Zürcher Hochschule
für Angewandte Wissenschaften

School of Engineering

Bachelor of Science (BSc) in Informatik

Modul Advanced Software Engineering 1 (ASE1)

LE 02 – Software Engineering Prozesse

Institut für Angewandte Informationstechnologie (InIT)

Walter Eich (eicw) / Matthias Bachmann (bacn)

http://www.zhaw.ch/de/engineering/institute-zentren/init/

Agenda

1. Motivation und Zielsetzung
2. Software-Prozessmodelle
3. Prinzipien agiler Softwareentwicklung
4. Tailoring des Prozesses für ein Projekt
5. Wrap-up

Lernziele

Sie sind in der Lage,

- die Charakteristiken der gängigen Software-Prozessmodelle (plangetrieben, agil) und deren Anwendungsgebiete (Home Ground) zu erläutern.
- Kriterien für mehr oder weniger Zeremonie in einem Software-Prozess (Tailoring) zu nennen.
- zu verstehen, dass ein Prozess dem Problem angepasst werden muss (agiles Credo).

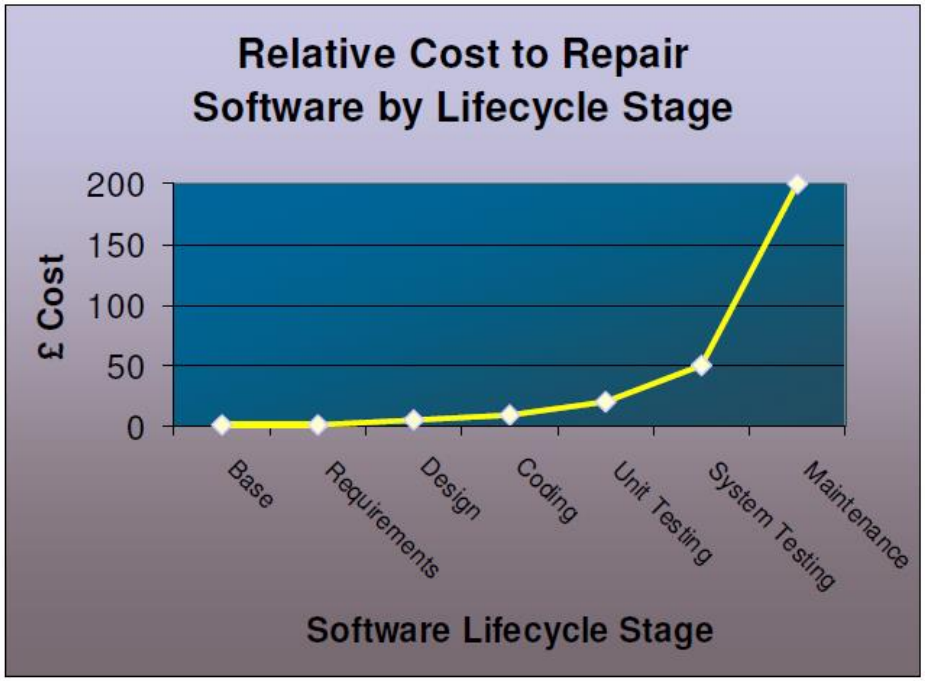
Motivation und Zielsetzung

- Der Produktlebenszyklus startet bei der ersten Idee und endet bei der Ausmusterung der Softwarelösung.
- Warum ist eine strukturierte Softwareentwicklung notwendig?
  - Fehlerentstehung und Fehlerkosten während der Softwareentwicklung
  - Strukturierung von Software-Entwicklungsprozessen
  - Anforderungen des Kunden sind die Basis für die Softwareentwicklung; durch sie wird definiert, was das Produkt leisten soll
- Je nach Projektkriterien (Grösse, Art, Typ, usw.) stehen zahlreiche Prozessmodelle zur Verfügung, die den wesentlichen Phasen des Software-Life-Cycles folgen (Vorgehensmodelle).
- V-Modell und iterativ-inkrementelle Entwicklung als Beispiele für Vorgehensmodelle.

Cost to Fix

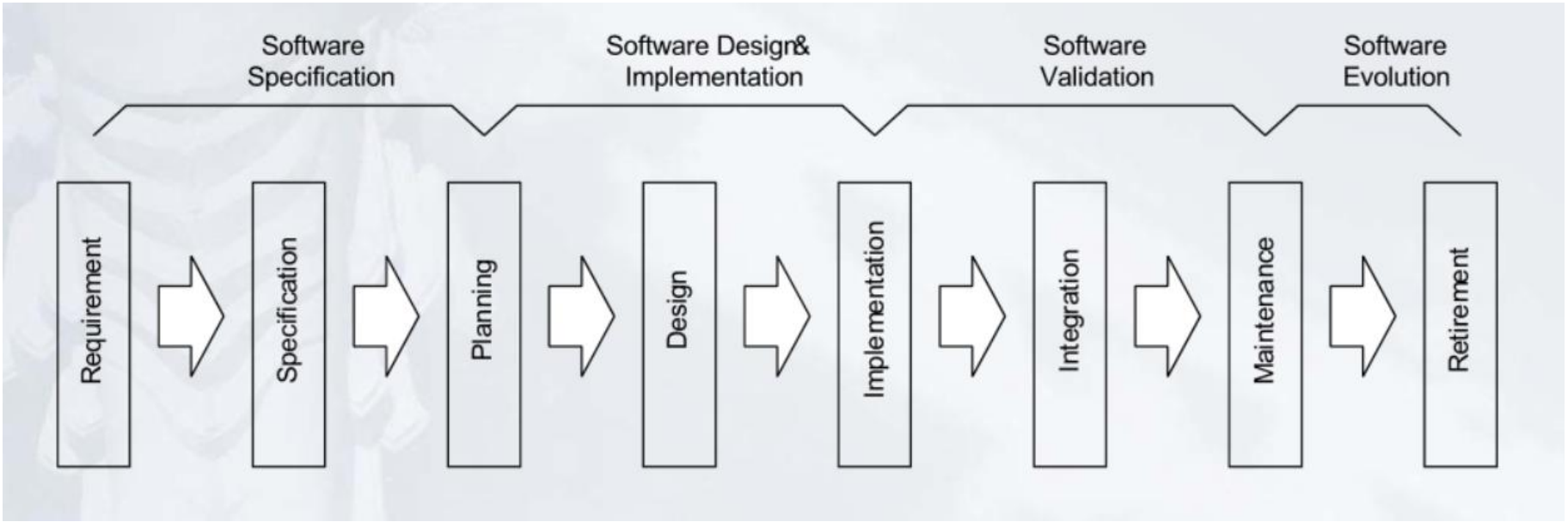
- Die Korrektur eines Fehlers welcher zu Beginn gemacht wurde, verursacht einen 200x grösseren Aufwand in der Wartung.

Base	1
Requirements	2
Design	5
Coding	10
Unit Testing	20
System Testing	50
Maintenance	200



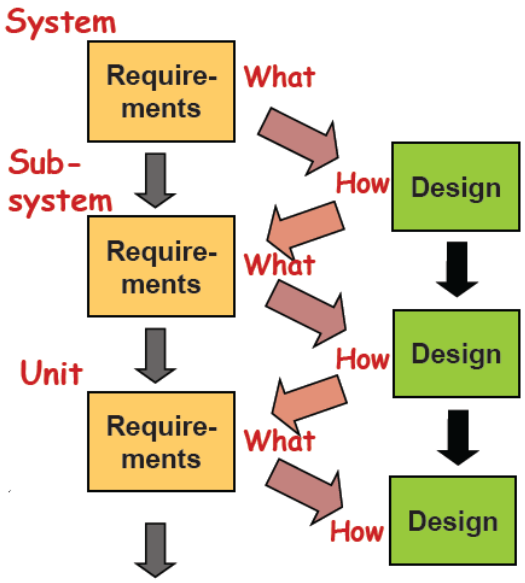
Übersicht über den Software Life-Cycle Prozess

- Ein Software Prozess ist eine Abfolge von Schritten (Phasen) mit all seinen Aktivitäten, Beziehungen und Ressourcen.
- Der Software-Life-Cycle beschreibt ein Basiskonzept für Software Engineering Prozesse.



Aspekte des Softwareentwicklungsprozess

- Ermitteln und Analyse der Anforderungen
  - Was genau soll die Software können?
- Architektur / Entwurf (Design)
  - Wie soll die Software erstellt werden?
- Implementierung/Umsetzung (Construction)
  - Ausführbaren Code erzeugen
- Test (Testing)
  - Unit Tests, Integrationstests, Systemtests, Akzeptanztests, ...
- Inbetriebnahme (Deployment, Configuration, Start of Operation)
  - Installation, Konfiguration, Schulung, organisatorische Umstellung
- Wartung/Betrieb (Maintenance/Operation)
  - Fehlerbeseitigung, Software ändern und erweitern



Agenda

1. Motivation und Zielsetzung
2. **Software-Prozessmodelle**
3. Prinzipien agiler Softwareentwicklung
4. Tailoring des Prozesses für ein Projekt
5. Wrap-up

Software-Prozessmodelle

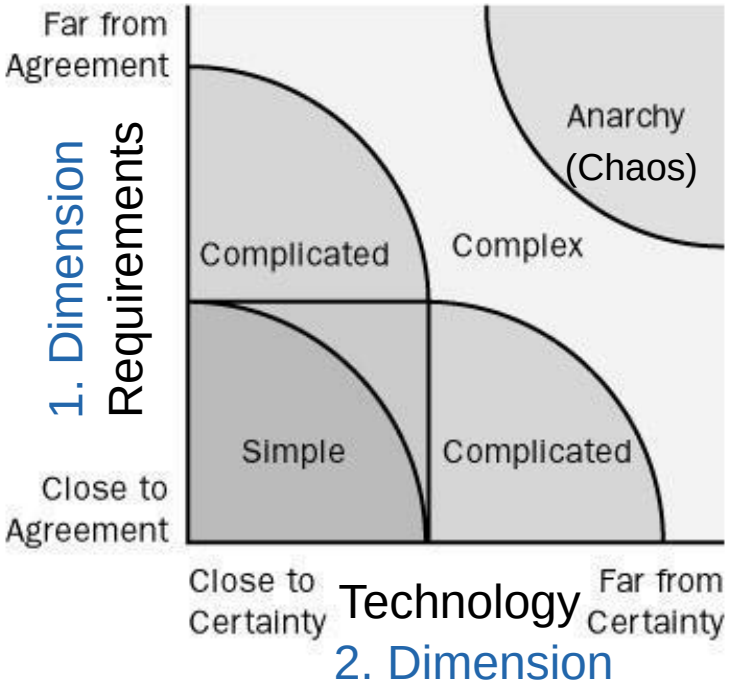
Leitfragen

- Sind eigentlich alle Software-Probleme gleich und falls nicht, wie lassen sie sich klassifizieren?
- Mit was für Strategien und Lösungsmustern versuchen die gängigen Software-Prozessmodelle diese Software-Probleme zu lösen?
- Was sind die Annahmen innerhalb dieser Software-Prozessmodelle und wie wird es praktiziert?
- Was steckt eigentlich hinter agiler Software-Entwicklung?
- Was sind die wichtigen Faktoren für das Tailoring eines Software-Prozessmodells für ein Projekt?

Begrifflichkeit

- Prozess
  - Ablauf eines Vorhabens mit der Beschreibung der Schritte (Aktivitäten), der beteiligten Personen (Rollen), der für diesen Ablauf benötigten Informationen und der dabei entstehenden Information (Artefakte).
- Entwicklung und Wartung von Software sind Prozesse.
- Ein (Software-)Prozessmodell ist die Beschreibung eines Software-Prozesses als präskriptives Modell für die Durchführung der Projekte.
  - Modellvorstellung über den Ablauf der Entstehung einer Software
  - Planung und Lenkung am Modell orientieren
  - Grundlage für eine erfolgreiche Software-Projektführung
- Im Kern besteht ein (Software-)Prozessmodell aus einem Vorgehensmodell ergänzt durch Organisationsstrukturen.

Wie können Software-Entwicklungs-probleme klassifiziert werden?



3. Dimension

Skills, Intelligence Level, Experience, Attitudes, Prejudices

Quelle: Agile Project Mangement with Scrum, Ken Schwaber, 2003

Ausgewählte Vorgehensmodelle (Familien) zur Lösung von Software-Problemen

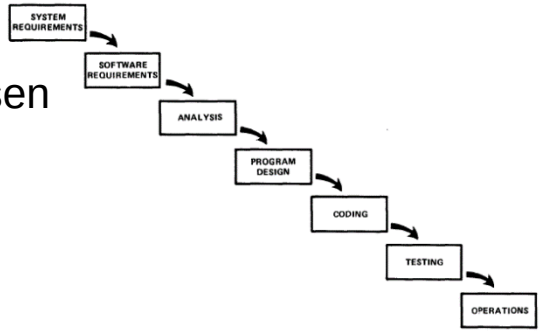
- Code and Fix
- Wasserfallmodell
- V-Modell
- Spiralmodell
- Rapid Application Development
- Iterative und inkrementelle Modelle
- Agile Software-Entwicklung

Code and Fix

- Definition
  - Code and Fix bezeichnet ein Vorgehen, bei dem Codierung oder Korrektur im Wechsel mit Ad-hoc-Tests die einzigen bewussten ausgeführten Tätigkeiten der Software-Entwicklung sind.
- Annahme, Paradigma
  - Erstens kommt es anders und zweites als man denkt!
- Vorteile
  - Entspricht dem Drang schnell voranzukommen.
  - Liefert schnell Ergebnisse.
  - Einfache Tätigkeiten (Codieren, Testen, Fixen).
- Nachteile
  - Projekt schlecht planbar (Funktionalität, Zeit, Kosten und Qualität) und keine Unterstützung für die Entwicklung im Team.
  - Aufwand für Korrekturen unangemessen hoch.
  - Schlecht wartbare Software.

Wasserfallmodell (Winston W. Royce 1970)

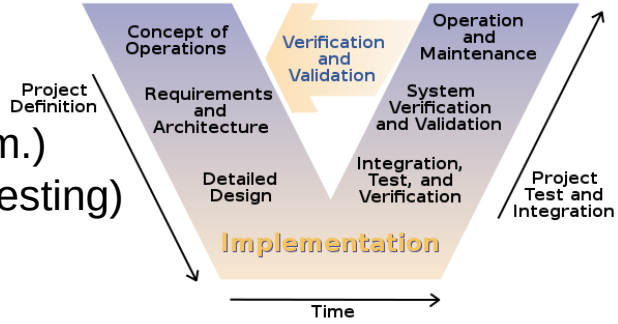
- Definition
  - Die Software-Entwicklung wird als Folge von Aktivitäten/Phasen betrachtet, die durch Teilergebnisse (Dokumente) gekoppelt sind. Die Reihenfolge der Aktivitäten ist fest definiert.
- Annahme, Paradigma
  - Ziel ist früh bestimmbar und Gelände ist bekannt (Metapher: ballistische Rakete).
- Vorteile
  - Hohe Planbarkeit (Funktionalität, Zeit und Kosten).
  - Einfaches, definiertes Vorgehen mit klarer Arbeitsteilung.
- Nachteile
  - Nicht-lokale Iterationen erschweren die Projektführung (bei Einbahnstrassenmodell).
  - Schlechtes Risikomanagement und Qualität als einzige Variable.



Quelle: Royce, Winston "Managing the Development of Large Software Systems“, Proceedings of IEEE WESCON, 1970

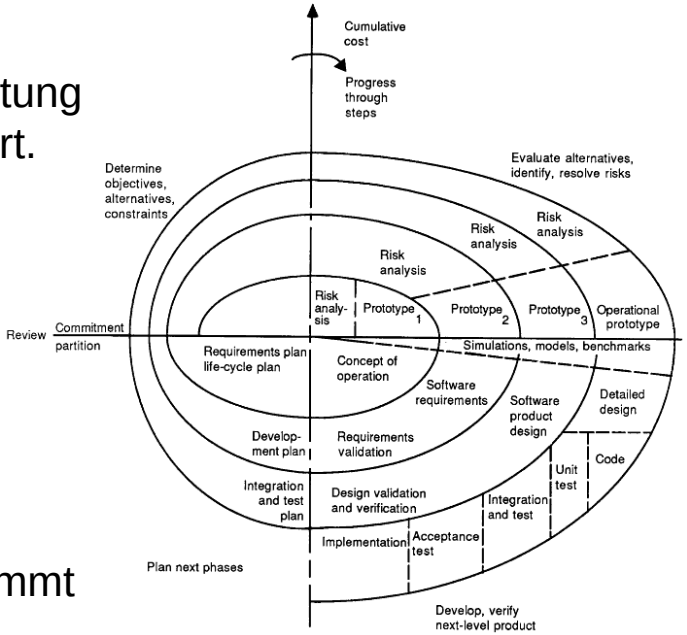
V-Modell (Barry Böhm 1979)

- Definition
  - Sequentielles Vorgehen wie Wasserfallmodell, aber umfassender (Projekt- und Konfigurationsmanagement u.m.) sowie mit korrelierenden qualitätssichernden Aktivitäten (Testing) ergänzt.
- Annahme, Paradigma
  - Gleich wie beim Wasserfallmodell (+ Validierung, Verifikation).
- \+ Vorteile
  - Kann durchgängig für System- und Software-Engineering verwendet werden.
  - Hohe Planbarkeit (Funktionalität, Zeit, Kosten und Qualität).
  - Klares, definiertes Vorgehen mit Qualitätssicherung.
- − Nachteile
  - Nicht-lokale Iterationen erschweren die Projektführung
  - Schlechtes Risikomanagement, wenn als Einbahnstrassenmodell verwendet.



Spiralmodell (Barry Böhm 1986)

- Definition
  - Generisches, risiko-orientiertes Modell, das eine Anleitung zur konkreten Ausprägung eines Prozessmodells liefert.
  - Planungszyklus:
    1. Planung
    2. Zielsetzung/Zielkorrektur
    3. Risikountersuchung
    4. Entwicklung und Prüfung
- Annahme, Paradigma
  - Wenn Risiken früh erkannt und bearbeitet werden, kommt es gut ;-)

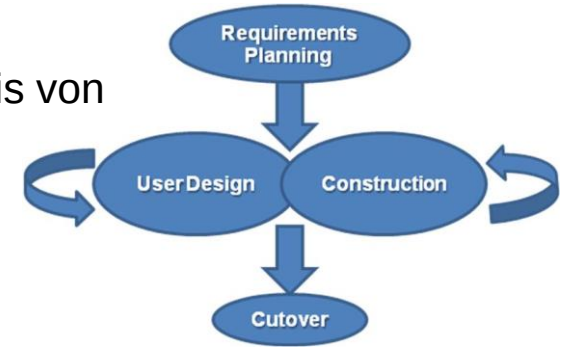


+ Vorteile
  - Risikogetrieben und hohe Adaptierbarkeit.

− Nachteile
  - Sehr generisch, muss konkret instanziiert werden für ein Projekt.

Rapid Application Development (RAD) (James Martin 1991)

- Definition
  - Prototyping orientiertes und iteratives Vorgehen auf der Basis von Joint Application Developement (JAD) und CASE-Tools.
  - Entwicklungszyklus:
    1. Requirements Planning Phase
    2. User Design Phase
    3. Construction Phase
    4. Cutover Phase



- Annahme, Paradigma
  - Baue zusammen mit dem Benutzer «rasche» Protoypen (GUI), bis er zufrieden ist ;-)

+ Vorteile
  - Benutzerorientierter und iterativ-inkrementeller Ansatz, der rasch zu benutzergerechten Lösungen führt.

– Nachteile
  - Sicherstellung der Software-Qualität und grosse Tool-abhängigkeit.

Kurze Denkpause

Aufgabe 1 (5’)

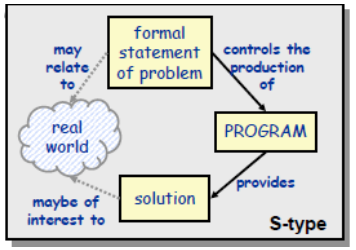
Diskutieren Sie in Murmelgruppen die Frage:

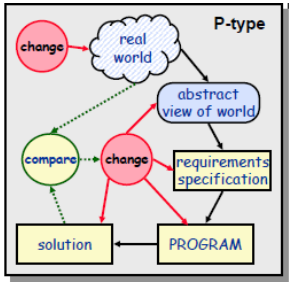
Für welche Softwareentwicklungs-Probleme eignet sich das Wasserfallmodell und für welche nicht?

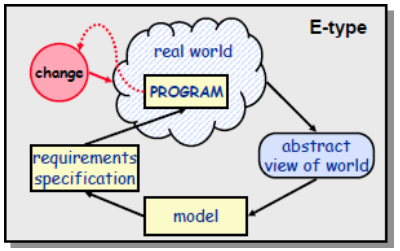
Drei Klassen von Software (Meir M. Lehman 1980)

- S-Typ: Vollständig durch formale Spezifikation beschreibbar (z.B. Sortieralgorithmus).
  - Erfolgskriterium: Spezifikation nachweislich erfüllt
- P-Typ: Löst ein abgegrenztes Problem (z.B. Schachprogramm).
  - Erfolgskriterium: Problem zufriedenstellend gelöst
- E-Typ: Realisiert eine in der realen Welt eingebettete Anwendung (z.B. CRM-Applikation).
  - Erfolgskriterium: Anwender zufrieden

- Software vom S-Typ ist stabil
- Software vom P- und E-Typ ist einer Evolution unterworfen durch den Wandel des Umfelds







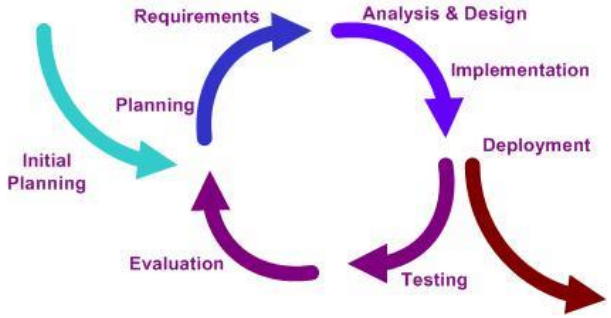
Quelle: adaptiert von Programs, Life Cycles, and Laws of Software Evolution, Meir M. Lehmann, 1980

Konsequenzen für P- und E-Software

- Bewegliche Ziele sind kein Unfall, sondern naturbedingt (unsere Projekte sind meistens vom E-Typ).
- Bei innovativen, neuen Produkten und Systemen müssen die Anforderungen zuerst gefunden werden – vielfach durch Prototyping und partielle Lösungen.
- Bei längeren Projekten sind Änderungen der Anforderungen während der Entwicklung wahrscheinlich.
- Software ist nie über mehrere Jahre hinweg gebrauchstauglich ohne Veränderung.
  - Wartung ist kein Unfall, sondern unvermeidlich.
  - Überlegungen zu Software immer auf die ganze Lebensdauer beziehen.

Iterativ-inkrementelle Modelle (ab den 90er Jahren mit OO)

- Definition
  - Software wird in mehreren geplanten und kontrolliert durchgeführten Iterationsschritten stückweise (inkrementell) entwickelt. Dabei kann der Gesamtumfang offen gestaltet werden.
- Annahme, Paradigma
  - Ziel und Gelände sind am Anfang unklar: Lenkwaffe.
- Vorteile
  - Flexibles Modell bei unklaren Anforderungen/Zielen.
  - Gutes Risikomanagement (Mitarbeiter und Technologie).
  - Frühe Einsetzbarkeit der Software und Feedback.
- Nachteile
  - Detaillierte «upfront» Planbarkeit hat Grenzen (Funktionalität, Zeit und Kosten).
  - Braucht eine Involvierung und Steuerung durch den Kunden über die ganze Projektdauer.



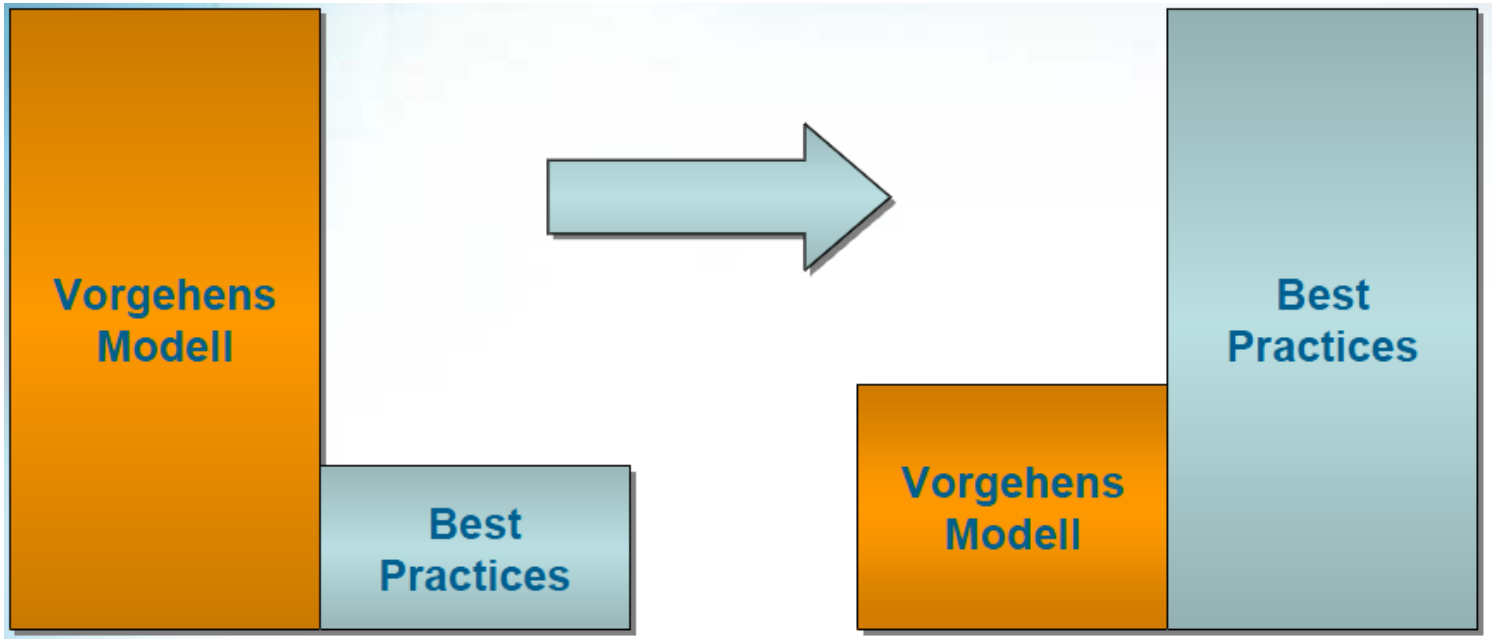
Agenda

1. Motivation und Zielsetzung
2. Software-Prozessmodelle
3. **Prinzipien agiler Softwareentwicklung**
4. Tailoring des Prozesses für ein Projekt
5. Wrap-up

Was ist agile Softwareentwicklung?

- Agile Softwareentwicklung ist keine einzige, umfassende Methode, sondern ist vielmehr eine Sammlung von Ideen (Werte, Prinzipien und Praktiken), um den Softwareentwicklungsprozess flexibler und schlanker zu machen, als das bei den klassischen Software-Prozessmodellen der Fall ist.

Schwergewichtige versus leichtgewichtige Software-Prozessmodelle



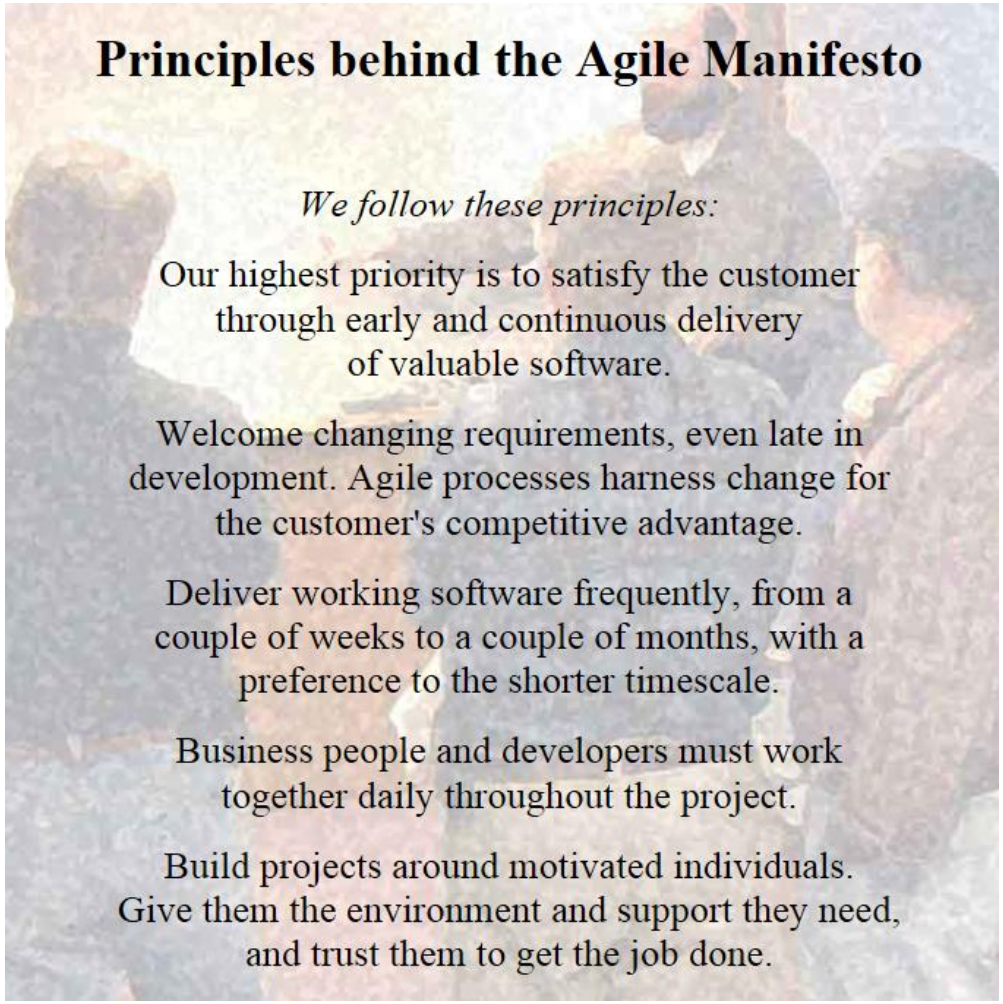
- Statt immer präziserer Vorgehensmodelle mit starren Vorschriften, einführen von agilen Prozessen mit wenigen essentiellen Aktivitäten und aus der Praxis bewährten Dingen
- Wissen in den Köpfen unserer Entwickler, Manager und Kunden als „Best Practices“

Agile Software-Entwicklung (ab 2000 mit XP u.a.)

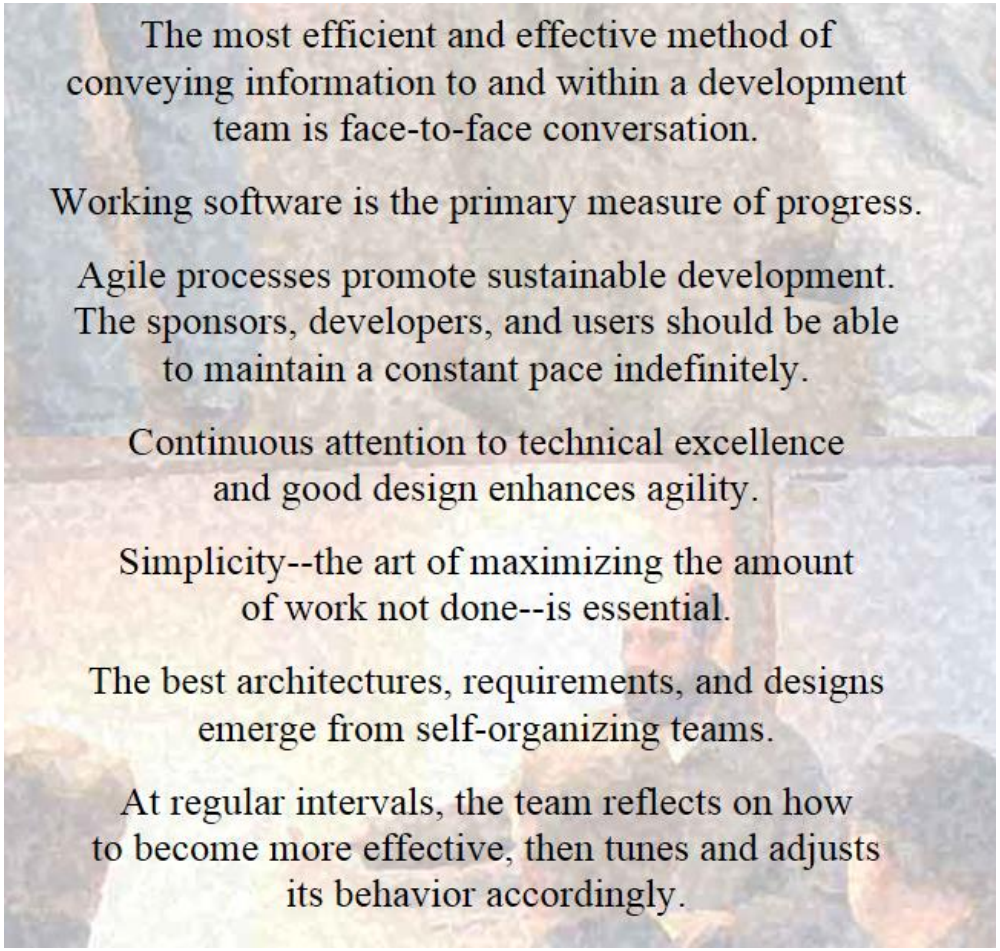
Die Architektur der agilen Prozessmodelle

Das Agile Manifest (www.agilemanifesto.org)

Das Agile Manifest (www.agilemanifesto.org)

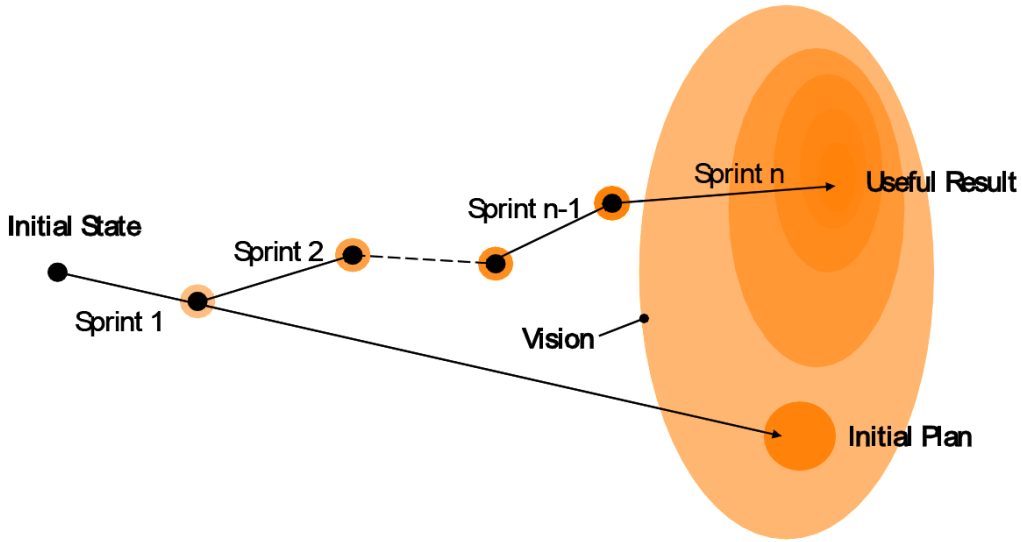


Das Agile Manifest (www.agilemanifesto.org)



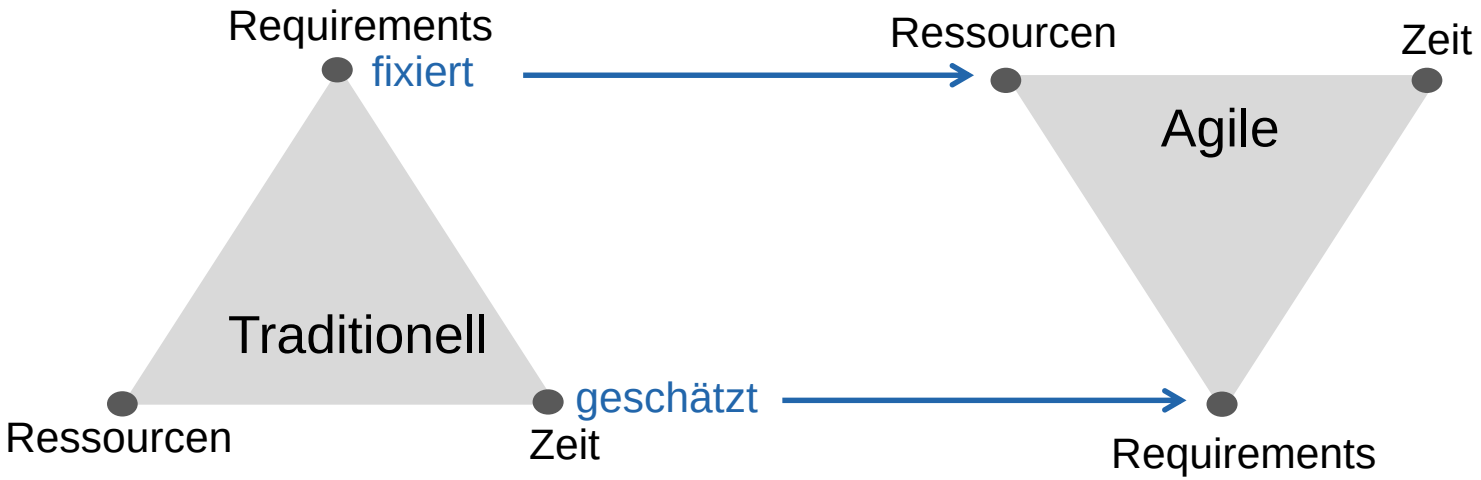
Strategie zur Prozesskontrolle in agilen Software-Prozessmodellen

- Definierte Prozesskontrolle (Klassisch, Plan-driven)
  - Merkmal: für einfache und völlig planbare Problemstellungen
  - Strategie: Steuerung
- Empirische Prozesskontrolle (Agil)
  - Merkmal: für komplexe, chaotische Problemstellungen
  - Strategie: Regelung , Deming-Cycle (Plan-Do-Check-Act)



Requirements Engineering agilen Software-Prozessmodellen

- Manifesto Prinzip 1: Unsere höchste Priorität ist es, den Kunden durch frühe und kontinuierliche Auslieferung wertvoller Software zufrieden zu stellen.
- Manifesto Prinizip 2: Heisse Anforderungsänderungen selbst spät in der Entwicklung willkommen. Agile Prozesse nutzen Veränderungen zum Wettbewerbsvorteil des Kunden.



Fix quality – deliver a small increment in a timebox – repeat.

Ein Assessment von Betrand Meyer (1/4)

(aus dem Buch: Agile! The Good, the Bad and the Ugly, 2014)

- The Bad and the Ugly
  - Deprecation of upfront tasks
  - User stories as basis for requirements
  - Feature-based development and ignorance of dependencies
  - Rejection of dependency tracking tools
  - Rejection of traditional manager tasks
  - Rejection of upfront generalization
  - Embedded customer
  - Coach as a separate role
  - Test-driven development
  - Deprecation of documents

Ein Assessment von Betrand Meyer (2/4)

(aus dem Buch: Agile! The Good, the Bad and the Ugly, 2014)

- The Hyped
  - Pair programming gut um Juniors/Neue einzubeiten
  - Open-space working arrangements
  - Self-organizing teams
  - Working at sutainable pace
  - Producing minimal functionality
  - Planning game, planning poker
  - Members and observers
  - Collective code ownership
  - Cross-functional teams

Ein Assessment von Betrand Meyer (3/4)

(aus dem Buch: Agile! The Good, the Bad and the Ugly, 2014)

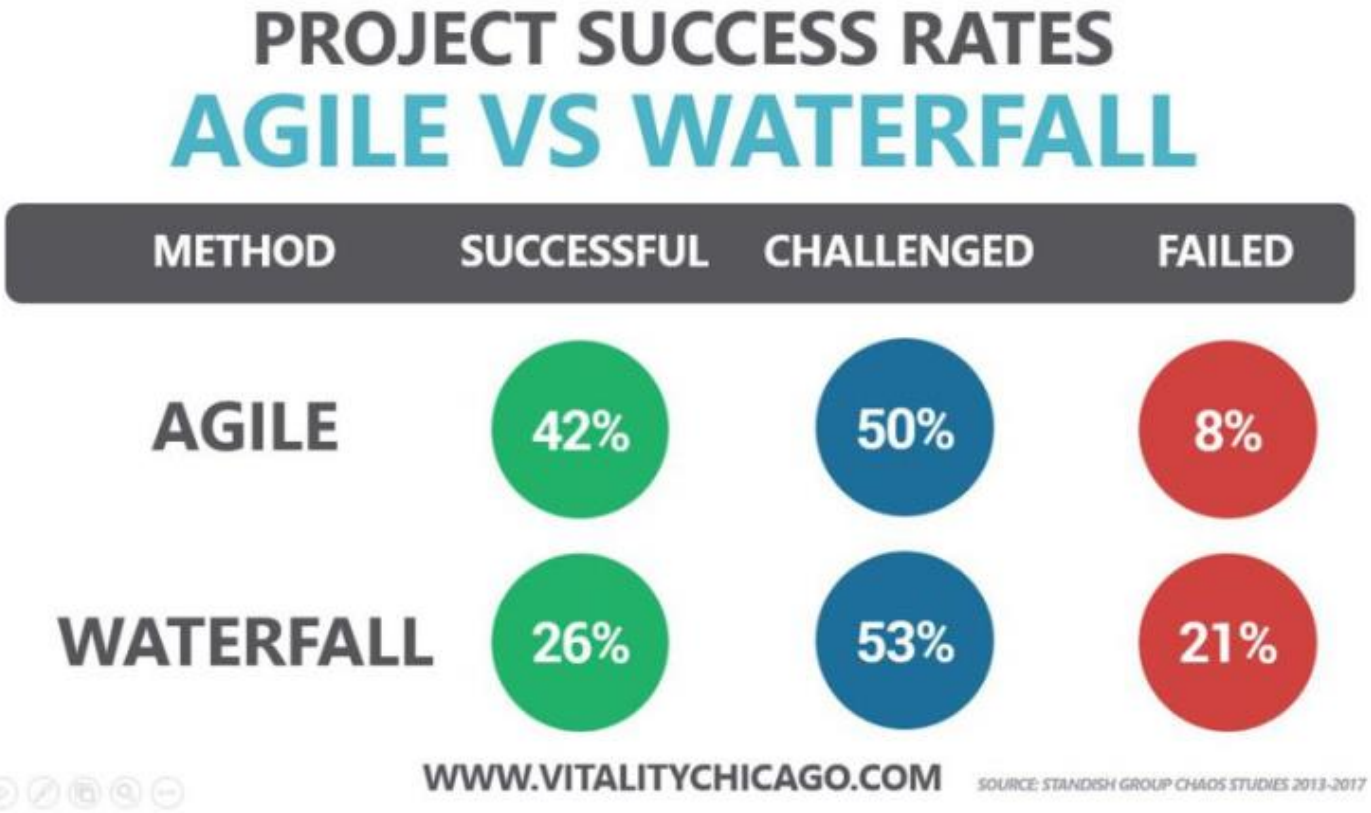
- The Good
  - Promoting refactoring is an important contribution of the agile approach, particularly of XP
  - Short daily meetings focused on simple verbal reports to progress - the “three questions” - are an excellent idea
  - Agile methods rightly insist on the importance of team communication to the success of projects
  - The practice of identifying and removing impediments, in particular as a focus of progress meetings, is a powerful agile insight
  - Lean’s identification of sources of waste in software development and insistence on removing them

Ein Assessment von Betrand Meyer (4/4)

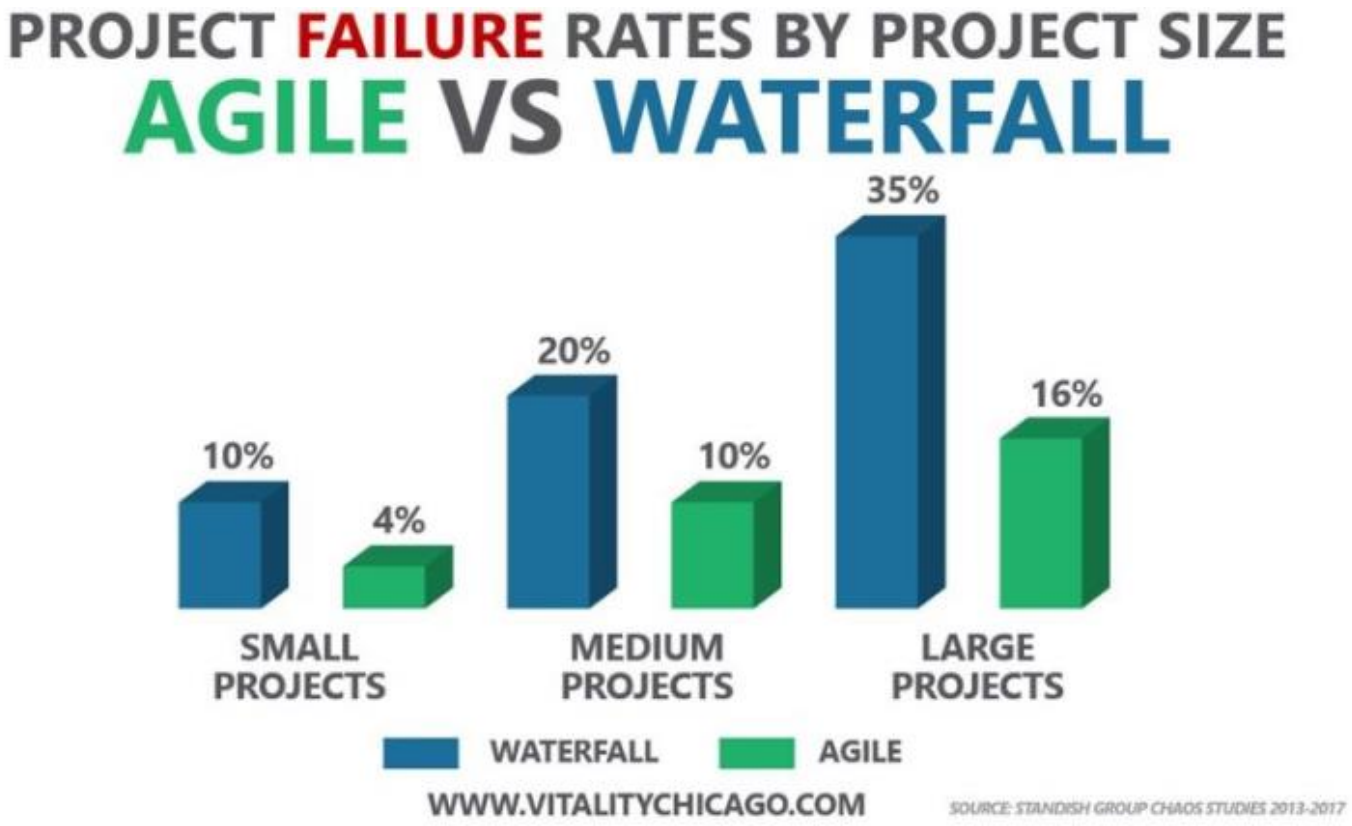
(aus dem Buch: Agile! The Good, the Bad and the Ugly, 2014)

- The Brilliant
  - Short iterations are perhaps the most visible influence of agile ideas
  - The related practice of continuous integration and the associated regression test suite artifact
  - The closed-window rule during an iteration
  - Time-boxing every iteration - not accepting any delays, even if some functionality has not been implemented
  - Scrum introduced the beneficial notion of a clearly defined product owner
  - The emphasis on delivering working software is another important contribution
  - The notion of velocity and the associated artifact of task boards to provide visible, constantly updated evidence of progress or lack thereof are practical
  - Associating a test with every piece of functionality is a fundamental rule which contributes significantly to the solidity of a software project

Hat die agile Softwareentwicklung Verbesserungen gebracht? (1/2)



Hat die agile Softwareentwicklung Verbesserungen gebracht? (2/2)



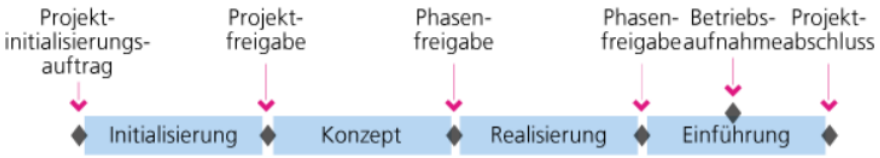
Agenda

1. Motivation und Zielsetzung
2. Software-Prozessmodelle
3. Prinzipien agiler Softwareentwicklung
4. **Tailoring des Prozesses für ein Projekt**
5. Wrap-up

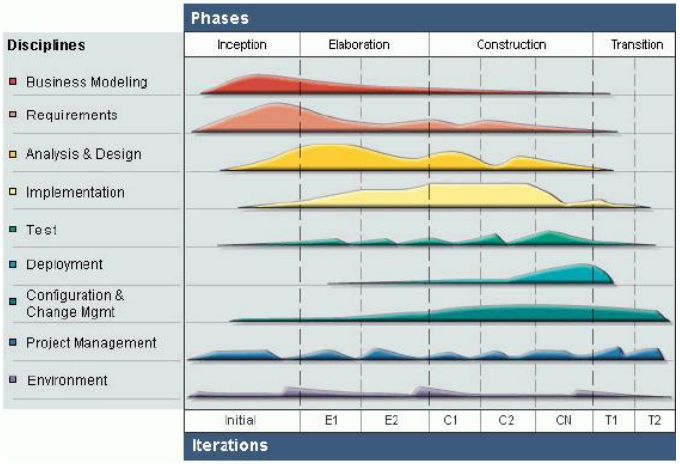
Charakterisierung von verbreiteten Software-Prozessmodellen

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering

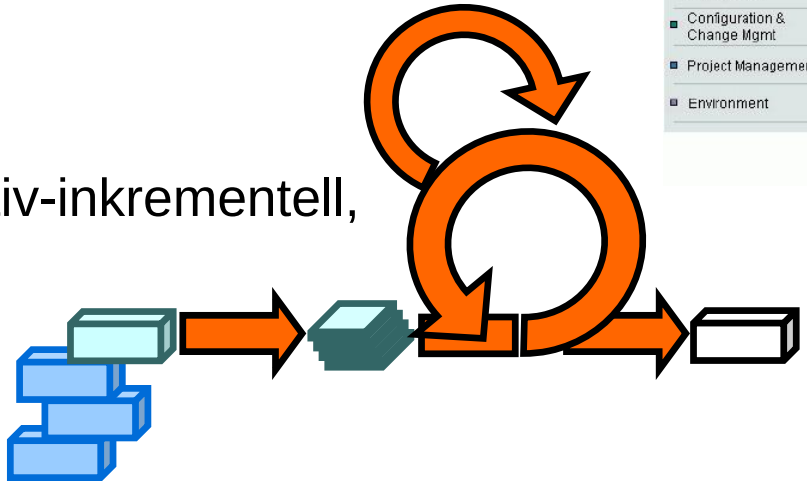
Hermes: Wasserfall/V-Modell (historisch), definiert oder empirisch (Tailoring)



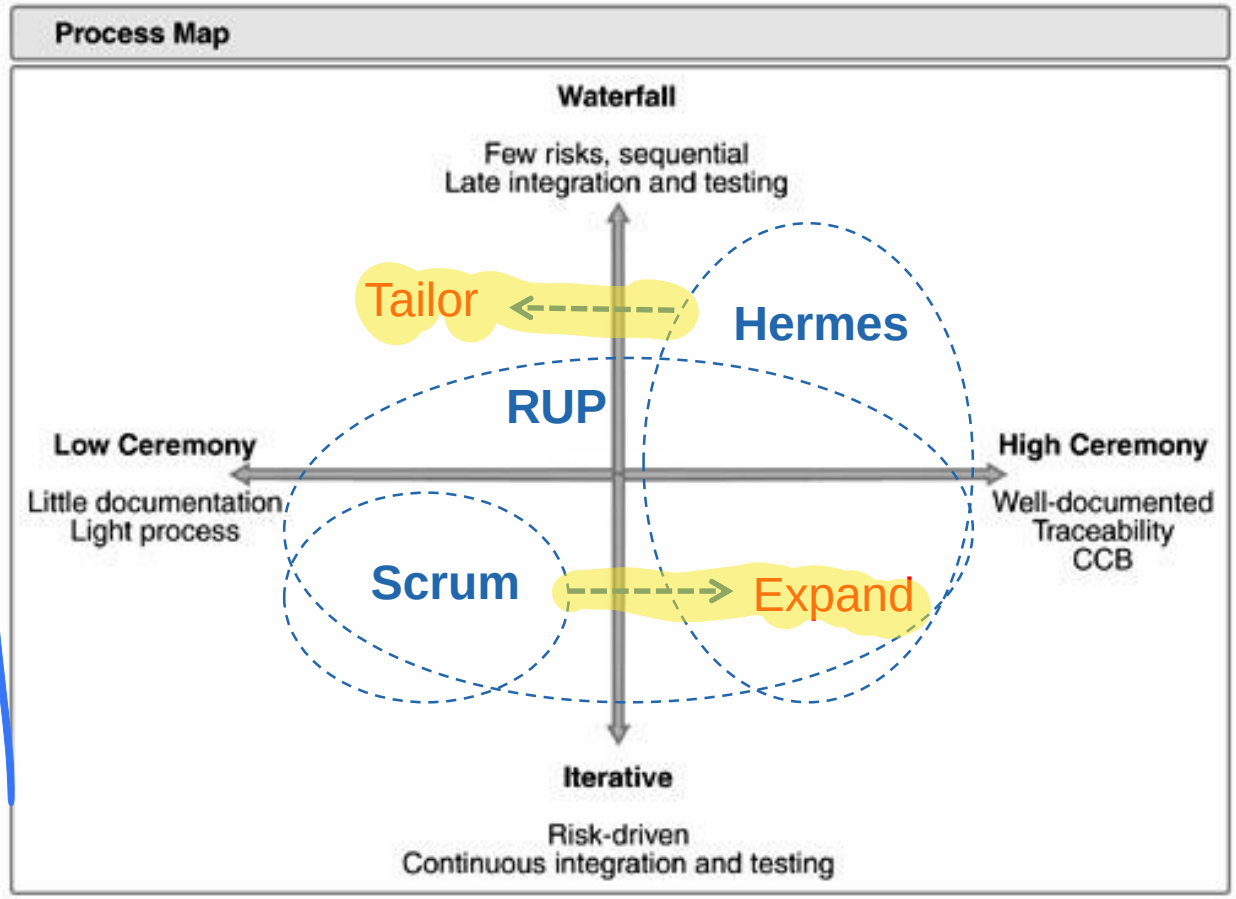
RUP: iterativ-inkrementell, definiert oder empirisch (Tailoring)



Scrum: iterativ-inkrementell, empirisch



Wie können die Software-Prozessmodelle verglichen werden?



Home Ground des Software-Prozessmodells

Quelle: The Rational Unified Process Made Easy, Per Kroll/Philippe Kruchten, 2003

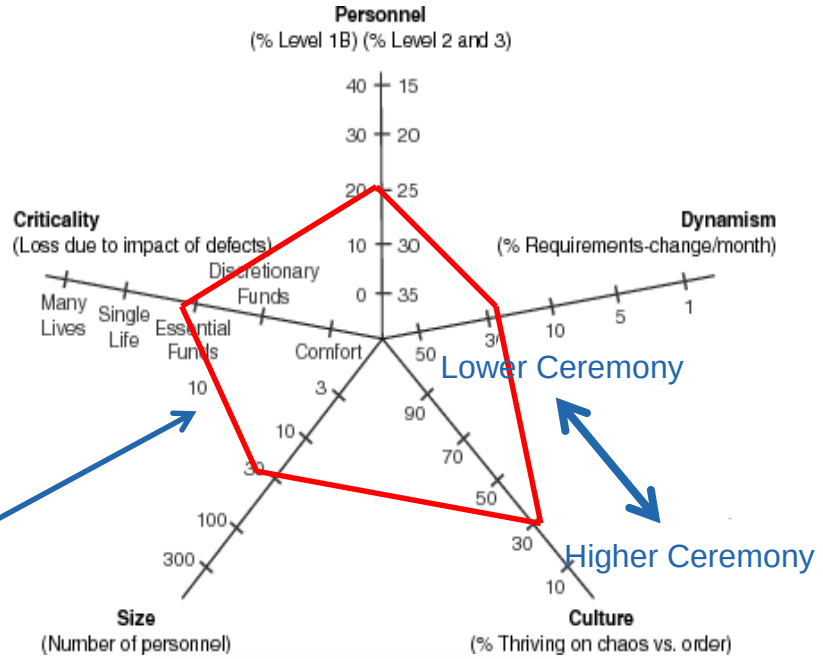
To be or not to be Agile – is that really the question?

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering

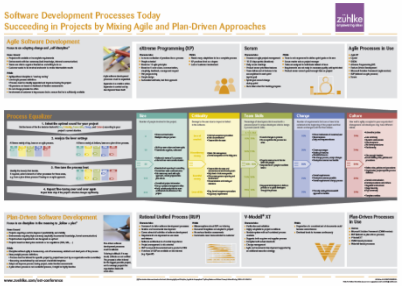
Fünf wesentliche Faktoren

- Teamstruktur (Personnel)
- Dynamik der Anforderungen (Dynamism)
- Entwicklungskultur (Culture)
- Teamgrösse (Size)
- Kritikalität (Criticality)

Quelle: Balancing Agility and Discipline, Barry Boehm/Richard Turner, 2004



Das Projektprofil (Problem) bestimmt das Tailoring!



siehe Zühlke Poster 2008

The Top Six Conclusions von Böhm/Turner

(aus dem Buch: Balancing Agility and Discipline, 2003)

1. Neither agile nor plan-driven methods provide a silver bullet.
2. Agile and plan-driven methods have home grounds where one clearly dominates the other.
3. Future trends are toward application developments that need both agility and discipline.
4. Some balanced methods are emerging.
5. It is better to build your method up than tailor it down.
6. Methods are important, but potential silver bullets are more likely to be found in areas dealing with peoples, values, communication, and expectations management.

Agenda

1. Motivation und Zielsetzung
2. Software-Prozessmodelle
3. Prinzipien agiler Softwareentwicklung
4. Tailoring des Prozesses für ein Projekt
5. **Wrap-up**

Wrap-up

- Die meisten Software-Entwicklungen heute sind komplexe Probleme (Zielvielfalt, Dynamik, Intransparenz, Rückkopplung), die eine problemangepasste Lösungsstrategie verlangen, damit sie erfolgreich sind.
- Ein Projekt sollte auf der Basis seiner Charakteristiken so agil wie möglich abgewickelt werden, aber nicht agiler!
- Ein Software-Prozessmodell – und insbesondere auch das RE - muss für die Problemstellung bzw. das Projekt angepasst werden (Tailoring).
- Für das Tailoring eines Prozesses ist ein risikobasierter Ansatz sinnvoll, der Kriterien wie Grösse, Kritikalität, Team Skills, Dynamik, Kultur und weitere problemspezifische Aspekte des Projektes berücksichtigt.