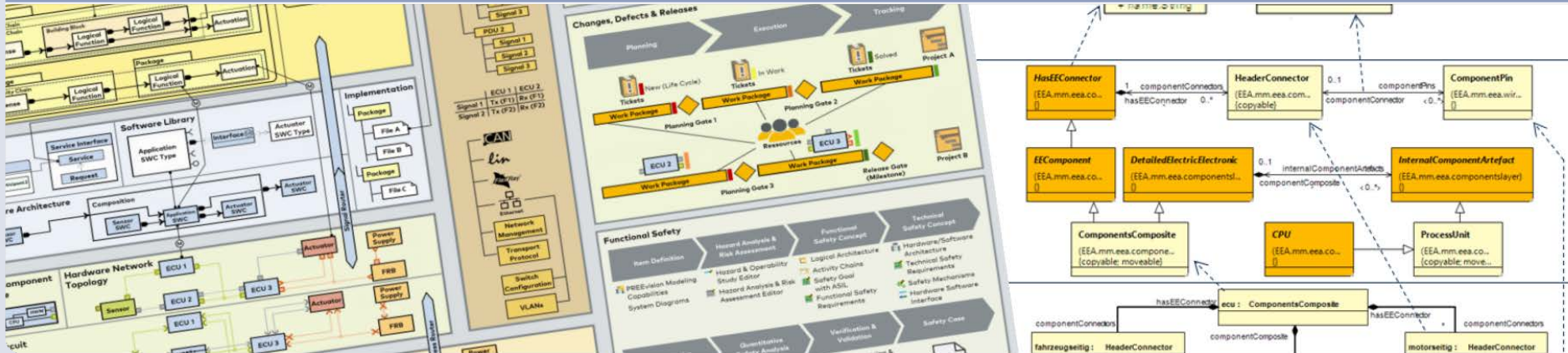


Vorlesung Software Engineering (SE)

Wintersemester 2017/2018

Kapitel 2 – Software Engineering



2. Software-Engineering



2. Software Engineering

2.1 Software

2.2 Software Engineering

2.3 Software Prozess

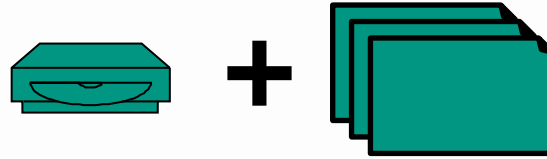
2.4 Software Vorgehensmodell

2.5 Software Methode

2.6 CASE

2.7 Zusammenhänge

- Computerprogramm und zugehörige **Dokumentation**.



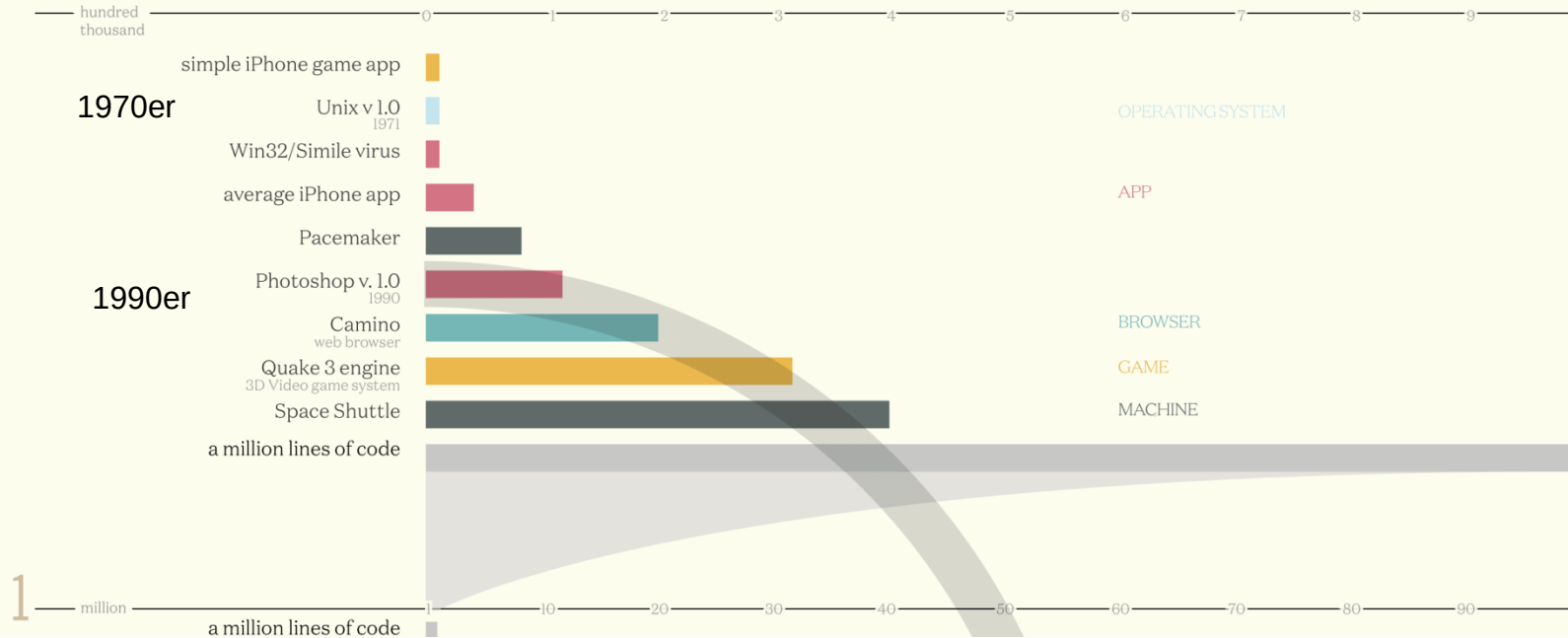
- Softwareprogramme können **für** einen speziellen **Kunden** oder für den allgemeinen **Markt** entwickelt werden.
- Software wird in einer **Programmiersprache** implementiert

Anzahl der Programmzeilen (Programmgröße) (1)

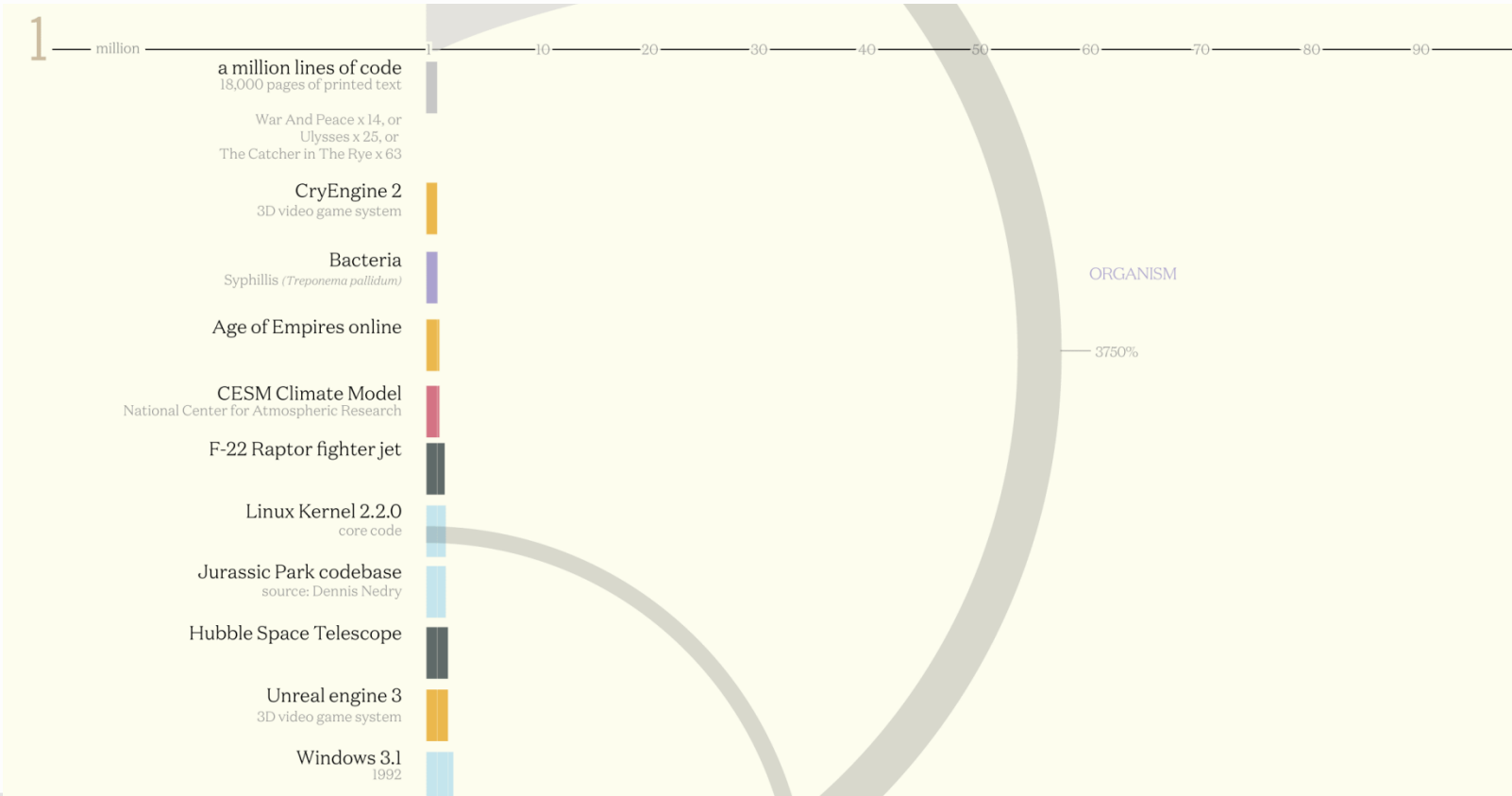
<http://www.informationisbeautiful.net/visualizations/million-lines-of-code/>

Codebases

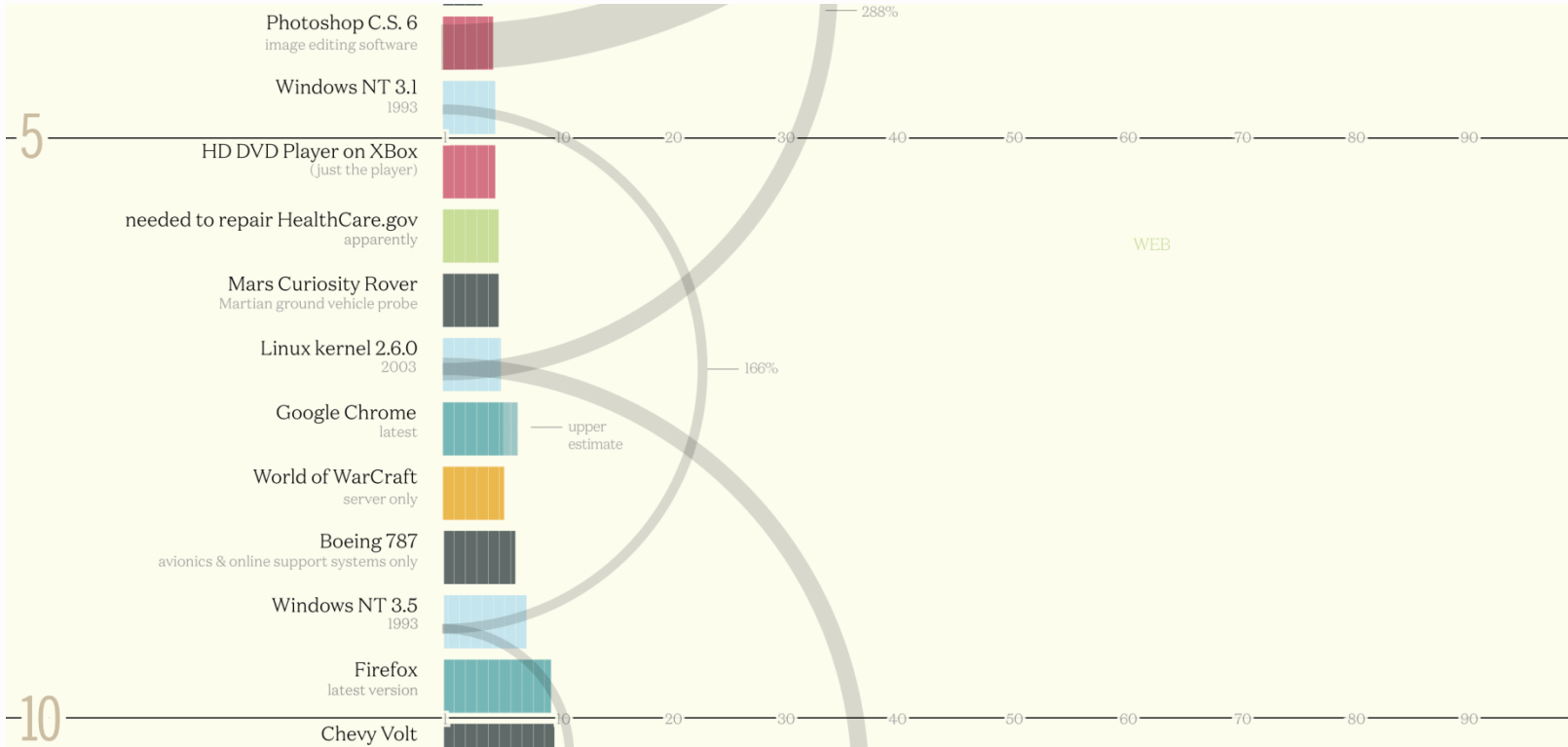
Millions of lines of code



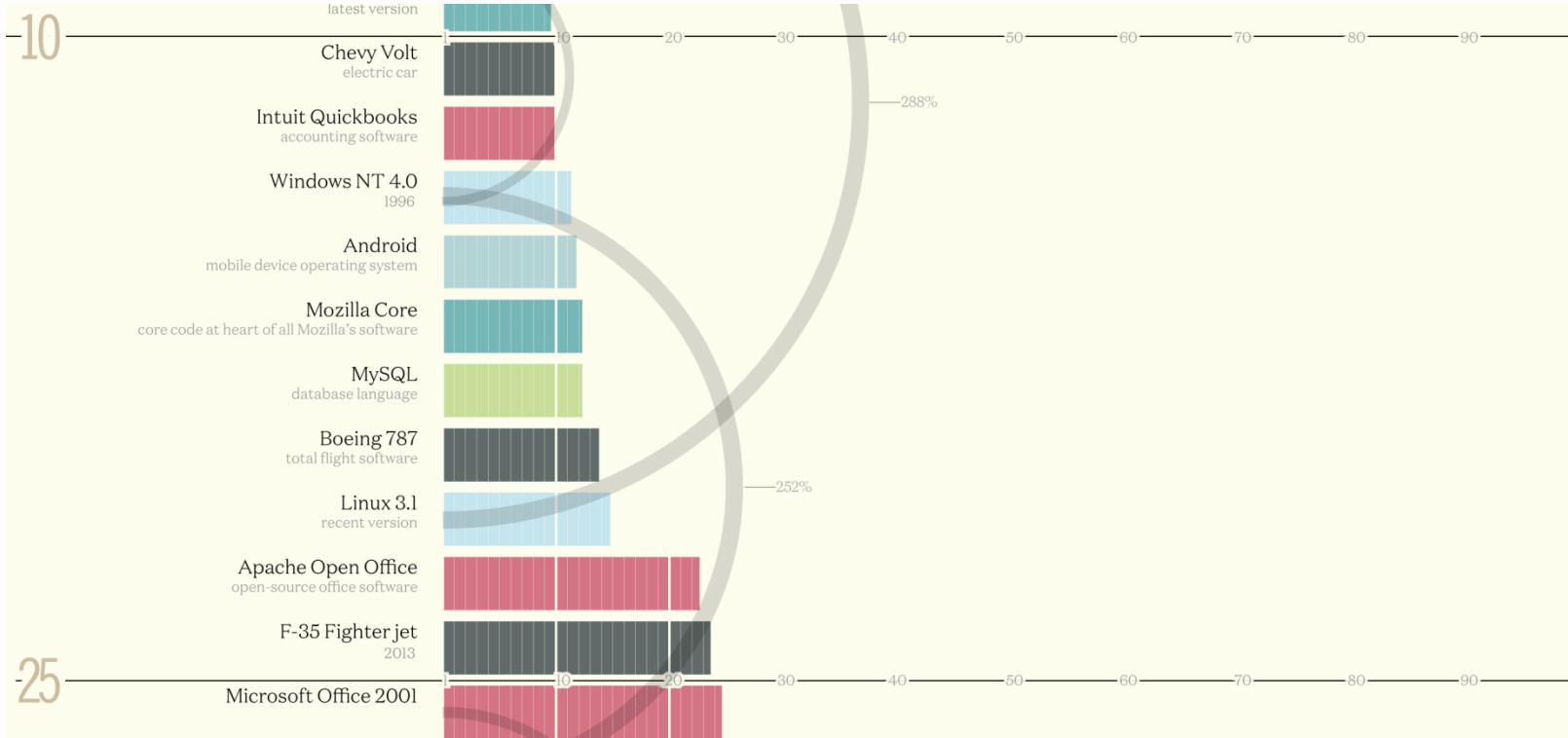
Anzahl der Programmzeilen (Programmgröße) (2)



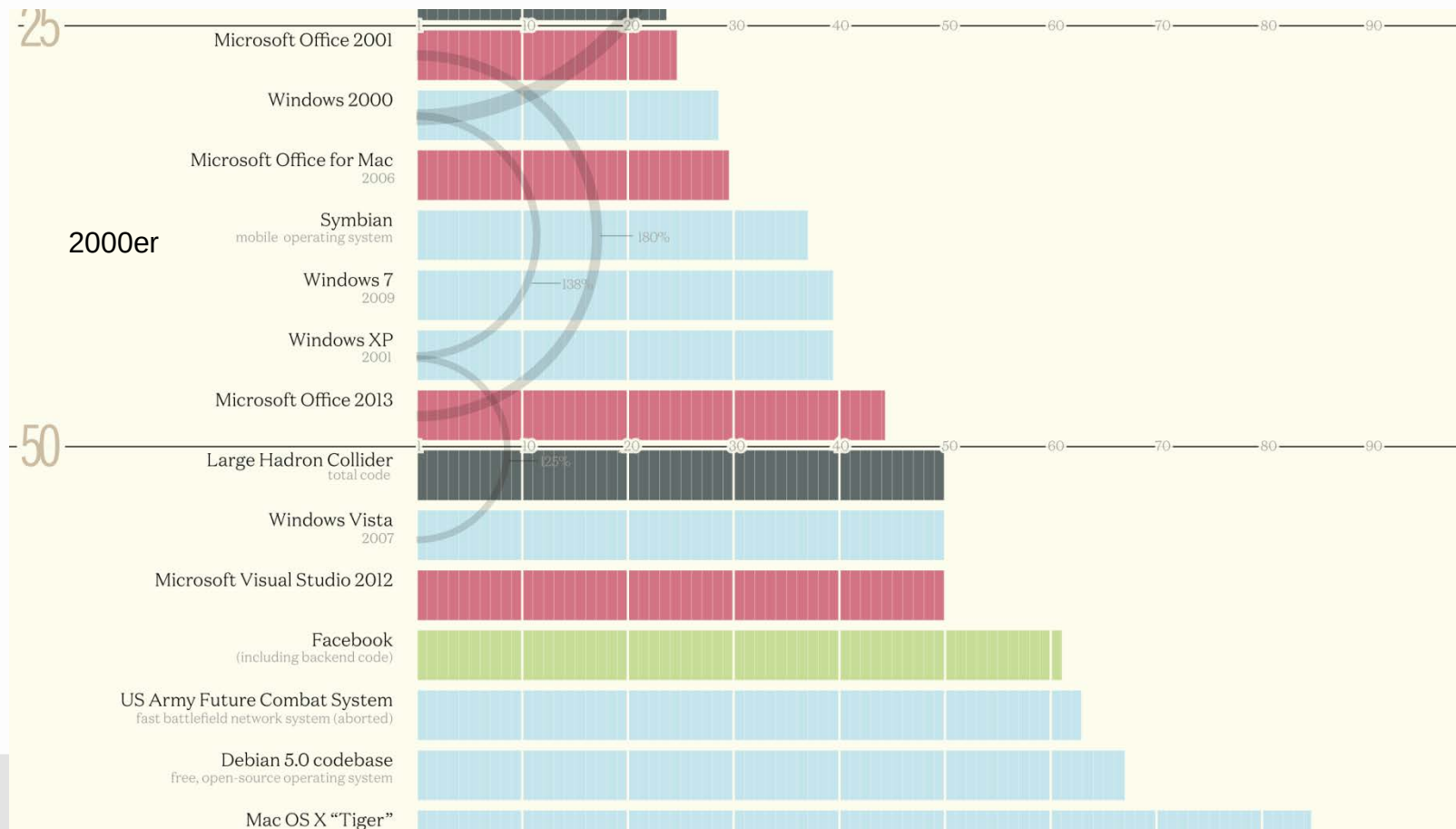
Anzahl der Programmzeilen (Programmgröße) (3)



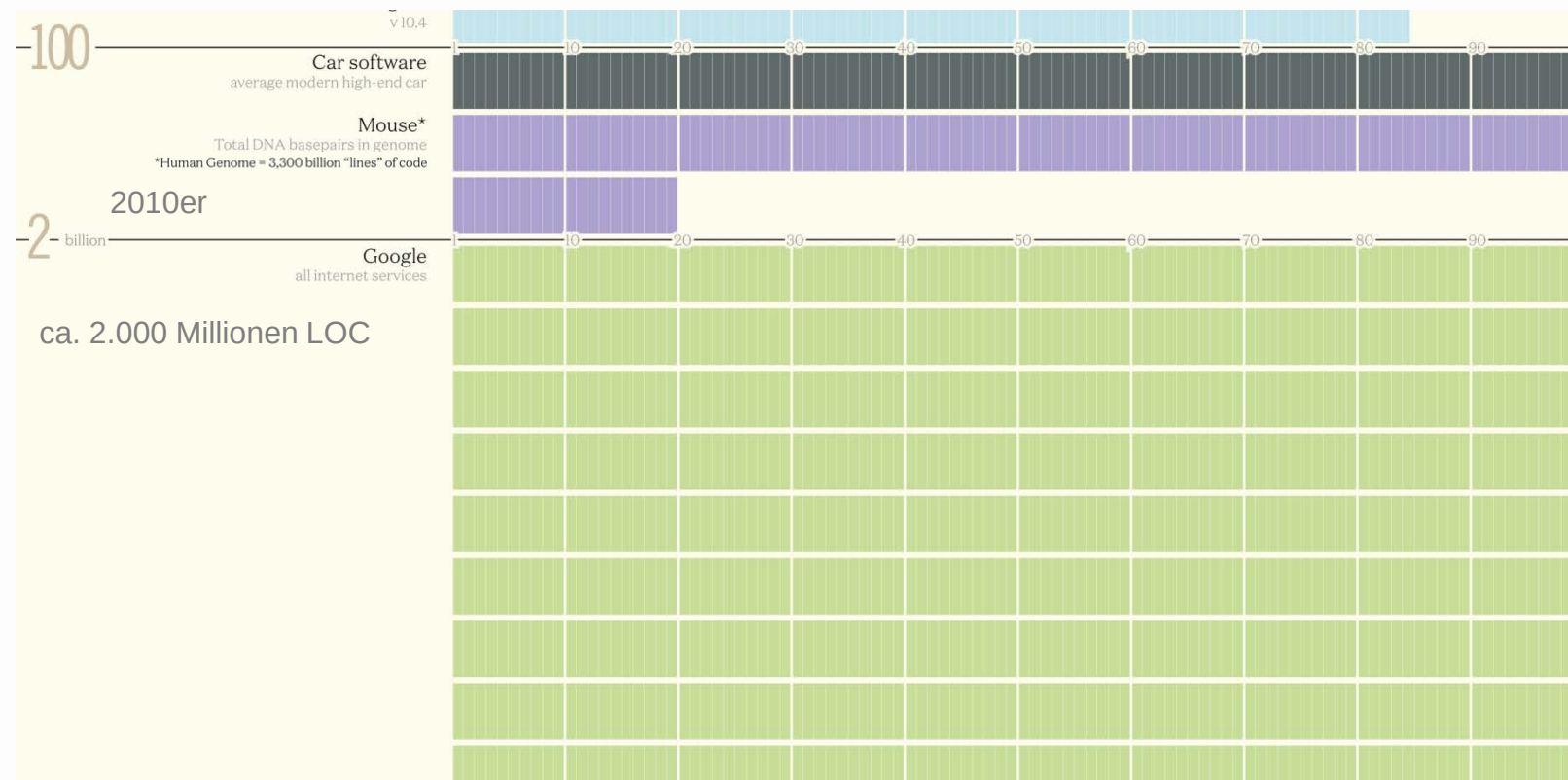
Anzahl der Programmzeilen (Programmgröße) (4)



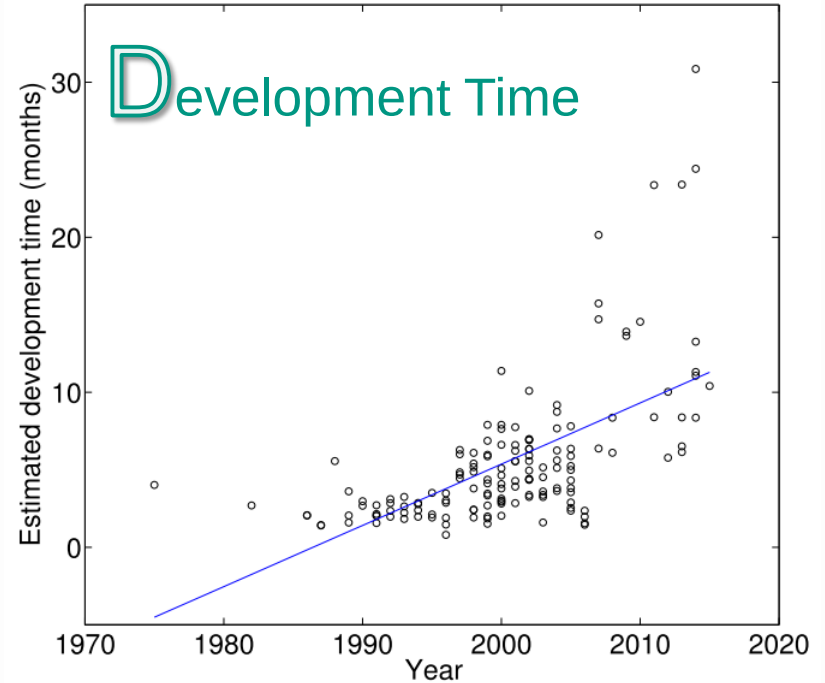
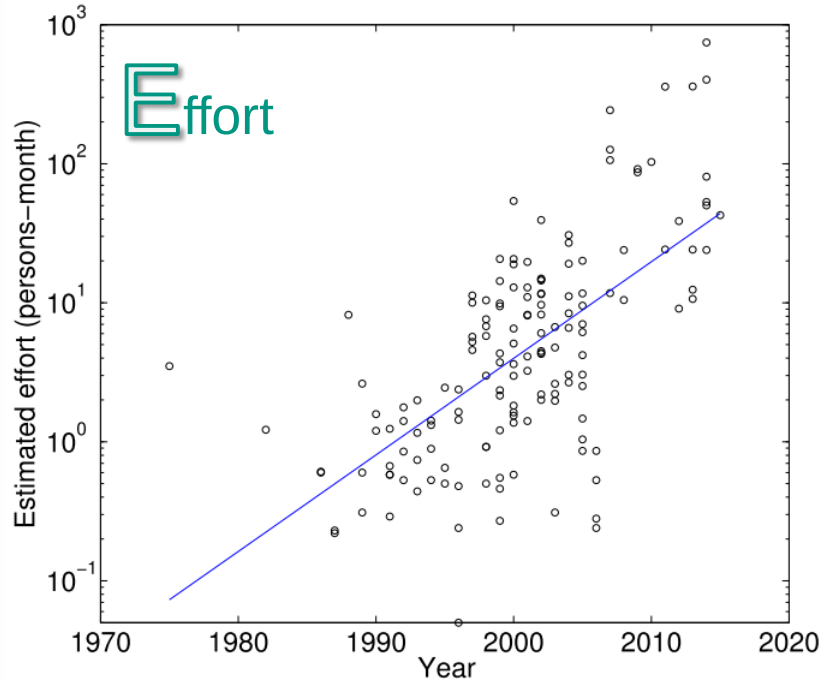
Anzahl der Programmzeilen (Programmgröße) (5)



Anzahl der Programmzeilen (Programmgröße) (6)

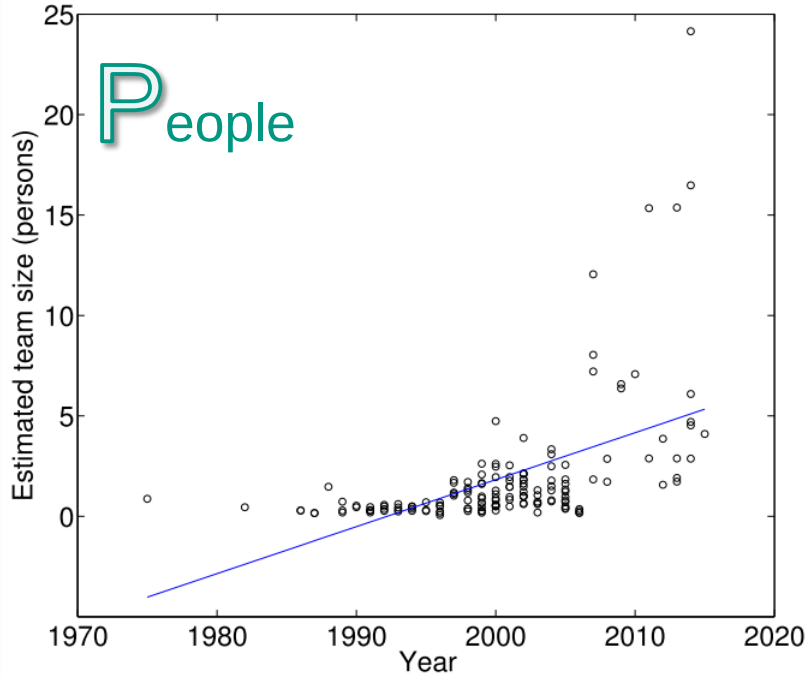


Zunehmende Komplexität (am Beispiel Malware) (I)



Quelle: RAID 2016: http://www.seg.inf.uc3m.es/accortin/RAID_2016.html

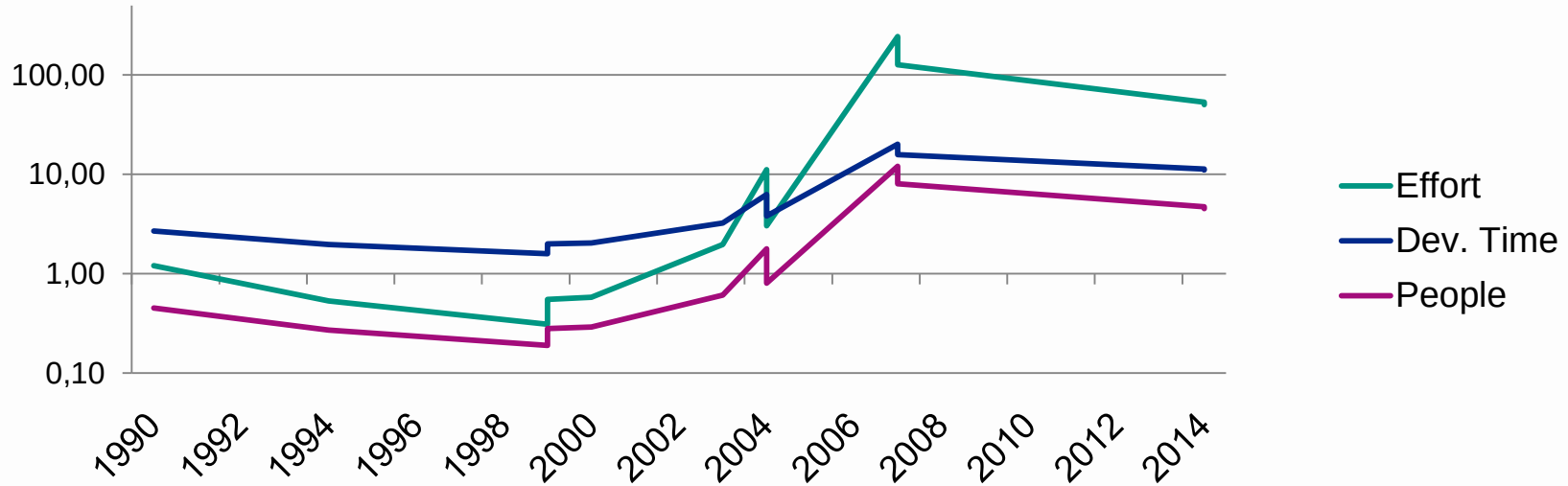
Zunehmende Komplexität (am Beispiel Malware) (II)



Sample	Year	Effort	Dev. Time	People
Anthrax	1990	1.20	2.68	0.45
Batvir	1994	0.53	1.97	0.27
AIDS	1999	0.31	1.59	0.19
liSWorm	1999	0.55	1.99	0.28
ILOVEYOU	2000	0.58	2.03	0.29
Blaster	2003	1.97	3.24	0.61
Mydoorn	2004	11.13	6.25	1.78
Sasser	2004	3.03	3.81	0.80
Zeus	2007	242.85	20.15	12.05
GhostRAT	2007	126.45	15.73	8.04
Tinba	2014	53.13	11.31	4.70
Dendroid	2014	50.20	11.07	4.53

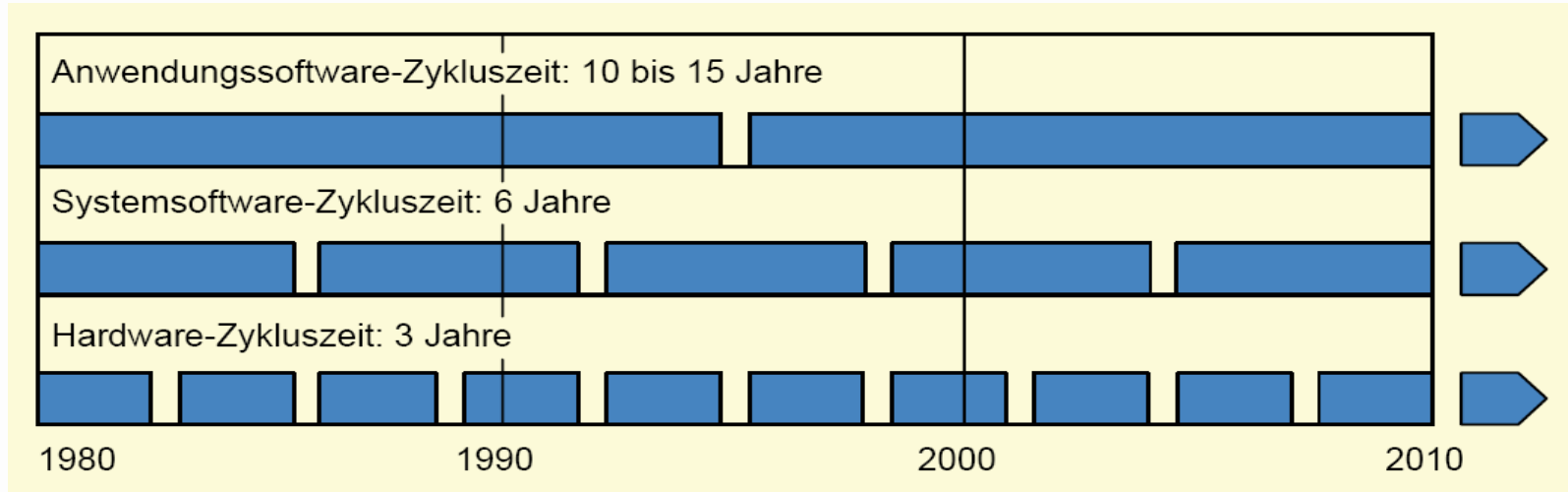
Quelle: RAID 2016: http://www.seg.inf.uc3m.es/accortin/RAID_2016.html

Zunehmende Komplexität (am Beispiel Malware) (III)



Selbst in diesem Software-Bereich zu beobachten:

- Anstieg der Komplexität (Aufwand, Entwicklungszeit, Teamgröße) seit *Jahrzehnten* stetig!



Wartbarkeit

- Weiterentwicklung:
Evolutionäres Vorgehen
- Weiterentwicklung wegen
veränderter
Kundenbedürfnisse

Zuverlässigkeit

- Verlässlichkeit
- Zugriffsschutz
- Betriebssicherheit

Effizienz

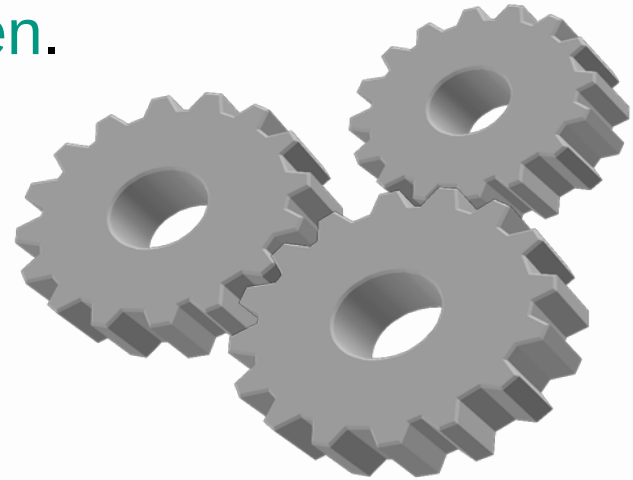
- Ressourcen nicht
verschwenderisch nutzen
 - Speicher (RAM, Festplatte)
 - Prozessorkapazität
(Reaktionszeit,
Verarbeitungszeit)

Benutzerfreundlichkeit

- Nutzbarkeit, Bedienbarkeit
- Dokumentation



Die *Informatik* beschäftigt sich mit der **Theorie** und den **Grundlagen**.



Softwarekrise (I)

Der Begriff der *Softwarekrise* etabliert sich seit den 60ern.
Besonders **spektakuläre Fälle** aus den letzten Jahrzehnten:

1979: **Studie zu Softwareprojekten** (USA), insges. ca. **7 Mio USD**:

75% der Ergebnisse nie eingesetzt;

19% der Ergebnisse stark überarbeitet;

6% benutzbar.

1981: **US Air Force Command & Control** Software überschreitet
Kostenvoranschlag fast um den Faktor 10 (**3,2 Mio USD**).

Softwarekrise (II)

Der Begriff der *Softwarekrise* etabliert sich seit den 60ern.
Besonders **spektakuläre Fälle** aus den letzten Jahrzehnten:

1984: **Überschwemmungskatastrophe** im französischen Tarntal, weil
Steuercomputer nach Überlaufwarnung Schleusen öffnet.

1996: Absturz der **Ariane 5** wegen eines Software-Fehlers.

1997: Entwicklung des **Informationssystems SACSS** für den Staat
Kalifornien abgebrochen.

Aufgelaufene Kosten: **300 Mio USD** (200 % des Voranschlags).

Mehrfach: Steuerungsprobleme beim **Airbus**.



Software Engineering

- ... ist eine technische Disziplin, die sich mit allen **Aspekten der Softwareherstellung** befasst.
- ... ist Teil der **Systementwicklung**
- Der Begriff wurde 1968 auf einer Konferenz vorgeschlagen.
 - Anlass: SW Krise durch **Komplexität**; d.h. man konnte keine SW entwickeln in einer Komplexität, welche vorhandene HW ausreizte.

Definition (I)

» *Software Engineering* ist die Entdeckung und Anwendung solider **Ingenieur-Prinzipien** mit dem Ziel, auf **wirtschaftliche Weise** Software zu bekommen, die **zuverlässig** ist und auf realen Rechnern läuft.

F.L. Bauer, NATO-Konferenz Software-Engineering 1968

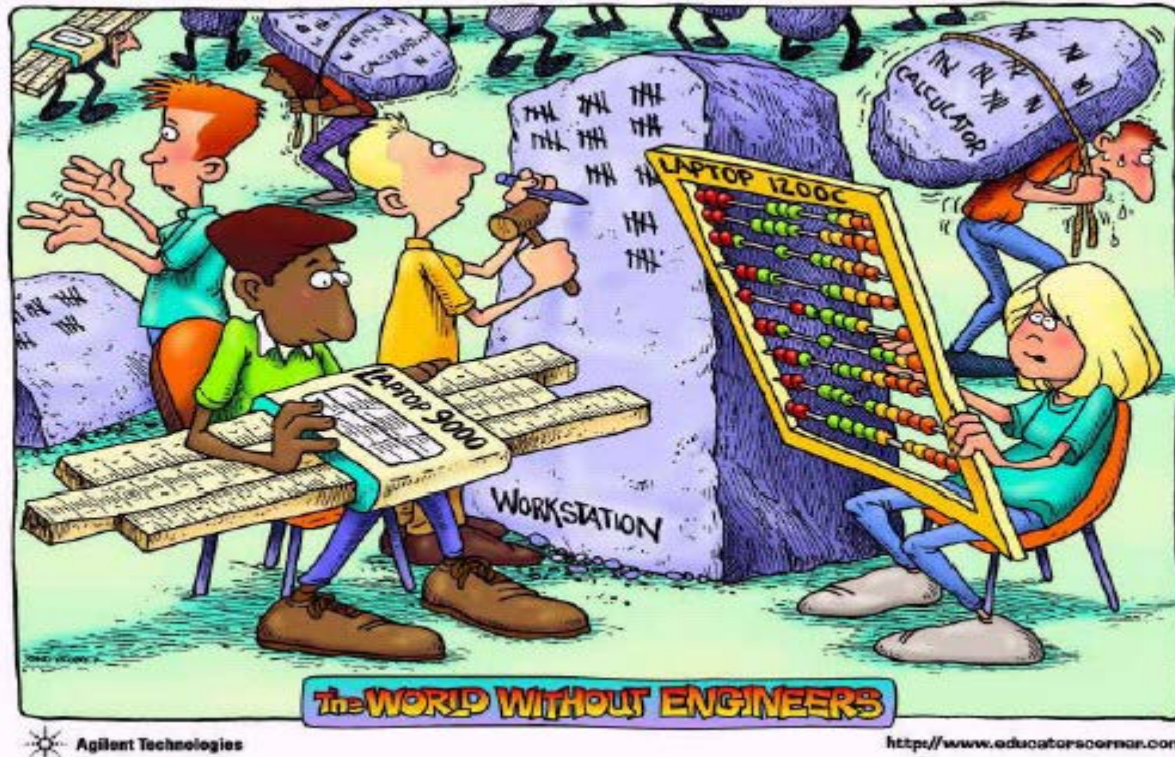
Teilbereiche

Spezifikation

Entwicklung

Management

Weiterentwicklung



» *Software Engineering* ist die Herstellung einer Software, wobei **mehrere Personen** beteiligt sind und/oder **mehrere Versionen** entstehen.

–D.L. Parnas

Ursache für Komplexität

» *Software Engineering* is the application of a **systematic**, disciplined, quantifiable **approach** to the development, **operation**, and maintenance **of software**; that is, the application of engineering to software [...]

IEEE Standard 610.12

systematisches Vorgehen!



„Ingenieur-typisches“ Vorgehen (I)

a) bei der Produktentwicklung

- **systematisch** nach bekannten **Methoden** vorgehen (s.u.):
 - Entwicklungsprozesse für **bekannte Situationen**
 - Entwicklungsprozesse für **Neuentwicklungen**
- dabei Einsatz geeigneter **Werkzeuge**
- **Zerlegung** komplexer Gesamtsysteme in handhabbare Subsysteme und **Komponenten**
 - Komponenten: abgeschlossene Bausteine mit **fester Funktionalität**, die sich zusammensetzen lassen
 - **normierte** Komponenten: leichter **(wieder-)verwendbar**
man denke z.B. an Schrauben, IC, Stecker, Dichtungsringe, ...

„Ingenieur-typisches“ Vorgehen (II)

b) bei der Gestaltung des Entwicklungsprozesses

- strukturiert in **Phasen**
 - erlaubt Spezialisierung und Arbeitsteilung
 - definierte Zwischenergebnisse („Meilensteine“)
 - wiederholbare Abläufe
- **Prozess** begleitende **Qualitätssicherung**
 - **Qualität**

» Qualität

lat. *qualitas*: Beschaffenheit, Eigenschaft, Zustand

Qualität ist der **Grad**, in dem ein Satz inhärenter Merkmale **Anforderungen erfüllt**.

— **DIN EN ISO 9000:2000**

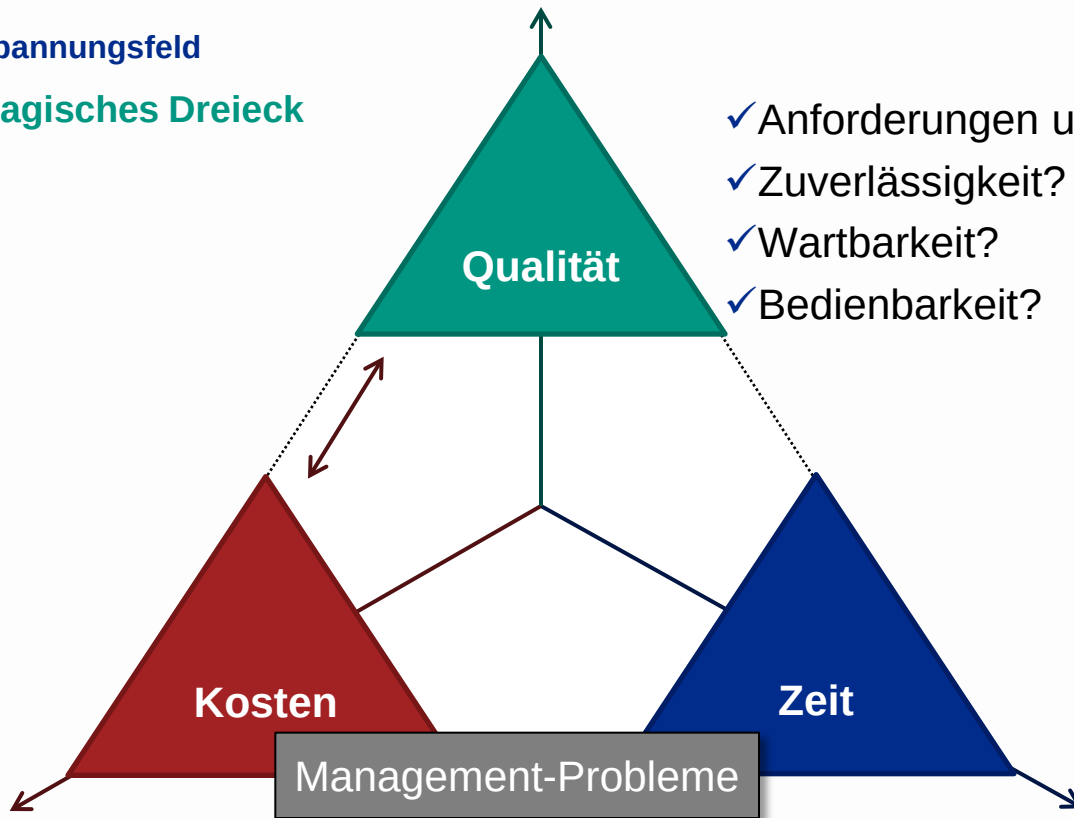
→ *Qualität* gibt an, in welchem **Maße** ein Produkt (Ware oder Dienstleistung) den bestehenden **Anforderungen entspricht**.

„Ingenieur-typisches“ Vorgehen (IV)

b) bei der Gestaltung des Entwicklungsprozesses

- strukturiert in **Phasen**
 - erlaubt Spezialisierung und Arbeitsteilung
 - definierte Zwischenergebnisse („Meilensteine“)
 - wiederholbare Abläufe
- **Prozess** begleitende **Qualitätssicherung**
 - **Qualität**
 - der **Erstellungsprozess** als Gegenstand der **Optimierung**
 - Einhalten von **Standards** wie **ISO 9000**, **Software Process Improvement and Capability Determination (SPICE)**, Capability Maturity Model Integration (**CMMI**)

Insgesamt Hauptaugenmerk auf **Qualität!**



✓ Anforderungen umgesetzt?

✓ Zuverlässigkeit?

✓ Wartbarkeit?

✓ Bedienbarkeit?

Informatik-Probleme

Ergonomie-Probleme

Management-Probleme

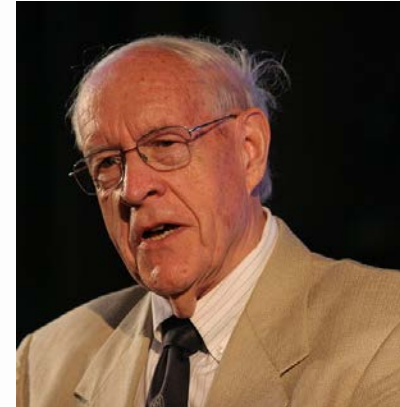
Der **Kunde** soll zurück kommen – nicht das Produkt!

- **Abgrenzung** von möglichst unabhängigen Teilaufgaben
- **Planung** und Verfolgung des Ablaufs
- **Koordination** durch einheitliche Begriffe und Notationen
- **Modulare Konstruktion** für parallele Entwicklungsarbeit
- Nutzung vorhandener Software: **Baukastensysteme**
- Technische und organisatorische **Infrastruktur** für Teamarbeit



» Adding manpower to a late software project makes it later.

Brooks's Law (1975)



Fred Brooks. by [David Monniaux](#),
licensed under [CC-BY-SA-3.0](#)

Teamarbeit erfordert zusätzlichen Aufwand

- Rollen im Team festlegen
- Kommunikation innen/außen

Management-Probleme

Aufteilung in unabhängige Teile

- zu erstellendes System
- Erstellungsprozess

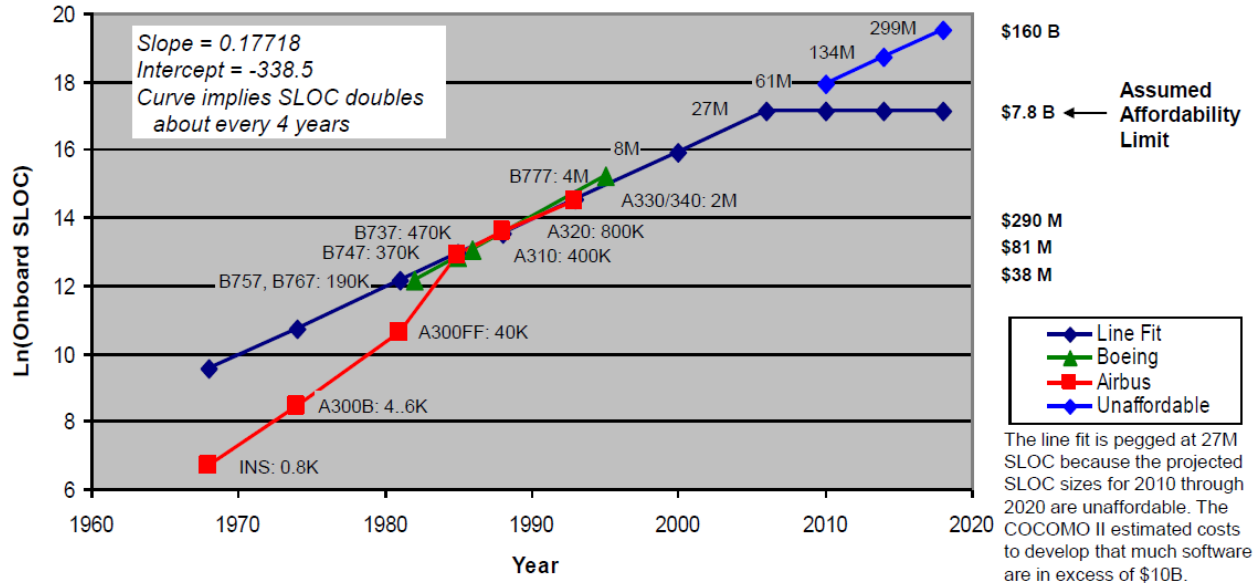
Informatik-Probleme

Verschiebung vom
programmierenden Benutzer
zum **fachfremden Programmierer**

Ergonomie-Probleme

Bedienbarkeit
usability

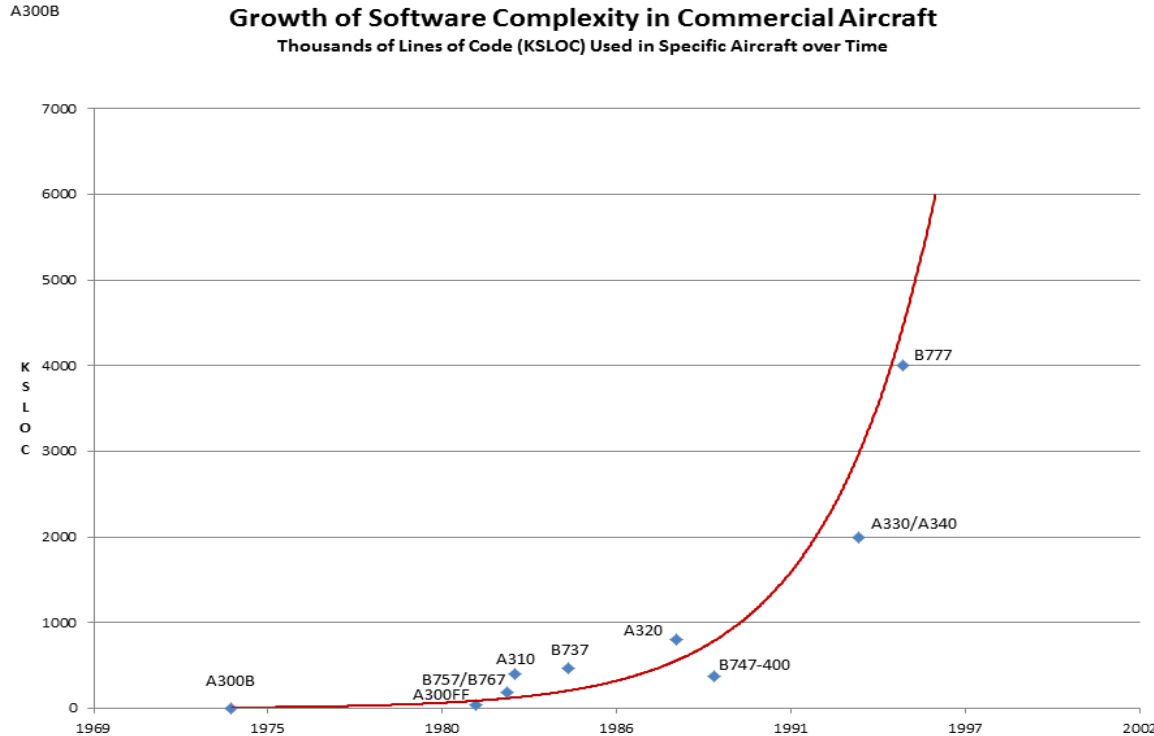
Growth of Software Lines of Code



Airbus data source: J.P. Potocki De Montalk, Computer Software in Civil Aircraft, Sixth Annual Conference on Computer Assurance (COMPASS '91), Gaithersburg, MD, June 24-27, 1991.

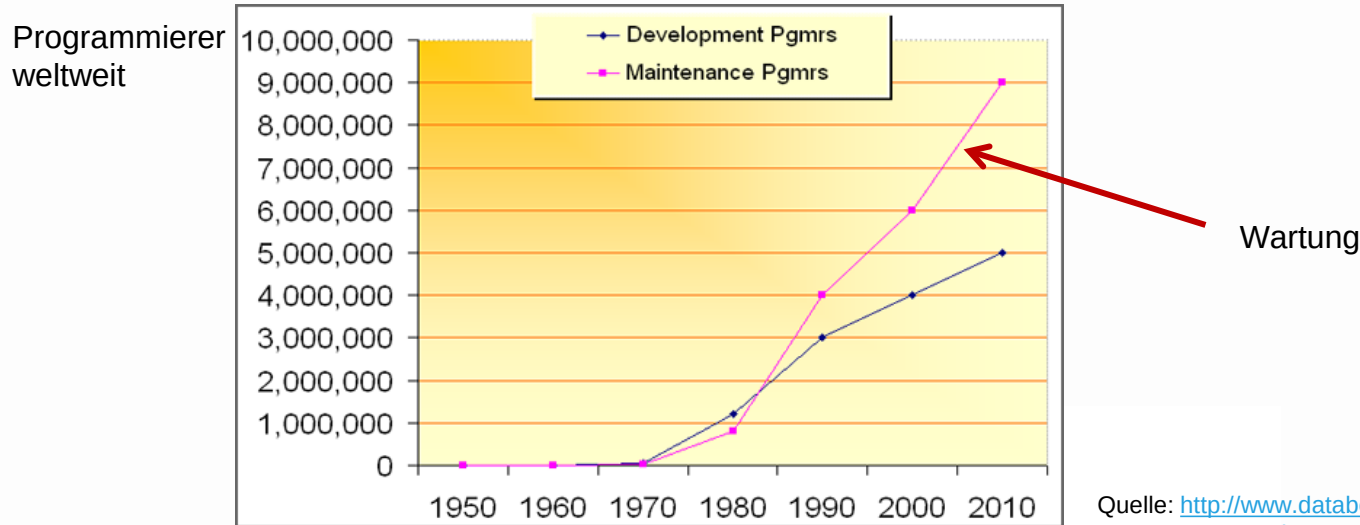
Boeing data source: John J. Chilenski. 2009. Private email.

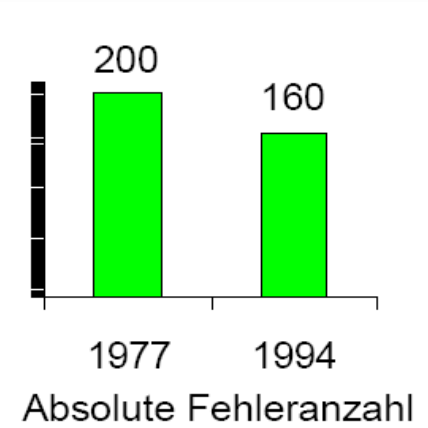
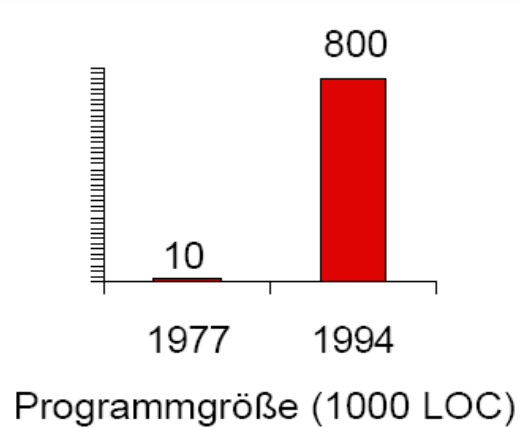
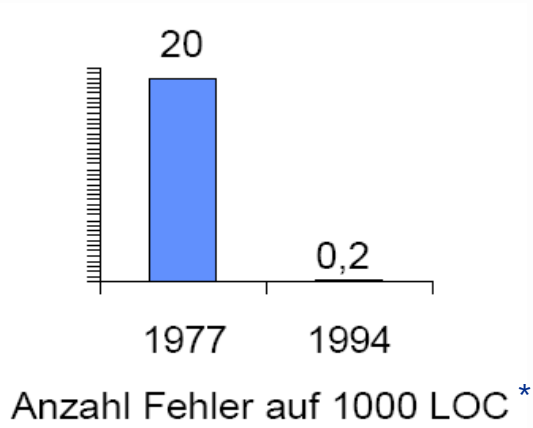
Quelle: <http://www.engineeringnewworld.com/?paged=2>



» As a system evolves, its complexity increases unless steps are taken to reduce it.

— Meir Lehman (Laws of Software Evolution)





Echte Qualitätsverbesserungen sind nur möglich, wenn man die Steigerung der **Programmkomplexität überkompensiert!**

LOC: Lines of Code

Für **50% der Ausfälle** im industriellen Sektor sind **Software-Fehler** verantwortlich

Nach **Cusumano** haben sich die gefundenen **Defekte in jeweils 1000 Zeilen Quellcode** folgendermaßen entwickelt:

1977: durchschnittlich 7- 20 Defekte

1994: durchschnittlich 0,2 - 0,05 Defekte

In **13 Jahren** konnte die **Defektrate** also ungefähr um das **100fache** **gesenkt** werden.

0,1%-Defektniveau bedeutet...

pro Jahr

- 20.000 fehlerhafte Medikamente
- 300 versagende
Herzschrillmacher

pro Woche

- 500 Fehler bei medizinischen
Operationen

pro Tag

- 16.000 verlorene Briefe
in der Post
- 18 Flugzeugabstürze

pro Stunde

- 22.000 Schecks nicht korrekt
gebucht

Hängt von Prozess und Software ab:

- Verteilung der **Entwicklungskosten** (allgemein)



Test:
ca. 40-50%

- Kosten bei **evolutionärer Entwicklung**



- Kosten der Weiterentwicklung (ca. 25% Systementwicklung)



- Kosten der **Produktentwicklung**



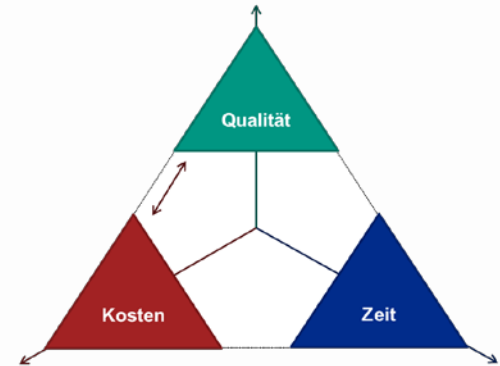
- Kostenschätzung: **COCOMO** (siehe Vorlesung SSE)

Qualität

Zeit



Kosten



Definition



Der *Softwareprozess* ist ein **Satz von Tätigkeiten**, die zusammenhängende Ergebnisse erzeugen, mit dem **Ziel** eines **Softwareprodukts**.

Tätigkeiten werden von **Softwareentwicklern** durchgeführt

Softwarespezifikation

- **Definition** der Funktion der SW und Randbedingungen für Einsatz

Softwareentwicklung

- Erstellen von SW, die **Spezifikation erfüllt**

Softwarevalidierung

- Validieren der SW um **sicherzustellen**, dass sie tut was der Kunde will

Softwareevolution

- **Weiterentwicklung** der SW

Definition



Ein *Vorgehensmodell* ist eine **Vereinfachte Beschreibung** eines Softwareprozesses (aus einer bestimmten Perspektive).

Zweck

Hilfe bei Erstellung von SW

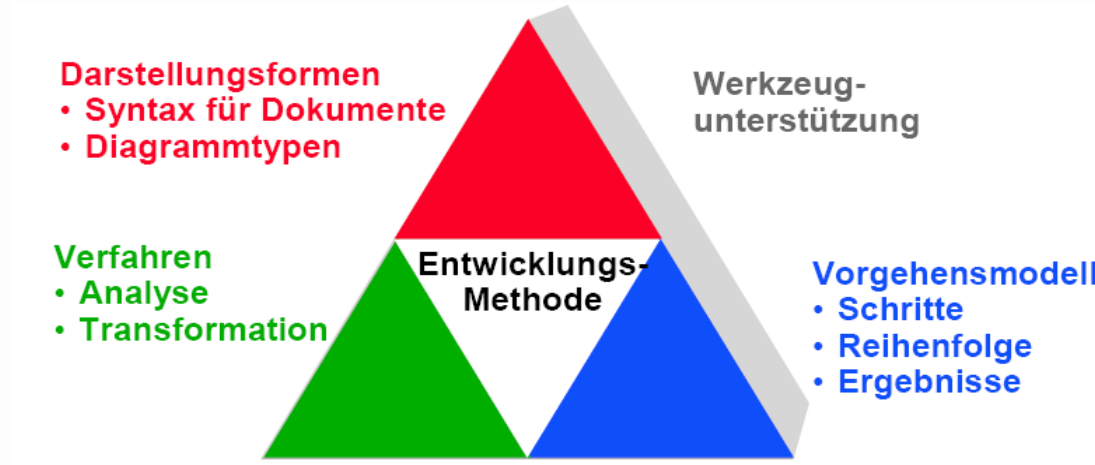
- **Bessere Planbarkeit** der Entwicklung
- **Bessere Struktur** des Produkts

Ziel

- **Abstraktion** des tatsächlichen Prozesses (Modell)
- Beschreibung von **Tätigkeiten und Rollen** (Personen)



Ein Vorgehensmodell ist Teil einer Entwicklungsmethode.



Ein Vorgehensmodell ist Teil einer Entwicklungsmethode.

Arbeitsablaufmodell

- Abfolge von Aktivitäten, Eingabe, Ausgabe, Abhängigkeiten

Rolle/ Aktions-Modell

- Rolle des Menschen im Prozess, Verantwortlichkeiten der Rollen

Datenfluss- oder Aktivitätsmodell

- Menge von Aktivitäten mit Datenumwandlung je Aktivität

Wasserfall-Modell

- Schwerpunkt auf Phasen

V-Modell

- Teilung: Entwurf und Test
- Große Projekte (SW+HW)

Spiralmodell

- Fokus auf Risiken

Evolutionäre Entwicklung

- Fokus auf schnelle Änderungen
- Agile Methoden (XP, Scrum, Kanban, usw.)

Prototypen

- Entwicklung ist durch Prototypen getrieben
- Prototyp != Releasegegenstand

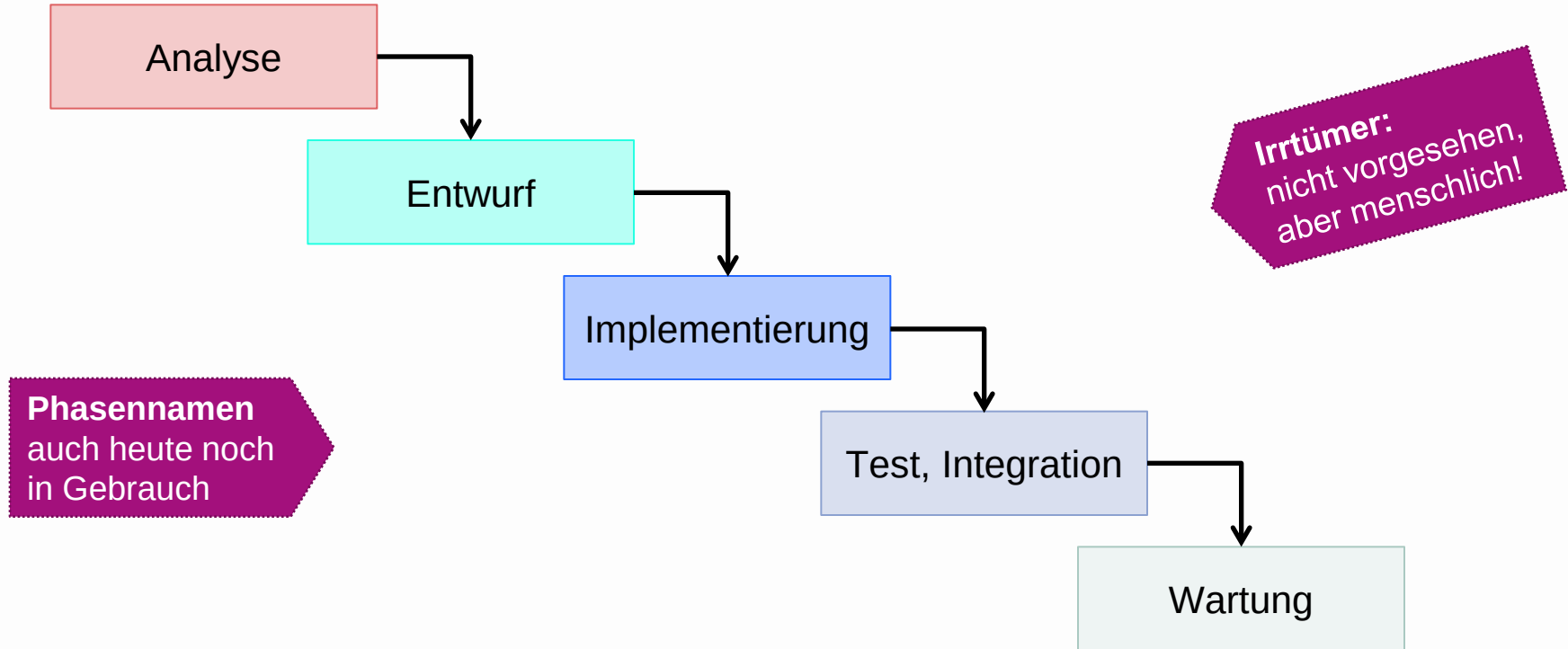
Formale Umsetzung

- Fokus auf formaler Beschreibung

Zusammensetzung aus wieder verwendbaren Komponenten

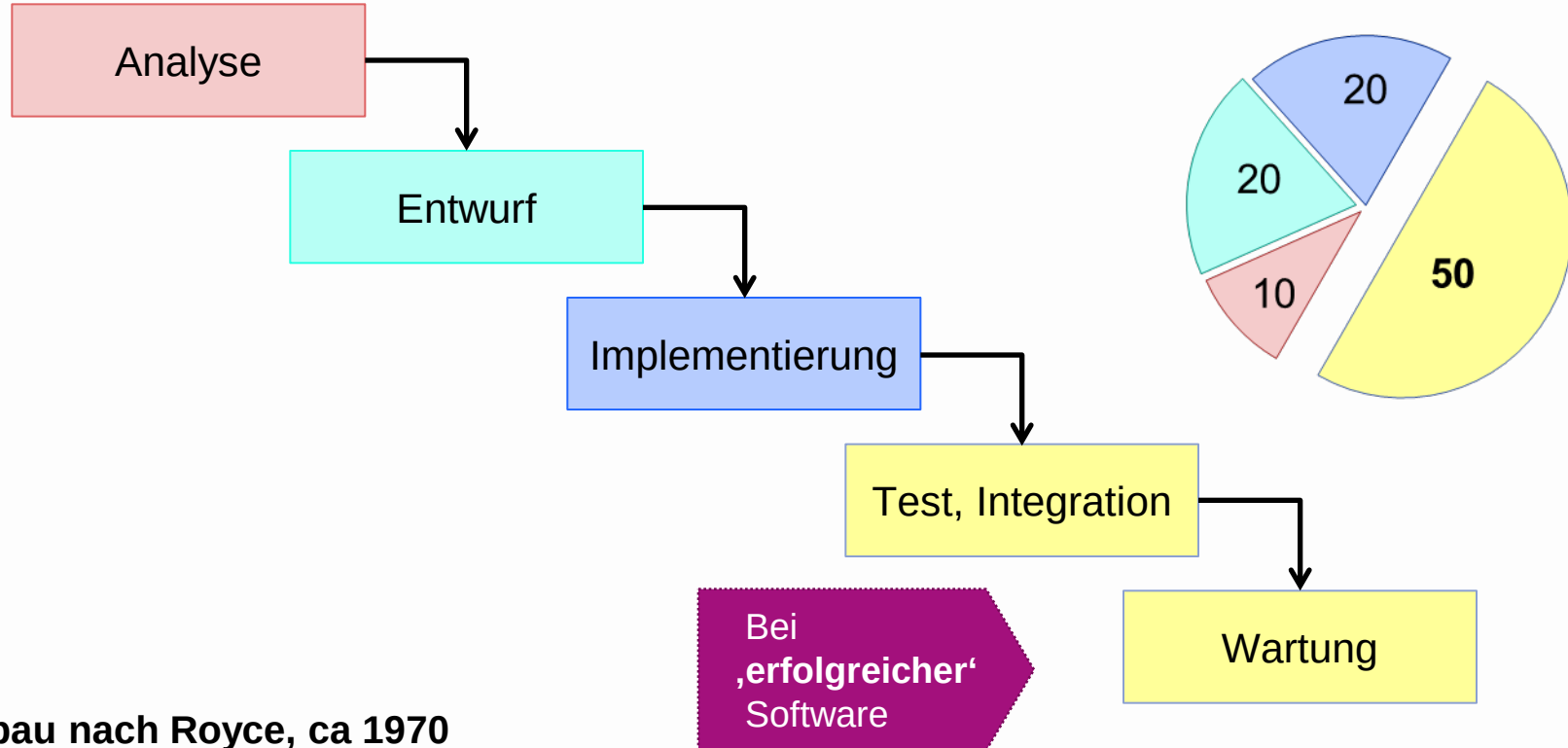
- Fokus: Baukastensystem

Wasserfallmodell



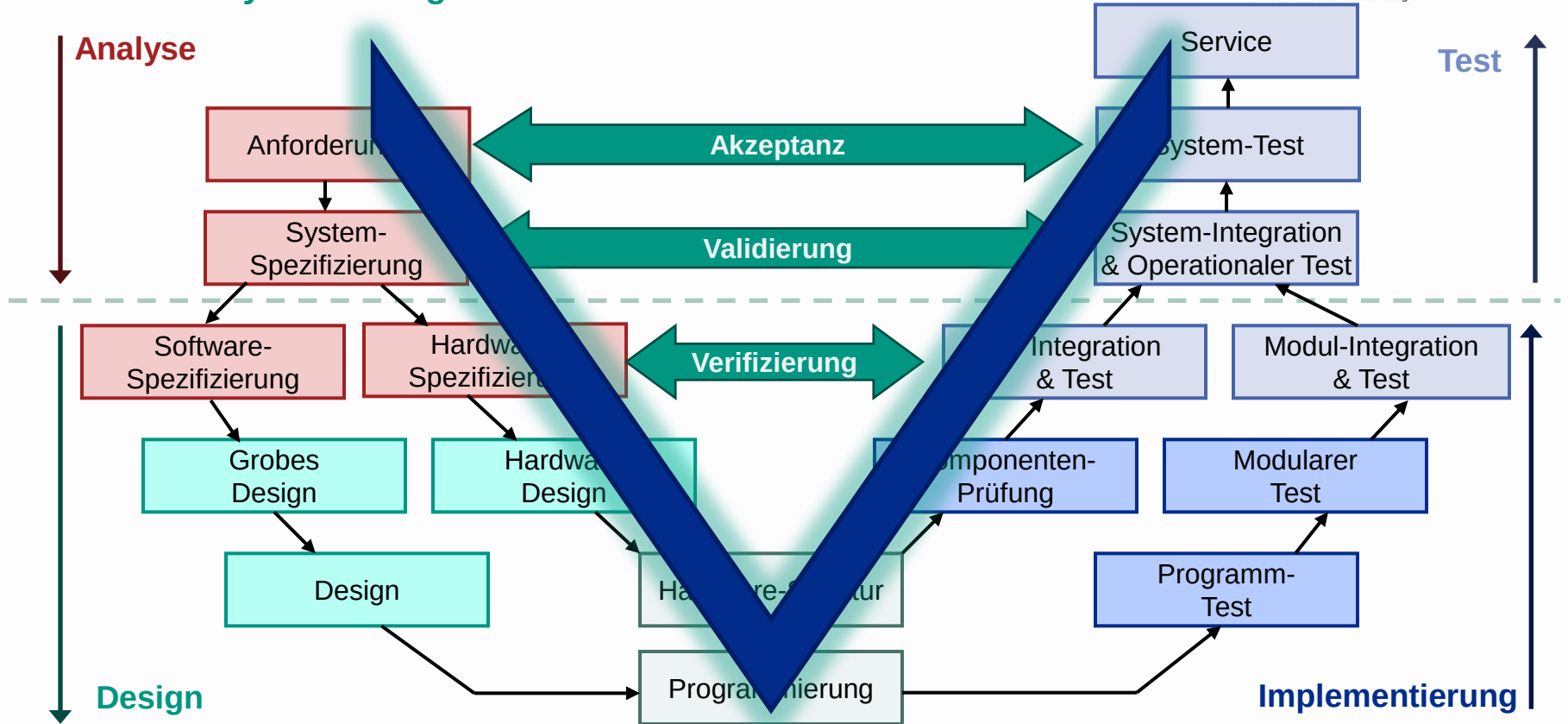
Aufbau nach Royce, ca 1970

Wasserfallmodell

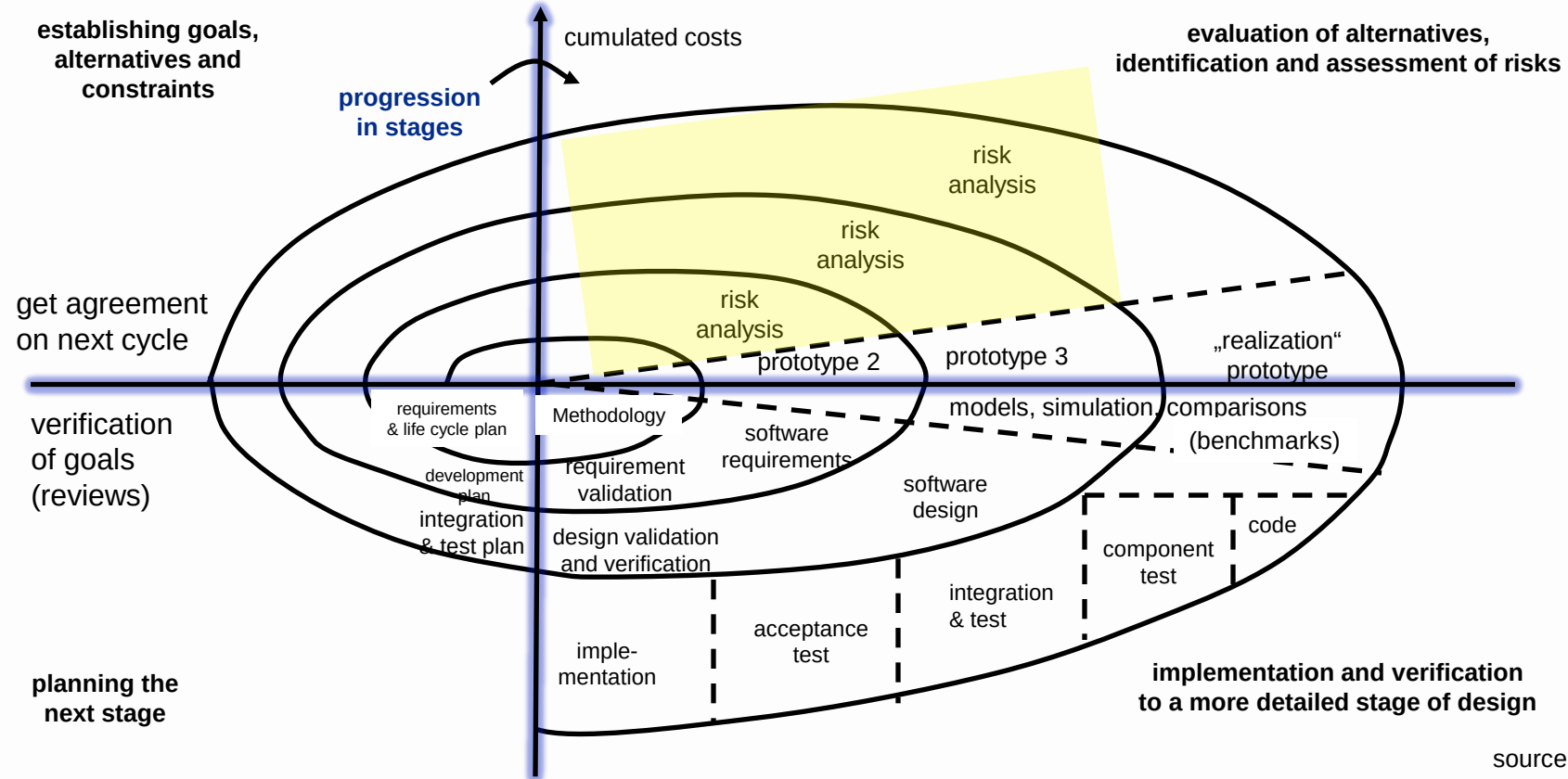


Aufbau nach Royce, ca 1970

V-Modell – System Design Prozess



Spiral model of the software life cycle



source: [Boe84]

Wesentliche ‚Zutaten‘

1) Minimale ‚Dokumentation‘

- Story-Board (**Use-Cases**)
- Testfälle und Programmcode

2) Keine vorausschauende Planung

- Nur **sofort Notwendiges** im Entwurf berücksichtigt
YAGNI: „*You Aren't Gonna Need It*“
- ggfs Umbau des Programmcodes (**Refactoring**)
- **Vorgabe der Prioritäten** durch Nutzer (welche Story?)
- **kurze Planungszyklen** (wenige Wochen)

YAGNI

Wesentliche ‚Zutaten‘

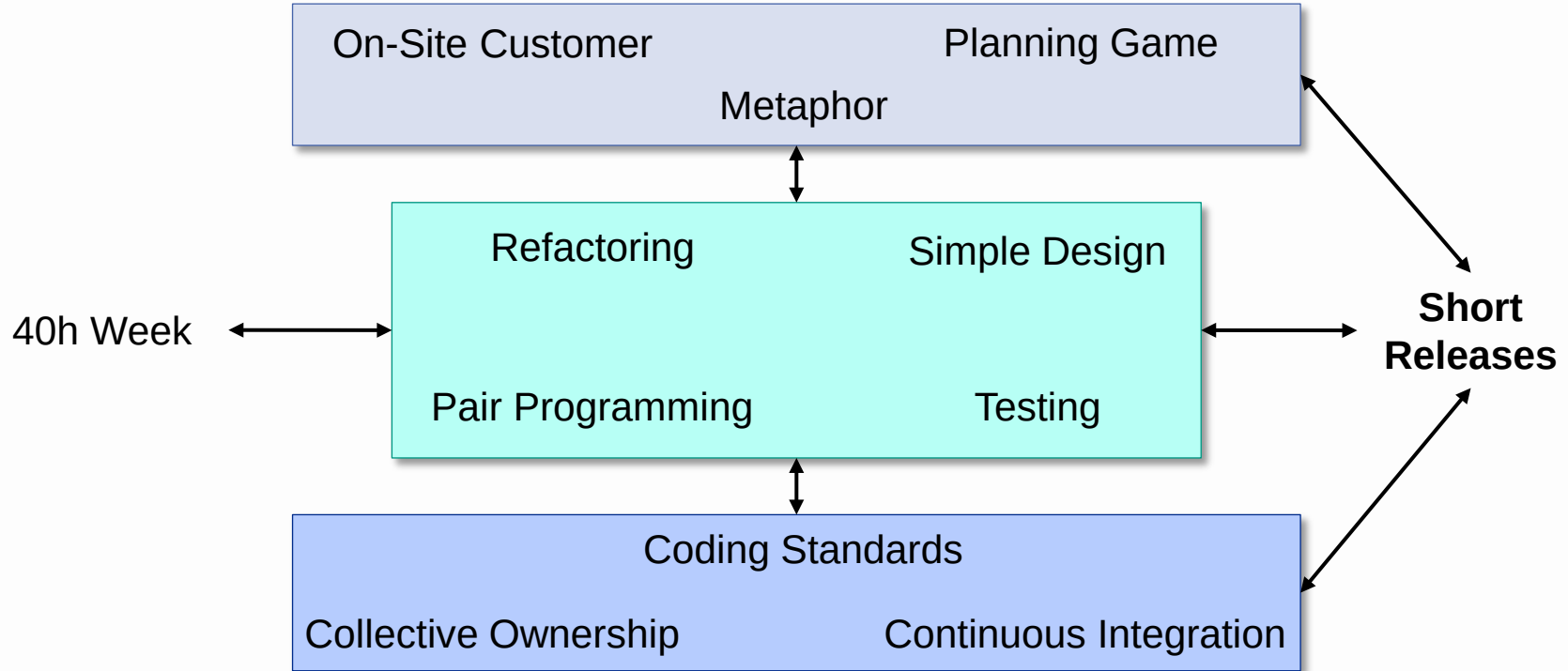
3) Techniken

- ‚Pair Programming‘
- nach jeder Ergänzung alle Testfälle ausführen
- fortlaufende Integration (Continuous Integration)
→ stets ausführbarer Prototyp
- So einfach wie möglich
„*Keep It Simple and Stupid*“



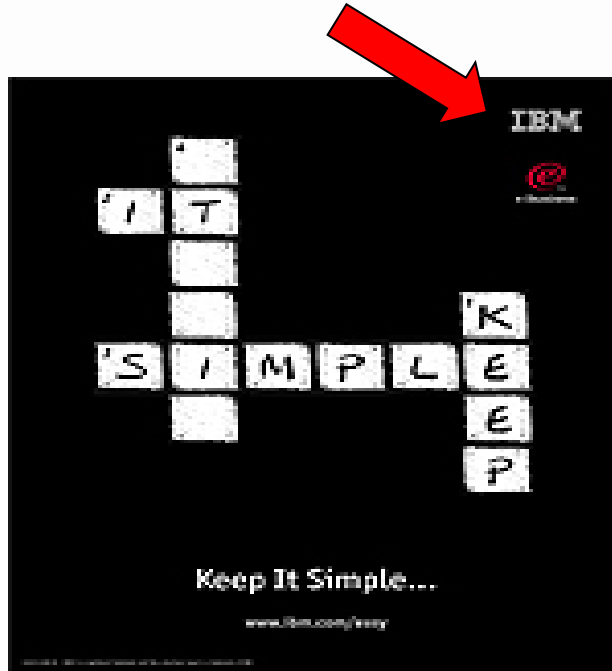
KISS

Extreme Programming [XP] (II)



<http://www.extremeprogramming.org/>

10. Nov. 2003



http://www-3.ibm.com/ibm/easy/eou_ext.nsf/Publish/650

Scrum (Begriff)

Herkunft

Scrum im Rugby (dt: »Gedränge«)

Ball befreien durch **eng verbundene**
Bewegung in **selbstorganisiertem** Team

- Metapher für Vorgehen
- Ist eine Form der Agilen Softwareentwicklung



Scrum to England by Eoln Gardiner

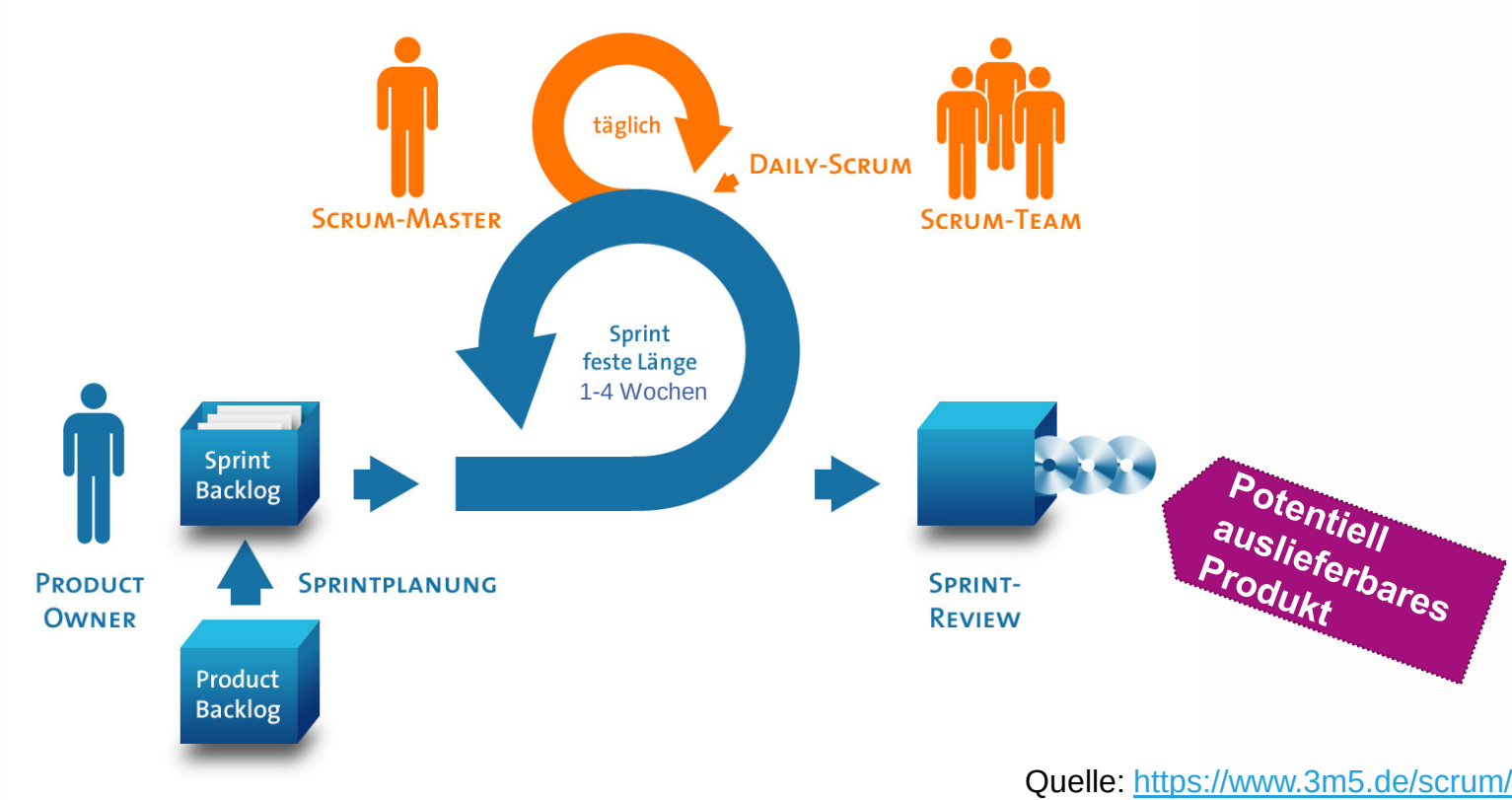
<https://www.flickr.com/photos/18091975@N00/3654141771/>

Wir erschließen bessere Wege, Software zu entwickeln, indem wir es selbst tun und anderen dabei helfen. Durch diese Tätigkeit haben wir diese Werte zu schätzen gelernt:

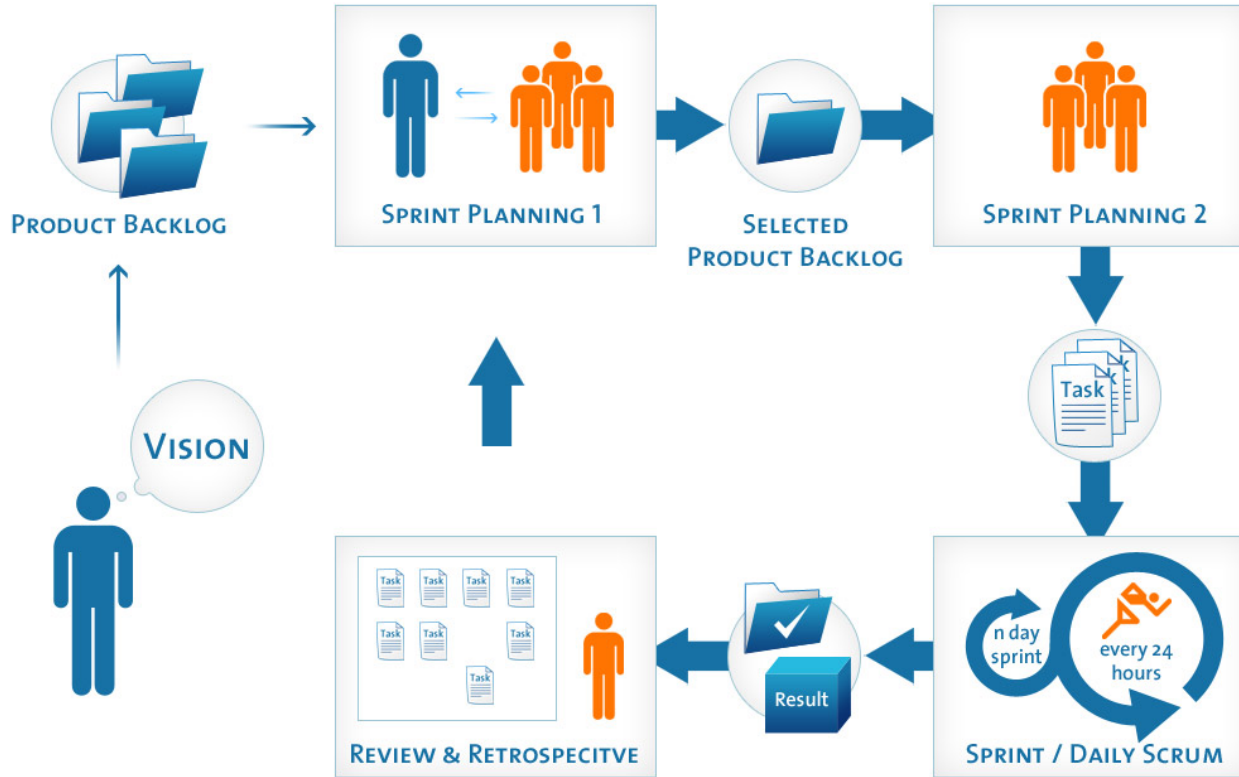
- **Individuen und Interaktionen** stehen über Prozessen und Werkzeugen
- **Funktionierende Software** steht über einer umfassenden Dokumentation
- **Zusammenarbeit mit dem Kunden** steht über der Vertragsverhandlung
- **Reagieren auf Veränderung** steht über dem Befolgen eines Plans

Das heißt, obwohl wir die Werte auf der rechten Seite wichtig finden, schätzen wir die Werte auf der linken Seite höher ein.





Quelle: <https://www.3m5.de/scrum/>



Quelle: <https://www.3m5.de/scrum/>

Scrum (Rollen)



Product Owner



PRODUCT
OWNER

- Repräsentiert Kunden mit Anforderungen (Konzeption, Vision)
- Verwalter und Bearbeiter des Product Backlog und Release-Plans
- Entscheidungsgewalt (was kommt als nächstes).

Entwicklungsteam



SCRUM-TEAM

- Entwickelt Produkt innerhalb festgelegter Anzahl von Sprints
- Agiert autonom gemäß festgelegten Standards und Prozessen

Scrum Master



SCRUM-MASTER

- Vermittler/moderiert zwischen PO und Team (mit Product Backlog)
- Keine Befehlsgewalt, sondern schult und führt (Hilfe bei Problemlösung)
- Passt Standards und Prozesse an.

Scrum (Artefakte)

Product Backlog

- Beinhaltet Roadmap: Liste aller *User Stories* nach Priorität und Gewichtung
- Product Owner ist für Liste und deren Anpassung verantwortlich

User Story

Pseudo-Anforderung: **grobe Beschreibung** der Eigenschaften und Produktmerkmale. (Sollte sog. INVEST-Merkmale aufweisen.)

Sprint Backlog

- User Stories, die im Sprint umgesetzt werden sollen
- Zeigt, welche Akteure gerade womit beschäftigt sind
- Wird täglich aktualisiert

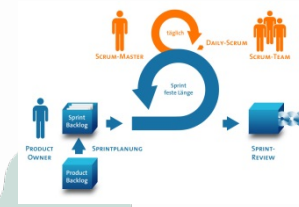
I	Independent	Sie sollten unabhängig von einander sein
N	Negotiable	Sie sollten verhandelbar sein
V	Valuable	Sie sollten einen konkreten Wert für den Customer haben
E	Estimatable	Sie sollten schätzbar sein
S	Small	Sie sollten klein sein
T	Testable	Sie sollten testbar sein

Quelle: http://en.wikipedia.org/wiki/INVEST_%28mnemonic%29



Sprint

- Zentraler Aspekt in Scrum, um den sich die anderen Abläufe richten
- Kleine, feste Zeitperiode (max. 30 Tage)
- Enthält Ziele, die spezifisch und messbar sind
- Werden in einer überschaubaren Menge in User Stories beschrieben
- Scrum Team konzentriert sich ausschließlich auf diese Aufgaben
- Am Ende entsteht ein fertiges Teilprodukt (Product Increment)
- Timebox (fester zeitlicher Rahmen)



Scrum Meetings (I)



Sprint Planning

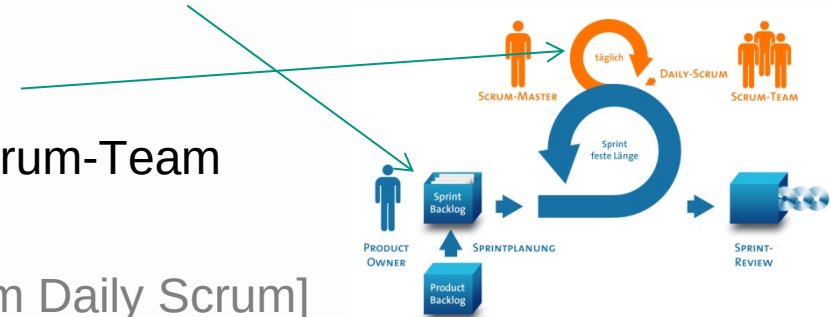
- Umfang des Springs → Backlog Items aufnehmen in Sprint Backlog
- Wie machen wir es?

Daily Scrum

Viertelstündiger Informationsaustausch im Scrum-Team

Jeder Entwickler beantwortet drei Fragen:

1. »Was habe ich geschafft?« [seit letztem Daily Scrum]
2. »Welchen Hindernissen bin ich begegnet?«
3. »Was will ich heute schaffen?«



Hinweis

Nur Abgleich, keine Diskussion/Beurteilung! Diese erfolgt im...

Scrum Meetings (II)

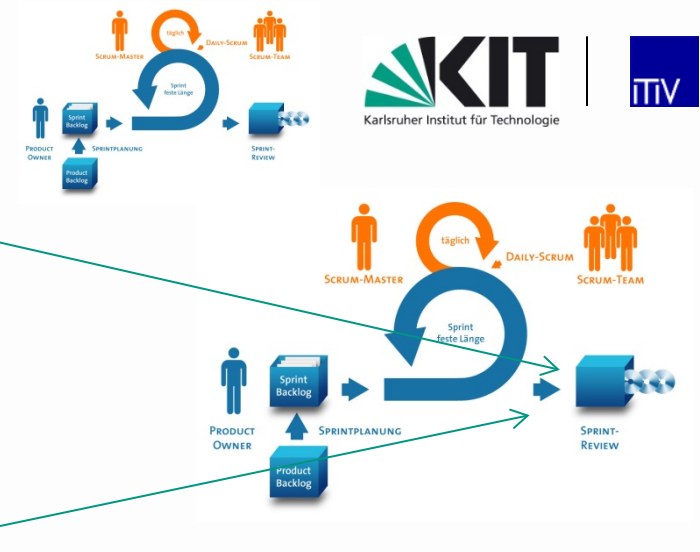
Sprint Review

Team und Product Owner tauschen sich aus.
Sichtbare Ergebnisse werden vorgestellt
Ideen und Vorschläge eingebracht

Sprint Retrospektive

Sprint wird betrachtet, Verbesserungsvorschläge gemacht, und Trends Erkannt
Ablauf wird darauf hin angepasst

➤ Empirische Verbesserung (**Continuous Improvement**)



Scrum (Stärken)

Stärken

Agile Entwicklung: Kurze Iterationen

➤ Transparenz

- Schnelles Erkennen und Reaktion auf Probleme
- Gute Messbarkeit des Fortschritts

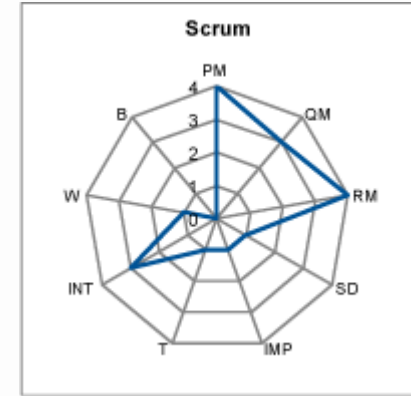
➤ Beurteilung

- Regelmäßiges Feedback (Produkt *und* Vorgehen)

➤ Flexibilität

- Kein endgültiger Plan – Kontinuierlich Änderungen möglich
- Produkt aus anpassbaren Produktteilen

➤ Risiken geringer, Verbesserung einfacher



Stärken von Scrum: PM, QM, RM

Quelle: <https://www.computerwoche.de/a/scrum-das-rahmenwerk,2352556>

Wasserfall

XP
SCRUM



V

Spiral

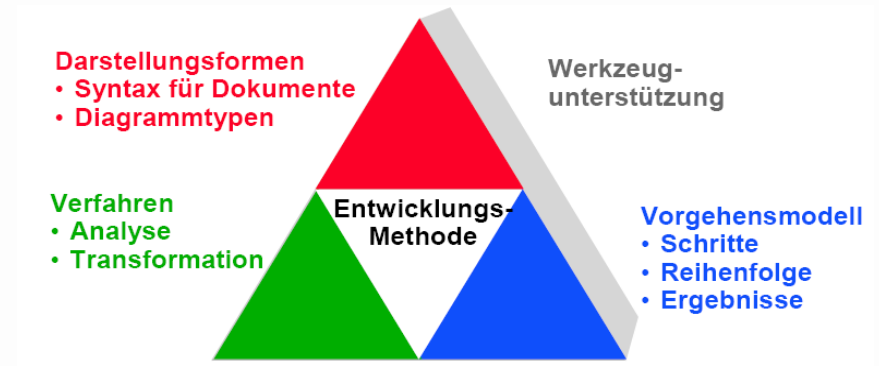


Eine *Methode* ist ein **strukturierter Ansatz** für die Softwareentwicklung.

Es handelt sich oft um **grafische Modelle** eines Systems, die zur **Systemspezifikation** oder zum **Systementwurf** verwendet werden.

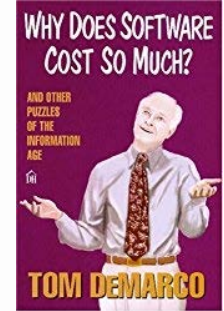
Ziel

- **Kostengünstige** Systeme
- Hohe **Qualität**



70er: Funktionsorientierte Methoden

- **Bestimmung** der Funktionalen **Komponenten**:
 - Strukturierte Analyse [SA] DeMarco 1978
 - Jackson Structured Design Jackson 1983



80er, 90er: Objektorientierte Methoden

- **erweitern Funktionsorientierte** Methoden
- Sprache: Unified Modeling Language (**UML**)
 - Booch 1994
 - Rumbaugh 1991
 - Fowler and Scott 1997



Beschreibung von Systemmodellen

- Definition der verwendeten **Notation**
- Bsp: Klassenmodelle, Datenflussmodelle, Zustandsmodelle, usw.

Regeln

- **Beschränkungen**, die für Systemmodelle immer gelten
- Bsp: Name innerhalb eines Modells muss eindeutig sein

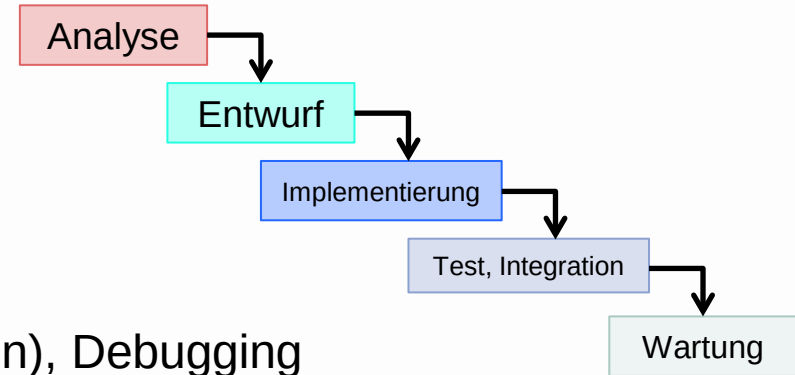
Empfehlungen

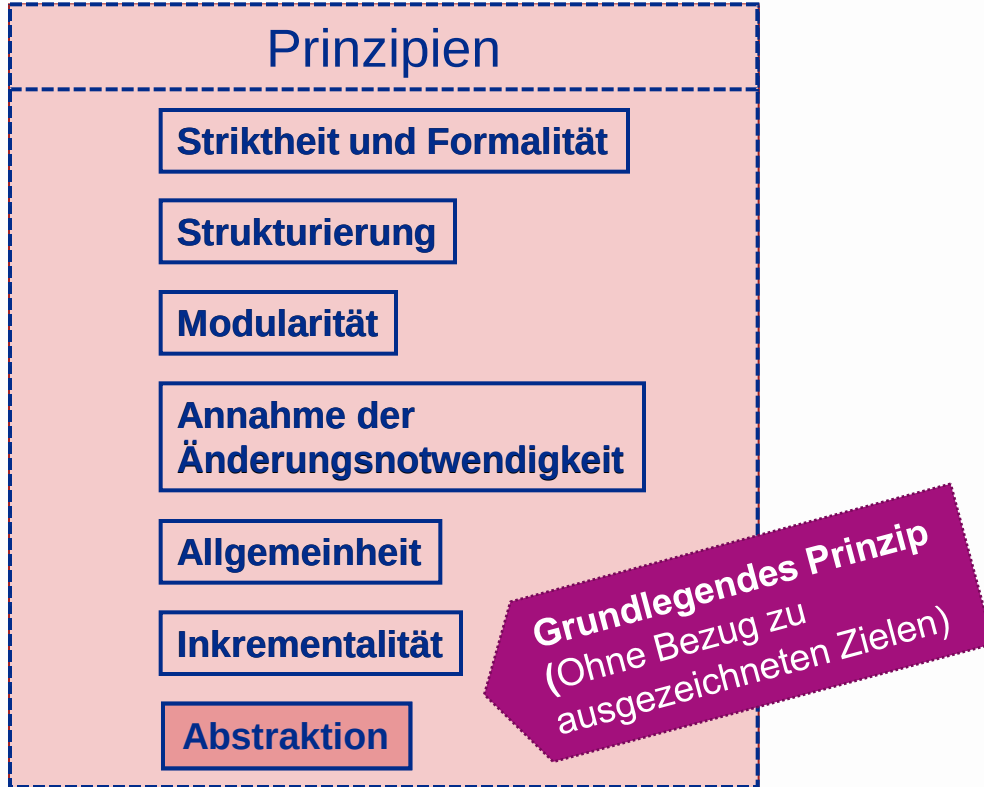
- **Verfahren** eines **guten Entwurfstiles** (Entwurfsmuster, ...)
- Führt zu einem gut organisierten Systemmodell
- Bsp: Kein Objekt sollte mehr als 7 untergeordnete Objekte haben

Anleitung zum Vorgehen

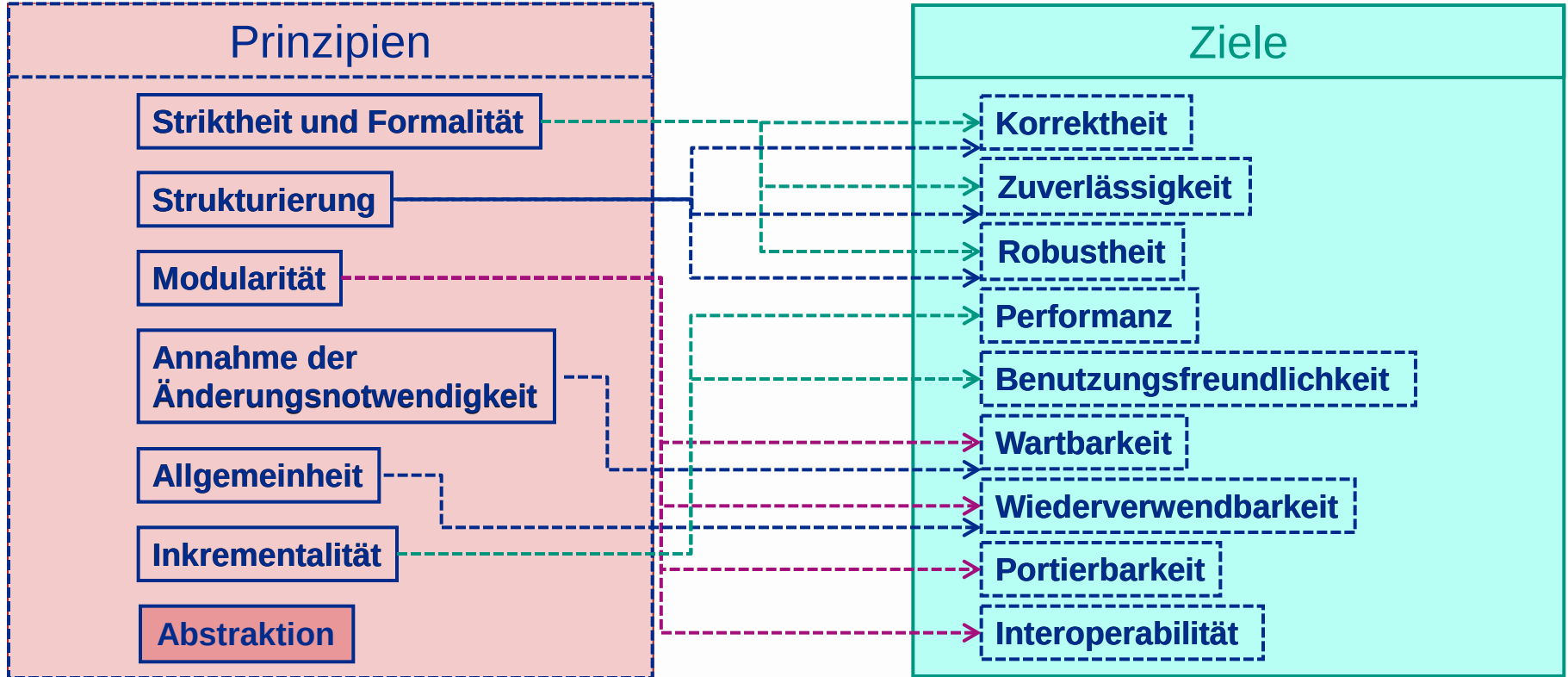
- **Beschreibung** der durchzuführenden **Aktivitäten**
- **Organisation** der Aktivitäten
 - Bsp: Objektattribute definieren, bevor man Operationen definiert

- **CASE: Computer-Aided Software Engineering**
dt.: computerunterstütztes Software-Engineering
- Werkzeuge (Softwareprogramme)
- Unterstützte Aufgaben:
 - Anforderungsanalyse
 - Systemmodellierung
 - Fehlerbehebung
 - Tests, Testfallgeneratoren
 - Codegenerator
 - Implementierung (Programmeditoren), Debugging
 - ...





Prinzipien des Software Engineering



Vorüberlegung: Warum so viele Ansätze für SW-Projekte?

Vergleich – Brückenbau

heute optimiert: bis ins Detail geplant – keine Probleme!

- ✓ im Kostenrahmen
- ✓ gemäß Zeitplan
- ✓ „doesn't crash“

Grund: *Frozen Design*

Bei Software nicht zutreffend!

- Welche Faktoren müssen zudem beachtet werden?



CHAOS Report

- vom IT-Beratungsunternehmen *The Standish Group* konzipiert; **erstmalig 1994** veröffentlicht
- **Studie zu Erfolgsraten** von **SW-Projekten** und deren Management
- als jährlicher **Schnappschuss** vom industriellen **Entwicklungs-Status** im IT-Bereich
- zum Erkennen von **Erfolgsfaktoren**

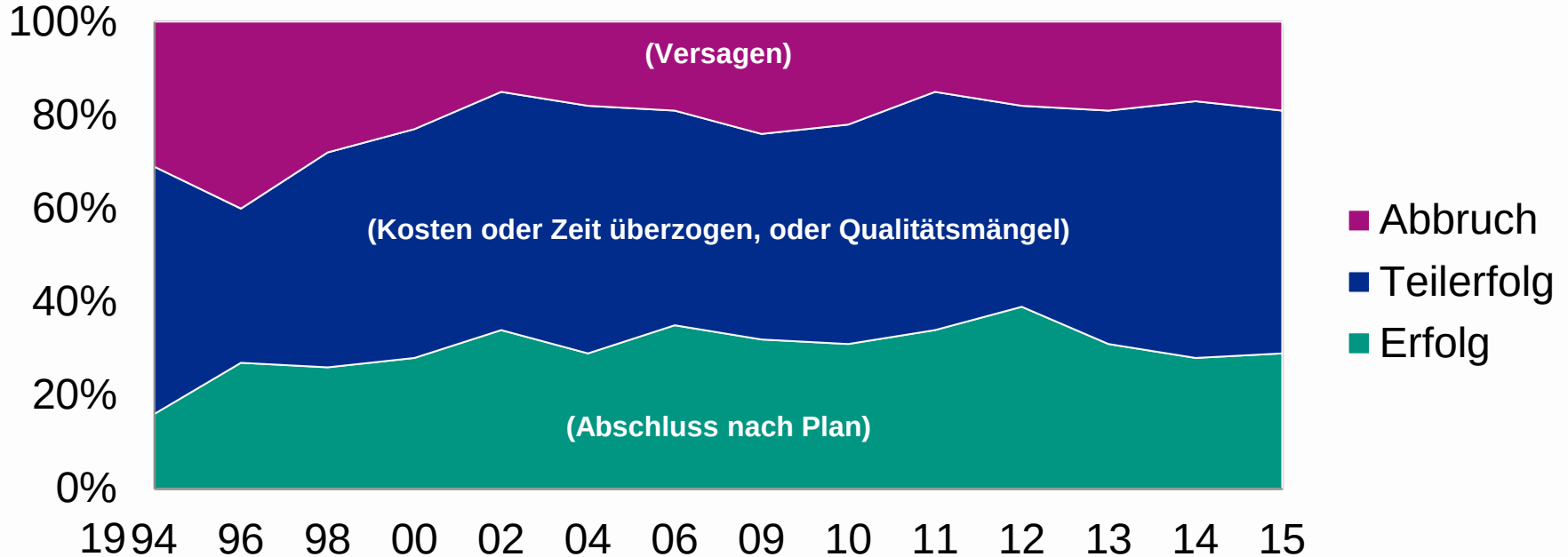


<http://www.standishgroup.com/outline>

Gliederung

- a) In drei Typen (Successful, Challenged, Failed)
- b) Nach Entwicklungs-Paradigmen:
 - **Ad hoc** (*kein Muster*)
 - **Iterativ** (Aufgabenspezifische Einteilung in Zeitfenster)
 - **Agil** (Iterativ; zudem kollaborativ, selbstorganisiert, ›schlank‹)
Beispiel: XP, Scrum, Kanban
 - **Traditionell** (Sequenziell)
Beispiel: Wasserfall
 - **Lean Development** (,Wertstromfluss‘, mit Kundenbestimmung)

Abschluss von Software-Projekten



Quelle: The Standish Group

Vergleich von Vorgehensmodellen (Quelle: Chaos Report 2015)

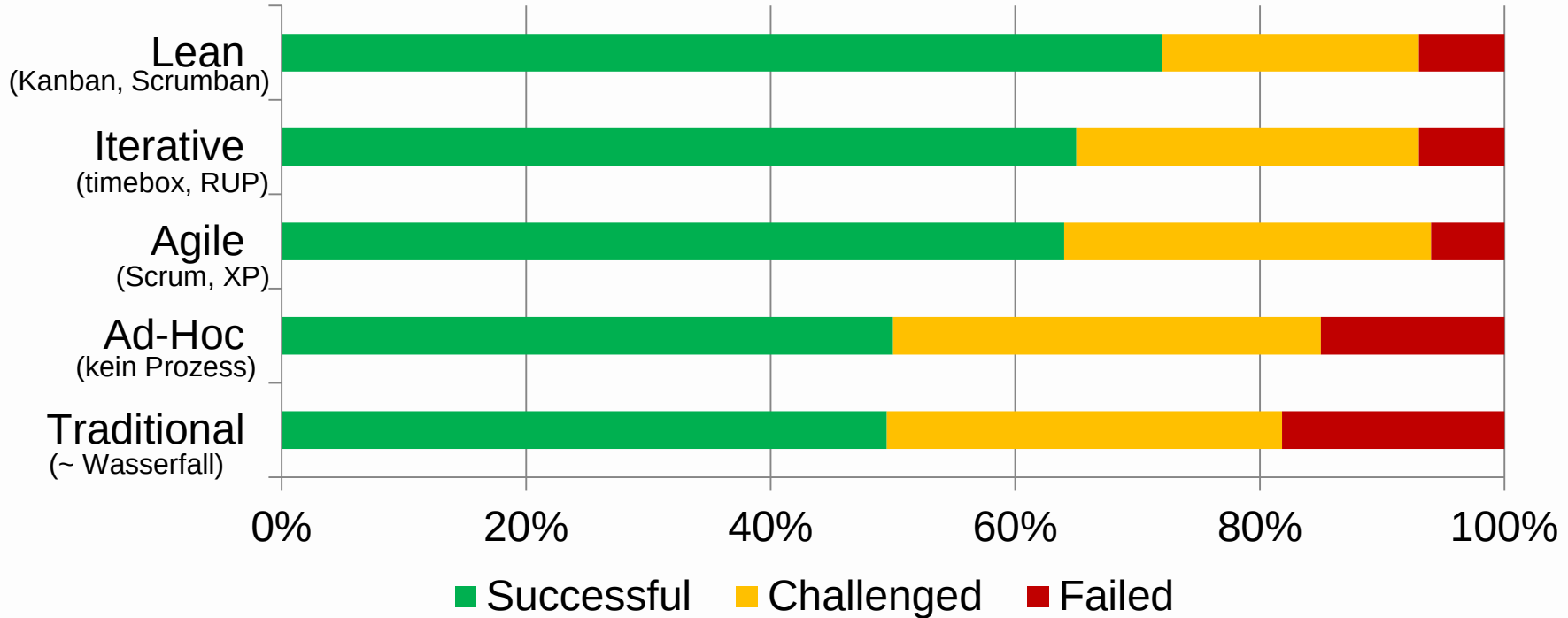
CHAOS RESOLUTION BY AGILE VERSUS WATERFALL

SIZE	METHOD	SUCCESSFUL	CHALLENGED	FAILED
All Size Projects	Agile	39%	52%	9%
	Waterfall	11%	60%	29%
Large Size Projects	Agile	18%	59%	23%
	Waterfall	3%	55%	42%
Medium Size Projects	Agile	27%	62%	11%
	Waterfall	7%	68%	25%
Small Size Projects	Agile	58%	38%	4%
	Waterfall	44%	45%	11%

The resolution of all software projects from FY2011–2015 within the new CHAOS database, segmented by the agile process and waterfall method. The total number of software projects is over 10,000.

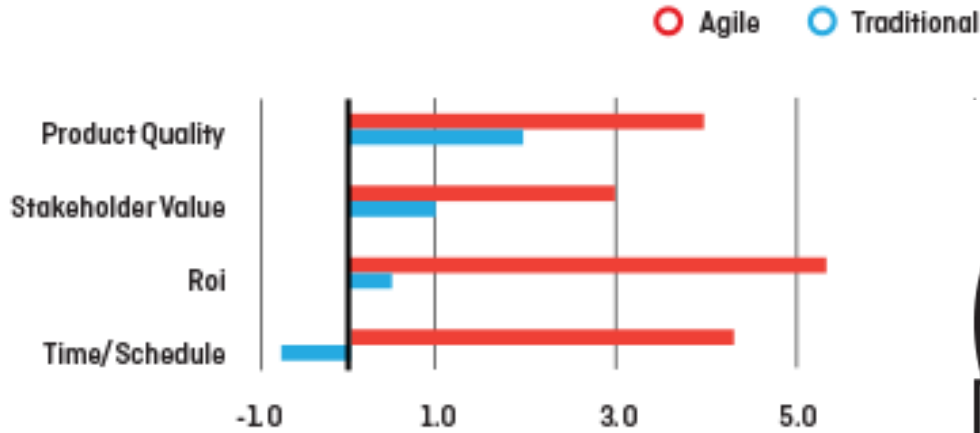
Quelle: The Standish Group
<https://www.infoq.com/articles/standish-chaos-2015>

Vergleich von Vorgehensmodellen (Quelle: Ambysoft 2013)

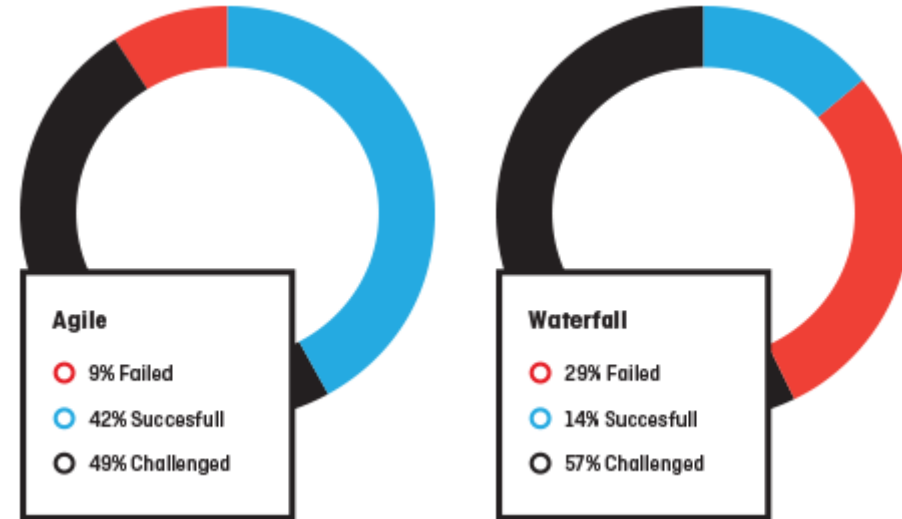


Quelle: Scott W. Ambler www.ambysoft.com/surveys/

Agile vs. Waterfall

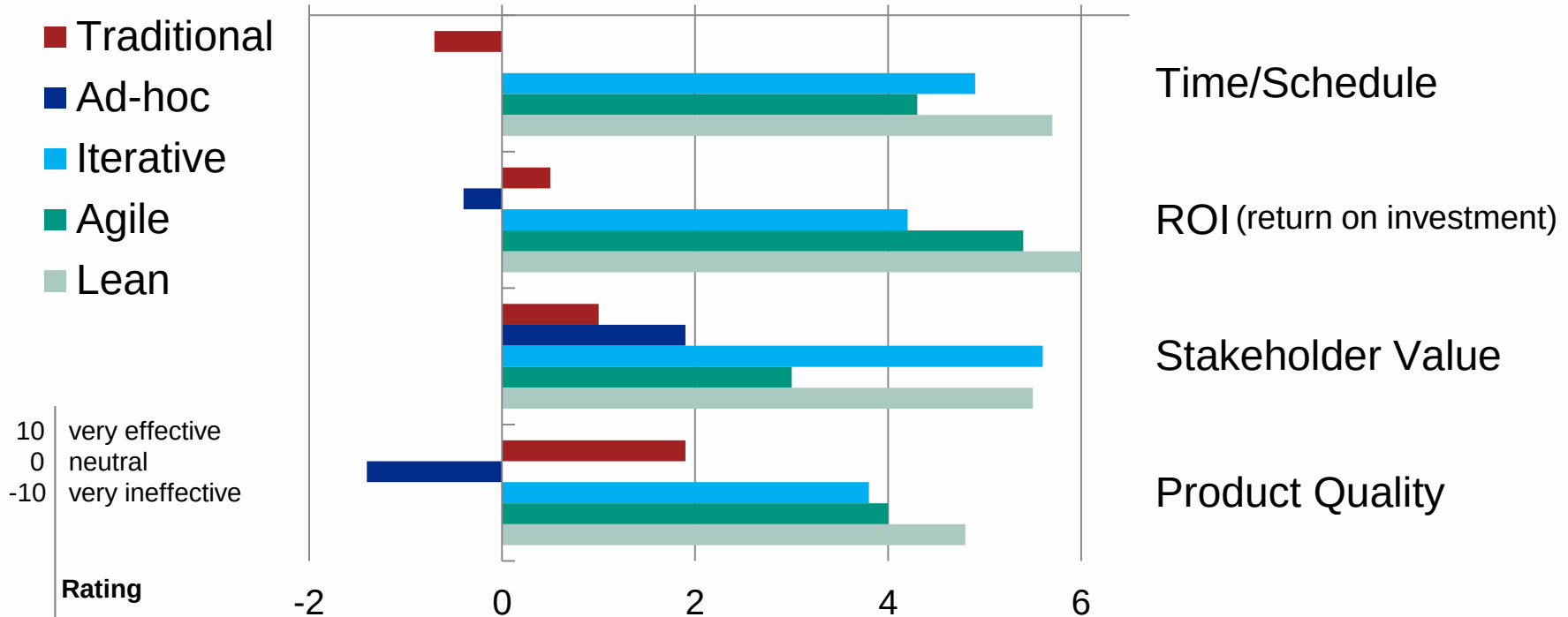


Ambyssoft 2013 Project Success Rates Survey



Quelle: Chaos Report 2015

Beurteilung von Auslieferungs-Paradigmen



Quelle: Scott W. Ambler www.ambysoft.com/surveys/

Fragen?

Methoden



CASE

Prinzipien