
Vorlesung „Methoden des Software Engineering“

Block D „Qualitätssicherung“

Qualitätsmanagement und Software-Test

Martin Wirsing

Einheit D.1, 19.12.2006

Ziele

- Grundlegende Begriffe des Qualitätssicherung kennen lernen
- Grundbegriffe des Testens kennen lernen

Wiederholung: Qualitätsmerkmale von Software

- Korrektheit
- Zuverlässigkeit
- Robustheit
- Benutzerfreundlichkeit
- Wartbarkeit
- Effizienz (Performanz)
- Dokumentation
- Portabilität

Qualitätsmanagement

Qualitätsmanagement umfasst alle Tätigkeiten, um die Qualität von Prozessen und Produkten sicherzustellen.

- **Qualitätsplanung**

Festlegung von überprüfbaren Qualitätszielen und Planung von Maßnahmen zur Erreichung dieser Ziele (dokumentiert im QS-Plan)

- **Qualitätssicherung**

Umsetzung der Maßnahmen des QS-Plans

QS-Maßnahmen

Die Qualitätsziele können erreicht werden durch

- **konstruktive QS-Maßnahmen:**

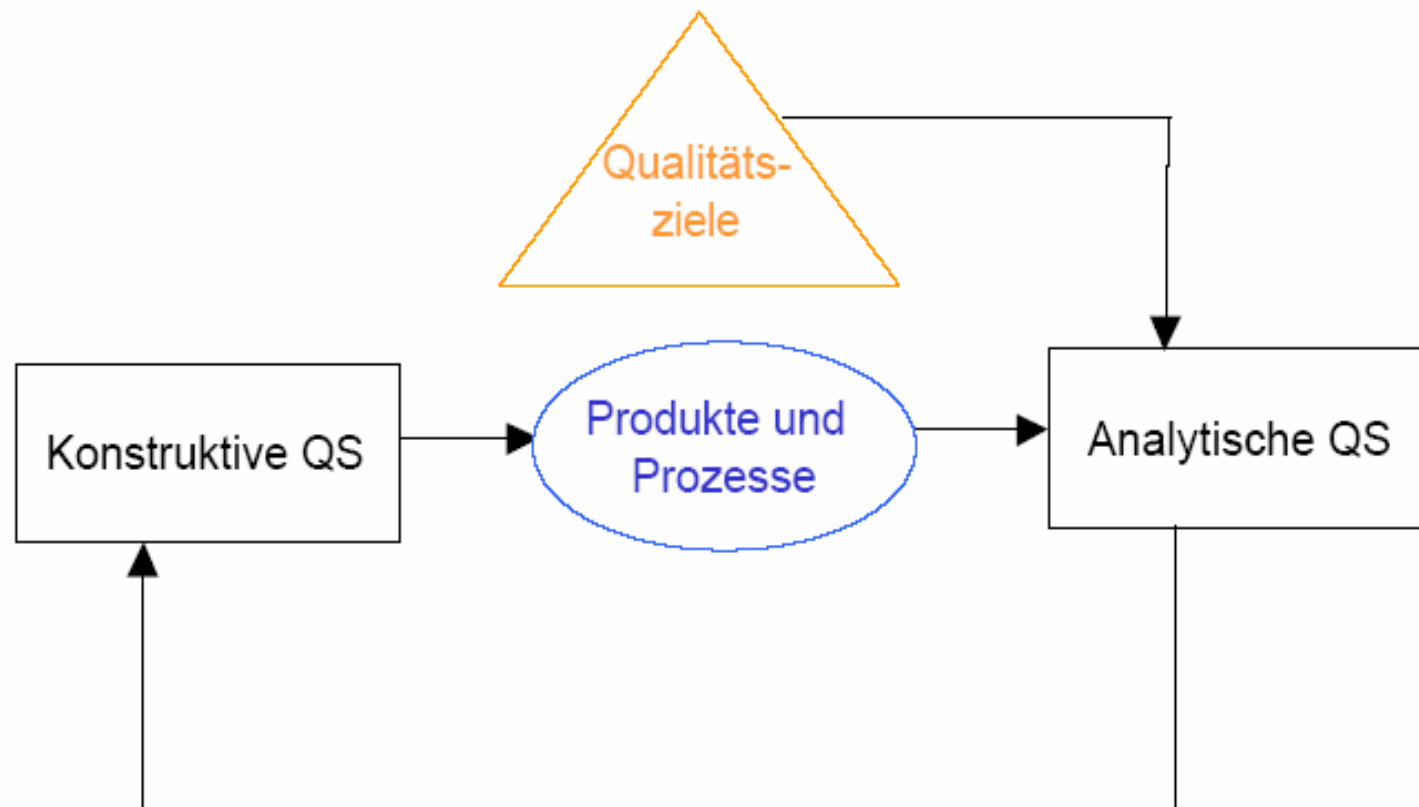
Methoden, Sprachen, Werkzeuge, Richtlinien, Checklisten, die dafür sorgen, dass das Produkt bzw. der Erstellungsprozess bestimmte Eigenschaften besitzt.

- **analytische QS-Maßnahmen:**

Prüfung und Messung des existierenden Qualitätsniveaus. Die Qualität wird gemessen, aber nicht verbessert.

Qualitätszyklus

Der Qualitätszyklus



Prinzip der Qualitätszielbestimmung

- **Festlegung von Qualitätszielen vor Beginn der Entwicklung**
 - das Team weiss, wohin es arbeitet
 - der Kunde weiss, worauf er sich einläßt

- **Die Qualitätsziele werden im QS-Plan festgelegt**
 - die Kritikalität des Systems ist ein wichtiger Anhaltspunkt für die Festlegung der Ziele

Prinzip der quantitativen Qualitätssicherung

- Ingenieurmäßige Qualitätssicherung ist undenkbar ohne die Quantifizierung von Soll- und Istwerten (Rombach, 93).
- Qualitätsziele müssen überprüfbar und messbar sein (vgl. Anforderungsanalyse)

Beispiel:

- *Kein überprüfbares Ziel:*

Das System muss performant sein.

- *Überprüfbar:*

Die Verarbeitungszeit eines Vertrages durch das System muss unter 8 Sekunden liegen

Prinzip der maximal konstruktiven Qualitätssicherung

Der Aufwand für die analytische QS soll durch eine möglichst gute konstruktive QS gering gehalten werden

- besser vorbeugen als heilen
- Fehler, die nicht gemacht werden können, brauchen auch nicht behoben zu werden

Beispiel

Beim Einsatz einer Programmiersprache mit statischer Typprüfung können Typfehler während der Laufzeit nicht auftreten

Prinzip der frühen Fehlererkennung und – behebung

- **Fehler in den frühen Projektphasen sind die teuersten**
 - Anforderungen des Kunden wurden nicht richtig verstanden
 - Inkonsistenzen im Anforderungsdokument
 - Eine verzögerte Fehlerentdeckung führt zu einem exponentiellen Kostenanstieg.

- **Aufmerksamkeit in die frühen Phasen der SW –Entwicklung investieren**
 - Fehler durch konstruktive Maßnahmen verhindern
 - Fehler, die dennoch gemacht werden, durch sorgfältige Prüfungen der Dokumente in den frühen Phasen rechtzeitig erkennen

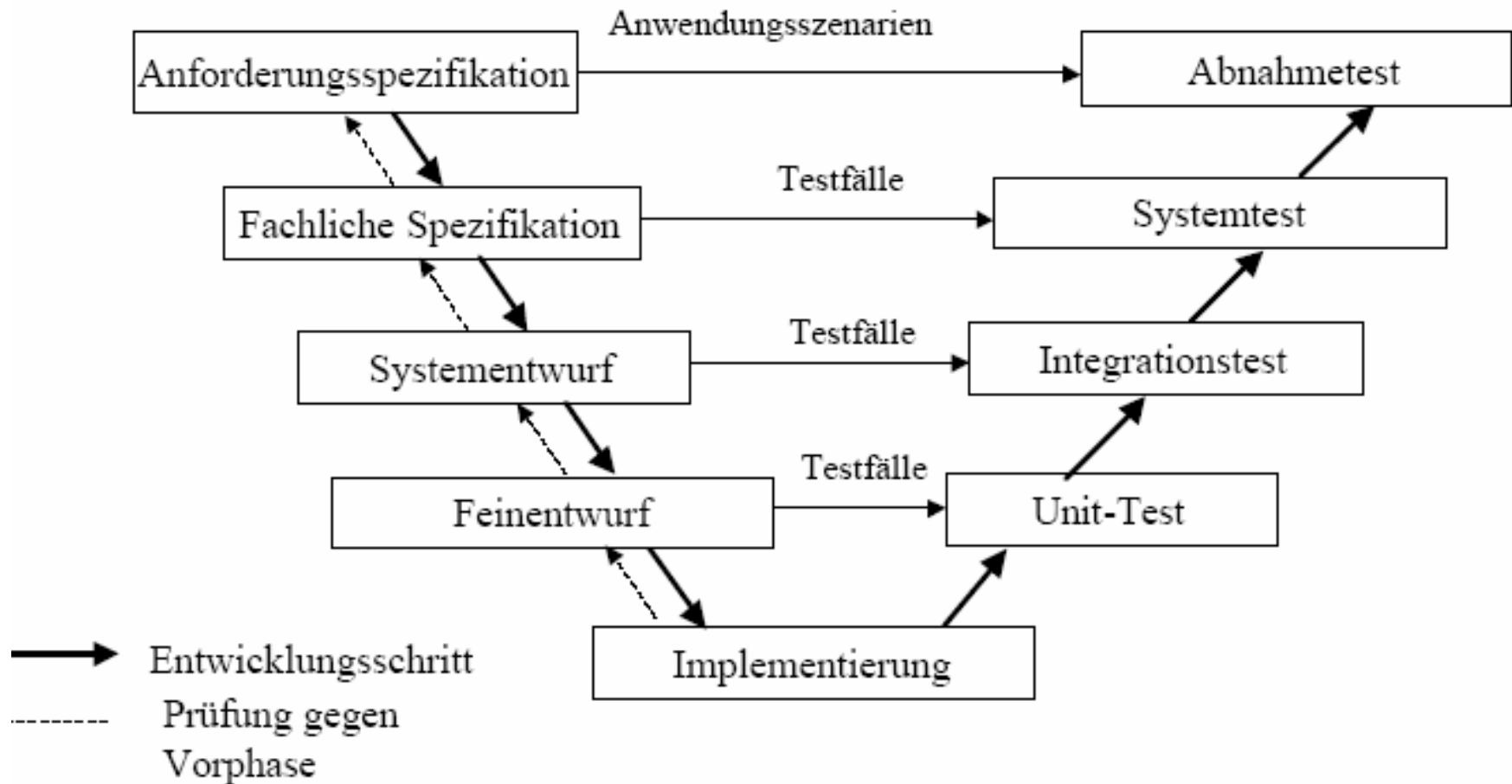
Prinzip der unabhängigen Qualitätssicherung

- Ziel der analytischen Qualitätssicherung ist es, Fehler und Mängel aufzudecken.
- Myers: „Testing is a destructive process, even a sadistic process“
- Entwickler können nicht gleichzeitig konstruktiv und destruktiv denken.
⇒ Die Entwickler eines Teilprodukts sollten nicht die analytische QS für dieses Teilprodukt vornehmen

Prinzip der entwicklungsbegleitenden Qualitätssicherung

- **Einbettung der QS in den organisatorischen Ablauf der SW -Entwicklung**
 - Ein Teilprodukt steht der nächsten Phase erst dann zur Verfügung, wenn eine bestimmte Qualität erreicht ist.
- **Beispiel**
 - **V-Modell**

V-Modell: Einbettung der QS in das Wasserfallmodell

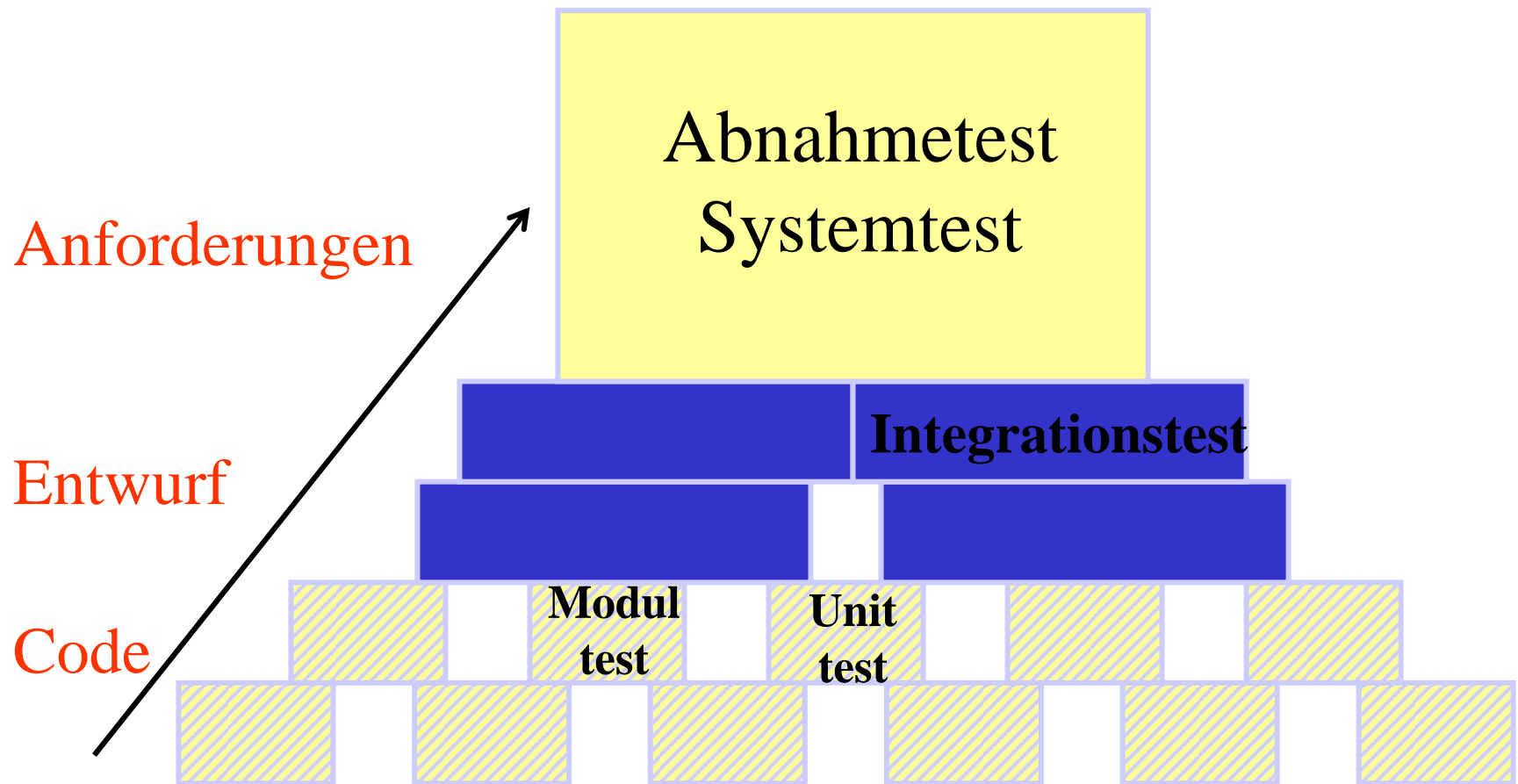


Arten von Tests

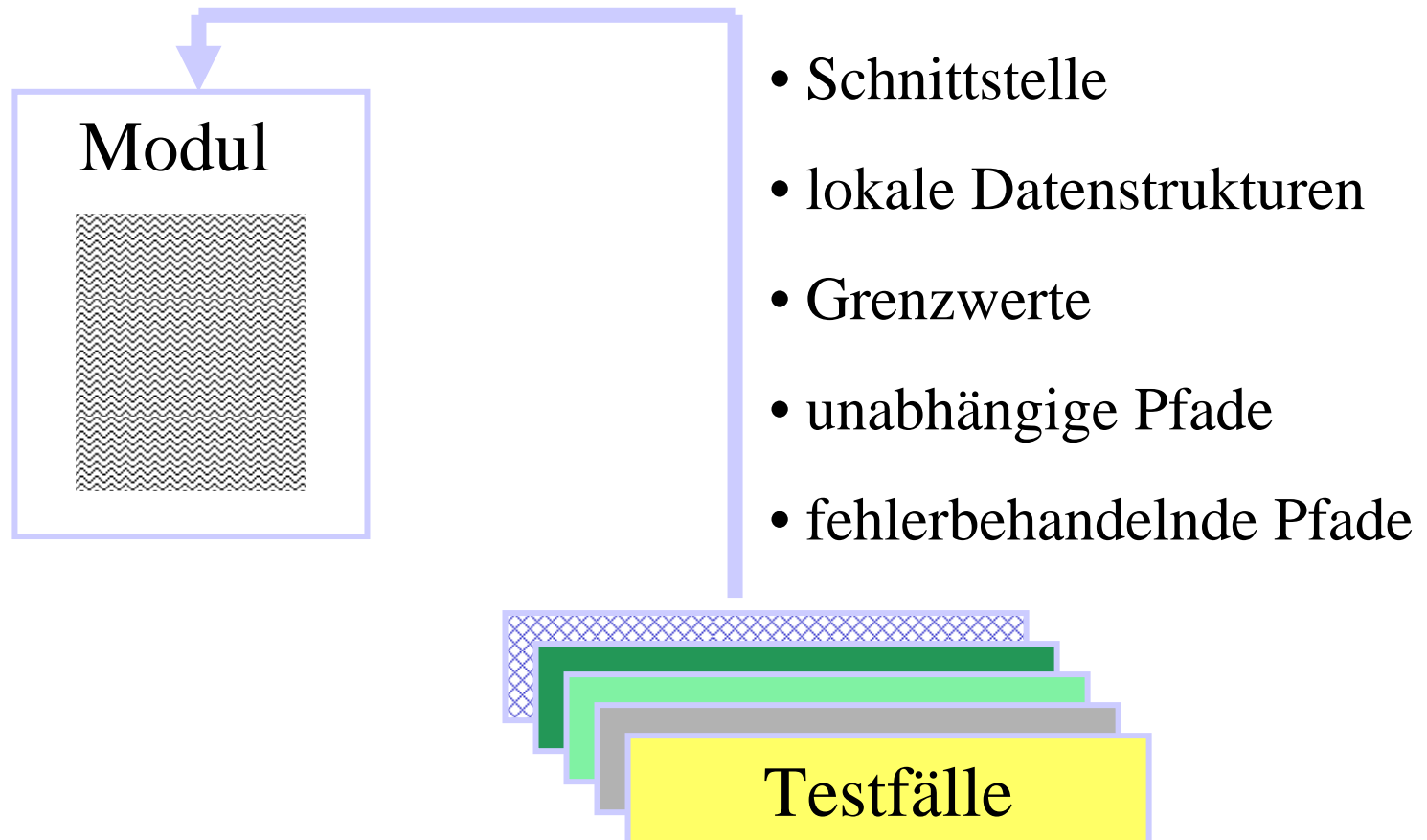
Man unterscheidet beim Testen u.a.

<i>Unit-Test</i>	Test der einzelnen Methoden einer Klasse
<i>Modul-Test</i>	Test einer Menge von Klassen mit einer bestimmten Aufgabe
<i>Integrationstest</i>	Test der Integration mehrerer Module
<i>Systemtest</i>	Test des Gesamtsystems (im Labor)
<i>Abnahmetest</i>	Test des Gesamtsystems (durch den Benutzer)
<i>Feldtest</i>	Test während des Einsatzes

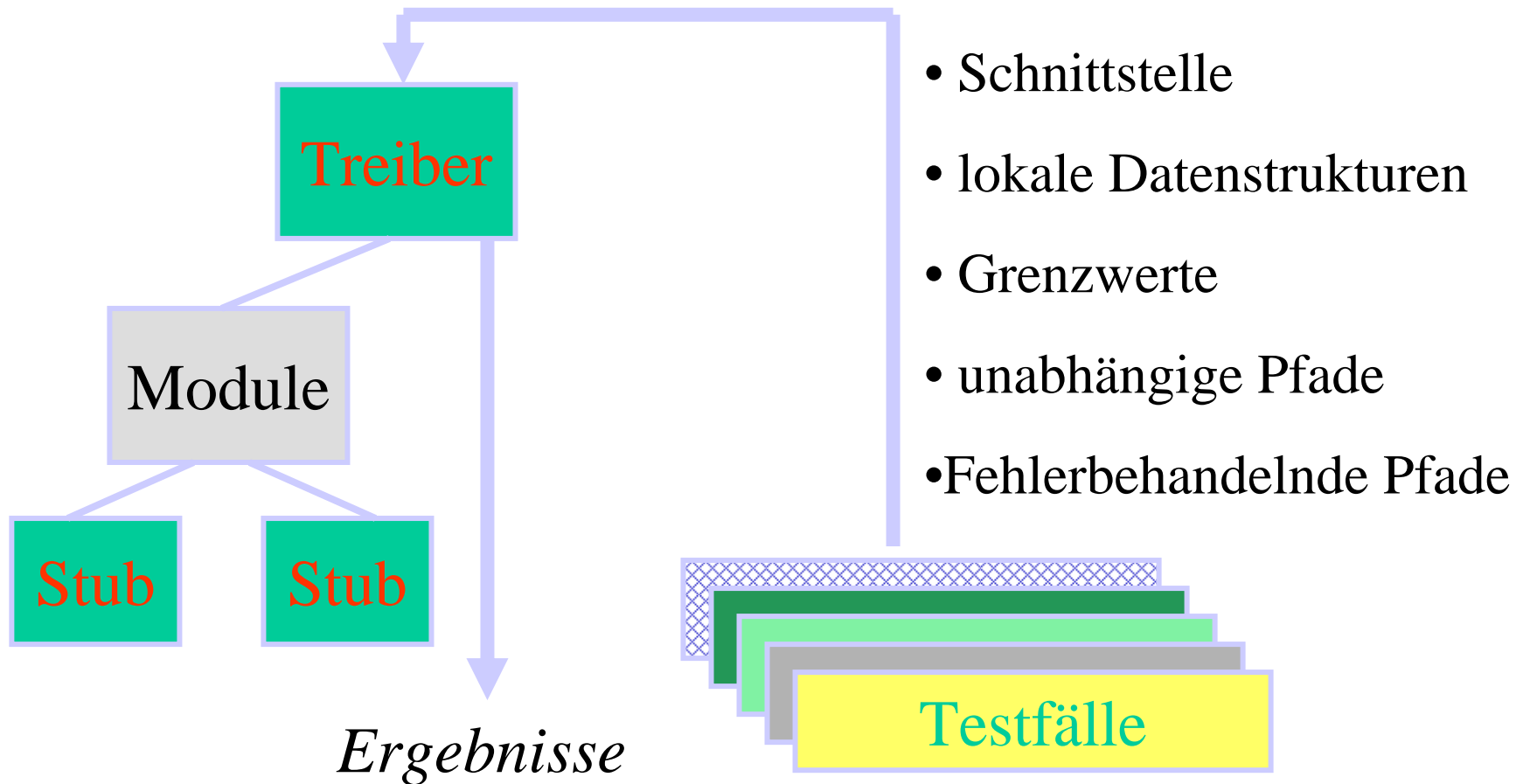
Teststrategie: Hierarchie



Modultest/Unittest



Modultest-Umgebung

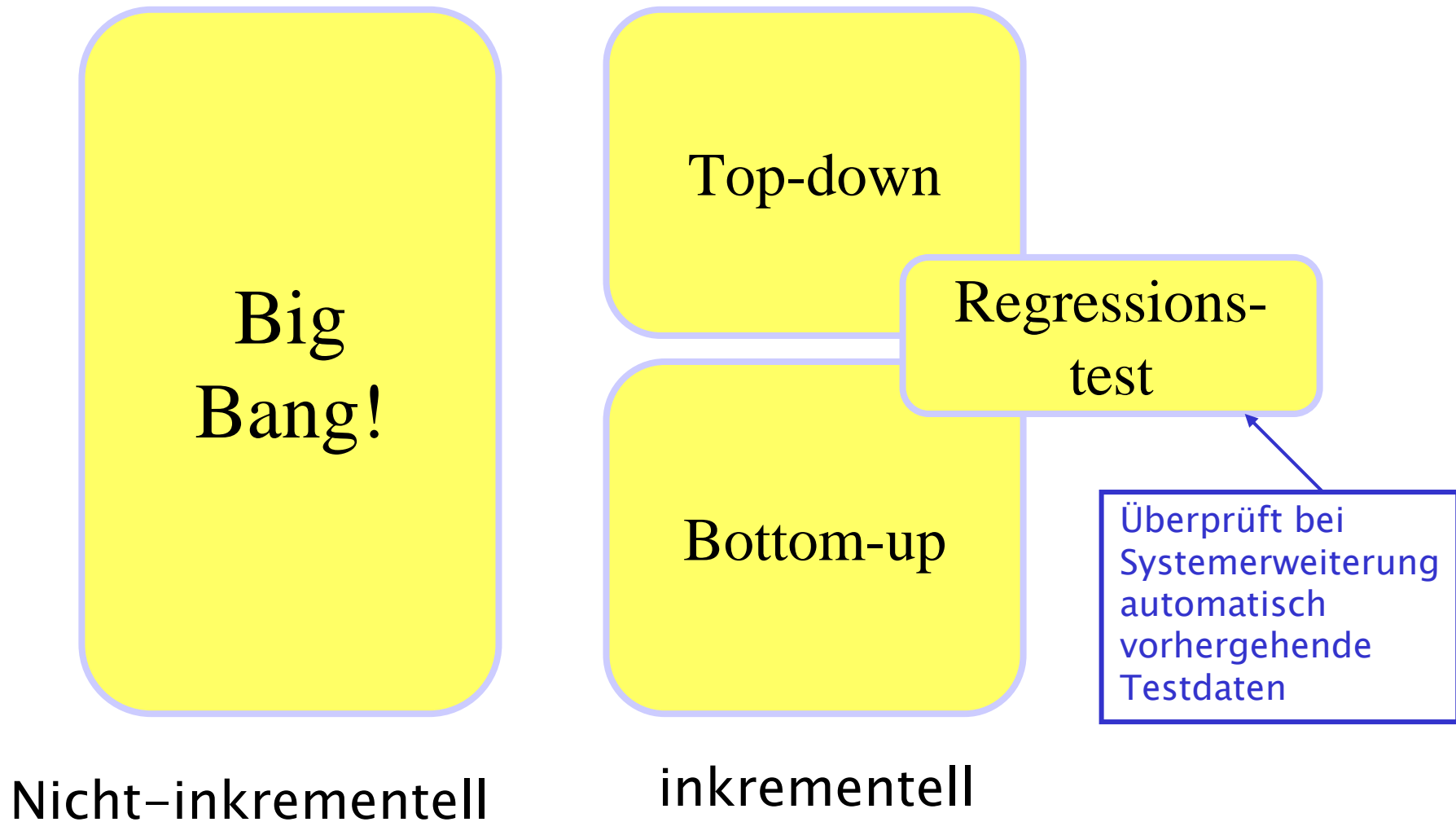


Warum Integrationstest?

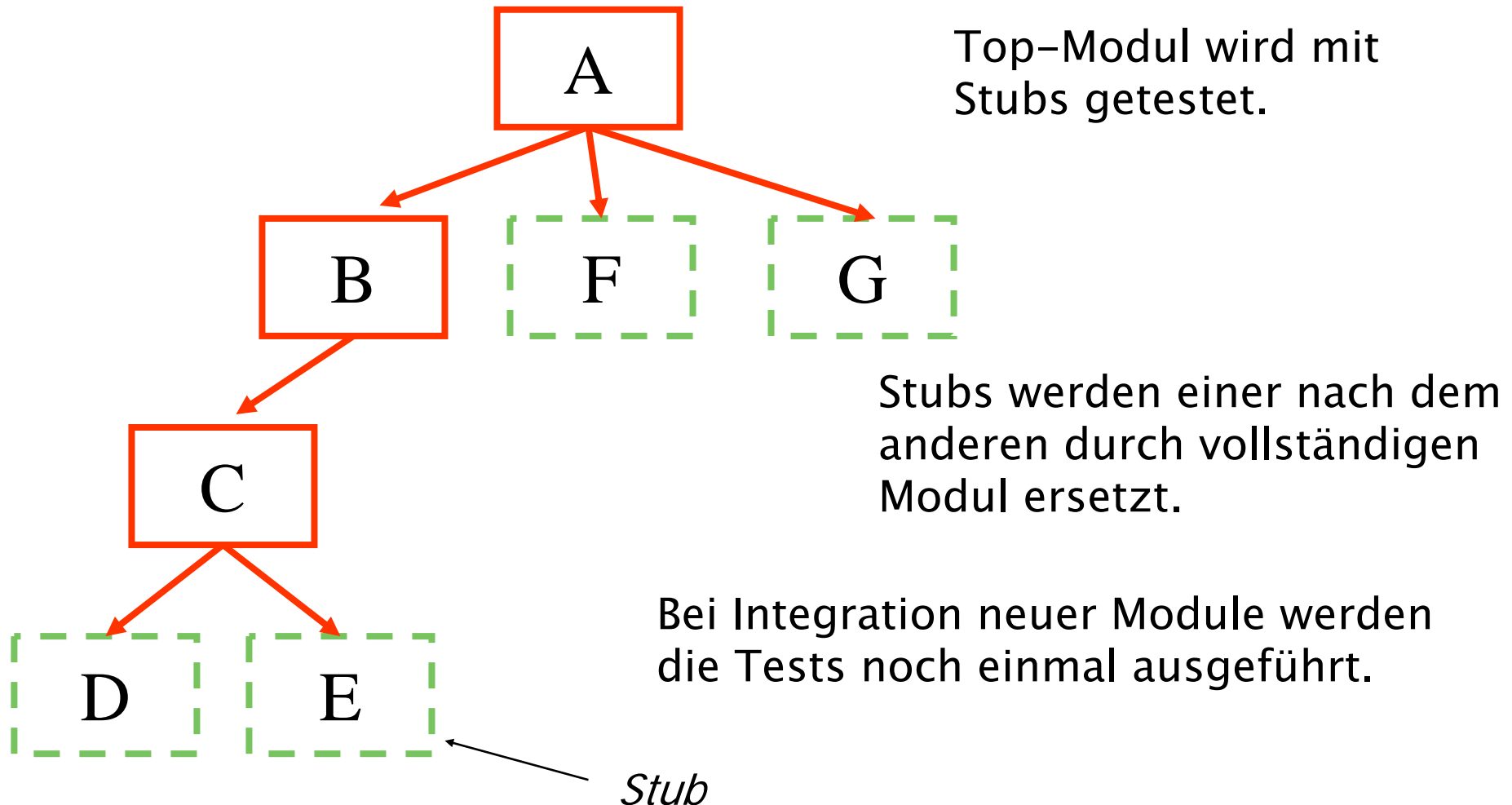
- Wenn alle Module korrekt sind, warum sollen sie dann nicht zusammenpassen?
- Was ist mit
 - undokumentierten Seiteneffekten,
 - unterschiedlicher Interpretation von Operationen,
 - Betriebssystemfeatures , wie z.B. Speicherverbrauch,
 - Einführung von Nebenläufigkeit (race conditions),
 - Verzögerungen der Kommunikation?

☞ Einheiten/Module zu integrieren erfordert Schnittstellen und deren Überprüfung

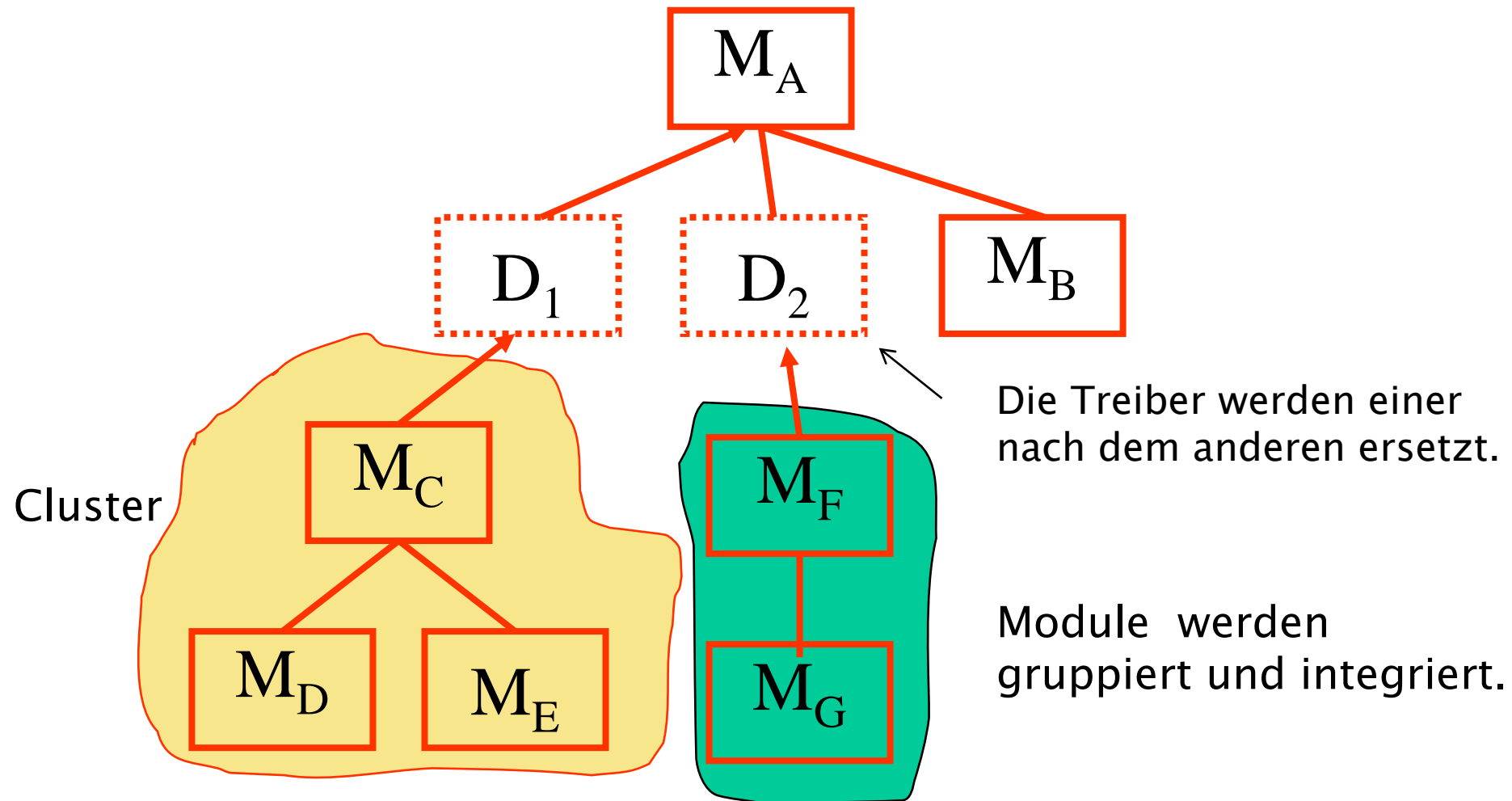
Integrationstest: Vorgehensweisen



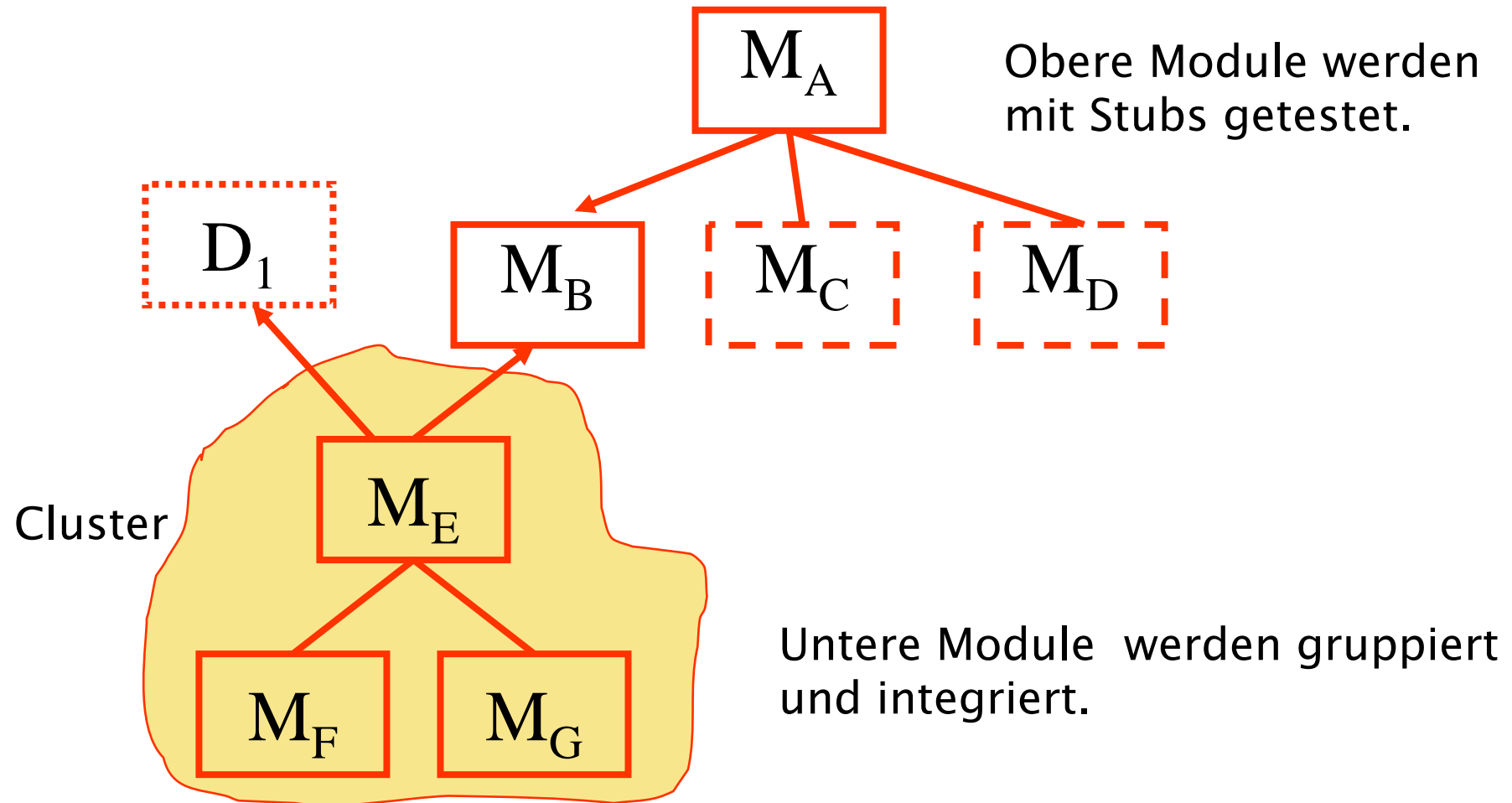
Top-down-Integration



Bottom-up-Integration



Sandwich- Test

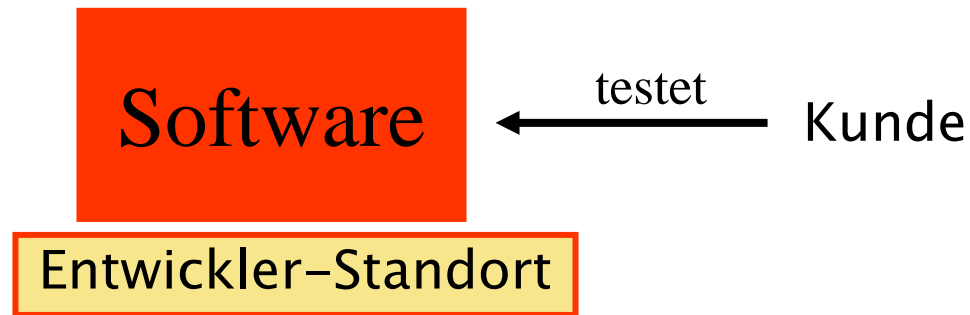


Höhere Tests

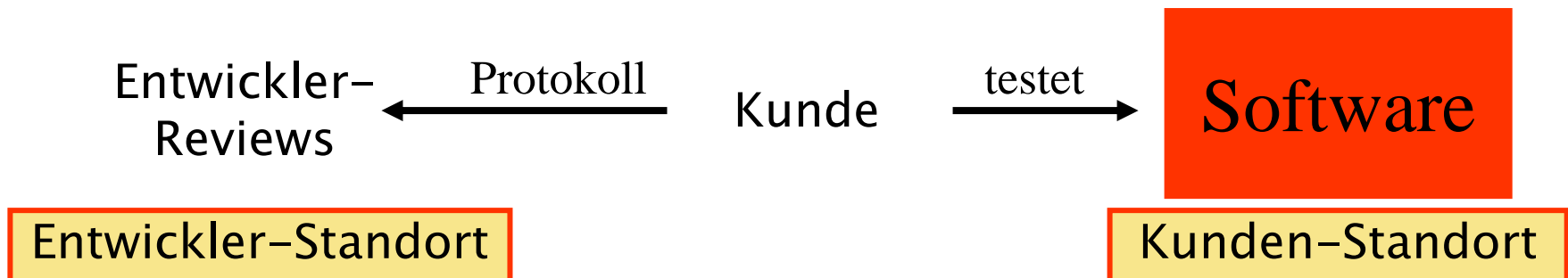
- **Abnahmetest (Validierungstest)**
 - Alpha- und Beta-Test
- **Systemtest**
- **Andere spezialisierte Testarten**
 - Performanztest
 - Sicherheitstest
 - ...

Alpha & Beta-Test

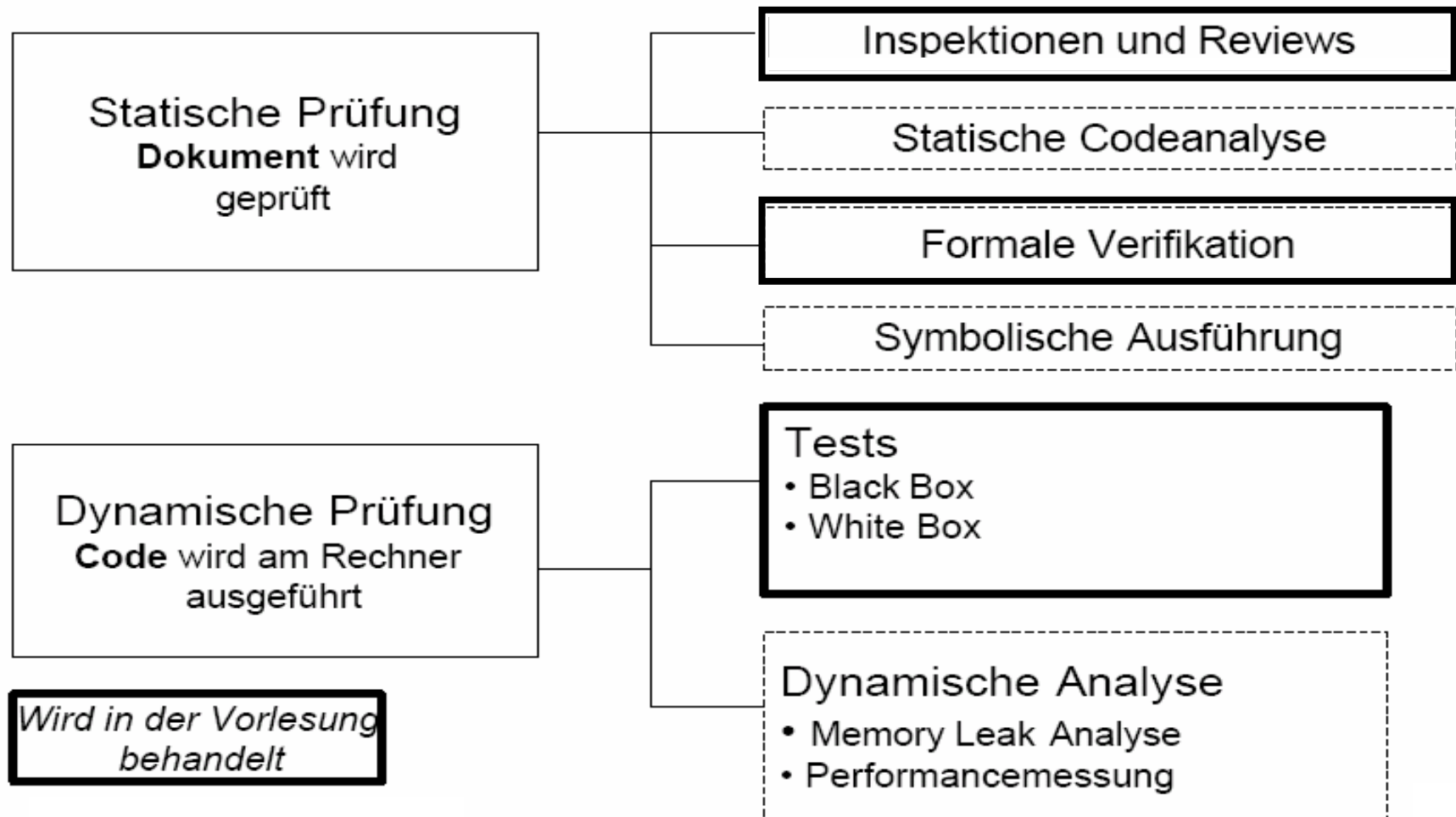
Alpha-Test



Beta-Test



Verfahren der analytischen Qualitätssicherung



Inspektionen und Reviews

- **Prüfmethode der Inspektion (Wiederholung)**
 - entwickelt von M.E. Fagan bei der IBM (1976)
 - formalisierte Evaluationstechnik zur Überprüfung von
 - Softwareanforderungen
 - Entwurf
 - Code
 - Überprüfung erfolgt durch eine Gruppe mit Moderator
 - der Moderator ist kein Vorgesetzter der Autoren!
- **Ziele:**
 - Aufdecken von Fehlern und Verletzungen von Standards im Dokument
 - Entscheidung über Freigabe

Inspektionen: Empirische Ergebnisse

- Es gibt zahlreiche Veröffentlichungen über empirische Ergebnisse von Software-Inspektionen
- Folgende Faustregel wird darin bestätigt
 - 50 bis 75% aller Entwurfsfehler können durch Inspektionen gefunden werden
 - Code-Inspektionen sind ein sehr kosteneffektiver Weg, um Defekte auszugleichen
- Beispiel: Studie bei der NASA Aufdecken von Fehlern pro Std. Aufwand
 - durch Code-Reading: 3,3 Fehler
 - durch Test: 1,8 Fehler

Dynamische Prüfverfahren: Testen

- **Testen ist der Prozess, ein Programm auf systematische Art und Weise auszuführen, um Fehler zu finden. [Glen Myers]**
- **Ein Fehler ist**
 - eine Abweichung zwischen Ist-Verhalten (im Testlauf festgestellt) und Soll-Verhalten (in der Spezifikation gefordert) oder ein
 - ein nicht erfülltes, vom Kunden vorausgesetztes Qualitätskriterium.
- **Testen ist destruktiv:**

Es zeigt nicht die Korrektheit eines Programms, sondern seine Inkorrektheit

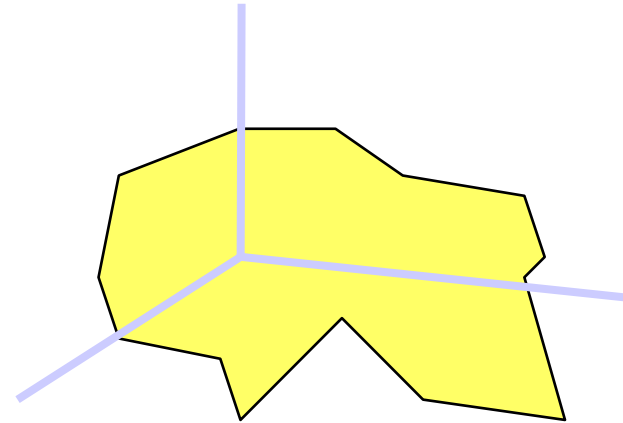
Testen

- Ein **Testdatum** ist ein Satz von Eingabe- und Zustandswerten für ein Softwareobjekt.
- Ein **Testfall** ist eine allgemeine Beschreibung eines Testdatums oder einer Folge von Testdaten zusammen mit zugehörigen Ausgabe-Sollwerten für ein zu testendes Software-Objekt .
- Eine **Testsuite** ist eine Menge von Testfällen.
- Testfälle sollten auf systematische Weise gewählt werden, um eine gewisse Wahrscheinlichkeit zu bieten, Fehler aufzudecken.
- Nach der Methode der Testfall-Ermittlung unterscheidet man
 - Black-Box-Testen
 - White-Box-Testen
 - Glass-Box-Testen (Kombination aus White- und Black-Box-Testen)

Testfallentwurf

“Bugs lurk in corners
and congregate at
boundaries...”

Boris Beizer



Ziel

Fehler zu finden

Kriterien

in einer **vollständigen** Weise

Einschränkung

mit einem **Minimum** an
Aufwand und Zeit

“Adäquate” Menge von Testfällen

- Test der Schnittstellen der Komponenten
 - alle öffentlichen Methoden
- Ist ein Testfall pro Methode genug?
Im allgemeinen nicht!
- Wieviele Testfälle braucht man?
Das hängt von der Methode ab!

Weitere Begriffe zum Testen

Definitionen (1)

- P (Programm), D (Definitionsbereich), R (Wertebereich)
 - $P: D \rightarrow R$ (möglicherweise partiell)
- Korrektheit definiert durch eine Spezifikation $SP \subseteq D \times R$
 - P(d) **korrekt**, falls $\langle d, P(d) \rangle \in SP$
 - P **korrekt**, falls alle P(d) korrekt sind.

Definitionen (2)

- **Failure (Fehlverhalten)**
 - $P(d)$ ist nicht korrekt
 - kann undefiniert sein (Fehlerzustand) oder das falsche Resultat
- **Error (Defect, Fehler)**
 - Alles, was ein Fehlverhalten erzeugen kann
 - Tippfehler
 - Vergessen einer Bedingung
 - Falsche Interpretation einer API-Methode”
- **Fault (Fehler)**
 - Inkorrekter Zwischenzustand eines Programms

Definitionen (3)

- **Testdatum** d ist ein Element von D
- **Testfall** t ist ein Paar $\langle d, r \rangle$ aus SP
- **Testmenge** T ist eine endliche Untermenge von D
- **Testsuite** S ist eine endliche Teilmenge von SP ; d.h.
eine Testmenge mit ihren Resultaten.
- Ein **Testdatum** d ist **erfolgreich (für P , SP)**, wenn $P(d)$ korrekt ist.
- Eine **Testmenge** T ist **erfolgreich (für P , SP)**, wenn P für alle d in T korrekt ist.
- Ein **Testfall**/eine **Testsuite ist erfolgreich** (für P), wenn ihre Testdaten/Testmenge erfolgreich sind für (für P , SP).
- **Ideale Testsuite** T
 - Wenn P nicht korrekt ist, dann gibt es ein Element d von T so, dass $P(d)$ nicht korrekt ist.
- *Wenn für ein Programm eine ideale Testsuite existieren würde, könnten wir die Korrektheit des Programm durch Testen beweisen.*

Bemerkung: Das ist nur möglich bei Programmen über endlichen Zustandsmengen, siehe Modelchecking

Testkriterium

- Ein Kriterium C definiert eine Menge endlicher Untermengen von D (Testmenge)
 - $C \subseteq 2^D$
- Eine Testmenge T erfüllt C , wenn sie ein Element von C ist.

Beispiel

$C = \{ \{x_1, x_2, \dots, x_n\} \mid n \geq 3 \wedge \exists i, j, k, (x_i < 0 \wedge x_j = 0 \wedge x_k > 0) \}$

$\{-5, 0, 22\}$ ist eine Testmenge, die T erfüllt

$\{-10, 2, 8, 33, 0, -19\}$ erfüllt T

$\{1, 3, 99\}$ erfüllt T nicht.

Eigenschaften von Kriterien

- **C ist konsistent**
 - Für alle T1, T2, die C erfüllen,
ist T1 erfolgreich gdw T2 erfolgreich ist
 - Beide erbringen die “gleiche” Information
- **C ist vollständig**
 - Wenn P nicht korrekt ist, gibt es eine Testmenge von C, die nicht erfolgreich ist.
- **C ist vollständig und konsistent**
 - Identifiziert eine ideale Testmenge
 - Erlaubt, dass Korrektheit bewiesen werden kann!

Bemerkung: Keine dieser Eigenschaften ist entscheidbar.

Es gibt keinen Algorithmus zum Beweis der Programmkorrektheit.

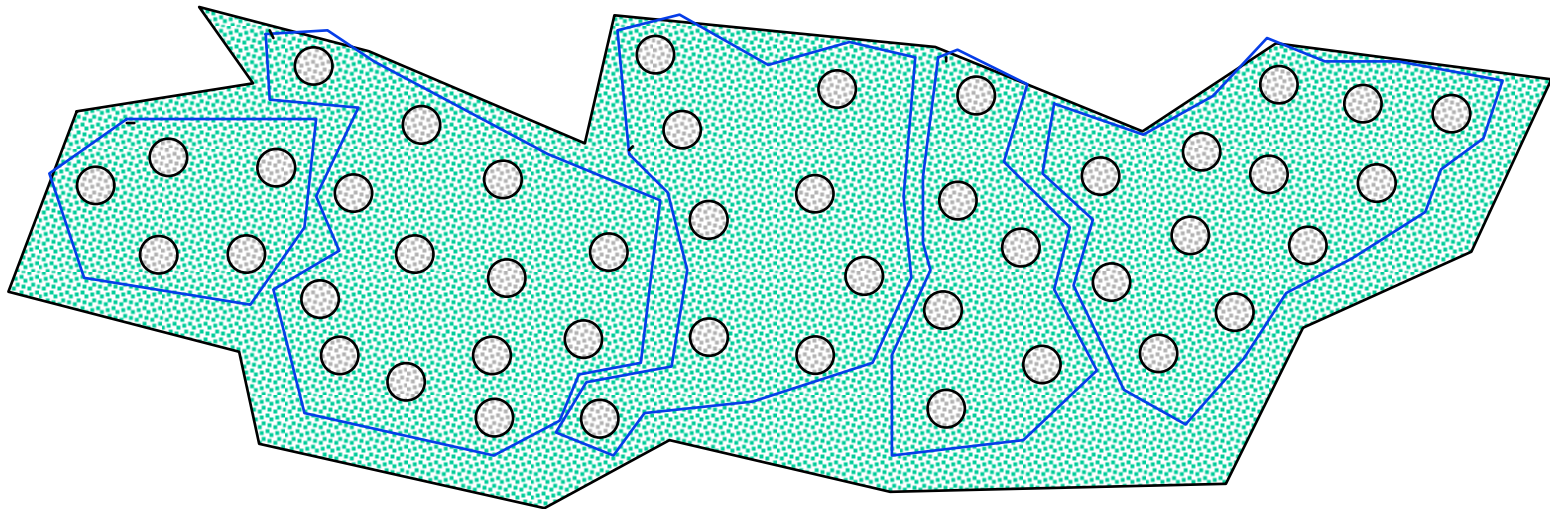
Empirische Testprinzipien

Empirische Testprinzipien

- bilden einen Kompromiss zwischen Unmöglichkeit und Inadäquatheit
- Definieren Strategien um signifikante Testfälle auszuwählen
 - signifikant=großes Potenzial zur Fehlerentdeckung

Prinzip der vollständigen Abdeckung

- Die Elemente von D in Teilbereiche D_1, D_2, \dots, D_n gruppieren, so dass alle Elemente von D_i ein ähnliches Verhalten aufweisen:
 - $D = D_1 \cup D_2 \cup \dots \cup D_n$
- Für jeden Teilbereich einen Repräsentanten als Testfall wählen.
- Beispiel einer Partition:



Zusammenfassung

- **Qualitätsmanagement** umfasst alle Tätigkeiten, um die Qualität von Prozessen und Produkten sicherzustellen.
- Die Qualitätssicherung dient zur Umsetzung der Maßnahmen des QS-Plans und umfasst **konstruktive QS-Maßnahmen** wie Methoden, Sprachen, Werkzeuge, und **analytische QS-Maßnahmen** wie Verifikation und Testen.
- Beim Testen unterscheidet man zwischen **Unit-Test/ Modul-Test, Integrationstest, Systemtest, Abnahmetest** und **Feldtest**.
- Grundbegriffe beim Testen sind **Testfall, Testsuite** und ihre **Konsistenz und Vollständigkeit**.

Literatur

- **Helmut Balzert: Lehrbuch der Software-Technik, Band 2, Spektrum Akademischer Verlag, 1998.**
- **Jens Coldewey: Reviews reviewed, Objekt Spektrum, Aug.-Okt. 2000**
- **C. Ghezzi et al.: Software Engineering, 2nd ed.. Prentice Hall 2003, Kap. 6**
- **Peter Liggesmeyer: Software-Qualität. Akad.Verlag, 2002**
- **Andreas Mieth: Qualitätsmanagement, in: P. Brössler, J. Siedersleben (Hrg): Softwaretechnik, Praxiswissen für Software-Ingenieure, Hanser 2000**
- **Ernest Wallmüller: Software-Qualitätssicherung in der Praxis, Hanser 1990**
- **Internet-Seiten mit Testbegriffen: Fachgruppe 2.1.7 der Gesellschaft für Informatik (GI) <http://www.fbe.hs-bremen.de/spillner/begriffe/home.html>**