Ausgewählte Kapitel der Softwaretechnik

Dr. Lutz Prechelt
prechelt@ira.uka.de
Tel. 608-4068
Raum 370 Informatikbau

13. April 1999

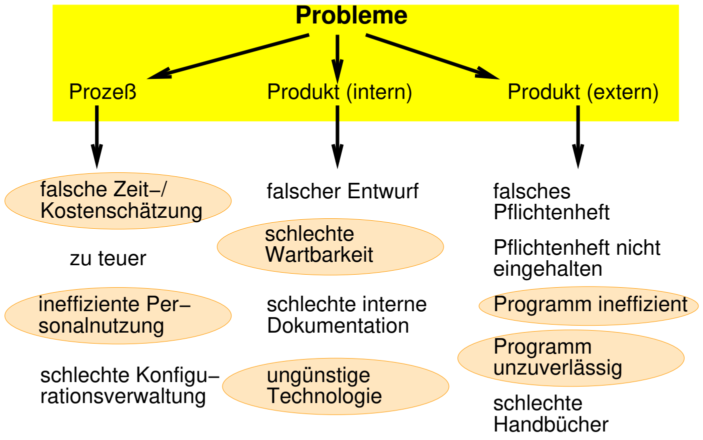
http://wwwipd.ira.uka.de/~prechelt/swt2

Inhalt:

Aufbau jedes Abschnitts:

- Einordnung und Ziele
- Ansätze
- Ergebnisse
- Probleme und Hindernisse
- Techniken

1 Messen und Maße



1.1 Einordnung und Ziele

- **Messen**: Anwendung eines Maßes auf Objekt,
  also Prozeß.
- **Maß**: Abbildung, nämlich objektive Zuweisung Symbol an Objekt zur Charakterisierung von Attribut.
- **Metrik**: mathematischer Begriff, Abstandsmaß.
  SWT-Maße eher Normen ähnlich, sind aber meist keine.
- Ziele von Messen:
  - Beschreibung. (Verständnis).
  - Beurteilung. (Vergleich, Verbesserungen).
  - Vorhersage. (Planung, Verbesserungen).

1.2 Ansätze

1.2.1 Skalen

- Nominalskala: qualitativ.
- Ordinalskala: qualitativ, geordnet (Median).
- Intervallskala: quantitativ (Differenzbildung, Mittelwert).
- Verhältnisskala: mit Nullpunkt (alle Operationen).
- Absolutskala: Anzahl.

Eigenschaft des Maßes, nicht der Größe!

Oftmals unsinnige Aussagen:

- „20 Prozent Verbesserung der Qualität“.
- „Diese Kunden sind doppelt so zufrieden wie jene“.
- „Heute doppelt so warm wie gestern“.

Wir nehmen gerne voreilig eine Verhältnisskala an.

Testfragen:

- „Was ist Qualität Null?“.
- „*Wie* zufrieden sind sie denn?“.

20 Grad Celsius gegenüber 10 Grad Celsius gestern
ist gleich
68 Grad Fahrenheit gegenüber 50 Grad Fahrenheit.
Die Zunahme ist 3.5 Prozent:
von 283 Kelvin auf 293 Kelvin!

1.2.2 Anforderungen an Maße

- Wohldefiniertheit. Exakte, objektive Definition.
- Aussagekraft. Größe von Interesse.
- Sinnhaltigkeit. Qualitativ korrekte Abbildung. („Repräsentationseigenschaft“: Erhaltung von Relationen)
- Genauigkeit. Zuverlässig genau genug bestimmbar.
- Berechenbarkeit. Algorithmische Form.
- Praktische Durchführbarkeit. Eingaben verfügbar.

Nehmen, was man kriegen kann, z.B. Zuverlässigkeit (spät), Komplexität (geringe Aussagekraft).
Hauptkriterium: Zweckmäßigkeit.

1.2.3 Versuch einer Axiomatisierung

Axiomatisierung von Komplexitätsmaßen (Weyuker):
$P, Q, R \qquad |P| \qquad P = Q \qquad P \equiv Q \qquad P; Q$

1. $\exists P, Q : |P| \neq |Q|$
   „Nicht konstant“.
2. $\forall c \geq 0 : |\{P \mid |P| = c\}| < \infty$
   „Fein genug“: Programme in genügend viele Komplexitätsklassen unterteilen.
3. $\exists P, Q : P \neq Q \wedge |P| = |Q|$
   „Nicht zu fein“: verschieden und dennoch gleiche Komplexität.
4. $\exists P, Q : P \equiv Q \wedge |P| \neq |Q|$
   „Implementationsabhängig“: verschiedene Komplexität trotz gleicher Funktionalität.

5. $\forall P, Q : |P| \leq |P; Q| \wedge |Q| \leq |P; Q|$
   „Monotonität“: Zufügen verringert Komplexität nicht.
6. a.) $\exists P, Q, R : |P| = |Q| \wedge |P; R| \neq |Q; R|$
   b.) $\exists P, Q, R : |P| = |Q| \wedge |R; P| \neq |R; Q|$
   „Interaktion“: Programmstücke interagieren verschieden komplex mit Kontext.
7. $\exists Q, \mathit{Permutation} : P = \mathit{Permutation}(Q) \wedge |P| \neq |Q|$
   „Anordnungsabhängigkeit“: Anweisungsreihenfolge relevant.
8. $\forall Q, \mathit{Renaming} : P = \mathit{Renaming}(Q) \Rightarrow |P| = |Q|$
   „Namensunabhängigkeit“: Benennung nicht relevant.
9. $\exists P, Q : |P| + |Q| < |P; Q|$
   „Mehr als Summe der Teile“: Programm evtl. komplexer als Summe seiner Teile.

Bemerkungen:

- Nicht hinreichend. Unsinnsmaß kann alles erfüllen.
- Nicht notwendig. Z.B. Datenflußkomplexität verletzt Monotonität.
- Evtl. Maß nützlich, nur nicht als Komplexitätsmaß.
- Manche Forderungen dubios, z.B. Namensunabhängigkeit.
- Dennoch nützliche Leitlinie: Begründe jede Verletzung.

1.2.4 Goal, Question, Metric

Basili und Weiss 1984:

- **Definiere Ziele.**
  z.B. Fehler verringern.
  z.B. Wartbarkeit verbessern.
- **Finde Fragen.**
  z.B. Welche/wieviele Fehler machen wir? Wann?
  z.B. Welche Programme sind schlecht wartbar?
- **Definiere Maße.**
  z.B. Fehlerprotokolle: Art, Zeit, Arbeitsschritt, Grund
  Maße erfinden (intuitiv, induktiv), validieren.

1.2.5 Auflösung, Genauigkeit und Fehler

- Auflösung: Anzahl oder Abstand von Stufen.
  Beispiel: Ganze Minuten versus ganze Stunden.
- (Un)Genauigkeit: erwartete oder höchste Abweichung.
  Beispiel: $Zeitaufwand = \sum_{t \in Tage} 8 \cdot Leute(t)$
- Systematischer und statistischer Fehler (z.B. Stoppuhr).
- Genauigkeitsangabe: Fehlerverteilung, Konfidenzbereich.
- Genauigkeit durch Auflösung beschränkt.
- Auflösung suggeriert Genauigkeit.
- Genauigkeit oft schwer anzugeben.

1.2.6 Arten von Softwaremaßen

- Externe Produktmaße.
  Wartbarkeit, Zuverlässigkeit, Benutzbarkeit, Effizienz, etc.
- Interne Produktmaße.
  Größe, Modularität, Komplexität, etc.
- Prozeßmaße.
  Produktivität, Kostenverteilung, Dauer, etc.
- Ressourcenmaße.
  Produktivität und Fehlerraten eines einzelnen, Auslastung, etc.

- Qualitativ oder quantitativ.
  z.B. „Strukturierungsniveau“: mitGoto, strukturiert, modular, objekt-orientiert.
  Auch dann nicht quantitativ, wenn mit 0, 1, 2, 3 kodiert!
  Quantitativ: Intervall-, Verhältnis-, Absolutskala.
- Automatisierbar oder nicht automatisierbar.
  z.B. LOC vs. Fehlerart.
- Direkt vs. indirekt.

1.3 Ergebnisse

1.3.1 PSP

Positives Beispiel.

- Simple Maße zur Verbesserung von Planung und Qualität.
- Direkt: LOC, 10 Fehlerklassen, 6 Prozeßphasen, Zeiterfassung.
- Indirekt: Fehlerdichten, Fehlerentfernungsquoten und -raten, Produktivität, Effektivität von Inspektionen.
- Maße wohldefiniert, aussagekräftig, Grundlage von Verbesserungen

1.3.2 o-o

Beispiel zum Nachdenken.

„This was a project in which we had good domain experience and about six years of metrics, both team productivity and other analogous software of similar scope and functionality. The difference with this project was that we switched from a functional design methodology to o-o.

First the good news: the overall team productivity (SLOC/person-month) was almost three times our previous rate.

Now for the bad news: the delivered SLOC was almost three times greater than estimated, based on the metrics from our previous projects."

1.3.3 GQM

Erfahrungen mit Meßvorhaben.

1. Verstehe Deinen Anwendungsbereich (Arbeitsumfeld und Projektgegenstand).
2. Validiere Daten rechtzeitig.
3. Minimiere Arbeitsaufwand für Datenerfassung.
4. Berücksichtige bei Analyse die Umgebung.
5. Unterschätze nicht den Aufwand.
6. Daten nicht gegen Leute verwenden.
7. Eventuell Hawthorne-Effekt.

1.4 Probleme und Hindernisse

- Mangel an Wohldefiniertheit.
  z.B. LOC: Layout? Leerzeilen? Fastleerzeilen? Kommentarzeilen? Trennerzeilen?
- Mangel an Übertragbarkeit.
  z.B. Produktivität bei WWW-Browser vs. bei Fabriksteuerung.
- Fehlinterpretation.
  z.B. Programmzeilen/Zeit = „Produktivität".
- Mangel an aussagekräftigen Maßen.
  z.B. für Komplexität z. Vorhersage von Wartungsaufwand.

- Mangel an Eingabemeßwerten.
  - hoher Erfassungsaufwand.
    z.B. Häufigkeit/Art mündlicher Anfragen bei Kollegen.
  - zu spät verfügbar.
    z.B. Wartungsaufwand
  - prinzipiell nicht meßbar.
    z.B. Anzahl ausgelieferter Fehler

1.5 Die Klassiker: Lines of Code, Fehleranzahl, Personentage

1.5.1 Beschreibung und Zweck

- LOC: Größe, meist für Quellcode.
  Zählweise? Schätzbar und Eingabe für viele Schätzungen.
- Fehleranzahl: Statische Defekte oder dynamische Versagen.
  Basisgröße für viele Qualitätsmaße von Prozeß und Produkt.
- Personentage: Aufwand an Arbeitszeit.
  Basisgröße für Maße von Kosten, Zeitbedarf, Produktivität etc.

1.5.2 Probleme und Grenzen

- Definitionen in der Praxis meist ungenau.
  z.B. Was ist ein Fehler? Personentage pauschal oder feinkörnig?
- Übertragbarkeit gering, da viele Faktoren wichtig.
- LOC: nur pro Programmierer/Sprache/Bereich genau. Sonst ungenau oder gar irreführend.
- Fehleranzahl: nicht reproduzierbar, da subjektive Klassifikation. Nützlichkeit sinkt mit steigender Objektivität.
- Personentage: Schlechte Übertragbarkeit, wenn Personen in mehreren verwandten Projekten zugleich oder bei SW-Wiederverwendung. Stets schlechte Übertragbarkeit zwischen Organisationen.

1.5.3 Wichtigkeit

Fast unersetzliche Basismaße.

- LOC: Mißt Arbeitsmenge, da LOC/Tag im Prinzip konstant bei festen Bedingungen. Hinweise auf Schlamperei oder organisatorische Verhinderung von Arbeit.
- Fehleranzahl: Maßgeblich für Produktivität und Qualität. Basis wichtiger Vorhersagen (z.B. kritische Module) und Prozeßindikatoren (wo klemmts?).
- Personentage: zentrale Aspekte „Kosten“ und „Zeit bis Markteintritt“ basieren darauf.

1.5.4 Fazit

Genaue Definition nötig, aber dann sehr nützlich:

- LOC: Bei konstanten Umgebungsbedingungen konsistenter Normierungsfaktor für viele Prozeß, Produkt und Ressourcenmaße. Nützlich, weil automatisierbar und im Voraus schätzbar.
- Fehleranzahl: Begrenzte Genauigkeit aber extrem hoher Nutzwert der Information, besonders wenn Fehlerklassifizierung. Nützlich, weil Fehler stark auf Qualität und Produktivität einwirken.
- Personentage: Begrenzte Genauigkeit, aber unersetzlich zur Beschreibung der Kosten. Auf Umweg schätzbar, deshalb wichtig für Planung.

1.5.5 Beispiel: Definition von LOC

- Richtlinien:
  - Jede Anweisung auf mind. eine Zeile für sich.
  - Wenn zu lang, sinnvoll in möglichst wenige Stücke zerlegen.
  - Jede Variablendeklaration auf getrennter Zeile.
    Dito Elemente von Verbunden, Parameter von Prozeduren.
  - Deklaration Verbundtyp mit n Elementen oder Prozedur mit n Parametern umfaßt also $n + 1$ Zeilen.
  - For-, While-, Repeat- etc. Anweisung ist zwei Zeilen länger als ihr Rumpf (Kopfzeile und END-Zeile).
  - If-Then-Else-Anweisung ist vier Zeilen länger als ihr Rumpf, bei fehlendem Else-Teil drei Zeilen.

- Zählweise: Jede physische Zeile zählt als 1, ausgenommen
  - Leere Zeilen.
  - Zeilen, die ausschließlich Kommentar oder…
  - …Blocktrenner enthalten (BEGIN, END).

Je nach Programmiersprache ferner Richtlinien für Case, Elseif-Ketten, Import-/Exportlisten, Schabloneninstanziierungen etc.

1.6 Vorhersage und Validation

- Meßziel meist Vorhersage.
- Zu erhalten durch Vorhersagesystem.
- Vorhersagesystem:
  - mathematisches Modell (abstrakte Repräsentation)
  - Prozeduren z. Bestimmung ungekannter Parameter
  - Prozeduren z. Interpretation der Resultate

1.6.1 Beispiel: Vorhersagesystem

- Vorhersageziel: Personenzeitaufwand f. Erstellung SW-Komponente k mit bekannter Größe.
- Modell: $T = b + G(k)/p$
  - Personenzeitaufwand $T(k)$
  - konstanter Basiszeitaufwand b
  - Komponentengröße $G(k)$
  - Programmiergeschwindigkeit p (gemessen in Größe pro Zeit).
- Modellannahmen:
  Es gibt Arbeiten, deren Dauer von Größe unabhängig und Herstellungsgeschwindigkeit ist von Größe unabhängig und beides ist von Art unabhängig (innerhalb Gültigkeitsbereich).

- Benötigt:
  Prozeduren zur Bestimmung von $b, G(k)$ und p.
- z.B. Bestimmung $G(k)$ durch LOC.
- z.B. Bestimmung b und p durch lineare Regression aus historischen Daten (in Minuten).

$G(k)$	$T(k)$	
78	33	
565	207	
330	191	
187	93	
1160	524	Summe
290	131	Durchschnitt

Berechne Regressionsgrade nach Regressionsformeln

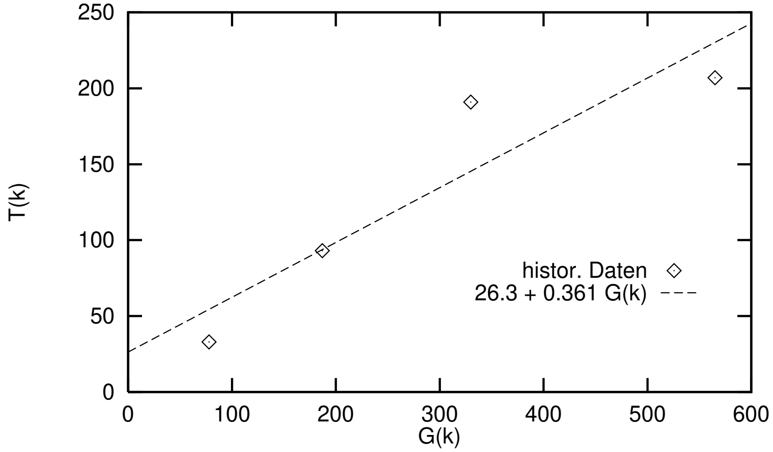
$$\beta_1 = \frac{\sum_{i=1}^{n} x_i y_i - n\bar{x}\bar{y}}{\sum_{i=1}^{n} x_i^2 - n\bar{x}^2} \qquad \beta_0 = \bar{y} - \beta_1 \bar{x}$$

Wir erhalten

$$p = \frac{199950 - 4 \cdot 290 \cdot 131}{469178 - 4 \cdot 290} = \frac{47990}{132778} = 0.361$$

$$b = 131 - 0.361 \cdot 290 = 26.3$$

Damit können wir Vorhersagen treffen in der Form $T(k) = 26.3 + 0.361 \cdot G(k)$.



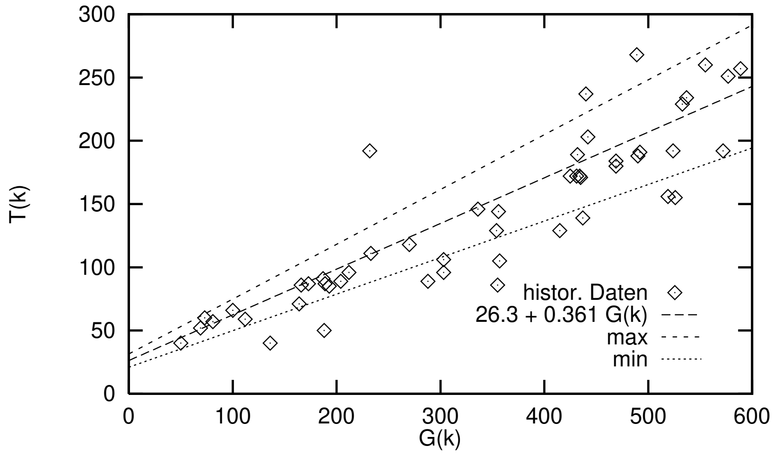
1.6.2 Validation

- „Ist Vorhersagesystem für seinen Zweck brauchbar?“
- Fragen:
  Wie genau ist es? Mit welcher Zuverlässigkeit?
- z.B. Abweichungen von mehr als 20% in maximal 10% der Fälle.
- Beurteilung am Zweck orientiert.
  z.B. bei Zuverlässigkeit: Was empfindet *Benutzer* als Fehler?
- Kern der Validation oft: Gültigkeit der Maße.
  Spiegeln Maße intuitiv richtig das zu messende wider? z.B. LOC muß Leerzeilen ausschließen.

1.6.3 Beispiel: Validation

Validiere obiges Beispiel $T(k) = b + G(k)/p$

- LOC sei akzeptabel definiert.
- Interpretation nicht nötig.
- Genauigkeit festlegen, je nach Zweck.
  z.B. $\pm 20\%$ mit Zuverlässigkeit 90%.
- Sammle Datenpunkte.
- Prüfe Schranken, ggf. mit statistischem Test.



50 Datenpunkte, 5 außerhalb des 20%-Bereichs erlaubt, 14 außerhalb sind vorhanden.

Validierung liefert oft Einsichten, die dann zu besseren Vorhersagesystemen führen.

Im Beispiel liegen z.B. lediglich die Daten systematisch zu hoch.

1.6.4 Häufige Fehler bei Validation

- Nicht Übereinstimmung mit Meßziel wird geprüft, sondern Korrelation mit bekannten Maßen.
  Weitgehend nutzlos, es sei denn neues Maß ist einfacher oder früher.
- Keine Hypothese, sondern Validation gegen irgendwas, worüber Daten zur Hand sind.
  z.B. Validation von Modularitätsmaß über Vorhersagekraft für Entwicklungskosten, obwohl keine Aussage über Zusammenhang vorgegeben.
- Validation aufgrund derselben Daten, aus denen Vorhersagesystem entwickelt wurde. Im Prinzip unsinnig; nur akzeptabel falls konservative Vorgehensweise.

1.6.5 Wahl von Vorhersagesystemen

Nützlich und validierbar sind

- Interne Produktattribute früher Phasen zur Vorhersage interner Attribute später Phasen.
  z.B. Größe und Modularität einer Spezifikation zur Vorhersage von Größe und Strukturiertheit einer Implementation.
- Frühe Prozeßattribute zur Vorhersage späterer Prozeßattribute.
  z.B. Anzahl von Fehlern aus Grobentwurfsinspektion für Vorhersage Entwurfskorrekturen während der Implementation.
- Interne Produktattribute zur Vorhersage von Prozeßattributen.
  z.B. Strukturiertheit des Codes zur Vorhersage Zeitbedarf für Wartungsarbeiten.

- Interne Produktattribute und Prozeßattribute zur Vorhersage externer Produktattribute.
  z.B. Zahl Fehler aus Codedurchsichten zur Vorhersage der Zuverlässigkeit.

Vorhersage externer Attribute aus internen Attributen grundsätzlich problematisch.
Einschränkungen nötig (Architekturmodelle, Entwurfsmethoden, Programmierkonventionen etc.).
Mitnutzung von Prozeßinformation nützlich.

1.7 Beispiel: Objektorientierte Entwurfsmaße

- Vorgestellt von Chidamber und Kemerer (CK) 1991.
- 1994 diskutiert anhand zweier großer industrieller Klassensammlungen (eine in C++, eine in Smalltalk).
- Basili, Briand, Melo (BBM) validierten 1995 dieselben Maße bezüglich Vorhersage der Fehleranfälligkeit.
  - Praktikum: Studenten, 4 Monate.
  - 8 Implementationen eines Videothekleihsystems
  - OMT, C++ und Bibliotheken
  - Vorhersage: mindestens 1 Fehler in Testphase.
  - Unterscheidung Subsysteme (GUI, TUI, DB) und un-/leicht/stark veränderte Klassen.

Beachte unterschiedliche Forschungsansätze von CK und BBM.

1.7.1 Gewichtete Methoden pro Klasse (WMC)

„Weighted Methods per Class“: WMC $= \sum_{Methodsi} c_i$ wobei c_i die Komplexität der Methode i ist (bei CK und BBM stets 1).

Bedeutungen und Beobachtungen laut CK:

1. Vorhersage Aufwand zur Entwicklung und Wartung.
2. Hohe Zahl Methoden bedeutet großen Einfluß auf Unterklassen (wegen Vererbung).
3. Klassen mit vielen Methoden meist anwendungsspezifischer, schlechter wiederverwendbar.

Beobachtungen: Anzahl Klassen nimmt mit steigendem WMC in beiden Firmen exponentiell schnell ab, trotz verschiedener Anwendungsgebiete, Leute und Sprachen.

BBM:

- Signifikanz $\alpha = 0.06$
- 0.7% der Varianz erklärt.
- Bei neuen und stark veränderten Klassen 2.4% Varianz erklärt.
- Bei UI 4.7% der Varianz erklärt.

1.7.2 Tiefe im Vererbungsbaum (DIT)

„Depth of Inheritance Tree“: DIT = Länge des längsten Pfades von Klasse zur Wurzel ihres Vererbungsbaums.

Bedeutungen laut CK:

1. Je tiefer in Hierarchie, desto schwieriger Verhalten vorherzusagen.
2. Tiefere Bäume haben höhere Entwurfskomplexität.
3. Je tiefer eine Klasse in Hierarchie, desto höher ihr Potential für Wiederverwendung von Methoden.

Beobachtungen: Mediane für DIT von 1 bzw. 3. DIT-Verteilung erlaubt angeblich, zu entscheiden, ob ein Entwurf „kopflastig“ oder „fußlastig“ sei.

Beide sind kopflastig: schlechte Wiederverwendung zugunsten leichterer Verständlichkeit.

- Signifikanz von $\alpha = 0.0000$ für „hohe DIT sagt Fehler voraus".
- 6.5% Varianz erklärt.
- „Kopflastig" also im Experiment klar besser!

1.7.3 Zahl von Unterklassen (NOC)

„Number of Children": NOC = Zahl direkter Unterklassen der Klasse.

Bedeutungen laut CK:

1. Je mehr Unterklassen, desto mehr Wiederverwendung.
2. Je mehr Unterklassen, desto höher Wahrscheinlichkeit falscher Abstraktion in der Oberklasse und Wahrscheinlichkeit, daß Vererbung mißbraucht wurde.
3. Je mehr Unterklassen, desto mehr Einfluß auf Gesamtentwurf. Dann sorgfältiger testen.

Beobachtungen: Ähnliche Verteilungen in beiden Firmen. Etwa 70% Klassen ohne Kinder. NOC zum Prüfen, ob Entwurfsphilosophie eingehalten wird.

BBM:

- Signifikanz von $\alpha = 0.0000$, für „hohe NOC bedeutet wenig Fehler" (weil sorgfältiger entworfen?)
- 14% Varianz erklärt.

1.7.4 Kopplung zwischen Objektklassen (CBO)

„Coupling Between Object Classes": CBO = Anzahl anderer Klassen, deren Methoden oder Instanzvariablen benutzt werden.

Bedeutungen laut CK:

1. Hohe Kopplung vermindert spätere Wiederverwendungsmöglichkeiten.
2. Hohe Kopplung vermindert Modularität und Kapselung und erhöht Empfindlichkeit gegen Änderungen.
3. Erlaubt vorherzusagen, wie schwierig oder kritisch Testen wird.

Beobachtungen: Median bei C++ gleich Null, bei Smalltalk gleich 9. CBO wächst mit der Größe der Klassenmenge.

BBM:

- Signifikanz von $\alpha = 0.0000$ für hohe Fehlerwahrscheinlichkeit bei hoher CBO.
- 6.8% Varianz erklärt.

1.7.5 Antwortmenge einer Klasse (RFC)

„Response for a Class“: RFC = Gesamtanzahl verschiedener Methoden, die direkt aufgerufen werden.

Bedeutungen laut CK:

1. Große RFC machen Testen schwieriger.
2. Große RFC bedeutet hohe Komplexität.

Beobachtungen: Klassenzahl fällt exponentiell mit steigendem RFC. Bei Smalltalk viel höher als bei C++. Wenige Klassen mit sehr hoher RFC: Schnittstellenklassen, gut als strukturelle Treiber zum Systemtesten. RFC erlaubt Untersuchung des Entwurfs von Klassenhierarchien.

BBM:

- Signifikanz von $\alpha = 0.0000$ für „hohe Fehlerwahrscheinlichkeit bei hoher RFC“.
- 6.5% der Varianz erklärt, 25% für neue und stark geänderte, 36% bei UI.

1.7.6 Mangel an Zusammenhang zwischen Methoden (LCOM)

„Lack of Cohesion in Methods“: LCOM = $max(|N| - |S|, 0)$, wobei N Menge von Methodenpaaren ohne gemeinsam benutzte Instanzvariable (no sharing), S entsprechend mit sharing.

Bedeutungen laut CK:

1. Zusammenhang wünschenswert, weil er Kapselung unterstützt.
2. Mangel an Zusammenhang zeigt an, daß Klasse wahrscheinlich gespalten werden sollte.
3. Hilft beim Entdecken von Entwurfsfehlern.
4. Geringer Zusammenhang erhöht Komplexität und Wahrscheinlichkeit von Fehlern beim Entwicklungsprozeß.

Beobachtungen: Median von 0 bzw. 2. Maximum bei C++ 200, bei Smalltalk 17. Beurteilt, ob Zusammenhangsprinzip gut eingehalten wird.

BBM:

- Keine signifikante Vorhersagekraft, vermutlich wegen Abschneiden negativer Werte. Das ist also Entwurfsfehler in Definition des Maßes.

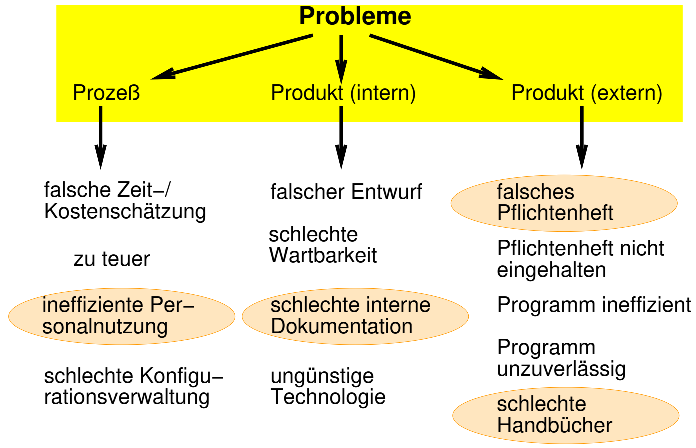
1.7.7 Diskussion

- Unterschiedlich nützliche Maße für Vorhersagesysteme.
- Sehr verschiedene Sichtweise von CK und BBM:
  CK: Daten zum Anregen von Ideen, wenig greifbare Ergebnisse.
  BBM: Konkretes Ziel, strukturiertes Vorgehen, klare quantitative Resultate.
- BBM Resultate: RFC, DIT, evtl. NOC gute Indikatoren für Fehleranfälligkeit, LCOM sollte umdefiniert werden.
- Gut wäre: BBM-Ansatz mit CK-Daten.

1.8 Zusammenfassung

- Maße erlauben Objektivierung.
- Beachte Skalenart und richtige Interpretation.
- Entscheidend: Aussagekraft im Vorhersagesystem.
- Muß an Zielen orientiert sein.
- Validation nötig.
- Methodische Fehler sind sehr häufig.
- Gute Maße oft schwer zu finden.
- Beispiel: o-o Entwurfsmaße.

2 Psychologie und Softwaretechnik



2.1 Einordnung und Ziele

Psychologische Effekte oft wichtig für Softwareerstellung.
Kenntnis Voraussetzung für gezielte Problemvermeidung oder Leistungsoptimierung.

Themen:

1. Lernen
2. Gedächtnis
3. Sprache, Denken, Problemlösen
4. Bedürfnisse und Motivation
5. Streß
6. Sozialpsychologie
7. (Psychologische Schulen)

2.2 Probleme und Hindernisse

- Erkenntnisse meist „offensichtlich".
- Aber nicht trivial.
- Faktenbewußtsein nötig für gezielte Anwendung.
- Schwierig, Haupteffekte zu finden und bei Maßnahmen Nebenwirkungen zu vermeiden.
- Intuition meist nützlich, aber manchmal irreführend.

2.3 Lernen

- Lernen: Permanente Verhaltensänderung als Folge von Übung.
- Vier Arten des Lernens:
  - Habituation.
  - Klassische Konditionierung.
  - Operative Konditionierung.
  - Komplexes Lernen.
- Vorsicht vor Habituation!
- Klassische Konditionierung zur Aufmerksamkeitssteuerung.
- Operative Konditionierung zur Motivationssteuerung.

Komplexes Lernen

- Bildung mentaler Abbilder der Welt: Abstraktionen
- und Manipulation dieser Abstraktionen.
- Mache Konzepte hinter Artefakten deutlich.
- Identifiziere Vor-Überzeugungen.
  Vermeide unechte Assoziationen.
- Induktives oder deduktives Lernen.

2.4 Gedächtnis

- Sensorisches Gedächtnis
- Kurzzeitgedächtnis
- Langzeitgedächtnis

z.B. 14 alphabetisch sortierte Wörter:

divers ein Experiment hat Kapazität Kurzzeitgedächtnis minus Objekt plus sieben unser von zufolge zwei.

2.4.1 Kurzzeitgedächtnis

Diversen Experimenten zufolge hat unser Kurzzeitgedächtnis eine Kapazität von sieben Objekten, plus oder minus zwei.

- Speicherung in “Bündeln” (chunks).
- z.B. Schachstellungen merken.
- z.B. Programmabschnitte merken, Aktionsfolgen merken.
- Folgerung: Abstraktionen erlernen, die große Bündel erlauben.
- Verwaltung als Schlange (first in, first out).
- Phonologisches K. und visuelles K.
- Gleichzeitig nutzbar!
- Fehler bei visueller Speicherung.

2.4.2 Langzeitgedächtnis

- Unbegrenzte Speicherdauer.
- Fast unbegrenzte Kapazität.
- Geringe Eingabebandbreite.
- Funktionalität: Erkennen und Assoziieren.
- Assoziationen semantisch, zeitlich oder räumlich.

- Speicherung in abstrakter Form.
- Verbesserung durch Lernen von “Vokabular”.
- Verbesserung durch Elaboration/Begründungen.
- Beides explizit herbeiführbar.
- Verbesserung durch Vermehrung der Zugriffspfade:
  - Begründungen
  - Taxonomien
  - Verbindende Geschichten oder Reime
  - Merkbilder
- Verbesserung durch Kompression: Schemata
- Verbesserung durch gleichen Kontext.

2.5 Sprache und Denken

2.5.1 Denken

2.5.2 Konzepte und Sprache

- Arten des Denkens: Aussagen, Bilder, Motorik.
- Aussage: Beziehung zwischen Objekten oder Objektklassen.
- Konzepte dienen kognitiver Sparsamkeit: Welt handhabbar machen.
- Objektkonzepte (Gegenstände, Abstraktionen): Substantive.
- Beziehungskonzepte (Zustände, Vorgänge): Verben.
- Eigenschaftskonzepte: Adjektive.
- Einzelobjekte: Namen, bestimmte Artikel, Pronomen
- Der Rest ist fast nur Syntax.

- Kategorisierung: Zuweisung Objekt an Konzept.
- Informationsgewinn.
- Fehlerrisiko.
- Also Vorsicht bei Konzeptbildung und Kategorisierung.

2.5.3 Prototypen und Kerne

- Prototyp: typisches.
- Kern: essentielles.
- Kern Teil von Prototyp oder auch nicht.
- Kern prüfbar: klassisches Konzept.
- Abstützen auf Prototyp: unscharfes Konzept.
- (Fuzzy Logic: klassische Konzepte!)
- Reale Welt: meist unscharfe Konzepte.
- Vermeide unscharfe Konzepte.
- Kennzeichne sie als solche.

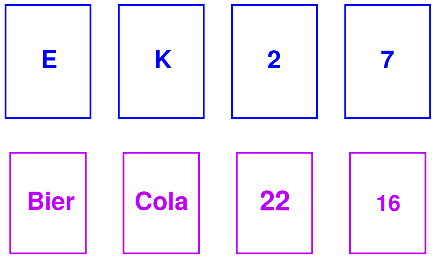
2.5.4 Erwerb von Konzepten

- Explizites Lehren: Erwerb von Kernen.
- Erfahrungen machen: Erwerb von Prototypen.
- Wisse was Du nicht weißt.
- Exemplarstrategie.
- Hypothesentesten.
- Top-Down-Lernen.
- Top-Down mächtig, aber gefährlich, z.B. bei Fehlersuche.
- Bewußte Auswahl treffen.

2.5.5 Schließen

- Taxonomien (ungleich Vererbung).
- Deduktion ist inhaltsabhängig.

Behauptung: „Wenn Vokal vorn, dann gerade Zahl hinten“



- Trick: pragmatische Regel.
  Aber Vorsicht!
- Trick: mentales Modell.
  Geeignete Metaphern nötig.
  Lernbar.

Induktives Schließen:

- z.B.: Johanna hat Informatik studiert.
- Johanna arbeitet in der Softwareabteilung von VW.
- (Schluß:) Johanna arbeitet als Informatikerin.
- Basisdatenregel.
- Konjunktionsregel.
- Beide werden manchmal mißachtet, besonders wenn nur abstraktes Wissen vorhanden und keine eigenen Erfahrungen.

- Basisdatenregel: Argument wird stärker wenn Prämisse
  „Johanna ist Mitglied der Gesellschaft für Informatik“
- Konjunktionsregel: Argument wird schwächer wenn Schluß
  „Johanna arbeitet als Informatikerin und verdient 85.000 DM im Jahr.“
- Verletzung der Konjunktionsregel:

  Das Pflichtenheft war vollständig in Textform abgefaßt, völlig ohne Tabellen oder Abbildungen. Es war unstrukturiert und unübersichtlich und wies zahlreiche inhaltliche Mängel und Fehler auf.

zu beurteilen sei die Wahrscheinlichkeit der folgenden beiden Aussagen:

1. Im Pflichtenheft waren 12 wesentliche Anforderungen nicht erfaßt.
2. Im Pflichtenheft waren 12 wesentliche Anforderungen nicht erfaßt und weitere 9 waren unklar oder mehrdeutig formuliert.

Grund des Fehlschlusses: Ähnlichkeit mit Prototyp.

- Andere Verletzung der Konjunktionsregel:

  Im Jahr 2007 wird es eine massive Überflutung in Nordamerika geben, der mehr als tausend Menschen zum Opfer fallen.

  empfindet man als weniger wahrscheinlich als

  Im Jahr 2007 wird es ein Erdbeben in Kalifornien geben, das eine massive Überflutung hervorruft, der mehr als tausend Menschen zum Opfer fallen.

- Grund: Hinweis auf plausible Begründung.

Weiteres Problem: Bestätigungsneigung

- Wir sammeln nur Informationen zur *Bestätigung* unserer Hypothesen.
- z.B. bei Prüfung deduktiver Schlüsse mit Allquantor: Suche nach Beispiel statt nach Gegenbeispiel.

2.5.6 Probleme lösen

- Abstandsverminderung.
- Zweck/Mittel-Analyse.
- Rückwärtsarbeiten.
- Unterschiede zwischen Experten und Laien:
  - Reiche Repräsentationen.
  - Lösungspläne.
  - Selten Rückwärtsarbeiten.

2.6 Bedürfnisse und Motivation

- Motivation = hoher Aktivitätsdrang.
- Nicht notwendigerweise positiv.
- Arten
  - Ehrgeiz.
  - Machtstreben.
  - Soziale Bedürfnisse.
  - Neugier.

2.6.1 Stärke von Motivation

- Drei Faktoren:
  1. Grundmotivation: Charakterzug.
  2. Erfolgsaussichten: erfahrungsabhängig.
  3. Subjektiver Wert: Frage der Wertsetzung.
- Erfolgsaussichten und Wert beeinflußbar.
- Eigene Grundmotivationen einschätzen lernen.

2.6.2 Motivation und Belohnung

Subjektiven Wert ändern.

- Ausbleiben von Belohnung $\Longrightarrow$ (tendenziell) Abnahme der Motivation.
- Oft intrinsische Motivation („Dinge geschehen machen“).
- Belohnung erhöht intrinsische Motivation.
- ... kann sie aber auch verringern (wenn nicht leistungsabhängig).
- Erlernte Hilflosigkeit möglich (mehrfaches Versagen).

2.6.3 Anwendung in der Softwaretechnik

Personeneignung oder Organisationsprinzipien.

- Neugier: Prototyp,Entwurf $\Longleftrightarrow$ Test,Dokumentation.
- Soziale Bedürfnisse: Anforderungsanalyse.
- Ehrgeiz: Vorsicht bei Belohnungssystemen.
- Machtstreben: Problematisch, aber Kontrolle über Umgebung als Surrogat, evtl. nützlich.

2.7 Streß

2.7.1 Ursache und Wirkung

- Ursprünglich „Kampf-oder-Flucht"-Reaktion für lebensgefährliche Situationen.
- Unschädlich wenn selten und kurz.
- Heute jedoch oft und langdauernd.
- Dadurch: hoher Blutdruck, Neigung zu Diabetes, Herzgefäßverengung und Magengeschwüren, evtl. stärkere Gedächtnisschwäche im Alter.

2.7.2 Streßvermeidung

- Stärke von Streß hängt ab von:
  1. Auftreten von Stressoren.
  2. Situationswahrnehmung.
  3. Emotionale Reaktionsbereitschaft.
  4. Maß möglicher Kontrolle über Situation.
  5. Vorhersehbarkeit der Situation.
- Alle Faktoren zur Streßverminderung nutzbar:
  - Persönlichkeitsschulung (Wahrnehmung, Reaktionsbereitschaft).
  - Organisationsverbesserung (Auftreten, Kontrolle, Vorhersehbarkeit).

2.7.3 Umgang mit Streß

- Problemorientiert:
  1. Problem definieren.
  2. Lösungsideen entwickeln, vergleichen, auswählen.
  3. Lösung umsetzen
- Emotionsorientiert:
  - Verhaltensstrategien (z.B. Sport, Alkohol, Hilfesuche)
  - Kognitive Strategien (z.B. Eintauchen, Rationalisieren, Wegschieben)

Problemorientierter Umgang ist eine erlernbare Fertigkeit.

2.8 Sozialpsychologie

2.8.1 Attribution

- Daten sammeln, Theorien bilden:
  Implizite Psychologie
- Theorien werden oft wie Fakten behandelt.
- Hauptunterscheidung: dispositive und situative Faktoren.
- Prüfmerkmale für Unterscheidung:
  - Konsens: Viele Personen $\Longrightarrow$ situativ.
  - Konsistenz: Oft bei einer Person $\Longrightarrow$ dispositiv.
  - Eigentümlichkeit: Oft in Situation $\Longrightarrow$ situativ.

Systematische Attributionsfehler:

- Fundamentaler Attributionsfehler: zuviel dispositiv.
- Falscher Konsens: unsere Meinung ist gar nicht typisch.
- Eigenerfolg und Fremdversagen.
- Relevanzvermutung.
- Verursachungsillusion.

Folgerung: intensive Kommunikation, insbesondere zur Vermeidung falscher Konsensvermutungen und von „Eigenerfolg und Fremdversagen“.

2.8.2 Heuristiken im sozialen Schließen

- Repräsentativitätsheuristik: Benutzung von Stereotypen.
- Problem: Über-Nutzung des Stereotyps möglich.
- Verfügbarkeitsheuristik: Kennen = häufig oder wichtig.
- Probleme: kürzliche Erfahrungen dominieren; oftmals verzerrte Informationsbelieferung (z.B. Flugzeugabsturz).
- Regressionsfehlschluß.

Auch in technischen Situationen problematisch.

2.8.3 Haltungen

- Affektive Komponente.
- Kognitive Komponente.
- Verhaltenskomponente.

Sagen Verhalten oft nicht gut voraus, wegen

- hoher Allgemeinheit.
- geringer Relevanz.
- fehlender Selbstattribution.
- Verhaltensbeschränkungen.

Oft führt Verhalten zu Haltung (statt umgekehrt)

- Verhaltensdiskrepanz (kognitive Dissonanz) führt zu
  - Senkung der Wichtigkeit eines dissonanten Elements oder
  - Zufügen konsonanter Elemente oder
  - Beseitigung eines dissonanten Elements.

  Das sind meist Haltungsänderungen (Bsp. „intelligent“ + schlechte Noten).
- „Teuer = gut“.
- „Meine Entscheidung = gut“.
- „Meine Situation = gut“.

Häufig Selbstachtung betroffen.

Vorurteile:

- Grund: Stereotypen plus selektive Wahrnehmung.
- Wichtige Folge: Selbsterfüllende Prophezeihungen.

- Beides *kann* positiv sein.

2.8.4 Soziale Einflüsse

- Konformität (Vorsicht bei Inspektionen).
- Soziale Förderung (bei komplexen Aufgaben kontraproduktiv).
- Soziales Faulenzen. (Einzelleistungen müssen erkennbar sein).
- Reziprozität (organisatorisch ausnutzen).
- Vor-Engagement bindet (erschwert Korrektur schwerer Fehler).

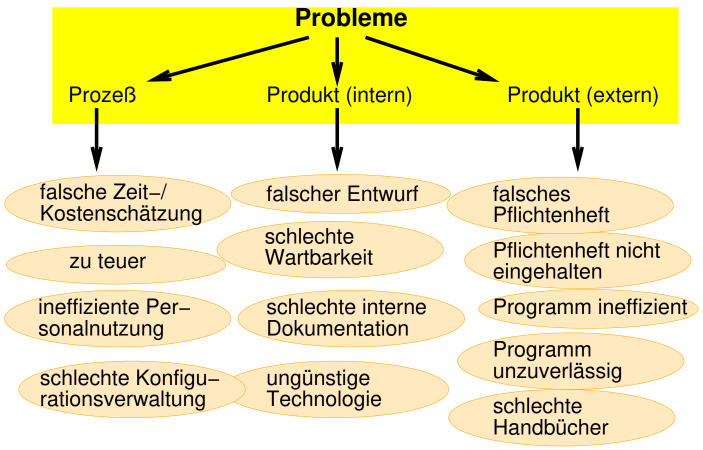
2.9 Nachtrag: Psychologische Schulen

- Strukturalismus
- Funktionalismus
- Behaviorismus, S-R–Psychologie
- Gestaltpsychologie
- Psychoanalyse
- Informationsverarbeitungssysteme, Psycholinguistik
- Neuropsychologie
- Kognitive Psychologie

2.10 Zusammenfassung

- Kognitive Mechanismen und Grenzen kennen.
- Kompetenz durch Besitz geeigneter Konzepte.
- Vorsicht vor logischen Fehlschlüssen.
- Gezielt mit Motivation umgehen.
- Gezielt mit Streß umgehen.
- Vorsicht vor sozialen Fehlschlüssen.
- Persönliche Arbeitsweise kann psychol. Erkenntnisse berücksichtigen.
- Organisationsstruktur kann das auch.

3 Softwaretechnik-Forschung



3.1 Einordnung und Ziele

3.1.1 Ingenieurwesen oder Wissenschaft?

Fred Brooks, 1996:

> The scientist *builds in order to study*,
> the engineer *studies in order to build*.

Informatik keine Wissenschaft.

Gefährlich, wenn wir anderes glauben:

- Entwürfe als Selbstzweck.
- Wir vergessen die Anwender.
- Gute Leute aus Anwendungen abgezogen.

ABER:

Unterscheidung gilt nicht für Fach, sondern für einzelne Arbeit.

Wissenschaft nötig, um das Ingenieurwesen mit Wissen zu untermauern.

3.1.2 Methoden zum Erkenntnisgewinn

- Festhalten
- Intuition
- Autorität
- Rationalismus
- Empirizismus
- wissenschaftliche Methode

3.1.3 Wissenschaft in der Softwaretechnik

Wir wissen wenig!

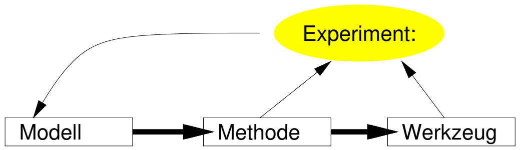
- Vorgehen: bisher meist Intuition plus Empirizismus.
- Beiträge: meist Entwürfe.
- Selten klare Hypothesen.
- Noch seltener Prüfungen.

Prüfungen sind enorm aufwendig.

3.2 Ansätze

3.2.1 Wissensarten: Modell, Methode, Werkzeug

1. **Modelle**.
   Abstrakte Repräsentation.
   Zweck: Verständnis verbessern, Vorhersagen.
   Bindet Wissensteile zusammen.
2. **Methoden**. Allgemeine Anleitung.
   Zweck: Probleme lösen.
   Softwaretechnik ist Methodenlehre.
3. **Werkzeuge**. Sind Beiträge, wenn Effekt verstehbar oder wenn Anwendbarkeitsbeweis.



Die drei Wissensarten in der Softwaretechnik und ihre Beziehungen.

Phasenmodell: Modell/Methode

Programmiersprache: Modell/Werkzeug

3.2.2 Arten von Beiträgen

1. Existenzbeweis.
2. Offensichtliche Verbesserung.
3. Theorem.
4. Validierter Entwurf.
5. Eigenschaftsbeweis.
6. Vergleich.

Am häufigsten: *un*validierter Entwurf.

3.2.3 Forschungsmethoden

1. Entwurf und Bau.
2. Mathematischer Beweis.
3. Kontrolliertes Experiment.
4. Fallstudie.
5. Feldbeobachtung.

Ergänzen sich gegenseitig.

3.2.4 Forschungsfelder

- **Anforderungsanalyse.**
  Wie kommuniziert man mit dem Anwender?
  Wie setzt man vage Anforderungen (Benutzerfreundlichkeit, Sicherheit) in konkrete um?
  Wie versteht man Prioritäten und Zufriedenheitskriterien des Anwenders?
  Wie stellt man Eindeutigkeit und Vollständigkeit sicher?
  Wie verteilt man Aufgaben zwischen System und Anwender?

- **Spezifikation.**
  Wie integriert man verschiedene Sichten und Darstellungen?
  Welche Darstellungen sind effizient und fehlerarm handhabbar? Warum?
  Welche erlauben die Verfolgung von Anforderungen?
  Welche sind anwendertauglich? Warum?
  Wie prüft man Eindeutigkeit und Vollständigkeit?

- **Entwurf.**
  Wie evaluiert man verschiedene Entwurfsansätze?
  Wie findet man einen guten Entwurf in Bezug auf Fehler, Implementationskosten, Implementationsgeschwindigkeit, Produkttestbarkeit, Produktwartbarkeit?
  Wie prüft man diese Qualität?
  Welche wiederkehrenden Muster in Entwürfen lassen sich entdecken? Warum?
  Welche Formalismen sind zur Darstellung von Entwürfen günstig? Warum?
  Wie dokumentiert man Anforderungen und Entwurfsentscheidungen?

- **Implementation.**
  Welche Sprachen haben die höchste Produktivität, die geringste Fehlerhäufigkeit? Warum?
  Welche Programmierstile optimieren Effizienz, Verständlichkeit, Erweiterbarkeit, Fehlerhäufigkeit? Warum?
  Wie unterstützt man Verfolgung von Entwurfsentscheidungen, Anforderungen?

- **Test.**
  Welche Testmethoden sind am effektivsten? Warum?
  Wie kann man Test automatisieren?
  Wie kann man Testaufbauten wiederverwenden?
  Wie kann man SW-Zuverlässigkeit schätzen?
  Wie kann man das Fehlerlokalisieren beschleunigen?

- **Reengineering.**
  Wie deckt man verlorene Anforderungen, Entwurfsentscheidungen, Entwurfsstrukturen auf?
  Wie stellt man Auswirkungen einer Änderung fest?
  Wie findet man die Stelle(n), wo man ein großes Programm ändern muß?

- **Dokumentenverwaltung.**
  Wie bewahrt man Bezüge zwischen Dokumenten?
  Wie verwaltet man Konfigurationen? Wie geht man mit unvollständiger Kontrolle um?
  Wie unterstützt man örtlich verteilte Softwareentwicklung?

- **Wiederverwendung.**
  Wie entwirft, validiert, katalogisiert man wiederverwendbare Software?
  Wie findet man eine Komponente?
  Wie entwirft, produziert, testet man mit Wiederverwendung?
  Wie wirkt Wiederverwendung auf Qualität, Kosten, Produktivität? Warum?
  Wie verwendet man Artefakte früher Phasen wieder (Anforderungen, Entwürfe)?

- **Softwareprozeß.**
  Wie sollte eine SW-Organisation arbeiten?
  Wieviel Vorschriften braucht man? Welche? Warum?
  Wie optimiert man Vorgehen durch Beobachtung?
  Wie wirkt sich das auf den einzelnen aus?
  Wie verhindert man unerwünschte Nebenwirkungen?

3.3 Probleme und Hindernisse

Zwei Hauptprobleme:

1. **Hoher Aufwand:**

- Menschen.
- Gruppen.
- Qualifikation.
- Dauer.

2. **Viele Variablen:**

- Versuchspersonen: Ausbildung, Fähigkeiten, Erfahrung; allg./hierfür.
- Aufgabenstellung: Größe, Anwendungsgebiet, Komplexitätsgrad, Besonderes.
- Randbedingungen: Termine, Qualität, Methodenwahl
- Arbeitsumfeld: maschinelle u. organisatorische Unterstützung.

Also:
1. Schon ein Datenpunkt teuer,
2. aber viele nötig!

Beispiel: Vergleiche zwei Entwurfsmethoden

- 3 Klassen von Versuchspersonen (mäßig kompetent, kompentent, sehr kompetent),
- 3 Schwierigkeitsgrade der Aufgabenstellung (einfach, mittel, schwierig),
- 8 Anwendungsgebiete,
- 2 Stufen von Qualitätsanforderungen (normal oder hoch),
- 2 Arten von Terminplänen (normal oder schnell),
- 8 Kombinationen sonstiger Methoden,
- 3 verschiedene Arbeitsumfelder.

$3 \cdot 3 \cdot 8 \cdot 2 \cdot 2 \cdot 8 \cdot 3 = 6912$ Versuche!
Beschneide Suchraum unterwegs.

3.4 Experimentelles Arbeiten

3.4.1 Formulierung der Forschungsfrage

1. Identifikation eines Problems.
2. Literaturstudie.
3. Machbarkeitsprüfung.
4. Formulierung.

Zumeist ineinander verquickt.

3.4.2 Kontrolle von Variablen

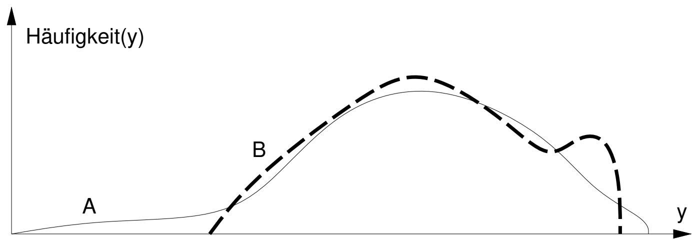
Garantiere Kausalität trotz zusätzlicher Variablen:

- Individuelle Unterschiede.
- Historie.
- Reifung (maturation).
- Instrumentation.
- Sterblichkeit (mortality).
- Statistische Regression.
- Anforderungscharakteristik.
- Erwartung (expectancy).
- Sequenzeffekte.
- Erfahrenheit (sophistication).

3.4.3 Zufallsstichproben und Vergleichsgruppen

- **Zufallsstichprobe** kontrolliert individuelle Unterschiede.
- **Vergleichsgruppe** kontrolliert Historie, Reifung und je nach Ansatz auch Erwartungen (Placebo, Blind, Doppel-Blind), Sequenzeffekte (Gegenbalance), Erfahrenheit.

3.4.4 Varianzreduktion



Vorsortieren

Paarungstechniken.

3.4.5 Versuchspläne

Intra-Subjekt $\Longleftrightarrow$ Inter-Subjekt

Vortest-Nachtest $\Longleftrightarrow$ nur-Nachtest

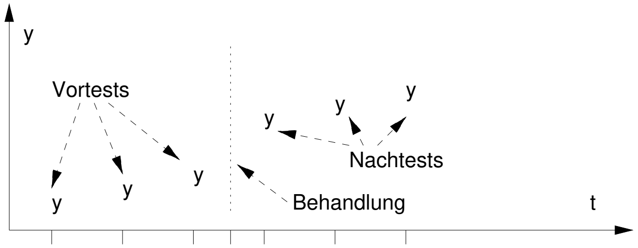
mehrere Behandlungsteile: (partielle) faktorielle Pläne

falsche Pläne z.B. Eingruppen-Vortest-Nachtest

Quasi-experimentelle Versuchspläne

Nichtequivalente Kontrollgruppe:
Gruppeneinteilung z.B. durch Selbstmeldung.

Unterbrochene Zeitreihe:
Mehrfache Vor- und Nachtests statt Kontrollgruppe.



Oft ähnlich akzeptabel wie Experimente (s. Tod durch Unfall).

3.4.6 Statistische Auswertung

- „Draufgucken“ auf Daten reicht nicht.
- Statistik liefert quantitative, stochastische Aussagen.
- Anwendung oftmals falsch.
- Wichtigste Klassen von Techniken:
  - Tests
  - Regression

statistischer Signifikanztest

Berechnet Wahrscheinlichkeit der Beobachtungen, falls Hypothese nicht gilt.

Beispiel: Entwurfsmethode E erzeugt weniger Fehler als A.
40 Versuchspersonen, 2 mal 20 Entwürfe, Fehler messen.

Hypothese: Fehlerzahl bei E-Entwürfen im Mittel kleiner als bei A-Entwürfen.

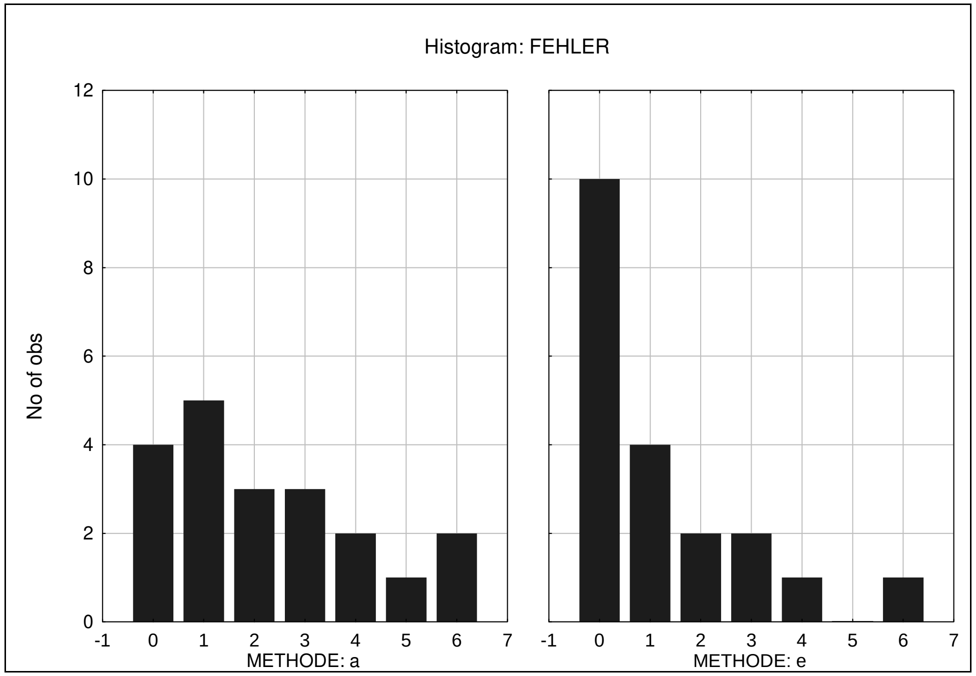
Nullhypothese: Kein Unterschied.

E: (0 4 0 2 0 0 2 1 1 0 0 0 1 3 6 0 3 0 1 0)

A: (0 4 1 2 2 3 5 1 0 0 6 6 1 3 3 1 2 1 4 0).

Mittelwerte:
E: 1.20
A: 2.25

Standardabweichungen:
E: 1.67
A: 1.94



2-Stichproben-t-Test

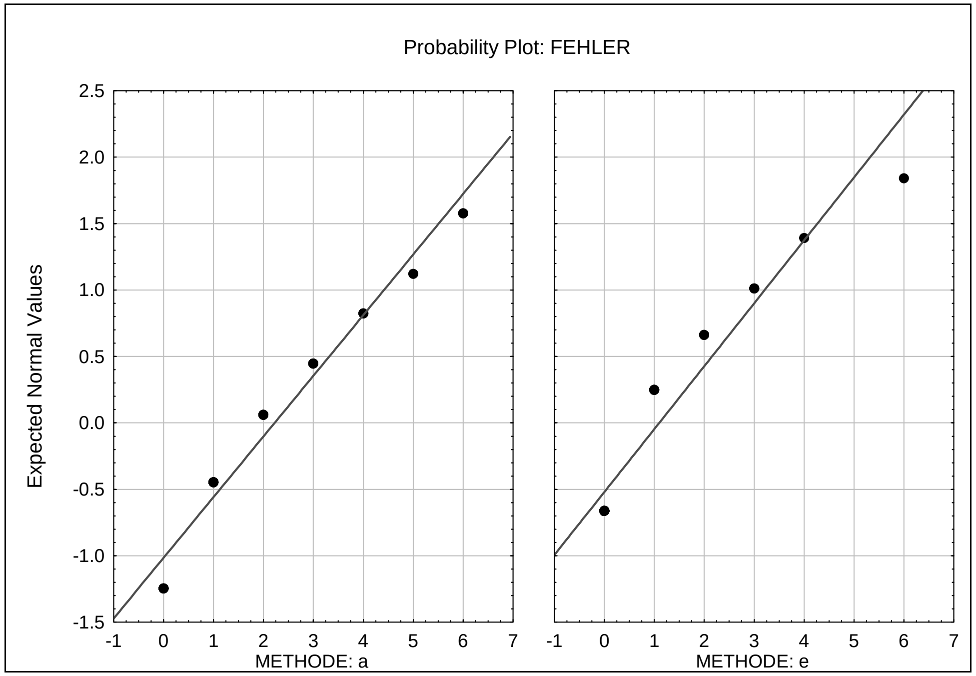
Testet:

- Unterschiede im Mittelwert

Anwendungsvoraussetzungen:

- Gleiche Varianzen
- Normalverteilung

Betrachte Normalquantilplot.



Trotzdem t-Test: liefert schwächere Unterschiede.

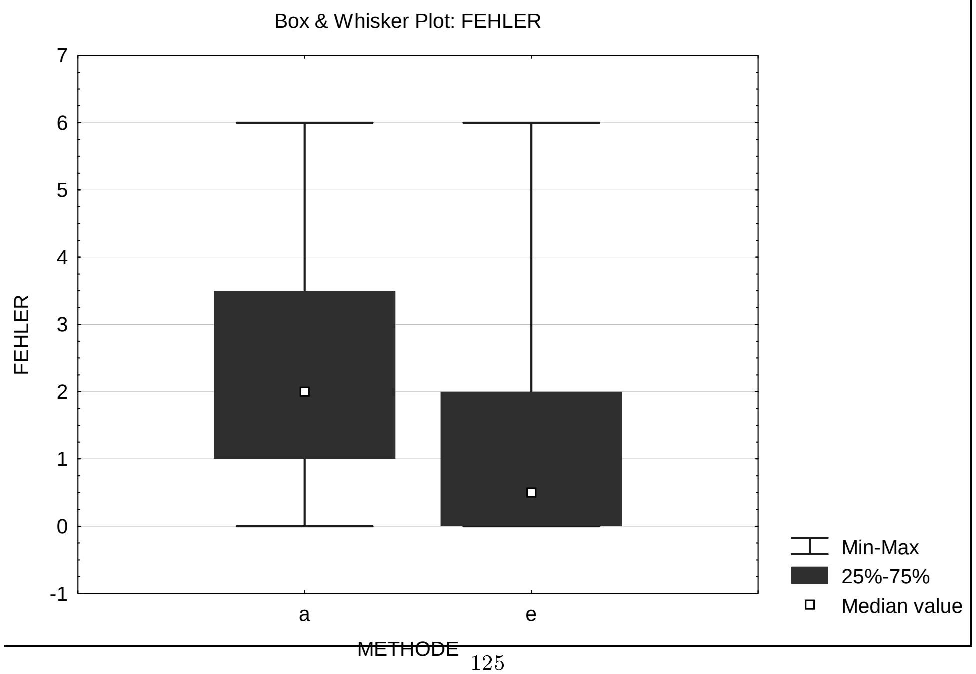
Hier: $p = 0.075$.

Signifikanzniveau. Üblich 0.05, ggf. zwischen 0.001 und 0.2.

Alternative: **nichtparametrische Tests**.

Mann-Whitney U-Test (Wilcoxon Rangsummentest).

Einzige Annahme: Ordinalskala.
Robust gegen Ausreißer.
Gute Grobdarstellung: Boxplots.



- U-Test: $p = 0.055$
- nicht strikt signifikant, entgegen Intuition.
- Mehr Versuchspersonen könnte helfen.
- Boxplots und U-Test sehr oft empfehlenswert.

Regression

- Quantitatives Modell aus Daten erzeugen. Modellklasse vorgeben.
- siehe Validation von Vorhersagesystemen.
- Geht auch für nichtlineare Funktionen z.B. $y = a + bx + cx^2$.
- Teuer, wenn nichtlinear in *Parametern*.
- Geht auch in mehreren Variablen z.B. $y = a + bx_1 + cx_2$.
- Angenommener Zusammenhang muß wirklich gelten!
  z.B. Korrelation über 0.7 bei linearer Regression.
- Beschreibt Effekt, nicht Mechanismus.
  Zusammenhang nicht unbedingt kausal!

3.4.7 Interne und externe Gültigkeit

Interne G.: Korrekte Durchführung u. Kontrolle d. Variablen.

Externe G.: Übertragbarkeit des Resultats.

Hindernisgründe:

- **Populationsgültigkeit**:
  experimentell zugängliche Population vs. Zielpopulation
  Modelle für Übertragung nötig!
- **Ökologische Gültigkeit**:
  Umstände des Experiments: Mehrfachbehandlung, Hawthorne-Effekt, Neuigkeitseffekt.
  Fehlen von Umständen: Druck, professionelle Arbeitsweise, Umfeld.

- **Zeitliche Gültigkeit**:
  Zyklische Effekte: Frühjahr, Montag, Freitag, Wach/Schlaf.
  Lineare Effekte: Reifung und Historie

Experimente wiederholen!

Feldstudien!

3.4.8 Veröffentlichung

Wissenschaft heißt Wissen *ansammeln*.
Deshalb Veröffentlichung entscheidend.

Wichtigste Gesichtspunkte:

Verständlichkeit **Reproduzierbarkeit**

3.5 Beispiel: N-Versions-Programmierung

Vorbildliche experimentelle Arbeit, Knight und Leveson.

3.5.1 Forschungsfrage

- N-Versions-Programmierung:
  - eindeutige Anforderungsbeschreibung.
  - N unabhängig entwickelte SW-Versionen.
  - Mehrheitsentscheid.
  - Hohe Zuverlässigkeit.

- enthält Annahme: statistisch unabhängige Versagen.
- Andernfalls Zuverlässigkeitszunahme geringer.

Annahme gültig?

Vermutung: Manche Teile inhärent schwieriger,
tendenziell gleiche Fehler,

Annahme falsch.

3.5.2 Experimentidee

- Lasse viele Personen je eine Version erzeugen.
- Prüfe mit sehr vielen Testfällen.
- Zähle gleichzeitige Versagen.
- Nimm statistisches Versagensmodell bei Unabhängigkeit.
- Prüfe Unabhängigkeitsannahme.

3.5.3 Experimentaufbau

- 27 Versionen: 15 undergraduates, 8 B.S., 4 M.S.
- 18 aus Irvine, 9 aus Virginia.
- je ca. 5 Std. Verstehen, 16 Implementation, 27 Test.
- 327 bis 1004 Zeilen.
- Keine Diskussion untereinander.

Problemstellung:

- Eingabe: Vektor von Radarreflektionen
- Ausgabe: Rakete oder nicht, plus 240 Bool-Teilergebnisse.
- Extrem gut geprüfte Anforderungsbeschreibung.
- Mathematische Kenntnisse nötig (Winkelrechnen).
- Sorgfalt mit Fließkommagenauigkeit.

Vorgehen:

- 15 Datensätze zum Testen an Versuchsperson.
- Abnahmetest: 200 zufällige Fälle.
  Simuliert gründlichen Test.
  Zufälligkeit vermeidet selektiven Fehlerfilter
- „Gold-Programm“ vorhanden.
- Untersuchung: 1 Million Testfälle
  Erzeugt mit speziellem Testgenerator.
  Simuliert gesamte Lebensdauer.

Versagensmodell:

N Versionen, Versagenswahrscheinlichkeiten p_i

$$P_0 = \prod_{i=1}^{N} 1 - p_i$$

$$P_1 = \sum_{i=1}^{N} p_i \frac{P_0}{1 - p_i}$$

$$P_m = 1 - P_0 - P_1$$

K Mehrfachversagen bei n Testfällen: Binomialverteilt

$$P(K = x) = \binom{n}{x} P_m^x \cdot (1 - P_m)^{n-x}$$

für große n normalverteilt (Mittelwert $n\,P_m$, Varianz $n\,P_m(1 - P_m)$).

$$z = \frac{K - n\,P_m}{\sqrt{n\,P_m(1 - P_m)}} \quad \text{standardnormalverteilt}$$

Nullhypothese: z modelliert Daten, wenn P_m aus Beobachtungen geschätzt.

3.5.4 Ergebnis

- Versionen sehr zuverlässig: alle über 99%
  23 über 99.9%, 6 fehlerfrei.
- 1255 mehrfache Versagen.
- $z = 100.51$
- Wahrscheinlichkeit kleiner als 1 Milliardstel.
- Also Unabhängigkeitsannahme falsch.

Schlüsse:

1. Annahme der Unabhängigkeit mit Konfidenz 99 Prozent zurückgewiesen.
2. Gilt für im Experiment betrachtete Anwendung.
   Übertragbarkeit unbekannt.
3. Jedenfalls Annahme von Unabhängigkeit jetzt unerlaubt.
4. Heißt nicht, daß N-Versions-Programmierung unnütz.
   Aber Zuverlässigkeitsberechnung muß verbessert werden.
5. Weitere Forschung über Ursachen gemeinsamer Versagen nötig.

3.5.5 Diskussion

- Forschungsfrage: bedeutsam, eng genug.
- Experimententwurf: gute interne Gültigkeit.
  N-Versions-Rahmenbedingungen eingehalten, Auswertung gründlich, Umfang der Tests schließt Zufälligkeit aus.
- Argumentation für externe Gültigkeit:
  Zuverlässige Versionen: keine Profis nötig.
  Kleiner Umfang ist realistisch wg. Kosten.

- Aufbereitung:
  Idee, Aufbau, Ergebnisse verständlich. Gründliche Einführung, Informationen über Details, Kritikpunkte diskutiert.
- Auslegung der Ergebnisse:
  nur äußerst vorsichtige Schlüsse.
  Konfidenz höher als angegeben, keine Übertragung des Resultats, keine Zurückweisung von NVP.

Starke Angriffe.

Furiose Antwort von Knight und Leveson in Software Engineering Notes, Januar 1990.

Sehr lesenswert!

3.6 Beispiel: Pseudokode versus Flußdiagramm

Mißlungene Forschung, Scanlan 1989:

- Sauber aufgebaut und durchgeführt.
- Erwartete Resultate erzielt.
- Aber irrelevant wegen falschem Entwurf.

3.6.1 Forschungsfrage

Verständlichkeit von Flußdiagrammen im Vergleich zu Pseudokode. Annahme: Flußdiagramme besser. Hypothesen:

1. weniger Verständnisfehler.
2. geben Studenten mehr Zutrauen in ihr Verständnis.
3. verringern Zeit zur Beantwortung von Fragen.
4. verringern Anzahl von Blicken.
5. je komplexer Algorithmus, desto nützlicher.

- Vorangegangen: Arbeit von Shneiderman et al.
  - Entwicklung, Verstehen, Debugging, Modifikation.
  - Fortran versus Flußdiagramm
  - Keine signifikanten Unterschiede
- Scanlans Kritik: Unterschiede verschüttet, weil
  - Zu einfache Programme.
  - Zu kleine Gruppen (10–30 Personen).
  - Ausgabeformat gefragt. Fehlt in Flußdiagramm.
  - Vor allem: keine Zeitmessung.

3.6.2 Experimentidee

- Ein einfacher, mittlerer, komplexer Algorithmus in Pseudokode und äquivalent als Flußdiagramm.
- Je zwei Fassungen A und B.
- Für jeden Algorithmus Katalog von Verständnisfragen.
- Experimentiervorrichtung zum Messen von Anzahl und Dauer der Blicke, Dauer der Antworten.
- Wende auf viele Versuchspersonen an.
- Prüfe Hypothesen anhand der Daten.

3.6.3 Experimentaufbau

Extrem sorgfältige Kontrolle über Variablen.

- 82 Versuchspersonen, genau aufgeschlüsselt.
- Algorithmen: nur Kontrollfluß fest, Inhalte zufällig, damit garantiert neu für Versuchspersonen.
- Je zwei Versionen A und B gegen Lerneffekt von Flußdiagramm zu Pseudokode (jede Person macht beides).
- Gleiche Schriftart gegen Lesbarkeitsunterschiede.
- Tageslichtfreier, schallgedämmter Versuchsraum.
- Vortest: 9-von-10 Fragen Verständnis der Experimentalprozedur.

- Experimentdurchführung mit voller Computersteuerung:
  - Papiervorlagen, Auf-/Zudeckautomat.
  - Entweder Antworteingabe oder Algorithmus sichtbar.
  - Zeitmessung, Ansichtzählung.
  - Einführung/Üben mit Sprachausgabe.

- Algorithmen: Behandlung von Gemüse
- einfach/mittel/komplex:
  2/4/6 Tests (Adjektive), 4/6/8 Prozeduraufrufe (Verben).
- Keine Schleife, kein komplexer Test, nur je 1 Prozeduraufruf.
- Zu beantwortende Fragen: 12/36/60 Stück
  z.B.: „What are the values (true/false/unknown) at all decisions in the algorithm when the vegetable is boiled?“


```
PROC                 |         IF HARD
  IF GREEN           |           THEN
    THEN             |             GRILL
      IF CRISPY      |           ELSE
        THEN         |             BOIL
          STEAM      |         ENDIF
        ELSE         |       ELSE
          CHOP       |         BAKE
      ENDIF          |     ENDIF
    ELSE             | ENDIF
      FRY            |END PROC
      IF LEAFY       |
        THEN         |
```


3.6.4 Ergebnis

Alle Hypothesen bestätigt:

1. Richtige Antworten z.B. bei komplexem Algorithmus 98.7% (Flußdiagramm) statt 91.7% (Pseudokode), Konfidenz in allen drei Fällen über 99.6%.
2. Zutrauen in Antworten: 3.94 Punkte statt 3.71 Punkten. Konfidenz > 99.9%.
3. Zeit: bei Flußdiagrammen niedriger für mittel/komplex (Konfidenz über 97%), nicht signifikant für einfach.
4. Blicke: 0.72 statt 0.82. Konfidenz > 99.5%.
5. Überlegenheit Flußdiagramme: für komplex höher als für einfach.

3.6.5 Diskussion

Handwerklich gut aufgebaut, aber unrealistisch und deshalb irrelevant.

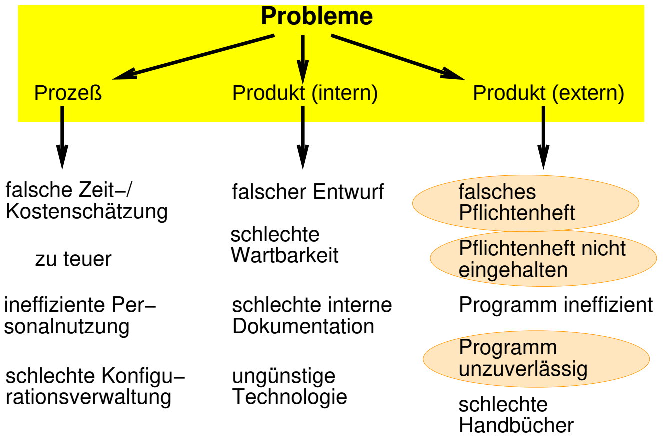
1. Algorithmen sehr kurz (max. 14 elementare Einzelteile).
2. Keine Schleife.
3. Pseudokode dient zu intuitivem Verständnis.
   Unmöglich wegen zufälliger Erzeugung der Algorithmen.
4. Welchen Sinn hat Programmverstehen von sinnlosem Programm?
5. Keine tiefergehenden Fragen (nach Zweck), nur nach Effekt.
6. Fragenkataloge zu umfangreich.
   (unter 1 mal Draufgucken pro Frage!)

Bitte sowas nicht nachmachen.

3.7 Zusammenfassung

- Wissensarten: Modell, Methode, Werkzeug.
- Arten von Beiträgen.
- Forschungsmethoden.
- Forschungsfelder und -fragen.
- Hoher Aufwand, viele Variablen.
- Experimente:
  Fragestellung, Kontrolle von Variablen, Zufallsstichproben, Vergleichsgruppen, Versuchspläne, statistische Auswertung, externe Gültigkeit, Veröffentlichung.
- Beispiele: N-Versions-Programmierung, Flußdiagramme.

4 Software- und Systemsicherheit



4.1 Einordnung und Ziele

- SW immer Teil von Systemen.
- Oft sicherheitskritisch.
- Trennung SW/Restsystem bei Sicherheitsbetrachtung unsinnig.

Systeme (und Software) müssen sicher sein!

Denke in Systemen!

4.2 Begriffe

- **Sicherheit (safety)**: Abwesenheit von Unfällen oder von unakzeptablen Risiken.
- **Unfall (accident)**: Unerwünschtes und ungeplantes Ereignis mit Verlust.
- **Risiko (risk)**: Wahrscheinlichkeit eines Unfalls mal Schadenshöhe.
  Qualitative Betrachtung meist vorzuziehen!
- **Versagen (failure)**: Fehlverhalten (bzgl. Spezifikation) eines Systems.

- **Zuverlässigkeit (reliability)**: Funktionieren gemäß Anforderungen unter allen vorgesehenen(!) Bedingungen.
- **Robustheit (robustness)**: Funktionieren ohne Unfall auch unter unvorhergesehenen Bedingungen.
- **Fehler (error)**: Ereignis: Entstehen eines Mangels.
- **Gefahr (hazard)**: Menge von Bedingungen, die gemeinsam mit anderen Bedingungen in der Umgebung unweigerlich zu einem Unfall führen.
- **Vorkommnis (incident, near miss)**: Ein Ereignis ohne Verlust, aber mit Potential für Verlust unter anderen Umständen.

- **Mangel, Defekt (fault, defect)**: Struktureller Fehler.
- **Verletzlichkeit (vulnerability)**: Schwachstelle im Entwurf eines Systems, an der Mängel besonders leicht zu Versagen führen können.
- **Schutz (security)**: Widerstandsfähigkeit gegen absichtliche Angriffe.
- **Schwaches Glied (weak link)**: Ereignisse, die die Funktionsfähigkeit beeinträchtigen oder zerstören.

4.3 Ansätze

4.3.1 Grundregeln

1. Baue Sicherheit von vornherein ein.
2. Betrachte das System als Ganzes, nicht seine Teile.
3. Nicht nur Systemversagen ist eine Gefahr.
   Zuverlässigkeit ist nicht gleich Sicherheit.
4. Verlaß Dich nicht auf Erfahrungen und Standards.
   Können die Sicherheit nicht garantieren.
   Analysiere das System!
5. Verwende qualitative statt quantitative Methoden.

6. Konflikte treten auf, Abwägungen nötig.
7. Vorrangfolge beim Umgang mit Gefahren:
   1. Gefahr eliminieren („intrinsische Sicherheit“),
   2. Gefahr reduzieren,
   3. Gefahr beherrschen (passiv, aktiv),
   4. Schaden verringern.

4.3.2 Hierarchische Sicht von Unfällen

1. **Mechanismen**: konkreter Hergang.
   Beschreibung.
2. **Bedingungen**: relevanter Umgebungszustand.
   Verstehen.
3. **Urgründe (root causes)**: allgemeine Umfeldbedingungen.
   Vermeiden.

Urgrund oft Mangel an Risikobewußtsein (z.B. Titanic).
Oft als Folge guter Sicherheitsvorkehrungen!
Nutze Ebenen 2 *und* 3 (leider 3 oft schlecht zugänglich).

4.3.3 Allgemeines Vorgehen

Ausgangspunkt: Klarheit über Anforderungen für Normalbetrieb.

1. Gefahrenbestimmung.
2. Risikoanalyse.
3. Entwurf.
4. Entwurfssicherheitsprüfung.
5. Bau.
6. Abnahmesicherheitsprüfung.

4.3.4 Gefahrenbestimmung

Allgemeine Prüfliste:

- Nutze historische Daten.
- Nutze veröffentlichte Gefahrenchecklisten, Standards, Vorgehensbeschreibungen.
- Untersuche Energiequellen, -flüsse, Materialien und Maßnahmen zu ihrer Kontrolle.
- Untersuche Mensch-Maschine-Schnittstelle.
- ...

- Untersuche Systementwicklung (insbes. SW-System):
  - Systemkonzept, Annahmen und Modelle korrekt?
  - Anforderungsanalyse vollständig und korrekt?
  - Spezifikation und Entwurf gut?
  - Implementation und Test sorgfältig?
  - Infrastruktur (Betriebssysteme, Übersetzer, etc.) geeignet und zuverlässig?
  - Weiterentwicklung bedrohlich (Fehlereinfügung, Komplexitätszunahme, Probleme bei Versions- oder Technologiewechsel)?
- ...

- Untersuche alle normalen und besonderen vorgesehenen Betriebszustände, insbesondere Übergangszustände wie Systemstart, -neustart, -stop, -test, -zusammenbruch, -wartung, -reparatur.
- Untersuche denkbare ungewöhnliche Umstände.
  - Unerwartete Umwelteinflüsse.
  - Infrastrukturversagen.
  - Hardwareversagen.
  - Mißbenutzung: Fehlinstallation, Fehlbedienung, Benutzung zu falschem Zweck, Langsamkeit, Mißachtung von Meldungen, vorsätzlicher Mißbrauch.

- Beachte langfristige Perspektive: Änderungen an
  - System,
  - technischem Umfeld,
  - organisatorischem/sozialen Umfeld.
- Gehe schrittweise durch Gesamtprozeß des Systems, suche:
  - Probleme,
  - Gegenmaßnahmen,
  - Rückfalloptionen für schlimmsten Fall.

4.4 Probleme und Hindernisse

- Kosten. Wirklich nötig?
- Negativer Geruch.
- Haltung: irgendwelche Sicherheitsmaßnahmen sind gleich vollständige Sicherheit.
- Haltung: Wir kriegen es ja doch nie sicher.
- Nicht sexy. Schwer zu verkaufen.
- Haltung: Sicherheit ist was für Spießer.

4.4.1 Risiken der Risikoanalyse

Nur drei Möglichkeiten, alle schlecht:

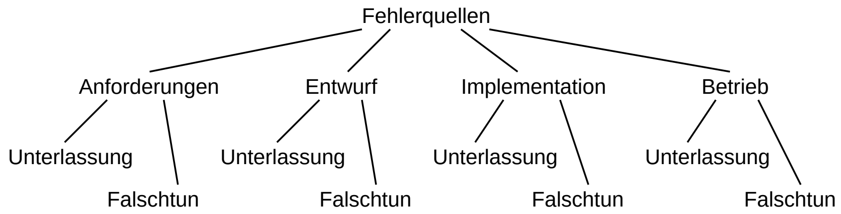
- Risiken unterschätzt:
  Schaden tritt ein.
- Risiken überschätzt:
  Vermeidbare Kosten. Verlust von Chancen.
- Risiken richtig eingeschätzt:
  Nutzen nicht beweisbar. Nachlassendes Risikobewußtsein.

4.4.2 Software-Mythen für sichere Systeme

1. Computer sind billiger als frühere Systeme.
   Hohe Kosten für sichere Software.
2. Software ist leicht zu ändern.
   Aber schwer korrekt zu ändern.
3. Computer sind zuverlässiger als frühere Geräte.
   HW ja, SW nein.
4. Hohe Zuverlässigkeit gleich hohe Sicherheit.
   Nein wegen falscher Anforderungsdefinition etc.
5. SW-Wiederverwendung steigert die Sicherheit.
   Vielleicht. Steigert aber auch Komplexität.

4.5 Beispiele für Sicherheitsprobleme

4.5.1 Fehlertaxonomie



Herkunftsarten von Fehlern, die zu Sicherheitsproblemen führen

4.5.2 Beispiele: Entwurfsprobleme

Dezember 1986, ARPAnet-Ausfall:
7 Verbindungen im Arpanet (heute Internet) zwischen Neu-England und dem Rest zugleich ausgefallen.
Entwurfsfehler, Unterlassung.

Osprey Schwenkrotorflugzeug abgestürzt:
Zwei von drei Rollraten-Gyrosensoren falsch verkabelt. Andere flogen mit einem falsch verkabelten ohne Warnung.
Entwurfsunterlassung und Implementationsfalschtun.

April 1990, U-Bahn ohne Fahrer:
Ausgestiegen wg. klemmender Tür. Zug fuhr los, weil Knopf festgeklebt.
Entwurfsunterlassung und Betriebsfalschtun (Aussteigen war unerlaubt).

Kenne Deine Benutzer!

1985–1987, Therac-25:
Sicherheitsprüfungen manchmal ausgelassen (SW-Fehler).
Falscher Zustand nach Editieren am Bildschirm möglich.
Mehrere Sicherheitsverriegelungen in HW von Therac-20 durch SW ersetzt.

Mehrfach 8000 bis 25000 rad anstatt 80 bis 200 rad
Mehrere Tote und Schwerverletzte.
Erste Korrektur: Cursortasten zum Editieren mechanisch unbrauchbar gemacht.
Entwurf und Implementation, Falschtun.

Tödliche Herzschrittmacher:
Mikrowellen setzen auf 214 Schläge pro Minute.
Ähnlich durch Diebstahlsicherung in einem Geschäft.
Arzt schaltet aus bei Umprogrammieren.
Anforderungen/Entwurf/Implementation?, Unterlassung.

März 1979, 5 Kernkraftwerke abgeschaltet wegen Simulationsfehler:
Simulationsprogramm Shock II hatte Fehler, erst lange nach Bau entdeckt. Einfache Summe anstatt Summe von Absolutwerten.
Entwurf, Falschtun.
Simulation kann nicht validiert werden. Vorgehen akzeptabel?

Hundertjährige:
Mary Bandar bekam Einladung zum Besuch des Kindergartens, war 104.
G.C. Blodgetts Autoversicherung dreimal so teuer wie zuvor, als er 101 wurde. „Teenager sind Leute unter 20.“
Was passiert Sylvester 1999 ???
Anforderungen/Entwurf/Implementation, Unterlassung.

April 1993, Kühlunfall im Stahlwerk:
Normalzeituhr als Zeitgeber. Kühlen von Stahl eine Stunde verkürzt als Sommerzeit begann. Großer Sachschaden.
Entwurf, Unterlassung.

Fortran auf Deutsch:
Ausgabe an *units* (z.B. 6 = Standardausgabe) und an *devices* (z.B. 6 = Platte). Verwechslung: auf Deutsch beides „Einheit“.
Klasse?

TENEX Paßwortprüfung:
Zeichenweise Prüfung überlistet durch Beobachten des Seitenwechsels.
Entwurf, Unterlassung.
Willentliche Angriffe, Thema für sich.

Unbefugte Dienstbenutzung, Bereicherung, Spionage, Dienstverhinderung, Zerstörung.

McDonalds Toaster:
Zeitgeber der McMuffin-Toaster induzierten Spannungsspitzen im Stromnetz. Uhren für Arbeitszeitmessung gingen bis 4 Stunden pro Tag vor. Gehälter zu hoch ausbezahlt. 5000 Toaster installiert, bevor Problem gefunden wurde.
Entwurf, Unterlassung.
Funktionierendes, zuverlässiges System (hier: Uhren) plötzlich gefährlich. Zuwenig robust.

Hohe Verbrauchsrechnungen:
Tampa, Florida: Stromrechnung für einen Monat 4.062.599 Dollar. Angebot von Ratenzahlung. Korrekt: 146,76 Dollar.
Bedienungsfehler.
Malcolm Graham: Wasserrechnung für einen Monat über 35 Millionen Liter. Neue Wasseruhr installiert und falsch eingestellt, Computer hatte Differenz akzeptiert.
Entwurf, Unterlassung.

BRAVO editor:
Kommando `edit` für Änderungsmodus. Im Änderungsmodus: `everything delete insert t`.
Rettung durch Tastaturlogbuch möglich.
Entwurf, Unterlassung.

Space Shuttle Missionsabbruch-Problem:
Simulatortest: Flugabbruch, doch nicht, später doch Flugabbruch, Endlosschleife.
Entwurf, Unterlassung.

4.5.3 Beispiele: Herstellungsprobleme

April 1981, Space Shuttle Startverzögerung:
Extrem subtiler Softwarefehler beim Initialisieren des Bordcomputersystems (Chance 1 zu 67).
Resultat: fünfter Rechner ein Zeitschritt hinter vier Hauptrechnern, weist deshalb alle Daten zurück. (Wird erst kurz vor Start zugeschaltet, 30 Stunden Countdown waren OK.)
Start verzögert um zwei Tage.
Implementationsfehler, Falschtun.

April 1993, Space Shuttle Startverzögerung:
11 Sekunden vor Start Countdown abgebrochen, weil angeblich ein Ventil nicht geschlossen hatte. SW hatte Sensordaten falsch interpretiert. Nächster Startversuch erst dreieinhalb Monate später.
Implementationsfehler, Falschtun.

Londoner Rettungsdienst:
Einsatzplanungssoftware für Krankenwagen nach Schwierigkeiten bei Entwicklung trotz mehrerer Ausfälle bei Tests in Betrieb genommen.
Resultat: Totalzusammenbruch der Einsatzplanung, Verzögerungen bis 11 Stunden, etwa 20 Tote.
Ursachen: System zu komplex, ungenügende Ausbildung der Leit- und Wagenmannschaften.
Entwurfsfalschtun und Betriebsunterlassung.

Versehentliche Bankbuchung:
Bank of New York buchte 32 Milliarden Dollar zuviel.
Resultat: Bank mußte sich 24 Milliarden Dollar für einen Tag leihen, bis der Softwarefehler gefunden und behoben war; Zinsen 5 Millionen Dollar!
Ursache: 16-bit Zähler übergelaufen. Die meisten Zähler waren 32 bit groß.
Implementation, Falschtun.

4.5.4 Beispiele: Betriebsprobleme

August 1991, Rakete gesprengt:
Rakete vom Kurs abgekommen, absichtlich im Flug gesprengt.
Ursache: Beim Laden der Software in die Rakete falschen Knopf gedrückt: Bodentestversion anstatt Flugversion geladen.
Entwurfsunterlassung und Betriebsfalschtun.

April 1993, Intercity-Zusammenstoß:
Karfreitag stieß aus Berlin kommender Intercity mit nach Berlin fahrendem anderen Zug frontal zusammen. Während Arbeitswoche wegen Bauarbeiten einspuriger Betrieb. Karfreitag eigentlich zweispurig, Fahrdienstleiter hatte aber Steueranlage nicht umgestellt.
Steuercomputer stellte korrekt Signal für Intercity auf Halt.
Fahrdienstleiter glaubt an Systemfehler, benutzt Zusatzsignal ohne Nachzufragen. Übersah, daß sich nicht fahrplanmäßiger Zug aus Gegenrichtung näherte.
Resultat: drei Tote und über 20 Verletzte, obwohl nur 30 km/h!
Betrieb, Unterlassung.

Chemiefabrik explodiert:
Operateur tippt Rezept. Verwechselt Tank 632 (ein Harz) mit dem Tank 634 (Dicyclopentadien). Vergißt das vorgeschriebene Nachprüfen.
Resultat: Überhitzungsalarm verständigt Feuerwehr, die ist aber überfordert. Explosion. 3 Tote, 11 Verletzte, Explosionsteile im Umkreis von 1 Kilometer, mehrere 10 Millionen Mark Sachschaden.
Operateur war erst drei Monate auf Arbeitsstelle, befand sich noch in Ausbildung.
Betrieb, Falschtun (des Operateurs) und Unterlassung (der Firma).

April/Mai 1989, Fehler in Fahrstuhltür:
Fehler in Verriegelungsschaltkreisen: Losfahren, obwohl nur eine der zwei Türen geschlossen. Zweimal im Abstand von 8 Wochen im gleichen Fahrstuhl ein Mensch getötet. Fehler bereits 11 Jahre zuvor erkannt, 1978 Reparaturvorschrift mit 1 Draht angegeben, jedoch nie ausgeführt (Gebäude: 1980 Eigentümerwechsel, 1988 Fahrstuhlwartungsfirmawechsel, keine Wartungslogbücher geführt).
Betrieb, Unterlassung.

4.5.5 Analyse

Augenfällig:

1. Viele Probleme wegen Unterlassungen.
   Checklisten helfen.
2. Meist *nicht* Programmierfehler.
3. Darf Software nicht isoliert betrachten.

Sie sind zuständig!

4.6 Allgemeine Suchtechniken zur Gefahrenbestimmung

Techniken der Anforderungsbestimmung oft geeignet. Unterscheide Verkettungstechniken (zeitorientiert) und Zerlegungstechniken (strukturorientiert).

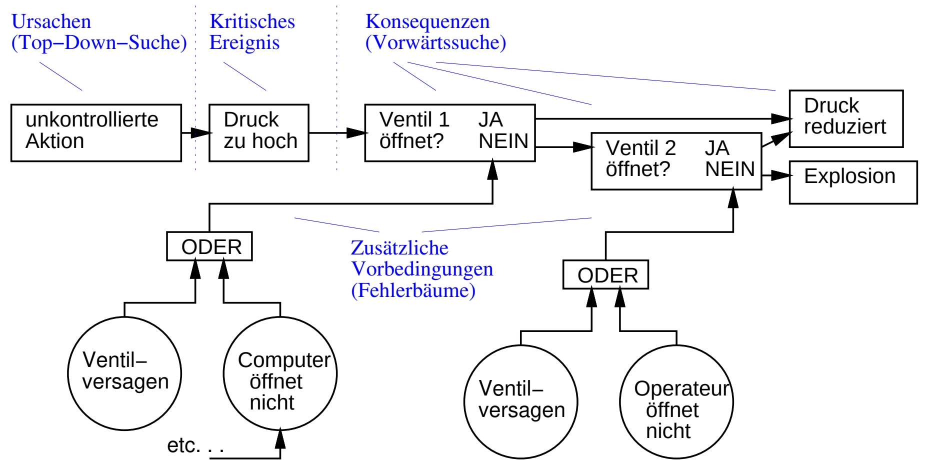
- **Vorwärtssuche:**
  Viele „uninteressante“ Folgezustände. Nur für sehr enge Bereiche nutzbar.
  Oft „Was geschieht, wenn dieses oder jenes schiefgeht?“.
- **Rückwärtssuche:**
  „Was könnte zu einer solchen Situation führen?“.
  z.B. Fehlerbaumtechnik. Ungünstig bei mehrfachen Ursachen.

- **Top-Down-Suche:**
  Zerlegung Zustand in Teile gemäß Systemstruktur.
- **Bottom-Up-Suche:**
  Bestimmt Wirkungen eines Zustands einer Komponente auf Gesamtsystem.
  Schwierig wegen Wechselwirkungen.
- **Kombinierte Suche:**
  Wechsle Methode von Schritt zu Schritt.
  Gespür und Erfahrung nötig.

4.7 Ursachen-Folgen-Analyse

Beispiel für eine kombinierte Suchtechnik:
Ursachen-Folgen-Analyse (cause consequence analysis)

- Ausgangspunkt: kritisches Ereignis.
- Bestimme sowohl Ursachen (Top-Down-Suche oder Rückwärtssuche) als auch mögliche Konsequenzen (Vorwärtssuche).
- Weitere kritische Ereignisse in Folgen: Bestimme zusätzliche Vorbedingungen z.B. mit Fehlerbäumen.



4.8 Gefahrenbestimmung mit formalen Notationen

Gründlichere Analyse.

4.8.1 Darstellung des Systems als endlicher Automat

- Reale Systeme: kontinuierliche Zustände.
  z.B. Druck.
- Abstraktion: unterscheide nur qualitativ verschiedene Zustände.
  z.B. Druck zu hoch, Druck akzeptabel.
- Verringert auch Zahl der Übergänge.
- Finde gefahrenrelevante Zusammenfassungen.

4.8.2 Gefahrenbestimmung im endlichen Automaten

- Vorwärtssuche findet alle gefährlichen Zustände.
  Unpraktikabel.
- Rückwärtssuche prüft Erreichbarkeit.
  Unpraktikabel.
- Rückwärtssuche nur bis zu möglicher Modelländerung: mache gefährlichen Zustand unerreichbar.
- Nachteil: Evtl. unnötige Änderungen.

4.8.3 Probleme dieses Ansatzes

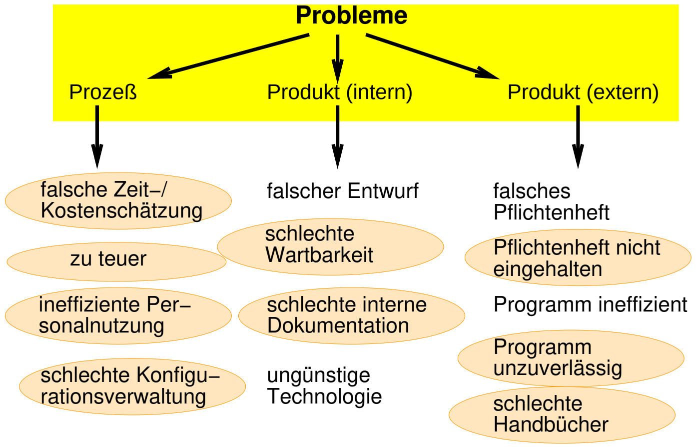
- Größe des Zustandsraums, Anzahl von Pfaden.
- Korrektheit des Modells fraglich.
- Validation durch Simulation teilweise möglich.

Trotzdem gründlicher.

4.9 Zusammenfassung

- Safety first!
- Gefahrenbestimmung, Risikoanalyse, Entwurf, Prüfung, Bau, Prüfung.
- Globale Sicht nötig.
- Gefahrenbestimmung: Prüflisten, Suchverfahren.
- Gefahren eliminieren, reduzieren, beherrschen, Schaden verringern.
- Implementationsfehler sind kein Hauptproblem.
- Zuverlässigkeit ist nicht gleich Sicherheit.

5 Softwareprozeß-Verbesserung



5.1 Einordnung und Ziele

5.1.1 Warum ist die Softwareproduktion so schlecht?

- **Anforderungen unklar**:
  Mißverständnisse und Fehler oder geringer Systemnutzen.
- **Entwurf schlecht**:
  Hoher Kommunikationsbedarf, Chaos.
- **Werkzeugunterstützung gering**:
  Information zu teuer, schlecht verfügbar, unzuverlässig.
- **Notationen ungenau**:
  Mißverständnisse, Korrektheitsbeweise unmöglich.
- **Softwareprozeß schlecht**.

Prozeßverbesserung überspannt übrige Themen.

5.1.2 Ziele von Softwareprozeß-Verbesserung

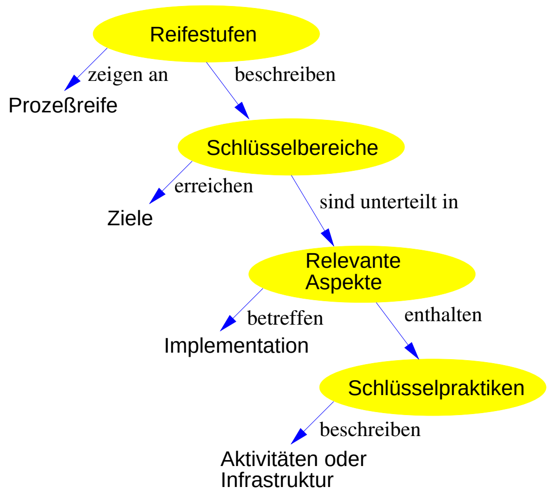
Grundlagen für guten Prozeß (für große Projekte!):

- **Wohldefiniertheit**:
  sonst Falsch-, Nicht-, Mehrfacharbeit.
  Information nicht verfügbar oder unverständlich.
- **Quantitatives Verstehen**:
  Einzige objektive Grundlage für Verbesserungen.
- **Kontinuierliche Änderung**:
  sonst Prozeß bald obsolet oder undefiniert.
  Software Engineering Process Group (SEPG).

5.2 Ansätze

5.3 Das Capability Maturity Model (CMM)

- SEI: staatlich gefördert, SWT-Praktiken verbessern.
- Annahme: Produktqualität hängt an Prozeßqualität.
- CMM beschreibt Stufen der SW-Prozeßreife.
- Beschreibungsmodell als Maßstab, Leitschnur für Verbesserungen, Entwicklungspfad.
- 5 Stufen: chaotisch bis kontinuierlich selbstverbessernd.
- Schlüsselbereiche, Schlüsselpraktiken.



Organisation des CMM

Relevante Aspekte sind:

- Selbstverpflichtung.
  Vorbereitung und Aufrechterhaltung der Maßnahmen.
- Fähigkeit.
  Vorbedingungen zur kompetenten Durchführung (Ressourcen, Organisationsstrukturen, Ausbildung).
- Aktivitäten.
- Messung und Analyse.
  Technische Untersuchung von Status und Effektivität.
- Implementationsüberprüfung. Management-Untersuchung der Übereinstimmung mit Vorgaben.

Einzelne Ziele gehen in folgende Richtungen:

- **Planung**: Vermeide unerwartetes.
- **Technische Aktivitäten**.
- **Management**: Zuständigkeiten für Probleme.
- **Zustimmung** aller zu Entscheidungen.

5.3.1 Stufe 1: ad-hoc

- Prozeß meist völlig instabil.
  Typisch: Bei Krisen nur noch Kodieren+Testen.
- Qualität und Fertigstellung unvorhersagbar.
- Hervorragende Manager können stabilisieren,
  Stabilität dann aber personenabhängig.
- Erfolge nur durch gutes Personal und heroischen Einsatz.

Weitaus häufigste Stufe

5.3.2 Stufe 2: wiederholbar

- Anforderungsverwaltung.
  Explizite Verwaltung, Arbeit mit Anforderungen konsistent halten.
- Software-Projektplanung.
  Planungen dokumentieren, Betroffene stimmen zu.
- Software-Projektverfolgung und -überblick.
  Soll/Ist-Verfolgung, ggf. Gegenmaßnahmen, Betroffene stimmen zu.
- Software-Unterauftragsverwaltung.
  Qualifikation prüfen, Pflichten einigen, Kommunikation.

- Software-Qualitätssicherung.
  Prüfen, Betroffene informieren, Management greift ein bei Konflikt.
- Software-Konfigurationsverwaltung.
  Änderungen überwachen, Beteiligte informieren.

5.3.3 Stufe 3: definiert

- Schwerpunkt auf organisationsweiten Prozeß.
- Prozeßdefinition.
  Standardprozeß warten, Anwendungsinformation sammeln.
- Ausbildungsprogramm.
- Integriertes Softwaremanagement.
  Projektspezifische Prozeßvariante definieren und benutzen.
- Koordination zwischen Gruppen.
  Verpflichtungen gemeinsam zustimmen, Probleme klären.
- Peer Reviews.

5.3.4 Stufe 4: verwaltet

- Quantitative Prozeßsteuerung.
  Leistung im Projekt überwachen, Leistung Standardprozeß bekannt.
- Software-Qualitätsmanagement.
  Meßbare Ziele f. Prozeßqualität definieren, Fortschritte überwachen.

5.3.5 Stufe 5: optimierend

- Defektvermeidung.
  Übliche Gründe von Fehlern finden, systematisch beseitigen.
- Verwaltung von Technologieänderung.
  Neue Technologien bewerten, ggf. einführen.
- Verwaltung von Prozeßänderung.
  Kontinuierliche Verbesserung, organisationsweite Beteiligung.

5.3.6 Managementsicht auf die Reifestufen

1. Prozeß undurchsichtig. Schwarze Magie. Kaum Einfluß.
2. An Meilensteinen durchsichtig. Reaktion auf Probleme.
3. Durchsichtig. Risiken grob abschätzbar.
4. Quantitativ verständlich. Genaue Vorhersagen.
5. Verstehen auch von Prozeßänderungen.

Prozeßreife ist zum Großteil Durchsichtigkeit.

5.4 ISO 9000

Anforderungen an Qualitätssicherungssysteme.

- ISO 9001: Herstellung und Wartung von SW.
  ISO 9000-3: Anwendungsrichtlinien für ISO 9001.
- Perspektive: Anforderungsbeschreibung.
- 20 Klauseln, kein Stufensystem.
- 3 Klauseln ohne direktes CMM-Pendant:
  - 4.7 control of customer-supplied product.
  - 4.15 handling, storage, packaging, delivery.
  - 4.19 servicing.

- Organisation ISO 9001 konform:
  erfüllt das meiste von Stufe 2, vieles von 3.
- Organisation auf Stufe 2:
  kann mit wenig mehr ISO 9001 erreichen.
- CMM und ISO 9001 kein Widerspruch.
  Für Verbesserungen CMM besser, weil genauer.

5.5 Bootstrap

Europäische CMM-Variante, berücksichtigt ISO 9001/9000-3.

5.6 Spice

Software Process Improvement and Capability dEtermination.

- Standard der ISO JTC1/SC7/WG10.
- Bewertungsprofil für konkretes Exemplar von Prozeß.
- Pro Anforderung: **(Graduell!)**
  - Angemessenheit der Maßnahmen.
  - Übereinstimmung mit Standard.

5.7 Weiterentwicklung

Alle Ansätze im Fluß.

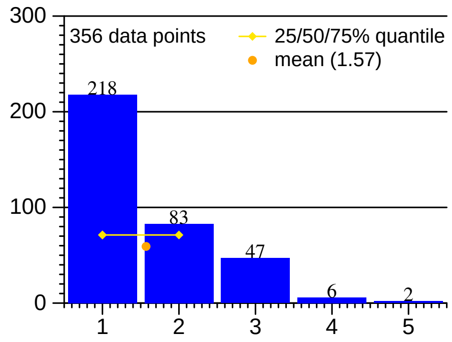
z.B. CMM: Kurs 1988, Buch 1989, 0.2 1990, 0.6/1.0 1991, 1.1 1993.

Tendenzen:

- **Vereinfachung**:
  Weniger Konzepte, konsistentere Darstellung.
- **Orthogonalisierung**.
- **Detaillierung**.
- **Konkretisierung/Objektivierung**:
  Wenig abstrakten Anweisungen oder subjektive Formulierungen.

- Andere CMMs (dieses: „for software“):
  engineering, software acquisition, people management, human interaction, systems engineering.
- Integration steht noch aus.
- Ebenso Integration mit Spice, ISO 9001.
  Erfolgt wohl bald.

5.8 Ergebnisse



Umfrage unter 167 Personen aus 61 Institutionen.

Ein bis drei Jahre seit CMM-Bewertung.

- Rücklauf 83% (138 Fragebögen).
- 47 SW-Ingenineure, 47 Manager, 44 SEPG-Mitglieder. Keine systematischen Unterschiede zwischen Gruppen.

Häufigkeit	Stufe 1	Stufe 2	Stufe 3
alte Bewertung	115	14	9
Eigeneinschätzung	87	36	15

4 Antwortmöglichkeiten von „exzellent“ bis „schlecht“.

Antworten „gut“ oder „exzellent“:

Frage	Stufe 1	Stufe 2	Stufe 3
Budgetplan einhalten	40%	56%	68%
Zeitplan einhalten	40%	56%	80%
gute Produktqualität	75%	80%	100%
hohe Produktivität	53%	68%	88%
hohe Kundenzufriedenheit	77%	69%	100%
Zufriedenheit des Personals	23%	50%	61%

Andere Aussagen über CMM-Bewertung:

- Aussagen zu Schwächen sehr zutreffend: 63%.
- Aussagen zu Stärken sehr zutreffend: 38%.
- War der Mühe wert: 74%.
- Empfehlungen zu ehrgeizig: 29%.

Was wurde erreicht?

- Beschränkte oder fast keine Verbesserungen: 44%.
  Mäßige Verbesserungen: 26%.
  Substantielle Verbesserungen: 31%.
- Aktionsplan aufgestellt: 96%.
  Aktionsteam aufgestellt: 89%.
- Änderungen in Pilotprojekten: 53%.
  Änderungen organisationsweit: 53%.

Erfahrungen kleiner Unternehmen und Organisationen

- **Unerreichbare CMM-Anforderungen.**
  insbesondere Abtrennung organisatorischer Gruppen und Rollen.
- **CMM-Schwächen.**
  zu wenig detailliert und anschaulich, schwierig zu verstehen und anzuwenden.
- **Implementationskosten.**
  Kostenabwägungen für Amortisation und Konkurrenzfähigkeit. Vor allem Ausbildung teuer.

- **Ressourcen.**
  Geld, Zeit, Personal schwer bereitzustellen.
- **Kultur.**
  Haltung der Mitarbeiter, hergebrachte Arbeitsweise. Kunden nicht aufgeschlossen.
- **Kunden.**
  Unterauftragsnehmer: eigener standardisierter Prozeß und Datensammlung schwer zu erreichen.

Gesamtbild:

- Vorteile von Kleinsein:
  Geringe Trägheit, kurze Projektdauer, gute Kommunikation.
- Nachteile überwiegen jedoch:
  Kosten skalieren nicht proportional herunter.
- Dennoch oft erfolgreiche Verbesserung:
  Rahmen kreativ an Bedürfnisse angepaßt, erfolgreich die Kernideen umgesetzt.

5.9 Probleme und Hindernisse

Schwierig, evtl. unerwünschte Nebenwirkungen

Schwierigkeiten:

- 49%: desillusioniert.
  72%: Bemühungen beeinträchtigt (Kosten).
- 77%: länger als erwartet.
  68%: teuer als erwartet.
- 1 bis 2 Jahre zur nächsten Reifestufe
- Zu wenig Hilfe beim *Wie*.

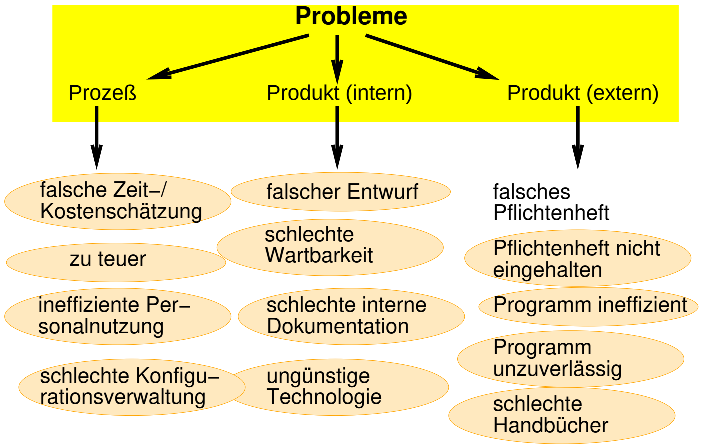
Unerwünschte Nebenwirkungen:

- Verselbständigung möglich:
  Selbstzweck statt Orientierung Produktivität/Qualität.
- Mangel an Risikobereitschaft:
  - Hohe Reife nur bei Erfahrung.
  - Erfahrung nur auf „heimischem“ Gebiet.
  - Gebietswechsel senkt Prozeßreife.
  - Neue Techniken senken Prozeßreife.
  - Deshalb Scheu vor Einführung.
- Dadurch Verlust von Geschäftschancen:
  Gewinn bei gut beherrschten Techniken meist gering;
  bei neuen oft hoch.

5.10 Zusammenfassung

- Diverse Gründe f. schlechte SW.
  Prozeßqualität umfassend wichtig, da immer Rahmen.
- Ziele: definieren, verstehen, verbessern.
- CMM: Beschreibungsmodell für Prozeßreife.
  - Stufen: ad-hoc, wiederholbar, definiert, verwaltet, optimierend.
  - Schlüsselbereiche, Ziele, Schlüsselpraktiken, relevante Attribute.
- ISO 9001
- Bootstrap, Spice

6 Der persönliche Softwareprozeß



6.1 Einordnung und Ziele

PSP: Anwendung von CMM auf 1 Person.
Ergänzung und Zugang, nicht Ersatz.

Ziel: disziplinierter, selbstoptimierender Prozeß, Stufe 5.

Vorzüge:

- Einzelpersonen flexibler zu ändern:
  schneller umzusetzen.
- Konkrete Techniken leichter anzugeben.
- Verbesserungen konkreter wahrzunehmen, Hohe eigene Kontrolle über Arbeitsmethoden:
  gute Motivation.

6.2 Ansätze

6.2.1 Grundideen

Unterschiede zwischen PSP und USP (umfassender SW-Prozeß):

- **Wenig Kommunikation**.
- **Hohe Flexibilität**:
  Keine Abstimmung mit Gruppe, kein schwächstes Glied, Prozeß einfacher.
- **Selbstähnlichkeit**:
  Datenübertragung, -uminterpretation, mehr Techniken zulässig.
- **Individualität**:
  „Standard“ ist kein Kompromiß, besser und motivierender.

6.2.2 Die Schlüsselbereiche im PSP

6.2.3 Stufe 2: wiederholbar

- Anforderungsverwaltung.
- **Software-Projektplanung.**
  Zeitplanung per Größenschätzung, Arbeitsphasen.
- **Software-Projektverfolgung und -überblick.**
  Viele Meilensteine, Plankorrekturen.
- Software-Unterauftragsverwaltung.
- **Software-Qualitätssicherung.**
  Durchsichten, Fehlerdatensammlung, geplante Tests.
- Software-Konfigurationsverwaltung.

6.2.4 Stufe 3: definiert

- Schwerpunkt auf organisationsweiten Prozeß.
- **Prozeßdefinition.**
- **Ausbildungsprogramm.**
- Integriertes Softwaremanagement.
- Koordination zwischen Gruppen.
- **Peer Reviews.**

6.2.5 Stufe 4: verwaltet

- **Quantitative Prozeßsteuerung.**
- **Software-Qualitätsmanagement.**

6.2.6 Stufe 5: optimierend

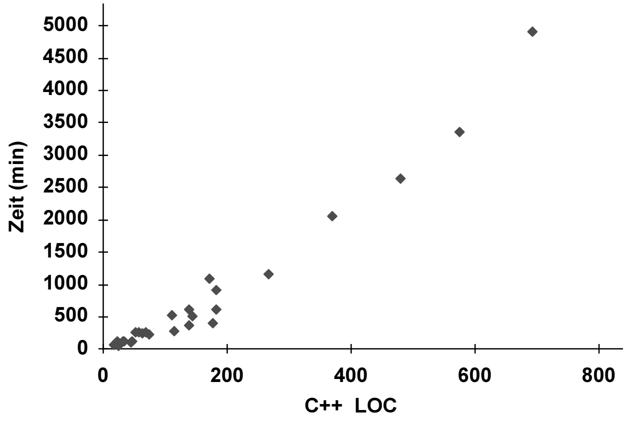
- **Defektvermeidung.**
  Analyse der Defektdaten, Selbsterziehung, Änderung der Prüflisten.
- **Verwaltung von Technologieänderung.**
  Kontinuierliches Ausprobieren, Bewerten, ggf. Einführen.
- **Verwaltung von Prozeßänderung.**
  Regelmäßige geordnete Analyse der Prozeßdaten, Abstellen von Schwächen.

6.2.7 Der PSP-Kurs

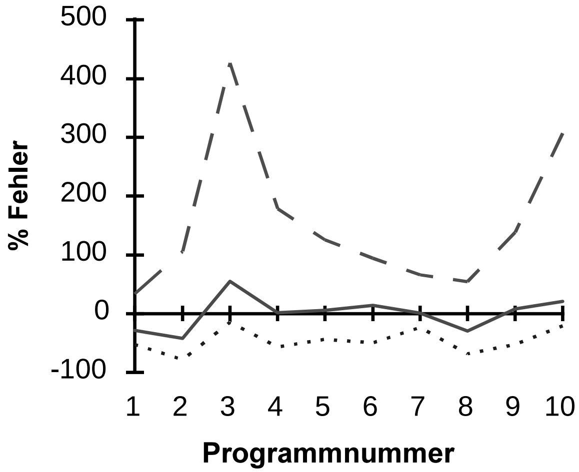
- 15 Wochen, 1 Tag pro Woche(!), erreicht Stufe 4.
- Gemäß Buch, zum Selbststudium geeignet.
- 10 Programmierübungen (lernen, üben),
  5 Prozeßübungen (definieren, analysieren, verbessern).
- PSP 0: Zeit- und Fehlererfassung.
- PSP 1: Größenmessen, -schätzen, Terminplanung.
- PSP 2: Qualitätsmanagement (Durchsichten, Vermessung, Kosten, Entwurfsprüfungsverfahren)
- PSP 3: inkrementelle Entwicklung.
- Resultat: vor- und selbstdefinierter Prozeß auf Stufe 4.
  Übergang zu Stufe 5 vorgezeichnet.

6.3 Ergebnisse

6.3.1 Planung: Grundlagen und Effekte



Korrelation Anzahl LOC mit Entwicklungszeit (C++ Code)



Schätzfehler für LOC

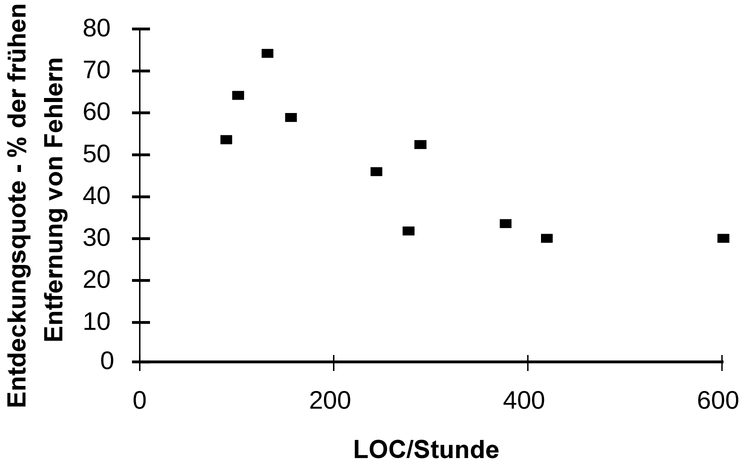
6.3.2 Qualität: Grundlagen

Annahmen des PSP:

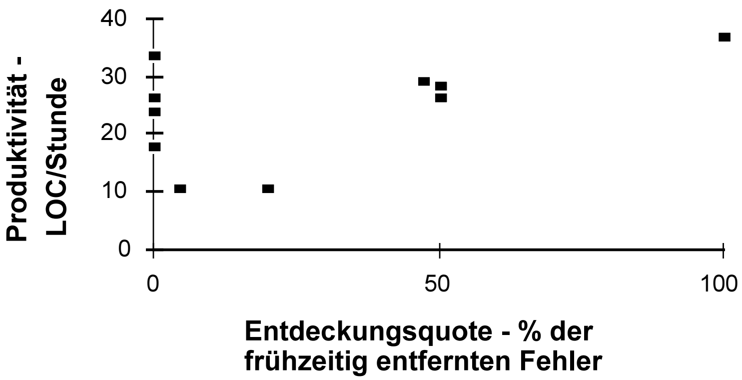
1. Jede Fehlerentfernungsstufe findet Prozentsatz.
   Also: wenig Fehler einfügen,
   viele Enfernungsstufen haben,
   frühe Stufen optimieren.

2. Auch triviale Fehler schädlich.

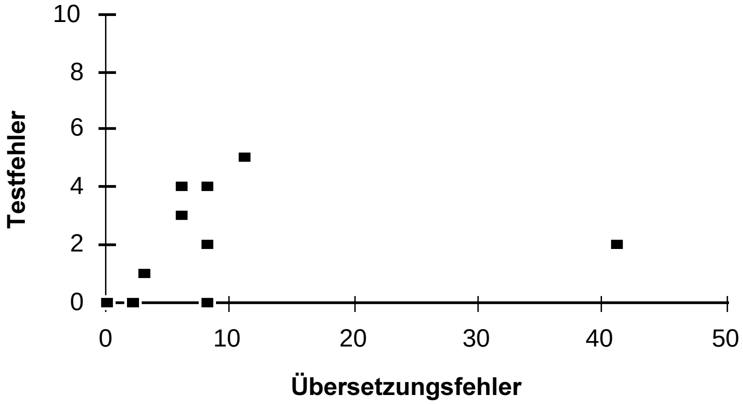
3. Durchsichten sind effizient.



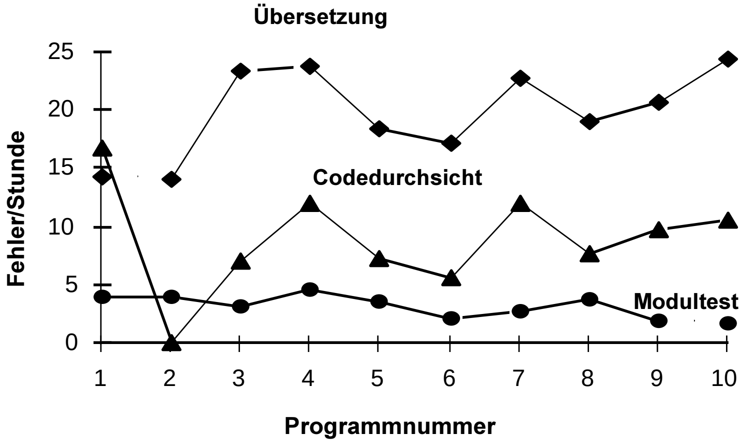
Fehlerausbeute gegen Begutachtungsgeschwindigkeit bei Student 12



Zusammenhang von Fehlerausbeute und Produktivität bei Student 20

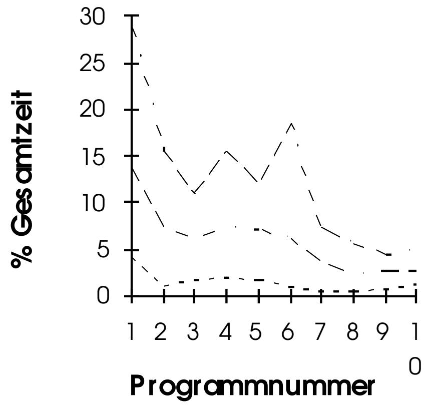


Zusammenhang der Anzahl gefundener Fehler bei Übersetzung und Test bei Student 20

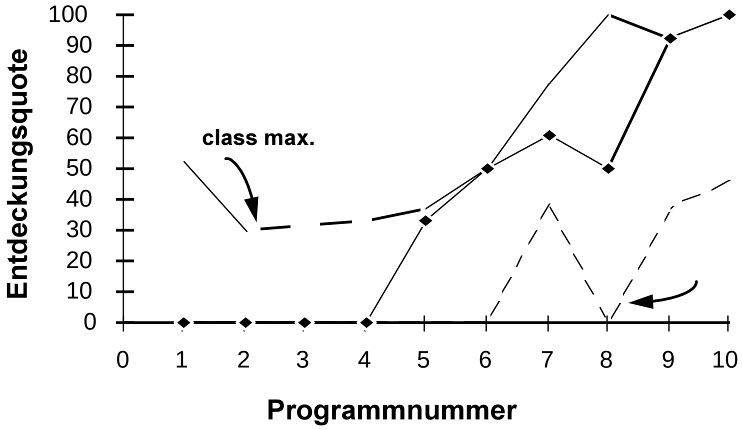


Vergleich der Fehlerentfernungsgeschwindigkeiten bei Übersetzung, Codedurchsicht und Modultest

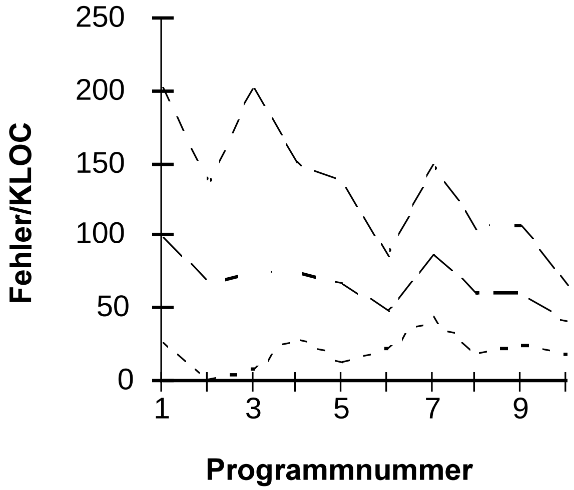
6.3.3 Qualität: Effekte



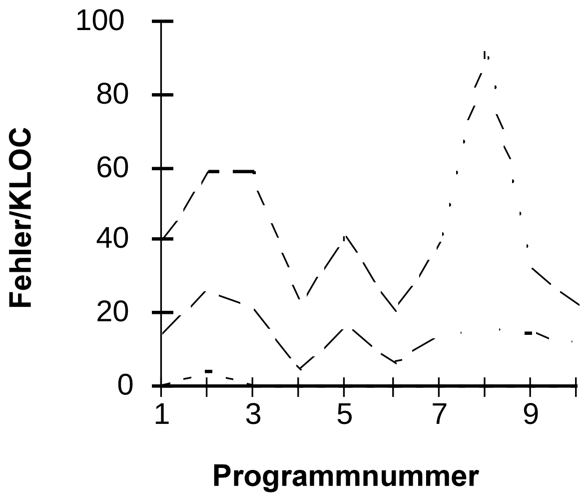
Entwicklung des Zeitanteils für die Übersetzung über die 10 Übungsprogramme des PSP-Kurses hinweg



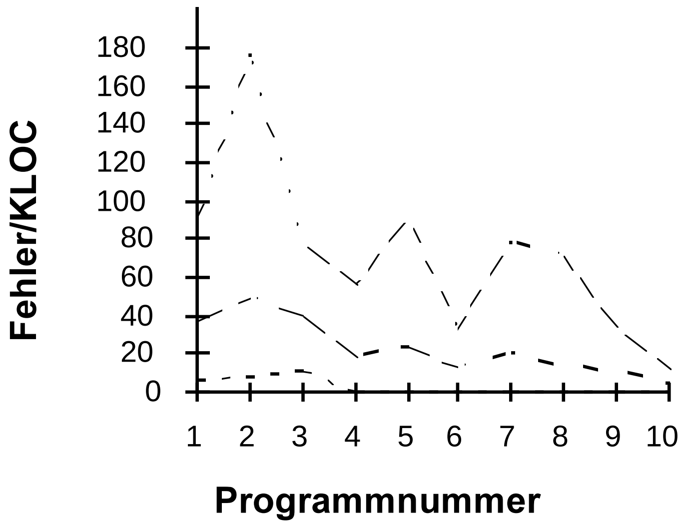
Entwicklung der Fehlerausbeute bei Student 3



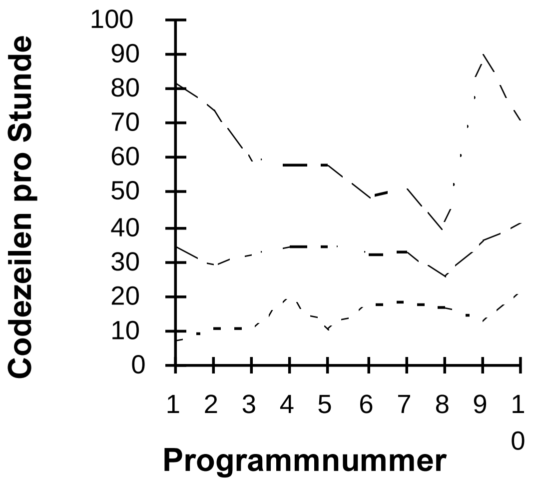
Entwicklung der Zahl beim Kodieren eingefügter Fehler im PSP-Kurs



Entwicklung der Zahl beim Entwurf eingefügter Fehler im PSP-Kurs



Entwicklung der Zahl der Testfehler im PSP-Kurs



Entwicklung der Produktivität in LOC/Std.

6.4 Probleme und Hindernisse

- Hauptproblem: Durchhalten.
- Nur möglich über
  - Einsicht in Nutzen.
  - Disziplin.
- Weiteres Problem: Betriebsblindheit.
- Abhilfe: Gegenseitige Prozeßbegutachtung.

6.5 Maße im PSP

Einfache, aber effektive Maße. Möglich, weil immer gleiche Person.

6.5.1 Produktgröße

Lines of Code: sinnvoll, weil gut schätzbar. Unterscheidung:

- Basis (base, B), zugefügt (added, A), gelöscht (deleted, D), geändert (modified, M).
- wiederverwendet (reused, R)
- total (total, T = B + A - D + R)
- neu (new and changed, N = A+M)
- neu wiederverwendbar (new reused)
- Umfangsbeschreibung: A+M, Zeitschätzung B, A, M, D, R.

6.5.2 Produktivität

Eigentlich: subjektiver Anwendernutzen pro Ressourceneinsatz.

Faktisch: Volumen pro Zeit.

Volumen nach Anzahl Anforderungen (z.B. Funktionspunkte, nicht automatisierbar) oder Menge Programm (**LOC/h**).

Aus Prozeßsicht interessiert

- Planungseignung, denn Plan ist Basis der Entscheidungen.
- Aufspüren von Problemen. LOC/h gut, weil prozeßnah. Ebenso Messen von Prozeßverbesserungen.

LOC/h nutzt Konsistenz eines Programmierers mit sich selbst.

LOC/h nutzbar bei

- Gesamtprozeß
- Kodieren
- Codedurchsicht
- Test
- Entwurf (auf Umwegen im Nachhinein)

6.5.3 Produktqualität

Leicht meßbar nur Zuverlässigkeit per statischer Defektanzahl.

- Schätzung anhand der gefundenen Fehler.
- defects/KLOC in jeder Phase.
- Schätzung für ausgeliefertes Produkt ungenau.
- total defects/KLOC
test defects/KLOC
- Annahme: konstante Defektentferungsquoten pro Phase.
- Stimmt nicht, wenn Phasen sich an Fehlermengen anpassen.
- Alternativ statistische Zuverlässigkeitsmessung möglich.

6.5.4 Prozeßqualität

Produktivität, Produktqualität siehe oben.

Planungsgenauigkeit: cost/performance index (CPI)
Planzeit/Realzeit pro Phase.
Zeit Fehlschätzung oder Prozeßinstabilität.

Qualitätsmanagement: yield (Ausbeute, Entfernungsquote)
Nützlich, obwohl nur ungenau meßbar.

6.5.5 Prozeßsteuerung

- Begutachtungsgeschwindigkeit in KLOC/h.
- Begutachtungs-Versagens-Verhältnis (*appraisal to failure ratio*, A/FR).

Optimalen Bereich identifizieren und ansteuern.

6.5.6 Qualitätsproduktivität

- **Defektentfernungsgeschwindigkeit** in entfernten Defekten pro Stunde (*defects removed per hour*).
- **Defektentfernungs-Hebelwirkung** (*defect removal leverage*, DRL(A/B))

6.5.7 Diskussion

- LOC: Vernünftig, aber teilweise spät.
- Zeit: Vernünftig.
- Fehler: gleichartige Gewichtung irreführend.
  Verschieden schwer zu finden.
  Verschieden schwere Auswirkung auf Qualität.
  Gute Einteilung ist offenes Problem.

6.6 Größenschätzung

- Zeitschätzung über Größenschätzung. Annahme: hohe Korrelation (stimmt, siehe Ergebnisse).
- Problem:
  1. Schätzung soll früh sein.
  2. Genaue Größenschätzung verlangt Details.
  3. Details nicht früh bekannt.
- Also nicht direkt schätzen, sondern Details durch Vergleich mit Bekanntem ersetzen: Ersatzgrößen.
- Dann historische Daten nutzbar.

6.6.1 Gesamtvorgehen

- Ausgangspunkt: konzeptueller Entwurf.
  Identifiziert Hauptkomponenten.
- Nur für Größenschätzung, aber wiederverwendbar.
- Schätze Basisprogramm, Änderungen, neue Objekte.
- Ergibt initiale Schätzung.
- Korrektur auf Basis historischer Daten.
  (lineare Regression)

Beispiel:

```
Base Program                                                LOC
  Base size (B)                                             295
  LOC deleted (D)                                            25
  LOC modified (M)                                           30
Projected LOC (P)
  Base_additions (BA)      (Type) (#meth) (size)            LOC
    readFromNetwork                                          40
    openAgain                                                15
  Total_base_additions ___________________________           55
  New_objects (NO)          Type   #meth   size             LOC
     Networkcomm            I/O     10      M               160
     BasicNetwork           I/O      8      S                96
     Crypt                  Comp     3      L                75
  Total_new_objects (NO)__________________________          331
```



```
Reused objects                                              LOC
     Longint                                                780
  Reused_total (R)                                          780
Estimation:
  Projected LOC (P)   BA+NO                                 386
  Estimated New&changed LOC (N)  beta0+beta1(P+M)           488
  Estimated total LOC (T)  B+N-D-M+R                       1508
  Prediction interval                                 448 - 528
  Prediction interval level (%)                              70
```

Trick: geschickte Auswahl von Stellvertretern (Ersatzgrößen).

- Zerlegung in gut vorstellbare Teile.
- Vergleich mit historisch bekanntem.

Zweite lineare Regression liefert Zeitschätzung.

6.6.2 Bestimmung der Objektgrößenbereiche

- Sorten von Klassen, die unterschiedlich geschätzt werden sollten. z.B Ein/Ausgabeklassen, Berechnungsklassen, Kontrollflußklassen, Datenklassen, Initialisierungsklassen etc. Individuell zu wählen.
- Größenklasse charakterisiert mittlere Größe der Methoden. Soll intuitiver Vorstellung entsprechen.
- Intuition richtet sich nach Häufigkeiten.
- Bei Normalverteilung: 38% innerhalb plus/minus 0.5 Standardabweichungen um Mittelwert, je 24% von plus (oder minus) 0.5 bis 1.5, je 7% Rest.
- Größendaten sind meist log-normal verteilt (Prüfen!).

Beispiel:

- 13 Klassen mit LOC/Methode im Mittel von
  10.33, 6, 21.75, 29, 8.33, 6, 22.25,
  28.33 12.33, 55.8, 20.5, 16.4, 23.
  Mittelwert = 20 und Standardabweichung = 12.84
- Logarithmen (ln) von LOC/Methode:
  2.335, 1.792, 3.08, 3.367, 2.12, 1.792, 3.102,
  3.344, 2.512, 4.022, 3.02, 2.797, 3.135
  Mittelwert = 2.802 und Standardabweichung = 0.6346

- Ergibt Bereichsgrößen (in lnLOC) von
  VS = 2.802 - 2*0.6346 = 1.533
  S = 2.802 - 0.6346 = 2.167
  M = 2.802
  L = 2.802 + 0.6346 = 3.437
  VL = 2.802 + 2*0.6346 = 4.071
- also in LOC:
  VS = exp(1.533) = 4.63
  S = exp(2.167) = 8.73
  M = exp(2.802) = 16.48
  L = exp(3.437) = 31.09
  VL = exp(4.071) = 58.62

Weitere Überlegungen:

- Genug Daten nötig.
  Sonst evtl. besser direkt schätzen.
- Objektarten evtl. unterschiedliche Streuung:
  (Humphrey, C++)

Art	VS	S	M	L	VL
Comp	2.3	5.1	11	25	54
I/O	9	12	16	22	29

- Beachte auch Wechsel des Anwendungsbereichs.
- Daten altern! (d.h. Relevanz sinkt).

6.6.3 Schätzkorrektur mit Regression

Überlegungen:

1. Systematischer Fehler: nur Methoden geschätzt,
   Attribute und Sonstiges ignoriert.
2. Schätzer machen auch systematische Fehler.
   z.B. Übersehen von Komponenten oder Komponentengröße
   meist zu gering (oder zu hoch).

Regression korrigiert systematischen Fehler.

Statistischer Fehler (Schwankung) bleibt.

Vorhersagesystem: Größe aus Größenschätzung.
Modell: geschätzt s, dann Größe a*s + b
Prozedur: lineare Regression.

6.7 Entwurfsprüfung

For every problem there is a solution
that is simple, neat, and wrong.

- Zentral im PSP: Fehlermanagement.
  Fehler vermeiden, ansonsten zumindest früh entdecken.
- Wichtigste Suchmethode: Durchsichten.
- Entwurfsdurchsicht umfaßt auch Feinentwurf,
  evtl. bis hin zu Pseudokode.
- Keine Entwurfsmethode vorgeschlagen,
  aber Entwurfsprüfungsmethoden und Entwurfsnotationen.
- Notationen: Mengenausdrücke, boolesche Ausdrücke,
  Prädikatenlogik, KV-Diagramme, endliche Automaten.

Entwurfsprüfungsmethoden:

- Zustandsmaschinenprüfung (Vollständigkeit, Eindeutigkeit).
- Entwurfskonventionen (Sorgfalt, Zusammenhang).
- Symbolische Ausführung (Programmlogik).
- Induktionsbeweis (Schleifen, Rekursionen).
- Ausführungstabellen (execution tables) als letzter Ausweg.
- Verfolgungstabellen (trace tables).
- Schleifenverifikation
- Formale Verifikation.

6.7.1 Allgemeine Überlegungen

Auswahlaspekte:

- Kosten.
  - Kritische Abschnitte im Entwurf.
  - Detailtiefe der Anwendung.
- Anwendbarkeit und Eignung der Methoden.
- Beherrschung der Methoden.
  ggf. Neuentwurf statt Prüfung

6.7.2 Zustandsmaschinen als Notation und Prüfungsmethode

- Meist zu viele Zustände für direkte Behandlung.
- Zerlegung in wenige disjunkte Teile möglich,
  bei Beibehaltung fast aller qualitativen Zustandsunterschiede.
- Dann Prüfung mit moderatem Aufwand möglich.
- Keine einfachen Regeln für Zerlegung.
  Evtl. mehrere verschiedene Zerlegungen hintereinander benutzen.

Beispiel: Stapelklasse

Bis zu `maxentries` Stapeleinträge zulässig
Operationen `empty, top, push, pop`.

Sinnvolle Zustände:

- `none`: Stapel ist leer, also `entries = 0`
  (Fallunterscheidungen bei `empty`, `top` und `pop`).

- `full`: Stapel ist ganz voll, also `entries = maxentries`
  (Fallunterscheidung bei `push`).

- `some`: Stapel ist teilweise voll
  (Fallunterscheidung bei `empty`).

Zustandsübergänge:

- Vom Zustand `none`
  in den Zustand `none`: `empty, top, pop`;
  in den Zustand `some`: `push`;
  in den Zustand `full`: kommt nicht vor.

- Vom Zustand `some`
  in den Zustand `none`: `pop` wenn `entries = 1`;
  in den Zustand `some`: `push` wenn `entries < maxentries - 1`, `pop` wenn `entries > 1`, `empty, top`;
  in den Zustand `full`: `push` wenn `entries` gleich `maxentries - 1`.

- Vom Zustand `full`
  in den Zustand `none`: kommt nicht vor;
  in den Zustand `some`: `pop`;
  in den Zustand `full`: `push, empty, top`.

Gibt Anleitung für Entwurf der Operationen:

1. **Vollständigkeit:** Jede Operation braucht für jeden Zustand eine Aktion und einen Folgezustand.
2. **Orthogonalität:** Zustände dürfen nicht überlappen.
3. **Korrektheit:** Operationen erzeugen richtige Folgezustände.
4. **Konsistenz:** Folgezustände eindeutig.

Vollständige Zustandsmaschine mit orthogonalen Zuständen heißt ordentliche Zustandsmaschine.
Schreibe Entwurf in einer Form, die das sicherstellt.

- Die Operation `empty`
  im Zustand `none`: true.
  im Zustand `some`: false.
  im Zustand `full`: false.
- Die Operation `top`
  im Zustand `none`: Fehler.
  im Zustand `some`: Oberstes liefern.
  im Zustand `full`: Oberstes liefern.
- Die Operation `push`
  im Zustand `none`: aufstapeln.
  im Zustand `some`: aufstapeln.
  im Zustand `full`: Fehler.

Vollständigkeit leicht zu prüfen.

Sehr effektive Methode!

- Steigert Disziplin bei Entwurf.
- Ermöglichst systematische Prüfung.
- Vermeidet „clevere“ Entwürfe.

Anschließend kann man meist wieder vereinfachen.

6.7.3 Entwurfskonventionen

Beitrag zur Qualitätsverbesserung

Produktkonventionen:
Vereinheitliche äußeres Erscheinungsbild.

1. Direkt qualitätsverbessernd.
2. Verletzungen zeigen Bereiche mangelnder Sorgfalt an.

Standards im Entwurfsdokument:
Verbessere Lesbarkeit und Eindeutigkeit des Dokuments

1. Direkt qualitätsverbessernd (Wartungsphase!).
2. Verletzungen zeigen Bereiche mangelnder Sorgfalt an.

Drittens evtl. Richtlinien für Gestaltung wiederverwendbarer Komponenten.

6.7.4 Symbolische Ausführung und Induktionsbeweis

Für exakt spezifizierte kurze Anweisungsabschnitte.

```
a := b + 1;        (* now a = b+1 *)
c := 2*a;          (* now c = 2b+2 *)
IF a > 7 THEN
  c := c + 1       (* now c = c+1 if b+1 > 7  else c = 2*b+2 *)
ELSE
  a := b           (* now a = b   if b+1 <= 7 else a = b+1 *)
FI;
```

liefert als Ergebnis, daß

```
a = IF NOT b > 6 THEN b ELSE b+1
b = b
c = IF b > 6 THEN c+1 ELSE 2*b+2
```

Verwandt: Induktionsbeweis, evtl. weniger formal.

6.7.5 Ausführungstabellen

Ausführung eines Programmes „von Hand“.
Simpel, universell und aufwendig.

Tabelle: Programmschritt, Bedingungswert, Umgebungswerte, Variablen.

```
Schritt      Bedingung    a    b    c
(START)                        5
a := b + 1;               6
c := 2*a;                           12
IF a > 7 THEN   false
ELSE
  a := b                  5
FI;
```


Warum kein Debugger?

1. ATs auch für höhere Abstraktionsniveaus geeignet.
   Auch bei fehlenden Teilen möglich.
2. Mehr Übersicht über Zustand.
3. Vor allem: Langsamkeit erzeugt Mitdenken!

6.7.6 Verfolgungstabellen

Kombiniert Ausführungstabellen, symbolische Ausführung und Induktionsbeweise.

1. Identifiziere Eingabeklassen mit qualitativ verschiedenem Programmverlauf.
2. Beschreibe jede davon durch Bedingungen
   z.B. „Die Eingabe für `a` ist größer als 1“ anstatt
   „Die Eingabe für `a` lautet 17“.
3. Prüfe jede Eingabeklasse separat.
4. Arbeite dabei soweit möglich symbolisch, aber wie in einer Ablauftabelle.

Recht aufwendig, aber gute Abdeckung.
Nicht mehr Fälle als bei Test, aber findet auch Fehler*stellen*.

6.7.7 (Halb)formale Verifikation

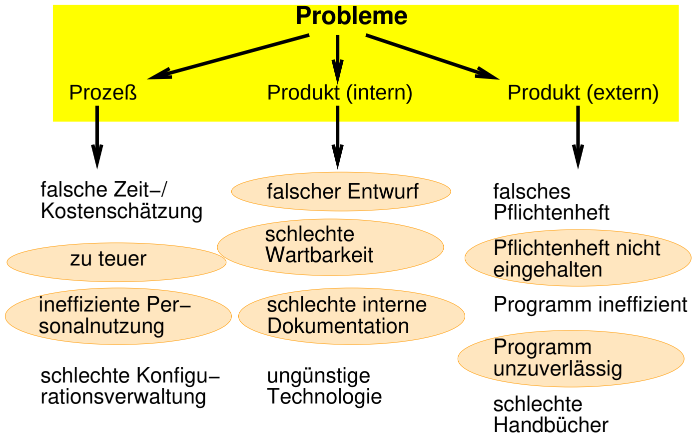
Prüfung detailliert formal spezifizierter Stücke.

- Leistungsfähig, aber schwierig und teuer.
- Kosten geringer, wenn Beweis gleich bei Konstruktion.
- Nur kritischste Stücke verifizieren
  z.B nur Schleifenbedingung und -fortschalten.
- Halbformale Anwendung möglich: Gleiche Grundmethoden und Schlußweisen, aber Auslassung von Details.
  z.B. Cleanroom-Methode.

6.8 Zusammenfassung

- Anwendung des CMM auf einzelnen SW-Ingenieur.
- Stufe 5 recht schnell erreichbar.
- PSP-Kurs: 15 Wochen, 10 Übungsaufgaben.
  Gute Ergebnisse. Stufe 4.
- Schwerpunkte: Planung, Qualitätsmanagement.
- Simple Maße gezielt und sinnvoll eingesetzt.
- Größenschätzungsmethode.
- Entwurfsprüfungsmethoden.

7 Die Cleanroom-Methode



7.1 Einordnung und Ziele

- von Harlan D. Mills, IBM, seit 1980.
- Name wegen Fehlervermeidung statt Fehlerbeseitigung.
- Ziel: hohe, quantifizierte Zuverlässigkeit zu geringen Kosten.
- Halbformale und formale Methoden basierend auf mathematischer Logik.
- Trennung von Qualitätserzeugung und Qualitätsprüfung.
- Für alle Sprachen, o-o, CASE.
- Neuentwicklung, aber auch Wartung und Reengineering.

7.2 Ansätze

- Kleine Teams:
  - parallele Entwicklung.
  - genaue Spezifikation.
  - hohe Motivation, enge Zusammenarbeit, Effizienz.
- Separation von Entwicklung und Test:
  - Entwicklungsteams.
  - Testteams.
  - disjunkte Aufgaben.

- Exakte Spezifikation:
  - Fehlervermeidung.
  - Eindeutige Fehlerbestimmung.
  - Verifikation.
  - Z, VDM, box method, Spezialgrammatiken.
- Schrittweise Verfeinerung:
  - Spezifikation (black box).
  - Zustandsbeschreibung (state box).
  - Prozeßbeschreibung (clear box).

- Begutachtung/Verifikation:
  - Entlang aller Verfeinerungen
  - State box und clear box.
  - Mathematisch gegründet.
  - Selten mathematischer Beweis.
  - Konstruktion oder Begutachtung des Beweises in Inspektion.
- Inkrementelle Entwicklung.
- Statistisches Testen:
  - Benutzungsmodellierung.
  - Testfälle: Stichprobenerzeugung.
  - Quantitative Zuverlässigkeitsaussage (Zertifikation)

7.3 Ergebnisse

- Produktivitätsverbesserungen: wenige Prozent bis Faktor 5.
- Testfehlerrückgang: Faktor 2 bis Faktor 100.
- z.B. 350 KLOC Betriebssystem:
  1 Fehler/KLOC, +70% Produktivität.
- Kontrolliertes Experiment (15 Teams, 6 Wochen, ca. 1300 Zeilen):
  - Mehr Funktionalität in gegebener Zeit, mehr Meilensteine eingehalten.
  - Programme weniger komplex, besser kommentiert, nicht länger.
  - Signifikant weniger Fehler.
  - 86% vermißten Testen, Qualität aber nicht schlechter.

7.4 Probleme und Hindernisse

Cleanroom schlecht geeignet, wenn

- formale Spezifikation schwer möglich,
- Bestimmung der Korrektheit von Ausgaben teuer.

Nötige Randbedingungen:

- Hoher Ausbildungsstand.
- Definierter Softwareprozeß (CMM Stufe 3).

7.5 Spezifikation und Entwurf mit Kastenstrukturen

- **Definiere die black box:**
  - Definiere Eingaben.
  - Definiere Ausgaben anhand Eingabehistorien.
- **Definiere die state box:**
  - Definiere Zustände zur Repräsentation der Eingabehistorien.
  - Wähle Zustandsdaten für Ebene aus.
  - Formuliere black box neu.
  - Verifiziere state box.

- **Definiere die clear box:**
  - Definiere Datenabstraktion für Zustandsdaten.
  - Formuliere state box neu.
  - Verifiziere clear box.

Weiter mit black boxes der Verfeinerungen.

Beispiel

•*black box 1:* `dreieckstyp` (a, b, c)
Vorbedingung: a, b, c positive reelle Zahlen
Nachbedingung: GLEICHSEITIG / GLEICHSCHENKLIG / SONSTIGES / KEINDREIECK $\iff$
Tripel (a, b, c) Seitenlängen eines gleichseitigen / nichtgleichseitigen gleichschenkligen / nichtgleichschenkligen Dreiecks / kann überhaupt nicht Tripel von Seitenlängen eines Dreiecks sein.

○*clear box 1:* `dreieckstyp` (a, b, c)
IF `alle seiten erfüllen dreiecksungleichung` (a, b, c)
THEN Liefere `echter dreieckstyp` (a, b, c)
ELSE Liefere KEINDREIECK

•*black box 2:* `alle seiten erfüllen dreiecksungleichung` (a, b, c)
Vorbedingung: a, b, c positive reelle Zahlen.
Nachbedingung: True, falls jede Seite kürzer als die Summe der beiden anderen; False andernfalls.

•*black box 3:* `echter dreieckstyp` (a, b, c)
Vorbedingung: (a, b, c) sind die Seitenlängen eines echten Dreiecks
Nachbedingung: ...

⋆*Verifikation clear box 1*:
(a, b, c) kann Dreieck bilden $\Longleftrightarrow$
die beiden kürzeren Seiten $x + y$ zusammen länger als längste, z.
$z < x + y$ (also: „Dreiecksungleichung gilt für die Seite z“) ist hinreichend für Dreieck. Außerdem ist notwendig:
Dreiecksungleichung gilt für jede Seite. $z < x + y$ gilt insbesondere, falls Dreiecksungleichung für jede Seite gilt.
Damit liefert `alle seiten erfüllen dreiecksungleichung` die Aussage, daß es sich wirklich um ein Dreieck handelt; Vorbedingung von `echter dreieckstyp` wird erfüllt, und clear box 1 ist korrekt.

∘*clear box 2:* `alle seiten erfüllen dreiecksungleichung` (a, b, c)
Liefere ($a < b + c$ AND $b < a + c$ AND $c < a + b$)

⋆*Verifikation clear box 2*:
3 verschiedene, korrekte Fälle geprüft, UND-verknüpft; ist offensichtlich korrekt.

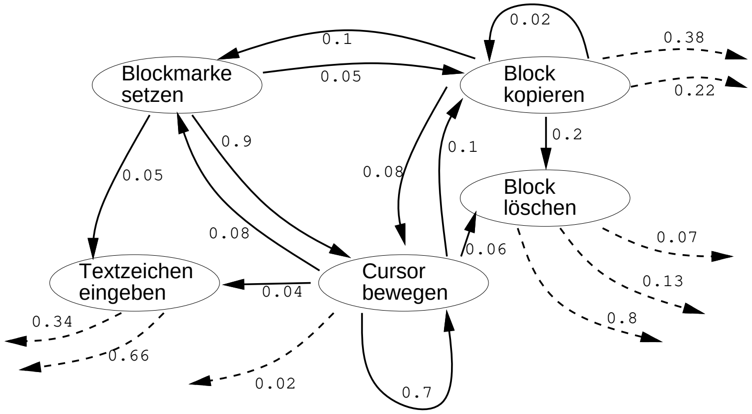
∘*clear box 3:* `echter dreieckstyp` (a, b, c)
IF $a = b = c$
THEN Liefere GLEICHSEITIG
ELSE IF $a = b$ OR $a = c$ OR $b = c$
THEN Liefere GLEICHSCHENKLIG
ELSE Liefere SONSTIGES

⋆*Verifikation clear box 3*:
SONSTIGES ist als NOT gleichschenklig korrekt geprüft laut black box 1.
Gleichseitig ist ein Spezialfall von gleichschenklig und muß deshalb zuerst geprüft werden, dies geschieht hier.
Deshalb ist clear box 3 korrekt.

7.6 Statistisches Testen und Zertifizierung

7.6.1 Benutzungsmodellierung

- Basis ist Beschreibung des Benutzungsprofils (aus Anforderungen)
- Mathematische Beschreibung mit Markov-Ketten (endlicher Zustandsraum, diskrete Ereignisse)



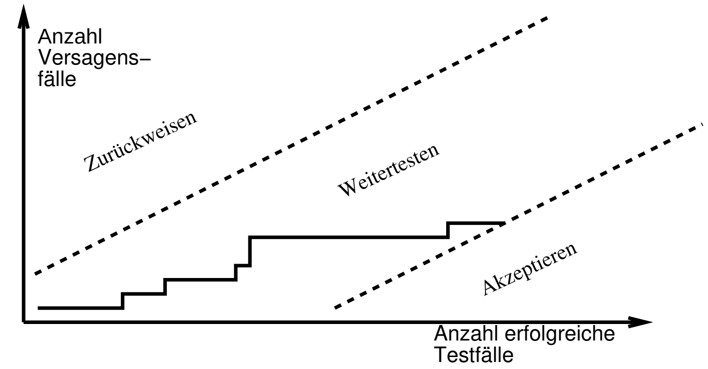
Ausschnitt aus Benutzungsmodell für einen Texteditor

7.6.2 Testen

- beliebige Zahl von Testfällen automatisch erzeugt
- Ausgaben???
- Evtl. nur Plausibilitätsprüfung
- Prüfe pro Testfall Korrektheit
- Registriere Intervalle zwischen Versagen

7.6.3 Zertifizierung

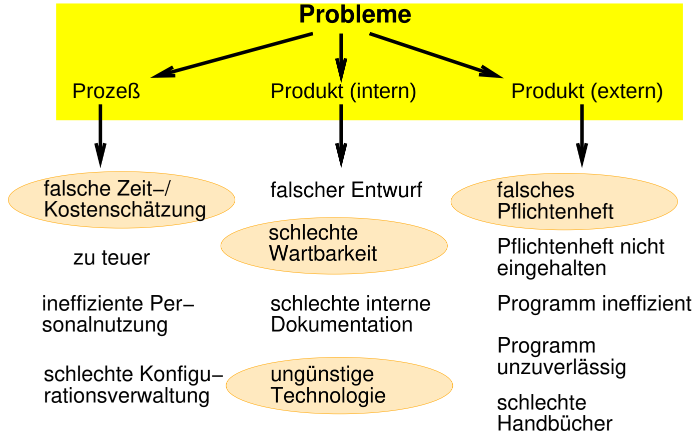
- Teste bis Zuverlässigkeitsaussage erhalten, z.B.
- „MTTF(Programm) $\geq m$ mit Konfidenz k“.
- Prüfung mit statistischen Methoden.
- Erweiterung: Fehlermodelle, Zuverlässigkeitsmodelle.
- Dann möglich, Aussagen von alten Programmversionen mit einzubeziehen.



7.7 Zusammenfassung

- Exakte Spezifikation
- Schrittweise Verfeinerung mit Kastenspezifikation
- Verifikation in Inspektionen
- Statistischer Test basierend auf Benutzungsmodellierung
- Zertifizierung
- Resultat: Sehr geringe Fehlerraten, hohe Produktivität

8 Anforderungsbestimmung (Requirements Engineering)



8.1 Einordnung und Ziele

"Was sollen wir eigentlich bauen?"

Auch bekannt als

- Requirements Elicitation
- Requirements Gathering
- Anforderungsdefinition
- Requirements Definition
- Anforderungsanalyse
- Requirements Engineering

Arten von Anforderungen:

1. Funktionale Anforderungen
2. Qualitätsanforderungen
3. Randbedingungen (HW-, SW-Umgebung, Betriebsbedingungen etc.)

Oft kritischste Phase:

- Fehler sehr teuer, da
  - evtl. systemweite Auswirkung und
  - oft erst spät bemerkt.
- Projekt eventuell totaler Fehlschlag, denn
  - SW-Ingenieure bemerken selbst schwerste Fehler gar nicht,
  - Anwender bemerken sie erst nach Systemeinführung.

Korrekte Anforderungsbestimmung ist extrem wichtig!

Wichtige Hinweise:

1. Sichtweise „Ermittle *Was*, aber nicht *Wie*" ist Idealisierung.
2. Funktionale und nichtfunktionale Anforderungen beeinflussen einander.
3. Anforderungen sind unvollständig.
4. Anforderungen ändern sich.

Viele Aussagen gelten deshalb nur mit Einschränkungen.

Fragestellungen:

- **Wie kommuniziert man mit den Anwendern?** Erfahrungshintergrund, Terminologie.
- **Wie entdeckt man Anforderungen?** Anwender kennen sie selber nicht!
- **Wie setzt man vage Anforderungen in konkrete um?** klare Richtlinien f. Systembau nötig, Prüfbarkeit!
- **Wie versteht man Prioritäten und Zufriedenheitskriterien des Anwenders?** Wichtig für Projektsteuerung u. Verständnis vager Anforderungen.

- **Wie stellt man Eindeutigkeit sicher?** Konflikte entdecken und entfernen.
- **Wie geht man mit Unvollständigkeit um?** Terminierung der Anforderungsbestimmung mit/ohne bewußte Lücken.
- **Wie löst man Interessenkonflikte?** Zusätzliche Erschwernis bei allen anderen Teilproblemen. z.B Datenschutzfragen, Arbeitsverteilung, Arten von Arbeiten, Kompetenzen.

Keine allgemeinen Lösungen bekannt.

8.2 Ansätze

8.2.1 Arbeitsphasen

Die Anforderungsbestimmung umfaßt drei Schritte:

1. **Ermittlung der Anforderungen**. Hauptproblem!
2. Beschreibung der Anforderungen.
   Notationen, Konventionen.
3. Analyse der Anforderungen.
   Prüfe Plausibilität, Qualität, Vollständigkeit.

(2) und (3) gute Hilfen für (1)!

8.2.2 Anwendungsbereiche

Generell: Bestimmung um so schwieriger, je tiefer Eingriff in Leben der Anwender.

- **Betriebliche Informationssysteme**.
  Komplexer Kontext, „fremde“ Anwender, Interessenkonflikte.
- **Benutzerschnittstellen**.
  Kontexte sehr divers, evtl. unbekannt, Psychologie.
  Eigene Disziplin.
- **Software für technische Anwendungen**.
  Vertrautere Anwender, evtl. konkretere Problemstellung.

- **Steuerungssysteme**.
  Technische Randbedingungen enger.
- **Eingebettete Systeme**.
  Noch enger, jedoch evtl. Benutzerschnittstelle.

Herausforderung: Finde völlig neue Anwendungen.

Beispiel: regelbares Ventil durch binäres Ventil + Software.

Anf.bestimmung einfacher $\Leftrightarrow$ Korrektheit kritischer.

8.3 Ergebnisse

- Forschung schwierig: große Unterschiede zwischen Anwendern, direkter Methodenvergleich nicht möglich.
- Deshalb meist nur Fallstudien.
- Studie über Methoden und Hilfsmittel:
  - Projekterfolg höher, wenn mehr verschiedene Methoden.
  - Kosten weiterer Methoden übersteigen irgendwann Nutzen.
  - Direkte Kommunikation erfolgreicher als indirekte.

8.4 Probleme und Hindernisse

Komplexer Prozeß, von Menschen durchgeführt, von menschlicher Umwelt abhängig. Informatik-Abstraktionen erst hinterher anwendbar.

Probleme beim **Verstehen des Anwendungsbereichs**:

- Anwender erkennen ihre Probleme nicht.
- Anwender können DV-Vorschläge nicht kritisieren.
- Neuartige DV-Lösungen übersteigen Vorstellungsvermögen.
- Entwickler verstehen viele Anwenderabstraktionen nicht.
- Entwickler haben oft nur technisches Interesse, verstehen nichttechnische Anforderungen nicht.

Kommunikationsprobleme:

- Verschiedene Begriffswelten und Erfahrungshintergründe.
- Verschiedene Beschreibungsmethoden und Notationen.
- Schwierigkeiten, sich auf Abstraktionsniveau zu einigen. Verstehen nicht, welches der andere gerade benutzt oder benutzen will.

Soziale Hindernisse:

- Verschiedene Gruppen von Anwendern haben unterschiedliche Interessen. Gefährlich, wenn Konflikte verwischt statt gelöst werden.
- Anwender sehen oft Notwendigkeit der Anforderungsbestimmung nicht ein und haben kein Interesse an Details.
- Entwickler haben oft keine Leistungsanreize auf Kundenzufriedenheit hin.
- Entwickler haben oft Überlegenheitsgefühl.

Technische Probleme, Projektmanagement:

- Einfluß von Marktkräften.
  Zeit bis Markteintritt, Features, Produktprofil etc.
- Entwicklungsprozeß bei sich ändernden Anforderungen.
  Alternativen vorzeichnen.
- Einbindung umfangreicher Fertigkomponenten.
  Rückwirkung auf Anforderungen

8.5 Analyse der Anforderungen

Qualitäts- und Plausibilitätsprüfung.

- Vollständigkeit. Komplett oder alle Lücken bekannt?
- Konsistenz.
- Eindeutigkeit. Alle Anforderungen von allen verstanden?
- Nachprüfbarkeit.
- Rückverfolgbarkeit. Gründe für Anforderungen bekannt?
- Modifizierbarkeit/Orthogonalität.
- (kostengerechte) Realisierbarkeit.
- Kostenschätzbarkeit.

Methoden:

- Formale Vollständigkeits- und Konsistenzprüfungen.
- Eindeutigkeit durch mathematische Schreibweise.
  Vorsicht vor Illusionen.
- Werkzeugunterstützung für Rückverfolgung, Modifikationen.
- Ansonsten: Wissen, Erfahrung, Gefühl, Überschlagsrechnungen.

8.6 Grundtechniken

8.6.1 Befragungstechniken

- **Interviews mit Benutzern**.
  Direkte, präzise, unverfälschte Auskünfte.
- **Interviews mit Kontaktpersonen**.
  Effizienter, fehleranfälliger.
- **Fragebogen**.
  Effizient, viele Meinungen, nicht interaktiv deshalb Vorverständnis nötig.
- **Gespräche im Marketing und auf Messen**.
  Aktives Interesse, produktorientiert, Detaillierung und Nachfragen schwierig.

8.6.2 Sammeltechniken

Eher anwendergetrieben

- **Hotline**.
  Schwachstellen an Benutzerschnittstelle und Dokumentation.
- **Wünsche von einzelnen Benutzern**.
  Fehlende Funktionalität, aus konkreten Problemen.
- **Wünsche von organisierten Benutzergruppen**.
  Wünsche gefiltert und vorverdaut.

8.6.3 Gruppentechniken

Balancierte Aktivität, partnerschaftlich.

- **klassische Gruppensitzungen**:
  Team, Analytiker verantwortlich.
- **Partizipatorischer Entwurf**:
  Demokratischer Prozeß der Anwender, Analytiker als Helfer, Anwender verantwortlich.

8.6.4 Beobachtungstechniken

Anforderungsbestimmung implizit.

- **Benutzbarkeitslabors**.
  existierende Software und Prototypen, vor allem Benutzerschnittstellen.
- **Protokollauswertung**.
  Automatisch gesammelte Daten. Ausbügeln fehlererzeugender Schwächen.
- **Feldbeobachtung**.
  Vielseitig, genau. Hoher Aufwand, Anwender evtl. befangen.
- **„in die Lehre gehen"**.
  Anwender als Experte, Befangenheit geringer, Nachfragen.

8.6.5 Hilfsmittel

- **Reservierte Räume**.
  Übersicht über Zwischenergebnisse, diskussionsfördernd.
  Zusammengehörigkeitsgefühl: Barrieren abbauen.
- **Prototypen**.
  Konzepte prüfen, früher Entwurfseinfluß.
  Volle Prototypen, Scheinprototypen, Benutzerschnittstellenprototypen.
- **Papierprototypen**.
  Unterstützt Vorstellungskraft, Vermeidet Mißverständnisse.
- **Elektronisches Pflichtenheft**.
  Für eifrige Computernutzer, mehr Personen einbindbar, wichtig bei gleichzeitigem Entwurf.

- **Elektronische Diskussionsgruppen**.
  Effizient da asynchron, kein Verlust von Beiträgen da schriftlich. Jedoch Steuerungsbedarf.
- **Video** (Anwender mit nach Hause bringen)
  Einbeziehung zusätzlicher Analytiker, Klären von späteren oder viel späteren Fragen. Verwaltungskosten beachten.
- **Video** (im Benutzbarkeitslabor)
  Sammle Bibliothek „kritischer Anwendungsfälle".

8.7 Umgang mit Risiken und Konflikten

8.7.1 Anforderungsbestimmung als Konfliktlösungsprozeß

- Übliche Sicht von Anforderungsbestimmung:
  Hauptproblem ist Anforderungen finden und genau beschreiben.
- Alternative Sicht:
  Hauptprobleme sind Unsicherheiten, Risiken und **Konflikte**.
  Auflösung durch Verhandlungen.
  Basiert auf Risikomodell (Spiralmodell); geht über Anforderungsanalyse hinaus; geeignet für sich ändernde Anforderungen.
- Deckt übliche Sicht weitgehend mit ab.
  Zusätzlicher Vorteil: Unklarheiten klarer.

8.7.2 Theorie W

Ausgangspunkt: Managementtheorien.

- **Theorie X**: („wissenschaftliches Management“)
  Effiziente Abläufe und Vorhersagbarkeit herstellen.
  Problem: unterdrückt Kreativität und Initiative.
- **Theorie Y**:
  Fördere Kreativität und Initiative.
  Problem: Erzeugt Konflikte, behandelt sie aber nicht.
- **Theorie Z**:
  Investiere vorab in gemeinsame Werte. Konsensprinzip.
  Problem: schlecht anwendbar, wenn mehrere Parteien beteiligt.

Theorie W:

Für Softwareentwicklung: Parteien mit verschiedenen Interessen (Entwickler, Warter, Kunde, Benutzer).

Interessen sind Wurzel von Konflikten. Müssen erkannt und behandelt werden. Ideal: Entscheidungen, mit denen alle zufrieden sind.

„Mache jeden zum Gewinner (make everyone a winner)“

Drei Arten von Situationen:

- Gewinn-Verlust (win-lose) Situation:
  Nullsummenspiel. Häufig, aber nicht immer.
- Verlust-Verlust (lose-lose) Situation:
  z.B. bei überhöhten Erwartungen, unrealistischen Plänen.
  Passiert bei schlechtem Risikomanagement.
- **Gewinn-Gewinn (win-win) Situation**:
  Alle Parteien profitieren.
  z.B. Beteiligung des Softwareerzeugers am Gewinn des Kunden.

Versuche, win-win Situationen zu erzeugen und zu erhalten.

Prinzipien:

- **Verstehe, wie die Beteiligten gewinnen wollen.**
  Identifiziere die entscheidenden Beteiligten (z.B. Unterauftragsnehmer)
  Versetze Dich in die Gewinnsituationen der anderen (z.B. für richtige Beurteilung der Anwenderwünsche).

- **Erzeuge angemessene Erwartungen.**
  Hart-Weich ist leichter als Weich-Hart.

- **Passe Aufgabenzuweisung an Gewinnbedingungen an.**
  Entdecke vorhandene und erzeuge neue win-win Situationen:

  - Optionen entlang von Gewinnmöglichkeiten ausrichten, z.B. die Verantwortung für die Qualitätssicherung auf die Programmwarter verlagern.
  - Aufgaben an Belohnungen koppeln.
  - Aufgabenbereich erweitern, z.B. Qualitätssicherung als vorbeugende und organisationsweite Maßnahme.
  - Besondere Unterstützung zusichern.
  - Neue Optionen auftun, z.B. Inkremente definieren bei nichterfüllbarem Plan.

Hilfsprinzipien von Theorie W:

1. **Plane gründlich und folge Deinem Plan.**
   Wie werden win-win Situationen hergestellt und erhalten:
   Warum? (Ziele), *Was?* (Produkte), *Wann?* (Meilensteine), *Wer und Wo?* (Verantwortlichkeiten), *Wie?* (Vorgehen), *Wieviel?* (Ressourcen).
   Korrekturen während Planausführung sind kritisch!
2. **Identifiziere und kontrolliere Deine Risiken.**
   1. Risikobestimmung (Risikoentdeckung, -analyse, und -priorisierung).
   2. Risikobehandlung (Planung, Durchführung und Überwachung).

   Oft schlechte Risikobehandlung oder -priorisierung.

8.7.3 Das WinWin-System

CSCW-Werkzeug, drei Hauptkonzepte:

- **Gewinnbedingung (win-condition, WC)**.
  Gewünschtes Ziel eines oder mehrerer Beteiligter.
  z.B. „Zuverlässigkeit“ oder „Projektkosten unter DM 1.2 Mio.“

- **Konflikt-/Risiko-/Unsicherheitsbeschreibung (conflict/risk/uncertainty specification, CRU)**.
  Konflikt zwischen WCs oder mit WC verbundene Risiken oder Unsicherheiten.
  z.B. Konflikt zwischen Funktionalität und Budget, Risiko bei Fähigkeit, eine Anforderung zu erfüllen oder Unsicherheit bezüglich Bedeutung einer Anforderung.

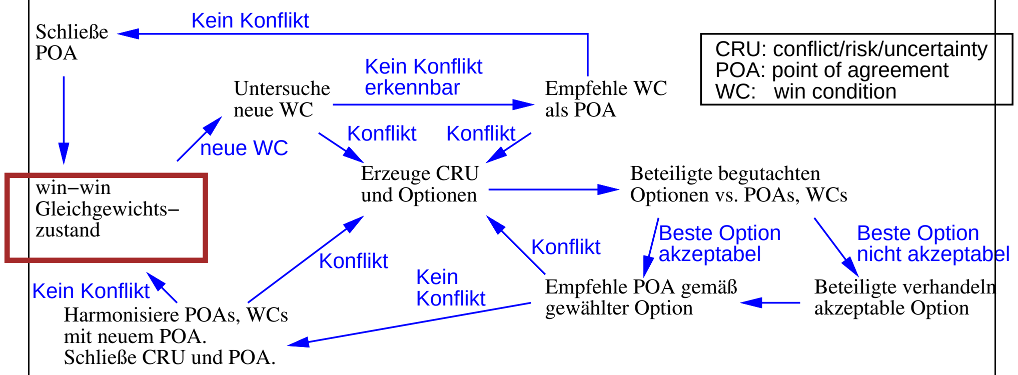
- **Übereinkunftspunkt (point of agreement, POA).** Menge vereinbarter Bedingungen, die die WCs einhalten. Definiert die Ziele der Systemkonstruktion. z.B. auf bestimmte Funktionalität verzichten, um Budget einzuhalten.

Vorgehensmodell:

- **Gleichgewichtszustand (Equilibrium)** zwischen WCs:
  1. keine Konflikte zwischen WCs und POAs
  2. alle POAs miteinander konsistent
  3. keine unaufgelösten CRUs oder offenen POAs

  Dies ist ganz zu Beginn erfüllt.
- neue WC kann Gleichgewicht stören (durch Konflikt, Risiko oder Unsicherheit).

Modell unterstützt ständige Änderungen der Anforderungen!



Zustandsübergangsdiagram des WinWin-Systems

Hilfsmittel:

- **Wahlmöglichkeitsübersicht (option summary)**
  Vom Analytiker aufgestellt, Teil von CRU. Benennt Vor- und Nachteile. Schnelle Konzentration auf relevante Optionen.
- **Bereichstaxonomie**
  Projektspezifisch, liefert einheitliche Terminologie, dadurch Konflikte leichter erkennbar.
  Teile: Produktzerlegung, Produktattribute, Prozeßattribute.

8.7.4 Konfliktidentifikation für Qualitätsanforderungen

Restaufgaben bei WinWin:

1. WCs identifizieren.
2. Konflikte zwischen ihnen erkennen.
3. Handlungsoptionen finden, die in win-win Situationen führen.

wissensbasiertes Werkzeug QARCC zur Erkennung von Qualitätsanforderungskonflikten.

Wissensbasisteile:

1. Liste der typischen **Qualitätsanforderungen**:
   Sicherheit/Betriebsgarantien (assurance), Interoperabilität, Benutzbarkeit/Benutzungsfreundlichkeit, Geschwindigkeit, Weiterentwickelbarkeit/Portabilität, Kosten und Zeitplan, Wiederverwendbarkeit.
2. **Verfeinerung** dieser Anforderungen:
   Schlüsselt in konkretere Anforderungen auf. z.B Sicherheit/Betriebsgarantien: Zuverlässigkeit, Genauigkeit, Korrektheit, Verfügbarkeit, Robustheit, Gefährdungsfreiheit, Datensicherheit, Datenschutz.

3. Liste von **Strategien** zu jedem Attribut.
   Produktstrategien (Architekturstrategien) und Prozeßstrategien.
   z.B. **Produktstrategien** für gute Benutzbarkeit: Verteilbarkeit (groupware), fehlervermeidende, konsistente, flexible, benutzeranpaßbare Benutzerschnittstellen, on-line Hilfe, äußere Modularität, Navigationshilfen, Benutzerprogrammierschnittstellen, Undo-Funktion.
   Zugehörige **Prozeßstrategien**: Bau von Prototypen, Benutzungsprotokollierung und -modellierung, Benutzerbeteiligung beim Entwurf, Benutzung von Benutzerschnittstellenwerkzeugen.

Substruktur der Produktstrategien:

- Definition: Beschreibung der Strategie.
- Vorbedingung: Anforderungen für die Anwendbarkeit der Strategie.
- Nachbedingung: Resultate, die erreicht werden müssen.
- Effekte: positive/negative Wirkungen auf Qualitätsattribute.
- Optionen: Methoden zur Auflösung von Konflikten.

Funktionsweise von QARCC zusammen mit WinWin:

1. QARCC extrahiert Taxonomieelemente, wenn Benutzer WC eingibt.
2. Für ggf. Qualitätsattribute bestimme Produktstrategien.
3. Für Strategie bestimme negativ beeinflußte Qualitätsattribute.
4. Suche sonstige win-conditions mit diesen Attributen.
5. Gib ggf. Nachricht an beide Benutzer: WCs, Strategie.
6. Erläuterung der Umstände durch Vorbedingung und Nachbedingung. Lösungsvorschläge aus Optionen-Feld bilden Grundlage für Verhandlungen.

Fallstudie: QARCC fand sieben Konflikte, Menschen nur zwei. QARCC fand ferner drei nicht signifikante Konflikte.

Guter Ansatz, geschickt hybrid automatisch/manuell.

Skalierbar?

8.8 Integration verschiedener Sichten

Ziel:

- Jeder Anwender benutzt „Lieblingssicht“.
- Werkzeug bildet auf andere Sichten ab.
- Gemeinsame Sicht als Basis.
- Maschinelle Analyse auf Inkonsistenzen.

Ansatz: Metamodellierungssysteme,

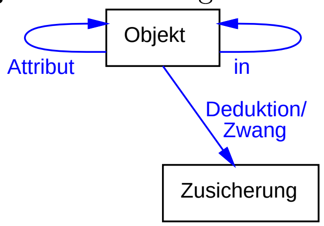
8.8.1 Metamodellierung

- Modell: abstrakte Beschreibung eines Zustandes.
- Modellierungssprache ist Metamodell.
  praktisch aber unflexibel.
- Metamodellierungssprache ist Metametamodell (M2-Modell).
  Kann Zusammenhänge zwischen Modellierungssprachen beschreiben.
- Grundlage für Sichtenintegration:
  Jede Sicht eine Sprache, M2-Modell gemeinsame Grundlage.

8.8.2 Beispiel: Telos

2 Entitätstypen, 3 Beziehungstypen:

- **Objekte**, über Namen identifiziert.
- **Zusicherungen**, also Formeln.
- **Attribute**.
- **Klassifikation** mittels „in“–Beziehung.
- **Regelbeziehung**. Deduktionsregeln und Zwänge.



Beispiele für Telos-Deklarationen:

```
Department in Class with
  attribute head: Manager
end
Employee in Class with
  attribute dept: Department;
            boss: Manager
end
Manager in Class isA Employee end
Employee with
  rule BossRule:$
      forall e/Employee m/Manager
        (exists d/Department (e dept d) and (d head m)) ==>
        (e boss m) $
```


8.8.3 Vorgehen zur Sichtenintegration

1. Definiere M2-Modell.
2. Definiere gewünschte Sichten.
3. Ermittle Ist-Zustand und Anforderungen in Interviews.
   Verwende jeweils günstigste Sicht.
4. Definiere Abbildungen zwischen Sichten.

Bemerkungen zum M2-Modell:

- Aufstellung aufwendig.
- Entscheidet über Flexibilität.
- Gutes M2-Modell kann wiederverwendet werden.
- Gutes M2-Modell muß einfach sein.
- Dann wird es wichtiges Kommunikationshilfsmittel.

8.8.4 Anforderungsanalyse durch Sichtenintegration

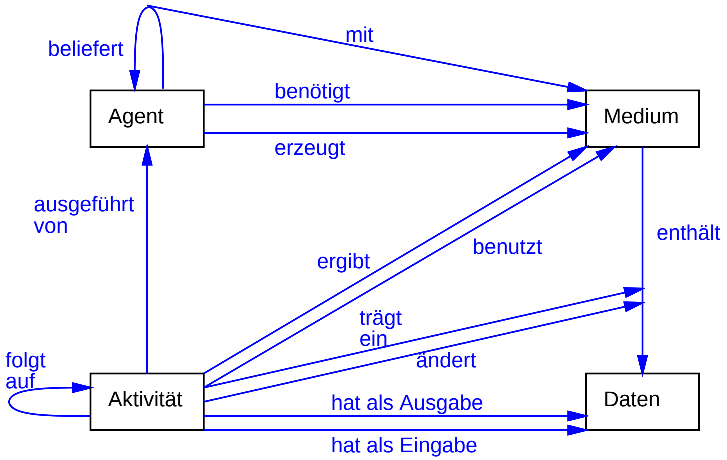
Definiere **Abfrageklassen** für jede Art erwarteter Inkonsistenzen.

Untersuche gefundene Instanzen.

8.8.5 Beispiel: PFR mit ConceptBase

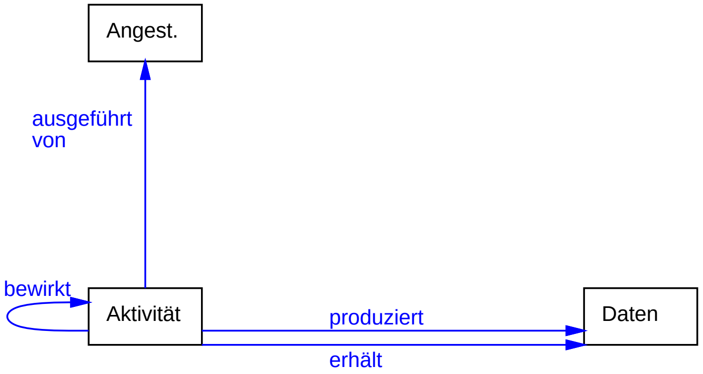
- Metamodellierungssystem ConceptBase: deduktives Datenbanksystem mit Telos als Kern.
- Beispiel: Analyse von Geschäftsprozessen.

- Methode PFR (analysis of presence and future requirements)
  - Sitzung zur Entwicklung von gemeinsamem Ausgangspunkt.
  - Untersuche zentrale Elemente in Interviews, Prozeßstudien, Dokumentstudien. Benutze diverse „Quellsichten“.
  - Kläre Widersprüche der Zusammenschau. Lokale Zustimmung.
  - Sitzung für Bekanntmachung und globale Zustimmung. Benutzt Zielsicht(en).

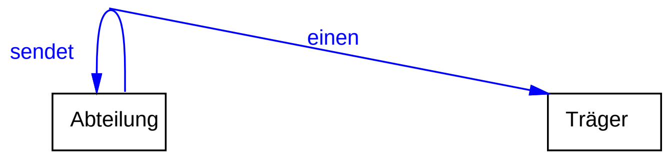


Beispiel für ein M2-Modell

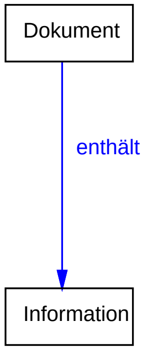
Einfach, aber schwer aufzustellen. Wiederverwendbar.



Modell der Aktivitätsfolgensicht



Modell der Informationsaustauschsicht



Modell der Dokumentenstruktursicht

Finde Inkonsistenzen gezielt durch Abfrageklassen, z.B.: Medien, die ein Agent erhält, obwohl er keine Daten davon nutzt.

```
QueryClass UnbenutztesMedium isA Medium with
  attribute
    nichtbenutzer : Agent
  constraint
    c:$ exists belieferung
          (belieferung in Agent!beliefert) and
          (belieferung an nichtbenutzer) and
          (belieferung mit this) and
          not exists aktion, info
            (aktion ausgefuehrt_von nichtbenutzer) and
            (this enthaelt info) and
            ((aktion hat_als_eingabe info) or
             (aktion hat_als_ausgabe info)) $
end
```


Liefert Paare (Medium `UnbenutztesMedium`, Agent `nichtbenutzer`). Mögliche Bedeutungen:

- Fehler: Eingabe- oder Ausgabebeziehung nicht erfaßt.
- Sichtenabweichung: Beteiligte sehen Prozeß unterschiedlich.
- Vermeidbar: Prozeß kann optimiert werden.
- Korrekt: aber dennoch sinnvoll (z.B. Träger sammeln).

Fehler, Sichtenabweichung: Probleme in Anforderungsbestimmung.
Vermeidbar: Optimierungsmöglichkeit (als Abfallprodukt).
Korrekt: unerwünschte Ausgabe, Modellierung zu schwach.

Suche nach Verbesserungsmöglichkeiten, z.B.: Agenten, die Dokument mit lauter bekannten Daten erhalten.

```
QueryClass MehrfachVersorgter isA Agent with
  attribute
    redundant : Medium
  constraint
    c:$ exists belieferung
          (belieferung in Agent!beliefert) and
          (belieferung an this) and
          (belieferung mit redundant) and
          forall info
            (redundant enthaelt info) ==>
              exists zweitmedium
                (zweitmedium <> redundant) and
                (zweitmedium enthaelt info) $
end
```


Liefert Paare (Agent `Mehrfachversorgter`, Medium `redundant`).
Mögliche Bedeutungen:

- Unnötig: kann optimiert werden.
- Auswahl: Agent soll Medium wählen. Sinnvoll?
- Prüfung: auf Gleichheit der Daten. Nötig? Automatisierbar?
- Dopplung: Agent soll wirklich dieselbe Tätigkeit mehrmals durchführen.

Unnötig: Verbesserungsmöglichkeit.
Auswahl, Prüfung: u.U. Verbesserungsmöglichkeiten.
Dopplung: unerwünschte Ausgaben, Modellierung zu schwach.

ca. 80 Anfrageklassen, davon ca. 70 wiederverwendbar.

Insgesamt:

- Metamodellierung gutes Werkzeug.
- Hohe Arbeitsersparnis, insbesondere bei großen Datenmengen.
- Zuverlässigkeitsgewinn der Prüfung.
- Leistungsengpässe möglich.

8.9 Zusammenfassung

- Anforderungsbestimmung, -beschreibung, -analyse.
- Sehr kritische Phase; je stärker menschliche Beteiligung, desto mehr.
- Probleme: Verstehen des Bereichs, Kommunikation, soziale Hindernisse, technische Randbedingungen.
- Befragungs-, Sammel-, Gruppen-, Beobachtungstechniken.
- Diverse Hilfsmittel.
- Anforderungsbestimmung als Konfliktlösung.
- Integration von Sichten.