

Software Engineering I

1. Einführung

Prof. Dr. Eckhard Kruse

DHBW Mannheim

Organisation

- Software Engineering I: WS+SS 2022/2023
 - Vorlesung: 96 h (WS 6 h, SS 4 h pro Woche, inkl. je 1h begl. Selbststudium)
 - **Selbststudium: 174 h!**
 - Termine: s. Kalender, inkl. etwas Puffer
- Vorlesung + Übungen + Teamarbeit: Software-Engineering-Projekt
- Fragen: Am besten direkt in der VL
 - eckhard.kruse@dhbw-mannheim.de, Raum 344 B, Tel. (0621) 4105 1262
- Verteilung der Folien
 - nach jeder Vorlesung per E-Mail-Verteiler
- Leistungsnachweis
 - Programmentwurf
 - 1 Note (Ende SS): Gesamtergebnisse aus Projektarbeiten (Code, Doku, Präsentationen usw.), aus WS+SS (50/50)
- Literatur („further reading“), z.B.:
 - Ludewig, Licher: *Software Engineering: Grundlagen, Menschen, Prozesse, Techniken*
 - Ian Sommerville: *Software-Engineering (englisch)*

Software macht's möglich ...



Software macht's möglich ...

A photograph of a grassy field at night or dusk. A bright, glowing trail of light extends from the bottom left towards the top right, representing the path of an Ariane 5 rocket during its failed launch. The field is dotted with small, scattered lights.

4.6.1996: Fehlgeschlagener Start der ersten Ariane 5
Finanzieller Schaden: ca. 500 Mio. US\$

4. Juni 1996: Erster Start der Ariane 5



- Während des Fluges läuft ein unnötiges Kalibrierprogramm, welches von der alten Software der Ariane 4 übernommen wurde.
- Die gemessenen Werte der Ariane 5 überschreiten die in der Ariane-4-Software vorgesehenen Wertebereiche → Variablenüberlauf
- Die Exception wird erkannt → Fehlerbehandlung: Anhalten des Steuerungscomputers, Umschalten auf zweites redundantes System.
- Im zweiten System läuft die gleiche Software → identische Fehlerbehandlung ...

Die Softwarekrise ...

- Software wird immer komplexer.
- Der Aufwand für Softwareentwicklung und Testen steigt rasant.
- Kleine Fehler können große Folgen haben.

Folgen: Software wird nicht termingerecht fertig, Kosten laufen aus dem Ruder und die Software-Qualität ist schlecht.



Wir haben eine „Softwarekrise“!

Der Begriff „Softwarekrise“ wurde Ende der 1960er geprägt!

Die Softwarekrise geht weiter!

- Software wird immer komplexer.
- Der Aufwand für Softwareentwicklung und Testen steigt rasant.
- Kleine Fehler können große Folgen haben.

Folgen: Software wird nicht termingerecht fertig, Kosten laufen aus dem Ruder und die Software-Qualität ist schlecht.

- Software wird „überall“ eingesetzt → Fehler in technischen Systemen sind immer häufiger auf Software-Fehler zurückzuführen
 - Autos, Unterhaltungselektronik, Haushaltsgeräte, Telefone...
- Software wird immer wichtiger in sicherheitskritischen Anwendungen
 - Verkehr (fly/drive-by-wire, Automatisierung), Industrie (z.B. Chemie, Pharma, ...), medizinische Anwendungen usw.
- Fehlerhafte Software ist als Normalität akzeptiert:
 - Updates/Patches als Standard, Kunde=Tester
 - Fehlerfreies Produkt von Anfang an: ist das überhaupt noch ein Ziel?

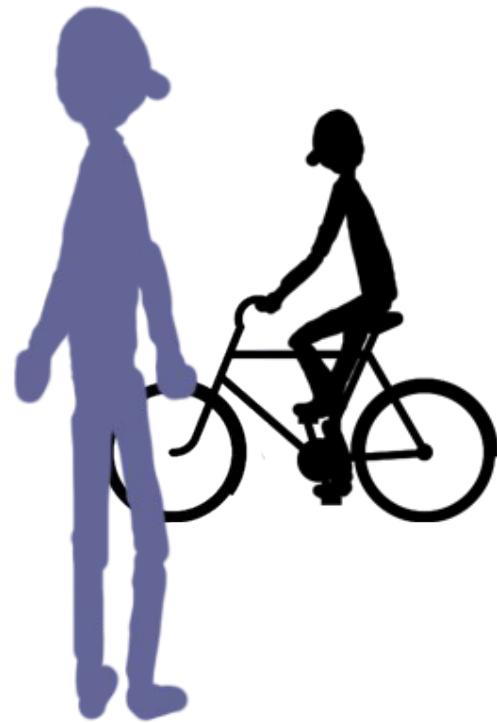
Software-Engineering als Rettung

Software-Engineering:

„Zielorientierte Bereitstellung und systematische Verwendung von Prinzipien, Methoden und Werkzeugen für die arbeitsteilige, ingenieurmäßige Entwicklung und Anwendung von umfangreichen Softwaresystemen unter Berücksichtigung von Kosten, Zeit und Qualität“

[s. H. Balzert, Lehrbuch der Software-Technik, Spektrum Akademischer Verlag, Heidelberg 2001]

Wie lernt man Software-Engineering?

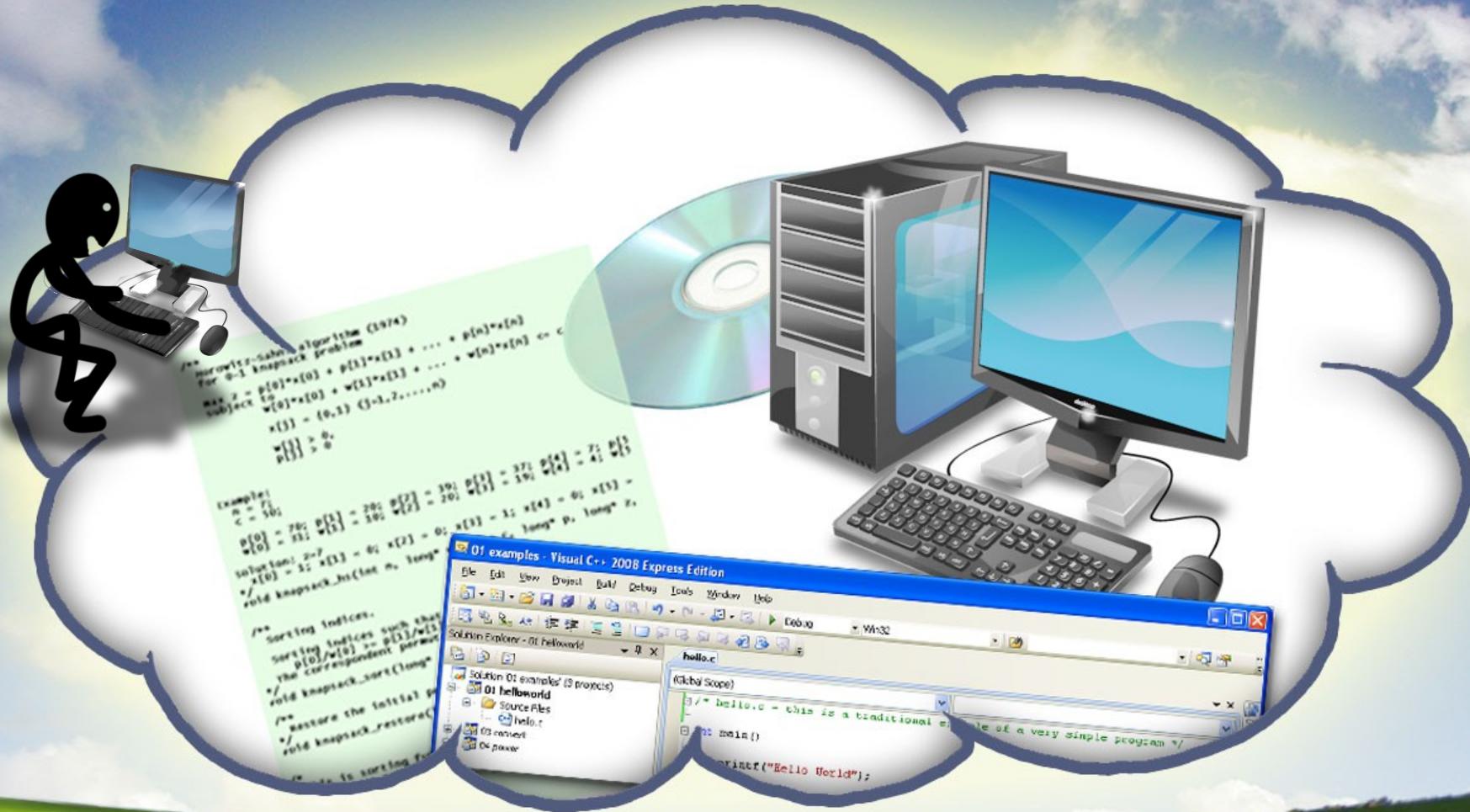


Eigentlich selbstverständlich...

Bewährte Regeln für effizientes gemeinsames Arbeiten,
Besprechungen usw. → gilt auch für diese Vorlesung:

- Pünktlichkeit (Vorlesungsbeginn, Pausenende)
- Anwesenheit: von Körper + Geist
- Anzahl gleichzeitig redender Personen ≤ 1 (Ausnahme Teamarbeit)
- Konzentration auf das Geschehen
 - Laptops nur ggf. zum Mitschreiben und für die Übungen
- Handys ausschalten, keine Telefonate
- ggf. Feedback zum Arbeitsprozess
 - Stoff zu schnell / zu langsam? Pausenbedarf?

Software-Engineering = Programmieren?



Was möchte eigentlich der Kunde?



Was möchte der Kunde?



Was möchte der Kunde?

Ein Programm?

Nein, eine zuverlässig funktionierende Lösung für seine Aufgaben!

z.B.

- *eine Möglichkeit für seine Kunden Zimmer bequem übers Internet zu buchen*
- *eine Steuerung seiner Produktion für maximalen Durchsatz und Qualität*
- *eine sichere, zentrale Verwaltung aller in der Firma anfallenden Daten.*
- ...
- *einen finanziellen Nutzen, der die Kosten der Software deutlich übersteigt.*

Klarer Business case

Return on investment

Qualität
(Bedienbarkeit,
Performance,
Zuverlässigkeit...)

Total cost of ownership
→ Kosten über gesamte
Systemlaufzeit

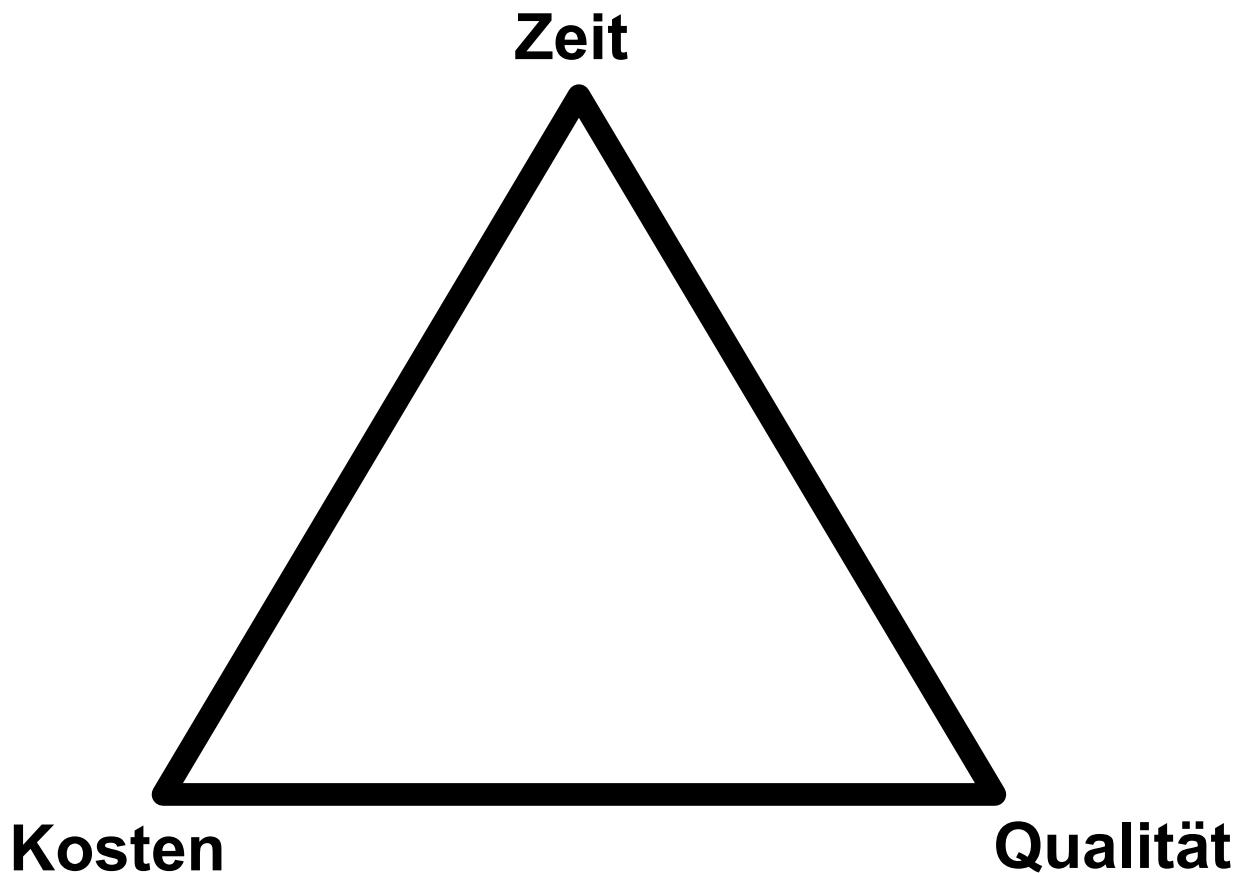
Software-Engineering

Software-Engineering:

„**Zielorientierte** Bereitstellung und systematische Verwendung von Prinzipien, Methoden und Werkzeugen für die arbeitsteilige, ingenieurmäßige Entwicklung und Anwendung von umfangreichen Softwaresystemen **unter Berücksichtigung von Kosten, Zeit und Qualität**“

[s. H. Balzert, Lehrbuch der Software-Technik, Spektrum Akademischer Verlag, Heidelberg 2001]

Das magische Dreieck



Projektaufgabe - Was möchte der Kunde?



Software-Engineering-Projekt

P.1 Bildung von Projektteams

- a) Bilden Sie Projektteams für die Themen.
- b) Die Teams sind die Lieferanten für das gewählte Thema.
- c) Die Teams agieren außerdem als Kunden für ein anderes Team.
- d) Vorschlag: Benennen Sie pro Team einen Projektleiter für die Lieferantenrolle und einen für die Kundenrolle. (Kann ggf. auch noch während der Projektlaufzeit geändert werden.)
- e) Teamzusammensetzung per E-Mail an Dozenten.

Hinweise zur Teambildung:

- Sind die Teams ausgewogen? (Neben den Einzelteams sollte auch der Kurs zusammen als Team funktionieren.)
- Akzeptieren Sie Kompromisse. Im Berufsleben kann man sich seine Teams meist nicht aussuchen.

Übung

Das magische Dreieck

- a) Überlegen Sie, aus welchen anderen Lebensbereichen Sie das „magische Dreieck“ aus Kosten, Zeit, Qualität kennen.
- b) Überlegen Sie, wie sich folgende Entscheidungen in einem SW-Projekt auf die drei Faktoren auswirken könnten:
 1. Verdopplung des Umfangs der Systemtests
 2. Outsourcing in ein Niedriglohnland
 3. Erweiterung des Funktionsumfangs in einer frühen Projektphase
 4. Erweiterung des Funktionsumfangs in einer späten Projektphase
 5. Einstellung zusätzlicher, aber unerfahrener Mitarbeiter
 6. Zusätzliche, regelmäßige Treffen für Projektreviews mit dem Kunden
- c) Nennen Sie weitere Beispiele für potenzielle Entscheidungen und Abwägungen bzgl. der drei Faktoren – insbesondere im Hinblick auf die eigenen Projektideen.

http://en.wikipedia.org/wiki/Project_triangle

"... There are also numerous spinoffs to this triangle, the most common including:

- *College: Work, Sleep, Play – Pick two.*
- *Men: Handsome, High-Earner, Faithful – Pick two.*
- *Women: Single, Sane, Sexy, Smart – Pick any three. (also called The four S's of dating)*
- *Operating System: Fast, Efficient, Stable - Choose two.*
- *Bicycle Parts: Strong, Light, Cheap - Pick any two.*
- *Opensource Software Development: Speed/Time, Inclusiveness/Openness, Quality*
- *Schedule, Scope, Resources – Pick two.*

..."

Übung

Was ist ein gutes Team?

Nennen Sie Kriterien für ein gutes Projektteam.

- a) Warum muss eine Gruppe gleichgesinnter Freunde nicht notwendigerweise ein gutes Projektteam sein?
- b) Welche Kompetenzen sind für SW-Projekte wichtig und sollten im Team vorhanden sein?
- c) Schwieriges Team = viel Lernpotenzial. Erläutern Sie diese Aussage.

Was ist ein gutes Team?

Übung?

- Eine Gruppe gleichgesinnter Freunde muss nicht unbedingt ein optimales Projektteam sein.
 - Projektteams leben von Vielfalt und verschiedenem Wissens-/Interessenmix.
 - Außenstehende können neue Sichtweisen (gegen die Betriebssblindheit) einbringen.
 - zu viel Harmonie kann "einlullen" und Risiken werden möglicherweise nicht gesehen
- Wertvolle Kompetenzen: Analysieren, verhandeln, erklären, hinterfragen, dokumentieren, schlichten, energetisieren, programmieren, tüfteln, ...
- Im Berufsleben gibt es immer wieder schwierige Teams. Hier lässt sich ggf. das Arbeiten in solchen Teams in einem geschützten Rahmen (ohne schlimme Konsequenzen) erlernen.

Software:

- Alles 'Nichtphysische' im Computersystem: Computerprogramme, Prozeduren, Regeln, Dokumentation, Daten

Softwaresystem:

- Ein System ist ein Ausschnitt aus der realen Welt bestehend aus Teilen, die in Beziehung stehen
SW-System: Ein System, dessen Komponenten aus Software bestehen

Software-Produkt:

- Ein Produkt ist ein in sich abgeschlossenes, i.a. für einen Auftraggeber bestimmtes Ergebnis eines erfolgreich durchgeföhrten Herstellungsprojektes.
- SW-Produkt: Produkt, das aus Software besteht.

Zeitplan (Vorschlag)

Woche

- 1 Teams, Themen, Definition Kunden
- 1-2 Lastenheft
- 2-4 Vorstudie(n) + Mockups, Pflichtenheft → Auftrag
- 4-5 Entwurf/Architektur, (grobe) Projektplanung+Aufwandsschätzung
- 5-10 Moduldesign + Implementierung
- 8-10 Testspezifikation + Testen
- 11 Übergabe, Abgabe Projekttagebücher
(Die Arbeitsergebnisse gehen in die bewertete PL nach dem SS ein!)

Übung

Warum ist Software besonders?

Überlegen Sie, worin sich das Entwickeln von Software von traditionellen Engineeringprojekten, die der Entwicklung von „hardware-basierten“ Produkten dienen, unterscheiden könnte.

- a) Was sind die Besonderheiten des Endprodukts „Software“?
- b) Welche Arbeitsschritte während der SW-Entwicklung sind zusätzlich/spezifisch für Software, welche Schritte könnten entfallen?
- c) Was ist einfacher bei der Entwicklung von Software?
- d) Was könnte Software-Engineering schwieriger machen?

Softwarebesonderheiten

- Fertigung/Produktionskosten (= DVD brennen / auf Server zum Download stellen) spielen praktisch keine Rolle (und sind kein Kostenfaktor) → prinzipiell leicht änderbar.
- Qualität schwer messbar
- Zuverlässigkeit (von Komponenten) lässt sich nicht wie bei Hardware einfach in Ausfallraten (z.B. MTBF = mean time between failure) angeben.
- Praktisch nicht 100%ig testbar
- „Alterung“: Zugrunde liegende Betriebssysteme/Technologien entwickeln sich schnell → große Auswirkung auf Softwareprodukt. Kontinuierlicher Aufwand für Aktualisierung
- Patches / Updates über Internetdownload möglich

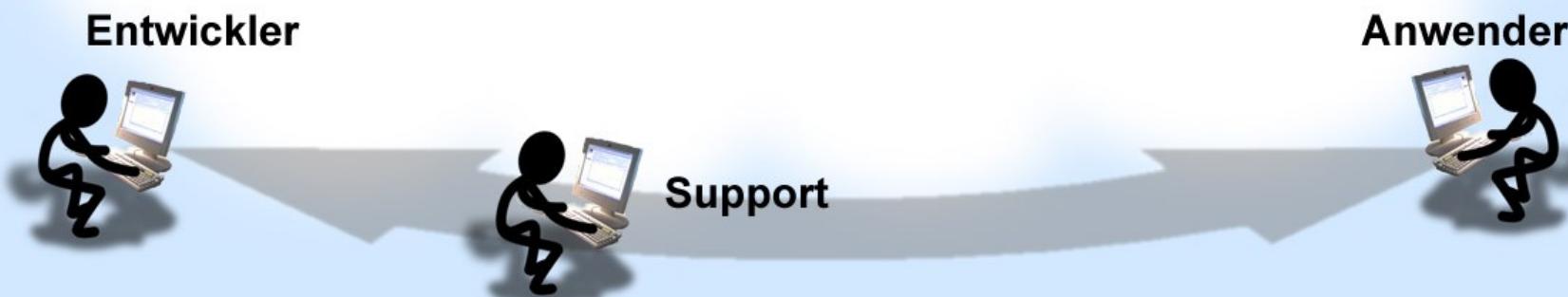
s. Studienplan:

- Theoretische Grundlagen des Software-Erstellungsprozesses kennen.
- Komplexe Problemstellungen analysieren und rechnergestützte Lösungen umsetzen und dokumentieren können.
- Projektphasenmodell und Methoden der Phasen kennen und anwenden können.
- Ergebnisse der jeweiligen Phasen in ihren Inhalten und Zielrichtungen erfassen und dokumentieren.
- Konkrete Ergebnisse innerhalb der einzelnen Projektphasen mit geeigneten Tools erarbeiten.
- Gruppendynamische Prozesse bei der Bearbeitung größerer Aufgaben innerhalb von Projektgruppen erfahren.

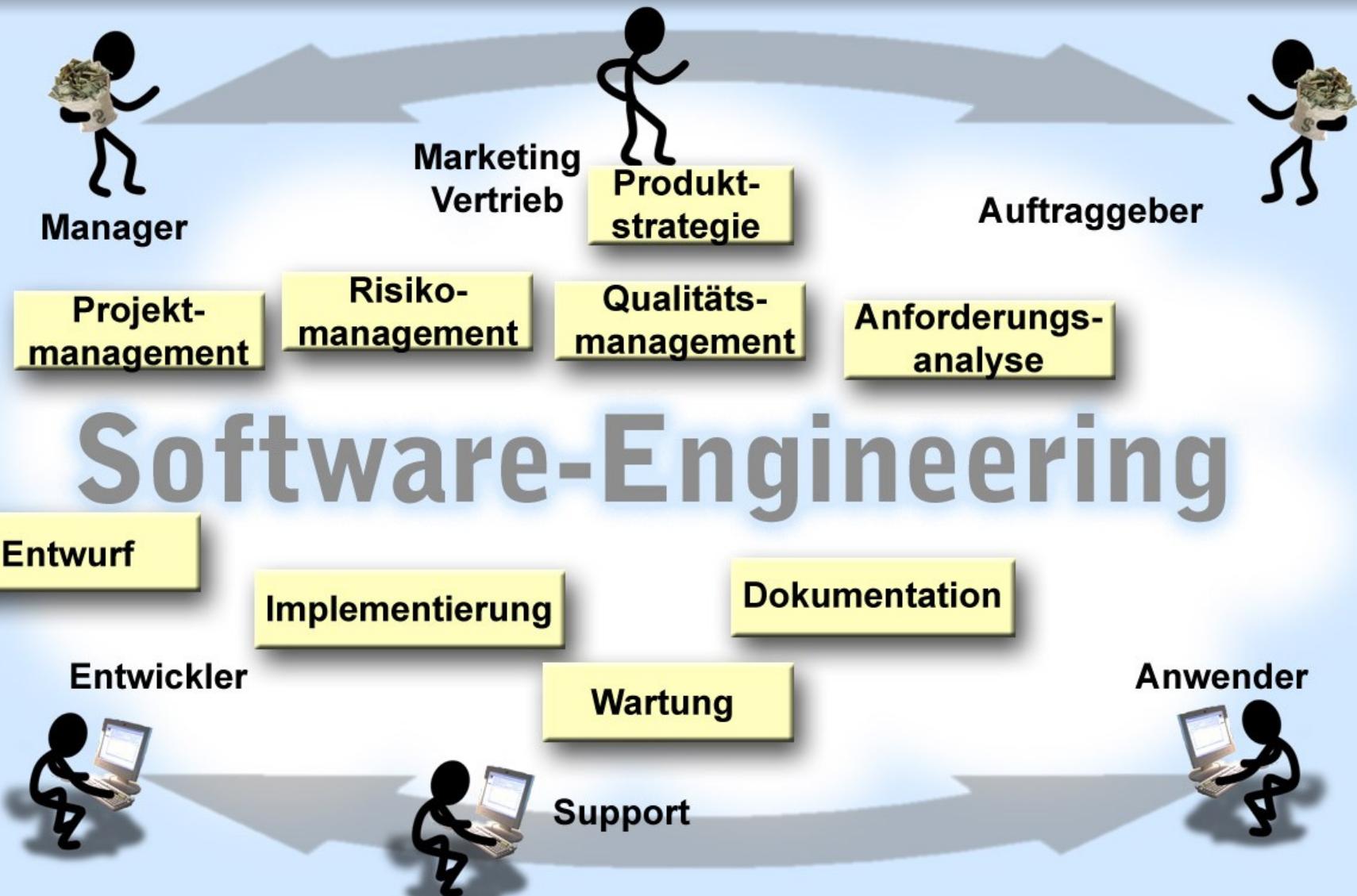
SW-Engineering Umfeld



Software-Engineering



SW-Engineering: Viele Aufgaben



Aufgaben im SW-Engineering

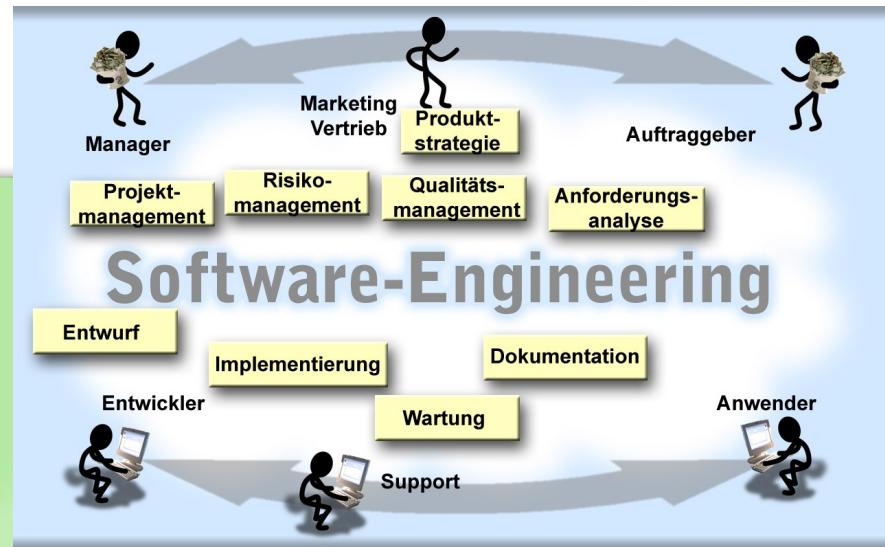
Übung?

Übung

Aufgaben im SW-Engineering

Diskutieren Sie die Rollen und Aufgaben im Umfeld des Software-Engineerings.

- Was sind Merkmale der verschiedenen Rollen: Welche Qualifikation müssen entsprechende Mitarbeiter haben? Was für Aufgaben müssen sie typischerweise wahrnehmen?
- Können Sie die genannten Rollen weiter differenzieren? Welche Rollen/Unterrollen könnte es noch geben?
- Können Sie die Aufgabenbereiche näher charakterisieren? Gibt es weitere Aufgabenbereiche / Unterbereiche?

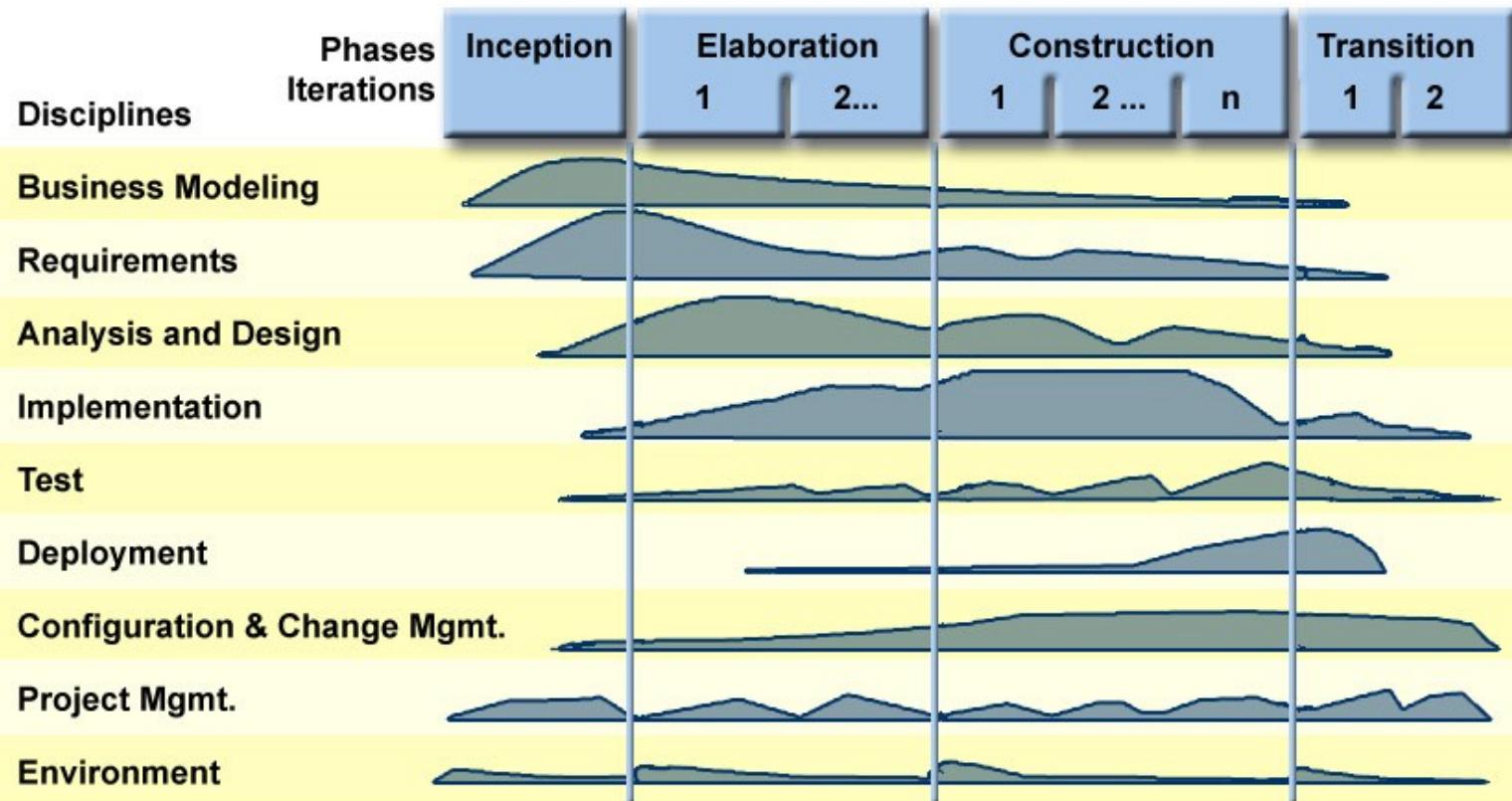


- Planen
- Verwalten
- Kommunizieren
- Analysieren
- Spezifizieren
- Dokumentieren
- Entwerfen
- Programmieren/Implementieren
- Kontrollieren (Qualität)
- Testen
- Konfigurieren
- Installieren
- Instandhalten

- **Analyst:** Probleme erfassen, kommunizieren, reden+aufschreiben, Kenntnis der Anwendungsdomäne
- **Modellierung/Designer:** Abstraktionsfähigkeit, UML, Design Patterns, Erfahrung, Informationsmodelle
- **UI Designer:** HTML, Usability, Gestalterische Fähigkeiten
- **Datenbankexperte:** Technische Kenntnisse, Modellierung
- **Architekt:** Programmiererfahrung, Entwurfsmuster, kommunizieren
- **Entwickler (für Programmiersprache X)** C/Java/...-Kenntnisse, Erfahrung!, guter Programmierstil
- **Techn. Schreiber:** Klare schriftl. Ausdrucksweise, Einfühlungsvermögen in den Leser, Erfahrung mit System-/Benutzer-Dokumentation
- **Tester** Kritische Einstellung, Sonderfälle erkennen, Erfahrung in Testplanung, Testtools
- **Qualitätsmanager:** Organisieren, kommunizieren, kritisch hinterfragen, planen
- **Projektleiter:** Organisieren, kommunizieren, präsentieren, entscheiden, Kompromisse aushandeln, schlichten, motivieren, Blick fürs Ganze, Zielorientierung, Erfahrung mit Vorgehensmodellen, Planungstools

(Ohne Anspruch auf Vollständigkeit!)

Beispiel: Rational Unified Process



Software Engineering I

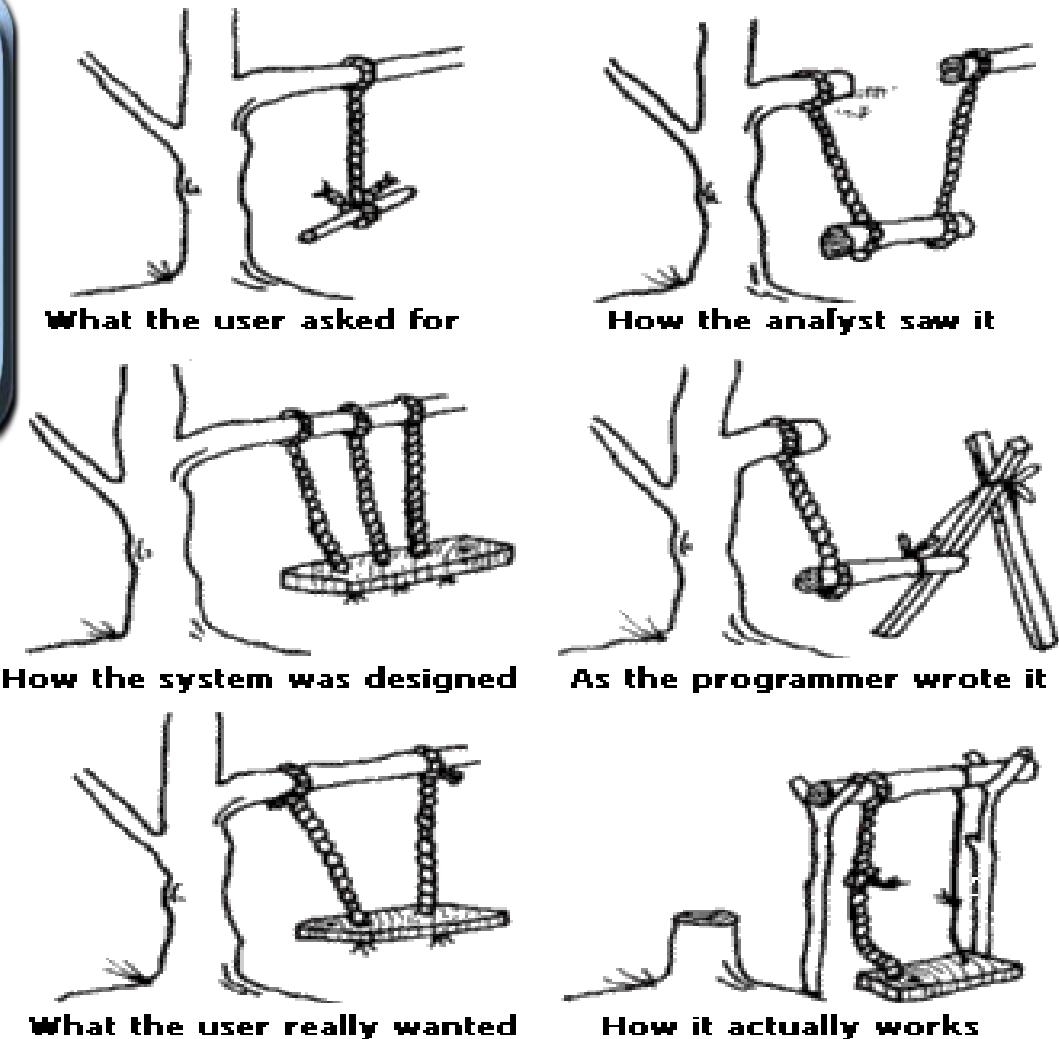
2. Anforderungsanalyse und Spezifikation

Prof. Dr. Eckhard Kruse

DHBW Mannheim

Der Wert eines Softwaresystems...

The primary measure of success of a software system is the degree to which it meets the purpose for which it was intended.
[Nuseibeh&Easterbrook '00]



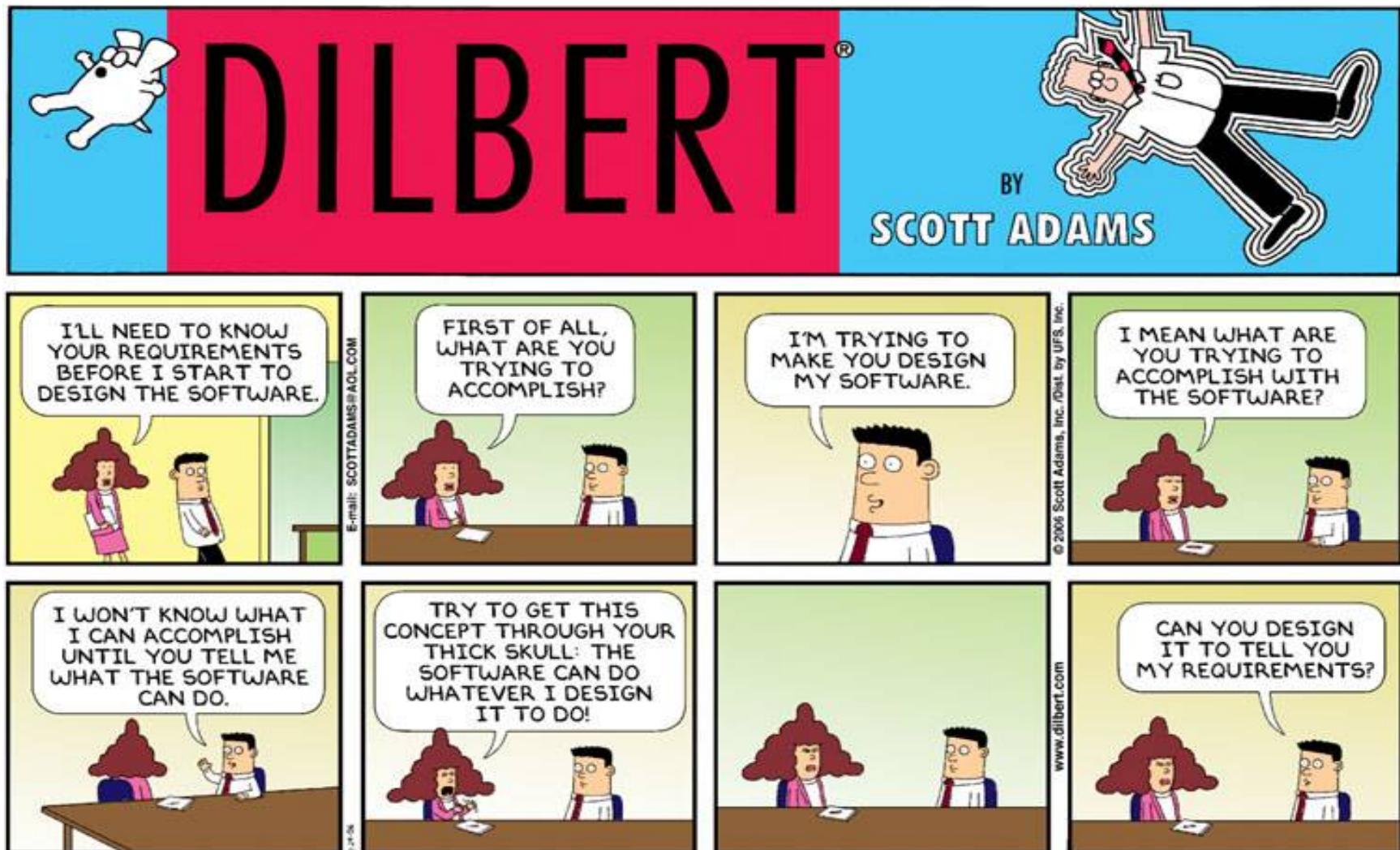


Software-Engineering-Projekt

P.2 Kundensicht: Definition des eigenen Profils

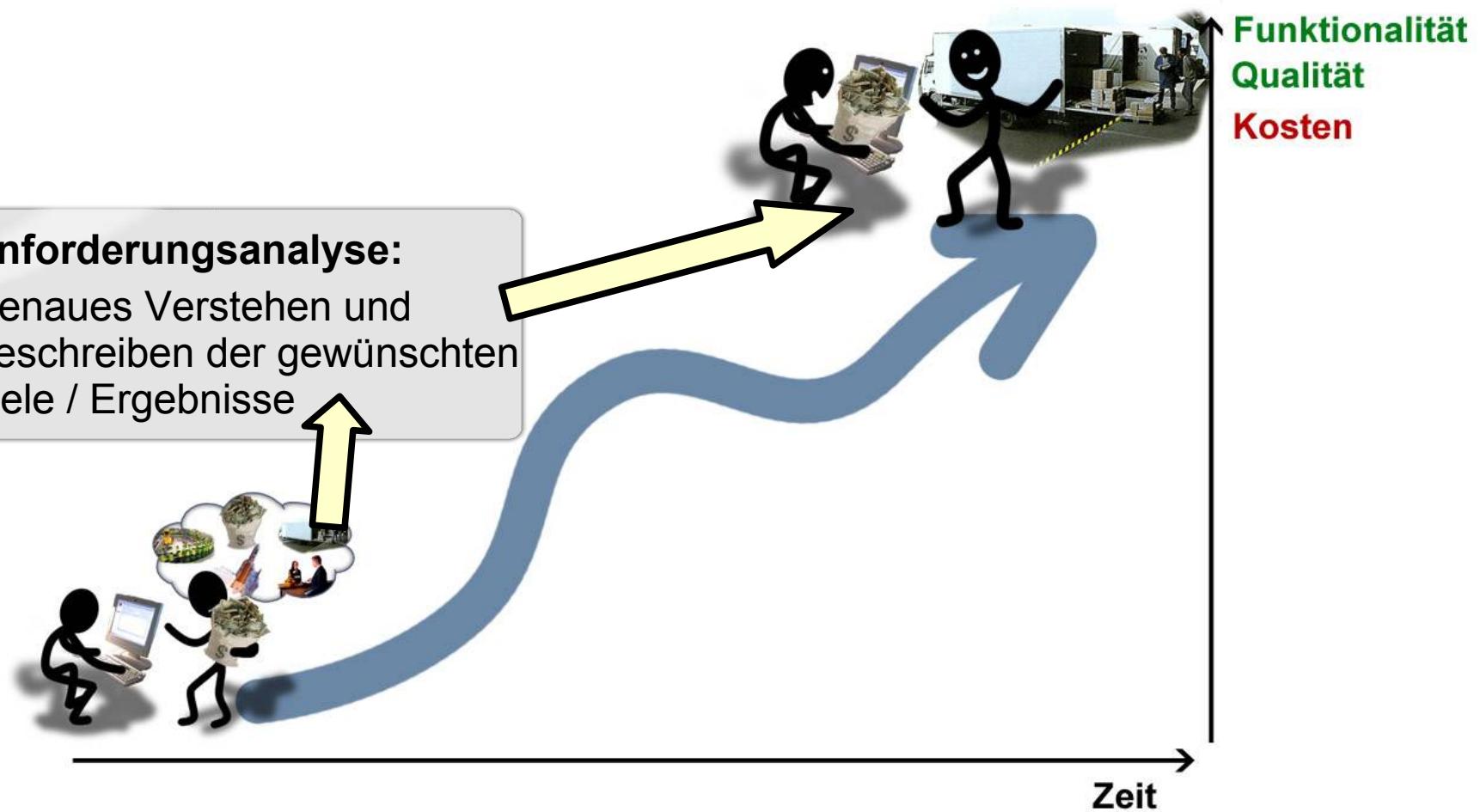
- a) Als Kunde sind Sie an einer in der Themenstellung vorgegebenen Softwarelösung interessiert. Definieren Sie Ihr Kundenprofil und Hintergründe für den Softwareauftag.
 - a) In welcher Branche agieren Sie?
 - b) Warum/wofür benötigen Sie die Software?
 - c) Was sind typische Anwender/Anwendungsszenarien?
 - d) Wann wäre der Softwareeinsatz ein Erfolg?
- b) Erstellen Sie zwei, drei Folien, die Ihr Profil prägnant wiedergeben und mit denen Sie sich der "Öffentlichkeit" bzw. dem Lieferanten präsentieren können.
- c) Ein Mitglied des Teams soll mit diesen Folien eine 5-minütige Kurzvorstellung von Ihrer Firma geben.

Anforderungen – woher nehmen?

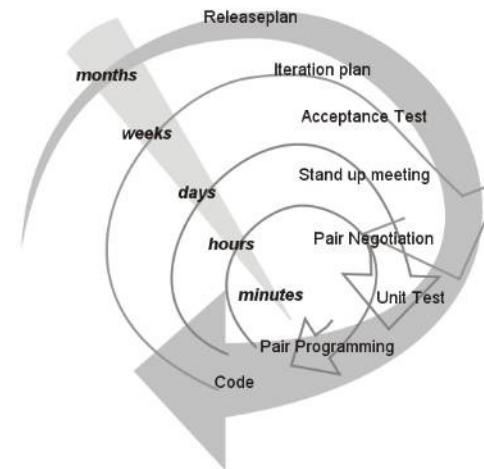
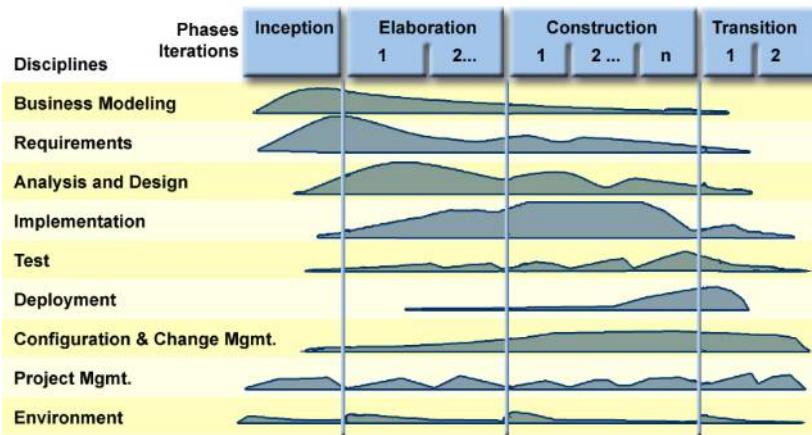
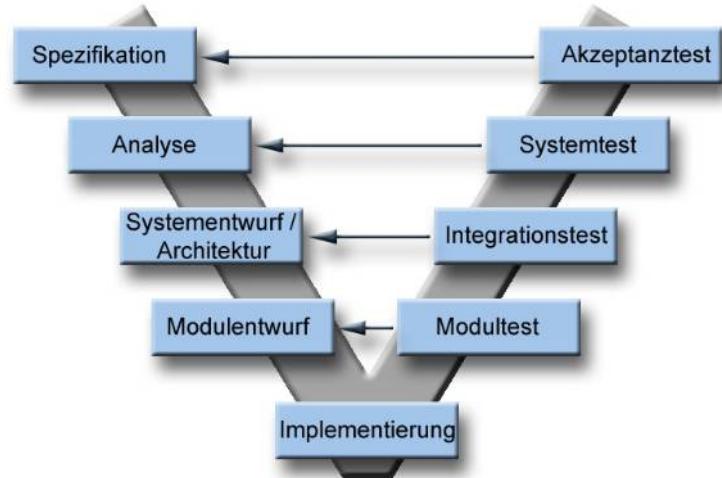


© Scott Adams, Inc./Dist. by UFS, Inc.

Am Anfang die Anforderungen



1. Phase in allen Vorgehensmodellen

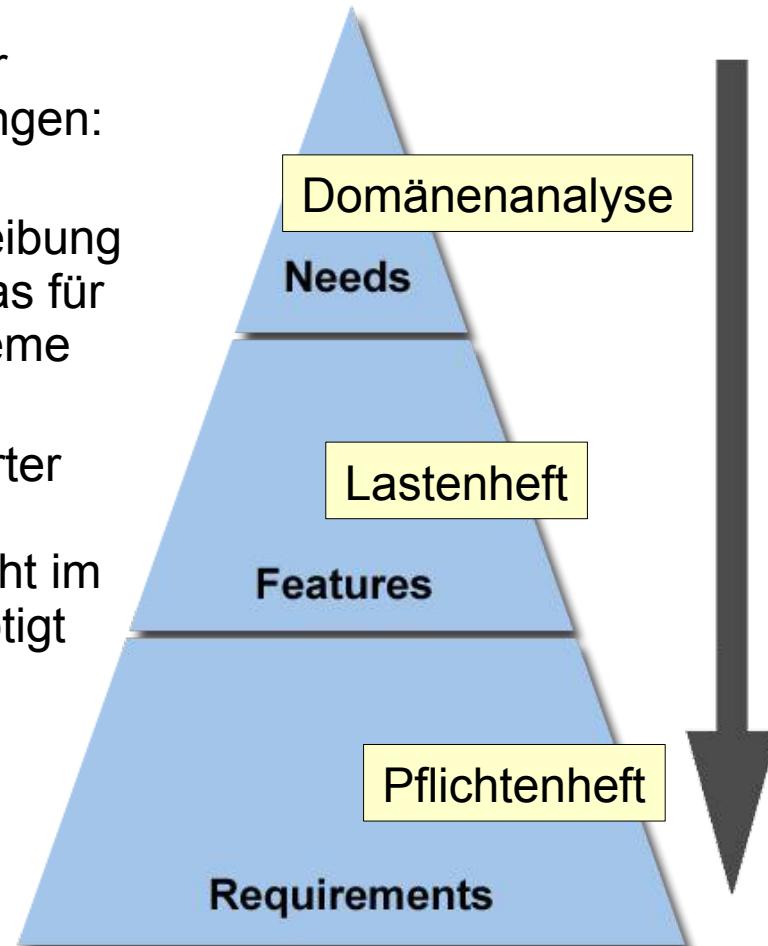


Anforderungen: Detaillierung

Needs: Bestandsaufnahme existierender Geschäftsprozesse, Problembeschreibungen: Warum soll Software entwickelt werden.

Features: Lastenheft mit grober Beschreibung von Hauptfunktionen neuer Software: Was für Funktionen sollen die gefundenen Probleme lösen.

Requirements: Pflichtenheft mit detaillierter Beschreibung der Aufgaben der neu zu entwickelnden Software: Immer noch steht im Vordergrund „was für Funktionalität benötigt wird“ und nicht „wie die gewünschte Funktionalität zu realisieren ist“.



In der **Domänenanalyse (domain analysis)** wird das Anwendungsgebiet hinsichtlich grundlegender Fragen untersucht: Was sind wesentliche Vorgänge, Datenmodelle, Konzepte, Anwender/Stakeholder usw., die bei der Entwicklung von Lösungen in dem Gebiet eine Rolle spielen?

- Insbesondere wichtig bei Anstrengung von Wiederverwendung, mehreren Projekten in der Domäne oder Aufbau einer Produktlinienarchitektur.
- Oft in Zusammenhang damit: Stakeholderanalyse
- Ziel: Wissen (z.B. von Experten in dem Bereich) sichtbar machen, dokumentieren und strukturieren.
- Ggf. auch Betrachtung bereits existierender Systeme (und deren Architekturen)
- Gemeinsamkeiten und Unterschiede verschiedener Systeme in dem Bereich aufzeigen
- Ergebnis: Domänenmodell, strukturierte (ggf. semiformale) Beschreibung von dem Anwendungsgebiet, Informationsmodellen, Abläufen usw.

Lastenheft und Pflichtenheft

DIN 69905, "Projektabwicklung, Begriffe"

- Das **Lastenheft** beschreibt ergebnisorientiert die Gesamtheit der Forderungen an die Lieferungen und Leistungen eines Auftragnehmers.
- Das **Pflichtenheft** detailliert die Auftraggebervorgaben und beschreibt das Realisierungsvorhaben unter Berücksichtigung konkreter Lösungsansätze.



Das **Lastenheft** führt grob alle fachlichen Anforderungen auf, die das fertige (Software-)Produkt aus Sicht des Auftraggebers erfüllen soll.
Es ist das grundlegende Dokument im Entwicklungsprozess.

Beispielstruktur eines Lastenhefts:

1. **Ziel:** Welche Ziele sollen mit dem Produkt erreicht werden?
2. **Einsatzbereich:** Welche Anwendungsbereiche und Zielgruppen werden addressiert?
3. **Funktionen:** Was sind die Hauptfunktionen aus Sicht des Auftraggebers? (Verwende systematische Nummerierung/Kennung für spätere Referenzierbarkeit)
4. **Daten:** Welche Arten von Daten sind für die Software von zentraler Bedeutung. (z.B. Benutzerdaten, Produktdaten,)
5. **Leistungsmerkmale:** Welche Anforderungen gibt es an Daten und Funktionen bezüglich Datenvolumen, Antwortzeiten, Durchsatz, Genauigkeit, usw.?
6. **Qualitätsanforderungen:** Welche Anforderungen gibt es z.B. in Hinblick auf Zuverlässigkeit, einfache Bedienung, Effizienz, Wartbarkeit?
7. **Ergänzungen:** Weitere wichtige Punkte, z.B. technische Randbedingungen?



Software-Engineering-Projekt

P.3 Kunde: Erstellen eines Lastenhefts

Entsprechend Ihres Kundenthemas möchten Sie eine Softwareentwicklung in Auftrag geben. Als Grundlage für eine Ausschreibung bzw. für die Diskussion mit potenziellen Auftragnehmern ist ein Lastenheft zu erstellen.

- a) Überlegen Sie, welche groben Vorstellungen Sie an das Ergebnis haben.
Machen Sie ein Brainstorming, recherchieren Sie, welche Lösungen es (z.B. bei anderen Firmen in der Branche) gibt usw.
- b) Was muss unbedingt geliefert werden? Welche Punkte sind optional und können evtl. verhandelt werden?
- c) Optional: Erstellen Sie ein kurze Präsentation (2-3 Folien), mit der Sie einen groben Überblick über Ihr Vorhaben geben können (z.B. Ihrer Firmenleitung).
- d) Welche Arbeitsschritte sind für die Erstellung des Lastenheftes erforderlich?
Wie organisieren Sie sich im Team?
- e) Senden Sie Ihr Lastenheft rechtzeitig dem potenziellen Lieferanten, damit er sich auf ein erstes Treffen mit Ihnen vorbereiten kann.



Software-Engineering-Projekt

P.4 Lieferant: Analyse des Lastenhefts

Von einem potenziellen Kunden haben Sie im Rahmen einer Ausschreibung ein Lastenheft für eine zu entwickelnde Softwarelösung erhalten. Da das Thema genau in Ihren Kompetenzbereich fällt und sechs Mitarbeiter bis Anfang Februar noch nicht ausgelastet sind, möchten Sie diesen Auftrag unbedingt erhalten.

- a) Untersuchen Sie das Lastenheft. Lässt sich die geforderte Lösung in der zur Verfügung stehenden Zeit realisieren? Wo sollten Abstriche gemacht werden?
- b) Haben Sie eigene Ideen zu dem Thema, die Sie dem Kunden schmackhaft machen wollen?
- c) Bereiten Sie sich auf das erste Treffen mit dem Kunden vor. Sie wollen zeigen, dass Sie die richtige Entwicklungsfirma für das Projekt sind, gleichzeitig aber auch nur Ergebnisse versprechen, die Sie hinterher tatsächlich liefern können.

Erstes Treffen Kunde-Lieferant



Software-Engineering-Projekt

P.5 Kunde-Lieferant: Erstes Treffen

Zwischen dem Kunden und dem Lieferanten kommt es zu einem ersten persönlichen Treffen. Da der Kunde eine gewisse Offenheit bezüglich Details der zu entwickelnden Lösung besitzt, hat er vom Lieferanten noch kein Pflichtenheft oder konkretes Angebot eingefordert. Statt dessen möchte er mit dem Lieferanten das Lastenheft besprechen bzw. dessen Ideen vorgestellt bekommen, um dann ggf. weitere Schritte zu planen.

- Was sind Ihre gegenseitigen Erwartungen? Besteht Aussicht auf einen Vertragsabschluss? Was sind die nächsten Schritte bis zu einem konkreten Angebot / Auftrag?
- Idealerweise können Sie sich darauf einigen, dass der Lieferant auf Basis des Lastenheftes und ggf. besprochener Änderungen ein Pflichtenheft erstellt, welches Grundlage des Vertrages wird.

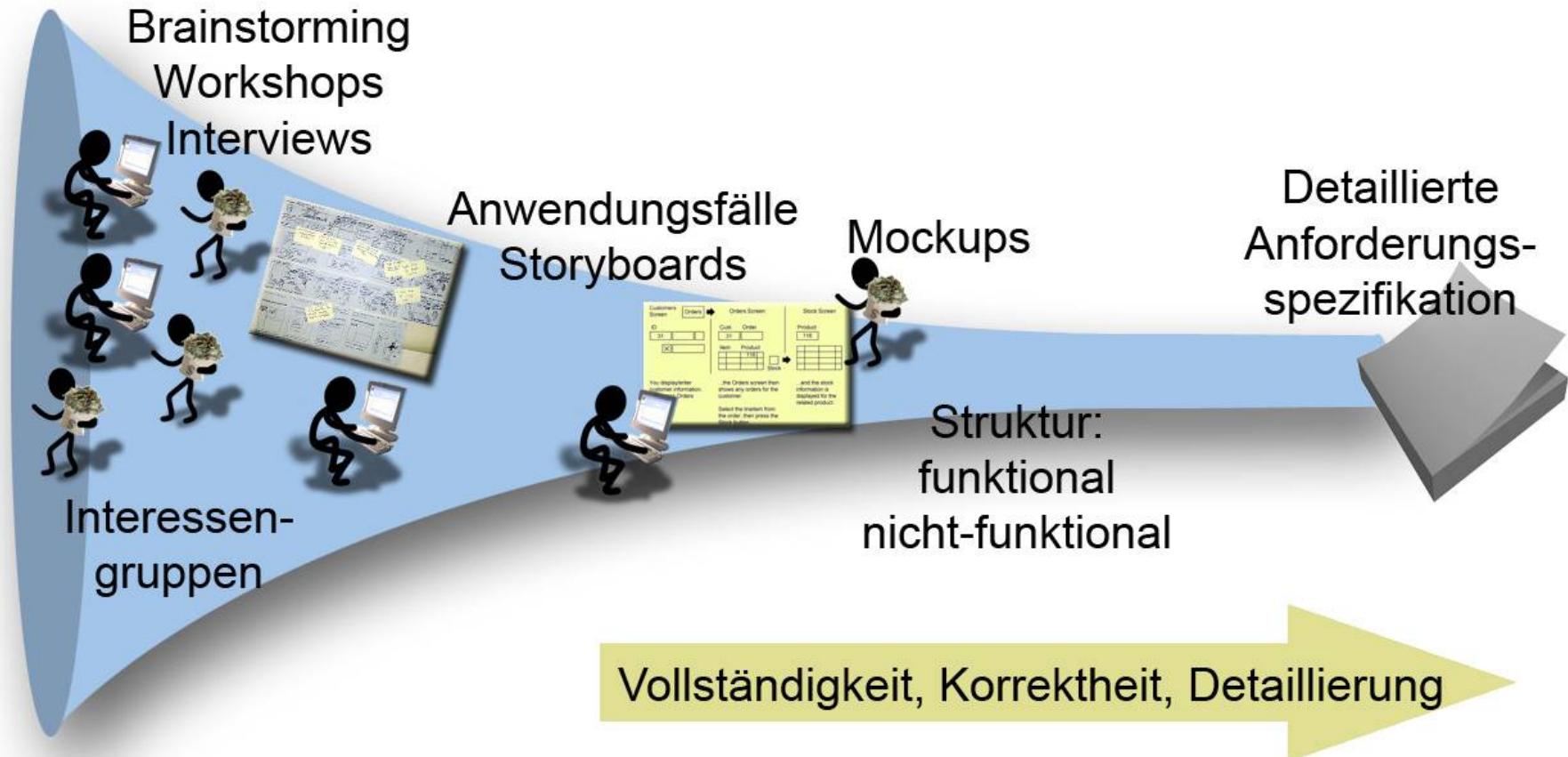
Systematische Anforderungsanalyse

Woher kommt der 'Input'?

Was zeichnet ein gutes Lastenheft / Pflichtenheft aus?

Wie kann eine hohe Qualität sichergestellt werden?

Anforderungsanalyse Vorgehen



Das **Pflichtenheft** definiert die Anforderungen an das zu liefernde (Software-)Produkt. Basierend auf dem Lastenheft wird es vom Auftragnehmer in Abstimmung mit dem Auftraggeber durch Ausarbeitung weiterer Anforderungsdetails erstellt. Das Pflichtenheft ist verbindliche Grundlage des abzuschließenden Vertrages.

Beispielstruktur eines Pflichtenhefts (nach Balzert, Softwaretechnik):

1. **Ziele:** Was muss das Produkt unbedingt leisten? Was ist wünschenswert? Was ist nicht Produktbestandteil?
2. **Einsatzbereich:** Anwendungsbereiche? Zielgruppen?
3. **Umgebung:** Welche Hardware- / Softwareumgebung muss unterstützt werden?
4. **Funktionen:** Detaillierte, systematische Auflistung aller wesentlichen Produktfunktionen
5. **Daten:** Detaillierte, systematische Auflistung aller wesentlichen zu handhabenden Daten
6. **Leistungsmerkmale:** Auflistung aller nicht-funktionalen Anforderungen (Performanz, Datenvolumen, Skalierbarkeit, Bedienbarkeit, Zuverlässigkeit usw.)
7. **Benutzeroberfläche:** Grundlegende Anforderungen an die Benutzeroberfläche?
8. **Qualitätsziele:** Wie wird die Produktqualität sichergestellt? Welche Standards/Normen werden befolgt?
9. **Testszenarien:** Anhand welcher Testfälle wird die Erfüllung der Anforderungen überprüft?
10. **Entwicklungsumgebung:** In welcher Umgebung / mit welchen Technologien wird die Software entwickelt?
11. **Ergänzungen:** Weitere wichtige Punkte, z.B. Lizenzen, Patente, 3rd Party Komponenten
12. **Glossar:** Definition von verwendeten Abkürzungen und Fachbegriffen



Software-Engineering-Projekt

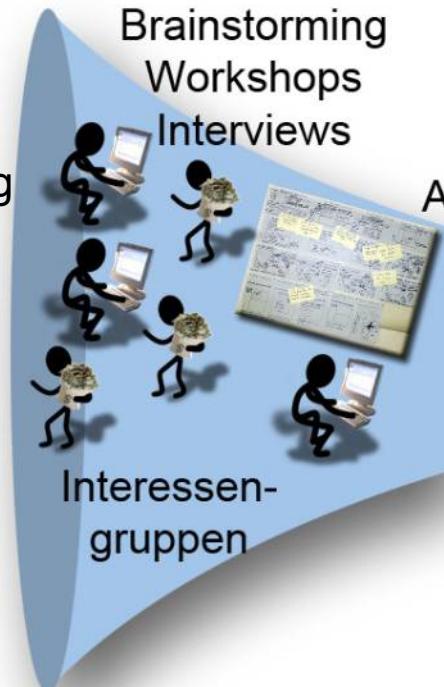
P.6 Lieferant: Pflichtenheft

Erstellen Sie auf Basis des Lastenheftes und des Gespräches mit Ihrem Kunden ein Pflichtenheft. Senden Sie das Pflichtenheft dem Kunden rechtzeitig, damit dieser vor dem nächsten Treffen noch genügend Zeit für die Durchsicht hat.

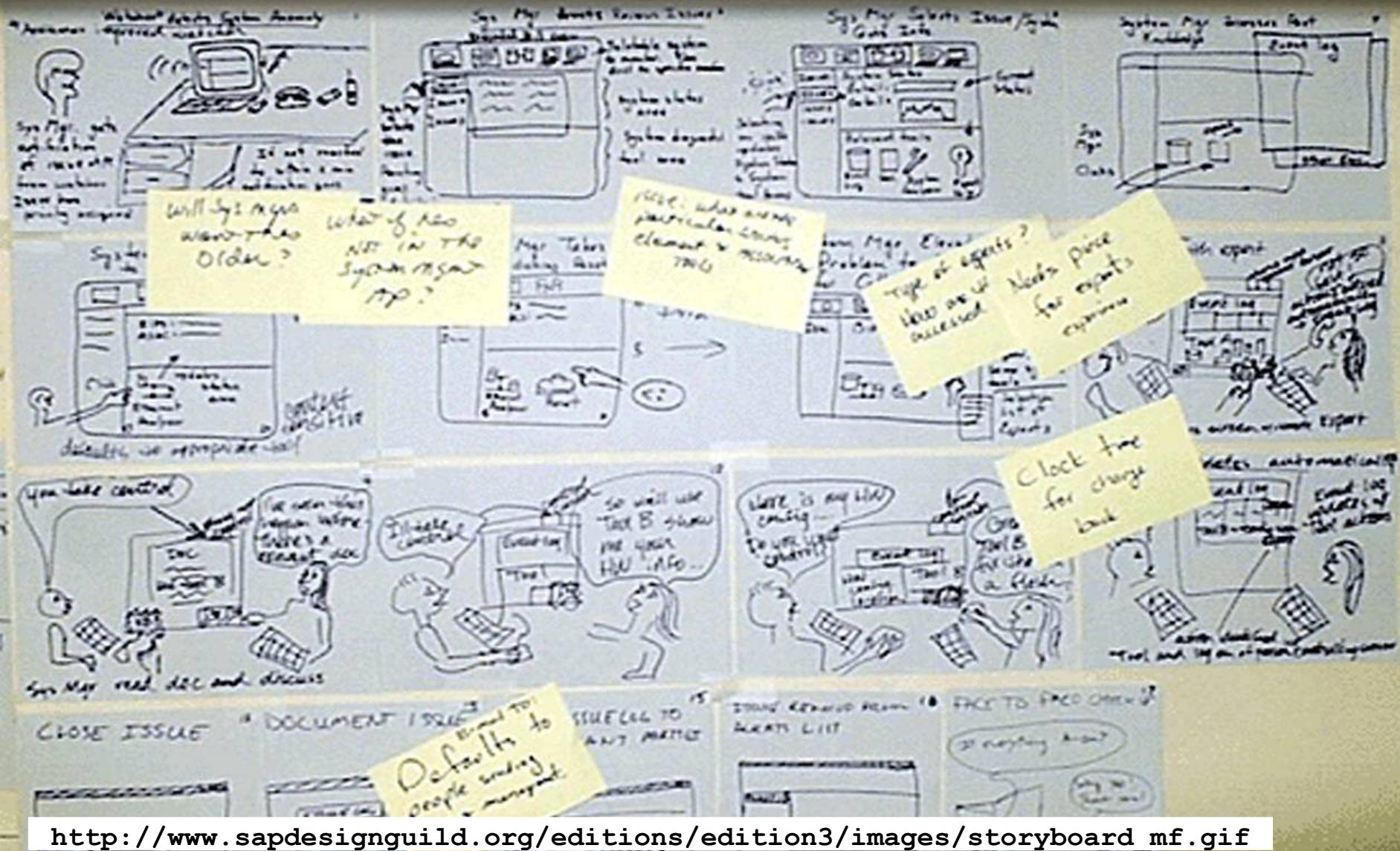
Ermittlung von Anforderungen

Ermittlung der Anforderungen (Requirements elicitation):

- Verschiedene Interessengruppen (stakeholders) einbeziehen:
 - z.B. Anwender, Administrator, Manager ...
 - Interviews, Diskussionen
- Verschiedene Sichten (Viewpoints) berücksichtigen:
 - Anwendersicht: z.B. einfache Bedienbarkeit
 - Administrator: z.B. Sicherheit, einfache Installation und Verwaltung
 - Manager: z.B. Total cost of ownership
 - ...
- Verschiedene Techniken verwenden:
 - Workshops
 - Brainstorming
 - Anwendungsfälle (typische Funktionsabläufe), Storyboards
 - (UI) Mockups
- Umfeldanalyse:
 - Andere Produkte auf diesem Markt, Wettbewerber?
 - Trends (Technologien, Benutzererwartungen)
 - Richtlinien und Standards (im Unternehmen, im Marktumfeld)

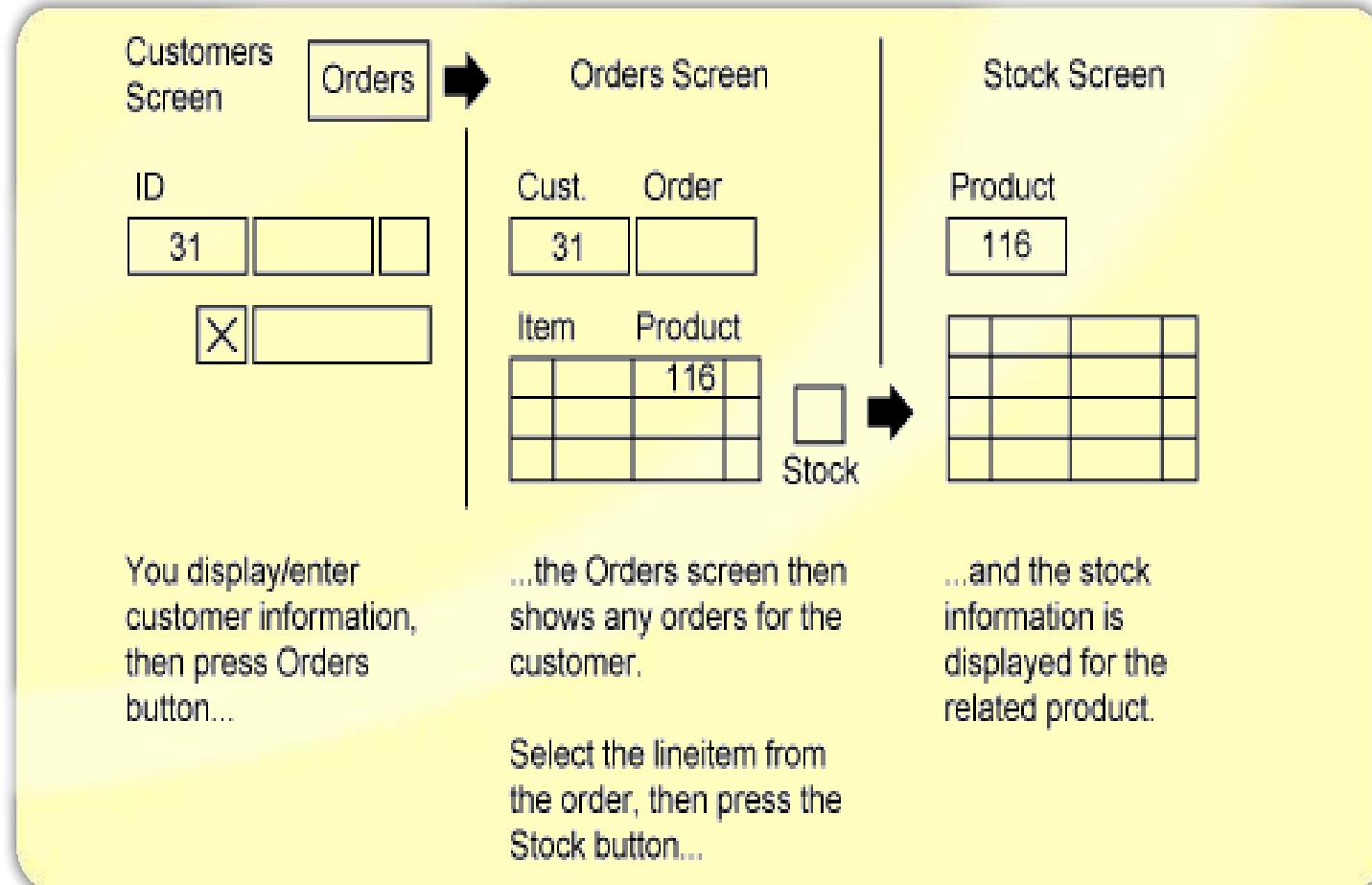


Storyboard (Brainstorming)



http://www.sapdesignguild.org/editions/edition3/images/storyboard_mf.gif

Storyboard



<http://sqltech.cl/doc/dev2000/guide21/gua3.gif>

Ein **Anwendungsfall (Use case)** beschreibt einen typischen Vorgang aus mehreren zusammenhängenden Arbeitsschritten, die von einem **Akteur (Actor)** durchgeführt werden, um ein bestimmtes Ziel zu erreichen / eine Aufgabe zu erledigen.

Beispiel:

#15 Adding a User

A new user needs an account to access the system. A new user is added to the system by creating an account consisting of a user name, password, and home directory.

Priority: 5 Estimation: Easy

#16 Logging onto the System

A user logs onto the system by entering their username and password. If they enter an incorrect value for either field, they are provided an error message indicating. if they enter three incorrect values, their account is locked and the user must inform the IT department to restore their account.

Priority: Medium Estimation: 3

#135 Paying for a Book

A customer may pay for a book using a credit card (VISA,AMEX,MC), PayPal, or by check.

Priority: 12 Estimation: 1

UML Use Case Diagram

UML Use Case Diagram (Anwendungsfalldiagramm)

System Boundary (Systemgrenze):

System, das die benötigten Anwendungsfälle bereitstellt und mit dem die Anwender interagieren.

Actor (Akteur):

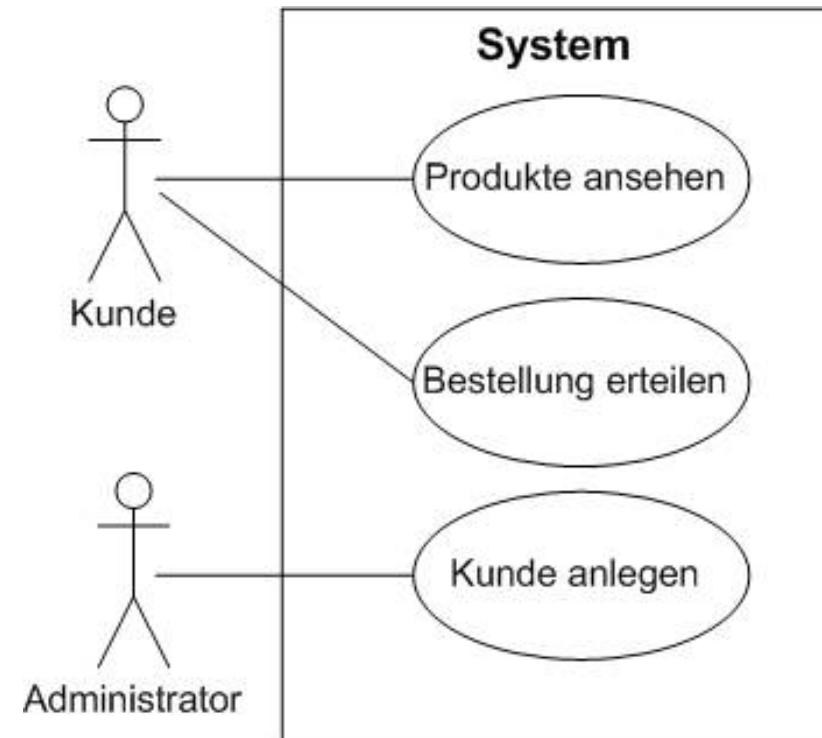
Typ oder Rolle, die ein externer Benutzer oder ein externes System während der Interaktion mit einem System einnimmt.

Use Case (Anwendungsfall):

Eine abgeschlossene Menge von Aktionen, die von einem System bereitgestellt werden und definierten Nutzen für einen oder mehrere Actors erbringen.

Association (Assoziation):

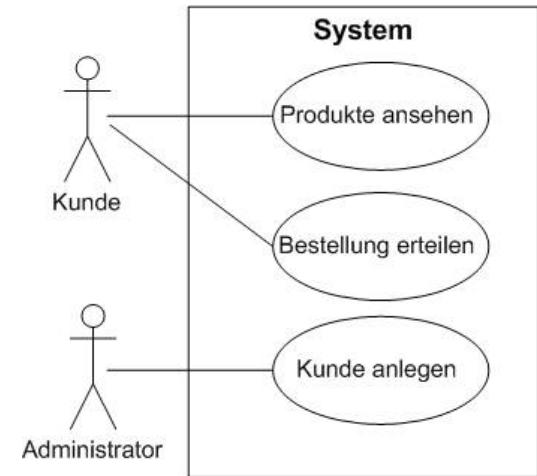
Beziehung zwischen Akteuren und Anwendungsfällen.



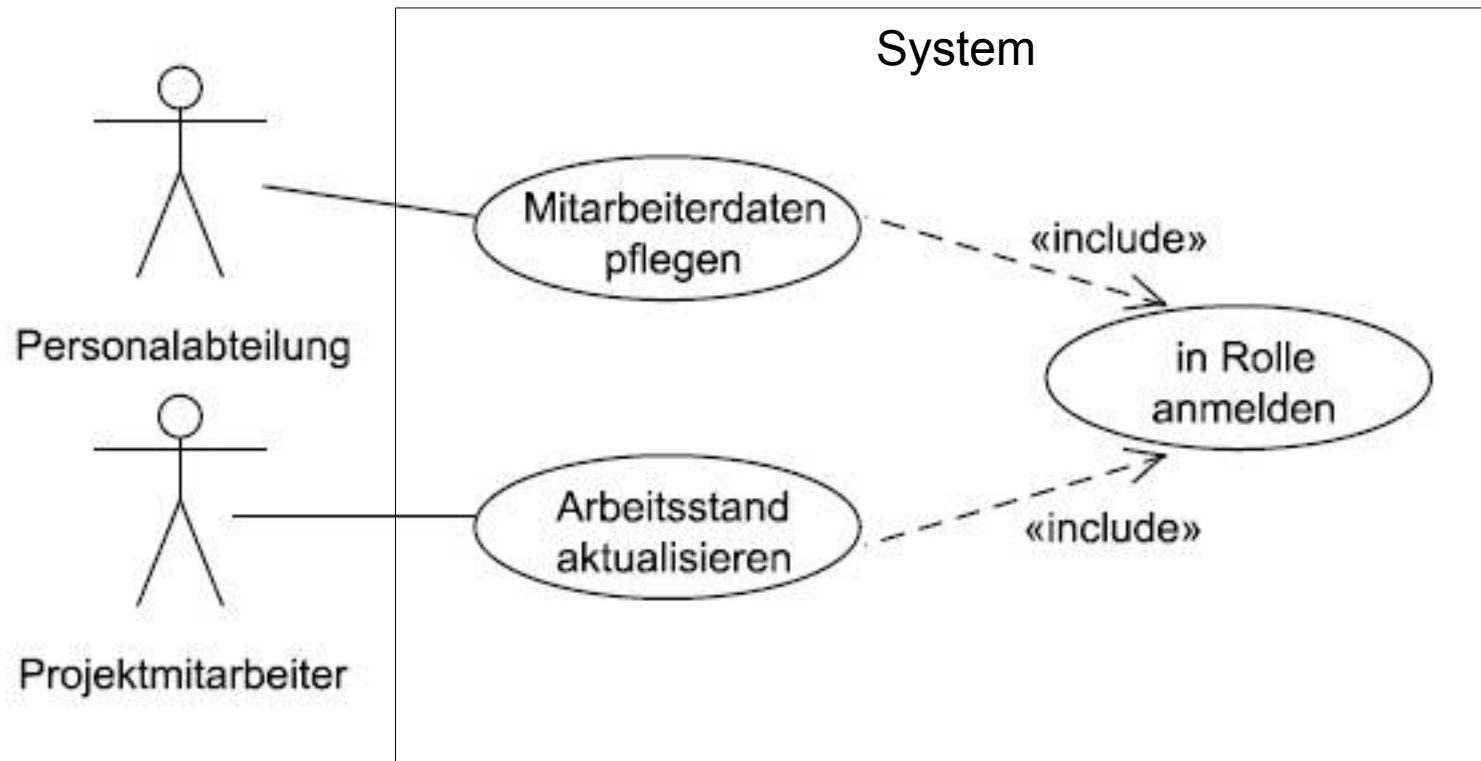
Use Case Details - Checkliste

Jeder Use Case sollte detailliert dokumentiert werden:

- Name (typisch: Substantiv+Verb)
- Eindeutige Kennung/Nummer
- Priorisierung (wie wichtig ist der Use Case?)
- Kurze Beschreibung
- Beteiligte Actors
- Vorbedingungen (z.B. System ist gestartet, in Modus xy...)
- Typischer Ablauf (ggf. mit Alternativen, z.B. Fehlerfall)
- Nachbedingungen (gewünschtes Ergebnis)
- Referenzen: Verweise auf weitere Informationen/Quellen und zu anderen Use Cases
- Autor, Version, Datum
- Fachverantwortlicher (Kundenseite)



Use Case Hierarchy



Benutzerorientierte Analyse

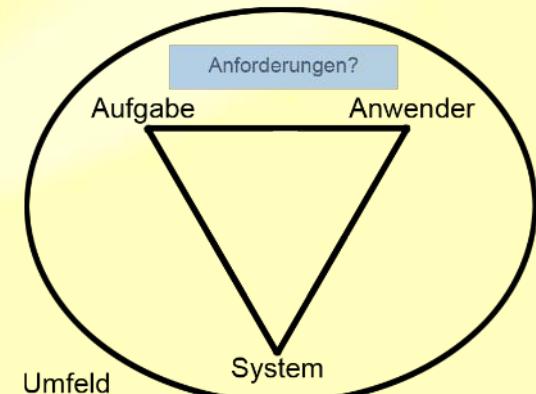
- Benutzeranalyse
 - Modell von künftigen Anwendern entwickeln
- Aufgabenanalyse
 - Welche Aufgaben (mit welchem Ziel) sollen vom Anwender mit dem System erledigt werden?
 - Anwendungsszenarien, typische Abläufe
- Verhalten und Wünsche des Anwenders
 - Anwenderverhalten bei bisherigen Systemen, Prototypen, beobachten
 - Interviews, Fragebögen
 - Anwenderverhalten bei Konkurrenzprodukten (kompetitive Analyse)
- Usability Ziele
 - Frühe Festlegung von Usability Zielen
 - Frühe/regelmäßige Überprüfung mit Mockups und Prototypen



User Story

Eine **User Story** ist ein knappe (ein, zwei Sätze) aus Anwendersicht in Alltagssprache formulierte Anforderung, die eine gewünschte Funktion beschreibt und begründet. User Stories sind insbesondere in der agilen Softwareentwicklung üblich.

- *Der Support-Mitarbeiter möchte jederzeit eine Übersicht aller Bestellungen des Kunden anzeigen können, um zu wissen, welche Produkte der Kunde seit wann verwendet.*
- *Wenn der Anwender das Fenster schließt und die Datei noch nicht gespeichert ist, soll eine Sicherheits-/Speichern-Abfrage kommen, damit Dateiänderungen nicht versehentlich verloren gehen.*



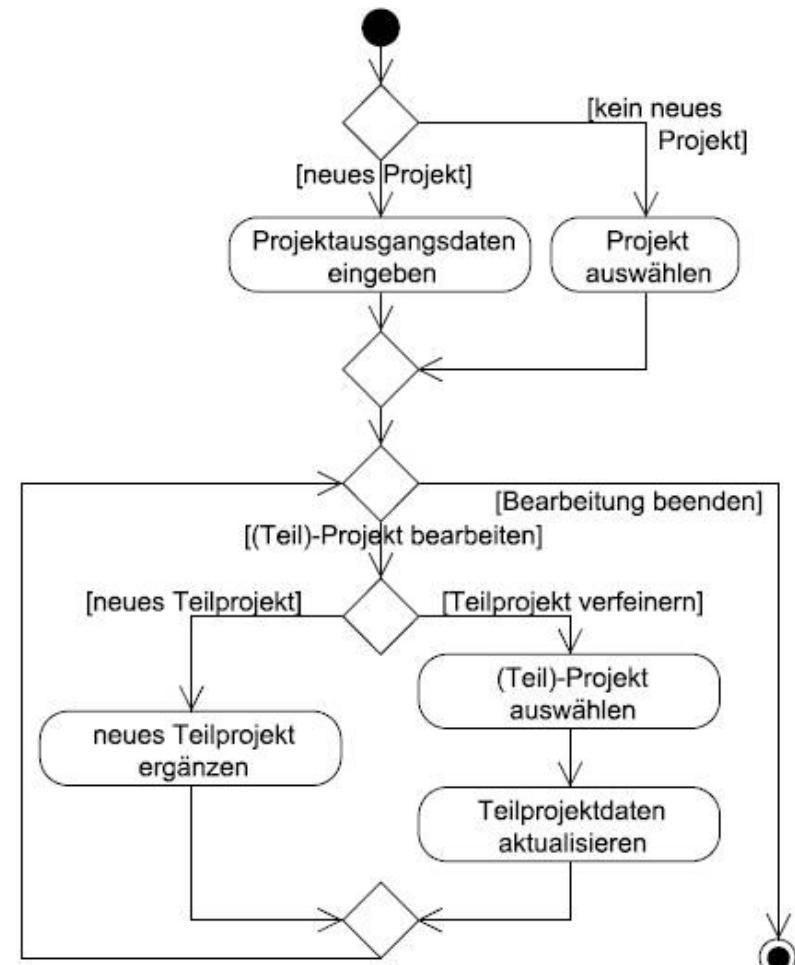
User-Story-Template

(Mike Cohn: User Stories Applied. For Agile Software Development)

"As a (role) I want (something) so that (benefit)."

UML Activity Diagram

Um Abläufe zu beschreiben lassen sich **UML Aktivitätsdiagramme (Activity Diagrams)** einsetzen.



Außerdem möglich:
Hierarchische Gliederung → zusätzliche
Diagramme für die detaillierte
Beschreibung einzelner Aktivitäten

Lastenheft und Pflichtenheft



Funktionale Anforderungen

- Wozu soll die Software dienen?
 - Aus Sicht des Kunden/der Benutzer (Anwender, Administrator etc.)
- Entscheidend: "Was" macht die Software – nicht "wie" macht sie es.
- Strukturierte Liste aller wesentlichen Funktionen / Features
- ggf. zusätzlich: typische, anschauliche Anwendungsfälle
- Angemessene Detaillierungsebene
 - die wesentlichen Abläufe, Sichten und Interaktionsschritte
 - aber nicht jeden einzelnen Mausklick

Nicht-funktionale Anforderungen

Nicht-funktionale Anforderungen (Qualitätsmerkmale)

- Leistung: Performanz, Durchsatz, Skalierbarkeit, Ressourcenbedarf
- Benutzersicht: Einfache Bedienung (Usability), Look&Feel, einfache Installation
- Zuverlässigkeit, Ausfallsicherheit, Verfügbarkeit
- IT-Sicherheit (Security), Datenschutz
- Technologieumfeld, Kompatibilität, Standards: z.B: unterstützte Browser, Betriebssysteme, PC-Anforderungen
- Total cost of ownership: Wartbarkeit, Erweiterbarkeit, Portierbarkeit

Sonstige Anforderungen im Lasten-/Pflichtenheft

- Dokumentation, Training, Einarbeitung
- Abnahmekriterien, geforderte Testprotokolle
- Vorgehen bei Übergabe/Deployment
- Nachweis der Qualitätssicherung
- Wartung und Support (Garantien)

Übung

2.1 Anforderungsfehler

Anforderungsfehler, d.h. falsch erfasste Anforderungen, unterscheiden sich in verschiedener Hinsicht von Softwarefehlern, die während des Entwurfs und der Implementierung auftreten.

- Diskutieren Sie die Unterschiede.
- Wie kann sichergestellt werden, dass es möglichst keine Anforderungsfehler gibt bzw. die Konsequenzen begrenzt werden?

Anforderungsfehler

- Entstehen früh im Prozess → potenziell hohe Kosten, da in der Folge viel Arbeit davon abhängt.
- Werden nicht beim Testen im Rahmen der normalen Softwareentwicklung erkannt, sondern potenziell erst bei der Endabnahme / im späteren Betrieb.
- Kunde/Auftraggeber trägt i.d.R. die Hauptverantwortung für derartige Fehler

Wie vermeiden?

- Saubere Anforderungsanalyse
- Kunden eng einbeziehen
- ggf. Iterationen / Zwischenchecks
- Werkzeuge zum Anforderungsmanagement

Lastenheft und Pflichtenheft

Was zeichnet ein gutes Lastenheft / Pflichtenheft aus?

- Verständlichkeit: Klare Struktur, zusammenhängender, "lesbarer" Text, Motivation/Hintergründe/Erläuterungen; Zusammenfassen, was zusammengehört.
- Vollständigkeit
- Genauigkeit und Korrektheit: Möglichst präzise Angaben, Keine Widersprüche, Prioritäten, sorgfältig überprüft, Glossar/Begriffsdefinitionen
- Effizienz: Möglichst wenig Redundanz, keine überflüssigen Angaben (die evtl. in andere Dokumente gehören, z.B. Implementierungsdetails)
- Nachverfolgbarkeit: Kennungen für Anforderungen, Dokumentenhistorie

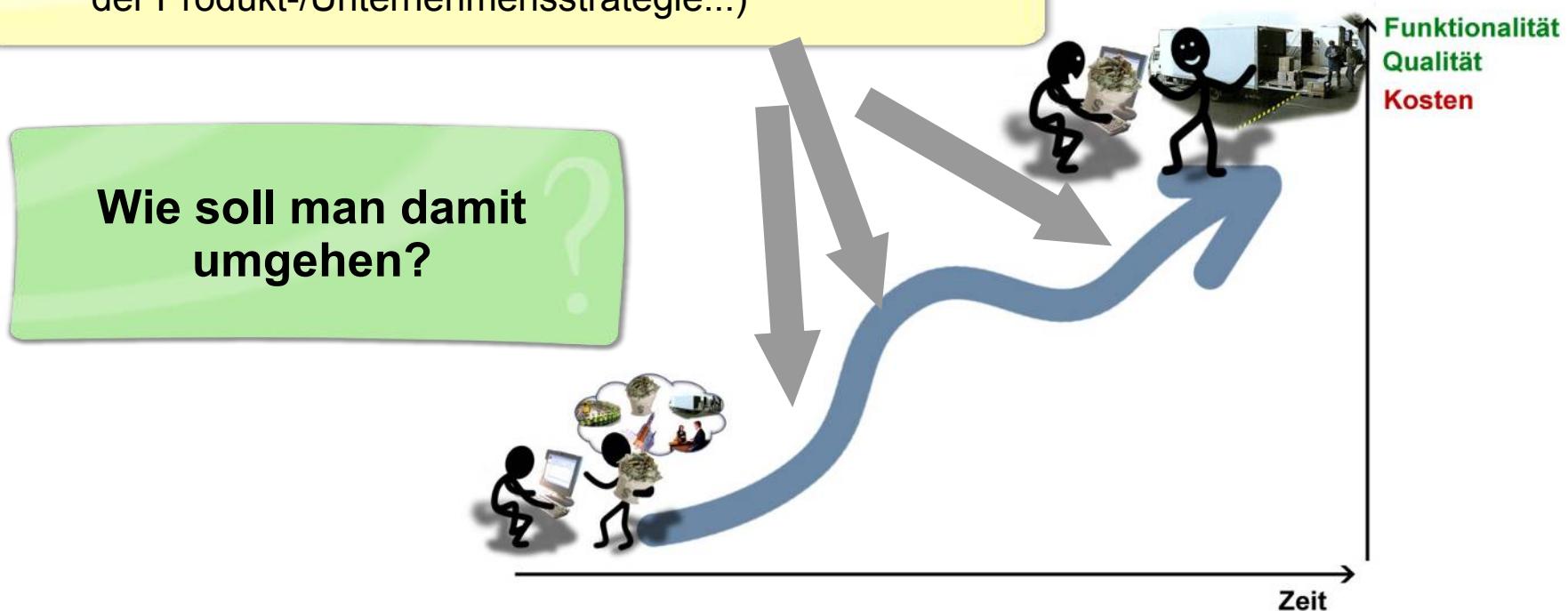
Typisch: Verwendung von vorgegebenen, unternehmenseigenen Dokumentenschablonen (Templates), um sicherzustellen, dass alle Punkte erfasst werden.

Das Pflichtenheft ist Bestandteil des Vertrages (und somit ggf. maßgeblich bei rechtlichen Auseinandersetzungen). Sorgfalt ist sehr wichtig!

Anforderungsmanagement

Anforderungen können sich über die Projektlaufzeit und auch nach Release des Produktes verändern:

- Fehler bei der Anforderungsanalyse werden erkannt
- Neue, unvorhergesehene Kundenwünsche
- Anforderungen von außen (neue Standards, Änderung der Produkt-/Unternehmensstrategie...)



Anforderungsmanagement

Anforderungen können sich über die Projektlaufzeit und auch nach Release des Produktes verändern:

- Fehler bei der Anforderungsanalyse werden erkannt
- Neue, unvorhergesehene Kundenwünsche
- Anforderungen von außen (neue Standards, Änderung der Produkt-/Unternehmensstrategie...)

gewünscht:

- Systematische Verfolgung von Anforderungen und Änderungen
- definierter Entscheidungsprozess (mit Kosten-Nutzen-Abwägung)
- Möglichkeit zur Umplanung des Projektes

Flexibilität hängt auch vom Vorgehensmodell ab!

Übung

2.2 Werkzeuge für Anforderungsmanagement

- a) Recherchieren Sie im Internet nach Werkzeugen für Anforderungsanalyse und Anforderungsmanagement.
- b) Was für besondere Möglichkeiten bieten diese Werkzeuge, z.B. im Vergleich mit normalen Büroanwendungen wie z.B. Textverarbeitung.
- c) Diskutieren Sie Für und Wider des Einsatzes dieser Werkzeuge in Ihrem Projekt.



Software-Engineering-Projekt

P.7 Kunde: Review Pflichtenheft

Begutachten Sie das von Ihrem Softwarelieferanten erstellte Pflichtenheft. Sind aus Ihrer Sicht alle wichtigen Punkte hinreichend detailliert erfasst? Gibt es Punkte, die vor Vertragsabschluss noch geklärt oder überarbeitet werden müssen?

Kunde-Lieferant: Vertragsabschluss?



Software-Engineering-Projekt

P.8 Kunde-Lieferant: Vertragsabschluss?

Der Kunde und Lieferant möchten das Pflichtenheft bzw. evtl. erforderliche Änderungen besprechen. Wenn die Besprechungen erfolgreich verlaufen, kann (nach ggf. noch kleineren Änderungen am Pflichtenheft) der Kunde den Auftrag an den Lieferanten erteilen. Falls es noch größere Differenzen gibt, müssen weitere Treffen und Diskussionen geplant werden.

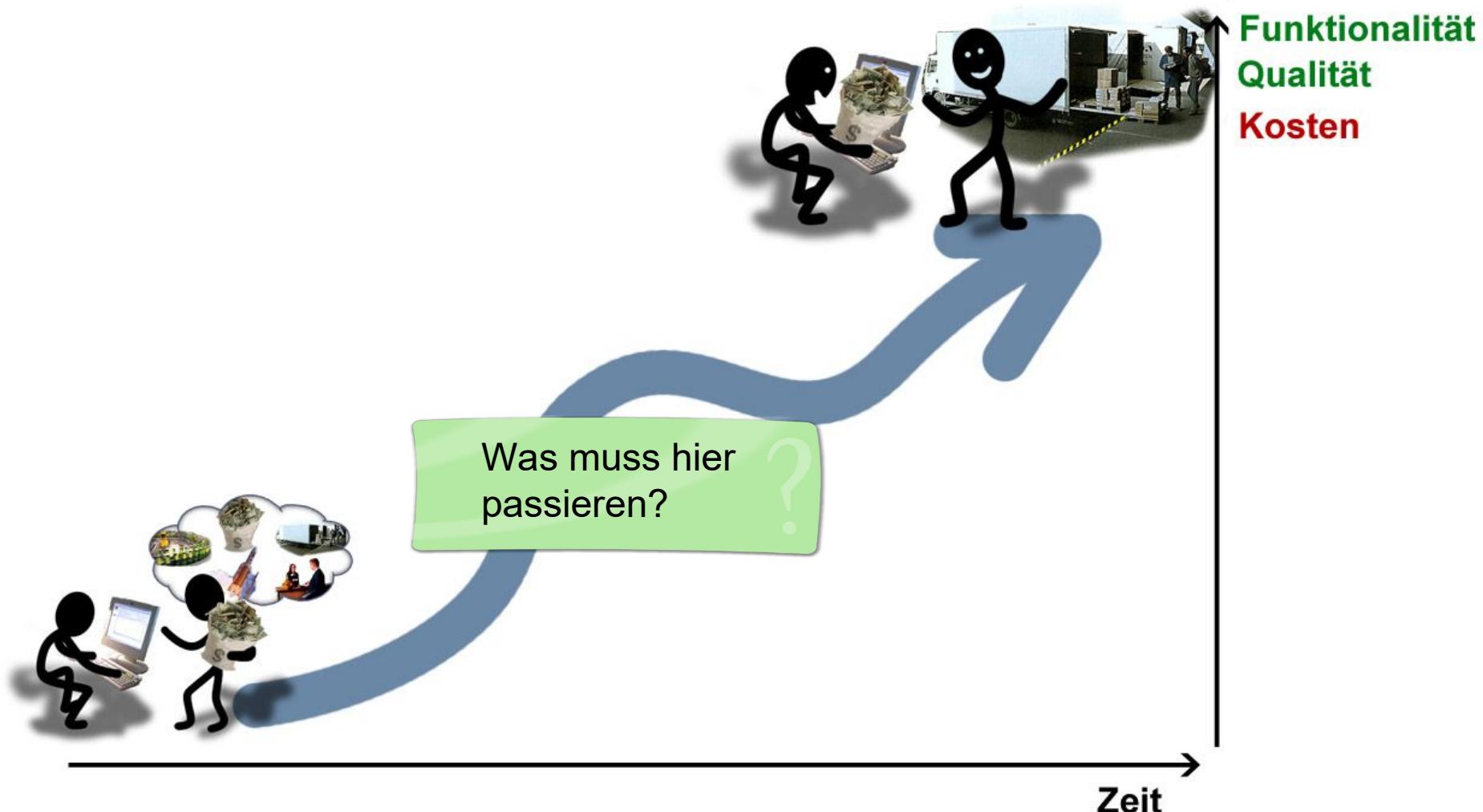
Software Engineering I

3. Vorgehensmodelle und Phasen

Prof. Dr. Eckhard Kruse

DHBW Mannheim

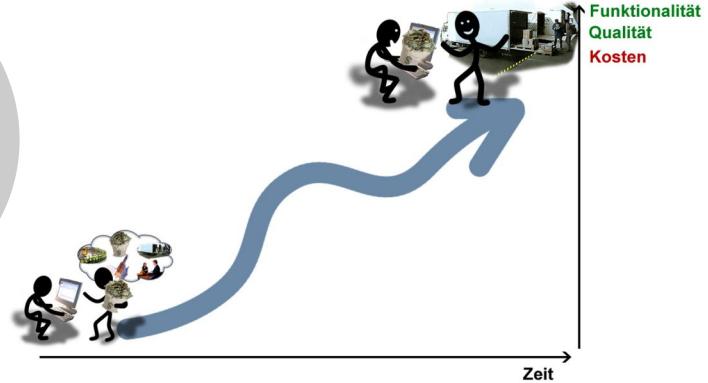
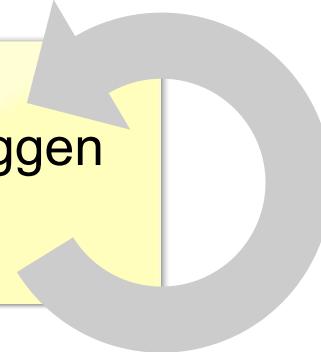
Vorgehensmodelle



„Code and fix“

Beispiel

1. Code schreiben
2. compilieren, testen, debuggen
3. Code verbessern
4. Nicht fertig? Gehe zu 2.



Reicht das?

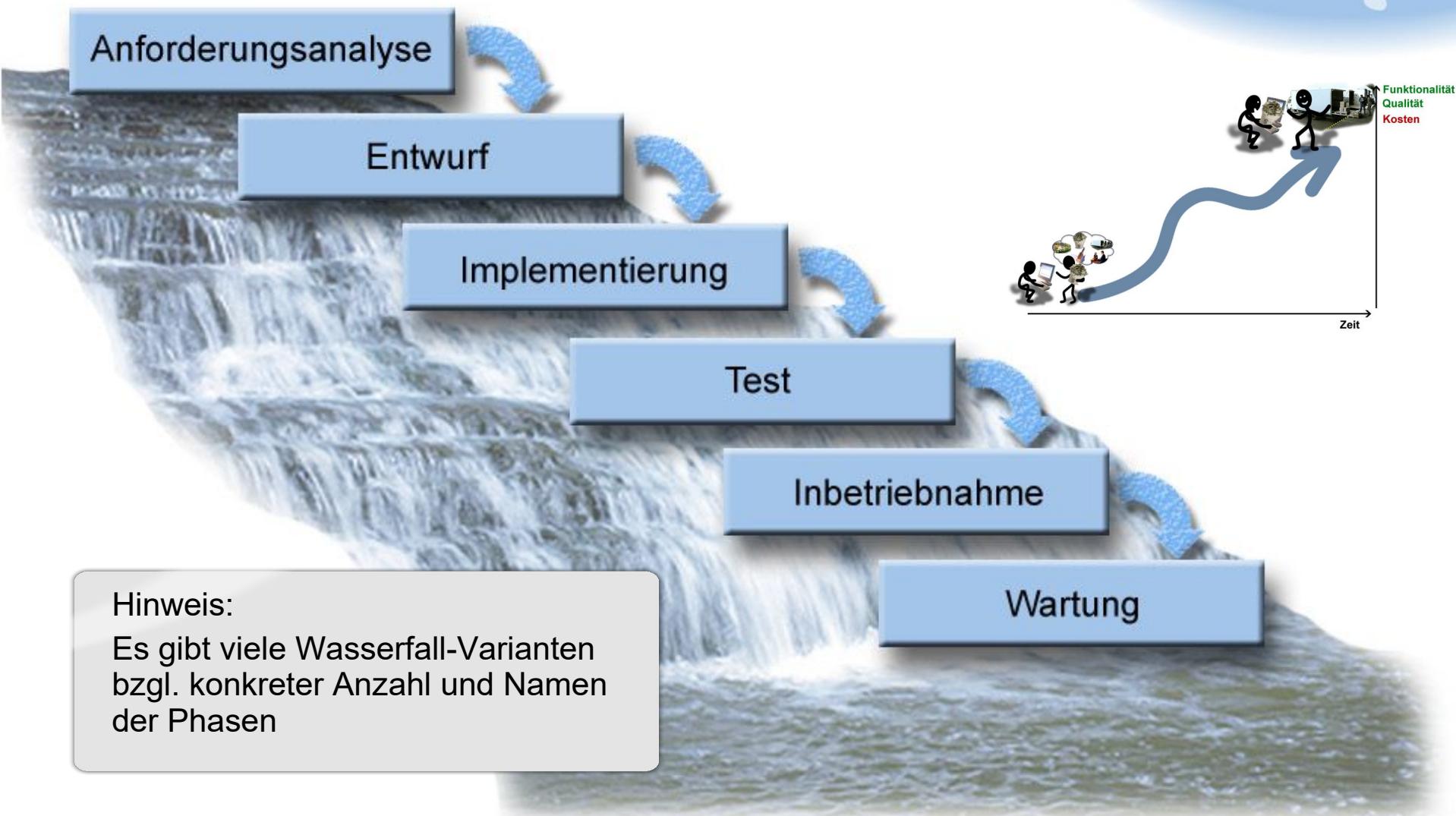


- Keine dokumentierten Anforderungen / Interaktion mit dem Kunden?
- Keine Planung: Wann wird es fertig? Zu welchem Preis?
- Wartung? Dokumentation?
- ...

→ “Hacker“-Ansatz: Bestenfalls für Mini-Projekte (und tolerante Kunden) geeignet.

Wasserfallmodell

Definition!



Übung

3.1 Wasserfallmodell

Untersuchen Sie das Wasserfallmodell:

- a) Recherchieren Sie im Internet nach dem Wasserfallmodell. Was für Information und Meinungen finden Sie dazu?
- b) Nennen Sie Vor- und Nachteile des Modells (auch im Vergleich zu „Code und Fix“).
- c) Können Sie sich Verbesserungen für dieses Modell vorstellen?

Wasserfallmodell - Bewertung

Übung?



Gut:

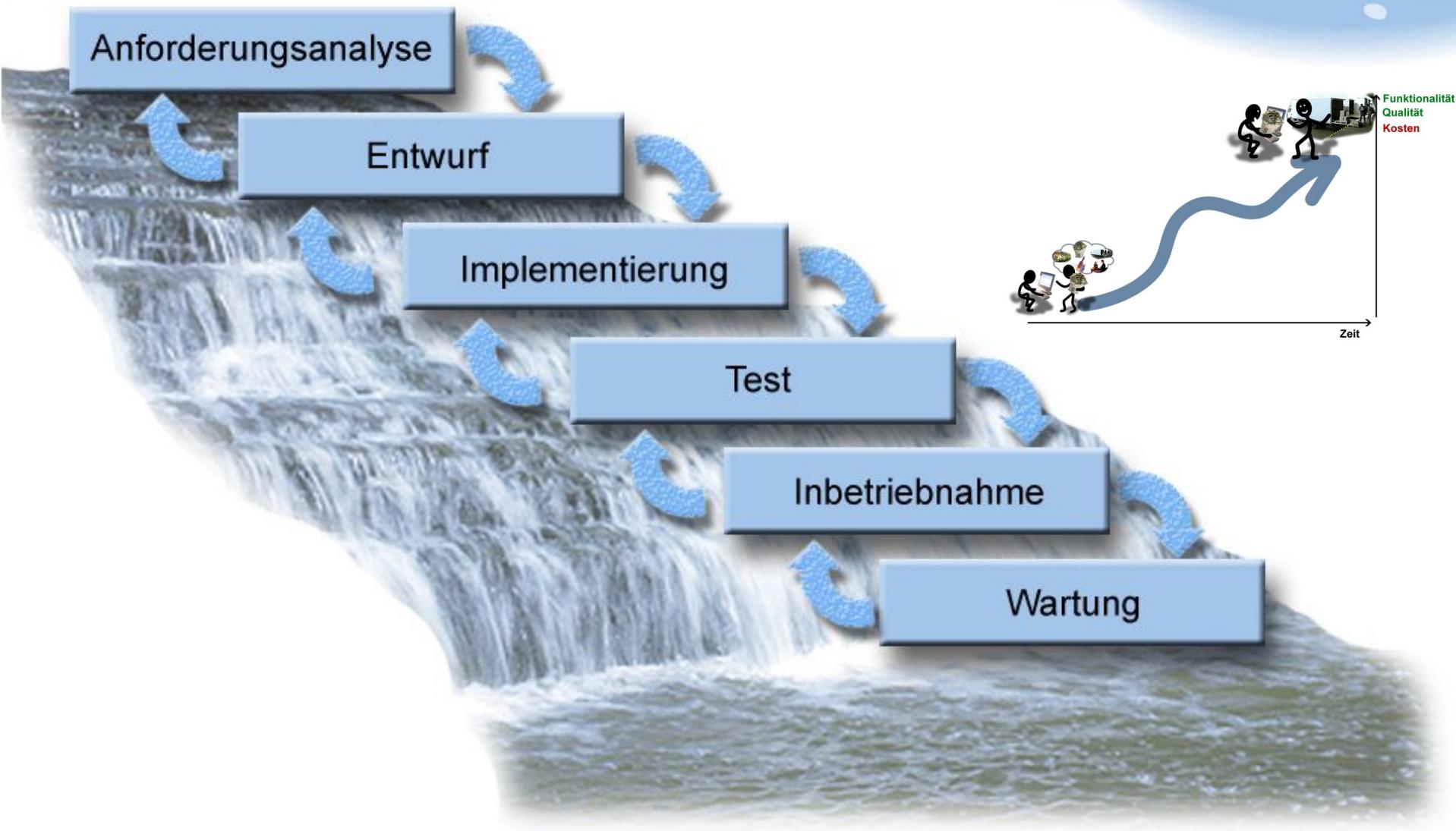
- Definierte Phasen
- Neben Implementierung, werden auch Analyse, Testen etc. explizit berücksichtigt.
- Gute Grundlage für Projektplanung

Schlecht:

- sehr idealisiert, reale Welt braucht oft mehrere Schleifen
- Anforderungen ändern sich
- neue Erfahrungen/Erkenntnisse schwer zu berücksichtigen
- „Planwirtschaft“

Wasserfallmodell mit Rücksprung

Definition!



Im Software-Engineering unterscheidet man verschiedene **Phasen**, wie z.B. **Planung, Anforderungsanalyse, Entwurf, Implementierung, Integration, Inbetriebnahme und Wartung**. Bzgl. der Anzahl und genauen Bezeichnung der Phasen gibt es viele Varianten.

Abhängig vom **Vorgehensmodell** werden die Phasen verschieden abgearbeitet. Unterschiede gibt es z.B. bzgl. der Bedeutung der einzelnen Phasen, ihrer zeitlichen Reihenfolge und Abhängigkeiten, eventuelle Unterteilung in Teilabschnitte, Zyklen usw.

Übung

3.2 Phasen im Software-Engineering

Im Software-Engineering unterscheidet man typischerweise die folgenden Phasen: Planung, Anforderungsanalyse, Entwurf, Implementierung, Test, Inbetriebnahme und Wartung.

- a) Überlegen Sie, welche Aktivitäten in den einzelnen Phasen stattfinden könnten.
- b) Überlegen Sie, welche Personen/Rollen (z.B. Kunde, Software-Entwickler, Architekt, Business Analyst, Unterlieferant ...), in den einzelnen Phasen mitarbeiten sollten und welche Aufgabe sie dort haben.
- c) Was ist wichtig für die einzelnen Phasen? (Was tun? Was lassen?)
- d) Was passiert, wenn eine Phase weggelassen/nicht richtig bearbeitet wird? Gehen Sie dazu der Reihe nach die Phasen durch.

Bevor in einem Projekt in eine detaillierte Anforderungsanalyse durchgeführt wird, sollte eine grobe **Planung** aller Projektphasen erfolgen bzw. die grundsätzliche **Machbarkeit** des Projektes untersucht werden.

- Bei kleinen Projekten kann die erste Planung u.U. sehr schnell erfolgen
- Bei großen Projekten sind möglicherweise umfangreiche Vorstudien, Technologieuntersuchungen, Wirtschaftlichkeitsanalysen usw. erforderlich, die selbst wiederum als kleines Projekt gemanaget werden sollten.
- **Ergebnisse:** Wirtschaftlichkeitsbetrachtung, Go/No-go-Entscheidung, ein erster, grober Projektplan, Lastenheft (seitens des Kunden).

Die **Anforderungsanalyse (requirements engineering)** hat zum Ziel, alle wesentlichen Anforderungen an das zu entwickelnde System systematisch zu ermitteln und zu dokumentieren.

- Neben den Anforderungen des Autraggebers sind in der Regel auch unternehmensinterne Anforderungen (z.B. verwendete Standardtechnologien, Unternehmens-/Produktstrategie) zu berücksichtigen.
- Anforderungen werden typischerweise in Prioritäten eingestuft (z.B. 1=“Muss“, 2=“wünschenswert“, 3=“optional“)
- **Ergebnis:** Pflichtenheft, welches als Bestandteil des Vertrages mit dem Kunden verbindlich ist.

Der **Entwurf** (engl. **design**) dient dazu, auf Basis der Anforderungen schrittweise eine entsprechende technische Realisierung zu entwerfen. Hierzu wird das System schrittweise vom Groben ins Feine in Komponenten und Verantwortlichkeiten zerlegt.

- Der Entwurf erfolgt auf verschiedenen Ebenen (je komplexer das System, desto mehr Ebenen).
- Typische Begriffe sind Systementwurf (=Gesamtsystem), Architektur (Gesamtsystem oder Teilsysteme / Komponenten), Design (auf Komponenten- / Modulebene). (Die Grenzen der Begriffe sind fließend.)
- Der Fokus liegt auf der Aufteilung in Komponenten/Module, Verantwortlichkeiten und Schnittstellen untereinander – nicht auf deren internen Realisierung.
- **Ergebnisse:** Entwurfsdokumente, Diagramme (z.B. UML)

Übung

3.3 Ende des Entwurfs

Mit den Phasen Planung, Anforderungsanalyse und Entwurf sind prinzipiell alle Vorarbeiten erfolgt, um die eigentliche Programmierung zu beginnen.

- a) Überlegen Sie, wieviel Zeit/Arbeitsaufwand des Projektes in diese Phasen investiert werden sollte.
- b) Die strikte Trennung aller Phasen ist in der Praxis selten. Nennen Sie Gründe, warum die Grenzen zwischen den frühen Projektphasen bzw. zwischen dem Entwurf und der Implementierung verwischen könnten.
- c) Wann wäre Ihrer Meinung nach ein Schritt zurück von einer späteren zu einer früheren Phase sinnvoll, wann nicht? Nennen Sie Beispiele.

Die **Implementierung (implementation)** ist die Umsetzung des Entwurfs durch Programmierung (Codierung) der einzelnen Komponenten / Module.

- Zur Implementierung gehört auch die Durchführung von grundlegenden Tests auf Modulebene sowie die Dokumentation des Quellcodes.
- Während der Implementierung können noch Entwurfsentscheidungen innerhalb der Module erforderlich sein. Die Schnittstellen / Verträge zwischen den Komponenten sollten aber möglichst nicht mehr verändert werden müssen.
- **Ergebnisse:** Dokumentierter Quellcode, kompilierte Komponenten/Systemteile, Testprotokolle.

Während der **Integration** werden die einzelnen Komponenten zum ablauffähigen Gesamtsystem zusammengesetzt und getestet.

- Das Kompilieren aller Quelldateien und anschließende Verbinden (Linken), Einbinden von Libraries usw. zum Gesamtsystem bezeichnet man als Build-Prozess. Bei großen Systemen kann der Buildprozess sehr komplex und aufwändig werden, so dass es z.B. spezielle Mitarbeiter gibt, die ausschließlich für den Buildprozess verantwortlich sind.
- Nach dem Build erfolgt überlicherweise die Installation der Software auf einem Testsystem für die Durchführung des Integrationstests.
- **Ergebnisse:** Ablauffähiges Gesamtsystem, Protokolle vom Build und den Integrationstests

Übung

3.4 Implementierung und Integration

- a) Die Phasen Implementierung und Integration sind in der Praxis typischerweise eng verzahnt. Warum?
- b) Was könnten wichtige Qualifikationen für Mitarbeiter sein, die in der Implementierung bzw. in der Integration arbeiten? Worin unterscheiden sich die Anforderungsprofile?
- c) Manche Softwarefirmen praktizieren „Daily Builds“. Was versteht man darunter? Welche Vorteile und Nachteile könnte es haben?

Bei der **Installation und Inbetriebnahme (Deployment)** wird die Software auf dem Zielsystem installiert, konfiguriert und ggf. abschließend getestet. Nach erfolgreicher Installation kann der produktive Einsatz der Software beginnen.

- Grundsätzlich sind zwei Szenarien zu unterscheiden:
 - Standardsoftware, die für einen breiten Markt bestimmt ist
 - Kundenspezifische Software, die eine maßgeschneiderte Lösung für das Kundenproblem darstellt.
- Vor der endgültigen Auslieferung (Release) einer Software werden oft Vorabversionen zu Testzwecken ausgeliefert:
 - Stichwörter: Alpha, Beta, Release Candidate, Release

Übung

3.5 Inbetriebnahme

Die Inbetriebnahme von maßgeschneiderter Software für einen speziellen Kunden und von Standardsoftware für einen breiten Markt weist wichtige Unterschiede auf. Untersuchen Sie diese Unterschiede:

- a) Nennen Sie Beispiele für beide Fälle.
- b) Diskutieren Sie die unterschiedlichen Anforderungen und mögliche Probleme in beiden Fällen. Was ist im Vergleich zwischen den beiden Szenarien jeweils einfacher oder schwieriger?

Nach der ersten Installation der Software und bis ihrem Einsatzende fallen regelmäßig Aufgaben wie Fehlerkorrekturen, Anpassungen oder kleinere Erweiterungen an. Diese werden als **Wartung** bezeichnet.

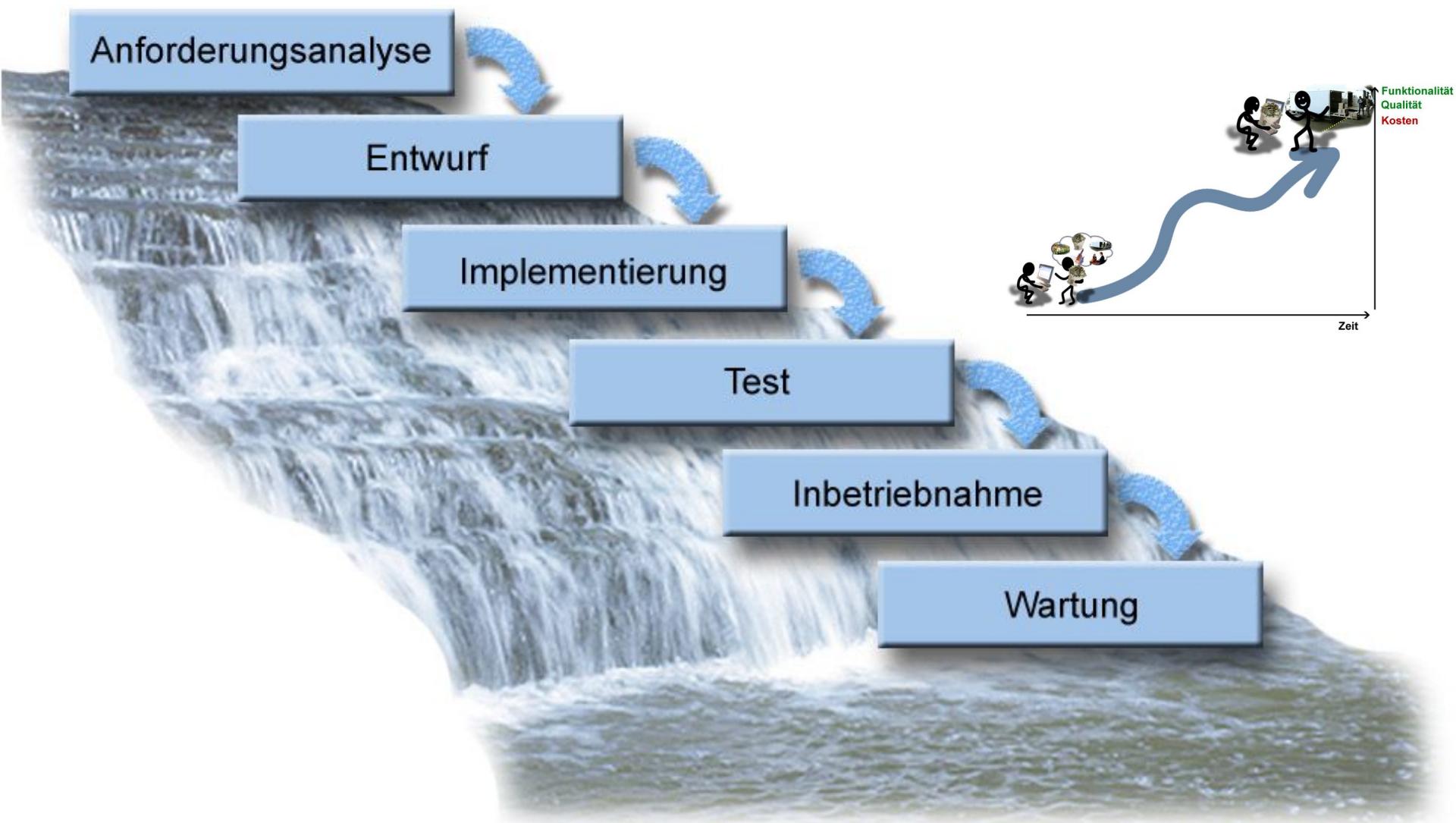
- Wartung macht oft einen Großteil der Gesamtkosten für die Softwareentwicklung aus.
- Heutzutage ist wohl keine frisch releasete Software „so fertig“, dass sie danach komplett ohne Wartung auskäme.
- Beispiele für Betriebssystemwartung: Microsoft-Patches und Updates.
- Bei größerem Änderungsbedarf, z.B. Entwicklung einer komplett neuen Produktversion spricht man nicht mehr von Wartung.
- **Ergebnis:** Ein System, das stets auf dem neuesten Stand ist und mit den aktuellen Technologien, Hardware, Betriebssystemversionen usw. funktioniert.

Übung

3.6 Wartung

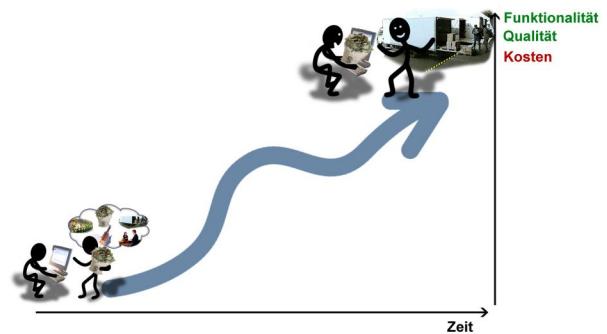
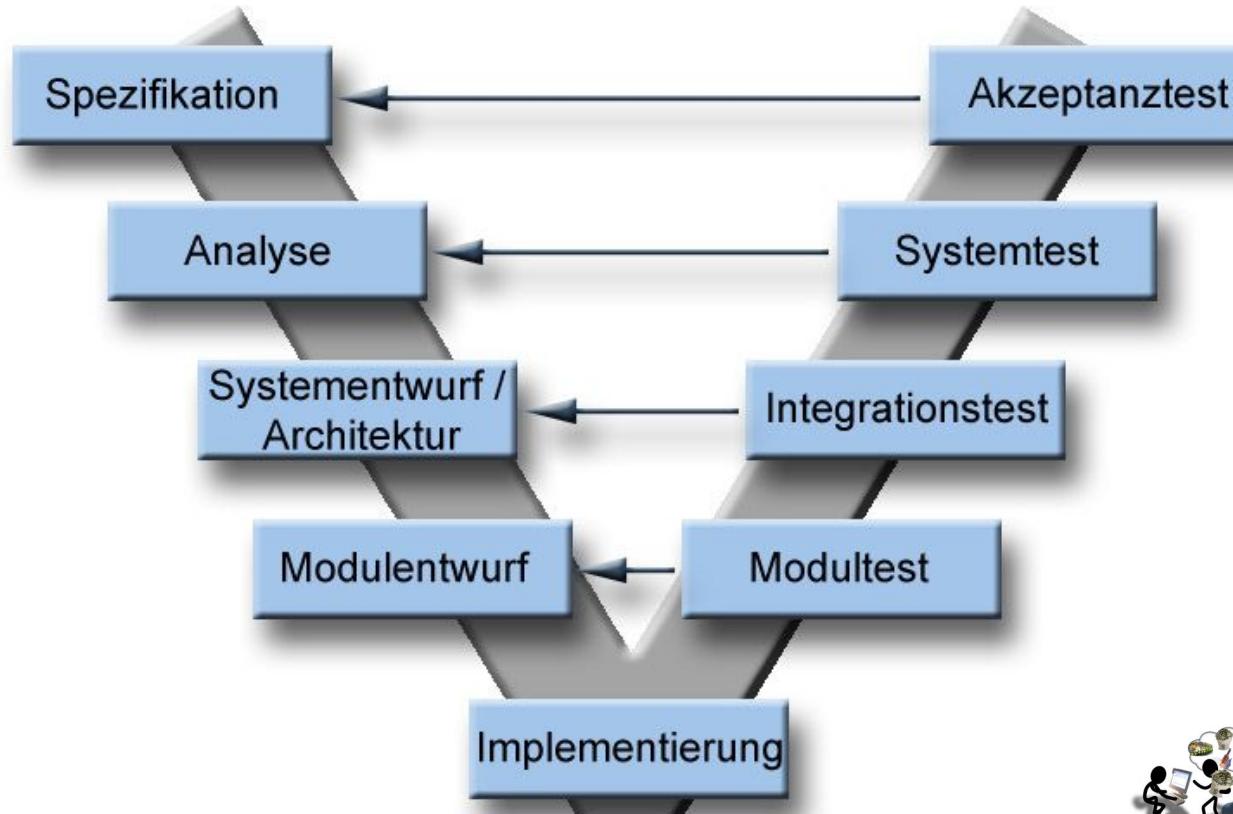
- a) Recherchen Sie im Internet über Software-Wartung (Software maintenance). Welche Aussagen finden Sie dort, z.B. über die Bedeutung der Wartung, Kosten und Verfahren.
- b) Von welchen Faktoren hängen Wartungskosten ab? Wie kann man Wartungskosten senken?

Wasserfallmodell



V-Modell

Definition!



Übung

3.7 V-Modell

Untersuchen Sie das V-Modell:

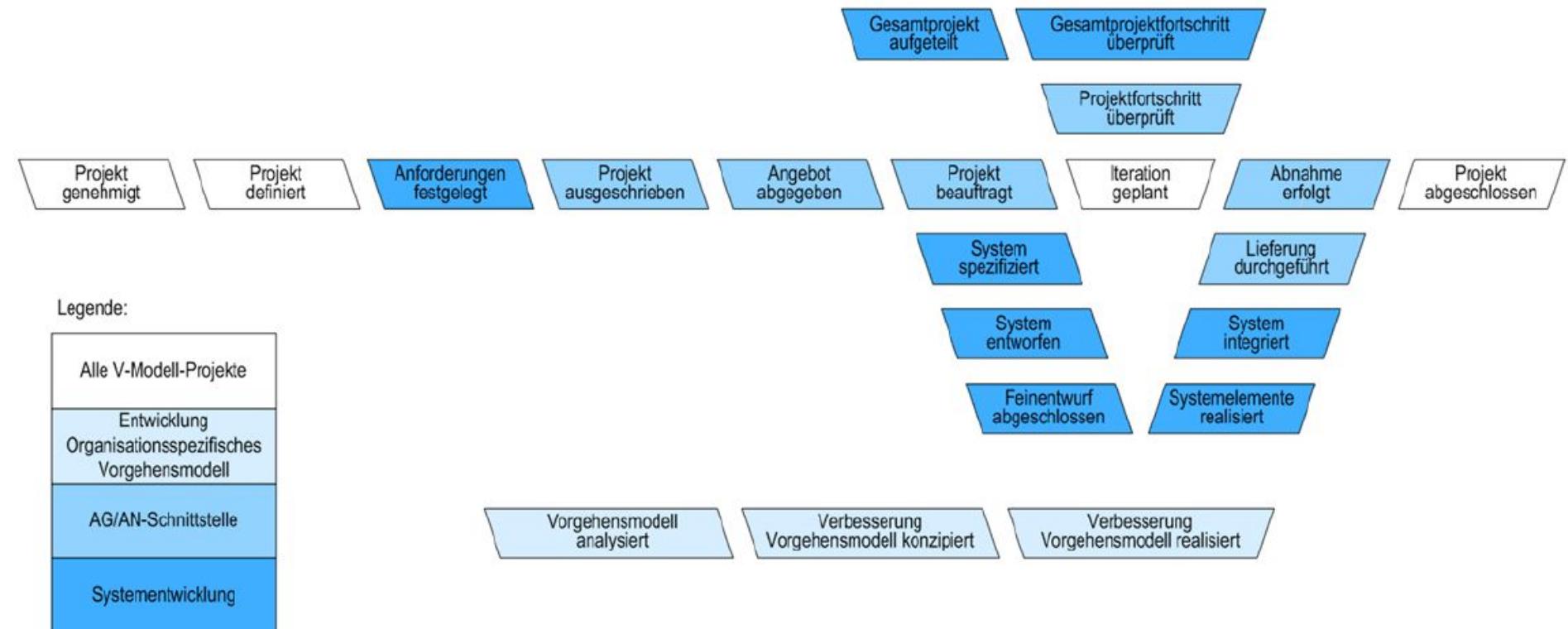
- a) Recherchieren Sie im Internet nach dem V-Modell. Was für Informationen und Meinungen finden Sie? Gibt es Varianten? Ist das V-Modell „modern“?
- b) Wie würden Sie das Modell beurteilen? Diskutieren Sie Vor- und Nachteile des Modells.

- Ursprung: Modell für SW-Projekte bei den Bundesbehörden. (parallele unabhängige Entwicklung ähnlicher Konzepte aber auch in den USA)

Aktueller Standard: V-Modell XT

- Zentrale Aspekte: Rollen, Produkte und Aktivitäten.
- XT = Xtreme Tailoring: Modulare, flexible Vorgehensbausteine zur Anpassung an Projekttypen
- Mehrere Projektdurchführungsstrategien, z.B. inkrementelle, agile und komponentenorientierte Entwicklung
 - (Teil-)Produkte müssen zu Entscheidungspunkten vorliegen (genaue Entwicklungsreihenfolge wird nicht betrachtet)
- Zusätzliche Themen: Migration, Ergonomie, Systemsicherheit, Ausschreibung, Vertragsschluss, Abnahme

V-Modell XT



- Akquisiteur
- Änderungssteuerungsgruppe
(Change Control Board)
- Änderungsverantwortlicher
- Anforderungsanalytiker (AG)
- Anforderungsanalytiker (AN)
- Anwender
- Assessor
- Ausschreibungsverantwortlicher
- Einkäufer
- Ergonomieverantwortlicher
- HW-Architekt
- HW-Entwickler
- KM-Administrator
- KM-Verantwortlicher
- Lenkungsausschuss
- Logistikentwickler
- Logistikverantwortlicher
- Projektkaufmann
- Projektleiter
- Projektmanager
- Prozessingenieur
- Prüfer
- QS-Verantwortlicher
- Qualitätsmanager
- SW-Architekt
- SW-Entwickler
- Systemarchitekt
- Systemintegrator
- Systemsicherheitsbeauftragter
- Technischer Autor

Rolle: Anforderungsanalytiker

Beispiel

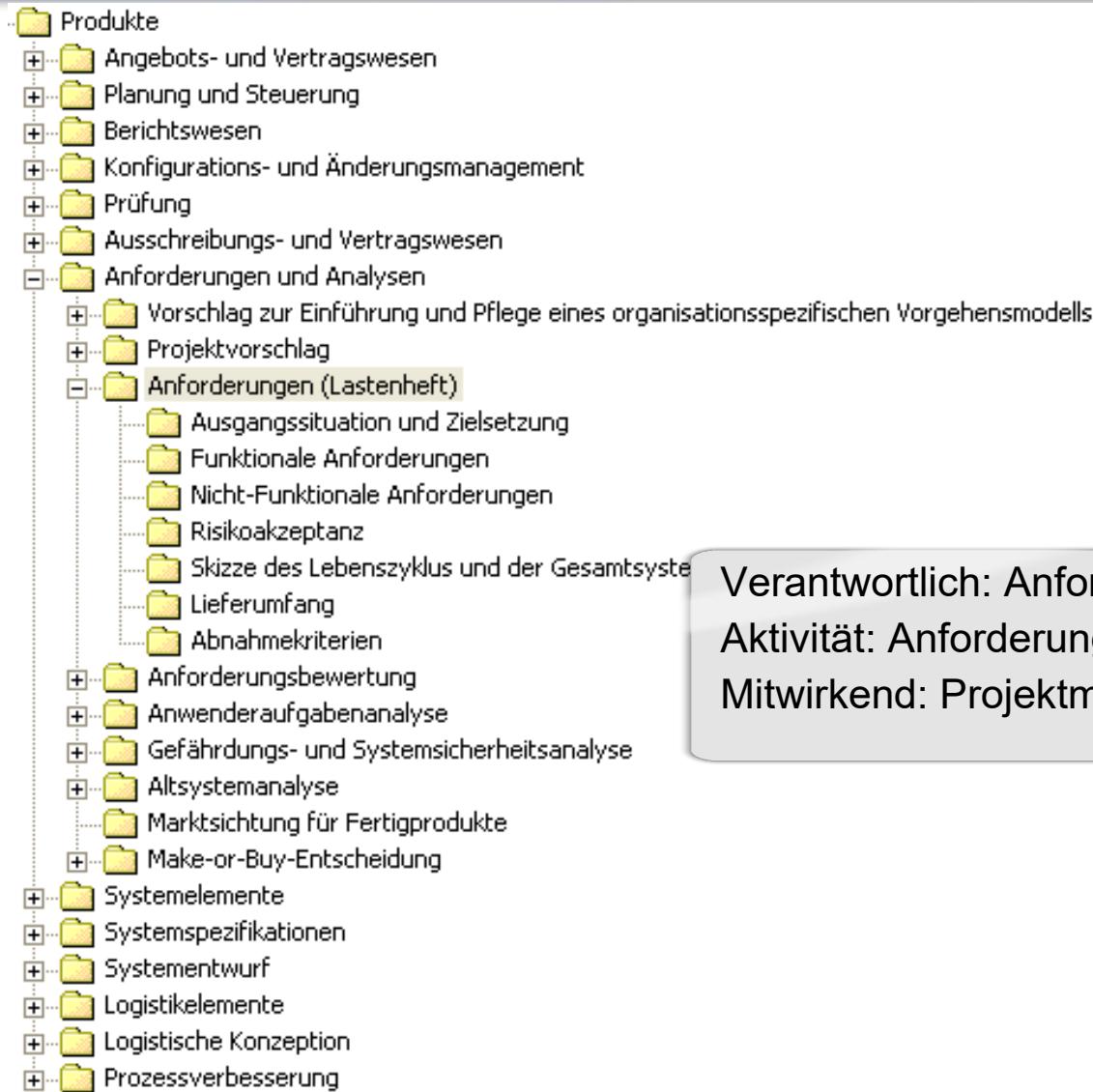
Der **Anforderungsanalytiker** (AG) ist nach erfolgreicher Beauftragung des Projekts für die Erstellung des Produktes Anforderungen (Lastenheft) verantwortlich. Er hat die Qualität der Anwenderanforderungen sicherzustellen und die Voraussetzungen für die Verfolgbarkeit und die Veränderbarkeit der Anforderungen über alle Lebenszyklusabschnitte zu schaffen. Der Anforderungsanalytiker (AG) hat die Grundlagen des Fachgebietes "Requirements Engineering" bei der Aufgabendurchführung zu beachten.

Aufgaben und Befugnisse

- Erarbeiten der Grundlagen für die Erstellung und das Management von Anforderungen,
- Auswahl und Einrichten der Werkzeuge für die Erfassung und Verwaltung der Anforderungen,
- Analyse von Geschäftsprozessen,
- Mitarbeit bei Realisierungsuntersuchungen,
- Analyse von Bedrohung und Risiko,
- Durchführung von Schwachstellenanalyse und Sicherheits- und Leistungsanalyse,
- Erfassen und Beschreiben der funktionalen und nicht-funktionalen Anforderungen,
- Abstimmen und Harmonisieren der erfassten Anforderungen mit allen Beteiligten,
- Systematisieren und Priorisieren der erfassten Anforderungen,
- Erstellen von Abnahmekriterien,
- **Erstellen des Entwurfs eines Anforderungsdokuments,**
- Qualitätssicherndes Überprüfen der Anforderungen nach vorgegebenen Qualitätskriterien,
- Überprüfen des Systementwurfs auf Einhaltung der Anwenderanforderungen,
- Mängelbeseitigung bei Anforderungen,
- Aufbereiten der Anforderungen für das Anforderungscontrolling,
- Bewerten von Anforderungen nach vorgegebenen Kriterien,
- Analyse der operationellen Notwendigkeit und der technischen Machbarkeit von Anforderungen,
- Bewerten der Anforderungen nach deren Wirtschaftlichkeit (Kosten-Nutzen-Analysen),
- Erstellen eines ausschreibungsreifen Anforderungsdokumentes.

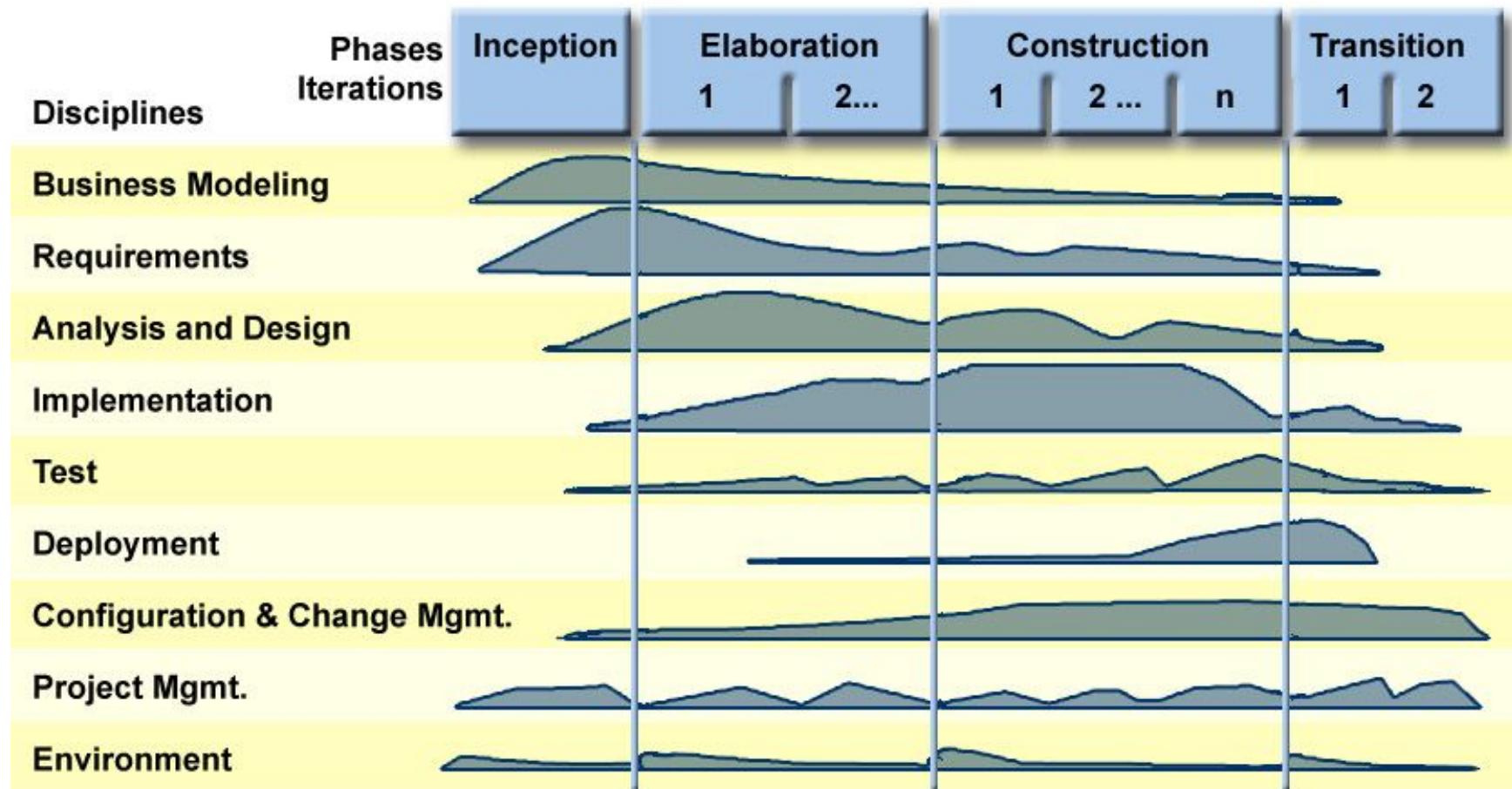
V-Modell XT - Produkte

Beispiel



Verantwortlich: Anforderungsanalytiker (Auftraggeber)
Aktivität: Anforderungen festlegen
Mitwirkend: Projektmanager, Projektleiter, Anwender

RUP Entwicklungsmodell*



* RUP = Rational Unified Process

RUP - Erläuterungen

- Business Modeling: Beschreibung des geschäftlichen Umfeldes, Geschäftsprozesse, Abläufe
- Requirements: Anforderungsanalyse
- Analysis/Design: Semiformale/grafische Beschreibung der Anforderungen (z.B. UML Use Cases), Grobentwurf Systemarchitektur usw. (verschiedene Sichten)
- Implementation/Test
- Deployment: s. Inbetriebnahme
- Configuration & Change Management: Verwaltung der verschiedenen Versionen des Softwarequellcodes oder anderer Komponenten.
- Project Management: Steuerung des Entwicklungsprozesses
- Environment: Bereitstellen der Entwicklungsumgebung und benötigter Ressourcen (Rechner, Werkzeuge, ...)

Übung

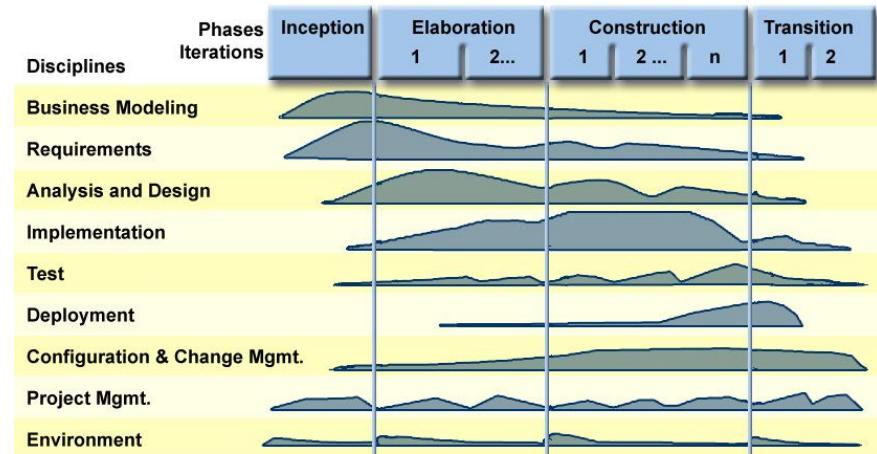
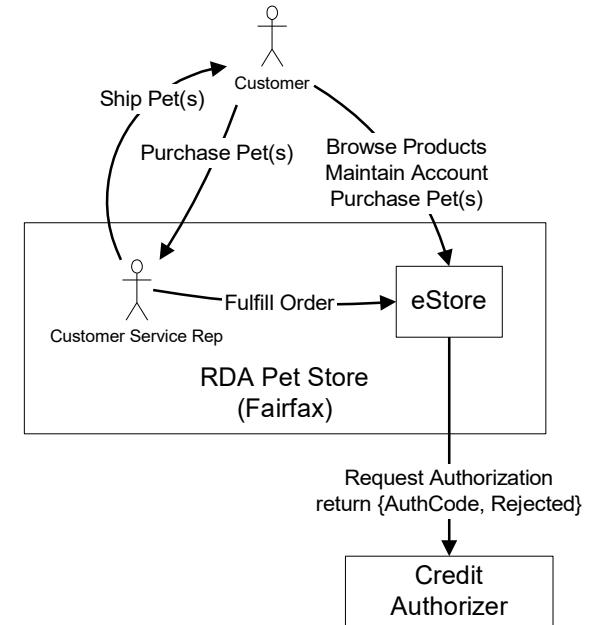
3.8 Rational Unified Process

Untersuchen Sie das RUP-Vorgehensmodell:

- a) Recherchieren Sie im Internet nach RUP. Was für Informationen und Meinungen finden Sie?
- b) Nennen Sie Unterschiede zu den anderen Modellen. Welche zusätzlichen Aspekte gibt es? Was erscheint Ihnen wichtig?
- c) Nennen Sie Vor- und Nachteile des Modells im Vergleich mit den anderen vorgestellten Modellen.

RUP

- Fokus auf Anwendungsfälle (Use Cases) und Architektur
- Objektorientierung und Beschreibung mit UML
- Iteratives, inkrementelles Vorgehen
- Rollen (roles), Produkte (products), Aufgaben (tasks)
- RUP ist ein "Framework", in dessen Rahmen verschiedene Vorgehensweisen eingebettet werden können.
- von Rational (nun bei IBM) entwickelt: IBM bietet Werkzeuge zur Unterstützung an.
- Es gibt Varianten/Ableger, z.B. Open Unified Process



Bekannte (Prozess-)Probleme?

Auszug aus **OpenUP –The Best of Two Worlds**, Bjorn Gustafsson

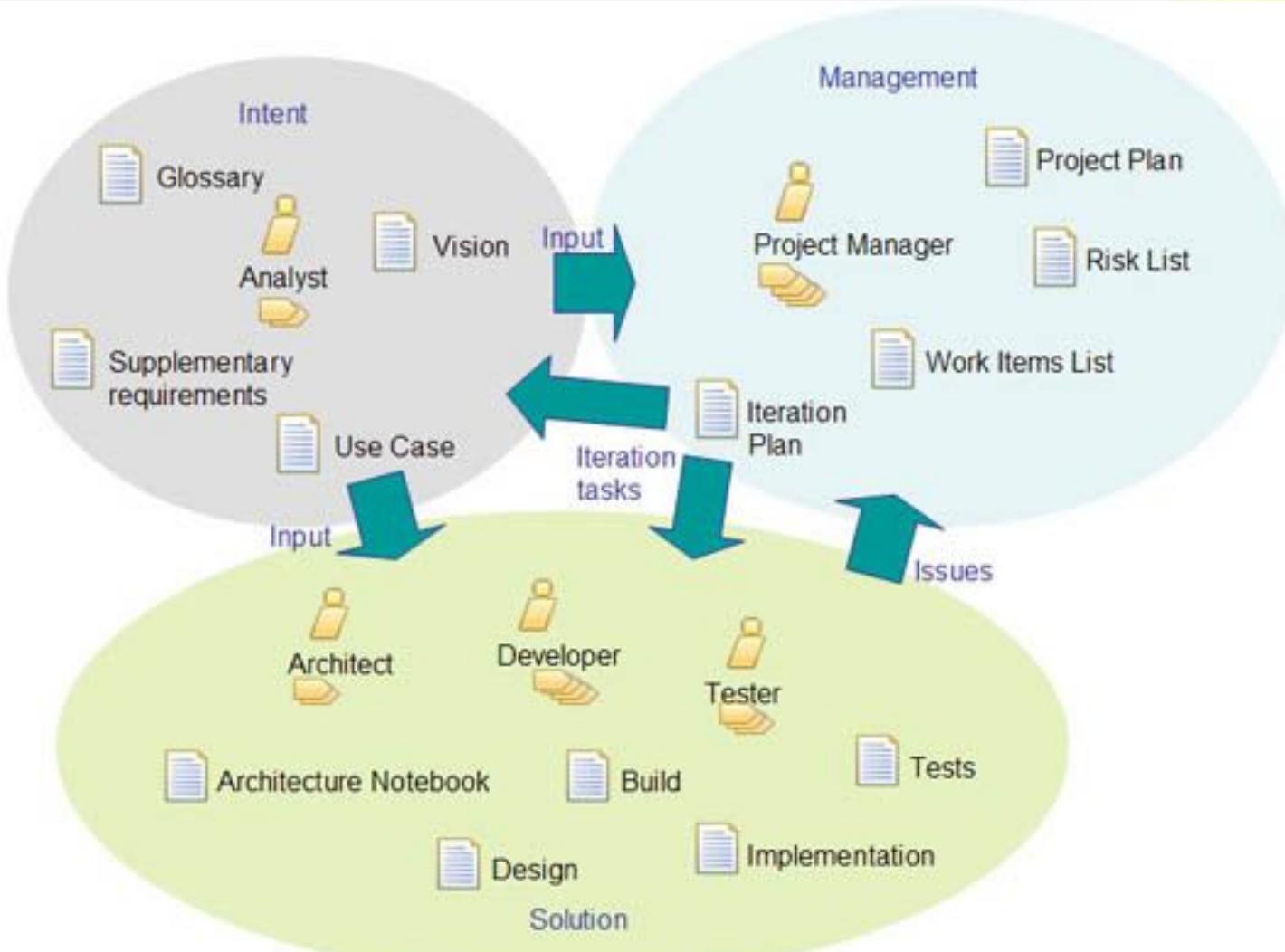
<http://www.methodsandtools.com/archive/archive.php?id=69>

Some common problems can be observed in troubled projects:

- **The project team doesn't share a clear vision of how the system will appear to its users.** Without a clear vision of the final system, there is no guiding framework for the work in the project. The team's analysts have no means to calibrate their requirements to the scope and effort of the project, which results in ill-fitting requirements statements; and the development team can not properly prioritize their work.
- **Requirements do not drive development work.** Some development cultures regard requirements as "incidental" input to the project only, and drive the development work based on other, internal and technical, conditions. This is commonly found in "silo" organizations where there are separate teams for requirements capture and development.
- **The system's architecture has not been articulated.** Although projects that only evolve and maintain existing systems may not need to pay much attention to architecture, there are many projects that do. As Dean Leffingwell points out: "... how much architecture a team needs depends in large part on what the team is building".
- **Plans are not connected to the engineering reality.** Plans are often created and maintained separately from the actual project work. We have all seen nebulous Gantt charts that project managers spent days or even weeks creating, with hundreds of line items in nifty breakdown structures, purportedly believed to bring the project to "completion" at some well-defined point in time. Of course, these plans become outdated even before they are pinned on the wall.
- **Risks are ignored.** All projects run the risk of building the wrong product or not being able to build the product as envisioned, yet very few projects acknowledge this uncertainty and actively try to reduce it.

OpenUP Elements

Beispiel



aus: <http://www.methodsandtools.com/archive/archive.php?id=69p3>

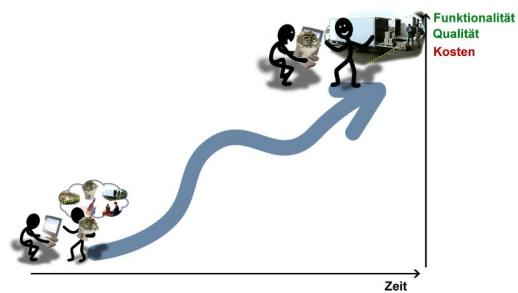
Agile Methoden (agile methods) sind ein Sammelbegriff für verschiedene Vorgehensmodelle, die die folgenden Punkte betonen:

- Teamarbeit (überschaubare Teams), enge Zusammenarbeit auch mit dem Kunden
- Kommunikation wichtiger als Dokumentation
- Inkrementelle Entwicklung: Viele kleine Entwicklungsschritte mit wenig Planung
- Iteration (Wiederholung): In jedem Schritt wird von Anforderungsanalyse bis Testen alles im "Miniatuurformat" durchlaufen. Am Ende steht immer ein ausführbares Programm (wenn auch anfangs mit minimaler Funktionalität)
- Leichtgewichtige Prozessmodelle, minimale langfristige Planung, Flexibilität.

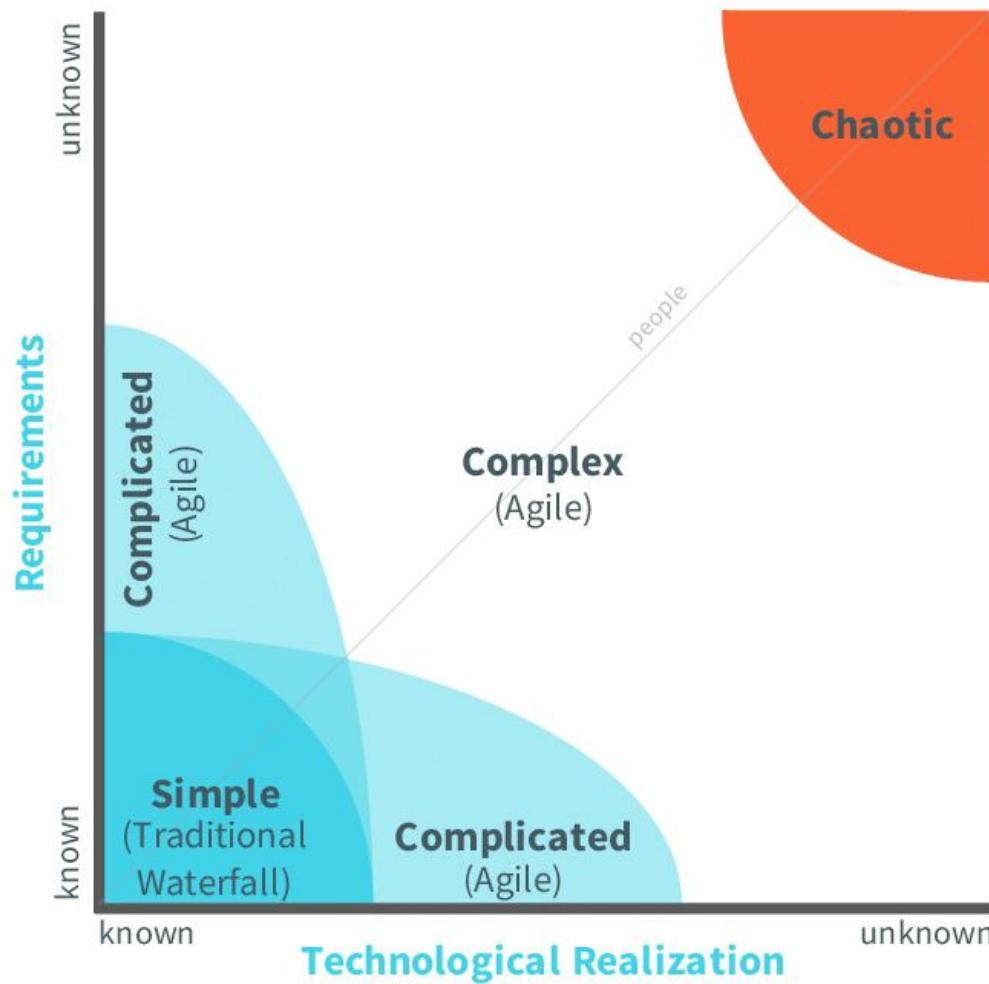
Beispiele:

- Scrum
- XP - Extreme Programming
- FDD – Feature Driven Development
- RAD - Rapid application development
- Crystal Clear Software Development
- DSDM - Dynamic Systems Development Method

→ Agile Methoden waren/sind auch eine Reaktion auf übertrieben bürokratische, ineffiziente Entwicklungsprozesse mit Übermaß von Dokumentation, die niemand liest.



Stacey Matrix für Software-Entwicklung



Bildquelle: Ömer Uludag, TU München

Übung

3.9 Agile Methoden

Recherchen Sie im Internet nach agilen Vorgehensmodellen.

- a) Welche Methoden finden Sie (Namen der Methoden)? Welche scheinen verbreitet, welche eher selten?
- b) Worin unterscheiden sich die Ansätze?
- c) Welche Eigenschaften/Mechanismen werden als positiv herausgestellt? Was wird kritisch diskutiert?

Agile Manifesto

2001: Treffen von Schlüsselpersonen der agilen Softwareentwicklung:
<http://agilemanifesto.org/>

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.

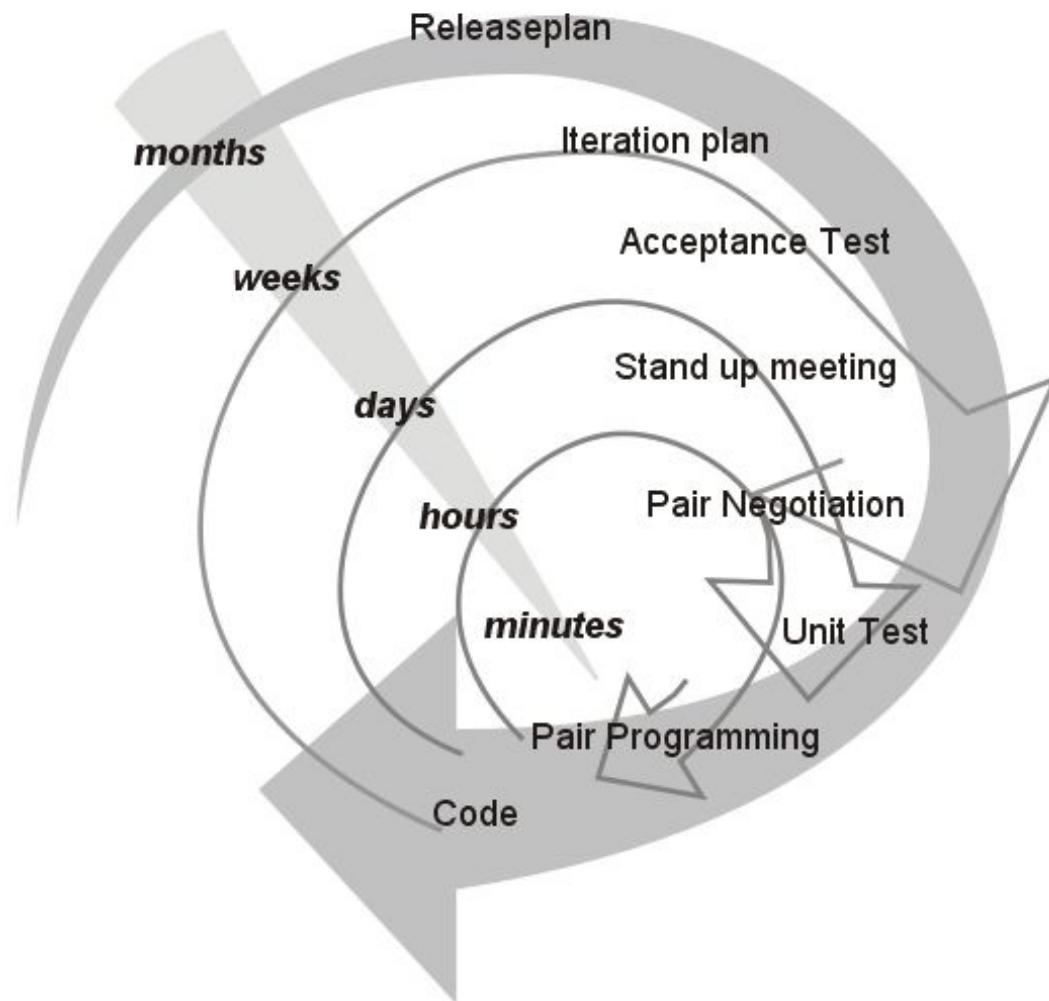
Agile Manifesto - Prinzipien

2001: Treffen von Schlüsselpersonen der agilen Softwareentwicklung:
<http://agilemanifesto.org/>

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity - the art of maximizing the amount of work not done - is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

XP - Extreme Programming

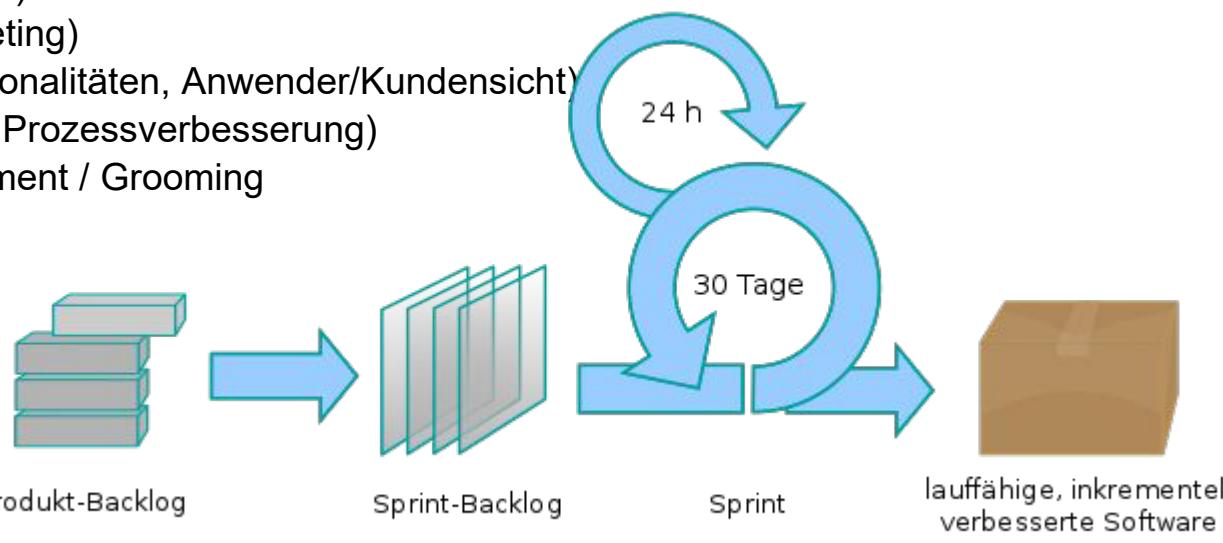
Beispiel



http://en.wikipedia.org/wiki/Extreme_Programming

Scrum

- Agiles Vorgehensmodell, weit verbreitet
- Inkrementell: „Sprints“ (30 Tage)
- Artefakte:
 - Product Backlog (Anforderungsliste)
 - Sprint Backlog (Plan für Sprint)
 - Product Increment (Sprintergebnisse)
- Aktivitäten:
 - Sprint Planning (was+wie)
 - Daily Scrum (15min-Meeting)
 - Sprint Review (→ Funktionalitäten, Anwender/Kundensicht)
 - Sprint Retrospective (→ Prozessverbesserung)
 - Product Backlog Refinement / Grooming
- Rollen
 - Product Owner (Anforderungen, Kundensicht)
 - Entwicklungsteam
 - Scrum Master („Prozessmoderator“)
- Teamgröße<10, Werkverträge...?



Bildquelle: https://de.wikipedia.org/wiki/Scrum#/media/File:Scrum_process-de.svg

Übung

3.10 Vorsorgen für später?

Im agilen Vorgehen wird propagiert, dass man nur die tatsächlich angeforderten Funktionen entwirft und implementiert, anstatt zu versuchen, Mechanismen vorzusehen, die eventuelle zukünftige Erweiterungen erleichtern sollen.

- a) Diskutieren Sie Vor- und Nachteile dieses Ansatzes.
- b) Was ist Ihr Fazit, Ihre Empfehlung?

- Programmierer versuchen nicht, bei der Realisierung des aktuellen Releases bereits künftige Releases mit zu berücksichtigen
- Lässt sich meist eh nicht genau vorhersehen
- Programmierer sind zum ständigen Umbau der bereits erstellten Software bereit
- Refactorings + Redesign
- Problem: Lokale Optimierung vs. optimale Systemarchitektur
- Rapid-Prototyping-Vorgehensweise

Wahl eines Vorgehensmodells

Entscheidungskriterien:

- Welches Modell wurde (im Unternehmen, vom Team) bisher verwendet? Wie erfahren ist das Team damit? Hat sich das Modell bewährt?
- Wie groß ist das Projekt? Wie hoch sind die Projektrisiken? Wie hoch sind die Qualitätsansprüche?
 - sicherheitskritische Lösungen erfordern z.B. zwingend einen starren, sehr genauen dokumentierten Prozess (→ Zertifizierung).
- Gibt es äußere Randbedingungen, unbedingt zu verwendende Standards?
 - Unternehmensrichtlinien, z.B. Unternehmen strebt CMMI-Zertifizierung an.
- Wie genau sind die Anforderungen am Anfang bekannt? Kann es noch größere Änderungen im Laufe des Projektes geben?
 - Extrembeispiel: Forschungsprojekte → Projektergebnis ist am Anfang vage
- Grundsatzentscheidung: Agil oder Traditionell?
- Was ist der richtige Umfang/Aufwand für den Prozess (welche Dokumente usw. sind erforderlich?)
- Sind Abweichungen vom Modell erlaubt? Vereinfachungen (z.B. bestimmte Phasen weglassen)?

Egal, welches Modell letztlich gewählt wird, wichtig ist:
Es gibt ein definiertes, geplantes Vorgehen und das Team versteht es und befolgt es!

Software-Engineering-Projekt



P.9 Lieferant: Vorgehensmodell

Sie haben den Auftrag erhalten und wollen nun entscheiden, welches Vorgehensmodell die Basis Ihrer Projektplanung werden soll.

- Welches Vorgehensmodell erscheint Ihnen grundsätzlich geeignet? Warum? (Vorteile/Nachteile?)
- Welche Teile des Vorgehensmodells wollen Sie verwenden, wo nehmen Sie Vereinfachungen oder Änderungen vor? Welche Bausteine des Modells sind für Sie wichtig, welche können Sie weglassen?
- Der Qualitätsmanager Ihres Unternehmens, der auch für die Definition der internen Prozesse verantwortlich ist, möchte die Vorgehensweise in ihrem Projekt begutachten. Grundsätzlich hat er auch Bereitschaft signalisiert, für dieses Projekt von den Standardprozessen abzuweichen und etwas Neues auszuprobieren. Bereiten Sie ein kleine Präsentation (zwei, drei Folien) vor, um Ihren Qualitätsmanager von Ihrer Vorgehensweise zu überzeugen.

Software Engineering I

4. Planung und Projektmanagement

Prof. Dr. Eckhard Kruse

DHBW Mannheim

Warum planen?

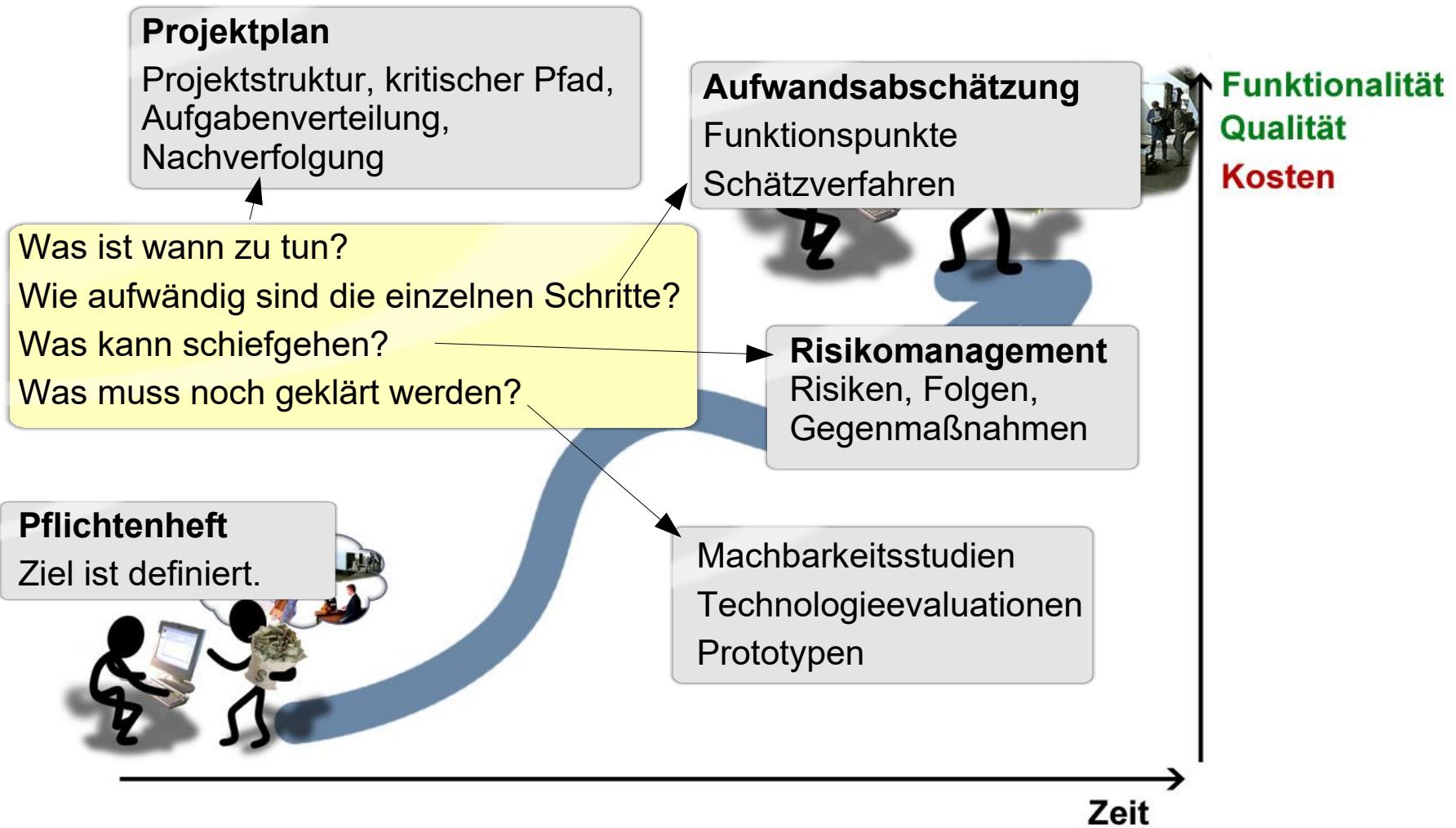
*"The nice thing about **not planning** is that failure comes as a complete surprise rather than being preceded by a period of worry and depression."*

Überzeugt? Oder gibt es Gegenargumente?

Planung

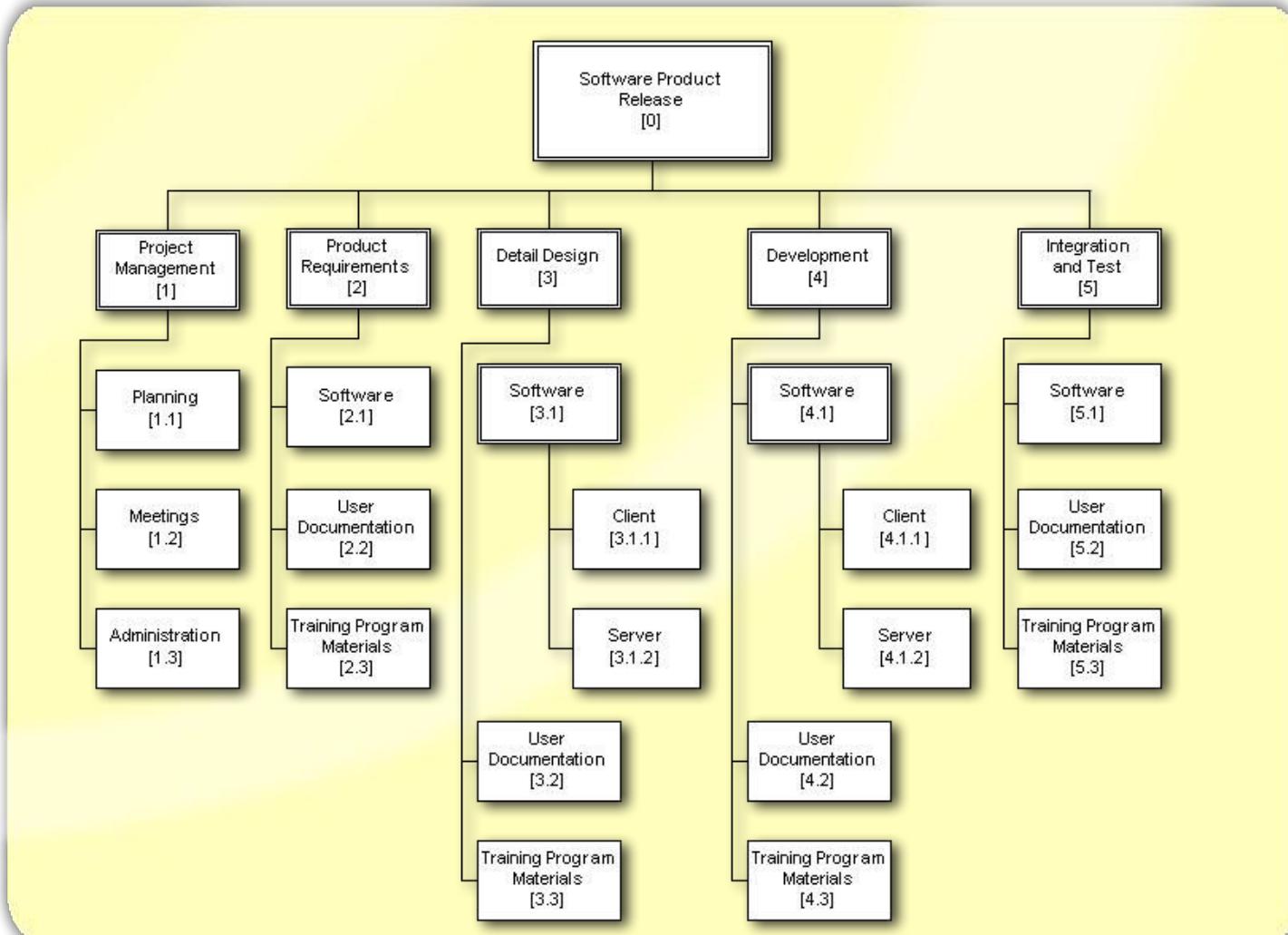


Planung



Projektstrukturplan (grob)

Beispiel



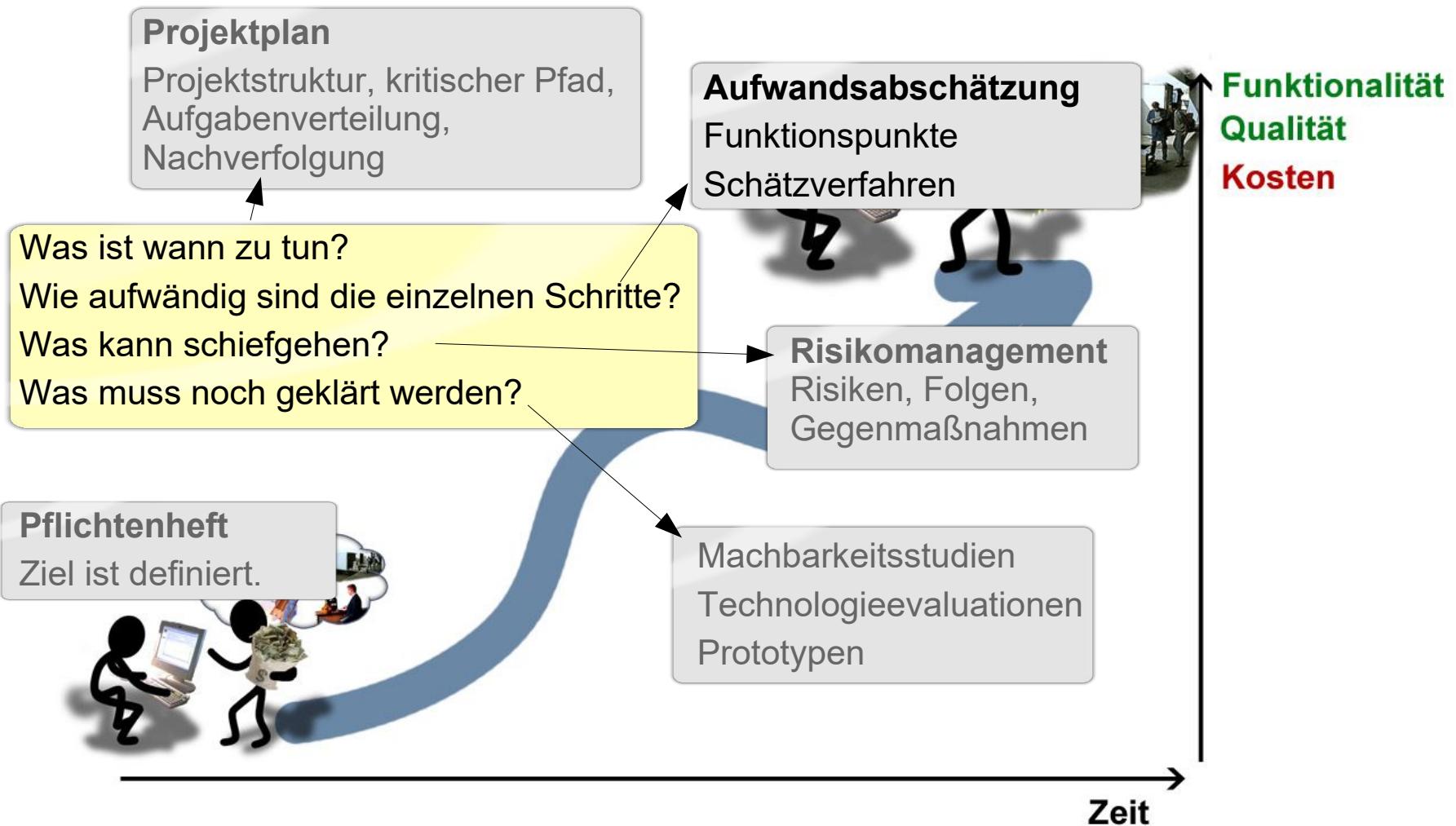
Projektplanung

Der **Projektstrukturplan (PSP)** (*work breakdown structure, WBS*) gliedert das Projekt in überschaubare, gut planbare Arbeitspakete und beschreibt deren Abhängigkeitsbeziehungen. Er ist die Basis für die Termin- und Kostenplanung.

- Definition der Arbeitspakete: Fokus liegt auf den Ergebnissen (deliverables), nicht so sehr auf den detaillierten einzelnen Arbeitsschritten.
- Ablaufplan/Terminplan: Vorgänge (Zeiträume), Meilensteine (Zeitpunkte) und Abhängigkeiten.
- Kritischer Pfad: Sequenz der Vorgänge von Projektstart zu Projektende, bei denen es keinen Zeitpuffer gibt. D.h. eine Verzögerung eines kritischen Projektvorganges führt direkt zu einem späteren Projektende.

- Projektplan: Template/Beispiel aus industrieller Softwareentwicklung
- Nachverfolgung: Gate Model

Aufwandsabschätzung



Aufwandsschätzung

Die Kosten von Softwareentwicklungsprojekten werden zu einem Großteil durch den personellen Aufwand bestimmt.

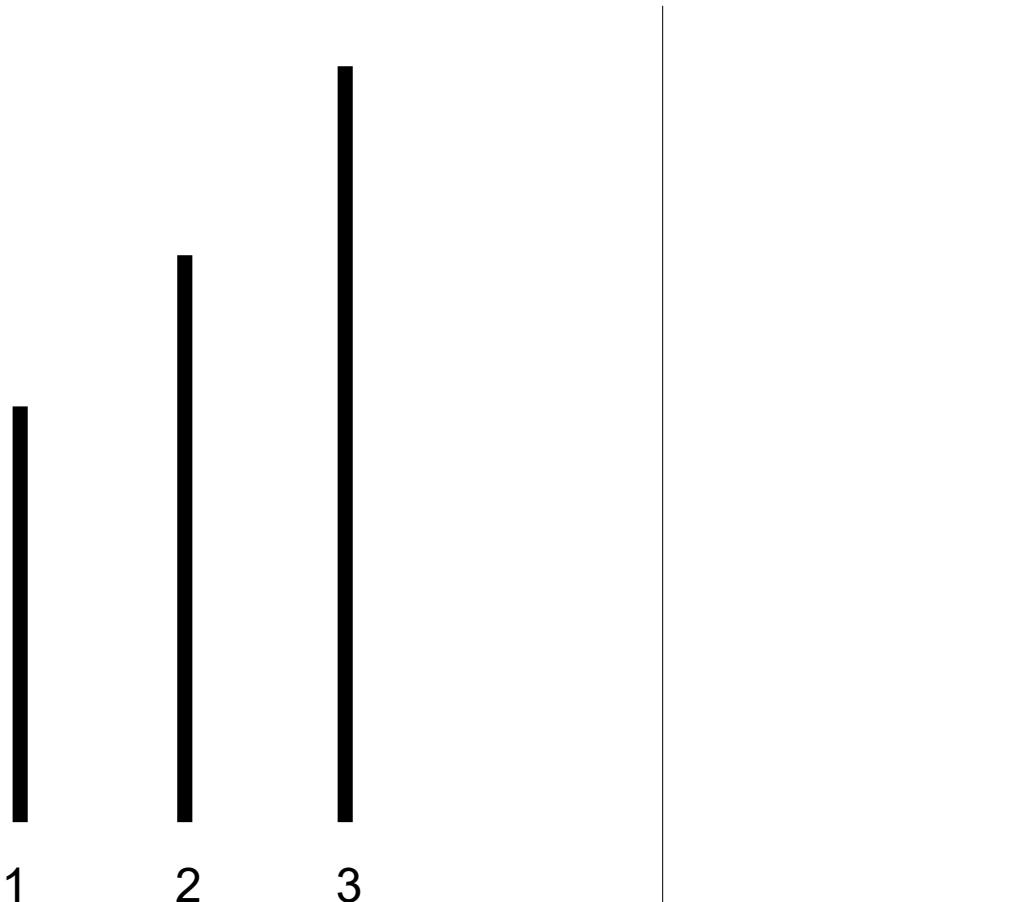
- Übliches Maß: Personentage, -wochen, -monate, -jahre.
- Personenjahre berücksichtigen Urlaub und Fehlzeiten. D.h. 1 Personenjahr ist z.B. 220 Personentage, 1650 Personenstunden (je nach Wochenarbeitszeit).

Wieviele Personentage müssen für die einzelnen Arbeitspakete geplant werden?

Vorgehen:

- Schrittweise, hierarchische Aufteilung der Arbeitspakete in Einzelaktivitäten z.B. in der Größenordnung Manntage bis max. Mannwochen
- Aufwandsschätzung für jede Einzelaktivität, z.B. mit erfahrenen Entwicklern aus dem Team
 - bester Fall, schlechtester Fall, wahrscheinlichster Fall
 - Risiken, Unsicherheiten
 - Sicherheitsmarge (da Schätzungen meist zu optimistisch)
- Aufaddieren, Durchrechnen, Zeitplanung über gesamtes Projekt, Meilensteine
- Besonderes Augenmerk auf kritischen Pfad

Schätzen Sie mal...



Solomon Asch, (1956). *Studies of independence and conformity: A minority of one against a unanimous majority.* Psychological Monographs, 70.

Aufwandsschätzung

Maße für Funktionalitätsumfang:

- Funktionspunkte (function points) = Definition "atomarer" Einzelfunktionen
- Anzahl Klassen / Komponenten, grobe Größe
- Anzahl Codezeilen (Lines of Code)

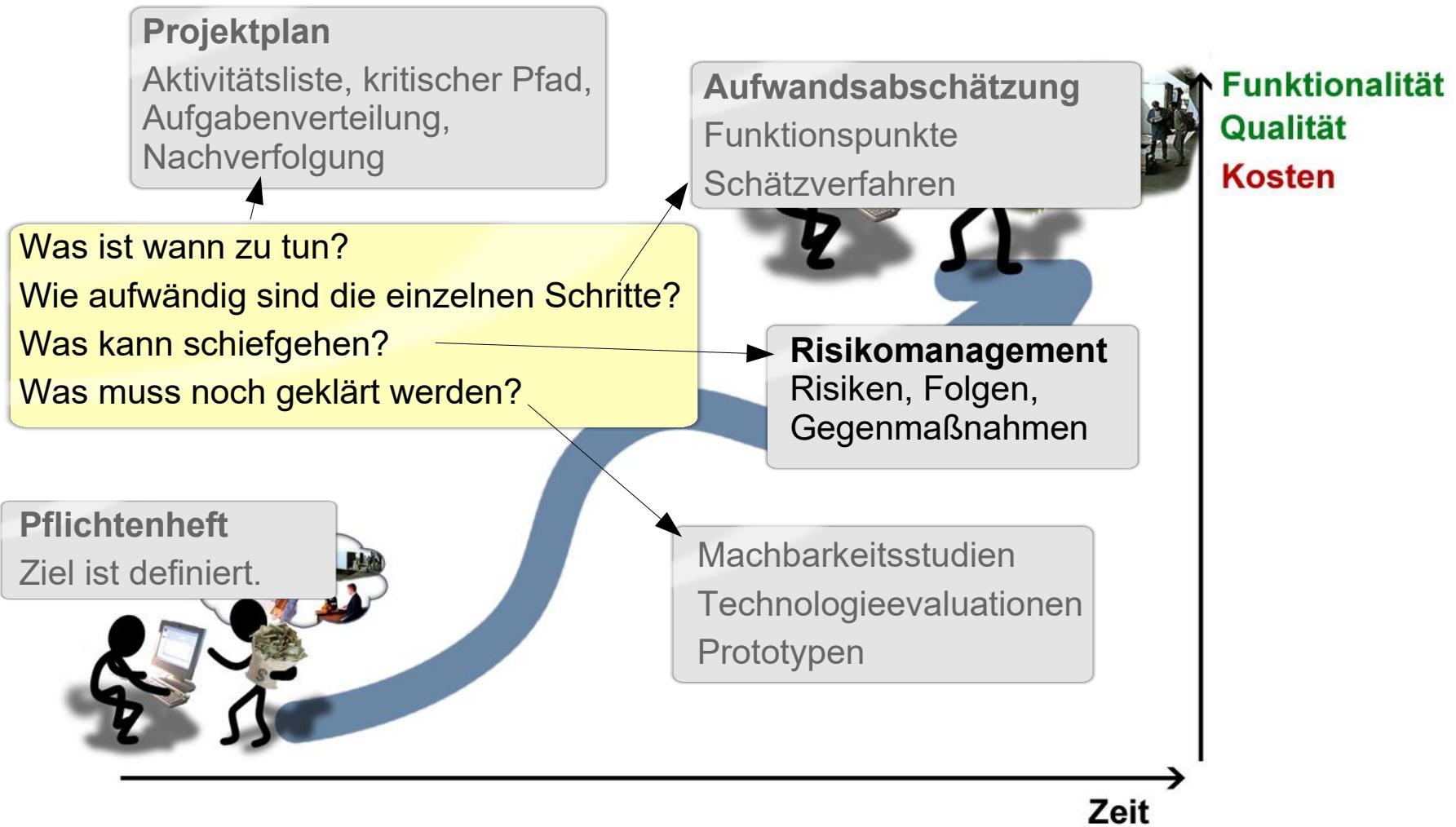
Vorsicht: Bei rein quantitativen Angaben erfolgt keine Unterscheidung zwischen einfachen und schwierigen Programmieraufgaben!

Aufwandsschätzung: Erfahrung ist entscheidend

- Vergleich mit früheren Projekten, ähnlichen Aufgabenstellungen
- Gefühl, was einfach und was aufwändig ist.
- Gefühl für Risiken, technische Probleme usw.

Dokumentierte Aufwandsschätzungen und tatsächlicher Umsetzungsaufwand können wertvolles Material für zukünftige Projekte liefern.

Risikomanagement



Risikomanagement

Risikomanagement ist der Teil des Projektmanagements, der sich mit der Identifizierung, Analyse und Beherrschung von Risiken für die geplante Projektabwicklung beschäftigt.

Vorgehen:

- Projektrisiken identifizieren und auflisten
- Mögliche Auswirkungen (auf den Projekterfolg) analysieren
- Ggf. bereits vorbeugende Maßnahmen initiieren
- Gegenmaßnahmen überlegen, falls der Risikofall eintritt
- Risikomanagement als regelmäßigen Prozessschritt durchführen, da sich Projektrisiken verändern können.

Lieferant: Projektplanung



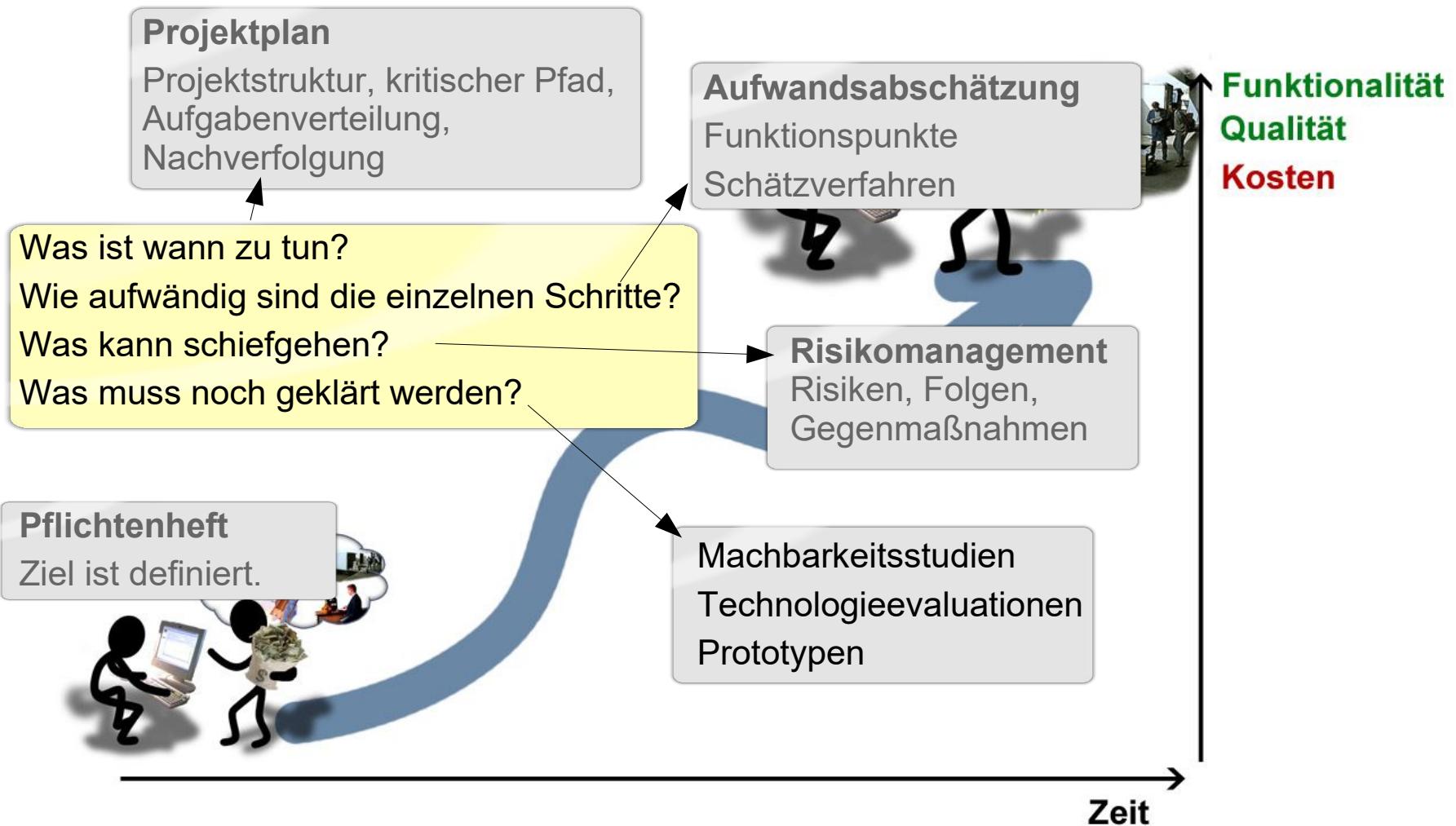
Software-Engineering-Projekt

P.10 Lieferant: Projektplanung

Auf Basis des ausgewählten Vorgehensmodells wollen Sie Ihr Projekt sorgfältig strukturieren und planen:

- a) Was sind die wesentlichen (Zwischen-)Ergebnisse/Produkte Ihres Projektes? Wann sollten diese vorliegen?
- b) Welche Aktivitäten müssen stattfinden? Welche Abhängigkeiten gibt es? (-> kritischer Pfad)
- c) Welche Rollen gibt es im Projekt, wer übernimmt welche Aufgaben?
- d) Welche Projektrisiken gibt es? Wie gehen Sie damit um?
- e) Wann und wie wollen Sie kontrollieren, ob Sie noch im Plan liegen bzw. ob Korrekturmaßnahmen erforderlich werden?
- f) Optional: Bereiten Sie eine kurze Präsentation vor, um Ihr Management zu überzeugen, dass Ihr Projekt solide geplant ist und ein Erfolg wird.

Machbarkeitsstudien



Machbarkeitsstudie

Um Fehlinvestitionen zu verhindern und Risiken zu minimieren, kann bei Zweifeln an dem Erfolg eines geplanten Projektes eine **Machbarkeitsstudie** vorangestellt werden.

Eine Machbarkeitsstudie soll Fragen zum geplanten Projekt klären, z.B. in Bezug auf:

- Wirtschaftlichkeit
- Technische Machbarkeit
- Marktsituation, Unternehmens- und Produktstrategie
- Patentrechtliche Fragen
- Aufwandsabschätzung
- Make or Buy Entscheidungen
- Grundlage für Angebotserstellungen

Feasibility Study Template

Beispiel

1 General Information

1.1 Project Administration

1.2 Project Goal

1.3 Project Result

2 Business

2.1 Customer issue

2.2 Solution

2.3 Benefits

 2.3.1 End-Customer Benefit

 2.3.2 Company Benefit

2.4 Targeted Customers

2.5 Market

2.6 Business Impact

2.7 Relation to Product Portfolio

2.8 Competition Evaluation

2.9 Intellectual Property

 2.9.1 Patent Search: Results of Interest

 2.9.2 Potential New Intellectual Property

3 Technology

3.1 User Operations

3.2 Implementation Approach

3.3 Project Main Requirements

3.4 Other Alternatives

4 Project Management

4.1 Target Dates

4.2 Estimated Cost

4.3 Project Organization

4.4 End Customer contacts

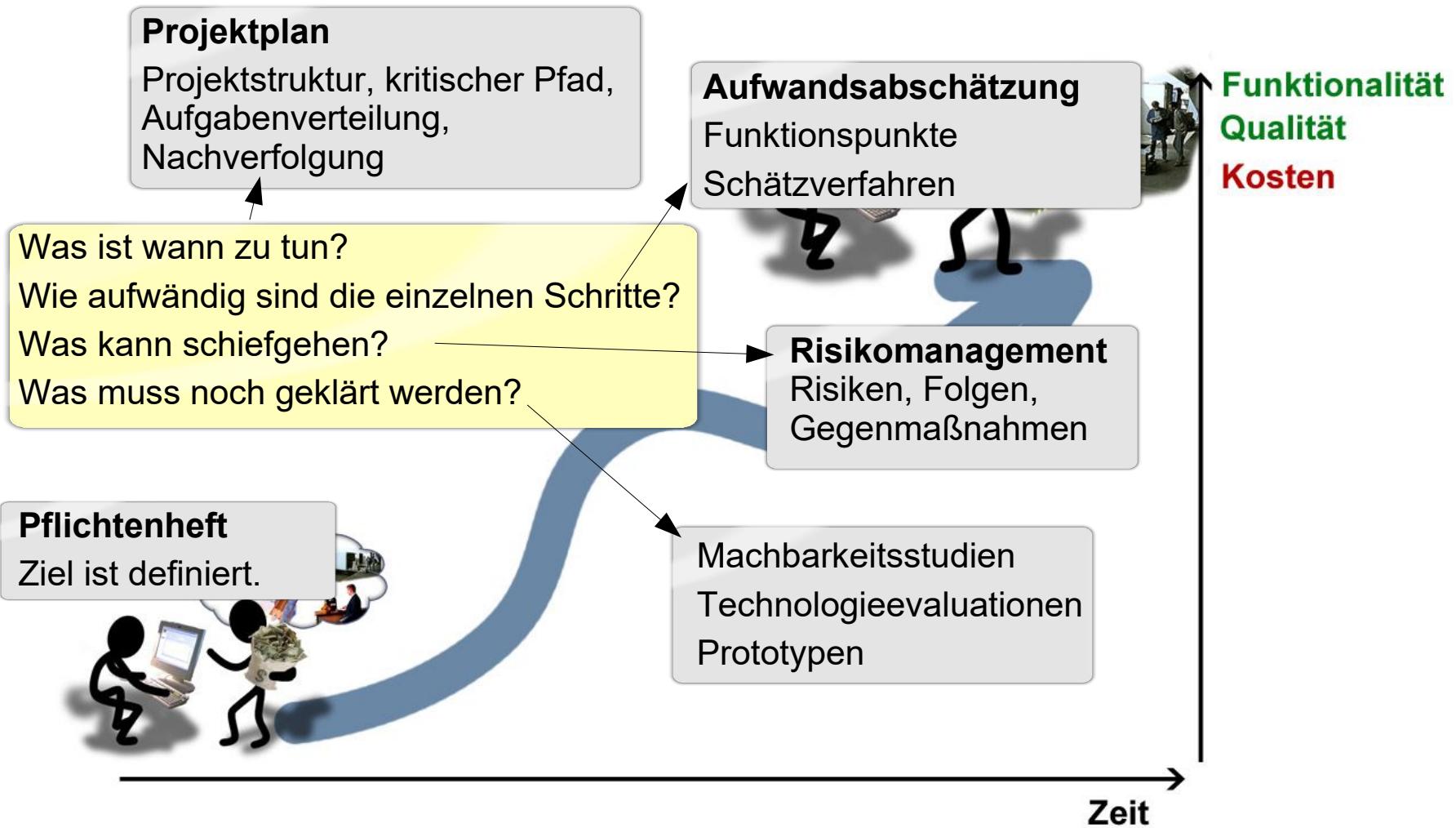
5 Risks

6 References

Change History

Review History

Planung



Prototyp

Ein **Prototyp** ist die rudimentäre, ansatzweise Implementierung (von Teilen) des zu entwickelnden Softwareprodukts. Im Vordergrund bei der Prototypentwicklung steht, möglichst schnell einen ersten Eindruck des Produktes zu erhalten - ohne Anspruch auf Vollständigkeit, hohe Software-Qualität oder gute Softwarearchitektur.

Einsatzbereiche eines Prototyps (insbesondere in den frühen Projektphasen):

- Demonstration (z.B. der Benutzerschnittstelle) bei den Interessengruppen des Projektes
- Ausprobieren neuer Ideen – z.B. im Rahmen der Anforderungsanalyse
- Aufwandsabschätzung, wie komplex bestimmte Entwicklungsaufgaben werden können
- Evaluation verschiedener technischer Lösungsansätze

Die eigentliche Produktentwicklung sollte (in aller Regel) nicht auf Basis des Prototyp-Codes erfolgen, sondern mit einem neuen, sauberen Entwurf!

Prototypen - Begriffe

- Ein **Demonstrationsprototyp** vermittelt einen ersten, sehr groben Eindruck von dem Produkt.
- Ein **horizontaler Prototyp** betrachtet eine komplette Systemschicht (z.B. die Benutzeroberfläche) ohne die abhängige Funktionalität in anderen Systemschichten (z.B. ohne Business Logic).
- Ein **vertikaler Prototyp** realisiert eine ausgewählte Funktion des Systems durch alle Schichten (z.B. Benutzeranmeldung: Betrifft UI, Administration, Datenhaltung, IT-Sicherheit usw.)
- Ein **explorativer Prototyp** ist eine Spielwiese, um neue Ideen auszuprobieren und in der Anforderungsanalyse offene Punkte zu untersuchen.
- Ein **experimenteller Prototyp** dient zur Untersuchung technischer Realierungsmöglichkeiten und soll den Nachweis erbringen, dass konzipierte Lösungsvorschläge in der Praxis auch anwendbar sind.
- Der **evolutionäre Prototyp** ist im Gegensatz zu den anderen Prototypen kein Wegwerfprodukt, sondern soll stufenweise bis zum fertigen Produkt verfeinert und weiterentwickelt werden.

Mock-up

Ein **Mock-up** ist ein Vorführmuster, um das Aussehen des zu entwickelnden Produktes zu demonstrieren (allerdings ohne Funktionalität).

- Einsatzbereich insbesondere in der Anforderungsanalyse, um Kunden/Anwender-Feedback zu erhalten.
- Mock-ups von Softwareprodukten stellen die Benutzerschnittstelle dar (= den sichtbaren Teil der Software) und können z.B. einfach mit Zeichenprogrammen o.ä. realisiert werden.
- Ein Mock-up ist kein Prototyp (denn ein Prototyp umfasst immer auch tatsächlich implementierte Funktionalität).



Software-Engineering-Projekt

P.11 Lieferant: Use Cases und Mock-up

Um sicherzugehen, dass Sie die Anforderungen Ihres Kunden richtig erfasst haben, bereiten Sie Use Cases (Anwendungsfälle) und ein Mock-up („Vorführmodell“) vor:

- Charakterisieren Sie typische Anwendungsszenarien des Systems.
- Entwerfen Sie das Ausehen der Benutzeroberfläche, z.B. mit einem Malprogramm. Was sind wesentliche grafische Elemente, welches sind die Bedienelemente, Buttons usw.?

Kunde-Lieferant: Präsentation Mock-up



Software-Engineering-Projekt



P.12 Kunde-Lieferant: Präsentation Mock-up

Der Lieferant stellt dem Kunden seine Entwürfe vor. Ist das Projekt auf dem richtigen Weg oder gibt es bei den Anforderungen noch Missverständnisse, die geklärt werden müssen?

Software Engineering I

5. Entwurf

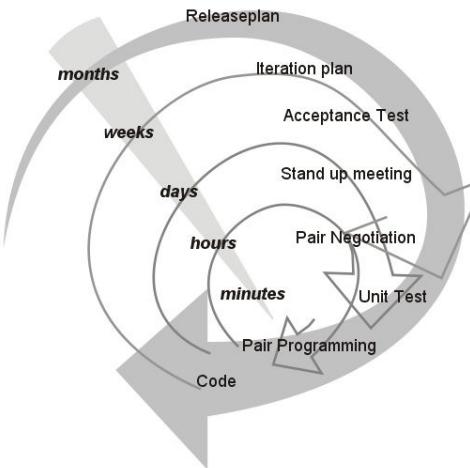
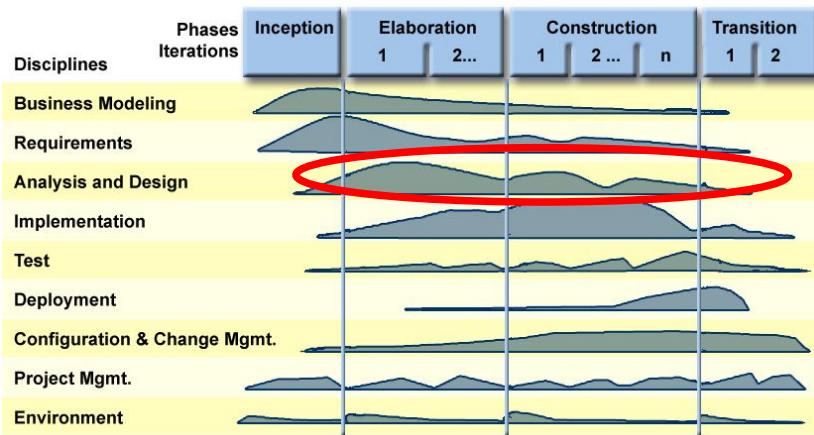
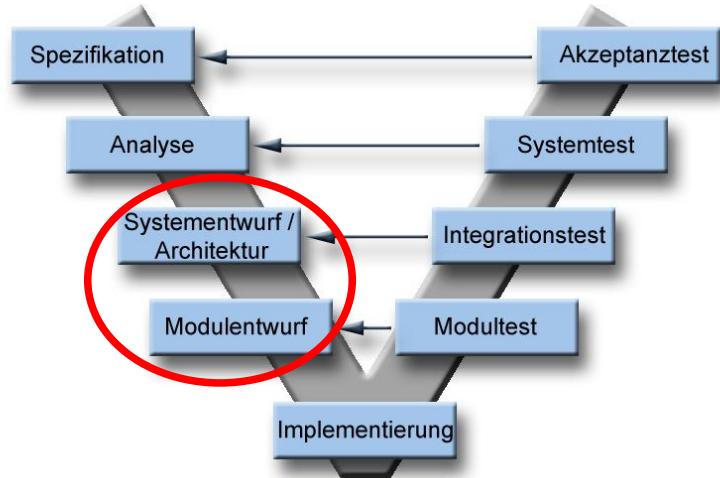
Prof. Dr. Eckhard Kruse

DHBW Mannheim

Der **Entwurf** (engl. **design**) dient dazu, auf Basis der Anforderungen schrittweise eine entsprechende technische Realisierung zu entwerfen. Hierzu wird das System schrittweise vom Groben ins Feine in Komponenten und Verantwortlichkeiten zerlegt.

- Der Entwurf erfolgt auf verschiedenen Ebenen (je komplexer das System, desto mehr Ebenen).
- Typische Begriffe sind Systementwurf (=Gesamtsystem), Architektur (Gesamtsystem oder Teilsysteme / Komponenten), Design (auf Komponenten- / Modulebene). (Die Grenzen der Begriffe sind fließend.)
- Der Fokus liegt auf der Aufteilung in Komponenten/Module, Verantwortlichkeiten und Schnittstellen untereinander – nicht auf deren internen Realisierung.
- **Ergebnisse:** Entwurfsdokumente, Diagramme (z.B. UML)

Enwurf in den Vorgehensmodellen



Entwurf – ein weiter Begriff

Beispiel

Entwurf
eines Sortieralgorithmus

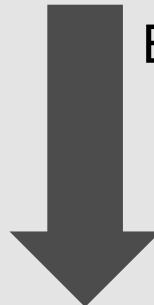


Entwurf
eines komplexen
Softwaresystems für die
industrielle Automatisierung

Was ist gleich?
Was ist verschieden?

Softwaresysteme sind hierarchisch aufgebaut:

- (Produktlinie)
- Produkt/System
- Subsystem
- Komponente
- Modul
- Klasse



Entwurf

Wo wäre hier der Entwurf
eines Algorithmus
einzzuordnen?

Beim **Systementwurf (engl. system design)** wird die Software-Architektur, d.h. der Bauplan der Software, entwickelt. Das Gesamtsystem wird auf einer groben Ebene in Schichten/Teilsysteme/Komponenten zerlegt, die verschiedene Aufgaben übernehmen (z.B. Datenhaltung, Benutzerschnittstelle, Geschäftslogik).

Wichtige Prinzipien:

- Dokumentation der Architekturentscheidungen: Was sind die Ziele und wie sollen sie erreicht werden (und warum nicht anders)
 - "Separation of concerns": Aufteilung verschiedener Funktionen in verschiedene Softwarekomponenten.
 - Definition von Schnittstellen (auf grober Ebene): Was/wie kommunizieren die unabhängigen Komponenten?
 - Grundlegende technische Fragen: Betriebssystem, 3rd party-Komponenten, Frameworks
 - Unterscheidung verschiedener Sichten, z.B. Komponenten, Deployment, Informationsmodell, Data+Control flow / collaboration
- Ergebnis: Architekturendokument

Aufteilung in Komponenten

Eine Komponente...

- dient einem klar umrissenen Zweck: “separation of concerns”
- fasst eng zusammengehörige Funktionalität zusammen: “high cohesion”
- verbirgt interne Implementierungsdetails: “information hiding”
- ist nur lose, über wenige Schnittstellen mit anderen Teilsystemen gekoppelt: “low coupling”
- besteht aus öffentlicher Schnittstelle und Rumpf (ihre Realisierung)
- sollte sich als eigenständige Einheit entwickeln, übersetzen und testen lassen (ggf. mit hilfsweisen Dummy-Implementierungen für benötigte externe Komponenten)
- ist typischerweise in der Verantwortung einer Person/eines Teilteams.

Beim Entwurf unterscheidet man verschiedene **Sichten (Views)** auf das System, die verschiedene Aspekte der Architektur in den Mittelpunkt stellen.

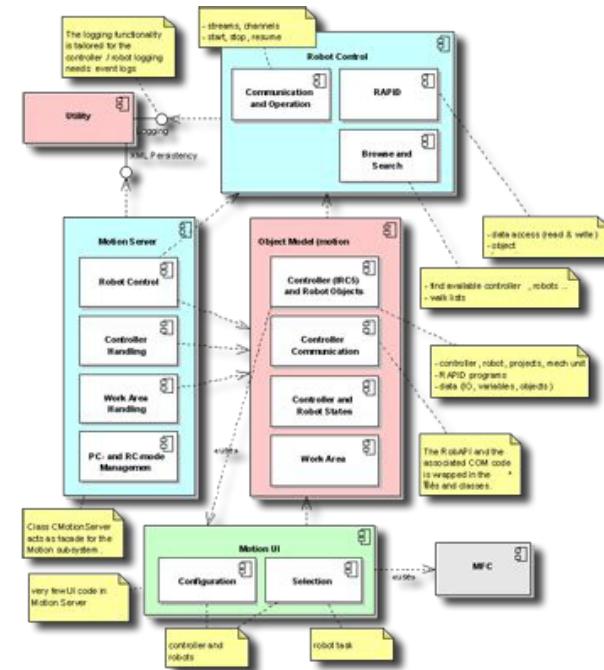
Wichtige Sichten (Notwendigkeit/Umfang hängt vom konkreten Projekt ab):

- **Logische Struktur:** Hierarchische Aufteilung des Systems. Unterscheidung verschiedener Granulationsebenen.
 - z.B. Komponentendiagramm, Klassendiagramm, Objektdiagramm
- **Physikalische Sicht:** Abbildung der Software auf (ggf. logische) Hardwarekomponenten, z.B. Client, Server
 - z.B. Verteilungsdiagramm (Deployment Diagram)
 - insbesondere bei verteilten Systemen
- **Teilsysteme:** Zerlegung des Quellcodes in unabhängig bearbeitbare Teile mit definierten Schnittstellen
 - Paketdiagramm (Package Diagram)
- **Datenfluss:** Wie fließen Daten durch das System
 - z.B. Aktivitätsdiagramm, Kommunikationsdiagramm
- **Kontrollfluss:** Welche Operationen rufen sich gegenseitig in welcher Reihenfolge auf, wo gibt es Nebenläufigkeiten
 - z.B. Aktivitäts-, Sequenz- und Kommunikationsdiagramm

UML (Unified Modelling Language) ist eine Notation und Semantik zum Entwurf, zur Visualisierung und Dokumentation von Modellen für die (insbesondere objektorientierte) Softwareentwicklung.
UML ist ein Standard der OMG (<http://www.omg.org/uml>).

- Sehr weit verbreitet, UML-Wissen ist "Pflicht" für Softwareentwickler
- Einsatz in der Anforderungsanalyse: Use Cases, Modellierung von Geschäftsprozessen
- Einsatz im Entwurf: Diverse Sichten auf die Architektur
- Einsatz in der Implementierung (z.B. Codegenerierung aus den Modellen)
 - aber weniger häufig, u.a. Problem des *Roundtrip-Engineering*
- Es gibt zahlreiche UML-Werkzeuge für die Erstellung von UML Diagrammen und die Generierung von Code (ggf. in die IDE integriert)
- Aktuelle Version: UML 2.5.1

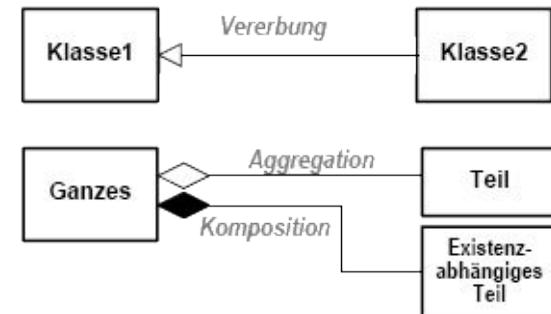
Wichtig: UML + Annotationen + textuelle Beschreibung



UML Strukturdiagramme (1/2)

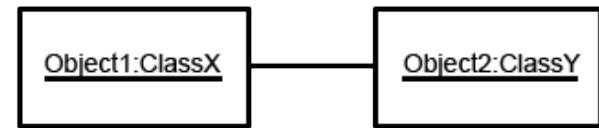
Klassendiagramm (Class diagram)

Statische Struktur von Klassen und Beziehungen
(Assoziationen, Aggregation, Komposition, Spezialisierung/
Generalisierung)



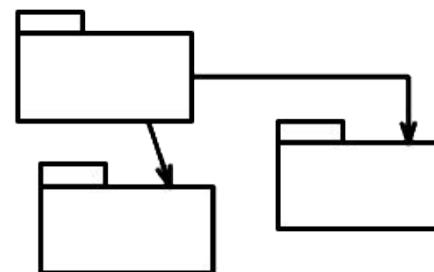
Objektdiagramm (Object diagram)

Struktur von Instanzen+Verbindungen, Laufzeitszenarien



Paketdiagramm (Package diagram)

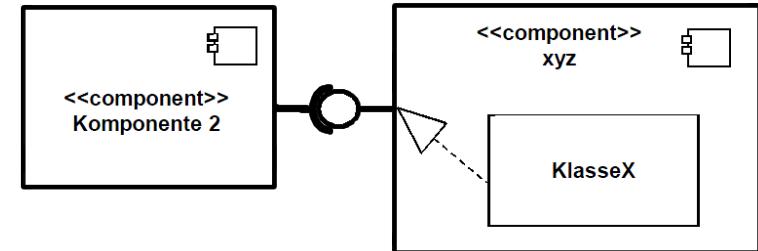
Zerlegung des Gesamtsystems in Pakete/Teilsysteme
(typisch: Source code packages)



UML Strukturdiagramme (2/2)

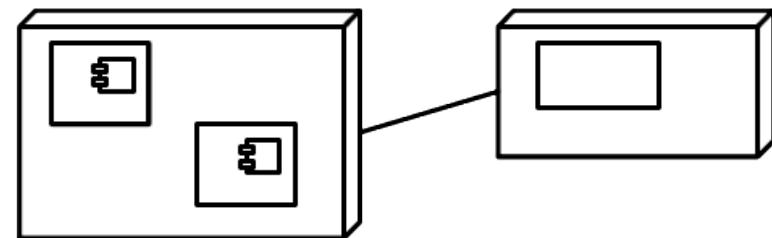
Komponentendiagramm (Component diagram)

Komponenten und Schnittstellen (Anbieter/Nutzer)



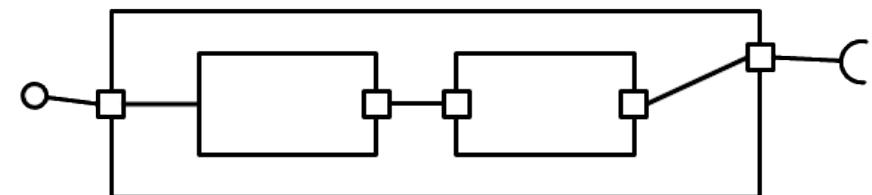
Verteilungsdiagramm (Deployment diagram)

Verteilung von Systembausteinen
auf die Hardware/Infrastruktur



Kompositionssstrukturdiagramm (Composite structure diagram)

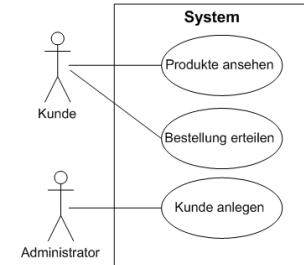
Laufzeitsicht auf Systembausteine, ihre
Zusammenarbeit und Schnittstellen,
z.B. interne Klassenstruktur



UML Verhaltensdiagramme (1/2)

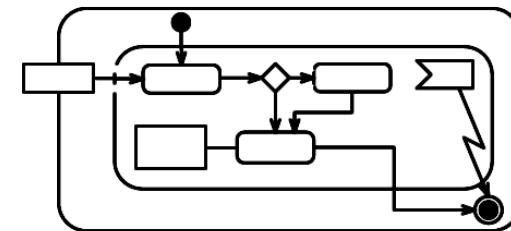
Anwendungsfalldiagramm (Use Case diagram)

Übersicht alle Vorgänge/Prozesse aus Sicht der Anwender (Akteure) und anderer externer Systeme



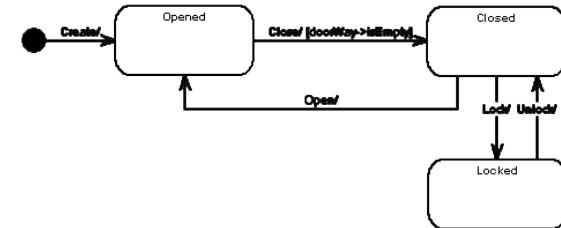
Aktivitätsdiagramm (Activity diagram)

Abläufe innerhalb von Systembausteinen, z.B. Algorithmen, Prozessbeschreibungen (ähnlich wie Flußdiagramme)



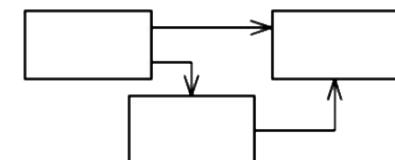
Zustandsdiagramm (State diagram)

Zustände eines Systembausteins, Zustandsübergänge und zugehörige Aktivitäten.



Kommunikationsdiagramm (Communication diagram)

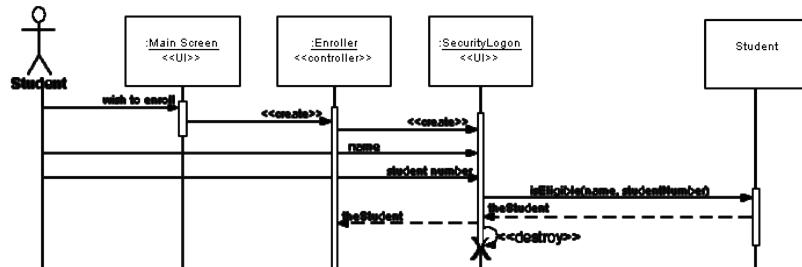
Zusammenarbeit zwischen Instanzen von Systembestandteilen (früher: Kollaborationsdiagramm)



UML Verhaltensdiagramme (2/2)

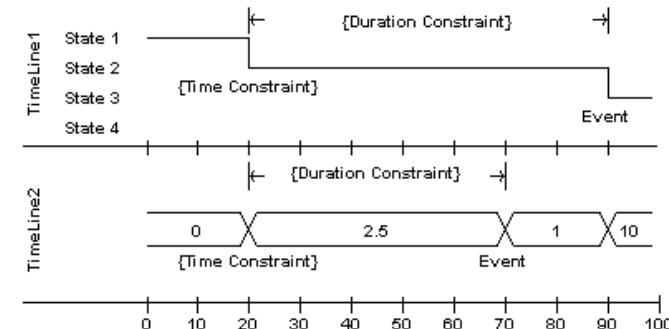
Sequenzdiagramm (Sequence diagram)

Nachrichtenaustausch zwischen Instanzen von Systembausteinen, meist konkrete Szenarien



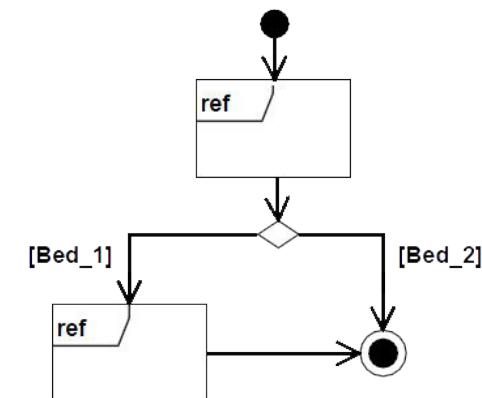
Zeitdiagramm (timing diagram)

Zeitablauf, Zustände/Werte über der Zeit, Zeitabhängigkeiten zwischen Ereignissen



Interaktionsübersichtsdiagramm (Interaction Overview diagram)

Aktivitätsdiagramm auf hoher Abstraktionsebene: Zusammenspiel verschiedener Interaktionen + Referenzen auf Detaildiagramme



- Arc42 (gutes Material, gute Grundlage für das Teamprojekt)
 - <http://www.arc42.de>
 - Dokumententemplates
 - Beispiel: <https://www.dokchess.de/>
- Software Architecture Document Template from CMU SEI
 - <https://wiki.sei.cmu.edu/confluence/display/SAD/Software+Architecture+Documentation+Template>
- Industry example, architecture documentation
- eStore Example
- Template
- RUP
- Design ready checklist

Lieferant: Entwurf



Software-Engineering-Projekt

P.13 Lieferant: Entwurf

Entwerfen Sie schrittweise die Architektur der Softwarelösung:

- a) Welche Ziele verfolgt die Architektur? Inwieweit dient die Architektur der Umsetzung der funktionalen und nicht-funktionalen Anforderungen?
- b) Wie sieht die Systemarchitektur aus: Was sind wesentliche Elemente/Komponenten? Welche Sichten sollten beschrieben werden?
- c) Welches sind die grundlegenden zu verwendenden Technologien (z.B. Betriebssystem, Datenbanken, Programmiersprache). Sind evtl. Technologieevaluationen und/oder Prototypen erforderlich, um Entscheidungen fundiert treffen zu können?
- d) Beschreiben Sie die Architektur und wesentliche Entscheidungen in einem Dokument. Charakterisieren Sie typische Anwendungsszenarien des Systems.



Software-Engineering-Projekt

P.14 Lieferant: Entwurf Review

Ihre Entwurfsphase ist abgeschlossen und das Dokument zur Systemarchitektur liegt vor. Im Rahmen der Qualitätssicherung in Ihrem Unternehmen soll ein erfahrener Architekt aus einer anderen Abteilung ein Assessment vornehmen.

- a) Bereiten Sie eine kurze Präsentation vor, mit der Sie den Assessor von der Qualität Ihrer Softwarearchitektur überzeugen können.
- b) Aktualisieren Sie ggf. Ihre Projektplanung und Ihr Risikomanagement.

Software Engineering I

6. Implementierung

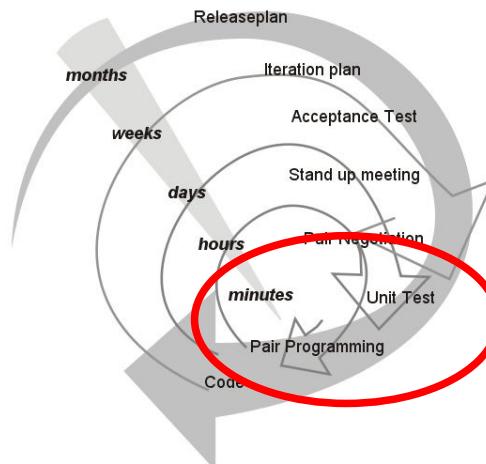
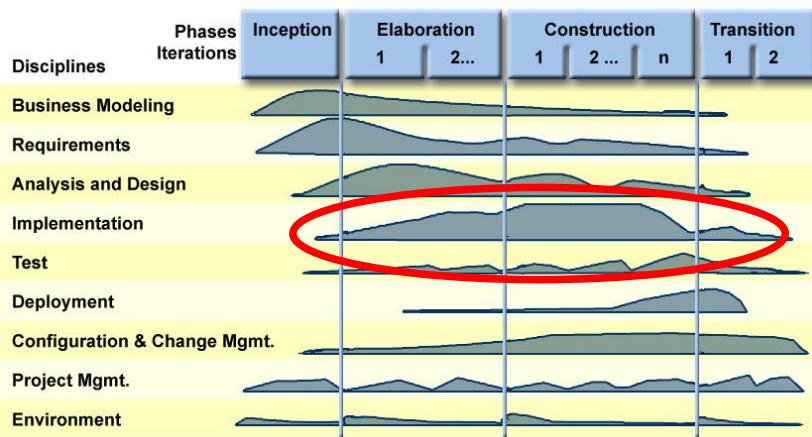
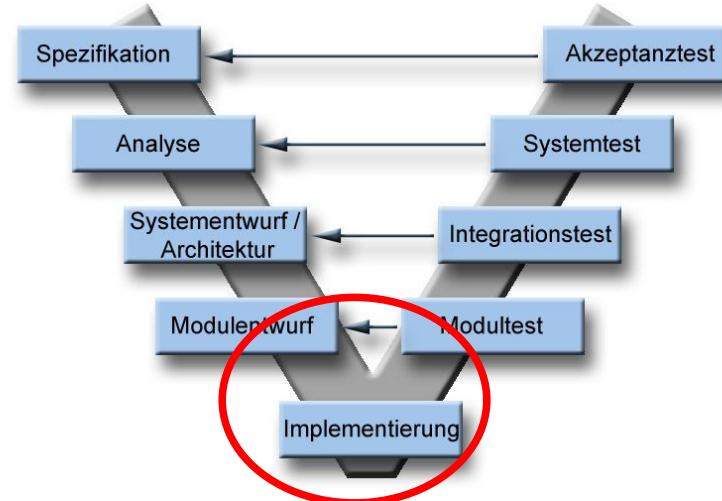
Prof. Dr. Eckhard Kruse

DHBW Mannheim

Die **Implementierung (implementation)** ist die Umsetzung des Entwurfs durch Programmierung (Codierung) der einzelnen Komponenten / Module.

- Zur Implementierung gehört auch die Durchführung von grundlegenden Tests auf Modulebene sowie die Dokumentation des Quellcodes.
- Während der Implementierung können noch Entwurfsentscheidungen innerhalb der Module erforderlich sein. Die Schnittstellen / Verträge zwischen den Komponenten sollten aber möglichst nicht mehr verändert werden.
- **Ergebnisse:** Dokumentierter Quellcode, kompilierte Komponenten/Systemteile, Testprotokolle.

Implementierung in den Vorgehensmodellen



Implementierung: Nur Programmieren?

Programmierer:

- Verständnis der Systemarchitektur + Entwurfsrichtlinien
- Feinentwurf: Wie werden die einzelnen Module realisiert
- Dokumentation
- Programmierung: Codieren auf Modulebene
- Einhalten der Programmierrichtlinien
- Kommunikation, Abstimmung mit dem Team
- Code Reviews
- Debugging
- Modultests (Unit tests): Schreiben von Testcode, Durchführen der Tests, ggf. Dokumentation in Testprotokollen
- Konfigurationsmanagement und Build-Prozess: Einpflegen des eigenen Codes, Befolgen der Prozesse
- Bug Tracker und Change Requests: Einpflegen von Fehlern, Auswerten/Korrigieren von Fehlern, die den eigenen Aufgabenbereich betreffen
- Refactoring

Architekt:

- Beratung, Steuerung
- Unterstützung Feinentwurf
- Änderungsbedarf?

UI Designer:

- GUI Entwurf
- Usability
- Grafiken, HTML, CSS ...

Konfig.manager:

- Build (daily? weekly?)
- Überwachung CMS
- Prozesse, Fristen

Change control board

- Bug tracker, change request

Qualitätsmanager:

- Abnahme der Tests
- Codierrichtlinien
- Prozesse

Was ist guter Code?

**Wodurch zeichnet sich qualitativ
hochwertiger Code aus?
Worauf sollten Programmierer achten?**



Was ist guter Code?

- Zielgerichtet und effizient
 - dient der Umsetzung der Spezifikation (funktional und nicht-funktional)
 - folgt dem Entwurf
 - ist dabei so einfach wie möglich
 - und korrekt
- Verständlich
 - verwendet Kommentare + ggf. externe Dokumentation
 - sinnvolle Variablen- und Methodennamen
 - hält Stilkonventionen ein
 - verwendet ggf. Standardvorgehen (Patterns)
- Sicher und zuverlässig
 - Ausnahmezustände (Exceptions) werden getestet/abgefangen
 - Code ist gut (automatisiert) testbar (z.B. Testinterfaces)

Coding Conventions schreiben einen einheitlichen Stil bei der Programmierung vor, um so die Verständlichkeit des Codes zu verbessern und die Softwarequalität (vor allem Wartbarkeit, Korrektheit, Zuverlässigkeit) zu steigern.

Allgemeine Vorschriften

- Formatierung (Einrücken, Klammerung usw.)
- Namenskonventionen (Variablen, Funktionen)
- Dateisystemstruktur und Dateinamen
- Überprüfen und Behandeln von Ausnahmen (Exceptions)
- Sicherheit (z.B. nur sichere Funktionen verwenden → s. Buffer overflow)
- Lokalisierbarkeit (Native language support)

Anwendungsspezifische Vorschriften:

- Zu verwendende Schnittstellen, verbotene Schnittstellen
- Empfohlene Entwurfsmuster (Patterns)
- ...

Übung

6.1 Coding Conventions

- a) Recherchieren Sie nach Coding Conventions and zugehörigen Tools.
- b) Wie beurteilen Sie die Einführung von Coding Conventions, wenn diese nicht durch Analysewerkzeuge automatisch überprüft werden?

Unter **(Software-)Konfigurationsmanagement (software configuration management)** versteht man die Verwaltung der in der Softwareentwicklung entstehenden Produkte wie Source Code, Konfigurationsdateien, Dokumente usw. unter Berücksichtigung des definierten Entwicklungsprozesses und verwendeter Werkzeuge.

- Versionierung, Labeling (und ggf. Konfliktbehandlung) aller Artefakte
- Definition und Verfolgung von Prozessen
- Zugriffskontrolle, verschiedene Zugriffsrechte
- Protokollierung aller Vorgänge
- Integration mit anderen Werkzeugen, z.B. mit der Entwicklungsumgebung und der System-Build-Umgebung.

Bei einem **Code Review** wird ein Teil des Quellcodes während oder nach der Entwicklung von/mit einem Gutachter (typischerweise ein anderer Entwickler → Peer Review) manuell durchgegangen, um Fehler zu finden bzw. die Qualität des Codes sicherzustellen.

- Vorteile: Bessere Code-Qualität, gegenseitiges Lernen
- Nachteile/Probleme: Aufwand, erfordert Offenheit der Entwickler
- Typischerweise nur für zentrale, wichtige Systemkomponenten
- Begriffe: Code Review, Peer Review, Code Walkthrough
- Kriterien: Nachvollziehbarer Entwurf, Einhalten von Coding Standards, Kommentierung, Verständlichkeit des Codes usw.
- ggf. gezielte (dokumentierte) Maßnahme im Qualitätsmanagement, z.B. als Ergänzung zum Testen



Software-Engineering-Projekt

P.15 Lieferant: Code Review

Führen Sie einen Code Review durch, indem Sie ausgewählte Teile Ihres Codes präsentieren und erläutern.

- a) Welche Code-Teile bieten sich für einen Code Review an (= versprechen Nutzen bzgl. Qualitätsverbesserung)?
- b) Bereiten Sie sich auf den Review vor. Was ist die Aufgabe des Codes? In welchem Zusammenhang ist seine Funktion zu verstehen? Worin bestehen die Schwierigkeiten und die gewählten Lösungsansätze?
- c) Führen Sie den Review durch.
- d) Arbeiten Sie Änderungen/Verbesserungsvorschläge in den Code ein.

Mit dem **(System) Build (Prozess)** bezeichnet man das Kompilieren und Linken aller am System beteiligten Softwarekomponenten (sowohl aus Eigenentwicklung als auch 3rd party Bibliotheken) zu einem ausführbaren System.

- Routineaufgabe (z.B. täglich oder wöchentlich), um möglich frühzeitig und regelmäßig ein testbares Gesamtsystem zur Verfügung zu haben.
- Sollte daher weitestgehend automatisiert ablaufen.
- Für große Systeme potenziell schwieriger, aufwändiger Prozess (z.B. → speziell verantwortlich Mitarbeiter als 'Build Manager')
- Hunderte bis tausende Dateien betroffen.
- Sorgfalt! Jeder einzelne Mitarbeiter kann durch seine Module den Build-Prozess scheitern lassen.
- hilfreich: lokale (teilweise) Buildumgebung für die beteiligten SW-Entwickler.

Don't break the build!



How to recognize a good programmer?

<http://www.inter-sections.net/2007/11/13/how-to-recognise-a-good-programmer/>
How do you recognise good programmers if you are a business guy?
(Assumption: The CV does not help much)

Positive Indicators:

- Passionate about technology
- Programs as a hobby
- Will talk your ear off on a technical subject if encouraged
- Significant (and often numerous) personal side-projects over the years
- Learns new technologies on his/her own
- Opinionated about which technologies are better for various usages
- Very uncomfortable about the idea of working with a technology he doesn't believe to be "right"
- Clearly smart, can have great conversations on a variety of topics
- Started programming long before university/work
- Has some hidden "icebergs", large personal projects under the CV radar
- Knowledge of a large variety of unrelated technologies (may not be on CV)

How to recognize a good programmer?

<http://www.inter-sections.net/2007/11/13/how-to-recognise-a-good-programmer/>
How do you recognise good programmers if you're a business guy?
(Assumption: The CV does not help much)

Negative Indicators:

- Programming is a day job
- Don't really want to "talk shop", even when encouraged to
- Learns new technologies in company-sponsored courses
- Happy to work with whatever technology you've picked, "all technologies are good"
- Doesn't seem too smart
- Started programming at university
- All programming experience is on the CV
- Focused mainly on one or two technology stacks (e.g. everything to do with developing a java application), with no experience outside of it

Lieferant: Alpha-Version



Software-Engineering-Projekt



P.16 Lieferant: Alpha-Version

Eine erste lauffähige (aber noch nicht funktional vollständige) Version Ihrer Software ist fertig. Präsentieren Sie diese Alpha-Version dem Kunden.

- a) Ist der Kunde zufrieden? Ist das Projekt auf dem richtigen Weg?
- b) Aktualisieren Sie ggf. Ihre Projektplanung und Ihr Risikomanagement.

Software Engineering I

7. Test und Qualitätssicherung

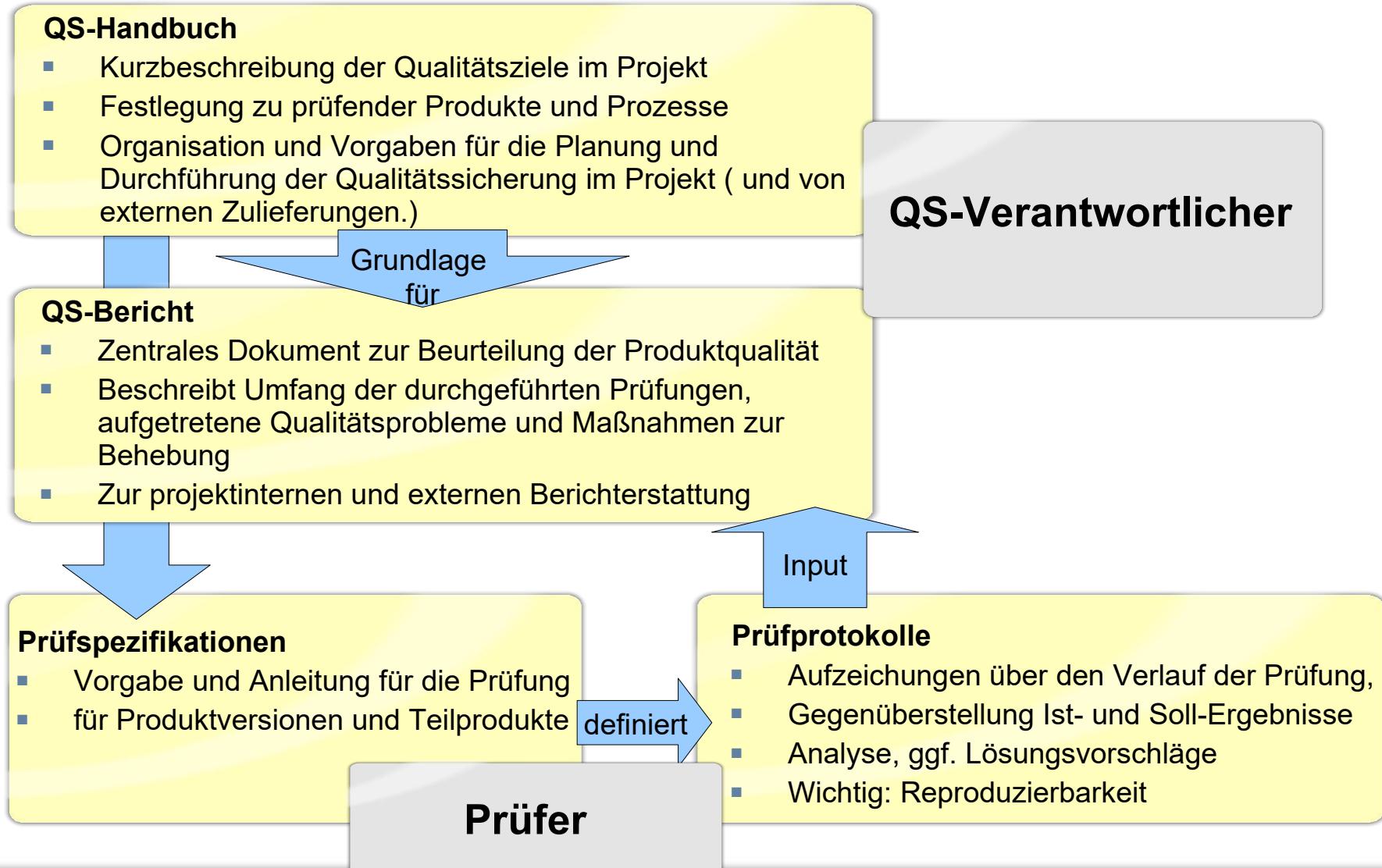
Prof. Dr. Eckhard Kruse

DHBW Mannheim

Softwarequalität ist die Gesamtheit der Merkmale und Merkmalswerte eines Softwareprodukts, die sich auf dessen Eignung beziehen, festgelegte oder vorausgesetzte Erfordernisse zu erfüllen.
(DIN ISO 9126)

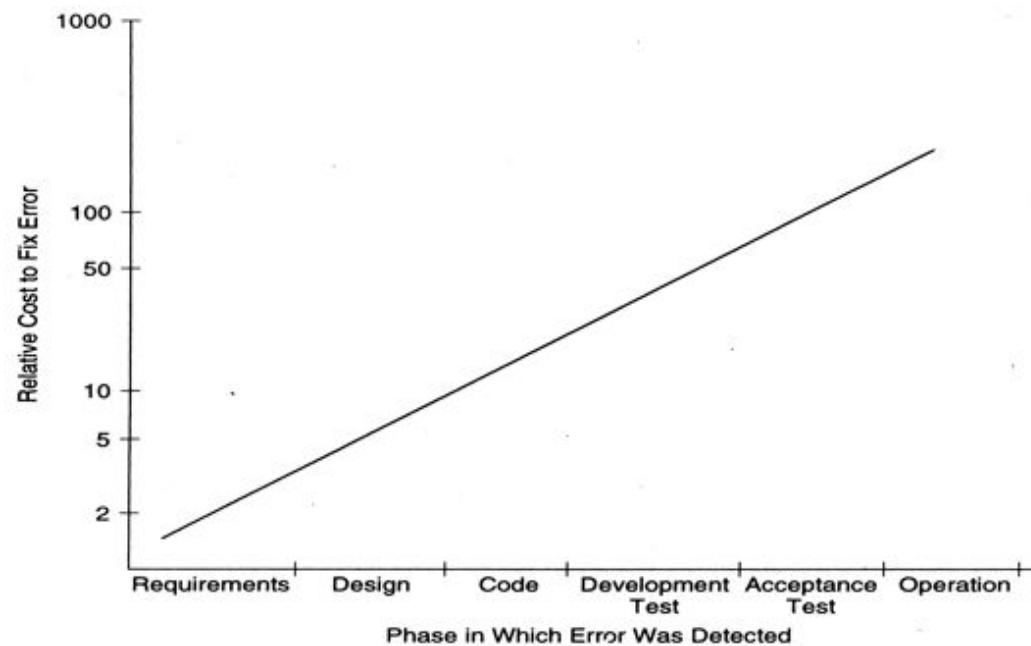
- Spezifikation der gewünschten Eigenschaften ist Voraussetzung, um die Qualität zu beurteilen.
- Wichtige Software-Qualitätsmerkmale:
 - Korrektheit: Software entspricht den Anforderungen
 - Zuverlässigkeit: Geringe Ausfallzeiten/Ausfallwahrscheinlichkeit
 - Robustheit: Unempfindlich gegenüber (kleineren) Fehlern/Störungen
- Qualität ist schwer messbar (im Gegensatz zu Hardware, z.B.: Fertigungstoleranzen, statistische Ausfallwahrscheinlichkeiten, Verfügbarkeit)
- Maßnahmen zur Qualitätssteigerung, z.B.:
 - Testen (und ggf. formale Verifikation)
 - Audits und Reviews
 - Definierte Entwicklungsprozesse

Qualitätssicherung im V-Modell



Kosten von Fehlern

- Fehler können in jeder Phase des Entwicklungsprozesses auftreten.
- Je größer die Zeit zwischen Ursache und Behebung des Fehlers, desto größer die Kosten
- Qualitätssicherung ist von Anfang an wichtig!



Verifikation und Validierung

Definition!

Verifikation (**Verification**) ist die Prüfung, dass ein Softwareartefakt die korrekten, d.h. spezifizierten Ergebnisse liefert.

Are we doing the product right?

- Modultests, Komponententests, Integrationstests
- Reviews mit Entwicklern
- Formale Verifikation: Mathematische Beweisverfahren, um die Korrektheit von Software (meist nur von kleinen Teilsystemen) zu beweisen.

Validierung (**Validation**) ist die Prüfung der Eignung einer Software bezogen auf ihren Einsatzzweck.

Are we doing the right product?

- Abnahmetest, Akzeptanztest (user acceptance test): Werden aus Sicht des Kunden/Anwender mit der Software die angestrebten Ziele erreicht?
- Reviews mit Kunden/Anwendern

Nicht-funktionale Eigenschaften

Die Softwarequalität bzgl. nicht-funktionaler Eigenschaften lässt sich oft nur sehr schwierig nachweisen/quantifizieren.

Beispiel Zuverlässigkeit:

- Wahrscheinlichkeit, dass sich ein System in einem gegebenen Zeitraum erwartungsgemäß verhält
- Hardware-Zuverlässigkeitsmaße, z.B.
 - Mean time between failure (MTBF)
 - Mean time to repair (MTTR)
 - z.B. Hard disk: $MTBF = 500000$ h
 - Und wenn das nicht reicht → Redundanz

Und Software-Zuverlässigkeit?
MTBF, Redundanz, ...?



Fault/Defect/Bug:

Fehlerhafte Stelle im System, die ein Fehlverhalten auslösen kann.

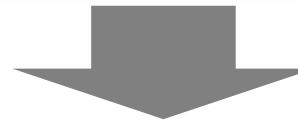
Damit **Faults** nicht automatisch zu **Failures** führen:

- Fault tolerance
- Error handling / Exception handling
- Redundancy



Error

Diskrepanz zwischen einem tatsächlichen Wert/Verhalten im System und dem gewünschten, theoretischen Wert/Verhalten.



Failure

Tatsächlich auftretendes, fehlerhaftes Verhalten des Systems, d.h. das Programm ist zu einem bestimmten Zeitpunkt nicht mehr in der Lage, seine korrekte Funktion zu erledigen.

Testspezifikation

Die Testspezifikation beschreibt die durchzuführenden Tests und die erwarteten Ergebnisse.

- Definition von verschiedenen Testfällen:
 - oft in direkter Anlehnung an die Use Cases, d.h. typische Abläufe aus Anwendersicht
- Für jeden Testfall: Schritt-für-Schritt-Auflistung der Vorbedingungen, Eingabe und des erwarteten (überprüfbaren) Ergebnisses.
 - "*Anwender klickt auf Menüpunkt xy*"
 - "*Neues Fenster wird eingeblendet, Daten werden gespeichert usw.*"
- Wichtig: Auch Sonderfälle und unerlaubte Eingaben sollten getestet werden.
- Typisch: Black-Box-Testing = Testen aus Sicht der Anwenders, der kein Wissen über den inneren Aufbau des Systems hat.

Unter **Regressionstests** (regression tests) versteht man die regelmäßige Wiederholung von identischen Testfällen, um auch in bereits getesteten Softwareteilen eventuell nachträglich entstandene Fehler (z.B. aufgrund von Nebenwirkungen von Änderungen) zu erkennen.

- Aufgrund der regelmäßigen Wiederholung (und des damit verbundenen Aufwands) sollten Regressionstest möglichst automatisiert erfolgen.
- Sinnvoller Ansatz z.B.: Automatisiertes Durchführen von Regressionstests im Rahmen des Buildprozesses: Jedes erfolgreiche Build wird automatisch einer Reihe von Systemtests unterzogen.

Unter **Testabdeckung** (test coverage / code coverage) versteht man den prozentualen Anteil von getesteten Softwareteilen im Verhältnis zum Gesamtumfang der Software.

- Meist nur ein geschätztes, theoretisches Maß
- Die Abdeckung kann sich auf verschiedene Kriterien beziehen (mit zunehmendem Aufwand):
 - Funktionen
 - Befehle
 - Bedingungen
 - Ausführungspfade
- 100%ige Testabdeckung lässt sich nur sehr schwer garantieren (komplexe analytische Verfahren, formale Methoden)

- Quality Plan – Document template
- Test plan – template
- Testspezifikation: Einfaches Template

Lieferant: Code Freeze + Testen



Software-Engineering-Projekt

P.17 Lieferant: Code Freeze+Testen

Die Implementierung aller Funktionen des Softwaresystems ist abgeschlossen. Die verbleibenden Entwicklungsaktivitäten sollen sich auf das Testen und Beheben von Fehlern konzentrieren. Um nicht in der letzten Projektphase noch neue Risiken und Instabilität in das System hineinzubringen, wird typischerweise ein „Code Freeze“, d.h. ein Einfrieren des Codes beschlossen. Erforderliche Änderungen der Implementierung (um Fehler zu beheben) sind nur noch nach Absprache und Abwägen möglicher Risiken möglich. Überlegen Sie, wie Sie Ihr System systematisch testen wollen.

- Wie dokumentieren Sie erkannte Fehler und den Status der Fehlerbehebung (Schwere des Fehlers, soll er behoben werden, Prioritäten usw.).
- Bereiten Sie sich vor (ggf. mit ein, zwei Folien), dem Qualitätsmanager zu berichten, wie Sie in der Testphase obige Schritte umsetzen, um eine hohe Softwarequalität sicherzustellen.

Kunde-Lieferant: Übergabe + Abnahme



Software-Engineering-Projekt

P.18 Kunde-Lieferant: Übergabe und Endabnahme

Der Lieferant hat einen Termin beim Kunden, um das Produkt zu übergeben und den Kunden einzuweisen. Der Kunde macht eine Endabnahme. Ist alles in Ordnung, so dass die endgültige Rechnungsstellung und Bezahlung erfolgen können?



Software-Engineering-Projekt

P.19 Lieferant: Interne Abschlusspräsentation

Ihr Management möchte in einem Abschlussvortrag über Ihr Projekt informiert werden. War das Projekt ein Erfolg, konnten die Ergebnisse wie gewünscht an den Kunden ausgeliefert werden und ist der Kunde zufrieden? Lessons learnt: Was lief gut, was hätte rückblickend anders gemacht werden sollen? Können Sie das Management überzeugen, dass Sie sich mit diesem Projekt für weitere Projektleitungsaufgaben qualifizieren?