Software Engineering

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

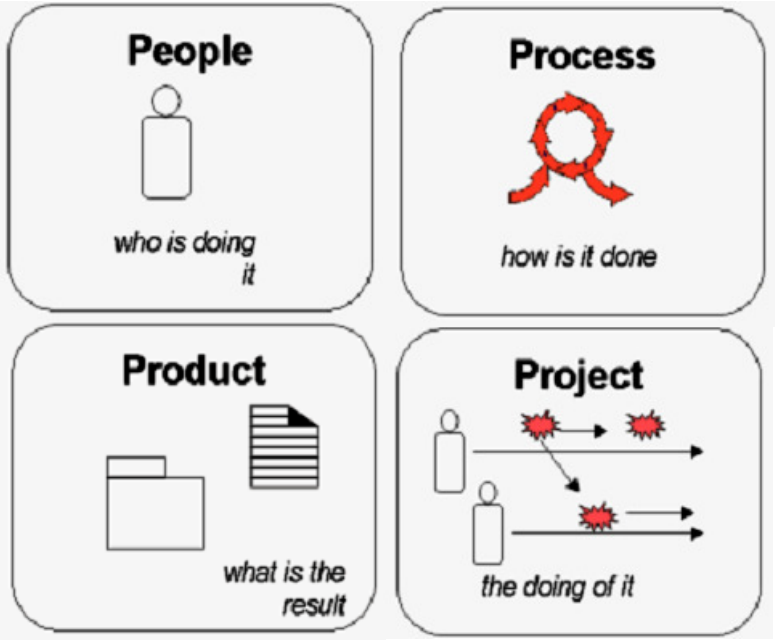
Francesca Saglietti

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

4 Aspekte des Software Engineering

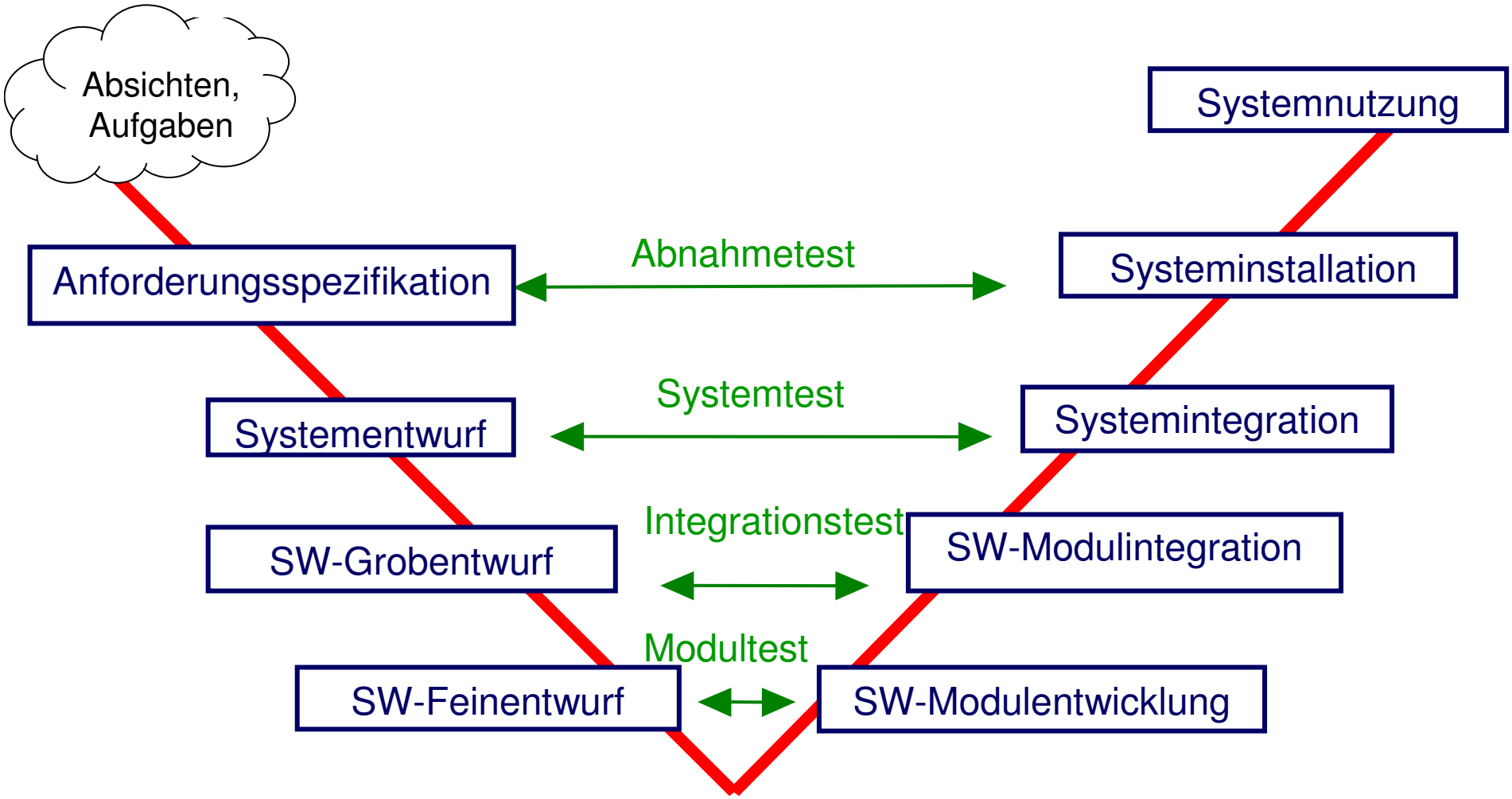


- **Personal**
  Wer entwickelt?
- **Prozess**
  Wie wird entwickelt?
- **Produkt**
  Was ist das Ergebnis?
- **Projekt**
  Gesamte Koordination

Alle 4 Aspekte kontrollieren, also **messen**!

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

V-Modell


Absichten,
Aufgaben
Anforderungsspezifikation
Systementwurf
SW-Grobentwurf
SW-Feinentwurf
SW-Modulentwicklung
SW-Modulintegration
Systemintegration
Systeminstallation
Systemnutzung
Abnahmetest
Systemtest
Integrationstest
Modultest

Vorkenntnisse / Studiengänge

- Zum Verständnis der ingenieurwissenschaftlichen Prinzipien zur Softwareerstellung sind Kenntnisse aus dem **Grundstudium** erforderlich:
  - vor allem Programmierkenntnisse,
  - aber auch Kenntnisse zu Programmiersprachen und –konzepten

- Deshalb wenden sich die Lehrveranstaltungen des Lehrstuhls hauptsächlich an Studenten des **Hauptstudiums**, die diese Grundkenntnisse bereits erworben haben
  - Diplom-Informatiker
  - Computational Engineering (Bachelor und Master)
  - Wirtschaftsinformatiker
  - Maschinenbauer
  - Mechatroniker
  - Computerlinguisten
  - demnächst Informations- und Kommunikationstechniker

Unterschied zum Grundstudium

- **Grundstudium**
  - hauptsächlich **Programmieren im Kleinen**
    Algorithmik 1,2 (objektorientierte / funktionale / logische Programmierung
    Softwaresysteme 1,2 (systemnahe Programmierung, Datenbanken)
  - nur ansatzweise **Programmieren im Großen**
    Softwaresysteme 3: Einzelne beispielhafte Ansätze

- **Hauptstudium: Software Engineering als Ingenieurwissenschaft**
  - pro Phase und Tätigkeit des Software-Lebenszyklus:
    viele **alternativen** Vorgehensweisen besprochen / verglichen
  - Instrument zur **Entscheidungsfindung** über optimales Vorgehen im Einzelfall (Preisleistungsverhältnis: Einfamilienhaus oder Schloß?)
  - Betrachtung **menschlicher** Faktoren (kognitiv-psychologisch)
  - **Messung** von Kenngrößen:
    Entwurfsqualität, Komplexität, Testüberdeckung, …

Lehrangebot

- **Hauptvorlesung**
  breit gefächertes Bild
  vorhandener
  ingenieurwissenschaftlicher Ansätze
  und ihrer Grenzen
  auch solchen Zuhörern,
  die eine einzige Lehrveranstaltung
  zur SW-Technik besuchen

- **Spezialvorlesungen und**
- **Seminare**
  zur Vertiefung
  sich gegenseitig ergänzend
  zu Besonderheiten
  unterschiedlicher Phasen
  im SW-Lebenszyklus

Lehrangebot

- **Hauptvorlesung** **mit zentralen Übungen**
  - Grundlagen des Software Engineering
- **Spezialvorlesungen** **mit zentralen Übungen**
  - Software Verification and Validation
  - Fehlertolerierende Softwarearchitekturen
  - Softwarezuverlässigkeit
- **Praktische Übungen**
  - Software Engineering in der Praxis
  - Rechnerübungen zu Grundlagen des Software Engineering
  - Rechnerübungen zu Analyse und Design mit UML

Grundlagen des Software Engineering

- 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,
  - **Nutzen**, **Grenzen** und **Komplementarität** aufgezeigt,
  - ihre **Eignung** in Abhängigkeit von den vorliegenden Anforderungen bewertet.

Inhaltsübersicht

Seminare

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.

- Software Engineering zur Vermeidung SW-bedingter Unfälle
- Denkfallen in der Software Entwicklung

Seminare im Hauptstudium

- Software-Entwurf und -Test mit der Unified Modelling Language (UML)
- Softwaretestverfahren
- Model-based Software Engineering
- Softwaremetriken
- Design Patterns in industriellen Anwendungen
- Softwarespezifikation – Sprachen, Methoden, Werkzeuge

Lernen aus Fehlerszenarien

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

- Verstrahlung von Patienten durch Therac 25 (1985-87)
- AT&T Telefon-Rechnerausfall in den USA (1990)
- Scheitern des Patriot-Raketenabwehrsystem (Scud-Rakete, 1991)
- Unfall eines Airbus A-320 bei der Landung in Warschau (1993)
- Versagen des elektronischen Stellwerks im Bahnhof Hamburg-Altona (1995)
- Explosion der Ariane 5 Rakete (1996)
- Antriebloses US-Kriegsschiff Yorktown/Titan (1997, 1999)

Lernen aus Fehlerszenarien

- LH Airbus 320 Unglück bei Landung in Ibiza (1998)
- Scheitern der Mars Probe Mission: Verlust der Mars Climate Orbiter Sonde (1999)
- Absturz des Dispositionssystems der Berliner Feuerwehr (2000)
- Geschädigte Taucher durch Aladin Nitrox-Tauchcomputer (1999, 2002)
- Notlandung der Soyuz Raumkapsel (2003)
- Scheitern der Mars Spirit Mission (2004)

Fehlertolerierende Softwarearchitekturen

- Bei hohen Zuverlässigkeits- / Verfügbarkeitsanforderungen insbesondere für sicherheitskritische Systeme lohnenswert,
  - während der Entwicklung **mehr Hilfsmittel** bereitzustellen und
  - während des Betriebes **mehr Hilfsmittel** einzusetzen,

  als es im Falle einer vollständig korrekten Implementierung erforderlich wäre
  - **Redundanz** zur Tolerierung sporadischer Softwarefehler im Betrieb

- Die Vorlesung befasst sich sowohl mit
  - **konstruktiven** Fragestellungen beim Entwurf fehlertolerierender Software
  - **deterministische** Aussagen zur Beherrschung **vorgegebener Fehlerklassen**

  als auch mit
  - **analytischen** Ansätzen zur Bewertung des zu erwartenden Fehlverhaltens
  - **probabilistische** Aussagen über **Zuverlässigkeitskenngrößen** (Überlebenswahrscheinlichkeit, Verfügbarkeit) vorgegebener Architekturen

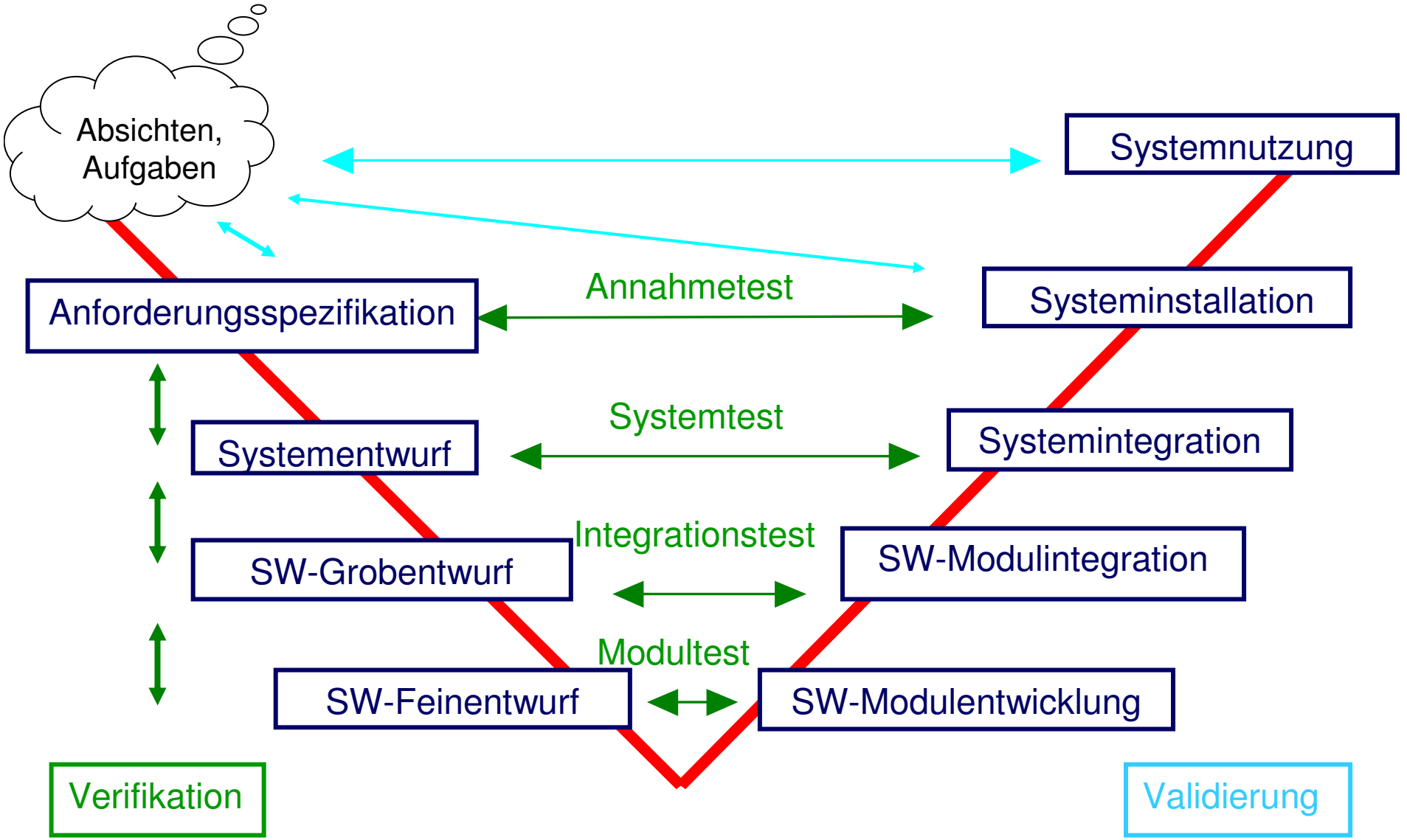
Software Verification and Validation

- Entsprechend dem Grad der Zuverlässigkeits- und Sicherheitsanforderungen Nachweisverfahren unterschiedlicher Rigorosität
  - Testverfahren
  - Beweisverfahren

  - **Verifikation** stellt sicher
    dass die Software
    bezüglich der definierten Aufgabenstellung korrekt entwickelt wurde
  - making things right

  - **Validierung** stellt sicher
    dass die Aufgabenstellung korrekt definiert wurde
  - making right things

Software Verification and Validation



Softwarezuverlässigkeit

- systematische Vorgehensweisen zur quantitativen Bestimmung der erzielten Zuverlässigkeit eines Softwaresystems

sowohl für

- **kommerzielle Softwarepakete**, für die das Argument “Time-to-market“ unter Umständen gegenüber einer qualitätserhöhenden Verlängerung der Testphase überwiegt
- momentan erzielte Zuverlässigkeit schätzen und die zu künftigen Zeitpunkten voraussichtlich erzielbare Zuverlässigkeit vorauszusagen, um damit dem Projektleiter wertvolle und anschauliche Indikatoren im Hinblick auf den optimalen Zeitpunkt zur Produktfreigabe zu bieten

als auch

- für **sicherheitsrelevante Softwaresysteme**, deren Einsetzbarkeit im Rahmen eines Genehmigungsverfahrens fundiert nachzuweisen ist
- also die nachzuweisende Zuverlässigkeitsanforderung mit Hilfe **stichprobenartiger Abläufe** statistisch fundiert zu belegen ist

Software Engineering in der Praxis

- Zur Unterstützung komplexer Softwareerstellung und –analyse
  - Einsatz automatisierter Hilfsmittel für **Entwicklung** und **Management**
- Aufgrund der anwachsenden logischen Komplexität
  - systematische Vorgehensweisen nicht manuell zu realisieren
  - mühsam und fehleranfällig
- Lösung
  - industrieller Einsatz von **Werkzeugen**, d. h. unterstützender Programme, die entsprechende Schritte des Software Engineering zu automatisieren erlauben
- Praktische Übungen
  - individuelle praktische Erprobung der vorgestellten Verfahren unter möglichst realen industriellen Randbedingungen

Software Engineering in der Praxis

- Unterstützung folgender Tätigkeiten des Software-Lebenszyklus
  - Modellierung und Simulation des Systemverhaltens, Analyse erreichbarer Zustände durch **klassische und zeit-behaftete Petri Netze**
  - Anforderungsanalyse durch vollautomatisches Nachweisen bzw. Widerlegen von Sicherheits- bzw. Lebendigkeitseigenschaften mittels **Model Checkers**
  - Verifikation durch maschinenunterstützte interaktive Korrektheitsbeweisführung mittels **Theorem Provers**
  - objektorientierte Analyse und Design durch graphische Editierung von **UML**-Modellen und automatische Generierung von Programmskeletten
  - Ermittlung quantitativer Indikatoren projektspezifischer **Komplexität** mittels automatischer Bestimmung prozess- und produktbasierter Software**metriken**
  - Evaluierung struktureller Testphasen durch automatische Ermittlung kontrollflussbasierter **Testüberdeckungsmaße** mittels Codeinstrumentierung
  - Configuration Management mittels maschineller **Verwaltung** zeitlich aufeinanderfolgender Versionen, Releases und kundenspezifischer Varianten
  - Dokumentation der Wartungs- und Pflegemaßnahmen durch Aufzeichnung, Kategorisierung und Weitergabe eingehender **Fehlermeldungen**, Änderungswünsche sowie getätigter Aktionen bei Fehlersuche und -behebung