

# Requirements Engineering

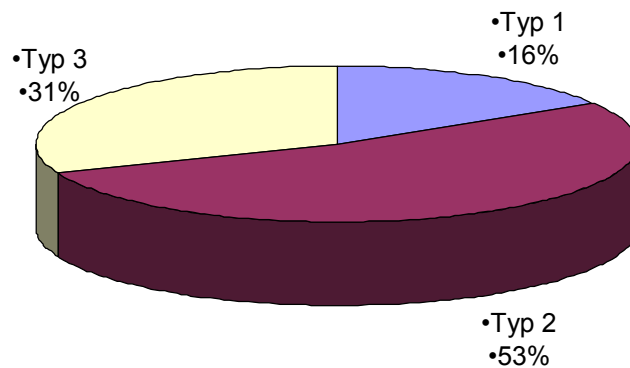
Die Wiege des Projekt-(Miss-)Erfolgs

**Dipl.-Ing. Johannes Bergsmann**

Gerichtlich beeideter Sachverständiger für Informatik

„Sag mir, wie Dein Projekt anfängt,  
und ich sage Dir, wie es endet“

## Situation in der Software-Entwicklung:



Chaos Report, The Standish Group

Typ 1: Projekt abgeschlossen  
- Im Zeitrahmen  
- Im Kostenrahmen  
- Mit geforderter Qualität

Typ 2: Projekt abgeschlossen  
- Teurer als geplant *oder*  
- Länger als geplant *oder*  
- Geringere Qualität als gefordert

Typ 3: Projekt abgebrochen

von 8300 untersuchten Projekten wurden  
**über 80% nicht erfolgreich** abgeschlossen!

(iX 4/2000 Seite 175)

Faktoren, die von 365 befragten DV-Managern für den Erfolg bzw. Misserfolg von insgesamt **8380 Softwareprojekten** verantwortlich gemacht werden: (Standish Group International Inc.: CHAOS-Report)

⇒ **Verteilung der Projekte:**

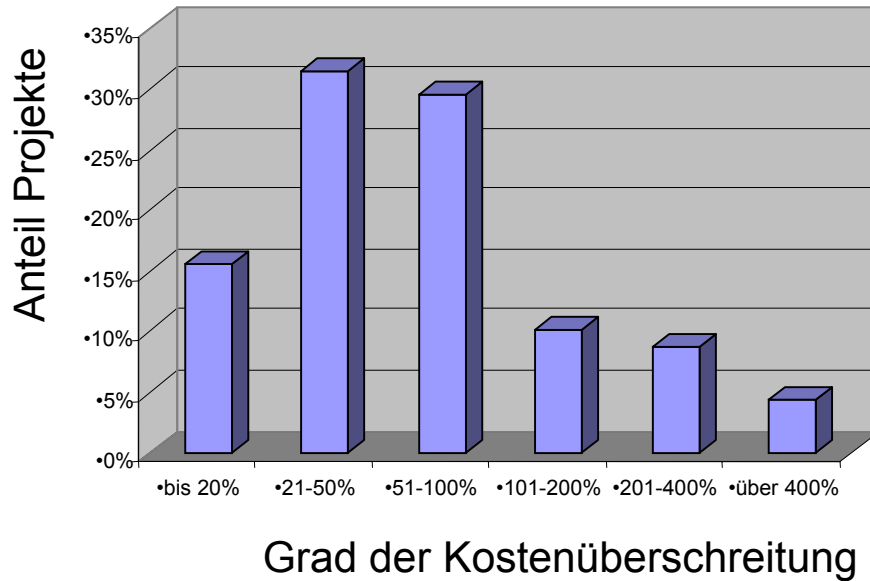
- 16,2% Erfolg, 52,7% Misserfolg, 31,1% Abbruch
- 53% der Projekte überschritten das Budget um mehr als 50%
- 4,4% überschritten das Budget sogar um mehr als 400%

⇒ **Erfolgsfaktoren:**

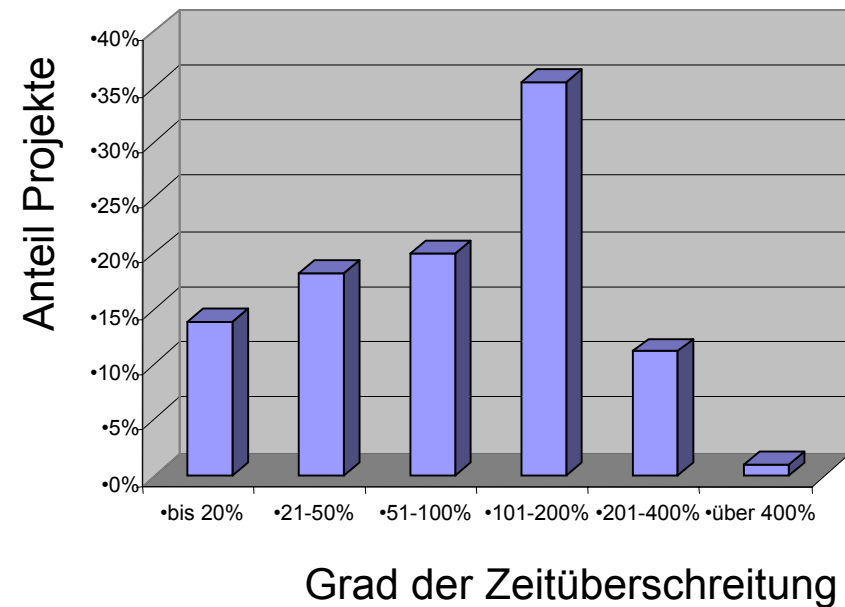
Benutzerbeteiligung, Managementsupport, klare Anforderungen, sorgfältige Planung

⇒ **Misserfolgsfaktoren:**

mangelnder Kundeninput, unvollständige Anforderungen und Spezifikationen, Änderung von Anforderungen



Chaos Report, The Standish Group



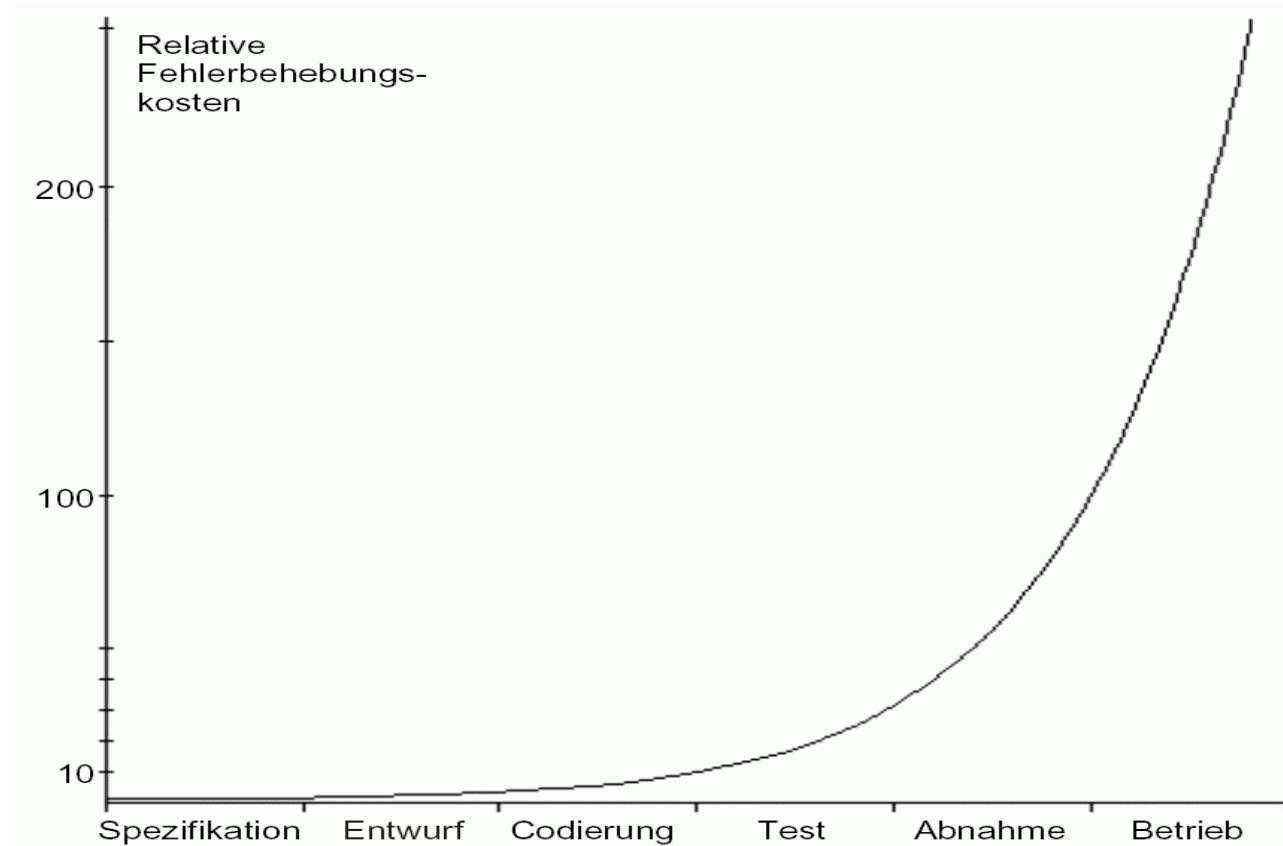
## ⇒ Probleme

- System von morgen nach Anforderungen von heute bauen
- Ungenaue Anforderungen
- Unterschiedliche Anforderungen unterschiedlicher Benutzer
- Dynamische Entstehung von Anforderungen während der Entwicklung
- Sprachliche Hürden

## ⇒ Resultat

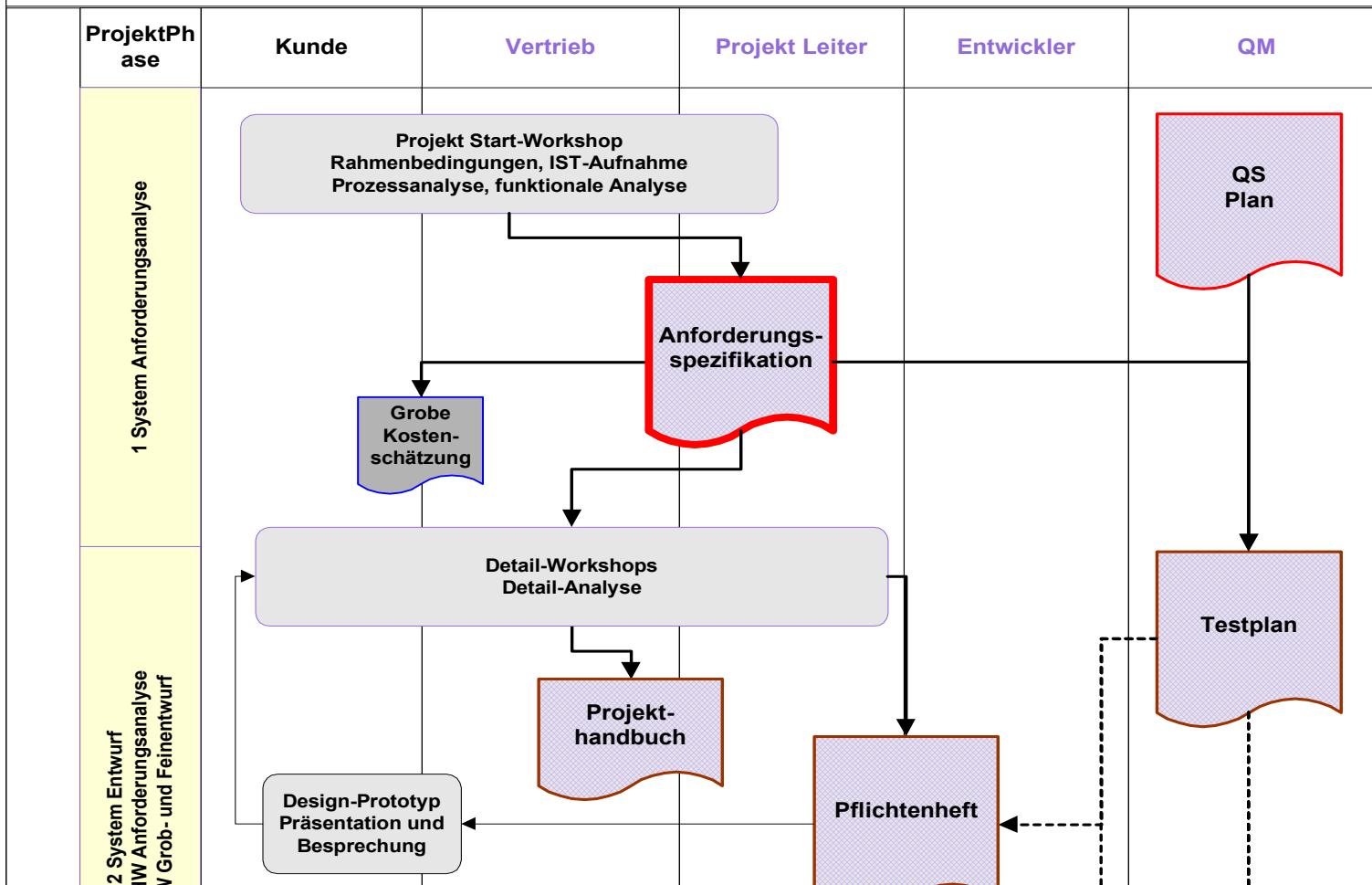
- Fehlerhafte Software
- Nicht bedarfs-gerechte Software

⇒ Die Fehlerbehebungskosten steigen exponentiell mit der Zeit

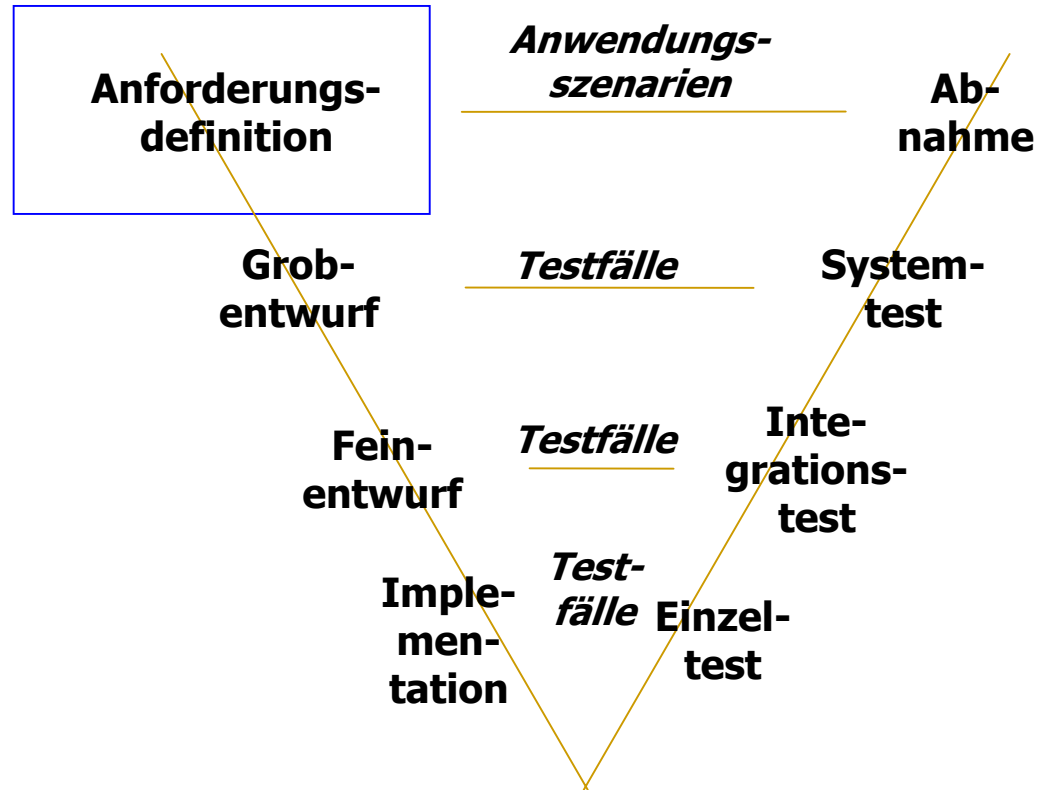
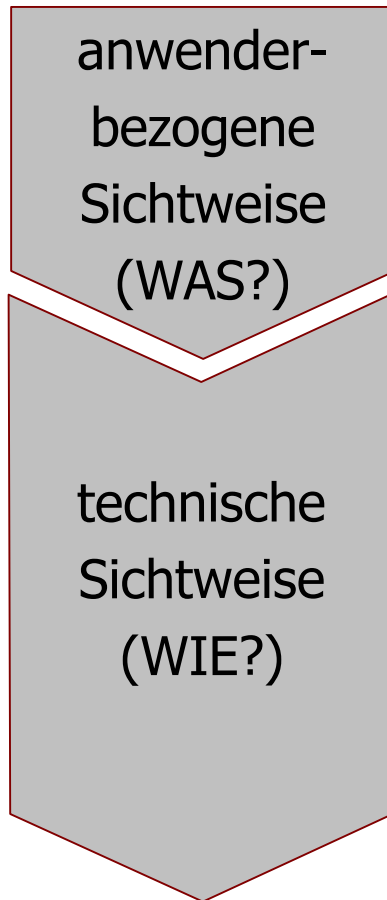


# Einordnung in IT-Projekt generell

## Projekt - Vorgehensmodell



# Einordnung in SW-Engineering-Prozess



# Unterschiedliche Sichten

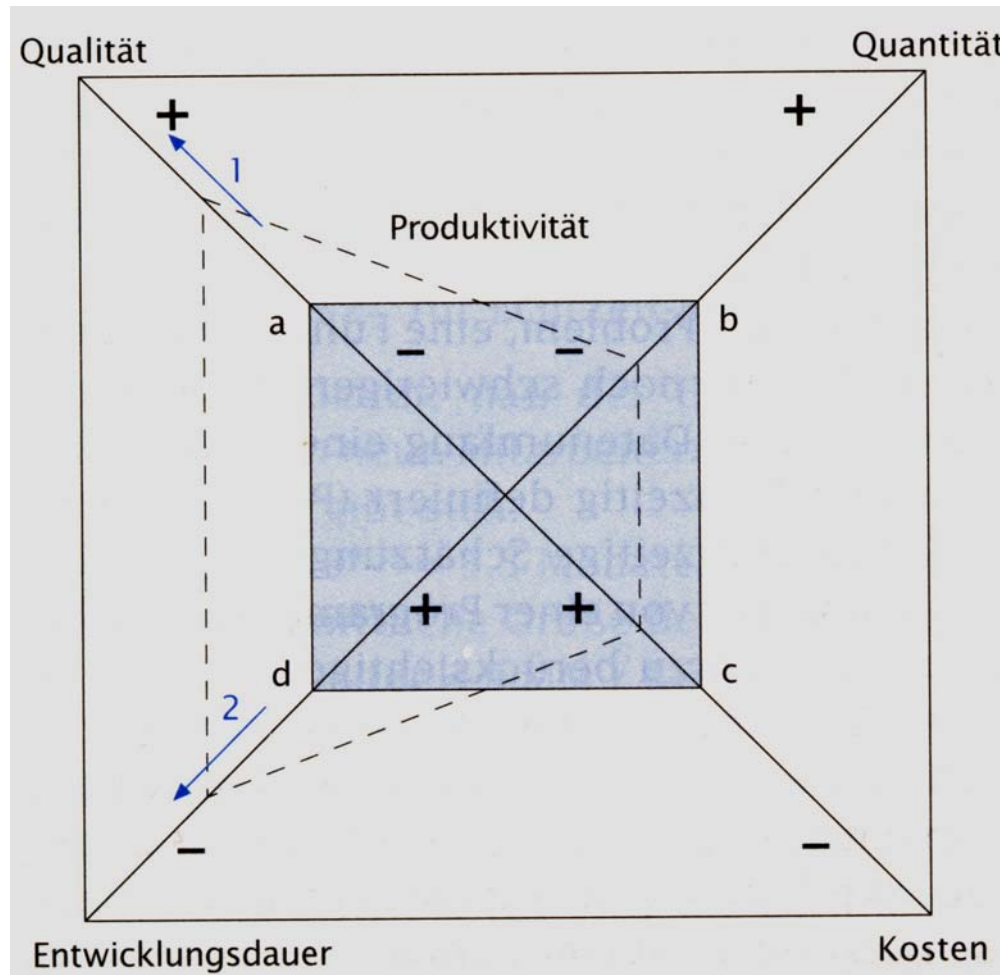
## ⇒ Verschiedene Benutzer / Interessensgruppen („Stakeholders“)

- Betriebsorganisation (BO) / Personalentwicklung
- Anwender / Endbenutzer
- Marketing (Corporate Identity)
- Vertragspartner
- Techniker (Installation, Wartung)

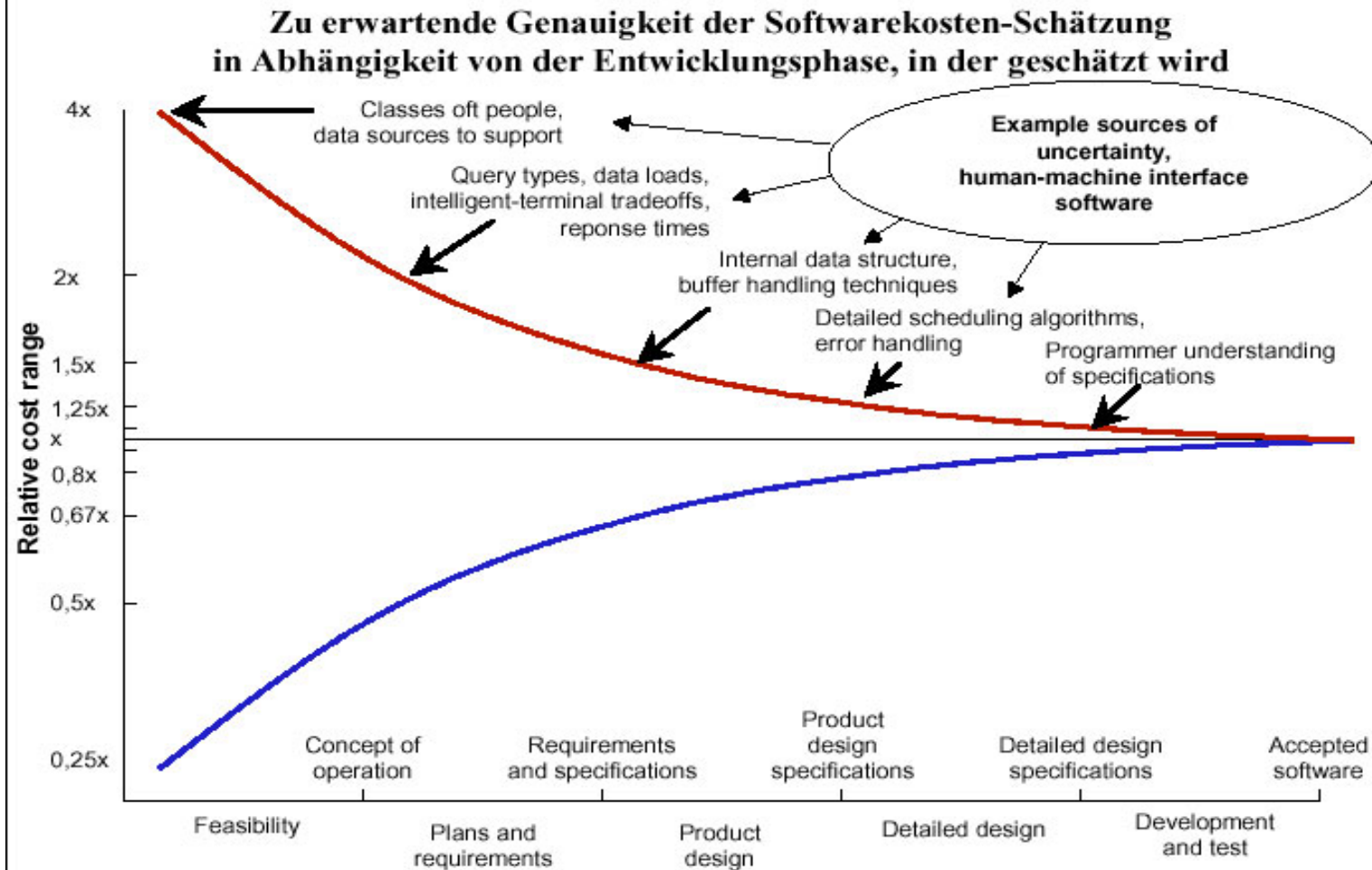
## ⇒ Mögliche Hürden

- Unterschiedliche Ziele innerhalb der Organisation
- Verschiedene Abteilungen
- Unterschiedlichste Kenntnisstände
- Häufig mangelnde interne Kommunikation
- Unternehmenshierarchie und Verantwortungen

# Planungsparameter und ihre Abhängigkeiten



# RE als Basis für Projektkostenschätzung?



Vgl.: Barry W. Boehm : „Software Engineering Economics“. Englewood Cliffs, N.J. 1981 S. 311

# RE als Basis für Projektkostenschätzung?

- ⇒ RE ist als Basis für Projektkostenschätzung nur bedingt geeignet
  - Die Bandbreite liegt zwischen  $\frac{1}{4}$  und dem 4-fachen der tatsächlichen Projektkosten
  - Anbieter müsste einen Sicherheitsaufschlag von bis zu 400% machen
- ⇒ Auftraggeber geht zumeist von niedrigeren Kosten aus und kalkuliert daher das Budget oft falsch
- ⇒ Vergleich zwischen Untersuchung von Standish Group und Analyse von Böhm zeigt, dass Projektverträge generell zu früh abgeschlossen werden.
- ⇒ Kosten und Zeit werden oft fixiert, bevor noch klar ist, was überhaupt realisiert werden soll!

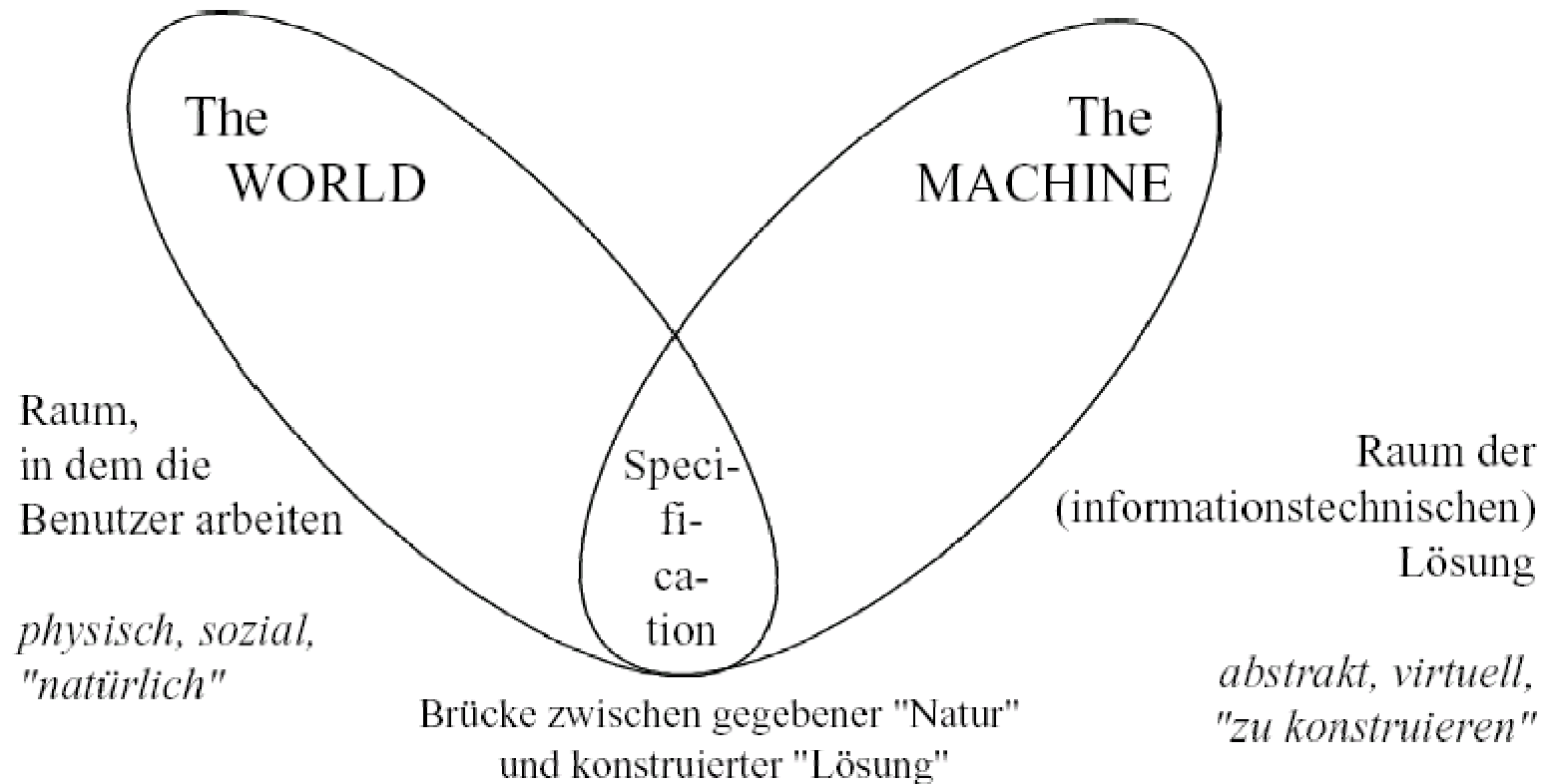
# Benutzerbeteiligung?

## Schlüssel zum Erfolg oder Hemmschuh der Entwicklung?

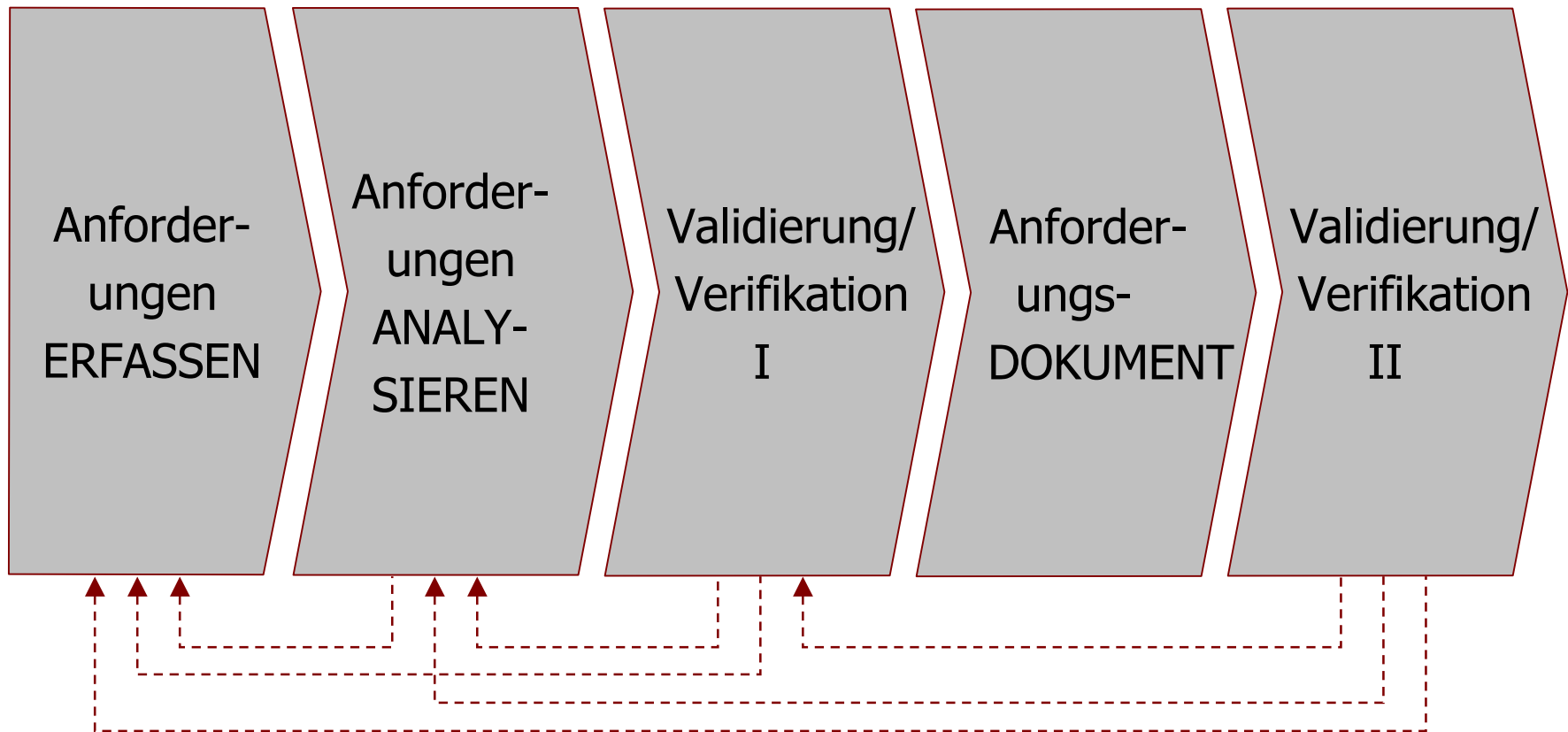
- ⇒ die **Beteiligung von** späteren oder potentiellen **Benutzern** als Mitarbeiter in SW-Projekten, um
  - **bessere Ausrichtung** der Software auf tatsächliche Bedürfnisse,
  - **höhere Akzeptanz** bei Benutzern und
  - **größere Effizienz** in den Projekten zu erreichen (Prototyping)
  
- ⇒ **Grund** für diese Erwartung ist die Annahme: Benutzer haben
  - **bessere Kenntnis** der Anwendung, der Geschäftsziele oder Anwendungsziele und –bedingungen und **Anforderungen**
  - **höhere Motivation zur Erreichung eines Nutzens** durch das Projekt oder Produkt
  - **höhere Motivation zu effizienter Entwicklung**

MODELLE:

[Michael Jackson, ICSE 95]



# Requirements-Engineering-Prozess



# Anforderungen ERFASSEN

## ⇒ Quellen lokalisieren

- Anwender, Management, ...
- Rolle im Unternehmen / in Bezug auf System
- Rechte / Kompetenzen
- Wer gibt Randbedingungen vor?

## ⇒ Vorhandene Dokumentation einbeziehen

## ⇒ Randbedingungen

- Firmen-Grundsätze
- Hard- und Softwarevoraussetzungen
- Einsatz- / Betriebsbedingungen
- Absehbare Veränderungen

# Anforderungen ERFASSEN

⇒ „Wer will weshalb was“

⇒ Aspekte:

- Funktion / Qualität
- Leistung / Quantifizierung
- Randbedingungen
- Priorisierung

⇒ Vom Analysten/Programmierer/...

- messbare Spezifikation
- ggf. Prototyp oder UserStory



# Anforderungen **ERFASSEN** - funktionale

## ⇒ **Daten**

- Struktur
- Verwendung
- Erzeugung
- Speicherung
- Übertragung
- ...

## ⇒ **Verhalten**

- Sichtbares dynamisches Systemverhalten
- Zusammenspiel der Funktionen

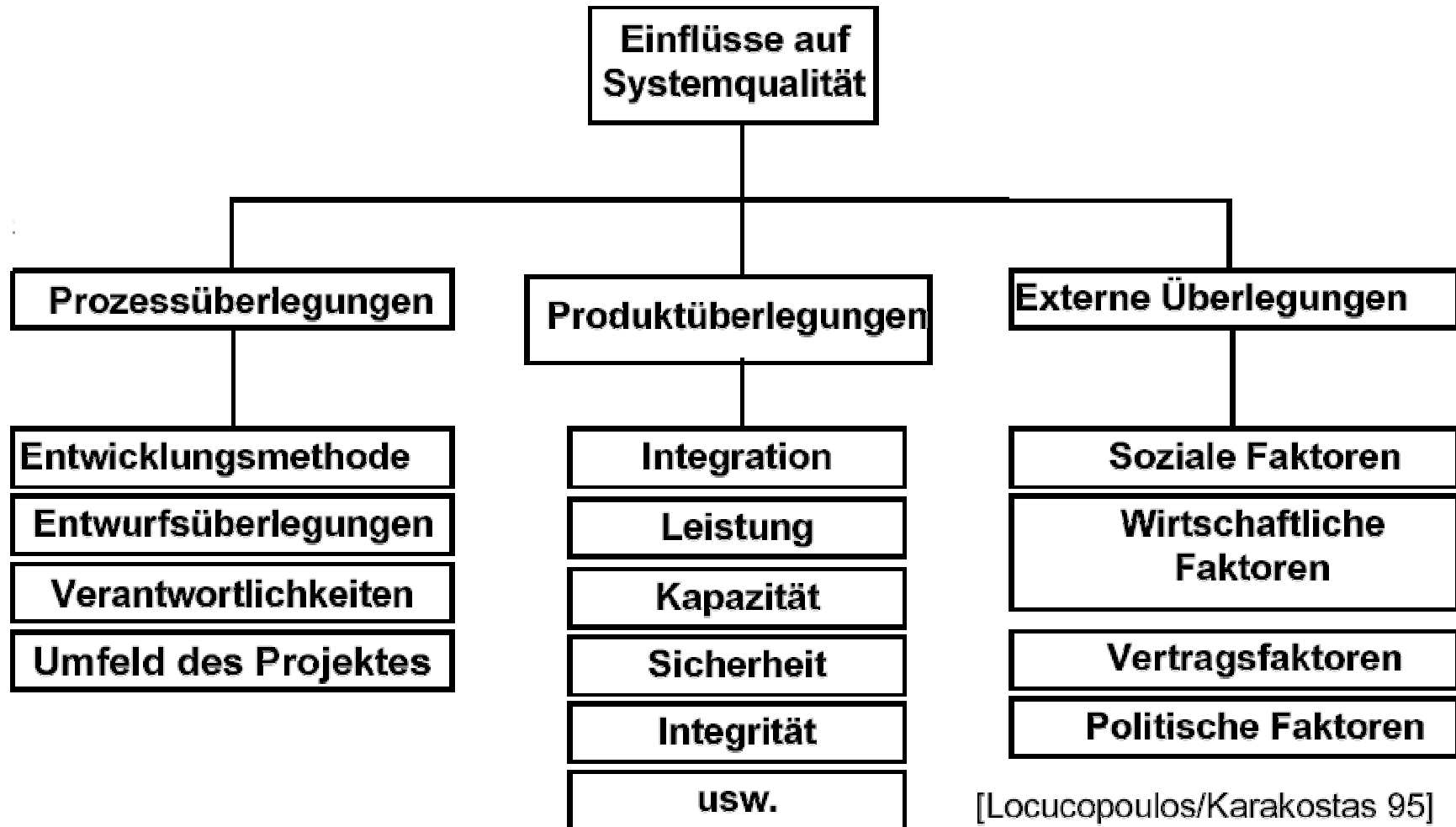
## ⇒ **Funktionen**

- Ausgabe
- Verarbeitung
- Eingabe von Daten

## ⇒ **Fehler**

- Normalfall
- Fehlerfälle

# Anforderungen - nicht funktionale



# Anforderungen– nicht funktionale

## ⇒ Leistung

- Datenmengen (durchschnittlich / im Extremfall)
- Verarbeitungs- /Reaktionsgeschwindigkeit (durchschn. / Extremfall)
- Verarbeitungszeiten und -intervalle
- wo immer möglich: messbare Angaben!

## ⇒ Qualität

- geforderte Qualitäten (z.B. Benutzerfreundlichkeit, Zuverlässigkeit)

## ⇒ Randbedingungen

- einzuhaltende / zu verwendende Schnittstellen
- Normen und Gesetze
- Datenschutz, Datensicherung
- Explizite Vorgaben des Auftraggebers.

# Anforderungen ERFASSEN – Grundprinzip

⇒ There is no complete and well defined set of Requirements waiting to be discovered!



⇒ Requirements-Engineering ist meist einfach nur harte Arbeit!

# Anforderungen ERFASSEN - Probleme

- ⇒ **Unterschiedliche Vertreter des Kunden haben unterschiedliche Vorstellungen über das zu spezifizierende System.**  
Häufig auch Auffassungen und Begriffsbildung im Anwendungsbereich nicht einheitlich.  
Requirements-Engineering beinhaltet daher auch Konsensbildung zwischen divergierenden Vorstellungen der beteiligten Personen.
- ⇒ **Die Kundenvertreter haben zwar eine Vorstellung, was sie wollen, aber sie können ihre Vorstellung nicht formulieren.**  
Manchmal „lösen“ sie dieses Problem, indem sie anstelle ihrer eigentlichen Anforderungen bestehende Lösungen mit vergleichbaren Eigenschaften beschreiben.
- ⇒ **Die Kundenvertreter wissen oft nicht oder nur sehr vage, was sie eigentlich wollen.**

# Techniken zur Anforderungserfassung

- ⇒ **Begriffe klären und Glossar mit Definitionen der wichtigen Begriffe des Anwendungsbereichs erstellen.**  
Begriffliche Grundlage schaffen, dass alle das Gleiche meinen, wenn sie reden.
- ⇒ **Soll-Prozeßabläufe untersuchen.**  
Feststellen, welche äußeren Ereignisse auf das zu spezifizierende System einwirken können und wie das System auf diese Ereignisse reagieren soll.
- ⇒ **Anwendungsszenarien bilden und durchspielen.**  
Alle Interaktionen der Umgebung (Menschen, Schnittstellen, ...) mit dem zu spezifizierenden System in Form von Szenarien aufschreiben und ‚durchspielen‘. Szenarien eignen sich gut zur Gewinnung und Diskussion von Anforderungen.
- ⇒ **Den Anwendungsbereich modellieren.**  
Feststellen relevanter Objekte für das geplante System (diejenigen, über die das System Information speichern muss, damit es seine Aufgabe erfüllen kann.)  
Herausfinden, welche Eigenschaften und welche Beziehungen dem zu spezifizierenden System bekannt sein müssen.

# Regeln zur Anforderungserfassung

- ⇒ IST-Zustandserfassung möglichst gering halten
  - Nur die als Basis für das neue System notwendigen Informationen
- ⇒ Nachteile einer zu detaillierten IST-Zustandserfassung:
  - Gefahr des ‚alten Weins in neuen Schläuchen‘
  - Unnötige Zusatzkosten
  - Innovative neue Wege werden ev. nicht erkannt
- ⇒ Generell: Erfassung und Dokumentation ist kein Selbstzweck
  - Daher nur so viel, wie unbedingt notwendig erfassen
  - Dies dafür jedoch sorgfältig und konsequent
- ⇒ Jedoch: In der Regel wird zu wenig und nicht zu viel erfasst

# Anforderungen ANALYSIEREN

- ⇒ Klassifizierung & scoping
- ⇒ Interaktionen (matrixweise Darstellung):
  - Abhängigkeiten
  - Überlappungen
  - Widersprüche
- ⇒ Prioritäten-Zuweisung
- ⇒ Risiko-Analyse
- ⇒ Verifikation möglich (unit tests)?
- ⇒ Unterschiedliche Darstellungen (graphisch und textuell)

# Klassifikation von Anforderungen

## ⇒ **Muß – Anforderungen**

sind unverzichtbar und müssen in jedem Fall erfüllt werden

## ⇒ **Soll - Anforderungen**

sollten erfüllt werden, sind aber bei zu hohen Kosten verzichtbar

## ⇒ **Wunsch – Anforderungen**

werden nur erfüllt, wenn dies mit vertretbaren Kosten möglich ist.

Definition nach Pomberger

⇒ Validierung durch alle Beteiligten!

⇒ Überprüfung von

- Adäquatheit (das beschreiben, was der Kunde will bzw. braucht)
- Vollständigkeit (alles beschreiben, was der Kunde will bzw. braucht)
- Widerspruchsfreiheit (sonst ist die Spezifikation nicht realisierbar)
- Verständlichkeit (für den Kunden UND für die Informatiker)
- Eindeutigkeit (damit Fehler durch Fehlinterpretationen vermieden werden)

⇒ Mögliche Fehlerquellen

- Fehlende oder widersprüchliche Information
- unrealistisch Anforderung
- ungenaue Formulierung

# Anforderungs-DOKUMENT erstellen

## ⇒ Zielsetzung:

- Allgemeine System-Referenz
- Kommunikation zwischen Anwendern, Entscheidungsträgern und Programmierern
- Pflichtenheft / User Stories daraus ableiten

## ⇒ Standard-Layout

## ⇒ Darstellung:

- Natürlichsprachlich textuell
- Ergänzend formale Spezifikations-Sprachen (Grammatiken, reguläre Ausdrücke)
- Graphisch

- ⇒ Authorisierung durch jeden Beteiligten
- ⇒ Abnahme durch Vertragspartner/Entscheidungsträger
- ⇒ insbesondere wichtig bei präskriptivem Vorgehen (Vertragscharakter)



- ⇒ Einleitung
- ⇒ Übersicht des Systems
- ⇒ Einzelne Requirements
- ⇒ Resultierende System-Spezifikationen
- ⇒ Rahmenbedingungen
- ⇒ Index

## ⇒ Einleitung

- Ziel und Sinn des Dokuments
- Version des Dokuments, Historie
- Anwendungsbereich des Systems, Gültigkeitsbereich des Dokuments
- Definitionen & Abkürzungen:
  - Technische Termini
  - Anwendungsbereich-spezifische Ausdrücke
  - Sprachliche Definitionen (sollen, können, müssen, ...)
- Literaturangaben und Verweise
- Übersicht des restlichen Dokuments

## ⇒ Übersicht des Systems

- Motivation
- Ziel & erwartete Anwendung (Hauptfunktionen)
- Beschreibung der künftigen Anwender (Nutzerprofile)
- Allgemeine Einschränkungen (insb. externe), Risiken
- Voraussetzungen und Abhängigkeiten

## ⇒ Einzelne Requirements

- Anforderungen:
  - Funktional (z.B. Erfassungsmaske mit Name, Adresse, Datum)
  - non-funktional (z.B. Reaktionszeit < 5 Sek., Ergonomienorm)
  - Interface (sowohl UI als auch API)
- Vollständige Beschreibung mit
  - Begründung
  - Quelle
  - Priorität
  - Klassifizierung
  - messbare Spezifikation
  - ggf. Prototyp oder UserStory
- Zusammenfassung (textuell und graphisch)

## ⇒ Resultierende System-Spezifikationen

- System models (Datenfluss, Geschäftsprozesse)
- Detaillierte Darstellung der Funktionalität
- (Unternehmens-) Umgebung
- Systemweite Anforderungen  
(Verfügbarkeit, Sicherheit, Kostenrahmen)
- Data Dictionary
- Bezug Spezifikation - Requirement

## ⇒ Rahmenbedingungen

## ⇒ Index

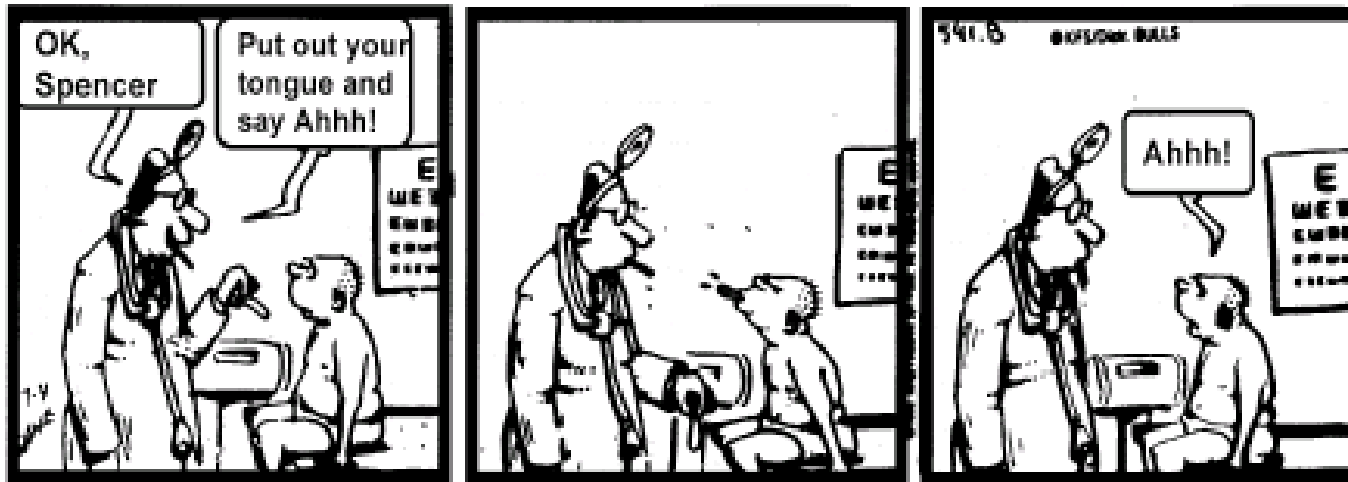
# Stil-Ratschläge für Spezifikationen

- ⇒ Kurze Sätze, da das menschliche Kurzzeitgedächtnis begrenzt ist
- ⇒ Nur eine Anforderung pro Satz formulieren, 'und' vermeiden
- ⇒ Jargon vermeiden, Abkürzungen sparsam verwenden
- ⇒ Kurze Absätze (max. ca. 7 Sätze)
- ⇒ Listen benutzen, anstatt Aufzählungen in einem Satz
- ⇒ Terminologie konsistent benutzen;  
Wortwiederholungen sind erwünscht!
- ⇒ Verschachtelte logische Zusammenhänge vermeiden
  - Wenn X oder Y und in diesem Fall Z gegeben ist, jedoch nicht wenn ...)
  - besser Pseudocode, Entscheidungstabellen, ... verwenden.

# Stil-Ratgeber für Spezifikationen

## ⇒ Mehrdeutigkeiten vermeiden

- Mehrdeutigkeiten sind ein häufig auftretendes Problem
- werden häufig nicht als solche erkannt



## ⇒ Aufgaben:

- Verwaltung von Anforderungen (Abhängigkeiten, Tracing, Fortschritt)
- Automatisierte Erstellung von Anforderungsdokumenten
- Versionsverwaltung

## ⇒ Software:

- Eigenentwicklungen (DBMS + Frontend)
- Grafik-/Diagramm-Tools (Micrografx, Visio, ...)
- CASE-Tools, insb. UML (z.B. Rational Rose)
- Spezielle RE-Software (z.B. RequisitePro)
- Sonstige Werkzeuge (z.B. Aris-Toolset, ...).

## ⇒ Vorteile:

- Einfache, strukturierte Verwaltung
- Leichte Änderbarkeit
- gewisse Plausibilitätsprüfungen möglich

## ⇒ Nachteile

- Kreativität leidet
- Eigene Plausibilitäts-Prüfungen werden vernachlässigt
- Man verlässt sich zu sehr auf das System

⇒ Das unterstützende Software-System ist nur so gut wie der, der es bedient!

# RE-Varianten: Präskriptiv vs. Explorativ

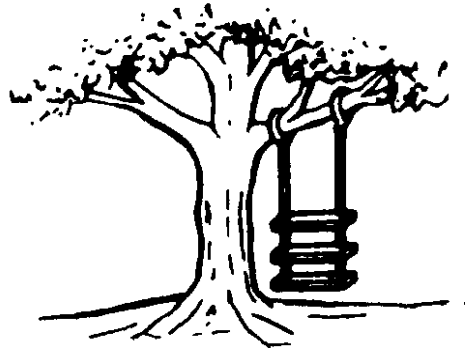
## ⇒ Präskriptiv

- Anforderungsanalyse / Pflichtenheft mit Vertragscharakter
- i.d.R. linearer RE-Prozess
- Alle Anforderungen sind zu erfüllen
- Typische ‚Pauschalprojekte‘

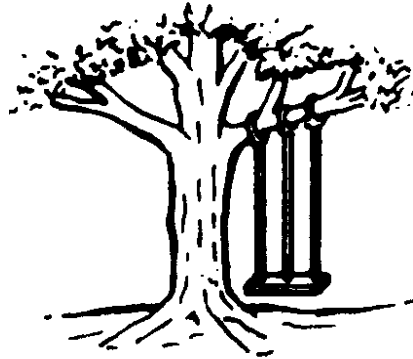
## ⇒ Explorativ

- Beispiel: eXtreme Programming
- iterativer, inkrementeller RE-Prototyp-Prozess
- hohe Integration aller Betroffenen
- starke Priorisierung der Aufgaben
- Typische ‚Nach Aufwand – Projekte‘.

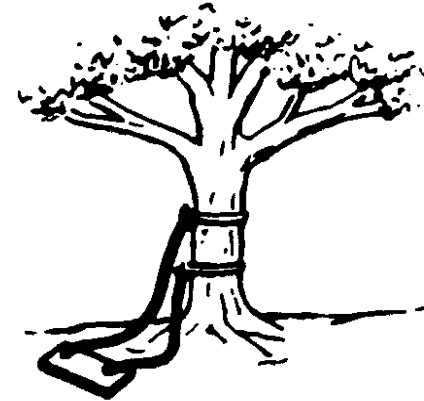
# Einfach zum Nachdenken



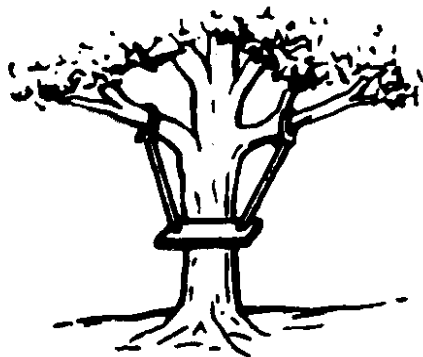
**AS PROPOSED BY THE  
PROJECT SPONSOR**



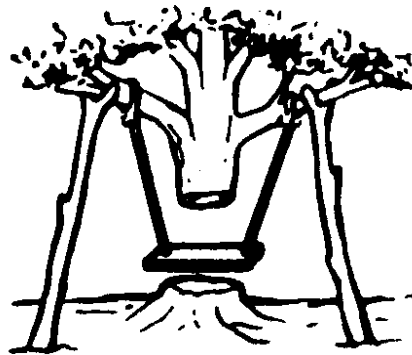
**AS SPECIFIED IN THE  
PROJECT REQUEST**



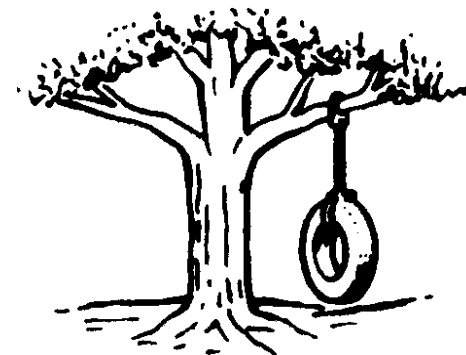
**AS DESIGNED BY THE  
SENIOR ANALYST**



**AS PRODUCED BY  
THE PROGRAMMERS**



**AS INSTALLED AT  
THE USER'S SITE**



**WHAT THE USER WANTED**