Prof. Dr. Holger Schlingloff

Institut für Informatik der Humboldt Universität

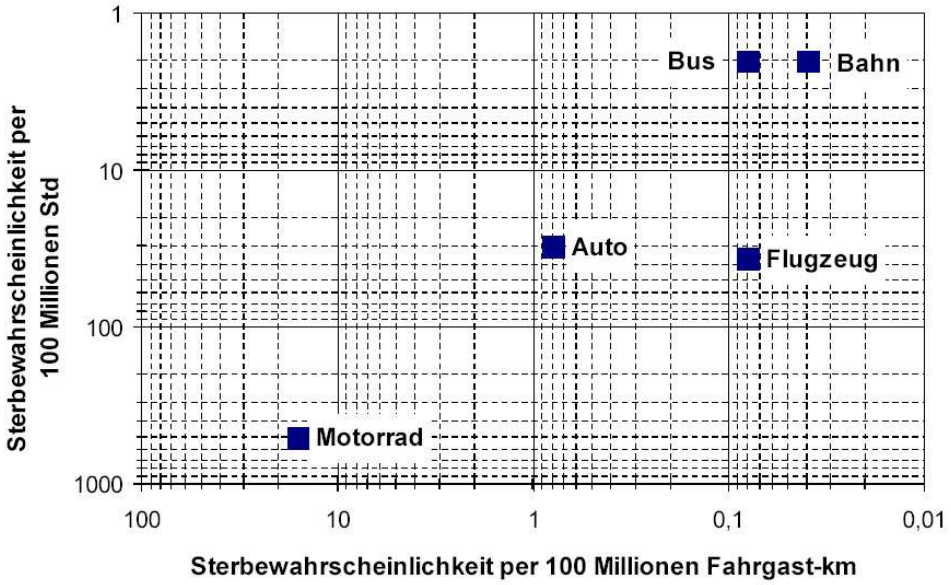
und

Fraunhofer Institut für Rechnerarchitektur und Softwaretechnik

Fraunhofer Institut
Rechnerarchitektur
und Softwaretechnik

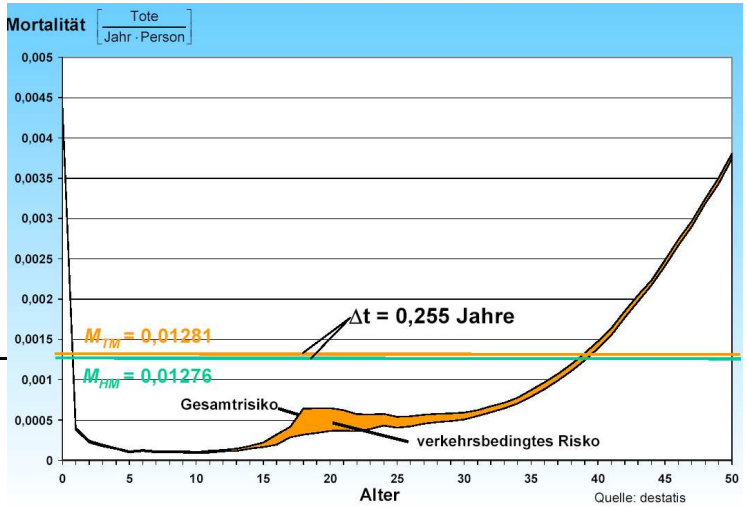
E. Schnieder: 4. Bieleschweig Workshop, 14.-15.9.04: NEUE UND HERKÖMMLICHE MAßE ZUR QUANTIFIZIERUNG DES RISIKOS IM EISENBAHNVERKEHR

	U.S. Automobiles	U.S. Commercial Aircraft
Deployed Units	~100,000,000	~10,000
Operating hours/year	~30,000 Million	~55 Million
Cost per vehicle	~$20,000	~$65 Million
Mortalities/year	42,000	~350
Accidents/year	21 Million	170
Mortalities / Million Hours	0.71	6.4
Operator Training	Low	High
Redundancy Levels	Brakes only	All flight-critical systems



Verkehrsmittel	Sterbefälle pro 100 Millionen Stunden Reisezeit der Fahrgäste	Sterbefälle pro 100 Millionen Kilometer Reisewegs der Fahrgäste
Motorrad	500	16
Auto	30	0,8
Bus	2	0,08
Flugzeug	36,5	0,08
Bahn	2	0,04

Quelle: SNCF

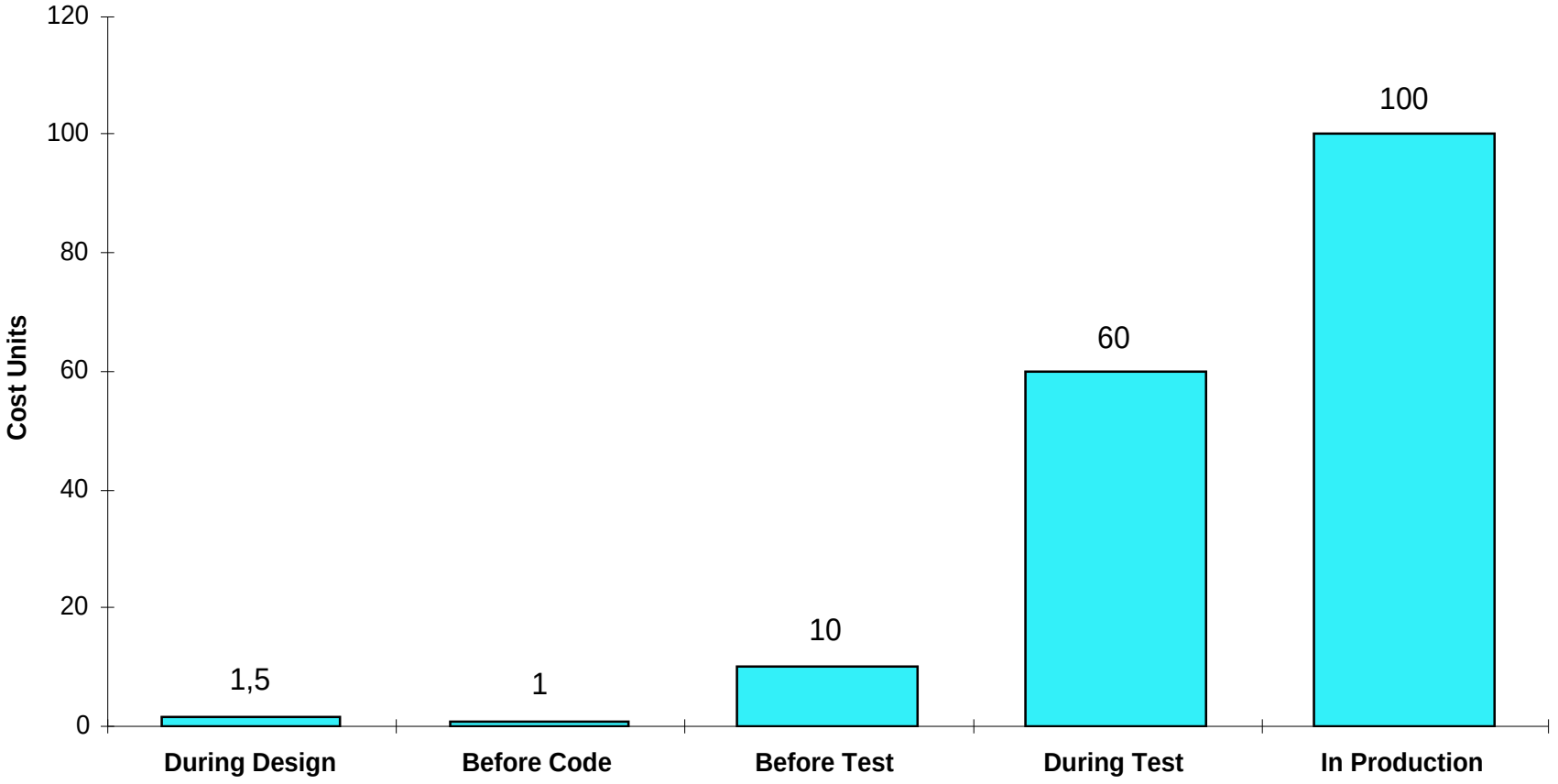


Quelle: http://www.ifev.bau.tu-bs.de/Workshops_Tagungen/Bieleschweig/bieleschweig4/pdf/Bieleschweig4_Folien_Schnieder.pdf

Fraunhofer FIRST Institut Rechnerarchitektur und Softwaretechnik

- Produktzyklus
  - Alle Aktivitäten, die mit der Entstehung, Verwendung und Entsorgung des Produktes zu tun haben
- Produktentwicklungszyklus
  - Alle Aktivitäten, die für die Erstellung des Produktes (der Software) nötig sind

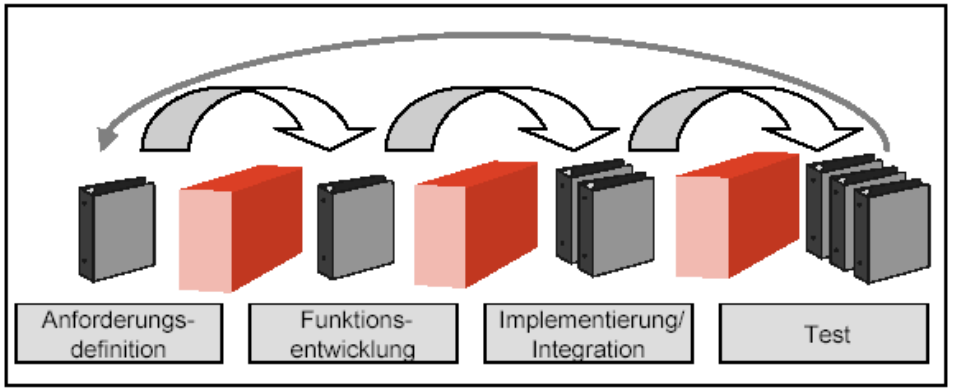
- Phasenmodelle
  - verschiedene gebräuchlich, kein „einzig wahres“ (Wasserfall, V-Modell, Spiralmodell, OO-Modell, XP, …)
  - Idee: Vorgehensweisen standardisieren („Checklisten“)
  - projektspezifische Anpassungen („Tayloring“)
  - integrierte QS

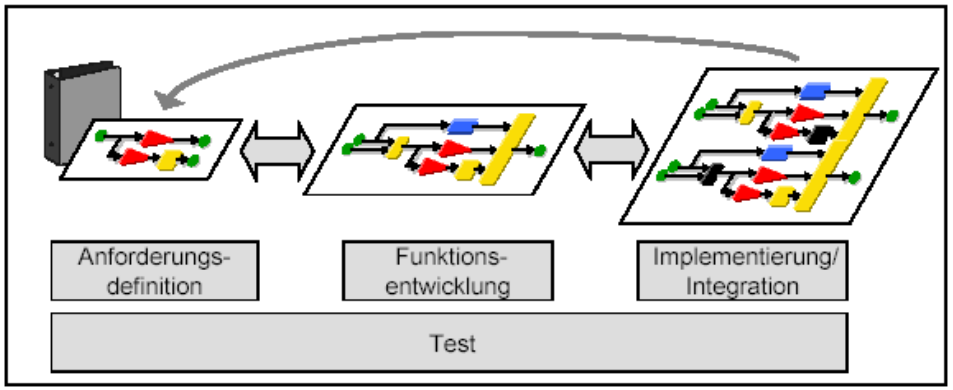


Source: Tom Gilb, Software Engineering Management, Daten der Standard Chartered Bank

- Anforderungsphase
- Entwicklungsphasen
- Inbetriebnahmephase

- Integration!
- durchgängige Modellierung!





QS: Anforderungsreview

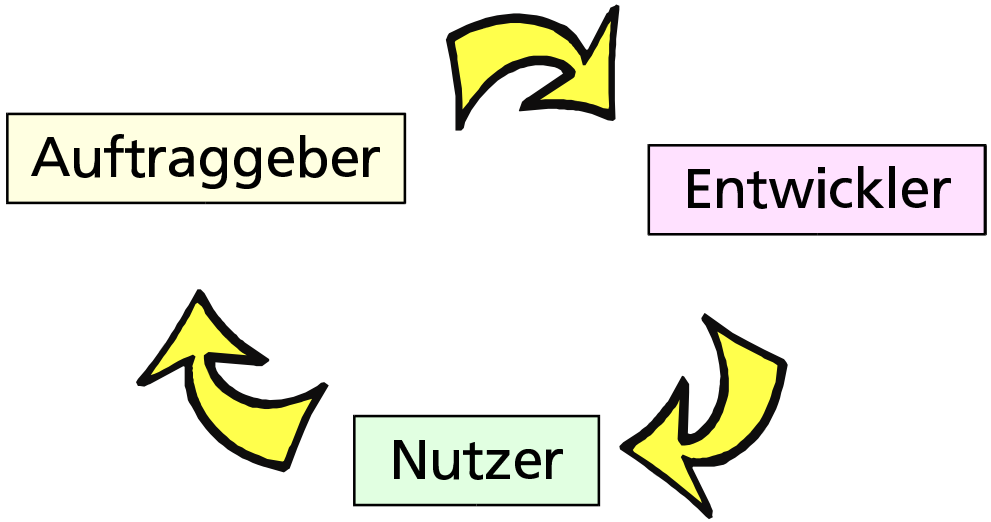
Objekt: Anforderungsdefinition

Methode: Testfähige Formulierung und Formalisierung der Anforderungen

- Besonders wichtig in den frühen Planungsphasen eines Projekts
- Entscheidend für Benutzerakzeptanz und Gesamterfolg

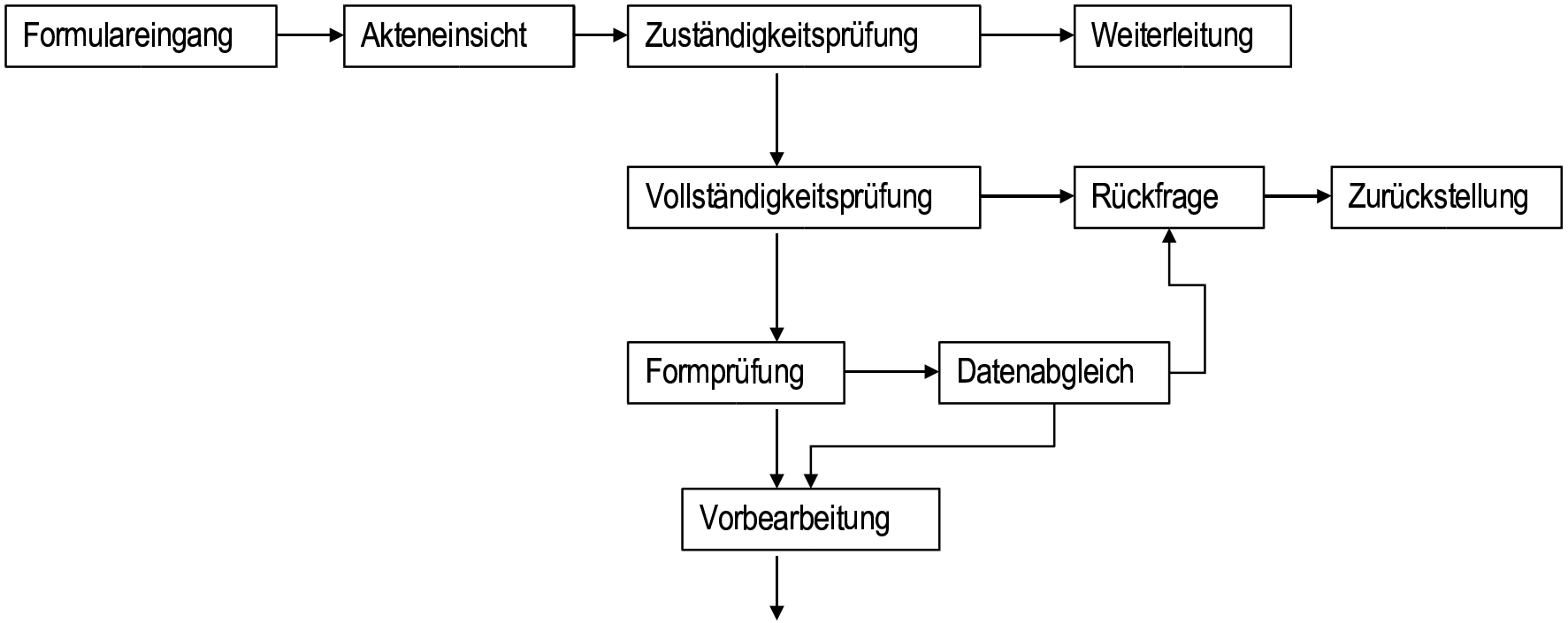
- Vorgehen: Progress reviews, Prototyping, Use cases

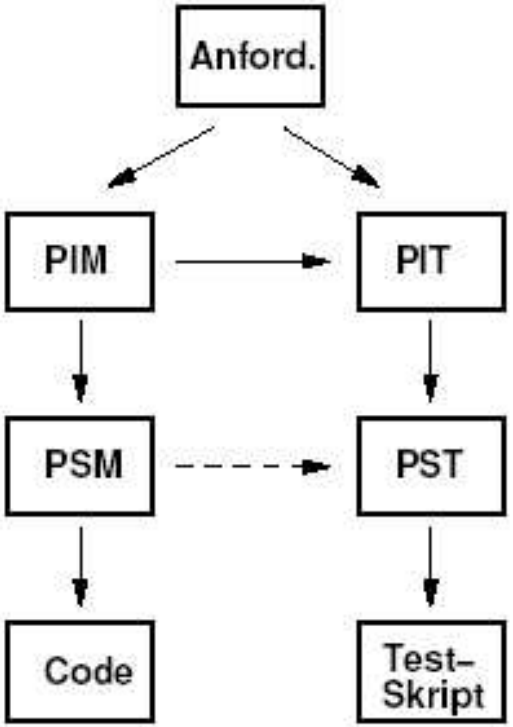
  Benutzerbeteiligung!

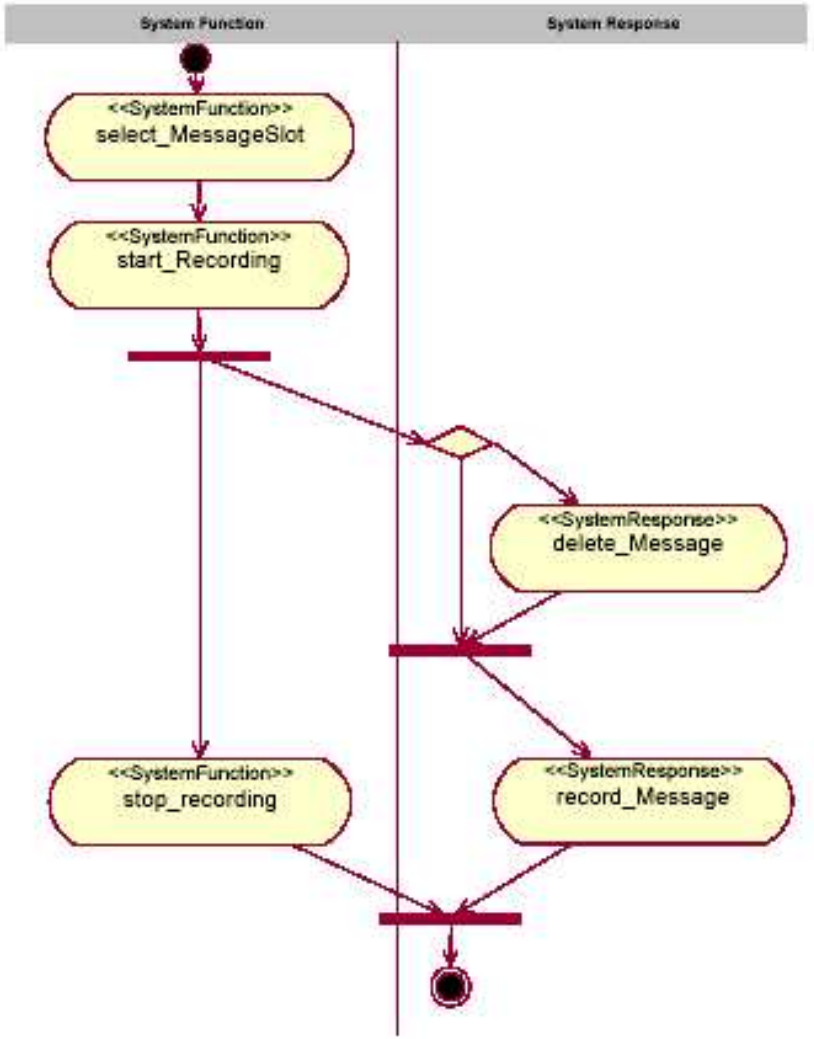


- z.B. als Transitionssystem oder Interaktionsdiagramm

Formales Benutzungsmodell







Fraunhofer FIRST Institut
Rechnerarchitektur
und Softwaretechnik

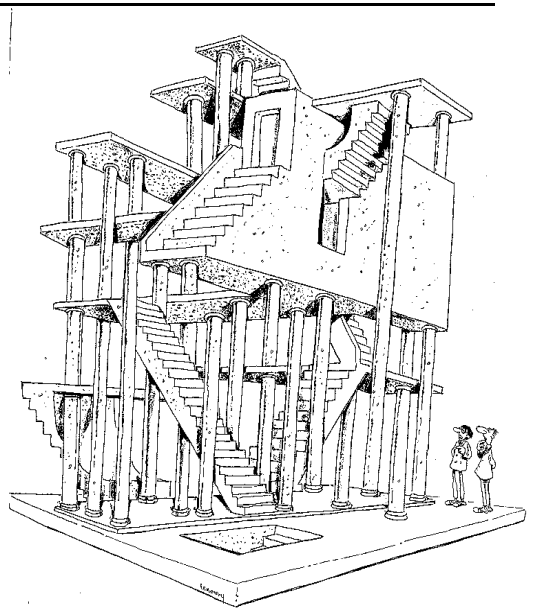
- Funktionsvollständigkeit
  - Ist der Aufgabenbereich klar abgegrenzt?
- Beschreibungsvollständigkeit
  - Wurden alle relevanten Werte definiert?
- innere Vollständigkeit
  - Sind alle Querbezüge vorhanden?
- äußere Vollständigkeit
  - Liegen alle Zusatzdokumente vor?
- Versäumnisfreiheit
  - Wurde nichts wichtiges ausgelassen?

- terminologische Konsistenz
  - Sind alle Begriffe eindeutig?
- innere Konsistenz
  - Existieren widersprüchliche Anforderungen?
- externe Konsistenz
  - Sind alle externe Objekte mit den Anforderungen vereinbar?
- zirkuläre Konsistenz
  - Gibt es zyklische Referenzen?

QS: Design Validierung

Objekt: Systementwurfsdokument

Methode: Testfallbeschreibung

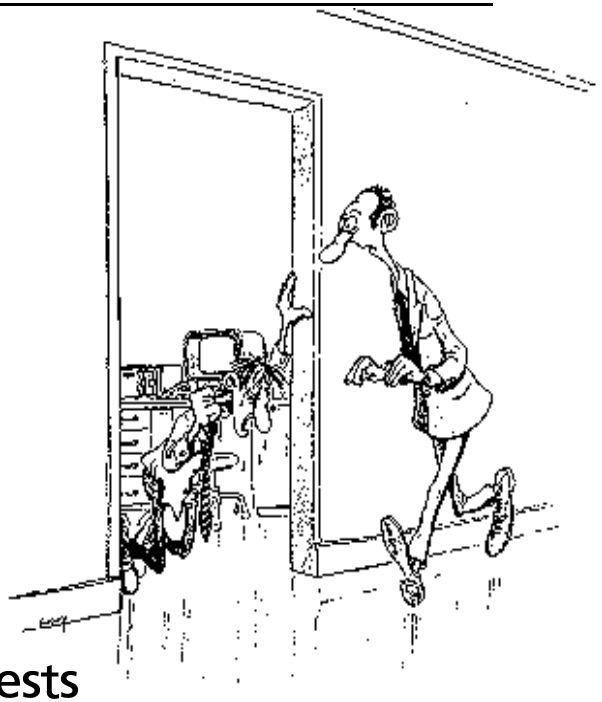


- Systementwurf:
  - Definition von Ein- und Ausgabeformaten
  - Definition von Dateien und Datenbasen
  - Definition der Ablauflogik
  - Definition der Modul-Dekomposition
- Testfälle:
  - Ein- und Ausgabedatensätze
  - Daten-Wörterbuch-Konzept
  - Aufrufhierarchie-Tests
  - Schnittstellen-Überprüfung

QS: Funktionstests, strukturelle Tests

Objekt: Einzelmodule

Methode: black-box und glass-box Skripten

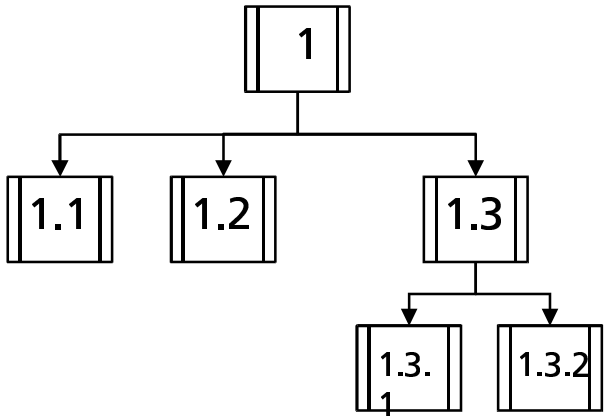


- Kombinatorische Explosion bei erschöpfenden Tests (exhaustive / exhausting)
- Verschiedene Testüberdeckungs-Begriffe
  - (Datenüberdeckung, Anweisungsüberdeckung, Pfadüberdeckung, Zweigüberdeckung, ...)
- Äquivalenzklassenbildung, Grenzwertanalyse, Entscheidungstabellen
- Werkzeugunterstützung möglich (Coverage, Stubgeneratoren)

Tests: Systemtests

Objekt: Teilsysteme

Methode: Stümpfe und Treiber



- Top-down Integration: Module als Stümpfe, Rapid Prototyping
- Bottom-up Integration: Testtreiber als Master, hierarchisches Vorgehen

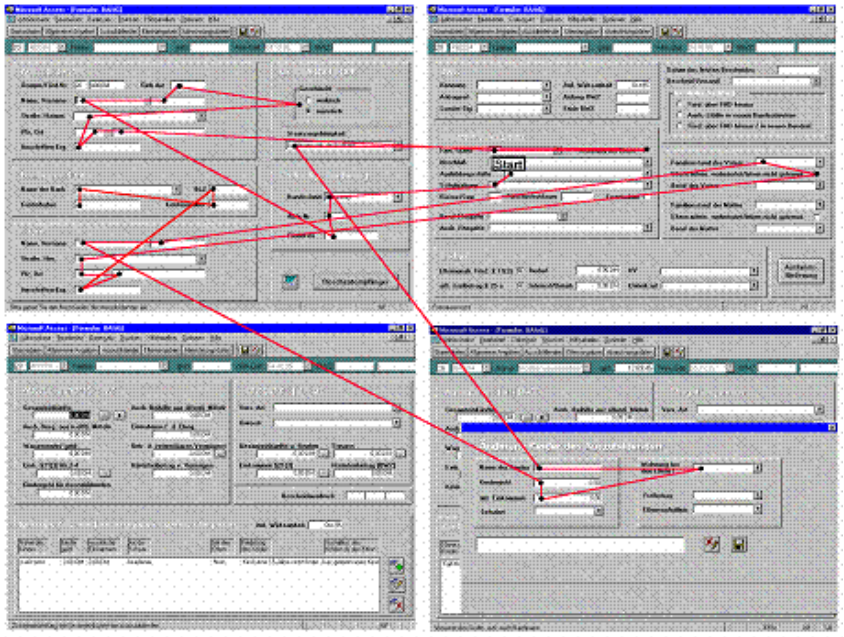
Test: Akzeptanztest

Objekt: Gesamtsystem

Methode: Benutzertestprotokolle

- Alpha- und Beta- Tests: mit bzw. ohne Entwicklerbeteiligung
- Protokollierung *aller* Benutzeraktionen! (ErgoLab)
- Installationsvorgang, Hilfesystem, Migrationsroutinen mit berücksichtigen!

- Desktop-System
  - Fensteraktivierung
  - Mausbewegungen
  - Blickrichtung



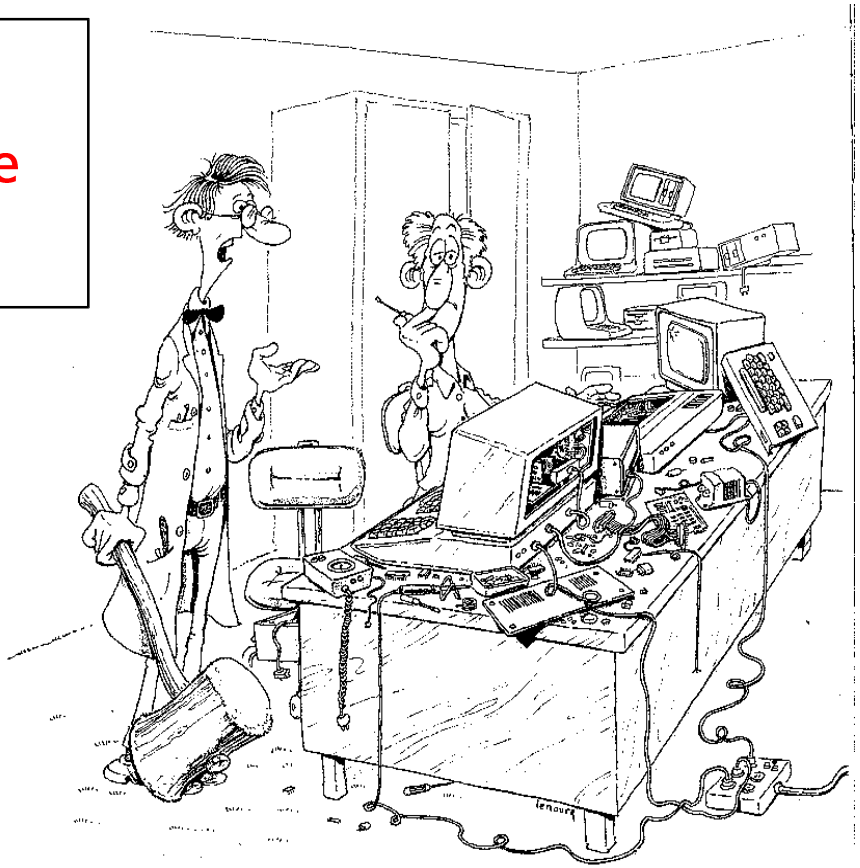
- Cockpitsoftware
  - Knöpfe (System & Umgebung!)
  - Fahrverhalten
  - Gesamtsituation incl. Sichtverhältnisse, Akustik etc.

    „ menschliches Versagen“ ist fast immer „fehlerhaftes Design“

QS: Konfigurations- / Einsatztest

Objekt: Eingebettete Hard/Software

Methode: HW-in-the-loop Tests



- kombinatorische Explosion n^m
- Beispielnutzer nicht repräsentativ
- HiL-Teststände, generierte Testfälle

- ISO 9000, CMM / CMMI, SPICE / OO-Spice, Bootstrap,…
- jeder Standard ist besser als kein Standard
  - durchschnittlich *immer* eine Verbesserung feststellbar

- z.B. Basiselemente ISO 9000-3
  - QM-Politik, Qualitätsmanager, QM-Handbuch
  - QM-Aufzeichnungen, Dokumentation
  - Dokumentenverwaltung, Versionskontrolle
  - Dokumentierte Prozesse,
  - Phasenpläne, Projektpläne, Testpläne, Wartungspläne
  - Schulung und Mitarbeiterbeteiligung

QS-Elemente nach DIN ISO 9000-3

Rahmen eines QS

- Verantwortung der obersten Leitung
- QS-System
- Interne Qualitätsaudits
- Korrektur-maßnahmen

Phasenübergreifende QS-Elemente

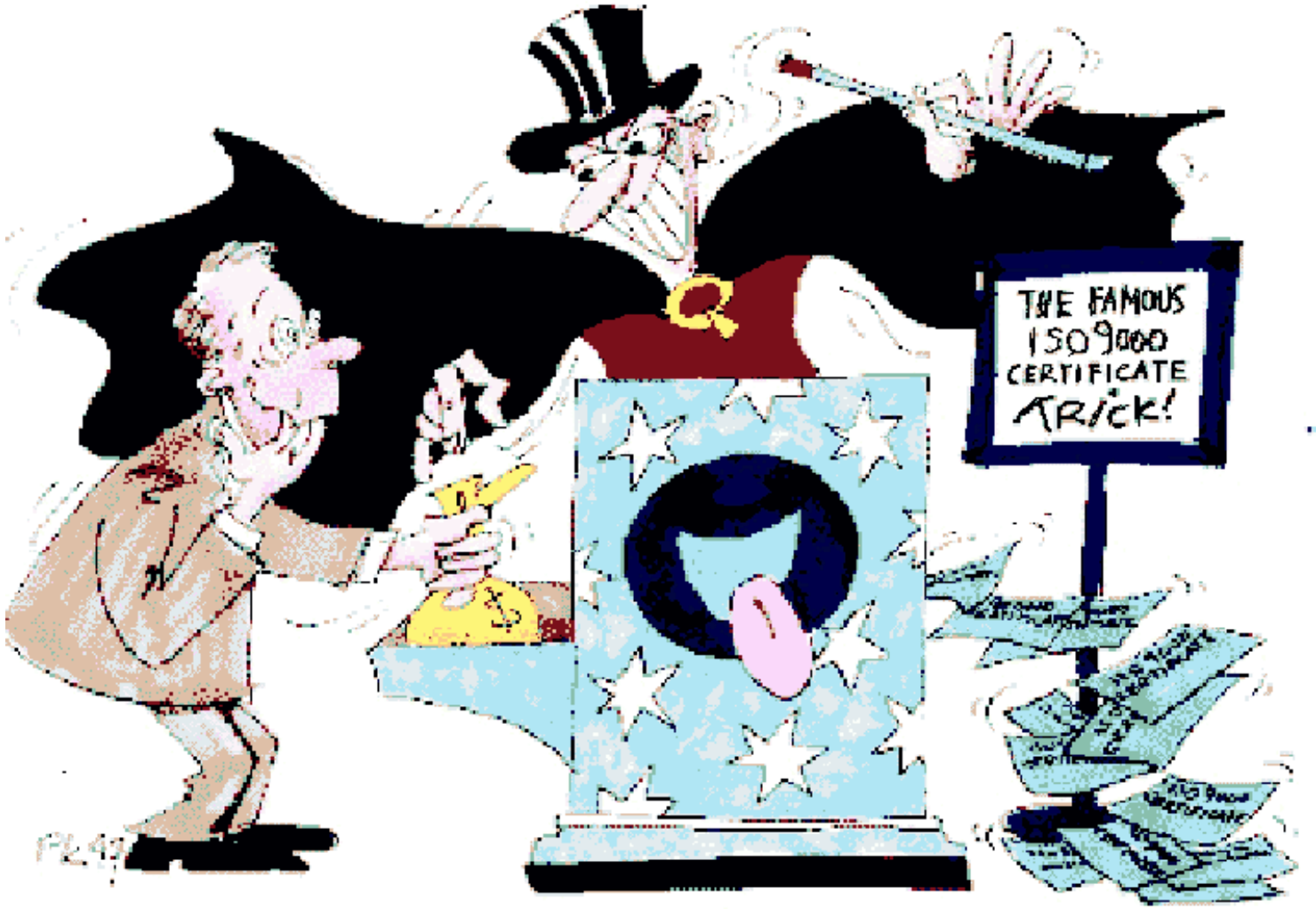
- Konfigurations-management
- Dokumentenlenkung
- Qualitäts-aufzeichnungen
- Messungen
- Regeln, Praktiken, Übereinkommen
- Werkzeuge und Techniken
- Beschaffung
- Bereitgestelltes Softwareprodukt
- Schulung

Phasenbezogene QS-Elemente

- Vertragsüberprüfung
- Spezifikation
- Entwicklungsplanung
- QS-Planung
- Design und Implementierung
- Testen und Validierung
- Abnahme
- Vervielfältigung, Lieferung und Installierung
- Wartung

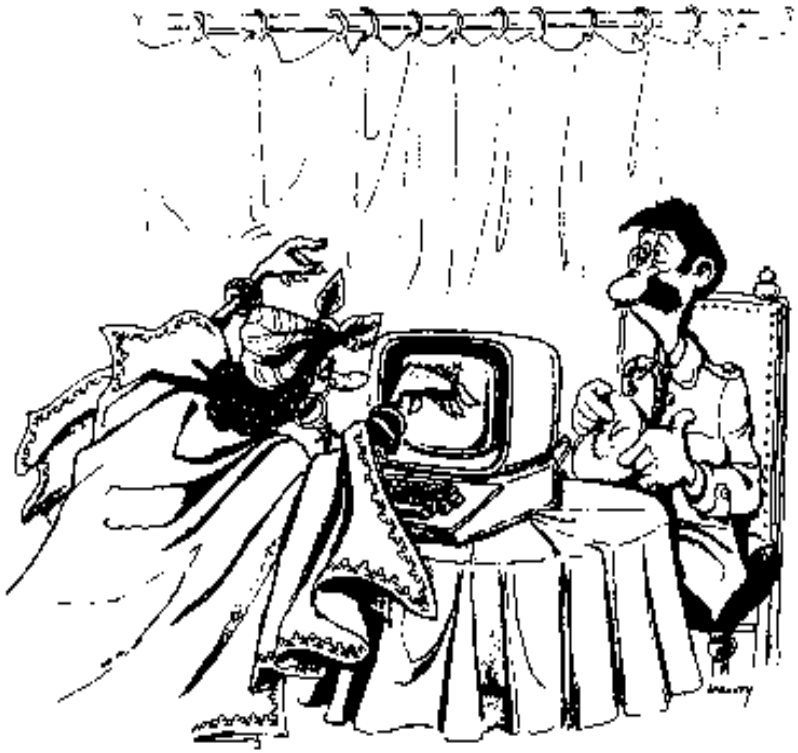
Level	Focus	Key Process Areas
Level 5 Optimizing	Continuous improvement	Process Change Management Technology Change Management Defect Prevention
Level 4 Managed	Product and process quality	Software Quality Management Quantitative Process Management
Level 3 Defined	Engineering process	Organization Process Focus, Org. Process Definition Peer Reviews, Training Programs, Intergroup Coordination, SW Product Engineering Integrated Software Management
Level 2 Repeatable	Project management	Requirements Management, SW Project Planning SW Project Tracking and Oversight SW Subcontract Management, SW Quality Assurance
Level 1 Initial	Heroes	SW Configuration Management No KPAs at this time

Source: www.software.org/quagmire/descriptions/sw-cmm.asp



Fraunhofer Institut
Rechnerarchitektur
und Softwaretechnik

- Begriffe
- Der Vorgang des Testens
  - Warum wer was wozu wie wann testet

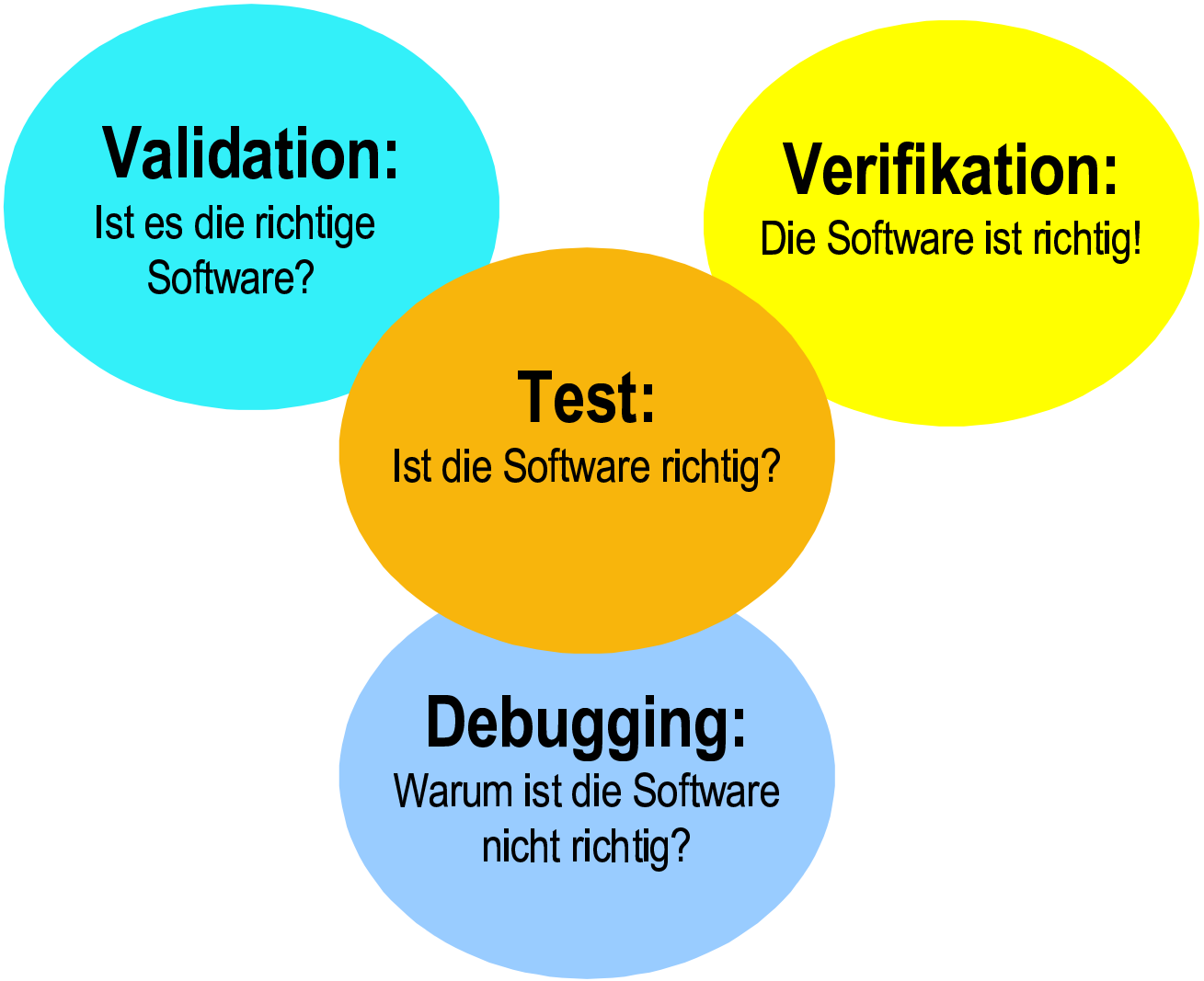


Experimentieren = Ausführen *einzelner* Versuche zur Erlangung einer neuen Erkenntnis

Probieren = experimentelles Feststellen der *Qualität* eines Objekts

Testen = *systematisches* Probieren nach verschiedenen Qualitätskriterien

Prüfen = Testen einer *Serie* gleichartiger Objekte



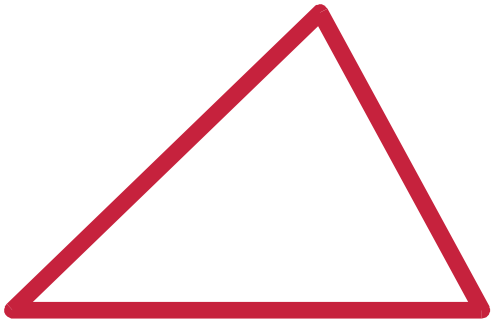
- systematisches Ausführen von Software mit dem Ziel, Fehler zu finden
- Test erfolgreich ⬅ ➡ Fehler gefunden

- Softwareentwicklung konstruktiv, -test destruktiv?
  - in gewisser Hinsicht ja

  aber
  - es erfordert viel Kreativität, die Denkmuster und –fehler anderer Menschen zu durchschauen

- Entwicklung und Test setzen Spezifikation um
  - „ überflüssige Doppelarbeit“?
  - „ unvollständig“?
  - „ einfacher“ ?
  - „ weniger Aufwand“?

- Funktion mit drei int-Eingabewerten (x,y,z), die als Längen von Dreiecksseiten interpretiert werden
- Berechnet, ob das Dreieck gleichseitig, gleichschenklig oder ungleichseitig ist (3-Zeiler)



- Schreiben Sie für diese Funktion Testfälle auf! (jetzt!)

1. Haben Sie einen Testfall für ein zulässiges gleichseitiges Dreieck?
2. Haben Sie einen Testfall für ein zulässiges gleichschenkliges Dreieck? (Ein Testfall mit 2,2,4 zählt nicht.)
3. Haben Sie einen Testfall für ein zulässiges ungleichseitiges Dreieck? (Beachten Sie, dass Testfälle mit 1,2,3 und 2,5,10 keine Ja-Antwort garantieren, da kein Dreieck mit solchen Seiten existiert.)
4. Haben Sie wenigstens drei Testfälle für zulässige, gleichschenklige Dreiecke, wobei Sie alle drei Permutationen der beiden gleichen Seiten berücksichtigt haben? (z.B. 3,3,4; 3,4,3; 4,3,3)

1. Haben Sie einen Testfall, bei dem eine Seite gleich Null ist?
2. Haben Sie einen Testfall, bei dem eine Seite einen negativen Wert hat?
3. Haben Sie einen Testfall mit 3 ganzzahligen Werten, in dem die Summe zweier Zahlen gleich der dritten ist? (D.h., wenn das Programm 1,2,3 als ungleichseitiges Dreieck akzeptiert, so enthält es einen Fehler.)
4. Haben Sie wenigstens drei Testfälle für Punkt 7, wobei Sie alle drei Permutationen für die Länge jeweils einer Seite als Summe der beiden anderen Seiten berücksichtigt haben? (z.B. 1,2,3; 1,3,2; 3,1,2.)
5. Haben Sie einen Testfall mit drei ganzzahligen Werten größer Null, bei dem die Summe aus zwei Zahlen kleiner als die dritte ist? (z.B. 1,2,4 oder 12,15,30)

1. Haben Sie wenigstens drei Testfälle für Punkt 9, wobei Sie alle drei Permutationen berücksichtigt haben? (z.B. 1,2,4; 1,4,2; 4,1,2.)
2. Haben Sie einen Testfall, in dem alle drei Seiten gleich Null sind (d.h. 0,0,0)?
3. Haben Sie wenigstens einen Testfall mit nichtganzzahligen Werten?
4. Haben Sie wenigstens einen Testfall, in dem Sie eine falsche Anzahl von Werten angeben (z.B. zwei statt drei ganzzahlige Werte)?
5. Haben Sie zusätzlich zu jedem Eingangswert in allen Testfällen die erwarteten Ausgabewerte angegeben?

1. Test mit maximalen Werten
2. Test auf Zahlbereichs-Überlaufbehandlung
3. Test mit unzulässigen Eingabezeichenfolgen

- Durchschnittswerte erfahrener Programmierer: 7-8
- *„ Diese Übung sollte zeigen, dass das Testen auch eines solch trivialen Programms keine leichte Aufgabe ist. Und wenn das wahr ist, betrachten Sie die Schwierigkeit, ein Flugleitsystem mit 100.000 Befehlen, einen Compiler oder auch nur ein gängiges Gehaltsabrechnungsprogramm zu testen.“* (1979)
- Heute: 1-10 MLoC

Mangelnde Qualität kostet Geld!

- Benutzerakzeptanz, Kundenverlust
- Fehlerkorrektur- und Folgekosten
- Gewährleistung, Produkthaftung
- Sicherheitsprobleme

Gates' Mathematik

Genie oder Unfähigkeit

Redmond (ag) – Microsofts Java-Compiler widersetzt sich nicht nur Suns Kompatibilitätstest, er definiert auch die Fakultät etwa von 5 neu.

Bill Gates ist ein Freund von kleinen Zahlen. Er hat nicht nur erfolgreich die Anzahl verbreiteter Betriebssysteme reduziert, er schickt sich nun auch an, das mathematische Modell der Fakultät zu revolutionieren. Tüftler haben entdeckt, daß ihre Java-Programme nach der Optimierung mit Microsofts Just-in-time-Compiler neue Ergebnisse liefern – die Zahlen werden kleiner. Die Fakultät von 5 ist dadurch auf überschaubare 15 geschrumpft, vor Gates war sie noch dreistellig.

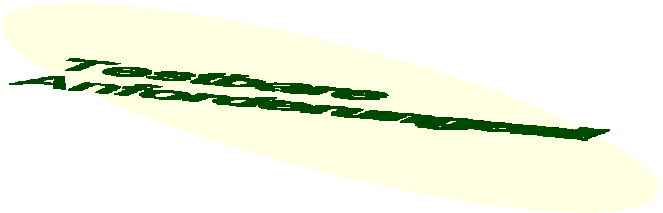
FIRST
Fraunhofer Institut Rechnerarchitektur und Softwaretechnik

- Qualität = Übereinstimmung mit den Anforderungen
  - ➡ nur spezifizierte Anforderungen testbar!

  z.B.: Funktionalität,
  Zweckdienlichkeit, Robustheit,
  Zuverlässigkeit, Sicherheit, Effizienz,
  Benutzbarkeit, Geschwindigkeit,
  Preisgünstigkeit, ...

- „ Design for Testability“
  - Testgerechte Beschreibung der Anforderungen
  - Beteiligung der QS bei der Anforderungsanalyse

- Quantifizierung der Anforderungen
  - nicht: „akzeptable Antwortzeiten sind wichtig“
  - sondern: „Antwortzeit höchstens 5 Sekunden,
  - in 80% der Fälle kleiner als 3 Sekunden“
- Kategorisierung der Anforderungen
  - (unerläßlich / wichtig / erwünscht / irrelevant / unerwünscht)
- Formalisierung so weit wie möglich
  - Diagramme, Formeln, Pseudocode, …
  - Pre- und Postconditions, Invarianten

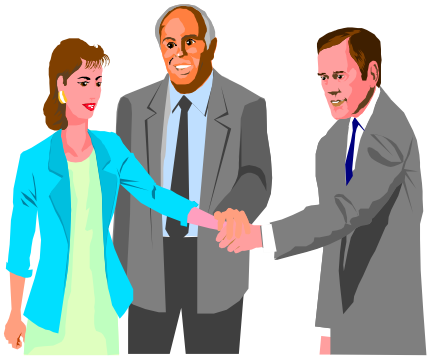


- „ Bugs“ schleichen sich nicht ein, sie werden gemacht
- Niemand kennt das Produkt so genau wie der Entwickler

- Motivation des Entwicklers: Debugging und Verifikation
- Motivation des Testers: Fehler identifizieren

- QS-Abteilung
  - Trennung von Produkt-verantwortung und Qualitätsverantwortung
  - Produktivität statt Qualität
  - mangelnde Fehleranalysen

- Qualitätsrunden
  - Peer Review oder Walkthrough
  - aktive Beteiligung aller Mitwirkenden
  - Gleichberechtigung
  - Konsensbildung



Moderation durch unabhängige Berater!

- Konzentration auf Benutzersicht

  (*Relevante* Testfälle, Benutzbarkeitsprobleme)

- Systematische Vorgehensweise

  (Testplanung und -dokumentation)

- Einbeziehung *aller* Komponenten

  (auch: Dokumentation, Installationsroutinen usw.)

- Automatisierung wo möglich

  Kosten-Nutzen-Rechnung

- Fehlerverfolgungssystem

  effektiver Einsatz Managementaufgabe!

Oberstes Ziel: Fehlervermeidung

➔ Tester nicht für Fehleranalyse und -lokalisierung zuständig!

… und schon gar nicht für die Erforschung der Ursachen!

Individualverantwortlichkeit

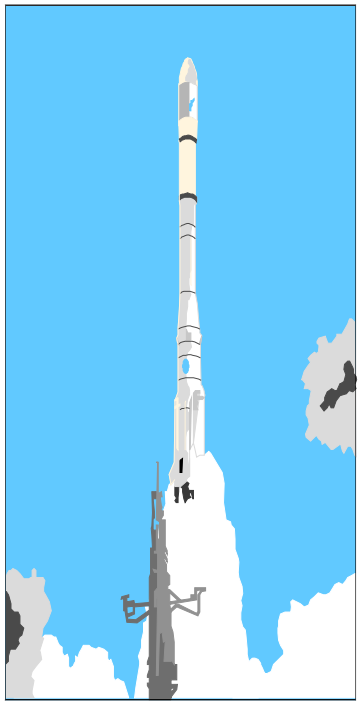
- Diskriminierungen
- Schuldzuweisungen
- verminderte Produktivität

Teamverantwortung

- Kooperation
- Qualitätsbewußtsein
- Produktverbesserung

- Globale Fehlerrückverfolgung

Explosion durch Sprengung,
weil
Schräglage durch Ruder, **weil**
vermeintliche Fehlbahn, **weil**
falsche Lagedaten, **weil**
Sensorausfall, **weil**
Übertragungsfehler, **weil**
Zahlbereichsüberlauf, **weil**
undokumentierte Anforderung,
weil
ungetestetes Ariane4-Bauteil,
weil
Termin- und Kostendruck



Handlungsempfehlungen:
...
Fehlerkorrekturmaßnahmen,
klare Aufgabenverteilung,
Software-Redundanz,
Leistungsbedarfsermittlung,
Ausnahmebehandlung,
formale Dokumentation,
vollständige
Integrationstests,
QS-getriebene Entwicklung

- Natürliche Sprache mehrdeutig!
- Beispiel:

> „ alle 30 Sekunden sollen die Werte der Sensoren abgelesen werden; wenn die Standardabweichung 0,25 überschreitet, soll die Normalisierungsprozedur ausgeführt werden, anschließend sollen die Werte an das Analysepaket weitergegeben werden.“

Akzeptanztest ergibt falsche Analysewerte.

Problem: Komma!

- (a) Festlegung der Schnittstellen und Ereignisse
- (b) Festlegung des reaktiven Verhaltens

(a)

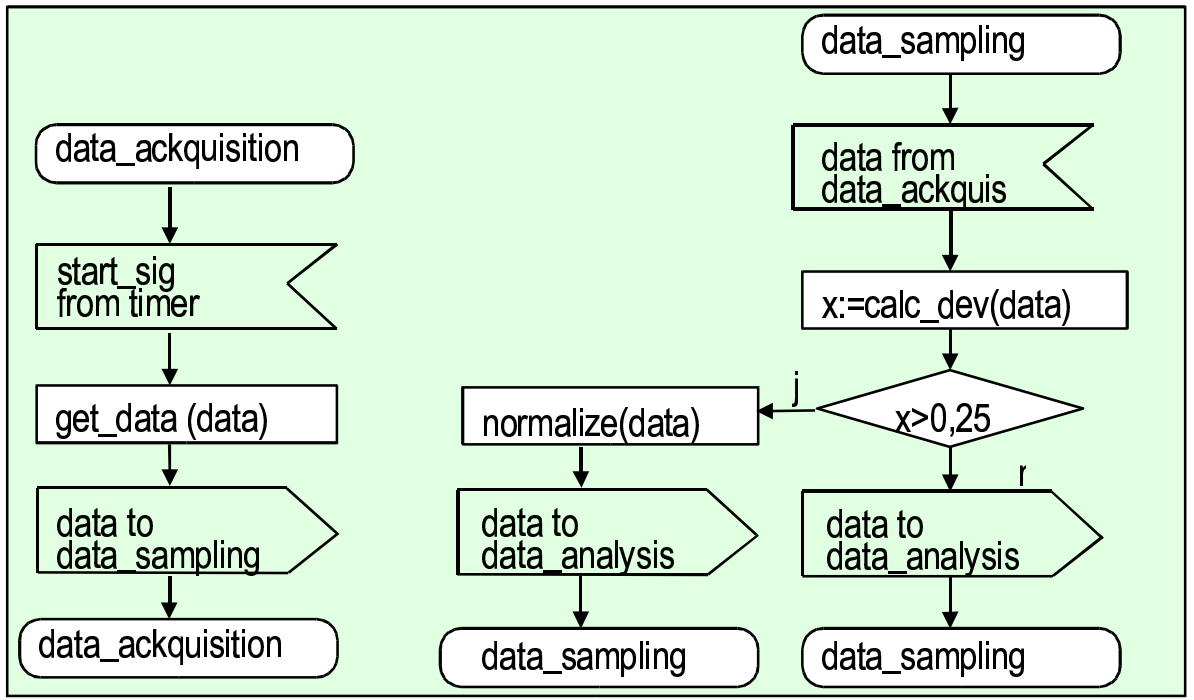
Methoden:

get_data (...)
calc_dev(...)
normalize (...)
set_timer (...)

Signale:

data: ...
deviation: ...
start_sig: ...

(b)



Noch besser:

Deklarative statt operationaler Beschreibungsformen
(*was* statt *wie*)

Alle Werte sollen normalisiert analysiert werden.

Logische Spezifikation der gewünschten Eigenschaften
(formale Grundlage)

For all x exists y: y=normalize(x) and sometime analyzed(y)

- normative Vorgaben IEC61508, EN50128
  - SIL4: formale Methoden „HR“
- „ ausführbare Pflichtenhefte“
  - Beschreibung von Anforderungen in standardisierter natürlicher Sprache (Anforderungsmuster)
  - Verfeinerungen und automatische bzw. semiautomatische Transformation in temporale Logik
  - Semantische Netze zur Modellierung der Relationen
  - Ableitung von Gefährdungen und HiL-Tests
  - Zulassungsunterstützung
- bislang reines Forschungsprojekt
  - Dissertation im Bereich Bahntechnik
  - Anwendungsbeispiel: Achszähler