



*AG Software Engineering
AG Algorithmic Bioinformatics*

Projektmanagement im Softwarebereich

SeqAn

Björn Kahlert

Institut für Informatik
Freie Universität Berlin
10.04.2012

Vorbemerkungen

- Hauptquellen
 - Prof. Dr. Lutz Prechelt,
Vorlesung Softwaretechnik
 - Prof. Dr. Lutz Prechelt,
Vorlesung Softwareprozesse
 - Prof. Dr. Oscar Nierstrasz,
Vorlesung Introduction to Software Engineering

Software Engineering

Computer programs, procedures, rules, and possibly associated documentation and data pertaining to the operation of a computer system.

IEEE Standard Glossary of
Software Engineering Terminology
/ANSI83/

The systematic approach to the development, operation, maintenance, and retirement of software.

- Ingenieurmäßiges Vorgehen (d.h. auf Basis wissenschaftlicher Erkenntnisse)
- Systematische Entwicklung
 - Ermittlung der Anforderungen
 - Pflege und Fortentwicklung

IEEE Standard Glossary of Software Engineering Terminology /ANSI 83/

Software Engineering



Projectmanagement
Project Management

14-16h

Qualitätssicherung
Quality Assurance

Anforderungen
Requirements

Entwurf
Design

Implementierung
Implementation

Wartung
Maintenance

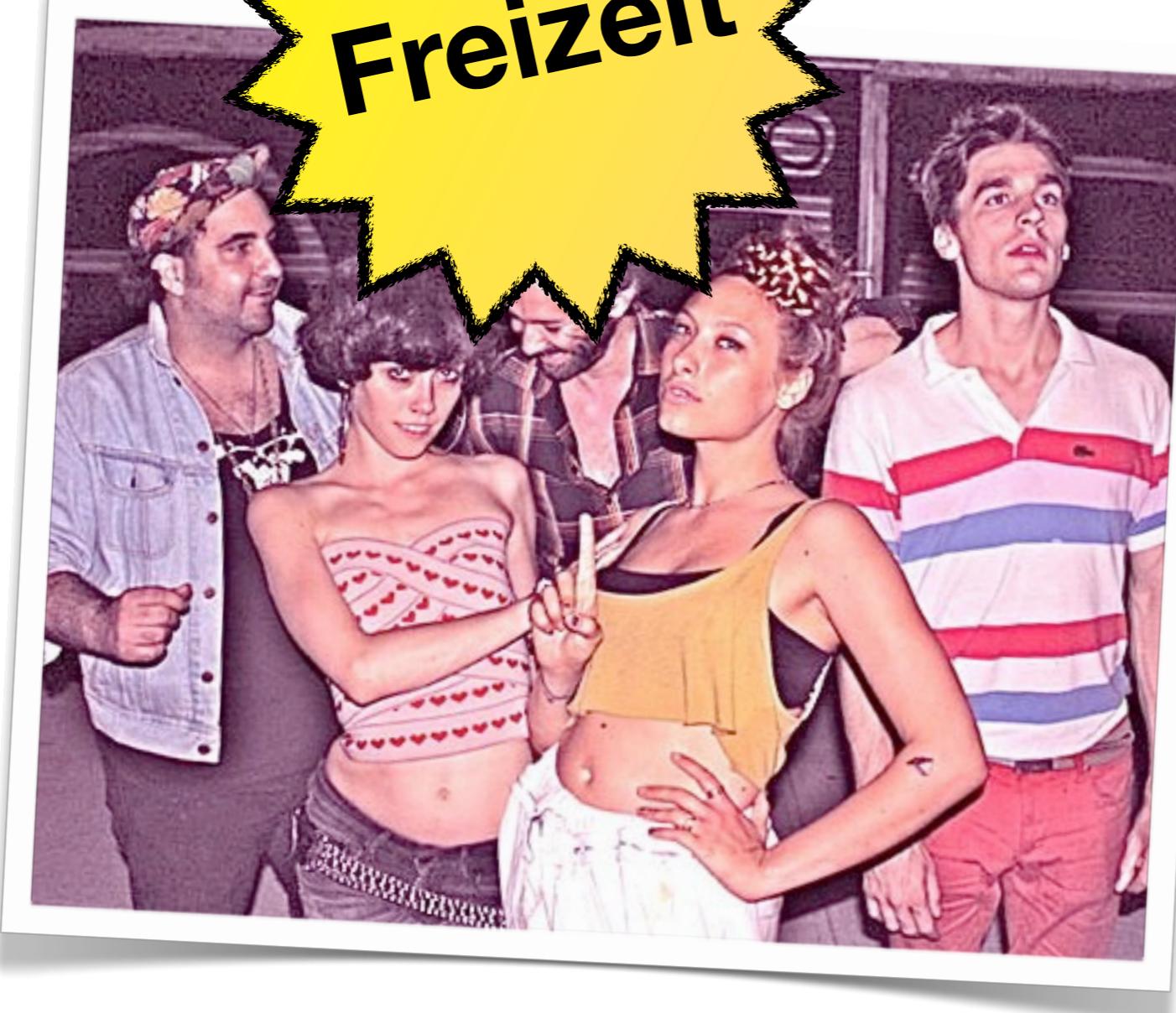
9-12 h

- “Am Schluss hat sich rausgestellt, dass wir zur Hälfte Sachen gebaut hatten, die der Kunde nicht brauchte, aber dafür mit dem Rest nie genau das richtige.”
 - Anforderungsbestimmung
- “Die meisten Defekte sind keine Programmierfehler, sondern Missverständnisse.”
 - Software-Entwurf, konstruktive Qualitätssicherung
- “Das Schlimmste ist, dass wir beim nächsten Projekt wieder die gleichen Sachen falsch machen werden.”
 - Prozessmanagement

Wozu?



Freizeit



- 
- Keine Bezahlung für
 - einen Hochschulabschluss
 - erworbene Wissen
 - irgendwelche Taten
 - irgendwelche Problemlösungen
 - Sonderlich für
 - Problemlösungen, bei denen der Nutzen der Lösung höher ist als die Kosten
 - besser: viel höher
- Probleme und ihre Wichtigkeit verstehen
- Probleme **effizient** lösen
- Kosten und Nutzen von Technologie und Methoden abschätzen

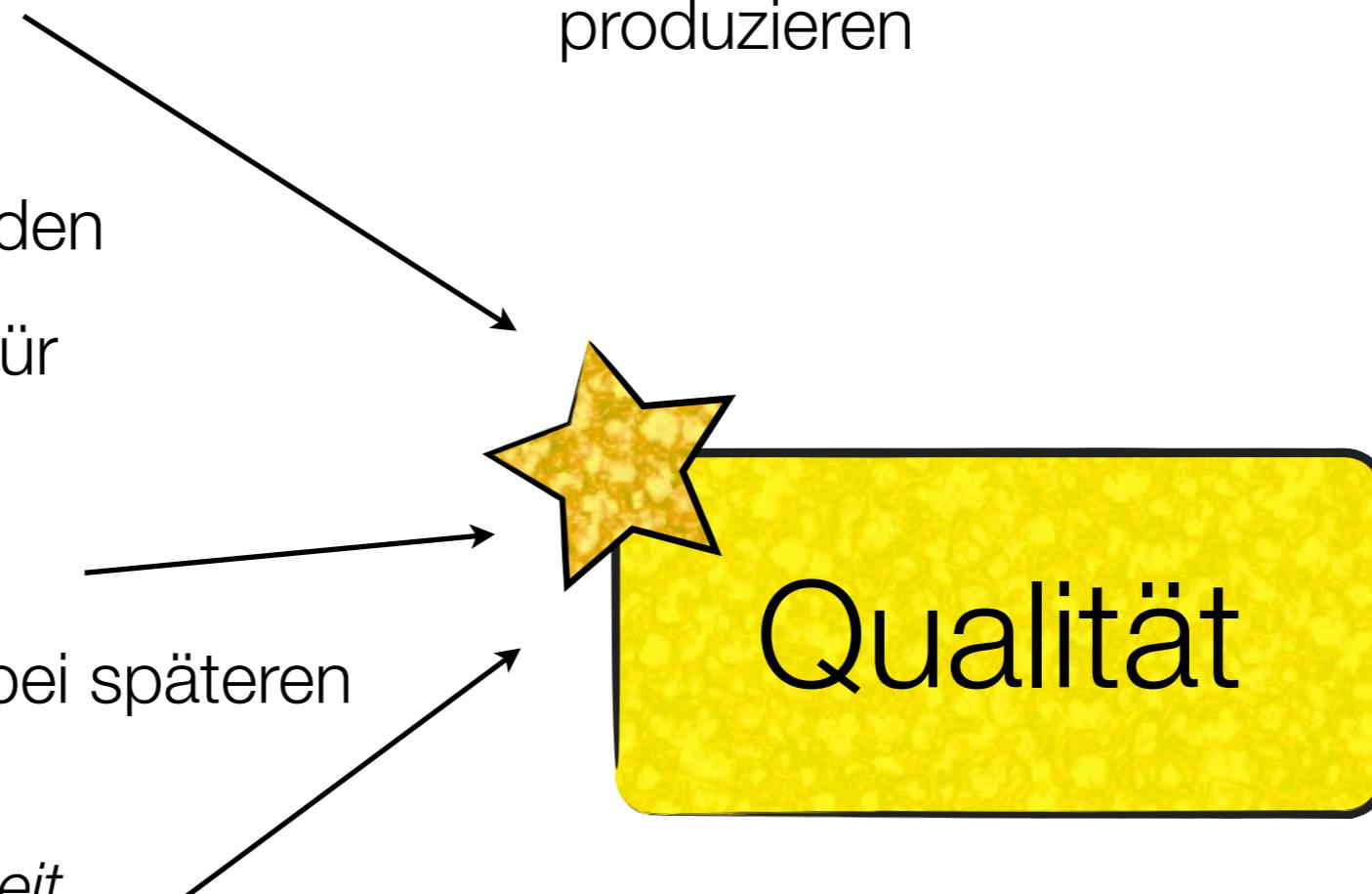
Kosten-/Nutzen-Optimierung

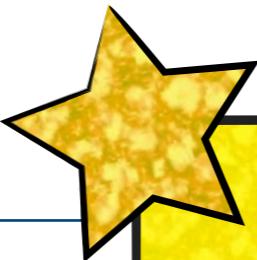
Kosten

- Software mit möglichst wenig Arbeitsaufwand fertigstellen
 - *durch Wiederverwendung, Qualitätssicherung, Risikomanagement*
- Künftige Kosten vermeiden
 - keine hohen Kosten für Defektkorrekturen
 - durch hohe Qualität
 - keine hohen Kosten bei späteren Änderungen
 - durch gute Wartbarkeit

Nutzen

- Wertvolle Anforderungen aufdecken und umsetzen
- dazu passende SW hoher Qualität produzieren





Qualität

Extern / aus Benutzersicht

- Benutzbarkeit
 - Bedienbarkeit, Erlernbarkeit, Robustheit, ...
- Verlässlichkeit
 - Zuverlässigkeit, Verfügbarkeit, Sicherheit, Schutz
- Brauchbarkeit
 - Angemessenheit, Geschwindigkeit, Skalierbarkeit, Pflege, ...

Intern / aus Entwicklersicht

- Zuverlässigkeit
 - Korrektheit, Robustheit, Verfügbarkeit, ...
- Wartbarkeit
 - Verstehbarkeit, Änderbarkeit, Testbarkeit, Korrektheit, Robustheit
- Effizienz
 - Speichereffizienz, Laufzeiteffizienz, Skalierbarkeit

Mittel zum Zweck



Privatkunde

- Geldverschwendungen

Benutzer

- Frustration
- Abnahme der Produktivität, Sicherheit und des Komforts

Unternehmenskunde

- Risiko
- Abnahme der Konkurrenzfähigkeit

Projektbeteiligten

- Überstunden
- Frustration
- Zerstörung der Motivation
- Störung der beruflichen Fortentwicklung

Grobunterteilung



1
Perspektive

2
**Lebensdauer
des Wissens**

3
**Art der
Aktivität**

4
Aspekt

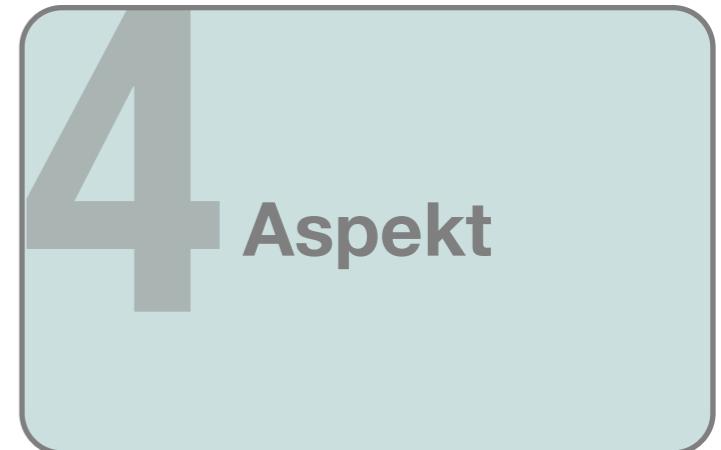


Produkt

- Gesamtheit der greifbaren Arbeitsergebnisse
 - meist genannt "Dokumente" oder "Artefakte"
 - z.B. Programmcode, Code für Test/Build/Installation etc., Benutzerdokumentation, interne technische Dokumentation, Pläne und Protokolle, etc.

Prozess

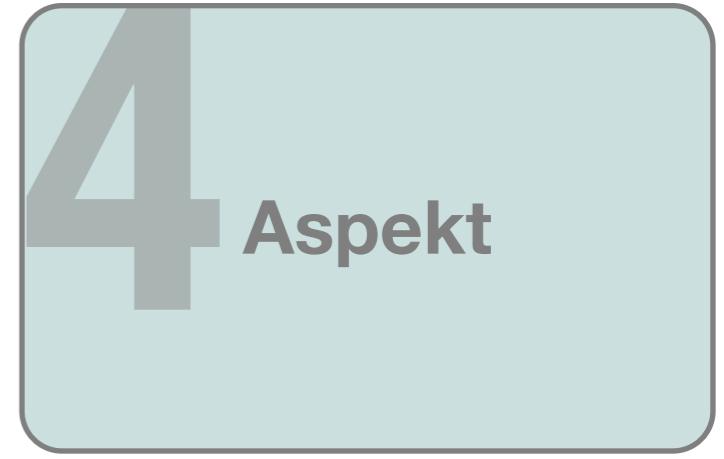
1. Für ein Projekt: Gesamtheit aller konkreten Abläufe
2. Allgemein: Gesamtheit der zugrunde liegenden allgemeinen Konzepte u. Verfahrensweisen ("Prozessmodell")
 - Rollen, Aktivitäten, Methoden, Werkzeuge, etc.



Grobunterteilung



	Prinzip Grundansatz, an dem man sein Handeln orientiert	<ul style="list-style-type: none">• sehr abstrakt & allgemein• Lebenslang richtig & nützlich
	Methode Planmäßig angewandte Vorgehensweise zum Erreichen festgelegter Ziele	<ul style="list-style-type: none">• abstrakt & auf Bereich spezialisiert• einige Jahre (ca. 2-15) relevant
	Verfahren Präzise und recht konkrete Vorgehensvorschrift	<ul style="list-style-type: none">• konkret oder leicht konkretisierbar• für spezifischen Bereich / Zweck
	Werkzeug Program, das bei einem Verfahren oder einer Methode unterstützt	<ul style="list-style-type: none">• nur für kurze Zeit relevant





Kosten/Nutzen-Analyse

- Ob? Was? Wozu?

Anforderungen

- Ob? Was? Wozu?

Entwurf

- Wie?

Implementierung

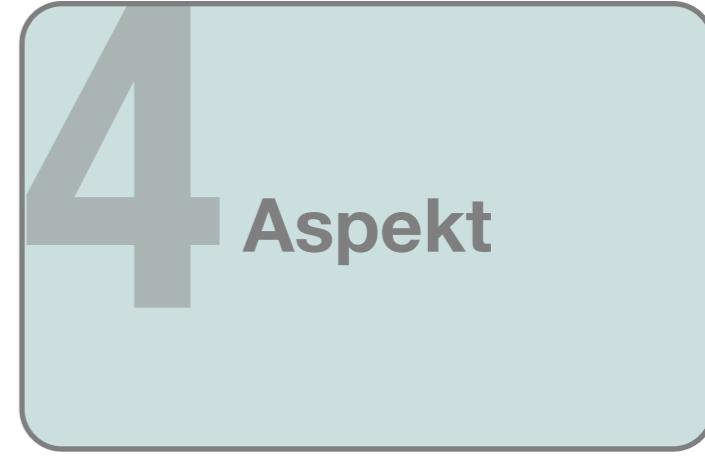
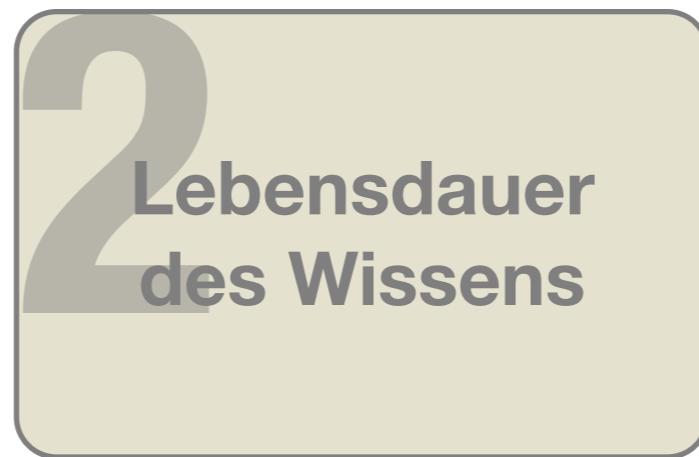
- Bauen

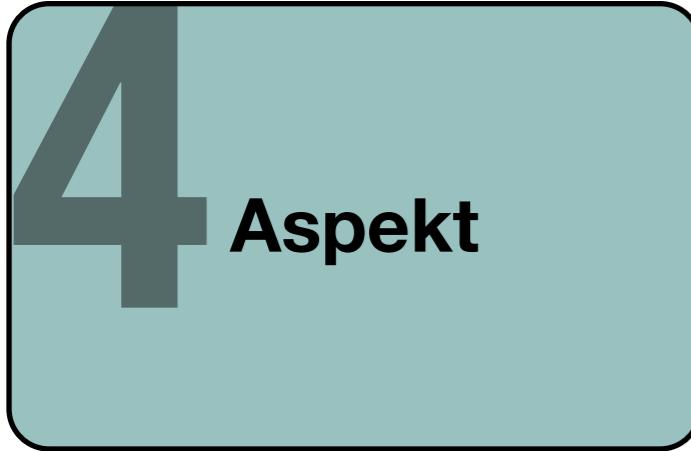
Qualitätssicherung
(analytisch)

- Validieren

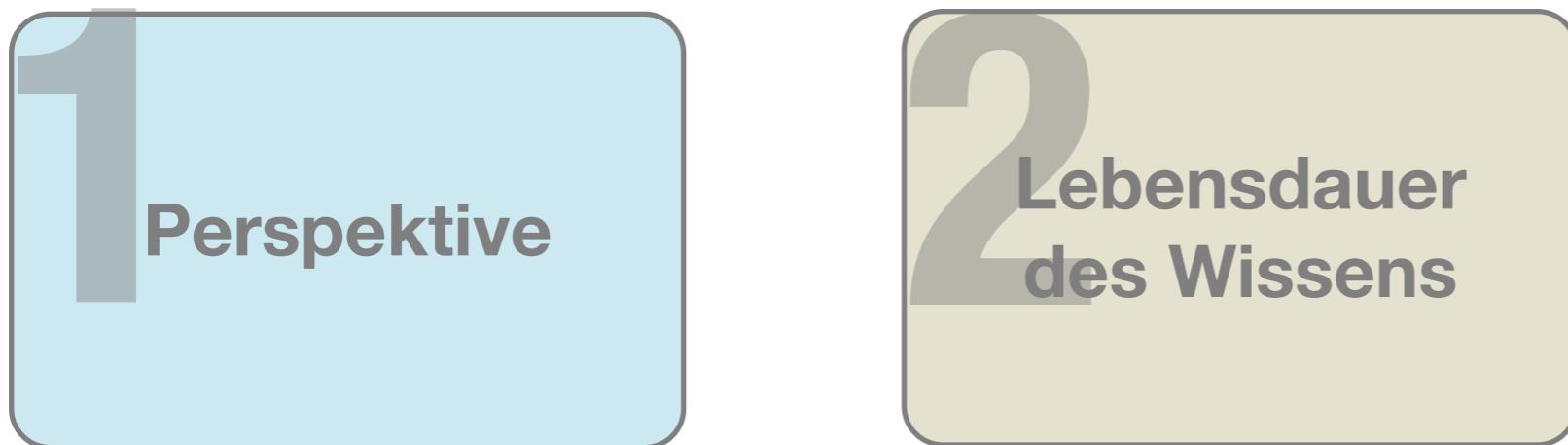
Projektmanagement

- Verwalten





- Beide Aspekte werden von vielen Softwareingenieuren und Projektmanagern leider gern ignoriert
- Technische Aspekte sind nicht alles!



Softwaretechnik wird von Menschen betrieben

- Menschliche Eigenschaften sind wichtige Treiber für die Softwaretechnik
- Alle Beteiligten: Entwickler, Benutzer, Kunden, Manager (und -innen)
- z.B. haben Menschen Beschränkungen (etwa bei Wissen, Kurzzeitgedächtnis, Kommunikationsvermögen), die entscheidend für die Gestaltung der Methoden der SWT sind

- Außerdem zeigen Menschen variables Verhalten: sie sind manchmal ängstlich, aggressiv, beeinflussbar, einfallslos, eitel, faul, fleißig, ignorant, kreativ, müde, mutig, souverän, redselig, schweigsam, wachsam, u.a.m.

Software wird von Menschen benutzt

- sozio-technisches System
- z.B. Programmcode, Code für Test/Build/Installation etc., Benutzerdokumentation, interne technische Dokumentation, Pläne und Protokolle, etc.

- Nur gewisse Softwaretechnik-Methoden sind für gewisse Situationen ideal!
 - Die übrigen meist problematisch / unbrauchbar.
- Nähe und Anzahl Kunden
- Art der Benutzer / Benutzung
- Größe / Komplexität der Software und des Projekts
- Wie kritisch ist (nicht technisches) Domänenwissen?
- Müssen Näherungslösungen verwendet werden?
 - Wie kritisch ist Effizienz?
 - Wie kritisch ist Verlässlichkeit?

Art der Benutzer

- Fall 1: Geschulte, professionelle Benutzer
 - SW-Ingenieure (für ein API-basiertes SW-Produkt)
 - Fluglotsen (Flugsicherungssystem)
- Fall 2: Gering oder gar nicht ausgebildete Benutzer
 - MS Word
 - Navigationssystem
 - Solche SW ist viel schwieriger gut hinzubekommen: Erfordert gutes Verständnis für Benutzbarkeitsgestaltung

Domänenwissen

- Fall 1: Eine rein softwaretechnische Domäne
 - Beispiel: Betriebssystemkern, Compiler
 - Hier verstehen die SW-Ingenieure ohne fremde Hilfe die Anforderungen recht gut
- Fall 2: Eine Domäne mit komplexer Fachlichkeit
 - Beispiel: SW zur Berechnung des Kreditausfallrisikos einer Bank
 - Hier sind die SW-Ingenieure bei den Anforderungen naiv
 - Die Anforderungen müssen von Fachspezialisten geliefert werden
 - Kommunikationsproblem!

Faustregeln

1. Wenn es eine interaktive Benutzungsschnittstelle hat, ist es nie rein technisch.
2. Die Domäne ist immer vielschichtiger als man denkt.

Domänenwissen ist wichtig

Wichtige Erkenntnis:

- In vielen SW-Projekten kann man die Anforderungen nur verstehen, wenn man Wissen im betroffenen fachlichen Bereich ("Anwendungsdomäne") hat
 - Und ohne Verständnis der Anforderungen kann man offensichtlich kaum für brauchbare Ergebnisse garantieren...
- Es gibt viele solche fachlichen Bereiche
- In mindestens einem davon sollte man sich auskennen!

Kooperation ist unumgänglich

- Ein größeres Softwareprojekt ist stets so komplex, dass keine einzelne Person noch ein komplettes Verständnis davon entwickeln kann
 - Viele Gruppen von Beteiligten
 - Arbeit im Team (sogar in vielen parallelen Teams) ist unvermeidlich
 - Schriftliche Dokumentation ist unvermeidlich

Emergente Eigenschaften



- Die emergenten Eigenschaften komplizierter Systeme lassen sich durch keine Theorie komplett vorhersagen
 - Man versteht und beherrscht sie nur allmählich aufgrund von Erfahrung
- Etwas Neues ist also nur verstehbar/beherrschbar, wenn es Bekanntem genügend ähnelt
- **Deshalb ist erfolgreiches Ingenieurwesen davon abhängig, immer möglichst viel so zu machen "wie üblich"**
 - **und möglichst wenig "neu" oder "ganz anders"**
- denn ein einzelnes Projekt soll ja zuverlässig und auf Anhieb klappen; nicht "nur allmählich aufgrund von Erfahrung" (s. oben)

Normales/radikales Vorgehen

- Etwas so zu machen "wie üblich" nennen wir "**normales Vorgehen**"
 - N1 Wir wissen, wie das geht. Wir tun es routinemäßig.
 - N2 Wir verstehen, worauf es dabei ankommt (Erfolgsfaktoren)
 - N3 Wir kennen die typischen Probleme dabei (Risikofaktoren)
 - N4 Wir verstehen, wofür das Vorgehen geeignet ist und wofür vielleicht nicht (Anwendbarkeitsbereich)
 - N5 Wir dürfen zuversichtlich sein, einen Erfolg zu erzielen
- Etwas "ganz anders" zu machen oder etwas "ganz Neues" zu tun nennen wir "**radikales Vorgehen**"
 - R1 Wir wissen nur grob (allgemein, nicht konkret), wie das geht
 - R2 Wir verstehen nur teilweise, worauf es dabei ankommt
 - R3 Wir wissen noch nicht, welche Probleme uns bevorstehen
 - R4 Wir verstehen höchstens ansatzweise Eignung und Grenzen
 - R5 Wir können allenfalls hoffen, einen Erfolg zu erzielen

Sollte Hauptbestandteil eines SW-Projekts sein

Erleichtert Innovationen;
erhöhtes Risiko kann besser mit Ausbildung bewältigt werden

Wichtigster Ansatz für normales Vorgehen:

Wiederverwendung

- von bewährten Komponenten
- von bewährten Entwurfsüberlegungen/Entwürfen
- von bewährten Anforderungen
- von bewährten Prozesselementen
- von bewährten Werkzeugen

Achtung: Wiederverwendung hat zwei Hauptziele • Senkung der Kosten

- Senkung des Risikos
- Aus Sicht des normalen Vorgehens ist nur die Risikosenkung von Interesse
- Die Kostensenkung gibt es quasi gratis dazu

Aus dieser Sicht sind also zwei Fragen an die Softwaretechnik zu richten:

- Welches Wissen für normales Vorgehen ist verfügbar?
- Welche Ideen und Methoden helfen dabei, ein (teilweise) nicht-normales Vorgehen möglichst oft zum Erfolg zu führen?

Welt der Problemstellungen

- Produkt (Komplexitätsproblem)
 - Anforderungen (Problemraum)
 - Entwurf (Lösungsraum)
- Prozess (psycho-soziale Probl.)
 - Kognitive Beschränkungen
 - Mängel der Urteilskraft
 - Kommunikation, Koordination
 - Gruppendynamik
 - Verborgene Ziele
 - Fehler

Welt der Lösungsansätze

- Technische Ansätze ("hart")
 - Abstraktion
 - Wiederverwendung
 - Automatisierung
- Methodische Ansätze ("weich")
 - Anforderungsermittlung
 - Entwurf
 - Qualitätssicherung
 - Projektmanagement

Anmerkung

- Die Taxonomie ist alles andere als kanonisch.
- Aber sie liefert einen nützlichen Orientierungsrahmen.

Welt der Problemstellungen

- "Mit welchen Phänomenen muss die SWT fertig werden?"
 - Enthält all das, was Programmieren-im-Großen unterscheidet vom Programmieren-im-Kleinen
 - Alle diese Probleme sind essenziell
 - d.h. sie lassen sich nicht beseitigen,
 - sondern nur abmildern.
 - Berühmter Aufsatz dazu:
 - Fred(erick) Brooks: "No Silver Bullet: Essence and Accidents of Software Engineering", IEEE Computer 20(4), April 1987, www.computer.org

Welt der Problemstellungen

- Produkt (Komplexitätsproblem)
 - Anforderungen (Problemraum)
 - Entwurf (Lösungsraum)
- Prozess (psycho-soziale Probl.)
 - Kognitive Beschränkungen
 - Mängel der Urteilskraft
 - Kommunikation, Koordination
 - Gruppendynamik
 - Verborgene Ziele
 - Fehler

Welt der Lösungsansätze

- Technische Ansätze ("hart")
 - Abstraktion
 - Wiederverwendung
 - Automatisierung
- Methodische Ansätze ("weich")
 - Anforderungsermittlung
 - Entwurf
 - Qualitätssicherung
 - Projektmanagement

- Komplexität bedeutet
 - ein System besteht aus vielen Einzelteilen und
 - es hat **emergente Eigenschaften**
 - Phänomene, die man nicht verstehen kann, indem man die Einzelteile betrachtet
 - sondern die erst aus deren Zusammenwirken entstehen
- "Komplex" heißt also praktisch so viel wie
 - "schwierig aufgrund vielfältiger Zusammenhänge"
- Komplexitätsprobleme bei SW-Entwicklung stammen meist aus der Komplexität des Produkts und liegen
 - im Problemraum (Anforderungen; Was wollen wir bauen?)
 - oder im Lösungsraum (Entwurf; Wie sollen wir es strukturieren?). Meistens in beiden.

- Problem: Herausfinden, **was** genau gebaut werden soll
- Teilprobleme:
 - Fachexperten können Bedarf nicht gut genug **ausdrücken**
 - (bei sehr innovativen Anwendungen gibt es gar keine Fachexperten)
 - und haben zuwenig Fantasie, sich SW-Möglichkeiten auszumalen
 - Verschiedene Gruppen von Fachexperten bringen **widersprüchliche Anforderungen** ein
 - Fachexperten und Technikexperten benutzen verschiedene, manchmal inkompatible Terminologie und sehr verschiedene **Darstellungsformen**
 - und verstehen einander nur sehr schwer
 - Anforderungen ändern sich im Laufe der Zeit
 - oft schon lange vor Abschluss des Entwicklungsprojekts

- Problem: Herausfinden, wie die SW strukturiert werden sollte, um die Anforderung gut zu erfüllen
 - Es gibt viele Möglichkeiten mit verschiedenen Stärken und Schwächen
 - Man muss darunter die günstigen entdecken und erkennen
- Teilprobleme:
 - Lösungsmöglichkeiten (er)kennen
 - Inkzeptable herausfiltern
 - Wirkung der Möglichkeiten auf die Qualitätseigenschaften verstehen
 - Prioritäten der Qualitätseigenschaften verstehen
 - Prioritäten gegeneinander abwägen ("Äpfel und Birnen")
 - Meist geht das nur per Umrechnung in Kosten
 - Die ist aber sehr oft dubios und sehr künstlich

Welt der Problemstellungen

- Produkt (Komplexitätsproblem)
 - Anforderungen (Problemraum)
 - Entwurf (Lösungsraum)
- Prozess (psycho-soziale Probl.)
 - Kognitive Beschränkungen
 - Mängel der Urteilskraft
 - Kommunikation, Koordination
 - Gruppendynamik
 - Verborgene Ziele
 - Fehler

Welt der Lösungsansätze

- Technische Ansätze ("hart")
 - Abstraktion
 - Wiederverwendung
 - Automatisierung
- Methodische Ansätze ("weich")
 - Anforderungsermittlung
 - Entwurf
 - Qualitätssicherung
 - Projektmanagement

- Entstehen aus zwei Quellen:
 - Menschen haben Schwächen und verhalten sich "merkwürdig"
 - Zur SW-Konstruktion müssen Menschen zusammenarbeiten
- Diese Probleme entstehen aus dem Prozess heraus
 - und hängen nur wenig direkt vom Produkt ab

Kognitive Beschränkungen



- Die Denkfähigkeiten eines Menschen sind begrenzt
- Wichtigste Einschränkung:
 - Limitiertes Arbeitsgedächtnis
(Kurzzeitgedächtnis plus Aufmerksamkeitslenkung)
 - es kann nur ca. 7 ± 2 Elemente (chunks, Bündel) aufnehmen

- Problem: Oft kann keiner der Beteiligten in einem Projekt eine Situation gut genug beurteilen, um eine fällige Entscheidung richtig zu treffen
- Tritt hauptsächlich in zwei Situationen auf:
 - Radikales Vorgehen (technischer Bereich und Prozessbereich):
 - Emergente Eigenschaften lassen sich nicht abschätzen
 - Rückgängigmachen falscher Entscheidungen (Prozessbereich):
 - Hier stehen psychologische Hürden im Weg (Verzerrungen der Wahrnehmung)

- Problem: Es ist schwierig, vorhandene Information stets korrekt und rechtzeitig an die Person weiterzugeben, die sie braucht
 - Inhaltliche Information (Anforderungen, Entwurfsentscheidungen etc.)
 - Organisatorische Information (Abstimmung, Koordination)
- Hauptquellen:
 - kognitive Beschränkungen oder Mängel der Urteilskraft
 - soziale Phänomene wie individuelle Abneigungen, z.B.
 - gegen Kommunikation allgemein (Introversion),
 - gegen bestimmte Personen (Aversion),
 - gegen die Kommunikation bestimmter Arten von Informationen (Überbringung schlechter Nachrichten) usw.

- Problem: Arbeiten in einer Gruppe beeinflusst Haltungen und Entscheidungen oft in unvernünftiger Art und Weise
- Beispiel:
 - "Groupthink" führt dazu, dass gewisse (notwendige) kritische Fragen nicht mehr gestellt werden
 - z.B. bei Vorherrschen der Ansicht "Durchsichten kosten zuviel Zeit"
 - Selbsterfüllende Prophezeiungen sind in Gruppen besonders wirksam
 - z.B. "Wir überziehen unseren Liefertermin immer."

Verborgene Ziele

Problem: Menschen handeln nicht immer nur im Interesse des Projekts, sondern haben persönliche (meist unausgesprochene) Ziele

- die zum Teil den Interessen des Projekts zuwider laufen
- Beispiele für verborgene Ziele
- Selbst viel lernen
- Spaß bei der Arbeit haben
- möglichst elegante Lösungen finden (auf die man stolz ist)

- Achtung bei Kollegen finden; Achtung bei Vorgesetzten finden • Gehaltserhöhung bekommen
- früh nach Hause gehen; faul sein
- spät nach Hause gehen; Privatleben verdrängen
- bestimmte Kollegen schlecht aussehen lassen
- bestimmten Kollegen aus dem Weg gehen
- u.a.m.



Fehler

- Problem: Will ein Mensch X tun, so tut er oft Y
 - ohne es zu wollen oder zu merken und
 - ohne dass Y ein brauchbarer Ersatz für X wäre • Man nennt die Wahl von Y dann einen *Fehler*
- Resultat von Fehlern sind oft Defekte in Dokumenten • Resultate von Defekten Versagen der SW
- Fehler sind nur Erscheinungsformen der oben schon genannten Probleme
- sind aber häufig genug und wichtig genug, um eine eigene Erwähnung zu rechtfertigen

Welt der Lösungsansätze

- Was haben SW-Ingenieure und Softwaretechnik- Forscher(innen) an Ideen entwickelt, um den Problemen zu begegnen?
 - Wir geben nur eine Überblick und reißen grundlegende Ideen an.
 - **Hauptgegenstand dieser Vorträge!**

Welt der Problemstellungen

- Produkt (Komplexitätsproblem)
 - Anforderungen (Problemraum)
 - Entwurf (Lösungsraum)
- Prozess (psycho-soziale Probl.)
 - Kognitive Beschränkungen
 - Mängel der Urteilskraft
 - Kommunikation, Koordination
 - Gruppendynamik
 - Verborgene Ziele
 - Fehler

Welt der Lösungsansätze

- Technische Ansätze ("hart")
 - Abstraktion
 - Wiederverwendung
 - Automatisierung
- Methodische Ansätze ("weich")
 - Anforderungsermittlung
 - Entwurf
 - Qualitätssicherung
 - Projektmanagement

Technische Ansätze ("hart")

- Diejenigen Ansätze, die sich weit genug konkretisieren lassen, um in Notationen oder Werkzeuge überführt zu werden
- Durch diese Konkretheit ("hartes" Wissen) sind technische Ansätze einfacher zu verstehen und anzuwenden als die "weichen" methodischen Ansätze
 - Methoden bieten aber mehr Anleitung für ihre Anwendung
- Anmerkung: Wir benennen im Folgenden nicht einzelne Ansätze, sondern Kategorien von Ansätzen
 - Die Kategorien selbst sind nicht sehr konkret
 - Die einzelnen technischen Ansätze sind es dann aber sehr wohl

- Abstraktion: Konzentration auf wesentliche Eigenschaften durch Weglassen von Details
 - Andere Sicht: Gruppierung gleichartiger Dinge gemäß einer Gemeinsamkeit und Ignorieren der sonstigen Unterschiede
 - Vorgehensweise: Bildung eines Konzepts (Begriffs) und Benennung dieses Konzepts durch eine Bezeichnung
- **Kernprinzip der gesamten Informatik!**
- Verwendung für alle möglichen Zwecke
 - z.B. Beschreibung von Produkten/Produktteilen, Vorgehensweisen, konkreten Vorgängen etc.
- Arten von Begriffen:
 - Allgemeine Begriffe
(z.B. Elemente von Notationen wie etwa die Konstrukte einer Programmiersprache)
 - Vorhabensspezif. Begriffe (z.B. einzelne Module eines Systems)
 - Begriff/Abstraktion reduziert den Baustein auf seine Schnittstelle
 - Modulname ist die Bezeichnung der Abstraktion

Wiederverwendung

- Spezialfall von Abstraktion:
 - Benutze nicht nur den Begriff mehrmals, sondern auch damit verbundene Details
- Wiederverwendung geht nicht nur bei Programmcode, sondern auch für z.B.
 - Arbeitsprodukte wie Anforderungen, Anforderungsmuster, Architekturen, Teilentwürfe, Entwurfsmuster, Testfälle
 - Prozesshilfsmittel wie Dokumentschablonen, Vorgehensbeschreibungen, Checklisten, Prozessmuster, Softwarewerkzeuge
- **Wiederverwendung ist die Hauptquelle von Produktivitätsverbesserungen in der Softwaretechnik!**

- Übertrage eine wiederholt zu erledigende Tätigkeit dem Computer
 - spart Zeit (und damit Kosten)
 - vermeidet triviale Durchführungsfehler
- Kann als Spezialfall von Wiederverwendung betrachtet werden
- Ist das Automatisierungsprogramm flexibel für viele Situationen einsetzbar, so nennt man es **Softwarewerkzeug**

Welt der Problemstellungen

- Produkt (Komplexitätsproblem)
 - Anforderungen (Problemraum)
 - Entwurf (Lösungsraum)
- Prozess (psycho-soziale Probl.)
 - Kognitive Beschränkungen
 - Mängel der Urteilskraft
 - Kommunikation, Koordination
 - Gruppendynamik
 - Verborgene Ziele
 - Fehler

Welt der Lösungsansätze

- Technische Ansätze ("hart")
 - Abstraktion
 - Wiederverwendung
 - Automatisierung
- Methodische Ansätze ("weich")
 - Anforderungsermittlung
 - Entwurf
 - Qualitätssicherung
 - Projektmanagement

Methodische Ansätze ("weich")

- Methodische Ansätze sind nur schwach vorstrukturiert
 - deshalb als "weich" empfunden (kein "hartes Wissen")
 - Benötigen menschliche Intelligenz zur Durchführung
- Abgrenzung zu technischen Ansätzen:
 - Kein Teil des Problems lässt sich abspalten und universell lösen
 - (alles, was sich doch abspalten und universell lösen lässt, zählen wir also nicht mehr zu den methodischen Problemen)
 - Die Übergänge sind aber dennoch fließend

Anforderungsermittlung

- Einsicht: Man darf sich nicht auf intuitiven Eindruck darüber verlassen, was gebaut werden sollte,
 - sondern sollte die Anforderungen systematisch ermitteln
- Prinzipien:
 - **Erhebung** der Anforderungen bei allen Gruppen von Beteiligten
 - **Beschreibung** in einer Form, die die Beteiligten verstehen
 - **Validierung** anhand der verschriftlichten Form
 - **Spezifikation:** Übertragung in eine zur Weiterverarbeitung günstige Form
 - **Trennung** von Belangen: Anford. kopplungsarm ausdrücken
 - **Analyse auf Vollständigkeit:** Lücken aufdecken und schließen
 - **Analyse auf Konsistenz:** Widersprüche aufdecken und lösen
 - **Mediation:** Widersprüche, die auf Interessengegensätzen beruhen, einer Lösung zuführen (Kompromiss oder Win-Win)
 - **Verwaltung:** Übermäßige Anforderungsänderungen eindämmen, Anforderungsdokument immer aktuell halten.

- Einsicht: Besser nicht einfach drauf los programmieren,
 - sondern Struktur u. Funktionsweise vorab im Ganzen festlegen,
 - um bessere Qualität zu ermöglichen und
 - um arbeitsteilige Realisierung zu erleichtern
- Prinzipien:
 - **Trennung von Belangen (separation of concerns)**: Belange (insbes. Funktionen) so von einander trennen, dass man sie einzeln verstehen, realisieren und verändern kann
 - **Architektur**: Treffe globale Festlegungen für die Gestaltung nicht abtrennbarer Belange (meist: nichtfunktionale Anforderungen)
 - **Modularisierung**: Verberge die Umsetzung von Belangen möglichst hinter einer Schnittstelle (information hiding)
 - dadurch wird der Belang lokal und ist einfacher zu ändern
 - **Wiederverwendung**: Erfinde Architekturen und Entwurfsmuster nicht immer wieder neu
 - **Dokumentation**: Entwurf u. Entwurfsentscheidungen schriftlich festhalten (jeweils: Zweck, Alternativen, Argumentation)

- Einsicht: Man macht oft Fehler, die zu schweren Mängeln in der SW führen können
 - Solchen Mängeln sollte man vorbeugen Prinzipien:
- Konstruktive Qualitätssicherung
 - Ergreife **vorbeugende Maßnahmen**, die vermeiden helfen sollen, dass überhaupt erst etwas falsch gemacht wird
 - Oft auch genannt **Qualitätsmanagement** (inkl. prüfende Maßnahmen) oder **Prozessmanagement**
- Analytische Qualitätssicherung
 - Verlasse dich nie ganz auf die Vorbeugung, sondern verwende zusätzlich **prüfende Maßnahmen**, die entstandene Mängel aufdecken sollen, damit man sie beheben kann
 - Wichtigste Vertreter (bislang): **Test, Durchsichten**

- Einsicht: Bau einer größeren SW verlangt nach Planung, Leitung und Koordination
- Prinzipien:
 - **Zielsetzung**, Prioritäten, Pläne, individ. Aufgaben müssen allen Beteiligten klar sein und sie sollten sich damit identifizieren
 - **Stabile Anforderungen**: Anforderungen dürfen sich nicht zu schnell verändern
 - **Iteration**: Projekt muss in kurzen Abständen wohldefinierte Ergebnisse hervorbringen (Meilensteine)
 - **Kommunikation**: Informationen müssen rechtzeitig die richtige(n) Person(en) erreichen
 - **Konflikte** zw. Beteiligten auf zielführende Weise lösen
 - **Risikomanagement**: Identifiziere wichtige unerwünschte Ereignisse, leite vorbeugende Maßnahmen ein oder bereite Gegenmaßnahmen vor; überprüfe regelmäßig
 - **Normales Vorgehen**: Vermeide radikales Vorgehen, es sei denn, der erwartete Nutzen rechtfertigt es

Welt der Problemstellungen

- Produkt (Komplexitätsproblem)
 - Anforderungen (Problemraum)
 - Entwurf (Lösungsraum)
- Prozess (psycho-soziale Probl.)
 - Kognitive Beschränkungen
 - Mängel der Urteilskraft
 - Kommunikation, Koordination
 - Gruppendynamik
 - Verborgene Ziele
 - Fehler

Welt der Lösungsansätze

- Technische Ansätze ("hart")
 - Abstraktion
 - Wiederverwendung
 - Automatisierung
- Methodische Ansätze ("weich")
 - **Anforderungsermittlung**
 - Entwurf
 - Qualitätssicherung
 - Projektmanagement

Anforderungsermittlung

Requirements Engineering

- Einsicht: Man darf sich nicht auf intuitiven Eindruck darüber verlassen, was gebaut werden sollte
 - sondern sollte die Anforderungen systematisch ermitteln

Requirement / Anforderung



A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document.

The set of all requirements forms the basis for subsequent development of the system or system component

Etwas, das dein System* haben muss, um jemanden die Lösung eines Problems zu ermöglichen / ein Ziel zu erreichen.

* entweder Computer-System (Systemanforderungen) oder sozio-technisches System (Benutzeranforderungen)

[IEEE Std]

Functional requirements:

- What the system does: the interactions between the system and its environment; independent from implementation

Nonfunctional requirements:

- Observable aspects of the system that are not directly related to functional behavior
- e.g. performance or reliability aspects, etc.
- oft dominant!
- typisch für größere Softwaresysteme:
 - Die Komplexität entsteht vor allem aus den nichtfunktionalen

- Anforderungen

Safety/security requirements ("shall not" properties)

- A kind of nonfunctional requirement: Behavior the system must never exhibit
- e.g. "must be impossible to apply reverse thrust in mid-flight"

Constraints ("Pseudo requirements"):

- Imposed by the client or environment in which the system operates
- Often concern the technology to be used (language, operating system, middleware etc.)

Requirements Engineering



[...] Requirements Engineering is the branch of systems engineering concerned with real-world goals for, services provided by, and constraints on software systems.

Requirements Engineering is also concerned with the relationship of these factors to precise specifications of system behaviour and to their evolution over time and across system families...

- Anforderungserhebung ist Teil von Requirements Engineering

[Zave94]

Requirements Engineering

1. Requirements Elicitation
2. Requirements Specification
3. Requirements Validation
4. Requirements Management

Requirements Engineering



- 1. Requirements Elicitation**
2. Requirements Specification
3. Requirements Validation
4. Requirements Management

- Which problem needs to be solved?
 - identify problem Boundaries
- Where is the problem?
 - understand the **Context/Problem Domain**
- Whose problem is it?
 - identify **Stakeholders** (Betroffene, Beteiligte)
- Why does it need solving?
 - identify the stakeholders' **Goals**
- How might a software system help?
 - collect some **Scenarios**
- When and how does it need solving?
 - identify **Development Constraints**
- What might prevent us solving it?
 - identify **Feasibility and Risk**

1. Requirements Elicitation

2. Requirements Specification

3. Requirements Validation

4. Requirements Management

- What is the external behavior? **Functional Modeling**
 - Create scenarios and use case diagrams
 - Talk to client, observe, get historical records, do thought experiments
- What is the structure of the system? **Object Modeling**
 - Create class diagrams
 - Identify objects
What are the associations between them?
Multiplicity? What are the attributes of the objects?
What operations are defined on the objects?
- What is its behavior? **Dynamic Modeling**
 - Create sequence diagrams
 - Dynamic Modeling
 - Identify senders and receivers
Show sequence of events exchanged between objects
Identify event dependencies and event concurrency
 - Create state diagrams
 - Only for the dynamically interesting objects

1. Requirements Elicitation
2. Requirements Specification

3. Requirements Validation

4. Requirements Management

- Verification checks the equivalence of different formal representations
- Validation checks if a system fulfills
 - Validation also checks:
 - Did we understand all the important **Requirements**?
 - Did we understand all the relevant **Domain properties**?
 - Problem with requirements validation
 - Requirements change during requirements elicitation
 - Requirements Management
 - **Requirements change all the time!**

Welt der Problemstellungen

- Produkt (Komplexitätsproblem)
 - Anforderungen (Problemraum)
 - Entwurf (Lösungsraum)
- Prozess (psycho-soziale Probl.)
 - Kognitive Beschränkungen
 - Mängel der Urteilskraft
 - Kommunikation, Koordination
 - Gruppendynamik
 - Verborgene Ziele
 - Fehler

Welt der Lösungsansätze

- Technische Ansätze ("hart")
 - Abstraktion
 - Wiederverwendung
 - Automatisierung
- Methodische Ansätze ("weich")
 - Anforderungsermittlung
 - **Entwurf**
 - Qualitätssicherung
 - Projektmanagement

Entwurf

- Einsicht: Man sollte vor dem Kodieren über eine günstige Struktur der Software nachdenken
 - und diese als Koordinationsgrundlage schriftlich festhalten
- Prinzipien:
 - **Trennung von Belangen (separation of concerns)**: Belange (insbes. Funktionen) so von einander trennen, dass man sie einzeln verstehen, realisieren und verändern kann
 - **Architektur**: Treffe globale Festlegungen für die Gestaltung nicht abtrennbarer Belange (meist: nichtfunktionale Anforderungen)
 - **Modularisierung**: Verberge die Umsetzung von Belangen möglichst hinter einer Schnittstelle (information hiding)
 - dadurch wird der Belang lokal und ist einfacher zu ändern
 - **Wiederverwendung**: Erfinde Architekturen und Entwurfsmuster nicht immer wieder neu
 - **Dokumentation**: Entwurf u. Entwurfsentscheidungen schriftlich festhalten (jeweils: Zweck, Alternativen, Argumentation)

- Man weiß (mehr oder weniger) was man bauen will. D.h.
- Man kennt die funktionalen Anforderungen
 - z.B. beschrieben durch Use Cases etc.
 - und konkretisiert durch das Analysemodell
- Man kennt die nichtfunktionalen Anforderungen
 - wenige oder viele, lasch oder stringent
 - betreffend z.B. Effizienz, Bedienarten, Bedienbarkeit, Robustheit, Sicherheit, Schutz, Erweiterbarkeit, Technologievorgaben, Zuverlässigkeit, Verfügbarkeit, Erlernbarkeit u.a.m.

Was müssen wir als nächstes leisten?



- Herausfinden wie man es bauen kann/sollte. D.h.:
 - Entscheiden wie/wo/wodurch die funktionalen Anforderungen (Funktionen) umgesetzt werden
 - z.B. Technologieauswahl, Modulzerlegung, Wiederverwendung
 - Wir klammern Technologieauswahl und Wiederverwendung aus
 - Frage 1 ist also: **Wie zerlegt man ein System klug in Teile?**
- Herausfinden, wie man dabei die nichtfunktionalen Anforderungen alle erfüllen kann
 - nichtfunktionale Anforderungen haben meist globalen Charakter, werden also nicht von nur wenigen Modulen realisiert
 - Frage 2 ist also: **Wie findet man eine Gesamtstruktur mit den gewünschten globalen Eigenschaften?**

Entwurf Architektur

- Das Problem bei der Suche nach einer geeigneten Gesamtstruktur ist folgendes:
 - Die globalen Eigenschaften sind meist emergente Eigenschaften
 - d.h. sie lassen sich nicht den Einzelteilen zuweisen, sondern entstehen erst aus deren Zusammenwirken
 - z.B. Speicherbedarf, Robustheit, Verfügbarkeit, Sicherheit
 - Emergente Eigenschaften sind sehr schwierig im Voraus abzuschätzen
 - Es gibt bereits mehrere erprobte Gesamtstrukturen!

- Das Problem bei der Suche nach einer geeigneten Gesamtstruktur ist folgendes:
 - Die globalen Eigenschaften sind meist emergente Eigenschaften
 - d.h. sie lassen sich nicht den Einzelteilen zuweisen, sondern entstehen erst aus deren Zusammenwirken
 - z.B. Speicherbedarf, Robustheit, Verfügbarkeit, Sicherheit
 - Emergente Eigenschaften sind sehr schwierig im Voraus abzuschätzen
- Es gibt bereits mehrere erprobte Gesamtstrukturen!

- Meist gibt es ein paar dominante nichtfunktionale Eigenschaften
- und eine bekanntermaßen dazu gut passende Gesamtstruktur
- Dann benutzt man solche bekannten Gesamtstrukturen
 - oft "Architekturen" genannt
- Wenn Sie ein ganz ungewöhnliches, neuartiges System bauen wollen, haben Sie es allerdings evtl. schwer!
 - Deshalb lohnt es sich, Neuartigkeit zu begrenzen (wo das geht)
 - Siehe normales vs. radikales Vorgehen

The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

- Es geht stets um eine globale Betrachtung!
 - Die aber, wenn man sie gründlich macht, bis auf die Detailebene herunterreicht
 - Deshalb gibt es z.B. so viele UML-Diagrammarten
→ verschiedene Sichten und Detailniveaus

<http://www.sei.cmu.edu/architecture/definitions.html>

Klient-/Dienstgeber-Architektur (client/server arch.)

- eine einfache Sorte verteilter Architekturen

Ereignisgesteuertes System (event-based arch.)

- eine Sorte lose gekoppelter Architekturen

Ablage-basierte Architektur (repository arch.)

- noch eine Sorte lose gek. A.; oft mit Ereignissteuerung verbunden

Unterbrechungsorientiertes System (interrupt-based system)

- eine Architektur für kleinere Echtzeitsysteme

Mehrschicht-Architektur (layered arch.)

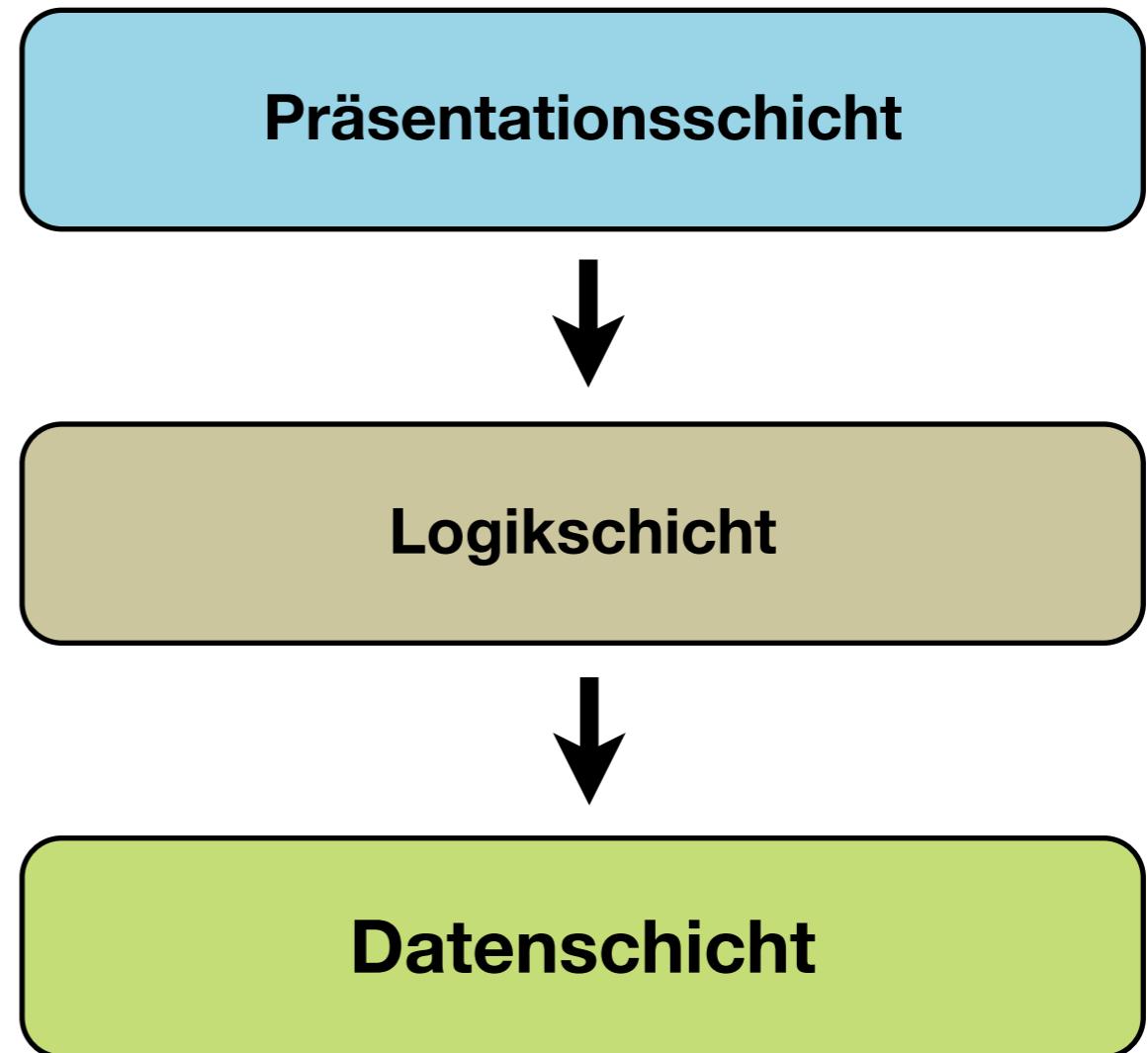
- ein allgemeiner A.stil, der mit vielen anderen Architekturideen verbunden werden kann

JavaEE-Architektur, CORBA-Architektur, .NET-Architektur

- technologiezentrierte Architekturen

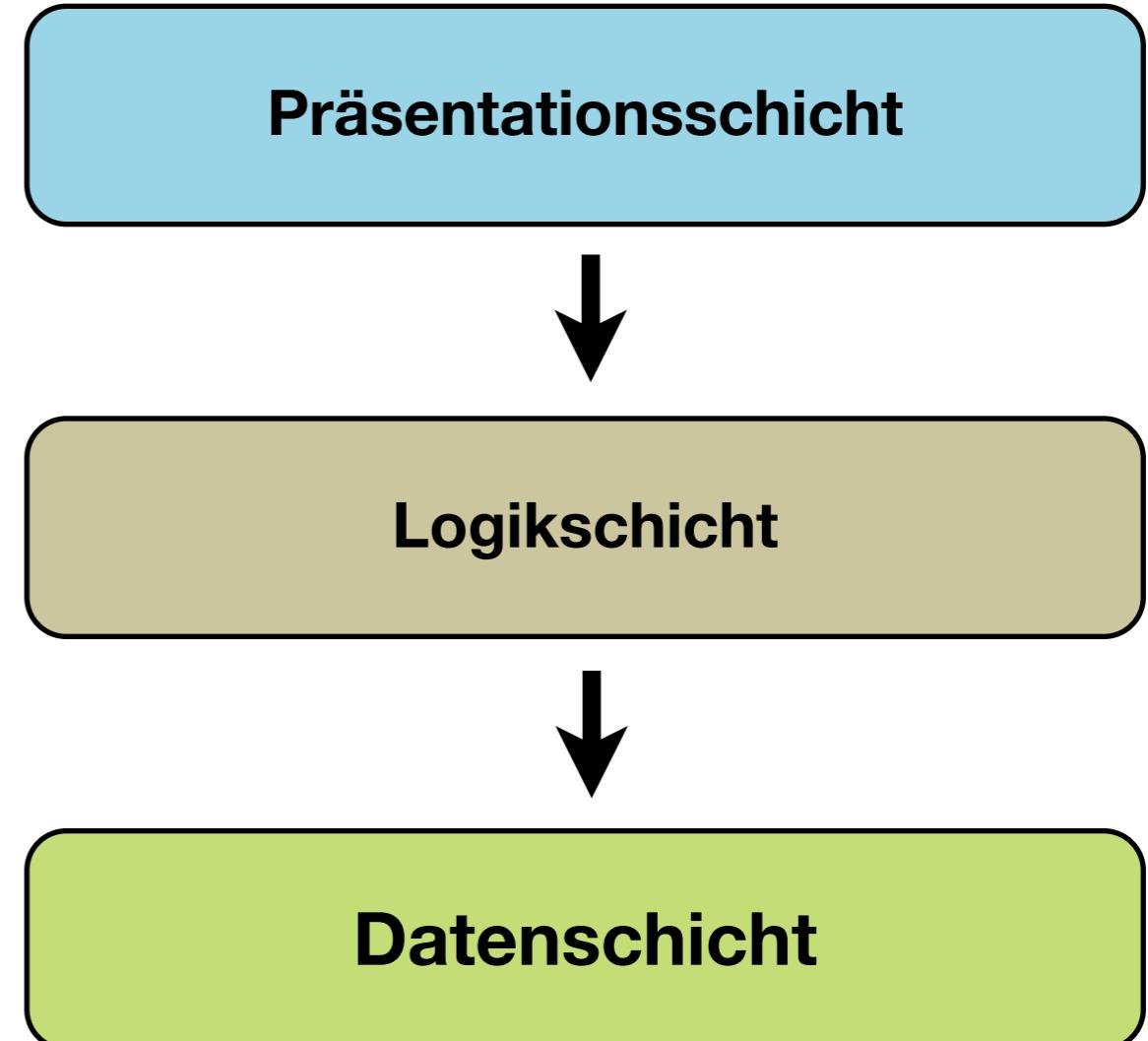
Beispiel: Schichten-Architektur

- Ein besonders häufiger Fall:
3-Schichten-Architektur
 - Die Präsentationsschicht kapselt die Interaktionen mit Benutzern oder Systemen
 - Die Logikschicht enthält die Geschäftslogik
 - Die Datenschicht kümmert sich um die persistente Speicherung aller Daten
- Jedes Modul benutzt nur Module seiner Schicht und der Schichten darunter (und liefert Dienste für die höheren Schichten)
- Höhere Schichten werden von niederen niemals benutzt
- **Vorlage für Internet-Protokoll-Stack (OSI-Modell)**



Beispiel: Schichten-Architektur

- Vorteile:
 - Sehr übersichtliche, aufgeräumte Grobstruktur
 - Kompletaustausch ganzer Schichten ist möglich
 - Unnötige Kopplungen zwischen Modulen werden vermieden
- Nachteile:
 - Kann umständlich oder unnatürlich sein
 - Manche Abstraktionen sind evtl. nur vorhanden, um die Schichtung herzustellen
 - Kann Laufzeit-ineffizient sein
 - wenn man zu viel abstrahiert hat und zu oft "weiterreichen" muss
 - z.B. ISO/OSI 7-Schichtenmodell



Entwurf Design Patterns

- Patterns are abstractions
 - Understanding a pattern reduces a number of elements to a single idea
 - This saves mental resources (7+-2) and simplifies understanding
 - and communication
- Patterns provide reuse
 - If I know the patterns solution previously, I do not have to invent my own solution: Reuse of ideas!
 - The solution idea will always be adapted to the specific context in which the pattern is being used

Design Patterns

- Structural Patterns (Strukturmuster)
 - Adapters, Bridges, Facades, and Proxies are variations on a single theme:
 - They reduce the coupling between two or more classes
 - They introduce abstract classes to enable future extensions
 - They encapsulate complex structures
- Behavioral Patterns (Verhaltensmuster)
 - Here we are concerned with algorithms and the assignment of responsibilities between objects: Who does what?
 - Behavioral patterns allow us to characterize complex control flows that are difficult to follow at runtime
- Creational Patterns (Erzeugungsmuster)
 - Here our goal is to provide an abstraction for a (possibly complex) instantiation process
 - We want to make the system independent from the way its objects are created, composed, and represented

Beispiel: Observer Pattern

Also known as Publish/Subscribe pattern

Problem:

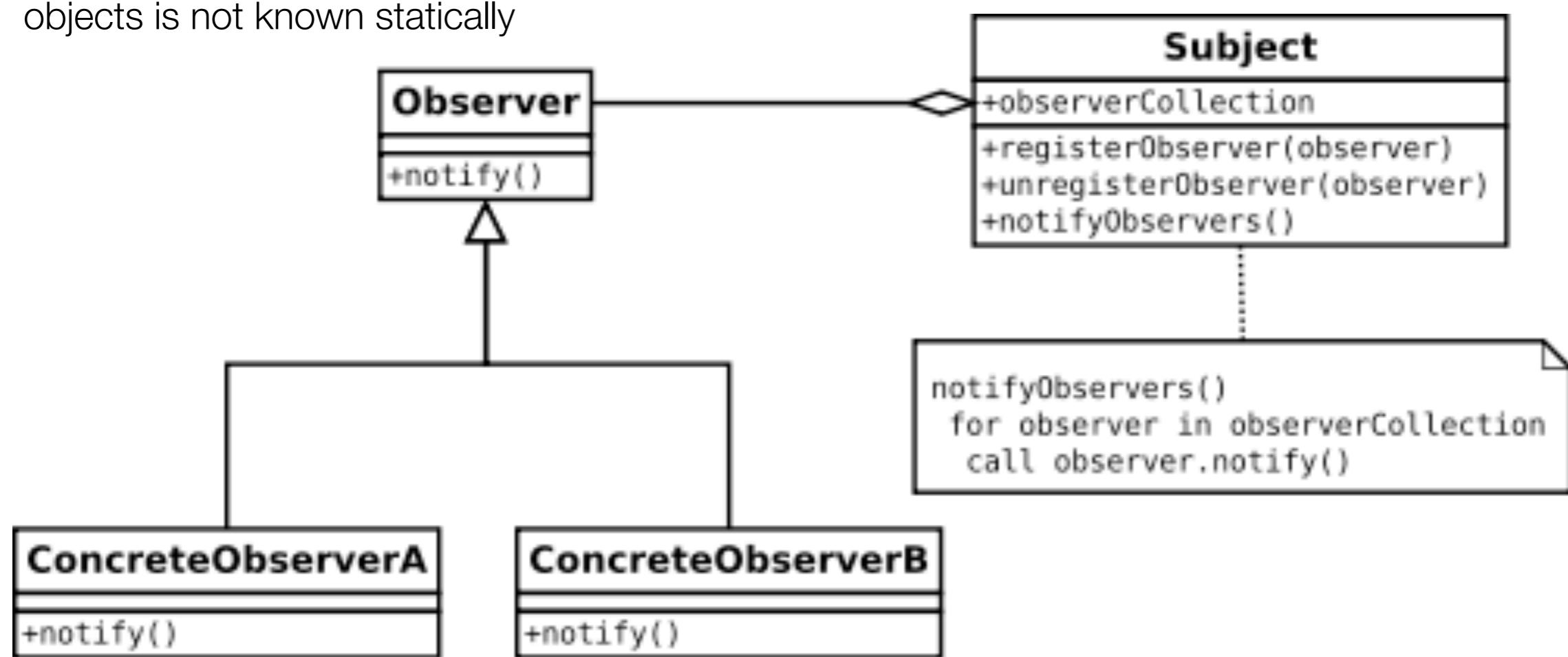
Whenever one particular object changes state, several dependent objects must be modified

- The number and identity of the dependent objects is not known statically

Solution idea:

All dependents provide the same notification interface and register with the state object

- All state objects (called subjects) provide the same registration interface



Observer

9.jpg

Subject

Name	Date Modified	Size	Kind
9.jpg	Dec 3, 2009 9:15 PM	1.7 MB	JPEG image
10.jpg	Dec 3, 2009 9:15 PM	2.3 MB	JPEG image
11.jpg	Dec 3, 2009 9:15 PM	2.6 MB	JPEG image
12.jpg	Dec 3, 2009 9:15 PM	1.5 MB	JPEG image
13.jpg	Dec 3, 2009 9:15 PM	2.4 MB	JPEG image
14.jpg	Dec 3, 2009 9:15 PM	3.4 MB	JPEG image
15.jpg	Dec 3, 2009 9:15 PM	1.3 MB	JPEG image
16.jpg	Dec 3, 2009 9:15 PM	1.2 MB	JPEG image
17.jpg	Dec 3, 2009 9:15 PM	1.3 MB	JPEG image
18.jpg	Dec 3, 2009 9:15 PM	1.5 MB	JPEG image
19.jpg	Dec 3, 2009 9:15 PM	4.1 MB	JPEG image
20.jpg	Dec 3, 2009 9:15 PM	938 KB	JPEG image
21.jpg	Dec 3, 2009 9:15 PM	2.6 MB	JPEG image
22.jpg	Dec 3, 2009 9:15 PM	1.3 MB	JPEG image
23.jpg	Dec 3, 2009 9:15 PM	1.8 MB	JPEG image
24.jpg	Dec 3, 2009 9:15 PM	2.4 MB	JPEG image

Beispiel: Strategy Pattern



Beispiel: Strategy Pattern

Also known as Policy pattern

Problem:

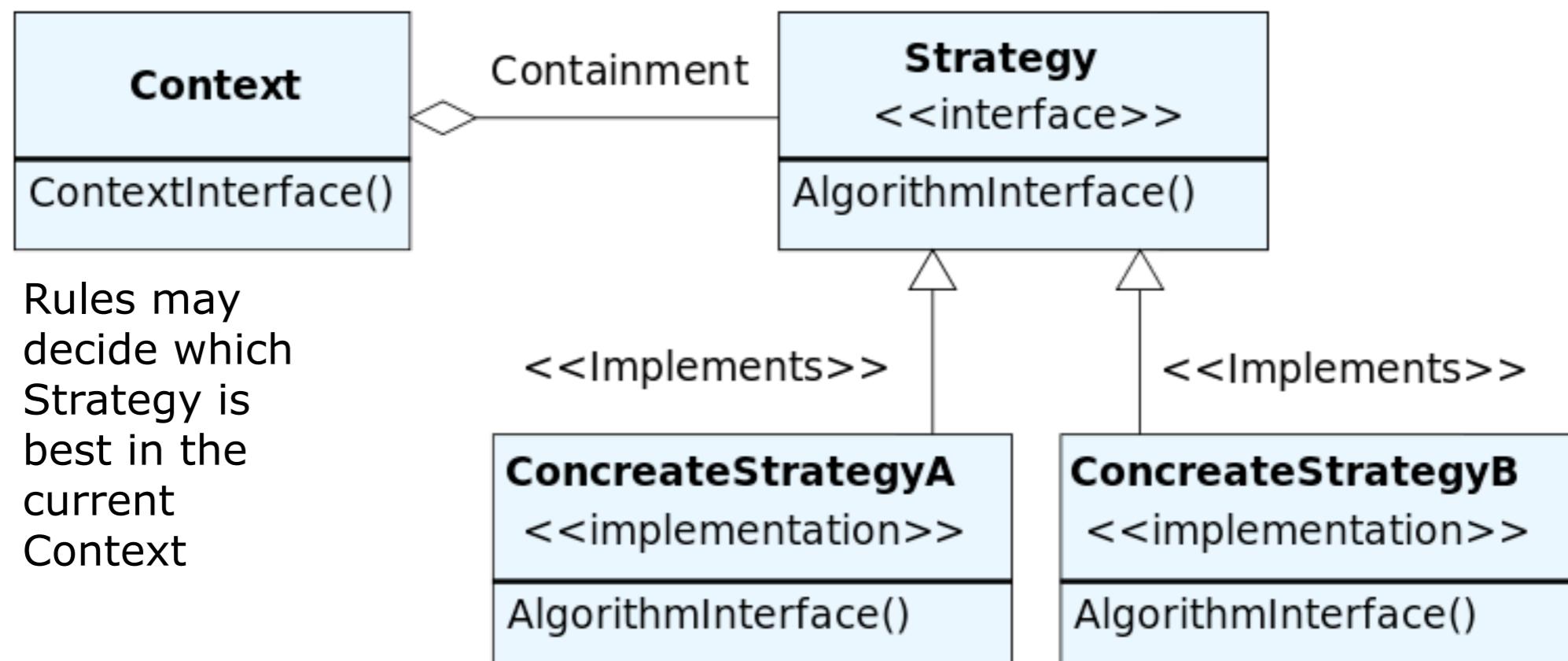
There are multiple ways of doing something.

- We want to add, use, and exchange them freely (perhaps even dynamically), depending on context:
- Different algorithms for identical results (e.g. sorting)

- Different variants of equivalent results (e.g. codecs, file formats)
- Different purposes (e.g. access control policies)

Solution idea:

- Dynamically associate an object whose interface is fixed for all variants.



Welt der Problemstellungen

- Produkt (Komplexitätsproblem)
 - Anforderungen (Problemraum)
 - Entwurf (Lösungsraum)
- Prozess (psycho-soziale Probl.)
 - Kognitive Beschränkungen
 - Mängel der Urteilskraft
 - Kommunikation, Koordination
 - Gruppendynamik
 - Verborgene Ziele
 - Fehler

Welt der Lösungsansätze

- Technische Ansätze ("hart")
 - Abstraktion
 - Wiederverwendung
 - Automatisierung
- Methodische Ansätze ("weich")
 - Anforderungsermittlung
 - Entwurf
 - **Qualitätssicherung**
 - Projektmanagement

Qualitätssicherung





Qualitätssicherung (QS, engl. quality assurance, QA)

- Gesamtheit aller Maßnahmen, die nicht darauf zielen, ein Produkt *überhaupt* fertig zu stellen, sondern darauf, es *in guter Qualität* fertig zu stellen

2 grundlegende Herangehensweisen:

- **Analytische QS:** Prüfend
 - Untersuche (Teil)Produkte nach ihrer Fertigstellung auf Qualität
 - Bessere nach, wo Mängel sind
- **Konstruktive QS:** Vorbeugend
 - Gestalte den Konstruktionsprozess und sein Umfeld so, dass Qualitätsmängel seltener werden
 - Beseitige bei entdeckten Mängeln nicht nur den Mangel selbst, sondern auch seine Ursache(n)

Analytische QS:

- Untersuche jedes Teilprodukt so früh wie möglich
- Je früher ein Mangel entdeckt wird, desto weniger Schaden richtet er an [Zugriff auf "Warum QS?"]
 - z.B. Anforderungsmängel erst nach Auslieferung zu beseitigen kostet oft über 1.000-mal die Kosten der Anforderungsbestimmung
- Analytische QS ist in der SWT Bestandteil aller Aktivitäten
- Beginne damit vor der eigentlichen Entwicklungsarbeit
 - Gestaltung von Organisationsstrukturen
 - Auswahl von Prozessen, Technologie, Strategie
- Lerne aus Projekterfahrungen
- Post-mortem: Zusammentragen von Erfahrungen nach Projektende und Umsetzen in Prozessverbesserungen



Extern / aus Benutzersicht

- Benutzbarkeit
 - Bedienbarkeit, Erlernbarkeit, Robustheit, ...
- Verlässlichkeit
 - Zuverlässigkeit, Verfügbarkeit, Sicherheit, Schutz
- Brauchbarkeit
 - Angemessenheit, Geschwindigkeit, Skalierbarkeit, Pflege, ...

Intern / aus Entwicklersicht

- Zuverlässigkeit
 - Korrektheit, Robustheit, Verfügbarkeit, ...
- Wartbarkeit
 - Verstehbarkeit, Änderbarkeit, Testbarkeit, Korrektheit, Robustheit
- Effizienz
 - Speichereffizienz, Laufzeiteffizienz, Skalierbarkeit

Analytische QS

- Dynamische Verfahren (Test)
 - Defekttest
 - Benutzbarkeitstest
 - Lasttest
 - Akzeptanztest
- Statische Verfahren
 - Manuelle Verfahren
 - Durchsichten
 - Inspektionen
 - Automatische Verfahren
 - Modellprüfung
 - Quelltextanalyse

Konstruktive QS

- Test- und Durchsichtsmgmt.
- Prozessmanagement
- Projektmanagement, Risikomanagement

Analytische QS

Extern / aus Benutzersicht

- Benutzbarkeit
 - Bedienbarkeit, Erlernbarkeit, Robustheit, ...
- Verlässlichkeit
 - Zuverlässigkeit, Verfügbarkeit, Sicherheit, Schutz
- Brauchbarkeit
 - Angemessenheit, Geschwindigkeit, Skalierbarkeit, Pflege, ...

Intern / aus Entwicklersicht

- Zuverlässigkeit
 - Korrektheit, Robustheit, Verfügbarkeit, ...
- Wartbarkeit
 - Verstehbarkeit, Änderbarkeit, Testbarkeit, Korrektheit, Robustheit
- Effizienz
 - Speichereffizienz, Laufzeiteffizienz, Skalierbarkeit



Ziel des Defekttests ist Herbeiführen von **Versagen (failure)** • Also einem falschen Programms

- Falsch im Sinne der Spezifikation (soweit vorhanden), der Anforderungen (wenn klar) oder der Erwartungen (andernfalls)
- Versagen entsteht aufgrund eines **Defekts (defect, fault)** im Programm
- Eventuell führen erst mehrere Defekte gemeinsam dazu
- Nicht jeder Defekt muss überhaupt zu einem Versagen führen
- Ein Defekt entsteht aufgrund eines **Fehlers (error)** der Softwareentwickler
- Ein Fehler ist entweder ein **Falschtun (commission)** oder ein **Versäumnis (omission)**
- Nicht unbedingt beim Kodieren, vielleicht auch bei Anforderungen oder Entwurf
- Fehlern liegt entweder ein **Irrtum (misconception)** oder ein **Versehen (blunder)**



Defekttest: Teilprobleme

- **Wie wählt man Zustände und Eingaben aus?** • Wer wählt Zustände und Eingaben aus?
- Wie wählt man Testgegenstände aus?
- Wie ermittelt man das erwartete Verhalten?
- Wann wiederholt man Tests?
- Wann/wie kann und sollte man Tests automatisieren? • Wann kann/sollte man aufhören?



Wie wählt man Zustände und Eingaben aus?

Vorgehensweisen:

- **Funktionstest (*functional test, black box test*)**
 - Äquivalenzklassen
 - Fehlerfälle
 - Extremfälle
- Strukturtest (*structural test, white box test*)
- Bekannte Versagensfälle
- Allgemeine Erfahrung, Intuition

```
if (x > 0) n  
else if (x < 0) m  
else if (x = 0) l  
else assert
```

Wird ein Versagen im Test nicht aufgedeckt, wird genau dieser Fall (falls reproduzierbar) erkannt.
• Evtl. wird dieser Testfall automatisiert und seine Durchführung künftig nach jeder Änderung wiederholt.

Allgemeine Erfahrung, Intuition

- Jemand mit genug Testerfahrung "riecht" vielversprechende Testfälle
- Solcher Intuition sollte man folgen: Oft viel effektiver als schematische Anwendung von Testfällen
- Außerdem gibt es ein paar oft anwendbare Faustregeln:
 - Leere Eingaben
 - Riesige Eingaben
 - Völlig unsinnige Eingaben



Funktionstest (functional test, black box test)

Wählt Testfälle durch Betrachtung der Spezifikation (Schnittstelle) der Komponente

- Für jeden Fall mit andersartigem Verhalten wähle mindestens einen Testfall
- Gruppen solcher Fälle:
 - Äquivalenzklassen gleichwertiger Eingaben
 - Fehlerfälle
- Heuristik: Randfälle

Funktionstest Beispiel 1: sqrt()

- Spezifikation (java.lang.Math):

```
"public static double sqrt(double a)
```

- Returns the correctly rounded positive square root

- Testeingaben?

- 0 1 Fixpunkte

- -0,164 Fehlerfall

- 0,5 (wird größer) • 1,5 (wird kleiner)

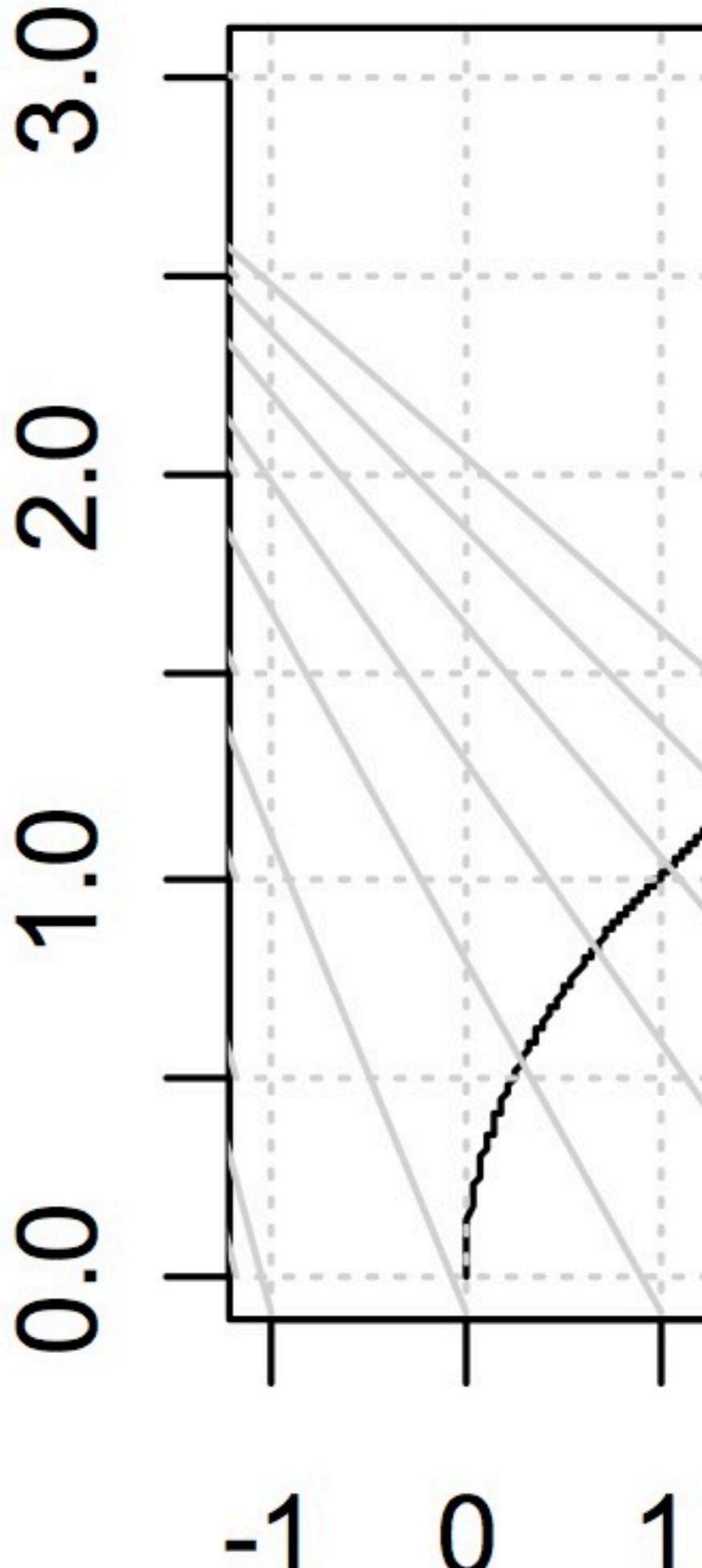
- Randfälle:

- Double.NaN

- Double.POSITIVE_INFINITY

-1 0 1 2 3 4 5

- Beachte: Kein Zustand relevant, sehr klare S





Funktionstest

Beispiel 2: addKeyListener()

- Jede *Component* in einem Java Swing GUI (z.B. ein JButton) kann einen oder mehrere *KeyListeners* haben
 - Ein *KeyListener* ist ein Objekt mit Methoden, die bei Tastendrücken aufgerufen werden:
 - *keyPressed(KeyEvent e)*:
Invoked when a key has been pressed
 - *keyReleased(KeyEvent e)*, *keyTyped(KeyEvent e)* entsprechend
 - Wir testen das Zufügen von *Components*
 - Spezifikation (`java.awt.Component`)
`"public void addKeyListener(KeyListener l)`
 - Adds the specified key listener to receive key events from this component."
 - Testfälle: Ersten *KeyListener* zufügen, weiteren K. zufügen, gleichen K. nochmal zufügen
 - Zerlegung ist bei weitem nicht eindeutig



Funktionstest

Beispiel 2: addKeyListener()

- Testfälle: Ersten *KeyListener* zufügen, weiteren K. zufügen, gleichen K. nochmal zufügen
 - Zerlegung ist bei weitem nicht eindeutig
 - Aber was ist das "erwartete Verhalten"?
- 1. Listener sind zugefügt laut `getKeyListeners()`? oder
- 2. Listener werden aufgerufen, wenn ein Tastendruck passiert?
- Ist die Reihenfolge relevant? Ist sie sichergestellt?
- Was passiert, wenn ich den selben Listener zweimal zugefügt hatte?
- Was passiert (soll passieren), wenn ein Listener eine Ausnahme wirft?
- Ist `null` zufügen erlaubt? Was ist die Moral von der Geschichte?
- Oft wird erst beim Testen die Spezifikation richtig.
- Das ist meist ein Zeichen für einen schlechten Softwareentwurf.
- Denn Klärung beim Entwurf wäre insgesamt billiger gewesen.
- Unbedingt die Dokumentation entsprechend nachbedenken.



Wann/wie kann und sollte man Tests automatisieren?

Was ist Testautomatisierung?

- Testen umfasst:
 1. Testfall auswählen
 2. Testfall durchführen
 3. Ergebnis(se) prüfen
 4. Erfolg oder Versagen feststellen
- Testautomatisierung bedeutet, einen oder mehrere dieser Schritte programmgesteuert auszuführen
 1. Testeingabe erzeugen
 2. Testeingabe abarbeiten
 3. Ergebnisse überprüfen
 4. Resultat protokollieren und ggf. Versagen anzeigen



Warum Testautomatisierung?

- **Mehr Testen:**

Manuelles Testen kann immer nur einen winzigen Teil des Verhaltensraums eines

- Vollautomatisiertes Testen kann den Anteil erhöhen
- wenn auch die Erzeugung der Testfälle mit automatisiert ist

- **Wiederholt Testen:**

Defekte können auch nach Tests in die Software eingefügt werden

- werden also nur gefunden, wenn später erneut getestet wird • **Zuverlässiger Tester**
- Manuelles Testen ist seinerseits fehleranfällig • Oft werden Versagen schlichtweg übersehen



Rückfalltesten (Regressionstesten)

- Automatisiertes Rückfalltesten automatisiert alle Schritte außer der Testf
- Testfälle werden (1) manuell implementiert
- Automatisierung (2) führt aus, (3) prüft Ergebnis und (4) protokolliert
- Sichert Integrität des Systems nach Veränderungen ab
- Paradebeispiel: Refactoring (siehe Agile Prozesse)
- Probleme:
 - 1. Das Implementieren/Automatisieren der Testfälle ist sehr aufwendig
 - Vor allem das Prüfen der Resultate
 - Grobe Faustregel: 10x so viel Arbeit wie eine manuelle Durchführung
 - 2. Rückfalltesten ist nur mäßig wirksam
 - Es gäbe viel mehr Defekte mit neuen Testfällen zu finden als durch die Wiederholung von alte

Achtung: Teuer, lange Amortisationszeit

Test-Werkzeuge

Test Driven Development

Testframework-Ansatz

Testframework =

Anwendungsspezifische Bibliothek von Test-Hilfsoperationen

- Erleichtert stark das Schreiben der Testtreiber
- Automatisiert jeden gewünschten Teilschritt nach Bedarf
- Ermöglicht sauber entworfene und änderungsfreundliche Testsuites
 - für GUIs (mit Aufnahme/Wiedergabe-Werkzeug)
 - für programmiersprachlich
- Problem:
- Hohe Vorab-Investition



Wann kann/sollte man mit dem Testen aufhören?

- Im Prinzip: Wenn die Kosten zum Aufdecken weiterer Defekte den Nutzen, sie erzeugt, übersteigen
 - Konkret kennt man aber weder die Kosten noch den Nutzen • es gibt aber zumindest eine obere Grenze für die Defektanzahl
- Typische Lösungen in der Praxis:
 - Häufig: Test endet, wenn der Zeitplan erschöpft ist
 - bzw. wenn weitere Überziehung nicht mehr akzeptiert wird
 - Manchmal: Test endet, wenn neue Versagen "selten" werden • Sinnvoll, wenn ein Konsens darüber besteht, was "selten" ist
 - Manchmal: Test endet, wenn die SW laut Guru "gut genug" ist • d.h. ein Experte einen Konsens darüber bringen muss, was "gut genug" ist
 - Selten: Test endet, wenn alle Testfälle des Testplans bestanden werden
 - Testplan wird in der Entwurfsphase gemacht und später ergänzt

Extern / aus Benutzersicht

- Benutzbarkeit
 - Bedienbarkeit, Erlernbarkeit, Robustheit, ...
- Verlässlichkeit
 - Zuverlässigkeit, Verfügbarkeit, Sicherheit, Schutz
- Brauchbarkeit
 - Angemessenheit, Geschwindigkeit, Skalierbarkeit, Pflege, ...

Intern / aus Entwicklersicht

- Zuverlässigkeit
 - Korrektheit, Robustheit, Verfügbarkeit, ...
- Wartbarkeit
 - Verstehbarkeit, Änderbarkeit, Testbarkeit, Korrektheit, Robustheit
- Effizienz
 - Speichereffizienz, Laufzeiteffizienz, Skalierbarkeit



- Teil des *Usability Engineering* (Benutzbarkeits-Gestaltung)
- Prüft, ob echte Benutzer in der Lage sind, die Funktionen des Systems zu nutzen
 - und wo dabei Schwierigkeiten auftreten
- Verfahren ist meist die Beobachtung solcher Benutzer
 - bei freier Benutzung
 - beim Lösen vorgegebener Aufgaben
- anschließende Verbesserung der Software
- und erneute Beobachtung
- Usability Engineering ist sehr wichtig und wertvoll
 - wird aber dennoch vielerorts nicht betrieben



Extern / aus Benutzersicht

- Benutzbarkeit
 - Bedienbarkeit, Erlernbarkeit, Robustheit, ...
- Verlässlichkeit
 - Zuverlässigkeit, Verfügbarkeit, Sicherheit, Schutz
- Brauchbarkeit
 - Angemessenheit, Geschwindigkeit, Skalierbarkeit, Pflege, ...

Intern / aus Entwicklersicht

- Zuverlässigkeit
 - Korrektheit, Robustheit, Verfügbarkeit, ...
- Wartbarkeit
 - Verstehbarkeit, Änderbarkeit, Testbarkeit, Korrektheit, Robustheit
- Effizienz
 - Speichereffizienz, Laufzeiteffizienz, Skalierbarkeit



Fragen:

- Leistungstest: Hält das System die nötigen Antwortgeschwindigkeiten ein?
- Wichtig bei zeitkritischen Systemen und Mehrbenutzersystemen
- Lasttest: Kann das System viele Benutzer zugleich schnell genug bedienen?
 - Wichtig bei Mehrbenutzersystemen, insbes. im WWW
 - Stresstest: Überlebt das System auch Überlasten, massenhaft unsinnige Eingaben?
 - Wichtig bei lebens- und geschäftskritischen Systemen
 - Deckt sogar Defekte in Betriebssystemkernen auf („fuzzing“), siehe "Month of the Kernel"
<http://projects.info-pull.com/mokb/>



Akzeptanztest

- Dient dazu, dem Kunden zu demonstrieren, dass das Produkt nun tauglich ist
- Beachte den Wechsel des Ziels:
 - Defekttests und Benutzbarkeitstests sind erfolgreich, wenn sie Mängel aufdecken
 - denn das ist ihr Zweck
 - Akzeptanztests sind hingegen erfolgreich, wenn Sie keine (oder nur geringe) Mängel aufdecken
- Akzeptanztests sollten direkt aus den Anforderungsdokumenten (meist Use Cases) entnommen werden
- Ein kluger Kunde macht das selbst (oder überträgt es Dritten) und legt die Testfälle erst später fest



Gemeinsame Schwäche aller Testverfahren

- Alle testenden Verfahren führen zunächst nur zu Versagen • Das Versagen muss Defekt zurückgeführt werden (**Defektlokalisierung, Debugging**)
 - Siehe oben: Heuristiken für die Defektlokalisierung
 - Das kann sehr **aufwändig** sein:
 1. Das in Frage kommende Codevolumen ist evtl. sehr groß
 2. Evtl. spielen mehrere Defekte zusammen
 3. Oft wirkt der Defekt zeitlich lange bevor man das Versagen sieht
 - In dieser Hinsicht sind **statische und konstruktive Verfahren günstiger**:
 - Die Aufdeckung des Mangels geschieht hier meist direkt am Defekt
 - Die Lokalisierungsphase entfällt deshalb



Manuelle statische Verfahren

- Bei manuellen statischen Verfahren werden SW-Artefakte von Menschen **gelesen** Mängel aufzudecken
- 1. Arbeitsergebnisse werden gründlich auf Mängel hin durchgesehen
 - Mängel können sein: • Defekte
 - Verletzungen von Standards
 - Verbesserungswürdige Lösungen
- 2. Mängel werden zusammengetragen
- 3. Mängel werden beseitigt
 - evtl. nicht alle (wg. Kosten/Nutzen-Abwägung)
- 4. Nach kritischen Korrekturen evtl. erneute Prüfung



Vorteile manueller statischer Verfahren

1. Im Gegensatz zum Test benötigen solche Verfahren keinen ausführbaren Code
 - sondern sind anwendbar auf **alle Arten von Dokumenten**: Anforderungen, Entwürfe, Dokumentationen
 - und sogar auf Prozessdokumente wie Projektpläne u.ä.
2. Dadurch werden Mängel **früher aufgedeckt**, was viel Aufwand spart
3. Außerdem haben die Verfahren **Zusatznutzen** neben der Aufdeckung von Mängeln
 - **Kommunikation**: Verbreitung von Wissen über die Artefakte (und damit verbundene Anwendungsideen) im Team
 - **Ausbildung**: Wissenstransfer über die Dokumente



Kleine Codedurchsicht (halbformell)

Peer Review:

- Entwickler A hat 4 zusammengehörige Klassen fertig gestellt • Und erfolgreich übergeben
- Er sendet Entwicklerin B eine Email und bittet, die 4 Klassen (insgesamt 600 Zeichen) begutachten
- B kennt die Anforderungs- und Entwurfsdokumente, aus denen sich ergibt, was die Klassen tun sollen
- B nimmt sich dafür 3 Stunden Zeit
- B meldet entdeckte Mängel per Email an A zurück:
 - 2 vergessene Funktionen
 - 2 Zweifel an Bedeutung von Anforderungen
 - 4 Fehler in Steuerlogik
 - 5 übersehene Fehlerfälle
 - 4 Vorschläge zur Verbesserung der Robustheit
 - 3 Verstöße gegen Entwurfs-/Kodier-/Kommentierrichtlinien

Geht auch formeller = Inspektion
Informeller: selber durchsichten



Modellprüfung: Zustandsautomaten

- Für voll spezifizierte (Zustands)Automaten kann der gesamte Zustandsraum untersucht werden
- Prüfung von Sicherheitseigenschaften, d.h. ob der Automat etwas tun kann, was er nicht sollte
- d.h. z.B. Beantwortung von Fragen der Art "Kann eine Abfolge A, B auftreten?"
- z.B. für automatische Steuerung eines Containerkrans "Kann ANHEBEN, LOSLASSEN geschehen?" (also ohne ABSINKEN dazwischen)
- Es gibt spezialisierte Programme, die selbst große Zustandsräume (10^{20} Zustände) untersuchen können
- Forschungsgebiet "model checking"



Quelltextanalyse

- Für viele gängige Programmiersprachen
- Defekte oder Schreibfehler erkennen
- z.B. für Java
- dubiose *catch*-Kontrolle
- verdächtige Verwendung von Variablen
- siehe z.B. "FindBugs"
- z.B. für C/C++
- Verletzungen der Sicherheit
- Verletzungen der Performance
- Verdächtige Verwendung von Variablen
- etc.
- Große Unterschiede zu anderen Tools

The screenshot shows a Java code editor with the following code:

```
140
141 int x = Integer.parseInt("42");
142 int
143 int
144 Syst 4 quick fixes available:
145 }
146 |
147 }
```

A yellow tooltip is displayed over the line `int` at line 142, containing the message: "The value of the local variable x is". Below this, a list of "4 quick fixes available:" is shown:

- ✖ Remove 'x' and all assignments
- ✖ Remove 'x', keep side-effect assignments
- @ Add `@SuppressWarnings('unused')`
- @ Add `@SuppressWarnings('unused')`

Below the code editor, the status bar shows: <terminated> export w. At the bottom, two INFO messages are visible:

```
[INFO] Packaging webapp
[INFO] Assembling webapp [usability_analyzer]
```

Konstruktive Qualitätssicherung

- Motto: "Vorbeugen ist besser als Heilen"
- Kann auf einzelnes Produkt hin orientiert sein:
 - **Projektmanagement**
 - Meist recht pragmatischer Ansatz ("Vorbeugen ist besser als auf die Füße kotzen")
 - Siehe spätere Präsentationen
- oder auf die Verbesserung des Prozesses einer Organisation über alle Projekte hinweg:
 - **Prozessmanagement**
 - abstrakter; längerfristige Ausrichtung; Gefahr des "Verkünstelns"
 - Dazwischen gibt es, quasi als Rohstoff für beides, **Prozessmodelle**
 - die allgemein das Zusammenwirken von Rollen und Tätigkeiten zu einem Softwareprodukt darstellen
 - und projektunabhängig dessen grobe Leitlinien festlegen



Gemeinsamkeiten von Projekt- und Prozessmanagement

- Leitlinien:

1. Gestalte den Konstruktionsprozess und sein Umfeld so, dass Qualitätsmängel seltener werden
2. Beginne damit vor der eigentlichen Entwicklungsarbeit
 - Gestaltung von Organisation und Arbeitsumfeld
 - Auswahl von Prozessen, Technologie, Strategie
3. Beseitige bei entdeckten Mängeln nicht nur den Mangel selbst, sondern auch seine Ursachen

- Urgrundanalyse (root cause analysis)
- Vorgehensweise:
- Vorstrukturierung des Arbeitsprozesses vorgeben
- Produktvorgaben: Standards, Schablonen
- Prozessvorgaben: Rollen- und Ablaufbeschreibungen
- Laufenden Prozess überwachen
- idealerweise quantitativ: Einhaltung der Vorgaben; Qualität

Extern / aus Benutzersicht

- Benutzbarkeit
 - Bedienbarkeit, Erlernbarkeit, Robustheit, ...
- Verlässlichkeit
 - Zuverlässigkeit, Verfügbarkeit, Sicherheit, Schutz
- Brauchbarkeit
 - Angemessenheit, Geschwindigkeit, Skalierbarkeit, Pflege, ...

Intern / aus Entwicklersicht

- Zuverlässigkeit
 - Korrektheit, Robustheit, Verfügbarkeit, ...
- Wartbarkeit
 - Verstehbarkeit, Änderbarkeit, Testbarkeit, Korrektheit, Robustheit
- Effizienz
 - Speichereffizienz, Laufzeiteffizienz, Skalierbarkeit

Analytische QS

- Dynamische Verfahren (Test)
 - Defekttest
 - Benutzbarkeitstest
 - Lasttest
 - Akzeptanztest
- Statische Verfahren
 - Manuelle Verfahren
 - Durchsichten
 - Inspektionen
 - Automatische Verfahren
 - Modellprüfung
 - Quelltextanalyse

Konstruktive QS

- Test- und Durchsichtsmgmt.
- **Prozessmanagement**
- Projektmanagement, Risikomanagement

Prozessmanagement: Grundidee

- Ziel von Prozessmanagement ist letztlich die lernende, sich selbst kontinuierlich Organisation
- Aus der Vogelschau soll jedes Projekt wie folgt ablaufen (*Quality Improvement Loop*)
 1. Charakterisiere Projektumgebung (**Characterize**)
 - → Welcher Projekttyp? Welche passenden Modelle existieren?
 2. Setze quantifizierbare Ziele (**Set Goals**)
 - → Was soll erreicht werden?
 3. Wähle die geeignete Vorgehensweise aus (**Choose process**)
 - → Wie soll das Ziel erreicht werden?
 4. Führe Projekt aus (**Execute**)
 - → mit QS konstruktiv und analytisch
 5. Analysiere Projektergebnisse (**Analyze**)
 - → Was hat funktioniert? Was nicht?
 6. Ergänze Erfahrungsdatenbank (**Package**)
 - → Welche Erfahrungen/Modelle sollen wiederverwendet werden?



Prozessmanagement: Praxis

- Allerdings braucht eine Organisation die meisten Erfahrungen nicht selbst zu machen
- sondern kann allgemein bekannte Erfahrungen wiederverwenden und für sich anpassen
- Deshalb gibt es zahlreiche Leitlinien dafür, worauf ein Prozessmgmt. generell basiert



- CMM-SW: Capability Maturity Model for Software
 - Entwickelt in den 1980er Jahren am SW Engineering Institute (SEI) in Pittsburgh, USA
 - inzwischen **CMMI**: Capability Maturity Model Integration
 - Entwicklungsansatz: Welche Prozesseigenschaften minimieren das Fehlschlagsrisiko und maximieren die erwartete Qualität?
 - Unterscheidet **5 Reifelevels**
- 1 Initial (initialer Prozess):
 • ad-hoc: SW-Entwicklung erfolgt "irgendwie"; Erfolg ist nicht erklärbar
 • 2 Repeatable (wiederholbarer Prozess):
 • Planung, I
 • 3 Define
 • Konstrukt
 • 4 Manag
 • Durch Me
 • 5 Optimi
 • Kontinuierl

Prinzipieller Aufbau der CMM Modells

Σ: 5 – 13 Jahre
 1 - 3 Jahre

1 - 3 Jahre

Stufe 5	Prozess-Charakteristiken
Optimierender Prozess (optimizing)	Rückgekoppelter Prozess; quantitative Basis für kontinuierliche Verbesserung
Stufe 4	Prozess-Charakteristiken
Gesteuerter Prozess (managed)	Quantitativer Prozess; ganz gute quantitative Kontrolle über die Produktqualität; Prozess durch Metriken gesteuert
Stufe 3	Prozess-Charakteristiken
Refinierter Prozess	Qualitativer Prozess; Zentralisierung
Kontinuierlicher Prozess	Organizational



CMM-SW: Stufen aus Managersicht

Wie viel und welchen Einblick hat ein Projektmanager in den Projektstatus?

- Initial (initialer Prozess):
 - fast keinen: Prozess ist weitgehend undurchsichtig;
 - Erfolge und Mißerfolge sind gleichermaßen überraschend
 - Repeatable (wiederholbar)
- Gültiges Anforderungsdokument; Einblick punktuell über Meilensteine, dazwischen nichts
- Defined (definierter Prozess):
 - Zusätzlich: Qualitätsinformation aus Durchsichten; wohldefinierter (also verständlicher) Prozess
 - Managed (gesteuerter Prozess):
 - Kontinuierliche Einsicht durch Messungen
 - Optimizing (selbstverbessernder Prozess):
 - Einblick sowohl in Projektschwächen als auch Prozessschwächen



CMM: Probleme und Risiken

- Prozessverbesserung ist aufwendig
- Es sind Investitionen nötig, die sich nicht sofort auszahlen • Der Aufstieg um eine CMM-S 3 Jahre
- Kleine Organisationen müssen vom CMM abweichen
 - und zum Teil eigene Lösungen zur Prozessverbesserung finden
 - da einige vorgesehene Prozesse und Institutionen nur für große Organisationen tragbar s
- Hohe Reifestufen können risikoscheu machen (Tom DeMarco)
 - Bei einem sehr innovativen Projekt (viel radikales Vorgehen) sind - bisherige Messungen aussagekräftig und
 - definierte Prozesse oft nicht anwendbar
 - Deshalb fällt eine Organisation dann meist auf Stufe 2 zurück
 - Falls Reifestufen zum "Wert an sich" geworden sind, wird die Organisation tendenziell vor zurücksschrecken



ISO 9000

- Normen für zertifiziertes Qualitätsmanagement
- Für SW relevant: ISO 9001 und ISO 9000-3
- Beschreibt Mindestanforderungen an ein Qualitätsmanagement
- Grobe Norm: Nur einige Dutzend Seiten
- Zertifikat wird von Auditor vergeben und ist 3 Jahre gültig
- ISO 9000-3: Inhaltsrichtlinien für diverse Dokumentarten
- Hauptsächlich Dokumentation
- CMM-SW hat 600 Seiten!
- Vertrag zwischen Auftraggeber und Lieferant
- Spezifikation
- Qualitätssicherungsplan
- Wartungsplan
- Konfigurationsmanagementplan

Analytische QS

- Dynamische Verfahren (Test)
 - Defekttest
 - Benutzbarkeitstest
 - Lasttest
 - Akzeptanztest
- Statische Verfahren
 - Manuelle Verfahren
 - Durchsichten
 - Inspektionen
 - Automatische Verfahren
 - Modellprüfung
 - Quelltextanalyse

Konstruktive QS

- Test- und Durchsichtsmgmt.
- Prozessmanagement
- Projektmanagement, Risikomanagement

Wo sind wir?: Prozessmodelle

- Einsicht: Man sollte die Gesamt-Vorgehensweise nicht in jedem Projekt neu erfinden
 - sondern sich auf vorhandene Erfahrungen abstützen
- Prinzipien:
 - **Planung und Koordination:**
 - Es senkt das Risiko, wenn alle Beteiligten im Voraus erkennen können, was wann getan werden muss
 - Es ist schwierig, dies im Vorhinein korrekt abzuschätzen
 - **Korrekturen:** Versuche, den Prozess so zu gestalten, dass die unvermeidlich auftretenden Fehler ausgeglichen werden können
 - **Iteration:** Es senkt das Risiko, wenn das Projekt in kurzen Abständen evaluierbare Versionen hervorbringt



Die wichtigsten Entscheidungen für ein SW-Projekt

- Projektziele
- Zeitplan und Budget
- Projektorganisation
- **Verwendetes Prozessmodell**
- Verwendete Technologie, Werkzeuge und Methoden
- Teammitglieder



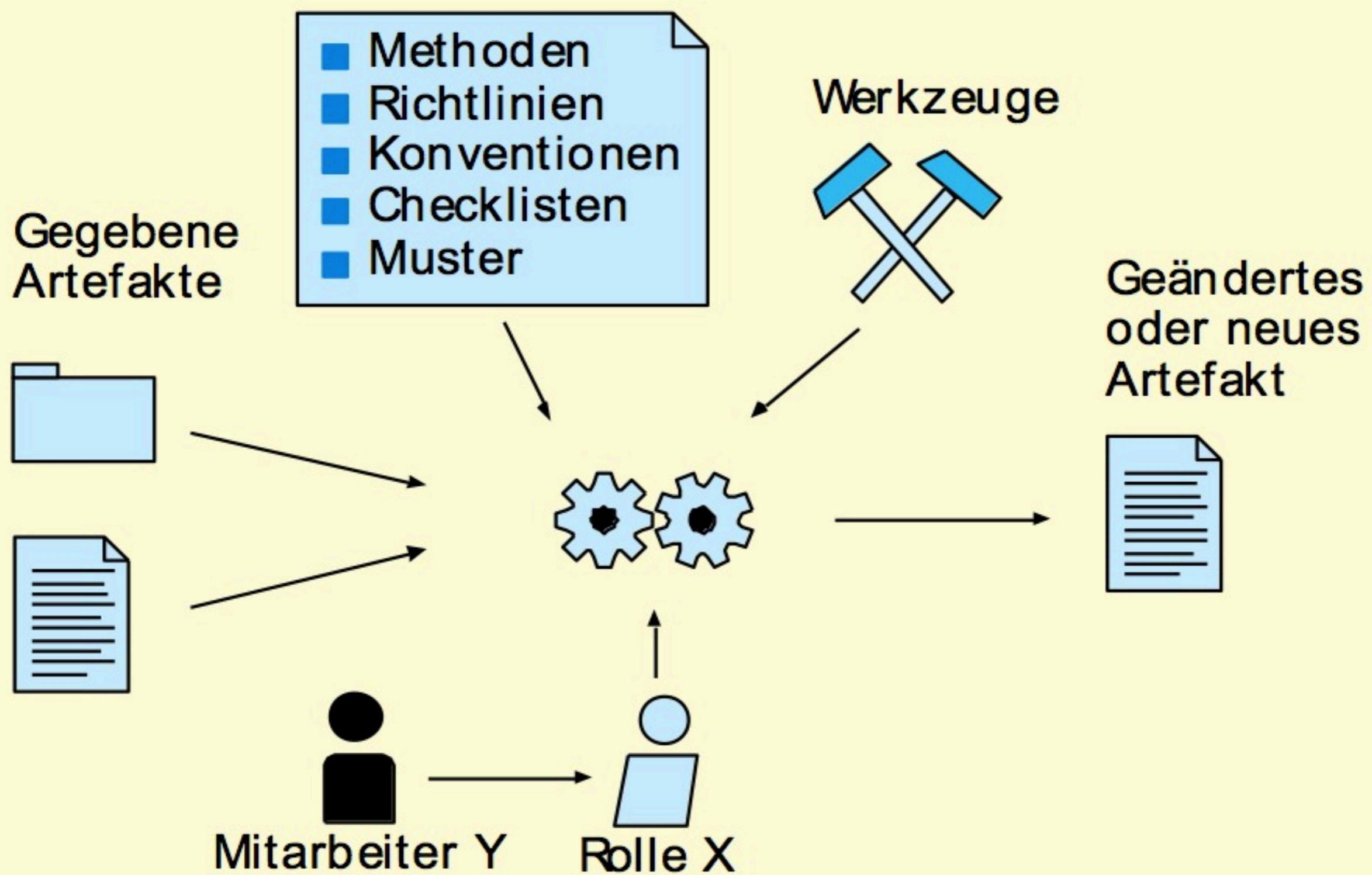
Definitionen: Prozess, Prozessmodell

Software-Prozess:

- Die Abläufe, die in einem Softwareprojekt geschehen
- entweder *deskriptiv* gemeint
(also beschreibend, was tatsächlich geschieht)
- oder *präskriptiv* (also als Vorschrift, wie es abzulaufen hat)
- auch Teile des Gesamtprozesses werden oft Prozess genannt
- z.B. der Testprozess

Softwareprozess-Modell (Software-Prozessmodell):

- Eine *Schablone* die die Gemeinsamkeiten der Abläufe in vielen ganz verschiedenen erfasst
- meistens präskriptiv gemeint





Der Urahn: Das Wasserfallmodell

- Es gibt eine kleine Zahl verschiedener Aktivitäten, z.B.
 - 1. Planung
 - 2. Anforderungsbestimmung
 - 3. Architekturentwurf
 - 4. Feinentwurf
 - 5. Implementierung
 - 6. Integration
 - 7. Validierung
 - 8. Inbetriebnahme
- Diese Aktivitäten werden der Reihe nach durchlaufen ("Phase")
 - Jede Phase nur einmal
 - Phase N beginnt nach Abschluss von Phase N-1
- Dokumenten-getriebener Prozess
 - alle Ergebnisse jeder Phase liegen in Dokumenten vor
- Am Ende jeder Phase erfolgt eine gründliche Prüfung der Ergebnisdokumente
 - und dann die Übergabe in die nächste Phase ("Meilenstein")
 - eventuell mit anderem Personal!
- Annahme:
 - Mängel in Phase N werden spätestens in Phase N+1 aufgedeckt
 - und können dann leicht in den Dokumenten beider Phasen korrigiert werden





Schwächen des Wasserfallmodells

1. Bei unklaren Anforderungen:

- Wenn Anforderungen in der Anforderungsbestimmung nicht gut verstanden werden, braucht man als Hilfe Implementierung und die Validierung
- Im Wasserfallmodell führt das zu Chaos, weil späte Änderungen der Anforderungen total das Prozessmodell entweder die Arbeitsweise mit gründlich ausgearbeiteten Dokumenten wird enorm teuer
- oder die Dokumente werden nicht mehr korrekt gepflegt

2. Bei veränderlichen Anford.:

- Das gleiche gilt, wenn sich Anforderungen irgendwann im Projektverlauf plötzlich von außen ändern können

3. Bei nicht beherrschten Architekturen:

- Ähnlich wie bei unklaren Anf.

4. Durch "Über die Mauer werfen":

- Kommunikation nur über Dokumente
 - Desaster, wenn Dokumente nicht gelesen werden
 - Verschiedenes Personal
 - Verständnis für Phänomene von Phase N ist in N+1 weitgehend verloren
- Eigentlich ist das Wasserfallmodell nur eine Legende.



1. Reparatur: Iteration

- Modernere Prozessmodelle empfehlen ein iteratives Vorgehen • Projektergebnis nicht "Rutsch" anfertigen,
sondern sich in mehreren Schritten "herantasten"

Vorteile:

- Senkt Komplexität in einzelnen Schritt (→ beherrschbarer)
- Kann mit unklaren oder veränderlichen Anforderungen etc. umgehen
- Verlangt engere Kommunikation der Beteiligten
- und senkt dadurch die Neigung zum "Über die Mauer werfen"

Nachteil:

- Bewirkt eine gewisse Doppelarbeit (wg. "Zwischenlösungen") und hat deshalb theoretisch höheren Aufwand
- Verlangt engere Kommunikation der Beteiligten



Beispiele iterativer Prozessmodelle

- Prototypmodell:
 - Baue anfangs ein (Teil)System "zum Wegwerfen", um kritische Anforderungen besser zu verstehen
 - Danach Wasserfallmodell
- Inkrementelles Modell:
 - Baue das Gesamtsystem schrittweise
 - In jedem Schritt werden nur neue Teile hinzugebaut, aber es wird (theoretisch) nie etwas Existierendes verändert
- Iteratives Modell (evolutionäres Modell):
 - Baue das Gesamtsystem schrittweise
 - In jedem Schritt werden neue Teile hinzugebaut und wo nötig auch existierende verändert
- Spiralmodell (Risikomodell):
 - Tue in jeder Iteration das, was am stärksten zur Verringerung des kritischsten Projektrisikos beiträgt
 - Hat alle obigen Modelle als Spezialfälle



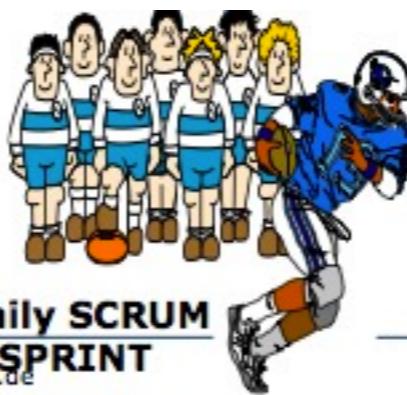
2. Reparatur: Allgemeine Ziele anstatt konkreter Pläne

- Manche Prozessmodelle planen nicht gleich den Inhalt aller Iterationen von Anfang an
 - sondern geben nur grobe Ziele der Entwicklung vor
 - Dies verbessert insbesondere die Anforderungsänderungen zu akzeptieren Bezeichnung solcher Prozessmodelle:
 - Agile Prozesse (oder agile Methoden)
- Agile Methoden verlangen sehr enge Kommunikation!



Hauptunterschied zwischen Prozessmodellen: Wieviel Planung?

- Konkrete Prozessmodelle unterscheiden sich in vielerlei Hinsicht
- Wichtigste Dimension von Unterschieden: Wie präzise/strikt/weit_voraus wird
- Sehr: Wasserfallmodell
 - möglichst präzise und strikt, für das gesamte Projekt im Voraus
- Mittel: Iterative Modelle
 - Planen so weit und so präzise wie möglich
 - nicht strikt (nötige Veränderungen werden akzeptiert) • nur für wenige Iterationen im Voraus
- Wenig: Agile Prozesse
 - Nur so viel Planung wie unbedingt nötig • Lieber Ziele als Pläne (wg. Flexibilität)



ad hoc

- + wenig Planungsaufwand
- + individuelle Freiheit
- - Ergebnis unvorhersehbar - - abhängig von "Helden"

feingranulare Verträge

- ++ klare Arbeitsgrundlage + finanzielle Sicherheit
- - enorm aufwändig

Agiles Beispiel: SCRUM

- + mittelfristig geplant
- + reagiert schnell
- hängt an MA-Qualifikation
- Endprodukt nicht spezifiziert



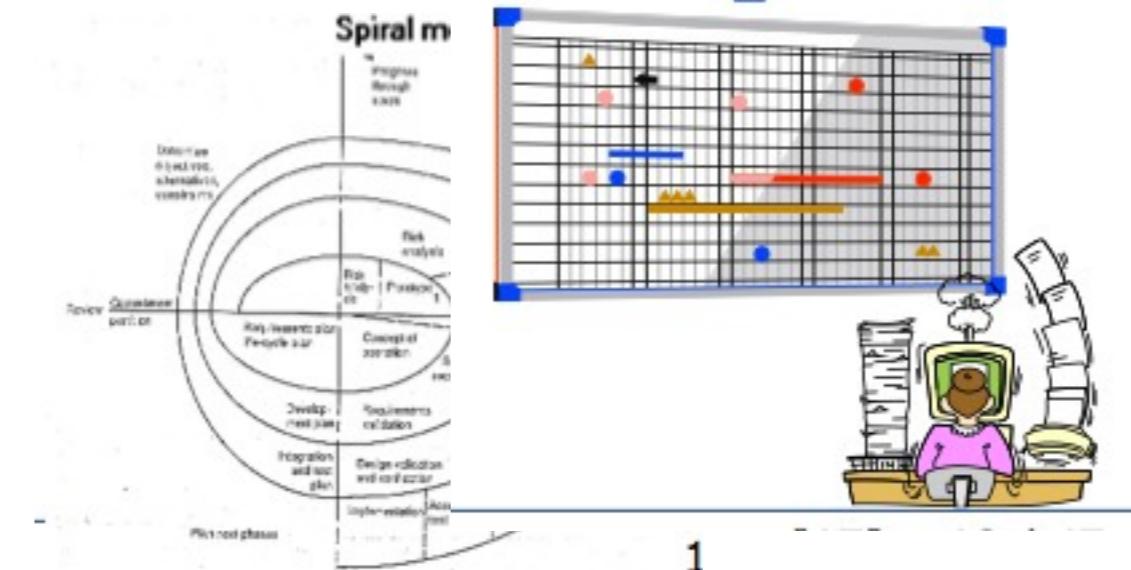
eXtreme Programming

- + früh Knergebnisse
- + flexibel für Änderungen - in sehr großen Projekten Zusatzplanung nötig
- viel Selbstdisziplin nötig



Meilensteinorientiertes Entwickeln

- ++ Risiken aktiviert langfristigen Vorhersagen
- kaum langfristig Änderungen aufwändig
- relativ aufwändig für realistische Annahmen





Literatur

- Wasserfallmodell:
 - <http://c2.com/cgi/wiki?WaterFall>
- Spiralmodell:
 - Barry Boehm: "A Spiral Model of Software Development and Enhancement", IEEE Computer 21(5):61-72, 1988.
- Royce's Method:
 - Walker Royce: "Software project management: A unified framework", Addison-Wesley Longman, 1998
 - <http://www.ibm.com/software/awdtools/rmc/index.html>
- V-Modell XT:
 - <http://www.v-modell-xt.de>
- XP:
 - Kent Beck, Cynthia Andres: "Extreme Programming Explained: Embrace Change", 2nd ed., Addison Wesley, 2004
 - 2. Ausgabe: hat andere Liste von Praktiken als die oben vorgestellten
 - www.extremeprogramming.org, www.xprogramming.com

Analytische QS

- Dynamische Verfahren (Test)
 - Defekttest
 - Benutzbarkeitstest
 - Lasttest
 - Akzeptanztest
- Statische Verfahren
 - Manuelle Verfahren
 - Durchsichten
 - Inspektionen
 - Automatische Verfahren
 - Modellprüfung
 - Quelltextanalyse

Konstruktive QS

- Test- und Durchsichtsmgmt.
- Prozessmanagement
- **Projektmanagement,**
Risikomanagement

Projektmanagement



Ähnlich für versch. Aufgaben (SW-Entwurf, pmi.org):

- "A project is a temporary endeavour, undertaken to create a unique product or service." (PMI)
- bitte nicht Projekt mit Produkt verwechseln
- "Project Management is the application of knowledge, skills, tools and techniques to project activities to meet the requirements of the stakeholders."
- "Project management includes
 - (1) identifying requirements,
 - (2) establishing clear objectives,
 - (3) balancing the competing demands for time, cost and quality, andan approach to the different concerns and expectations of the various stakeholders."
- DIN 69901:
 - "Gesamtheit von Führungsaufgaben, -organisation, -techniken und -mitteln für die Abwicklung eines Projektes"

Project Management

Das magische Dreieck

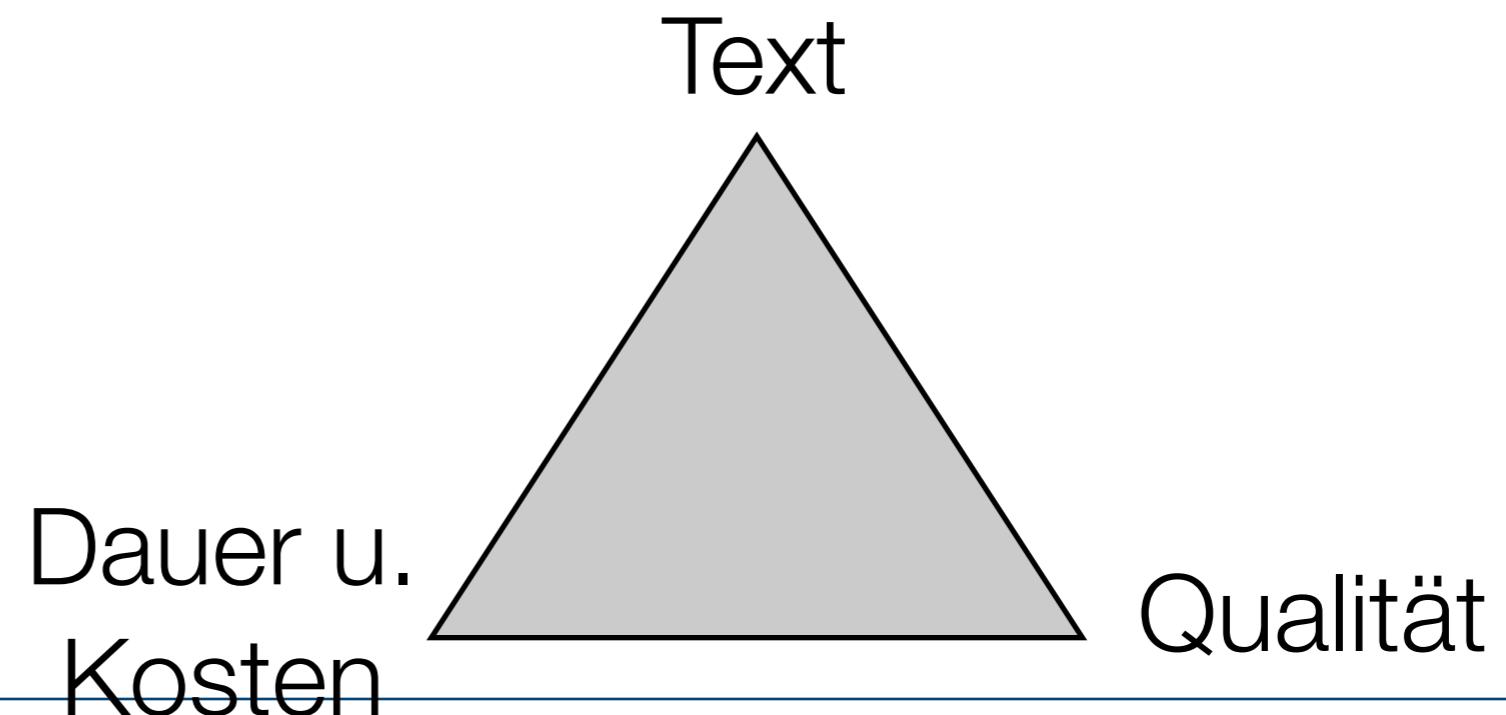


Klassisches Projektmanagement versucht, widerstreitende Kräfte zu balancieren: **Qualität, Kosten, Projektdauer**

- höhere Qualität verlangt meist längere Zeit und/oder erhöht Kosten etc.

Umfang

- Im SW-Projektmgmt. spaltet man Qualität oft auf in
- Umfang (Funktionalität) und
 - ist bei SW leichter unterwegs reduzierbar als z.B. bei einer Brücke
- Qualität (z.B. Zuverlässigkeit, Bedienbarkeit)
- Dagegen hängen die Kosten eng mit der Projektdauer zusammen (Personalaufwand)
- und treten weniger als separate Entscheidungsgröße auf
- Deshalb ist bei SW-Projekten das Dreieck meistens **Umfang, Qualität, Projektdauer**
- manchmal sieht man auch ein Viereck





Aufgabenfelder des Projektmanagements (PM)

Laut Project Management Institute besteht PM aus den folgenden 9 Aufgabenfeldern:

1. Integrierende Aufgaben
2. Umfangsmanagement
3. Zeitmanagement
4. Kostenmanagement
5. Qualitätsmanagement
6. Personalmanagement
7. Kommunikationsmgmt.
8. Risiko mgmt.
9. Beschaffungsmgmt.

(integration mgmt) (scope mgmt)

(time mgmt)

(cost mgmt)

(quality mgmt)

(human resource mgmt) (communication mgmt) (risk mgmt) (procurement mgmt)

- Es folgt eine Kurzübersicht über deren Themen
- anschließend betrachten wir ein paar Fragestellungen genauer



Vorbemerkung: Wann wie viel?

- Viele der nachfolgend beschriebenen Aufgaben sind nur für Großprojekte im vollen sinnvoll
- Bei mittelgroßen Projekten sollten viele davon vereinfacht werden
 - aber welche und wie stark hängt vom Einzelfall ab
- Bei kleinen Projekten sollten evtl. sogar manche ganz entfallen
 - aber ob und welche hängt vom Einzelfall ab
- Übertriebenes Projektmanagement ist schädlich! Zu wenig Projektmanagement ist a
- Augenmaß ist gefragt!
- (Das ist allerdings ohne Erfahrung ziemlich viel verlangt...)



Integrierende Aufgaben (integration mgmt)

- Diese Tätigkeiten verbinden die Tätigkeiten der restlichen Felder miteinander
- Entwickeln des umfassenden Projektplans
 - enthält Teilpläne aus den anderen Aufgabenfeldern
 - Anmerkung: Ein solcher Plan existiert immer (evtl. nicht detailliert, evtl. nur in Köpfen; evtl. sogar vage)
- Projektleitung
 - konkrete Anweisungen geben, Entscheidungen fällen etc.
- Projektüberwachung
 - kontinuierlich den Ablauf gegenüber den Plänen verfolgen, • bei Abweichungen geeignete Maßnahmen ergreifen
- Planänderungs-Überwachung und -Steuerung
 - Änderungen am Plan auf Wirkungen abklopfen und entweder zurückweisen oder komplett ins einfädeln



Umfangsmanagement (scope mgmt)

- Umfangsdefinition
- Was ist Aufgabe des Projekts? Was nicht? • BeiSW: → Anforderungsbestimmung
- Erarbeiten einer Arbeitszerlegung (WBS: Work Breakdown Structure)
- In welche handhabbaren Teile sollte man das Gesamtvorgehen (d.h. den konkreten Prozess) unterteilen?
- Wie fügen sich diese Teile zum ganzen Vorgehen zusammen?
- Bei SW hauptsächlich: → Entwurf + Prozessmodell
- Umfangsverwaltung(beiSW: → Anforderungsverwaltung)
- Änderungen am Umfang (durch externe Einflüsse, z.B. Kundenwunsch) werden nicht einfach vorgenommen, sondern die Auswirkungen überprüft.
- Akzeptierte Änderungen werden sorgfältig in die Umfangsbeschreibung eingearbeitet
- und Beteiligte geeignet benachrichtigt



Zeitmanagement (time mgmt)

- Entwickeln einer Aktivitätenliste
- Zu jedem Teilprodukt laut WBS gehören eine oder mehrere Aufgaben (Aktivitäten, also Prozesse)
- (Zeit)Aufwandsschätzung für die Aktivitäten
 - Wie hoch ist der Aufwand (z.B. Personentage)?
 - Wie lange dauert die Erledigung (Kalendertage)?
- Aufstellung eines Zeit- und Arbeitsplans (schedule)
 - Wie hängen die Aufgaben von einer zu anderen ab?
 - Wer erledigt von wann bis wann welche Aktivität?
- Zeitplanüberwachung (schedule control)
 - Wird der Plan eingehalten?
 - Kontinuierliche Fortschreibung des Zeitplans



Kostenmanagement (cost mgmt)

- Kostenschätzung
- Budgetaufstellung
- Kostenüberwachung
- Bei SW-Projekten sind außer den Personalkosten meistens nur wenige Posten relevant (Lizenzen, HW, Räume etc.)
- Dadurch hängen die Kosten so eng an den Zeitaufwänden, dass diese Aufgabe kaum gelöst werden muss



Qualitätsmanagement (quality mgmt)

- Qualitätsplanung (quality planning)
- Aufstellen von Qualitätsanforderungen
(bei SW: → Anforderungsbestimmung)
- Planen, wie man sie erfüllt
(bei SW ungefähr: → Qualitätsmanagement (=konstruktive QS))
- Qualitätssicherung (qua
- bei SW: → (analytische) Qualitätssicherung
- Qualitätssteuerung (quality control)
- wenn die Qualität nicht stimmt, geeignete Maßnahmen ergreifen
- z.B. korrigieren, Teil wegwerfen und neu bauen, Aufgabe an andere Person übergeben, Entwändern, etc.



Personalmanagement (human resource mgmt)

- Personalplanung
- Definition von Rollen, Verantwortlichkeiten, Weisungsbefugnissen
(→ Prozessmodell & Organisationsplanung)
- Aufstellung eines Personalzeitplans
 - Wie viele Leute mit welcher Qualifikation brauchen wir von wann bis wann?
 - z.B. Analysten überwiegend am Anfang des Projekts, Programmierer jedoch nicht von Anfang an
- Team einwerben, Teamformung
 - Teilnahme an einem Projekt sollte freiwillig sein
 - Mitglieder müssen ein Gemeinschaftsgefühl entwickeln
- Teamführung
 - Leistung der Mitglieder verfolgen, Feedback geben, Konflikte auflösen, Arbeit bei Änderungen koordinieren



Kommunikationsmanagement (communication mgmt)

- Kommunikationsplanung
- Welche Beteiligten brauchen regelmäßig welche Information?
- Informationsverteilung
- Den Betroffenen relevante Information stets zügig zukommen lassen
- Fortschrittsberichte
- Statusberichte und Planfortschreibung regelmäßig erarbeiten und allen Betroffenen zuleiten
- Interaktion mit Beteiligten
- Besprechungen und Schriftverkehr mit allen Beteiligten (z.B. Projektteam, Auftraggeber, Anwalt, Interessierte), um deren Bedürfnisse zu erfüllen und Angelegenheiten aller Art zu klären



Risikomanagement (risk mgmt)

- Risiken identifizieren
- Welche Ereignisse könnten die plangerechte Durchführung bedrohen?
- Risikoeinschätzung
- Risikogrößen abschätzen, Risikobehandlung priorisieren
- Was wird getan, um welchem Risiko vorzubeugen?
- Was wird getan, wenn welches Risiko eintritt?
- Kontinuierlich nach neuen oder veränderten Risiken Ausschau halten
- Fortführung und Wirkung eingeleiteter Vorbeugungen und Gegenmaßnahmen überwachen. G eingreifen
- Vorbeugung planen
- Gegenmaßnahmen planen
- Risikomanagement-Überwachung

Tim Lister: "Risikomgmt. ist Projektmgmt. für Erwachsene"

Beschaffungsmanagement (procurement mgmt)

- Planung und Durchführung der Beschaffung aller Dinge, die das Projekt braucht, abherstellt
- Betrifft bei SW-Projekten z.B. Möbel, Hardware, SW-Lizenzen • Ist aber meist nicht kompliziert



Zeitplanung • Schätzung des Gesamtaufwands

- Aufstellen eines Zeit- und Arbeitsplans • Risikomanagement
- Risikoermittlung und -einschätzung • Riskovorbeugung und -behandlung
- Personalmanagement
- Gruppen und Teams
- Psychologische Effekte
- Integrierende Aufgaben • Projektplan
- Projektleitung und -überwachung



Zeitplanung • Schätzung des Gesamtaufwands

- Aufstellen eines Zeit- und Arbeitsplans • Risikomanagement
- Risikoermittlung und -einschätzung • Riskovorbeugung und -behandlung
- Personalmanagement
- Gruppen und Teams
- Psychologische Effekte
- Integrierende Aufgaben • Projektplan
- Projektleitung und -überwachung



Schätzen: Rolle

- Zuverlässige (und halbwegs genaue) Aufwandsschätzungen sind eine wichtige Grundlage für wirtschaftlich erfolgreiche SW-Projekte
 - denn wenn ein Projekt mehr kostet als es Nutzen bringt, sollte man das vorher wissen (und es nicht durchführen)
 - Zusätzlich sind auch Zeitschätzungen sehr wichtig
 - weil der Nutzen oft stark abnimmt, wenn man zu spät an den Markt kommt
 - Zu schlechte (dann meist zu optimistische) Schätzungen führen zu
 - Streß und Frustration bei den Projektmitgliedern
 - langfristig Ausbrennen
 - Wirtschaftlichen Verlusten, auch bei den Auftraggebern



Schätzen: Probleme

- Leider ist das Schätzen von Softwareprojekten aufgrund von deren hoher Komplexität schwierig
- außerdem gibt es Störfaktoren, insbesondere unbekannte oder schwankende Projektanforderungen
- Wichtige Fragen beim Schätzen
 - Sind die Anforderungen schon bekannt?
 - Alle? Genau genug? Auch die nichtfunktionalen?
 - Ist die Architektur schon verstanden? (Standardarchitektur?)
 - Ist die Technologie vorgegeben?
 - Gibt es sonstige schwierige/ungewöhnliche Randbedingungen? ● z.B. verteilte Entwicklung, kein direkter Benutzerkontakt etc.
- Faustregeln:
 - Hochinnovative Projekte kann man nicht schätzen
 - Nur normales Vorgehen lässt sich schätzen, radikales nicht
 - Projekte mit fließenden Anforderungen kann man nicht verlässlich schätzen



- Zeitplanung
- Schätzung des Gesamtaufwands ("cost estimation")

● Aufstellen eines Zeit- und Arbeitspla

- Risikomanagement
- Risikoermittlung und -einschätzung • Riskovorbeugung und -behandlung
- Personalmanagement
- Gruppen und Teams
- Psychologische Effekte
- Integrierende Aufgaben • Projektplan
- Projektleitung und -überwachung

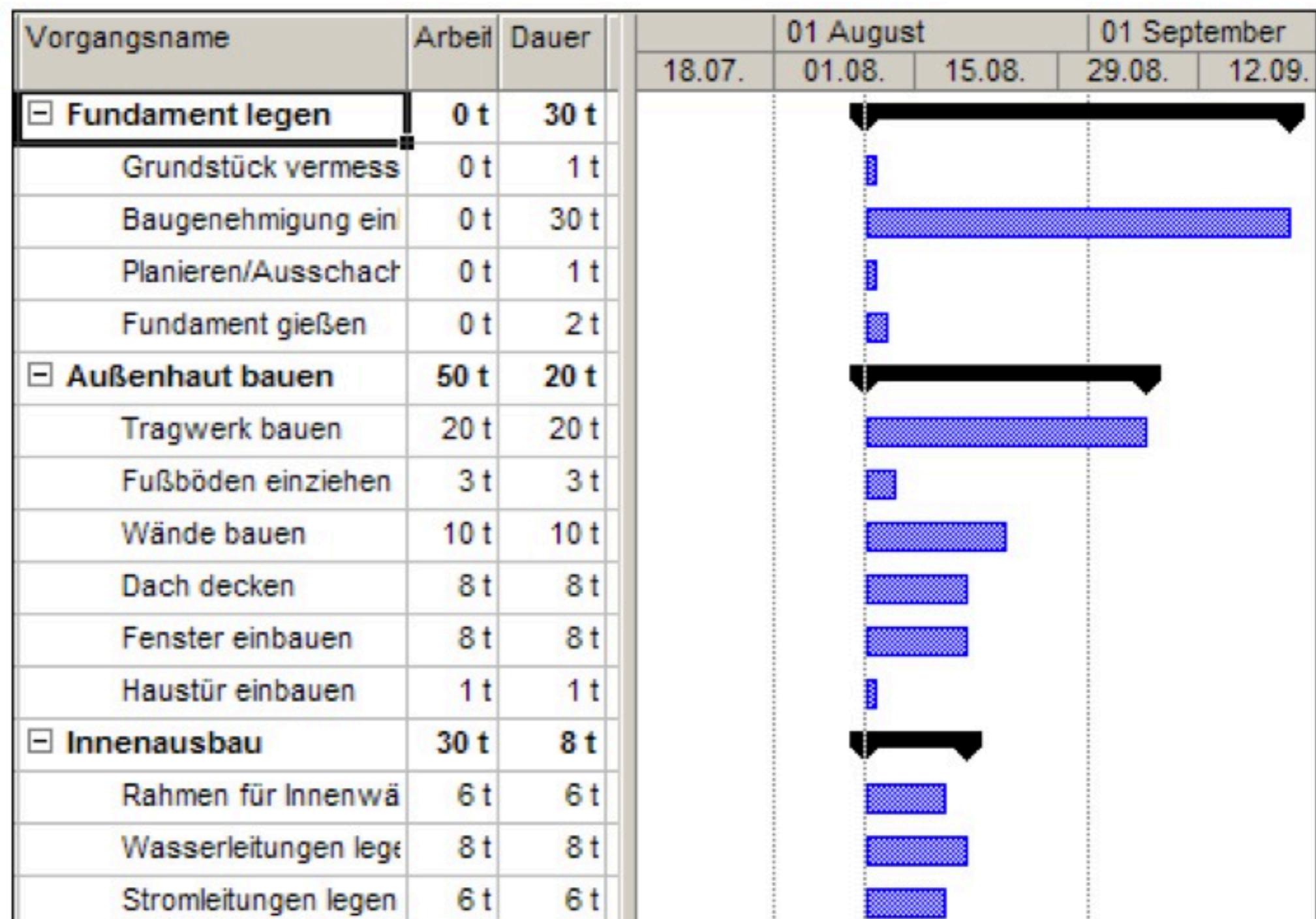


Planung: Problem

- Es genügt für das Projektmanagement nicht, den Aufwand für das Gesamtprojekt zu schätzen.
- Man muss auch herausfinden, wie es zu organisieren ist: Planung
- Einzelfragen:
 - Welche Teilaufgaben sind zu lösen?
 - Wie lange dauern die?
 - Welche Abhängigkeiten bestehen dazwischen (A muss vor B)?
 - Wie viele Leute mit welchen Qualifikationen brauchen wir? Wann?
 - Welche anderen Ressourcen (z.B. SW, Dienste, Rechner, Hilfspersonal) brauchen wir? Wann?
 - Sobald das Personal feststeht: Wer ist qualifiziert für welche Aufgabe, die jeweils gerade durchgeführt werden muss?



Teilaufgaben ("Vorgänge") mit Aufwand ("feste Arbeit" oder "feste Dauer") eingeben, Gantt-Diagramm generieren





Welche Vorgänger müssen vor einer Aufgabe erledigt sein? • kürzester Zeitplan wird automatisch berechnet

		Vorgangsname	Arbeit	Dauer	Vorgänger	I	01 August	01 September		
							01.08.	15.08.	29.08.	12.09.
1		■ Fundament legen	0 t	34 t						
2		Grundstück vermess	0 t	1 t						
3		Baugenehmigung ein	0 t	30 t	2					
4		Planieren/Ausschacht	0 t	1 t	3					
5		Fundament gießen	0 t	2 t	4					
6		■ Außenhaut bauen	50 t	38 t	1					
7		Tragwerk bauen	20 t	20 t						
8		Fußböden einziehen	3 t	3 t	7AA					
9		Wände bauen	10 t	10 t	8;7					
10		Dach decken	8 t	8 t	7					
11		Fenster einbauen	8 t	8 t	9					
12		Haustür einbauen	1 t	1 t	9					
13		■ Innenausbau	30 t	20 t						
14		Rahmen für Innenwä	6 t	6 t	7					
15		Wasserleitungen legen	8 t	8 t	14					
16		Stromleitungen legen	6 t	6 t	14;10					



MS Project: Automatische Planung

- Jetzt tritt die eigentliche Automatisierung in Aktion:
- Ressourcenabgleich:
 - Die Beteiligung jeder Ressource an ihren Aufgaben wird so reduziert (in %), dass die Ressource gleichzeitige Aufgaben überlastet ist
 - Kritischer Pfad:
 - Es wird berechnet, bei welchen Aufgaben jede Verzögerung zu einer Verzögerung des Gesamtprojekts führt
 - Die Abfolge dieser Aufgaben bildet den so genannten "kritischen Pfad" durch das Projekt
 - es kann (selten) mehrere kritische Pfade geben
 - Alle Aufgaben außerhalb kritischer Pfade haben Spielraum (slack time), d.h. Zeitpuffer
 - Verzögerungen dieser Aufgaben, die geringer als der Spielraum sind, verzögern das Projekt nicht
 - Aber das Nutzen eines Spielraums verkleinert andere Spielräume





Zeitplanung

- Schätzung des Gesamtaufwands ("cost estimation") • Aufstellen eines Zeit- und Arbe

- Risikomanagement
- Risikoermittlung und -einschätzung • Riskovorbeugung und -behandlung
- Personalmanagement
- Gruppen und Teams
- Psychologische Effekte
- Integrierende Aufgaben

• Projektplan

- Projektleitung und -überwachung



Welche Dokumente gibt es in einem SW-Projekt?

- Vereinbarungen mit Auftraggeber • Vertrag
- Vertragsgrundlagen wie Pflichtenheft (enthält Anforderungen) ● Produktorientierte (technische) Dokumente
- Anforderungen, Entwurf, Code, Testfälle, etc. ● Prozessorientierte technische Dokumente
- z.B. Testplan, Defektdatenbank, Konfigurationsdatenbank, Checklisten für Durchsichten etc.
- Prozessorientierte organisatorische Dokumente
- Die Bestandteile des Projektplans
- Der Projektplan soll also den Prozess zu organisieren helfen



Rollen des Projektplans

- Bildet eine Klammer um alle übrigen Dokumente • Er deklariert "was steht wo?"
- Richtet sich sowohl nach innen (an das Projektteam) als auch nach außen (an den Anforderer)
- Nach Außen: Definiert das WAS des Prozesses
 - Anforderungen an das Projekt (und damit auch an das Produkt)
- Nach Innen: Definiert das WIE des Prozesses • Entwurf der Projektdurchführung
 - Meist als zwei separate Dokumente
 - Da Auftraggeber das WIE nicht sehen will/muss
 - WAS: oft genannt *Pflichtenheft*
 - WIE: oft genannt *Projektplan*
 - Beide Begriffe werden sehr uneinheitlich benutzt
 - z.B. meint *Pflichtenheft* evtl. nur das Produkt-Anforderungsdokument, so dass dann keine Angaben zur Durchführung stehen



Nach Außen: Vereinbarung mit Auftraggeber (Pflichtenheft)

- Dokument, das mit dem Auftraggeber geschrieben wird
- Definiert Projektziele, -dauer, -kosten
- Definiert zu liefernde Ergebnisse
 - mit Details: was, wieviel, wann, wer, wie, wo?
- Definiert Mitwirkungspflichten des Auftraggebers
- Arten von Ergebnissen ("deliverables")
 - Dokumente (inkl. Programmcode)
 - ggf. in vorgeschriebener Form, z.B. installationsfertige CD
 - Abnahmetest (Vorführung von Funktionen und nichtfunktionalen Eigenschaften)
 - Abnahme von Entwurf und Implementierung
 - Einweisung/Ausbildung von Endbenutzern
 - Einweisung von Administratoren
 - Einweisung von Wartungspersonal



Nach Innen: Projektentwurf

- Gibt Zeitplanung und Prioritäten vor • siehe Einheit über MS Project etc.
- Enthält Zuständigkeitsplanung und -festlegung • siehe Einheit über Gruppen und Rollen
- Beschreibt Durchführungsregeln (Projektcharta) • verwendeten SW-Prozess,
- Benutzung von Werkzeugen,
- Berichtsverfahren,
- Entscheidungsverfahren, • Regeln für Besprechungen • etc.

Meist ist der Projektplan eine Sammlung von Einzelteilen anstatt ein geschlossenes Dokument

Übersicht

- Zeitplanung
- Schätzung des Gesamtaufwands ("cost estimation")
 - Aufstellen eines Zeit- und Arbeitsplans
- Risikomanagement
- Risikoermittlung und -einschätzung
 - Riskovorbeugung und -behandlung
- Personalmanagement
- Gruppen und Teams
- Psychologische Effekte
- Integrierende Aufgaben
 - Projektplan

• Projektleitung und - überwachung



Aufgaben der Projektleitung

- Nach Außen zum Auftraggeber:
 - Schätzen und Planen
 - Anfängliche Übereinkunft erarbeiten
 - Zufriedenheit managen
 - Änderungen an Anforderungen und Prioritäten verhandeln und mitentscheiden
 - Zeitplan-/Budgetänderungen verhandeln
- Nach Innen zum Projektteam:
 - Schätzen und Planen
 - Team aufbauen und formen
 - Entwicklungsleiten/überwachen
 - dies hat zahlreiche Aspekte



Haupttechniken für Projektleitung

- Planung
 - siehe Abschnitt neulich
- Risikomanagement
 - siehe Abschnitt neulich
- Taktisches Entscheiden
 - meist über Ressourcen-Zuordnungen (d.h. auch über Prioritäten) ● Personal- und Zeitbedarf für (Detailplanung)
 - Personal- und Zeiteinsatz für "unendliche" Aufgaben
 - Freigabe von Dokumenten/Komponenten
 - Auf Basis der Resultate der Qualitätssicherung
 - Kommunikation
 - Sammeln von Information ● Verteilen von Information ● Gemeinsames Entscheiden



Was macht das taktische Entscheiden schwierig?

- Theorie:
 - Wenn man erst mal einen detaillierten und korrekten Plan aufgestellt hat, sind nur noch triviale Entscheidungen nötig:
 - z.B. "Wenn A krank geworden ist, wer übernimmt die Aufgabe?"
 - Für die entstehende Verzögerung steht ein Zeitpuffer zur Verfügung
- (wg. Risikomanagement) ● Praxis:
 - 1. Der Plan ist nicht korrekt, sondern sehr ungenau
 - 2. Die Effekte selbst kleiner Abweichungen vom Plan können sich aufschaukeln (Nicht-lineare Dynamik)
 - 3. Es werden häufig von außen Änderungen verlangt
- Wir besprechen jetzt (a) Nichtlineare Dynamik und (b) Änderungsmanagement
- (b): Planänderungen bewirken unmittelbar Planabweichungen, setzen also Rückkopplungsschleifen (nichtlin. Dyn.) in Gang



Nichtlineare Dynamik

- Nichtlineare Systemdynamik kommt dadurch zustande, dass die Wirkungen einer Ursache zurückwirken können
 - verstärkend (positive Rückkopplung) oder • abschwächend (negative Rückkopplung)
- Konsequenzen:
 - Die Wirkung kleiner Ursachen kann sehr groß sein
 - Eine Wirkung kann sich im Laufe der Zeit schnell verstärken
 - Der Aufwand, um irgendetwas rückgängig zu machen, kann deshalb unerhört groß sein



What you should know!

- > How does Software Engineering differ from programming?
- > Why is the “waterfall” model unrealistic?
- > What is the difference between analysis and design?
- > Why plan to iterate? Why develop incrementally?
- > Why is programming only a small part of the cost of a “real” software project?
- > What are the key advantages and disadvantages of object-oriented methods?

Roadmap



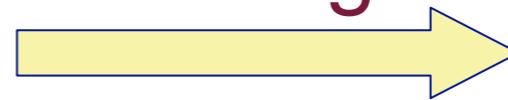
- > Course Overview
- > **What is Software Engineering?**
- > The Iterative Development Lifecycle
- > Software Development Activities
- > Methods and Methodologies

Why Software Engineering?

A naive view:

Problem Specification

coding



Final Program

But ...

- Where did the *specification* come from?
- How do you know the specification corresponds to the *user's needs*?
- How did you decide how to *structure* your program?
- How do you know the program actually *meets the specification*?
- How do you know your program will always *work correctly*?
- What do you do if the *users' needs change*?
- How do you *divide tasks up* if you have more than a one-person team?

What is Software Engineering? (I)

Some Definitions and Issues

“state of the art of developing quality software on time and within budget”

- > Trade-off between perfection and physical constraints
 - SE has to deal with real-world issues
- > State of the art!
 - Community decides on “best practice” + life-long education

What is Software Engineering? (II)

“multi-person construction of multi-version software”

— Parnas

- > Team-work
 - Scale issue (“program well” is not enough) + Communication Issue
- > Successful software systems must evolve or perish
 - Change is the norm, not the exception

What is Software Engineering? (III)

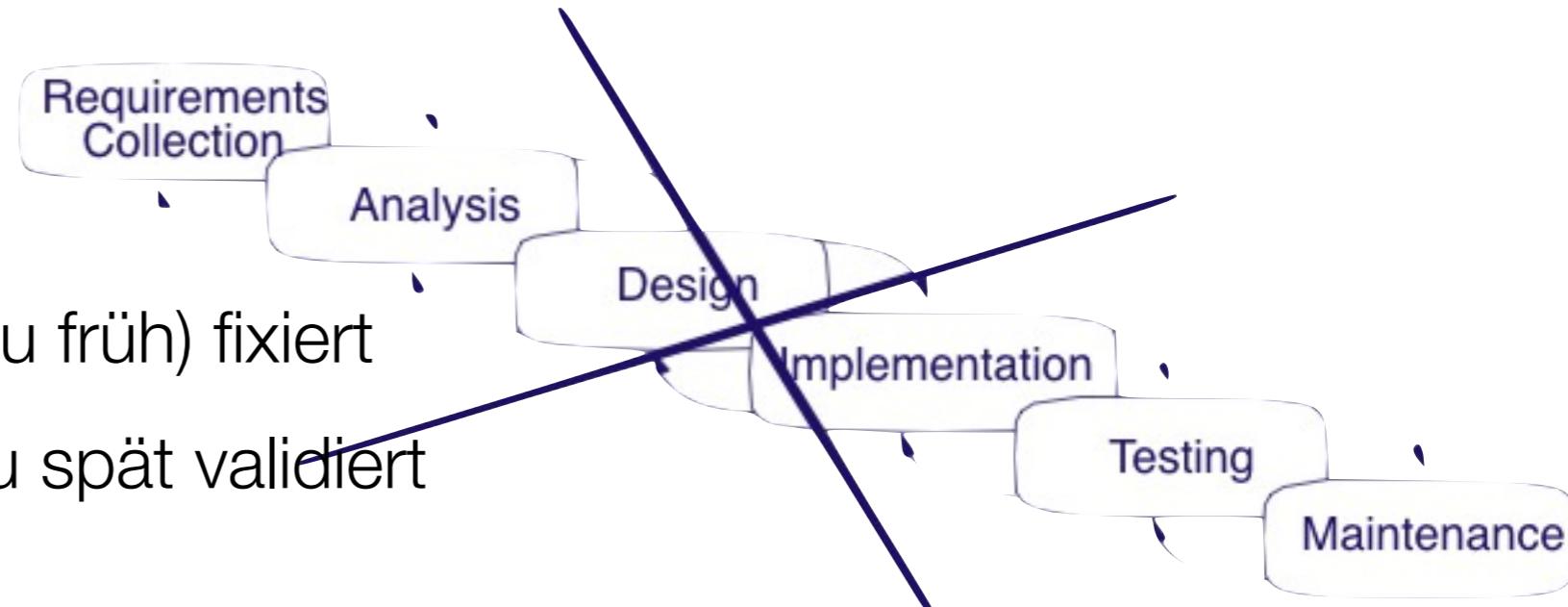
“software engineering is different from other engineering disciplines”

— Sommerville

- > Not constrained by physical laws
 - limit = human mind
- > It is constrained by political forces
 - balancing stake-holders

Klassisches Vorgehen

- Schritt für Schritt
- Nachteile
 - Anforderungen werden (zu früh) fixiert
 - Anforderungen werden zu spät validiert
 - Keine Iterationen möglich
 - Alle Anforderungen müssen scharf definiert sein
 - Arbeitsfähiges Zwischenprodukt steht erst spät bereit
 - Wenn Missverständnis spät entdeckt > Faktor 10 Regel

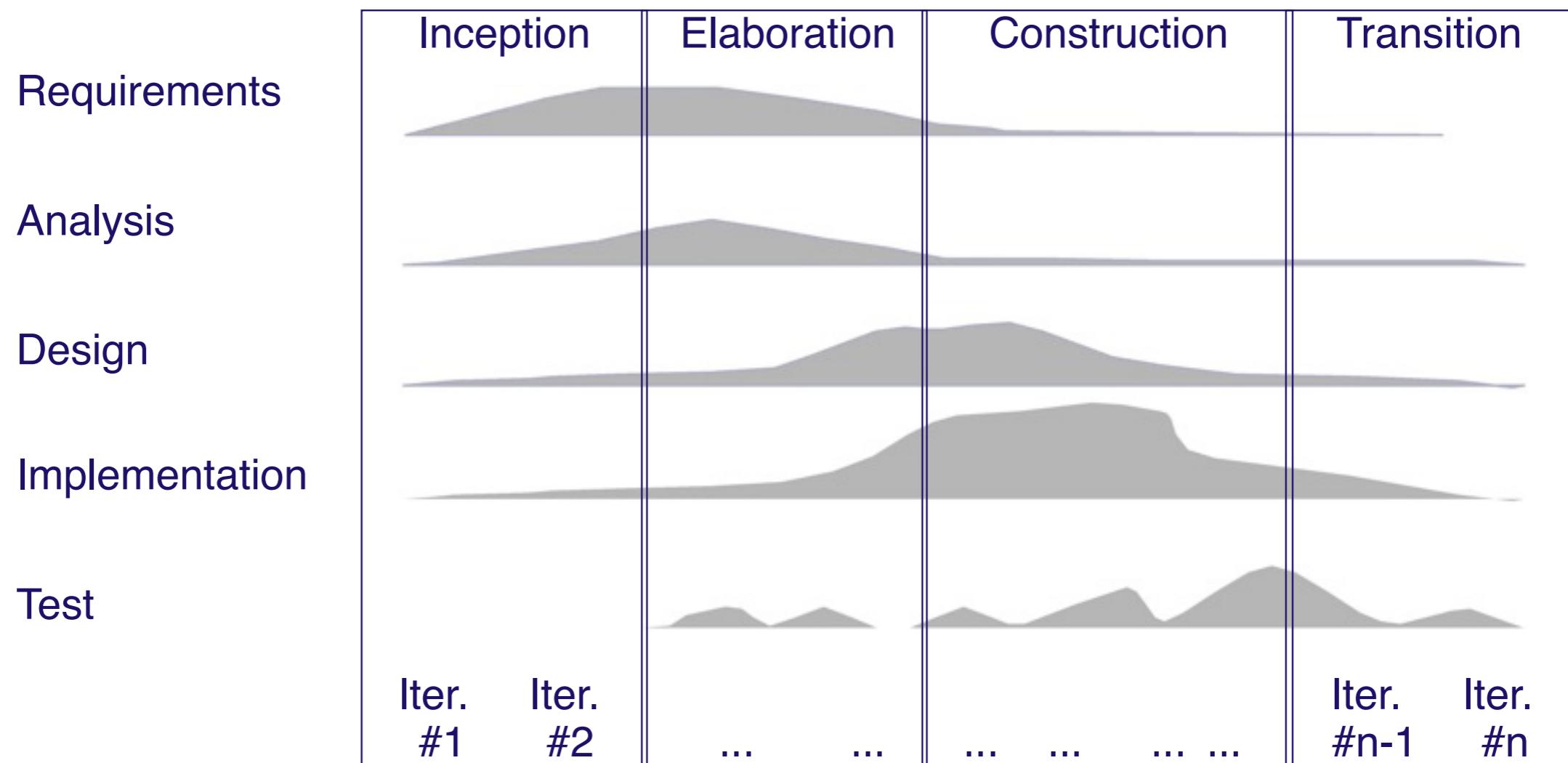


- Iterationen über
 - Anforderungen, Design und Implementierung
 - Ziel: Erreichung von Konsistenz
 - d.h. Bijektion zwischen Anforderungen <-> Design <-> Implementierung
- Inkrementelle Entwicklung
 - Funktion für Funktion
 - Ziel: Ständig lauffähiges Zwischenprodukt

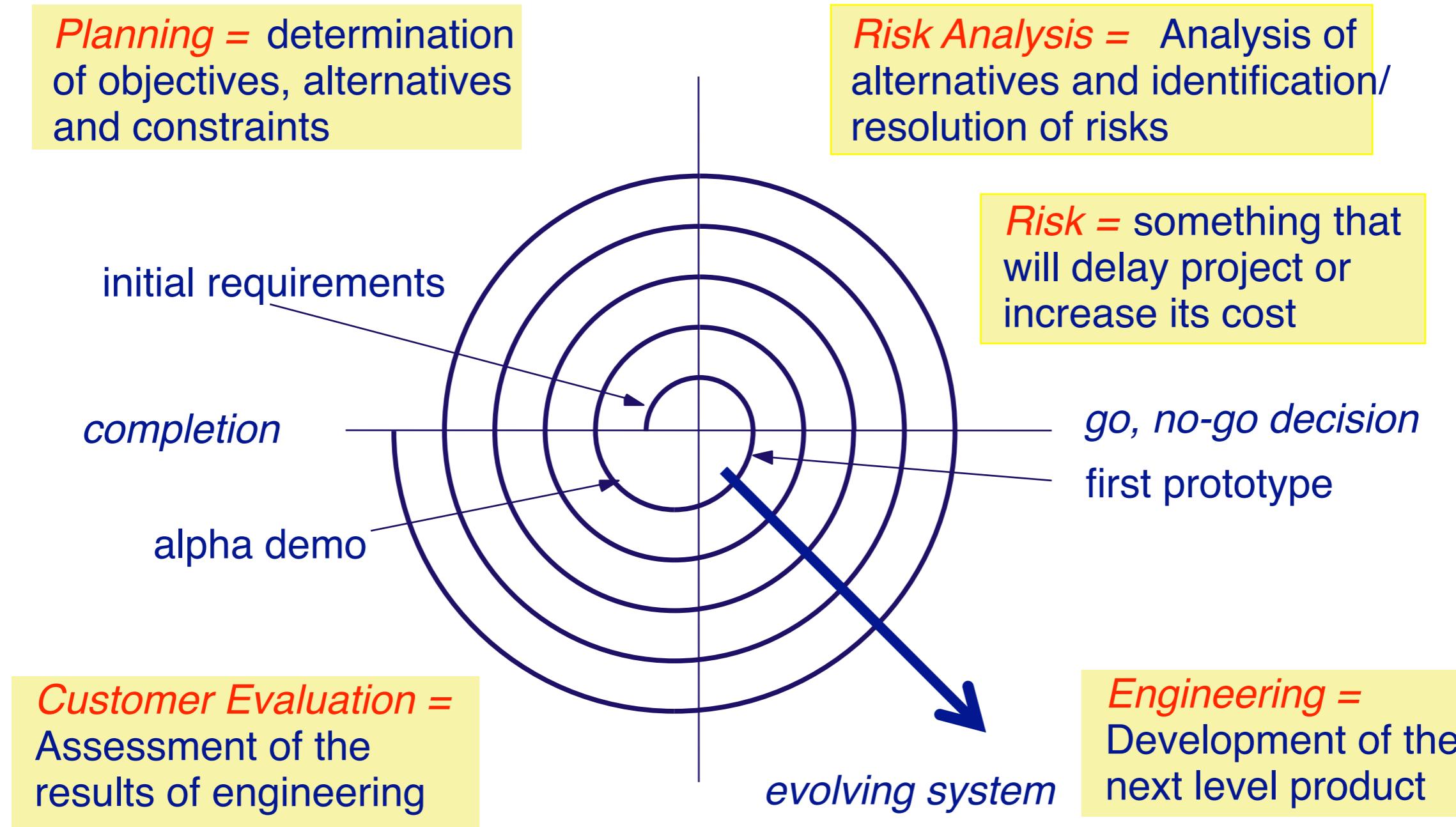
The Unified Process



- Wie entscheidet man über Anzahl Iterationen?
- Wann ist das Produkt fertig?



Boehm's Spiral Lifecycle



Roadmap



- > Course Overview
- > What is Software Engineering?
- > The Iterative Development Lifecycle
- > **Software Development Activities**
- > Methods and Methodologies

Requirements Collection

User requirements are often expressed *informally*:

- features
- usage scenarios

Although requirements may be documented in written form,
they may be *incomplete*, *ambiguous*, or even *incorrect*.

Changing requirements

Requirements *will* change!

- inadequately captured* or expressed in the first place
- user and business *needs may change* during the project

Validation is needed *throughout* the software lifecycle, not only when the “final system” is delivered!

- build constant *feedback* into your project plan
- plan for *change*
- early *prototyping* [e.g., UI] can help clarify requirements

Requirements Analysis and Specification

Analysis is the process of specifying *what* a system will do.

- The intention is to provide a clear understanding of what the system is about and what its underlying concepts are.

The result of analysis is a *specification document*.

Does the requirements specification correspond to the users' actual needs?

Object-Oriented Analysis

An object-oriented analysis results in models of the system which describe:

- > *classes* of objects that exist in the system
 - responsibilities* of those classes
- > *relationships* between those classes
- > *use cases* and *scenarios* describing
 - operations* that can be performed on the system
 - allowable *sequences* of those operations

Prototyping (I)

A *prototype* is a software program developed to test, explore or validate a hypothesis, i.e. *to reduce risks*.

An *exploratory prototype*, also known as a *throwaway prototype*, is intended to *validate requirements* or *explore design choices*.

- UI prototype — validate user requirements
- rapid prototype — validate functional requirements
- experimental prototype — validate technical feasibility

Prototyping (II)

An evolutionary prototype is intended to evolve in steps into a finished product.

- > iteratively “grow” the application, *redesigning* and *refactoring* along the way

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

Design

Design is the process of specifying *how* the specified system behaviour will be realized from software components. The results are *architecture* and *detailed design documents*.

Object-oriented design delivers models that describe:

- how system operations are implemented by *interacting objects*
- how classes refer to one another and how they are related by *inheritance*
- *attributes* and *operations* associated to classes

*Design is an iterative process,
proceeding in parallel with
implementation!*

Conway's Law

—“Organizations that design systems are constrained to produce designs that are copies of the communication structures of these organizations”

Implementation and Testing

Implementation is the activity of *constructing* a software solution to the customer's requirements.

Testing is the process of *validating* that the solution meets the requirements.

—The result of implementation and testing is a *fully documented* and *validated* solution.

Design, Implementation and Testing

Design, implementation and testing are iterative activities

- The implementation does not “implement the design”, but rather the design document *documents the implementation!*

- > System tests reflect the requirements specification
- > Testing and implementation go hand-in-hand
 - Ideally, test case specification *precedes* design and implementation

Maintenance

Maintenance is the process of changing a system after it has been deployed.

- > Corrective maintenance: identifying and repairing *defects*
- > Adaptive maintenance: *adapting* the existing solution to new platforms
- > Perfective maintenance: implementing *new requirements*

In a spiral lifecycle, everything after the delivery and deployment of the first prototype can be considered “maintenance”!

Maintenance activities

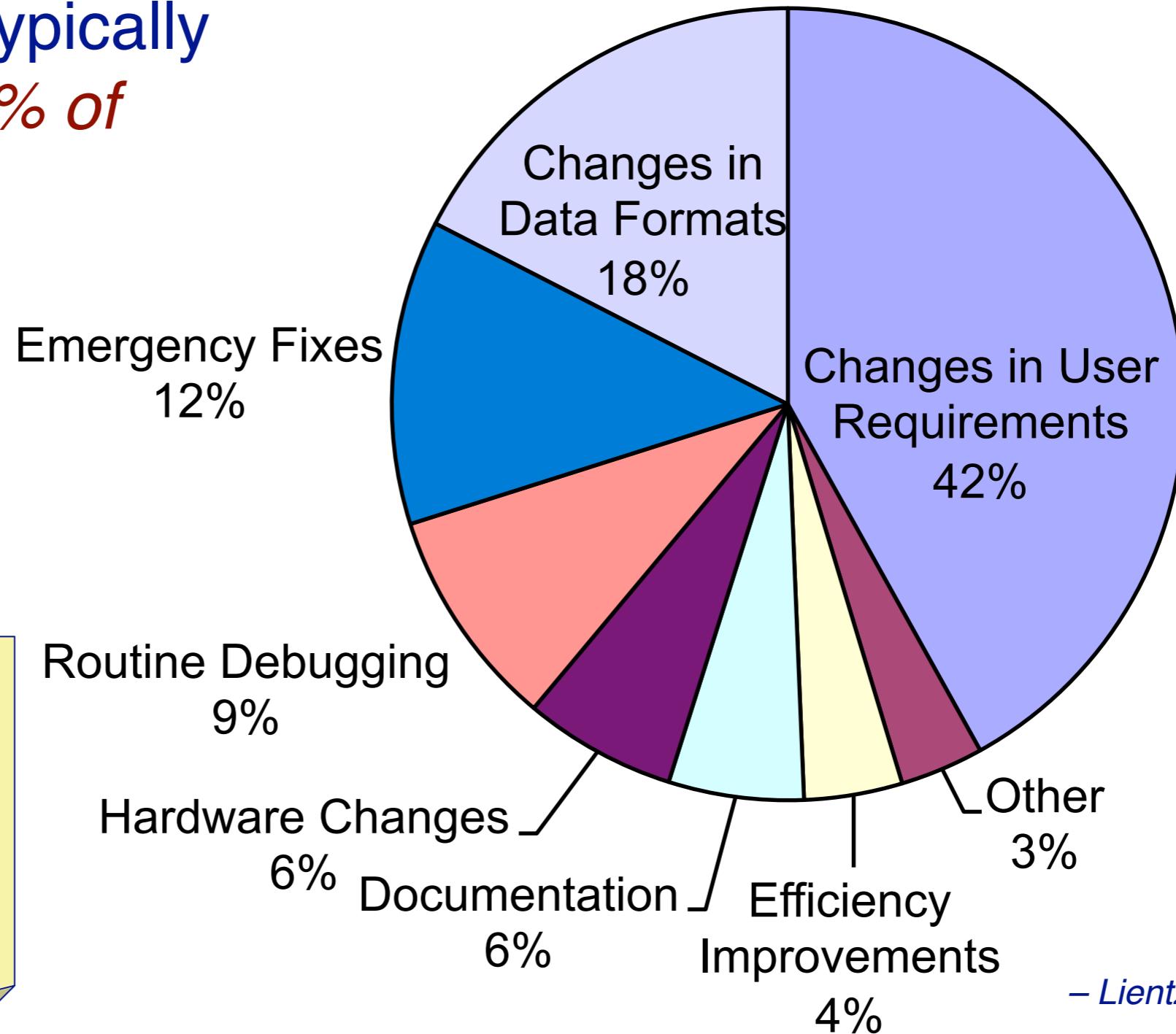
“Maintenance” entails:

- > configuration and version management
- > reengineering (redesigning and refactoring)
- > updating all analysis, design and user documentation

*Repeatable, automated
tests enable evolution
and refactoring*

Maintenance costs

“Maintenance” typically accounts for *70% of software costs!*



Means: most project costs concern continued development *after* deployment

– Lientz 1979

Roadmap



- > Course Overview
- > What is Software Engineering?
- > The Iterative Development Lifecycle
- > Software Development Activities
- > **Methods and Methodologies**

Software Development Activities

Requirements Collection

Knut Reinert,
April 2009
Berlin

User requirements are often expressed *informally*:

- features
- usage scenarios

Although requirements may be documented in written form, they may be *incomplete*, *ambiguous*, or even *incorrect*.

- Kund/inn/en
 - das sind die Leute, die das Geld geben
 - evtl. viele, mit unterschiedlichen Bedürfnissen
- Benutzer/innen
 - das sind die Leute, die später die Software einsetzen
 - nur manchmal mit Kund/inn/en identisch, meist nur teilweise
 - fast immer viele, mit unterschiedlichen Wünschen/Bedürfnissen
- Manager/innen
 - das sind die Leute, die während der Entwicklung die organisatorischen Entscheidungen fällen
 - hoffentlich nur eine/r (aber meist mit weiteren Vorgesetzten)
- Entwickler/innen
 - das sind die Leute, die die Software definieren und bauen
 - in verschiedenen Rollen (d.h. m. verschiedenartigen Aufgaben)

Prototyping

Knut Reinert,
April 2009
Berlin

A prototype is a software program developed to test, explore or validate a hypothesis, i.e. *to reduce risks*.

An exploratory prototype, also known as a *throwaway prototype*, is intended to **validate requirements** or *explore design choices*.

- UI prototype – validate user requirements
- rapid prototype – validate functional requirements
- experimental prototype – validate technical feasibility

Analytische Qualitätssicherung

- **Produkt**orientiert, d.h.
konkrete Qualitätsmängel aufdecken
und nachbessern
- Statisch
 - Durchsichten
 - Inspektionen
 - Modellprüfung
 - ...
- Dynamisch
 - Test



Konstruktive Qualitätssicherung

- **Prozess**orientiert: Qualitätsmängel von vornherein durch das Vorgehen vermeiden
 - “Vorbeugen ist besser als Heilen”
 - “Fehler, die nicht gemacht werden, brauchen auch nicht behoben zu werden”
- Durch:
 - Prozessmanagement
 - Verwendung von Entwurfsmustern
 - Code Reviews
 - Test Driven Development



Design

Knut Reinert,
April 2009
Berlin

Design is the process of specifying *how* the specified system behaviour will be realized from software components. The results are *architecture* and *detailed design documents*.

Object-oriented design delivers models that describe:

- how system operations are implemented by *interacting objects*
- how classes refer to one another and how they are related by *inheritance*
- *attributes* and *operations* associated to classes

Design is an iterative process, proceeding in parallel with implementation!



Berlin Center for Genome Based Bioinformatics

Algorithmische Bioinformatik, FU Berlin



Design, Implementation and Testing

Knut Reinert,
April 2009
Berlin

Design, implementation and testing are iterative activities

- The implementation does not “implement the design”, but rather the design document *documents the implementation!*

System tests reflect the requirements specification
Testing and implementation go hand-in-hand

- Ideally, test case specification *precedes* design and implementation



Knut Reinert,
April 2009
Berlin

Implementation and Testing

Implementation is the activity of *constructing* a software solution to the customer’s requirements.

Testing is the process of *validating* that the solution meets the requirements.

The result of implementation and testing is a *fully documented* and *validated* solution.

Knut Reinert,
April 2009
Berlin

Maintenance

Maintenance is the process of changing a system after it has been deployed.

- > Corrective maintenance: identifying and repairing defects (tool used e.g. bug tracker)
- > Adaptive maintenance: adapting the existing solution to new platforms (e.g. new OS)
- > Perfective maintenance: implementing new requirements

In a spiral lifecycle, everything after the delivery and deployment of the first prototype can be considered “maintenance”!

Repeating automated tests enable evolution and refactoring



Berlin Center for Genome Based Bioinformatics

Algorithmische Bioinformatik, FU Berlin



Berlin Center for Genome Based Bioinformatics

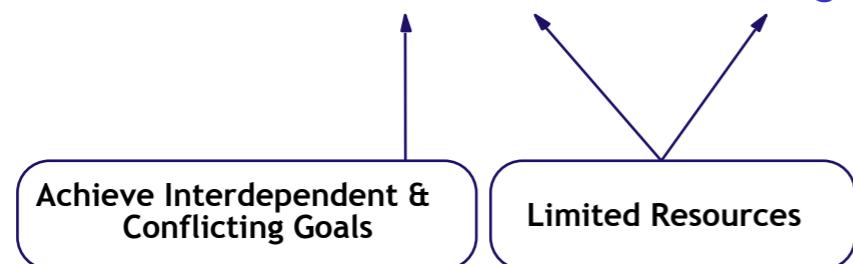
Algorithmische Bioinformatik

Why Project Management?

Knut Reinert,
April 2009
Berlin

Almost all software products are obtained via *projects*.
(as opposed to manufactured products)

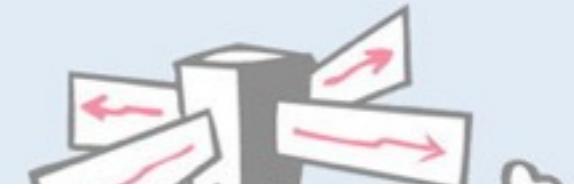
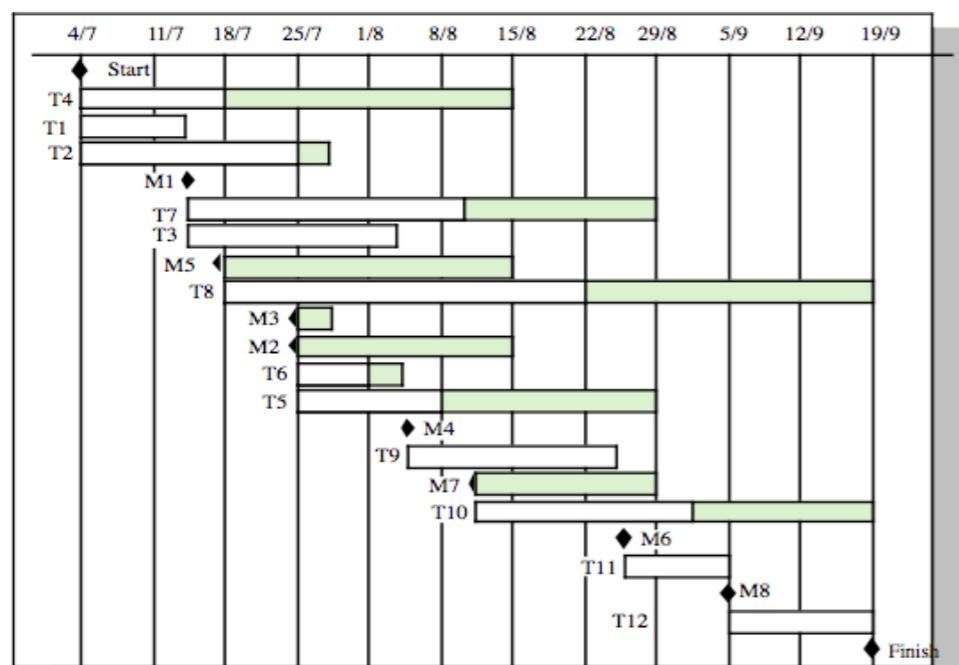
Project Concern = *Deliver on time* and *within budget*



The Project Team is the primary Resource!

Gantt Chart: Activity Timeline

Knut Reinert,
April 2009
Berlin



What is Project Management?

Knut Reinert,
April 2009
Berlin

Project Management = *Plan the work* and *work the plan*

Management Functions

- > *Planning*: Estimate and schedule resources
- > *Organization*: Who does what
- > *Staffing*: Recruiting and motivating personnel
- > *Directing*: Ensure team acts as a whole
- > *Monitoring (Controlling)*: Detect plan deviations + corrective actions



Berlin Center for Genome Based Bioinformatics

Algorithmische Bioinformatik, FU Berlin



Repeating automated enable even and refac

UML

Use Cases

Objektmodellierung /
Spezifikation

Wie modularisiert / zerlegt man
ein System am besten?

- Verringerung der Komplexität
- Information hiding

Vom Modell zur Implementierung

Prozess-/Projektmanagement

- keine Prozessmodelle genau erklärt

- Persönlichkeitstypen
- Wie schätzt man Zeitaufwände
- Risikomanagement
- Personalmanagement
 - Teams / Gruppen
 - Psychologische Effekte
- Methoden zur Entwicklung
sicherheitskritischer Systeme
- Methoden zur Maximierung der
Benutzbarkeit
- Software-Evolution