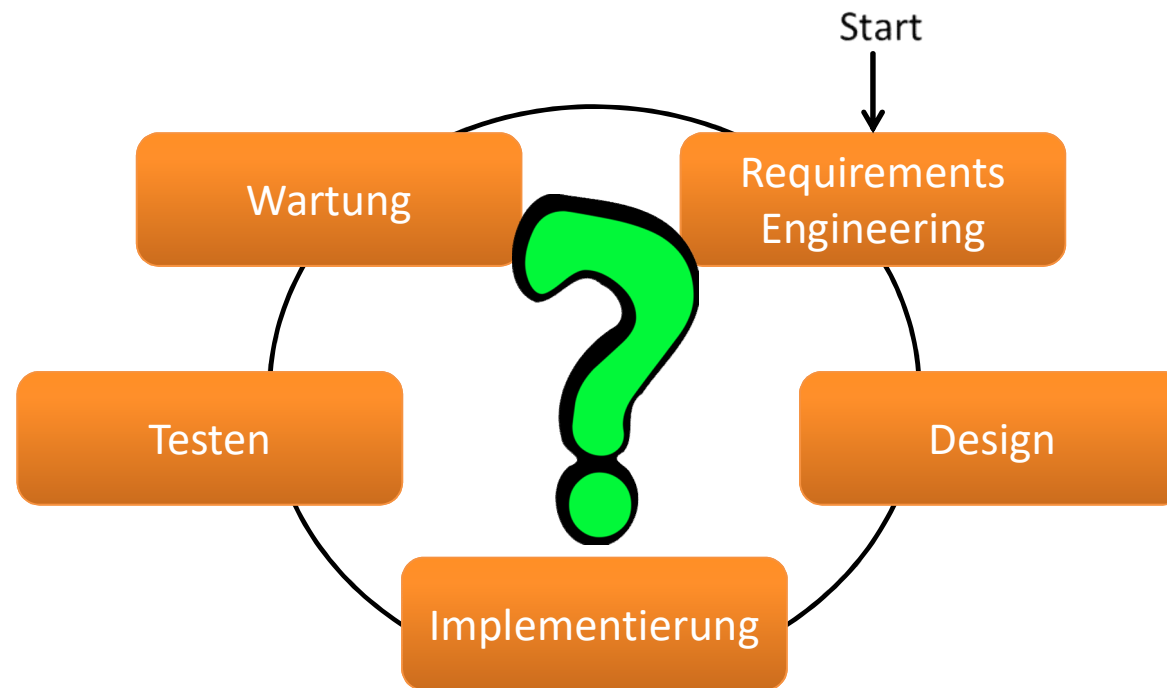


Softwareentwicklungsprozesse

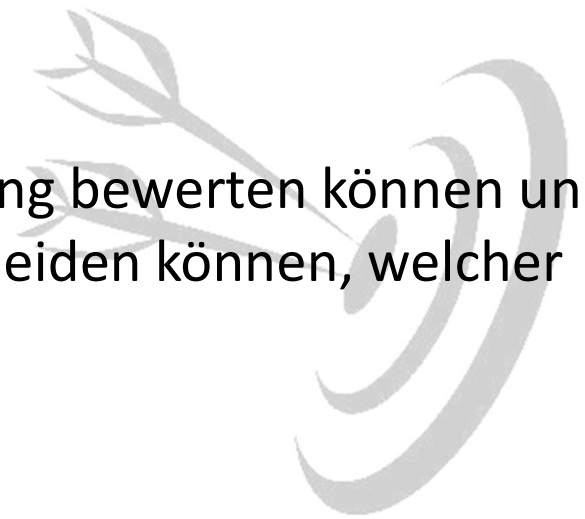
Authors of slides:
Norbert Siegmund
Janet Siegmund
Oscar Nierstrasz
Sven Apel

Einordnung



Lernziele

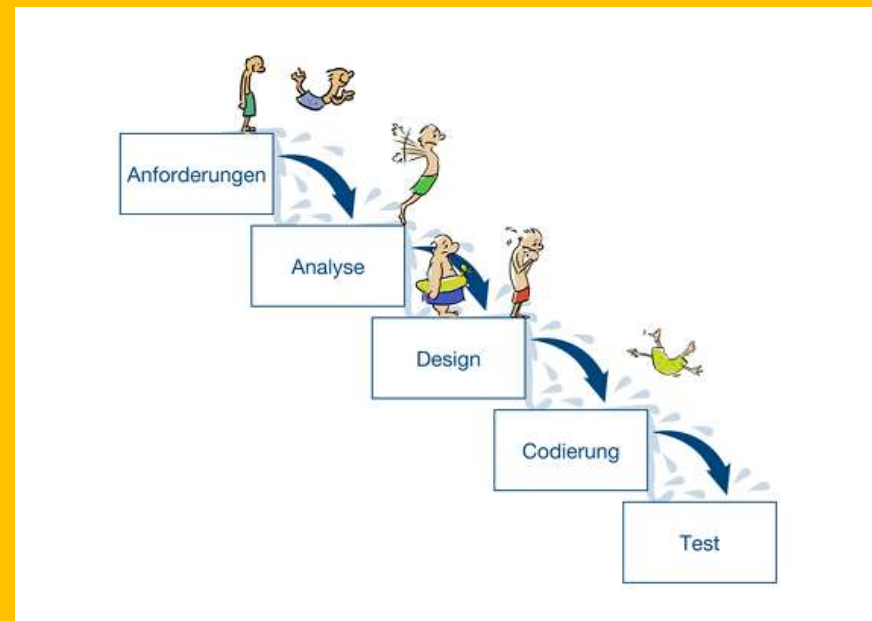
- Überblick und grundlegendes Verständnis für Entwicklungsprozesse haben
- Traditionelle Softwareentwicklungsprozesse (sequenzielle Modelle) kennen und deren Probleme verstehen
- Alternative, neue Ansätze der Softwareentwicklung bewerten können und abhängig von der Projekt- / Betriebsgröße entscheiden können, welcher Ansatz zu präferieren ist



Vorgehensmodelle

- Beschreiben *wie*, *wann*, *welche* der Tätigkeiten der Softwareentwicklung (Phasen des Lebenszyklus) ausgeführt werden
- Bisher: Brute-Force-Modell: Einfach drauf losprogrammieren
- Jetzt:
 - Traditionelle Entwicklungsmodelle
 - Große Teams, feste Anforderungen
 - Neuartige Modelle (agile Methoden)
 - Angepasst auf schnelle Entwicklungsphasen, kurze Release-Zeiten und sich ständig ändernde Anforderungen

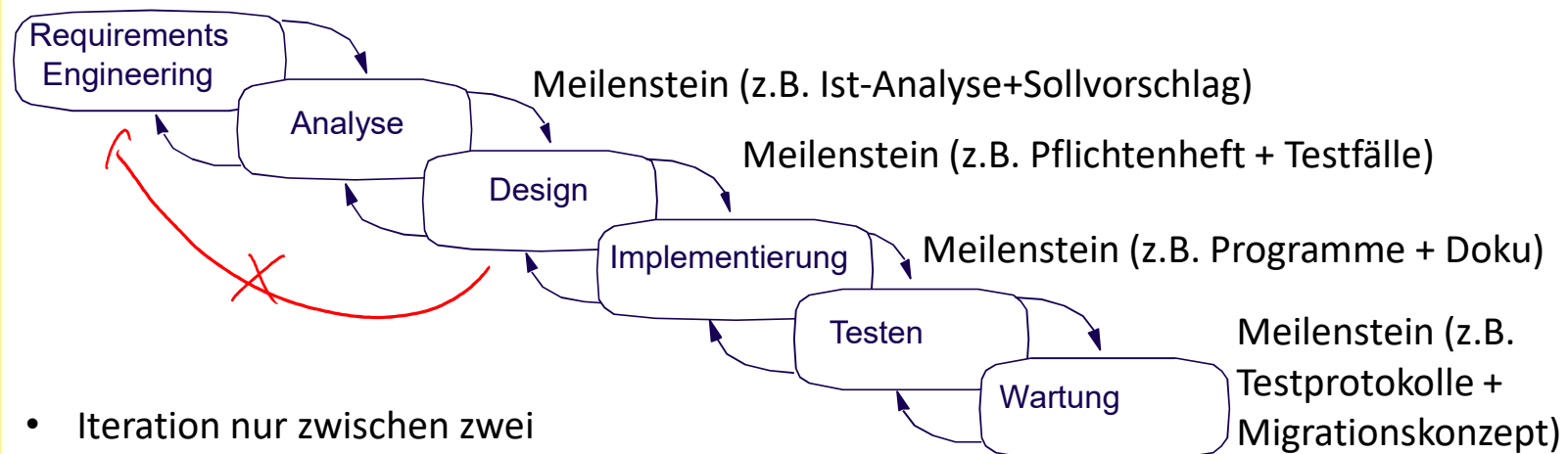
Sequentielle Modelle



Grundidee von Seq. Modellen

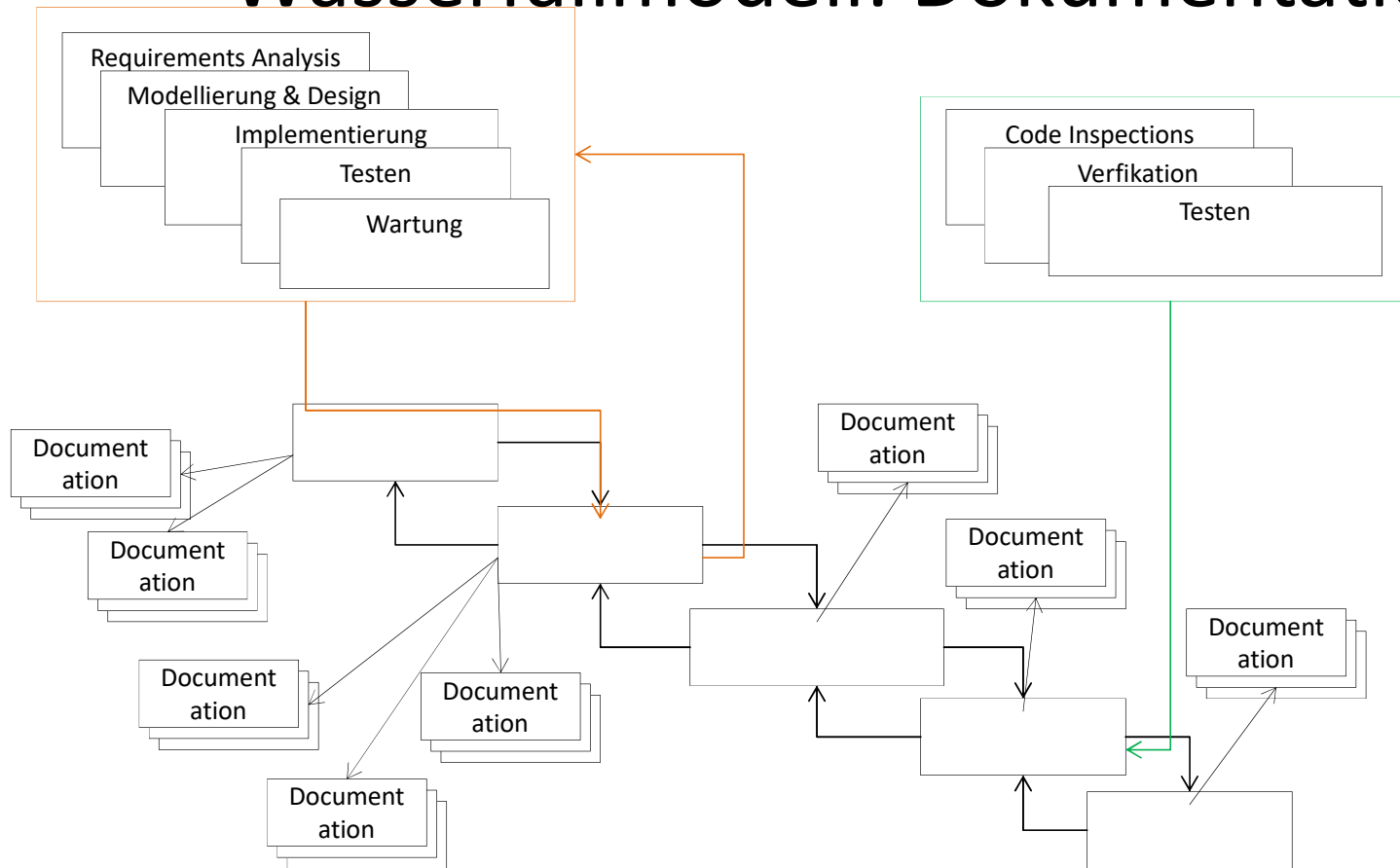
- Schritt für Schritt werden alle Phasen des Lebenszyklus nacheinander „abgearbeitet“
 - Planen / Konzipieren
 - Entwerfen / Ausarbeiten
 - Inbetriebnehmen / Warten
- Modelle:
 - Wasserfallmodell
 - V-Modell

Wasserfallmodell



- Iteration nur zwischen zwei aufeinanderfolgenden Schritten
- Abgeschlossener Schritt als sicherer Rückgriff

Wasserfallmodell: Dokumentation



Wasserfallmodell: Diskussion

- Vorteile:

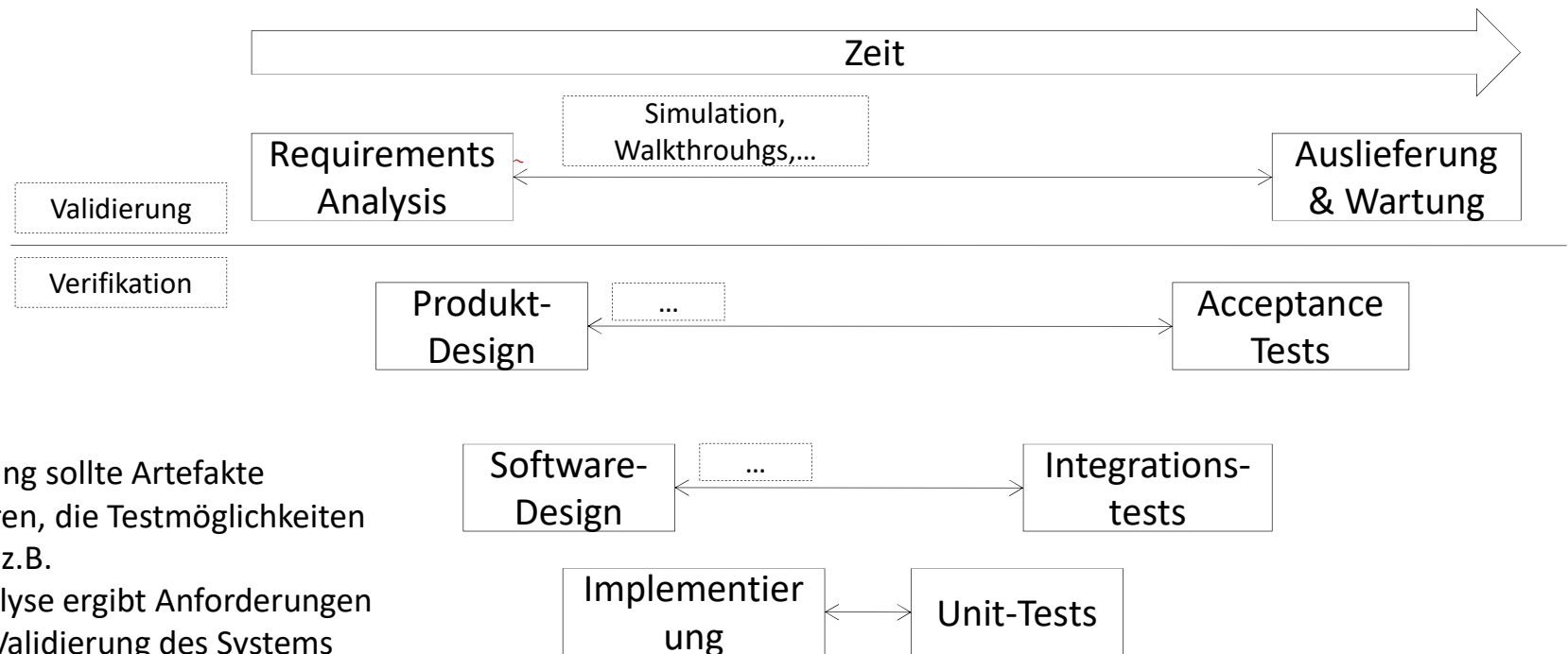
- **Dokumentation** nach jeder Phase verfügbar
- Klare **Trennung** der Phasen und Verantwortlichkeiten
- Analog zu Ingenieurprojekten (Brückenbau etc.)

Meist unrealistisch in der Praxis!

- Nachteile:

- **Starres** Vorgehen
- Reaktionen auf **geänderte** Anforderungen schwierig
- Anforderungen, Design, etc. **früh fixiert**, Änderungen nicht vorgesehen (aber Änderungen sind natürlich!)
- **Späte** Qualitätsprüfung (Baue ich überhaupt das **richtige Produkt?** Erster **Prototyp sehr spät** verfügbar!)

^{& V}V-Modell



Entwicklung sollte Artefakte produzieren, die Testmöglichkeiten schaffen, z.B.

- Analyse ergibt Anforderungen
➔ Validierung des Systems
- Design ergibt Grobspezifikation
➔ Verifikation der Komponenteninteraktion
- Feinentwurf ergibt Feinspezifikation
➔ Verifikation der Komponenten (Testen, usw.)

Artefakte vom V-Modell

- Anwenderanforderungen: Lastenheft
- Technische Anforderungen: Grobentwurf/Pflichtenheft
- SW-Architektur: Feinspezifikation
- SW-Entwurf: Codedokumentation (z.B. javadoc)
- Implementierungsdoku: Installationsanleitung
- Prüfprozedur und Prüfergebnis: Abnahmedokumente

Probleme sequentieller Modelle

- Anforderungen oft nicht klar und können sich ändern
- Bei großen Projekten dauert Entwicklung lange und es fehlt Erfolgskontrolle
- Erfordert viel Dokumentation → Overhead für kleine Projekte
- Gefahr von Missverständnissen
 - Prototyp erst am Ende des Projektes
- Je grundlegender ein Fehler, umso später wird er gefunden

Teilweise Lösung: Prototypen

Ein **Prototyp** ist ein Softwareprogramm, entwickelt, um zu testen, explorieren und validieren von Hypothesen (Reduzierung von Risiken).

Ein **explorierender Prototyp**, auch bekannt als **Wegwerfprototyp**, zielt auf die **Validierung von Anforderungen** oder **Erprobung von Designentscheidungen** ab.

- UI prototyp — validiert Nutzeranforderungen
- Rapid prototype — validiert funktionale Anforderungen
- Experimental prototype — validiert technische Machbarkeit



Weitere Arten von Prototypen

Ein evolutionärer Prototyp ist dafür gedacht sich zu entwickeln, so dass er in Schritten zum finalen Produkt ausreift.

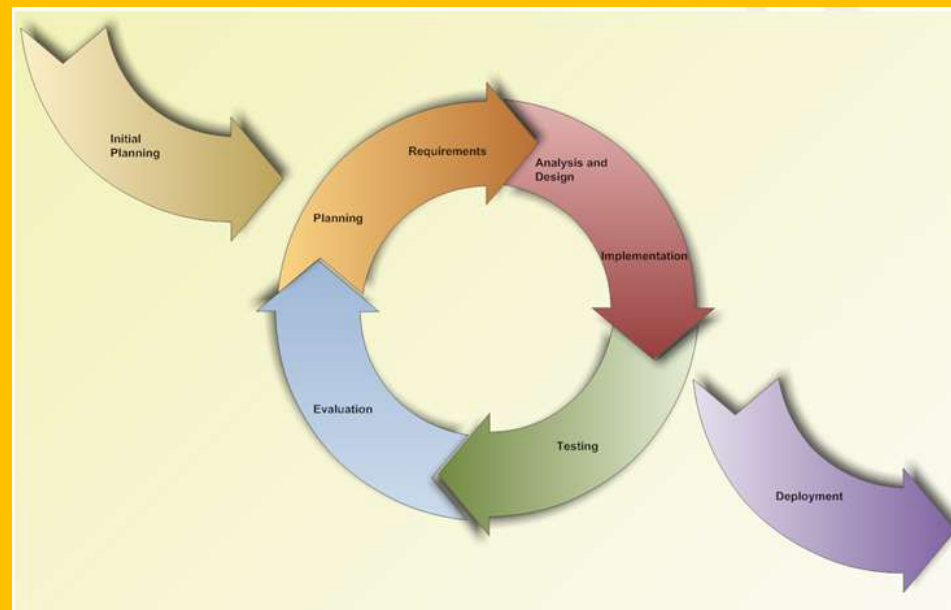
- Beim iterativen “Wachsen” der Anwendung ist **Redesign** und **Refactoring** ständiger Begleiter.

First do it,
then do it right,
then do it fast.

Horizontaler Prototyp wird für die Ebene mit dem größten Risiko erstellt (reduziert Missverständnisse) und realisiert alle Funktionen einer Ebene (gut, um Beziehungen zwischen Funktionen zu erkennen).

Vertikaler Prototyp setzt Kernfunktionalität um (Machbarkeits- / Effizienztest, Aufwandsabschätzung) und dient als Pilotsystem (gut, um eine komplexe Funktion besser zu verstehen).

Iterative Modelle



Idee von iterativen Modellen

- Lat. iterare: wiederholen
- Idee:
 - Vollständiges Analysieren und Planen ist a priori unmöglich, daher iterative “Annäherung” an das Ziel
 - Erkenntnisse aus jedem Iterationsschritt werden benutzt, so lange, bis definiertes Ziel erreicht ist
- Beim ersten Mal macht man typischerweise Fehler
- Darum integrieren, validieren und testen so oft wie möglich

Prototyp vs. Inkrementelle Entwicklung

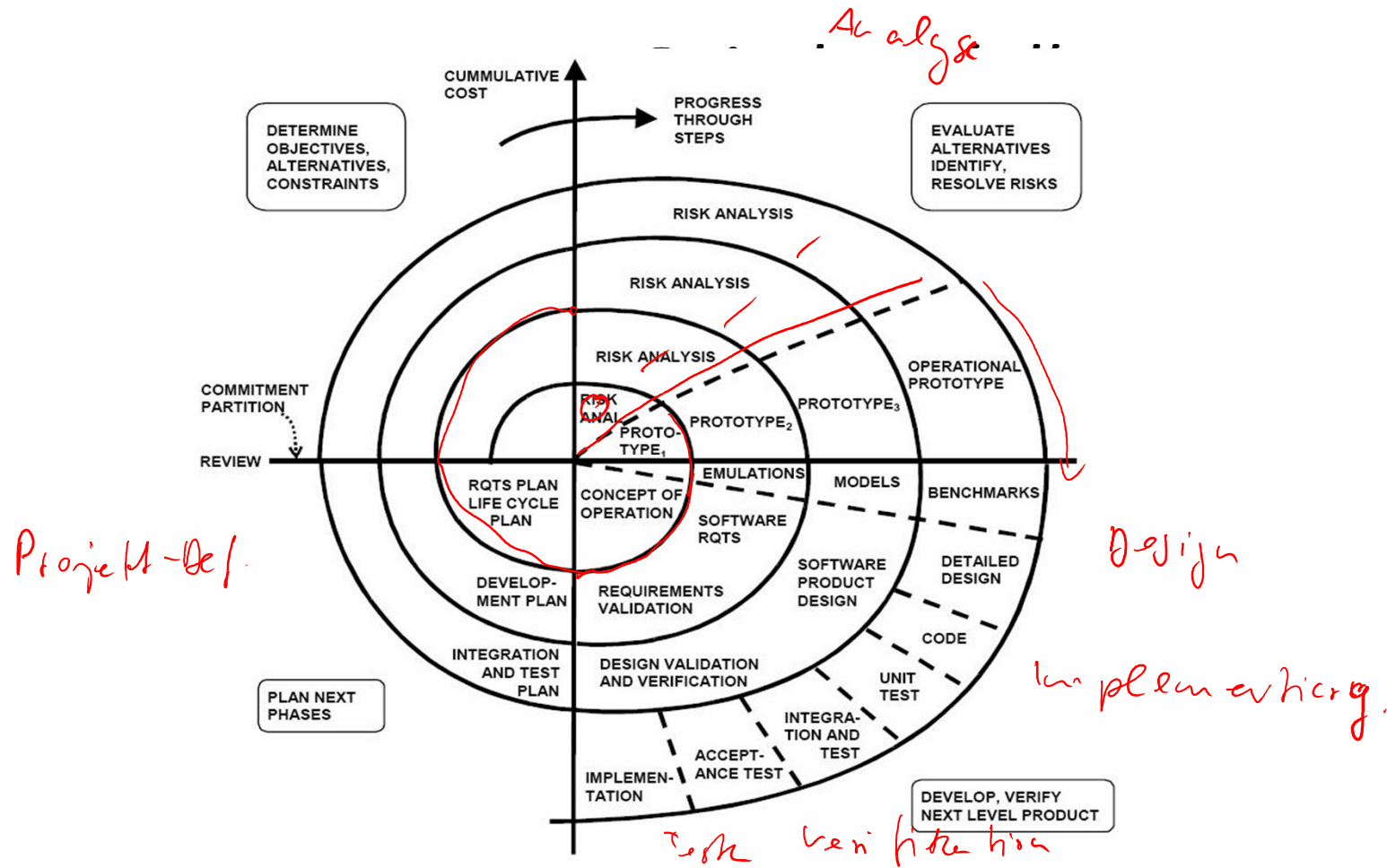
- Prototypen werden oft weggeworfen (da keine Qualitätskriterien eingehalten wurden, sondern nur zum Zeigen entwickelt wurden)
- Inkrement:
 - Software-Baustein(e), die zu einem existierenden System oder Subsystem hinzugefügt werden, um dessen Funktionalität oder Leistung zu vergrößern oder zu verändern
 - Inkrement kann Subsystem entsprechen
 - Inkrementelle Systementwicklung: beginnt mit Kernsystem, schrittweise Erweiterung

Hinweise für inkrementelle Entwicklung

- Falls möglich, habe **immer eine laufende Version des Systems**, selbst, wenn der größte Teil der Funktionalität nicht implementiert ist
- **Integriere** neue Funktionalität so früh wie möglich
- **Validiere** inkrementelle Versionen gegen Nutzeranforderungen

Spiralmodell

- Risiko-orientiertes Vorgehen
 - Suche alle Risiken, von denen das Projekt bedroht ist. Wenn es keine gibt, ist das Projekt erfolgreich abgeschlossen
 - Bewerte die erkannten Risiken, um das größte zu identifizieren
 - Suche einen Weg, um das größte Risiko zu beseitigen, und gehe diesen Weg. Wenn sich das größte Risiko nicht beseitigen lässt, ist das Projekt gescheitert
- Generisches Modell (kann bspw. auch zu Wasserfallmodell werden)



Spiralmodell

- Verbesserung des Wasserfallmodells
 - Gleiche Aufgaben wie im Wasserfallmodell
 - Jede Aufgabe wird durch Prototypen abgeschlossen
 - Fortschritt kann besser kommuniziert werden
 - Risikoanalyse erlaubt frühe Erkennung von Risiken
- War erfolgreich im Einsatz
 - Microsoft
 - IBM
 - US Militär (future combat system)

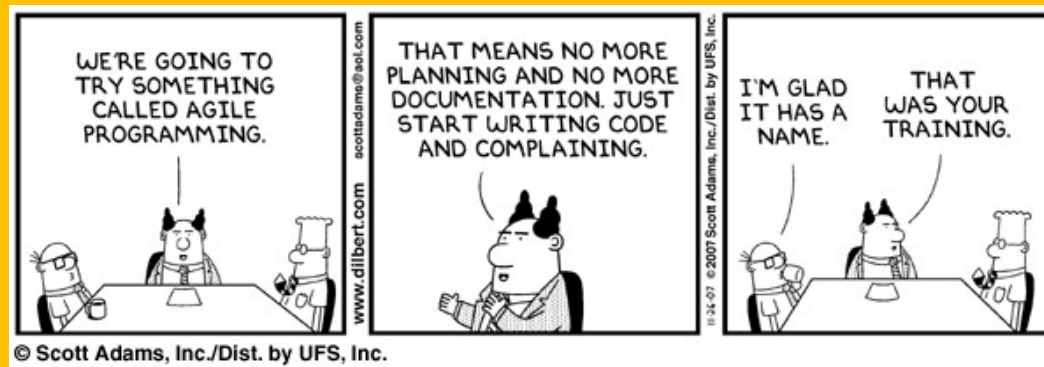
Weitere Modelle

- Unified Process (UP)
 - Inkrementelle Implementierung der funktionalen Anforderungen (wichtigste zuerst)
 - Kurze Iterationen (Wochen)
 - Jede Iteration endet mit vollständig laufendem System
- Evolutionäres Modell
 - Kunde bekommt früh Vorab-Version (erfordert Einbindung des Kunden, schwierige Gesamtplanung)
 - Ermöglicht früh Return-on-Investment

Aufgabe

- Welcher Entwicklungsprozess würde sich für NoMoreWaiting eignen?
 - Wasserfallmodell
 - V-Modell
 - Prototypen
 - Spiralmodell
 - Unified Process

Agile Softwareentwicklung



Agile Softwareentwicklung

- Relativ neuer Ansatz (ca. 1999)
- Im Spannungsfeld zwischen:
 - Qualität, Kosten und Zeit
 - Ungenauen Kundenwünschen und instabilen Anforderungen
 - Langen Entwicklungszeiten und überzogenen Terminen
 - Unzureichender Qualität
- Gegenbewegung zu schwergewichtigen, bürokratischen iterativen Prozessen, die oft zu viel Dokumentation erfordern

Manifesto für Agile Software Entwicklung

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.



Kent Beck

Ward Cunningham

Andrew Hunt

Robert C. Martin

Dave Thomas

Mike Beedle

Martin Fowler

Ben Jeffries

Steve Meller

James Grenning

Arie van Bennekum

Jim Highsmith

Jon Kern

Ken Schwaber

Jeff Sutherland

Alistair Cockburn

Brian Marick

[Agilemanifesto.org]

Manifest der agilen Softwareentwicklung

- Menschen und Kooperation vor Werkzeugen und (automatisierten) Prozessen
- Funktionsfähige Software vor umfassender Dokumentation
- Zusammenarbeit mit Kunden vor bürokratischen Vertragsverhandlungen
- Dynamische Reaktion auf Veränderungen vor statischer Planeinhaltung
- (trotzdem sind Prozesse, Dokumentation, ... vorhanden und wichtig)

Was ist Agile Softwareentwicklung?

- Menge von Softwareentwicklungsmethoden
 - Basierend auf iterativer und inkrementeller Entwicklung
- Meist kleine Gruppen (6-8)
- Kunde ist in Projekt integriert

Schwergewichtige Prozesse	Agile Prozesse
Dokumentenzentriert	→ Codezentriert
Up-Front Design	→ Minimale Analyse zu Beginn
Reglementiert	→ Adaptiv, Prozess wird angepasst
Abarbeitung eines Plans	→ Ständige Anpassung der Ziele
Lange Releasezyklen	→ Häufiges Deployment

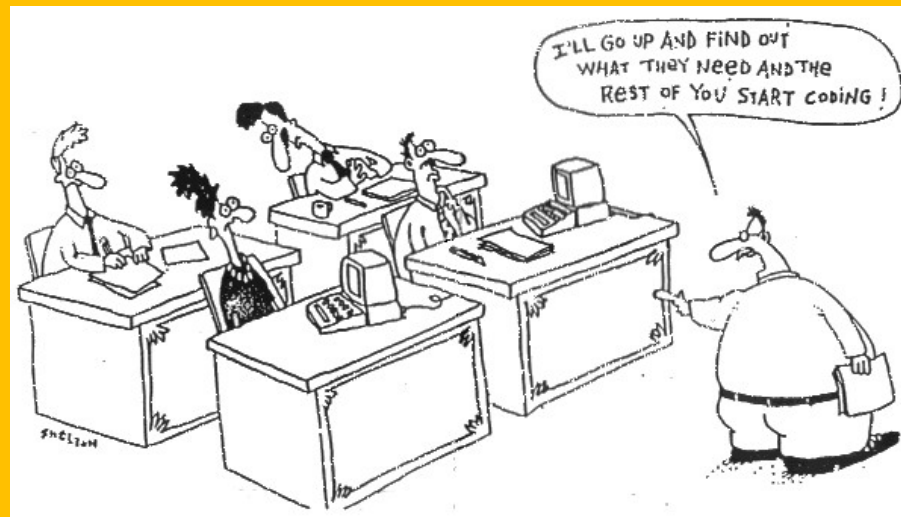
Die 12 agilen Prinzipien I

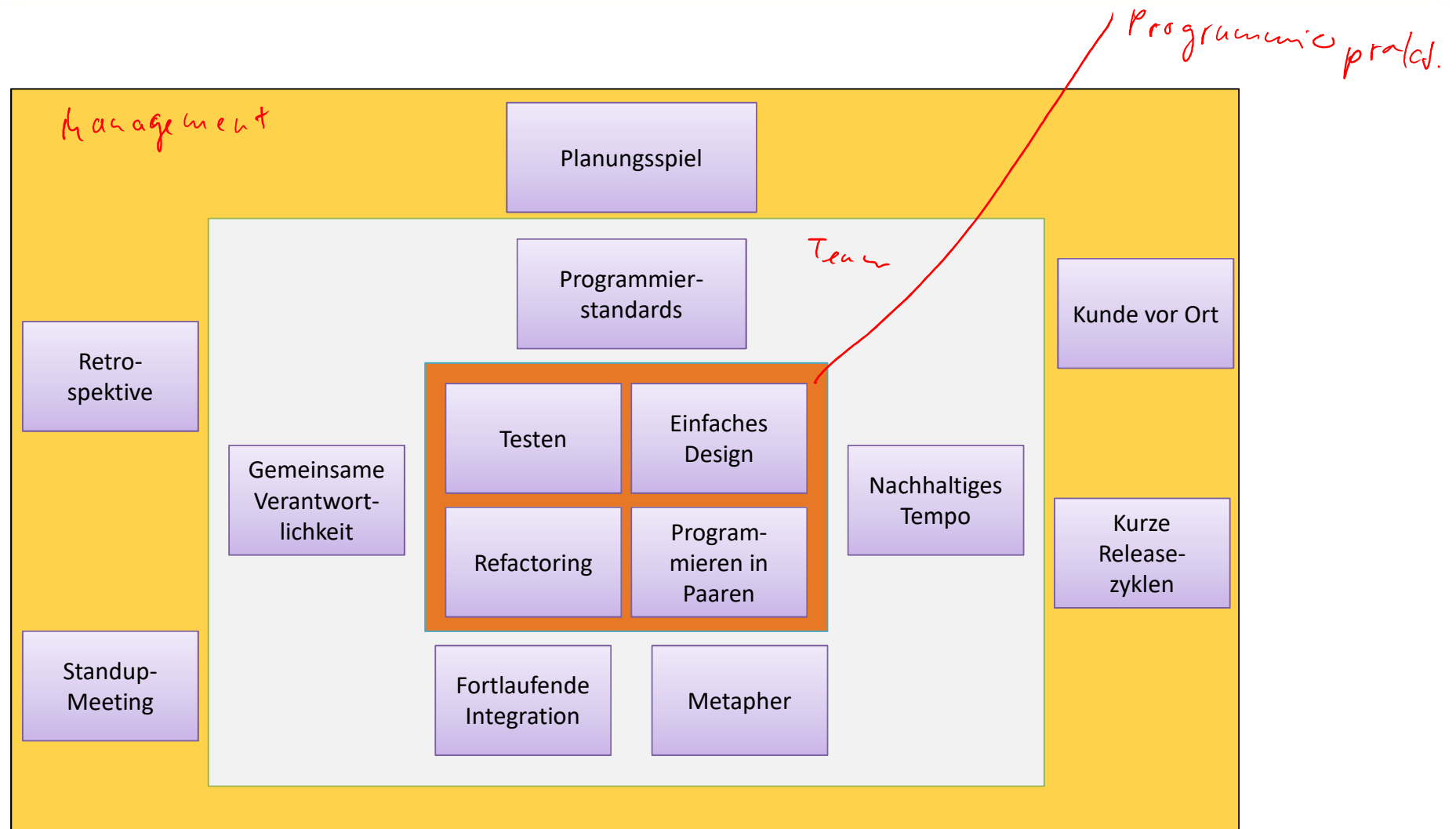
1. Kundenzufriedenheit durch frühzeitige und kontinuierliche Auslieferung wertvoller Software!
2. Änderungen begrüßen, selbst wenn sie spät in der Entwicklung kommen. Agile Prozesse nutzen Änderungen zugunsten des Wettbewerbsvorteils des Kunden.
3. Liefere funktionierende Software häufig (zw. wenigen Wochen und Monaten) aus.
4. Tägliche Zusammenarbeit von Kunden und Entwicklern.
5. Baue Projekte mit motivierten Mitarbeitern und gib ihnen die Umgebung und Unterstützung, die sie benötigen und vertraue ihnen, dass sie erfolgreich ihre Arbeit beenden.
6. Konversation von Angesicht zu Angesicht ist die effektivste und effizienteste Methode der Kommunikation!

Die 12 agilen Prinzipien II

7. Das primäre Maß für Fortschritt ist funktionierende Software.
8. Agile Prozesse bieten Kontinuität in der Entwicklung, so dass Investoren, Entwickler und Anwender ein beständiges Tempo aufrecht erhalten können.
9. Ständige Aufmerksamkeit gegenüber technisch hervorragender Qualität und gutem Design erhöht Agilität.
10. Einfachheit – die Kunst, unnötige Arbeit zu minimieren – ist essentiell.
11. Die besten Architekturen, Anforderungen und Designs stammen aus sich selbst organisierenden Teams.
12. Regelmäßig evaluiert das Team wie es noch effizienter arbeiten kann und passt sich entsprechend an.

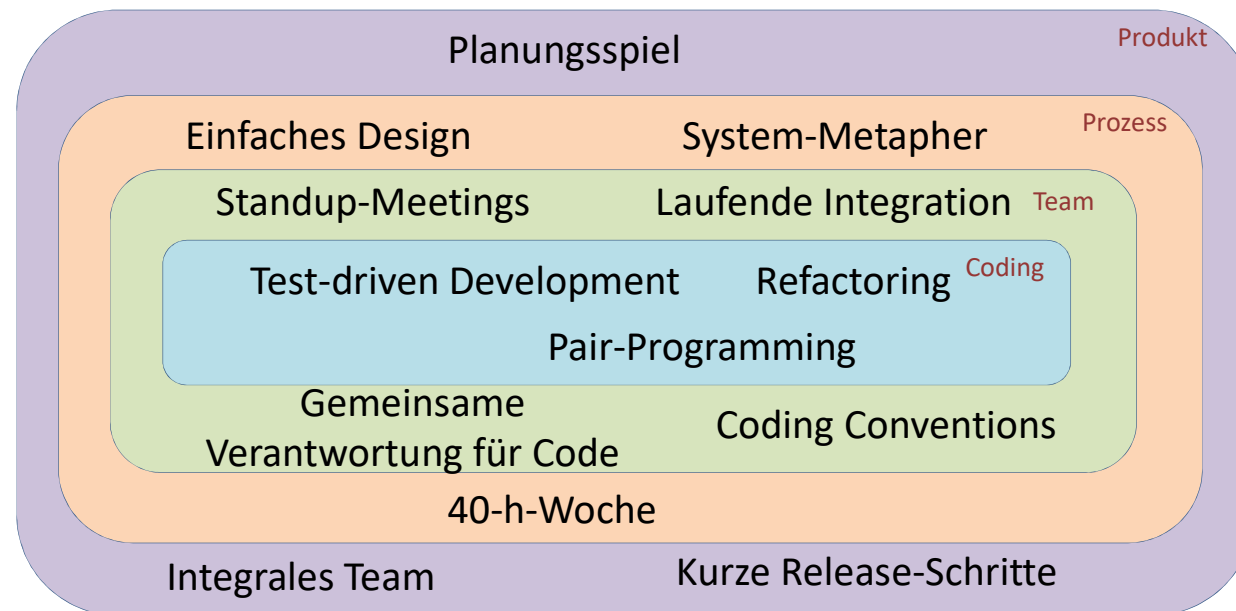
Extreme Programming (XP)





Extreme Programming (XP)

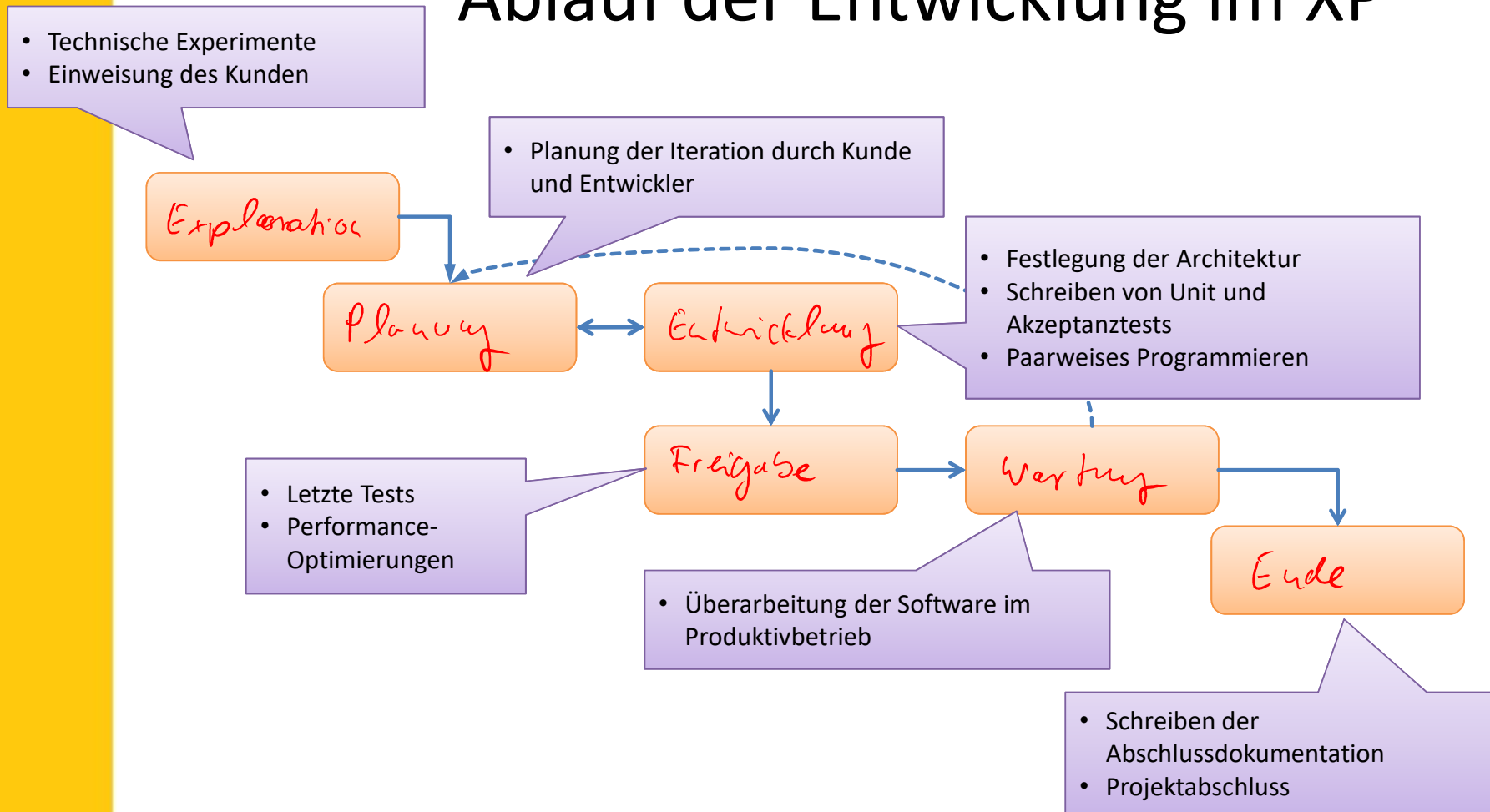
- Menge von Methoden (Praktiken), um qualitativ hochwertige Software zu entwickeln



Extreme Programming: Grundsätze

- Iterative Entwicklung (getrieben durch neue oder geänderte Features)
- Wenig Analyse- und Entwurfstätigkeiten, nur rudimentäre Spezifikationen, selbst-dokumentierender Code ("simple design")
- Frühes Programmieren, prototypisches Umsetzen einzelner "stories"
- Testfälle stehen am Anfang und ersetzen Spezifikation ("test first")
- Ständige Kommunikation der Entwickler mit Management und Benutzern, kurze Rückkopplungsschleifen, schnelle Rückmeldungen
- Schrittweise Änderungen, schrittweise angepasste Tests ("refactoring"), fortlaufende Integration ("continuous integration")
- Fahrer-/Beifahrer-Prinzip beim Programmieren ("Pair programming"); schnelle Code Reviews
- Gemeinsame Standards aller Entwickler, gemeinsames Eigentum am Code ("collective code ownership")

Ablauf der Entwicklung im XP



Planspiel / User Stories

- User stories sind ein oder mehrere Sätze in allgemeiner Sprache aus Sicht des Kunden / Nutzer der Software, welche eine benötigte Funktion oder Verwendung des Systems beschreibt.
 - Art Anforderungen und benötigte Funktionen auszudrücken
- Beschreibt **Wer, Was, Warum** einer Anforderung in einer einfachen Art und Weise.



As a librarian, I
want to be able ✓
to search for books
by publication year.

Bewertung/Anwendbarkeit des XP

- Nur für **kleine bis mittlere Teams** ausgelegt (bis zu 15 Entwickler)
- Erfordert **hochqualifizierte** Mitarbeiter
- Die XP-Praktiken sind untereinander **stark verkettet**
- Erzwingt ein „**Alles oder Nichts**“-Vorgehen
- Wenn kein Kunde vor Ort möglich, da keiner existent (Software für den Massenmarkt), sollten ausgewählte Teammitglieder diese Rolle **Vollzeit** übernehmen

Read-Me-driven Development

- Fokus auf Anwendung des Softwaresystems, um richtiges Produkt zu bauen
- Zuerst die Read-Me schreiben, dann erst mit Testfällen, Design, usw. anfangen

Behavior-driven Development

- Verhalten von Software wird beschrieben
- Dabei Schlüsselwörter, die Vorbedingungen und Nachbedingungen beschreiben
- Implementierung der Software anhand der Szenarien

Scrum – agile SW-Entwicklung



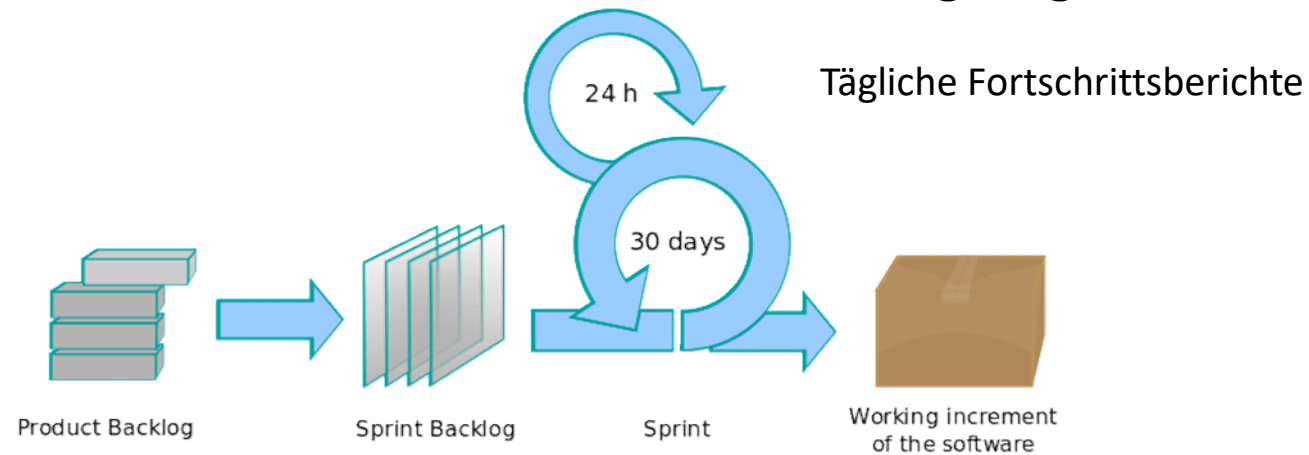
Was ist Scrum?

- SCRUM ist ein **agiles Managementframework**, bei dem Entwicklerteams als eine **Einheit** zusammenarbeiten, um ein gemeinsames Ziel zu erreichen.
- Ziel: Ordnung ins Chaos der Entwicklung zu bringen
 - Flexibel auf Änderungen der Anforderungen durch kurze Entwicklungszyklen reagieren
 - Enthält **keine** Praktiken, die vorschreiben, wie die Software entwickelt werden soll



Leitidee und Ablauf

- Ein Team ist besonders produktiv, wenn es sich mit seinem Produkt identifiziert und auch dafür die Verantwortung trägt.



Dokument über Anforderungen (Liste von Features, Bugs, etc.)

Dokument, welches die Aufgaben listet, die getan werden müssen, um das Feature zu implementieren.

Zeitraum der Realisierung

Laufendes Produkt mit neuem Feature

Prinzipien von Scrum

- Es gibt **keine Projektleiter!**
 - Team teilt sich selbständig Arbeit auf.
- **Pull-Prinzip**
 - Nur das Team kann entscheiden, wieviel Arbeit in gegebener Zeit geleistet werden kann
- **TimeBox**
 - Es existieren klare zeitliche Grenzen
- Potential **Shippable Code**/Produkt
 - Ergebnis sind immer fertig Produkte

Rollen in Scrum



Chicken: Let's start a restaurant!

Pig: What would we call it?

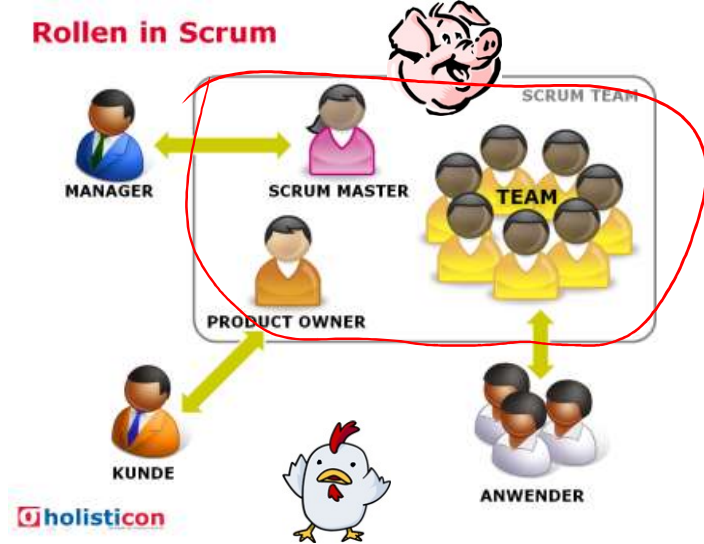
Chicken: Ham 'n' Eggs!

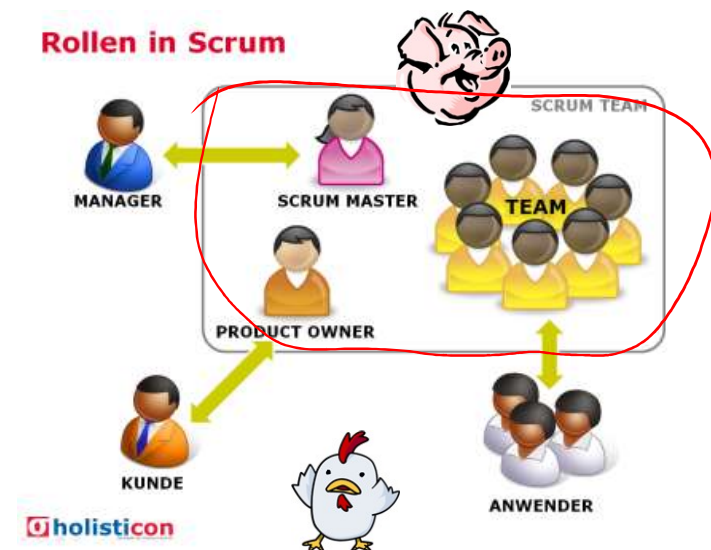
Pig: No thanks. I'd be committed, but you'd only be involved!



- Direkt am Prozess Beteiligte heißen „pigs“
 - Tragen Last und Risiko und arbeiten nach SCRUM-Regeln
 - Regelmäßige Treffen und Pflege der Artefakte
- Außenstehende heißen „chickens“
 - Nur anwesend bei Meetings, um sich zu informieren
 - Werden versucht aus dem Projekt herauszuhalten

Rollen in Scrum

- Produkt Owner (Der Visionär, kein Chef!)
 - Pflegt und priorisiert Product Backlog, fachlicher Ansprechpartner für Kunden (evtl. bei tägl. SCRUMs dabei)
 - Das Team (Die Lieferanten)
 - 5-10 Personen meist interdisziplinär
 - Selbst-organisierend, tägl. Meetings
 - ScrumMaster (Der Change Agent)
 - Verantwortung für SCRUM-Prozess
 - Moderiert, vermittelt, optimiert
 - Der Kunde (Der Geldgeber)
 - Der Anwender (Der Nutzer)
 - Der Manager (Der Big Boss)
- 
- Das Diagramm zeigt die Rollen in Scrum. Ein roter Rahmen umschließt die Rollen SCRUM MASTER, TEAM und PRODUCT OWNER, die als 'SCRUM TEAM' bezeichnet werden. Ein pinker Schwein-Charakter steht über dem Team. Ein Manager (blauer Anzug) steht links und ist mit dem SCRUM MASTER (rosa Kleid) verbunden. Ein Kunde (orange Anzug) steht unten links und ist mit dem PRODUCT OWNER (braune Kleidung) verbunden. Anwender (weiße Kleidung) stehen unten rechts und sind mit dem TEAM verbunden. Ein gelber Pfeil führt vom TEAM zum Anwender. Ein Logo 'holisticcon' ist unten links zu sehen.

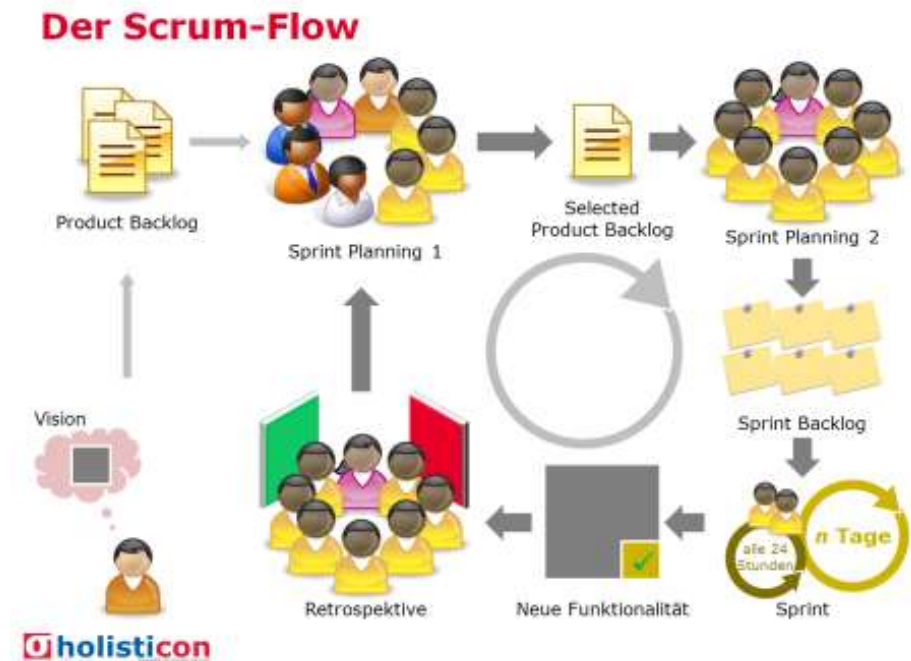


Vorbereitung für Scrum

- Ziel: Erstellen eines Backlogs
- Weg:
 - Produkt Owner erstellt Vision
 - Product Owner und Team erarbeiten gemeinsam Einträge für Backlog
 - **Produkt Owner** priorisiert Backlog-Items
 - **Team** schätzt(!) Aufwand für Backlog-Items

Sprint

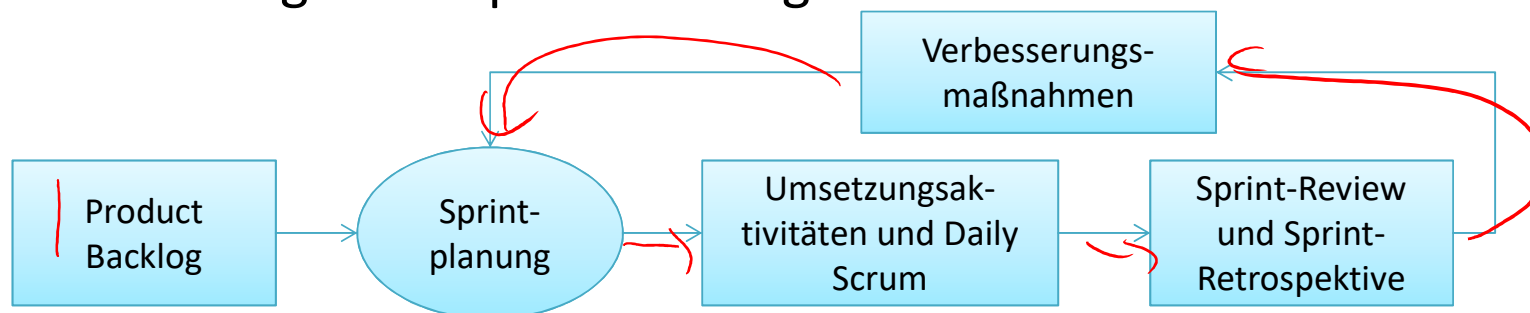
- Ziel: Umsetzung eines Teils des Backlogs in **auslieferbaren Code**
 - Phasen eines Sprints
 - Planung
 - Durchführung
 - Abschluss
-
- ```
graph LR; PB[Product Backlog] --> SP1[Sprint Planning 1]; SP1 --> SPB[Selected Product Backlog]; SPB --> SP[Sprint Planning]; SP --> PB
```



# Sprints

- Wandelt Anforderungen in lauffähige, getestete und dokumentierte Software um
  - Overhead für Scrum-spezifische Aktivitäten  $\leq 10\%$
- Vorgehen ist iterativ und inkrementell
  - Agile Entwicklungspraktiken (z.B. TDD) einsetzbar, aber nicht festgelegt
- | Während des Sprints keine Änderung an dessen Dauer, den Anforderungen im Sprint Backlog und der Teamzusammensetzung

time box

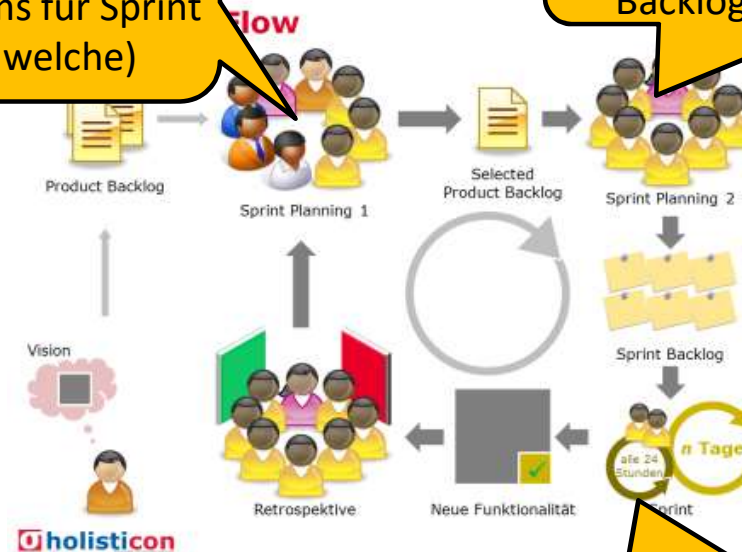




# Scrum – Sprint

**Sprint Planning 1:**  
PO, Team, Kunde und Management legen Product-Backlog-Items für Sprint fest (Team entscheidet welche)

**Sprint Planning 2:**  
Team diskutiert das Selected Product Backlog und skizziert Lösungen



**Daily Scrum:**  
Team stimmt sich **täglich** über die zu bearbeitenden Backlog Items ab

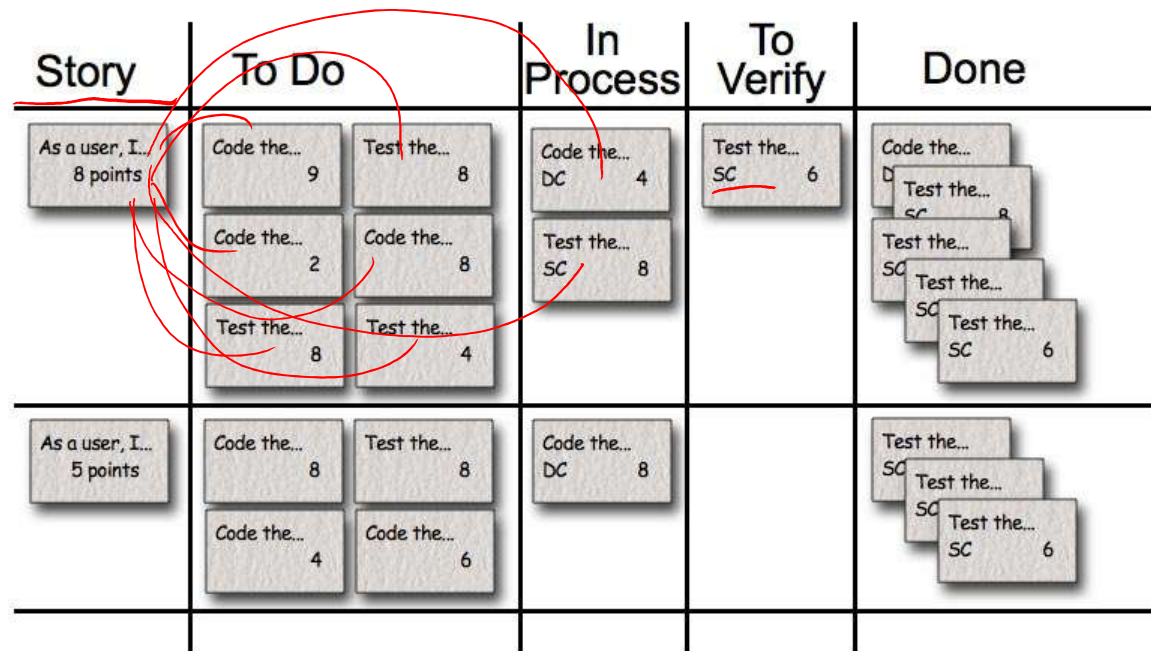
# Daily Scrum

- Ablauf:
  - Jedes Mitglied wählt Tagesaufgabe selbst
  - Jedes Mitglied informiert über eigenen Fortschritt
  - Jedes Mitglied berichtet über Blockaden und aufkommende Probleme
- Bedingungen:
  - Teamgröße i.d.R. nicht mehr als 8 Personen
  - 15-Minuten-Regel (SCRUM-Master moderiert)
  - Bei größeren Projekten „SCRUM of SCRUMs“

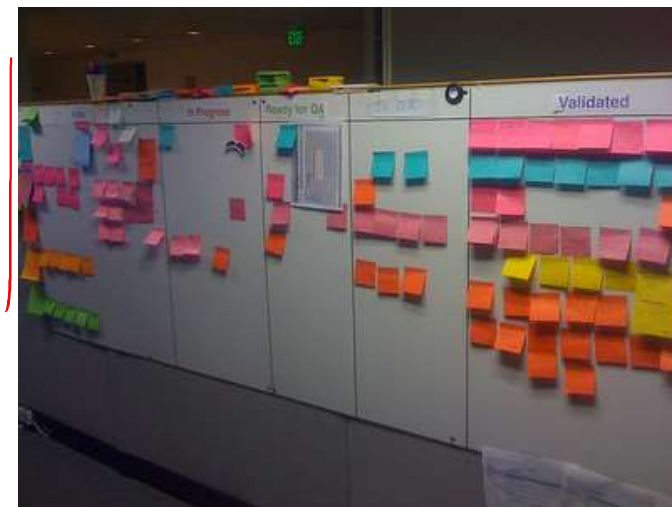
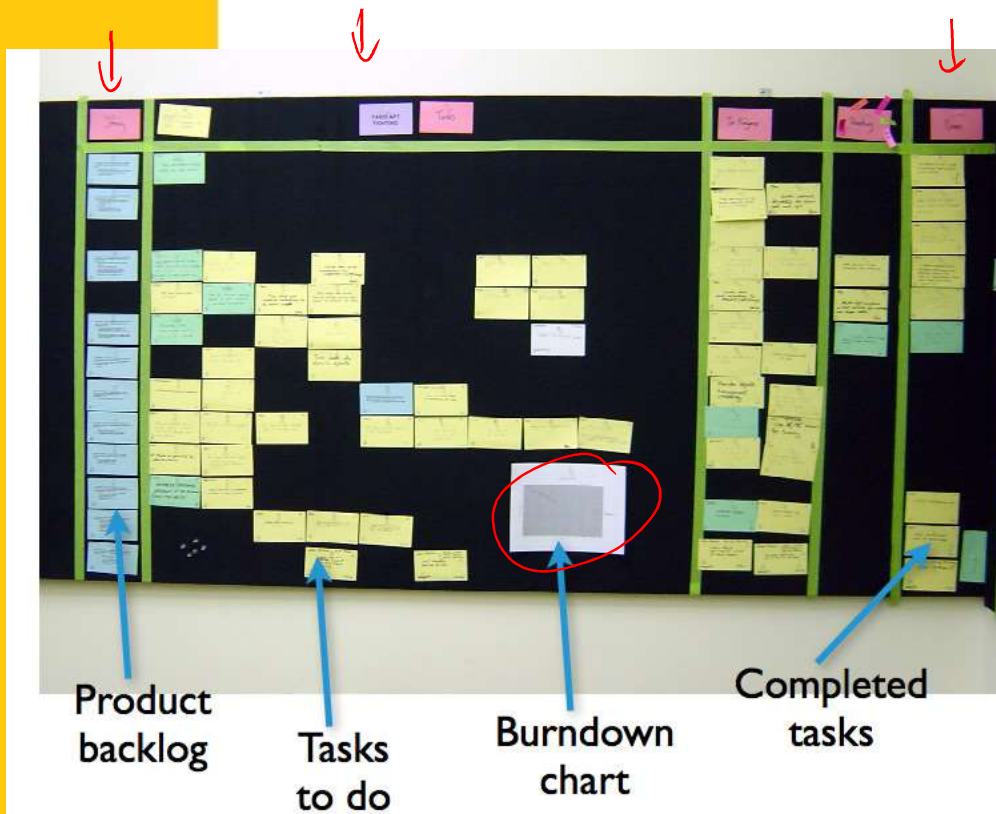


# Scrum – Task Board

- Entwickler heften neue Karten an und verschieben sie selbstständig (oft während / nach Daily Scrum)

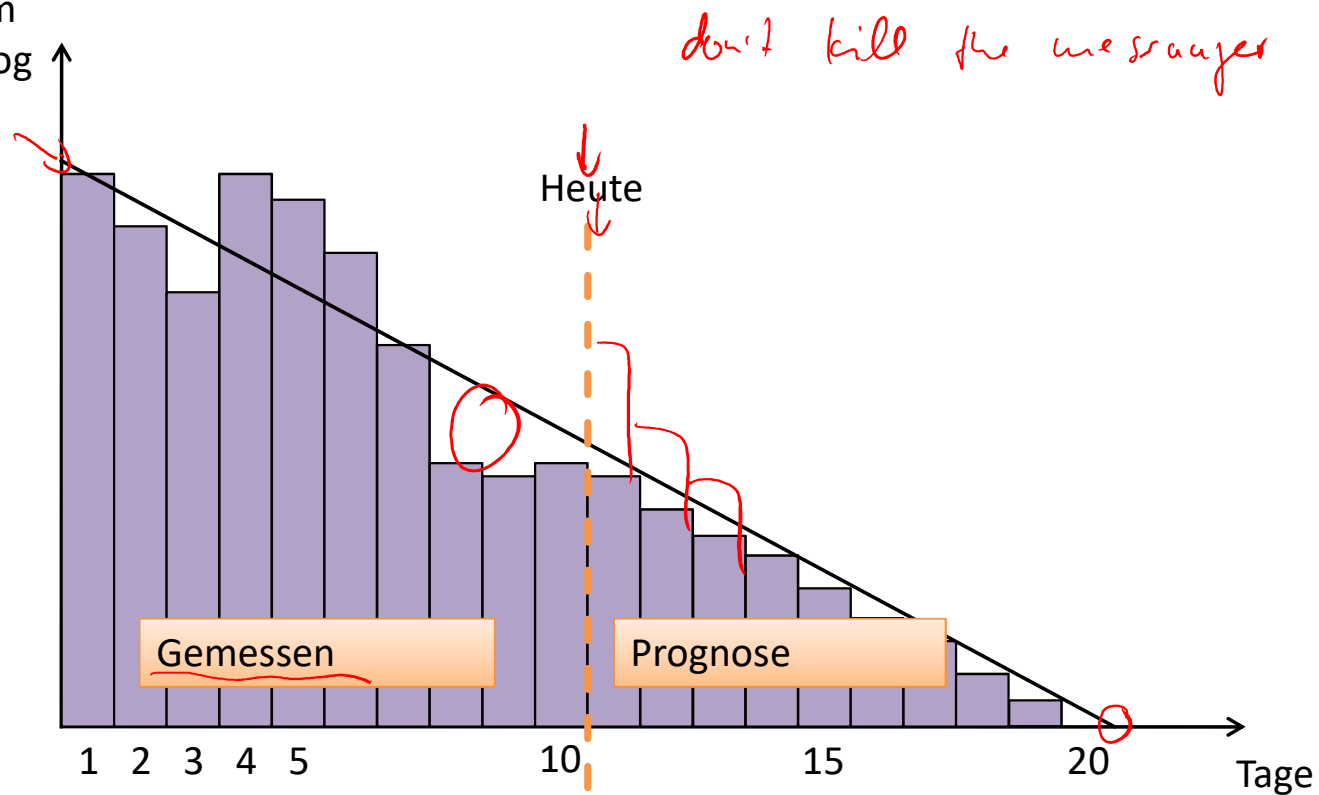


# Beispiele



# Sprint Burndown Charts

Aufwände im  
Sprint Backlog

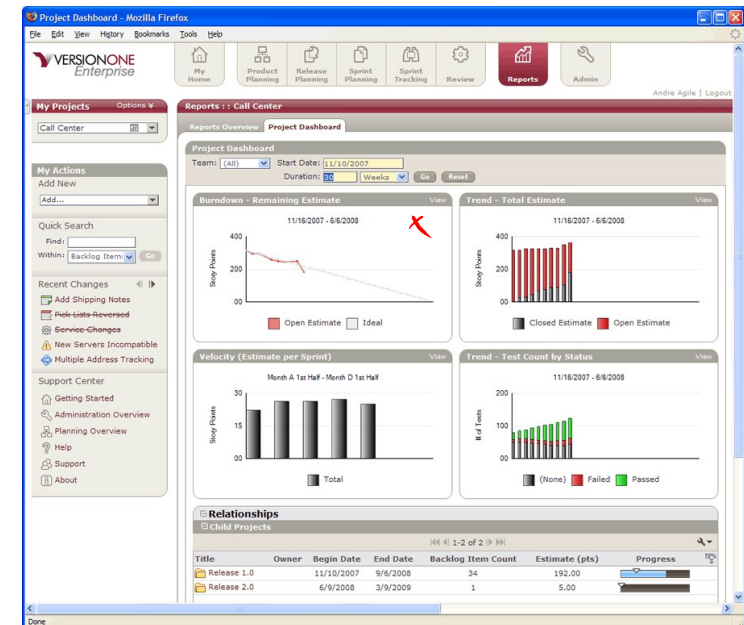


# Sprint-Abschluss

- Estimation-Meeting  
Ziel: **Aktualisierung** und **Anpassung** des Product Backlogs  
Teilnehmer: PO, SM, Team
- Sprint Review  
Ziel: Präsentation der **fertigen** Product-Backlog-Items
- Sprint Retrospektive  
Ziel: Verbesserung der **zukünftigen** Planung von Sprints

# Scrum – Überblick

- **Team** steht in Vordergrund und trägt Verantwortung
- **Tägliche** feste Meetings
- Sprint-Iterationen mit jeweils **shipable code**
- Continuous Improvement Process
  - Schätzungen in JEDEM Sprint anpassen
- Tool-support:  
Version One, Rally



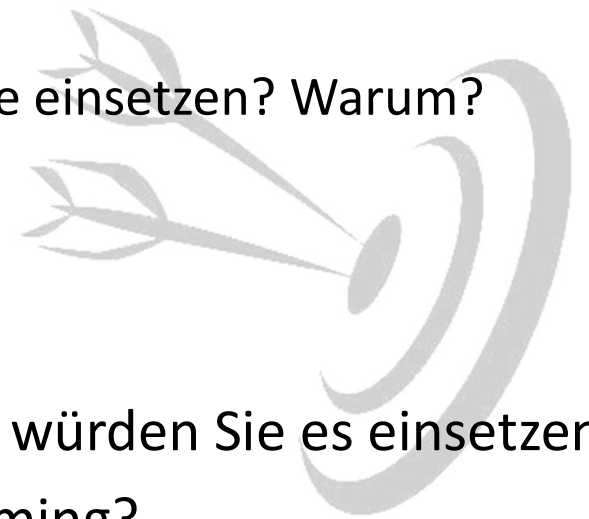
## Einordnung: Scrum vs. XP *extreme Programme*

- Scrum ist Projekt-Management-Methode
  - Spezifiziert den **Prozess** von Idee zum finalen Produkt
  - Unabhängig von der Entwicklungsmethode (z.B. Wasserfall)
- XP ist Entwicklungsmethode
  - Definiert, wie Software agil entwickelt wird
- Wenn Scrum für SW-Entwicklung eingesetzt wird, dann ähnelt es XP
- Kombiniert man beide erhält man:
  - Sprints, Artefakte, etc. von Scrum
  - Methoden der SW-Entwicklung (Pair-Programming, TDD, Refactoring, etc.) von XP



## Was Sie mitgenommen haben sollten:

- Nennen/Erläutern Sie X Softwareentwicklungs-prozesse
- Stellen Sie einen sequentiellen und einen iterativen Softwareentwicklungsprozess gegenüber
- [Anwendungsszenario]
  - Welchen Softwareentwicklungsprozess würden Sie einsetzen? Warum?
  - Horizontaler oder vertikaler Prototyp? Warum?
  - Iterativer oder sequentieller Prozess? Warum?
- Was sind die Eigenschaften von Scrum und wann würden Sie es einsetzen?
- Worin liegt der Unterschied zu Extreme Programming?



# Literatur

- Sommerville. Software Engineering.
- Ludewig and Lichter. Software Engineering: Grundlagen, Menschen, Prozesse, Techniken
- eXtreme Programming Explained: Embrace Change. Kent Beck. Addison-Wesley Pub Co; ISBN: 0201616416; 1st edition (October 5, 1999)

