



Qualitätssicherung von Software

Prof. Dr. Holger Schlingloff

Humboldt-Universität zu Berlin
und
Fraunhofer FIRST

Kapitel 2. Testverfahren



Abgrenzung

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**

Fragestellung beim Testen

Validation:

Ist es die richtige
Software?

Verifikation:

Die Software ist richtig!

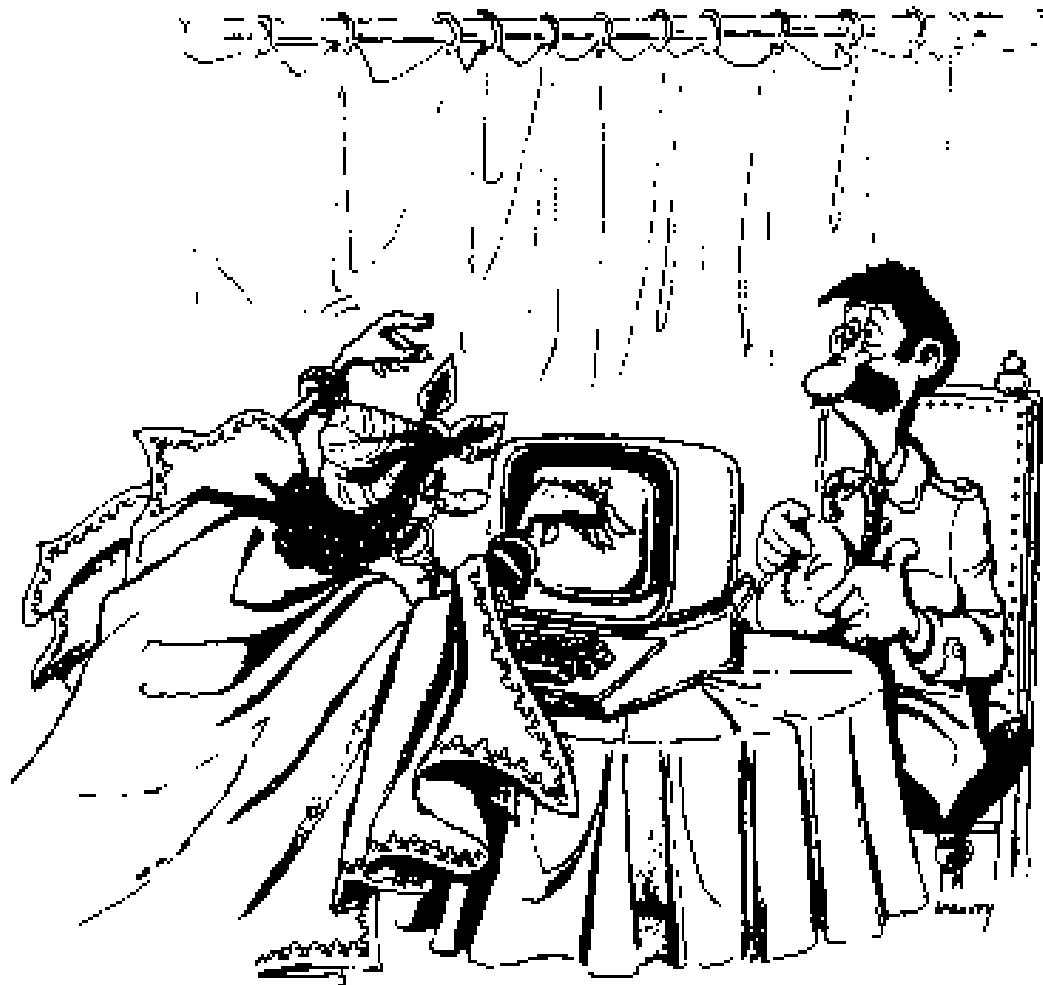
Test:

Ist die Software richtig?

Debugging:

Warum ist die Software
nicht richtig?

Der Vorgang des Testens



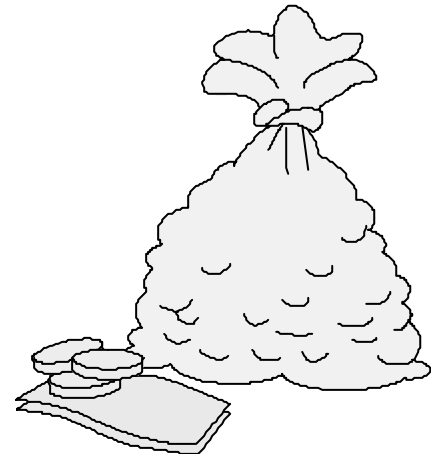
w-Fragen



- warum
 - was
 - wer
 - wozu
 - wie
 - wann
- sollen wir testen?

Warum sollen wir testen?

- Mangelnde Qualität kostet Geld:
- **Benutzerakzeptanz, Kundenverlust**
- **Fehlerkorrektur- und Folgekosten**
- **Gewährleistung, Produkthaftung**
- **Sicherheitsprobleme**



Genie oder Unfähigkeit

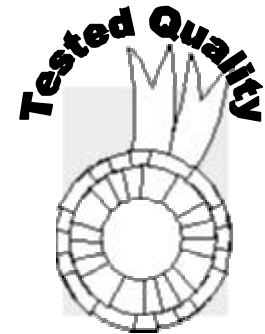
Redmond (ag) – Microsofts Java-Compiler widersetzt sich nicht nur Suns Kompatibilitätstest, er definiert auch die Fakultät etwa von 5 neu.

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.

Was sollen wir testen?

- Qualität = Übereinstimmung mit den Anforderungen

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



- **Anforderungsanalyse unabhängig von Marktverfügbarkeit**
- 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)

**Testbare
Anforderungen!**

Standard- versus Individual- Software

- **Standardsoftware:**
- **Massenprodukt**
- **Starr**
- **Separat**
- **Billig**




- **Individualsoftware:**
- **Kundenspezifisch**
- **Adaptierbar**
- **Integrierbar**
- **Teuer**



Beim Einsatz von COTS-Software reduziert sich Testen meist auf Probieren

Für Custom-Software sind i.a. Akzeptanz- und Korrektheitstests notwendig

Wer soll testen?

- „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 Produktverantwortung und Qualitätsverantwortung
- Produktivität statt Qualität
- mangelnde Fehleranalysen



Moderation durch unabhängige Berater!

• Qualitätsrunden

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



Wozu sollen wir testen?



Fehlervermeidung statt Fehlererkennung!

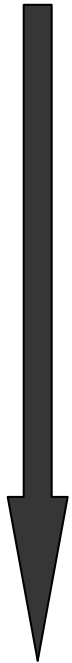
Individualverantwortlichkeit

- Diskriminierungen
- Schuldzuweisungen
- verminderte Produktivität

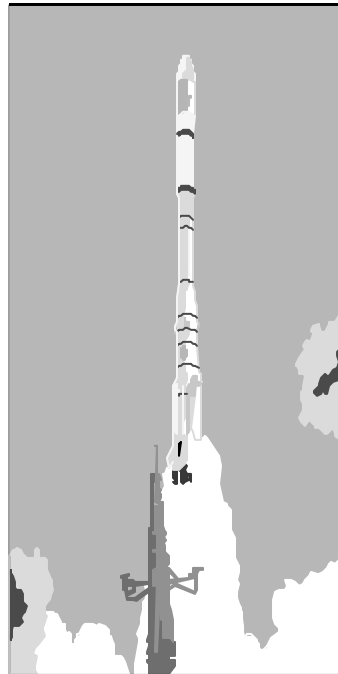
Teamverantwortung

- 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



Wie sollen wir testen?

- **Konzentration auf Benutzersicht**

(*Relevante* Testfälle, Benutzbarkeitsprobleme)

- **Systematische Vorgehensweise**

(Testplanung und -dokumentation)

- **Einbeziehung *aller* Komponenten**

(auch: Dokumentation, Installationsroutinen usw.)

- **Automatisierung wo möglich**

Notwendigkeit der Formalisierung

- 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!

Formale Anforderungsdefinition

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

(a)

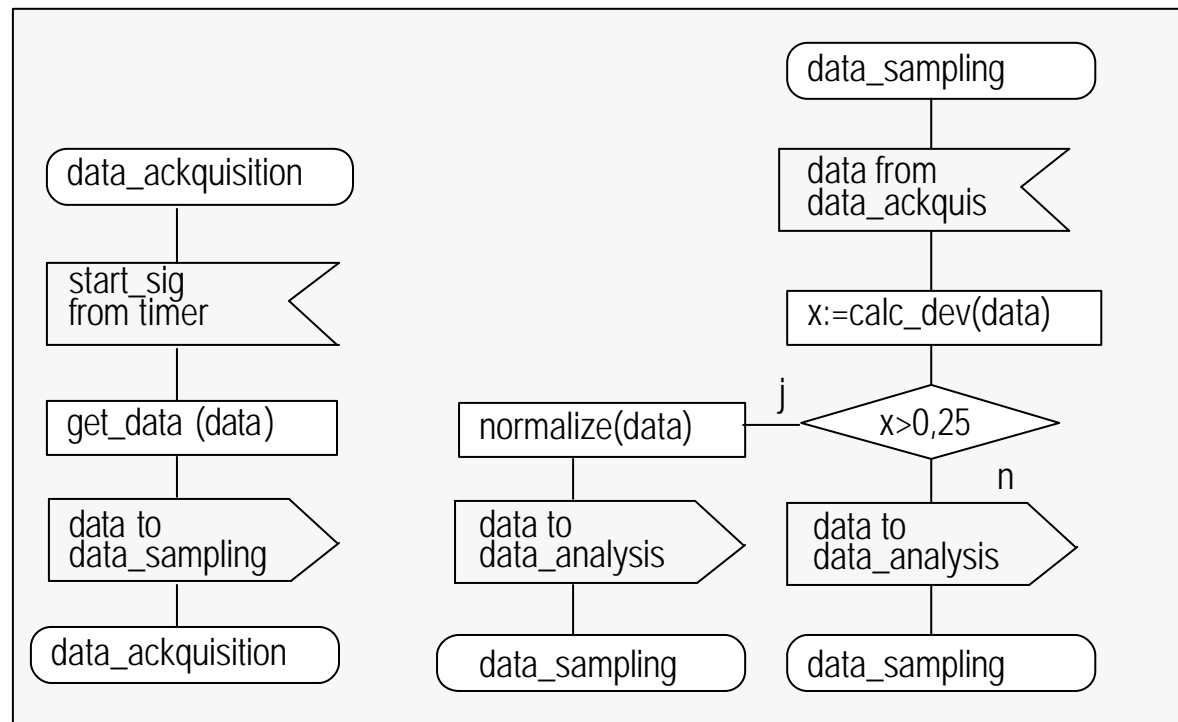
Methoden:

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

Signale:

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

(b)



Noch besser:

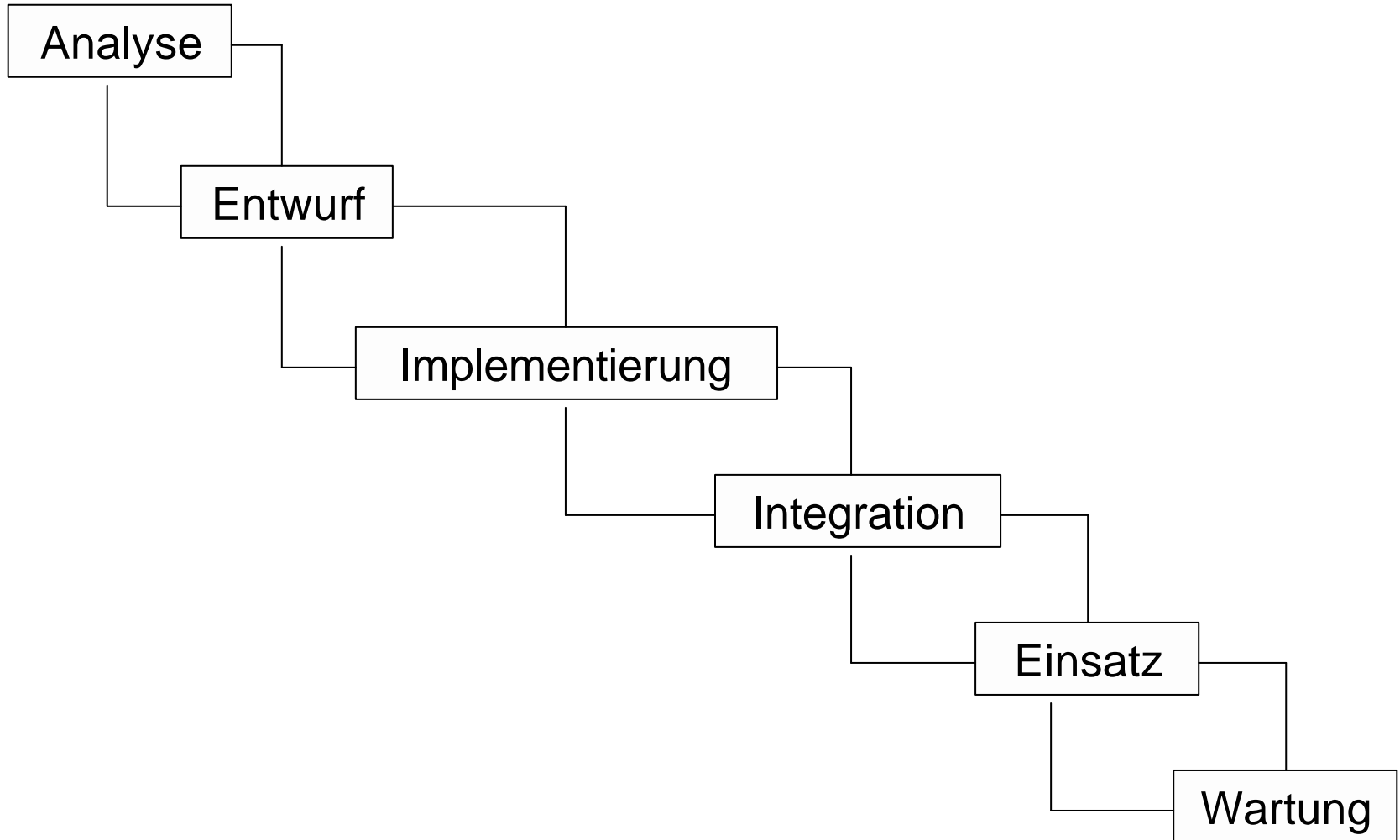
Deklarative statt operationaler Beschreibungsformen
(*was* statt *wie*)

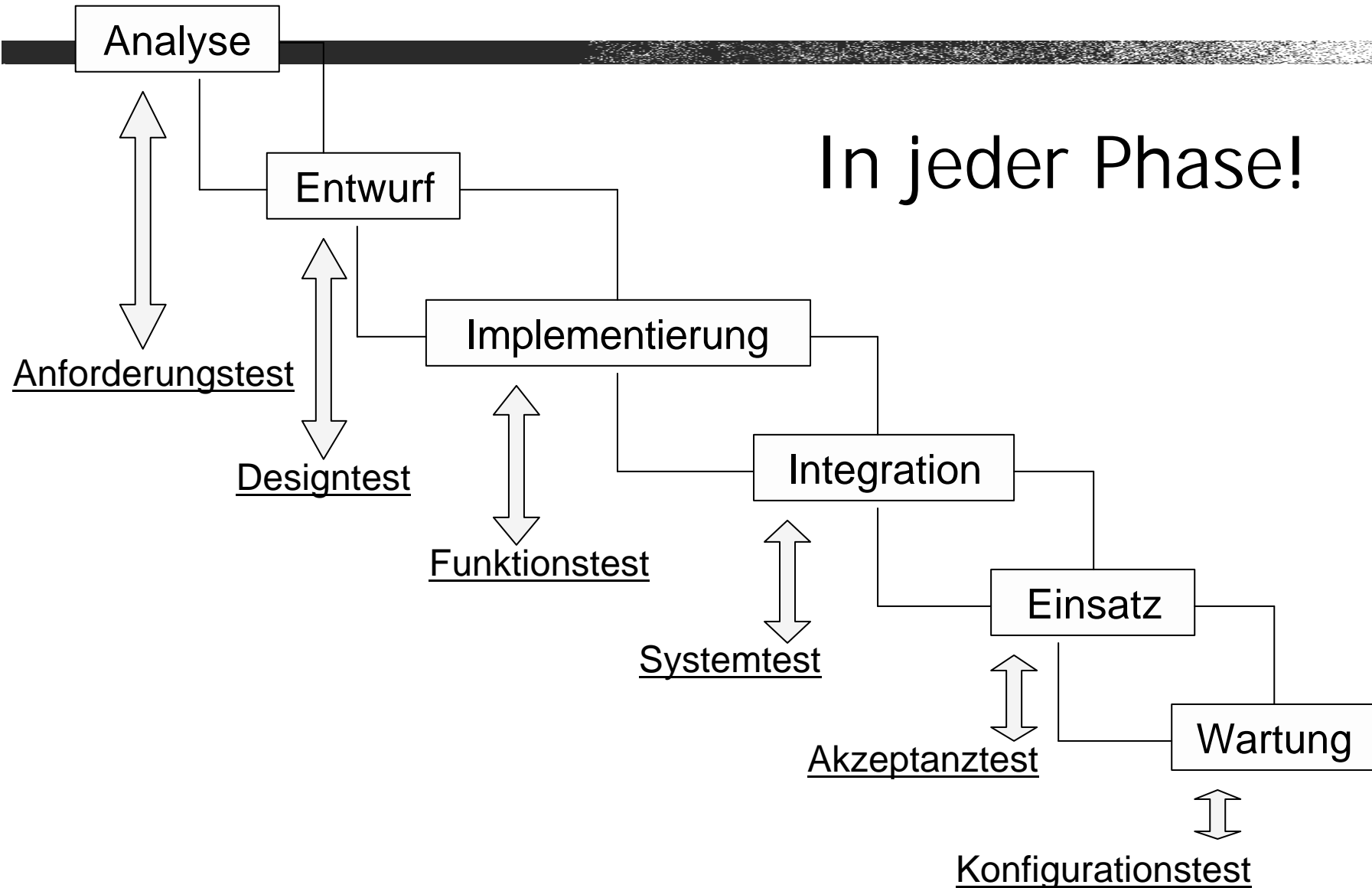
Alle Werte sollen normalisiert analysiert werden.

Logische Spezifikation der gewünschten Eigenschaften
(formale Grundlage)

For all x exists y : $y = \text{normalize}(x)$ and sometime analyzed(y)

Wann sollen wir testen?





Kapitel 2. Testverfahren

2.1 Testen im SW-Lebenszyklus

2.2 funktionsorientierter Test

- Modul- oder Komponententest
- Integrations- und Systemtests

2.3 strukturelle Tests, Überdeckungsmaße

2.4 Test spezieller Systemklassen

- Test objektorientierter Software
- Test graphischer Oberflächen
- Test eingebetteter Realzeitsysteme

2.5 automatische Testfallgenerierung

2.6 Testmanagement und -administration

Phasenbezogene Teststrategien

1. Anforderungsphase
2. Konstruktionsphase
3. Betriebsphase

1. Tests in der Anforderungsphase

Test: Anforderungstest

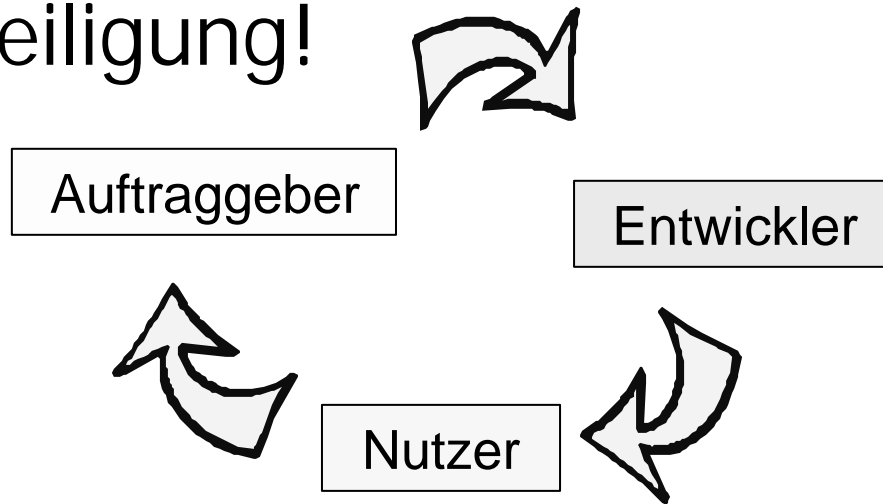
Objekt: Anforderungsdefinition

Methode: Testfähige Formulierung und
Formalisierung der Anforderungen



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

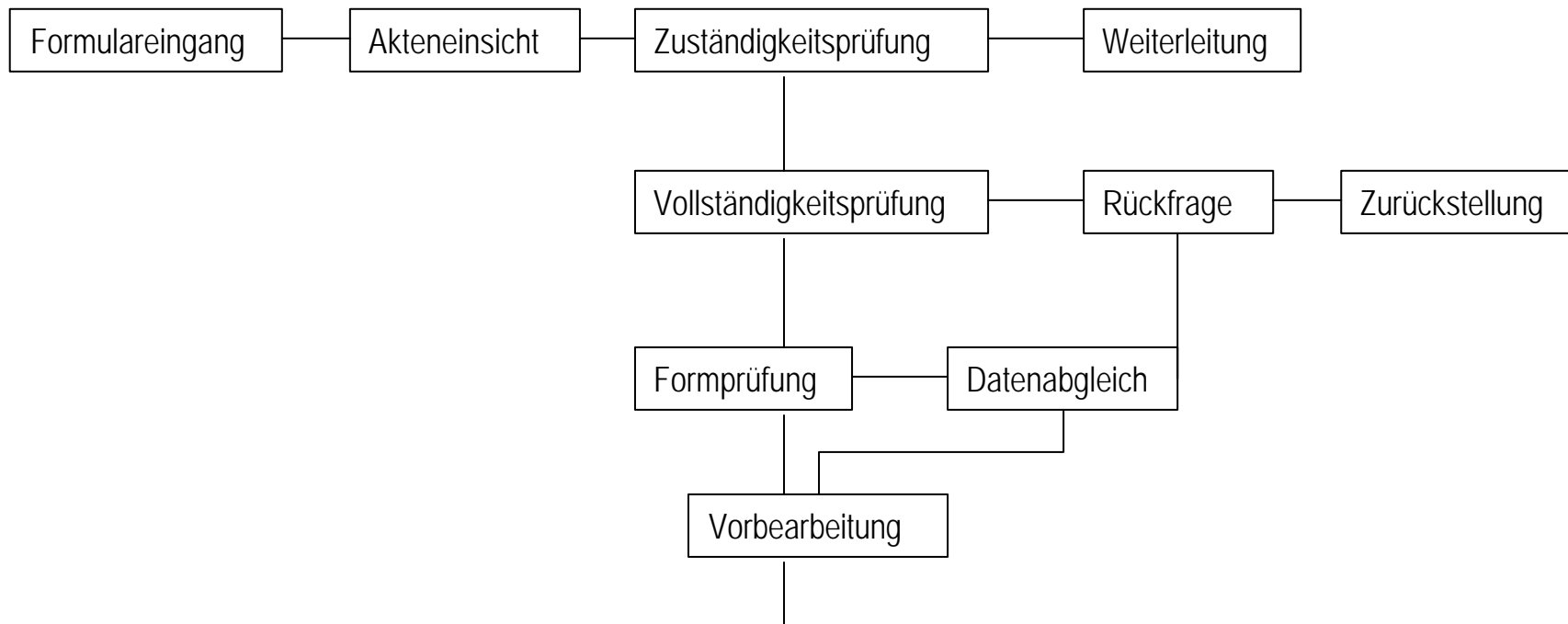
extrem wichtig:
Benutzerbeteiligung!



- **Vorgehen:** Progress reviews, Prototyping, Use cases

Formales Benutzermodell

- **z.B. als Transitionssystem:**



Vollständigkeit und Konsistenz der Anforderungsdefinition



- 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?

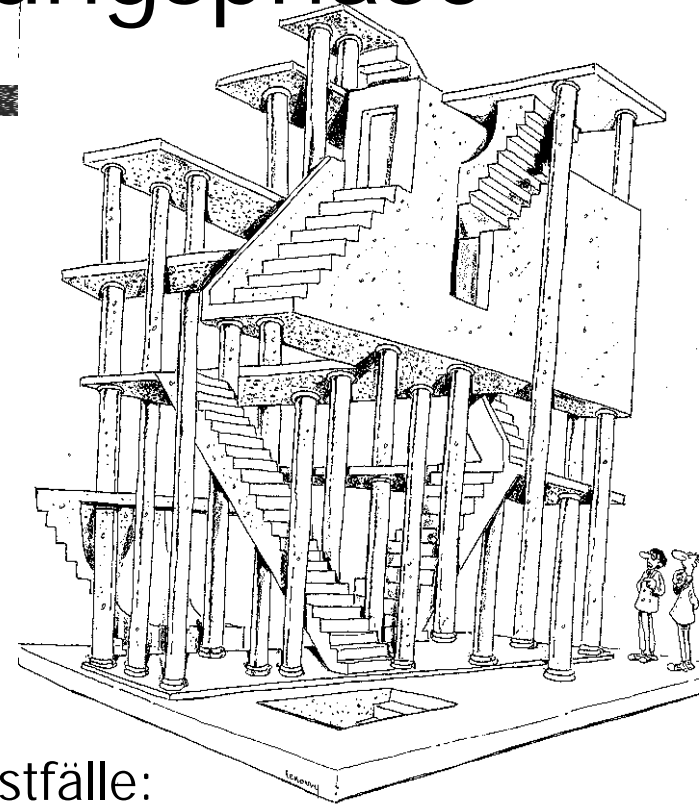
2. Tests in der Entwicklungsphase



Test: Designtest

Objekt: Systementwurfsdokument

Methode: Testfallbeschreibung



Systementwurf:

- Definition von E/A-Formaten
- Definition von Dateien und Datenbasen
- Definition der Ablauflogik
- Definition der Moduldekomposition

Testfälle:

- Ein- und Ausgabedatensätze
- Daten-Wörterbuch-Konzept
- Aufrufhierarchie-Tests
- Schnittstellen-Überprüfung

Tests: Funktionstests, strukturelle Tests

Objekt: Einzelmodule

Methode: black-box und glass-box Skripten

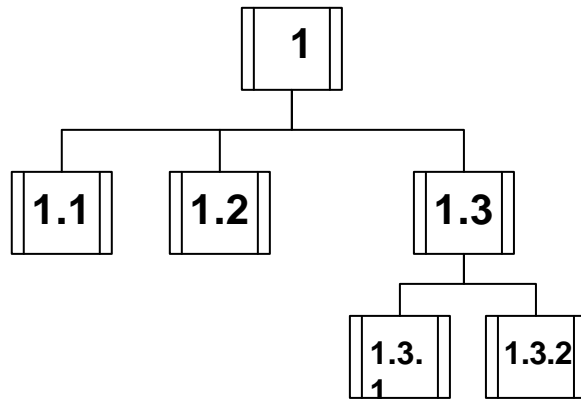


- Kombinatorische Explosion bei erschöpfenden Tests (exhaustive / exhausting)
- Verschiedene Testüberdeckungs-Begriffe
 - (Datenüberdeckung, Anweisungsüberdeckung, Pfadüberdeckung, Zweigüberdeckung, ...)
 - Äquivalenzklassenbildung, Grenzwertanalyse, Entscheidungstabellen
- Werkzeugunterstützung möglich

Tests: Systemtests

Objekt: Teilsysteme

Methode: Stümpfe und Treiber



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

3. Tests in der Betriebsphase

Test: Akzeptanztest

Objekt: Gesamtsystem

Methode: Benutzertestprotokolle



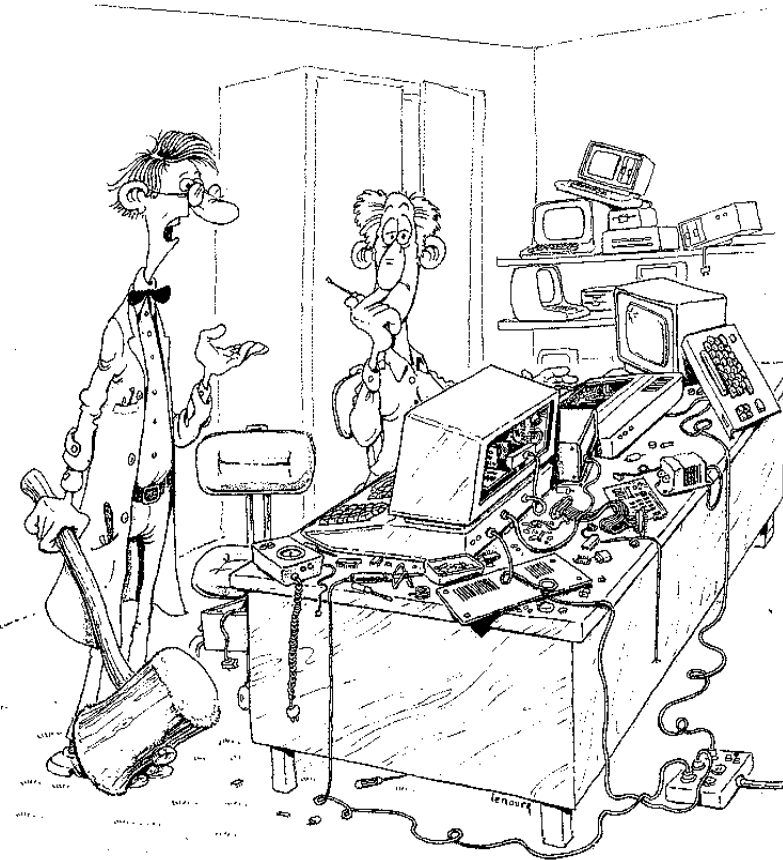
- Alpha- und Beta- Tests: mit bzw. ohne Entwicklerbeteiligung
- Protokollierung aller Benutzeraktionen!
- Installationsvorgang, Hilfesystem, Migrationsroutinen mit berücksichtigen

Test: Konfigurationstest

Objekt: Eingebettete Hard/Software

Methode: HW-in-the-loop Tests

- kombinatorische Explosion n^m
- PD-Benutzer nicht repräsentativ
- Testautomatisierung!



Bücher zum Testen

- Myers, Glenford J.: *The Art of Software Testing*. Wiley & Sons (1979) (auch in deutsch erhältlich: „*Methodisches Testen von Programmen*“; Neuauflage in Vorbereitung)
- Andreas Spillner, Tilo Linz: *Basiswissen Softwaretest*. DPUNKT Verlag (2003) (für ASQF Zertifizierung)
- Bart Broekman, Edwin Notenboom: *Testing Embedded Software*, Addison-Wesley (2003) (DC)
- Harry Sneed, Mario Winter: *Testen objektorientierter Software*. Hanser (2002)
- Edward Kit: *Software Testing in the Real World – Improving the process*. Addison-Wesley (1995)
- Dorothy Graham, Mark Fewster: *Software Test Automation - Effective Use of Test Execution Tools*. Addison-Wesley (2000)
- Cem Kaner, Jack Falk, Hung Q. Nguyen: *Testing Computer Software*. Wiley (2 ed. 1999)
- Marnie L. Hutcheson: *Software Testing Fundamentals - Methods and Metrics*. Wiley (2003)
- Georg E. Thaller: *Software-Test*. Heise (2002)