Software Engineering

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl für Informatik 11 (Software Engineering)

Francesca Saglietti
saglietti@informatik.uni-erlangen.de

Ringvorlesung Orientierung

Zielsetzung des Software Engineering

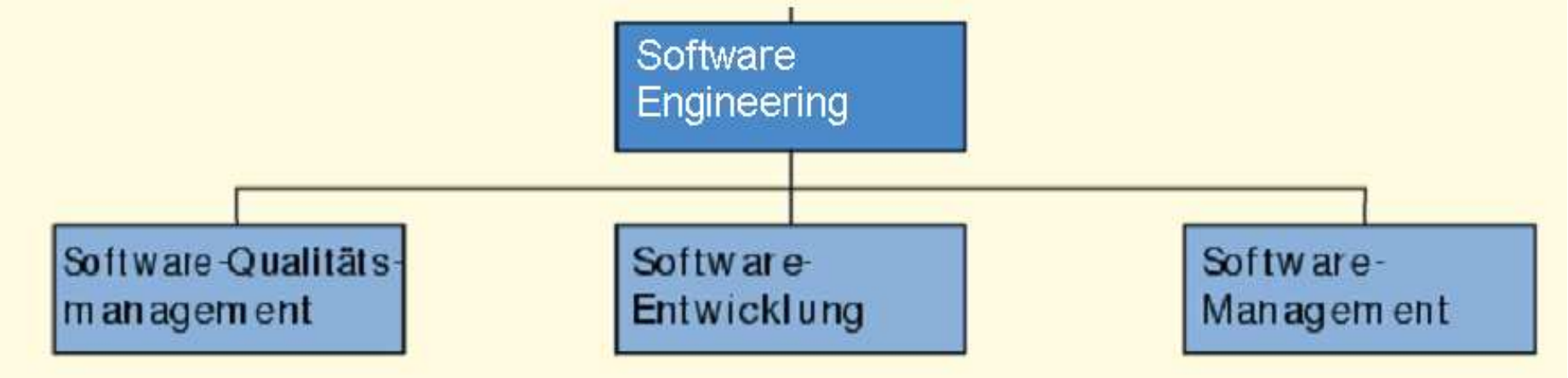
Gegenstand des **Software Engineering** ist
die **ingenieurmäßige** Entwicklung
komplexer, umfangreicher Softwaresysteme
hoher Qualität unter Berücksichtigung
der einzusetzenden **Arbeits-** und **Zeit**ressourcen.

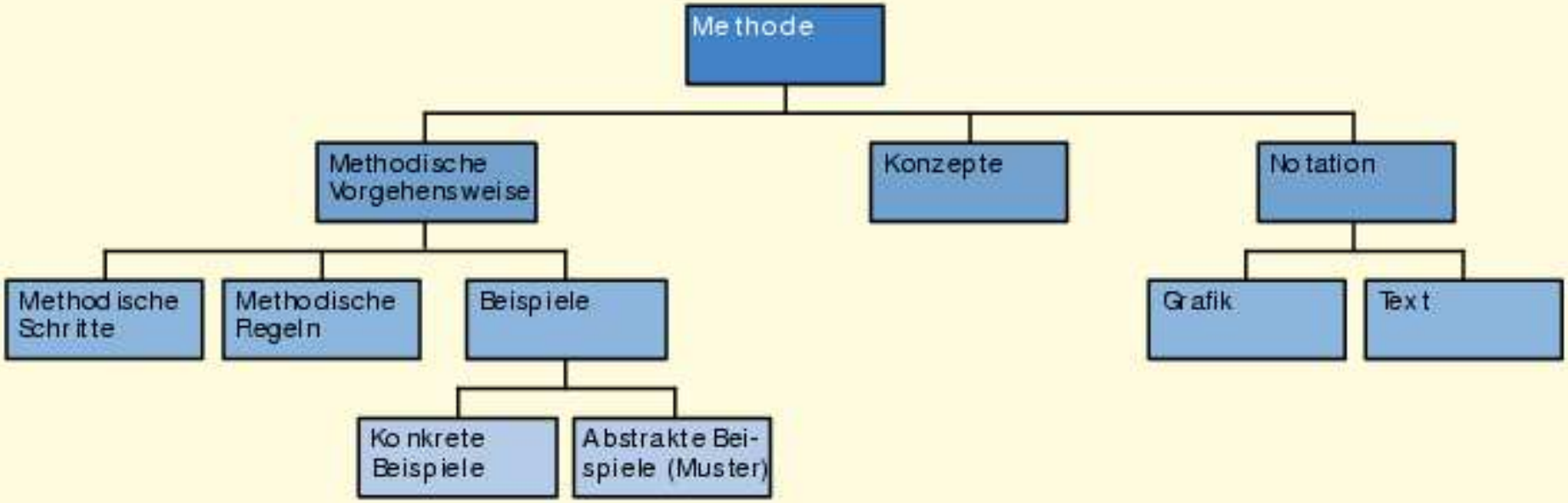
1. Problemkomplexität	Größe [SW] > 20 K LOC
2. Qualitätsvorgaben	Zuverlässigkeit, Wartbarkeit
3. Kostenlimits	Geld, Personal
4. Terminvereinbarungen	Lieferung, Genehmigung

Vorkenntnisse

- Zum Verständnis der ingenieurwissenschaftlichen Prinzipien zur Softwareerstellung sind Kenntnisse aus dem Grundstudium erforderlich:
  - vor allem Programmierkenntnisse,
  - aber auch Kenntnisse zu Programmiersprachen und –konzepten
- Aus diesem Grund werden sich die Lehrveranstaltungen des Lehrstuhls hauptsächlich an Studenten des Hauptstudiums richten, die diese Grundkenntnisse bereits erworben haben.

Software-orientiertes Fach




Methode
Methodische Vorgehensweise
Konzepte
Notation
Methodische Schritte
Methodische Regeln
Beispiele
Grafik
Text
Konkrete Beispiele
Abstrakte Beispiele (Muster)

Hauptvorlesung

- Grundbaustein der Lehre:
  jährlich wiederkehrende, 4stündige Vorlesung

- **Grundlagen des Software Engineering**
  soll auch solchen Zuhörern,
  die eine einzige Lehrveranstaltung zur SW-Technik besuchen,
  ein breit gefächertes Bild vorhandener
  ingenieurwissenschaftlicher Ansätze und ihrer Grenzen bieten.

- + 2 stündige Übung (derzeit 2 Übungsgruppen)

Ziel der Vorlesung

- Die Vorlesung befasst sich mit dem gesamten Software -Lebenszyklus und bietet eine Übersicht
  - konstruktiver und
  - analytischer Prinzipien und Verfahren;

- insbesondere werden
  - phasenspezifische und übergreifende Ansätze klassifiziert und eingeordnet,
  - ihre Nutzen, Grenzen und Komplementarität aufgezeigt,
  - ihre Eignung in Abhängigkeit von den vorliegenden Anforderungen bewertet.

Gliederung

- Einführung: Charakterisierung des Software Engineering
  - Definitionen, historischer Hintergrund
  - Analogie zu klassischen Ingenieurwissenschaften
  - Unterschiede zu anderen Ingenieurwissenschaften
  - Grundbegriffe
  - Phasen des Software-Lebenszyklus
  - Prozessmodelle
- Konstruktive Verfahren
  zur Vermeidung von Fehlern bei der Software-Erstellung
  - Anforderungsanalyse
  - Spezifikation
  - Entwurf
  - Codierung

Gliederung 2

- Analytische Verfahren
  zur Erkennung von Software-Fehlern
  - Informelle Verfahren
  - Statische Analyse
  - Softwaremetriken
  - Testen
- Verifikationsverfahren
  zum Nachweis der Software-Fehlerfreiheit
- Quantitative Verfahren
  zur Ermittlung von Qualitätskenngrößen
- Redundanzbasierte Verfahren
  zur Tolerierung von SW-Fehlern
- Projektorganisation und –Management

Spezialvorlesungen

Zur Vertiefung:
weitere Vorlesungen, sich gegenseitig ergänzend
zu Besonderheiten unterschiedlicher Phasen
im SW-Lebenszyklus, darunter etwa:

- Software-Verifikation und -Validierung
- Analyse und Entwurf mit UML
- Fehlertolerierende Softwarearchitekturen
- Management von Softwareprojekten

Seminare im Grundstudium

- Motivation:
  um Studierenden aus dem zweiten Studienjahr,
  die bereits einige Erfahrungen mit Softwareproblemen gesammelt haben, ansatzweise auf die Bedeutung des Software Engineering hinzuweisen und sie damit rechtzeitig für die Fächer im Hauptstudium zu motivieren.

- Proseminar Wintersemester 2002 / 2003:
  Software Engineering
  zur Vermeidung softwarebedingter Unfälle

Lernen aus Fehlerszenarien

“Learn from the mistakes of others,...
you'll never live long enough to make them all yourself.“

- Ariane 5 Explosion
- LH Airbus 320 Unglück bei Landung
- Therac 25 Verstrahlung mehrerer Patienten
- Scheitern der Mars Probe Mission
- Rechnerabsturz der Berliner Feuerwehr
- Scheitern des Patriot-Abwehrsystem bei Scud-Rakete
- Inkorrekte Diagnosen d. Fehlbedienung in Blutdatenbank
- Explosion einer chemischen Fabrik

Seminare im Hauptstudium

- In **Hauptseminaren** sollen Spezialgebiete des Software Engineering behandelt werden, zum Beispiel
- Software-Testen (Sommersemester 2002)
- Informationssicherheit (security)
- Standards zum Einsatz genehmigungspflichtiger Software (safety)
- aktuelle Fortschritte auf dem Gebiet formaler Methoden
- Bewertung der Software-Zuverlässigkeit

Software Engineering Praktikum

- Zur Ergänzung und Vertiefung:
  individuelle praktische Erprobung der vorgestellten Verfahren
  zur Unterstützung der Softwareerstellung und –analyse

- Aufgrund der drastisch anwachsenden logischen Komplexität:
  systematische Vorgehensweisen nicht manuell zu realisieren
  (mühsam und fehleranfällig)

- Lösung: industrieller Einsatz von **Werkzeugen**,
  d. h. unterstützende Programme,
  die entsprechende Schritte des Software Engineering
  zu automatisieren erlauben.

- zunehmend als Hilfsmittel für Entwicklung und Management
  industrieller Softwareprojekte herangezogen

Software Engineering Praktikum

- Zur Bewältigung komplexer Programm- und Systemanalysen Einsatz automatisierter Hilfsmittel unter möglichst realen industriellen Randbedingungen erproben. Beispiele für Aufgaben des Software Engineering, die durch industriell eingesetzte Werkzeuge unterstützt werden, sind:

- Analyse & Entwurf
- Verifikation & Validierung
- Ermittlung quantitativer Kenngrößen
- Prozessreife und Projektmanagement

Analyse

- Statische Programmanalysatoren
  Erfassung und Auswertung des Daten- und Kontrollflusses
  Aufzeichnung der durch Tests erzielten strukturellen Überdeckungsmaße
  Unterstützung bei der Generierung weiterer Testläufe

- Objektorientierte Analyse / Entwurf / Programmpflege
  Graphische Darstellung der Anforderungen
  Codegenerierung
  Rückübersetzung (reverse engineering).

- Analyse nebenläufiger Prozesse / (Echt-) Zeitverhaltens
  Modellierung von Kooperations- und Konfliktsituationen
  Analyse der Erreichbarkeit vorgegebener Markierungen
  Simulation.

- Fehlerbaumanalyse
  Ermittlung von Systemengpässen durch Herleitung
  minimaler Schnittmengen und Importanzgrößen

Software-Verifikation & -Validierung

- Theorem-Beweiser
- Nachweis der Korrektheit logischer Aussagen durch formale Herleitung der Beweisschritte, insbesondere Nachweis der Korrektheit sukzessiver Entwurfsverfeinerungen bzw. automatische Erzeugung manuell zu entlastender Zwischenschritte (sog. Beweisverpflichtungen).

- Beweis-Checker

- Model Checking
  Nachweis vorgegebener Sicherheits- bzw. Lebendigkeitseigenschaften bzw. Generierung konkreter Gegenbeispiele.

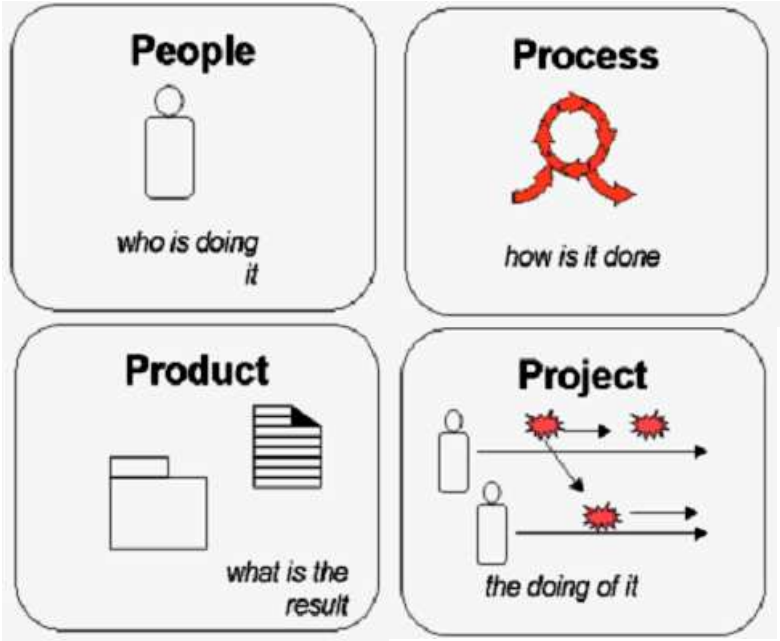
- **Bewertung der Softwarezuverlässigkeit**
  - Schätzung aktuell erzielter und künftig zu erwartender Zuverlässigkeitskenngrößen
  - durch Modellierung der mit sukzessiven Korrekturen bzw. Funktionalitätsanpassungen eingehenden Programmevolution

Projektmanagement

- Verwaltung der verschiedenen Versionen (Releases), Änderungswünsche der Kunden, Reklamationen, Dokumentation, Configuration Management, Termin- und Kostenverfolgung, Reifegrad, Kompatibilität.
-
- **Bewertung der Qualität des Softwareentwicklungsprozesses**
- Bewertung des Reifegrads des Herstellungsprozesses eines Unternehmens, Einstufung der sich der Prüfung unterziehenden Softwarefirmen in Klassen, die Qualität, Stil und Einheitlichkeit der Managementorganisation wiedergeben, z. B. nach CMM, Capability Maturity Model, Software Engineering Institute, Carnegie Mellon Univ. (Pittsburgh).
-

Heterogene Domänen

4 Ps



- **Personal**
  Wer entwickelt?
- **Prozess**
  Wie wird entwickelt?
- **Produkt**
  Was ist das Ergebnis?
- **Projekt**
  Gesamte Koordination

Alle 4 Aspekte kontrollieren, also messen!

"You cannot control what you cannot measure!" (**Tom De Marco**)