Software Engineering

Einführung

Dr. A. Zimmermann

Software Engineering (Softwaretechnik) ist eine Wissenschaft, die sich mit der Entwicklung und dem Betrieb von großen Softwaresystemen beschäftigt. Folgende Aspekte gehören u.a. dazu:

- Projekt-Management
- Programm-technische Aspekte
- Datenhaltung
- Organisatorische Maßnahmen
- Qualitätssicherung
- Dokumentation

Laut [1] gilt folgende Definition der Software Engineering:

Softwaretechnik: Zielorientierte Bereitstellung und systematische Verwendung von Prinzipien, Methoden und Werkzeugen für die arbeitsteilige, ingenieurmäßige Entwicklung und Anwendung von umfangreichen Softwaresystemen. Zielorientiert bedeutet die Berücksichtigung z. B. von Kosten, Zeit, Qualität.

Der Begriff **Software** ist sehr vielseitig, umfassend und kann auf unterschiedliche Art definiert werden, abhängig von der Sicht der Entwickler, Anwender, Stakeholder, Aufsichtsbehörden u.s.w.

Beispiele:

- Software = Sammelbezeichnung für Programme...
- Software = Menge von Programmen, Daten und begleitenden Dokumenten...
- Software = Programme, Prozeduren, Regeln, Daten, Dokumentation...

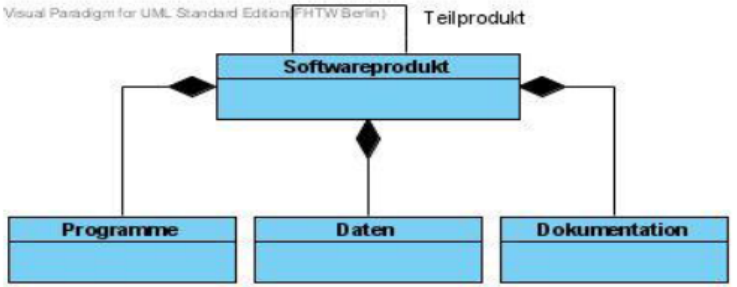
Zusammen mit dem Begriff **Software** verwendet man die Begriffe **Produkt** (Softwareprodukt) und **System** (Softwaresystem).

Der Begriff **Softwareprodukt** widerspiegelt die Sicht des Auftraggebers oder Käufers/Verkäufers. Es ist ein Ergebnis eines erfolgreichen Projektes, eines systematischen Vorgehens. Das ist eine **äußere** Sicht auf die Software.

Der Begriff **Softwaresystem** beschreibt die **interne** Sicht auf die Software. Die Betonung liegt auf den Komponenten der Software, auf den primären Funktionalitäten. In dem Fall betrachtet man die Software als Teil der reellen (oder gedanklichen) Welt. Man konzentriert sich auf der Kommunikation der Software mit dieser Welt, sowie auf den internen Zusammenhängen der Software.

Basisbegriff des Lehrgebiets: Software

- Software unterscheidet sich wesentlich von anderen technischen Gegenständen.



- Softwareprodukt: kundenorientierter Begriff
- Softwaresystem: entwicklungszentrierter Begriff

Der Begriff **großes** System ist auch wichtig. Er verlegt den Schwerpunkt auf die **Komplexität** des Systems. Kleine, nicht kommerzielle Systeme werden hier (in SE) nicht betrachtet.

Die Komplexität eines Systems beginnt mit dem einfachen Wachstum der einzelnen Komponenten (Befehlszeile, Datensatz, Transaktion, also quantitative Änderungen) und führt auf einem bestimmten Punkt zu einem Umschlag der Qualität. Somit können die großen Systeme nicht mit den Prinzipien und Methoden entwickelt werden, wie die kleinen Systeme. Hier fängt Software Engineering an.

Die Software-Komplexität lässt sich in folgende Bereiche gliedern:

- Komplexität der Daten (40 Datenbanken auf einem Server)
- Komplexität der Algorithmen (PKI, FEM, Simulationen)
- Komplexität der Funktionen (Tabellenkalkulation, CAD, DTP)
- Komplexität der Echtzeit-Systeme (CAM, Aufschließung von Öl- und Gasvorkommen)
- Komplexität der Systemumgebung (eingebettete Systeme wie Radaranlagen, Kraftwerksteuerung)
- Komplexität der GUI (CAD, Simulationen, Büroanwendungen, Web-Anwendungen)

Unterteilung der Software nach ihrer Größe (sloc = source lines of code):

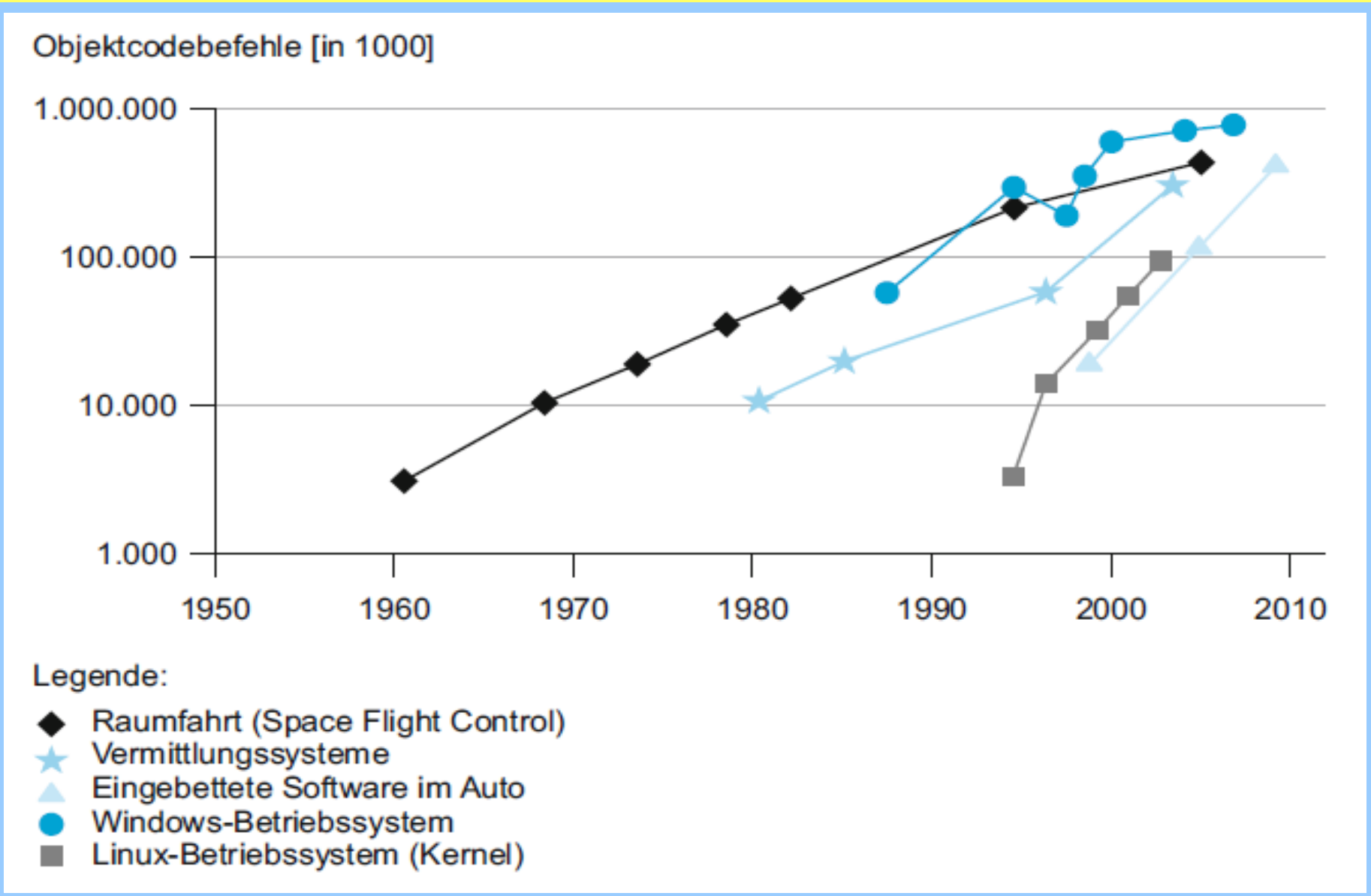
- Kleine Softwaresysteme – bis 2.000 sloc
- Mittlere Softwaresysteme – 2.000 sloc bis 100.000 sloc

iPhone Apps	10.000 sloc
UNIX v.1	10.000 sloc
Photoshop v.1	100.000 sloc
Space Shuttle	400.000 sloc

- Große Softwaresysteme – 100.000 sloc bis 1.000.000 sloc
- Sehr große Softwaresysteme – über 1.000.000 sloc

Age of Empires	1.000.000 sloc
Hubble Telescope	2.000.000 sloc

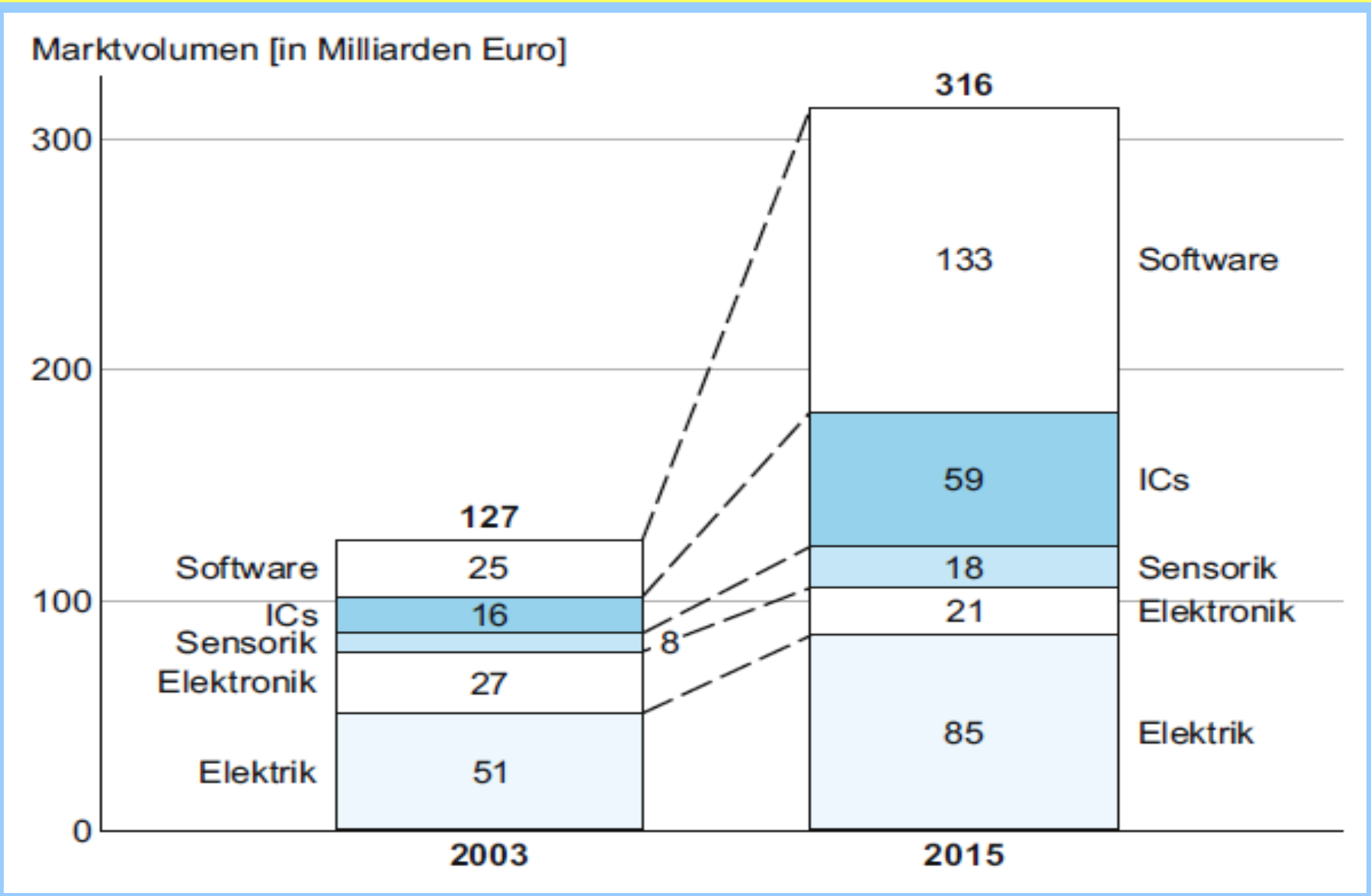
http://www.informationisbeautiful.net/visualizations/million-lines-of-code


Objektcodebefehle [in 1000]
1.000.000
100.000
10.000
1.000
1950
1960
1970
1980
1990
2000
2010
Legende:
Raumfahrt (Space Flight Control)
Vermittlungssysteme
Eingebettete Software im Auto
Windows-Betriebssystem
Linux-Betriebssystem (Kernel)

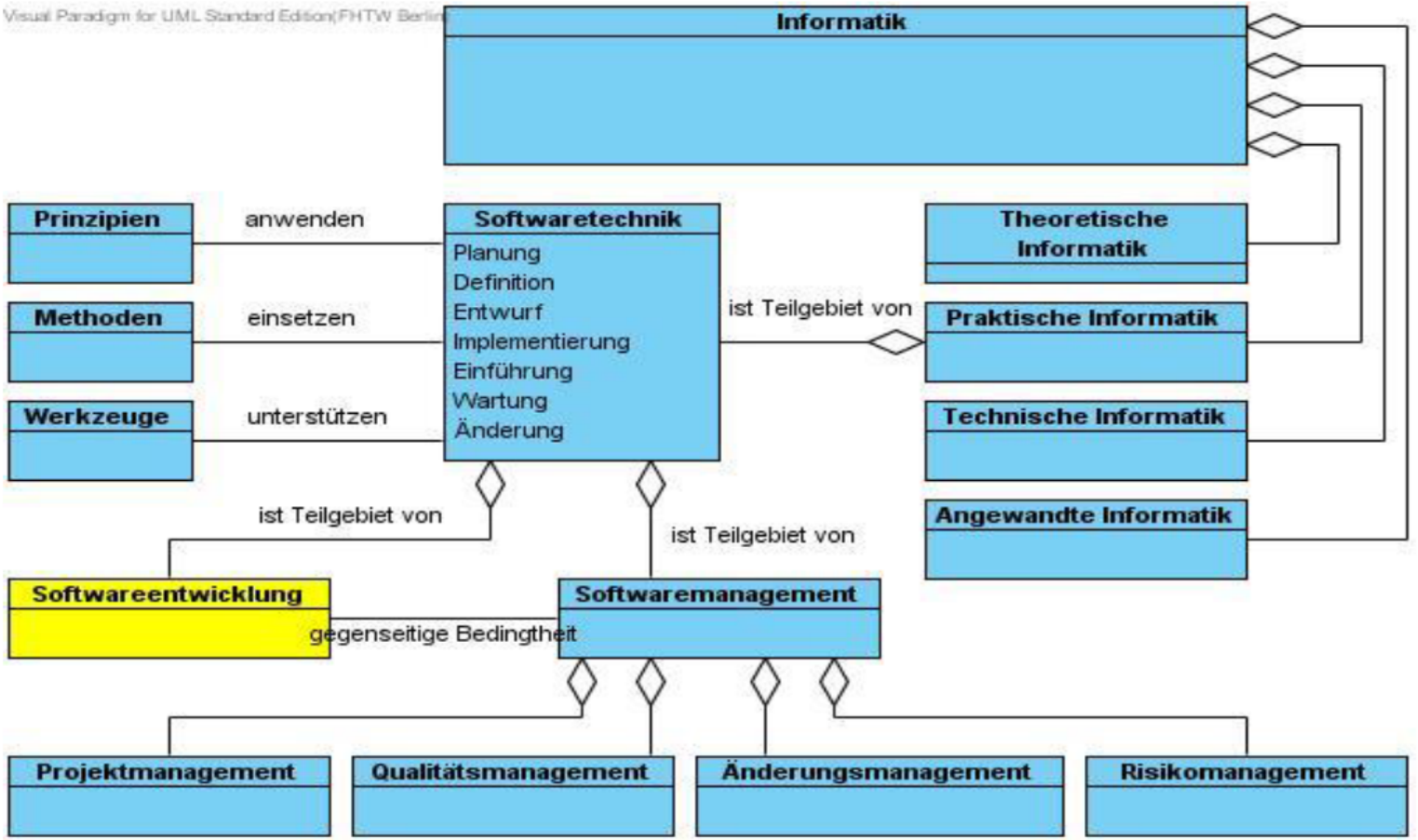
Die Software wird durch folgende Eigenschaften gekennzeichnet:

- Immaterielles Produkt (objektiver Entwicklungsfortschritt ist schwer festzustellen)
- Keine Abnutzung beim Einsatz (kein Verschleiß). Als Folge – es gibt keine Ersatzteile
- Alterung mit der Entwicklung der Hardware, mit den Änderungen der Anforderungen. Als Folge - ständige Anpassung
- Langfristige Benutzung (durchschnittlich 12 Jahre, bis 30 Jahre)
- Änderungen sind leichter und billiger als bei technischen Produkten
- Zunehmende Bedeutung von Software (Betriebssysteme sind in Handys, Kaffeemaschinen, Autos, Satellitenreceivern, Fernsehern)

- Zunehmende Qualitätsanforderungen an die Software
- Wachsende Komplexität
- Wachsende Nachfrage nach Arbeitskräften (etwa 20.000 Informatikabsolventen sind jährlich erforderlich, sind aber nur etwa 15.000 vorhanden)
- Anteil der Standardsoftware wächst (sinnvoll einzusetzen, wenn die Anforderungen standardisiert sind, billiger und weniger fehleranfällig als Individualsoftware)
- Zunehmendes Outsourcing (etwa 45% der Software wird im Ausland entwickelt)
- Wachsender Druck von Altlasten in dem Code der Systeme (Software von heute ist Altlasten von morgen)


Marktvolumen [in Milliarden Euro]
300
200
100
0
316
133
Software
59
ICs
18
Sensorik
21
Elektronik
85
Elektrik
127
Software
25
ICs
16
Sensorik
8
Elektronik
27
Elektrik
51
2003
2015

Einordnung des Lehrgebiets



Inhalt des Lehrgebiets (1/3)

- Einführung, Begriffe, Werkzeuge und Darstellungsmittel
- Anforderungsspezifikation
  - Begriffe und Methoden
  - Modelle
- Anforderungsanalyse (beinhaltet Erstellung von Analysemodellen)
  - Geschäftsprozesse und Anwendungsfälle
  - Informationsanalyse (und -modellierung)
  - Begriffe, Modelle, Ergebnisse, Maßnahmen
- Objektorientierte Analyse (führt zu Lösungsmodellen)
  - Analysemuster
  - Zuordnung von Operationen
  - Systemanwendungsfall-Operationen

4

Inhalt des Lehrgebiets (2/3)

- Entwurf der Systemarchitektur
  - Fachkonzeptschicht
  - Dialogschicht
  - Datenhaltungsschicht
- Aspekte der Softwarequalität
  - Qualitätsmerkmale
  - Qualitätssicherung

5

Inhalt des Lehrgebiets (3/3)

- Zusammengefasst werden im Verlauf der Lehrveranstaltungen folgende Fragen beantwortet:
  - Welche Prinzipien, Methoden und Techniken benötigt man für die Entwicklung komplexer Softwaresysteme?
  - Wie findet man heraus, welche Eigenschaften eine Software haben soll?
  - Wie beschreibt man diese Eigenschaften?
  - Wie strukturiert man die Software so, dass sie sich arbeitsteilig bauen und flexibel verändern lässt?
  - Was macht gute Software aus?
  - Wie deckt man Mängel in Software auf?
  - Wie beugt man Mängeln vor?

- Hinweis

  Das Ablegen der Leistungsnachweise mit befriedigenden oder besseren Ergebnissen setzt die regelmäßige Teilnahme an den Lehrveranstaltungen sowie regelmäßiges, intensives Selbststudium voraus.

Lehrzieltaxonomie

- Lehrziele der kognitiven Stufe „Kennen“

  Der Lernende ruft im Gedächtnis gespeicherte Informationen (z.B. Begriffe, isolierte Fakten, Abfolgen, Prinzipien, Mittel und Wege) ab. Typische beobachtbare Leistungen sind **bezeichnen, wiedererkennen, aufzählen**.

- Lehrziele der kognitiven Stufe „Verstehen“

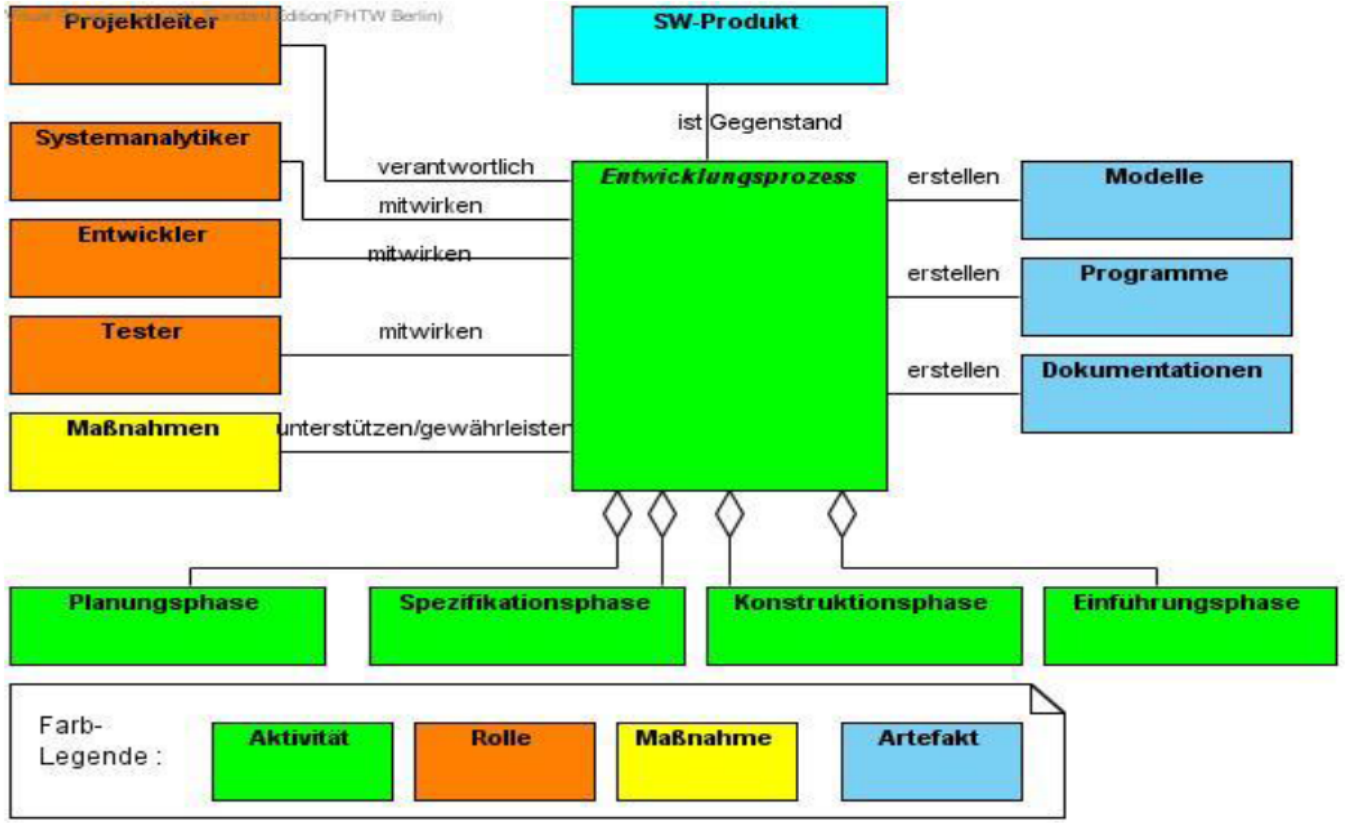
  Der Lernende verknüpft oder transformiert Informationen. Typische beobachtbare Leistungen sind **beschreiben, zusammenfassen, vergleichen, klassifizieren, begründen, erklären**.

- Lehrziele der kognitiven Stufe „Anwenden“

  Der Lernende überträgt erworbenes Wissen auf gegebene neue Situationen oder wendet sie zur Problemlösung an. Typische beobachtbare Leistungen sind **ausführen, anwenden, beurteilen, ermitteln, entwerfen, analysieren**.

Beschreibung einer Entwicklungs-Aktivität

Grafisches Modell:



Die Entwicklung der komplexen Softwaresysteme impliziert systematische Verwendung von bestimmten Prinzipien und Methoden.

- Ein Prinzip ist ein Grundsatz, den man seinem Handeln zugrunde legt.
- Eine Methode ist eine begründete und planmäßig angewandte Vorgehensweise zur Erreichung des Ziels im Rahmen der festgelegten Prinzipien.

Eigenschaften vom einem Prinzip:

- Es ist abstrakt und allgemeingültig.
- Es bildet eine theoretische Grundlage.
- Es wird aus der Praxis hergeleitet und in Praxis bestätigt.
- Es ist von dem Anwendungsgebiet oft unabhängig.

In den grundlegenden Arbeiten von weltbekannten Gelehrten wie N. Wirth, E. W. Dijkstra, D. Parnas, F. P. Brooks u.a. in den 1970er und 1980er Jahren wurden die Prinzipien für den Aufbau eines IT-Systems festgelegt. Diese Prinzipien spielen heute noch eine wichtige Rolle in dem Softwareentwicklungsprozess, insbesondere für Spezifikation, Entwurf und Implementierung. Die Prinzipien sind nicht isoliert, sie befinden sich in Wechselwirkung, einige Prinzipien sind Voraussetzungen für die anderen.

Die Prinzipien sind laut [1]:

- Abstraktion (Verallgemeinerung)
- Strukturierung
- Hierarchisierung
- Trennung von Zuständigkeiten (Bindung und Kopplung)
- Modularisierung
- Verbergen von Informationen (Geheimnisprinzip)
- Lokalität
- Verbalisierung

Abstraktion ist Verzicht auf besondere Eigenschaften und Einzelheiten des Objektes oder des Systems. Dabei werden die wesentlichen und grundlegenden Merkmale herausgefunden, und alles andere wird vernachlässigt.

Abstrakt bedeutet theoretisch, verallgemeinert, nicht konkret, nicht anschaulich.

Durch Abstrahieren wird ein Modell der realen Welt gebildet. Es ist eine sehr anspruchsvolle Tätigkeit für die Systemanalytiker. Ein Modell muss vollständig und widerspruchslos sein.

Gegenteil zur Abstraktion ist die Konkretisierung.

Vorteile:

- Erkennung, Ordnung, Klassifizierung und Gewichtung der wesentlichen Merkmale des Systems
- Trennen des Wesentlichen vom Unwesentlichen
- Bildung der Voraussetzungen für Allgemeingültigkeit

Strukturierung ist die Anordnung der Teile des Ganzen zueinander. Diese Teile (Komponenten) können wechsel- oder einseitig auf einander wirken.

Eine Struktur stellt die wesentlichen Merkmale des Systems dar.

Man unterscheidet folgende Arten von Strukturen:

- Statische Struktur – ab einem bestimmten Zeitpunkt ändert sich die Struktur nicht mehr. Beispiel: die Struktur der Daten einer Anwendung.
- Dynamische Struktur – die Struktur ändert sich im Laufe der Zeit. Beispiel: Speicherverwaltung der rekursiven Programme.

Die Strukturen werden oft als Graphen dargestellt.

Vorteile:

- Ein Mittel, Komplexität zu beherrschen
- Hohe Verständlichkeit
- Bessere Wartbarkeit
- Bessere Einarbeitung in das Softwareprodukt

Hierarchie ist eine Struktur, in der für jede Komponente Vorgänger- und Nachfolger-Komponente definiert sind. Jede Komponente muss exakt einen Vorgänger und kann mehrere (auch keine) Nachfolger haben. Eine Ausnahme bildet die Hauptkomponente (root), die keinen Vorgänger hat.

Viele Objekte in der Natur scheinen diesem Prinzip zu gehorchen (mindestens) aus der Sicht des menschlichen Verständnisses.

Vorteile:

- Viele Systeme können nach diesem Prinzip abgebildet werden.
- Die chaotischen Strukturen sind unmöglich.

Bindung ist ein Maß, in dem die Bestandteile (z.B. Klassen) einer Systemkomponente unter einander abhängig sind. Starke Bindung einer Komponente bedeutet, dass alle ihre Bestandteile die Informationen von einander brauchen und für eine Aufgabe ausgerichtet sind. Schwache Bindung bedeutet, dass die Bestandteile von einander (fast) unabhängig sind und unterschiedliche Aufgaben erfüllen.

Kopplung charakterisiert die Abhängigkeit zwischen den Komponenten eines Systems. Bei einem hohen Grad der Kopplung sind die Komponenten eng mit einander verbunden, d.h. es wird relativ schwer, die Änderungen an dem System durchzuführen. Umgekehrt sind die Änderungen in einem schwach gekoppelten System i.d.R. problemlos.

Folgende Regeln führen zu einer ausgeprägten Struktur des Systems:

- Bindungsgrad muss man maximieren.
- Kopplungsgrad muss man minimieren.

Vorteile:

- Hohe Qualität des Softwareproduktes
- Hohe Wartbarkeit
- Hohe Flexibilität

Modul ist eine weitgehend abgeschlossene Bau- oder Funktionsgruppe. Dementsprechend verlangt das Prinzip der **Modularisierung**, dass ein System aus den Modulen bestehen muss. Ein Modul hat eine fest definierte (möglichst standardisierte) Schnittstelle zu anderen Modulen, um Informationsaustausch zu gewährleisten.

Vorteile:

- Hohe Wartbarkeit
- Hohe Änderbarkeit
- Erleichterung der Arbeitsorganisation

Geheimnisprinzip verbietet den Zugriff auf die Daten, die für Erledigung einer bestimmten Aufgabe nicht nötig sind.

Dieses Prinzip wird stark in der OOP eingesetzt (Datenkapselung). Die Schnittstellen zwischen den Modulen müssen sehr sorgfältig dokumentiert sein.

Vorteile:

- Minimaler Fehleranteil
- Hohe Zuverlässigkeit des Systems
- Sicherheit der Daten

Lokalität bedeutet, dass alle Informationen, die zur Lösung eines Problems notwendig sind, sind an einer Stelle zu finden. Diese Probleme können bsp. Fehlersuche, Einarbeitung, Implementierung sein. Unter der Stelle wird der Arbeitsplatz oder die Web-Seite gemeint.

Dieses Prinzip bedeutet auch, dass die überflüssigen Informationen an dieser einen Stelle nicht vorhanden sind.

Lokalität widerspricht manchmal (sogar oft) dem Geheimnisprinzip.

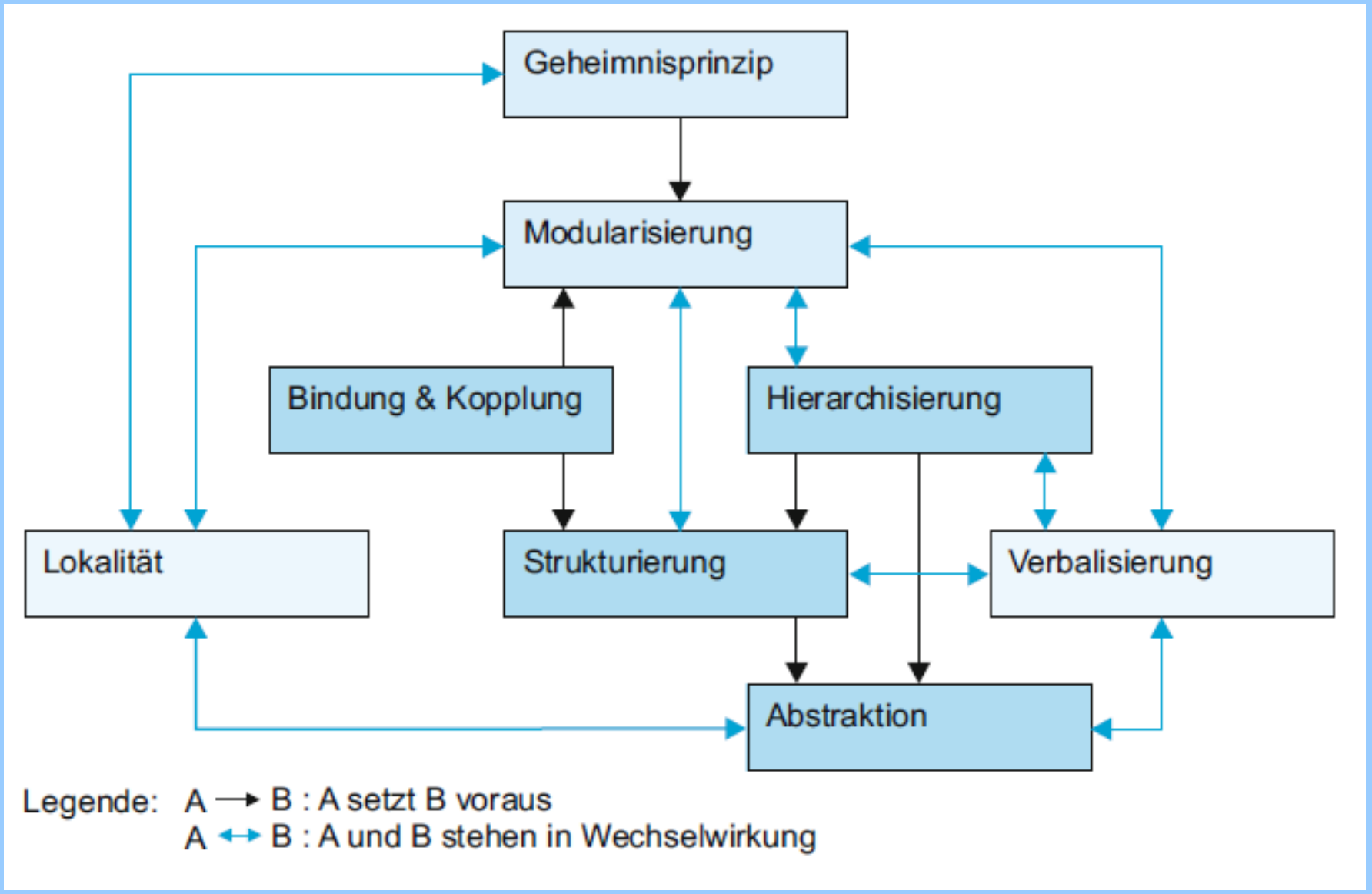
Vorteile:

- Hohe Wartbarkeit
- Hohe Änderbarkeit

Verbalisierung wird durch aussagekräftige Namensgebung, sinnvolle Kommentare und gut dokumentierte Vorgänge ausgeprägt.

Vorteile:

- Leichte Einarbeitung und Wiedereinarbeitung ins System
- Hohe Wartbarkeit
- Leichte Qualitätssicherung


Geheimnisprinzip
Modularisierung
Bindung & Kopplung
Hierarchisierung
Lokalität
Strukturierung
Verbalisierung
Abstraktion
Legende: A → B : A setzt B voraus
A ↔ B : A und B stehen in Wechselwirkung

Eigenschaften von Methoden:

- Sie basiert auf bestimmten Prinzipien.
- Sie ist von dem Anwender unabhängig.
- Sie ist auf Kenntnissen und Erfahrungen des Entwicklers gebaut (keine Intuition).
- Man kann sie erlernen.
- Sie wird in Software Engineering als Oberbegriff verwendet.

Man unterscheidet folgende fach-unabhängige Methoden:

- Laut **Top-Down-Methode** wird vom Abstrakten zum Konkreten (vom Allgemeinen zum Speziellen) vorgegangen.
- Laut **Bottom-Up-Methode** wird vom Konkreten zum Abstrakten (vom Speziellen zum Allgemeinen) vorgegangen.
- Laut **Outside-In-Methode** wird zuerst die Umwelt des Systems modelliert, und davon ausgehend die Systemkomponenten.
- Laut **Inside-Out-Methode** werden zuerst die Systemkomponenten modelliert, und davon ausgehend die Schnittstelle zur Umwelt des Systems.

In Software Engineering werden meistens die Top-Down- und Outside-In-Methoden verwendet.

Die Methoden sind von den Verfahren zu unterscheiden.

Die Verfahren sind ausführbare Vorschriften oder Anweisungen zum gezielten Einsatz von Methoden.

- Sie beschreiben konkrete Wege zur Lösung bestimmter Probleme oder Problemklassen.
- Sie beinhalten formale Vorschriften und bilden die Standards.

<table>
<tr><td rowspan="6">Ziel:
Ein Haus bauen</td><td rowspan="3">Methode:
Kredit nehmen</td><td>Bank wählen</td></tr>
<tr><td>Eigene Kreditwürdigkeit nachweisen</td></tr>
<tr><td>Vertrag schließen</td></tr>
<tr><td rowspan="3">Methode:
Eigenes Geld sparen</td><td>Einnahme/Ausgabe synchronisieren</td></tr>
<tr><td>Zusätzliches Einkommen sichern</td></tr>
<tr><td>Sparvertrag schließen</td></tr>
</table>