universitätbonn ROOTS

Kapitel 1

Software Engineering – Eine Einführung

Ziele

- Erklärung der Bedeutung von Software Engineering
- Erläuterung der Hauptprobleme von SE
- Motivation der Verwendung objektorientierter Techniken
- Definition der grundlegenden Konzepte von SE
- Exkurse
  - Software-Qualitäten
  - Software Engineering Prinzipien

Bedeutung von Software Engineering

- Software ist allgegenwärtig
  - Immer mehr Systeme werden durch Software kontrolliert

- Software ist ein wirtschaftlicher Schlüsselfaktor
  - Die Volkswirtschaft aller Industrieländer ist von Software abhängig
  - Die Ausgaben für Software haben einen beträchtlichen Anteil am Bruttosozialprodukt aller Industrieländer

- Software Engineering befasst sich mit
  - Theorien
  - Prozessen
  - Methoden
  - Werkzeugen
  - zur professionellen Softwareentwicklung

Ziele: Software-Kosten und Softwarequalität

- Softwarekosten übersteigen oft Systemkosten
  - Beispiel: PC
- Der Softwarewartung kostet mehr als die Entwicklung
  - Bis zu 80% der Gesamtkosten
- Schlechte Software schadet mehr als sie nutzt
  - Unzufriedene Benutzer
  - Direkte Schäden
- Software Engineering befasst sich mit
  - kosteneffektiver Software-Entwicklung
  - qualitativ hochwertiger Software

Einige Beispiele: Qualität und Kosten

- Jahr 1900 - Bug
  - Im Jahre 1992 erhielt Mary eine Einladung einen Kindergarten zu besuchen – im Alter von 104 Jahren!
- Fehlverhalten
  - Automatisiertes Gepäcksystem am Flughafen von Denver beschädigte Koffer
  - 3,2 Mio. $ über dem Budget, 16 Monate verspäteter Start mit zum größten Teil manuellem System
- Unnötig komplexe Lösung
  - Das C-17 Transportflugzeug nutzt 19 Onboard-Computer, 80 Mikroprozessoren und 6 Programmiersprachen
  - Es war 500 Mio $ über dem Budget
- Interface-Missbrauch
  - Wissend, dass es ein System gab, das die Abfahrt des Zuges mit offenen Türen verhinderte, fixierte ein Zugführer den Startknopf mit Klebeband in Startstellung.
  - Als er den Zug verließ um eine klemmende Tür zu schließen, ...
  - ... fuhr der Zug ohne ihn los als die Tür nicht mehr klemmte.

Er fand dies eine besonders clevere Art sicherzustellen, dass der Zug sofort losfuhr sobald die letzte Tür geschlossen war.

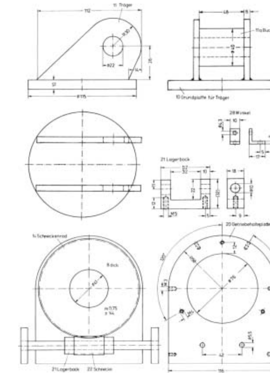
Hauptherausforderungen von SE: Komplexität und Änderung

- Komplexität von Softwaresystemen
  - Viele Funktionen
  - Viele, möglicherweise kollidierende Ziele
  - Viele Komponenten → Komplexität der Zusammenstellung
  - Viele Teilnehmer → Komplexität der Kommunikation
- Keine Einzelperson kann ein ganzes System verstehen

- Dauernde Änderung von Softwaresystemen
  - Software modelliert einen Ausschnitt der „realen Welt"
  - Die reale Welt ändert sich (Gesetzgebung, Geschäftsabläufe, ...)
  - Der Ausschnitt ändert sich (mehr Funktionalität hinzufügen, Systeme integrieren, ...)
  - Die Implementierungstechnologie ändert sich (neue Sprachen, Komponenten, ...)
  - Das Team ändert sich (Personen gehen, neue kommen hinzu, ...)
  - Das Management wechselt (neue Geschäftsausrichtung, ...)
- Alles kann sich jederzeit ändern!

Charakteristiken von anderen Ingenieurswissenschaften

- Domäne
  - Klar definierte Probleme
  - Ein Produkt muss *gebaut* werden
  - Hohe Qualitätsanforderungen

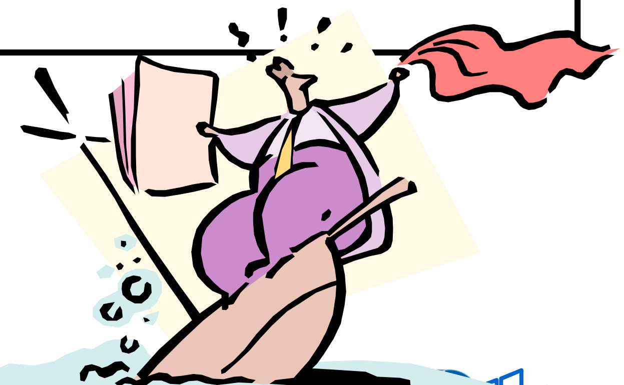


- Methoden
  - Systematische Verfahren und ihre disziplinierte Anwendung
  - Standardschnittstellen, -komponenten und -prozesse
  - Wissen um bereits verfügbare Teilkomponenten und gezielter Einsatz fertiger (Teil-) Lösungen
  - Empirische Methoden zur Bewertung von Lösungen

Software Engineering ist anders

● Andere Ingenieursbereiche	● Software Engineering
◆ Herstellung bestimmt die Endkosten	◆ Herstellung ist eine einfache Duplizierung
◆ Änderungen sind weniger häufig ⇨ 2000 Jahre von der Euklidischen zur nicht-Euklidischen Geometrie	◆ Änderungen geschehen dauernd ⇨ Manchmal innerhalb von Stunden (Kunde hat seine Meinung geändert)
◆ Änderungen sind möglich, aber sehr teuer ⇨ Redesign wird gründlich durchdacht ⇨ Auswirkungen werden genau analysiert	◆ Änderungen sind einfach ⇨ Kein durchdachtes Redesign ⇨ Auswirkungen werden nicht ausreichend bedacht

Andauernde Änderungswünsche zusammen mit der Leichtigkeit, Änderungen durchzuführen, führen zu ungenügend überdachten, ad-hoc „Lösungen“.



Kreatives Chaos versus diszipliniertes Engineering

Programmieren ist ein kreativer Akt.
Ich kann meine Kreativität nicht von Regeln begrenzen lassen.
Ich bin ein Künstler.

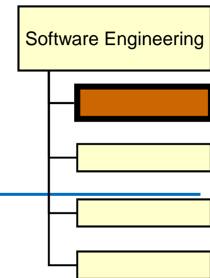
Wenn weniger Programmierer Kreativität mit chaotischer Vorgehensweise verwechseln würden, gäbe es mehr **brauchbare Software**!

Was ist Software-Engineering?

Was ist Software Engineering

- Modellierung
  - Modellierung ist das wichtigste Mittel um mit Komplexität umzugehen
- Problemlösung
  - Suche nach akzeptablen Lösungen erfordert experimentellen Vergleich von Modellen
- Wissensakquisition
  - Modellierung erfordert das Sammeln von Daten, deren Organisation und Formalisierung
- Rationale Management
  - Dokumentation der Überlegungen / Gründe für Entscheidungen erleichtert Verständnis der Auswirkungen von späteren Änderungen

Modellierung (1)



- ... hilft die Welt zu verstehen
  - Modelle nicht mehr existenter Systeme:
    - ⇨ Modell der Dinosaurier, basierend auf Knochen, Zähnen und den Regeln der Anatomie
  - Modelle von Systemen die angeblich existieren:
    - ⇨ Modell von Materie und Energie basierend auf Experimenten mit Teilchenbeschleunigern
  - Modelle realer Systeme:
    - ⇨ Software basierend auf Betrachtung der Anwendungsdomäne und Benutzer
    - ⇨ "Participatory design" (Beteiligungsorientiertes Design) als Weg, die relevanten Aspekte der Anwendungsdomäne zu verstehen
- ... möglichst einfache aber ausreichend genaue Darstellung des Systems
  - wichtigstes Mittel mit Komplexität umzugehen ☺
  - Abstraktion hilft den Blick auf die relevanten Aspekte zu richten
- ... ist ein Hauptgrund für Änderungen ☹
  - Je abstrakter desto mehr implizite Annahmen liegen dem System zugrunde.
  - Je mehr Annahmen desto mehr Potential für Änderungen

Was wir modellieren

Objektorientierte Modellierung

- Modellierung der Anwendungsdomäne
  - Problemorientierte Sicht
- Modellierung des Systems
  - Lösungsorientierte Sicht
  - Vergleich verschiedener Lösungen verlangt die Modellierung verschiedener alternativer Systeme

➔ „Problem domain object model"
  - Objekte
  - Beziehungen

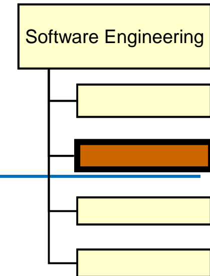
➔ „Solution domain object model"
  - Objekte
  - Beziehungen
  - Interaktionen

➔ Vorteile der OO Modellierung
  - Erstellung der beiden Modelle mit gleichen Mitteln möglich
  - Modell der Lösung ist eine Erweiterung des Problem-Modells
  - Nachvollziehbarkeit (‚Traceability')

„Modellierung“ in dieser Vorlesung

- Objektorientierte Konzepte
  - Technische Grundlagen
- Die Unified Modelling Language (UML)
  - Standardisierte graphische Notation für OO Modelle
- Objektorientierte Modellierung
  - Anwendungs-Prinzipien
- Entwurfsmuster (Design Patterns)
  - Typische „gute Entwurfs-Lösungen“
- Refactoring
  - Verletzung von Designprinzipien erkennen („Bad smells“)
  - Entsprechende verhaltenserhaltende Restrukturierung („Refactoring“)

Problemlösung



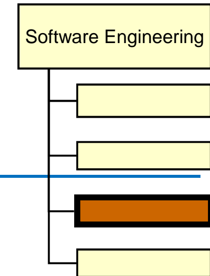
- Begrenzte Ressourcen und unvollständiges Wissen erfordern
  - ... den Vergleich verschiedener Lösungen/Modelle
  - ... empirische Bestimmung von Alternativen

- Techniken um Angemessenheit von Modellen zu bewerten
  - Benutzerzentrierter Entwurf (Participatory Design)
    - ⇨ Den Benutzer permanent in das Projekt einbinden und die Entwicklung leiten lassen
  - Analyse-Überprüfung (Analysis Review)
    - ⇨ Vergleiche Modell des Anwendungsbereichs mit Realität aus Sicht des Benutzers
  - Entwurfs-Überprüfung (Design Review)
    - ⇨ Vergleich des Lösungsmodells (solution domain model) mit Projektzielen
  - Implementierungs-Überprüfung (Code Review) und Testen
    - ⇨ System wird auf Basis des Lösungsmodells (solution domain model) validiert
  - Projektmanagement
    - ⇨ Vergleiche Prozessmodell (Zeitplan, Budget, ...) mit der Realität des Projektes

„Problemlösung“ in dieser Vorlesung

- Participatory design
  - Den Benutzer permanent in das Projekt einbinden und die Entwicklung leiten lassen
  - → Extreme Programming: Benutzer / Kunde als Teil des Teams
- Analysis-, Design-, Code-Review
  - → Extreme Programming: permanenter Review-Prozess durch „Pair Programming“
- Projektmanagement
  - Vergleiche Prozessmodell (Zeitplan, Budget, ...) mit der Realität des Projektes

Wissensakquisition



- Wissensakquisition ist nicht linear
  - Das Hinzukommen einer neuen Information kann alles vorher erworbene Wissen unbrauchbar machen → Hohes Fehlerrisiko ☹
- Risk-based development (Risikoorientierte Softwareentwicklung)
  - ... zielt darauf ab, mit hohem Risiko behaftete Komponenten und Aktivitäten zu erkennen.
  - ... plant sie frühzeitig im Projekt ein
- Issue-based development (Problemorientierte Softwareentwicklung)
  - ... erkennt an, dass jede Aktivität jede andere beeinflusst
    - ⇨ Analyse
    - ⇨ Systemdesign, Objektdesign
    - ⇨ Implementierung, Testen
    - ⇨ Auslieferung
  - ... ersetzt lineare durch parallele Ausführung von Aktivitäten
  - ... erleichtert es, auf Änderungen zu reagieren
  - ... ist leider schwieriger zu koordinieren

„Wissensaquisition“ in dieser Vorlesung

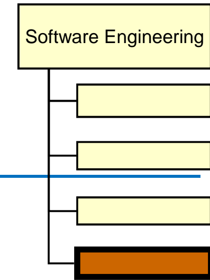
- Prozessmodelle
  - Risk-based development
  - Issue-based development
- Extreme Programming und „Agile Prozesse“
  - Verbindung beider Ansätze
- Es geht um den Umgang mit Änderungen!
- Das neue Wissen von heute kann
  - die gute Entscheidung von gestern hinfällig werden lassen.
  - die gestern verworfene Alternative neu ins Spiel bringen.

‚Rationale Management‘

- Rationale (engl.) = Begründung, Erklärung, Motivation

- Rationale Management = Erfassung von
  - Problemen
  - Lösungsalternativen
  - Argumenten für und Wider der Alternativen
  - getroffene Entscheidung
  - Begründung für die Entscheidung

‚Rationale Management': Sinn und Zweck



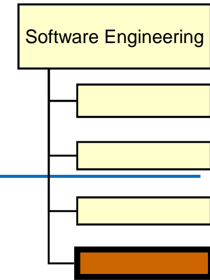
Statische Problem- und Lösungs-Domänen

- Mathematikbücher sind voller Beweise. Nie aber ist festgehalten, warum ein Beweis gerade so und nicht anders geführt wurde.
- Das ist nicht weiter schlimm, da sich die zugrundeliegenden Modelle sehr selten ändern
  - 2000 Jahre von euklidischer zu nicht-euklidischer Geometrie
  - 1500 Jahre von geozentrischem zu heliozentrischem Weltmodell

Dynamische Problem- und Lösungs-Domänen → Rationale Management

- Dokumentation der Gründe für Architektur und Entwurfsentscheidungen wichtig
- Sie ermöglicht die Auswirkung späterer Änderungen dieser Entscheidungen nachzuvollziehen
- Sie reduziert den Änderungsaufwand und die Gefahr von Fehlentscheidungen

Rationale Management: Problem



- Dokumentation von Alternativen
  - Aktuelles Lösungsmodell beinhaltet nur die gewählte Alternative
  - Betrachtet wurden aber evtl. N andere Möglichkeiten
  - Rationale Management muss also N andere Alternativen mit erfassen
- Dokumentation der Gründe gegen *jede* verworfene Alternative
  - Wichtigste Entscheidungshilfe bei Änderung: Ist etwa eine der vorherigen Annahmen ungültig geworden und somit auch die Schlussfolgerung die damals gezogen wurde?
- Dokumentation der getroffenen Entscheidung
  - Zusammen mit benutzten Bewertungskriterien
- Dokumentation impliziter Entscheidungen
  - Entscheidung aus Erfahrung oder Intuition, ohne explizit Alternativen zu betrachten
  - Nachträgliches Explizitmachen sehr aufwendig

➔ Rationale Management erfordert enormen Zusatzaufwand

Software-Qualitäten & Engineering-Konzepte

Exkurs

Folgende Inhalte sind „nur“ eine gebündelte Vorschau auf Themen die noch kommen.

Software Engineering Konzepte

- Teilnehmer und Rollen
- Systeme und Modelle
- Arbeitsprodukte
- Aktivitäten, Ressourcen
- Ziele, Anforderungen, Randbedingungen
- Notationen, Methoden, Methodologien

In Anlehnung an IEEE Standards zu Software Engineering (s. [IEEE 1997] in Brügge/Dutoit)

Software Engineering Konzepte

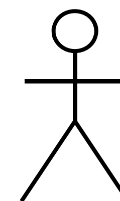
- Teilnehmer und Rollen
- Systeme und Modelle
- Arbeitsprodukte
- Aktivitäten, Ressourcen
- Ziele, Anforderungen, Randbedingungen
- Notationen, Methoden, Methodologien

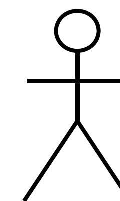
In Anlehnung an IEEE Standards zu Software Engineering (s. [IEEE 1997] in Brügge/Dutoit)

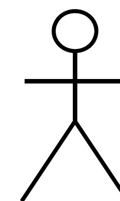
Teilnehmer und Rollen

- Teilnehmer (Participant)
  - Jede natürliche oder juristische Person die am Projekt teilnimmt
  - Beispiel: Alice, John, Marc, Theo, Lara, die Deutsche Bahn AG
- Interessensvertreter (Stakeholder)
  - Repräsentant einer Personengruppe, die ein eigenes Interesse an dem Projekt oder Produkt hat
  - Beispiel: Projektmanager, Entwickler, zukünftige Nutzer, durch das System zukünftig arbeitslos werdende Angestellte, ...
- Rolle (Role)
  - Die Rolle die ein Teilnehmer im Projekt spielt
  - Beispiel: Kunde, Endbenutzer, Entwickler, Projektmanager, Analyst, ...

- Oft abstrahieren wir Teilnehmer durch die Rollen die sie im Projekt oder beim Umgang mit dem System einnehmen







System und Modelle

- System
  - „reale Welt“
  - Beispiel 1: Fahrkartenautomat
  - Beispiel 2: Projekt

- Modell
  - Abstraktion des Systems
  - Beispiele zu 1: Blaupausen, Schaltpläne, Programme, UML-Diagramme, ...
  - Beispiele zu 2: Zeitpläne, Kostenplan, ...

Produkte (Products)

- Produkt
  - Während eines Projekts entwickelte Artefakte
  - Beispiele: Anforderungsdokumente, Programme, Testauswertungen, ...
- Interne Produkte
  - Nur für das Projektteam bestimmt
  - Beispiele: Demo-Prototypen, Testfälle, Testergebnisse, ...
- Auszuliefernde Produkte (Deliverables)
  - Für den Kunden bestimmt
  - Üblicherweise vertraglich festgelegt, zusammen mit Zeitplan für Abgabe
  - Beispiele: Fahrkartenautomat, seine Software, Handbücher

Aktivitäten, Aufgaben und Ressourcen

- Aktivität (Activity)
  - umfasst Menge von Aufgaben, die auf das gleiche Ziel gerichtet sind
  - Beispiel: Anforderungs-Erhebung, Auslieferung, Management
  - kann aus Teilaktivitäten bestehen
  - Beispiel: Auslieferung umfasst Softwareinstallation und Benutzerschulung
- Aufgabe (Task)
  - Atomare Aufgabe, die zugeteilt und erledigt wird
  - verwendet Ressourcen (Resources)
  - erzeugt Produkte
- Ressourcen (Resources)
  - Alles was zur Durchführung von Aufgaben benötigt wird
  - Beispiele: Teilnehmer, Zeit, Material, ...
- Projektplanung
  - Planung von Aktivitäten und Aufgaben
  - Zuordnung von Ressourcen für Aufgaben

Ziele, Anforderungen, Randbedingungen

- Ziele (Goals)
  - Leitprinzipien bzw. Hauptziele des Projektes
  - Definieren die wichtigen Attribute des Systems
  - Beispiel 1: Leitziel der Space-Shuttle Entwicklung ist hohe Sicherheit
  - Beispiel 2: Leitziel der Entwicklung eines Fahrkartenautomaten ist hohe Verfügbarkeit
  - **Unklar definierte Ziele und Zielkonflikte** machen einen großen Teil der Komplexität mancher Systeme aus
- Anforderungen (Requirements)
  - Eigenschaften, die das System besitzen muss
  - Funktionale Anforderungen (z.B. Ticket ausgeben)
  - Nicht-Funktionale Anforderungen (schnelle Reaktionszeit, Touchscreen)
- Randbedingungen (Constraints)
  - Weitere Bedingungen, die für den Benutzer nicht sichtbar sind
  - Z.B. Einbindung von Altsystemen

Notation, Methode, Methodologie

- Notation
  - Sprache zur graphischen oder textuellen Darstellung eines Modells
  - Beispiel: UML

- Methode
  - Reproduzierbares Verfahren zur Problemlösung
  - Beispiel: Kochrezept, Sortieralgorithmus, Rationale Management, Konfigurationsmanagement

- Methodologie
  - Menge von sich ergänzenden Methoden zur Lösung einer bestimmten Problemklasse
  - Beispiele: Unified Process, OMT, ...

universitätbonn ROOTS

Software-Qualitäten & Engineering-Prinzipien

Exkurs

Folgende Inhalte sind „nur“ eine gebündelte Vorschau auf Themen die noch kommen.

Überblick

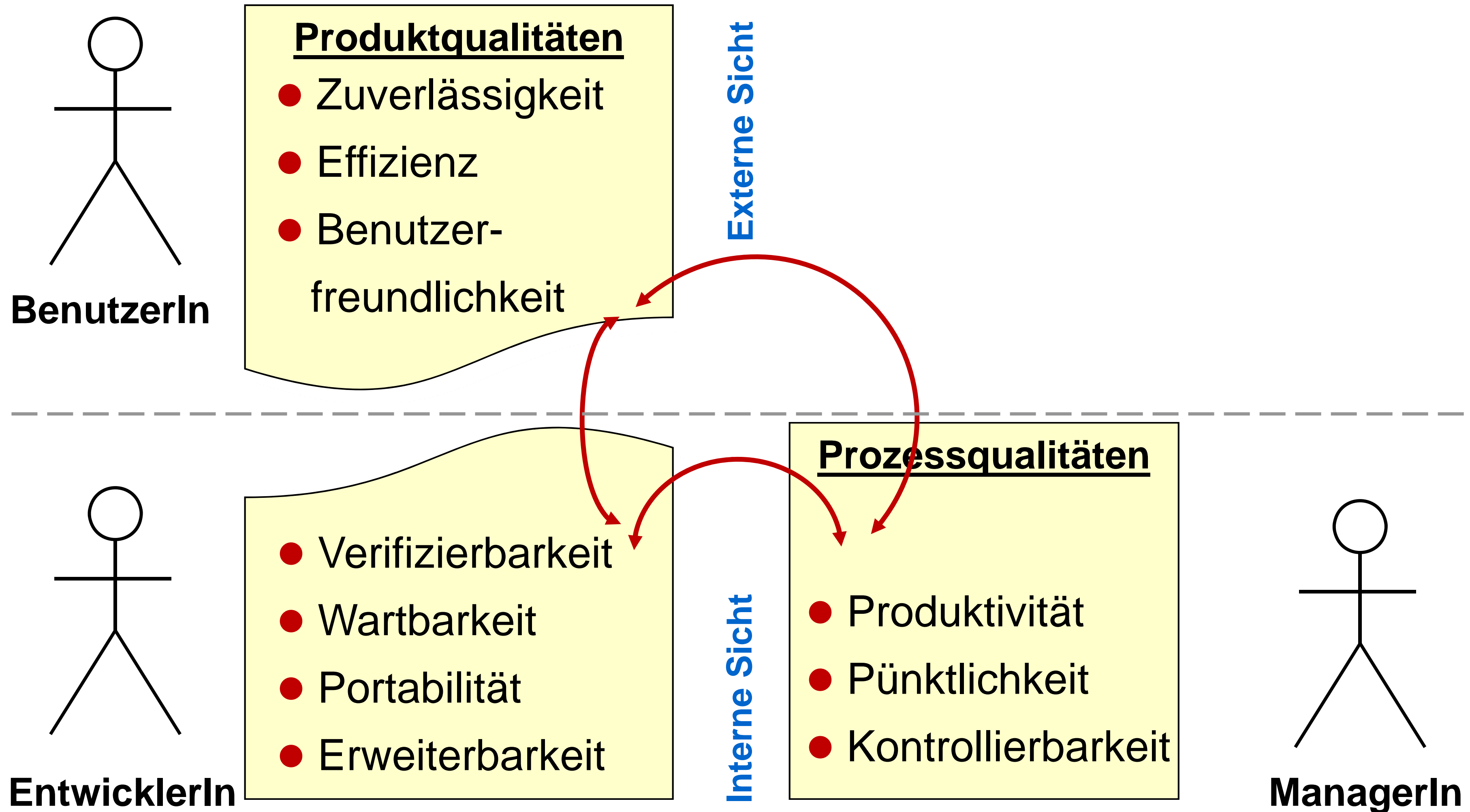
Das Ziel: “qualitätsorientiertes Software Engineering"

- Softwarequalität und Prozessqualität
  - Verschiedene Qualitätssichten durch verschiedene Interessensvertreter
  - Verschiedene Qualitäten in verschiedenen Anwendungsbereichen wichtig

Wie man “qualitätsorientiertes Software Engineering“ erreicht

- Engineering-Prinzipien
  - „Rigor and formality“
  - Trennung der Belange
  - Modularität
  - Abstraktion
  - Erwartung von Änderungen
  - Allgemeingültigkeit
  - Inkrementelle Vorgehensweise

Software-Qualitäten: Verschiedene Sichten

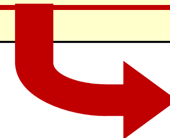


Qualitätsorientiertes Software Engineering

- Die gewünschten Qualitäten des
  - Softwareproduktes
  - Softwareprozesses

- ... müssen
  - von Anfang an klar sein
  - die treibende Kraft für den gesamten Prozess sein

> Das Erreichen der Qualitäten
> hängt ab vom Grad der Einhaltung
> guter Ingenieurs-Prinzipien



nächster Abschnitt

„Rigor and Formality“

- Formality = engl. Förmlichkeit, Formalismus
  - Formalisierung hilft Dinge präzise auszudrücken
  - ... dadurch Widersprüche und Inkonsistenzen zu entdecken
  - ... und gegebenenfalls sogar beweisen zu können.
- Beispiele
  - Typsysteme von Programmiersprachen
  - Sicherheitskritisches Verhalten
  - Nebenläufigkeit / Parallelität

- Rigor = engl. Härte, Starre, Strenge
  - gemeint ist konsequentes / diszipliniertes, systematisches Vorgehen
  - Besonders wichtig für Formalisierungen aber nicht nur.
  - Konsequentes, diszipliniertes Vorgehen im Laufe des Softwareentwicklung ist genauso wichtig. → Extreme Programming

Ingenieurs-Prinzipien: “Separation of Concerns” (Trennung der Belange)

- Möglichkeiten der Trennung
  - Trennung nach Zeit
    - ⇨ z.B. Zeitplanung des täglichen Lebens
  - Trennung nach Qualitäten
    - ⇨ z.B. anfängliche Konzentration auf bestimmte Qualitäten, später dann auf andere
  - Trennung nach Größe
    - ⇨ Entwicklung von Teilen des Systems geschieht getrennt

- Trennung ermöglicht ...
  - ... die Kooperation verschiedenster Leute im selben Projekt
  - ... die Bewältigung der Komplexität eines Projektes

> Der einzige Weg Komplexität zu bewältigen ist die Trennung in verschiedene Belange.

Ingenieurs-Prinzipien: “Separation of Concerns”

- Vergangenheit
  - Prozedurale Dekomposition
  - Funktionale Dekomposition
- Gegenwart
  - objekt-orientierte Dekomposition
- Problem
  - "tyrany of predominant decomposition"
  - unabhängig von Dekompositionsart gibt es immer „überschneidende Belange" („crosscutting concerns“)
    - ⇨ Sicherheit, Persistenz, ...
- Aktuelle Weiterentwicklung / Zukunft (?)
  - mehrdimensionale Trennung der Belange
  - "aspekt-orientierte Softwareentwicklung"
  - → http://aosd.net
  - → http://aosd.net/conference: AOSD.06, Universität Bonn, März 2006

Ingenieurs-Prinzipien: Modularität (2)

- Dekomposition
  - Top-down Methode
  - Aufteilen des Ursprungsproblems
  - Rekursive Dekomposition von Unterproblemen
  - Divide-And-Conquer

- Komposition
  - Bottom-up Methode
  - Elementare Komponenten werden zuerst entworfen
  - Projekt wird aus einzelnen Modulen zusammengesetzt

- Möglichkeit des Verstehens
  - Hilft beim modifizieren eines Systems

Ingenieurs-Prinzipien: Modularität (3)

- Kohäsion (engl. „Cohesion") → Zusammenhalt
  - Elemente eines Moduls werden gruppiert
  - Gruppen werden aus logischen Gründen gebildet
    - ⇨ Elemente kooperieren zum Erreichen eines Ziels
  - Kohäsion ist ein Maß für die gegenseitigen Abhängigkeiten von Elementen einer Gruppe
  - Hohe Kohäsion → hoher Zusammenhalt der Gruppe

- Kopplung (engl. „Coupling") → Abhängigkeiten
  - Kopplung ist ein Maß für die Abhängigkeiten zwischen den Modulen
  - Starke Kopplung → Viele unerwünschte Abhängigkeiten

> Modularität kann nur erreicht werden, wenn die Module einen hohen Zusammenhalt und geringe Abhängigkeiten besitzen.

Ingenieurs-Prinzipien: Abstraktion

- Definition
  - Die wichtigen Aspekte herausfinden unter Vernachlässigung der Details.
- Generelle Idee
  - Entwurf von Modellen als Abstraktion der Realität
- Gut für…
  - … den erfolgreichen Umgang mit der Komplexität im Allgemeinen.
  - … das Sichtbarmachen wesentlicher Aspekte einer Aufgabe.
- Gilt für…
  - … Softwareprozesse
    - ⇨ z.B. Kostenabschätzung (nur über Schlüsselfaktoren)
  - … Softwareprodukte
    - ⇨ z.B. Klassen sind Abstraktionen “realer” Objekte

Ingenieurs-Prinzipien: Voraussehen von Änderungen

- Generelle Idee
  - Software muss einfach erweiterbar und änderbar sein
- Änderungen in den Softwareprodukten
  - Benutzerfeedback kann zu neuen Anforderungen führen
  - Nicht alle Anforderungen waren im ersten Release erfüllt
- Änderungen im Softwareprozess
  - Personalwechsel kann zu unvollständigem Wissen führen

Ingenieurs-Prinzipien: Allgemeinheit

- Prinzip
  - Probleme wenn möglich verallgemeinern
- Pro
  - Mehr Wiederverwendungspotential
  - Möglicherweise weniger komplexe Lösung
  - Lösung ist möglicherweise durch bereits fertige Komponenten möglich
- Kontra
  - Könnte mehr Kosten verursachen in Bezug auf Geschwindigkeit, Speicherbedarf, Entwicklungszeit, Einarbeitungsaufwand, Wartungsaufwand, ...
  - Übermäßige Allgemeinheit ist eher schädlich...

Allgemeinheit ist ein fundamentales Prinzip in der Entwicklung von Werkzeugen und Paketen.

Ingenieurs-Prinzipien: Inkrementelle Vorgehensweise

- Definition
  - Ein Ziel wird durch immer bessere sukzessive Näherungen erreicht.

- Vorversionen der Software werden früh veröffentlicht
  - ⇨ Frühes Feedback der Kunden
  - ⇨ Vorher unbekannte Anforderungen stellen sich heraus

> Inkrementelle Vorgehensweise ist eine Grundvoraussetzung von Erweiterbarkeit