AG Software Engineering
AG Algorithmic Bioinformatics

Projektmanagement im Softwarebereich

SeqAn

Björn Kahlert

Institut für Informatik
Freie Universität Berlin
07.04.2014

Software Engineering

Software

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

Qualitätssicherung
Quality Assurance

Anforderungen
Requirements

Entwurf
Design

Implementierung
Implementation

Wartung
Maintenance

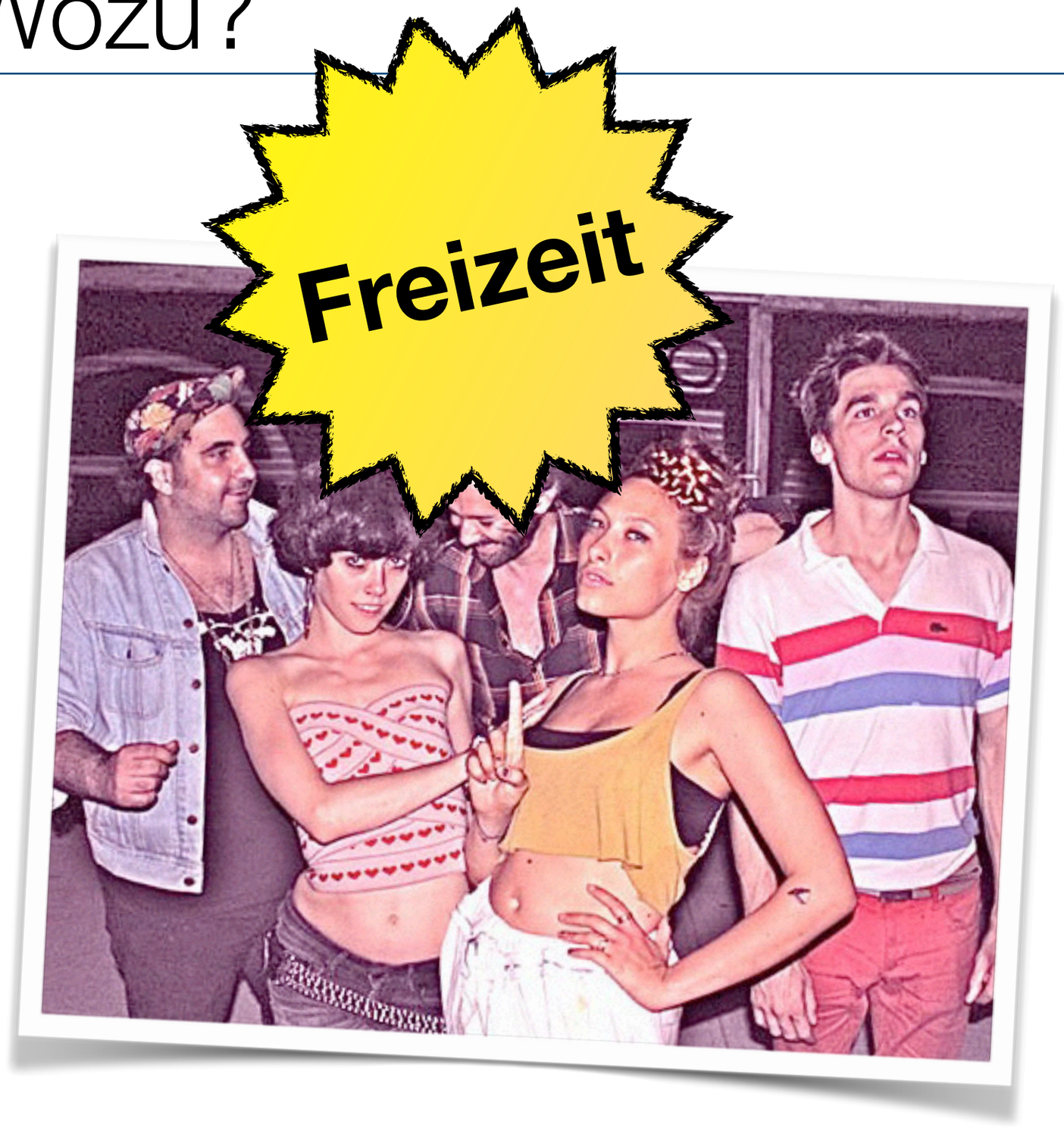
Probleme

- “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?



Wertschöpfung

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

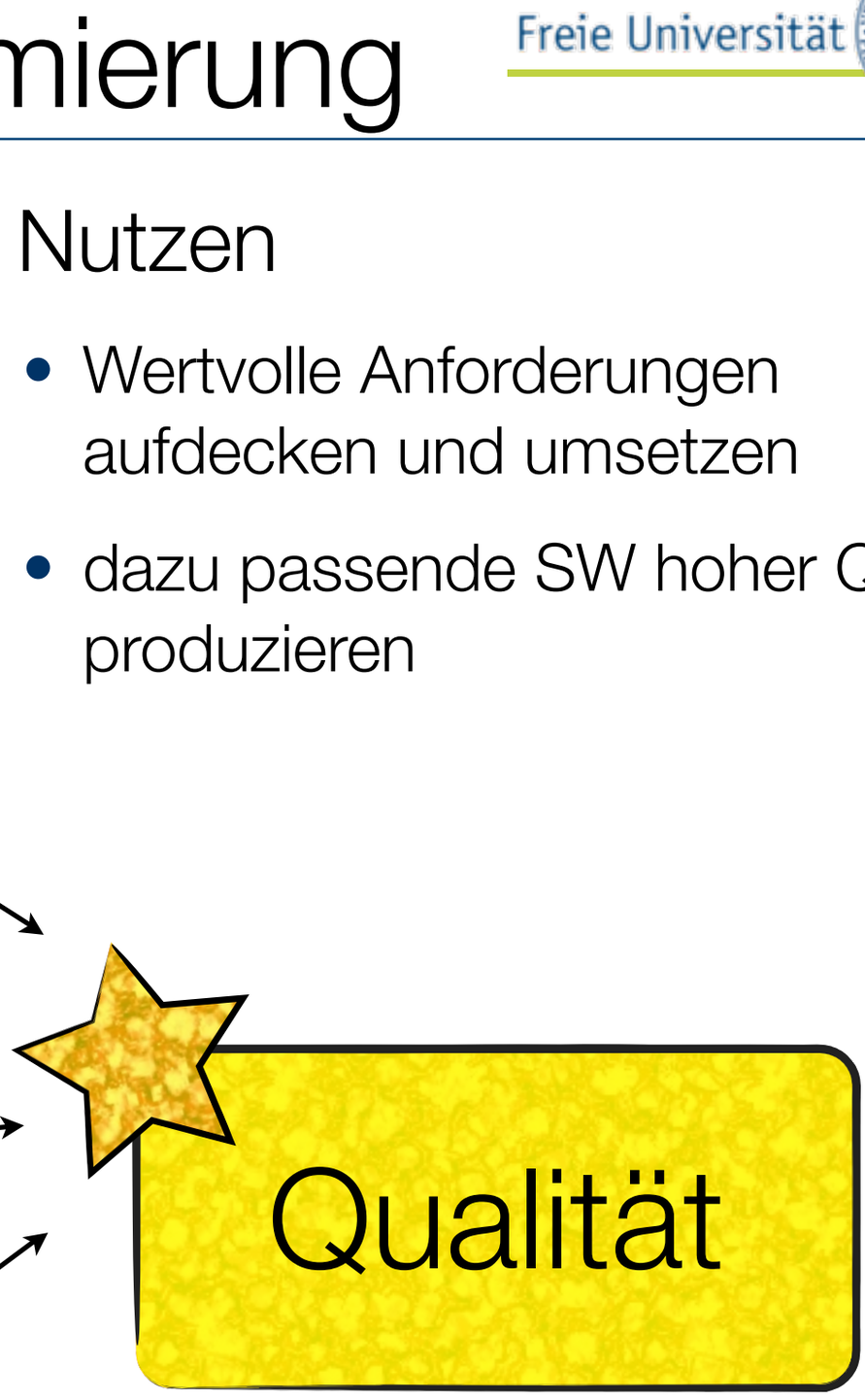
Kosten-/Nutzen-Optimierung

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



Taxonomie
"Die Welt der Softwaretechnik"

Freie Universität Berlin

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

Taxonomie
"Die Welt der Softwaretechnik"

Freie Universität Berlin

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

Produktprobleme

- Komplexität bedeutet
  - ein System besteht aus vielen Einzelteilen und
  - es hat emergente Eigenschaften
    (Eigenschaften, die sich nicht in den Einzelteilen finden lassen)
- "Komplex" heißt also praktisch so viel wie
  - "schwierig aufgrund vielfältiger Zusammenhänge"
- Komplexitätsprobleme bei SW-Entwicklung stammen meist aus der Komplexität des Produkts

Anforderungsprobleme

- 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 zu wenig 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

Entwurfsprobleme

- 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
  - Inakzeptable 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

Taxonomie
"Die Welt der Softwaretechnik"

Freie Universität Berlin

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

Prozessprobleme

- 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

Prozessprobleme

- **Kognitive Beschränkungen**
  - Die Denkfähigkeiten eines Menschen sind begrenzt.
  - Arbeitsgedächtnis: 7±2 Elemente
- **Mängel der Urteilskraft**
  - Oft kann keiner der Beteiligten in einem Projekt eine Situation gut genug beurteilen, um eine fällige Entscheidung richtig zu treffen
- **Kommunikation, Koordination**
  - Es ist schwierig, vorhandene Information stets korrekt und rechtzeitig an die Person weiterzugeben, die sie braucht.
- **Gruppendynamik**
  - Arbeiten in einer Gruppe beeinflusst Haltungen und Entscheidungen oft in unvernünftiger Art und Weise.
- **Verborgene Ziele**
  - Menschen handeln nicht immer nur im Interesse des Projekts, sondern haben persönliche (meist unausgesprochene) Ziele
- **Fehler**
  - Will ein Mensch X tun, so tut er oft Y

Welt der Lösungsansätze

- Was haben SW-Ingenieure und Softwaretechnik-Forscher(innen) an Ideen entwickelt, um den Problemen zu begegnen?

Taxonomie
"Die Welt der Softwaretechnik"

Freie Universität Berlin

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

Abstraktion

- Abstraktion: Konzentration auf wesentliche Eigenschaften durch Weglassen von Details
  - Andere Sicht: Gruppierung gleichartiger Dinge gemäß einer Gemeinsamkeit und Ignorieren der sonstigen Unterschiede
- **Kernprinzip der gesamten Informatik!**
- Verwendung für alle möglichen Zwecke
- Beispiel:
  - Man reduziert eine Softwarekomponente auf seine Schnittstelle

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!**

Automatisierung

- 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**

Taxonomie
"Die Welt der Softwaretechnik"

Freie Universität Berlin

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

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]

Anforderungstypen

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.)

Entwurf

- Einsicht: Man sollte vor dem Kodieren über eine günstige Struktur der Software nachdenken
  - 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)

Anforderungen bekannt

- Jetzt geht es darum, wie man sie realisieren kann. D.h.:
- Entscheiden wie/wo/wodurch die funktionalen Anforderungen (Funktionen) umgesetzt werden
  - Frage 1: **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: **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!

Architektur

- Rat: Bekannte Gesamtstrukturen / Architekturen verwenden
- 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

Bekannte Architekturen

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.)

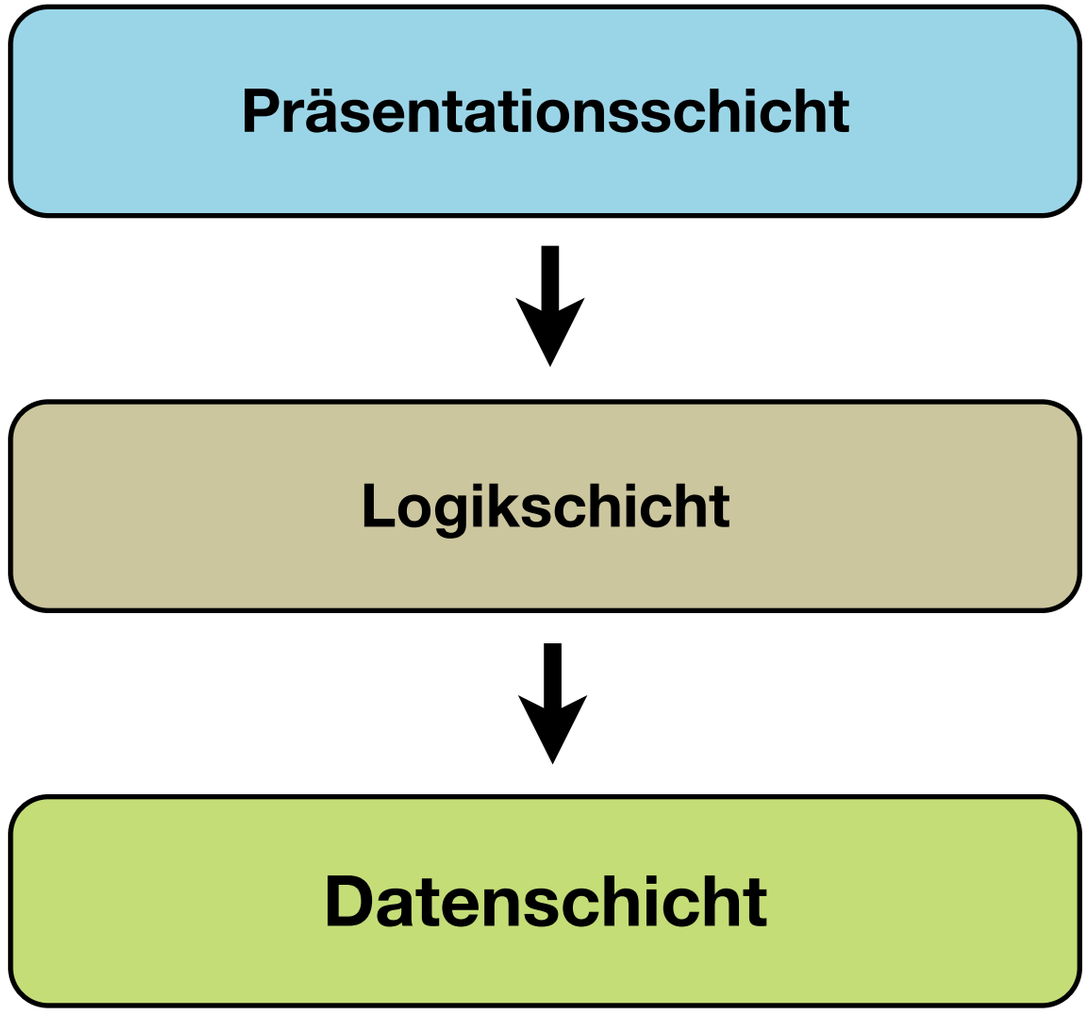
- ein allgemeiner A.stil, der mit vielen anderen Architekturideen verbunden werden kann

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

- technologiezentrierte Architekturen

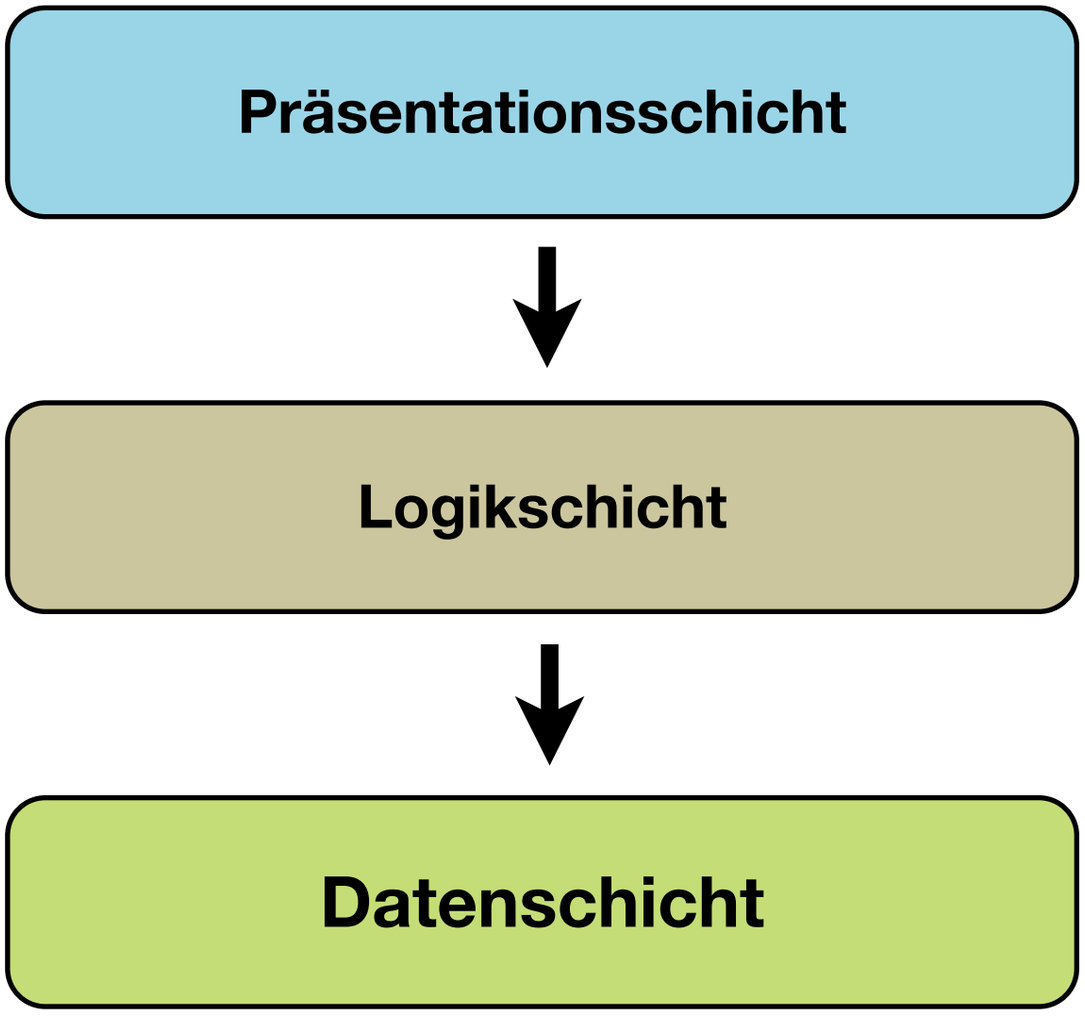
Beispiel: Schichten-Architektur

- 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



Beispiel: Schichten-Architektur

- Vorteile:
  - Sehr übersichtliche, aufgeräumte Grobstruktur
  - Komplettaustausch 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 responsibilies 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 we 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

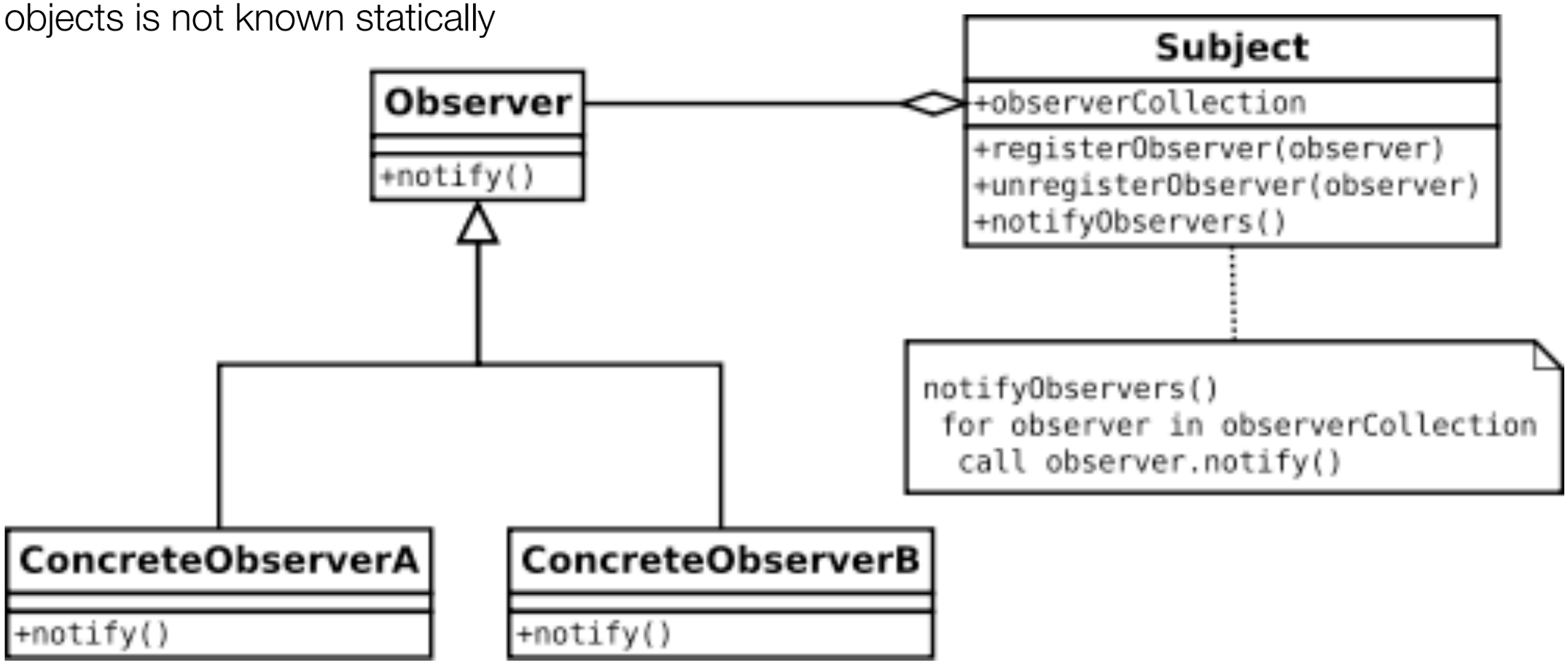
Also known as Publish/Subscribe pattern

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



Taxonomie
"Die Welt der Softwaretechnik"

Freie Universität Berlin

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

- Einsicht: Man macht oft Fehler, die zu schweren Mängeln in der SW führen können
  - Solchen Mängeln sollte man vorbeugen
- Prinzipien:
  - Analytische Qualitätssicherung
    - Verwende **prüfende Maßnahmen**, die entstandene Mängel aufdecken sollen, damit man sie beheben kann
    - Wichtigste Vertreter (bislang): **Test, Durchsichten**
  - 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**

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
  - Je früher ein Mangel entdeckt wird, desto weniger Schaden
    - **z.B. Anforderungsmängel erst nach Auslieferung zu beseitigen kostet oft über 1.000-mal mehr als bei der Anforderungsbestimmung**

Qualitätssicherung

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

Qualitätssicherung

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

Defekttest

Ziel des Defekttests ist Herbeiführen von **Versagen** (**failure**)

- Also einem falschen Verhalten des 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)** zu Grunde

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 mit dem Testen aufhören?

Defekttest: Zustände & Eingaben

- **Bekannte Versagensfälle**
- Wird ein Versagen im Test nicht aufgedeckt
  - sondern erst später, wird genau dieser Fall (falls reproduzierbar) nach der Korrektur in jedem Fall getestet
  - Evtl. wird dieser Testfall automatisiert und
  - seine Durchführung künftig nach jeder Änderung wiederholt

Defekttest: Zustände & Eingaben

- **Allgemeine Erfahrung, Intuition**
- Faustregeln
  - Leere Eingaben
  - Riesige Eingaben
  - Völlig unsinnige Eingaben
    - z.B. Binärdaten statt Text; irrwitzige Reihenfolgen von Operationen; etc.

Defekttest: Zustände & 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

Testen - Wann aufhören?

- Im Prinzip: Wenn die Kosten zum Aufdecken weiterer Defekte den Nutzen, sie entdeckt zu haben, übersteigen
  - Konkret kennt man aber weder die Kosten noch den Nutzen
- 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 kompetentes Team testet

Gemeinsame Schwäche aller Testverfahren

- Alle testenden Verfahren führen zunächst nur zu Versagen
  - Das Versagen muss dann auf einen Defekt zurückgeführt werden (Defektlokalisierung, Debugging)
- Das kann sehr aufwändig sein:
  - Das in Frage kommende Codevolumen ist evtl. sehr groß
  - Evtl. spielen mehrere Defekte zusammen
  - 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

Qualitätssicherung

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

Manuelle statische Verfahren

- Bei manuellen statischen Verfahren werden SW-Artefakte von Menschen gelesen mit dem Ziel 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

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

Durchsicht

Peer Review:

- Entwickler A hat 4 zusammengehörige Klassen fertig gestellt
  - Und erfolgreich übersetzt
- Er sendet Entwicklerin B eine Email und bittet, die 4 Klassen (insgesamt 600 Zeilen Code) zu begutachten
- B kennt die Anforderungs- und Entwurfsdokumente, aus denen sich ergibt, was die Klassen leisten sollten
  - 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

Qualitätssicherung

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

Konstruktive QS

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")
- oder auf die Verbesserung des Prozesses einer Organisation über alle Projekte hinweg:
  - **Prozessmanagement**
- Dazwischen gibt es, quasi als Rohstoff für beides, **Prozessmodelle**
  - die allgemein das Zusammenwirken von Rollen und Tätigkeiten zu einem Softwareprozess beschreiben
  - und projektunabhängig dessen grobe Leitlinien festlegen

Gemeinsamkeiten von Projekt- und Prozessmanagement

Freie Universität Berlin

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 Ursache(n) und ggf. deren Ursache(n)

- 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

Ziel konstruktiver 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

Die wichtigsten Entscheidungen für ein SW-Projekt

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

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 gut ausgeglichen werden können
- Iteration:
  - Es senkt das Risiko, wenn das Projekt in kurzen Abständen evaluierbare Versionen der Software hervorbringt

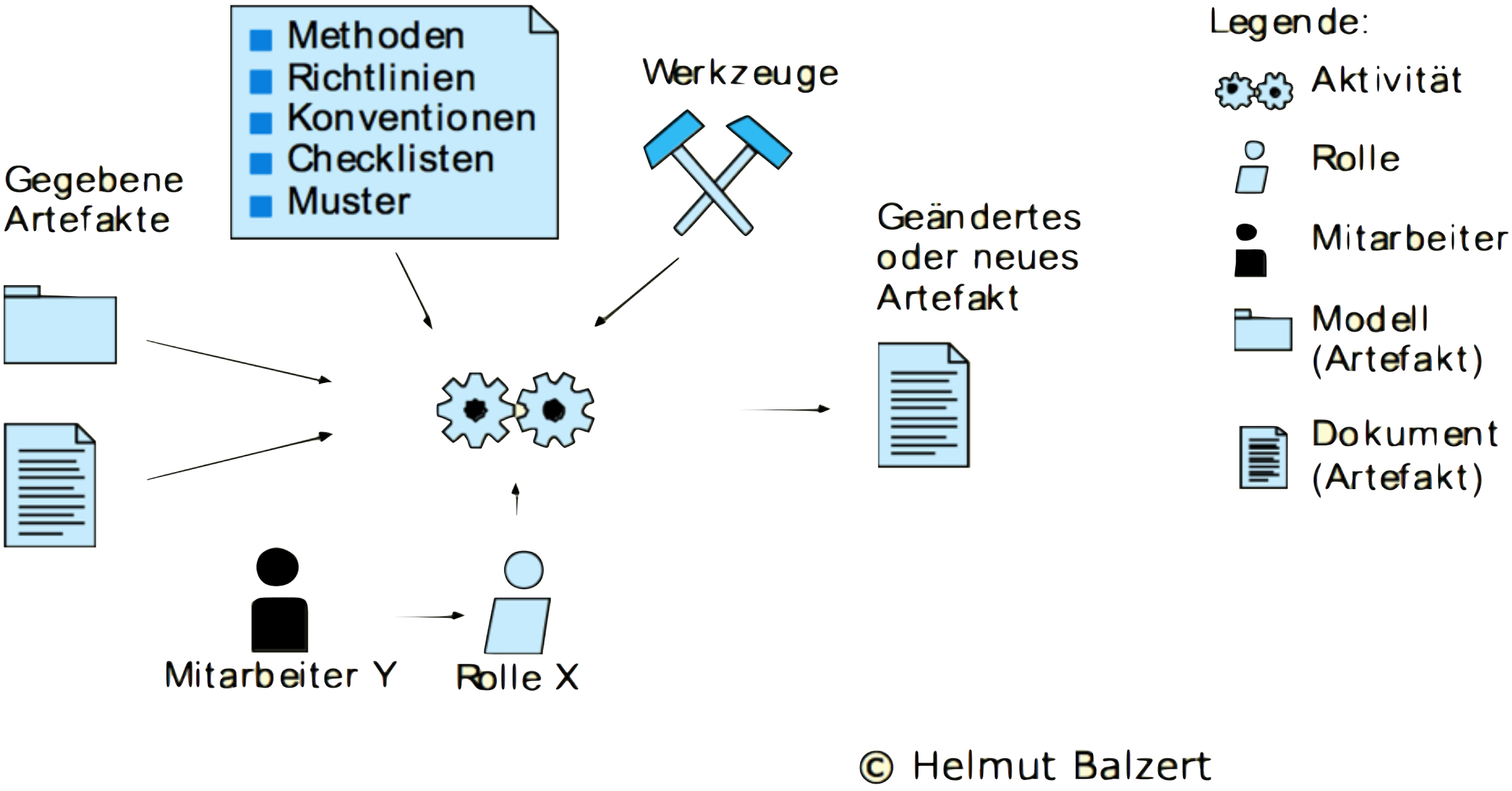
Definitionen: Prozess, Prozessmodell

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)

Softwareprozess-Modell (Software-Prozessmodell):

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



© Helmut Balzert

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 erst 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

- Bei unklaren Anforderungen:
  - Wenn Anforderungen in der Anforderungsbestimmung nicht gut verstanden werden, braucht man als Hilfe den Entwurf, die Implementierung und die Validierung
  - Im Wasserfallmodell führt das zu Chaos, weil späte Änderungen der Anforderungen total das Prozessmodell durchbrechen
    - Entweder die Arbeitsweise mit gründlich ausgearbeiteten Dokumenten wird enorm teuer
    - oder die Dokumente werden nicht mehr korrekt gepflegt
- Bei veränderlichen Anford.:
  - Das gleiche gilt, wenn sich Anforderungen irgendwann im Projektverlauf plötzlich von außen ändern können
- 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

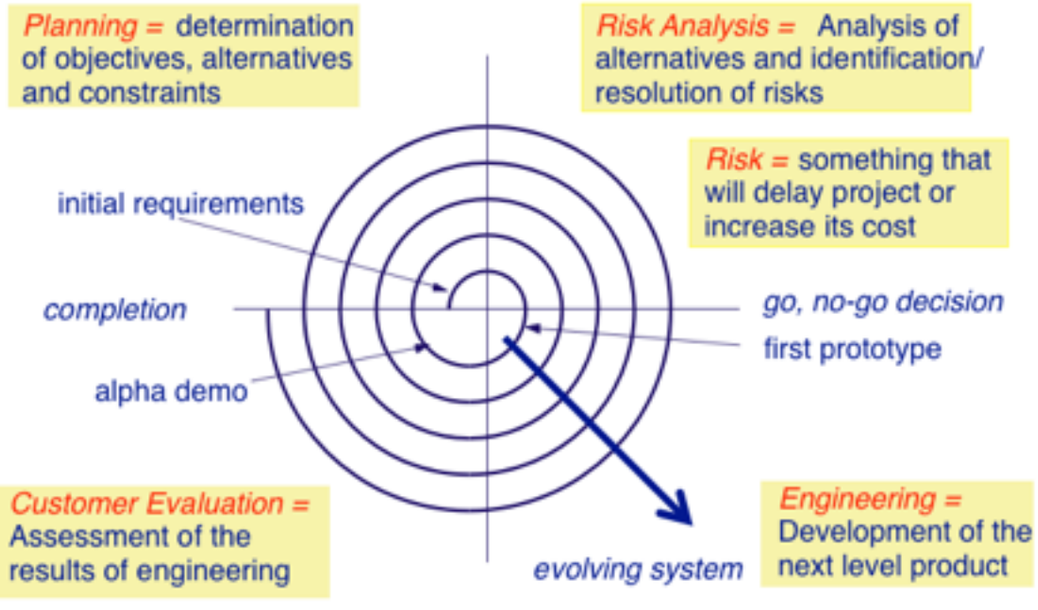
Eigentlich ist das Wasserfallmodell nur eine Legende.

1. Reparatur: Iteration

- Modernere Prozessmodelle empfehlen ein iteratives Vorgehen
  - Projektergebnis nicht "in einem Rutsch" anfertigen, sondern sich in mehreren Schritten "herantasten"
- **Vorteile**
  - Senkt Komplexität in einzelnem 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



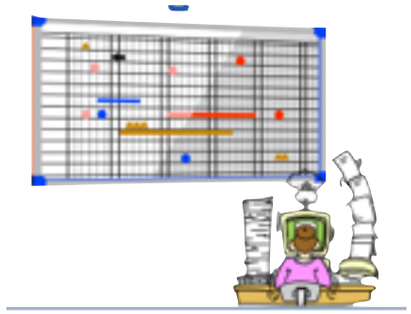
2. Reparatur: Allgemeine Ziele anstatt konkreter Pläne

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

Agile Methoden verlangen sehr enge Kommunikation!

- **zwischen <u>allen</u> Beteiligten**

Hauptunterschied zwischen Prozessmodellen: Wieviel Planung?



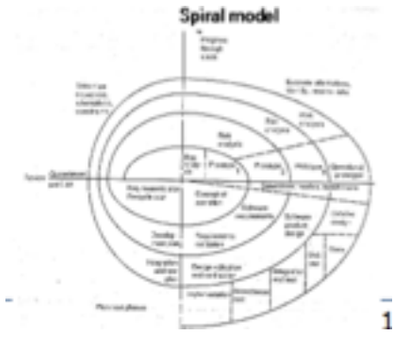
ad hoc
+ wenig Planungsaufwand
+ individuelle Freiheit
-- Ergebnis unvorhersehbar
-- abhängig von "Helden"

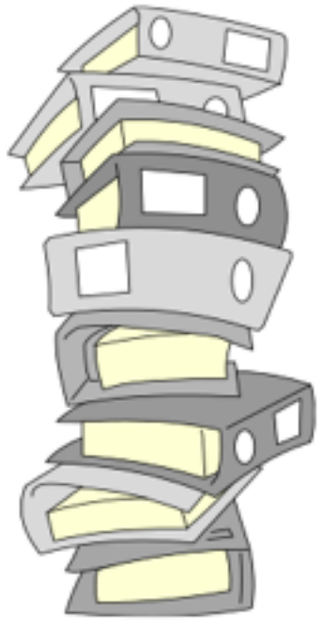
Agiles Beispiel: SCRUM
+ mittelfristig geplant
+ reagiert schnell
- hängt an MA-Qualifikation
- Endprodukt nicht spezifiziert

Meilenstein- u. Plangetrieben
+ langfristige Vorhersagen
+ gute Zustandskontrolle
- - Änderungen aufwändig
- - unrealistische Annahmen

eXtreme Programming
+ früh Kernergebnisse
+ flexibel für Änderungen
- in sehr großen Projekten Zusatzplanung nötig
- viel Selbstdisziplin nötig

Meilenstein- u. Risikogesteuert
++ Risiken aktiv ausgeräumt
+ Teilergebnisse früh
- kaum langfristig planbar
- relativ aufwändig





feingranulare Verträge
++ klare Arbeitsgrundlage
+ finanzielle Sicherheit
-- enorm aufwändig
-- Änderungen auch für AG schwer durchsetzbar

Nicht angesprochen

- 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
  - Wie schätzt man Zeitaufwände
  - Risikomanagement
- Personalmanagement
  - Persönlichkeitstypen
  - Gruppendynamik
  - Psychologische Effekte
- Methoden zur Entwicklung sicherheitskritischer Systeme
- Methoden zur Maximierung der Benutzbarkeit
- Änderungsmanagement
- Software-Evolution
  Wartung verursacht bis 70% der Gesamtkosten

Danke

Qualitätssicherung

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

Testen in SeqAn

Freie Universität Berlin

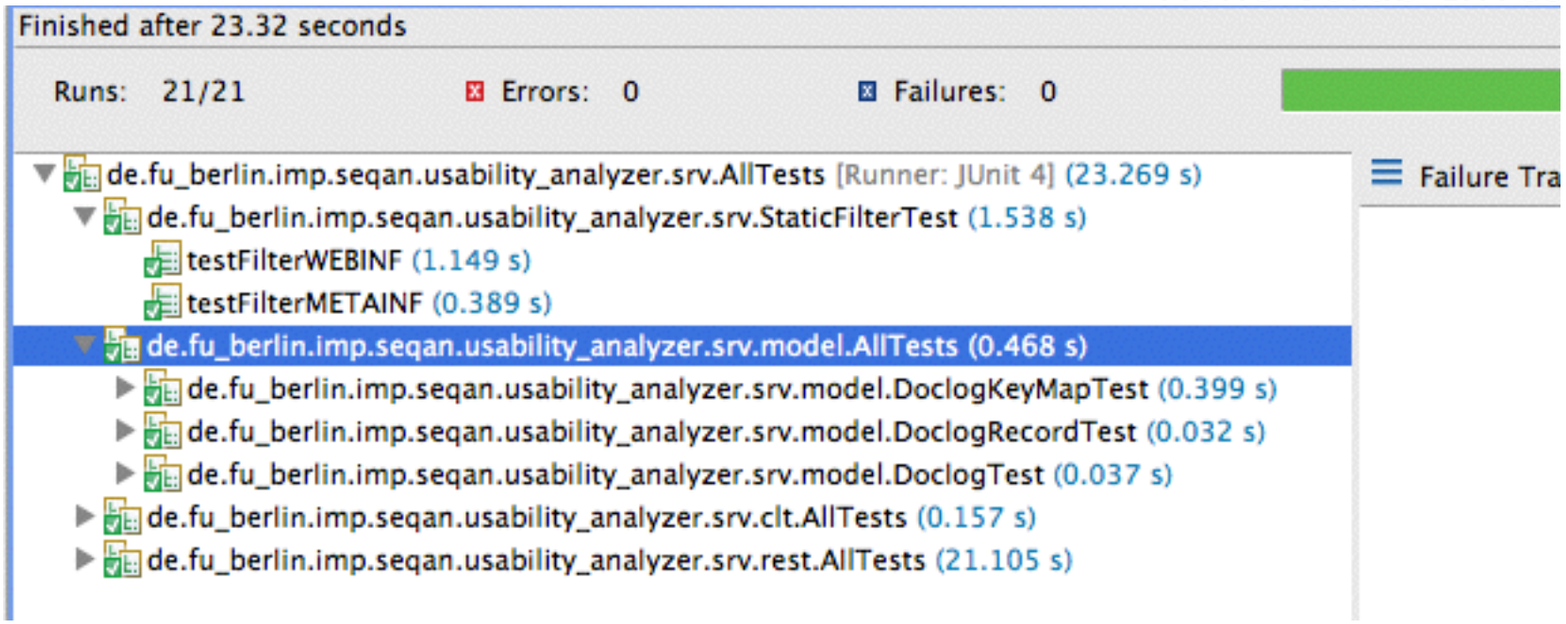
Test-Werkzeuge

Test Driven Development

Testframework-Ansatz

Testframework =
Anwendungsspezifische Bibliothek von Test-Hilfsop

- Erleichtert stark das Schreiben der Testtreiber
- Automatisiert jeden gewünschten Teilschritt nac
- Ermöglicht sauber entworfene und änderungsfre
- für GUIs (mit Aufnahme/Wiedergabe-Werkzeug) • für
- Problem:
- Hohe Vorab-Investition



http://trac.seqan.o