Software-Qualitätsmanagement

Kernfach Angewandte Informatik

Sommersemester 2004

Prof. Dr. Hans-Gert Gräbe

Inspektion - Planungsphase

- Festlegung und Einladung eines Inspektionsteams
- jedem Inspektor werden Rollen zu gedacht
  - Beispiele für Rollen:
    - Benutzer: Konzentration auf die Benutzersicht
    - System: Konzentration auf die Implikationen für das Gesamtsystem
    - Finanzen: Konzentration auf Kostenimplikationen, Termine ...
    - Qualität: alle Aspekte von Qualitätsmerkmalen
    - Service: Wartung und Installation
- Festlegung aller Referenzunterlagen für die Inspektion (Ursprungsprodukt, Erstellungsregeln, Checklisten)
- Aufteilung des Prüfobjekts in handhabbare Einheiten, wenn es für eine Sitzung zu umfangreich ist, d.h. mehr als zwei Stunden Sitzung benötigt.
- Festlegung von Terminen
- nach der Planung kann eine Einführungssitzung durchgeführt werden (*kick-off-meeting*)

Inspektion – Vorbereitungsphase

- Jedes Mitglied bereitet sich individuell vor.

Folgende Punkte sind von den Gutachtern zu beachten:

  - Die Vorbereitung muss bis zur Inspektionssitzung abgeschlossen sein.
  - Die Überprüfung ist entsprechend den Inspektionsregeln durchzuführen.
  - Jeder Prüfer sucht nach rollenspezifischen Defekten.
  - Gefundene Defekte sind zu notieren.
  - Für die Güte der individuellen Inspektion ist die empfohlene Arbeitsgeschwindigkeit zu beachten (ca. 1 Seite/h).
- Alternative: Ausschnittsüberprüfung
  - man prüft nur einen Teil des Objekts
  - Fehlerbeispiele zeigen dem Autor typische Schwächen auf
- Die Überprüfung unterscheidet leichte und schwere Defekte.

3. Inspektion - Vorbereitung

Maßnahmen, um sich auf schwere Defekte zu konzentrieren:

- Jeder Inspektor prüft nur festgelegte Aspekte, zur Unterstützung werden Checklisten bereitgestellt, die diese berücksichtigen.
- Im *kick-off-meeting* darauf hinweisen, dass primär schwere Defekte zu suchen sind.
- Die Checklisten auf eine physikalische Seite begrenzen
- In den Checklisten alle Fragen kennzeichnen (ein Symbol für leichte und eins für schwere Defekte).
- Jeder Inspektor markiert die Defektart (schwer/leicht)

- Jeder Inspektor führt eine Aufwandsanalyse (Zeit und Zahl der gefundenen potenziellen Fehler).

Die Inspektionssitzung

- **Ziele**:
  - Protokollierung der gefundenen Defekte
    - mit Angabe defektspezifischer Information (Kurzbeschreibung, Ort, Bezug zu Referenzdokumenten, leicht/schwer)
  - Identifizierung und Protokollierung zusätzlicher Defekte
    - etwa 20 % der Defekte werden in der Sitzung selbst gefunden
  - Protokollierung von anderen Verbesserungsvorschlägen und Fragen an den Autor
- Inspektionssitzung sollte wie eine Brainstormingsitzung ablaufen
- Jeder Inspektor protokolliert anonym die benötigte Zeit, die Anzahl gravierender Fehler und die Anzahl geprüfter Seiten.
- Keine Diskussion oder Kommentierung potentieller Defekte
- Für die Arbeitsgeschwindigkeit sollte ein Ziel gesetzt werden, z.B. wenigstens ein Defekt alle 30 Sekunden

Das Inspektionsprotokoll

- Es sollte einem formalen Schema folgend enthalten:
  - Inspektionsdatum
  - Name des Moderators
  - Prüfobjekt
  - Referenzunterlagen
  - Defekte mit folgenden Angaben:
    - Kurzbeschreibung des Defekts
    - Ort des Defekts
    - Bezug zu Regeln oder Checklisten
    - leichter oder schwerer Fehler
    - in der Sitzung identifiziert oder bei der Vorbereitung
    - Verbesserungsvorschläge
    - Fragen an den Autor
- Nach der Inspektion kann noch ein Prozess-Brainstorming geführt werden („dritte Stunde“)

Inspektion - Überarbeitungsphase

- Anhand des Protokoll führt der Autor folgende Aktivitäten aus:
  - Überarbeitung des Prüfobjekts
  - Änderung Fehlergrad schwer/leicht
  - Änderungsanträge für Referenzprodukte stellen
  - Metriken über „Benötigte Überarbeitungsstunden“ und „Anzahl der schweren Defekte“ an den Moderator melden
  - Im Inspektionsprotokoll vermerken, welche Aktionen pro Protokolleintrag unternommen wurden.
- Der Moderator prüft am Ende die Sorgfalt und Vollständigkeit der überarbeiteten Fassung (nicht aber die Korrektheit!).

- Nach erfolgreicher Nachüberprüfung und Überprüfung der **Freigabekriterien** erfolgt die **formale Freigabe** des Prüfobjekts.

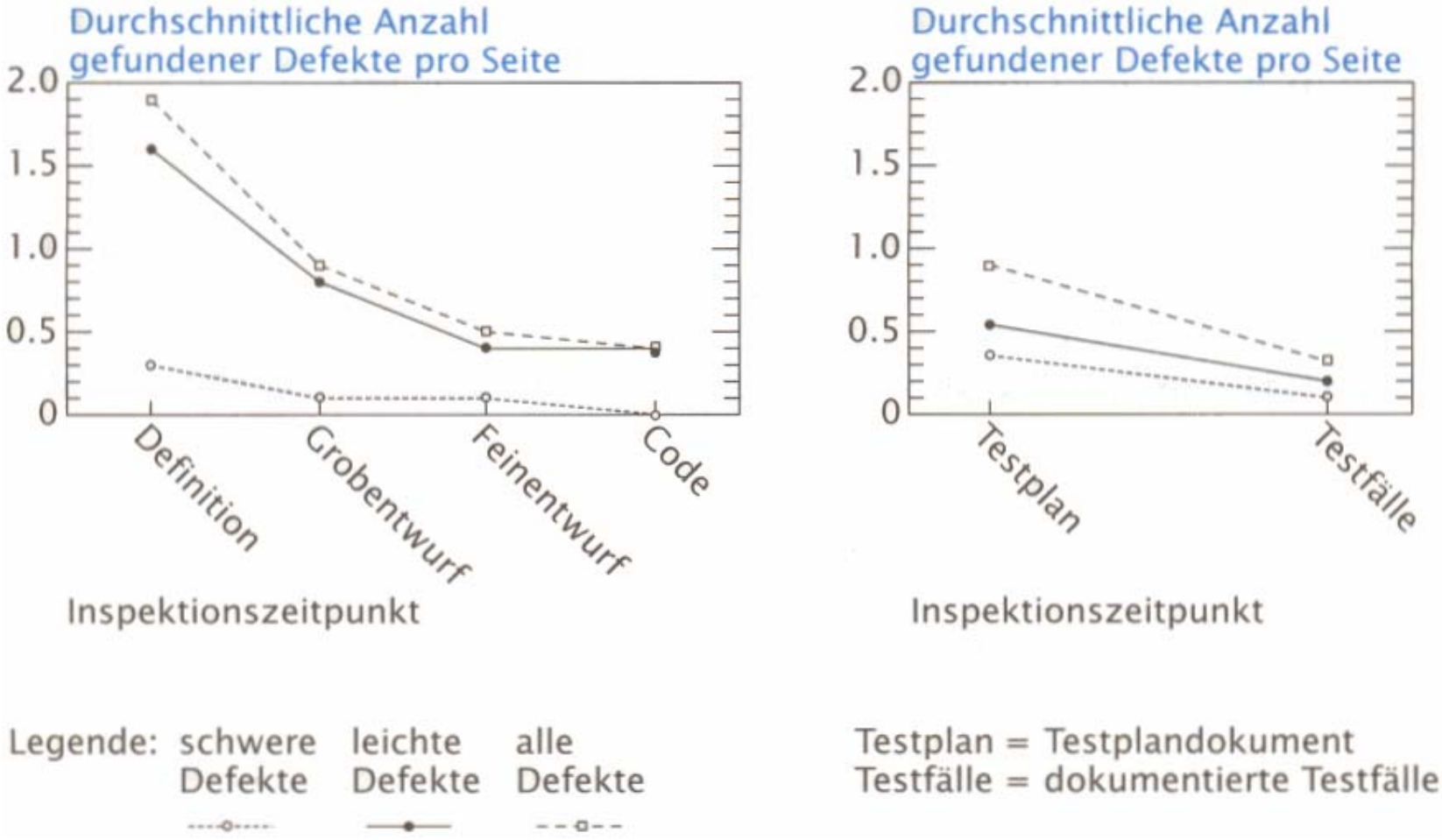
Inspektion – Freigabe des Dokuments

- Freigabekriterien sollen belastbare Abschätzung über die erreichte Qualität des Prüfobjekts ermöglichen.
- insb. Abschätzung der Zahl der verbliebenen schweren Defekte. (Durchschnittliche Inspektionseffektivität liegt bei etwa 40%)
  - Die Anzahl der unentdeckten Defekte ist etwa gleich der Anzahl der entdeckten Defekte pro Seite
  - Eine von sechs Korrekturen wird fehlerhaft ausgeführt
  - Für die weitere Nutzung ist eine schriftliche Fixierung der geschätzten Restdefektrate im Protokoll nützlich.
- Weiter ist die Datensammlung mit den Inspektionsmetriken zu ergänzen (V: Moderator)
- Alternativen bei Zurückweisung eines Produkts wegen zu vieler geschätzter Fehler:
  - Grundlegende Überarbeitung des Prüfobjekts (über die protokollierten Punkte hinaus)
  - Erstellung eines neuen Objekts
  - Wiederholung der Inspektion nach der Überarbeitung

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.

3. Inspektion - Empirische Ergebnisse



Inspektionseffektivität bezogen auf den Inspektionszeitpunkt

3. Inspektion – Empirische Ergebnisse

- 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 ≈ 11.8
- Weitere Vorteile:
  - Kürzere Entwicklungszeit (1700 Std. ≈ 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 Prüfungsmethoden

- **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, Produkteigenschaften zu überprüfen, welche durch automatische Werkzeuge nicht oder nur unzureichend festgestellt werden können.

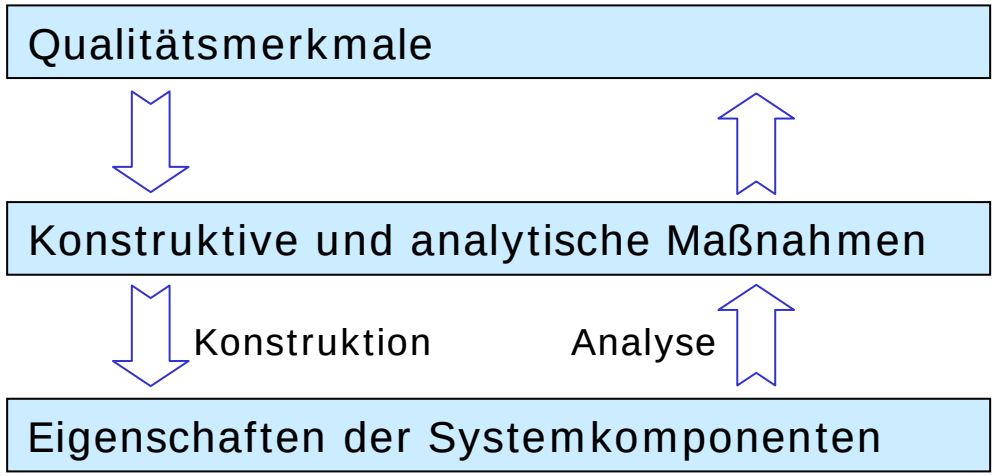
Die Effektivität hängt vor allem von folgenden Punkten ab

- 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 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

Was ist ein Fehler?

- Abweichung des Ist-Zustands eines Qualitätsmerkmals vom Soll-Zustand,
- Inkonsistenzen zwischen Spezifikation und Implementierung,
- Strukturelle Merkmale des Quellcodes, welche fehlerhaftes Verhalten des Programms verursachen.

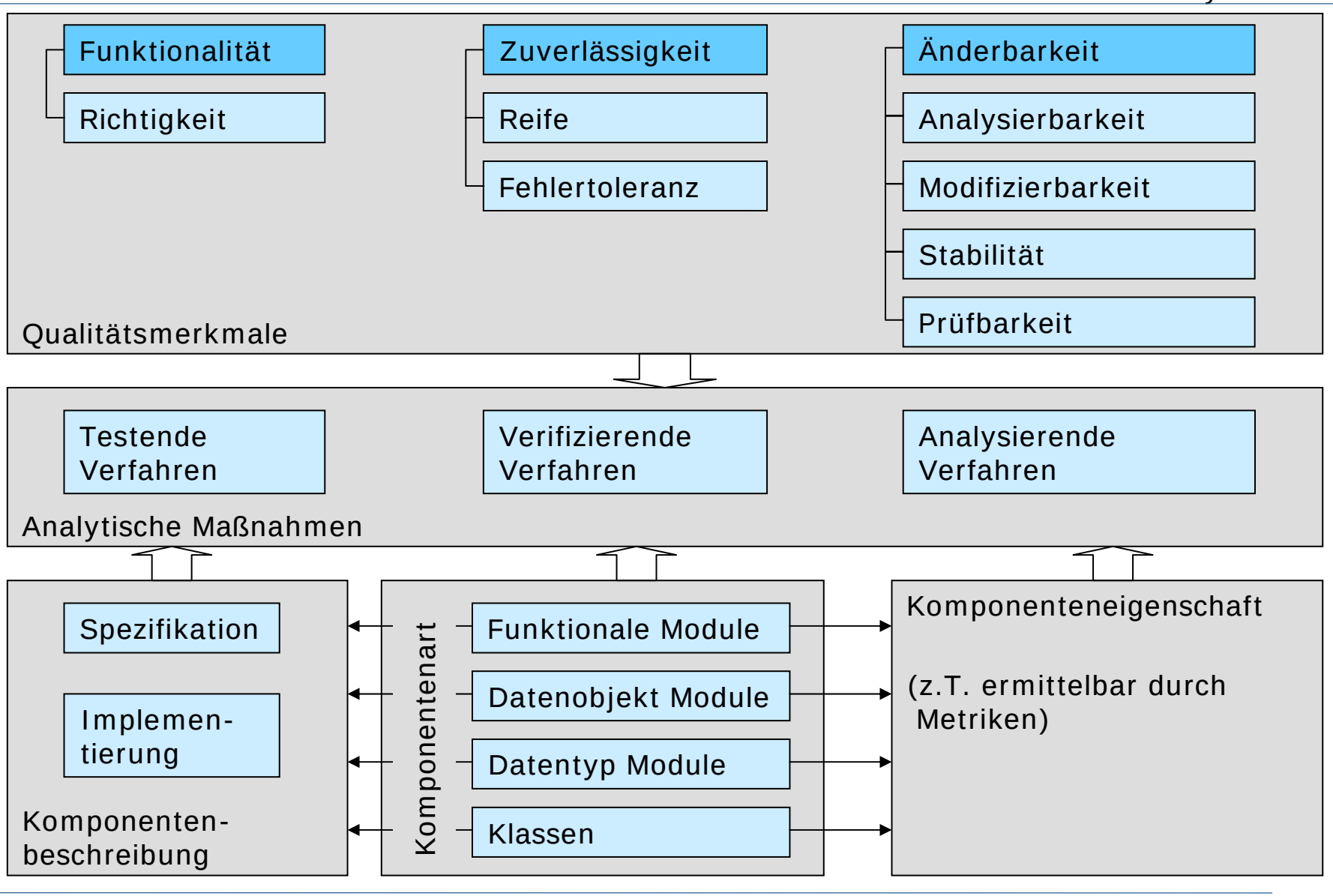
Konstruktives Ziel ➔ fehlerfreie SW-Komponenten

Analytisches Ziel ➔ Nachweis der Fehler bzw. deren Nichtexistenz

Art der Systemkomponenten bestimmt die analytischen Maßnahmen:

- Funktionale Module,
- Datenobjekt-Module,
- Datentyp-Module und
- Klassen

1. Einführung



2. Klassifikation analytischer Verfahren

- Testende Verfahren: ➔ Ziel: Fehler erkennen
  - Dynamische Testverfahren
  - Statische Verfahren
- Verifizierende Verfahren: ➔ Korrektheit einer Komponente beweisen
  - Verifikation
  - Symbolische Ausführung
- Analysierende Verfahren: ➔ Vermessung bzw. Darstellung von Eigenschaften von Komponenten
  - Analyse der Bindungsart
  - Metriken
  - Grafiken und Tabellen
  - Anomalienanalyse

Beispiel: C++-Prozedur (1)

Das Beispiel gilt für alle Testenden Verfahren!

```
/*Programmname: ZaehleZchn (Spezifikation/Header)

Aufgabe:

Die Prozedur ZaehleZchn liest solange Zeichen von der Tastatur, bis ein
Zeichen erkannt wird, das kein Großbuchstabe ist, oder Gesamtzahl den
größten durch den Datentyp int darstellbaren Wert INT_MAX erreicht.

Ist ein gelesenes ein Großbuchstabe zwischen A und Z, dann wird
Gesamtzahl um eins erhöht. Ist der Großbuchstabe ein Vokal, dann wird
auch VokalAnzahl um eins erhöht.

Ein-/Ausgabeparameter sind Gesamtzahl und VokalAnzahl.

Randbedingung:

Das aufrufende Programm stellt sicher, dass Gesamtzahl stets größer oder
gleich VokalAnzahl ist.

*/
```

Beispiel: C++-Prozedur (2)

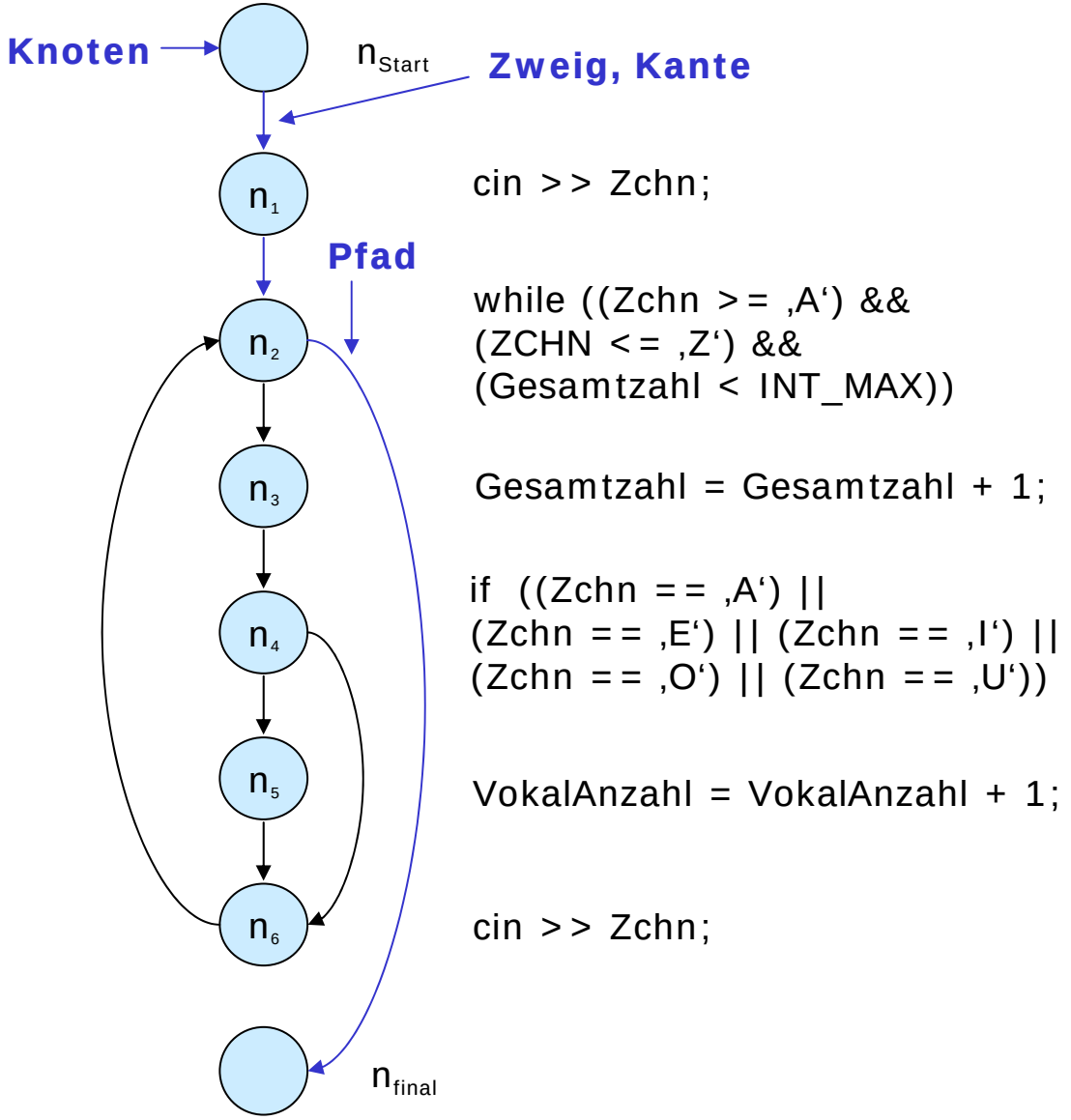
```
void ZaehleZchn(int &VokalAnzahl, int &Gesamtzahl);

#include „ZaehleZchn.h“
#include <LIMITS.H>
#include <iostream.h>

void ZaehleZchn(int &VokalAnzahl, int &Gesamtzahl)
{
    char Zchn;
    cin >> Zchn;
    while ((Zchn >= ‚A‘) && (ZCHN <= ‚Z‘) && (Gesamtzahl < INT_MAX))
    {
        Gesamtzahl = Gesamtzahl + 1;
        if ((Zchn == ‚A‘) || (Zchn == ‚E‘) || (Zchn == ‚I‘) ||
        (Zchn == ‚O‘) || (Zchn == ‚U‘))
        {
            VokalAnzahl = VokalAnzahl + 1;
        }
        cin >> Zchn;
    }
}
```

Beispiel: C++-Prozedur (3)

```
void main()
{
    int AnzahlVokale = 0;
    int AnzahlZchn = 0;
    cout << „Programm ZaehleZchn“ << endl;
    cout << „Zeichen bitte eingeben:“ << endl;
    ZaehleZchn(AnzahlVokale, AnzahlZchn);
    cout << „Anzahl Vokale: “ << AnzahlVokale << endl;
    cout << „Anzahl Zeichen: “ << AnzahlZchn << endl;
}
```

3. Kontrollflussgraph

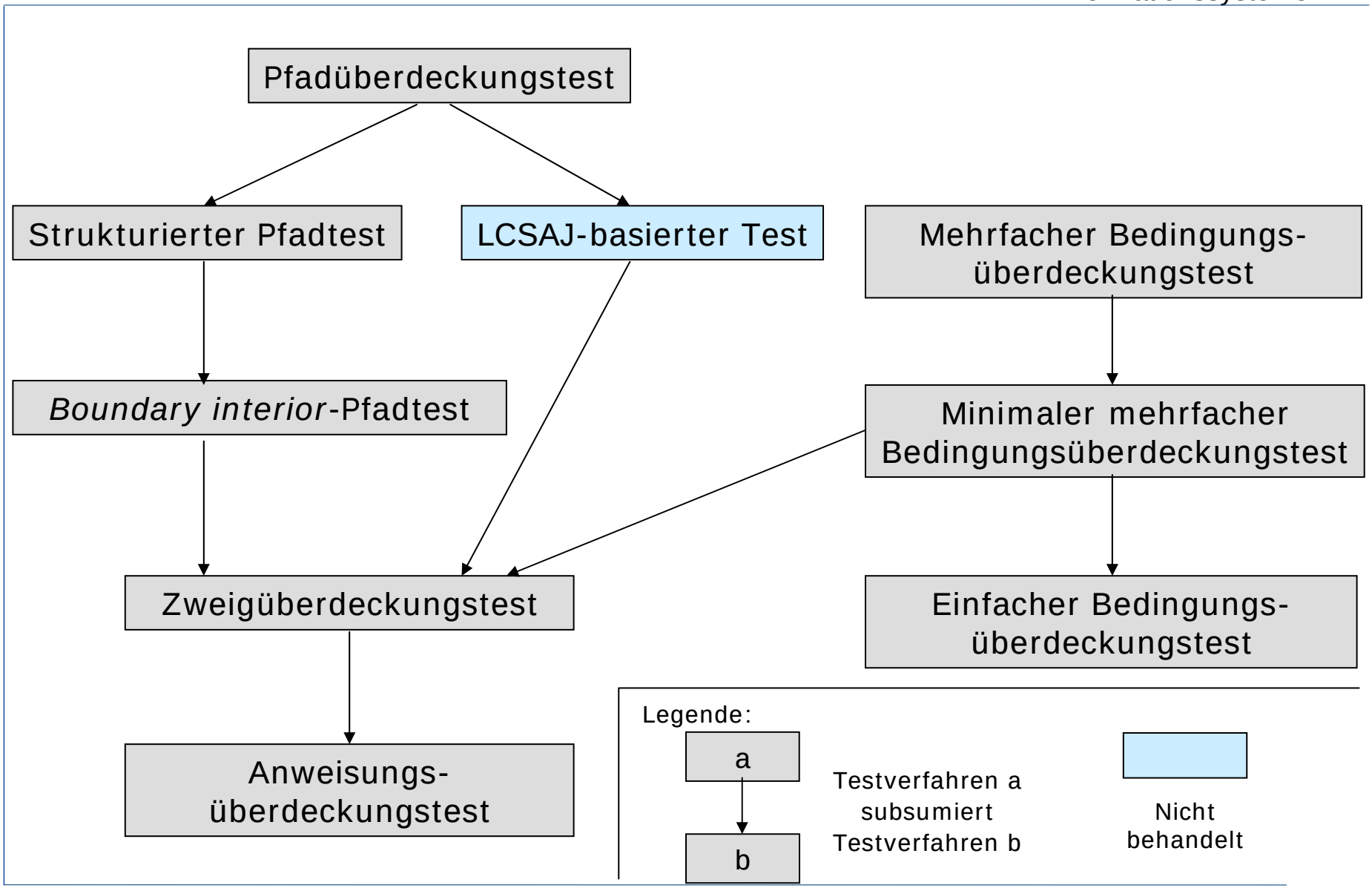
- auch Programm-ablaufplan
- Gerichteter Graph, bestehend aus Knoten und Kanten
- Besitzt einen Start- und einen Endknoten
- Folge von Knoten und Kanten vom Start- zum Endknoten heißt Pfad



Überblick

- Basieren auf der Kontrollstruktur des zu prüfenden Programms
- Gehören zu den Strukturtest-, White Box- oder Glass Box-Verfahren
- Sind dynamische Testverfahren, d.h. das Programm wird ausgeführt
- **Ziel** ➔ mit wenigen Testfällen alle Anweisungen, Zweige oder Pfade zu durchlaufen

4. Kontrollflussorientierte Strukturtestverfahren



4.1 Anweisungsüberdeckungstest

Eigenschaften:

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

Metrik:

$$C_{Anweisung} = \frac{Anzahl\ der\ ausgeführt en\ Anweisunge n}{Gesamtzahl\ der\ vorhandene n\ Anweisunge n}$$

Leistungsfähigkeit:

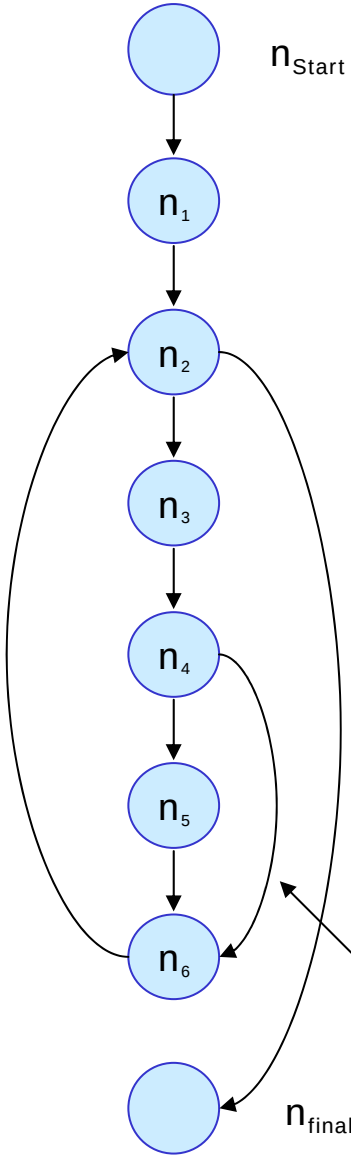
- Niedrigste Fehleridentifizierungsquote, 18 % der Fehler werden entdeckt

Bewertung:

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

4.1 Anweisungsüberdeckungstest

- Auch C_0 – Test (C = *Coverage*) genannt
- Verlangt Ausführung aller Anweisungen (**Knoten**)
- Folgender Testfall:
  Aufruf ZaehleZchn mit:
  Gesamtzahl = 0
  Eingelesene Zeichen: ‚A' - ‚1'
  Durchlaufener Pfad:
  (n_{Start}, n_1, n_2, n_3, n_4, n_5, n_6, n_2, n_{final})
- Testpfad enthält alle Knoten, aber nicht alle Kanten



cin >> Zchn;

while ((Zchn >= ‚A') &&
(ZCHN <= ‚Z') &&
(Gesamtzahl < INT_MAX))

Gesamtzahl = Gesamtzahl + 1;

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

VokalAnzahl = VokalAnzahl + 1;

cin >> Zchn;

Zweig (n_4, n_6) wird nicht notwendig ausgeführt

4.2 Zweigüberdeckungstest

Eigenschaften:

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

Metrik:

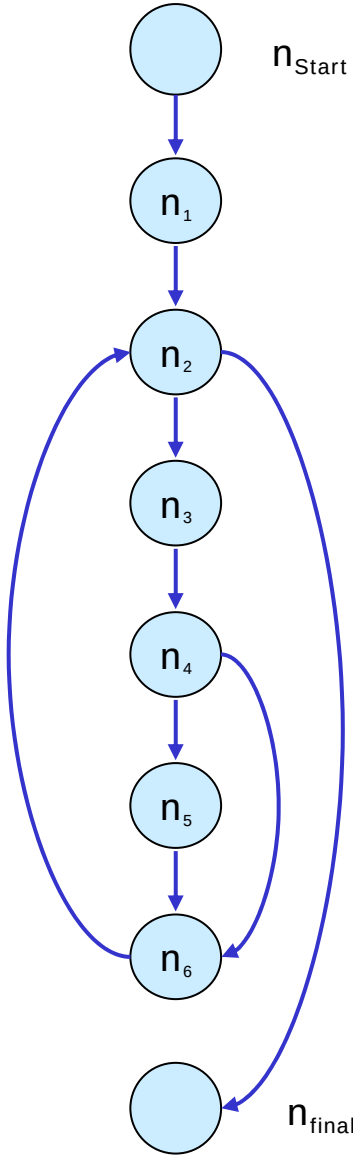
$$C_{Zweig} = \frac{Anzahl\ der\ ausgeführt en\ Zweige}{Gesamtzahl\ der\ vorhandene n\ Zweige}$$

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%
- Erfolgsquote ist höher als bei statischer Analyse

4.2 Zweigüberdeckungstest

- Auch C_1 – Test genannt
- Verlangt Ausführung aller Zweige (**Kanten**)
- Folgender Testfall:
  Aufruf ZaehleZchn mit:
  Gesamtzahl = 0
  Eingelesene Zeichen: ‚A' – ‚B' - ‚1'
  Durchlaufener Pfad:
  (n_{Start}, n_1, n_2, n_3, n_4, n_5, n_6, n_2, n_3, n_4, n_6, n_2, n_{final})
- Testpfad enthält alle Kanten, insbesondere die Kante (n_4, n_6)
- Zweigüberdeckung auch als Entscheidungs-überdeckung benannt




```
cin >> Zchn;

while ((Zchn >= ‚A') &&
(ZCHN <= ‚Z') &&
(Gesamtzahl < INT_MAX))

Gesamtzahl = Gesamtzahl + 1;

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

VokalAnzahl = VokalAnzahl + 1;

cin >> Zchn;
```

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

4.3 Pfadüberdeckungstest

Eigenschaften:

- Pfadanzahl wächst bei unbestimmten Wiederholungen (while, ...) explosiv.
- Ein Teil der konstruierbaren Pfade ist nicht ausführbar, da sich Bedingungen gegenseitig ausschließen können.

Leistungsfähigkeit:

- *Mächtigstes* kontrollflussorientiertes Testverfahren
- In einer vergleichenden Studie [Howden 78a, b, c] Erkennung von 18 von 28 Fehlern, um den Faktor 3 höhere Erkennung als Zweigüberdeckung
- Höhere Erfolgsquote nur durch Kombination mit anderen Verfahren

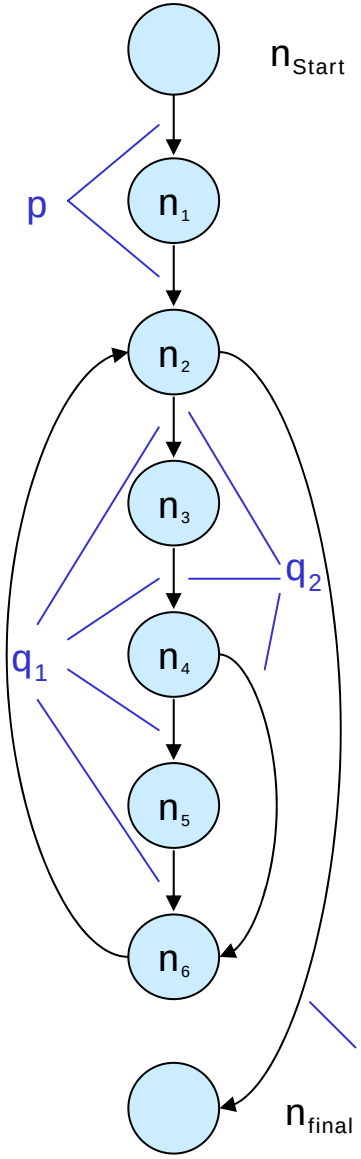
Bewertung:

- Keine praktische Bedeutung, da nur sehr eingeschränkt Durchführbar
- Bedeutung in Zusammenhang mit fehlerorientierten Testansätzen

4.3 Pfadüberdeckungstest

- Erfordert die Ausführung **aller** unterschiedlichen Pfade im Programm
- Entwickelt um Wiederholungen und Schleifen testen zu können
- Beispiel rechts:
  alle möglichen Pfade von ZaehleZchn beginnen mit Subpfad $p=(n_{Start}, n_1)$, dann folgen k Subpfade q, gefolgt vom Subpfad $r=(n_2, n_{final})$.

  die k Subpfade q haben die Form $(n_2, n_3, n_4, n_5, n_6)$ oder (n_2, n_3, n_4, n_6)
  Anzahl der Schleifendurchläufe in ZaehleZchn ist durch die maximale Differenz zwischen Gesamtzahl und INT_MAX begrenzt
  Anzahl an Testpfaden:
  $2^{2.147.483.647}-1$ unter der Annahme das der größte Werte für int Variablen 2.147.483.647 ist




```
cin >> Zchn;

while ((Zchn >= ‚A‘) &&
(ZCHN <= ‚Z‘) &&
(Gesamtzahl < INT_MAX))

Gesamtzahl = Gesamtzahl + 1;

if  ((Zchn == ‚A‘) ||
(Zchn == ‚E‘) || (Zchn == ‚I‘) ||
(Zchn == ‚O‘) || (Zchn == ‚U‘))

VokalAnzahl = VokalAnzahl + 1;

cin >> Zchn;
```

4.3 Pfadüberdeckungstest

Boundary Interior – Test

- Abgeleitet vom Pfadüberdeckungstest, für Programme ohne Schleifen sogar identisch
- Schleifen werden beim Test nur einmal überprüft
- Zwei Gruppen von Pfaden für jede Schleife im Programm:
  1. Grenztest-Gruppe (*boundary tests*):
     alle Pfade die die Schleife betreten, aber nicht wiederholen
  2. Gruppe zum Test des Schleifeninneren (*interior tests*):
     alle Pfade, die mindestens eine Schleifenwiederholung beinhalten
- Erlaubt gezielte Überprüfung von Schleifen
- Praktisch anwendbar (im Gegensatz zum Pfadüberdeckungstest)
- **Strukturierter Pfadtest**
  ➔ Verallgemeinerung des *Boundary Interior* Tests

4.4 Bedingungsüberdeckungstest

Ziel:

- Analysiert und überprüft die Bedingungen für Schleifen und bedingte Anweisungen

Eigenschaften:

- 3 verschiedene Varianten:
  1. Einfache Bedingungsüberdeckung:

     überdeckt alle atomaren Bedingungen, Evaluation dieser muss jeweils mindestens einmal „true“ und „false“ ergeben
  2. Mehrfach-Bedingungsüberdeckung:

     Überdeckung aller Variationen von atomaren Bedingungen
     ➔ hohe Anzahl an Testfällen (2^n Variationsmöglichkeiten)
  3. Minimale Mehrfach-Bedingungsüberdeckung:

     jede Bedingung (ob atomar oder nicht) muss jeweils mindestens einmal „true“ und „false“ annehmen

4.4 Bedingungsüberdeckungstest

Bewertung:

- Einfache Bedingungsüberdeckungstests:

  weder Zweig- noch Anweisungsüberdeckung enthalten
  ➔ alleinige einfache BÜ reicht nicht aus

- Mehrfach-Bedingungsüberdeckungstests:

  Zweigüberdeckung ist enthalten, jedoch ist Mehrfach-BÜ aufwendig zu realisieren

- Minimale Mehrfach-Bedingungsüberdeckungstest:
  - Zweigtest ist enthalten, zusätzlich werden invariante Bedingungen entdeckt,
  - Sinnvolle Weiterentwicklung des Zweigtests

Überblick

- Ebenso wie kontrollflussorientierte Verfahren dynamische Strukturtestverfahren
- Im Gegensatz zu kontrollflussorientierten Verfahren werden Datenbenutzungen getestet.
- Definition von Variablen sowie lesende und schreibende Zugriffe auf diese Variablen
- Eignen sich für Test von Datenobjekt- und Datentypmodulen sowie Klassen.
- Nur wenige Testwerkzeuge vorhanden.

Literatur

[Balzert, Balzert, Liggesmeyer 93]

Balzert Helmut, Balzert Heide, Liggesmeyer P., *Systematisches Testen mit Tensor*, Mannheim: BI-Wissenschaftsverlag

[Girgis, Woodward 86]

Girgis M. R., Woodward M. R., *An Experimental Comparison of the Error Exposing Ability of Program Testing Criteria*, in Proceedings Workshop on Software Testing, Banff, July 1986

[Howden 78a]

Howden W. E., *An Evaluation of the Effectiveness of Symbolic Testing*, in: Software-Practice and Experience, Vol. 8, 1978

[Howden 78b]

Howden W. E., *Theoretical and Empirical Studies of Program Testing*, in: IEEE Transactions on Software Engineering, Vol. SE-4, No. 4, July 1978

[Howden 78c]

Howden W. E., *Theoretical and Empirical Studies of Program Testing*, in: Proceedings of the 3rd International Conference on Software Engineering, Atlanta, May 1978

Literatur

[Liggesmeyer 90]

Liggesmeyer P., *Modultest und Modulverifikation – State of the Art*, Mannheim: BI-Wissenschaftsverlag 1990

[Liggesmeyer 93]

Liggesmeyer P., *Wissensbasierte Qualitätsassistenz zur Konstruktion von Prüfstrategien für Software-Komponenten*, Mannheim: BI-Wissenschaftsverlag 1993