

Softwaretechnik

Softwareentwicklungsprozesse



UNIVERSITÄT
LEIPZIG

Prof. Dr.-Ing. Norbert Siegmund
Software Systems

Übung: Anforderungen des SWT Praktikums

Wie ist das gemeint?

Alle Messdaten soll das Backend **automatisch** vom MQTT-Broker anfordern. Der MQTT-Broker stellt die Messdaten aller Geräte pro Sekunde in Watt zur Verfügung. Im Backend sollte sowohl für das temporäre, als auch für das langfristige Speichern der Daten eine Datenbank verwendet werden. Es sollen alle neuen Energiemessdaten **temporär** *Wie lange?* abgelegt werden. Ist das **Speicherlimit** *Wie hoch?* der Datenbank erreicht, werden die ältesten Daten zuerst **gelöscht**.

Behalten wir die Statistiken für gelöschte Daten?

Im Backend sollen die **statistischen Berechnungen** durchgeführt werden, da diese mitunter sehr aufwändig werden könnten. Es sollen der Idle-, Durchschnitts- und der *Wie bestimmen?* maximale Energieverbrauch für **alle Geräte** berechnet werden. Die Ergebnisse aller statistischen Berechnungen sollen **langfristig** gespeichert werden. Das Backend soll innerhalb eines **Microservices** gekapselt werden und über eine REST-API mit dem Frontend **kommunizieren**. *Vorgaben?*

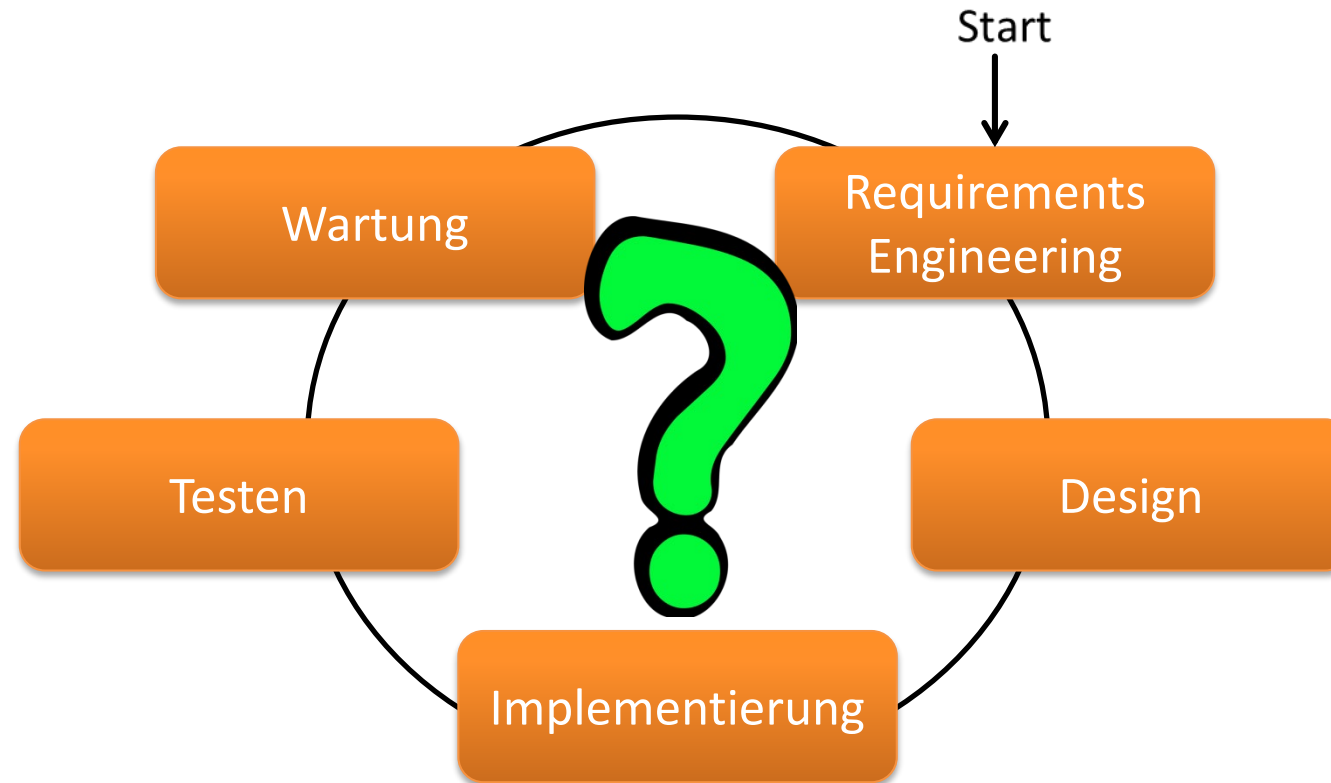
Einzel oder aggregiert?

*Verschlüsselt?
A-/Synchron?*

*Technologie?
Sprache?*

*Unbegrenzt? In Wochenscheiben?
Aggregiert über den gesamten Zeitraum?
Auswirkungen auf Speicherlimit?*

Einordnung



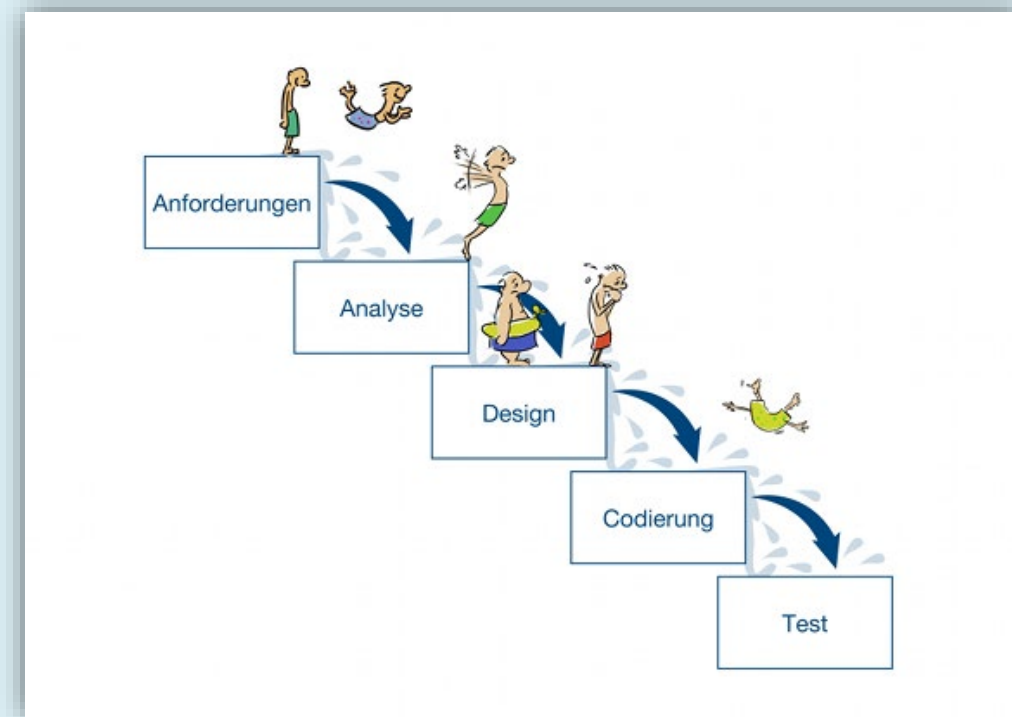
Lernziele

- Überblick und grundlegendes Verständnis für Entwicklungsprozesse haben
- Traditionelle Softwareentwicklungsprozesse (sequenzielle Modelle) kennen und deren Probleme verstehen
- Alternative, aktuelle 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

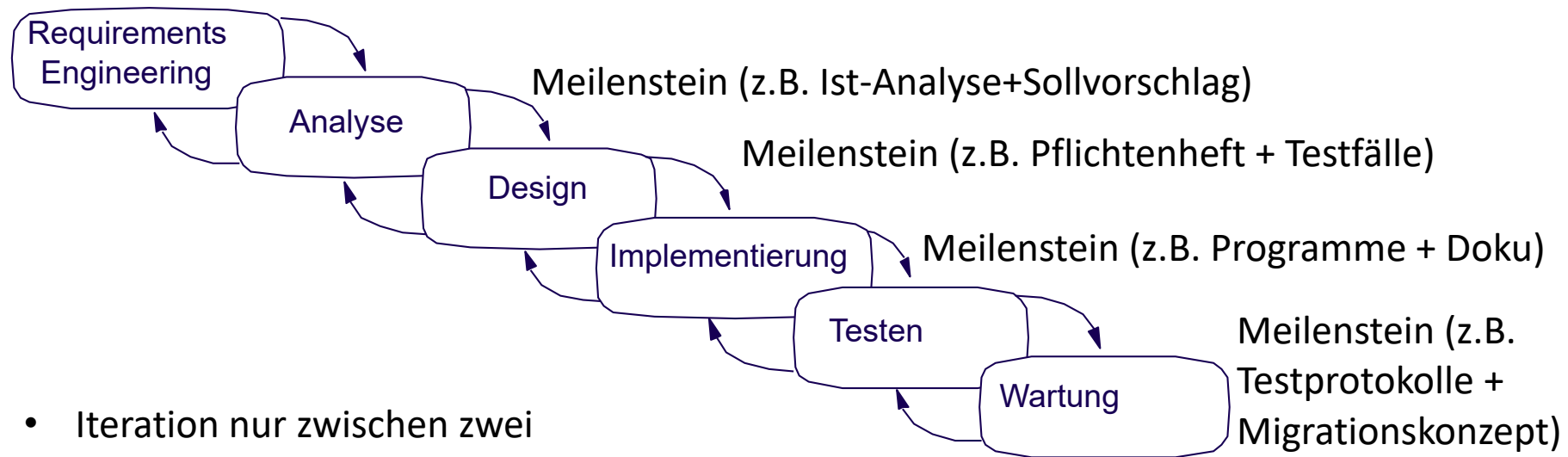
Sequentielle Modelle



Grundidee von Seq. Modellen

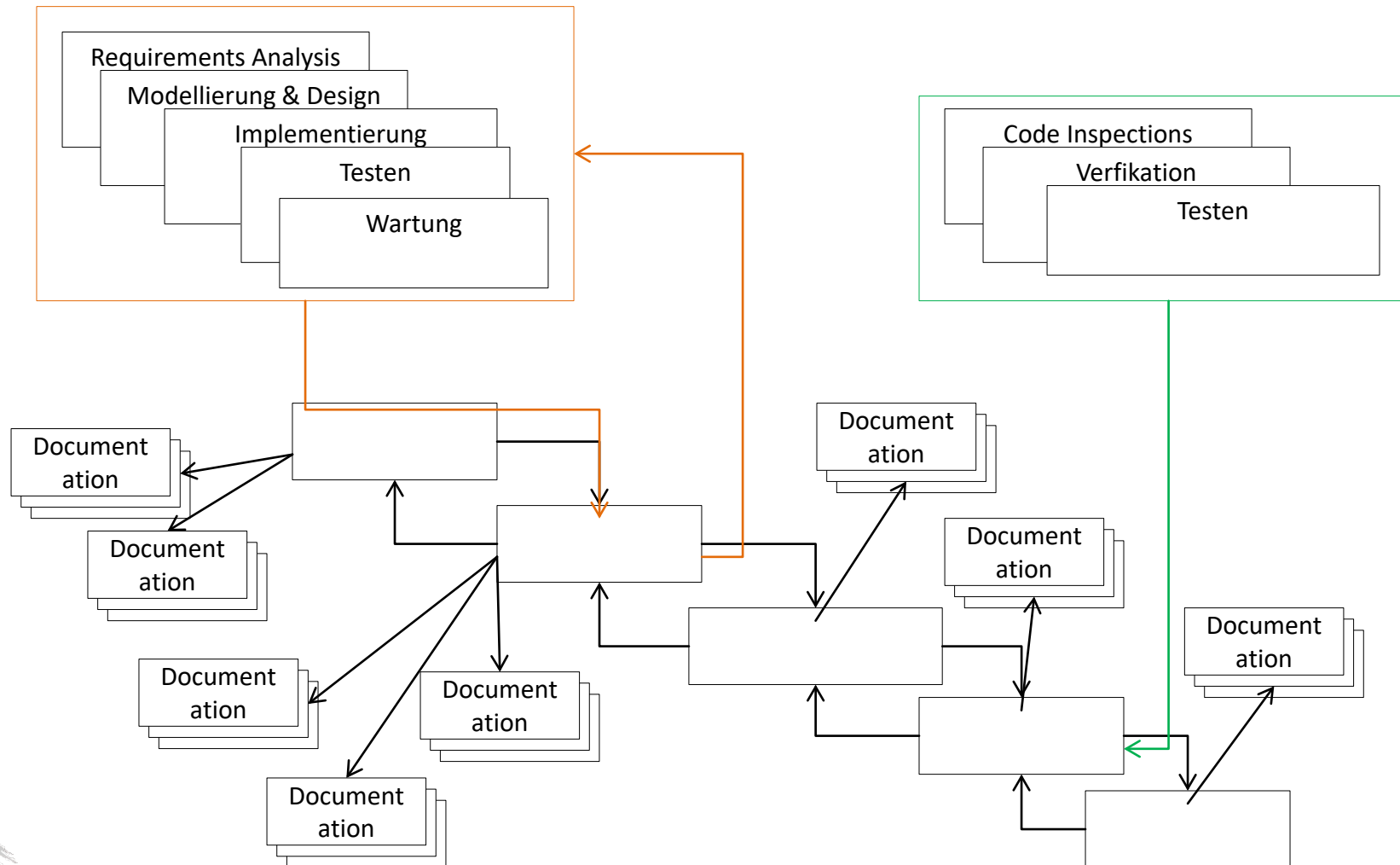
- Schritt für Schritt werden alle Phasen des Lebenszyklus nacheinander „abgearbeitet“
 - Planen / Konzipieren
 - Entwerfen / Ausarbeiten
 - Inbetriebnehmen / Warten
- Modelle:
 - Wasserfallmodell
 - V-Modell

Wasserfallmodell



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

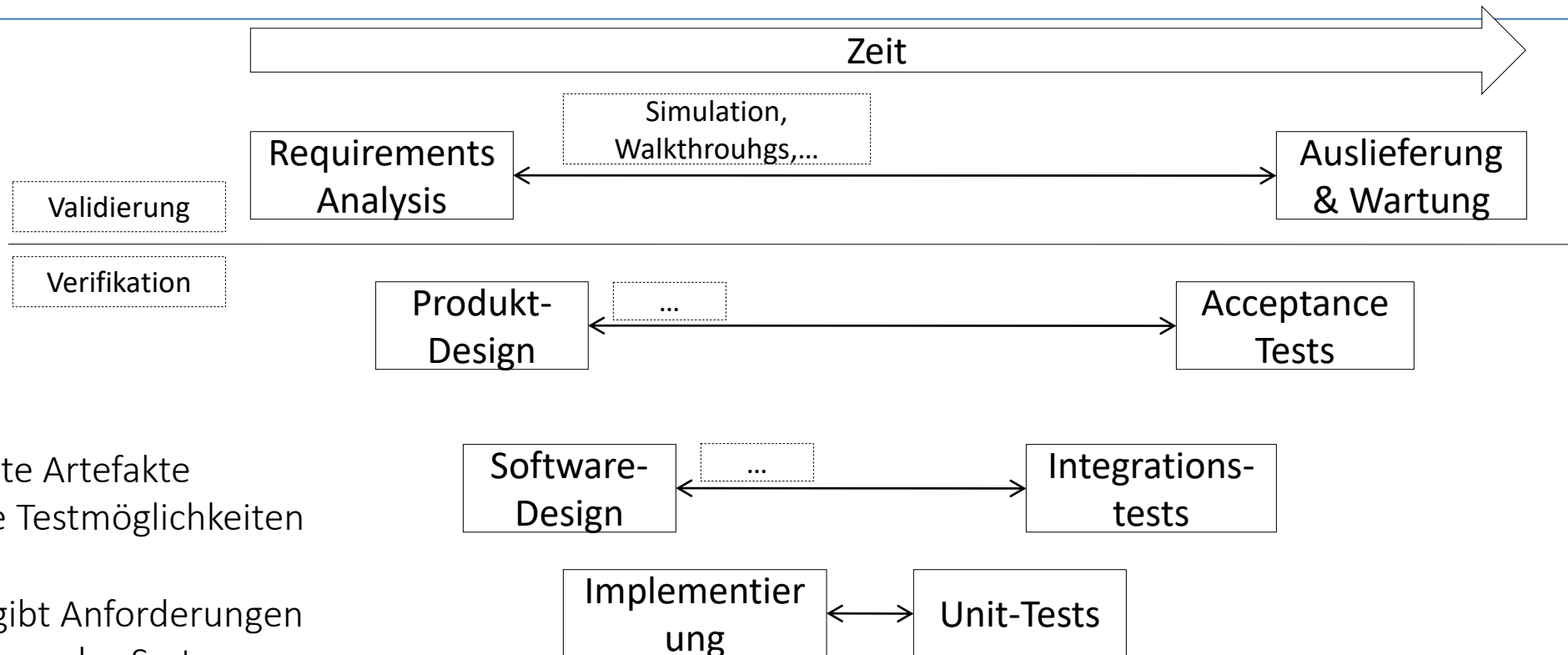
Wasserfallmodell: Dokumentation



Wasserfallmodell: Diskussion

- Vorteile:
 - *Dokumentation* nach jeder Phase verfügbar
 - Klare *Trennung* der Phasen und Verantwortlichkeiten
 - Analog zu Ingenieurprojekten (Brückenbau etc.)
- Nachteile:
 - *Starres* Vorgehen (veraltete Dokumentation)
 - 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!)
 - Meist unrealistisch in der Praxis

V-Modell



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

Analyse ergibt Anforderungen

➔ Validierung des Systems

Design ergibt Grobspezifikation

➔ Verifikation der Komponenteninteraktion

Feinentwurf/Implementierung 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)
- Implementierungsdokumentation: 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 Hypothesen zu testen, zu explorieren und zu validieren. Dient der Reduzierung von Risiken.


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

- UI prototype: 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.

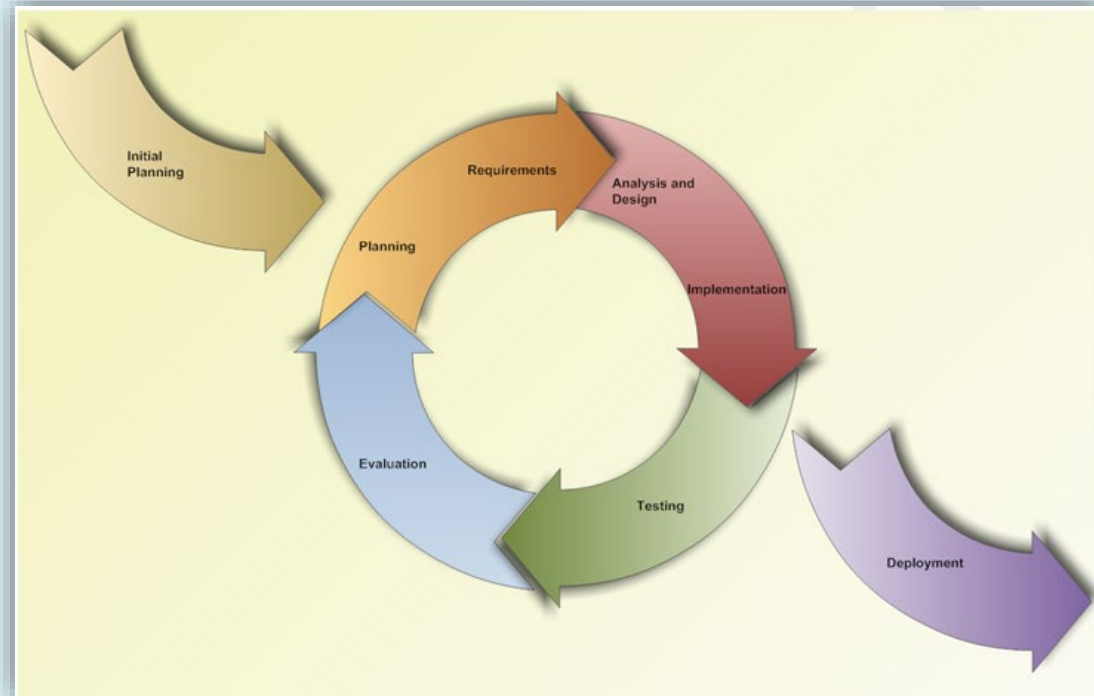


First do it,
then do it right,
then do it fast.

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

Prototypen

- Möglichst früh eine erste Version erstellen
- Insbesondere bei Unklarheiten bei Kundinnenwünschen
- Selbst wenn Kernfunktionalität fehlt
- Kunde kann Fortschritt erkennen und (Teil-) Lösung bewerten
- Erinnerung: horizontale vs. vertikale Prototypen

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

Prinzipien der inkrementellen 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

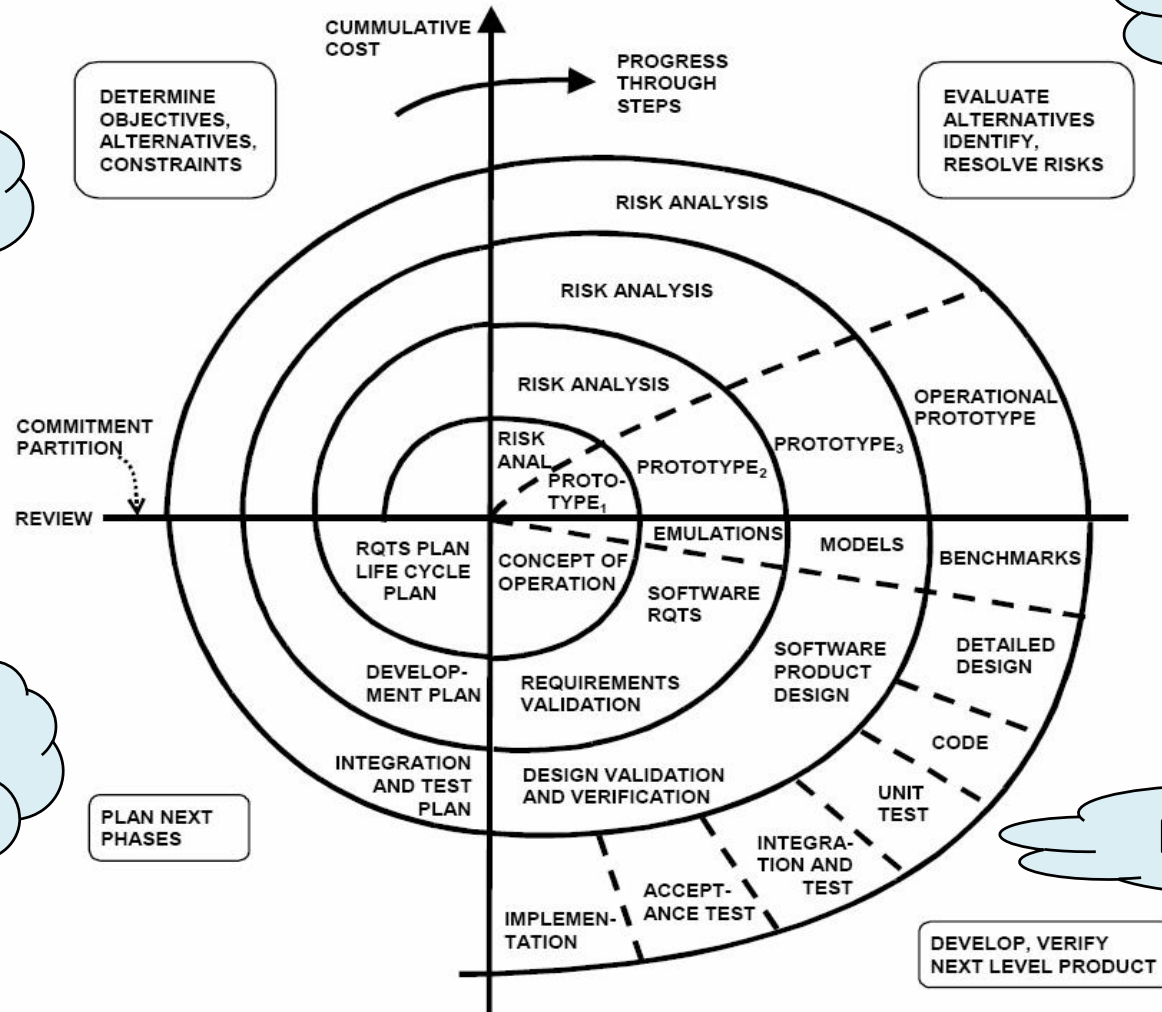
Spiralmodell

- 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

Identifiziere
und löse Risiken

Bestimme
Ziele



Projekt-
planung

Entwicklung und Testen

Spiralmodell

- Kombiniert Wasserfallmodell mit Prototypen und iterativer Entwicklung
 - 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
 - Kundin bekommt früh Vorab-Version (erfordert Einbindung der Kundin, schwierige Gesamtplanung)
 - Ermöglicht früh Return-on-Investment

Aufgabe

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

Agile Softwareentwicklung



Erinnerung: Gründe für das Scheitern

- Zeit aufgebraucht
- Budget aufgebraucht
- Falsches Produkt
- Schlechtes Produkt
- Nicht wartbar
- Ineffiziente Software
- Ökonomische Ursachen

Gründe:

- Dokumentation hat hohen Aufwand und veraltet schnell -> trotzdem folgen wir ihr
- Projektpläne sind ungenauer, ändern sich leider nicht -> trotzdem halten wir daran fest

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
Ron Jeffries
Steve Meller*

*Arie van Bennekum
James Grenning
Jon Kern
Ken Schwaber*

*Alistair Cockburn
Jim Highsmith
Brian Marick
Jeff Sutherland*

[Agilemanifesto.org]

Manifest der agilen Softwareentwicklung

- Menschen und Kooperation vor Werkzeugen und (automatisierten) Prozessen
- Funktionsfähige Software vor umfassender Dokumentation
- Zusammenarbeit mit Kundinnen 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

1. Kundinnenzufriedenheit 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 Kundinnen und Entwicklerinnen.
5. Baue Projekte mit motivierten Mitarbeiterinnen 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

7. Das primäre Maß für Fortschritt ist funktionierende Software.
8. Agile Prozesse bieten Kontinuität in der Entwicklung, so dass Investorinnen, Entwicklerinnen und Anwenderinnen 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 effektiver arbeiten kann und passt sich entsprechend an.

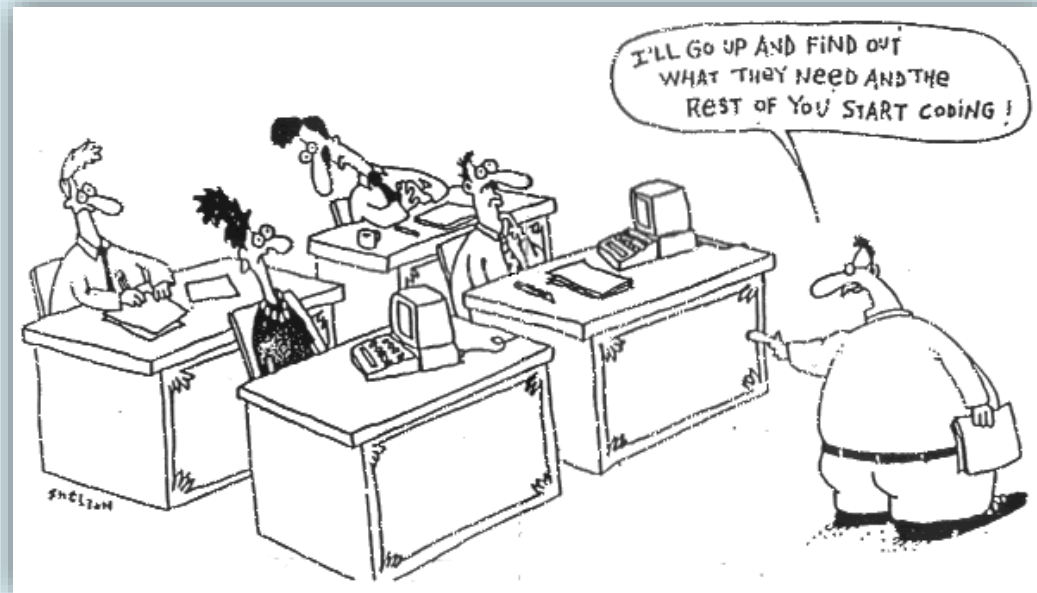
Projektrisiken

- Zeit und Budget
 - Kleine Iterationen mit kalkulierbaren Budget
 - Benötigte Funktionalität wird priorisiert
 - Auslieferbares Produkt nach jeder Iteration
- Falsches oder schlechtes Produkt
 - Stakeholder und Kundinnen sind in jeder Iteration involviert
 - Testen und Benutzen des Produktes am Ende jeder Iteration
 - Schnelle Rückmeldungen zur Neupriorisierung
- Wartbar, da ständig im Einsatz

Konsequenzen

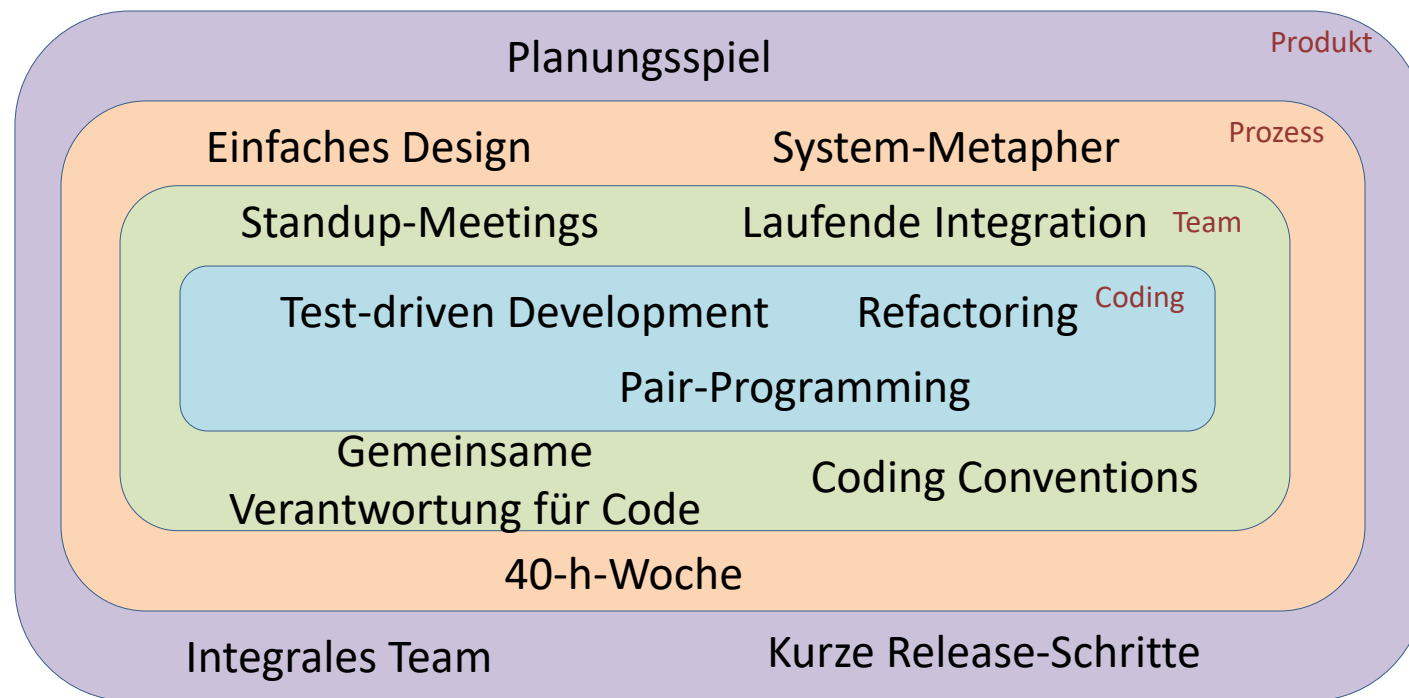
- Kein vollständiger Projektplan (wann [alle] Anforderungen erfüllt?)
- Anforderung als ständiger Input (wie sammeln und ordnen?)
- Evolutionäres Softwaredesign statt einmaliger Entwurf
- Häufige Refaktorisierung des Codes, um neue Anforderungen und geändertes Design umzusetzen (mit Reimplementierung existierender Funktionalität)
- Technische Schulden können sich aufhäufen
- CI/CD Prozess benötigt für schnelle Iterationen
- Regression und kontinuierliches Testing erforderlich
- Hoher Kommunikationsaufwand mit Teams und entsprechende Organisationsstruktur notwendig

Extreme Programming (XP)



Extreme Programming (XP)

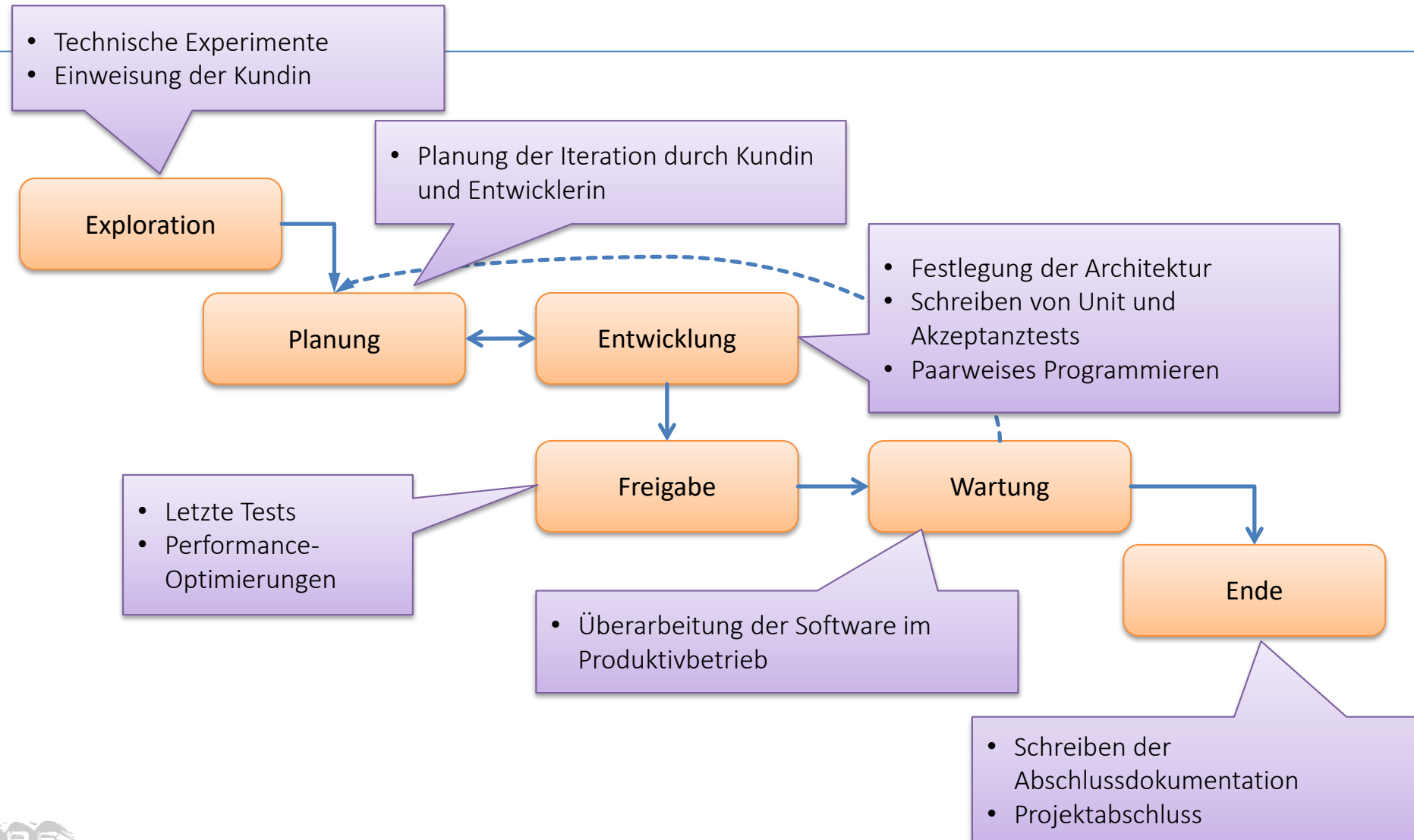
- Menge von Methoden (Praktiken), um qualitativ hochwertige Software zu entwickeln



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 Entwicklerin mit Management und Benutzenden, 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 Entwicklerinnen, 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 der Kundin / Nutzerin 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 Entwicklerinnen)
- Erfordert *hochqualifizierte* Mitarbeitende
- 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

Test-driven Development

- Erst die Testfälle, dann den Code
- Hauptsächlich bei Unit Tests
- Vorteile:
 - Entwickler nimmt mehr Sicht des Anwenders an
 - Denken in kleinen Modulen, die getestet werden können, dadurch bessere Modularisierung
- Probleme:
 - Schwer bei kompletten Funktionstest (z.B. Usability, Datenbankverbindungen)
 - Testen und Implementierung durch denselben Entwickler

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

Lean Software

Eine Variante von Agile

Idee: Verschwendung Eliminieren

Müll / Verschwendung ist es, eine Aktivität auszuführen, auf die verzichtet werden kann, ohne das sich etwas am Ergebnis ändert.

Fokus auf Verschlankung von Produkten und Prozessen:

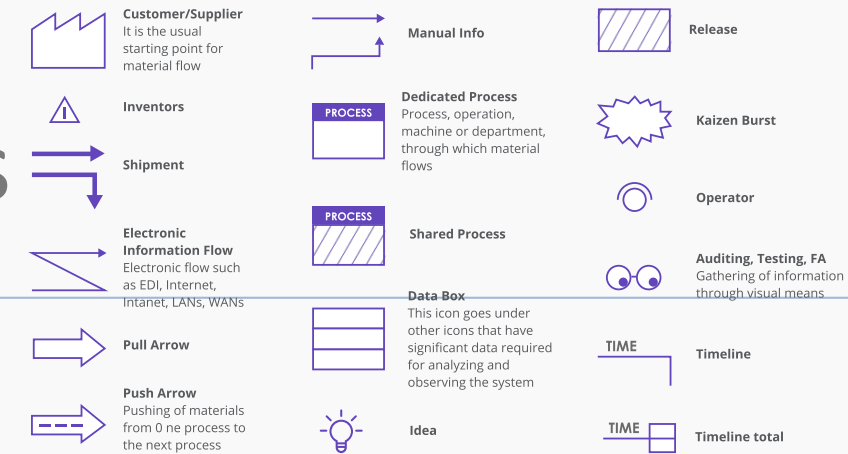
- Nur wertvolle Features berücksichtigen
- Inkrementelle Entwicklung (Probieren statt Dokumentieren)
- Entscheidungen (z.B. über Features) nach hinten schieben
- Möglichst schnell liefern, um sofortige Rückmeldung zu bekommen
- Verantwortung an Team übergeben
- Integrität des Kunden erhöhen (Endnutzer Testing)
- Das große Ganze sehen (besseres Verständnis von Szenarien)
- Qualität einbauen (CI, Test-Driven-Development, etc.)
- Wissen schaffen (mit wissenschaftlichen Methoden lernen, um beste Alternative zu finden)



<https://www.bund-berlin.de>

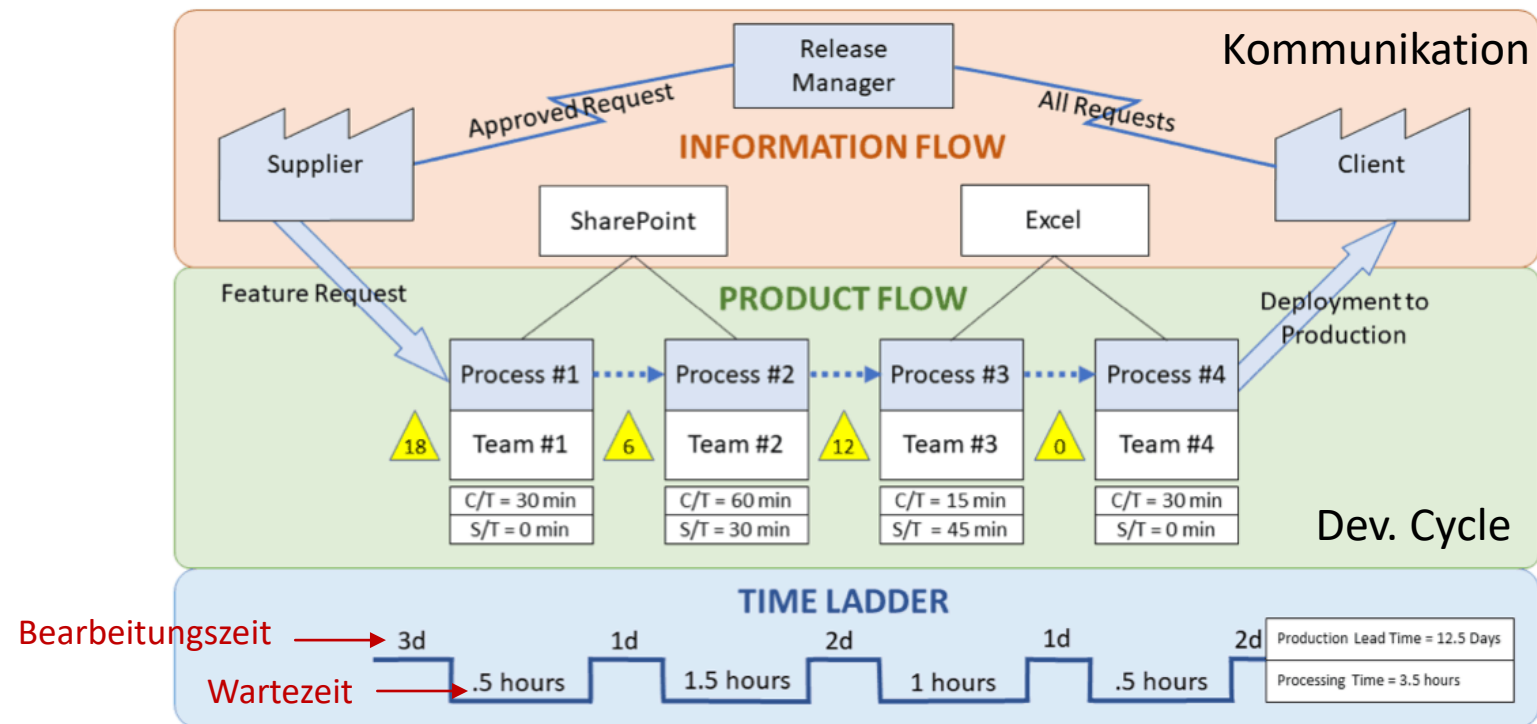
Value-Stream Maps

Methode, um derzeitigen Stand zu ermitteln, wie Informationen und Materialien vom Beginn des Produktionsprozesses bis zum Kunden fließen.
Neben Kanban, eine der wichtigsten Visualisierungen im Lean SW Development



- Bessere Kollaboration
- Vogelperspektive, um den Gesamtkontext zu erkennen
- Visuelles Feedback, um Problemstellen zu identifizieren
- Sichtbar, wo tatsächlicher Mehrwert (Value) generiert wird

Simplified Value Stream Map for Software Development

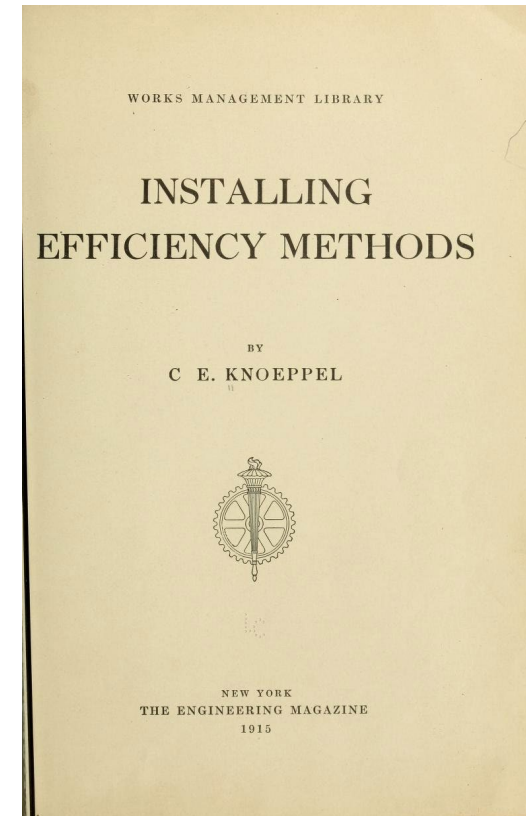


Weiterführende Infos: <https://www.plutora.com/blog/value-stream-mapping>

Historie: Toyota und Knoepfel

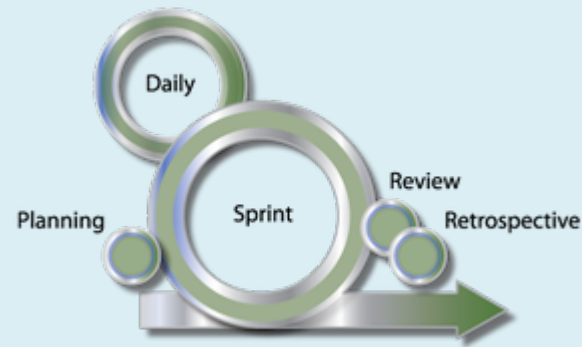


Zweiten Hälfte des 20. Jahrhunderts von Toyota
erfolgreich im Produktionssystem eingeführt



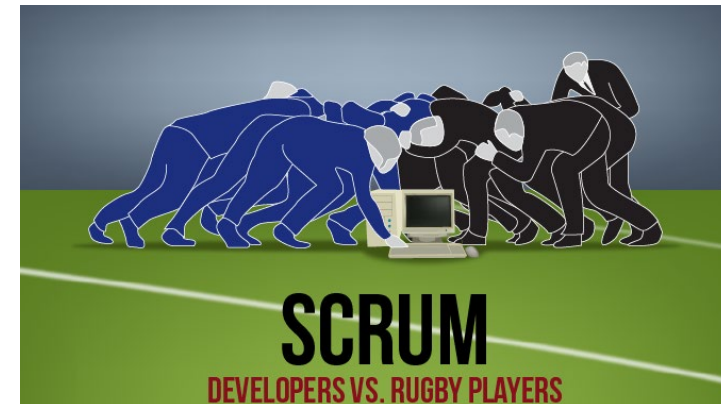
Ursprung: Buch „Installing Efficiency Methods“ von
Charles Knoepfel, 1915.

Scrum – Projektmanagement für agile und lean SW-Entwicklung



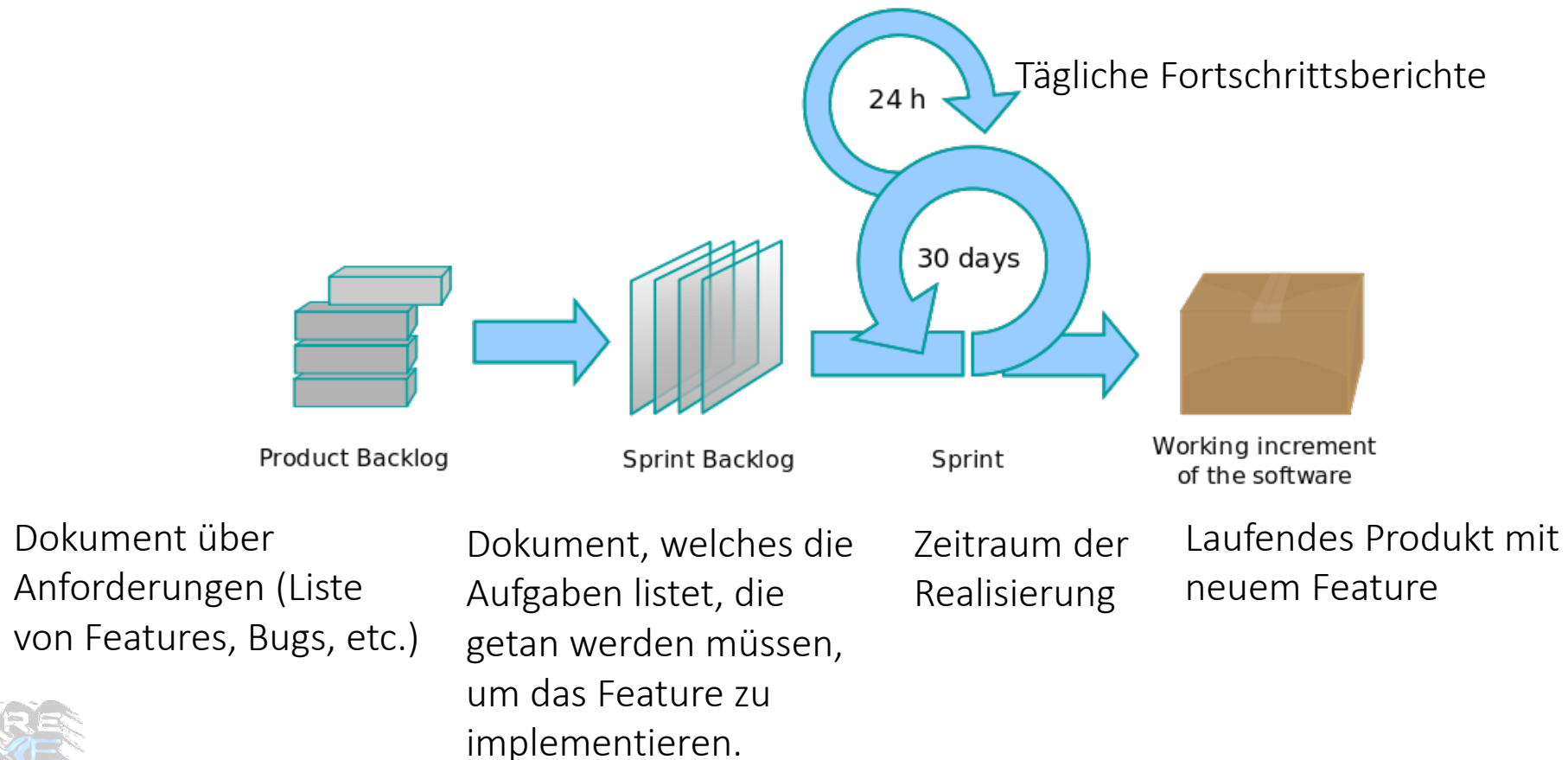
Was ist Scrum?

- 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.



Prinzipien von Scrum

- Es gibt *keine Projektleiterin*!
 - 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

- Produkt Owner (Die Visionärin, kein Chef!)
 - Pflegt und priorisiert Product Backlog, fachlicher Ansprechpartnerin für Kunden (evtl. bei tägl. SCRUMs dabei)
 - Anforderungsmanagement, Releasemanagement, Kosten und Nutzen Betrachtung
- Das Team (Die Lieferanten)
 - 5-10 Personen meist interdisziplinär
 - Selbst-organisierend, tägl. Meetings
- Scrum Coach*
 - Verantwortung für SCRUM-Prozess
 - Moderiert, vermittelt, optimiert
- Die Kunden (Geldgeber)
- Die Anwender (Benutzen das Produkt am Ende)
- Die Manager (Organisatorische Leitung des Teams)



*wird meist als Master, hergeleitet aus Mastery, bezeichnet

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

Anforderungen/Features als User Stories

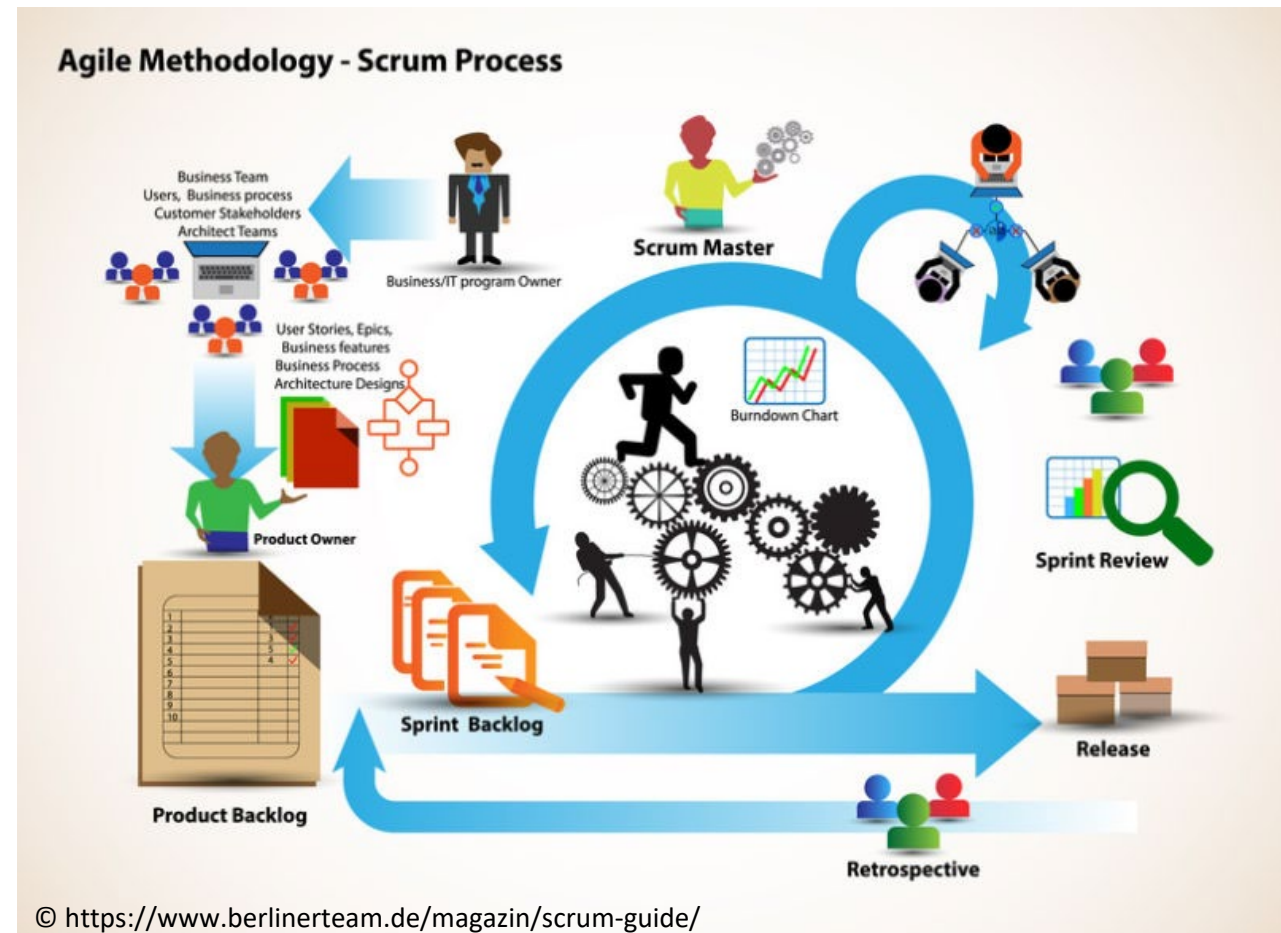
- Anforderungen sollten die INVEST Eigenschaften erfüllen:
 - Independent -> nicht abhängig von anderen Anforderungen
 - Negotiable -> kein Vertrag, Details verhandelbar
 - Valuable -> Wertvoll für die Kundin
 - Estimable -> Für Planung und Ranking
 - Small -> Wenige Personentage, -stunden
 - Testable -> Überprüfbarkeit

Tasks aus User Stories

- Folgen den SMART-Eigenschaften
 - Specific -> Aufgabe kann verstanden werden
 - Measurable -> Zustand ist überprüf- und messbar
 - Achievable -> Sollte auch Lösbar im Sprint sein
 - Relevant -> Sollte relevant für die User Story sein
 - Time-boxed -> Limitiert in einer Zeitspanne (Stunden oder Tage)

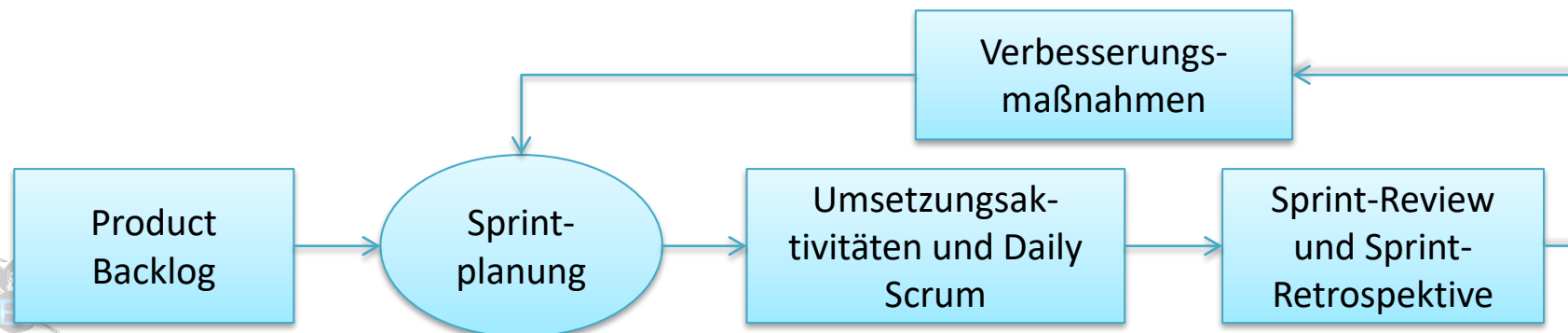
Sprint

- Ziel: Umsetzung eines Teils des Backlogs in *auslieferbaren Code*
- Phasen eines Sprints
 - Planung
 - Durchführung
 - Abschluss



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



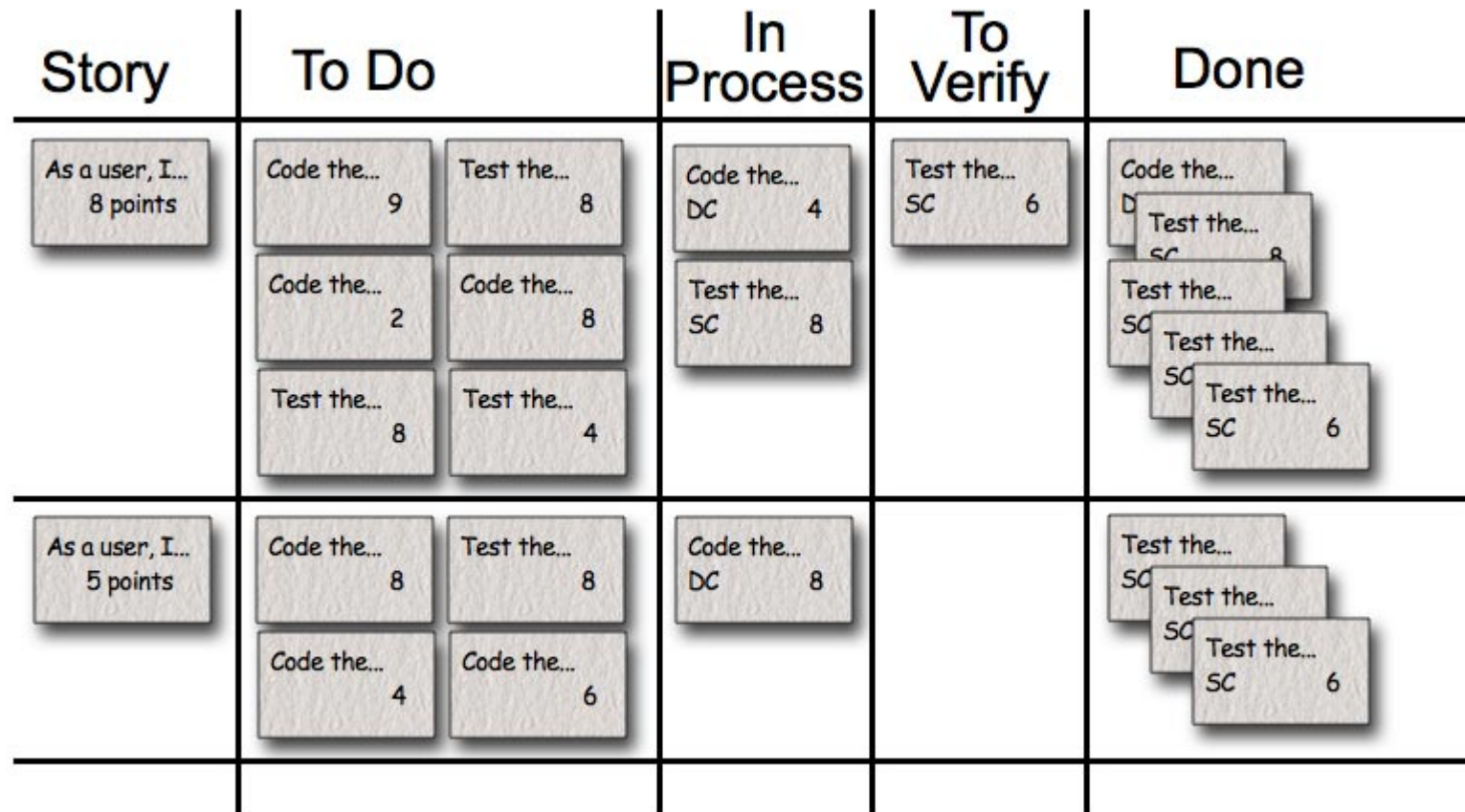
Daily Scrum

- 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“

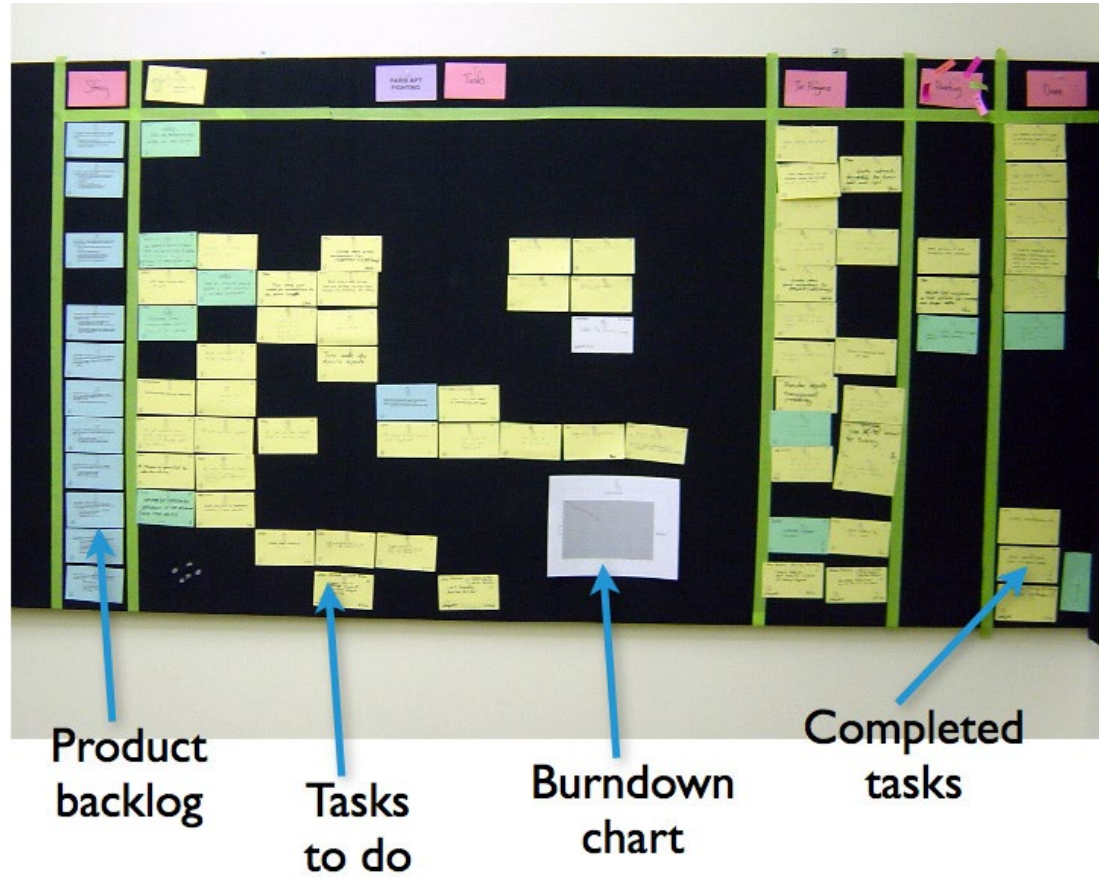


Scrum – Task Board

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

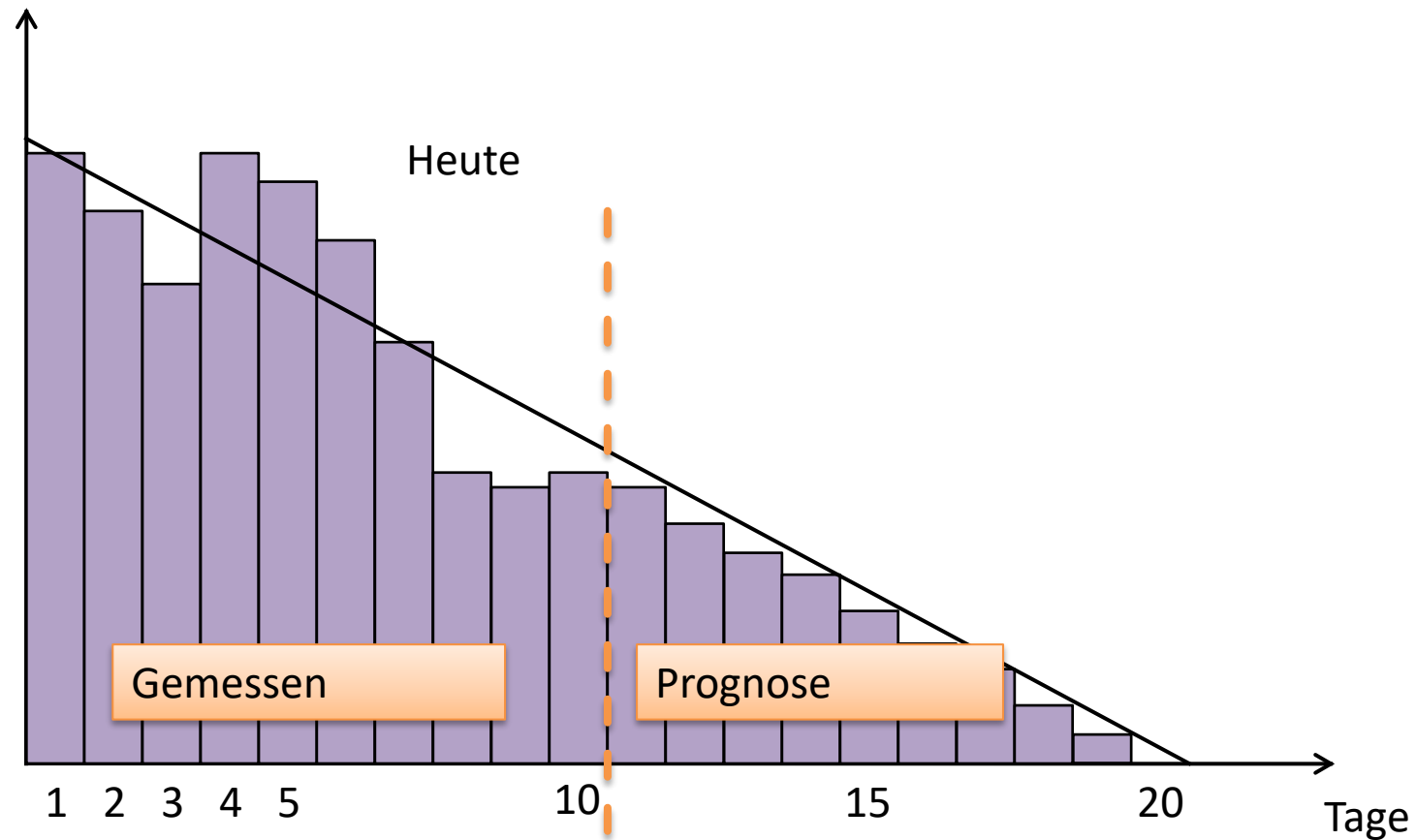


Beispiele



Sprint Burndown Charts

Offene 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

- 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

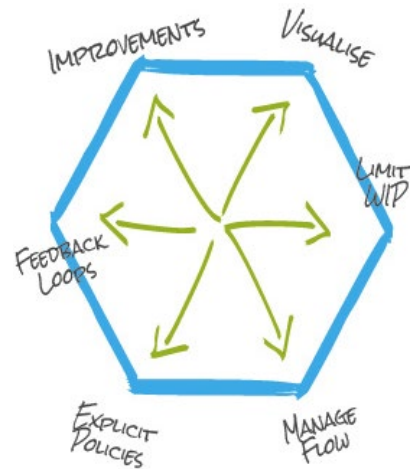
Kanban

Die Alternative zu Scrum



Ziel: Minimiere Laufende Arbeiten

- Kanban (japanisch für Hinweistafel, Anschlagswand, etc.)
- Visuelles Prozessmanagement
- Aus Automobilindustrie (Toyota) um „Just-In-Time Production“ sicher zu stellen



<https://www.agil8.com/blog/six-practices-kanban/>

Visuelles Projektmanagement

Pool of Ideas	Feature Preparation		Feature Selected	User Story Identified	User Story Preparation		User Story Development		Feature Acceptance		Deployment	Delivered
Epic 431	3 - 10		2 - 5	30	15		15		8		5	Epic 294
Epic 478	In Progress	Ready			In Progress	Ready	In Progress	Ready (Done)	In Progress	Ready		
Epic 562	Epic 444	Epic 662	Epic 602			Story 602-02 Story 602-03	Story 602-06 Story 602-04	Story 602-05 Story 602-01	Epic 401	Epic 609	Epic 694	Epic 386
Epic 439	Epic 589		Epic 302	Story 302-05 Story 302-02	Story 302-01 Story 302-06	Story 302-07 Story 302-08	Story 302-09	Story 302-05 Story 302-04	Epic 468	Epic 577	Epic 276	Epic 419
Epic 329	Epic 651		Epic 335	Story 335-09 Story 335-08	Story 335-10 Story 335-01	Story 335-04 Story 335-03	Story 335-05 Story 335-02	Story 335-06 Story 335-07	Epic 362		Epic 339	Epic 388
Epic 287			Epic 512	Story 512-04 Story 512-05	Story 512-07 Story 512-06	Story 512-02 Story 512-03	Story 512-01				Epic 521	Epic 287
Epic 606	Discarded										Epic 582	Epic 274
	Epic 511	Epic 213										
	Epic 221											

Policy

Business case showing value, cost of delay, size estimate and design outline.

Policy

Selection at Replenishment meeting chaired by Product Director.

Policy

Small, well-understood, testable, agreed with PD & Team

Policy

As per "Definition of Done" (see...)

Policy

Risk assessed per Continuous Deployment policy (see...)

Kernpraktiken I

- Visualisiere:
 - Geteiltes Kanban-Board für alle Aufgaben, Prozessschritte und Mitarbeiterinnen (siehe Folie zuvor)
- Kanban Limits: Limitiere „Work in Progress“
 - Begrenze Anzahl an Tickets pro Prozessschritt
 - Fokussiere auf die produktivste Aufgabe im Projekt
 - Pull-Ansatz: Nimm Item aus vorheriger Spalte
 - Reduziere Context-Switches
- Manage Flow:
 - Messe die Länge der Warteschlange, Mittlere Durchlaufzeit
 - Identifiziere Flaschenhälse zur besseren Planung

Kernpraktiken II

- Explizite Vorgaben:
 - Explizite Regeln, Vorgaben für alle über Bedeutung der Prozesse, wann der Transfer stattfindet in eine andere Spalte, etc.
- Feedback Loops:
 - Kontinuierliche Verbesserung durch Einbau von Mentoring, nicht zeitlich festgelegte Retrospektiven (auch zwischen Teams)
- Verbesserungen:
 - Keine neue Rollen und Verantwortlichkeiten am Anfang benötigt, aber kontinuierliche, inkrementelle Verbesserungen als Ziel, die auch Organisationen verändern können

Gemeinsamkeiten mit Scrum

- Beide Projektmanagementmethoden unterstützen das Agile Manifesto und Lean Software Development
- Reagieren flexibel auf Änderungen
- Fokus auf Qualität, klarer Zeitpunkt der Fertigstellung
- Beide verwenden das Pull-Prinzip:
 - Scrum für Projektplanung in Kanban für das Board
- Selbstorganisierende Teams
- Inkrementelle Softwareentwicklung mit auslieferbarer SW

Unterschiede Generell

- Kanban
 - Konzentration auf Visualisierung von Aufgaben
 - Begrenzen von laufenden Aufgaben
 - Optimierung der Effizienz
 - Projekt (oder User Story) möglichst schnell abschließen
- Scrum
 - Fixe Intervalle zur Auslieferung neuer Softwareinkremente
 - Direktes und schnelles Feedback von Kunden und Lernschleife
 - Regelmäßige Zeremonien, feste Rollen zur Einbeziehung des Kunden

Unterschiede zu Scrum I

Teams	Scrum	Kanban
Rollen	Fest (Product Owner, Scrum Coach)	Keine initialen Rollen vorgeschrieben
Größe	3-9 Mitglieder, cross-funktional, kollaborativ	Keine Vorgaben, cross-funktional oder spezialisiert

Meetings	Scrum	Kanban
TimeBox	Daily standup	Keine Vorgaben
Austausch	Team-Retrospektive nach Sprint	Keine Vorgaben
Steering	Review-Meeting nach Sprint	Keine Vorgaben
Forecast	Preplanning vor Sprint	Regelmäßige Nachschubmeetings

Unterschiede zu Scrum II

Artefakte	Scrum	Kanban
Anforderungen	Product Backlog	Backlog auf Board
Lieferzyklus	Sprint	Durchlaufzeit des Tickets
Board	Scrumboard (neu bei jedem Sprint)	Kanban-Board (dauerhaft)
Lieferung	Potential auslieferbares Produkt	Abgearbeitetes Ticket
Metriken	Velocity	Lead Time, Cycle Time, etc.

Kanban eher für Wartung und Evolution, wo keine komplexe Lösung erforderlich ist und kontinuierlich Verbesserungen erreicht werden sollen.

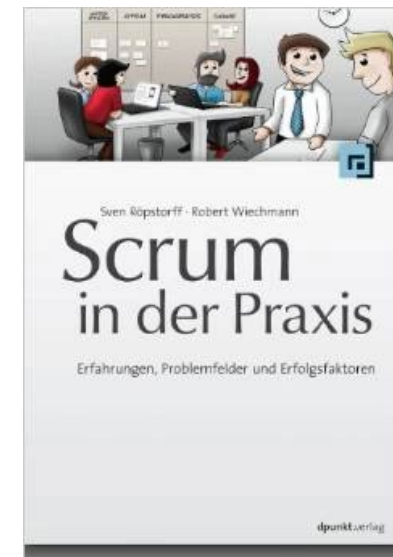
Scrum konzentriert sich auf komplexen SW-Entwicklungsprozess, um größere SW mit interdisziplinären Teams zusammen mit der Kundin zu entwickeln

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?
- Kennen Sie die Unterschied von Scrum und Kanban, um bei einen gegebenen Szenario die vielversprechendste Methode einzusetzen?
- Worin liegt der Unterschied zu Extreme Programming?

Literatur

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



Literatur 2

- Agile!: The Good, the Hype and the Ugly. Bertrand Meyer. Springer Publishing Company, Incorporated, 2014. ISBN 9783319051543.
- Stop Starting, Start Finishing!. Arne Roock. Illustrated Edition. Blue Hole Press, 2012. ISBN 9780985305161.
- Kanban: Successful Evolutionary Change for Your Technology Business. David J. Anderson. Blue Hole Press, 2010. ISBN 0984521402.
- Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise. Mark Lines, Scott W. Ambler. IBM Press, 2012. ISBN 978-0-13-281013-5