Technische Universität Berlin

Softwaretechnik und Programmierparadigmen

03 Einführung

Prof. Dr. Sabine Glesner
Software and Embedded Systems Engineering
Technische Universität Berlin

SESE

Inhalt

Einführung in die Softwaretechnik

- Grundlagen der Softwaretechnik
- Softwareentwicklungsprozesse
- Lernziele und Literatur

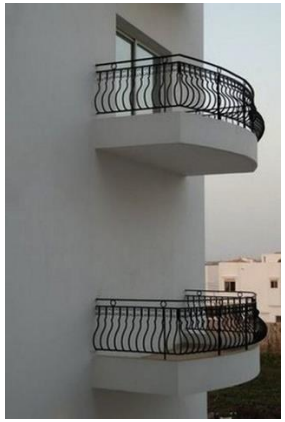
Inhalt

Einführung in die Softwaretechnik

- Grundlagen der Softwaretechnik
- Softwareentwicklungsprozesse
- Lernziele und Literatur

Motivation

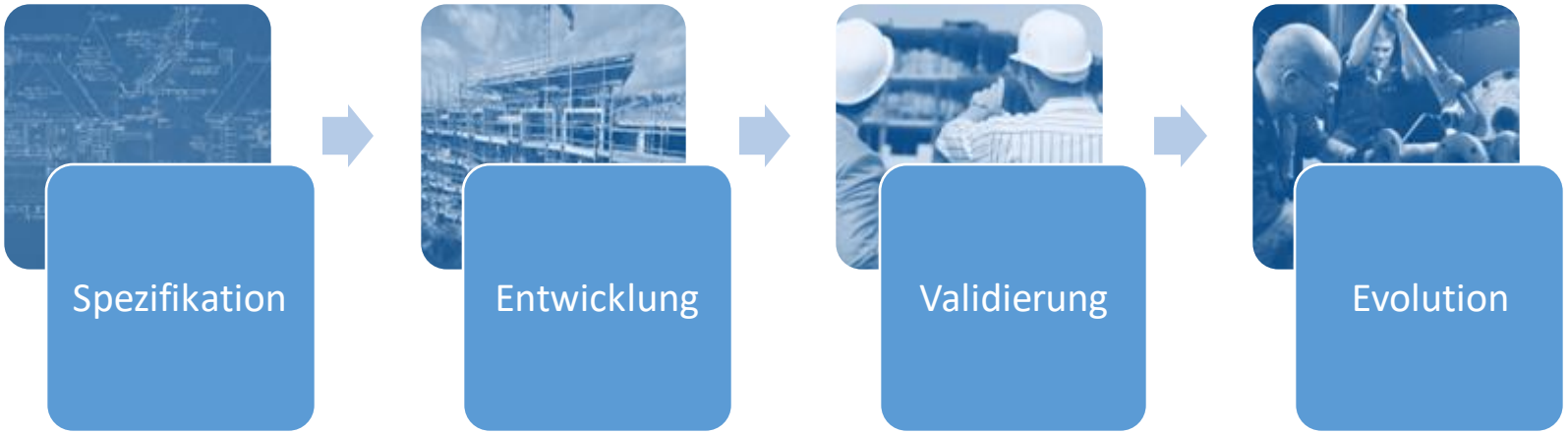
Planlos programmieren ist wie planlos bauen.



Nicht so clever.

Einführung

Stattdessen nähern wir uns der Lösung **schrittweise** in mehreren Phasen



Diese Phasen und ihre Verwaltung sind die **grundlegenden Aktivitäten** des *Software-Engineering*

Softwaretechnik

„Software-Engineering ist eine **technische Disziplin**, die sich mit **allen Aspekten** der Softwareherstellung beschäftigt.“
Ian Sommerville

Prozessaktivitäten

- Spezifikation, Entwicklung, Validierung, Evolution (Weiterentwicklung)

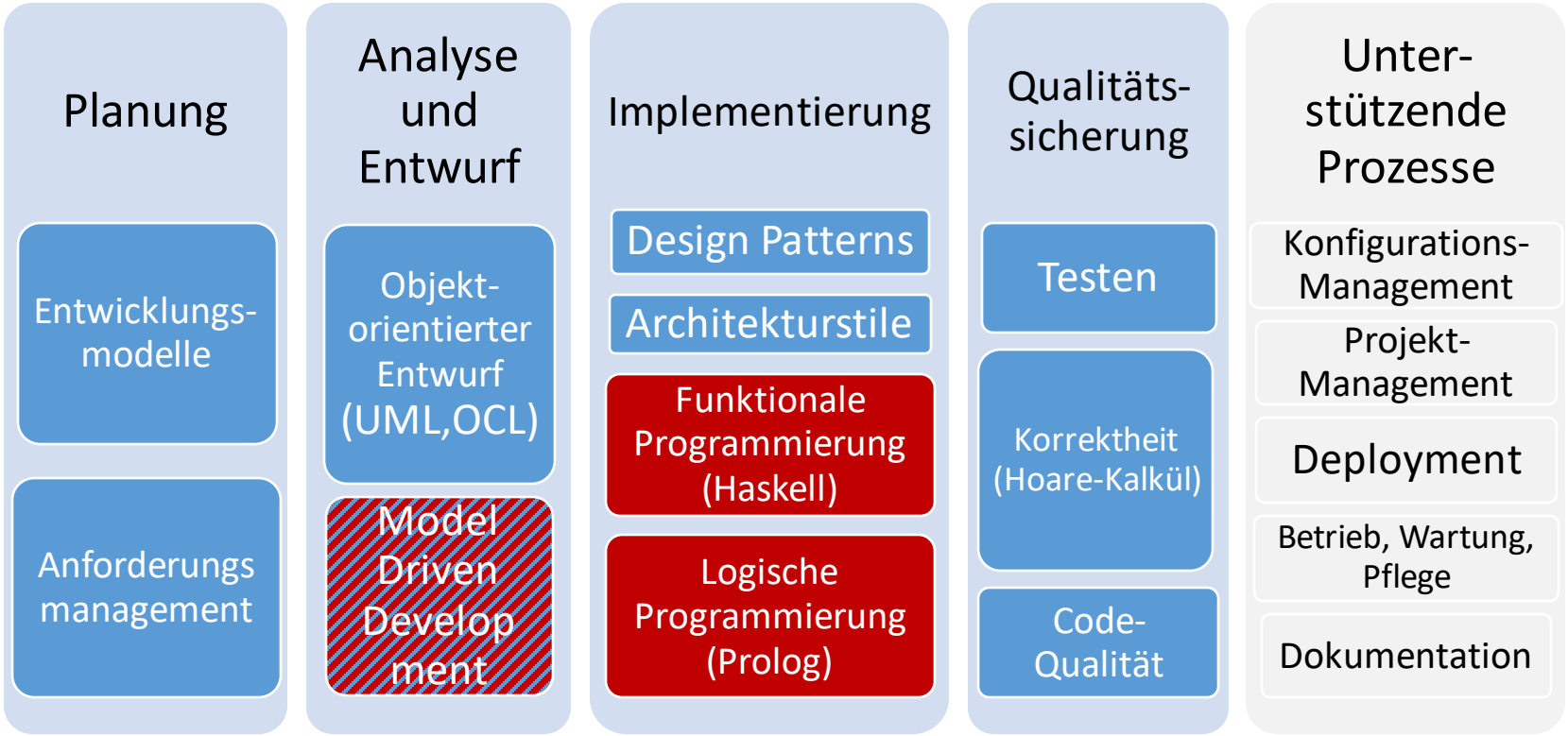
Softwareentwicklungsprozesse

- Reihenfolge und Häufigkeit der Prozessaktivitäten

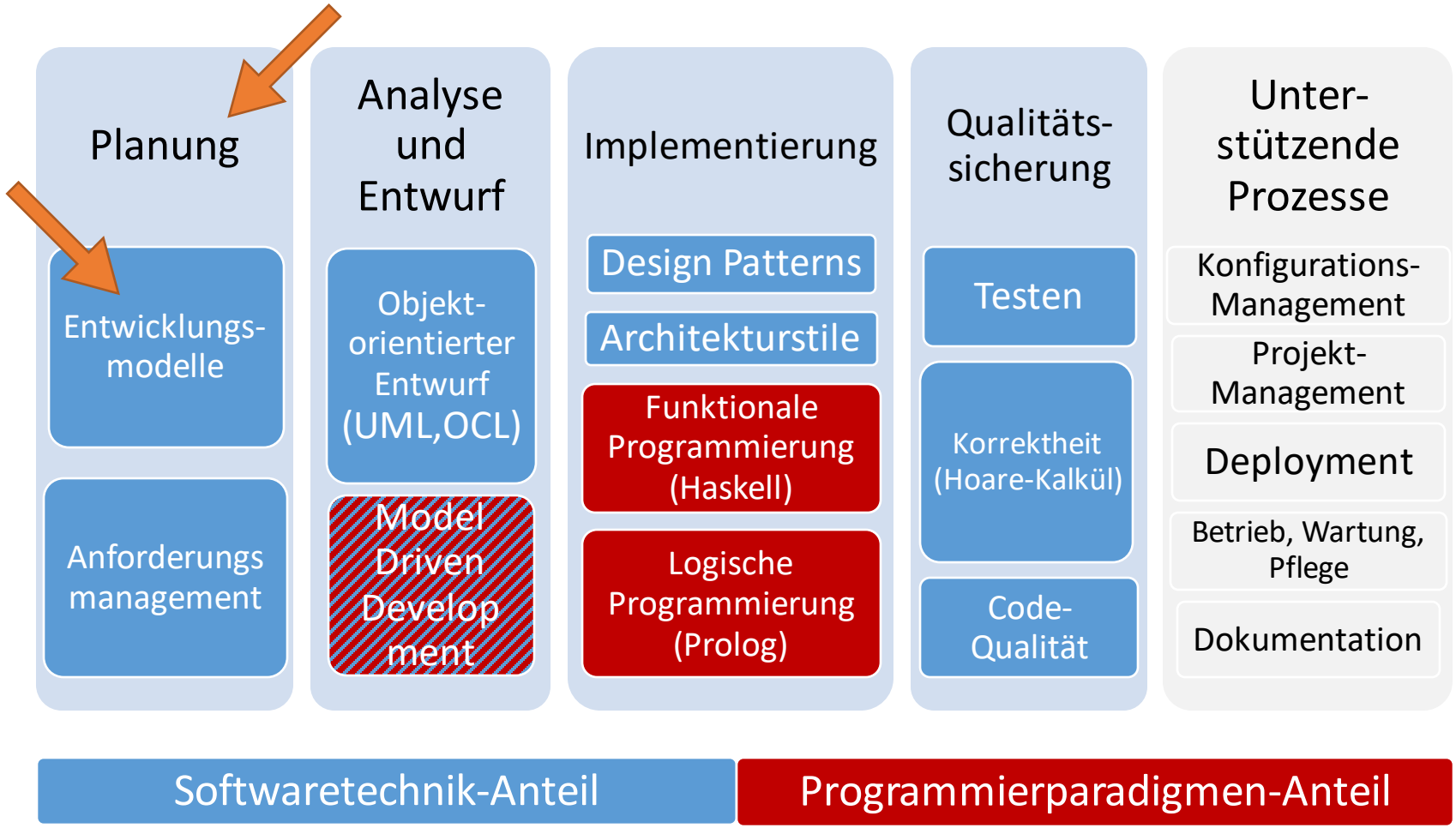
Methoden und Werkzeuge der Softwareentwicklung

- Anforderungsmanagement, Objekt-orientierter Entwurf, Design Patterns
- Software-Metriken, Testen und Verifikation

Programm in diesem Semester



Diese VL


Planung
Entwicklungs-modelle
Anforderungs management
Analyse und Entwurf
Objekt-orientierter Entwurf (UML,OCL)
Model Driven Develop ment
Implementierung
Design Patterns
Architekturstile
Funktionale Programmierung (Haskell)
Logische Programmierung (Prolog)
Qualitäts-sicherung
Testen
Korrektheit (Hoare-Kalkül)
Code-Qualität
Unter-stützende Prozesse
Konfigurations-Management
Projekt-Management
Deployment
Betrieb, Wartung, Pflege
Dokumentation
Softwaretechnik-Anteil
Programmierparadigmen-Anteil

Inhalt

Einführung in die Softwaretechnik

- Grundlagen der Softwaretechnik
- Softwareentwicklungsprozesse
- Lernziele und Literatur

Softwareentwicklungsprozesse

Ausgehend von informellen Anforderungen werden die **grundlegenden Aktivitäten** in einem **Softwareentwicklungsprozess** kombiniert.

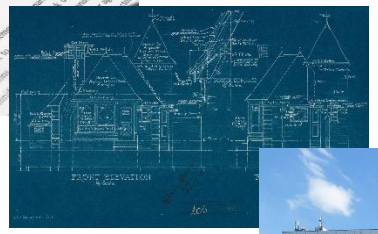
Spezifikation	Entwicklung	Validierung	Evolution
• Definition der zu entwickelnden Software und ihrer Einsatzbedingungen	• Entwurf und Programmierung der Software	• Überprüfung der Software auf Erfüllung der Anforderungen	• Weiterentwicklung und Anpassung an geänderte Anforderungen

Verschiedene Softwareentwicklungsprozesse legen den **Schwerpunkt** auf verschiedene Aspekte der Softwareentwicklung.

Es gibt keinen idealen Prozess für alle Situationen!

Beispiel

Anforderung



Spezifikation

Entwicklung

Validierung

Evolution

Sequenzielle Ausführung

Das Wasserfallmodell

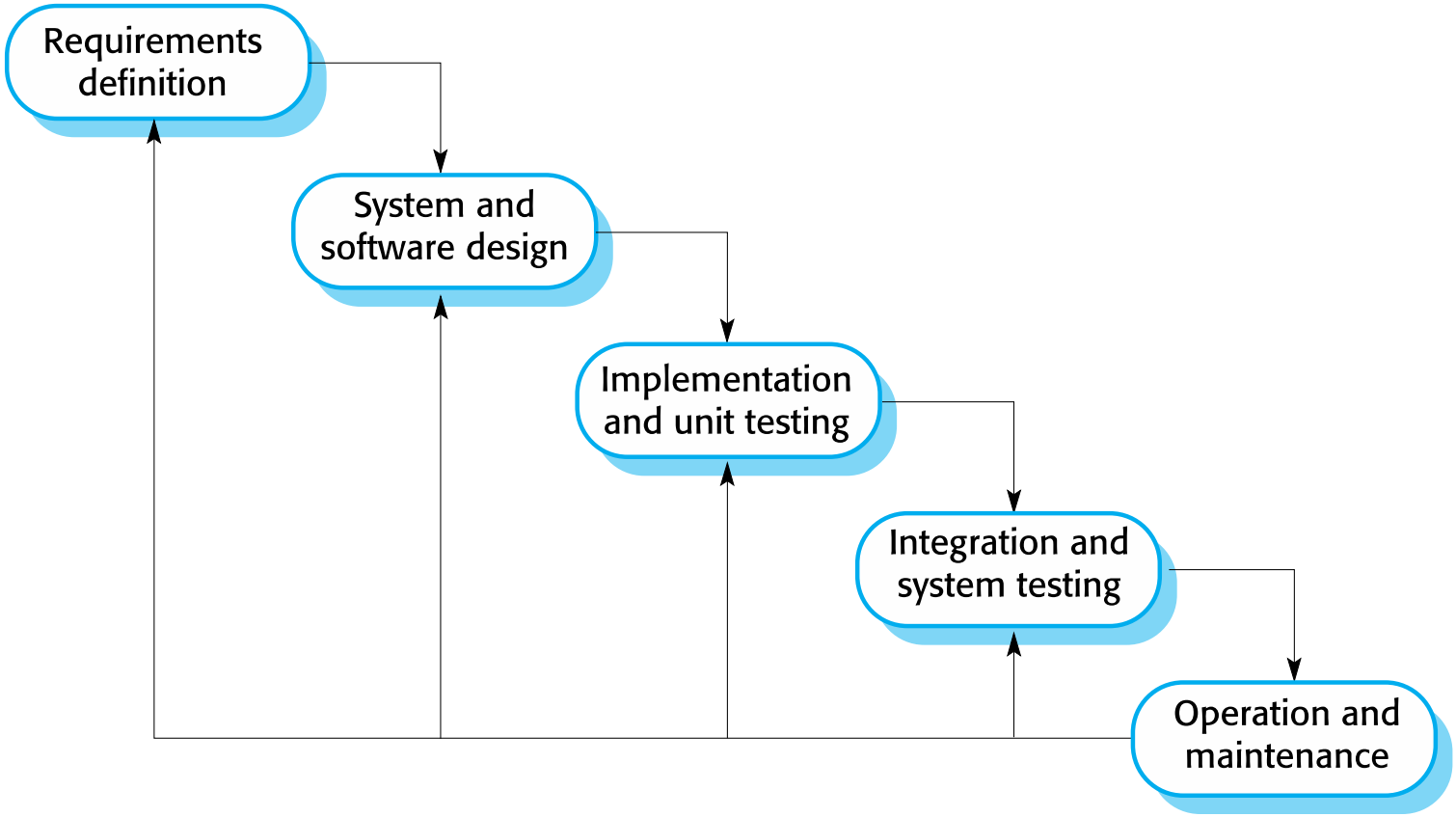
Aufteilung des Entwicklungsprozesses in feste, **sequentielle** Phasen

- *Vorteile*
  Klare Abgrenzung der einzelnen Phasen…
  …ermöglicht einfache Koordinierung der Arbeitsschritte

- *Nachteile*
  Umfangreiche Planung nötig
  Auf **Veränderungen** kann während des Prozesses schlecht reagiert werden…
  …darum müssen die Anforderungen weitestgehend stabil sein
  Fehler werden erst sehr spät erkannt (späte Integration und Systemtests)

Geeignet für stabile Großprojekte mit mehreren Entwicklungsteams

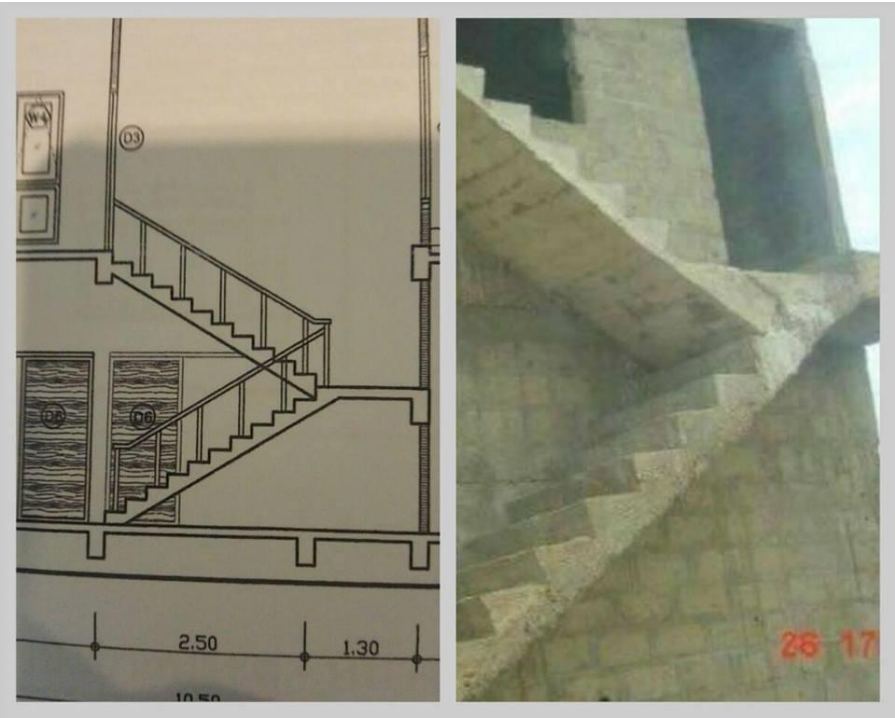
Das Wasserfallmodell



Ian Sommerville, Software-Engineering, Chapter 2

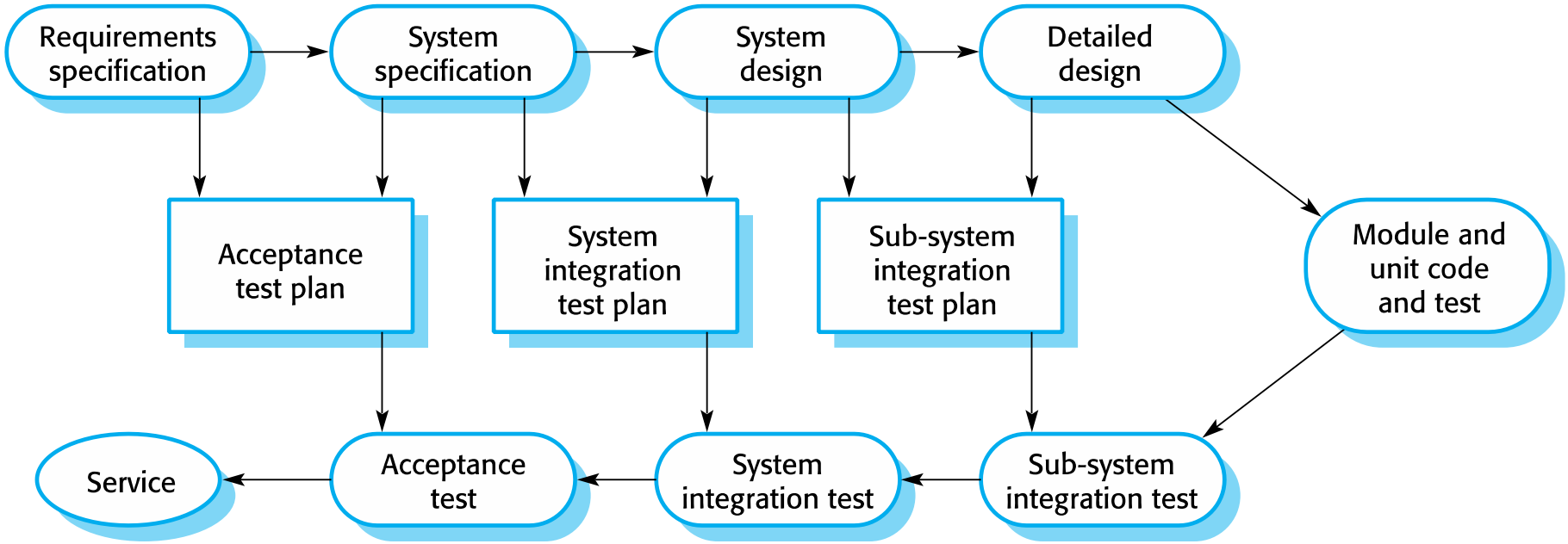
Validierung

Alles gut geplant, aber nur Quatsch implementiert…



…was nun?

Das V-Modell



Ian Sommerville, Software-Engineering, Chapter 2

Das V-Modell

Erweitert das Wasserfallmodell um vorher geplante **Testaktivitäten**

- Stärkerer Fokus auf die **Validierungsphase**

- *Entwickler-/Komponententests*

  Komponenten werden unabhängig von anderen Komponenten getestet

  Oftmals durch den Entwickler selbst, um Programmierfehler zu finden

- *Systemtest*

  Das System wird vollständig integriert getestet, eventuell mehrstufig

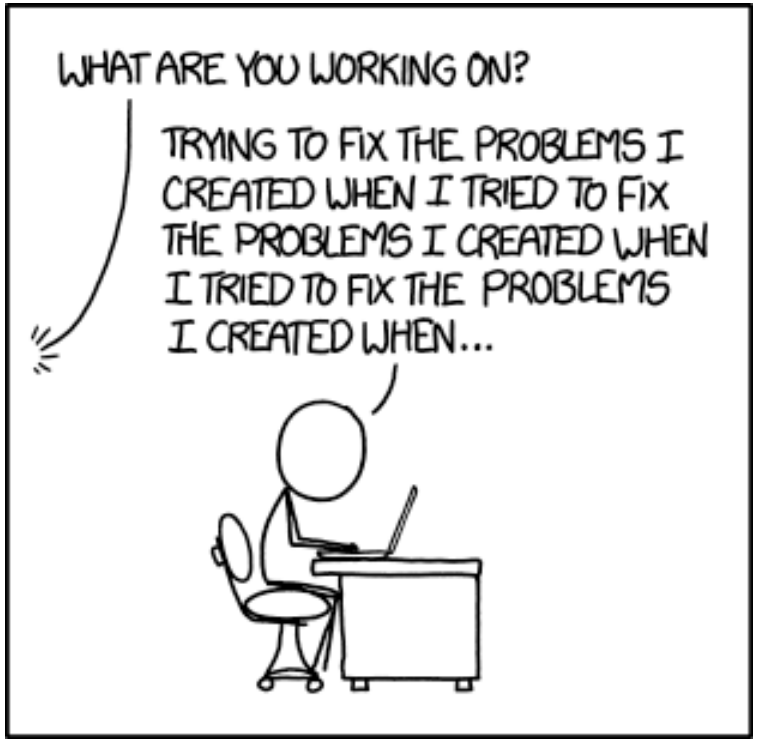
  Dient dem Auffinden von Schnittstellenproblemen und Fehlern in der Interaktion von Komponenten

- *Abnahmetest (Alpha-Test)*

  Testen des Gesamtsystems mit vom Kunden zur Verfügung gestellten Testdaten

  Dient der direkten Validierung mit dem Kunden

Evolution und Wiederverwendung

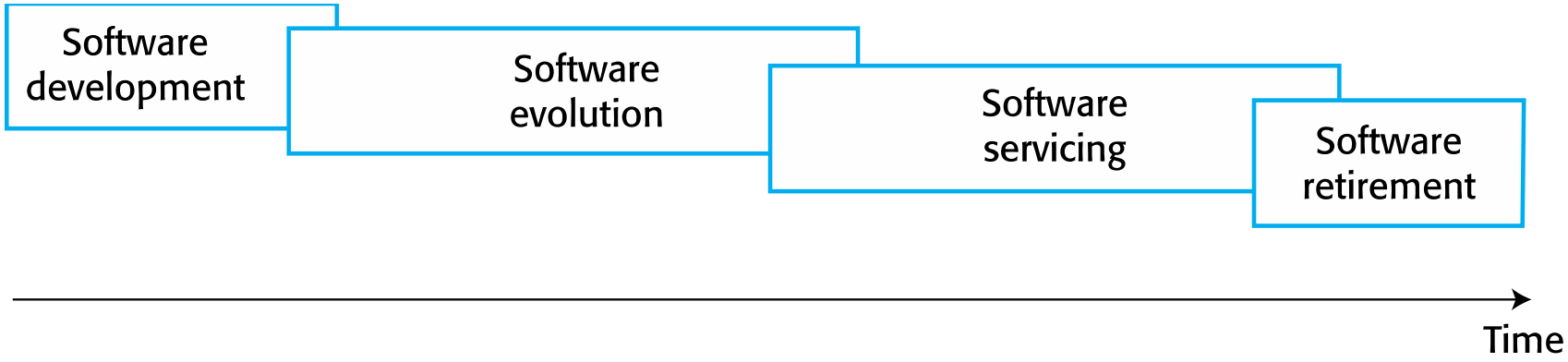


XKCD

Evolution und Wiederverwendung

Ein Großteil der Softwarekosten werden bei der **Evolution** verursacht.

- Änderung im Anforderungsprofil, aufgetretene Fehler, nachlassende Leistung, neue Hardwarebedingungen etc. erzwingen nachträgliche **Anpassungen** und **Verbesserungen** der Software



Während der späteren Instandhaltungsphase werden nur noch kleine, unerlässliche Änderungen durchgeführt.

Ian Sommerville, Software-Engineering, Chapter 9

Wiederverwendungsorientierte Prozesse

Während der Entwicklung entstehen zunächst **versunkene Kosten**

- Bis zur **Fertigstellung** ist die Software größtenteils wertlos

Die Entwicklung neuer Komponenten kann beschleunigt (und damit günstiger) werden, indem Komponenten **wiederverwendet** werden.

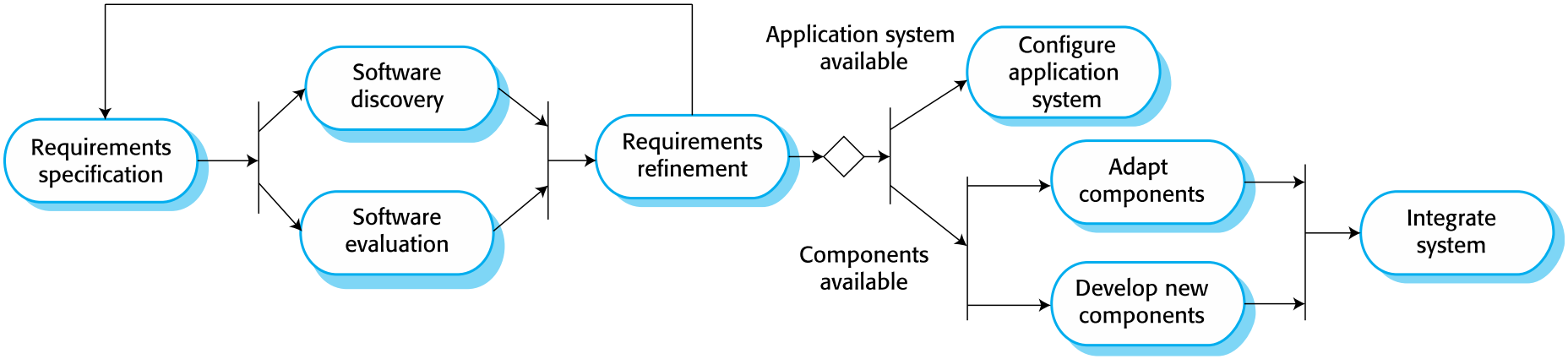
- *Analyse der Komponenten*

  Aus den bestehenden Komponenten werden zur Anforderungsspezifikation passende ausgesucht
- *Anpassung der Anforderungen*

  Eventuell werden die Anforderungen bestehende Komponenten angepasst
- *Ergänzung und Integration der Komponenten*

  Fehlende Komponenten werden neu erstellt und zusammen mit den angepassten Komponenten zum Gesamtsystem integriert

Wiederverwendungsorientierte Prozesse

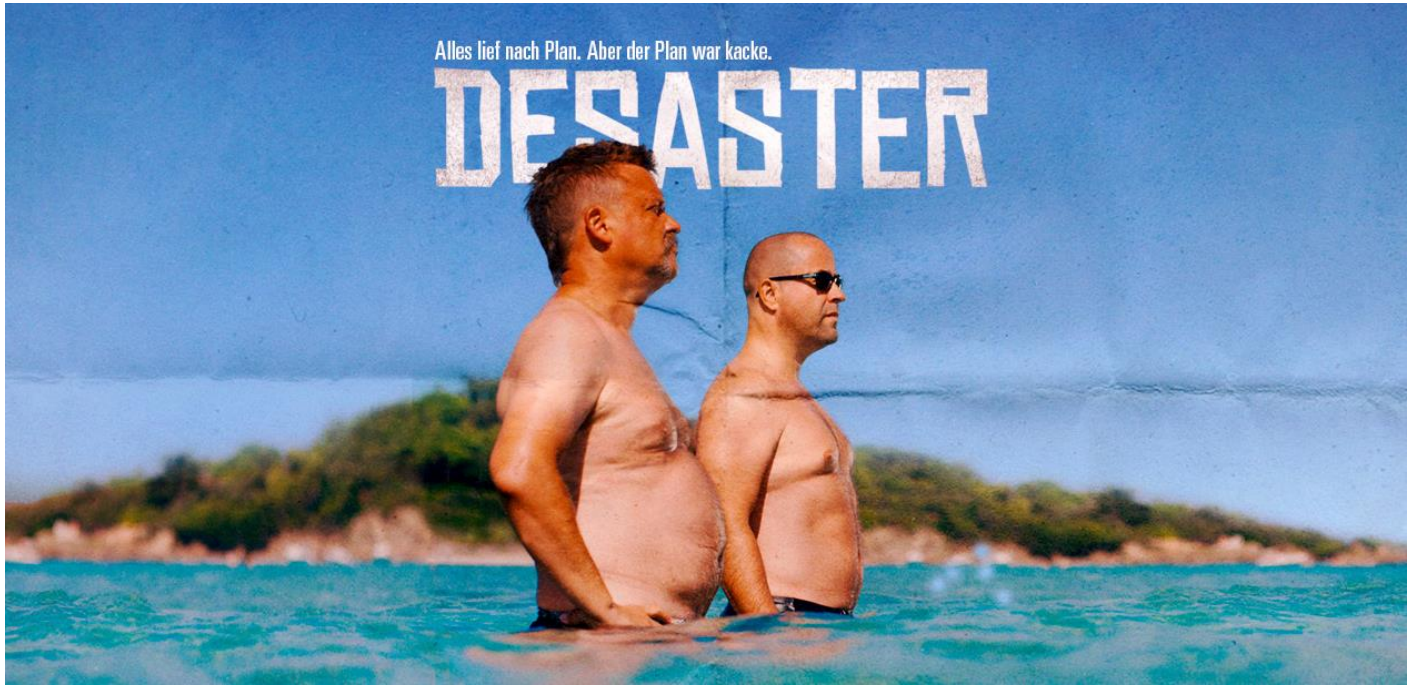


Ian Sommerville, Software-Engineering, Chapter 2

Inkrementelle Entwicklung

Alle genannten Prozesse benötigen frühe, gute Planung…

…und was mache ich, wenn ich mich voll *verplant* hab‘?



Spiralmodell nach Boehm

Inkrementelles Entwicklungsmodell mit frühen Prototypen

- Besonderer Fokus auf die **Transparenz** des Entwicklungsprozesses

Die Entwicklung wird durch eine **Risikobewertung** gesteuert

- In jeder Iteration werden die Risiken des Projekts **neu bewertet**
- Kann ein Risiko nicht **beseitigt** werden, gilt das Projekt als gescheitert

- *Vorteile*

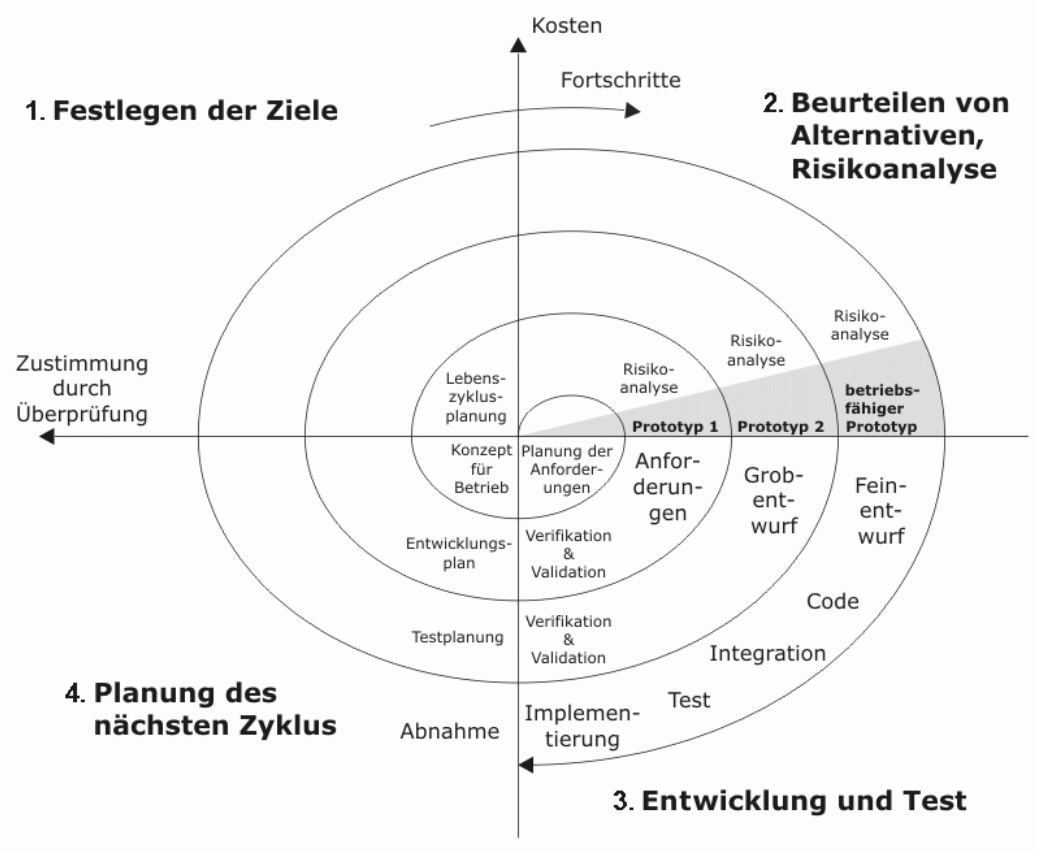
  Iterative Entwicklung reduziert Risiko

  Frühes Testen (auch durch Benutzer) von Prototypen

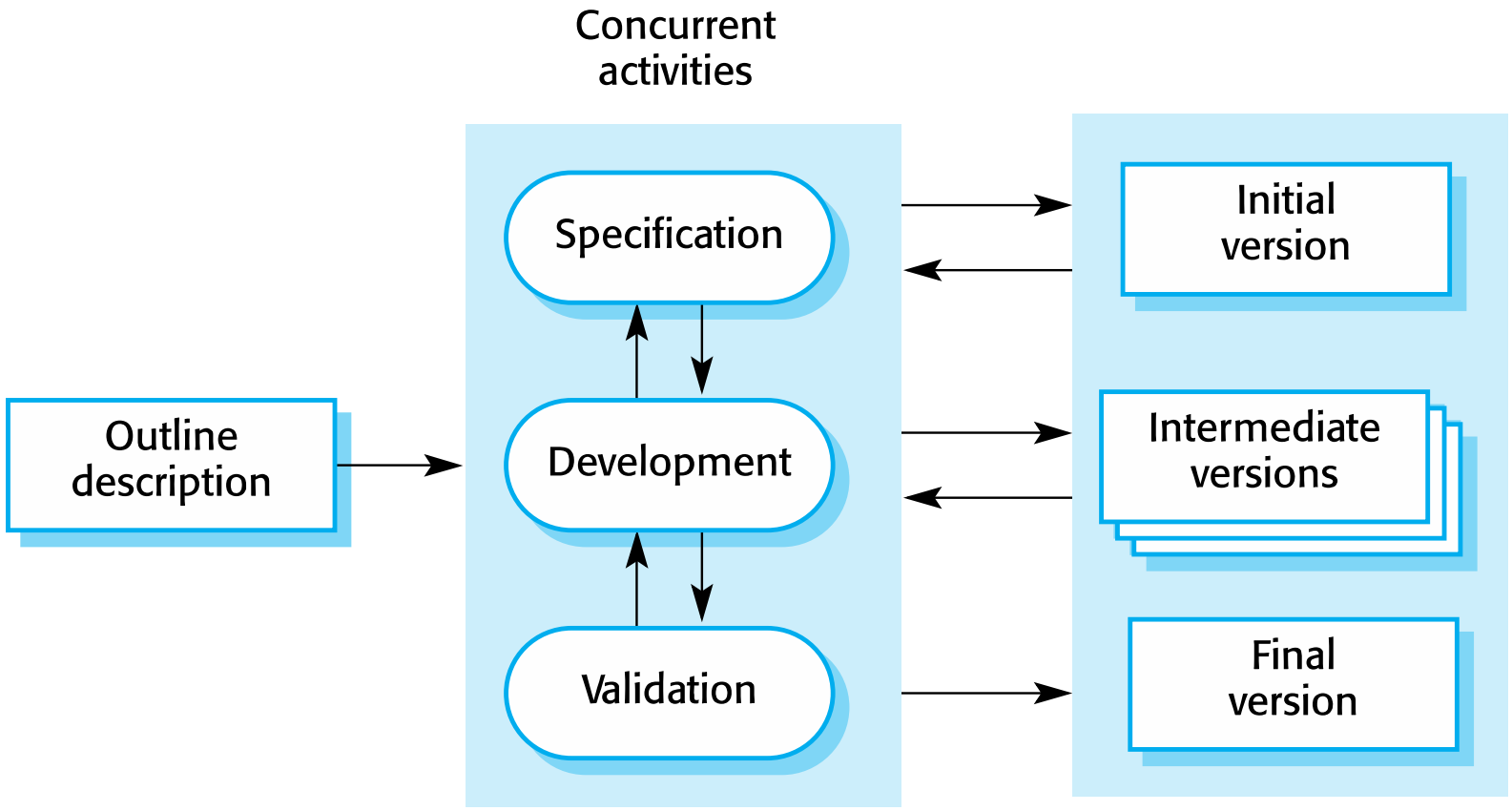
- *Nachteile*

  Risikoanalyse kann mitunter sehr teuer sein

Spiralmodell nach Boehm



Inkrementelle Entwicklung



Ian Sommerville, Software-Engineering, Chapter 2

Inkrementelle Entwicklung

Spezifikation, Entwicklung und Validierung werden **parallel** ausgeführt

- Zwischen den einzelnen Aktivitäten werden Rückmeldungen ausgetauscht, um Fehler früh zu beheben

Durch frühe Prototypen (Anfangsimplementierung) können Kundenwünsche während der Entwicklung berücksichtigt werden

- Bis zur finalen Version werden mehrere Zwischenversionen erstellt

- *Probleme der Inkrementellen Entwicklung*

  Bürokratische Umgebungen harmonieren nicht gut mit informelleren Prozessen (z.B. sicherheitskritische Software)

  Der Gesamtfortschritt ist nicht anhand eines starren Planes messbar

Inkrementelle Entwicklung ist das Fundament **agiler Entwicklung**!

Agile Entwicklungsprozesse

Agile Entwicklungsprozesse sind Prozesse mit **sehr kleinen Inkrementen** und **vielen Prototypen**, die den agilen Grundprinzipien folgen.

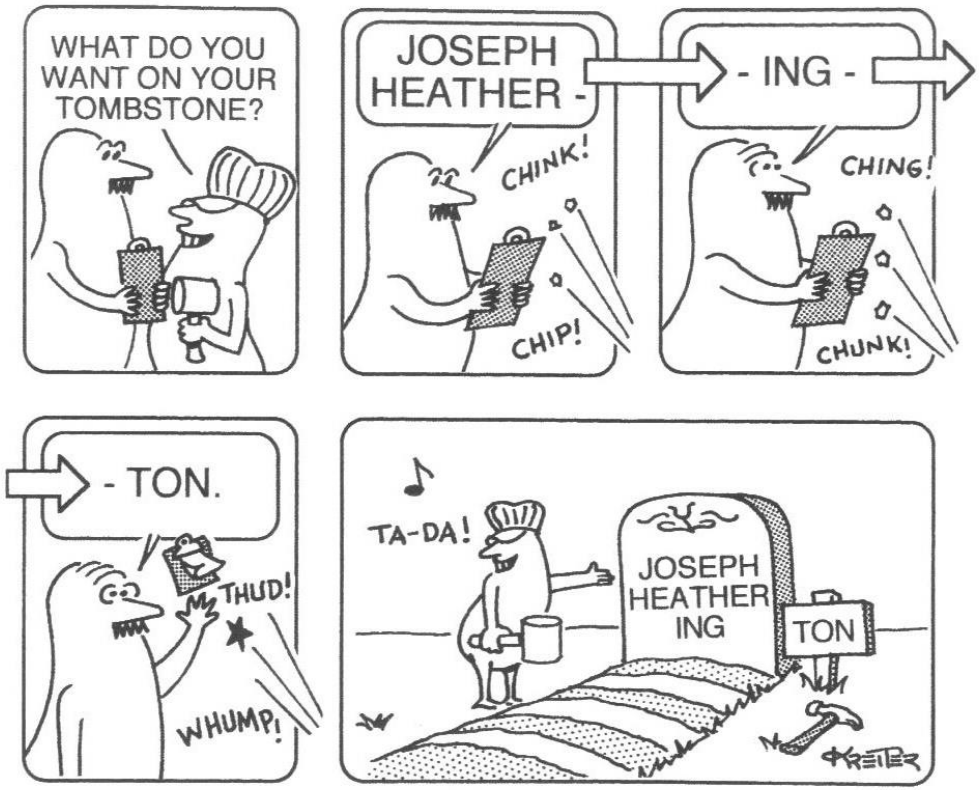
Beispiele

- Scrum
- Extreme Programming
- Kanban (der Softwareentwicklung)
- …

"**What's wrong** with the analogy between software and building construction?"
Frage von oscarkuo auf StackOverflow.com

„[Architect's clients] can't take delivery of your **finished** underground carpark and ask you **to add** an airport (also underground)"
Antwort von Martin Beckett

Wozu agile Entwicklungsprozesse?



Agile Entwicklungsprozesse ermöglichen **veränderliche Anforderungen**

Was ist agile Softwareentwicklung?

2002 prägten die „Snowbird 17“ die agile Softwareentwicklung mit ihrem

agilen Manifest

Individuen und Interaktionen über Prozesse und Werkzeuge

Funktionierende Software über umfassende Dokumentation

Zusammenarbeit mit dem Kunden über Vertragsverhandlung

Reagieren auf Veränderung über das Befolgen eines Plans

https://agilemanifesto.org/

Agile Prinzipien

https://agilemanifesto.org/

Aus dem agilen Manifest folgen 12 agile Grundprinzipien

Zufriedenstellung des Kunden durch frühe Auslieferung	Änderungen von Anforderungen sind willkommen	Funktionierende Software in regelmäßigen kurzen Zeitspannen	Tägliche Zusammenarbeit von Entwicklern und Experten
Vertrauen in und Unterstützung der Individuen im Projekt	Übermittlung von Informationen von Angesicht zu Angesicht	Das wichtigste Fortschrittsmaß ist funktionierende Software	Gleichmäßiges Tempo auf unbeschränkte Zeit
Ständiges Augenmerk auf technische Exzellenz und gutes Design	Einfachheit ist essenziell	Selbstorganisierte Teams	Team reflektiert regelmäßig bzgl. Effizienz und passt Verhalten an

https://agilemanifesto.org/

Exkurs: Push- vs. Pull-Prinzip

Konzepte aus der Logistik und Produktion

Push

- Produktion nach festgelegten Mengen / Zeiträumen
- Produkte werden in den Markt *hineingedrückt*

Pull

- Nachfrage bestimmt Produktion
- Produkte werden erst produziert (*pull*), wenn die Nachfrage existiert

Agile Prozesse orientieren sich meist am Pull-Prinzip.

Entwickler bestimmen die Menge der umsetzbaren Anforderungen.

Scrum

Eurosport

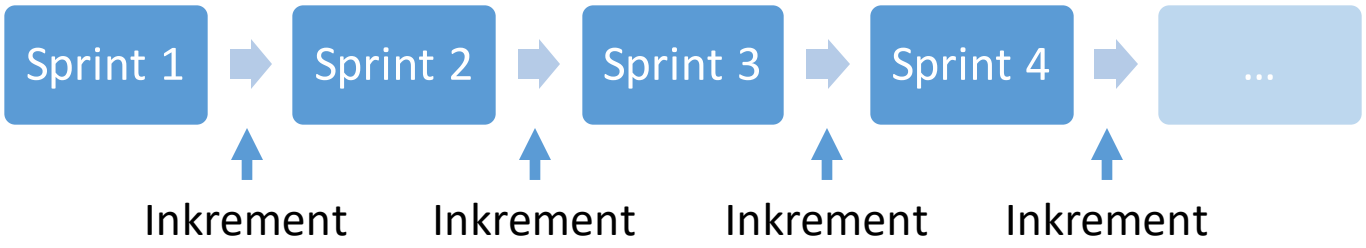
Scrum

Allgemeine Methode zur Verwaltung iterativer Prozesse

- Framework für das Projekt-Management
- Wird in vielen Bereichen neben der Softwareentwicklung eingesetzt
- Offizielle Dokumentation unter www.scrumguides.org

Sprint

- Iterative Planungseinheit
- **feste Länge**, meist 2 bis 4 Wochen
- Ergebnis: auslieferbares Produkt
- *Sprint* verfolgt ein zu Beginn festgelegtes Ziel



Scrum Artefakte

Product Backlog

- Liste aller bekannten Anforderungen an das Produkt (bspw. User Stories)
- Ist das einzige Anforderungsdokument
- Sortiert nach Mehrwert
- Ständige dynamische Anpassung (Löschen, Hinzufügen, Umsortieren)

Mehrwert

	User Story	Aufwandsschätzung	…
1	Als Kundin/-e möchte ich mich anmelden.	3	…
2	Als Administrator/-in möchte ich Kunden verwalten und Kennwörter zurücksetzen.	5	…
3	Als Kundin/-e möchte ich die Produkte im Warenkorb bezahlen, damit sie mir zugesendet werden.	9	…
…	…	…	…

Scrum Artefakte

Sprint Backlog

- Planungsdokument für einen *Sprint*
- Elemente des *Produkt Backlogs*
- Zusätzlich: Planungselemente zur Realisierung des *Sprintziels* (Beispiel: Benutzerverwaltung)

	User Story	Aufwandsschätzung	...
1	Als Kundin/-e möchte ich mich anmelden.	3	...
2	Als Administrator/-in möchte ich Kunden verwalten und Kennwörter zurücksetzen.	5	...
3	Als Kundin/-e möchte ich die Produkte im Warenkorb bezahlen, damit sie mir zugesendet werden.	9	...
...	...	...	...

Planungselement

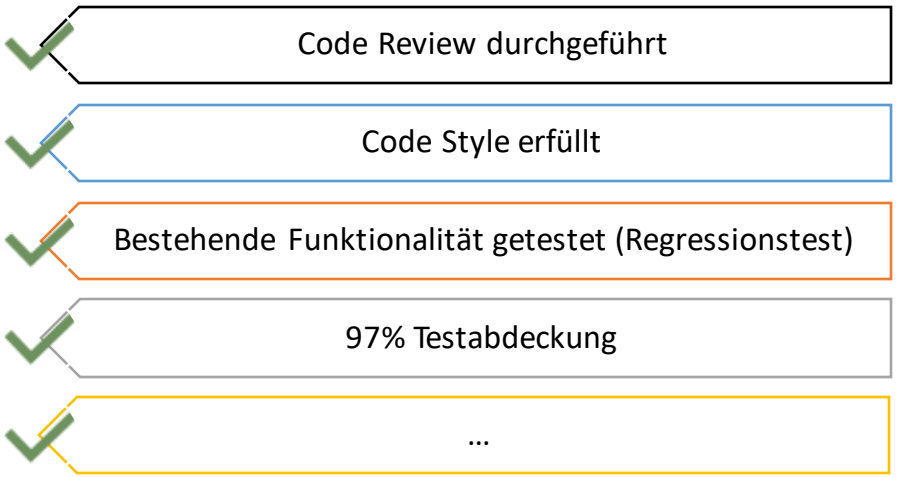
Elemente des Produkt Backlogs

	User Story	Aufwandsschätzung	...
1	Es wird eine Datenbank benötigt.	3	...
2	Als Kundin/-e möchte ich mich anmelden.	3	...
3	Als Administrator/-in möchte ich Kunden verwalten und Kennwörter zurücksetzen.	5	...
...	...	...	...

Scrum Artefakte

Definition of „Done“

- Kriterien zur Bewertung, wann Element aus Backlog „fertig“ ist
- Sollte einheitlich für mehrere *Scrum-Teams* eines Unternehmens sein



Rollen im Scrum

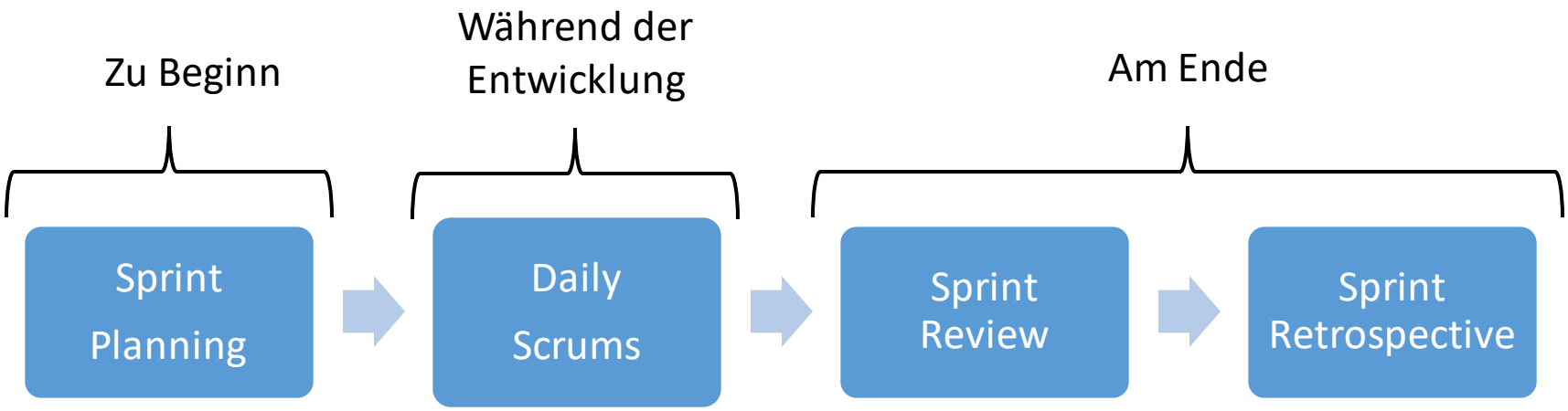
Bei Scrum steht das *Scrum-Team* im Vordergrund
- Kleines und selbstorganisiertes Team

Product Owner
- Verantwortung: **Maximierung des Produktwertes**
- Pflegt *Product Backlog*
- fachliche Schnittstelle des Teams nach außen (bspw. Kunde oder Management)

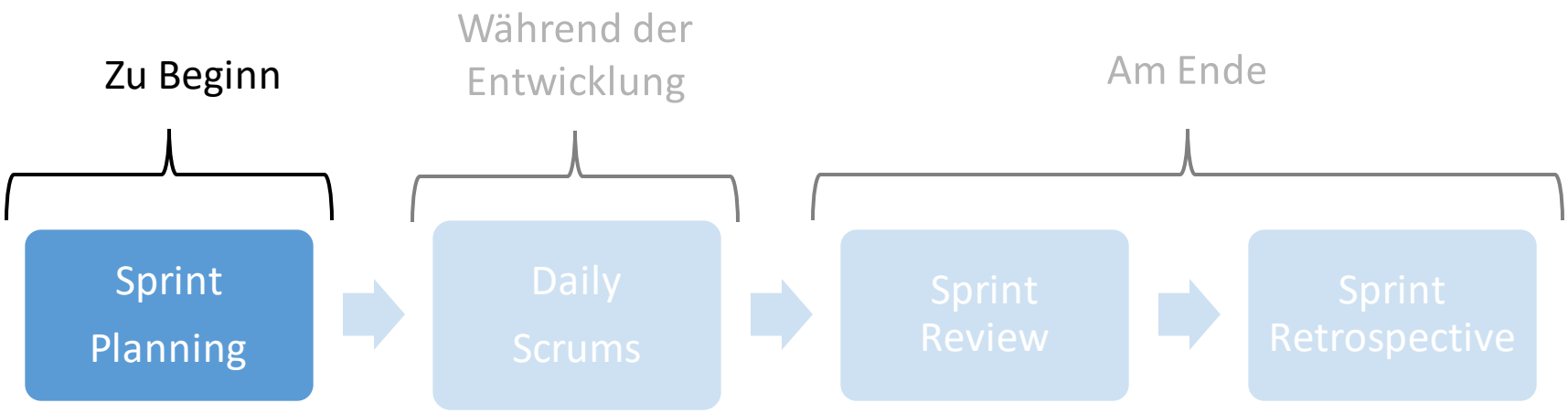
Entwicklungsteam
- Verantwortung: **Implementierung**
- Funktionsübergreifend: besitzt alle notwendigen fachliches Kompetenzen
- Keine Hierarchie, aber Berücksichtigung von Spezialisierung möglich

Scrum Master
- Verantwortung: Durchsetzung und Verbesserung des **Scrum-Prozesses**
- Übernimmt **organisatorische Aufgaben**
- Moderator und Schnittstelle eines Entwicklerteams

Sprint Meetings



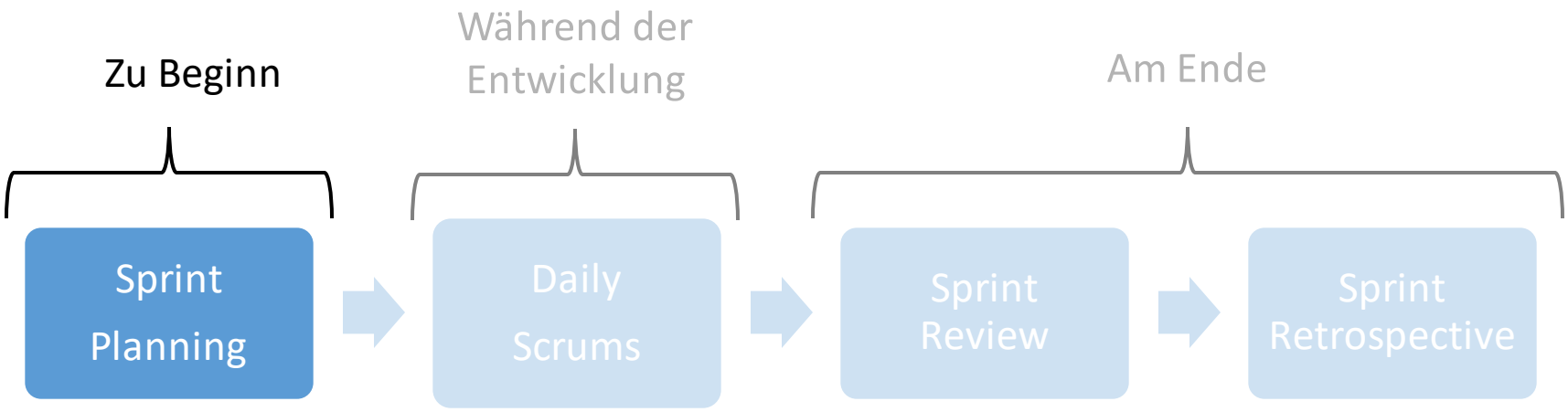
Sprint Meetings



Sprint Planung

- Teilnehmende: *Scrum-Team*
- Vorbereitung durch *Product Owner*
- Festlegung des *Sprintziels*
- Erstellen des *Sprint Backlogs* mithilfe des *Product Backlogs (Pull-Prinzip)*
- Ergebnis: *Sprintziel* und *Sprint Backlog*

Sprint Meetings



Sprint Planung – Pull Prinzip

- *Entwicklungsteam* wählt die Menge der umsetzbaren Elemente aus (nicht der *Produkt Owner*)

Product Backlog

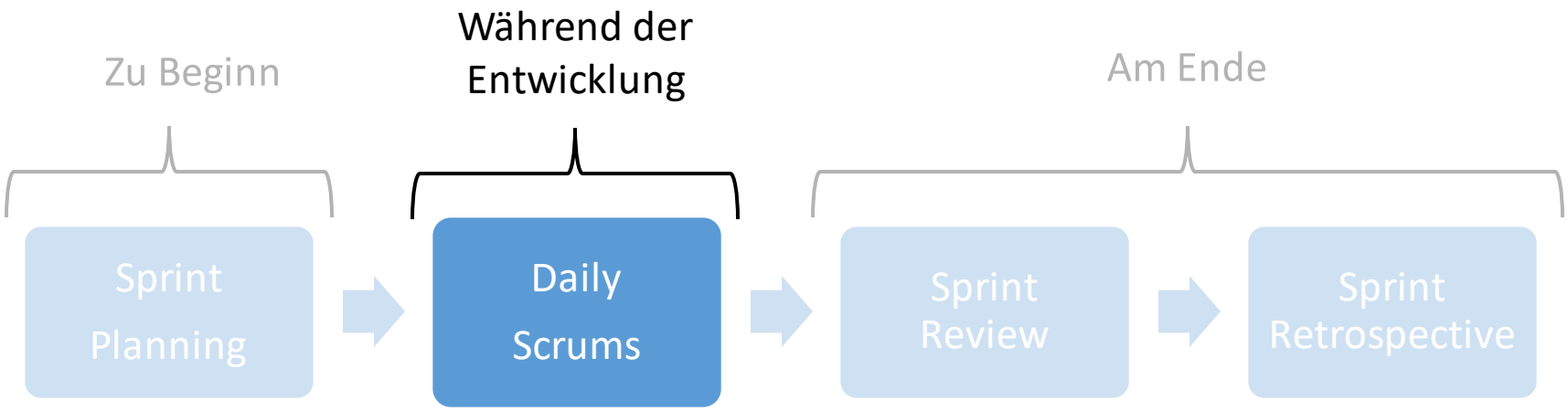
	User Story	Aufwandsschätzung	...
1	Als Kundin/-e möchte ich mich anmelden.	3	...
2	Als Administrator/-in möchte ich Kunden verwalten und Kennwörter zurücksetzen.	5	...
3	Als Kundin/-e möchte ich die Produkte im Warenkorb bezahlen, damit sie mir zugesendet werden.	9	...
...	...	...	...

Pull

Sprint Backlog

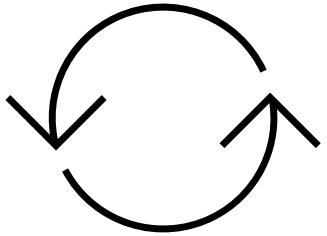
	User Story	Aufwandsschätzung	...
1	Es wird eine Datenbank benötigt.	3	...
2	Als Kundin/-e möchte ich mich anmelden.	3	...
3	Als Administrator/-in möchte ich Kunden verwalten und Kennwörter zurücksetzen.	5	...
...	...	...	...

Sprint Meetings

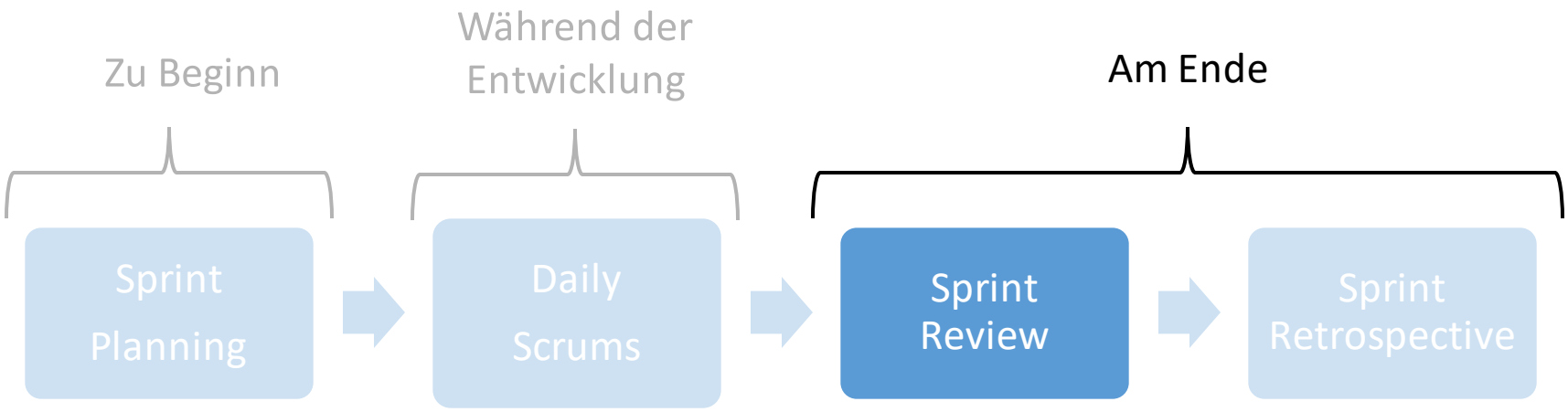


Tägliche Besprechung

- Teilnehmende: *Entwicklungsteam*
- Tägliche Besprechung des *Entwicklungsteams*
- Kurz: 15 Minuten
- Darstellung des aktuellen Stand
- Ergebnis: Verbesserte Kommunikation / Identifizierung von Problemen und Lösungsansätzen



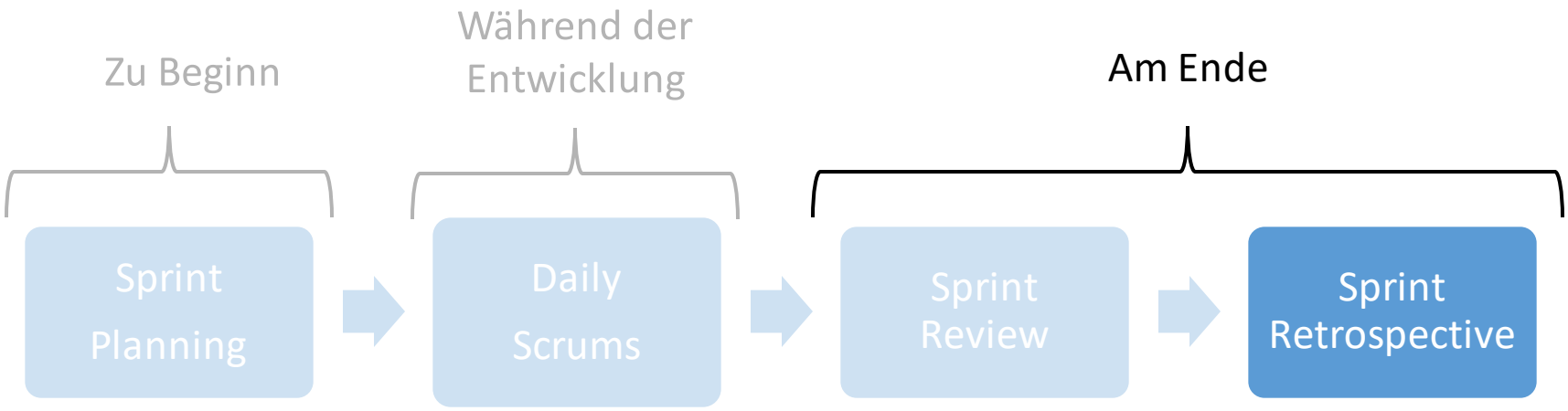
Sprint Meetings



Abschlussbesprechung

- Teilnehmende: *Scrum-Team* und Interessengruppe (bspw. Kunde)
- Durchgehen und Demonstration der Elemente des *Sprint Backlogs*
- Schnelles Feedback von bzw. Zusammenarbeit mit der Interessengruppe
- Diskussion über anstehende Aspekte anhand des *Product Backlogs*
- Ergebnis: Produktinkrement / überarbeiteter *Product Backlog*

Sprint Meetings



Retrospektive

- Teilnehmende: *Scrum-Team*
- Besprechung bezüglich Personen, Beziehungen, Prozess und Werkzeugen
- Ergebnis: Verbesserungen des Entwicklungsprozesses
- Realisierung durch *Scrum Master* im nächsten *Sprint*

Extreme Programming (XP)

Source

Extreme Programming (XP)

Eine iterative Entwicklungsmethode mit **extrem kurzen Iterationen**

- *Inkrementelle Planung*
  Am Ende einer Iteration steht ein fertiges Produkt (*Release*)
- *Kleine Releases*
  Funktionalitäten werden als ganzes nach und nach hinzugefügt
- *Kunde vor Ort*
  Der Kunde (oder ein Vertreter) steht für Rückfragen immer zur Verfügung
- *Best Practices*
  Extreme Programming stützt sich auf **bewährte Methoden**

Planung in XP

Planung eines Inkrements anhand von:

- User Stories (s. Requirements Engineering) und
- Story Points
- Folgt Pull-Prinzip

Aufwandsabschätzung anhand von *Story Points*

- Gesamte Team beteiligt
- Abschätzung ist relativ und Team abhängig
- Story Point: Relativer Aufwand einer User Story als abstrakte Zahl
- Aufwandsschätzung einer User Story in mehrere Runden (*Planning Game)*
- Ermöglicht Messbarkeit des Durchsatzes (auch *Velocity*) des Teams
- Durchsatz: Durchschnitt geleisteter Story Points in vorherigen Inkrementen

Best Practices in XP

Einfacher Entwurf

Nur das Notwendigste wird ausführlich spezifiziert

Test-First-Entwicklung

Vor der Entwicklung werden die Ziele in Tests festgelegt

Kontinuierliche Integration

Sobald eine Aufgabe abgeschlossen ist, wird sie in das Gesamtsystem integriert

Refactoring

Bestehender Code wird permanent gepflegt und verbessert

Paarprogrammierung

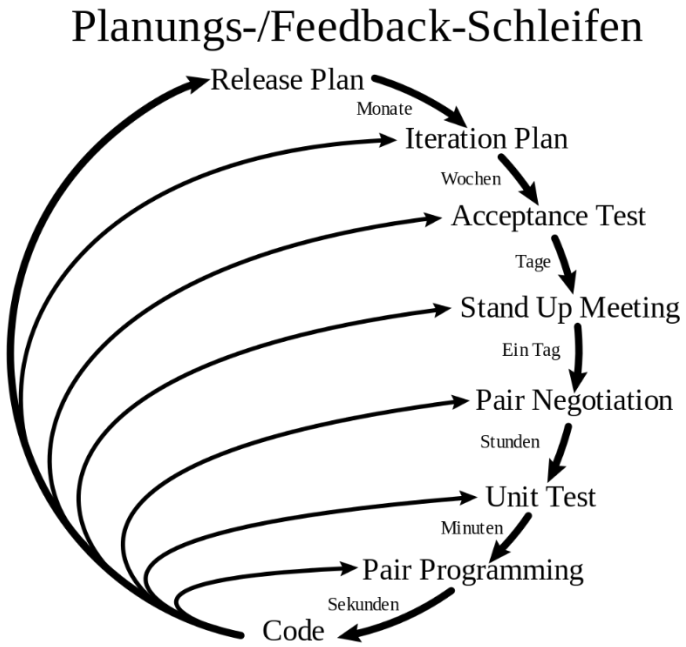
Entwickler arbeiten in Paaren und überprüfen sich gegenseitig

Kollektives Eigentum

Jeder Entwickler ist für den gesamten Code verantwortlich und darf ihn ändern

Direktes Feedback in XP

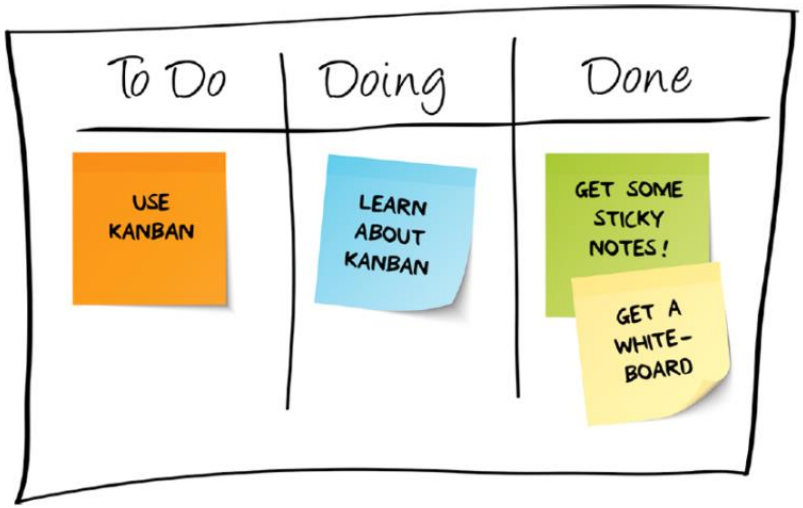
Es wird davon ausgegangen, dass der Kunde selbst die Anforderungen noch nicht genau kennt



Wikipedia

Kanban (Softwareentwicklung)

- Methode zur Organisation des Flusses paralleler Arbeiten
- Ursprünglich entwickelt von Toyota in der Fertigung
- Anpassung an Softwareentwicklung:
  Inkremente anhand von Anforderungen
- Pull-Prinzip

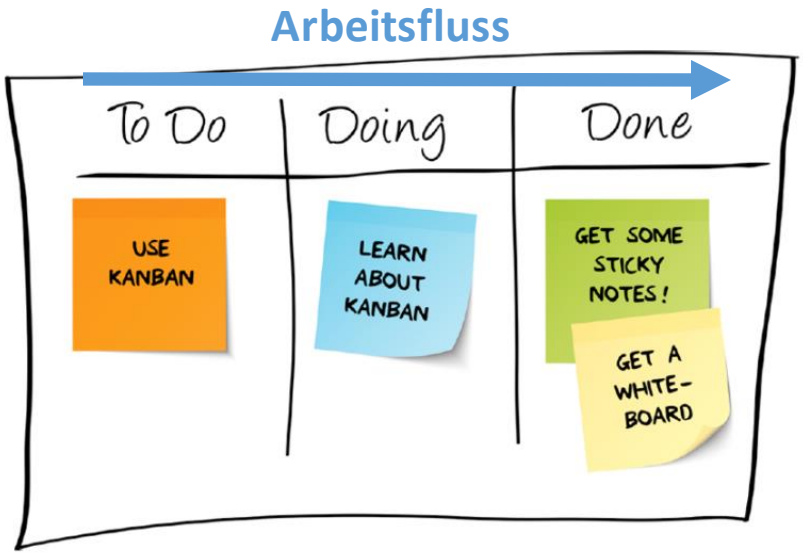


Source

Kanban - Visualisierung

Visualisierung mithilfe des Kanban-Boards

- Anforderungen durchlaufen Arbeitsschritte sequentiell
- Verdeutlichung der Prozessschritte anhand von Spalten
- Anforderungen (bspw. User Stories) als Karteikarten



Source

Kanban – Limitierung

Begrenzung der Anzahl paralleler Arbeiten für jeden Arbeitsschritt

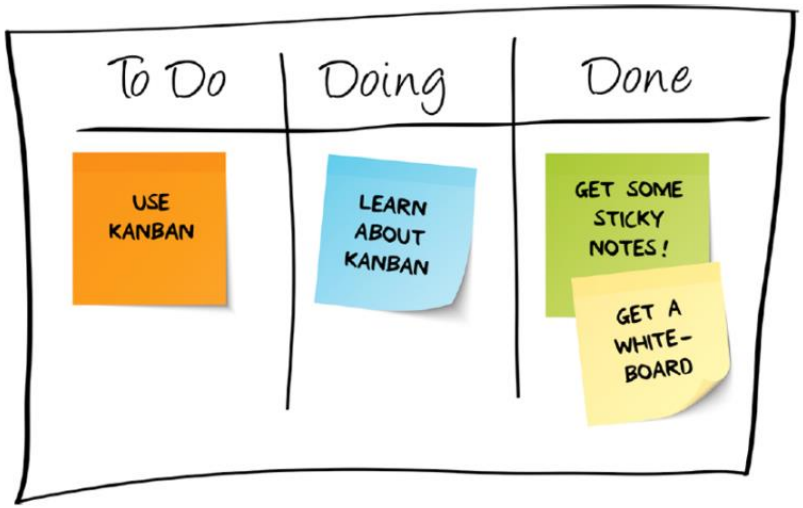
- Begrenzung ermöglicht das Pull-Prinzip
- Der Arbeitsschritt bestimmt die Anzahl der Elemente

Source

Kanban – Messung und Optimierung

Verwendung des Kanban-Boards ermöglicht Messung und Optimierung

- Messung von Warteschlangenlänge, Zykluszeit, Durchsatz möglich
- Identifizierung von Problemen wie bspw. Engpässen
- Optimierung einzelner Arbeitsschritte



Source

Fazit: Entwicklungsstile

Oft wird zwischen **plangesteuerten** und **agilen** Prozessen unterschieden.

- *Plangesteuerte Entwicklungsprozesse*
  - Die Abfolge der Aktivitäten und deren Ziele **im Voraus** fest eingeplant (*Push*)
  - Der tatsächliche Fortschritt lässt sich leicht anhand **des Planes** feststellen
  - Eignen sich am besten für große Softwareprojekte mit **festen Anforderungen**
  - **Kann** auch ein **inkrementeller** Prozess sein
- *Agile Entwicklungsprozesse*
  - Die Entwickler bestimmen die Menge der umsetzbaren Anforderungen (*Pull*)
  - Spezifikation und Entwicklung können sich **überschneiden**
  - Die Software wird **inkrementell** entwickelt
  - Frühe Prototypen ermöglichen direktes **Feedback** vom Kunden
  - Eignen sich gut für Umgebungen mit **wechselnden Anforderungen**

Plangesteuert oder agil?

Aussage	Empfehlung
Die Entwicklung benötigt eine sehr detaillierte Spezifikation.	Plangesteuert
Inkrementelle Auslieferung ist realistisch und schnelles Kundenfeedback gewünscht.	Agil
Das Entwicklerteam ist zu groß für informellen Austausch.	Plangesteuert
Die Anforderungen sind komplex und haben hohen Analysebedarf (z.B. Echtzeitsystem).	Plangesteuert
Das System ist sehr langlebig, umfangreiche Dokumentation erwünscht.	Plangesteuert
Kunde verlangt belastbare Spezifikation vorab (Pflichtenheft).	Plangesteuert
Das Team verfügt über viele erfahrene Programmierer die selbstständig arbeiten können.	Agil
Das System unterliegt externen Vorschriften (Genehmigung nach Standard).	Plangesteuert

Inhalt

Einführung in die Softwaretechnik

- Grundlagen der Softwaretechnik
- Planung
- Softwareentwicklungsprozesse
- Lernziele und Literatur

Lernziele

- ❑ Was versteht man unter Softwaretechnik?
- ❑ Was sind die grundlegenden Aktivitäten der Softwaretechnik?
- ❑ Wie sind die grundlegenden Aktivitäten im Wasserfallmodell angeordnet?
- ❑ Welche Vorteile hat das Wasserfallmodell/welche Nachteile?
- ❑ Wie unterscheiden sich Wasserfallmodell und V-Modell?
- ❑ Welche Tests sind im V-Modell eingeplant?
- ❑ Wie lassen sich die Entwicklungskosten neuer Software erheblich reduzieren?
- ❑ Welche Aktivitäten kommen hinzu, wenn bestehende Komponenten wiederverwendet werden sollen?
- ❑ Was versteht man unter inkrementeller Entwicklung?
- ❑ Was sind Vorteile, was Nachteile der inkrementellen Entwicklung?
- ❑ Welche Phase kommt bei dem Spiralmodell nach Boehm hinzu?
- ❑ Wann gilt ein Projekt nach dem Spiralmodell von Boehm als gescheitert?
- ❑ Was versteht man unter agilen Entwicklungsprozessen?
- ❑ Was ist der Unterschied zwischen plangesteuerten und agilen Entwicklungsprozessen?
- ❑ In welche 3 Phasen lässt sich der Scrum-Prozess gliedern?
- ❑ Wie läuft ein Sprint-Zyklus im Scrum-Prozess ab?
- ❑ Was ist der Unterschied zwischen dem Product Backlog und dem Sprint Backlog?
- ❑ Welche Rollen sind im Scrum Prozess vorgesehen?
- ❑ Welche Besonderheiten charakterisieren das Extreme Programming?
- ❑ Welche Best Practices finden beim Extreme Programming Anwendung?
- ❑ Wie wird im Extreme Programming mit Feedback umgegangen?
- ❑ Wann bietet sich planbasiertes Vorgehen an? Wann agiles?

Literatur

Diese Folien basieren zum Teil auf dem folgenden Buch:

IAN SOMMERVILLE, *Software-Engineering,* Pearson, 2012, 9. Aufl.

Link zur Zentralbibliothek

Mehr (englischsprachiges) Material gib es auf:

http://iansommerville.com/software-engineering-book/