemerstellung
- … bietet effiziente Methoden der Systemerstellung

1.2.3 Das V-Modell ...

KOMPASS als V-Modell der Deutschen Post

Matrix - Phasen und Prozesse in KOMPASS:

Prozess Systemerstellung

Projekt-definition	Projekt-initiali-sierung	Analyse	Design	Reali-sierung	Fachliche Abnahme	BQS-Maß-nahmen	Roll-out	Projekt-abschluss

Begleitende Prozesse

- Prozess Projektmanagement
- Prozess Anforderungsmanagement
- Prozess Qualitätssicherung
- Prozess Konfigurationsmanagement

1.2.3 Das V-Modell ...

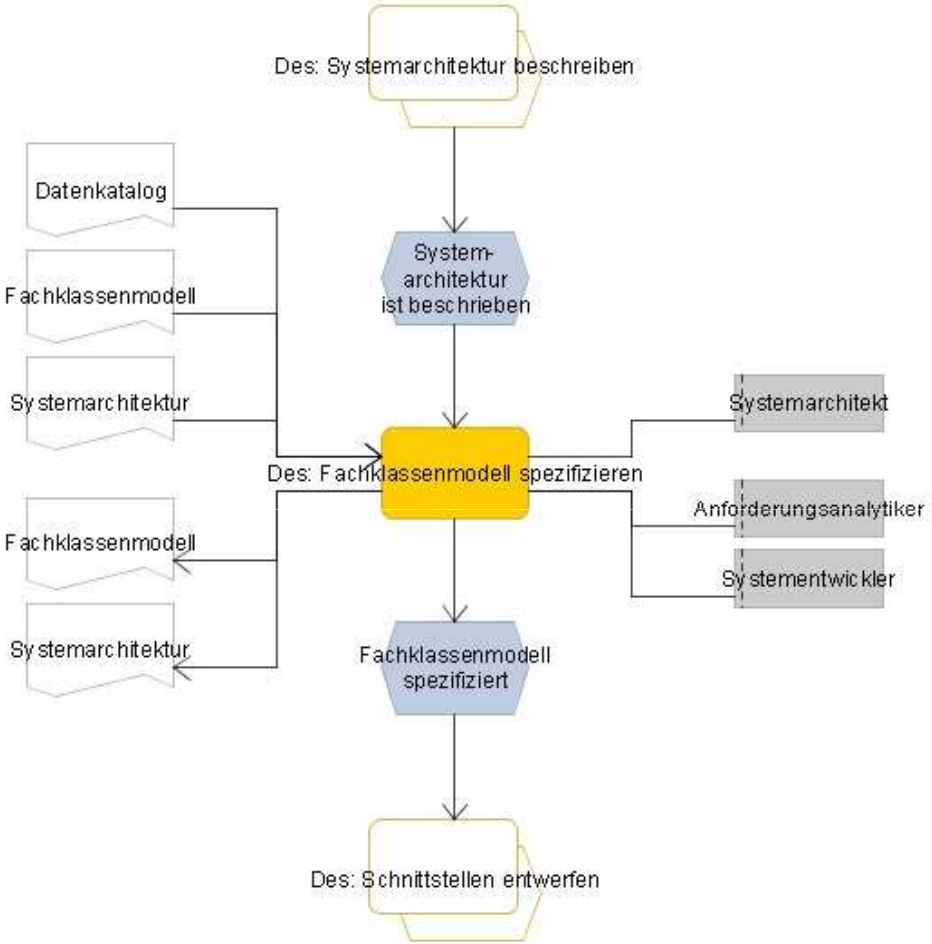
KOMPASS als V-Modell der Deutschen Post

Die 4 Querschnittsprozesse des KOMPASS

1.2.3 Das V-Modell ...

KOMPASS

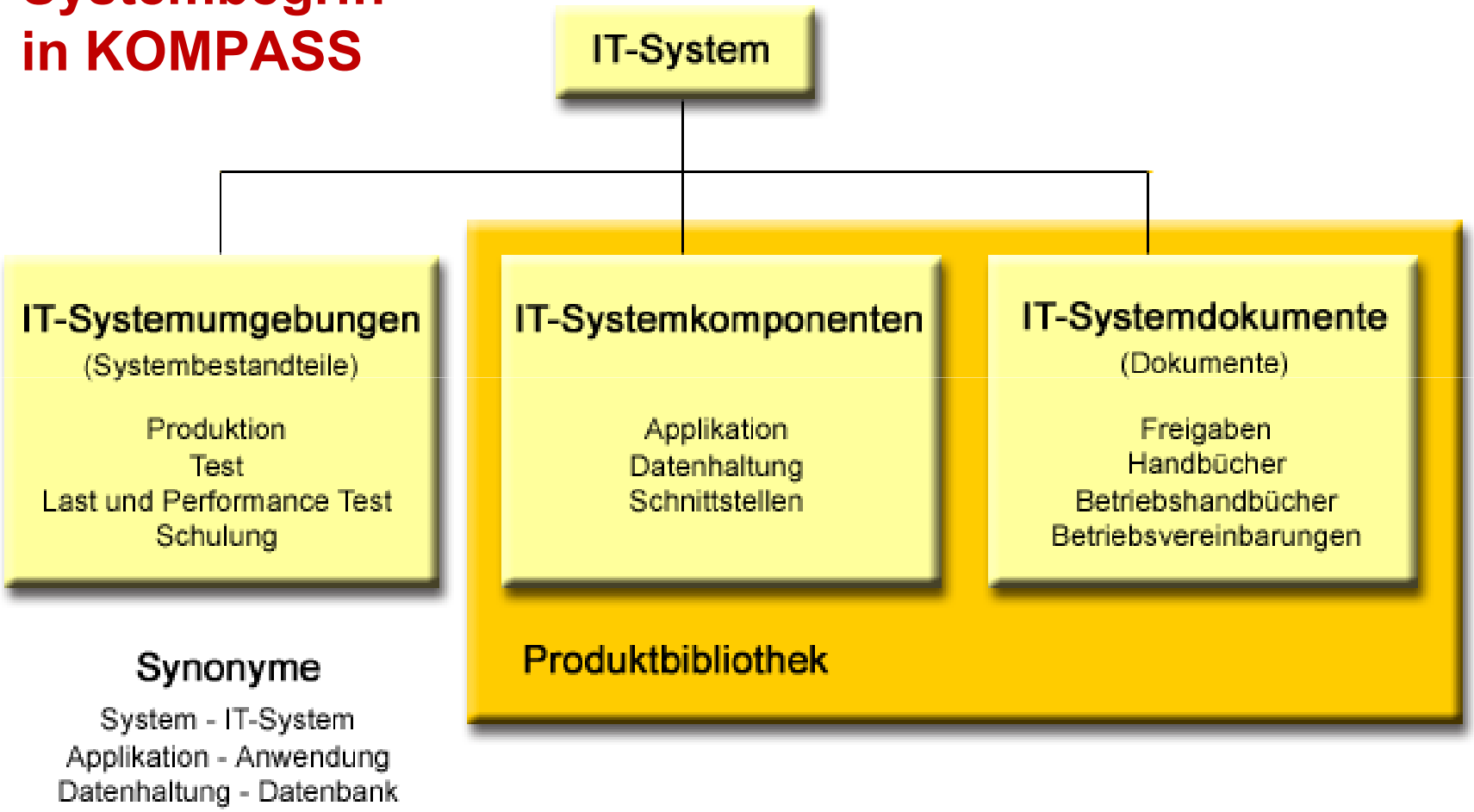
Aktivitäten in KOMPASS



Des: Fachklassenmodell spezifizieren 06.05.2009 00:50:00

1.2.3 Das V-Modell ...

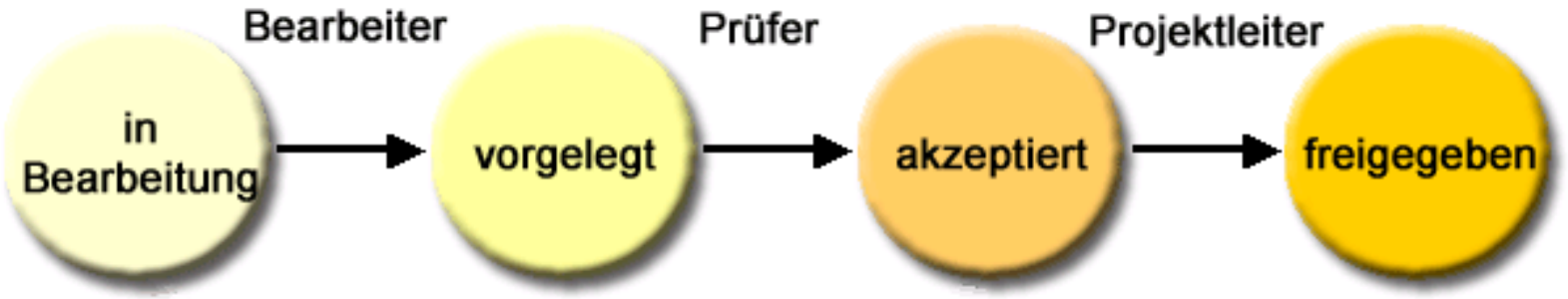
Systembegriff in KOMPASS



1.2.3 Das V-Modell ...

KOMPASS als V-Modell der Deutschen Post

Dokumentenstatus in KOMPASS



1.2.3 Das V-Modell ...

KOMPASS als V-Modell der Deutschen Post

- **Anforderungsmanagement**

  - Ein gemeinsames Verständnis der Anforderungen und eine projektbegleitende Kommunikation zwischen allen Stakeholdern ist einer der kritischen Erfolgsfaktoren für jedes Projekt.

    Nur der gemeinsame standardisierte und systematische Ansatz des Anforderungsmanagements auf Basis der Methoden des Requirements-Engineerings ermöglicht es dem UB BRIEF, alle Projekte gleichermaßen zu steuern und zum gemeinsamen Erfolg zu führen.

1.2.3 Das V-Modell ...

KOMPASS als V-Modell der Deutschen Post

➢ Herausforderung im Anforderungsmanagement (I)

- Während der frühen Phasen eines Projektes hat der fachliche Anforderer in der Regel nur eine vage Vorstellung eines Problems, eine geschäftliche Vision oder die Idee einer möglichen Lösung. Dasselbe gilt für den Realisierer, auch er bildet aus einer Idee eine möglichen Lösung.

  Aus verschiedenen Gründen sind aber beide Ideen nicht identisch. Daraus ergeben sich im Projektverlauf Missverständnisse, Fehlverhalten, abweichendes Design, Rework während der Testphase und andere Risikofaktoren für den Projekterfolg. Deshalb ist ein gemeinsames Verständnis der Anforderungen und der Lösungen zwischen Kunden und Lieferanten unabdingbar.

  Solche Anforderungen müssen entsprechend über den gesamten Projektverlauf verwaltet und in verschiedenen Detaillierungsebenen archiviert und für jeden zugänglich gemacht werden.

1.2.3 Das V-Modell ...

KOMPASS als V-Modell der Deutschen Post

➢ Herausforderung im Anforderungsmanagement (II)

Folgende Schwerpunkte müssen berücksichtigt werden:

- ➢ Der Scope des Projektes, also die Grenzen und Ziele, müssen gesetzt werden
- ➢ Detaillierte Kundenanforderungen müssen erhoben und dargestellt werden
- ➢ Die Umsetzung in Systemanforderungen muss vollzogen werden
- ➢ Die Konsistenz zwischen beiden Anforderungssystemen (Kundenanforderungen und Umsetzung in Systemanforderungen) muss gewahrt sein
- ➢ Alle Anforderungen müssen von der Phase "Projektinitialisierung" bis zur Phase "BQS Maßnahmen zur Betriebsüberführung" in der Realisierung nachvollziehbar sein (Traceability).

1.2.3 Das V-Modell ...

KOMPASS als V-Modell der Deutschen Post

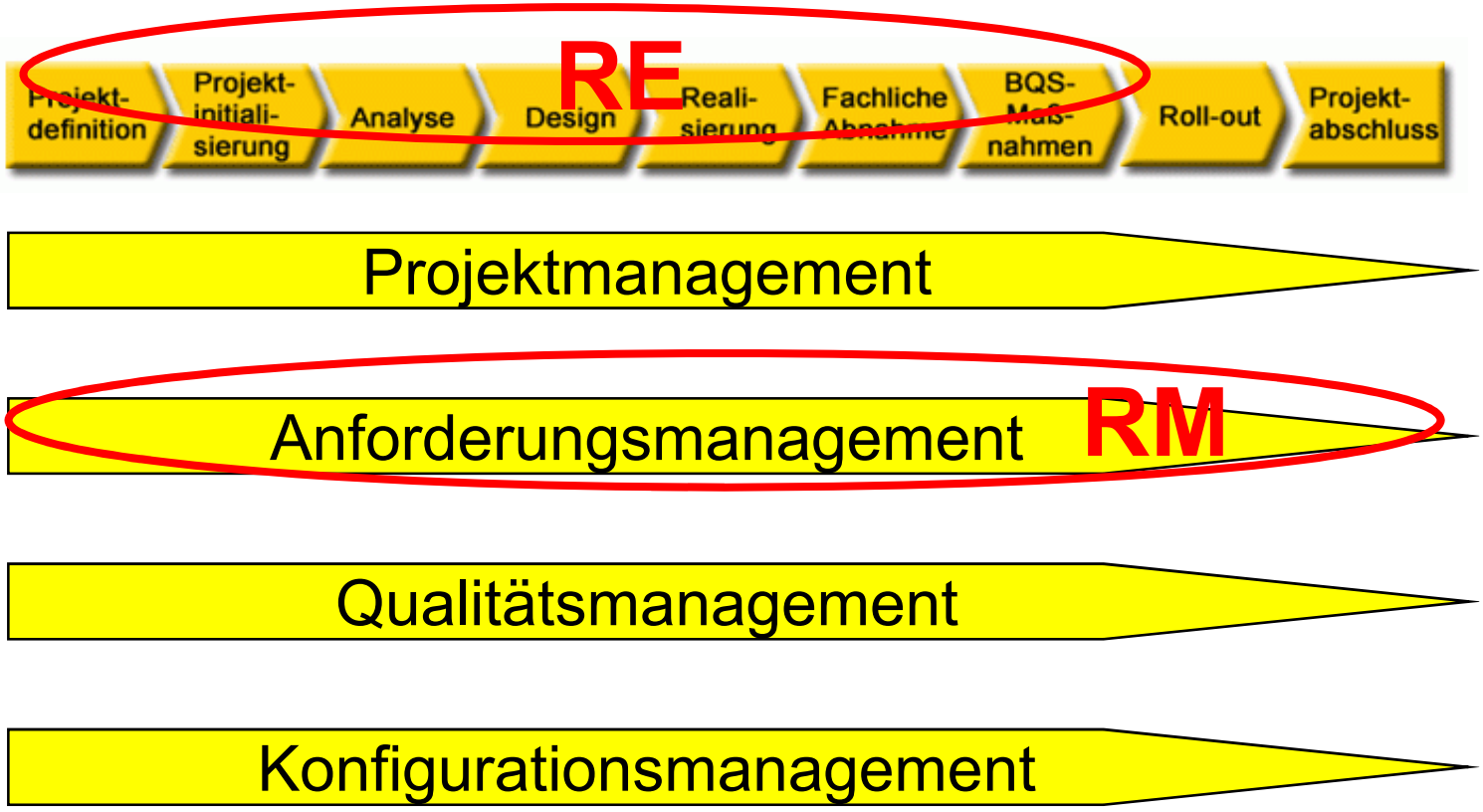
Unterschied
Requirements-Engineering / Requirements-Management

- Requirements-Engineering
  - Requirements Engineering umfasst die Anforderungsanalyse und das Requirements Management mit ingenieurmäßigem Vorgehen.
- Requirements-Management
  - Requirements Management umfasst Maßnahmen, welche die Anforderungsanalyse und die weitere Verwendung der Anforderungen unterstützen.

1.2.3 Das V-Modell ...

KOMPASS als V-Modell der Deutschen Post

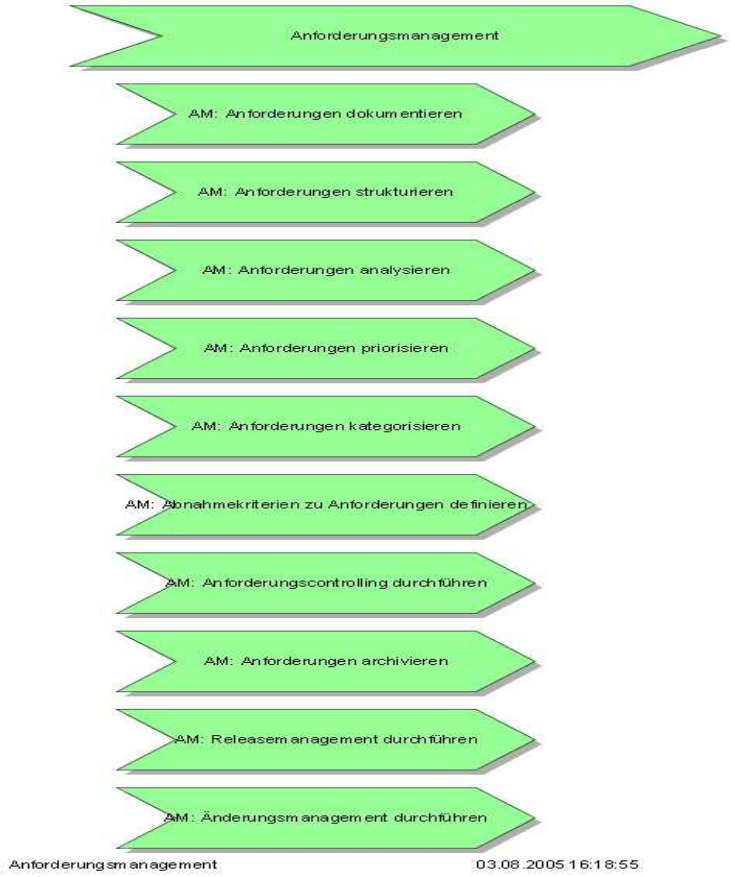
Requirements-Engineering / Requirements-Management



1.2.3 Das V-Modell ...

KOMPASS als V-Modell der Deutschen Post

AM: Anforderungen dokumentieren
AM: Anforderungen strukturieren
AM: Anforderungen analysieren
AM: Anforderungen priorisieren
AM: Anforderungen kategorisieren
AM: Abnahmekriterien zu Anforderungen definieren
AM: Anforderungscontrolling durchführen (...)
AM: Anforderungen archivieren
AM: Releasemanagement durchführen (...)
AM: Änderungsmanagement durchführen

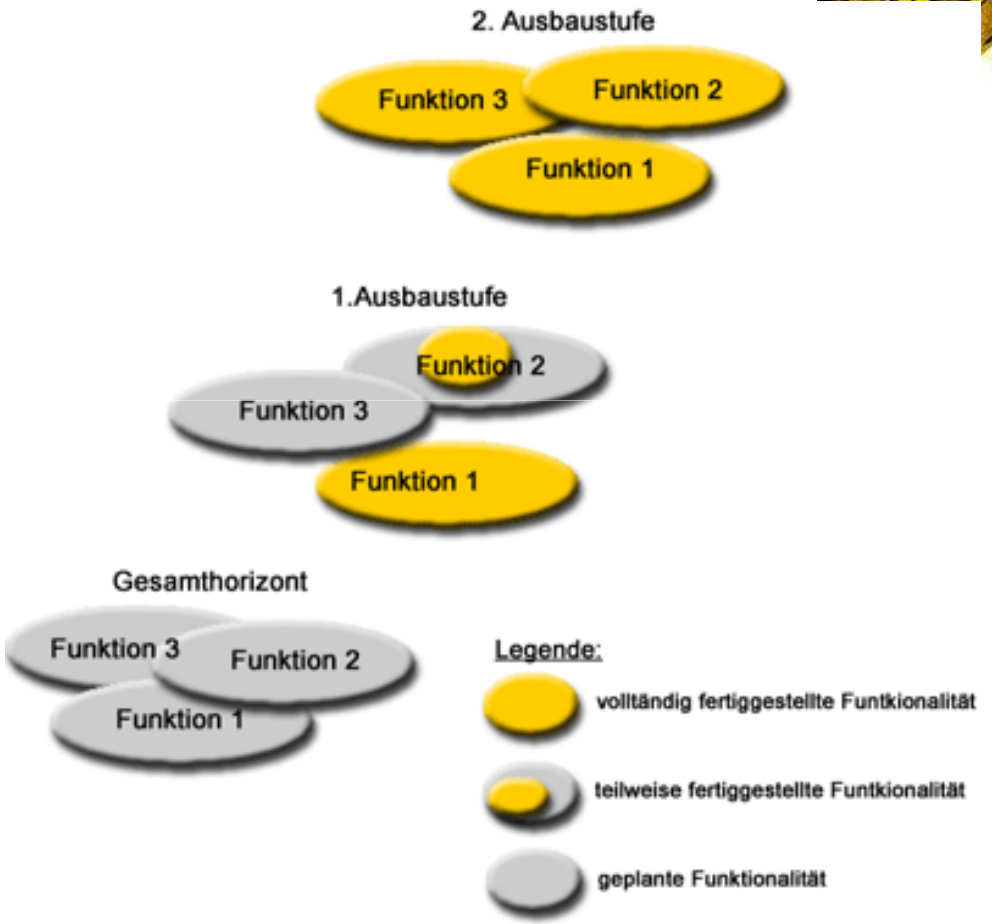


1.2.3 Das V-Modell ...

KOMPASS als V-Modell der Deutschen Post

KOMPASS ist aktuell, flexibel und methodenneutral

- Neutralität hinsichtlich Organisation, Methoden und Werkzeugen
- Flexibilität hinsichtlich der gewählten Methode (inkrementelle Entwicklung, OO-Entwicklung, Reengineering, Einsatz von Fertigprodukten, Wasserfallmodell)
- Erfahrungen aus der Praxis fließen kontinuierlich ein
- Versionierte Anpassung über einen Change Request Prozess
- Durchgängige Intranet-Verfügbarkeit mit Produkten zum Downloaden
- Konsequente Einführung und Verbreitung
- Ständige Abstimmung und Konsistenz zu IT-Guidelines



1.2.3 Das V-Modell ...

KOMPASS als V-Modell der Deutschen Post

Nutzen bei der Anwendung von KOMPASS

- Vertragsgrundlage
  - Eindeutige Festlegung von Lieferumfang und Arbeitsweise
- Eindämmung der Kosten über den Lebenszyklus
  - Straffung des Entwicklungsablaufs
  - Fehlentwicklungen werden frühzeitig erkannt
  - Reduzierung des Ressourceneinsatzes
- Verbesserung der Kommunikation
  - Verwendung definierter Begriffe
  - Transparenz der Arbeitsabläufe und Ergebnisse
  - Unterstützung der Fachseite bei der Formulierung der Anforderungen
- Verbesserung der Qualität
  - Definierte Zwischenergebnisse mit geeigneten Prüfmaßnahmen
  - Einheitliche Inhalte je Produkttyp

1.2.3 Das V-Modell ...

KOMPASS

- **Methode:**

  Wie soll es getan werden? In der Praxis gibt es eine Vielzahl von Methoden ("Auf welche Weise wird eine Aktivität getan?"). Durch die bewusst getroffene Eingrenzung der Anzahl und eine Überprüfung der Umsetzbarkeit im KOMPASS-Kontext wird dem Projektmanagement die Auswahl erleichtert.

- Aufwandsschätzungs-Methoden
  - Function Point Analyse
  - Rule of Thumb-Methode
  - Trendanalyse
  - UB BRIEF Function Point Aufwandsschätzverfahren
- Darstellungs-Methoden
  - Baumdiagramm
  - Organigramm
  - Unified Modelling Language
- Dokumenten-Review
  - Systematische Team-Reviews
- Kreativitäts-Methoden
  - Brainstorming
  - MindMapping
- Konfigurations- u. Versionsverwaltung
- Projektmanagement-Methoden
  - Balkendiagramm
  - Earned-Value-Analyse
  - Netzplan-Technik
- Qualitätsmanagement-Methoden
  - Projekt-Audit
  - Projekt-Review
- Requirements-Engineering-Methoden
  - Anforderungserhebungsmethoden
  - Natürlichsprachliche Anforderungsanalyse
  - Stakeholder-Analyse
  - Strukturierungsmethode - Volere
- Risikomanagement-Methoden
  - ABC-Analyse
  - Information Risk Assessment / Risk Treatment
  - Risikoassessment
- Testmethoden
  - Methoden zur Testfallermittlung - Funktionstest
  - Methoden zur Testfallermittlung - Integrationstest
  - GUI Testautomatisierung
- Übergreifende Methoden
  - Methode Business-Modelling-Discipline
  - Proof of Concept

1.2.3 Das V-Modell ...

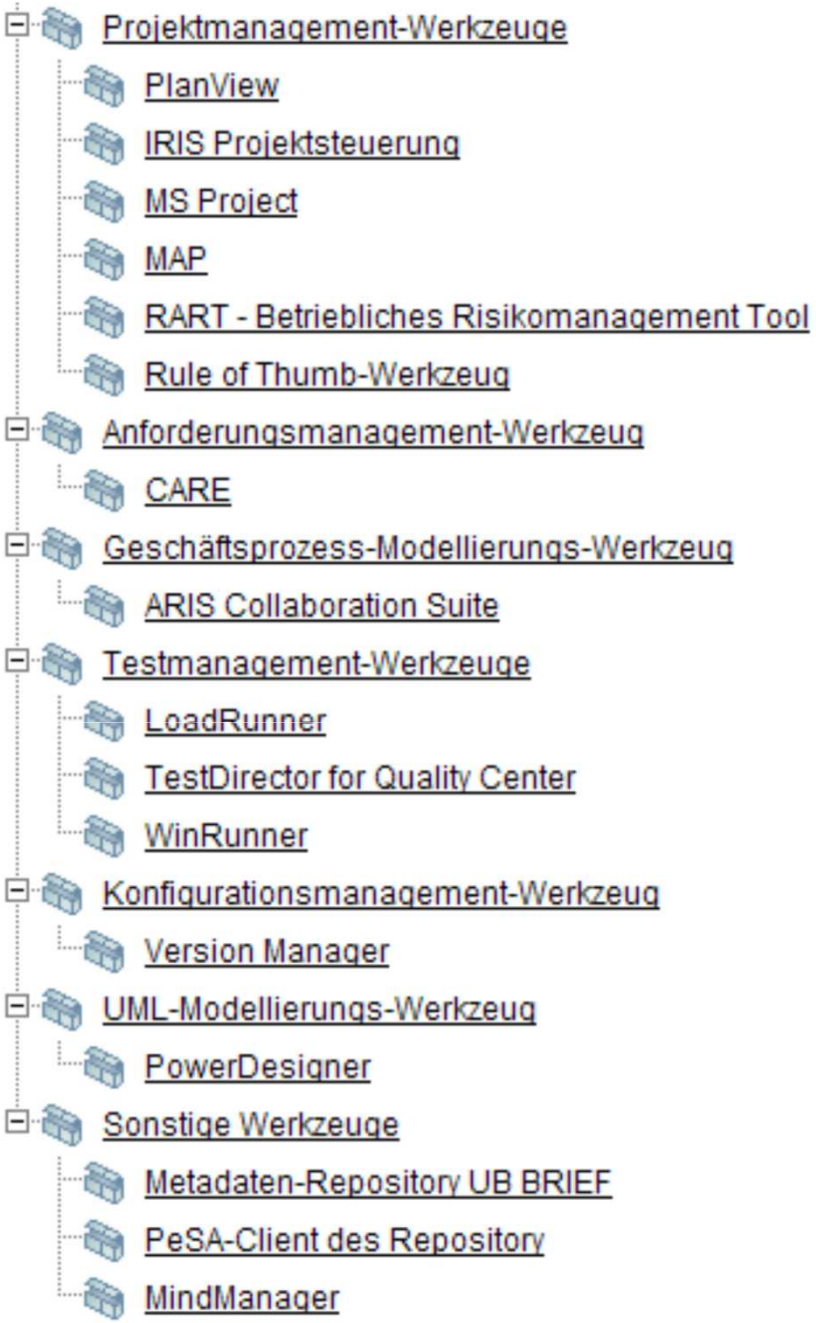
KOMPASS

- **Werkzeug:**

  Womit soll etwas umgesetzt werden? Bei der Auswahl machen insbesondere solche Methoden Sinn, die auch durch ein Werkzeug unterstützt werden

  ("Mit welchem Tool ist die Methode umzusetzen?"),

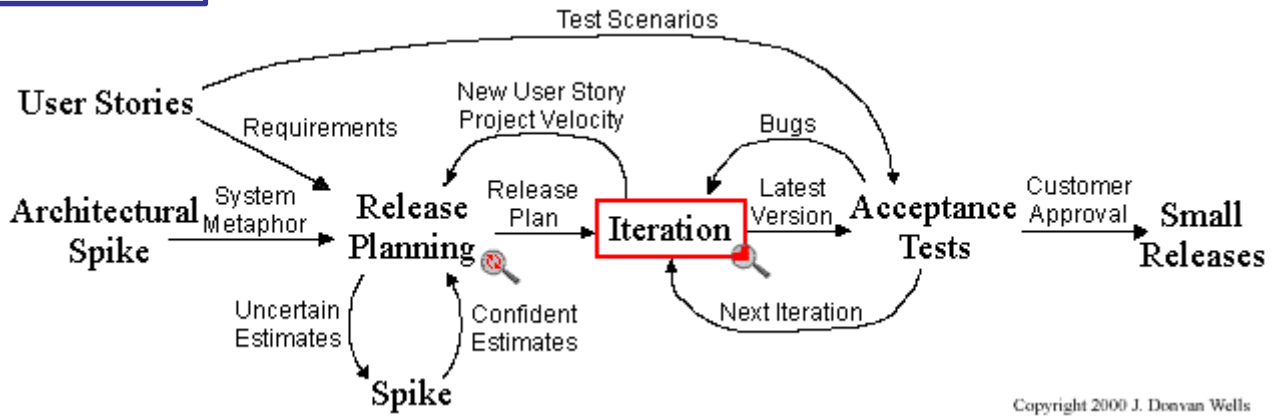
  falls dies vom Projektmanagement gewünscht wird



1.2.4 Extreme Programming XP

- XP (Extreme Programming) beschreibt eine agile iterative Vorgehensweise beim Softwareentwicklungs-prozess mit leichtgewichtiger Methodologie und wenig Dokumentation und Overhead.

Überblick



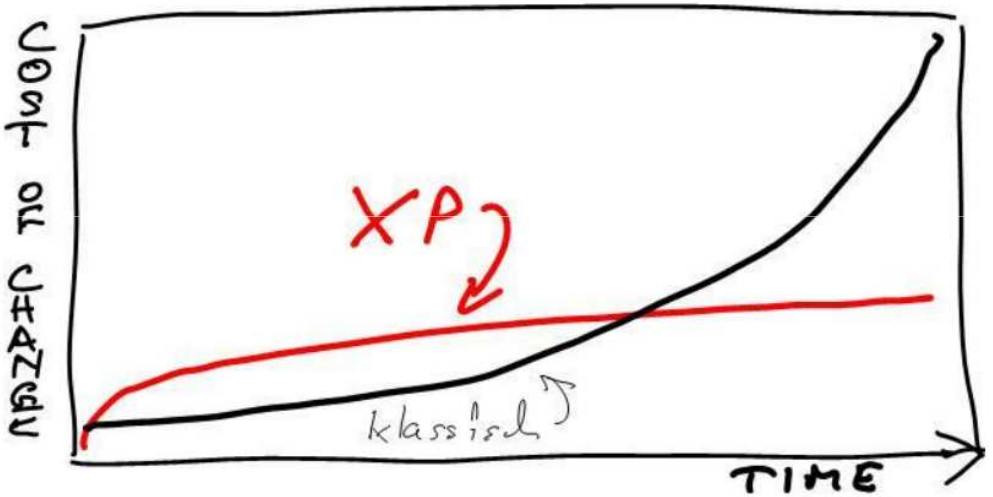
1.2.4 Extreme Programming XP...

Warum braucht man XP?

Risiken klassisch

- Zeitpläne
- Projekt abgebrochen
- Die Software wird unbrauchbar
- Fehlerrate
- Missverständnisse
- Vergoldung
- Personalwechsel

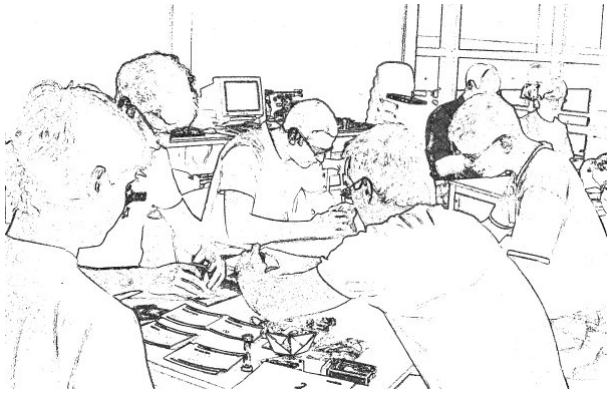
Kosten



Notwendigkeit
- Einfaches Design
- Automatische Tests
- Stetige Anpassung des Designs

1.2.4 Extreme Programming XP...

XP Team

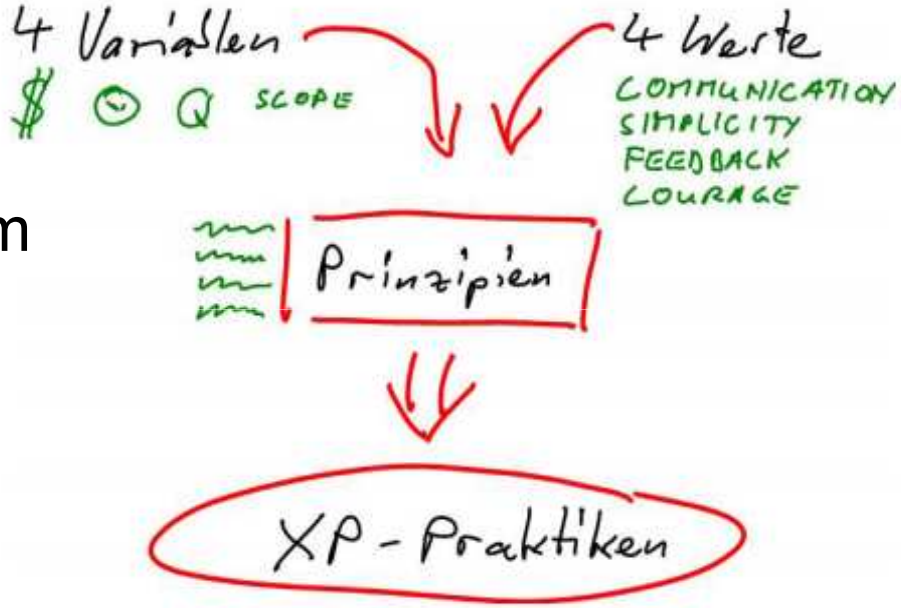


- Small group, at most 12 persons
- Group should fit into a single room
  - Ideally the group is co-located for the whole working time
- Special member roles:
  - **Representatives of the customer** (“on-site customer”)
  - **Tester:** Helps the customer to translate his stories into functional tests
  - **Coach:** Helps in keeping the XP discipline
  - **Tracker:** Continuously measures progress of the team and publishes it
  - **Consultant:** External provider of specific knowledge, active on the team only for a short time
  - **Big Boss:** Encourages the team

1.2.4 Extreme Programming XP...

Grundsätze

- Pair-Programming
- Testgesteuerte Entwicklung
- Alle Programmierer entwickeln das gesamte System
- Integration folgt unmittelbar auf Entwicklung



1.2.4 Extreme Programming XP...

4 Variablen

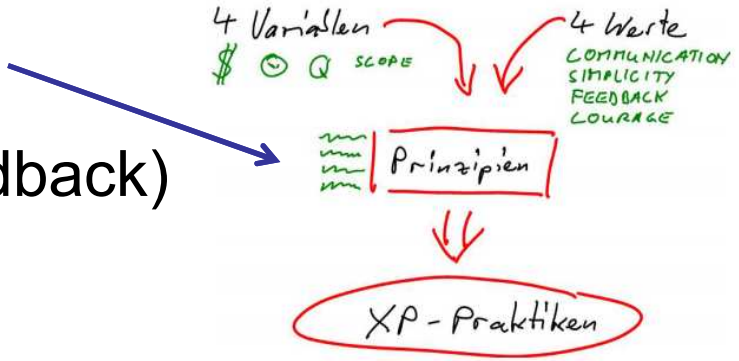
- Kosten
- Zeit
- Qualität
- Umfang (Scope)

4 Werte

- Kommunikation
- Einfachheit
- Feedback
- Mut

1.2.4 Extreme Programming XP...

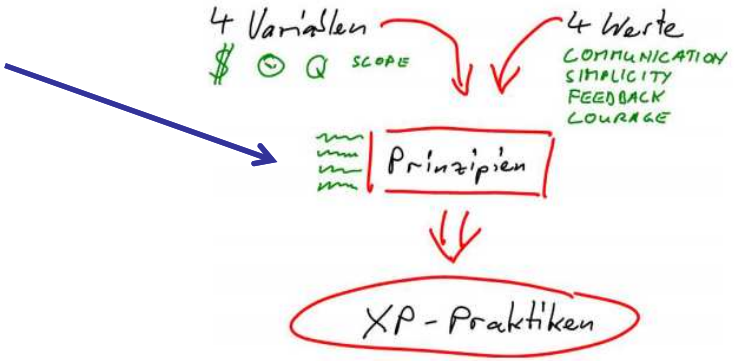
XP Prinzipien -1



- Unmittelbares Feedback (rapid feedback)
- Annahme der Einfachheit (assume simplicity)
- Inkrementelle Weiterentwicklung (incremental change)
- Änderungen willkommen heissen (embrace change)
- Qualitätsarbeit leisten (quality work)

1.2.4 Extreme Programming XP...

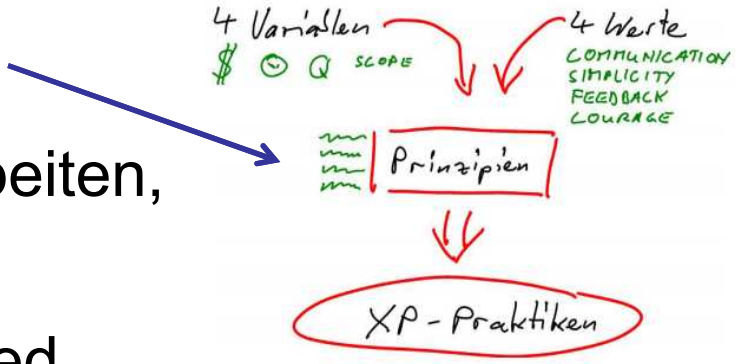
XP Prinzipien -2



- Lernen lehren (teach learning)
- Kleine Startinvestition (small initialinvestment)
- Play to win (DO NOT play not to lose)
- Experimente (concrete experiments)
- Offene, ehrliche Kommunikation (open, honest communication)

1.2.4 Extreme Programming XP...

XP Prinzipien -3



- MIT dem Instinkt der Mitarbeiter arbeiten, nicht dagegen
- Verantwortung akzeptieren (accepted responsibility)
- Lokale Adaption (local adaption)
- Reisen mit leichtem Gepäck (travel light)
- Ehrliche Messungen (honest measurement)

1.2.4 Extreme Programming XP...

A Development Episode (1)

- A looks at his stack of task cards. The top card says "Export Quarter-to-date Withholding". He addresses B.
- "Hi B, at this morning's standup meeting I heard you had finished the quarter-to-date calculation. Do you have time to help me with the export?" B agrees, and they are a pair now for some time.
- A and B need some additional information about the data structure and interrupt C for 30 seconds. They get the answer immediately.
- By looking at the existing export test cases, a simple abstraction (introduction of a superclass) is found by A and B. The code is restructured (refactored) with the superclass. All existing test cases are run successfully. Some other test cases may also profit from the new superclass; this is written down on the pair's to-do card.
- A and B together write the test case for the export function. During this, they note some ideas for the implementation on the to-do card.
- A and B run the test case, and it fails as expected, due to the missing implementation.

1.2.4 Extreme Programming XP...

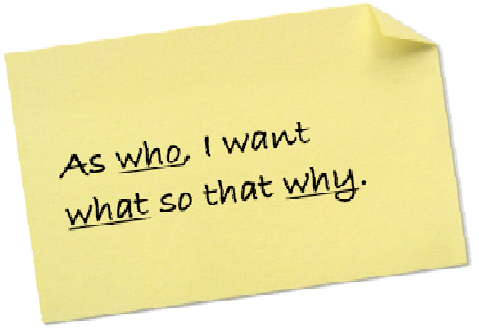
A Development Episode (2)

- During implementation, some other test cases come to the minds of either of the two, and are noted on the to-do card. Whoever of the two has the best ideas implements the export function and later the additional test cases, the other watches and comments. After a few iterations, all new test cases run successfully, as well as the old tests.
- What's left on the to-do-card? The two restructure (refactor) other test cases and make sure all tests run successfully.
- Look, the integration machine is free! The two go to integration machine, load their new code (tests and implementation) and run all the tests known. Strangely, a not too much related test fails. After a few minutes, it has been clarified why and all tests run successfully.
- The new version is released.

This is the whole XP lifecycle in a nutshell.

1.2.4 Extreme Programming XP...

User Stories



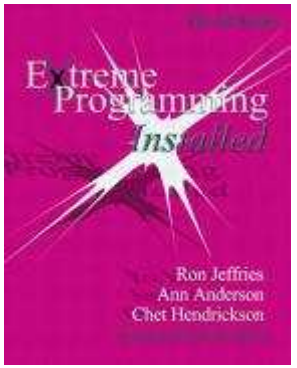
- *"Each user story is a short description of the behaviour of the system, from the point of view of the user of the system. In XP, the system is specified entirely through user stories."* (R. Jeffries, A. Anderson, C. Hendrickson, XP Installed, Addison-Wesley 2001, p. 24)
- Preparation:
  - Everybody (representing the customer) gets a card, tries to scribble on it and tears up the card.
- Process:
  - Customer writes story on card, possibly in iterations
  - Programmers "listen" - ask questions just for clarification
  - Stories are promises for conversation
- How many:
  - At least one per feature
  - One story implementable in a few days to a few weeks time

1.2.4 Extreme Programming XP...

Example User Stories

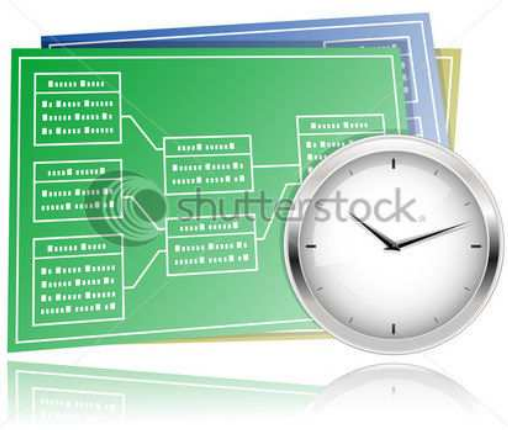
- “Union dues vary by union and are taken only in the first pay period of the month. The system computes the deduction automatically. The amount is shown in the attached table.”
- “When a transaction causes a customer’s account to go into overdraft, transfer money from the overdraft protection account, if any.”
- “Produce a statement for each account, showing transaction date, number, payee, and amount. A sample statement is attached - make the report look approximately like the sample.”

Examples from Jeffries/Anderson/Hendrickson: XP Installed



1.2.4 Extreme Programming XP...

Planning in XP



- **Releases:**
  - One to six month period
  - Software actually delivered to customer

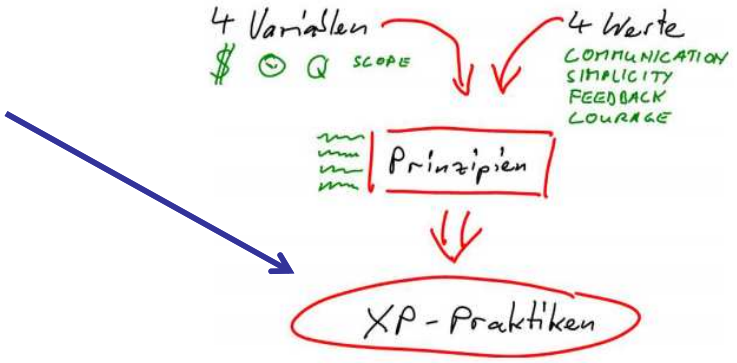
- **Iteration:**
  - One to three weeks period
  - Produces an intermediate version for the next release

- **Story:**
  - Closely linked to system features
  - One or several stories to be realized in one iteration

- XP suggests to plan ahead for one to two steps at most on all three levels

1.2.4 Extreme Programming XP...

XP Praktiken (XP Practices)

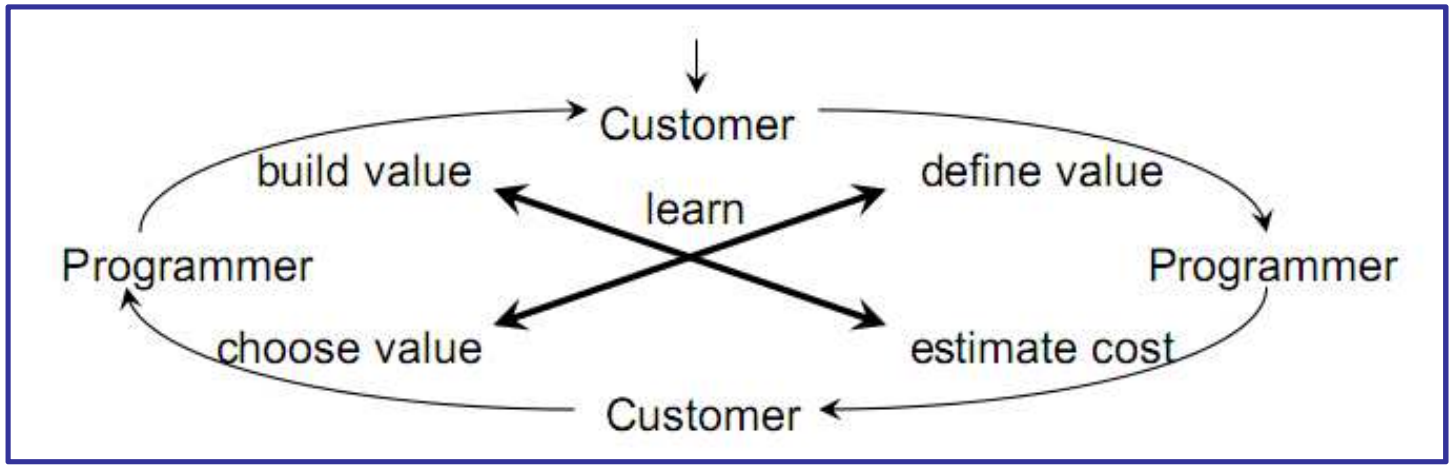


- The Planning Game (1)
- Small Releases (2)
- Metaphor (3)
- Einfaches Design (4)
- Testing (5)
- Refactoring (6)
- Pair-Programming (7)
- Collective Code Ownership (8)
- Continuous Integration (9)
- 40 Hour Week (10)
- On-Site Customer (11)
- Coding Standards (12)

1.2.4 Extreme Programming XP...

The Planning Game (1)

- XP develops the system in an evolutionary way, so the team has to plan what the features of the next iteration will be.
- This is a game between two parties, leading to a balanced solution:
  - Business people (customer): What are the valuable features? What are the priorities? When are the features needed?
  - Technical people (programmer): How expensive is a feature to implement? What are the consequences? What is a realistic schedule?



1.2.4 Extreme Programming XP...

Small Releases (2)

- Release = Software version handed over to customer

- Every release should be as small as possible, i.e. contain a small change to the previous release
  - Containing the most valuable business requirements
- Every release has to make sense as a whole
- Reduce release cycle:
  - One month or two (if possible)
- Potential problems:
  - Usually, the small releases will not go into productive use (to avoid instabilities, additional user training etc.)

– Therefore, specific quality assessment by customer is expected
  » Is this realistic?

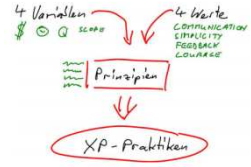
1.2.4 Extreme Programming XP...

Metaphor (3)

- Single overarching metaphor
  - – Naive: Terms mirroring the real world, e.g. contract, customer, ...
  - – Less naive: e.g. pension calculation as a spreadsheet with rows and columns
- "The metaphor in XP replaces much of what other people call 'architecture'."
- In practice more similar to a domain model as done in system analysis.
- Potential problems:
  - – Metaphor may become too complex to be helpful for large systems.
- Consequence:
  - – UML Class diagram of the problem domain and its metaphor may be helpful (see later)

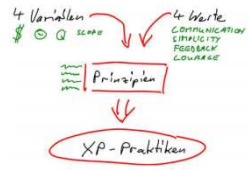
1.2.4 Extreme Programming XP...

Metaphor Formula 1



1.2.4 Extreme Programming XP...

Simple Design (4)



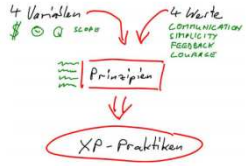
- "The right design for the software at any given time is the one that
  1. Runs all the tests
  2. Has no duplicated logic
  3. States every intention important to the programmers
  4. Has the fewest possible classes and methods."

"...And that, in simple terms, is what's wrong with your software design."

- XP mantras: "The simplest thing that could possibly work."
  "You ain't going to need it. (YAGNI)"
  – Erase (or better do not add in the first place) every-thing unnecessary
- Software design seen as a communication medium
  – Very similar to a traditional design specification...
- Design is represented in code
  – There is no separate documentation
  – Graphical sketches (e.g. UML) only used for short digression (essentially for finding the right questions)

1.2.4 Extreme Programming XP...

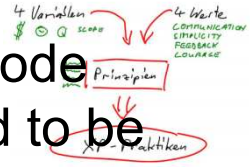
Testing (5)



- Test-First approach:
  - Tests are written before the program
  - Tests are used to clarify the usage of an interface, to define the expected effect, to single out problematic special cases, ...
  - Tests are the XP replacement for traditional software specification
- Automated tests:
  - Tests are kept in an executable infrastructure and can be run at any time
  - Tests give feedback and confidence to the programmer
  - “Test infected: Programmers love testing” (E. Gamma about the xUnit testing framework)
- Programmer-written unit tests: Must always run to 100%
- Customer-written functional tests based on “stories”: May run only partially for some time

1.2.4 Extreme Programming XP...

Refactoring (6)



- Adding new features in an arbitrary way will lead to ill-structured code
- When adding a new feature, the structure of the system may need to be adapted (refactored)
  - Refactoring may remove code, introduce new code but keeps the functionality unchanged (all tests run 100%)
  - Simple steps like combining parameters of a method into a data structure
  - Complex steps like applying design patterns
- This is done only when necessary to keep the solution simple:
  - Main reason for refactoring: To avoid duplication of logic
  - Possibly other reason: Smaller and more elegant design after introduction of new features
- How can we be sure not to destroy working functionality by refactoring?
  - Use automated tests
  - Refactor first, run the tests, then add the feature, run the enhanced tests

1.2.4 Extreme Programming XP...

Pair-Programming (7)

"*All production code is written with two people looking at one machine, with one keyboard and one mouse*." (K. Beck: Extreme Programming Explained - Embrace Change, Addison-Wesley, 2000, p. 58)

- **Two roles:**
  - Keyboard/mouse owner: Thinks tactically about the best way to implement the method under development
  - Observer: Thinks strategically about the overall approach and simplification
  - Roles in the pair may be switched after some time
- **Dynamic pairing:**
  - Pair partners should change frequently
  - Changes may take place even several times a day
- **Potential problems:**
  - Complex programming tasks are often better solved by a single person "meditating" over the solution
  - Some people simply do not like the pair situation

1.2.4 Extreme Programming XP...

Collective Code Ownership (8)

- Through pair programming, any piece of code has many authors
  - Side effect: Removes too complex code
- Everybody in the team has the right to add value to any portion of the code at any time.
- If somebody does not know some part of the system well, he/she should pair up with an expert on this part
- Practical tool for co-ordination: Stand-Up Meetings
  - Regular meetings in the morning of each day
  - Everybody has to stand in a circle (to keep the meeting short)
  - In turn, each team member reports what has been done yesterday and what the plans are for today
- Potential problems:
  - Collective ownership means no individual responsibility
  - Collectivism partially contradicts to human nature

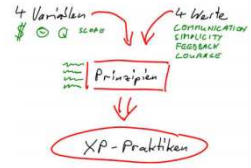
1.2.4 Extreme Programming XP...

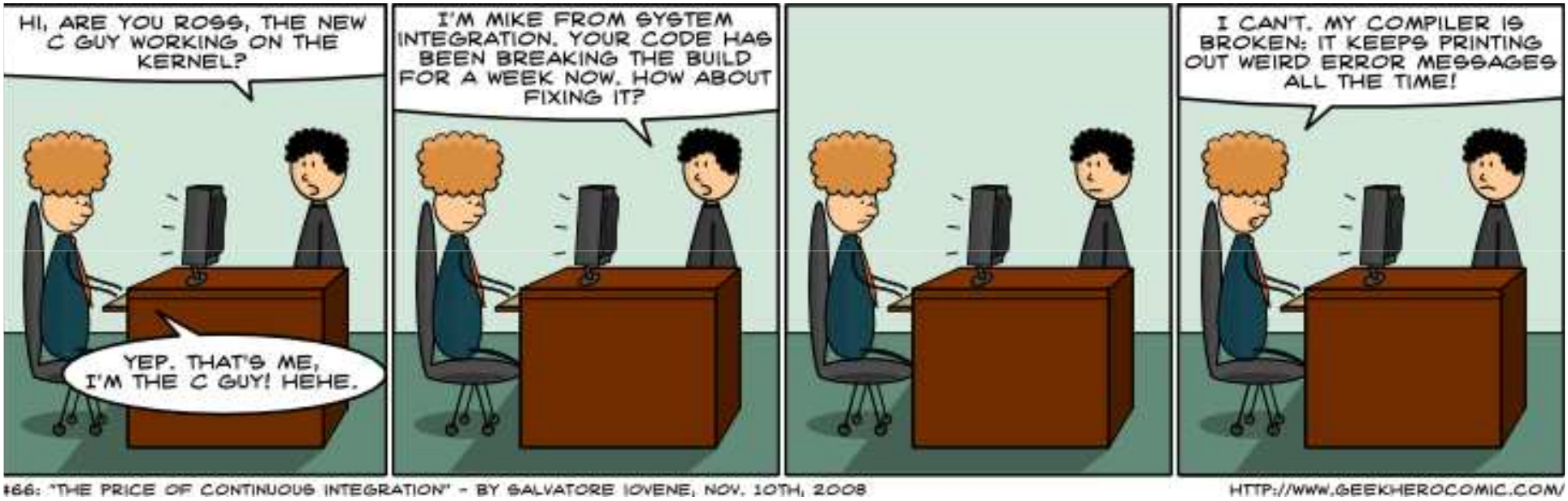
Continuous Integration (9)

- There is always an up-to-date running version of the full system.
- Integration is not a late development stage but done on a daily basis.
- Once a day, newly developed code is integrated into the common code basis
- Separate “integration machine”
  - Can be used only by one developer pair at a time
  - Keeps the current version of the integrated system
  - Programmers
    - » load new/modified code
    - » check for collisions
    - » run all tests, fixes problems, ... until all tests run 100%

1.2.4 Extreme Programming XP...

Continuous Integration (9) – Example ☺





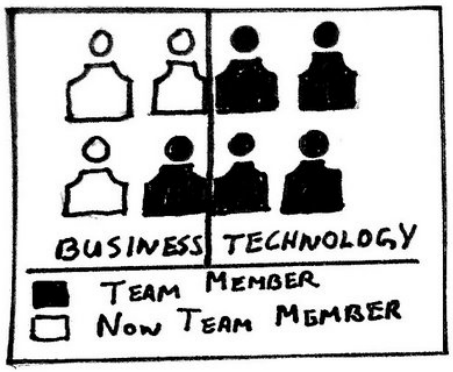
1.2.4 Extreme Programming XP...

40 Hour Week (10)

- Overtime considered a symptom for a serious problem of the project
- XP-rule: You can't work a second week of overtime.
- Basic idea: People should not get too exhausted
- Potential problems:
  - Customers may have the feeling the team does not work hard enough
  - Fully unrealistic when delays occur and important deadlines are approaching
  - The on-site customer is a single point of failure who has low chances for a 40-hour week

1.2.4 Extreme Programming XP...

On-Site Customer (11)



• "*A real customer must sit with the team, available to answer questions, resolve disputes, and set small-scale priorities. By "real customer" I mean someone who will really use the system when it is in production.*" (K. Beck: Extreme Programming Explained - Embrace Change, Addison-Wesley, 2000, p. 60)

- "Whole team": There may be several representatives of the customer
- Only representative of the customer required, no longer a "real customer"
- Potential problems:
  - Definitely the weakest point of the XP approach.
  - On-site customer tends to get overloaded, since he/she has the final responsibility for too many things
  - Customer organisations tend to assign the role to inexperienced people
  - On-site customer gets mentally separated from his organization and may feel too much as team member to be effective.

1.2.4 Extreme Programming XP...

Coding Standards (12)

- To enable collective ownership, coding standards for the team are necessary
  - Naming conventions
  - Conventions for embedded documentation
  - Code layout conventions
  - Framework for automated tests

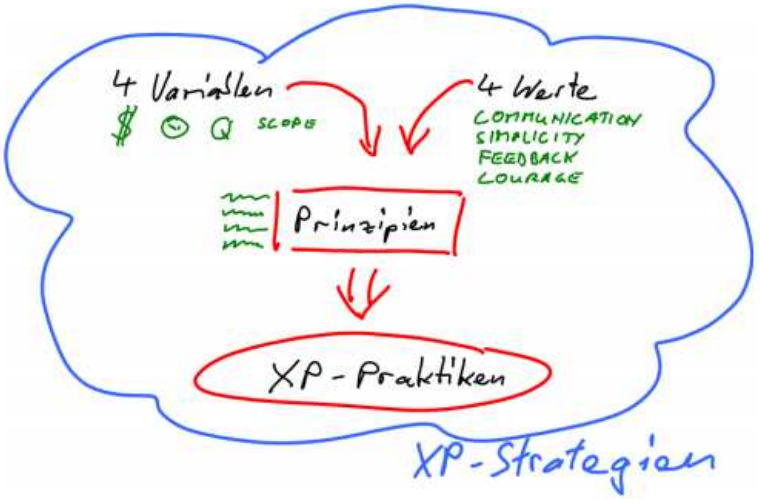
- These standards need to be discussed thoroughly, so that everybody accepts them without resistance

- Fortunately, for modern languages “style guides” often exist already.

1.2.4 Extreme Programming XP...

XP Management Strategie

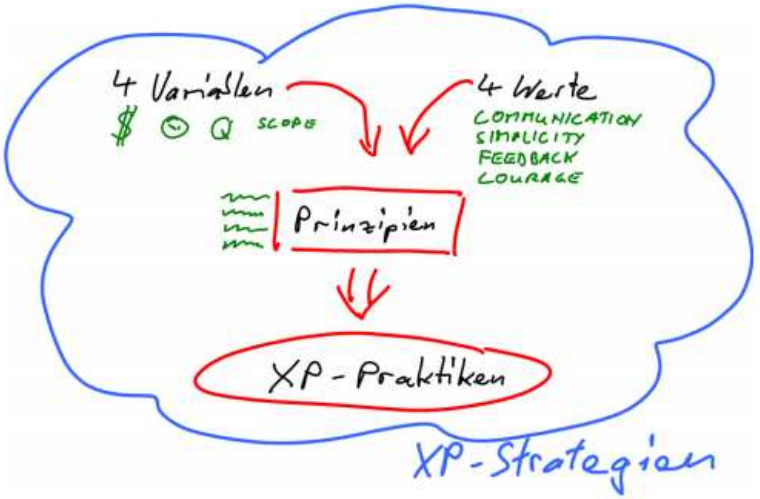
- Accepted responsibility
- Quality work
- Local adaption
- Travel light
- Honest measurement



1.2.4 Extreme Programming XP...

XP Planning Strategie

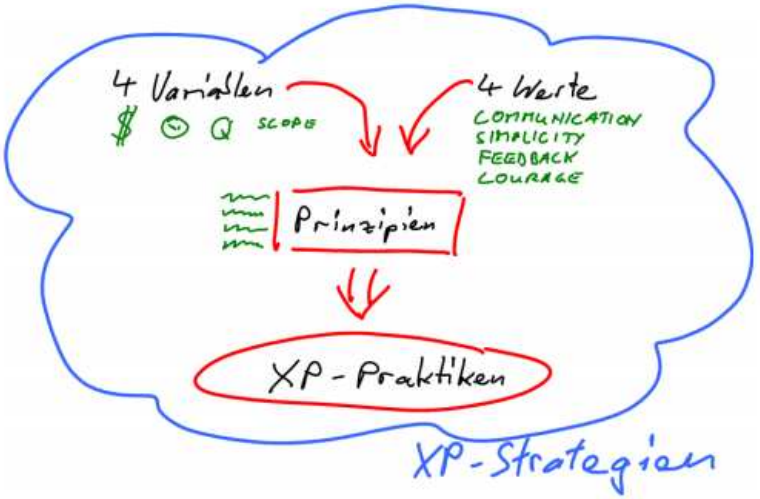
- Team zusammenbringen
- Umfang und Prioritäten wählen
- Kosten und Zeiten
- Vertrauen schaffen
- Mittel für kontinuierliches Feedback bereitstellen



1.2.4 Extreme Programming XP...

XP Entwicklungsstrategie

- Kontinuierliche Integration
- Collective Code Ownership
- Pair-Programming



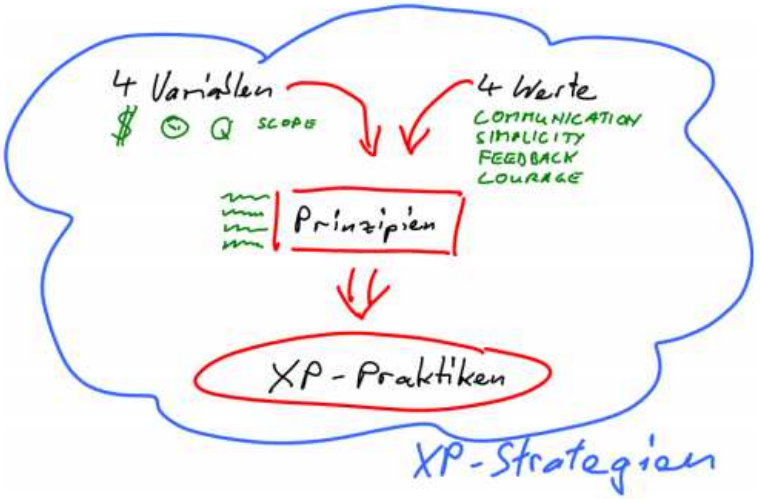
XP Designstrategie

- 1. Schreibe einen Test, damit Du weißt, wann Du fertig bist.
- 2. Designe und implementiere nur so viel, dass der Test läuft.
- 3. Wiederhole 1 und 2
- 4. Falls Du eine Möglichkeit siehst, das Design zu vereinfachen, mach es.

1.2.4 Extreme Programming XP...

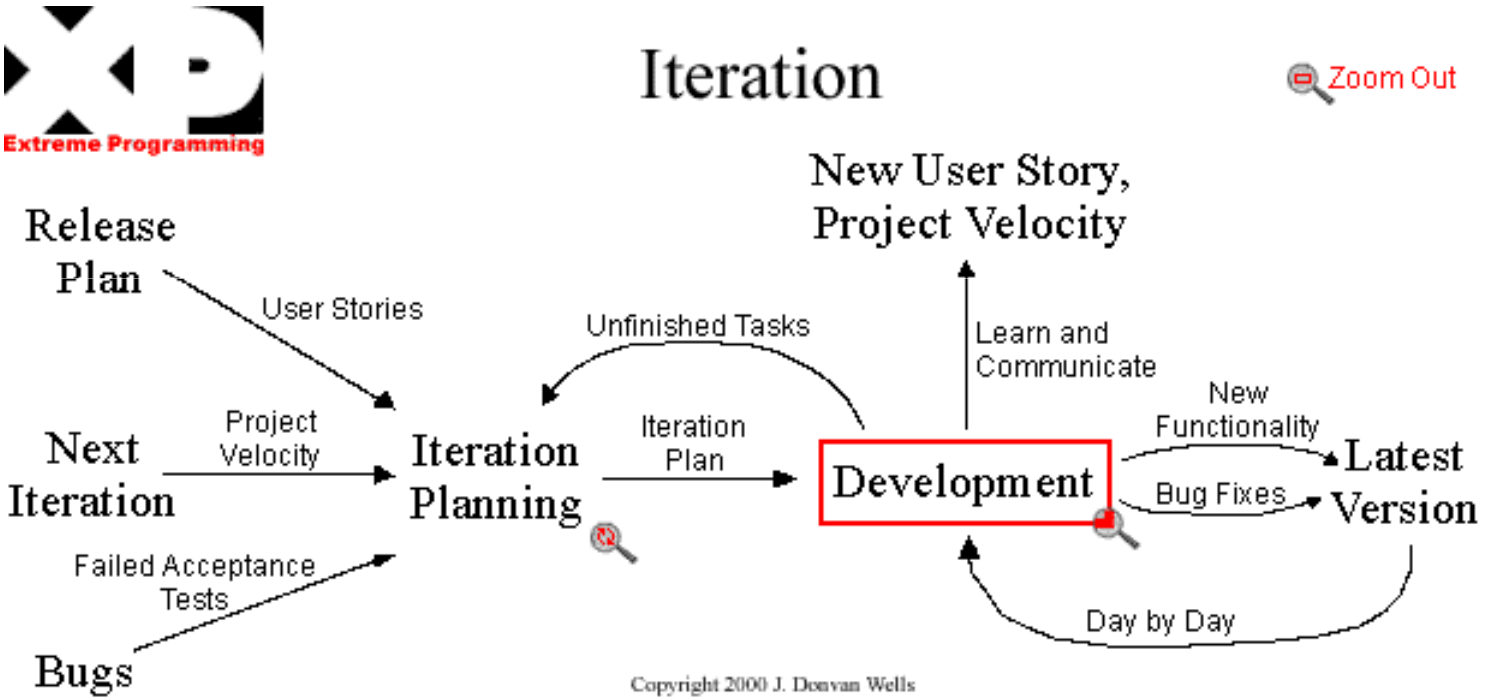
XP Teststrategie

- Automatisches Testen
- Kontrolle
- Modultests von Entwicklern
- Funktionstests von Kunden
- (Ev. Stress- oder Überlasttests)



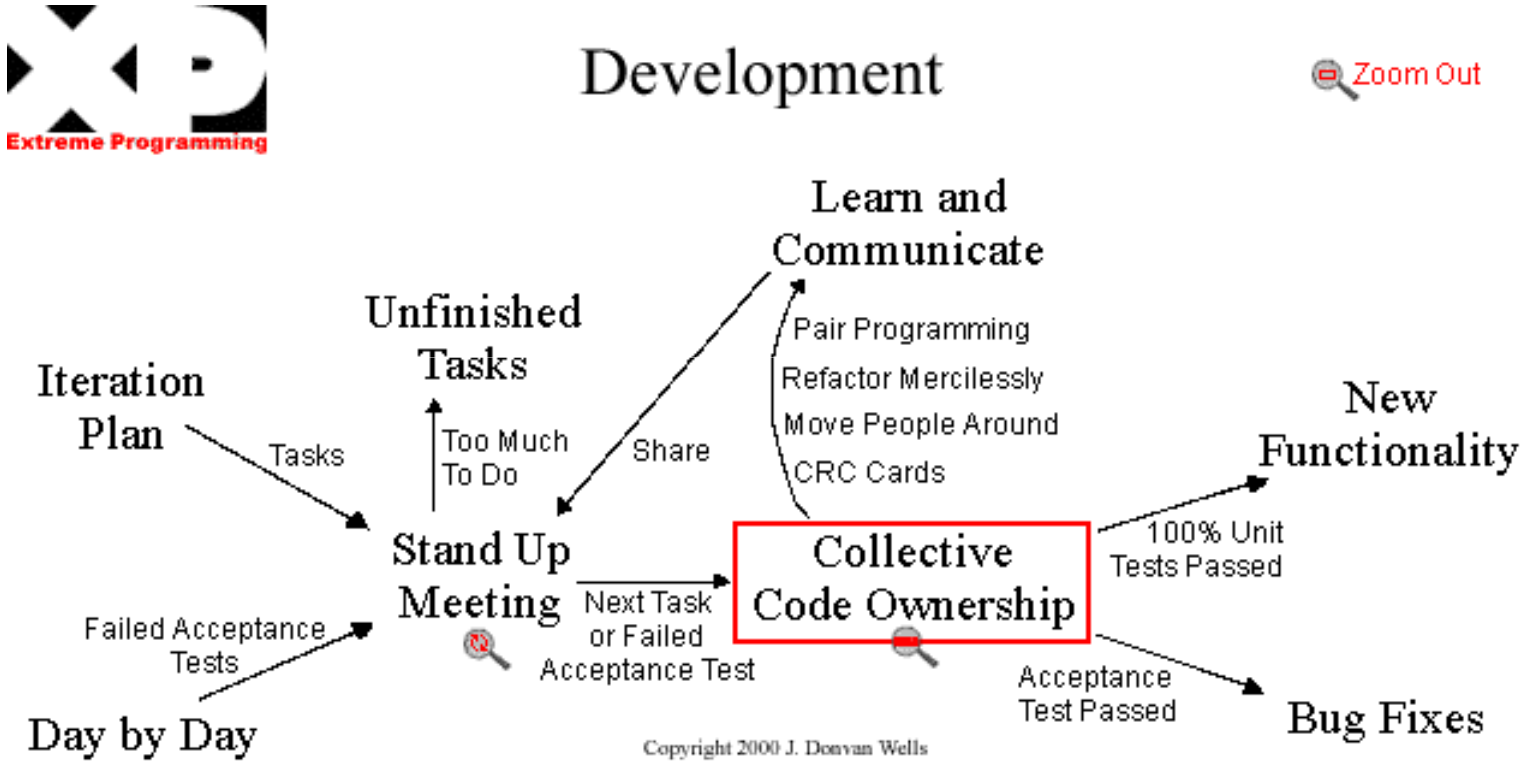
1.2.4 Extreme Programming XP...

Example: One Iteration of XP



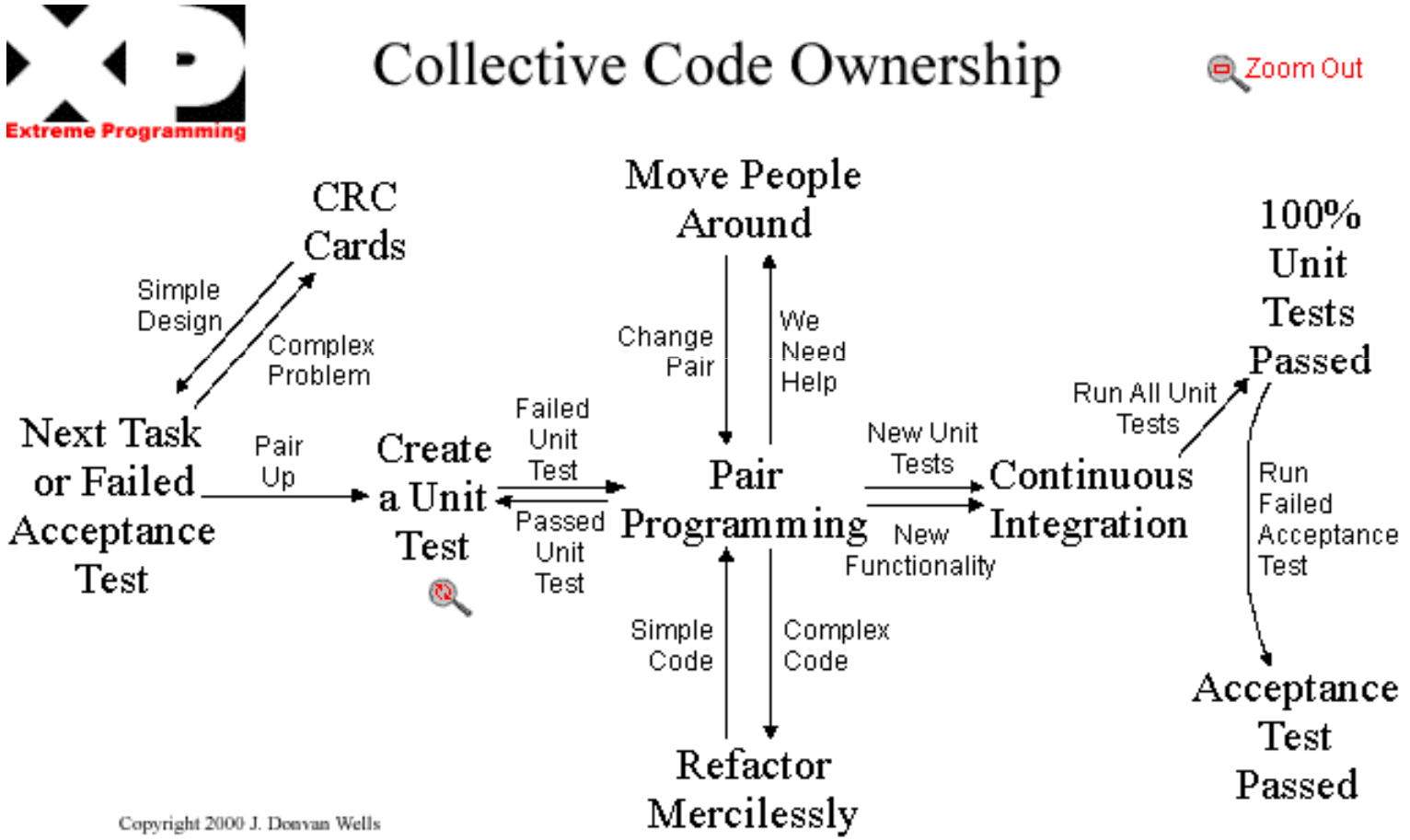
1.2.4 Extreme Programming XP...

Example: Development Overview



1.2.4 Extreme Programming XP...

Example: Programmers activities and work Style



1.3 Traditionelle Methoden zur Programmentwicklung

- Ab 1970: erste Ansätze zur systematischen SW-Entwicklung:
  - strukturierte Programmierung
  - schrittweise Verfeinerung und Top-down Entwurf

Strukturierte Programmierung

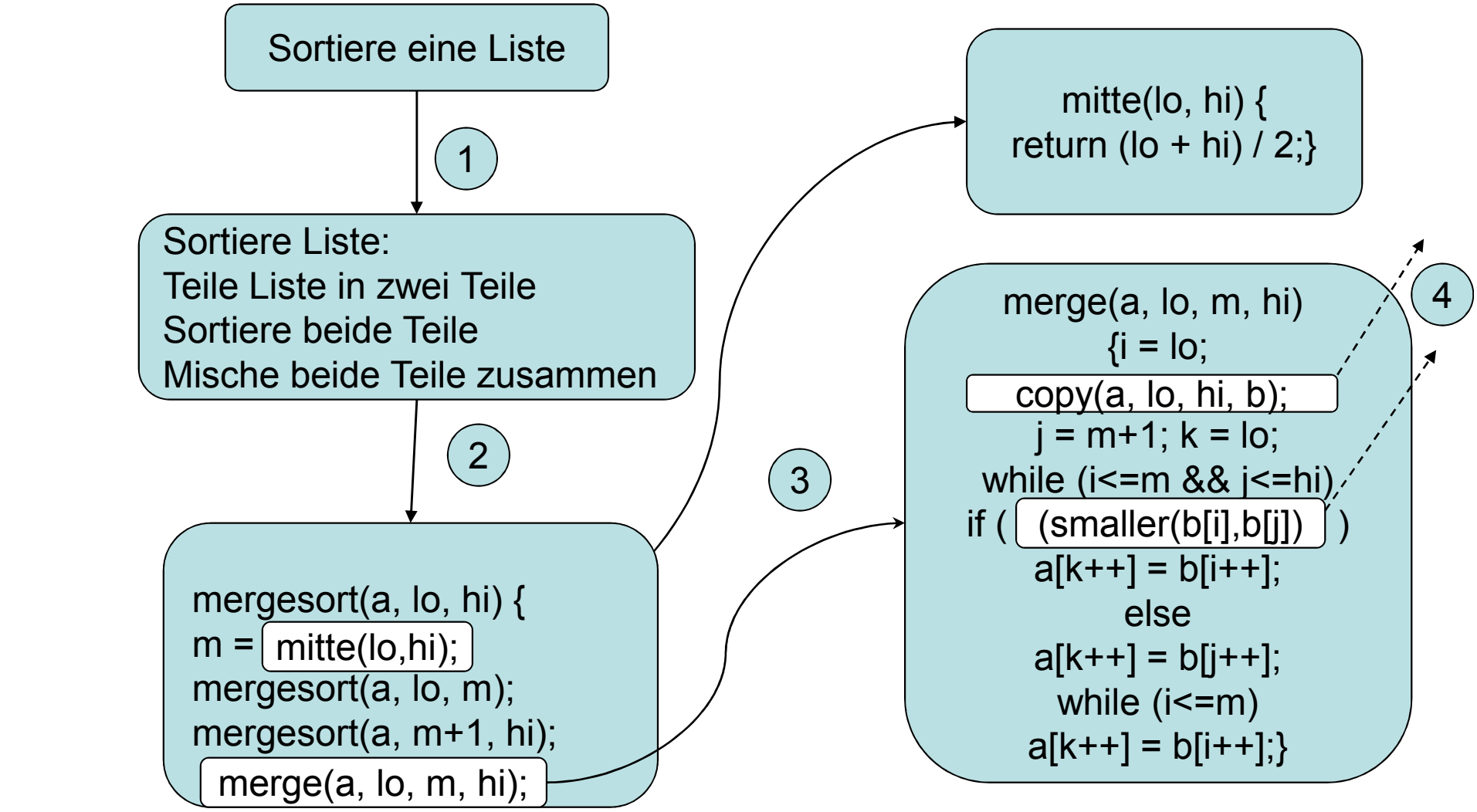
- Ziel: bessere Lesbarkeit und Korrektheit von SW
- Einhaltung von Regeln wie z.B.:
  - klare Ablaufstruktur des Programms (keine goto-Befehle)
  - klare Gliederung, z.B. durch Funktionen / Prozeduren
  - klar aufgebaute Datenstrukturen
  - selbsterklärende Bezeichner

- Durch moderne Programmiersprachen unterstützt, i.w. aber Frage des Programmierstils

Schrittweise Verfeinerung und Top-down Entwurf

- Strategie des *divide-et-impera*:
  - löse aus großem Problembereich kleinere Teilprobleme heraus und bearbeite diese unabhängig voneinander
  - rekursive Anwendung dieses Prinzips
    - führt zur weiteren Verfeinerung von Programmteilen
- Systementwurf erfolgt dabei vom Ganzen zu den Teilen hin
- Im Vordergrund steht die **funktionale** Untergliederung des Programmsystems:
  - komplexe Funktionen werden in Teilfunktionen zerlegt
- Problem: zur Funktionsstruktur ”querlaufende“ Datenstrukturen
  - gemeinsame Strukturierung von Funktionen und Daten nötig

1.3 Traditionelle Methoden zur Programmentwicklung ...



1.4 Daten- und funktionsorientierte Methoden

Geheimnisprinzip

- Schwierigkeit bei der Programmentwicklung:
  - Kommunikationsproblem der beteiligten Entwickler
  - ein Entwickler sollte nicht mit Detailwissen über die Module anderer Entwickler überfrachtet werden
- Lösung: Geheimnisprinzip
- Jedes Software-Modul besteht aus zwei Teilen:
  - Vereinbarungen, die für die Nutzung des Moduls durch andere notwendig sind (Spezifikation, Schnittstelle)
  - Vereinbarungen und Anweisungen, die für die Nutzung nicht notwendig sind (Konstruktion, Implementierung)
- Nur die Schnittstelle ist nach außen sichtbar (öffentlich)

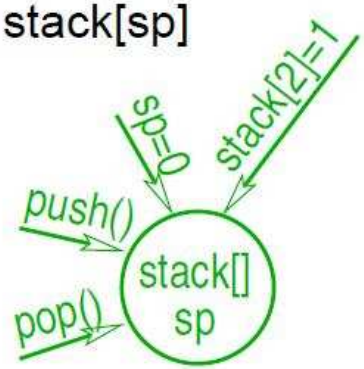
Datenabstraktion (Datenkapselung)

- Daten und darauf operierende Funktionen (Operationen) müssen immer gemeinsam definiert werden
- Datenstrukturen sind so in Module zu kapseln, daß der Zugriff von außen nur über definierte Operationen möglich ist
- D.h. Schnittstelle des Moduls besteht nur aus den Operationen
  - Aufbau der Datenstruktur und Programmierung der konkreten Zugriffe bleibt verborgen
  - damit: Module werden unabhängiger voneinander
    - z.B.: Datenstruktur kann ohne Einfluß auf andere Module verändert werden
- (Grundlegendes Prinzip auch der objektorientierten Programmierung)

Beispiel: Keller (vgl. EI_I)

Ohne Datenkapselung

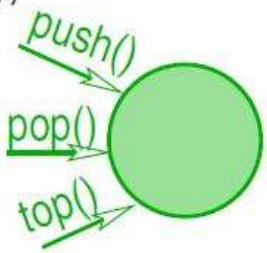
```
// Array mit Kellerelemente
int stack[100];
// Oberstes Element ist stack[sp]
int sp = −1;

// Operationen
void push(int i);
int pop();
```



Nutzer des Kellers sieht die Realisierung. Änderung der Implementierung ist nicht möglich (z.B. als Liste)

Mit Datenkapselung

```
// Abstrakter Datentyp
class Stack {
public:
    // Operationen
    void push(int i);
    int pop();
    int top();
};
```



Nutzer sieht nur die Schnitt−stelle, nicht die interne Realisierung.

Entity/Relationship-Modellierung (E/R)

- Ursprünglich: Entwurfstechnik für Datenbanken
- Ziel: **Datenmodellierung**
  - Modellierung der strukturellen Zusammenhänge der zu bearbeitenden Daten
    - Entitäten: Gegenstände der realen Welt, beschrieben durch Eigenschaften
    - Relationen: Beziehungen zwischen den Gegenständen
  - abstrahiert zunächst von funktionalen Zusammenhängen
    - diese müssen zusätzlich in anderer Form beschrieben werden
- Vorläufer der objektorientierten Modellierung

1.5 Objektorientierte Entwicklungsmethoden

1.5.1 Prinzipien der Objektorientierung

- Wirklichkeit wird als Menge interagierender **Objekte** modelliert
  - dies bedeutet eine Abstraktion der Wirklichkeit

- Ein Objekt
  - hat bestimmte **Eigenschaften** (**Attribute**) / einen **Zustand**,
  - reagiert mit einem bestimmten **Verhalten** auf die Umgebung,
  - steht in bestimmten **Beziehungen** zu anderen Objekten.

- Objekte werden zu **Klassen** zusammengefaßt
  - Abstraktion von der konkreten Ausprägung eines Objekts
  - Objekte mit gleichen Attributen und gleichem Verhalten werden gemeinsam betrachtet
    - Klasse ist Stellvertreter / Bauplan für diese Objekte

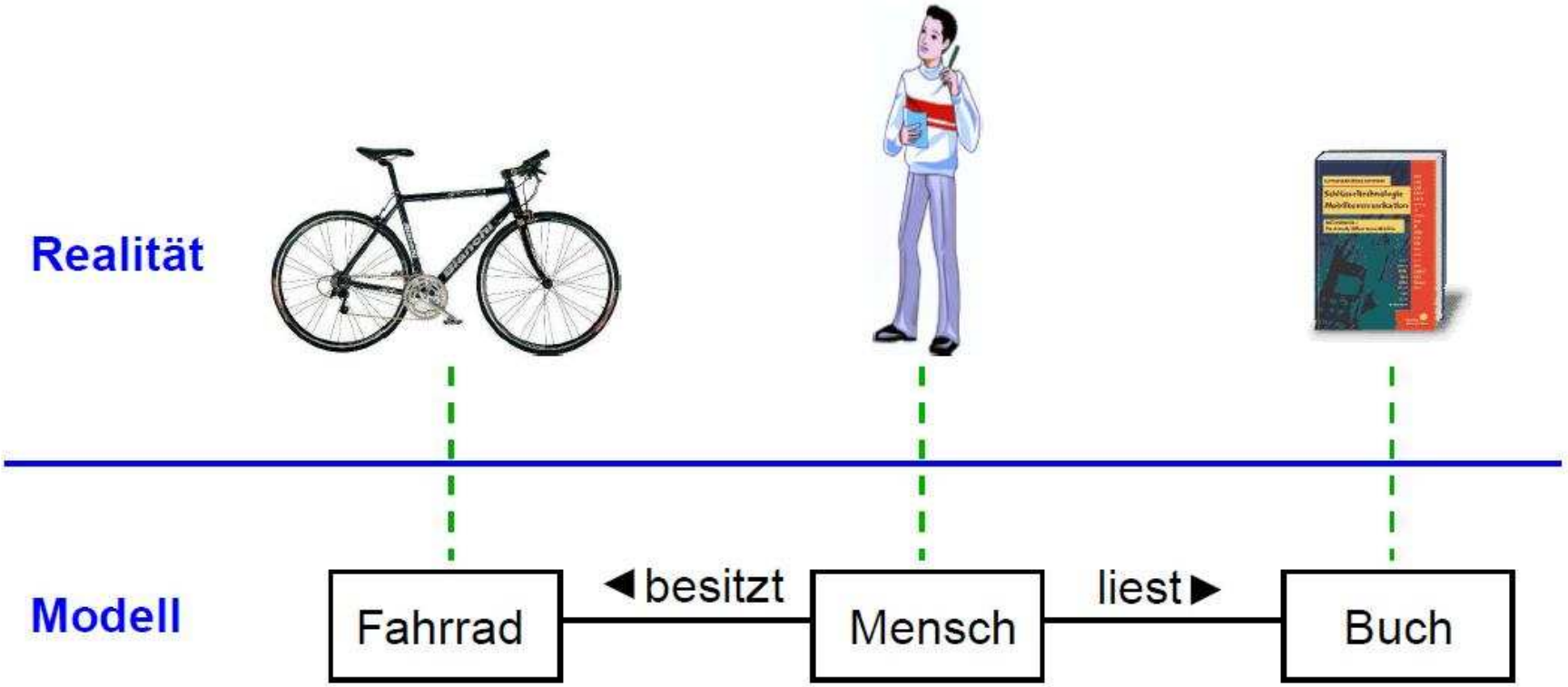
1.5.1 Prinzipien der Objektorientierung ...

”**Gleiche Attribute**“ bedeutet, daß die Objekte alle eine bestimmte Eigenschaft (z.B. Farbe, Alter, Telefonnummer, ...) besitzen, aber nicht, daß der Wert dieser Eigenschaft gleich sein muß (z.B. können Objekte einer Klasse unterschiedliche Farben haben, solange sie alle die Eigenschaft ”Farbe“ besitzen).

”**Gleiches Verhalten**“ bedeutet, daß sich das Verhalten identisch beschreiben läßt. Auch das bedeutet nicht, daß sich die Objekte immer gleich verhalten müssen, da das Verhalten durchaus auch vom Zustand des Objekts anhängen kann.

1.5.1 Prinzipien der Objektorientierung ...

Darstellung der Realität durch ein abstraktes Modell

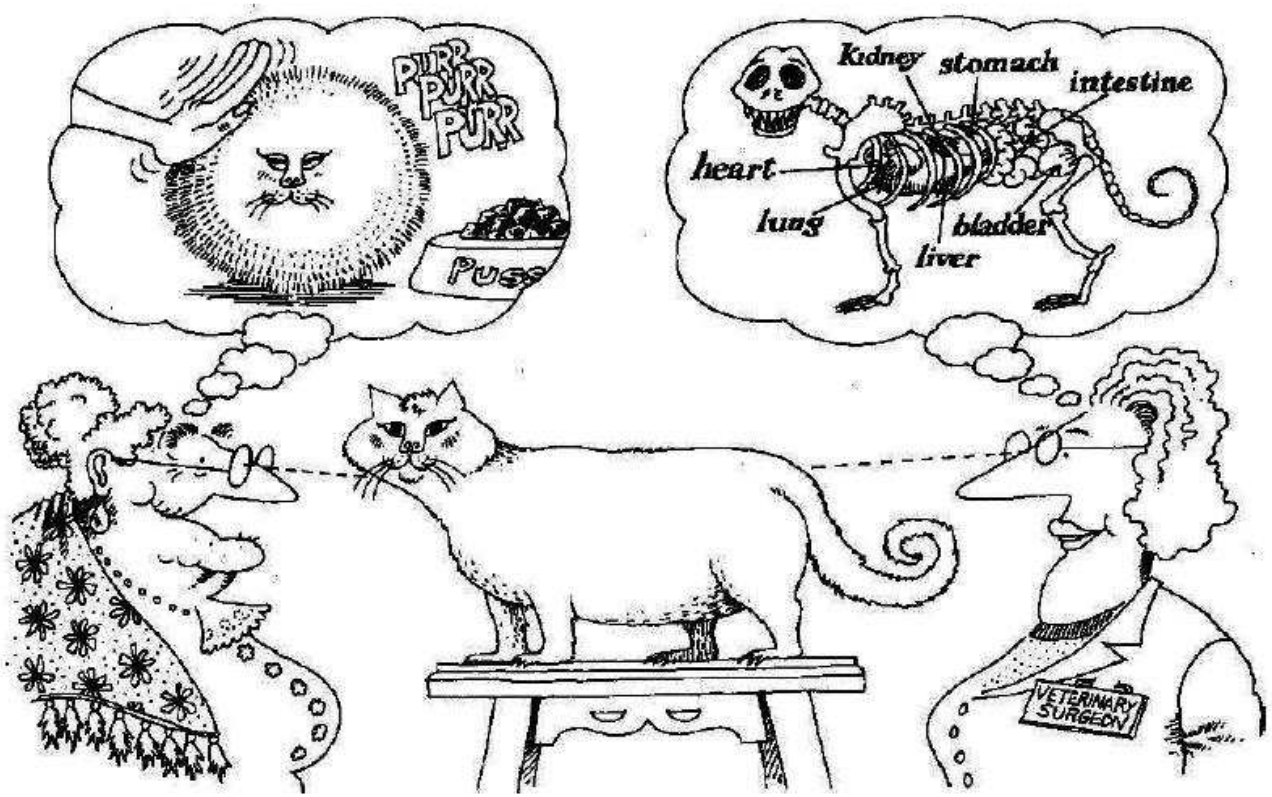


➢Reale Objekte werden durch Objekte und Klassen des Modells repräsentiert

1.5.1 Prinzipien der Objektorientierung ...

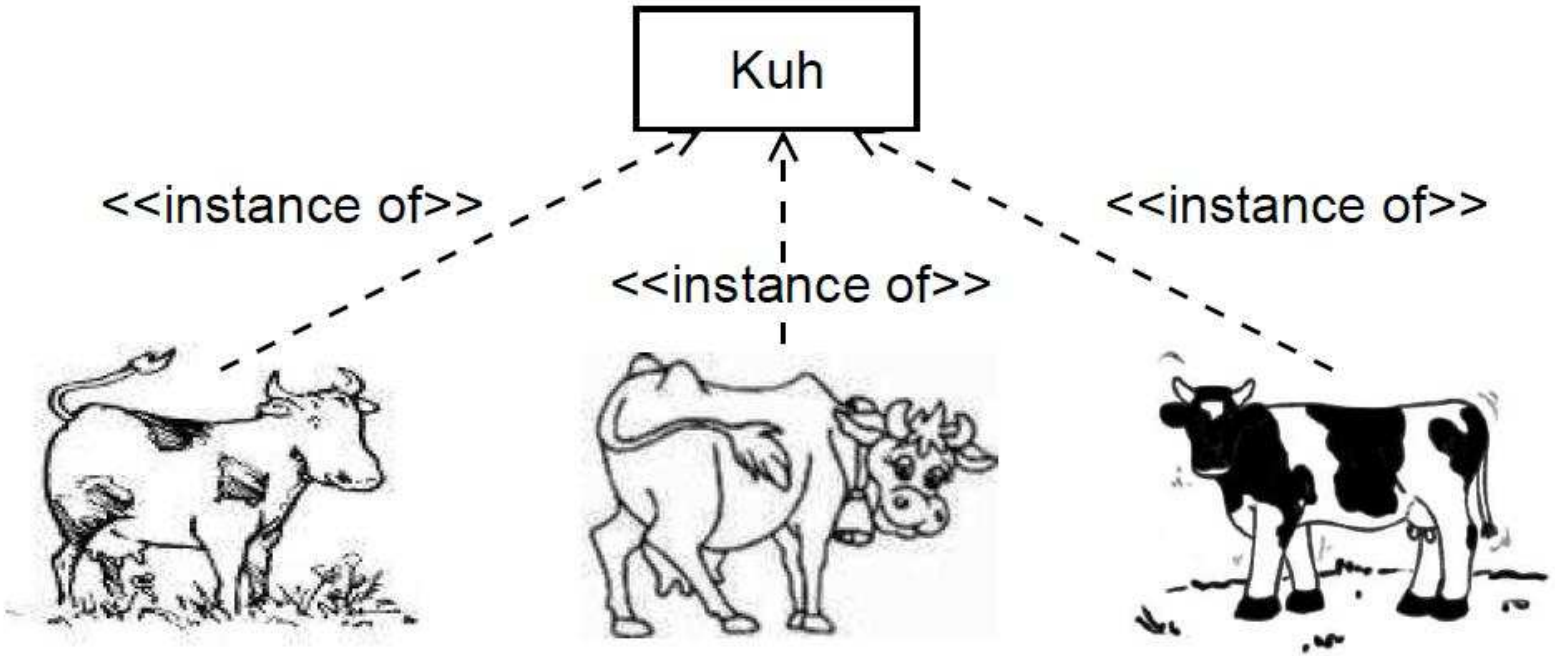
Darstellung der Realität durch ein abstraktes Modell ...

- Das Modell konzentriert sich auf die wesentlichen Merkmale eines Gegenstands (abhängig vom Betrachter!)



1.5.1 Prinzipien der Objektorientierung ...

Objekte und Klassen



➢Abstraktion: wir betrachten zusammenfassende Klassen statt individueller, gleichartiger Objekte **(Instanzen, Exemplare)**

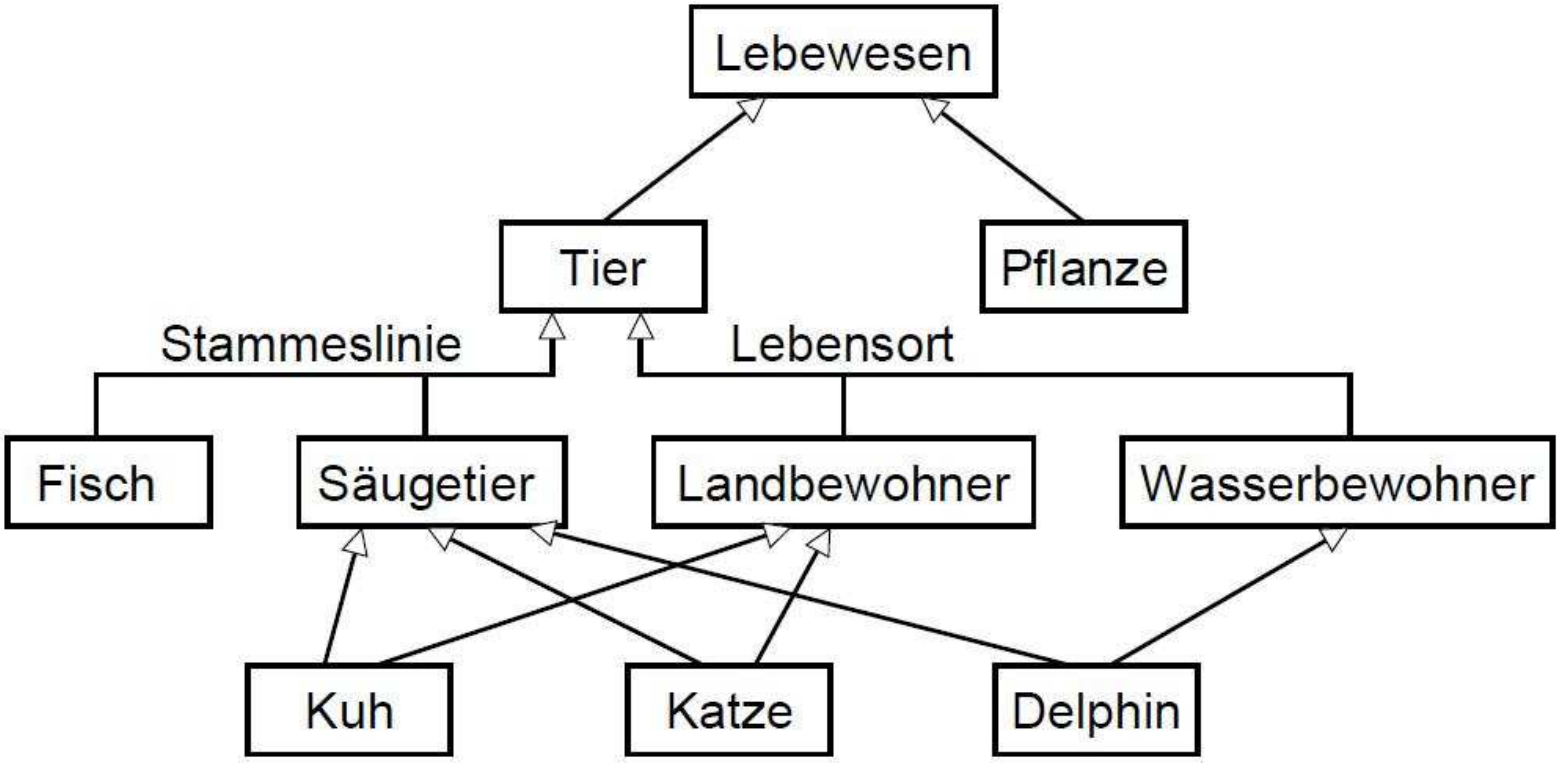
1.5.1 Prinzipien der Objektorientierung ...

Generalisierung (Oberbegriff-Beziehung)

- Zusammenfassung mehrerer Arten / Varianten von Objekten (eigentlich: Klassen!) unter einem Oberbegriff
  - bekannte Abstraktion aus dem täglichen Leben:
    - man sagt: ”Da fahren drei Autos“ statt” Da fahren ein Cabriolet, ein Kombi und eine Limousine“
  - spezielle Beziehung zwischen Klassen: ”ist ein“
    - ein Cabriolet ist ein Auto
    - ein Auto ist ein Fahrzeug
- Generalisierung erlaubt den Aufbau komplexer Taxonomien
- Die Unterstützung des Abstraktionsmittels ”Generalisierung“ ist charakterisierend für objektorientierte Methoden

1.5.1 Prinzipien der Objektorientierung ...

Beispiel einer Taxonomie (Klassenhierarchie)



1.5 Objektorientierte Entwicklungsmethoden ...

1.5.2 Einsatz der Objektorientierung im Entwicklungszyklus

- Beim fachlichen Entwurf
  - objektorientierte Analyse (OOA)
- Beim software-technischen Entwurf
  - objektorientierter Entwurf (OOD)
- Bei der Programmierung
  - objektorientierte Programmierung (OOP)
- Vorteil der objektorientierten Entwurfsmethode:
  - durchgängige Methodik ohne Paradigmenwechsel zwischen den Entwicklungsphasen
- (Aufbau der Vorlesung orientiert sich an obigen Phasen)

1.5.3 Objektorientierte Analyse (OOA)

- Ziel: Verstehen des Anwendungsproblems und Beschreibung durch ein objektorientiertes Modell
  - OOA ist im Kern die Modellierung der realen Welt zur Simulation im Computer
  - Modell-Objekte sind reale Objekte, aber auch Begriffe oder Ereignisse aus dem Anwendungsbereich
  - wichtig: geeignete Abstraktion für das Modell
- Noch nicht betrachtet: Fragen der Implementierung
  - z.B. Speicherung oder Darstellung der Objekte
- Häufiges Hilfsmittel zur Kommunikation mit dem Auftraggeber:
  - Prototyp der Bedienoberfläche, aus OOA-Modell abgeleitet

1.5.3 Objektorientierte Analyse (OOA) ...

Bestandteile des OOA-Modells

- **Statisches Modell**: Datenmodell
  - Klassen und Beziehungen, insbes. Generalisierung
  - Attribute der Klassen
  - Aufteilung in Pakete (Teilsysteme)

- **Dynamisches Modell**: Funktionsabläufe
  - *Use Cases*: Beschreibung der durchzuführenden Aufgaben auf sehr hohem Abstraktionsniveau
  - Spezifikation der globalen Abläufe von Aktivitäten
    - detaillierte Analyse der *Use Cases*
  - Ablauf der Kommunikation zwischen den Objekten
  - Spezifikation des lokalen Verhaltens von Objekten (Zustandsübergänge)

1.5.4 Objektorientierter Entwurf (OOD)

- Im OOA-Modell bleibt die technische Realisierung noch völlig unberücksichtigt
- Das OOD-Modell ist bereits nahe an der Implementierung, aber noch auf höherem Abstraktionsniveau
  - Effizienz und Wiederverwendung werden berücksichtigt
  - Modell ist Abbild der späteren Programmstruktur
    - jede Klasse, Attribut etc. kommt auch im Programm vor (mit gleichem Namen)
    - Klassen können direkt implementiert werden
- Das OOD-Modell besteht wie das OOA-Modell aus statischem und dynamischem Modell