Software Engineering

Kapitel 01 – Grundlagen des Software Engineering

Gliederung

1. Einführung: Was ist Software Engineering
2. Geschichte des Software Engineering
3. Motivation: Warum ist Software Engineering wichtig?
4. Definitionen für Software Engineering
5. Einordnung des Software Engineering
6. Ziele des Software Engineering
7. Zusammenfassung

Was ist Software-Engineering

Einführung, Motivation und Definition

Fragen an Sie…

- Was ist Software Engineering eigentlich /
- Was bedeutet Software Engineering aus Ihrer Sicht? Was gehört dazu?

- Brauchen wir Software Engineering? Warum/Wofür?

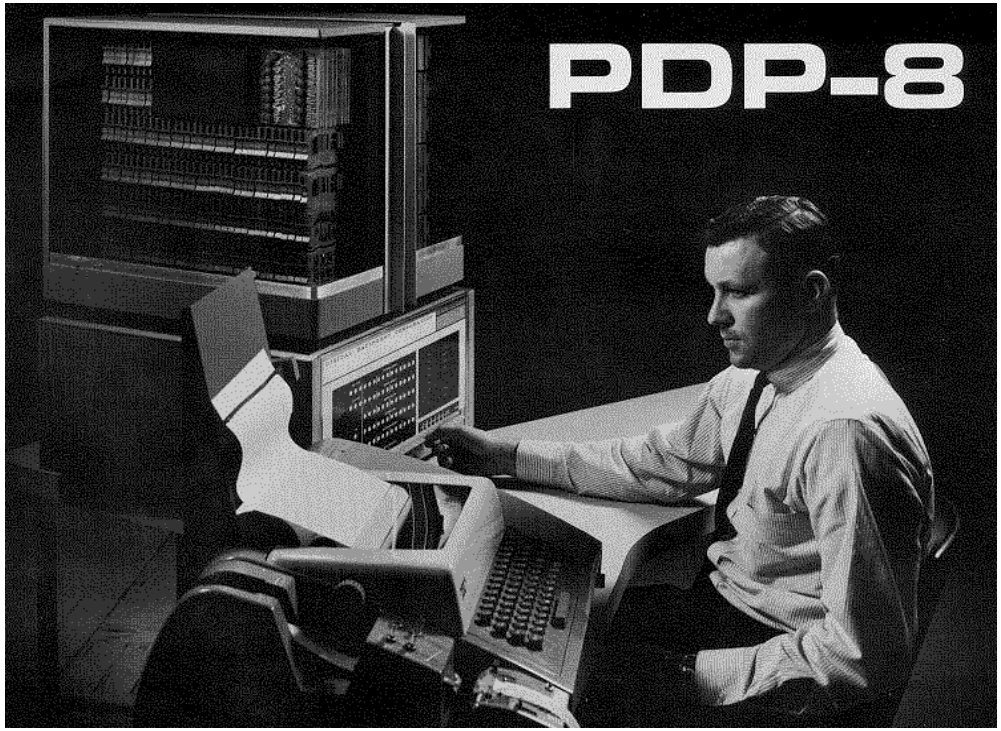
Was müssen wir tun, damit wir ein Softwareprojekt zum Scheitern bringen?

- Positive Formulierungen!

Geschichte des Software Engineering

Wie alles anfing…

Softwaretechnik im Jahr 1972 (1)



digital

pdp8/e & pdp8/m

small computer handbook

Softwaretechnik im Jahr 1972 (2)

- Damalige Antworten auf die Frage:
  „Wie sollte Software entwickelt werden?“

PROGRAMMING RULES

The most successful method of programming is to begin a program as simply as possible, test it, and then add to the program until it performs the required job. Before beginning the programming, the programmer should become familiar with the programs that he will be using. Refer to Chapter 4 for a description of the standard programs and refer to Appendix A for a complete list of the PDP-8/E compatible programs. For best results, the programmer should avoid the use of the following device codes:

1. Devide code 0 (reserved for processor)
2. Device code 3

Was meinen Sie zu dieser Aussage?

Geschichte des Software Engineerings (1)

- Ende der 60er:
  - Bedarf für Softwaretechnik neben der reinen Programmierung erstmals voll erkannt
  - Vorher sind zahlreiche große Programmentwicklungen (möglich durch verbesserte Hardwareeigenschaften) gescheitert
  - Arbeiten von Dijkstra 1968 (u.a. gegen Verwendung von GOTO) und Royce 1970 (Software-Lebenszyklus),
  - Top-Down-Entwurf, grafische Veranschaulichung (Nassi-Shneiderman-Diagramme)
- Mitte der 70er:
  - Top-Down-Entwurf für große Programme nicht ausreichend, zusätzlich Modularisierung erforderlich
  - Entwicklung der Begriffe Abstrakter Datentyp, Datenkapselung und Informat. Hiding
- Ende der 70er:
  - Bedarf für präzise Definition der Anforderungen an ein Softwaresystem,
  - Entstehen von ersten Vorgehensmodellen

Geschichte des Software Engineerings (2)

- 80er Jahre:
  - Vom Compiler zur Entwicklungsumgebung (Editor, Compiler, Linker, Debugger, …)
  - Weiterentwicklung der Modularisierung und der Datenkapselung zur objektorient. Programmierung (ab Mitte 80er C++)
- 90er Jahre:
  - Neue OO Programmiersprache Java (ab Mitte 90er)
  - Application Frameworks zur Vereinfachung von Design und Programmierung
- 1995:
  - „*Die 3 Amigos*“ definieren die **Unified Modeling Language** (UML) als Quasi-Standard
- 1997:
  - UML in Version 1.1 bei der OMG zur Standardisierung eingereicht und angenommen
  - UML ist keine Entwicklungsmethode (Phasenmodell), nur Beschreibungssprache
- 1999:
  - Entwicklungsmethode: Unified Process (UP) und Rational Unified Process (RUP)

Geschichte des Software Engineerings (4)

- Heute:
  - (Agile) Vorgehensweisen auf individuelle Projektanforderungen abgestimmt
  - CASE-Methoden und –Tools orientieren sich an der UML
  - Aktueller Stand Dezember 2017: UML 2.5.1
  - Aufbauend auf Analyse und Design erzeugen Codegeneratoren Programmgerüste
  - Haupttätigkeiten bei Softwareentwicklung sind Analyse und Design, vieles andere versucht man zu automatisieren

Was z.B.?

Warum ist Software Engineering nun wichtig?

Einige Motivationen:

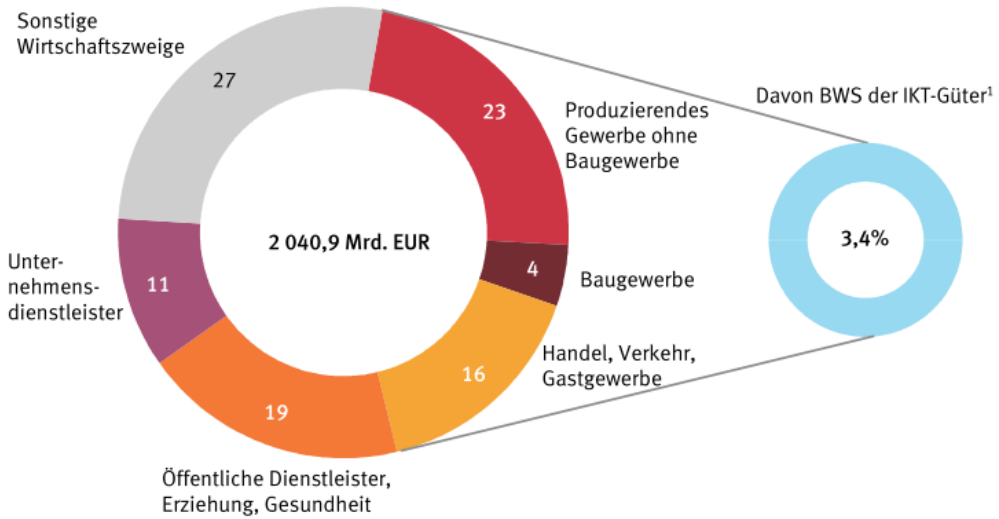
1. Wirtschaftliche Bedeutung
2. Umfang und Komplexität von Software
3. Qualität (Beispiele für Softwarefehler)
4. Stand der Digitalisierung
5. Vielfalt der Eigenschaften von Software

Warum ist Software Engineering so wichtig?

Software nimmt immer mehr Bedeutung in allen Bereichen des Alltags ein:

- Software als Wirtschaftsgut; Wirtschaft aller Industrieländer hängt von Software ab
- Anteil Softwarekosten gegenüber Hardwarekosten steigt stark
- Anteil der Software an der Wertschöpfung steigt erheblich (z.B. Automotive, Medizin)

Schaubild 1 **Bruttowertschöpfung in Deutschland 2009**
in %

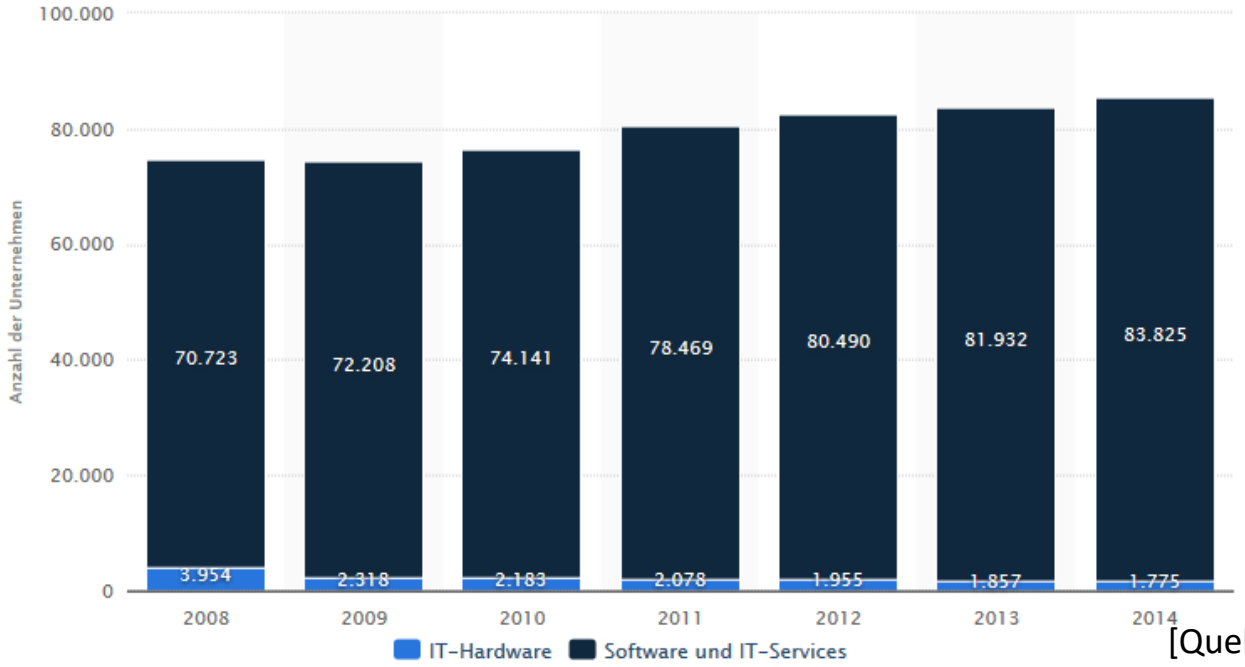


1 Ohne Handel und Leasing von IKT-Produkten.

[Quelle: Statistisches Bundesamt 2013]

Motivation: 1. Wirtschaftliche Bedeutung von Software in Deutschland

- 2014 gab es in Deutschland ca. 84.000 Unternehmen im Bereich Software und IT-Services
- 2015 gab es etwa 1 Mio. Beschäftigte im Bereich IKT (trotzdem existiert ein Mangel an Fachkräften)
- 2010 wurde ein Gesamtumsatz von ca. 281 Mrd. Euro im Bereich IKT erwirtschaftet
- 2015 machte deutscher Branchenführer SAP ca. 21 Mrd. Euro Umsatz



[Quelle: Statista 2016]

Motivation: 2. Umfang von Software

- Produkte:
  1. Mars Curiosity Rover: 5 Mio. Code-Zeilen
  2. Firefox: 10 Mio. Code-Zeilen
  3. MS Visual Studio: 50 Mio. Code-Zeilen
  4. Facebook: >61 Mio. Code-Zeilen (inkl. Backend)
  5. Luxusklasse Auto: 100 Mio. Code-Zeilen
  6. Google (alle Internet Services): 2 Mrd. Code-Zeilen

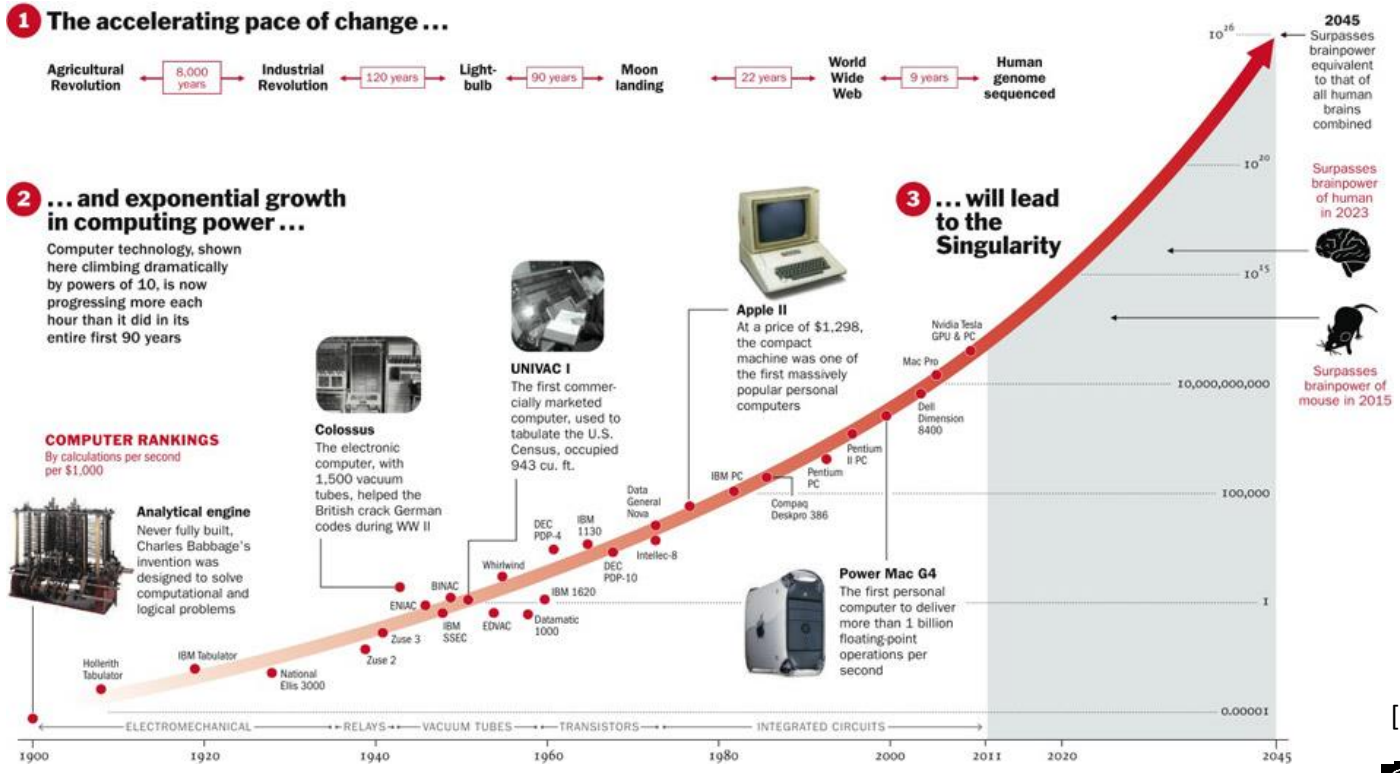
- Unternehmen (Anfang 2000):
  - Citicorp Bank: 400 Mio. Code-Zeilen
  - AT&T: 500 Mio. Code-Zeilen
  - General Motors: 2 Mrd. Code-Zeilen

[Quelle: http://www.informationisbeautiful.net/visualizations/million-lines-of-code/]
[Quelle: Prof. Dr. B. Rumpe, RWTH Aachen]

Motivation: 2. Komplexität von Software

Wachsende Komplexität:

- Immer neue Anwendungsgebiete (z.B. Data Mining, Data Warehouse, E-Business, Bildverarbeitung)
- Immer komplexere Aufgabenfelder (z.B. Weltraum-, Medizintechnik, KI, ML)



[Quelle: Time mag. 2.2011]

Motivation: 2. Komplexität von Software (Wirthsches Gesetz)

- Software wird in kürzerer Zeit langsamer, als Hardware schneller
- Software is a gas (N. Myhrvold) **Bedeutung?**
- Software expands to fill the available memory (C. N. Parkinson)
- Allgemeine Aussage: Komplexität von Software hebt Gewinne durch Hardwareentwicklung mehr als auf
- Plädoyer für schlanke Software:
  - lieber Rückbesinnung auf einfache Algorithmen und Werkzeuge
  - als “fatware” (unnötig große Software)

Beispiele für ständig langsamer werdende Software?

Motivation: 2. Komplexität von Software

- Wachsendes Outsourcing/Offshoring von Softwareentwicklung:
  - Firmen geben Softwareentwicklung an externe Software-Häuser ab
  - Schätzungen: 55% intern, 45% extern
- Wachsende Altlasten:
  - Jedes Jahr wird mehr Software entwickelt
  - Anwendungssoftware hat häufig Lebenszeit von >20 Jahren
  - Immer mehr Entwicklerkapazität wird durch Wartung und Pflege gebunden
- Wachsende Qualitätsanforderungen:
  - Softwarefehler und –ausfälle können immese Folgen haben (finanzieller Natur, Gefahr für Leib und Leben,…)
  - Gefahr: langfristige Schädigung des Rufs eines Software-Lieferanten

Motivation: 3. Qualität von Software

Finden Sie Beispiele!

- **Softwarefehler sind allgegenwärtig:**
  - Handy bis zu 600 Fehler (d.h. 3 Fehler pro 1.000 LoC)
  - Windows95 bis zu 200.000 Fehler (d.h. 20 Fehler pro 1.000 LoC)
  - Space Shuttle weniger als 1 Fehler pro 10.000 LoC
  - Gefundene Defekte in 1000 Zeilen Quellcode: (M. Cusumano, MIT 1990)
    - 1977: 7 - 20 Defekte
    - 1994: 0,05 - 0,2 Defekte

- **Bedeutung eines Defektniveaus von 0.1%:**
  - Pro Jahr: 20.000 fehlerhafte Medikamente;
300 versagende Herzschrittmacher
  - Pro Woche: 500 Fehler bei medizinischen Operationen
  - Pro Tag: 16.000 verlorene Briefe
18 Flugzeugabstürze
  - Pro Stunde: 22.000 falsch gebuchte Schecks

- Ergebnis: auch in Zukunft massive QM Anstrengungen notwendig

Kleiner Fehler, große Auswirkung:

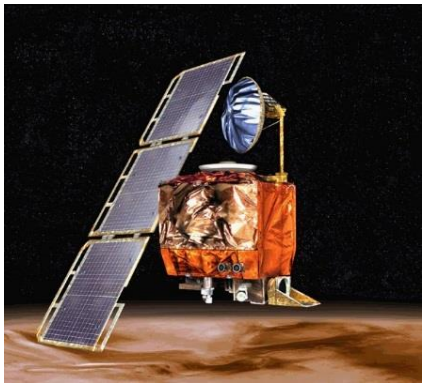
- Jungfernflug der Ariane-5
- 04. Juni 1996

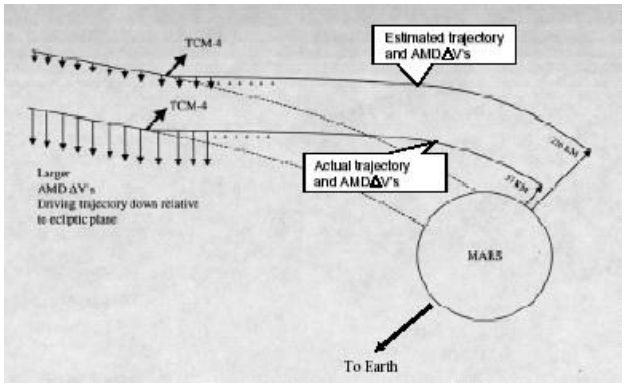
[Quelle: ESA]

- ca. 37 Sekunden nach Start wurde Selbstzerstörung ausgelöst
- Grund:
  - Software von Vorgänger (Ariane-4) teilweise übernommen
  - Geschwindigkeit der neuen Rakete höher
  - Unnötiges Kalibrierungsprogramm für Trägheitssensoren läuft
  - Pufferüberlauf bei Konvertierung von 64-bit float nach 16-bit int
  - Umschalten auf zweites, redund. System, Fehler wird gleich behandelt
- Schaden: 500 Mio. US$
- Lösung: z.B. Exception-Handling

Kennen Sie weitere Beispiele?

Beispiel 2:
Mars Climate Orbiter (1999)





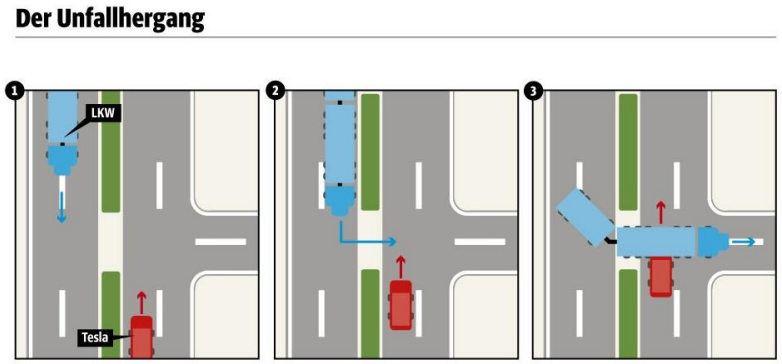
- Unentdeckte Modellierungsfehler bei Veränderung der Sondengeschwindigkeit
- Trajektorien-Korrektur-Maneuver 5 wurde nicht ausgeführt
- Der Systementwicklungs-Prozess achtete nicht genügend auf den Übergang von Entwicklung zum Betrieb
- Mangelnde Kommunikation zwischen den Projekt-Teilen
- Navigationsteam:
  - Unzureichend vertraut mit der Sonde
  - Unterbesetzt
  - Unzureichend trainiert
- Mangelhafte oder fehlende Verifizierung und Validierung der Software

Spektakuläre Softwarefehler sind kein Einzelfall Beispiele aus dem letzten Jahrtausend

- 1982: F-117 (Absturz eines Tarnkappenbombers wegen Vertauschung von Daten des Höhen- und Seitenruders)
- 1985-87: Therac-25 (Personen erhielten überhöhte Strahlendosis, da Software falsch dosierte)
- 1990: AT&T-Telefonnetz (70 Mio. Gespräche innerhalb von 9 Stunden nicht vermittelbar)
- 1991: Patriot-Raketen-Fehler (28 Tote wegen Rundungsfehler in Zeit nach Start, Rakete flog 0.57km zu weit)

Spektakuläre Softwarefehler sind kein Einzelfall
Beispiele der jüngeren Vergangenheit

- 2016: Tödlicher Unfall mit selbst fahrendem Tesla-Auto
  wahrscheinliche Unfallursache:
  - Autopilot konnte nicht zwischen weißer Lastwagenplane und hellem Himmel unterscheiden
  - Software hielt LKW für hohes Straßenschild



[Quelle: sueddeutsche.de, 01.07.2016]

[Quelle: Polizei]

Aber es kann noch schlimmer kommen…

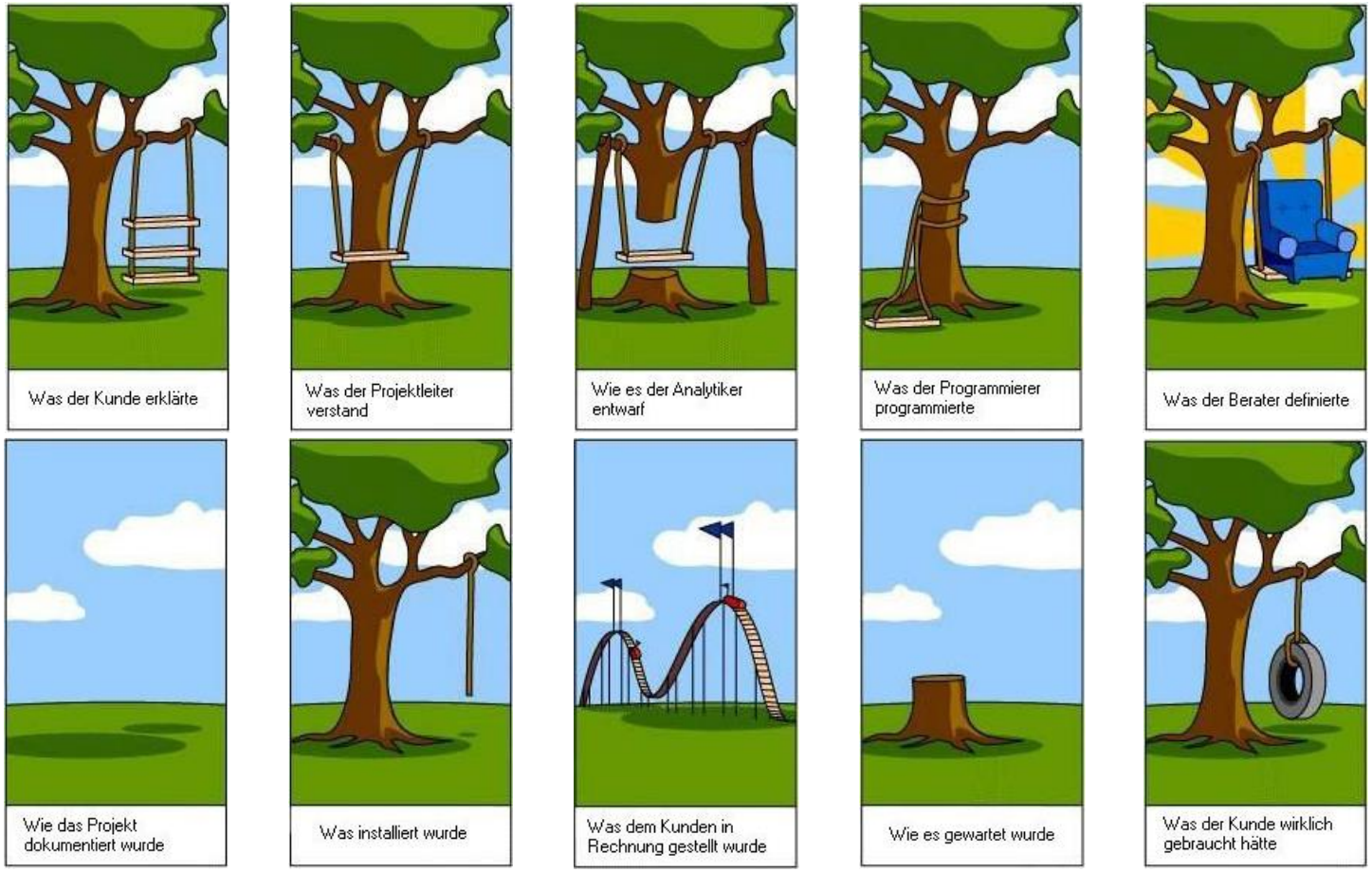
Russische Satelliten interpretieren am 26.09.1983 Lichtreflektion an Wolken
als Start von 5 US-Interkontinental-Raketen

→ Beinahe dritter Weltkrieg

→ Das russische Protokoll sah in einem solchen Fall eigentlich einen direkten Gegenschlag vor

[Quelle: http://mentalfloss.com]

Beispiel für Fehler in verschiedenen Projektphasen



Motivation: 4. Stand der Digitalisierung

- **Wearables**
  Brillen, Armbänder, Kleidung

- **Autonome Systeme**

  Audi Hockenheimring 10.2014

  [Quelle: https://www.youtube.com/watch?v=1AWAzAxEVbM]

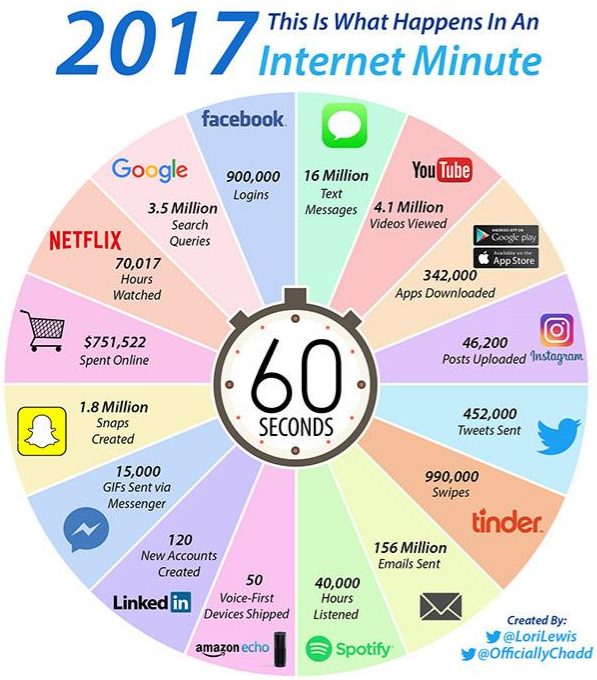
- **Predictive Crime Analytics**

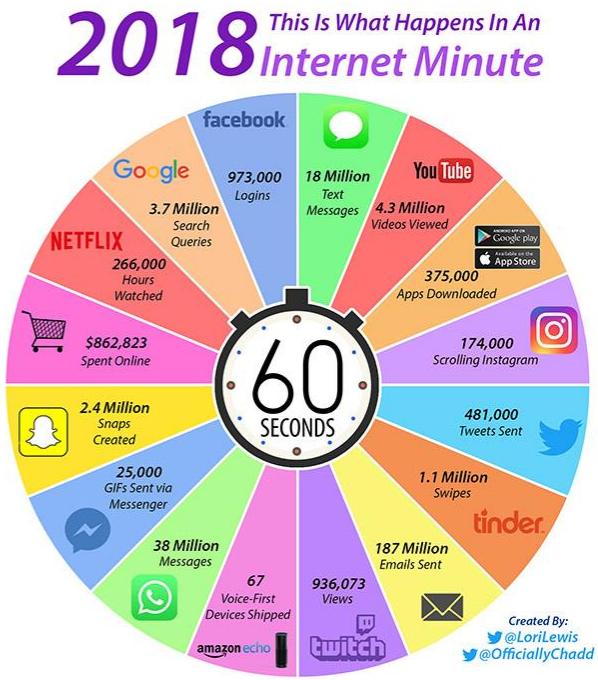
  [Quelle: https://www.youtube.com/watch?v=5n2UjBO22EI]

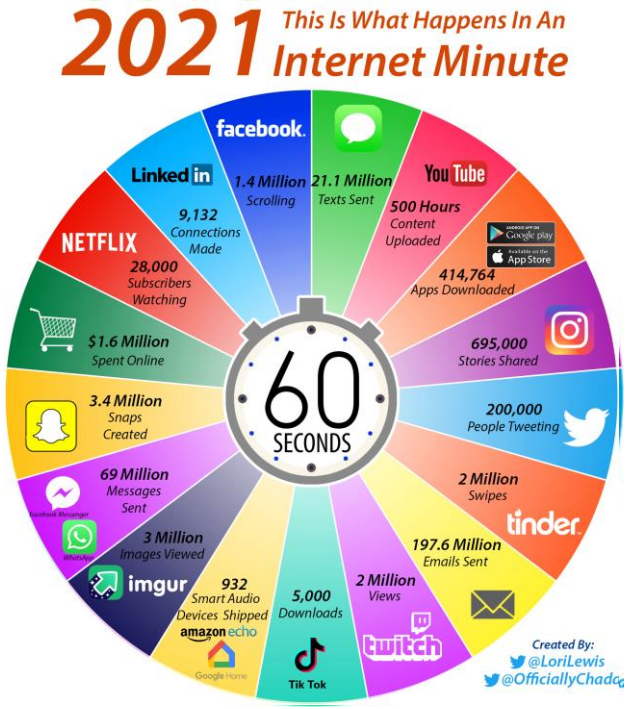
- **Brain reading**

  [Quelle: https://www.youtube.com/watch?v=76lIQtE8oDY]

Motivation: Stand der Digitalisierung Gegenwart …

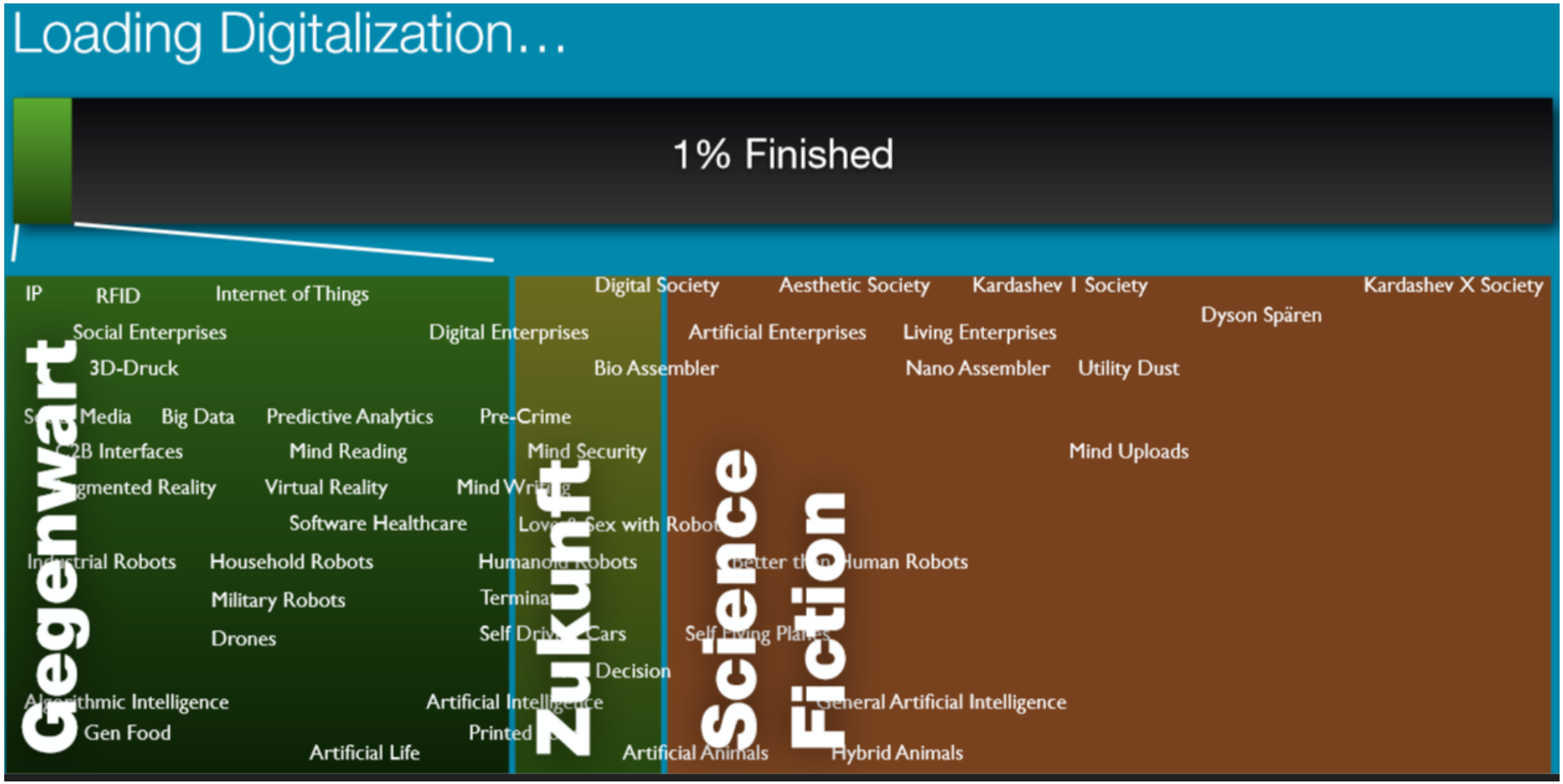






[Quelle: www.visualcapitalist.com]

Motivation: Stand der Digitalisierung Gegenwart und Zukunft



[Quelle: Prof. Tim Bruysten, Jahrestreffen SW Foren Leipzig, 2014]

Motivation: 5. Vielfalt von Software

• Produkt	vs	Individualentwicklung
• Systemsoftware (OS, Compiler)	vs	Anwendungssoftware
• Produktintegriert	vs	für reine Computersysteme
• Datenintensiv	vs	berechnungsintensiv
• Monolithisch	vs	verteilt
• Standalone	vs	integriert

→ Jeweilige Ausprägung durch Anforderungen und Domäne charakterisiert

Soll-Eigenschaften guter Software

- zuverlässig:
  - Software muss bei Versagen physische oder ökonomische Schäden vermeiden
- wartbar:
  - Software muss leicht erweiterbar und anpassbar an neue Anforderungen sein
  - Software sollte möglichst Plattform-unabhängig sein
  - Software-Code und –Produkt muss gut dokumentiert und lesbar sein
- benutzerfreundlich:
  - Software muss sich nach Bedürfnissen von Nutzern richten (z.B. ergonom., intuitiv)
- performant:
  - Software muss geforderte Funktionalität schnellstmöglich erbringen
- effizient:
  - Software muss ressourcenschonend gegenüber Host-System sein

Erweiterte Liste wichtiger Software-Eigenschaften

Merkmale und ihre Bedeutung:

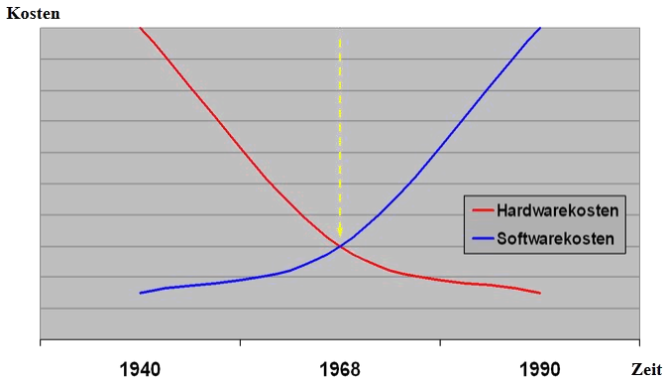
- Funktionalität (SW erfüllt Aufgabe, wegen der sie erstellt wurde)
- Benutzbarkeit (Erlernbarkeit, Softwareergonomie)
- Verfügbarkeit (Zuverlässigkeit, Ausfallsicherheit)
- Leistungsfähigkeit (Antwortzeiten, Skalierbarkeit)
- Sicherheit (Vertraulichkeit, Unverletzlichkeit)
- Modifizierbarkeit (Wartbarkeit, Erweiterbarkeit, Portierbarkeit)
- Testbarkeit (Überprüfbarkeit der korrekten Funktionsweise)

Definitionen für Software Engineering

und ihr Ursprung

Softwarekrise

- Mitte 60er Jahre: *Softwarekrise*
  - *Hardware schneller → Software wird wichtiger*
  - *Steigende Anforderungen, aber kein Fachpersonal*
  - *Softwareentwicklung durch "Bastelei"*
  - *Systeme werden komplex, unübersichtlich, kaum wartbar*
  - *Kosten für Software übersteigen erstmals Kosten für Hardware*
  - *Große Softwareprojekte scheitern*



- 1967: Weinberg's Second Law: Bedeutung?
  - If builders built buildings the way programmers wrote programs, then the first woodpecker that came along would destroy civilization.

Definitionsversuch 1: Software Engineering

Definition 1: Was gehört zur Softwaretechnik?

Softwaretechnik ist die Anwendung von Prinzipien, Fähigkeiten und Kunstfertigkeiten beim Entwurf und der Erstellung von Programmen und Systemen von Programmen

Künstler, Einzelkämpfer [Dennis, 1975]

Definition 2: Äußere Umstände

Softwaretechnik ist Programmierung unter mindestens einer der folgenden Bedingungen:

- Mehr als eine Person befasst sich mit Erstellung und/oder Gebrauch des Programms
- Mehr als eine Version des Programms wird erstellt werden

Teamarbeit, Pflege von Software [Parnas, 1985]

Definition 3: Ziele der Softwaretechnik

Die Ziele der Softwaretechnik sind die ökonomische Erstellung von Software, die zuverlässig ist und effizient auf realen Maschinen arbeitet.

Wirtschaftlichkeit, Ingenieurdisziplin [Bauer, 1990]

Sommerville:

"Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification to maintaining the system after it has gone into use."

Software Engineering als Ingenieursdisziplin? (1)

Was könnten Gemeinsamkeiten sein?

- Kostengünstige Lösungen erstellen
- Praxisnahe Probleme
- Anwendung von wissenschaftlichem Wissen
- Erstellung von Artefakten
- Im Dienst der Menschheit
- Prozesse, Projektplanung, Kostenkalkulation
- Entwicklung durch Teams (macht Projektmanagement notwendig)
- Systematische Analyse
- Standardisierte Methoden und Technologien zur Produktentwicklung
- Langjähriger Einsatz von Produkten (>10 Jahre)
- Betrieb/Wartung + Erweiterung

Software Engineering als Ingenieursdisziplin? (2)

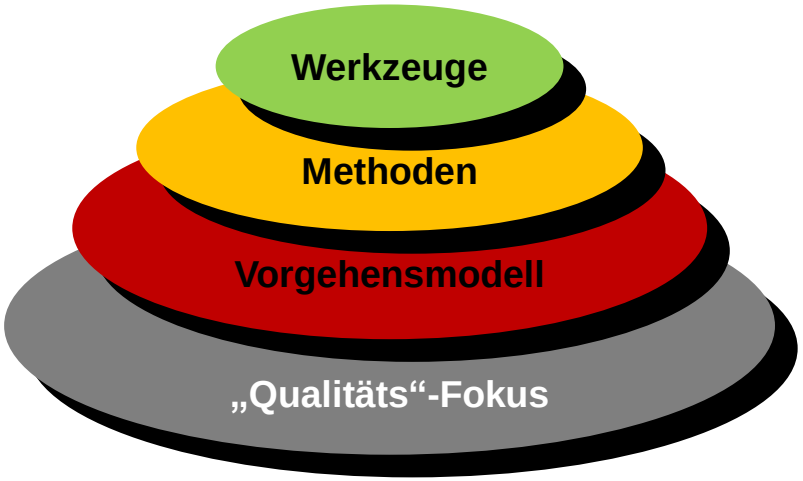
Was könnten Unterschiede sein?

- Software (als Produkt) hat keine physikalischen Eigenschaften
- SW unterliegt keinen physikalischen Gesetzen/Einschränkungen
- I.d.R. beliebig kopier- und modifizierbar
- Fehler oft einfacher korrigierbar
- ABER: kleine Fehler können große Auswirkungen haben (Beispiele s. F. 19-25)
- Es gibt keine Ersatzteile (Defekte sind immer Konstruktionsfehler)
- Keine Fehler durch physischen Verschleiß, trotzdem Alterung
  - Hardware nicht mehr verfügbar (z.B. Speichermedien)

Definitionsversuch 2: Software Engineering

Was ist <u>modernes</u> Software Engineering?

- Software Engineering ist der Einsatz qualifizierter Methoden, Werkzeuge und Vorgehensmodelle zum Erstellen und Betreiben von Software mit dem Ziel,
- Einerseits die Softwarekosten bei der Entwicklung, Wartung und Erweiterung von Programmsystemen zu senken,
- Andererseits zum Erreichen einer höheren Systemqualität



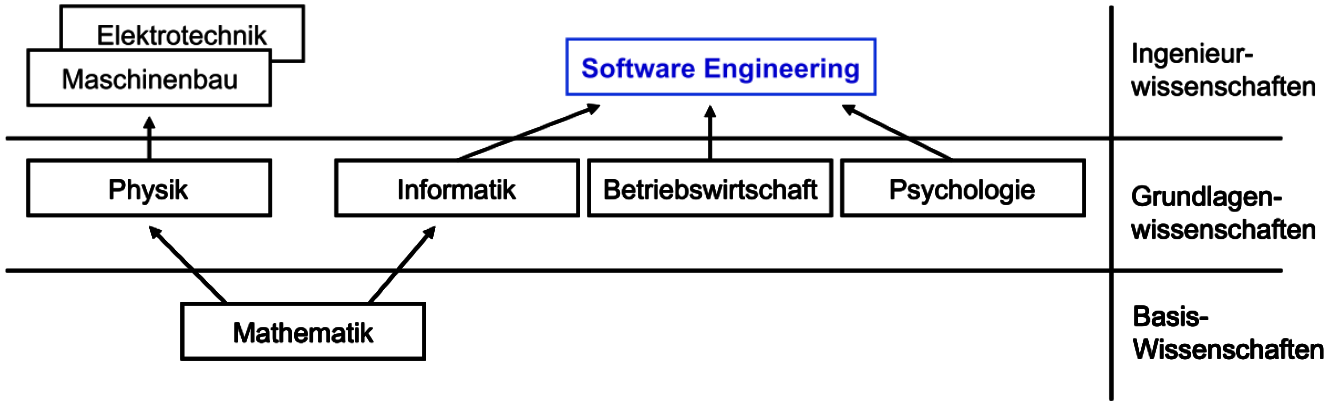
Lösungsansätze:

Werkzeuge:	Einsatz von OOP, CASE-Tools, ...
Methoden:	Beschreiben bewährte Verfahren, wie einzelne Aufgaben möglichst gut durchzuführen sind.
Vorgehensmodelle:	Prozessbeschreibungen, was in welcher Reihenfolge durchgeführt werden soll.
Dokumentation:	Einheitliche Notationen für Entwicklungsergebnisse (UML)

Einordnung von Software Engineering

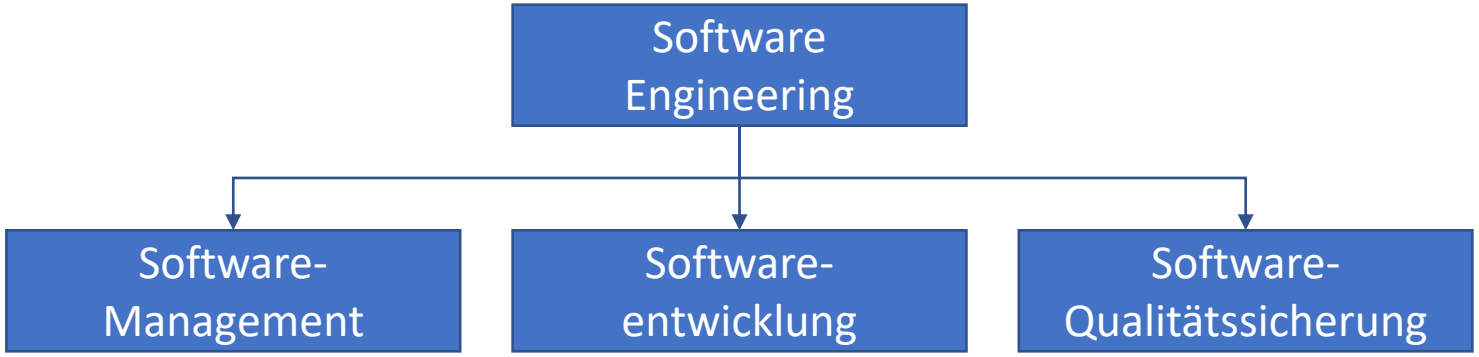
in den Kontext anderer Wissenschaften und der Informatik

Einordnung/Abgrenzung zu anderen Fachbereichen/Wissenschaften



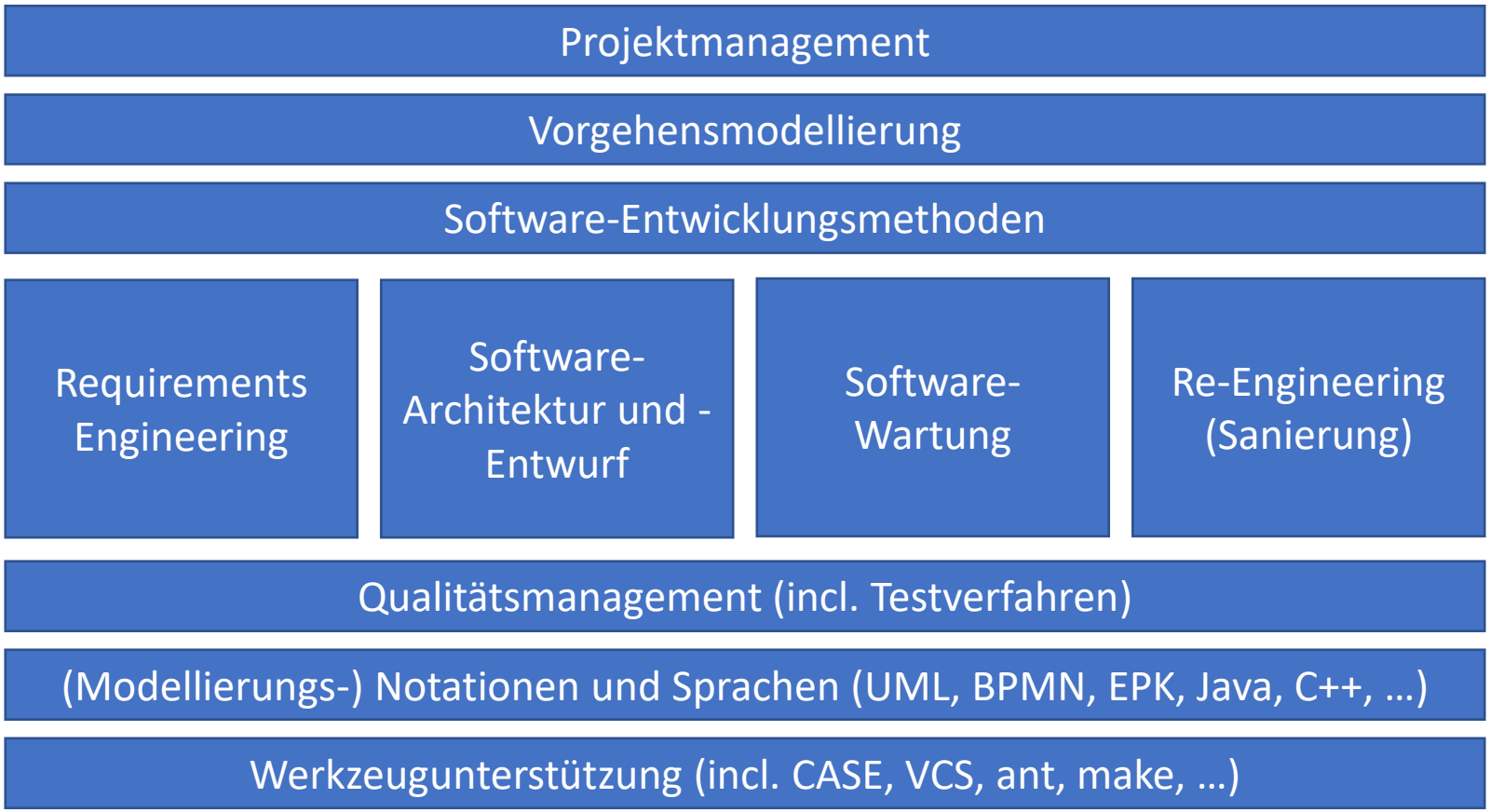
- SE ist der ingenieurwissenschaftliche Teil der Informatik
- So wie z.B. der Maschinenbau der ingenieurwissenschaftliche Teil der Physik ist
- Folgende Aspekte werden durch SE berücksichtigt:
  - Kosten
  - Termine
  - Qualität!?

Einordnung/Abgrenzung in der Informatik



- Software-Management:
  - Planung, Organisation, Leitung, Kontrolle von Softwareentwicklung
- Softwareentwicklung:
  - Aus einem geplanten Software-Produkt ein nutzbares Software-Produkt erstellen, das geforderte Eigenschaften aufweist
- Software-Qualitätssicherung:
  - Sicherstellung der geforderten Produkt- und Prozessqualität von Softwareentwicklung durch konstruktive und analytische Maßnahmen

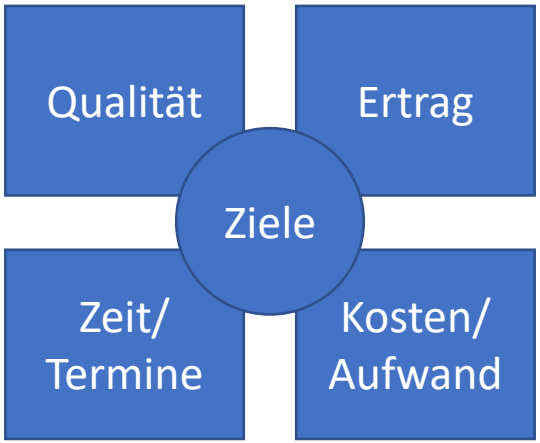
Themengebiete des Software Engineering



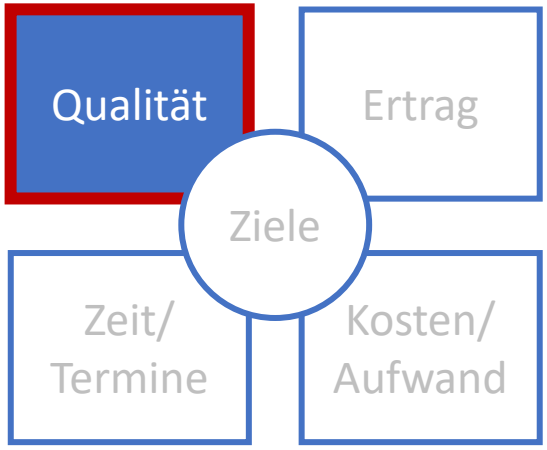
Ziele von Software Engineering

Ziele des Software Engineering

- Nach Balzert:
  - Softwareentwicklung erfolgt nach wirtschaftlichen Aspekten
  - Nicht Entwicklung selbst ist vorrangiger Zweck
  - Sondern die Erfüllung wirtschaftlicher Ziele



Ziele des Software Engineering

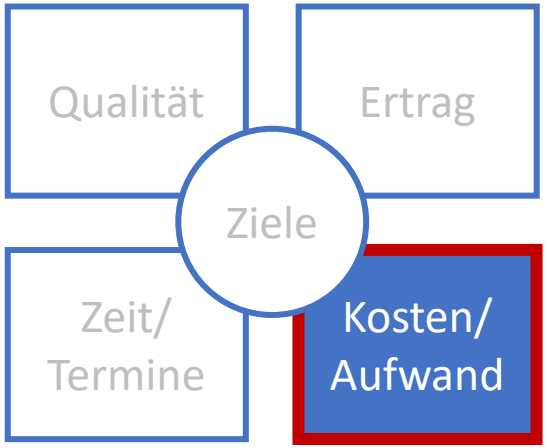


Qualität:

- Definition: Software-Produkt-Qualität ist die Gesamtheit von Eigenschaften eines Software Produkts, die sich auf die Eignung zur Erfüllung gegebener Erfordernisse beziehen.

- Erfordernisse des Anwenders sind entscheidend

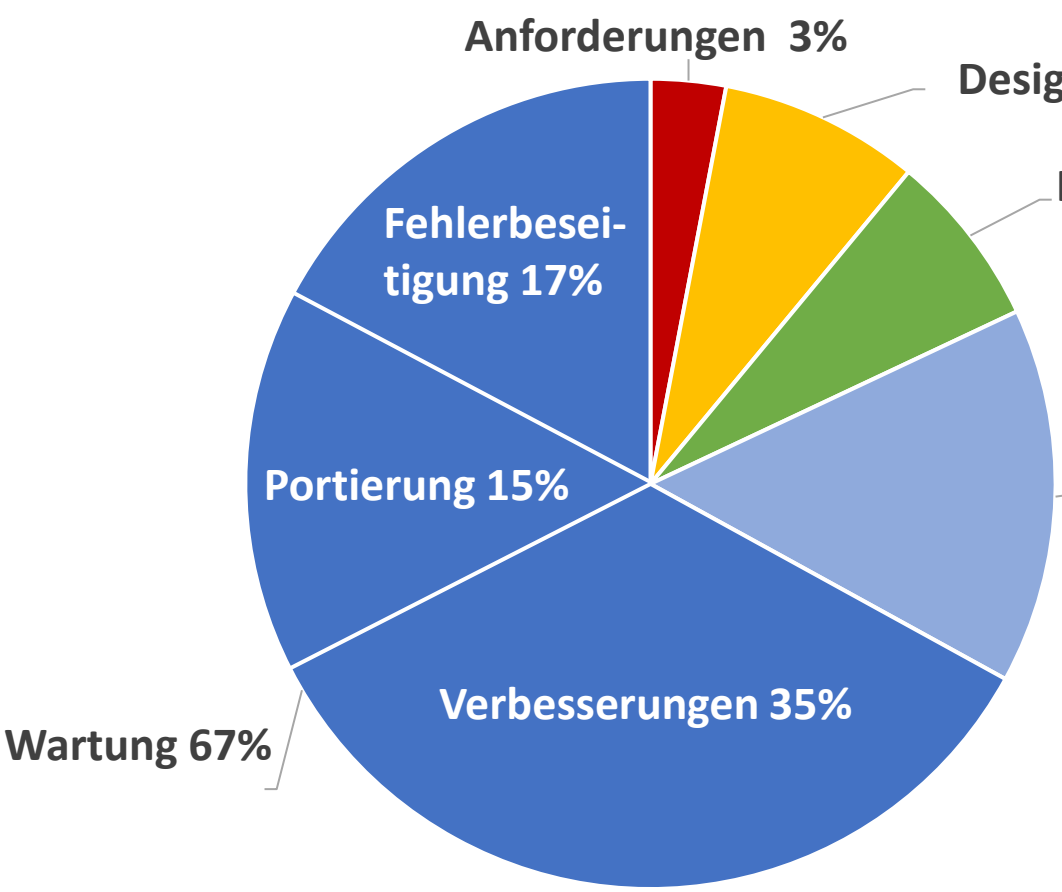
Ziele des Software Engineering



Kosten: (Welche Arten von Kosten?)

- Entwicklungskosten
  - Personal
  - Entwicklungsplatz, Rechner, Stromkosten, …
- Fehlerbeseitigung
  - Anpassung, Erweiterung
- Organisatorische Implementierung
  - Customizing (Beispiel: SAP)
  - Schulung
- Vertriebskosten
  - Marketing, Werbung, Sales, Promotion
  - Personal Selling
- Wartungskosten

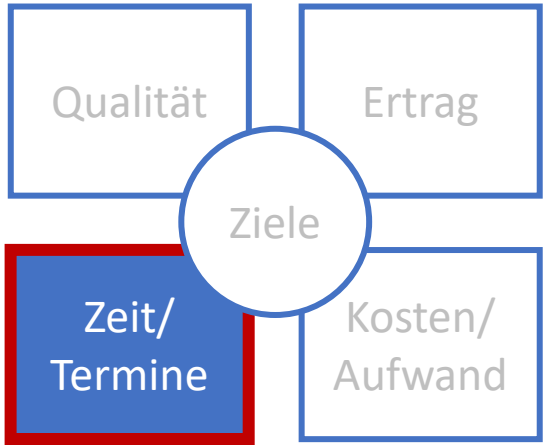
Aufschlüsselung der Kosten von Software: Entwicklung



Anteil der Codeerstellung im Vergleich zu Analyse, Design, Test ist gering

Folgerungen:
Qualität von Anforderungen, Design und Testen extrem wichtig
Software leicht anpassbar und erweiterbar gestalten

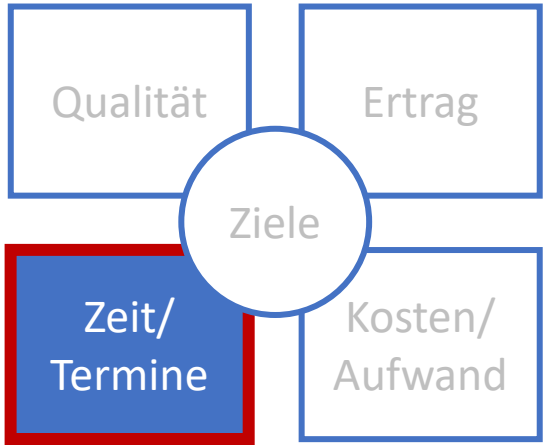
Ziele des Software Engineering



Ziele bezgl. Verfügbarkeit und Einsatzdauer:

- Ziel: kurze Entwicklungsdauer
  - Schnelle Marktpräsenz (time-to-market)
- Konflikt mit Qualität:
  - Sog. “Bananenprodukte” (d.h., sie reifen beim Kunden)
- Marktpräsenz/Einsatzdauer maximieren:
  - Lizenzeinnahmen (sog. “Milchkühe”, “Cash-Cows”)
  - Amortisation (nach welcher Zeit hat sich Investition rentiert?)
- Wichtig: Softwareentwicklung muss planbar sein

Ziele des Software Engineering



Wie verkürzt man die Entwicklungszeit?

- Einsatz von mehr Personal
- Einsatz von Spezialisten
- Mitarbeitermotivation, eingeschworenes Team

- Vorsicht bei der Berechnung der Entwicklungszeit ("chinesischer Dreisatz"):

$$T_{Entwicklung} = \frac{LOC}{P * M}$$

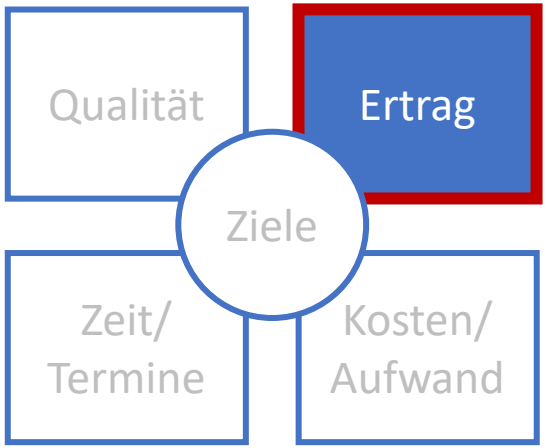
mit

LOC: Lines of Code

P: Produktiviät (LOC/MannJahr)

M: Personalkapaziät (#Mitarbeiter)

Ziele des Software Engineering



Umsatz und Kosteneinsparung:

- Softwarelieferant (-produzent)
  - Lizenzeinnahmen (Verkaufszahl ∗ Preis)
  - Lizenzgebühr für Nutzungsrecht
  - Wartungsgebühr
  - Beratung, Schulung
- Kunde/Anwender:
  - Kosteneinsparungen durch Rationalisierung
  - Stichwort: Amortisation der Anschaffungskosten
  - Schwer zu messende/ zu quantifizierende Komponenten:
    - Schnellere Reaktionszeit (Anfrage … Lieferung)
    - Bessere Information
    - Zufriedenere Endkunden

Zusammenfassung: Aufgabenbereiche des Software Engineering

- Software Engineering vs Programmieren (s. letztes Semester)
  - Projekt-Management (großer und komplexer Projekte)
  - Schätzung von Terminen und Kosten
  - Erfassen von Kunden-, Markt- und Gesetzesanforderungen
  - Änderungsmanagement
  - Qualitätsmanagement (hohes Qualitätsniveau sicherstellen)
  - Wartung und Weiterentwicklung von Produktivsystemen
  - Pflege eines guten Programmierstils
  - Entwicklungswerkzeuge
  - Basis-Prinzipien (Abstraktion, Strukturierung, Hierarchiebildung, Modularisierung)
  - …

Softwareentwicklung ist mehr als nur Programmieren!

Portfolio an SE-Techniken

- **Vergleichbar mit einem Werkzeugkoffer:**
  - Für jedes Problem das nützlichste Werkzeug
  - In der Hand eines Experten, der in dessen Umgang geschult ist

  - Aber: nicht jeder muss alle Werkzeuge beherrschen
  - Trotzdem: je mehr Werkzeuge man beherrscht, desto besser