

Software- Qualitätsmanagement

**Kernfach Angewandte Informatik und
Vorlesung im Modul 10-202-2319
Software-Management**

Sommersemester 2008

apl. Prof. Dr. Hans-Gert Gräbe

<http://bis.informatik.uni-leipzig.de/HansGertGraebe>

1. Allgemeines

- **Durchsprache**

- geringer personeller und organisatorischer Aufwand (Autor/Gutachter)
- Analyse von Dokumenten in einem frühen Entwicklungsstadium
- **Ziel:** Aufdecken von Defekten und Problemen im Ansatz, indem andere „mal draufschauen“.

- **Review**

- größerer personeller Aufwand (Moderator/Autor/mehrere Gutachter)
- Analyse von Dokumenten in einem fortgeschrittenen Entwicklungsstadium
- **Ziel:** Aufdecken von Defekten und Problemen durch genauere Betrachtung unter verschiedenen Aspekten

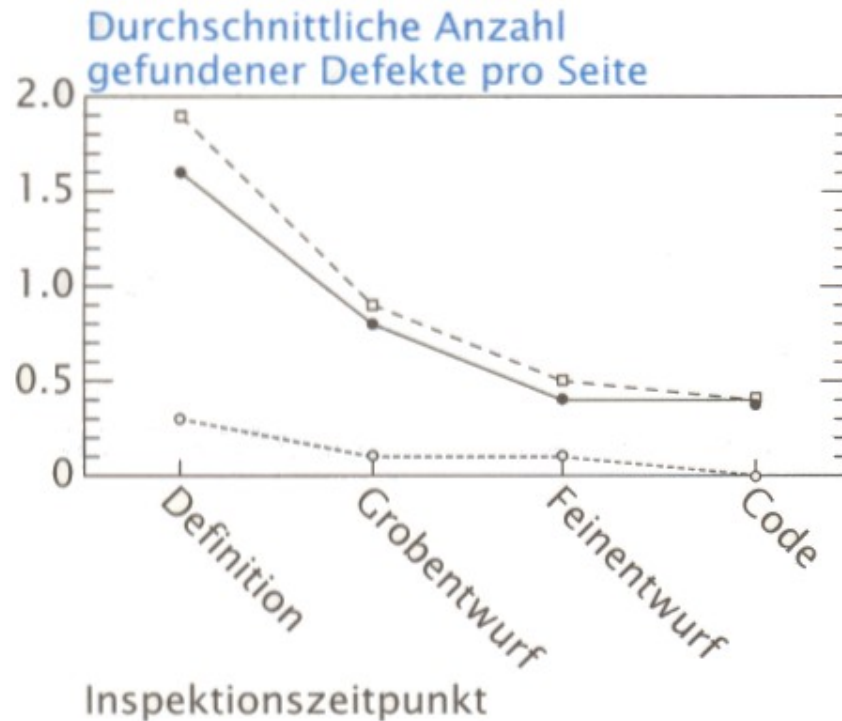
- **Inspektion**

- größerer personeller und organisatorischer Aufwand
- Analyse von Dokumenten in finalem Zustand
- **Ziel:** Freigabe von Teilprodukten für die nächste Entwicklungsaktivität

Beispiele für generische Freigabekriterien

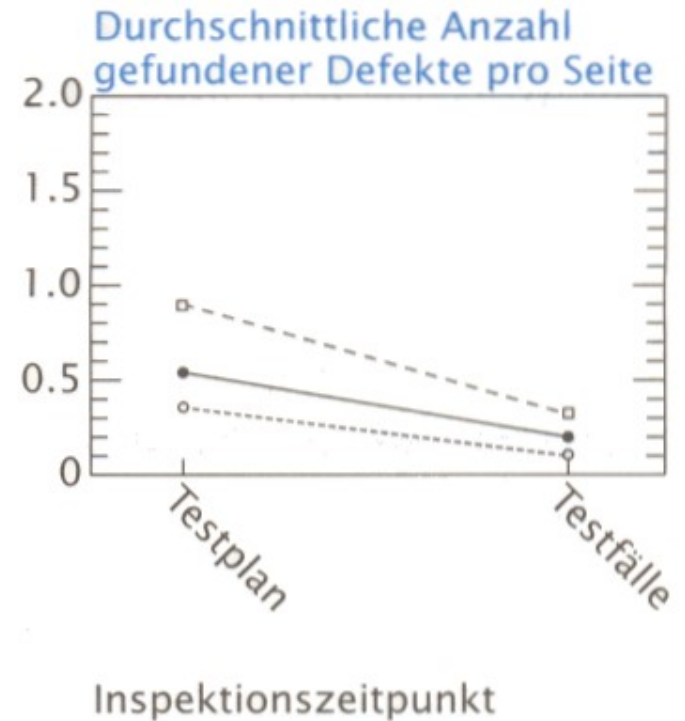
- Alle Überarbeitungen sind vollständig und sorgfältig durchgeführt.
- Alle notwendigen Änderungsanträge wurden gestellt.
- Die Datensammlung mit den Inspektionsmetriken ist vollständig und in der Datenbank erfasst.
- Restdefektrate ist kleiner als 0,25 schwere Defekte pro Seite (2 bis 3 für Anfänger).
- Die individuelle Prüfgeschwindigkeit (Seiten pro Stunde) und die Prüfgeschwindigkeit der Inspektionssitzung haben die bekannte optimale Prüfgeschwindigkeit im Durchschnitt um nicht mehr als 20% überschritten (sonst werden zu viele Defekte übersehen).
- Weder der Autor noch der Moderator haben ein Veto gegen die Freigabe eingelegt. Sie können dies tun, wenn sie subjektiv glauben, dass das freizugebende Prüfobjekt nicht nutzungstauglich ist.

Empirische Ergebnisse



Legende: schwere Defekte leichte Defekte alle Defekte

-----○----- —●— - - - □ - - -



Testplan = Testplandokument
Testfälle = dokumentierte Testfälle

- Faustregeln [Grady 92]
 - 50 bis 75% aller Entwurfsfehler können durch Inspektionen gefunden werden.
 - Code-Inspektionen sind ein sehr kosteneffektiver Weg, um Defekte aufzudecken.
- Die Investitionseffizienz (Verhältnis der ersparten Ingenieurstunden zu den Kosten) ist viel besser als für viele andere Investitionen.
 - Kosten für Training: 6 Personen * 8 Std. = 48 Std.
 - Kosten für Inspektion: 6 Personen * 16 Std. = 96 Std.
 - Eingesparte Kosten für spätere Defektbeseitigung: 1700 Std.
 - entspricht einer Investitionseffizienz von $1700/144 \approx 11.8$
- Weitere Vorteile:
 - Kürzere Entwicklungszeit (1700 Std. \approx 1.8 Monate)
 - Geringeres Risiko
 - Die Trainings- und Einführungskosten fallen pro Team nur einmal an.
- Frühe Inspektionen sind effizienter als spätere

Review

- Manuelle Prüfmethode, mit welcher Stärken und Schwächen eines schriftlichen Dokuments in Bezug auf Referenzunterlagen nach individueller Vorbereitung der Gutachter in einer Teamsitzung identifiziert werden, um diese durch den Autor beheben zu lassen.
 - Weniger stark formalisiert als Inspektion.
- **Ziel:** Feststellung von Mängeln, Fehlern, Inkonsistenzen, Unvollständigkeiten sowie Verstößen gegen Vorgaben und Richtlinien
- **Dokumente:** Prüfobjekt, Referenzdokumente
 - Dokumente: max. 50 Seiten, 5 Gutachter, 10 Seiten/Std.
 - Code: max. 20 Seiten, 3 Gutachter, 5 Seiten/Std.
- **Rollen:** Moderator, Autor, Protokollführer, 2-5 Gutachter

- **Vorgehen:** Beantragung, Eingangsprüfung, optionale Einführungs-Sitzung, individuelle Vorbereitung, Review-Sitzung, Überarbeitung, Bestätigung
 - Ablauf ist im Wesentlichen wie bei einer Inspektion
 - Eingangsprüfung wird vom Manager u. U. zusammen mit dem Moderator durchgeführt
- **Aufwand:** 15% (Code) – 20% (Dokumente) des Aufwands für die Erstellung des Prüfobjekts
- **Nutzen:** 60 – 70% der Fehler werden gefunden.
 - Reduktion der Fehlerkosten in der SE um 75% und mehr
 - Nettoeinsparungen in der Entwicklung um ca. 20%, in der Wartung um ca. 30%.

Durchsprache (Walkthrough)

- Manuelle informale Prüfmethode, um in einer Teamsitzung Fehler, Defekte, Unklarheiten und Probleme in schriftlichen Dokumenten zu identifizieren und durch den Autor beheben zu lassen.
 - Noch weniger stark formalisiert als Review.
- **Ziele:**
 - Identifikation von Defekten
 - Vermittlung von Inhalten (Ausbildung der Mitarbeiter)
- **Dokumente:** Prüfobjekt oder Teilprodukte
- **Rollen:** Autor, Gutachter
- **Vorgehen:** Gruppensitzung unter Leitung des Autors nach (optionaler) individueller Vorbereitung
- **Aufwand:** relativ gering
- **Nutzen:** Auf Grund des informellen Charakters schwer messbar

Vorteile

- geringer Aufwand
- auch für kleine Entwicklungsteams geeignet
- sinnvoll für „unkritische“ Dokumente bzw. Dokumente in frühen Entwicklungsphasen
- Durch Einbeziehung von Kunden/Nutzern als Gutachter können Unvollständigkeiten und Missverständnisse aufgedeckt werden
- gut geeignet, um das Wissen über ein Dokument auf eine breite Basis zu stellen.

Nachteile

- Es werden wenig konkrete Defekte identifiziert.
- Autor kann die Durchsprache dominieren
- Überarbeitung des Prüfobjekts liegt im Ermessen des Autors und wird nicht nachgeprüft.

Andere manuelle Prüfmethode

- **Stellungnahme:** Der Autor bittet ein oder mehrere Kollegen um Kommentare zu einem Prüfobjekt. Der Autor gibt den Kollegen Kopien des Prüfobjekts und erhält diese mit Kommentaren zurück.
- **Round-Robin-Review:** Ziel ist es, Argumente für die Güte des Prüflings zu sammeln. Jeder Gutachter versucht in der *Review*-Sitzung die anderen davon zu überzeugen, dass die Qualität des Prüfobjekts akzeptabel ist.
- **Peer-Review:** Gutachter werden in einem Raum „eingeschlossen“, untersuchen ein oder mehrere Prüfobjekte, und liefern Gutachten dazu. Das *Review*-Team bestimmt selbst die Aufgabenverteilung und die Vorgehensweise.

Zusammenfassung

Manuelle Prüfmethoden dienen dazu, Produkt- oder Prozesseigenschaften zu überprüfen, welche durch automatische Werkzeuge nicht oder nur unzureichend festgestellt werden können.

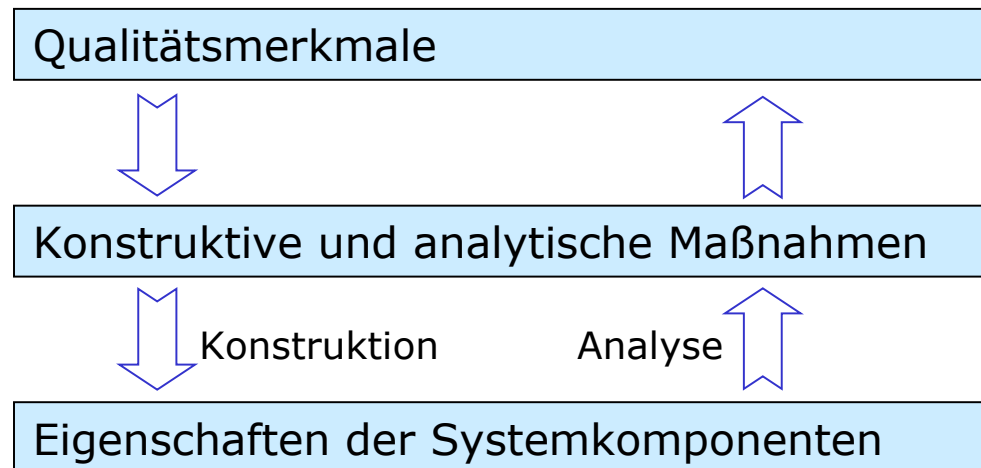
Die Effektivität hängt vor allem von folgenden Punkten ab

- Gutachter konzentrieren sich auf einzelne Aspekte
- Gutachter bereiten sich individuell und schriftlich vor
- In einer Team-Sitzung werden moderiert die Ergebnisse zusammengetragen und weiter analysiert. Lösungen werden dabei nicht diskutiert.
- Der Prüfaufwand ist in der Projektplanung berücksichtigt.

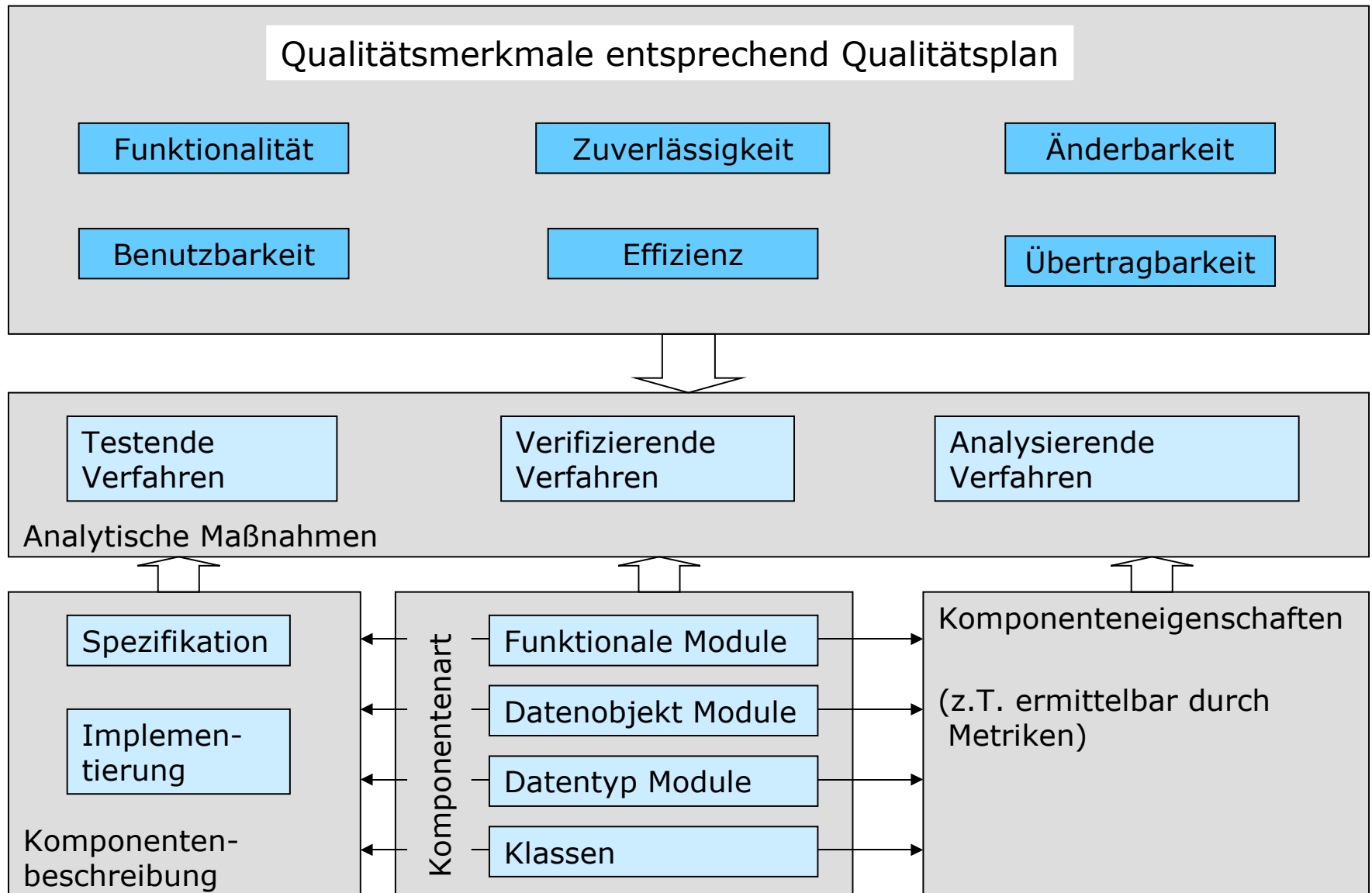
Der Aufwand (Review oder Inspektion) liegt bei 15-20% des Erstellungsaufwands für das entsprechende Produkt oder Dokument.

Software-Produktqualität ist abhängig von:

- Qualität der Systemkomponenten
- Qualität der Beziehungen zwischen den Komponenten



- Konstruktive Maßnahmen siehe VL „Software-Technik“
- analytische Maßnahmen beziehen sich im Wesentlichen auf Funktionalität, Zuverlässigkeit und evtl. Änderbarkeit



Fehler als zentraler Begriff

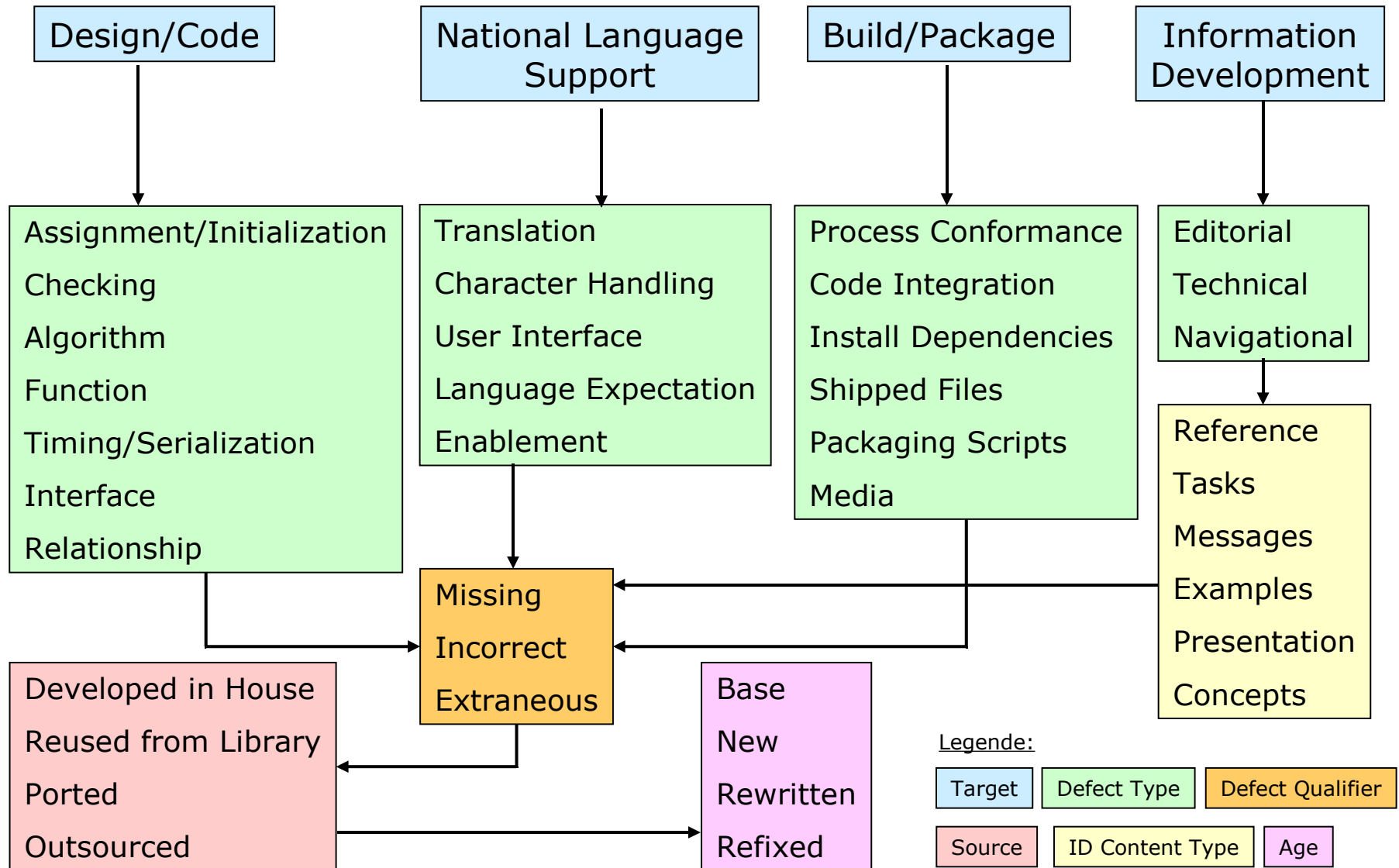
- Konstruktives Ziel: Entwicklung fehlerfreier Software-Komponenten
- Analytisches Ziel: Nachweis der Fehlerfreiheit von Software-Komponenten

Als **Fehler** wird

- jede Abweichung der tatsächlichen Ausprägung eines Qualitätsmerkmals von der vorgesehenen Soll-Ausprägung,
- jede Inkonsistenz zwischen der Spezifikation und der Implementierung und
- jedes strukturelle Merkmal des Programmtexts, das ein fehlerhaftes Verhalten des Programms verursacht, bezeichnet.

4. Produktqualität

2. Was sind Fehler?



Analyseverfahren

Die Art und die Eigenschaften der Systemkomponenten bestimmen die Auswahl geeigneter analytischer Maßnahmen:

- Funktionale Module
- Datenobjekt-Module
- Datentyp-Module
- Klassen

Unterschied der eingesetzten analytischen Maßnahmen insbesondere zwischen Komponenten mit komplexen Kontrollstrukturen und Komponenten mit komplexen Datenstrukturen.

Testende Verfahren haben das Ziel, Fehler zu erkennen

- Dynamische Testverfahren
- Statische Verfahren

Verifizierende Verfahren sollen die Korrektheit einer Komponente beweisen

- Verifikation
- Symbolische Ausführung

Analysierende Verfahren sollen Eigenschaften von Komponenten darstellen oder vermessen

- Analyse der Bindungsart
- Metriken
- Grafiken und Tabellen
- Anomalienanalyse

Testende Verfahren - Prinzipieller Zugang

- Das tatsächliche Verhalten wird **stichprobenartig** an Hand einer Menge von **Testfällen** untersucht und die Ergebnisse mit den erwarteten Ergebnissen (Spezifikation, Normen) verglichen und dokumentiert.
- Testfälle werden entsprechend den Testzielen speziell ausgewählt.
- Einsatz um
 - Programmfehler aufzufinden (Bugs)
 - Wiederauftreten von Fehlern zu vermeiden (Regressionstests)
- Fehlerfreiheit ist plausibel, aber nicht garantiert.
- Abzugrenzen von
 - **Verifikation** als strengem Korrektheitsbeweis
 - **Ausprobieren** als einer Entwicklungsmethode (trial and error)

Begriff des Programms

- Programm = schrittweise Transformation einer Menge von Eingabedaten in eine Menge von Ausgabedaten nach einem vorgegebenen Algorithmus
- Black-Box-Betrachtung: $f: X \rightarrow Y$
 - Spezifikation, funktionale Korrektheit
- Transformation = Abarbeiten einzelner Programmschritte, in denen die Daten entsprechend den angegebenen Instruktionen verändert werden.
 - zustandsorientierte Betrachtung: Datenfluss
 - übergangsorientierte Betrachtung: Kontrollfluss
- Programmstatus = Zustand der Gesamtheit der durch das Programm manipulierten Daten
 - Anweisungen und Deklarationen
 - Variablenbegriff als Wertcontainer
 - Sichtbarkeit und Lebensdauer
 - Compilezeit und Laufzeit

Statische und dynamische Testverfahren

Dynamische Testverfahren

- übersetztes und ausführbares Programm wird mit konkreten Eingabewerten ausgeführt
- evtl. Instrumentierung des Programms
- Test in realer Laufzeitumgebung
- Stichprobenverfahren (Testfälle)
- Ziele: Finden von Fehlern (debugging), Finden und Optimieren laufzeitkritischer Bereiche (profiling)
- Korrektheit kann so nicht bewiesen werden
- Klassifikation nach Herkunft der Testfälle

Statische Testverfahren

- Analyse des Quellcodes, evtl. testfallorientiertes Durchgehen
- typische Verfahren: manuelle Prüfmethode

Glossar

Prüfling, Testling oder **Testobjekt**

- das zu testende Programm oder die Komponente (Prüfling als allgemeiner Begriff)

Testverfahren

- grundlegendes Verfahren, mit dem einzelne Eigenschaften eines Testlings durch eine geeignete Anzahl von Testfällen untersucht wird

Testfall

- Satz von Testdaten, der die vollständige Ausführung eines zu testenden Programms bewirkt

Testdatum

- Eingabewert, der einen Eingabeparameter des Testobjekts instanziiert

Testtreiber

- Testrahmen, mit dem eine nicht extern zugängliche Funktion interaktiv aufgerufen werden kann.
- Können durch Testwerkzeuge automatisch generiert werden.

Instrumentierung

- Der Quellcode des Testlings wird für die Analyse der Testfälle mit zusätzlichem Protokollcode versehen oder dieser aktiviert.
- Instrumentierter Testling wird übersetzt. Protokoll enthält Informationen über das Laufzeitverhalten des Testlings während des Abarbeitens der einzelnen Testfälle.

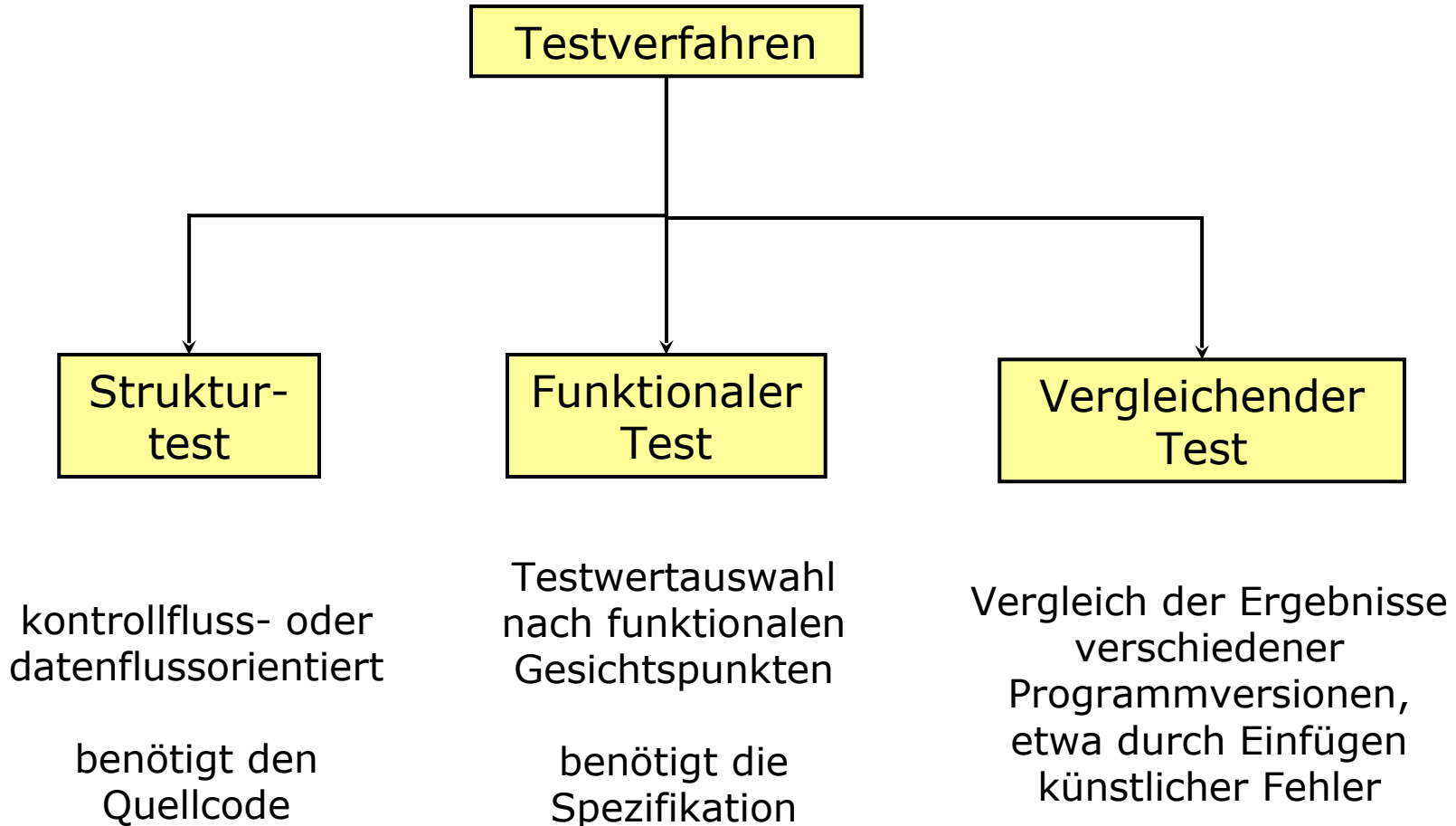
Überdeckungsgrad

- Maß für den Grad der Vollständigkeit eines Tests bezogen auf ein bestimmtes Testverfahren

Regressionstest

- automatische werkzeuggestützte Neuausführung bereits durchlaufener Tests nach Änderungen am Testling.

Klassifikation testender Verfahren



- **Strukturtest**

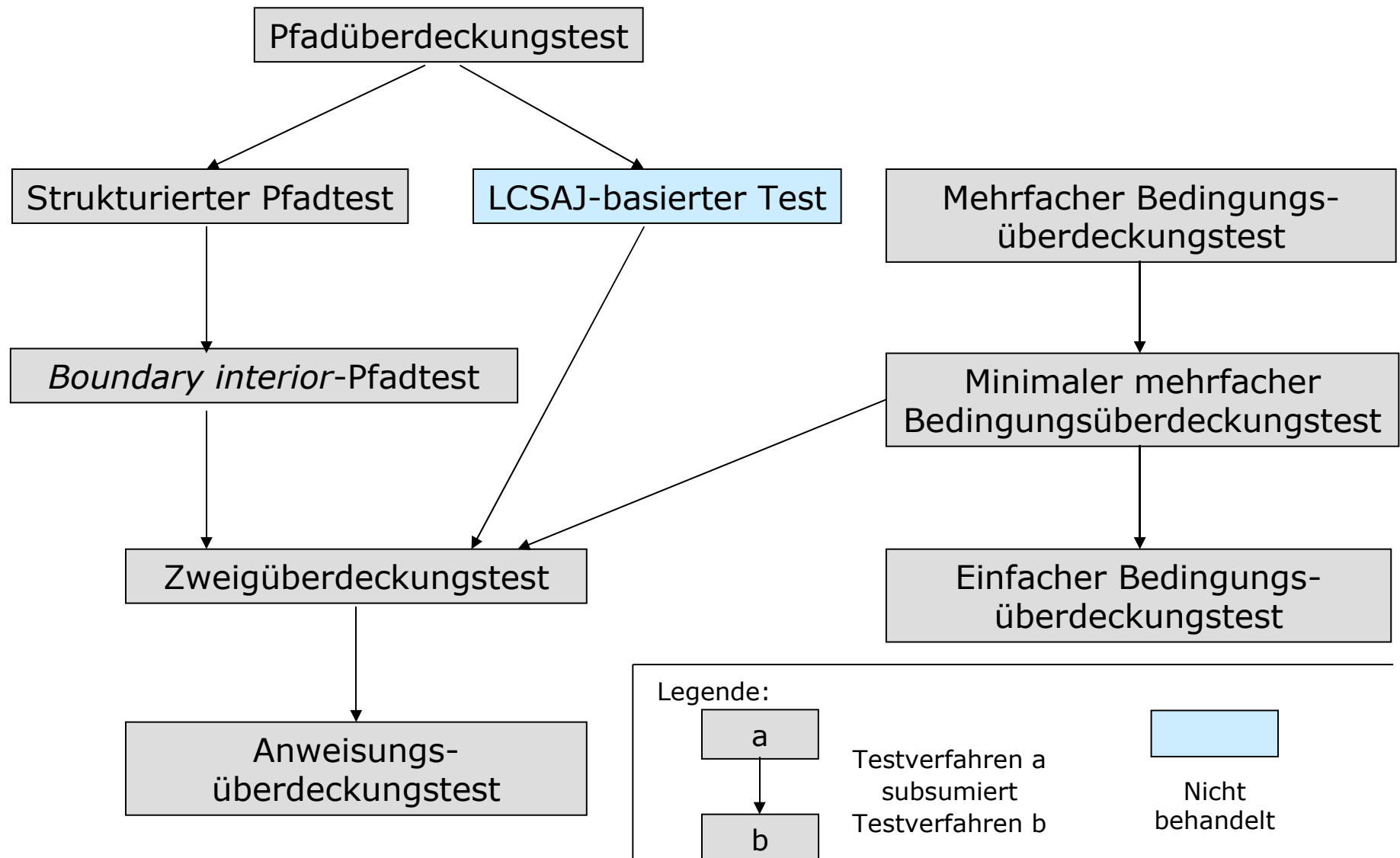
- kontrollflussorientiert (Monitoring des Programmflusses)
 - Anweisungsüberdeckung
 - Zweigüberdeckung
 - Pfadüberdeckung (volle Version kombinatorisch exponentiell!)
 - Bedingungsüberdeckung
- datenflussorientiert (Monitoring der Programmdaten)

- **Funktionaler Test**

- funktionale Äquivalenz
- Grenzwertanalyse
- Test spezieller Werte (Szenarios)
- Zufallstest
- zustandsübergangsgetriebene Tests

Überblick

- Basieren auf der Kontrollstruktur des zu prüfenden Programms
- Gehören zu den **Strukturtest-**, **White Box-** oder **Glass Box-Verfahren**
- **Ziel** ist, mit möglichst wenigen Testfällen alle Anweisungen, Zweige oder Pfade zu durchlaufen
- Sorgfältige Auswahl der Testfälle, um mögliche strukturelle Probleme genau zu überdecken.



Ein Beispiel

/* Funktion: ZaehleZchn

Aufgabe: Die Prozedur ZaehleZchn zählt die Zeichen sowie Vokale in einem String. Dazu werden die Klassenvariablen *Gesamtzahl* und *VokalAnzahl* der Klasse *count* inkrementiert.

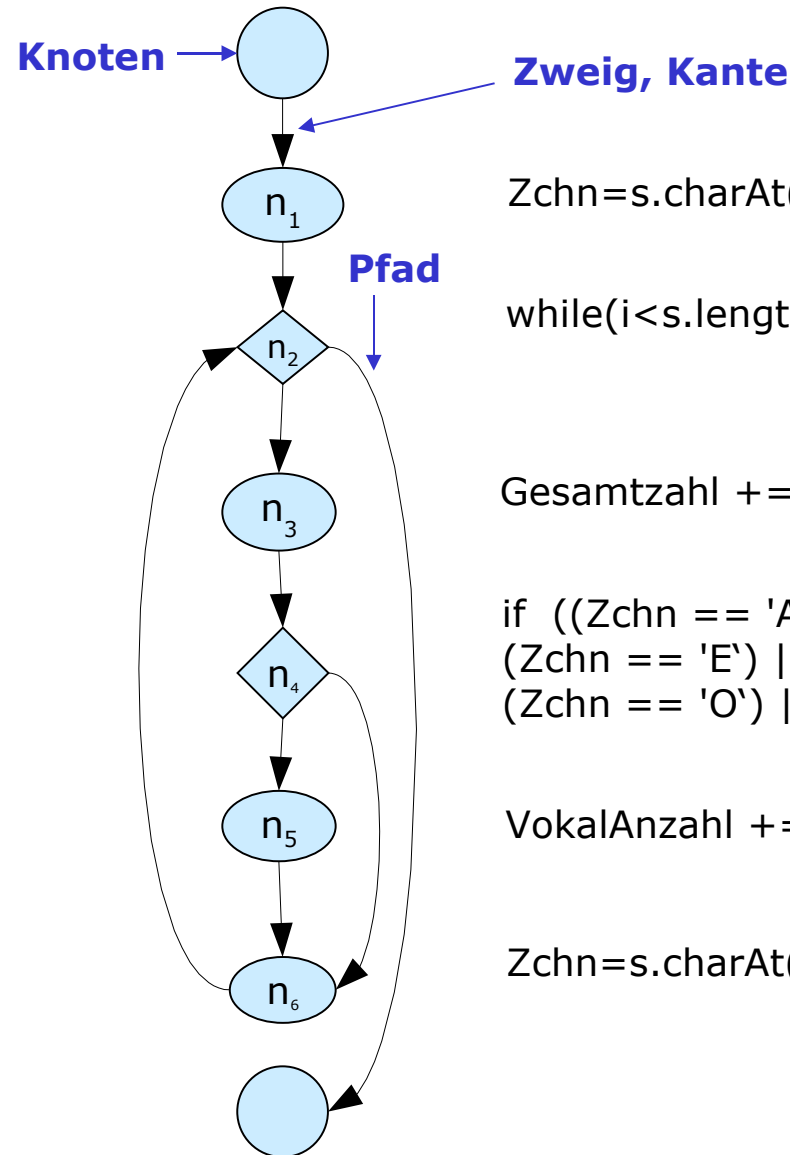
Randbedingung: Die Funktion stellt sicher, dass Gesamtzahl stets größer oder gleich VokalAnzahl ist. */

```
public static void main(String[] argv) {  
    Gesamtzahl=0;  
    VokalAnzahl=0;  
    for(int i=0; i<argv.length; i++)  
        zaehleZeichen(argv[i].toUpperCase());  
    System.out.println("Argumente enthalten "+Gesamtzahl+  
        " Zeichen, davon "+VokalAnzahl+" Vokale");  
}
```

```
public class count {  
  
    static int Gesamtzahl;  
    static int VokalAnzahl;  
  
    static void zaehleZeichen(String s) {  
        int i=0;  
        char Zchn=s.charAt(i++);  
        while(i<s.length()) {  
            Gesamtzahl+=1;  
            if ((Zchn == 'A') || (Zchn == 'E') || (Zchn == 'I')  
                || (Zchn == 'O') || (Zchn == 'U'))  
                VokalAnzahl+=1;  
            Zchn=s.charAt(i++);  
        }  
    }  
}
```

Kontrollflussgraph

- auch Programmablaufplan
- Gerichteter Graph, bestehend aus Knoten und Kanten
- Besitzt einen Start- und einen Endknoten
- Folge von Knoten und Kanten vom Start- zum Endknoten heißt Pfad



```
Zchn=s.charAt(i++)
```

```
while(i<s.length())
```

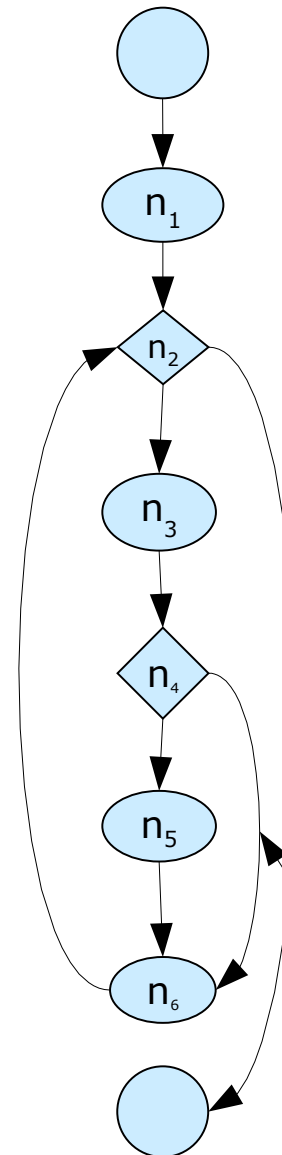
```
Gesamtzahl += 1;
```

```
if ((Zchn == 'A') ||  
(Zchn == 'E') || (Zchn == 'I') ||  
(Zchn == 'O') || (Zchn == 'U'))
```

```
VokalAnzahl += 1;
```

```
Zchn=s.charAt(i++);
```

- Auch C_0 – Test ($C = \text{Coverage}$) genannt
- Verlangt Ausführung aller Anweisungen (**Knoten**)
- Testmenge: $\{[„A“, „1“]\}$
- Ein Test reicht aus.
Testpfad enthält alle Knoten, aber nicht alle Kanten



Zchn=s.charAt(i++)

while(i<s.length())

Gesamtzahl += 1;

if ((Zchn == 'A') ||
(Zchn == 'E') || (Zchn == 'I') ||
(Zchn == 'O') || (Zchn == 'U'))

VokalAnzahl += 1;

Zchn=s.charAt(i++);

Zweig (n₄, n₆) wird nicht
notwendig ausgeführt

2.1 Anweisungsüberdeckungstest

Eigenschaften:

- 100prozentige Überdeckung bedeutet: jede Anweisung wurde mindestens einmal ausgeführt
- Wesentliche Aspekte eines Programms werden nicht geprüft

Metrik: *Überdeckungsgrad* =

Zahl der ausgeführten Anweisungen / Gesamtzahl aller Anweisungen

Leistungsfähigkeit:

- Niedrigste Fehleridentifizierungsquote, 18 % der Fehler werden entdeckt

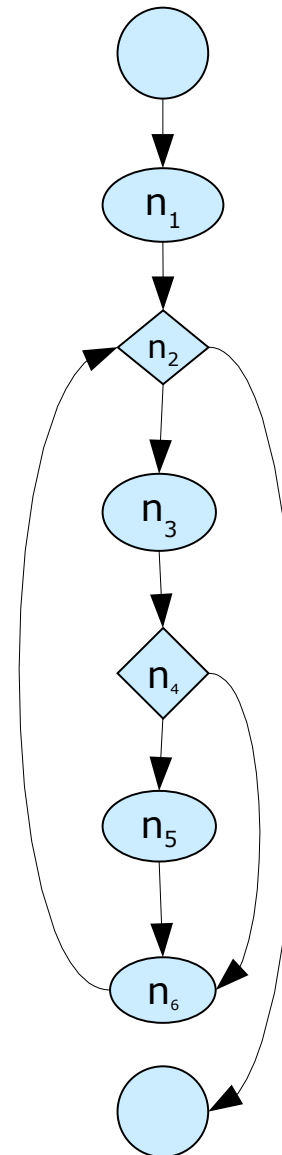
Bewertung:

- *Notwendiges*, aber nicht hinreichendes Testkriterium
- Nicht ausführbarer Code kann gefunden werden
- Eigenständig nicht geeignet, aber in Kombination mit anderen Verfahren

5. Testende Verfahren

2.2 Zweigüberdeckungstest

- Auch C_1 – Test genannt
- Verlangt Ausführung aller Zweige (**Kanten**)
- Testmenge:
 $\{[„A“, „B“, „1“]\}$
- Ein Test reicht aus. Testpfad enthält alle Kanten.
Insbesondere sind die Kanten $n_4--n_5--n_6$ (Durchlauf mit „A“) sowie n_4--n_6 (Durchlauf mit „B“) abgedeckt
- Zweigüberdeckung wird auch als Entscheidungsüberdeckung bezeichnet



Zchn=s.charAt(i++)

while(i<s.length())

Gesamtzahl += 1;

if ((Zchn == 'A') ||
(Zchn == 'E') || (Zchn == 'I') ||
(Zchn == 'O') || (Zchn == 'U'))

VokalAnzahl += 1;

Zchn=s.charAt(i++);

2.2 Zweigüberdeckungstest

Eigenschaften:

- 100prozentige Überdeckung bedeutet: jeder Zweig wurde mindestens einmal durchlaufen.
- Fehlende Zweige können nicht direkt entdeckt werden.

Metrik: *Überdeckungsgrad* =

Zahl der erfassten Kanten / Gesamtzahl aller Kanten

Leistungsfähigkeit:

- Höhere Fehleridentifizierungsquote als Anweisungsüberdeckung, ca. 34% der Fehler werden entdeckt, 79% der Kontrollflussfehler und 20% der Berechnungsfehler
- Leistungsfähigkeit schwankt in weitem Bereich zwischen 25% bis 75%

Bewertung:

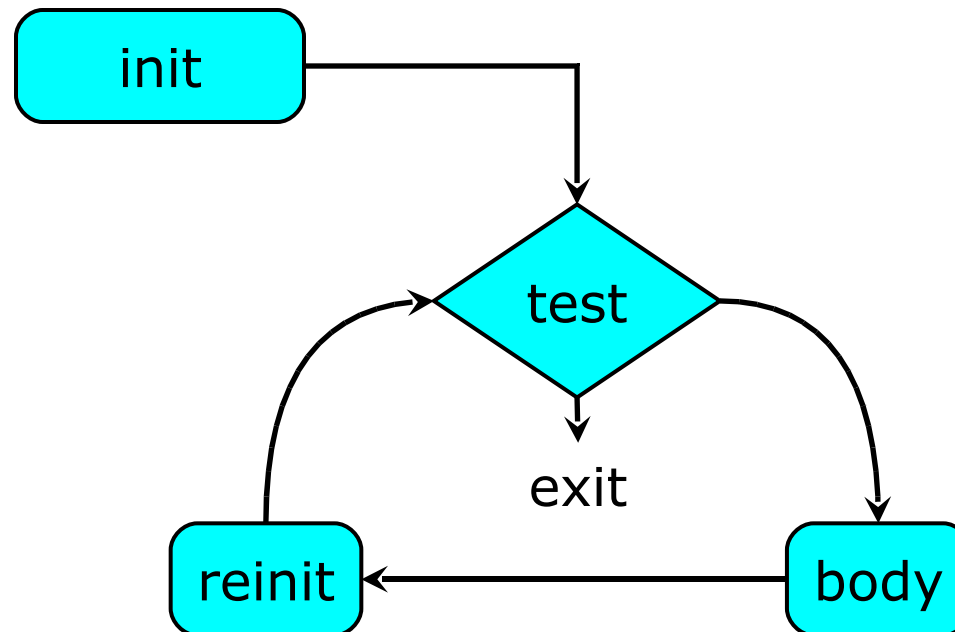
- Gilt als *das* minimale Testkriterium
- Nicht ausführbare Zweige können gefunden werden
- Korrektheit des Kontrollflusses an Verzweigungen wird kontrolliert
- Gezielte Optimierung häufig durchlaufener Programmteile möglich

Nachteile:

- Unzureichend für den Test von Schleifen
- Keine Berücksichtigung von Abhängigkeiten zwischen Zweigen
- Nicht geeignet für den Test komplexer Bedingungen
- Lösung der beiden ersten Nachteile: Pfadüberdeckungstest
- Lösung des letzten Nachteils: Bedingungsüberdeckungstests

Testen von Schleifen

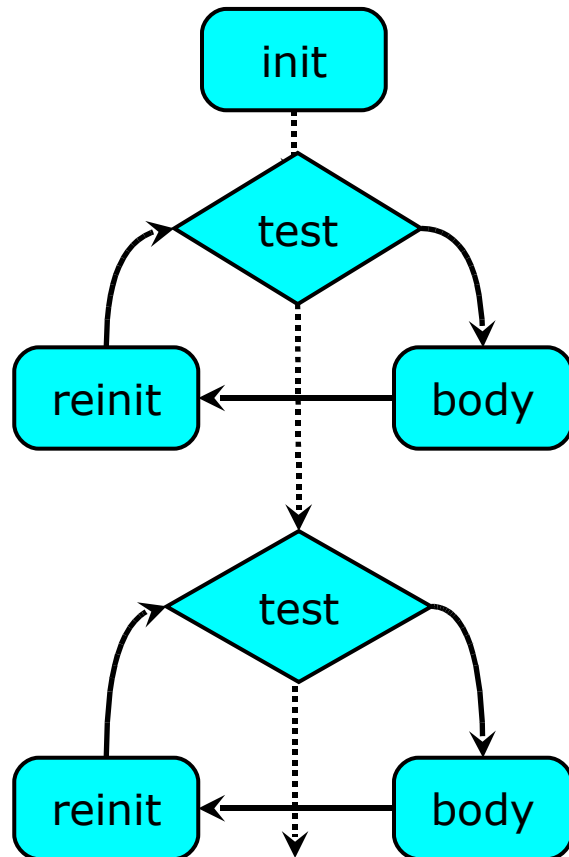
- Anweisungs- und Zweigüberdeckung haben Probleme mit dem Test von Schleifen
- Typische Schleifenstruktur:



5. Testende Verfahren

2.3. Pfadüberdeckungstests

Problem des Wachstums der Anzahl der Pfade



- Zahl der Testfälle von while-Schleifen ist nicht vorab bekannt oder nicht beschränkt
- Zahl der Testfälle konsekutiver Schleifen ist multiplikativ:
 $N = N_1 * N_2$
- Schleife mit Verzweigung im Körper: Ist N Schranke für Zahl der Schleifendurchläufe, so sind im worst case 2^N Testfälle erforderlich (exponentielles Wachstum)

5. Testende Verfahren

2.3 Pfadüberdeckungstest

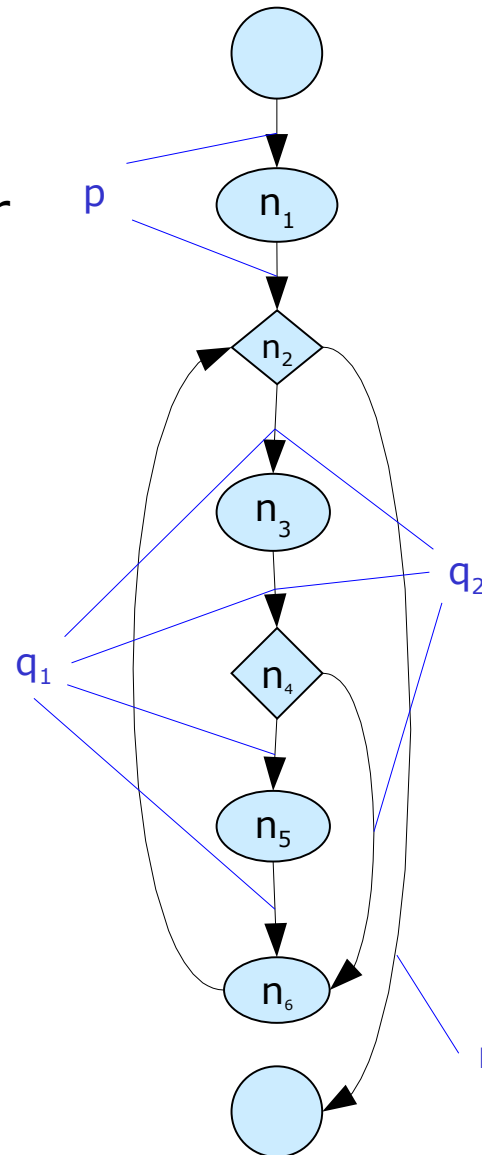
Brute force: Testbeispiele zur Ausführung **aller** unterschiedlichen Pfade im Programm

Beispiel:

Jeder Schleifendurchlauf trägt entweder zu Vokal oder zu Konsonant bei.

Pfade stehen in eindeutiger Korrespondenz zu den Worten über dem Alphabet $\{q_1, q_2\}$.

Anzahl der Testpfade bei Beschränkung auf N Schleifendurchläufe ist also $2^N - 1$.



Zchn=s.charAt(i++)

while(i<s.length())

Gesamtzahl += 1;

if ((Zchn == 'A') ||
(Zchn == 'E') || (Zchn == 'I') ||
(Zchn == 'O') || (Zchn == 'U'))

VokalAnzahl += 1;

Zchn=s.charAt(i++);