

Martin Glinz Harald Gall

Software Engineering

Kapitel 1

Einführung:

Software-Entwicklung und -Pflege als Problem



Universität Zürich
Institut für Informatik

1.1 Mehr als Programmieren

1.2 Software überall

1.3 Software als „Material“

1.4 Software-Entwicklung und -Pflege

1.5 Warum haben wir Probleme?

1.6 Software Engineering

Fallstudie „Gesundheitskarte“

- Deutsches Großprojekt zur Informatisierung des Gesundheitswesens
- Gesundheitskarte ist Träger für
 - Patientendaten (Blutgruppe, Impfungen, Allergien, Behandlungen...)
 - Versicherungsdaten
 - Rezepte
 - Arztbriefe
- Teilprojekt elektronisches Rezept
 - Arzt legt Rezept auf Karte ab
 - Apotheke liest Rezept von Karte und rechnet mit Kasse ab

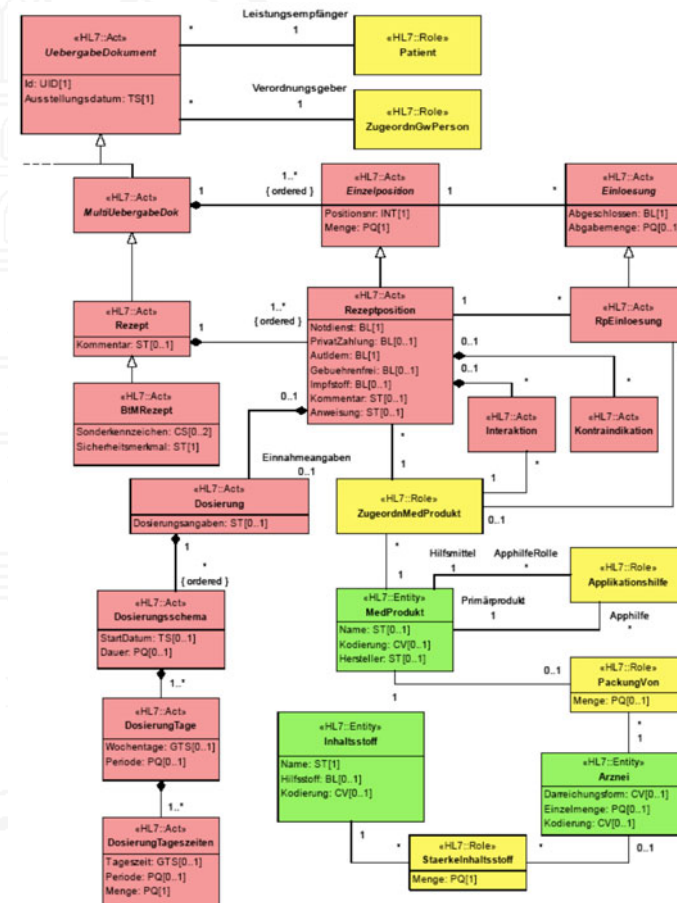


Lösung für das elektronische Arztrezept – so?

Faktisch so: e-Rezept im Projekt „Gesundheitskarte“

```

class eRezept {
    String arzt;
    String patient;
    String arznei;
    int menge;
    String dosierung;
}
    
```



e-Rezept im Projekt „Gesundheitskarte“: Lehren

- **Komplizierter als man denkt**
- **Mehr als Programme:** Anforderungen und Grobarchitektur in diesem Projekt umfassen 1200 Seiten Dokumentation
- Nur beherrschbar mit
 - **Technischen Mitteln**, die weit über Programmieren hinausgehen: Modelle, Spezifikationen, Entwürfe, Prüfpläne...
 - Geeigneten **Managementmitteln**: Prozesse, Projektmanagement,...
 - **Infrastruktur-Unterstützung**: Werkzeuge, Konfigurationsmanagement,...

1.1 Mehr als Programmieren

1.2 Software überall

1.3 Software als „Material“

1.4 Software-Entwicklung und -Pflege

1.5 Warum haben wir Probleme?

1.6 Software Engineering

Brauchen wir Software?

“Software is arguably the world’s most important industry. The presence of software has made possible many new businesses and is responsible for increased efficiencies in most traditional businesses.”

Grady Booch (Communications of the ACM, März 2001)

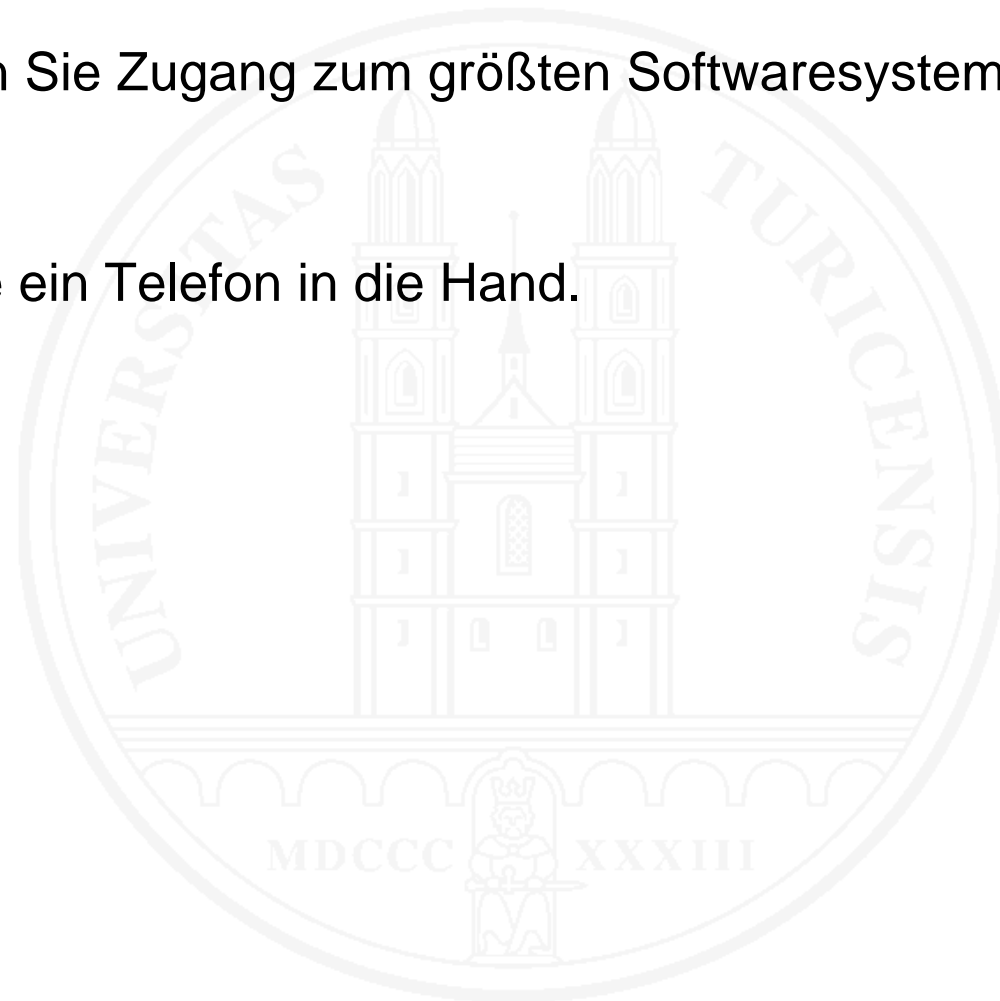
- Stellen Sie sich die Welt (und Ihr persönliches Leben!) **ohne Software** vor...
- ...und vergessen Sie nicht: Software hat es nicht nur in Ihrem PC
- **Software ist überall.**

Software ist überall

- Im **Alltag**: Telefon, Fernseher, Waschmaschinen, ...
- In **Fahrzeugen**:
 - Autos (\$675 Stahl vs. \$2500 Elektronik in neuen Autos von GM)
 - Lokomotiven (Re 4/4-460: 32 Prozessoren)
- In **Flugzeugen** (“fly-by-wire”)
- In der **Industrie**: Konstruktion, Produktionsplanung, Produktionssteuerung, Logistik, Bestellwesen, Buchhaltung,...
- In **Handel** und **Dienstleistungen**: Banken, Börse, Devisenmärkte, Reisebüros, elektronischer Handel mit Gütern und Information...
- In der **Energieversorgung**: Stromerzeugung, Stromverteilung
- In den **Medien**: Zeitungen, Fernsehen, Internet
- ...

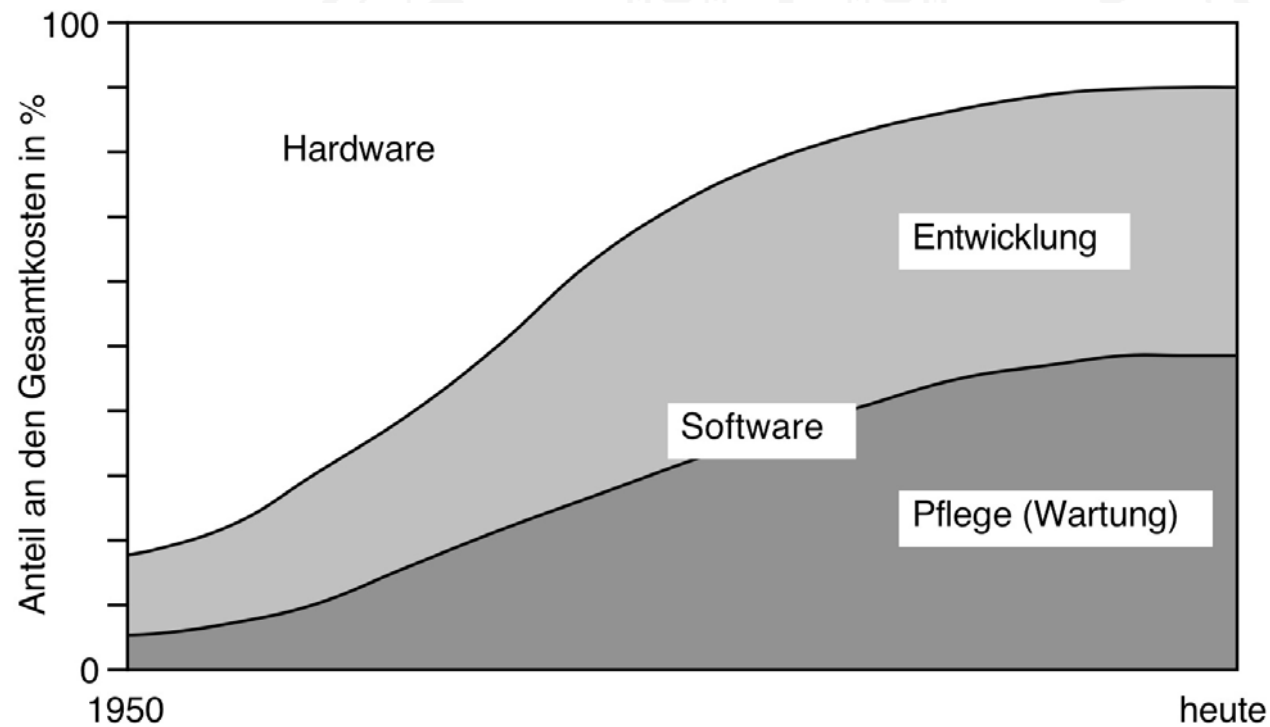
Preisfrage

- Wie erhalten Sie Zugang zum größten Softwaresystem der Welt?
- Nehmen Sie ein Telefon in die Hand.

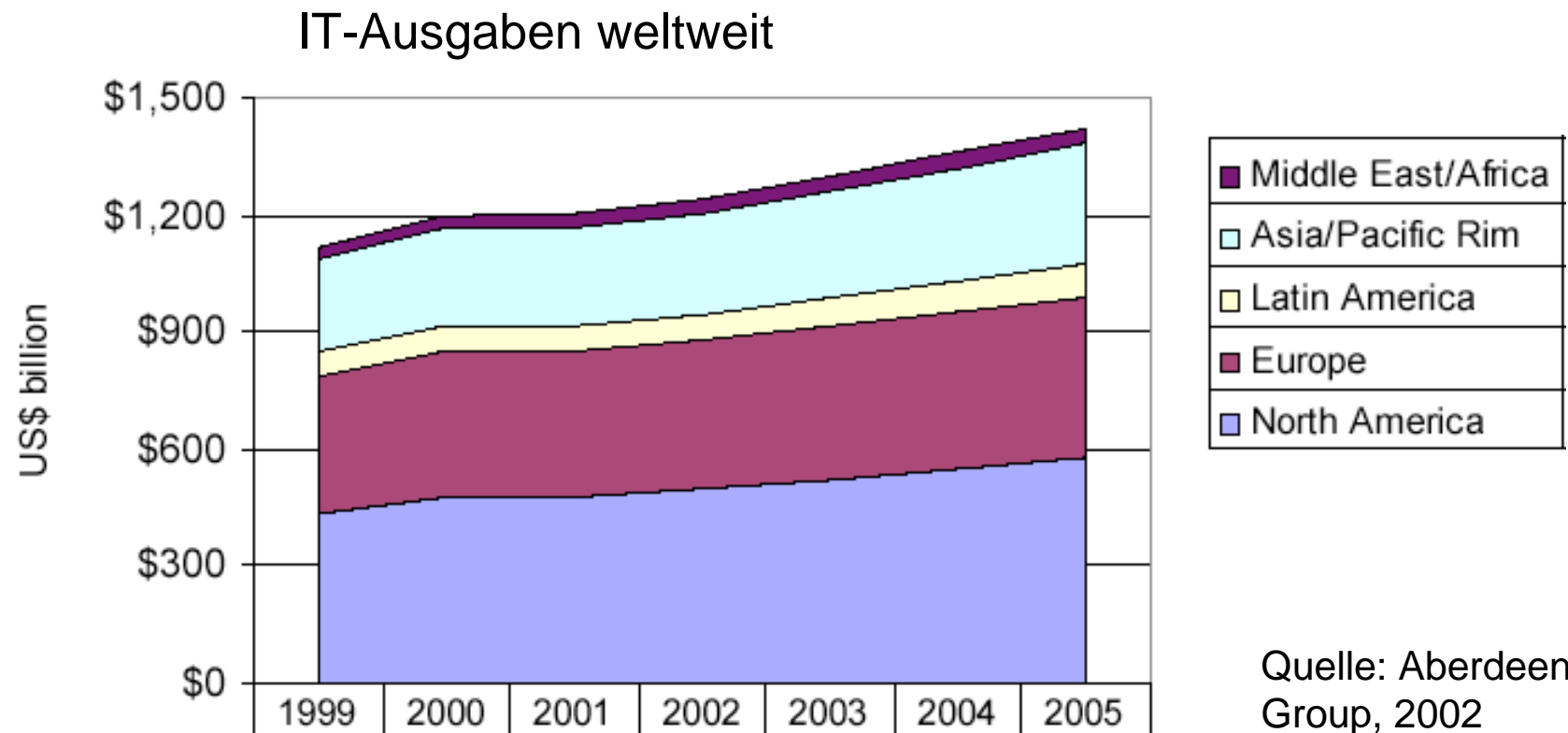


Die Dimension des Problems

- Ohne Software geht heute nichts mehr
- Software-Kosten dominieren die Kosten von Informatiksystemen



Ausgaben für Software weltweit: horrende Summen



○ Es gibt Verbesserungspotential

⇒ **Software Engineering**

1.1 Mehr als Programmieren

1.2 Software überall

1.3 Software als „Material“

1.4 Software-Entwicklung und -Pflege

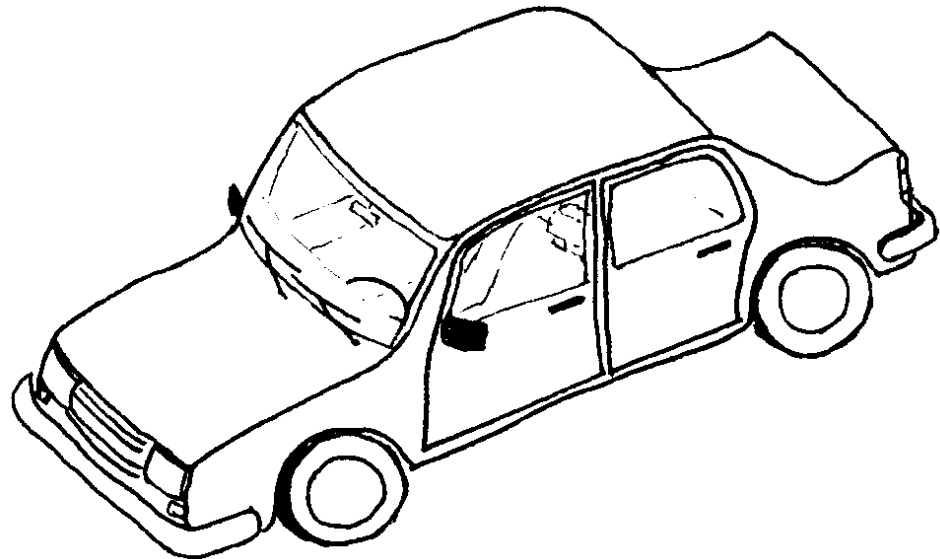
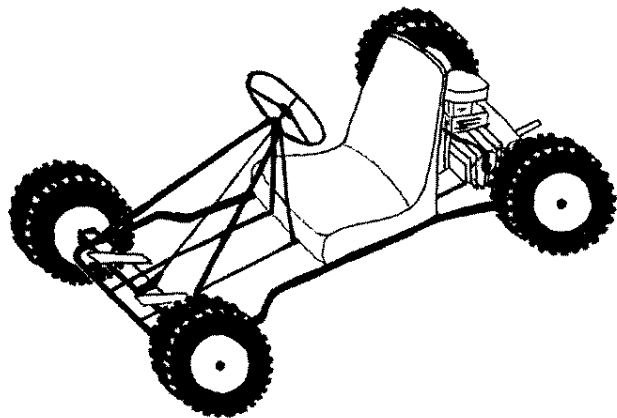
1.5 Warum haben wir Probleme?

1.6 Software Engineering

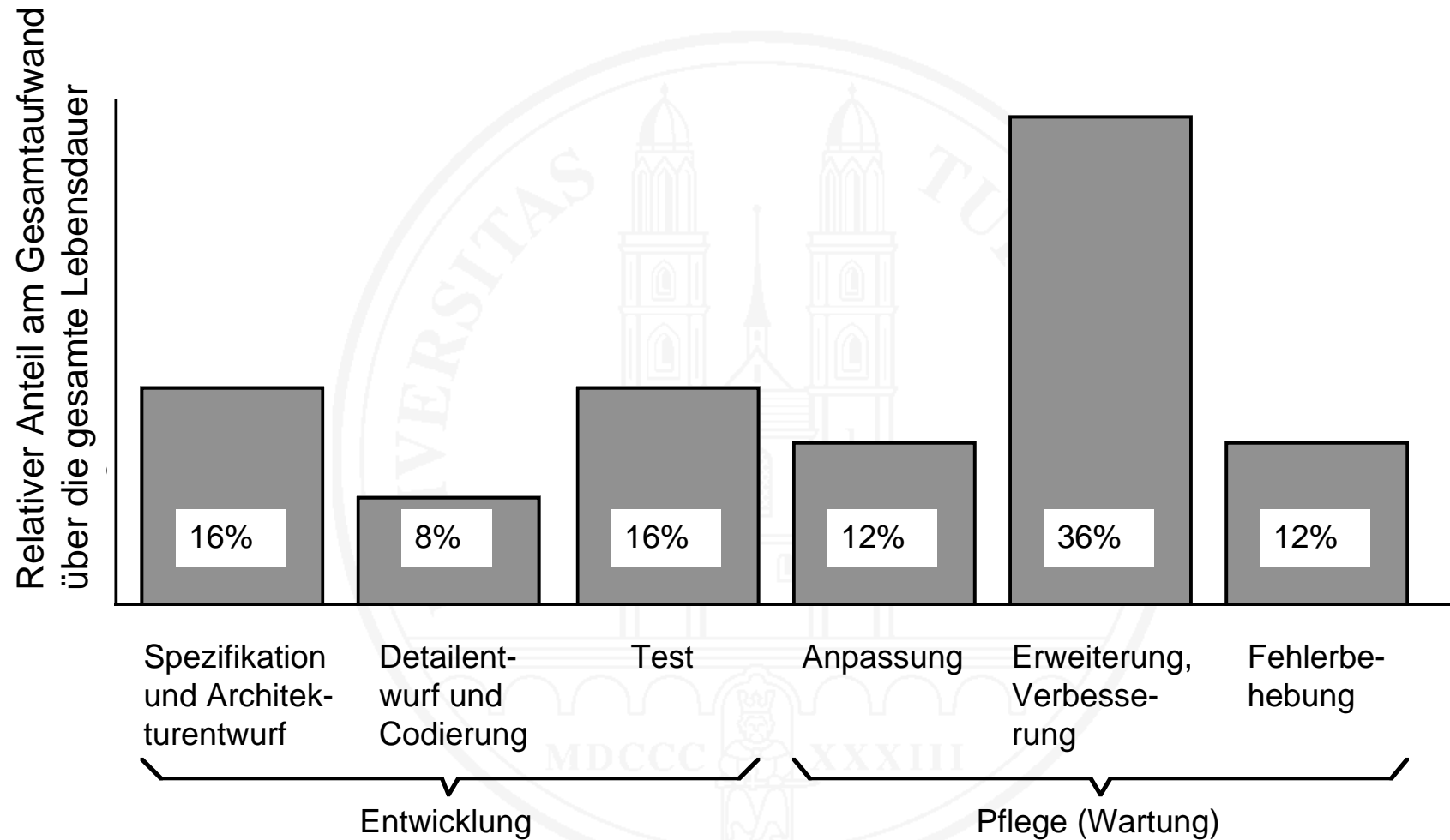
Was ist Software?

Software. Die Programme, Verfahren, zugehörige Dokumentation und Daten, die mit dem Betrieb eines Rechnersystems zu tun haben (IEEE 610.12).

⇒ *Mehr als nur Programme.*

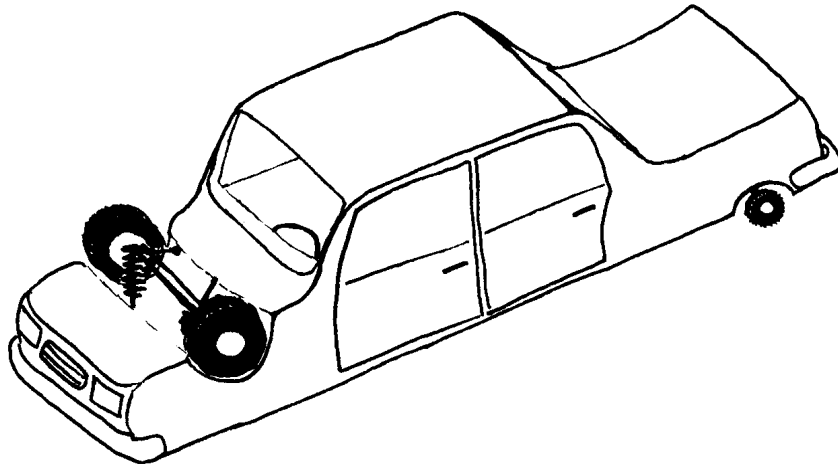


Verteilung des Aufwands über die Lebensdauer

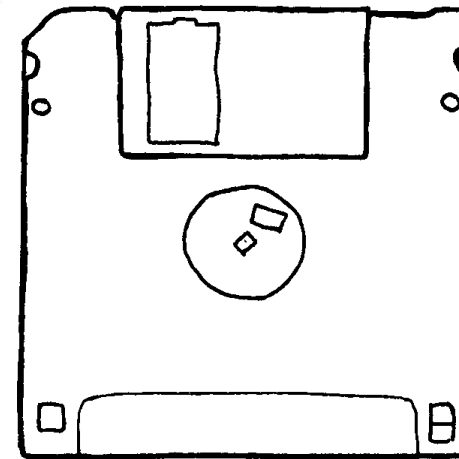


Ein immaterielles technisches Produkt – 1

Software kann man nicht anfassen.



Diese mechanische Konstruktion ist offensichtlich **falsch**.



Wie erkennen wir aber **Fehler** in der hier gespeicherten Software-Konstruktion?

Ein immaterielles technisches Produkt – 2

- Beobachtbar nur
 - in den **Wirkungen** beim Ablauf auf Rechnern
 - indirekt über die **Dokumentation** der Software

- Kein **Materialwert**
- Keine physikalischen **Grenzen**
- **Fehler** sind schwieriger erkennbar
- **Entwicklungsstand** und **Qualität** schwer zu beurteilen
- Scheinbar leicht zu **ändern**

Verhält sich unstetig

- kleinste Änderungen in der Software
 - ⇒ massive Änderungen im Verhalten
- Nachweis des wunschgemäßen Funktionierens schwierig

⇒ **schwieriger als andere technischen Produkte**

Wozu Software?

- Probleme lösen
- Automatisierung oder Unterstützung von
 - menschlicher Arbeit
 - technischen Abläufen
 - in beliebigen Anwendungsbereichen
- steht in ständiger Wechselwirkung mit
 - Arbeitsprozessen
 - Produktionsprozessen
 - Menschen

Konsequenzen

- Komplexes Problem ⇨ komplexe Lösung
- Immer **zwei Schwierigkeiten**
 - Problem im Kontext des jeweiligen Anwendungsbereichs verstehen
 - Problemlösung auf Software abbilden
- Problemlösungen **schaffen neue Realitäten und Bedürfnisse**
- ⇨ **Software konstruiert und verändert die Realität**

Systeme vs. Programme

- Mengen von Programmen genügen nicht; wir brauchen **Systeme**
- Beispiel: Das System e-Rezept
 - ist Teil des Systems Gesundheitskarte
 - interagiert mit anderen Teilsystemen, zum Beispiel
 - Sicherheit
 - Lesen/Schreiben der Karte
 - Das System Gesundheitskarte wiederum integriert
 - die Karte selbst
 - die Lese-/Schreibinfrastruktur für die Karte
 - bestehende Computersysteme bei Ärzten, Apotheken und Versicherungen
- ☞ Durch die (notwendige) **Integration** wird aus vielen **einfachen kleinen Teilproblemen** ein **komplexes und schwieriges Gesamtproblem**

1.1 Mehr als Programmieren

1.2 Software überall

1.3 Software als „Material“

1.4 Software-Entwicklung und -Pflege

1.5 Warum haben wir Probleme?

1.6 Software Engineering

Software-Entwicklung

Software-Entwicklung (software development) – Die Umsetzung der Bedürfnisse von Benutzern in Software.

Umfasst Spezifikation der **Anforderungen**, Konzept der **Lösung**, **Entwurf** und **Programmierung** der Komponenten, **Zusammensetzung** der Komponenten und ihre **Einbindung** in vorhandene Software, Inbetriebnahme der Software sowie **Überprüfung** des Entwickelten nach jedem Schritt.

- Das Problem verstehen
 - Anwendungsbereich verstehen
 - Anforderungen spezifizieren
- Lösung entwerfen
- Lösung umsetzen (→ programmieren)
- Lösung integrieren und in Betrieb nehmen

Zwischen- und End-
ergebnisse prüfen

Software-Pflege

- Die Benutzerbedürfnisse ändern sich schneller, als Software von Grund auf neu entwickelt werden kann
- Bestehende Software muss **geändert** und **weiterentwickelt** werden

Software-Pflege (oder Software-Wartung, software maintenance) – Modifikation und/oder Ergänzung bestehender Software durch neue Software mit dem Ziel,

- **Fehler** zu **beheben**,
- die bestehende Software an veränderte Bedürfnisse oder Umweltbedingungen **anzupassen**
- oder die bestehende Software um neue Fähigkeiten zu **erweitern**.

→ Kapitel 14: Software-Pflege und -Evolution

1.1 Mehr als Programmieren

1.2 Software überall

1.3 Software als „Material“

1.4 Software-Entwicklung und -Pflege

1.5 Warum haben wir Probleme?

1.6 Software Engineering

Was Software-Entwicklung und -Pflege schwierig macht

- Arbeit im Kleinen \neq Arbeit im Großen
 - Was im Kleinen Erfolg bringt, skaliert vielfach nicht
 - Viele Probleme entstehen erst beim Arbeiten im Großen
- $n \cdot \text{Klein} \neq 1 \cdot \text{Groß}$
 - Aufwand steigt überproportional mit Produktgröße
 - Quantensprünge im Wachstum
- Integration vieler heterogener Teile ist unausweichlich
 - Verstehensprobleme
 - Anpassungsprobleme
 - Viele Beteiligte mit unterschiedlichen Interessen
- Nicht linearer Prozess
 - Spezifikation, Entwurf und Realisierung sind miteinander verzahnt

Was Software-Entwicklung und -Pflege schwierig macht – 2

- Software **schließt** die **Lücke** zwischen
 - den spezifischen Problemen in einem Anwendungsbereich
 - und den unspezifischen Fähigkeiten von Rechnern

⇒ Erfordert **gleichzeitig Anwendungs- und Problemlösungswissen**
- Software ist einer **Evolution** unterworfen
 - Bedürfnisse ändern sich
 - Technologie ändert sich
 - Nie fertig
- Software wird **von Menschen gemacht**
 - **Können** und **Produktivität** der Beteiligten haben starken Einfluss
 - **Emotionale Fehleinschätzungen**
 - fehlendes Gefühl für immaterielle Produkte
 - Klein ist fein

Klein vs. Groß in der Software-Entwicklung – 1

Klein	Groß
Programme von 1 bis ca. 300 Zeilen	Längere Programme
Für den <i>Eigengebrauch</i>	Für den Gebrauch durch <i>Dritte</i>
<i>Vage Zielsetzung</i> genügt	<i>Genaue Zielbestimmung</i> erforderlich
<i>Ein Schritt</i> vom Problem zur Lösung genügt: Lösung direkt programmiert	<i>Mehrere Schritte</i> erforderlich
<i>Validierung am Endprodukt</i>	<i>Auf jeden Entwicklungsschritt muss ein Prüfschritt</i> folgen
<i>Eine Person</i> entwickelt: Keine Koordination notwendig, keine Kommunikationsbedürfnisse	<i>Mehrere Personen</i> entwickeln gemeinsam: Koordination und Kommunikation notwendig

Klein vs. Groß in der Software-Entwicklung – 2

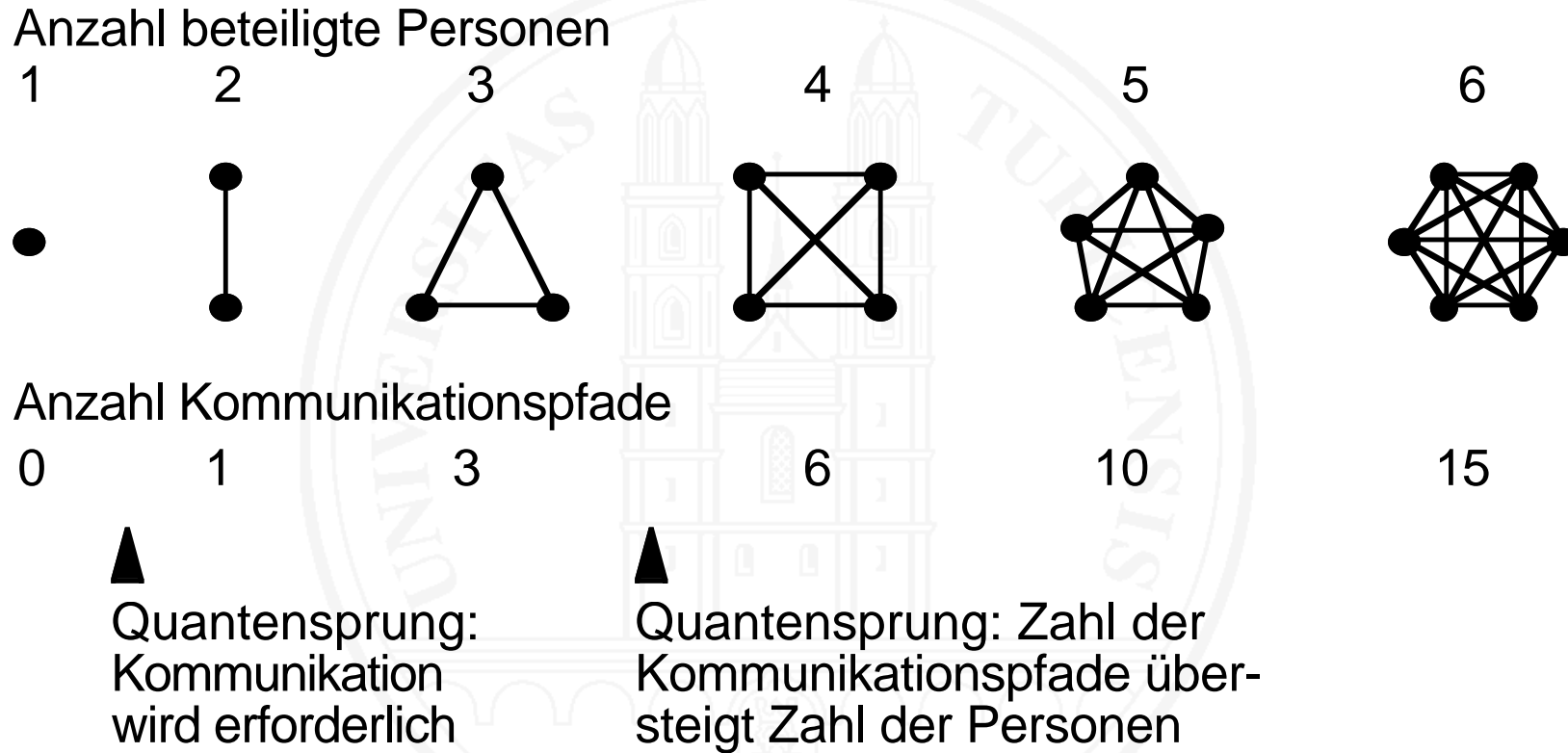
Klein	Groß
<i>Komplexität</i> des Problems in der Regel <i>klein</i> ,	<i>Komplexität</i> des Problems <i>größer bis sehr groß</i> ,
Behalten der Übersicht nicht schwierig	Maßnahmen zur Strukturierung und Modularisierung erforderlich
Software besteht aus <i>wenigen Komponenten</i>	Software besteht aus <i>vielen Komponenten</i> ; Maßnahmen zur Komponentenverwaltung erforderlich
In der Regel <i>keine Dokumentation</i>	<i>Dokumentation</i> zwingend
<i>Keine Planung</i> und <i>Projektorganisation</i> erforderlich	<i>Planung</i> und <i>Projektorganisation</i> zwingend erforderlich

Mini-Übung 1.1 (Aufgabe 1.2 im Skript)

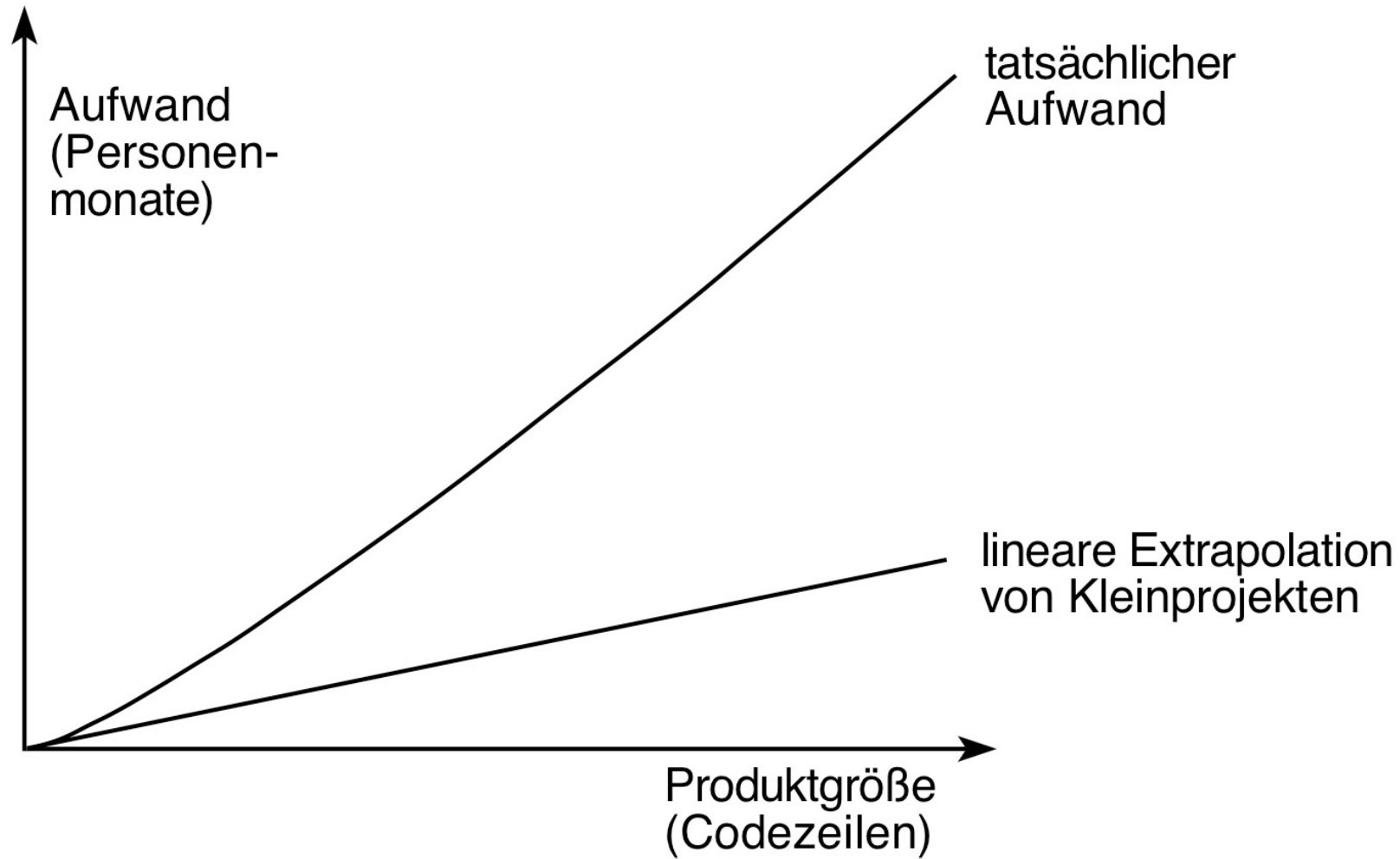
«Ein Mann braucht zum Bau einer 2 m langen Brücke 0,5 Tage. Wie lange brauchen 100 Leute für den Bau einer 2 km langen Brücke? Rechne.»

Begründen Sie, warum das eine Milchmädchenrechnung ist. Ziehen Sie Parallelen zur Entwicklung von Software.

Wachstum des Kommunikationsbedarfs



Der Aufwand steigt überproportional



Mini-Übung 1.2 (Aufgabe 1.4 im Skript)

Eine Kundenbetreuerin im Firmenkundengeschäft einer Bank hat auf der Grundlage eines Tabellenkalkulationsprogramms eine kleine persönliche Anwendung geschrieben, die sie bei der Überprüfung der Kredite der von ihr betreuten Firmen unterstützt. Die notwendigen Daten gibt sie jeweils von Hand ein.

Der Abteilungsleiter sieht diese Anwendung zufällig, ist davon angetan und beschließt, sie allen Kundenbetreuerinnen und -betreuern zur Verfügung zu stellen. Die notwendigen Daten sollen neu automatisch als den Datenbanken der Bank übernommen werden.

Mini-Übung 1.2 (Fortsetzung)

Die Kundenbetreuerin gibt an, für die Entwicklung ihrer Anwendung insgesamt etwa vier Arbeitstage aufgewendet zu haben. Der Abteilungsleiter veranschlagt daher für die Übernahme und die gewünschten Änderungen einen Aufwand von einer Arbeitswoche. Als die geänderte Anwendung endlich zur Zufriedenheit aller Beteiligten läuft, sind jedoch rund acht Arbeitswochen Aufwand investiert.

Der Abteilungsleiter erzählt die Geschichte einem befreundeten Berater als Beispiel, dass Informatik-Projekte nie ihre Termine einhalten. Darauf meint der Berater trocken, der investierte Aufwand sei völlig realistisch und normal.

Begründen Sie, warum.

- 1.1 Mehr als Programmieren
 - 1.2 Software überall
 - 1.3 Software als „Material“
 - 1.4 Software-Entwicklung und -Pflege
 - 1.5 Warum haben wir Probleme?
 - 1.6 Software Engineering**
-

Ein Blick zurück

- Ca. 1940 - 1960
 - Wenig verfügbare Rechner, geringe Verarbeitungskapazität
 - Software besteht im Wesentlichen aus einzelnen Berechnungsprogrammen
 - 1960 - 1970
 - Menge, Leistungsfähigkeit und Zuverlässigkeit der Hardware wachsen dramatisch
 - Informationsverarbeitung in großem Stil wird möglich
 - Ad hoc Verfahren und Kunsthandwerk in der Software-Entwicklung versagen bei der Arbeit an großen Systemen
- ⇒ Die «Software-Krise» ist da

Software Engineering

- Die Antwort auf die « Software-Krise»
- 1968 durch F.L. Bauer in Garmisch-Partenkirchen

Software Engineering ist das **technische** und **planerische** Vorgehen zur **systematischen Herstellung und Pflege von Software**, die **zeitgerecht** und unter **Einhaltung der geschätzten Kosten** entwickelt bzw. modifiziert wird (Fairley 1985).

Software Engineering. Die Anwendung eines **systematischen, disziplinierten** und **quantifizierbaren** Ansatzes auf die Entwicklung, den Betrieb und die Wartung von Software, das heißt, die **Anwendung der Prinzipien des Ingenieurwesens auf Software**. (IEEE 610.12)

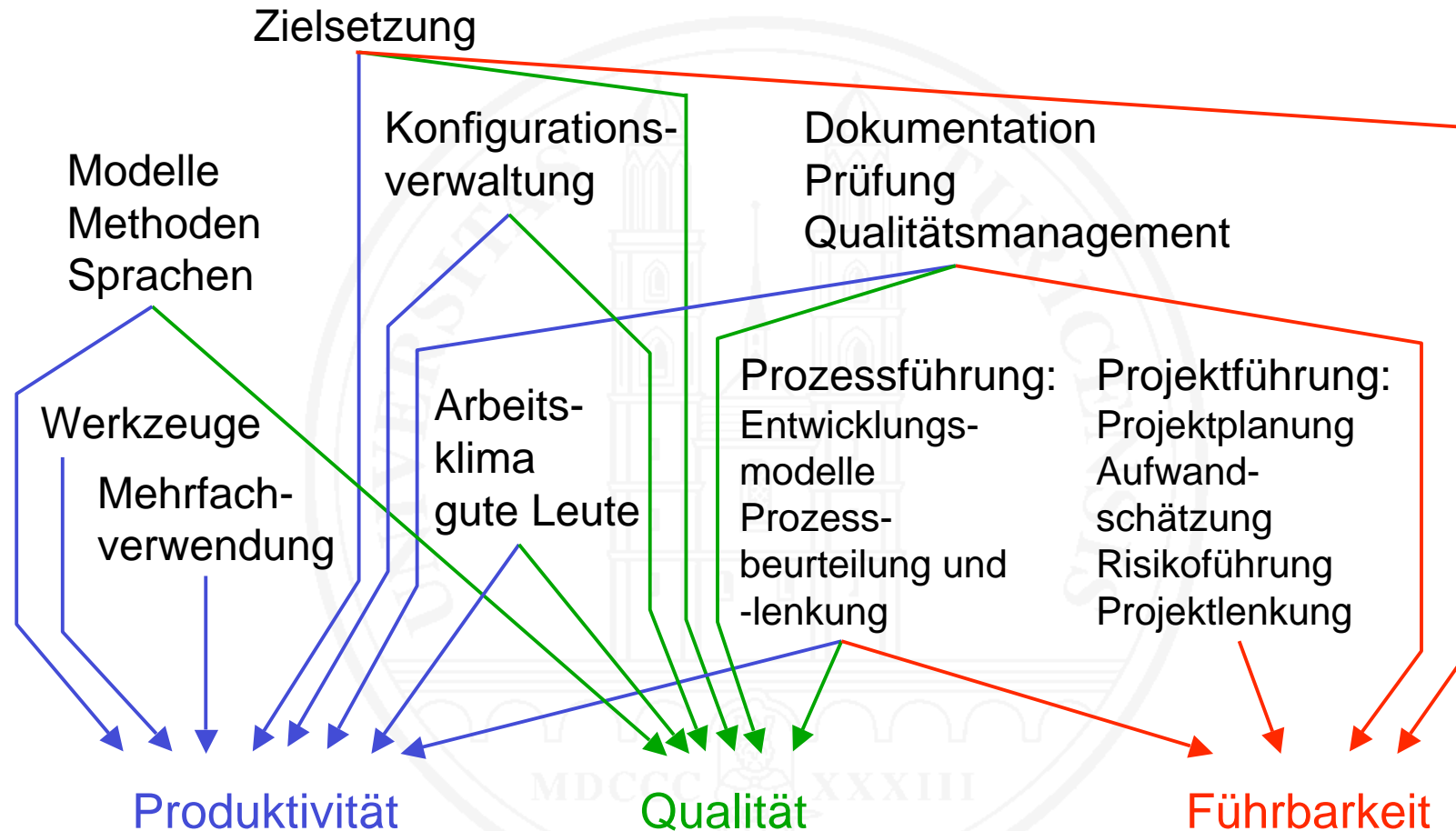
- Im deutschen Sprachraum auch: **Softwaretechnik**

Die Ziele des Software Engineerings

- Steigerung der **Produktivität**
- Verbesserung der **Qualität**
- Erleichterung der **Führbarkeit** von Projekten



Mittel des Software Engineerings und ihre Wirkung



In dieser Vorlesung: Die Mittel kennen lernen

Das Ende der Software-Krise?

- Die Software-Krise überwunden? Nein. Wir haben nach wie vor ...
 - ... häufige Termin- und Kostenüberschreitungen
 - ... ganz oder teilweise gescheiterte Projekte
 - ... zu wenig konsequent gelebtes Software Engineering
- Also nichts erreicht? Doch, sehr viel sogar.
 - Wir schaffen heute Software einer Größe und Komplexität, die vor 20 Jahren weit jenseits des Machbaren lag
 - Viele Routineprobleme werden heute gut beherrscht
- Sind wir schlechter als andere? Nein.
 - Software Engineering ist im Mittel schwieriger als das Engineering klassischer Systeme und Produkte
 - In der Beherrschung großer Non-Standardsysteme sind wir heute teilweise besser als klassische Ingenieure

Was tun?

- Auf der **Erzeugerseite**
 - Software Engineering **systematisch lernen** ...
... und **konsequent** in der Praxis **umsetzen**
 - Weg von der Kultur des Bastelns und Heimwerkens ...
... hin zu einer **Kultur professionellen Arbeitens**
- Auf der **Kundenseite**
 - **Bewusstsein** für den Zusammenhang zwischen Kosten, Terminen und Qualität stärken
 - Professionelles Software Engineering als **Selbstverständlichkeit** fordern ...
 - ... und bereit sein, den damit erzielten **Mehrwert** zu **bezahlen**

Literatur

Siehe Literaturverweise im Kapitel 1 des Skripts.

Im Skript [M. Glinz (2005). *Software Engineering*. Vorlesungsskript, Universität Zürich] lesen Sie bitte Kapitel 1.

Im Begleittext zur Vorlesung [S.L. Pfleeger, J. Atlee (2006). *Software Engineering: Theory and Practice*, 3rd edition. Upper Saddle River, N.J.: Pearson Education International] lesen Sie bitte in Kapitel 1: Why Software Engineering die Abschnitte 1.1, 1.2, 1.4-1.6 als Grundlage und die Abschnitte 1.7-1.16 als Vertiefung und Ergänzung.

