Vorlesung Software Engineering

Foliensatz Nr. 1 (19.10.11)

Arbeitsgruppe Software Engineering
Prof. Elke Pulvermüller

Universität Osnabrück
Institut für Informatik, Fachbereich Mathematik / Informatik
Raum 31/318, Albrechtstr. 28, D-49069 Osnabrück

elke.pulvermueller@informatik.uni-osnabrueck.de

http://www.inf.uos.de/se

Sprechstunde: mittwochs 14 – 15 und n.V.

Vorlesung Software Engineering

Organisatorisches

- **Vorlesung:**
  Umfang: 4+2 SWS (= 9 ECTS), Master- und Bachelor-Modul
  Zeit: mittwochs 12-14 und donnertags 12-14
  Ort: 31/449a (mittwochs) und 32/109 (donnerstags)

- **Prüfung (voraussichtlich!):**
  Termin: Mi 08.02.12 oder Do 09.02.12 / 13:00 – 15:00 Uhr
  Anmeldung über OPIUM

- **Informationen über Mailingliste + StudIP**
  Bitte in die Mailingliste eintragen sofern noch nicht geschehen!

Organisatorisches

- **Informationen:**

- Webseite: http://www-lehre.inf.uos.de/~swe/2011
- Übungen: http://www-lehre.inf.uos.de/~swe/2011/uebung.html
- Stud.IP: http://studip.serv.uos.de
- Prüfungsanmeldung in OPIUM: http://www.opium.uni-osnabrueck.de

- Folien der Vorlesung auf der Webseite (Passwort, s. “News” in Stud.IP)
- Literatur auf der Webseite (auch online verfügbare Lehrbücher)

- Mailingliste:
- Eintragen: https://list.serv.uni-osnabrueck.de/mailman/listinfo/swe11

  Hinweis: Teilnehmer, die bis Montag, 17. Oktober, in Stud.IP angemeldet waren, sind automatisch eingetragen worden und haben bereits eine Begrüßungsmail erhalten.
- „Spam“-Filter: Nur registrierte Absender dürfen auf der Liste posten.
- E-Mail-Adresse der Liste: **swe11@list.serv.uni-osnabrueck.de**
- Ein nicht-öffentliches „Archiv“ der Mailingliste ist auf der Übungsseite verlinkt.

Organisatorisches

- **Übung:**
  - Gruppe 1: dienstags 10:00 Uhr – 12:00 Uhr (69/E15)
  - Gruppe 2: dienstags 12:00 Uhr – 14:00 Uhr (69/E15)
  - **Übungsleitung: Wolfgang Runte**
  - **Tutoren: Sergey Krutikov, Daniel Neumann**

- **Übungsblätter:**
  - pro Blatt müssen mind. 50% der Punkte erreicht werden
  - n-1-Regelung, d.h. ein “Freischuss”
  - Bearbeitung der Übungsblätter in Zweiergruppen
  - Blattausgabe: dienstags in der Übung (und online auf der Übungsseite)

Organisatorisches

- **Ablauf:**
  - Für die Bearbeitung können die Rechner in 31/369 genutzt werden.
    Loginprobe?
    Ansprechpartner bei Problemen: Friedhelm Hofmeyer (31/319)

  - Alle Aufgaben sind schriftlich zu bearbeiten und vor dem Testat per E-Mail beim jeweiligen Tutor abzugeben.

  - Quellcode ist ausführlich per Javadoc zu dokumentieren!
    Zu dokumentieren sind Klassen, Autoren, Version, Attribute, Methoden, Parameter, Rückgabewerte, Exceptions.
    Wer's nicht macht bekommt Punktabzug!

  - Ausgabe des ersten Übungsblattes: Di., 25. Oktober, in der Übung.

Organisatorisches

- **Testate:**
  - Montag bis Mittwoch im Rechnerraum 31/369.
  - Eintragen für Testattermine (2er-Gruppen, Aushang vor 31/369 ab Mittwoch, 19. Oktober).
  - Möglichst gleichmäßig pro Tutor (sonst wird getauscht).
  - Anwesenheit beider Gruppenmitglieder im Testat ist Pflicht.
  - Im Zweifelsfall werden Einzelbewertungen vorgenommen.
  - Testate finden in 31/369 statt, d.h. Programme müssen in jedem Fall auf diesen Rechnern laufen.
  - Bearbeitete Aufgaben sind vor Beginn des Testats beim jeweiligen Tutor in elektronischer Form abzugeben. In einem für die Tutoren lesbaren (Standard-)Format (ASCII, PDF, OpenOffice, ... etc., welches sich im CIP-Pool öffnen und verarbeiten/anzeigen lässt).
    Im Zweifel sowohl als PDF-Datei als auch im Ursprungsformat.
    Möglichst ge-zipped per E-Mail.

  - Erste Testate: Montag, 31. Oktober, bis Mittwoch, 2. November.

Literatur

- Ludewig, Jochen und Lichter, Horst: Software Engineering – Grundlagen, Menschen, Prozess, Techniken, dpunkt.verlag, 2006
- Sommerville, Ian: Software Engineering, Addison-Wesley
- Balzert, Helmut: Lehrbuch der Software-Technik Band 1, Spektrum Akademischer Verlag
- Balzert, Helmut: Lehrbuch der Software-Technik, Band 2, Spektrum Akademischer Verlag
- Zuser, Wolfgang; Grechenig, Thomas; Köhle, Monika: Software Engineering mit UML und dem Unified Process, 2004

- Weitere Quellen, die in Teilen in die Vorlesung eingegangen sind:
  - Vorlesungsunterlagen Universität Lübeck: Softwaretechnik (Prof. Dosch), Software Engineering (Dr. Magnussen), Softwarekonstruktion (Dr. Dölle)
  - Klaeren, Softwaretechnik Skript, Universität Tübingen, 2009

Inhalt

Inhalt

Spezielle, teils weiterführende Literatur zum Kapitel:

- Bauer, Friedrich L.: Software Engineering – wie es begann. Informatik-Spektrum, 16:259-260, 1993
- Floyd, Christiane: Software-Engineering – und dann? Informatik-Spektrum, 17(1):29-37,1993
- Brooks, Frederick P.: The Mythical Man-Month: Essays on Software Engineering, Addison-Wesley, 1975
- Boehm, B: Software Engineering. In: IEEE Transaction on Computers 25(12): 1226-1241, 1976

1.1 Krisenstimmung

Fehlschläge bei der Software-Entwicklung

Handelsblatt (Nr. 153, 12. August 1998, S. 37):

Software ist das fehlerhafteste Produkt überhaupt. Es ist schon erstaunlich, wie es manche Anwender mit der Geduld einer tibetanischen Bergziege hinnehmen, was ihnen da Programmierer zumuten. [. . .] Programme machen, was sie wollen, nur nicht das, was der Anwender gerne hätte. Das System stürzt ab, Daten gehen verloren. Selbst die Gerichte haben kein Einsehen mit dem geplagten Anwender; sie vertreten schlicht den Standpunkt, dass Software nun einmal fehlerhaft sei. Gewährleistung ist für die Softwarehersteller kein Thema.

1.1 Krisenstimmung

Fehlschläge bei der Software-Entwicklung

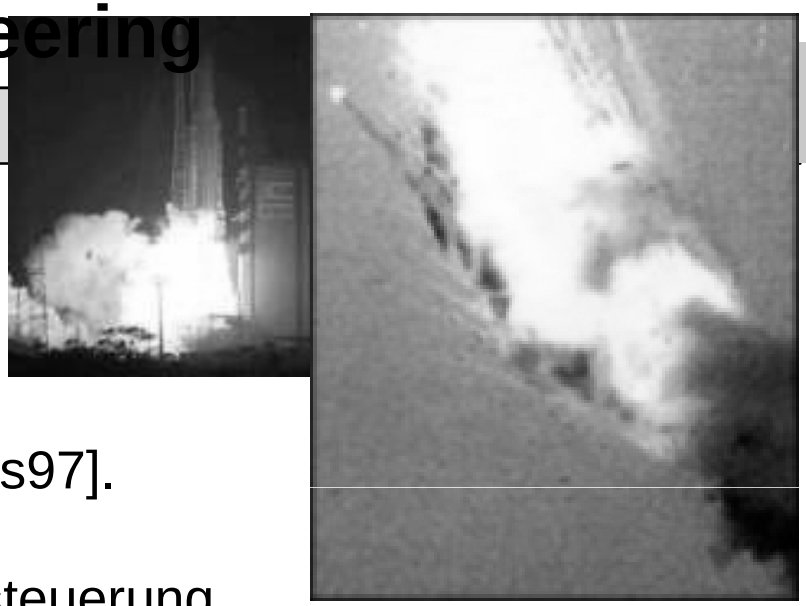
- Bei der Wahl des Oberbürgermeisters in Neu-Ulm 1994 wurde zunächst eine Wahlbeteiligung von 104% ermittelt. Später musste man feststellen, dass sich in die Auswertungssoftware ein mysteriöser Faktor 2 eingeschlichen hatte.
  [Partsch: Requirements Engineering Systematisch, S. 1]

- One of the first things the Air Force test pilots tried on early F-16 was to tell the computer to raise the landing while still standing on the runway. Guess what happened? Scratch one F-16.
  [ACM SIGSOFT Software Engineering Notes, vol. 11, no. 5 (1986), S. 10]

- ... in September 1997, the (USS) Yorktown suffered a systems failure during manoeuvres off the coast of Cape Charles, VA., apparently as a result of the failure to prevent a divide by zero in a Windows NT application. Atlantic Fleet officials said the ship was dead in water for about two hours and 45 minutes.
  [ACM SIGSOFT Software Engineering Notes, vol 24, no. 1 (1999), S. 31]

Software-Krise und Software Engineering

1.1 Krisenstimmung



- Durch eine falsche Software gerät die für 11 Milliarden Mark entwickelte Ariane 5 außer Kontrolle und muss gesprengt werden [JM97], [Nus97].

- Im neuen Flughafen Denver versagt die Softwaresteuerung der Gepäcktransportbänder derart, dass Koffer zerrissen werden [Swa96]. Der Flughafen wird mit 16-monatiger Verspätung eröffnet bei einem Verdienstausfall von rund einer Million Dollar pro Tag.

- In Warschau rollt 1993 ein Airbus 320 über die Landebahn hinaus, weil die Bordcomputer die Auslösung der Schubumkehr auf nasser Landebahn verweigern. Der Pilot kommt bei dem Aufprall des Flugzeugs ums Leben [Neu95, S. 46].

[JM97] J.-M. Jézéquel und B. Meyer. Design by Contract: The Lessons of Ariane. *IEEE Computer*, 30(1):129 - 130, 1997.

[Neu95] P. G. Neumann. Computer-Related Risks. Addison-Wesley, 1995.

[Nus97] B. Nuseibeh. Ariane 5: Who Dunnit? *IEEE Software*, 14(3):15 - 16, Mai 1997.

[Swa96] A. J.Swartz. Airport 95: Automated Baggage System? *Software Engineering Notes*, 21(2):79 – 83, 1996.

1.1 Krisenstimmung

Erfahrung, dass viele Programme

- **Fehlerhaft**
- **Unzuverlässig**
- **Wartungsunfreundlich**
- **Schwer portierbar**

sind.

⇒ Geburtsstunde 1967/1968 (NATO Science Committee):

“Software Crisis”, “Software Engineering”

1.2 Historische Entwicklung (Ursachen)

Historischer Hintergrund: Hardware

Technische Beschränkung (Speicher, Geschwindigkeit)

Leistungsfähige, zuverlässigere Rechner(hardware)

- 1940er Jahre, erste elektronische Rechner auf Basis von Relais (Rechner Colossus von Turing, Zuse Z1, MARK I von Aiken)
- 1950er Jahre, Rechner auf Basis von Röhren (erste Generation), z.B. ENIAC (Electronic Numerical Integrator and Computer) mit 17468 Röhren ca. 45 % der Zeit fehlerfreie Arbeiten
- ab 1957 Rechner auf Basis von Transistoren (zweite Generation), z.B. Siemens 2002 (Addition 90 µs, Multiplikation 120 µs) oder IBM 1400 Serie
- ab 1964 Rechner mit integrierten Schaltungen

1.2 Historische Entwicklung (Ursachen)

Historischer Hintergrund: Software

Die Charakteristika aus der Frühzeit der Computer (etwa 1942 – 1948):

- Fokus auf Hardware („automatische Rechenmaschinen“)
  Die Rechner wurden meist ohne Software ausgeliefert. Eine Programmierung erfolgt beim Kunden oder durch die Kunden.
- Einsatz für sehr eng umgrenzte Anwendungen (v.a. technisch-naturwissenschaftlicher Bereich, numerische Algorithmen)
- Programme waren reine Nachbildungen von Algorithmen, die aus der Mathematik bekannt waren (Entwicklung erster Programmiersprachen: z.B. FORTRAN ab 1954).
- Konstrukteure, Programmierer und Benutzer bildeten eine im wesentlichen geschlossene Gruppe mit durchweg gleicher Vorbildung.
- Die Programme behandelten eine eingeschränkte Problemklasse und griffen nicht unmittelbar in irgendwelche Handlungsabläufe ein.

1.2 Historische Entwicklung (Ursachen)

Historischer Hintergrund: Software

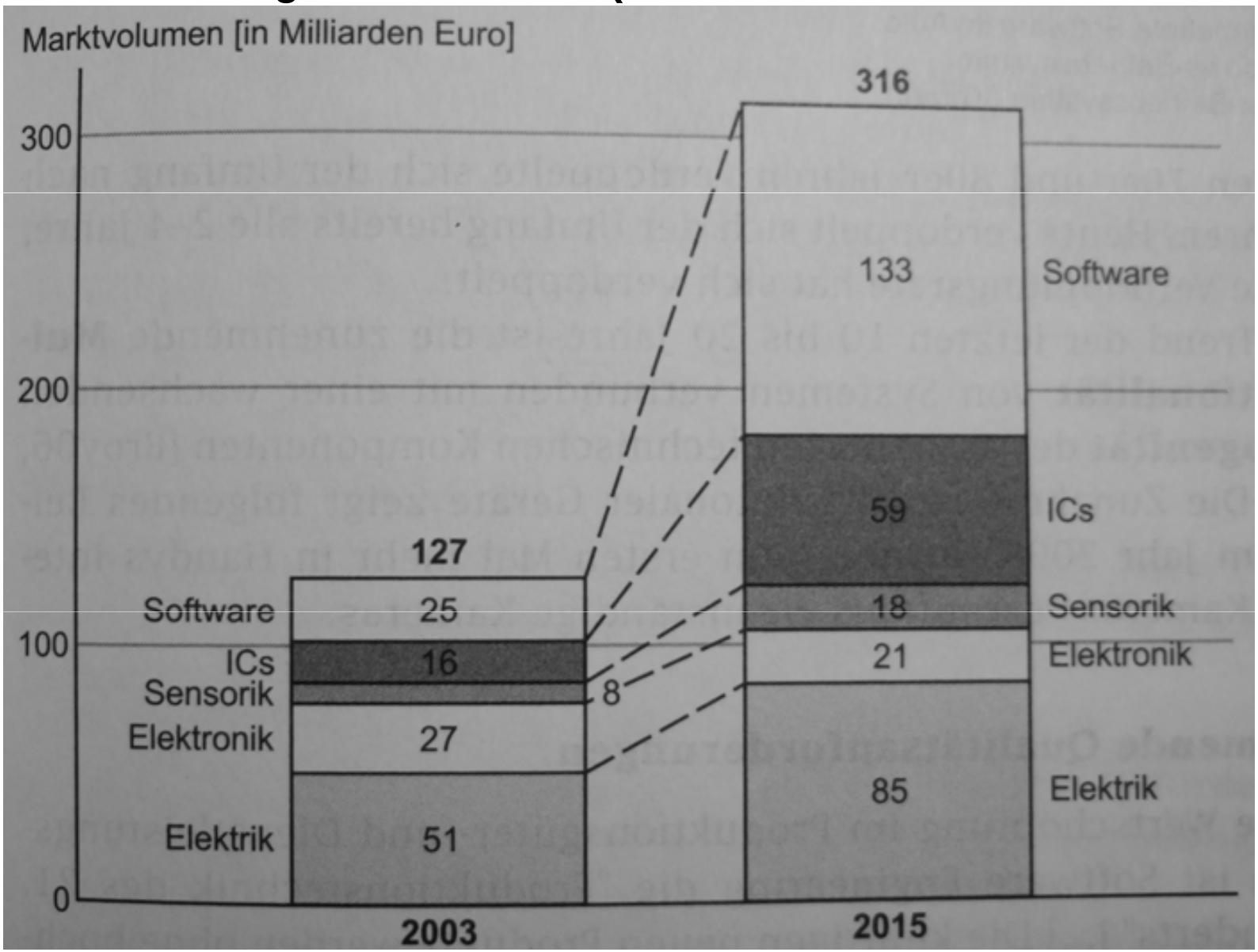
Die Charakteristika der Software Entwicklung ab den 1960er Jahren:

- Hardware und Software sind Teil einer ganzheitlichen Problemlösung; Aufgabenteilung und (komplexere) Algorithmen zur Lösung müssen erst gefunden werden
- Unüberschaubarer Einsatz-/Anwendungsbereich: die Programme greifen direkt in Abläufe ein, steuern diese und führen diese zum Teil auch selbst aus.
- Hersteller, Konstrukteure, Programmierer und Benutzer sind verschiedene Menschengruppen mit ungleicher Vorbildung (Kommunikationsprobleme!)
- Zunehmender Umfang von Softwaresystemen

⇨ **für die Software-Entwicklung fehlen ab den 1960ern geeignete Vorgehen zur Entwicklung großer Systeme mit den entsprechenden Folgen**

1.2 Historische Entwicklung (Ursachen)

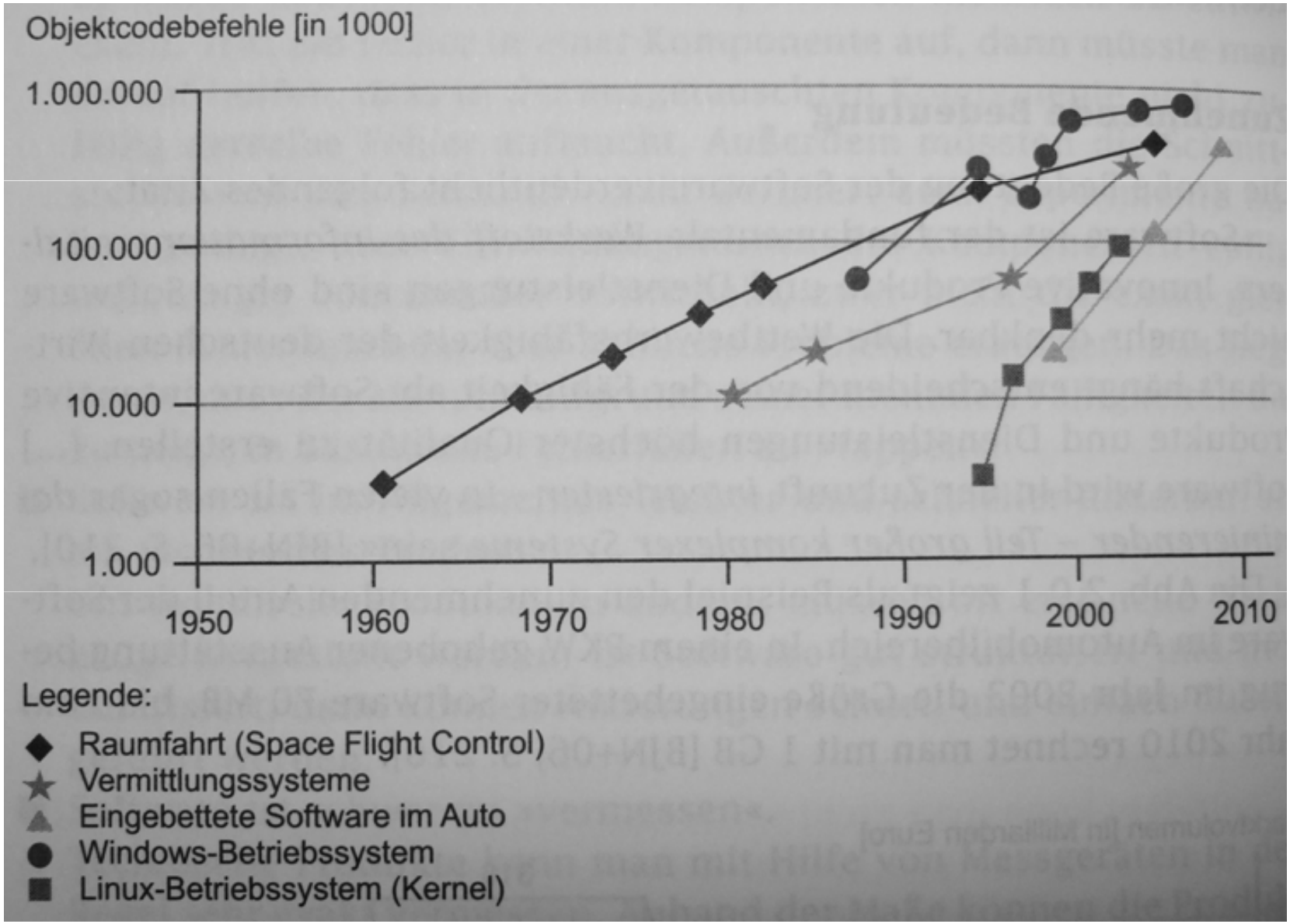
Historischer Hintergrund: Software (SW Marktvolumen Automobilbereich)



aus [Bal09] H. Balzert. Lehrbuch der Softwaretechnik, Spektrum Akademischer Verlag, 2009

1.2 Historische Entwicklung (Ursachen)

Historischer Hintergrund: Software (Entwicklung des SW Umfangs)



aus [Bal09] H. Balzert. Lehrbuch der Softwaretechnik, Spektrum Akademischer Verlag, 2009

Software-Krise und Software Engineering

1.2 Historische Entwicklung (Ursachen)

Prog. im Kleinen	Prog. im Großen	Auswirkungen
Genaue Spezifikationen liegen nicht vor oder werden nicht benötigt	Spezifikationen müssen erst erarbeitet werden	Spezifikationsmethoden, Formalismen, Spezifikationssprachen, Requirements Engineering
Entwicklung in Einzelarbeit	Entwicklung im Team	Brookssches Gesetz, Kommunikationsregeln, Gruppenorganisation, Schnittstellendefinition
einschrittige Lösung	Lösung in vielen Schritten	Projektmanagement, Organisation, Ermittlung des Projektfortschritts, Validierung nach jedem Schritt statt nur einmal
Lösung besteht aus einer Komponente	Lösung in viele Komponenten zerlegt	Schnittstellenbeschreibung, Administration von Komponenten
Entwickler und Benutzer einheitlicher Personenkreis	Entwickler und Benutzer ungleiche Personen/Vorbildung	Falschbedienung vorsehen, Dialoge planen, Handbücher, Tutorials
Kurzlebige Programme, Unikate	Langlebige Programme, Programmfamilien	Versionskontrolle, Release Management, Strukturierung, Dokumentation
Umfangreiche Werkzeuge und Automatisierung nicht wirtschaftlich	Werkzeuge und Automatisierung sind Wettbewerbsfaktoren	Ausbildung (Wissen über die Werkzeuge), Mehr Einarbeitungsaufwand, Werkzeugintegration, Erkennen “wiederkehrender Aufgaben” notwendig

angelehnt an [Kla2009] H. Klaeren. Skript Software Technik, Universität Tübingen, 2009

1.2 Historische Entwicklung (Ursachen)

Grösse von Software:

Klassifikation nach W. Hesse:

Kategorie	Anzahl Code-Zeilen	Personenjahre zur Entwicklung
sehr klein	bis 1.000	bis 0,2
klein	1.000 - 10.000	0,2 - 2
mittel	10.000 - 100.000	2 - 20
groß	100.000 - 1 Mio.	20 - 200
sehr groß	über 1 Mio.	über 200

- **Fließende Grenze zwischen „groß“ und „klein“ (geschätzt ca. bei 5-10 kLOC)**
- **Mit wachsender Aufgabenkomplexität zunehmend höhere Abstraktionsstufe und Automatisierung**

1.2 Historische Entwicklung (Ursachen)

Grösse von Software:

	Promotion	Core	User	Visitors	User-Kompl.
■ Sehr kleines Projekt	1	2	10	200	0.01
■ Kleineres Projekt	1	10	30	3000	0.25
■ Mittleres Projekt	4	100	2000	20.000	1
■ Große Hochschule	100	4000	40.000	1 Mio.	100
■ Tickets Staatl. Bahn	15	8000	4 Mio.	20 Mio.	200
■ ID Paraguay	50	15K	8 Mio.	> 8 Mio.	400
■ eHealth–Germany	1000	200K	80 Mio.	> 80 Mio.	4000

aus [Gre10] T. Grechenig, M. Bernhart, R. Breiteneder, K. Kappel: Softwaretechnik, Pearson, 2010

1.3 Software-Krise

Erkennung und Formulierung der Software-Krise (Software Crisis) auf der NATO-Tagung in Garmisch-Partenkirchen 1968 (NATO Science Committee [NR69]):

Software-Entwicklung geschieht vollkommen unsystematisch und ist damit ein unwissenschaftliches, höchst fehlerbehaftetes Vorgehen!

Vorgeschlagene Problemlösung:
Suche nach einem wissenschaftlichen Vorgehen, das eine systematische und qualitätsorientierte Entwicklung ermöglicht:

ingenieurswissenschaftliches Vorgehen $\Rightarrow$ Software Engineering
(im Deutschen oft: Softwaretechnik)

[NR69] P. Naur and B. Randell. Editors. *Software Engineering. Report on a conference sponsored by the NATO Science Committee*. Garmisch, Germany, 7th to 11th October 1968. NATO Scientific Affairs Division, Bruxelles, January 1969.

1.4 Begriff und Disziplin “Software Engineering”

- **Grundvorstellung:**
  Software Engineering = Softwareerstellung nach Ingenieursmethoden

- **Literaturzitate:**

  79 Boehm:
  Software Engineering ist die praktische Anwendung wissenschaftlicher Erkenntnisse auf den Entwurf und die Konstruktion von Computerprogrammen.

  75 Denis:
  Software Engineering ist die Anwendung von Prinzipien, Fähigkeiten, Kunstfertigkeiten auf den Entwurf und die Konstruktion von Programmsystemen.

- **Literaturzitate: (Fortsetzung)**

74 Parnas:
Software Engineering ist Programmierung unter mindestens einer der zwei Bedingungen:
(1) Mehr als eine Person schreibt und benutzt das Programm.
(2) Mehr als eine Version des Programmes wird erzeugt.

85 Farley:
Software Engineering ist die technische und organisatorische Disziplin zur systematischen Herstellung und Wartung von Softwareprodukten, die zeitgerecht und innerhalb vorgegebener Kostenschranken hergestellt und modifiziert werden.

75 Bauer:
Das Ziel der Softwaretechnik ist die wirtschaftliche Herstellung zuverlässiger und effizienter Software.

1.4 Begriff und Disziplin “Software Engineering”

Software **Engineering**

↓

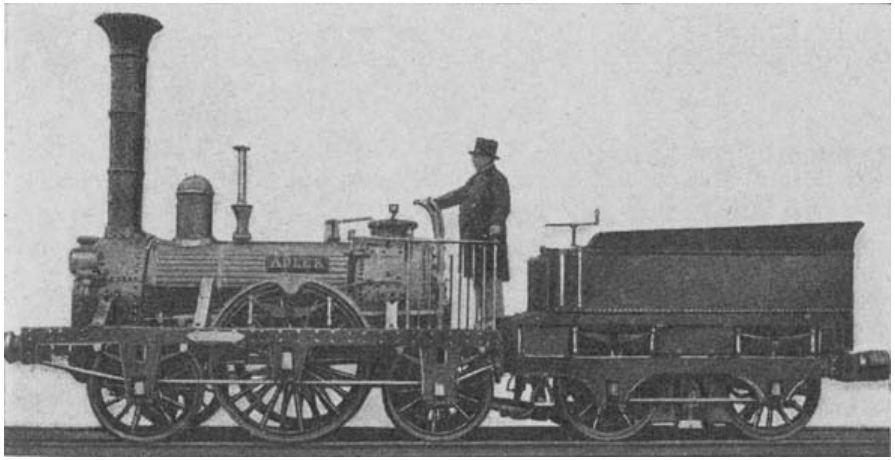
„... die schöpferische Anwendung wissenschaftlicher Prinzipien auf Entwurf und Entwicklung von Strukturen, Maschinen und Apparaten oder Herstellungsprozessen im Hinblick auf eine gewünschte Funktion, Wirtschaftlichkeit und Sicherheit von Leben und Eigentum“ [Encyclopedia Britannica]

- **Kreative Anwendung wissenschaftlicher Grundlagen zur Erbringung einer gewissen Funktion (Herstellung und Einsatz)**
  (Mathematik, Physik, Mechanik, Elektrotechnik, Chemie, …)
- **Wirtschaftliche Abwägungen**
  (Gewinnerzielung, Kostenminimierung, rationelle Fertigung)
- **Sicherheit und Qualität**
  (Sicherheit für Menschen und für Güter; zuverlässige Funktion)

1.4 Begriff und Disziplin “Software Engineering”

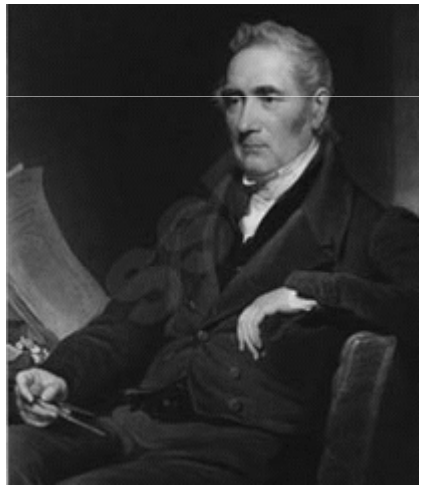
Was ist ingenieurmäßiges Vorgehen / Engineering?

Historisch: Wie ist der Begriff *Engineer* / Ingenieur entstanden?



Bedienende einer komplexen, auf mathematisch/physikalischen Grundsätzen aufbauenden Maschine
Engine ⇔ Engineer

Entwickelnde einer komplexen, auf mathematisch/physikalischen Grundsätzen aufbauenden Maschine
Engine ⇔ Engineer
(hier George Stephenson 1781 – 1848)



1.4 Begriff und Disziplin “Software Engineering”

Was sind die Ziele des ingenieursmäßigen Vorgehens?

Berechenbare und nachvollziehbare Bestimmung von:

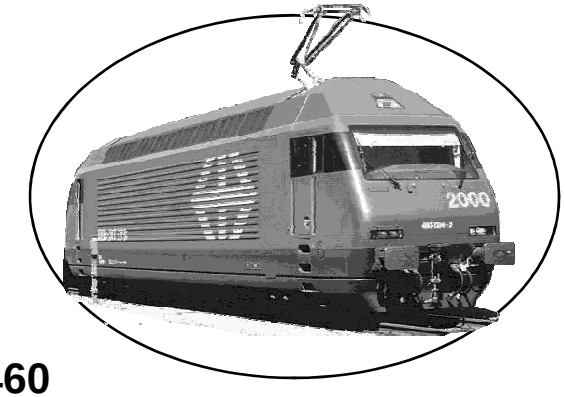
- Kosten, Gewinn
- Sicherheit, Sicherheitsbestimmungen
- Zeitplan, Aufwand, Produktionsplanung, (rationelle Produktion)
- Konstruktion, Produktion, Inbetriebnahme
- berechenbare Fertigungstechnologien
- Zuverlässigkeit, Einhaltung von Normen
- Benutzerfreundlichkeit, Bedienkomfort, Wartung, Außerbetriebsetzung
- …

1.4 Begriff und Disziplin “Software Engineering”

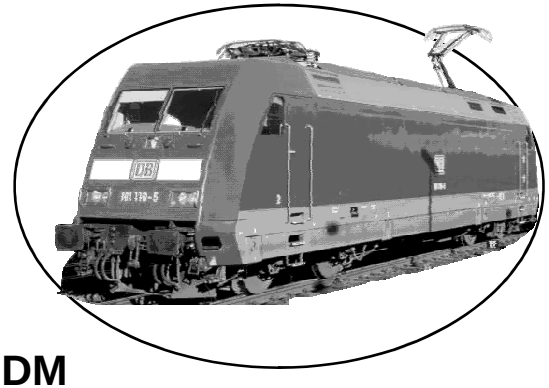
Was ist ingenieursmäßiges Vorgehen?

Abwägung (wirtschaftlicher) Eigenschaften

Beispiel: Hochleistungslokomotiven



SBB
BR 460
Kosten: 8,1 Mio. DM
Zuverlässigkeit: maximal
Umweltverträglichkeit: hoch
Bedienkomfort: maximal
Erweiterungsfähigkeit: ja
Aussehen: Optimiertes Design
Verkauf in vier weitere Länder
Renommierobjekt



DB AG
BR 101
Kosten: 5,6 Mio. DM
Zuverlässigkeit: hoch
Umweltverträglichkeit: sehr hoch
Bedienkomfort: mittel
Erweiterungsfähigkeit: nein
bereits abgeleitetes System
Aussehen: entfeinertes Design
Verkauf in zwei weitere Länder
partielles Renommierobjekt, spätere Verwendung ausschließlich im Gütertransport geplant

Was ist die besondere Kunst daran?

Die Produktinformationen werden bestimmt, bevor die Maschinen / Systeme überhaupt bestehen; basieren daher voll auf Erfahrungen! Trotzdem sind die Werte relativ genau vorherberechnet / abgeschätzt.

Beispiele:

- **Kosten** (Personal, Maschinen, Rohstoffe /-material, Zukaufteile, …)
- **Zeit** (Entwicklungsdauer, Produktionsdauer, Testdauer, Inbetriebnahme, …)
- **Komplexität** (Entwicklungskomplexität, Bedienkomplexität, …)

1.4 Begriff und Disziplin “Software Engineering”

Übertragung des ingenieursmäßigen Gedankens
auf den Entwurf und die Entwicklung von Software

- **Kreative Anwendung wissenschaftlicher Grundlagen zur Erbringung einer gewissen Funktion**
  Mathematik, Algorithmen, Modelle, Sprachen und ihre Konzepte

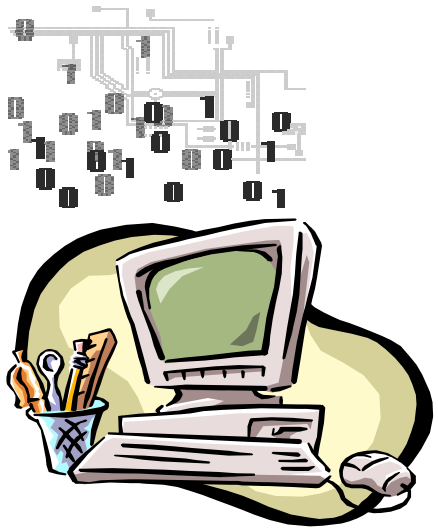
- **Wirtschaftliche Abwägungen**
  Kostenminimierung durch rationelle Fertigung und Automatisierung, Entwicklungswerkzeuge, Entwicklungsprozessmodelle, Qualitätssicherung und Metriken

- **Sicherheit**
  Entwicklungsprozesse, nachprüfbare, formale Modelle, Qualitätssicherung und Metriken

Module gelten als sehr frühes Konzept
zur Erreichung aller drei oben genannten Ziele.

Schwierigkeiten der Übertragung
auf den Entwurf und die Entwicklung von Software

- **Was sind die naturwissenschaftlichen Grundlagen, Naturgesetze ?**
- **Unstetige digitale Welt statt analoger mit stetigen Funktionen**
- **„Software ist soft“**



Stark divergierende Anforderungen an Software

- Software soll preiswert sein („Software kostet nichts.“)
- Software ist zum Teil sehr langlebig: 10 bis 15 Jahre (z.B. in Produkten wie Pkws oder Waschmaschinen oder Verwaltungsanwendungen)
- Software soll permanent aktuell sein (permanente Updates)
- Software soll möglichst modular einsetzbar sein
- Software soll leicht überschaubar sein
- Software soll persönlich sein (*Personalization*) und trotzdem weltweit einsetzbar
- Software soll zuverlässig funktionieren

Software-Krise und Software Engineering

Zusammenfassung und Ausblick

UNIVERSITÄT OSNABRÜCK

1 **Software-Krise und Software Engineering**
   - 1.1 Krisenstimmung
   - 1.2 Historische Entwicklung
   - 1.3 Software-Krise
   - 1.4 Begriff und Disziplin

2 **Grundlagen des Software Engineering**

3 **Projektmanagement**

4 **Konfigurationsmanagement**

5 **Software-Modelle**

6 **Software-Entwicklungsphasen, -prozesse, -vorgehensmodelle**

7 **Qualität**

8 **Fortgeschrittene Techniken**

→ **Wege im Umgang mit der Software-Krise**

Grundlagen und Prinzipien (insbesondere Modularisierung)