Achtung:
Deutsch/Englisch-Gemisch

Freie Universität Berlin

Vorlesung "Softwaretechnik"
Buchkapitel 16
Software-Prozessmodelle
(d.h. Softwareprozess-Modelle)

Stephan Salinger
(Foliensatz L. Prechelt, B. Brügge, A. Dutoit, K. Schneider)
Freie Universität Berlin, Institut für Informatik
http://www.inf.fu-berlin.de/inst/ag-se/

- Elemente von Prozessmodellen
  - Rollen, Artefakte, Aktivitäten
- Wasserfallmodell
- Reparatur 1: Iteration
  - Prototypmodell, Evolutionäre Modelle
- Reparatur 2: Flexiblere Planung
  - Agile Methoden
- Prozessmodell-Auswahlkriterien
- Anpassbare Prozessmodelle
  - Rational Unified Process
  - V-Modell XT
- Was bedeutet Agile Methode?
  - Beispiel: eXtreme Programming (XP)

Wo sind wir?: Taxonomie "Die Welt der Softwaretechnik"

Freie Universität Berlin

Welt der Problemstellungen:

- Produkt (Komplexitätsprob.)
  - Anforderungen (Problemraum)
  - Entwurf (Lösungsraum)
- Prozess (psycho-soziale P.)
  - Kognitive Beschränkungen
  - **Mängel der Urteilskraft**
  - **Kommunikation, Koordination**
  - Gruppendynamik
  - Verborgene Ziele
  - **Fehler**

Welt der Lösungsansätze:

- Technische Ansätze ("hart")
  - Abstraktion
  - Wiederverwendung
  - Automatisierung
- Methodische Ansätze ("weich")
  - Anforderungsermittlung
  - Entwurf
  - **Qualitätssicherung**
  - **Projektmanagement**

Wo sind wir?: Prozessmodelle

- Einsicht: Man sollte die Gesamt-Vorgehensweise nicht in jedem Projekt neu erfinden
  - sondern sich auf vorhandene Erfahrungen abstützen

- Prinzipien:
  - ***Planung und Koordination***: Es senkt das Risiko, wenn alle Beteiligten im Voraus erkennen können, was wann getan werden muss
    - Es ist schwierig, dies im Einzelfall korrekt abzuschätzen
  - ***Korrekturen***: Versuche, den Prozess so zu gestalten, dass die unvermeidlich auftretenden Fehler gut ausgeglichen werden können
  - ***Iteration***: Es senkt das Risiko, wenn das Projekt in kurzen Abständen evaluierbare Versionen der Software hervorbringt

Die wichtigsten Entscheidungen für ein SW-Projekt

Freie Universität Berlin

- Projektziele
- Zeitplan und Budget
- Projektorganisation
- **Verwendetes Prozessmodell**
- Verwendete Technologie, Werkzeuge und Methoden
- Teammitglieder

Definitionen: Prozess, Prozessmodell

Freie Universität Berlin

Software-Prozess:

- Die Abläufe, die in einem Softwareprojekt geschehen
  - entweder *deskriptiv* gemeint
    (also beschreibend, was tatsächlich geschieht)
  - oder *präskriptiv* (also als Vorschrift, wie es abzulaufen hat)
- auch Teile des Gesamtprozesses werden oft Prozess genannt
  - z.B. der Testprozess

Softwareprozess-Modell (Software-Prozessmodell):

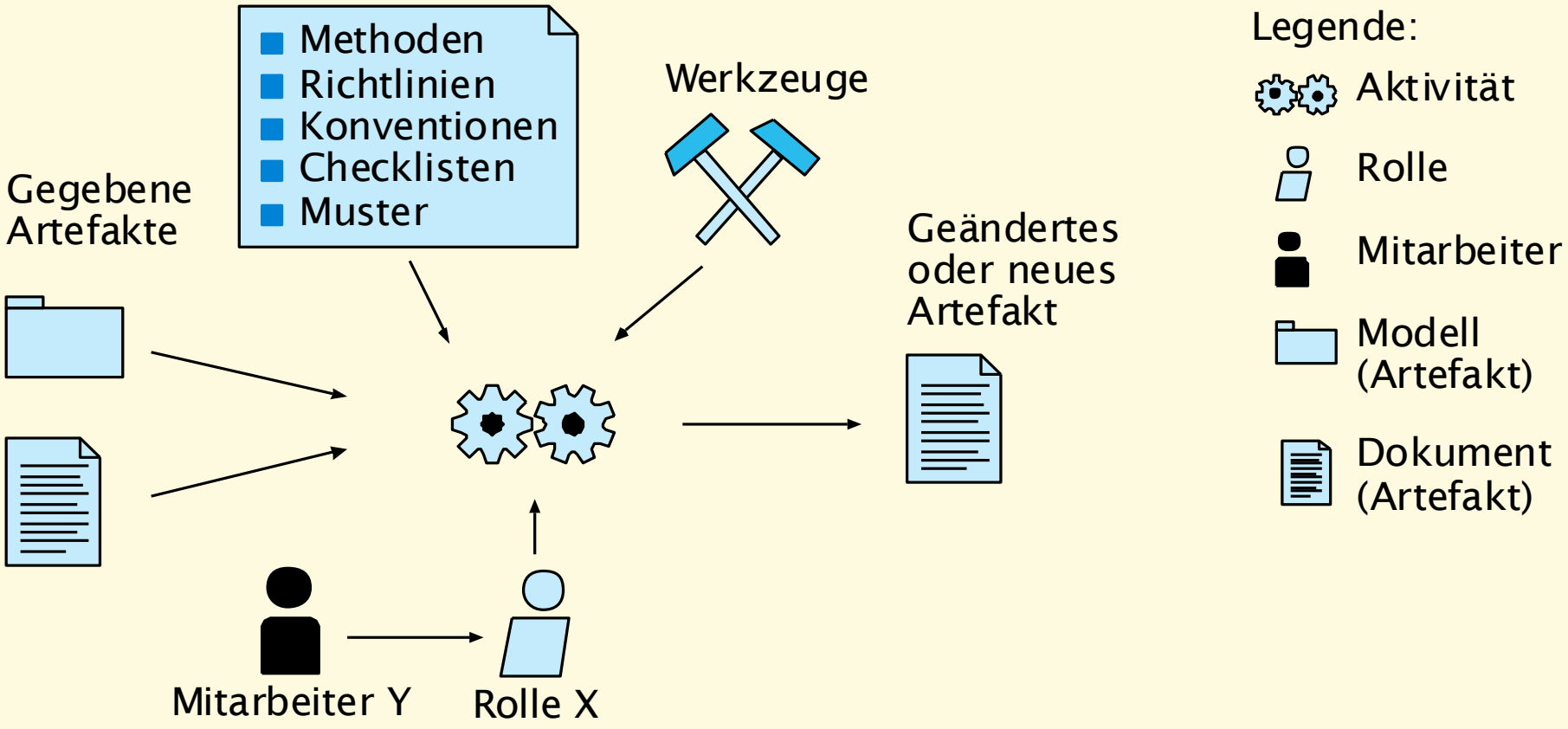
- Eine *Schablone* die die Gemeinsamkeiten der Abläufe in vielen ganz verschiedenen Projekten erfasst
  - meistens zumindest halbwegs präskriptiv gemeint

Definitionen: Rollen, Aktivitäten, Artefakte

Prozesse werden beschrieben mittels folgender Hauptkonzepte:

- **Rolle:**
  - Abstraktion der Aufgabe einer Person in einer bestimmten Situation
  - z.B. "Entwerferin", "Entwickler", "Tester", "Projektleiterin" usw.
    - die selbe Person hat in einem Projekt oft mehrere Rollen
- **Aktivität:**
  - Abstraktion für eine Sorte zielgerichteten Handelns in einem Projekt
  - z.B. "Anforderungsvalidierung", "Architekturentwurf", "Modulentwurf", "Kodieren eines Moduls", "Test eines Moduls" usw.
- **Artefakt:**
  - Abstraktion für eine Sorte von Arbeitsergebnis einer Aktivität
  - z.B. "UML-Klassendiagramm", "Programmcode", "Testfall", "Testbericht" usw.

Rollen, Aktivitäten, Artefakte (2)



© Helmut Balzert

Prozesse: Detailgrad, Verschiedenheit, Definiertheit

- Ein Prozess (deskriptiv) ist im Prinzip ungeheuer komplex
  - viel komplexer als das entstehende Produkt, da ja für jedes Detail im Produkt zahlreiche Abläufe vor sich gegangen sind
- Man meint deshalb mit "Prozess" in den meisten Fällen nur eine sehr starke Vergröberung des tatsächlich Ablaufenden

Aber wie grob?

- Vergröbert man zu wenig,
  hat jedes Projekt einen einmaligen Prozess
  - der zudem nicht vorher definiert ist
- Vergröbert man zu stark,
  haben alle Projekte den selben Prozess
  - und man lernt nichts mehr
- Eine sinnvolle Vergröberung ist so, dass sich zwar viele Projekte ähneln, aber andere noch verschieden aussehen
  - und dass die Unterschiede dazwischen interpretierbar sind

Elemente eines Prozessmodells

- Rollen
  - Welche gibt es?
    Wie genau sind sie definiert ("Arbeitsbeschreibungen")?
- Aktivitäten
  - Welche sind vorgesehen? Wie genau sind sie definiert?
  - Entscheidungsspielraum über Einsatz oder Art des Einsatzes?
  - Gibt es Checklisten?
    Vorgaben über Werkzeuge, Methoden, Richtlinien?
- Artefakte
  - Welche sind vorgesehen? Wie genau sind sie definiert?
    Wie verbindlich ist das?
  - Gibt es Vorlagen (Schablonen)?
- Steuerung der Aktivitäten
  - Gibt es einen festen Ablaufplan?
    Oder weichere Kriterien für die Abfolge von Aktivitäten?
  - Sind Eintritts- und Austrittsbedingungen definiert?

Der Urahn: Das Wasserfallmodell

- Es gibt eine kleine Zahl verschiedener Aktivitäten, z.B.
  1. Planung
  2. Anforderungsbestimmung
  3. Architekturentwurf
  4. Feinentwurf
  5. Implementierung
  6. Integration
  7. Validierung
  8. Inbetriebnahme
- Diese Aktivitäten werden der Reihe nach durchlaufen ("Phase")
  - Jede Phase nur einmal
  - Phase N beginnt erst nach Abschluss von Phase N-1
- Dokumenten-getriebener Prozess
  - alle Ergebnisse jeder Phase liegen in Dokumenten vor
- Am Ende jeder Phase erfolgt eine gründliche Prüfung der Ergebnisdokumente
  - und dann die Übergabe in die nächste Phase ("Meilenstein")
  - eventuell mit anderem Personal!

- Annahme:
  - Mängel in Phase N werden spätestens in Phase N+1 aufgedeckt
  - und können dann leicht in den Dokumenten beider Phasen korrigiert werden

Schwächen des Wasserfallmodells

1. Bei unklaren Anforderungen:
   - Wenn Anforderungen in der Anforderungsbestimmung nicht gut verstanden werden, braucht man als Hilfe den Entwurf, die Implementierung und die Validierung
   - Im Wasserfallmodell führt das zu Chaos, weil späte Änderungen der Anforderungen total das Prozessmodell durchbrechen
     - Entweder die Arbeitsweise mit gründlich ausgearbeiteten Dokumenten wird enorm teuer
     - oder die Dokumente werden nicht mehr korrekt gepflegt
2. Bei veränderlichen Anford.:
   - Das gleiche gilt, wenn sich Anforderungen irgendwann im Projektverlauf plötzlich von außen ändern können
3. Bei nicht beherrschten Architekturen:
   - Ähnlich wie bei unklaren Anf.
4. Durch "Über die Mauer werfen":
   - Kommunikation nur über Dokumente
     - Desaster, wenn Dokumente nicht gelesen werden
   - Verschiedenes Personal
   - Verständnis für Phänomene von Phase N ist in N+1 weitgehend verloren

Eigentlich ist das Wasserfallmodell nur eine Legende (W.W. Royce 1970).

1. Reparatur: Iteration

- Modernere Prozessmodelle empfehlen ein iteratives Vorgehen
  - Projektergebnis nicht "in einem Rutsch" anfertigen, sondern sich in mehreren Schritten "herantasten"

Vorteile:

- Senkt Komplexität in einzelnem Schritt (→ beherrschbarer)
- Kann mit unklaren oder veränderlichen Anforderungen etc. umgehen
- Verlangt engere Kommunikation der Beteiligten
  - und senkt dadurch die Neigung zum "Über die Mauer werfen"

Nachteile: (das sind aber beides in Wirklichkeit keine)

- Bewirkt eine gewisse Doppelarbeit (wg. "Zwischenlösungen") und hat deshalb theoretisch höheren Aufwand
- Verlangt engere Kommunikation der Beteiligten

Beispiele iterativer Prozessmodelle

- Prototypmodell:
  - Baue anfangs ein (Teil)System "zum Wegwerfen", um kritische Anforderungen besser zu verstehen
  - Danach folgt noch ein Prototyp oder Wasserfallmodell
- Inkrementelles Modell:
  - Baue das Gesamtsystem schrittweise
  - In jedem Schritt werden nur neue Teile hinzugebaut, aber es wird (theoretisch!) nie etwas Existierendes verändert
- Iteratives Modell (evolutionäres Modell):
  - Baue das Gesamtsystem schrittweise
  - In jedem Schritt werden neue Teile hinzugebaut und wo nötig auch existierende verändert
- Spiralmodell (Risikomodell):
  - Tue in jeder Iteration das, was am stärksten zur Verringerung des kritischsten Projektrisikos beiträgt
  - Das hat alle obigen Modelle als Spezialfälle

2. Reparatur: Allgemeine Ziele anstatt konkreter Pläne

Freie Universität Berlin

- Manche Prozessmodelle planen nicht gleich den Inhalt aller Iterationen von Anfang an
  - sondern geben nur grobe Ziele der Entwicklung vor
- Dies verbessert insbesondere die Bereitschaft, Anforderungsänderungen zu akzeptieren

Bezeichnung solcher Prozessmodelle:

- Agile Prozesse (oder agile Methoden)

Agile Methoden verlangen sehr enge Kommunikation!

Hauptunterschied zwischen Prozessmodellen: Wieviel Planung?

Freie Universität Berlin

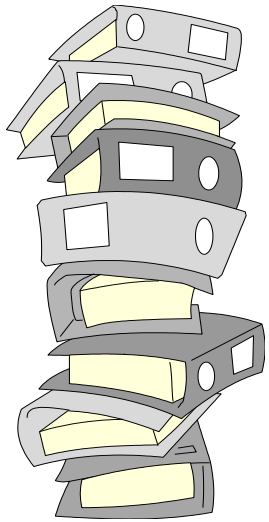
- Konkrete Prozessmodelle unterscheiden sich in vielerlei Hinsicht
- Wichtigste Dimension von Unterschieden:
  Wie präzise/strikt/weit_voraus wird geplant?

- Sehr: Wasserfallmodell
  - möglichst präzise und strikt, für das gesamte Projekt im Voraus
- Mittel: Iterative Modelle
  - Planen so weit und so präzise wie möglich
  - nicht strikt (nötige Veränderungen werden akzeptiert)
  - nur für wenige Iterationen im Voraus
- Wenig: Agile Prozesse
  - Nur so viel Planung wie unbedingt nötig
  - Lieber Ziele als Pläne (wg. Flexibilität)

Wieviel Planung?
nach Barry Boehm

Freie Universität Berlin

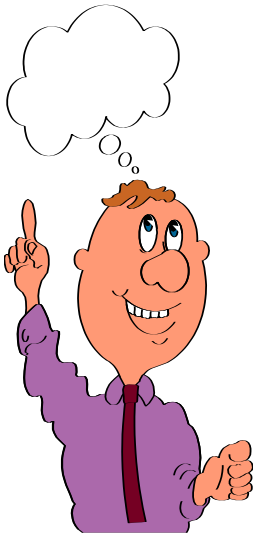
feingranulare Verträge
++ klare Arbeitsgrundlage
+ finanzielle Sicherheit
- - enorm aufwändig
- - Änderungen auch für AG schwer durchsetzbar



Wieviel Planung?
nach Barry Boehm

ad hoc
+ wenig Planungsaufwand
+ individuelle Freiheit
- - Ergebnis unvorhersehbar
- - abhängig von "Helden"

feingranulare Verträge



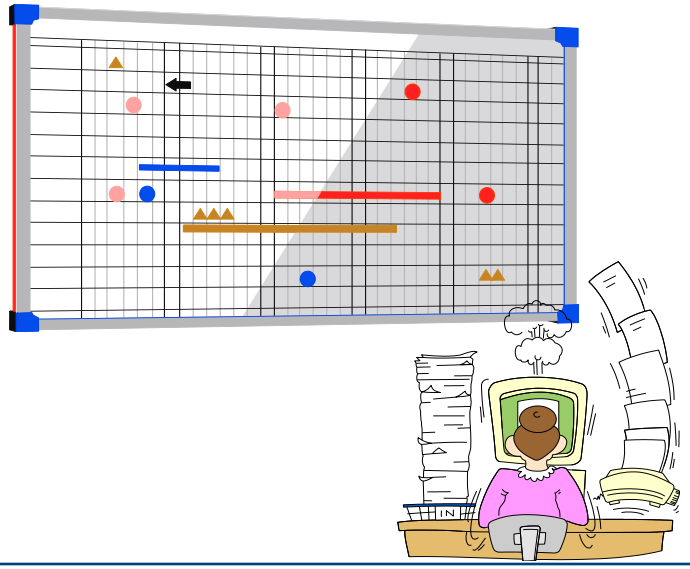
Wieviel Planung?
nach Barry Boehm

Freie Universität Berlin

Meilenstein- u. Plangetrieben
+ langfristige Vorhersagen
+ gute Zustandskontrolle
- - Änderungen aufwändig
- - unrealistische Annahmen
schwer zu eliminieren

ad hoc

feingranulare Verträge



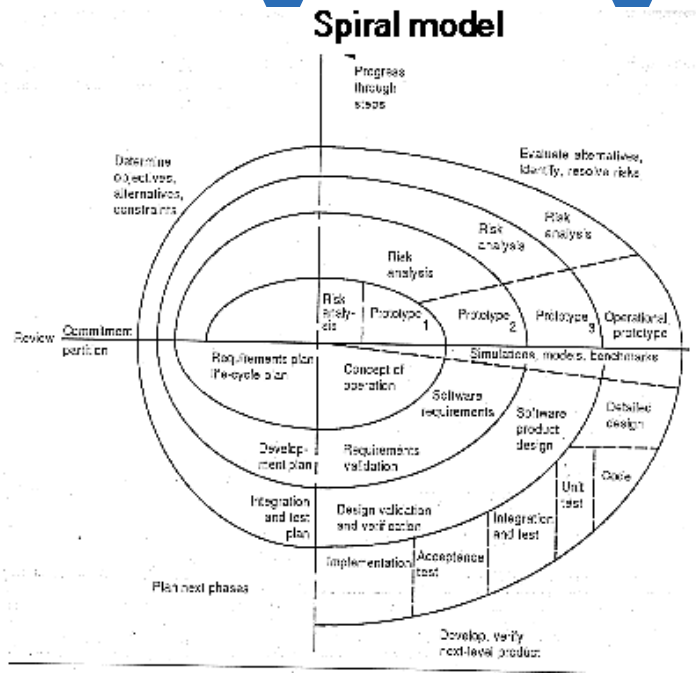
Wieviel Planung?
nach Barry Boehm

Meilenstein- u. Risikogesteuert
++ Risiken aktiv ausgeräumt
+ Teilergebnisse früh
- kaum langfristig planbar
- relativ aufwändig

ad hoc

Meilenstein- u. Plangetrieben

feingranulare Verträge



Wieviel Planung?
nach Barry Boehm

Freie Universität Berlin

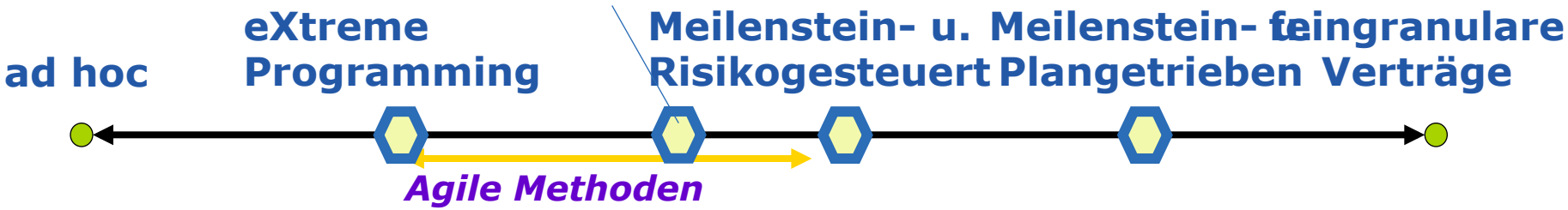
eXtreme Programming
+ früh Kernergebnisse
+ flexibel für Änderungen
- in sehr großen Projekten Zusatzplanung nötig
- viel Selbstdisziplin nötig

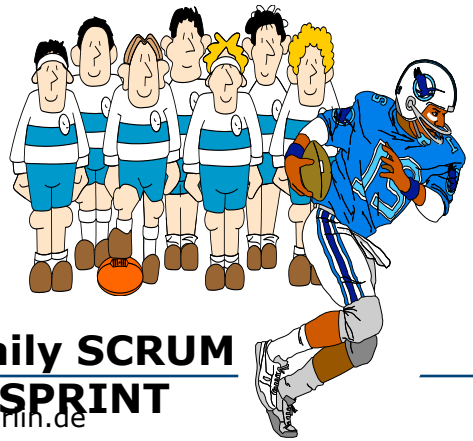
ad hoc **Meilenstein- u. Risikogesteuert** **Meilenstein- u. Plangetrieben** **feingranulare Verträge**

Was bedeutet agil?
Planungshorizont nach Maß n. Barry Boehm

Agiles Beispiel: SCRUM

+ mittelfristig geplant
+ reagiert schnell
- hängt an MA-Qualifikation
- Endprodukt nicht spezifiziert





Daily SCRUM + SPRINT

Prinzip
und Denkweise von Agilen Methoden

Freie Universität Berlin

nach Frühauf, Conquest 2001	**Bisheriger Ansatz**	**Agiler Ansatz**
Ständige Mitwirkg. des Kunden	unwahrscheinlich	kritischer Erfolgsfaktor
Etwas Nützliches wird geliefert	erst nach einiger (längerer) Zeit	mindestens alle sechs Wochen
Das Richtige entwickeln durch	langes Spezifizieren, Vorausdenken	Kern entwickeln, zeigen, verbessern
Nötige Disziplin	formal, wenig	informell, viel
Änderungen	erzeugen Widerstand	werden erwartet und toleriert
Kommunikation	über Dokumente	zwischen Menschen
Vorsorge für Änderungen	durch Versuch der Vorausplanung	durch "flexibel bleiben"

Wann sind welche Modelle angemessen?

Freie Universität Berlin

Strikte, stark planende Modelle:

- Wenn ein wohldefiniertes Resultat in definierter Zeit erreicht werden muss
- Wenn sehr große (insbesondere verteilte) Projektgruppen koordiniert werden müssen
  - Denn dann sind die Pläne und Dokumente zur Koordination schwer verzichtbar

Agile Modelle:

- Wenn hohe Unsicherheit über die Anforderungen besteht
  - Inhalt, Prioritäten
- Wenn Änderungen von außen häufig sind
  - Anforderungen, Zeitplan, Budget, Qualitätsziele etc.

Andere Kriterien für die Auswahl des Prozessmodells

Freie Universität Berlin

Kräfte in Richtung auf stark planende Modelle:

- Parallele Entwicklung von Hardware (z.B. Auto) und SW
- Örtlich verteilte Entwicklung (auch: in mehreren Firmen)

Kräfte in Richtung auf agile Modelle:

- Hat überhaupt keinen Zeitplan (z.B. Hobbyprojekt)
- Arbeit mit unausgereifter und unbeherrschter Technologie
  - und andere Arten stark radikalen Vorgehens

Sonstige:

- Wie viel u. welche Wiederverwendung ist geplant/gewünscht?
- Wie viel Reverse Engineering ist nötig?
- Welche sonstigen großen Risiken gibt es?

Universal-Prozessmodelle: Die eierlegende Wollmilchsau

Freie Universität Berlin

Es gibt Prozessmodelle, die den Anspruch erheben, universell geeignet zu sein, z.B.:

- Rational Unified Process (RUP) + Royce's Unified PM Approach
  - Fortentwicklung von Objectory (Ivar Jacobson) und Spiralmodell
  - RUP wird heute als Produkt vertrieben von IBM/Rational
  - Walker Royce's Methode ist ein Buch: "Software Project Management: A Unified Framework", Addison Wesley 1998.
- V-Modell XT
  - Entwickelt im Auftrag der deutschen Regierung

- Beide Modelle sind stark an Projektbedingungen anpassbar
- Beide Modelle sind sehr umfangreich, nicht leicht zu verstehen
  - Die meisten Leute, die behaupten, sich damit auszukennen, haben nur Grundkenntnisse

The Rational Unified Process (RUP)

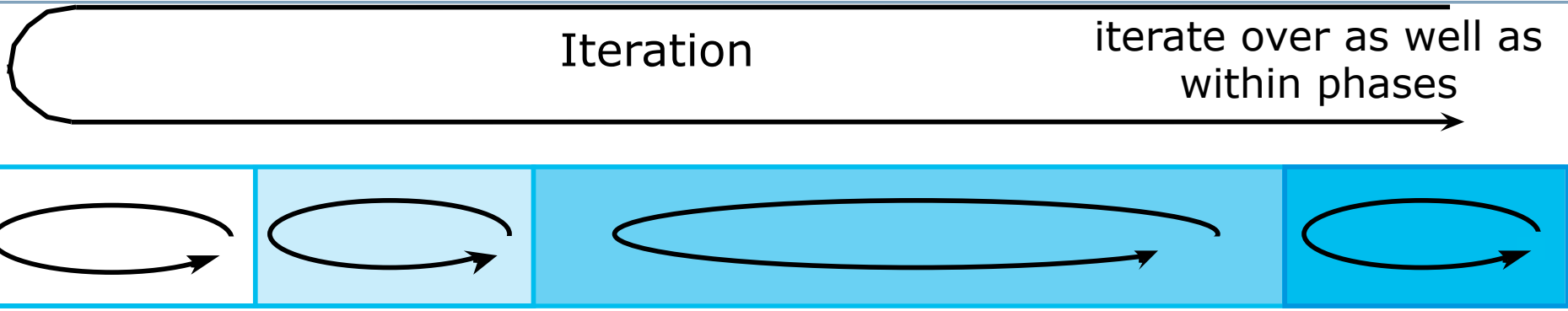
- A modern **evolutionary** (iterative) process model
- Easy access to a knowledge base with **guidelines**, **templates** and **tool mentors** for all critical development activities.
- Rather than focusing on the production of large amount of paper documents, the Unified Process emphasizes the development and maintenance of **models**.
- The Rational Unified Process is a guide for how to effectively use the **Unified Modeling Language** (UML).
- Is supported by **tools**, which automate large parts of the process.
- Is a **configurable process**. No single process is suitable for all software development.

The Rational Unified Process (RUP)

- Normally described from **3 perspectives**
  - 1. A dynamic perspective that shows (four) **phases** over time
  - 2. A static perspective that shows process **activities**
  - 3. A practice perspective that suggests **good practice**
- RUP is the interaction of all three perspectives

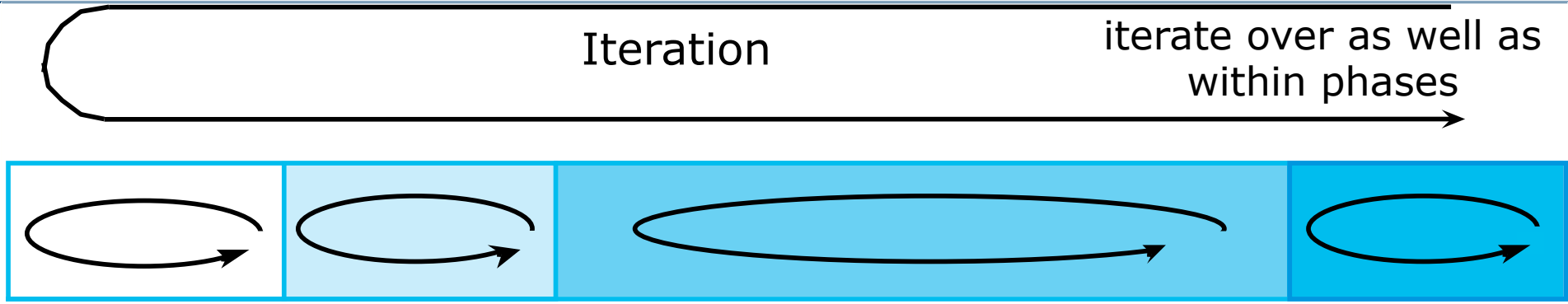
- In a concrete project, **RUP is tailored as appropriate**
  - only parts of the overall RUP definition will be used
  - (For instance, the full RUP defines about 30 roles)

Perspective 1: RUP phases



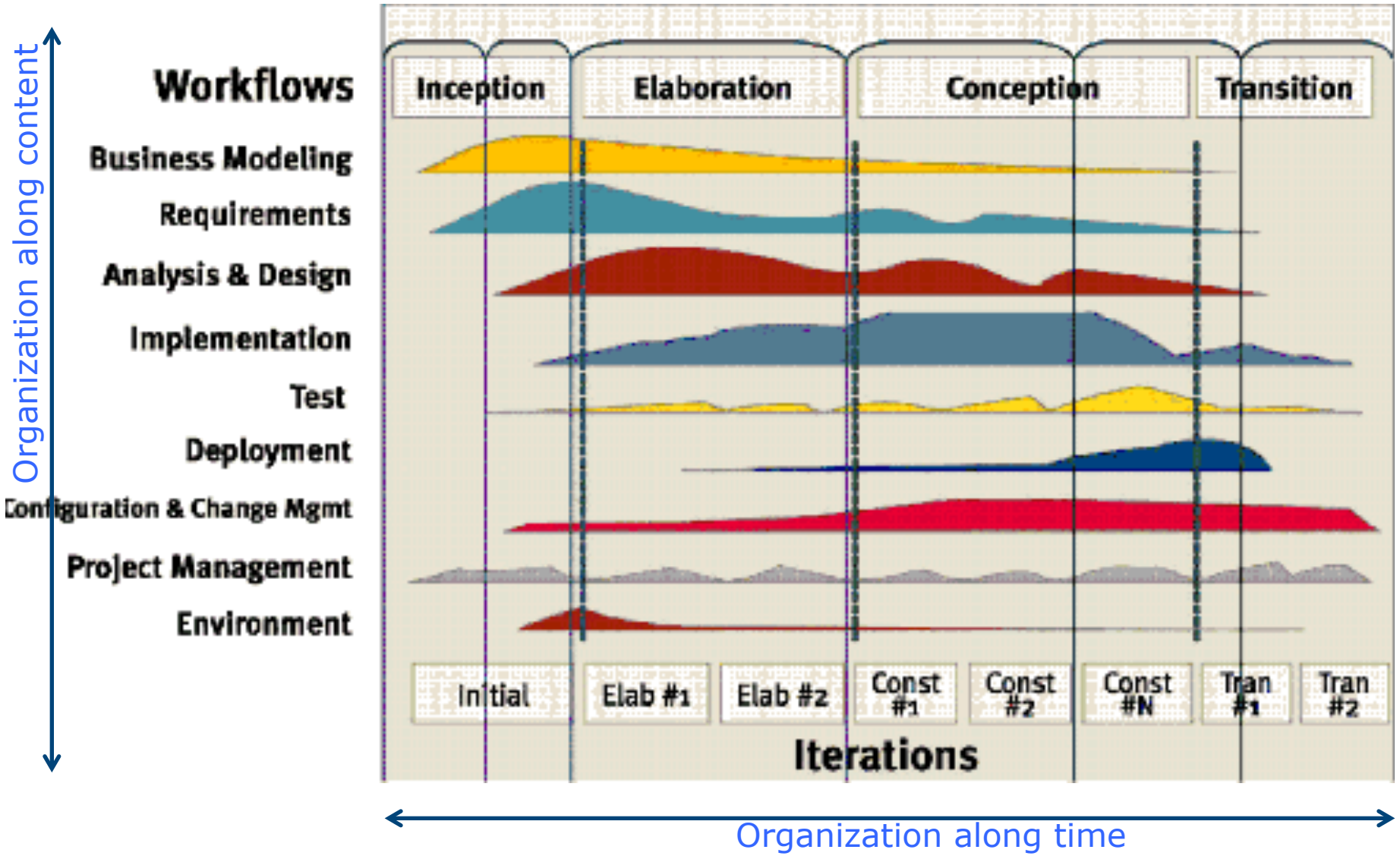
- Inception
  - Establish the **business case** for the system / Outcome:
    - A vision document (core project's requirements, key features, …)
    - A initial use-case model (10% -20% complete)
    - An initial risk assessment
    - …
- Elaboration
  - Analyze the problem domain, establish a sound **architectural foundation**, develop the **project plan**, and **eliminate the highest risk elements** / Outcome:
    - A use-case model (at least 80% complete)
    - A Software Architecture Description.
    - …

Perspective 1: RUP phases



- Construction
  - All remaining components and application features are **developed** and **integrated** into the product, and all features are thoroughly **tested** / Outcome:
    - The software product integrated on the adequate platforms
    - The user manuals
    - A description of the current release
    - …
- Transition
  - Deploy the system in its operating environment

Intensity of workflows in different phases and iterations (simplistic, but popular)



Perspective 3: RUP good practice

- Develop Software Iteratively
  - An iterative approach is required that allows an **increasing understanding** of the problem through successive refinements. RUP supports an iterative approach.
- Manage Requirements
  - RUP describes how to **elicit, organize, and document required functionality** and constraints
- Use Component-based Architectures
  - The process focuses on early development and baselining of a **robust executable architecture**.
  - RUP supports **component-based software development**.
- Visually Model Software
  - The process shows you how to **visually model software** to capture the structure and behavior of architectures and components.
  - Allows you to **hide the details** and write code using "graphical building blocks." Visual abstractions **help you communicate** different aspects of your software
  - **Unified Modeling Language** (UML)
- Verify Software Quality
  - Rational Unified Process assists you in the planning, design, implementation, execution, and evaluation tests.
- Control Changes to Software
  - RUP describes how to control, track and monitor changes to enable successful iterative development.

RUP: How much Planning?

- **The project plan is developed iteratively** (along the SW lifecycle)
  - The Plan is detailed and refined as the stakeholders increase their knowledge in the application and solution domain
- Planning errors are treated like software defects
  - the earlier they are resolved, the less impact they have
- **Work breakdown structure** is organized around software life cycle activities
  - Level 1: life cycle workflows (management, requirements, design, implementation, assessment, and deployment)
  - Level 2: phases (inception, elaboration, construction, transition)
  - Level 3: the artifacts produced during each phase
- Estimation:
  - Compute the initial estimate with a model, such as COCOMO II
  - Refine it with the project manager, developers, and testers

V-Modell

- Entwickelt ca. 1987–1992 im Auftrag des BMVg und BMI als "**Vorgehensmodell** zur Softwareentwicklung"
  - Überarbeitet 1997 als V-Modell 97 (Objektorientierung)
  - Überarbeitet 2005 als V-Modell XT ("extreme tailoring")
- Sehr umfangreiches Modell (V-Modell XT hat 9 Teile):
  - **1: Grundlagen des V-Modells** (Einführung, Konzepte, ...)
  - **2: Eine Tour durch das V-Modell** (Kleines Beispielprojekt)
  - **3: Referenz Tailoring** (Anpassung an mein Projekt)
  - **4: Referenz Rollen** (Wen brauche ich?)
  - **5: Referenz Produkte** (Wie sollen Projektergebnisse aussehen, was enthalten sie?)
  - **6: Referenz Aktivitäten** (Wie erstelle ich Produkte?)
  - **7: Referenz Konventionsabbildungen** (Ich kenne schon VM97/CMMI/... Wo im V-Modell XT finde ich Entsprechendes?)
  - **8: Anhang** (Werkzeuge, Methoden, Glossar...)
  - **9: Vorlagen** (Anwendungsbeschreibung)

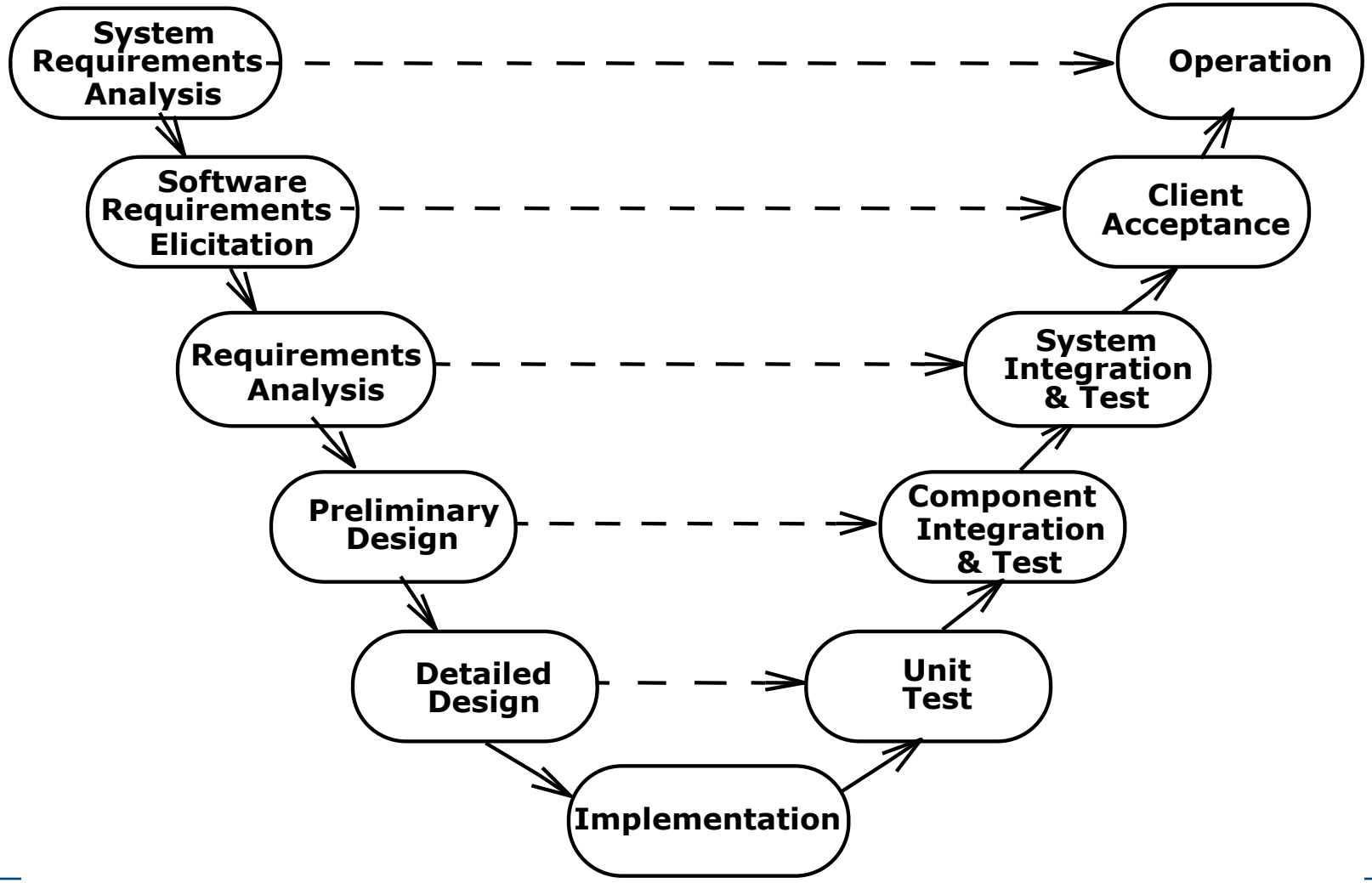
V-Modell Kern

- Das V-Modell wurde vor allem für sehr große Projekte konzipiert. V-Modell XT ist auch an kleinere anpassbar.
- Das V-Modell XT kennt 4 verbindliche (und 16 optionale) so genannte Vorgehensbausteine:
  - PM: Projektmanagement
  - QS: Qualitätssicherung
  - KM: Konfigurationsmanagement
  - PROB: Problem- und Änderungsmanagement
- Diese fassen zusammengehörige Aktivitäten zusammen

- Jedes Produkt (Dokument) durchläuft vier Zustände: geplant, in Bearbeitung, vorgelegt, akzeptiert
  - Qualitätssicherung ist also ein zentraler Bestandteil des V-Modells

V-Modell: Das andere V

Nicht namensgebende Korrespondenz zw. Entwicklungs- und Validierungsaktivitäten



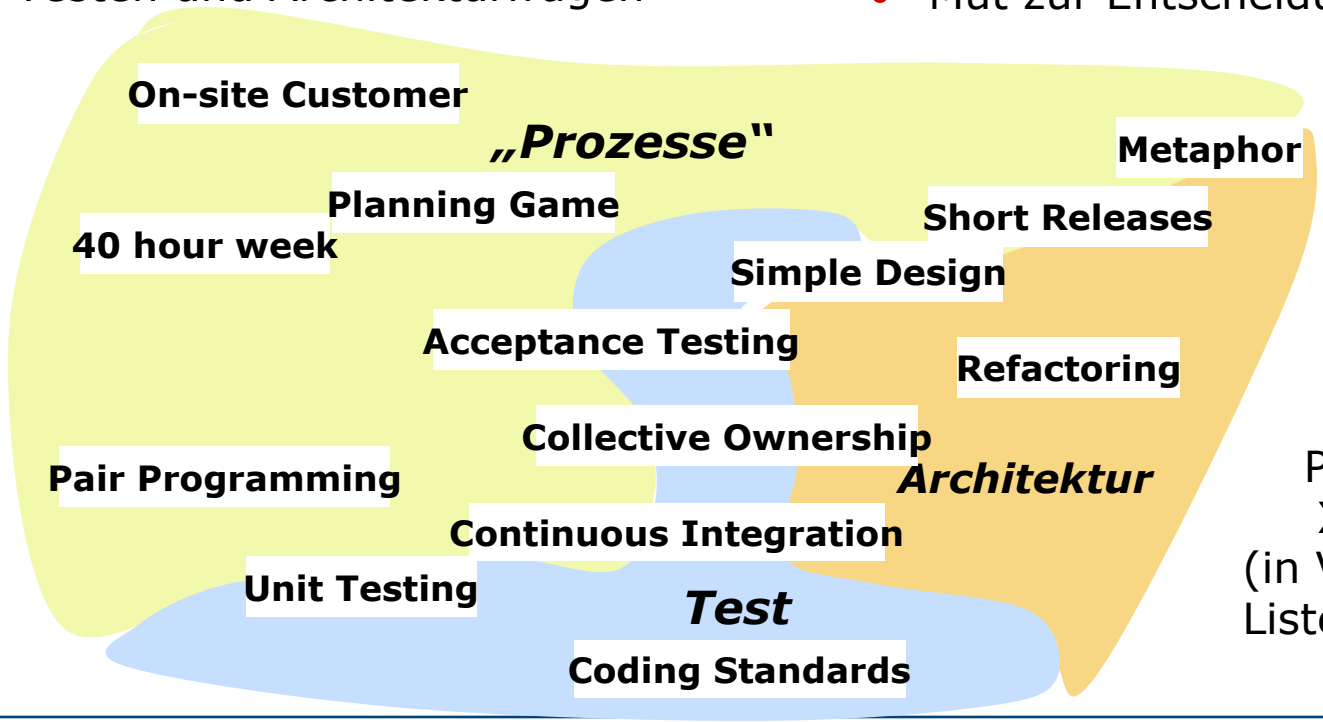
Nicht Fisch und nicht Fleisch?

- RUP und V-Modell können je nach Bedarf eher wasserfallartigen Charakter oder eher agilen Charakter entfalten
- Dadurch werden sie aber auch schwieriger zu verstehen
  - und es gibt leicht Streit über sie, weil sie gar nicht als so flexibel wahrgenommen werden

- Es fehlt uns also noch ein "reinrassiges" Gegenstück zum Wasserfallmodell
- Es gibt diverse ausgearbeitete Agile Methoden
  - z.B. für eher kleine Projekte:
    - **XP** (**"eXtreme Programming"**, Kent Beck)
    - SCRUM [nur Projektmanagement] (Jeff Sutherland, Ken Schwaber)
    - Crystal Clear (Alistair Cockburn)
  - z.B. für mittelgroße Projekte:
    - Scrum of Scrums (verschachteltes SCRUM)

eXtreme Programming (XP)

- XP ist ein Satz von Praktiken
  - Müssen alle genau eingehalten werden
    - verlangt sehr hohe Disziplin
  - Verstärken sich gegenseitig
  - Zusammenspiel von Prozess, Testen und Architekturfragen
- Werte und Prinzipien
  - Kommunikation u. Kundeneinbindung
  - Feedback und inkrementelle Entwicklung
  - Einfachheit
  - Mut zur Entscheidung



Praktiken von XP Version 1 (in Version 2 ist die Liste etwas anders)

Was heißt hier extrem?

Freie Universität Berlin

- "eXtreme Programming" bedeutet nicht, nur zu hacken
  - sondern erfordert ganz im Gegenteil ***extreme Disziplin***
- Der Name kommt daher, dass XP bewährte Praktiken aus der SW-Entwicklung ins Extrem führt
  - Beispiel: Codedurchsichten haben sich bewährt, daher verlangt XP ständige Code-Reviews (durch Pair Programming)

- XP ist die vielleicht bekannteste Agile Methode
  - die verbreitetste ist aber inzwischen Scrum
- XP ist eine sehr radikale Agile Methode
- XP ist nicht die einzige, beste oder einfachste Agile Methode
  - Eine beste gibt es auch nicht: Muss zur Situation passen!

Agilitätsfördernde Praktiken in XP

- Short Releases (Kurze Freigabezyklen):
  - alle paar Wochen wird eine benutzbare Version an den Kunden ausgeliefert
  - → kompletter Projektfehlschlag ist fast unmöglich
  - → Anforderungsmängel werden früh erkannt
- On-site customer (Kunde vor Ort):
  - entscheidngsbefugter Vertreter des Auftraggebers ist stets beim Projektteam
  - beschreibt Anforderungen, priorisiert, entscheidet, nimmt Ergebnisse ab
  - → Unklarheiten können *umgehend* ausgeräumt werden
- Pair programming (Paarprogrammierung):
  - sämtlicher Code wird von je zwei Programmierern gemeinsam geschrieben
  - → jede Klasse ist mindestens zwei Personen vertraut
  - → mehr Flexibilität bei "wer macht was wann?"
- Unit Testing (Modultests):
  - zu jedem Modul gibt es gründliche automatisierte Tests
  - → erhöhte Bereitschaft, neue Anforderungen zu akzeptieren
    - Leider müssen die Tests auch geändert werden…
  - → weniger Angst vor Änderungen am Code
- u.a.m.

Literatur

- Wasserfallmodell:
  - http://c2.com/cgi/wiki?WaterFall
- Spiralmodell:
  - Barry Boehm: "A Spiral Model of Software Development and Enhancement", IEEE Computer 21(5):61-72, 1988.
- Royce's Method:
  - Walker Royce: "Software project management: A unified framework", Addison-Wesley Longman 1998
  - http://www.ibm.com/software/awdtools/rmc/index.html
- V-Modell XT:
  - http://www.v-modell-xt.de
- XP:
  - Kent Beck, Cynthia Andres: "Extreme Programming Explained: Embrace Change", 2nd ed., Addison-Wesley 2004
    - 2. Ausgabe: hat andere Liste von Praktiken als die oben vorgestellten
  - www.extremeprogramming.org, www.xprogramming.com

Zusammenfassung

- Ein Prozessmodell beschreibt Aktivitäten, Rollen und Produkte im Softwareprozess
- Es gibt ein weites Spektrum verschiedener Prozessmodelle
  - von stark planungsorientierten (Extremfall: Wasserfallmodell)
    - eignen sich für Projekte, die sehr gut geplant werden können
    - und solche, die sehr gut geplant werden müssen (z.B. sehr große)
  - bis zu stark kommunikationsorientierten (z.B. eXtreme Programming)
    - eignen sich für Projekte mit hohem Änderungsbedarf unterwegs
- Es gibt recht universelle, für viele Fälle anpassbare Modelle
  - Rational Unified Process, V-Modell XT
  - Diese sind aber umfangreich und nicht einfach zu verstehen
- Prozessmodelle sind nicht gut oder schlecht, sondern zum Projekt passend oder unpassend

Danke!

Additional information on RUP: Perspective 2: Static workflows (1)

Freie Universität Berlin

- Business modelling
  - The business processes are modelled as use cases
- Requirements
  - Actors who interact with the system are identified
  - Use cases are developed to model the system requirements
- Analysis and design
  - A design model is created and documented using architectural models, component models, object models and sequence models
- Implementation
  - The components in the system are implemented and structured into implementation sub-systems
  - Automatic code generation from design models helps accelerate this process
- Test
  - Testing is an iterative process that is carried out in conjunction with implementation
  - System testing follows the completion of the implementation

Perspective 2: Static workflows (2)

- Deployment
  - The product is installed in users' workplaces
- Configuration and change management
  - Coordinate and protocol changes to the system
- Project management
  - Plans, coordinates, and manages resources
- Environment
  - This workflow is concerned with making appropriate software tools available to the software development team

RUP: How much Reuse?

- Make-or-buy decisions are treated as risks that should be confronted early in the life cycle (e.g., in the first iterations of the elaboration phase)
  - When components are reused in more than one project, the return on investment can be further increased
- Key priniciple: Minimize the amount of human-generated source code
  - Reuse commercial components (COTS)
  - Use code generation tools
  - Use high-level visual and programming languages
- Reuse is treated as a return on investment decision which decreases development time
  - Mature components and tools also reduce time to repair defects
  - Note: Immature components and tools increase quality problems drastically which off-sets any economic benefit

RUP: How much Modeling?

Modeling artifacts:

- Management Set (planning, monitoring):
  - Ad-hoc notations are used to capture the "contracts" among the stakeholders
  - Specific artifacts: Problem statement, software project management plan, configuration management plan, and status descriptions
- Requirements set
  - Visionary scenarios
  - Prototypes for user interfaces
  - Requirements analysis model
- Design set
  - Software architecture
  - Interface specifications
- Implementation set
  - Source code, components and executables needed for testing the system
- Deployment set
  - All the deliverables negotiated between the project manager and the client
  - In general it contains the executable code, the user manual and the administrator manual

Test artifacts are part of each of the above sets:

- Management set includes test plan and procedures
- Requirements set includes test specifications

RUP: How much Control?

- Royce's methodology focuses on three management metrics and four quality metrics

- Management metrics:
  - Work. Number of tasks completed (compared to the plan)
  - Cost. Amount of resources consumed (compared to the budget)
  - Team dynamics. Number of participants that leave the project prematurely or that are added

- Quality indicators:
  - Change traffic. How many change requests are issued over time?
  - Breakage. How much source code is reworked per change?
  - Rework. How much effort is needed to implement a change?
  - Mean time between failures. How many defects are discovered per hour of testing?