

## Kapitel 1

# Software Engineering – Eine Einführung

– Stand: 12.10.2010 –

---

# 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.

# Haupt Herausforderungen von SE: Komplexität und Änderung

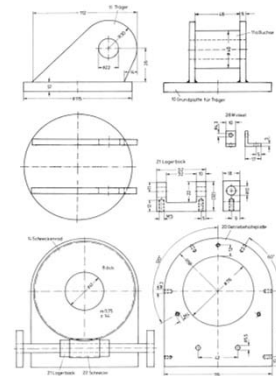
---

- 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 Ingenieurwissenschaften

- 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 Ingenieurbereiche

◆ Herstellung bestimmt die Endkosten

◆ Änderungen sind weniger häufig

⇒ 2000 Jahre von der Euklidischen zur nicht-Euklidischen Geometrie

◆ Änderungen sind möglich, aber sehr teuer

⇒ Redesign wird gründlich durchdacht

⇒ Auswirkungen werden genau analysiert

## • Software Engineering

◆ Herstellung ist eine einfache Duplizierung

◆ Änderungen geschehen dauernd

⇒ Manchmal innerhalb von Stunden (Kunde hat seine Meinung geändert)

◆ 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

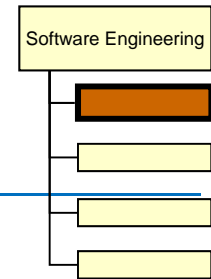
- Wissensakquisition

- ◆ 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

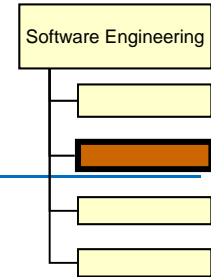
- Modellierung der **Anwendungsdomäne**
    - ◆ Problemorientierte Sicht
  - Modellierung des **Systems**
    - ◆ Lösungsorientierte Sicht
    - ◆ Vergleich verschiedener Lösungen verlangt die Modellierung verschiedener alternativer Systeme
- „Problem domain object model“
    - ◆ Objekte
    - ◆ Beziehungen
  - „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

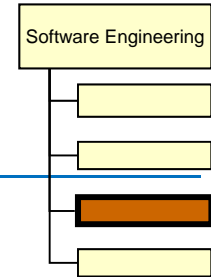
# „Problemlösung“ in dieser Vorlesung

---

- 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.

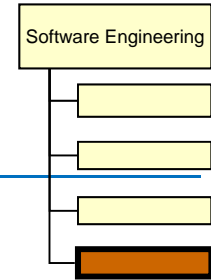
# „Rationale Management“

---

- 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

---



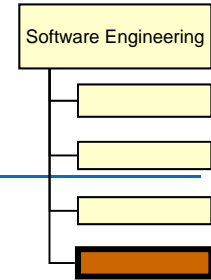
## Statische Problem- und Lösungs-Domänen

- 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