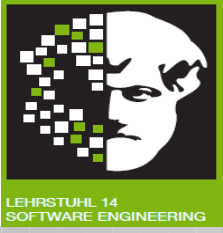
Willkommen zur Vorlesung
Softwarekonstruktion
im Wintersemester 2011/2012

Prof. Dr. Jan Jürjens

TU Dortmund, Fakultät Informatik, Lehrstuhl XIV



07. Testen

[inkl Beiträge von Prof. Volker Gruhn]

LEHRSTUHL 14
SOFTWARE ENGINEERING

Agenda

8 Testen
- Ursachen von SW-Fehlern
- Klassifikation von SW-Fehlern
- Einleitende Begriffe (rund ums Testen)
- Analytisches Qualitätsmanagement

Ursache von SW-Fehlern

- Fehler in der Kommunikation bei der
  - Kodierung
    - Es wird nicht das eigentlich Gemeinte gesagt / geschrieben
  - Übermittlung
    - Informationen werden falsch weitergegeben
  - Dekodierung
    - Das Gesagte / Geschriebene wird anders verstanden als gemeint

Klassifikation von SW-Fehlern

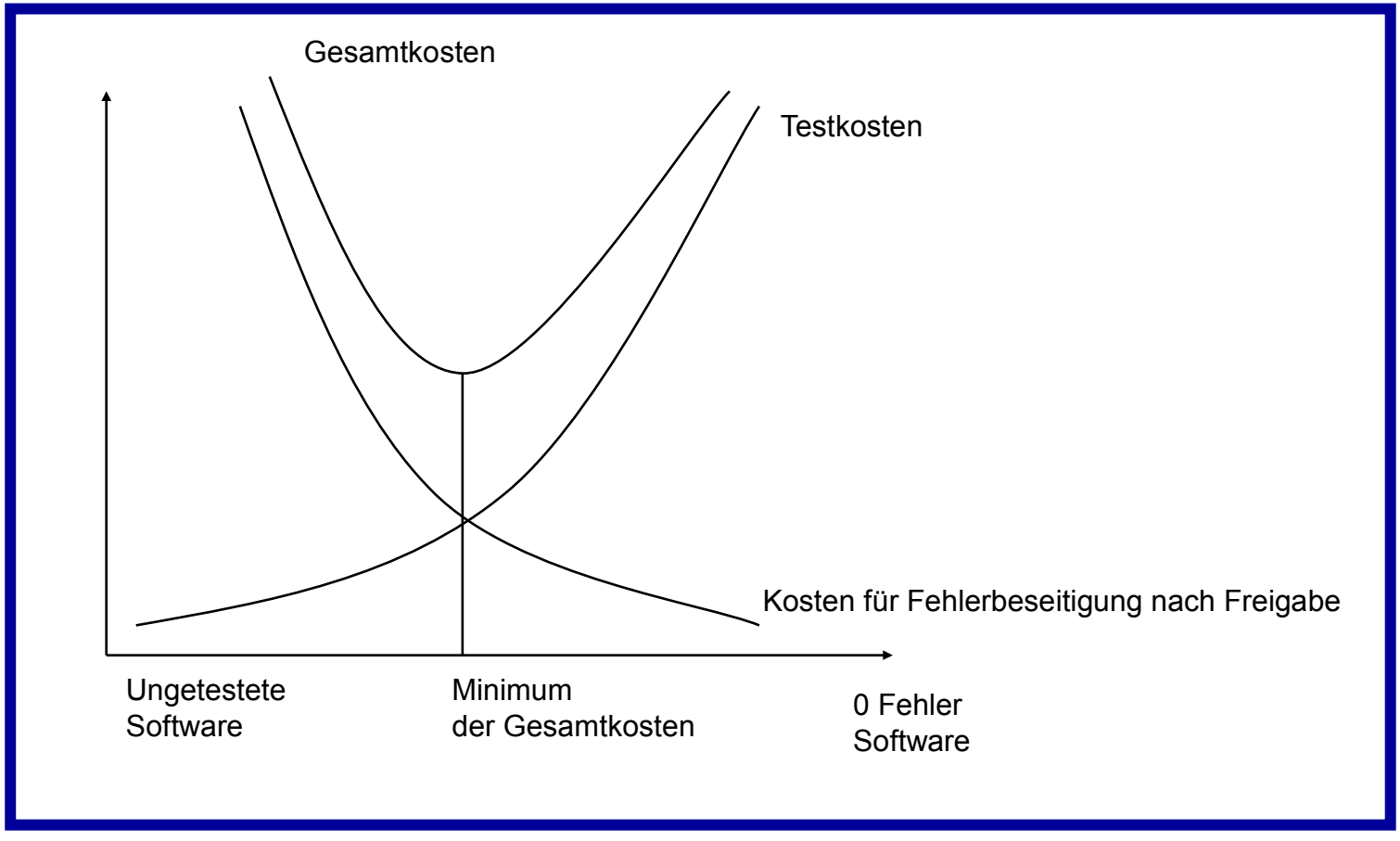
- Unterscheidung von Fehler und Mangel
  - Fehler
    - Nichterfüllung einer festgelegten Forderung, insbes. eines Qualitäts- und Zuverlässigkeitsmerkmals (DIN EN ISO 8402)
  - Mangel
    - Nichterfüllung einer beabsichtigten Forderung oder einer angemessenen Erwartung (DIN EN ISO 8402)
      - Mit der angemessenen Erwartung gibt's beliebige Probleme!
        - Konnte der Anwender einer Software mit ASCII-Oberfläche damit rechnen, dass die neue Software (mit grafischer Oberfläche) mehrfaches Klicken erfordert?
        - Musste er davon ausgehen, dass der Aufbau einer grafischen Oberfläche 2 Sekunden dauert?
        - Hat er zu recht erwartet, dass er die Standardeingaben mit dem Zahlenfeld erledigen kann?
    - Hier geht es oft um die nicht genau spezifizierte Benutzerfreundlichkeit
  - Unterscheidung wichtig für Auftragnehmer und Auftraggeber im Rahmen der Vertragsgestaltung

Klassifikation von SW-Fehlern

- Kosten höher
  - Je früher in der Entwicklungsphase der Fehler entsteht
  - Je später in der Entwicklungsphase der Fehler entdeckt wird
  - Summationseffekt von Fehlern und Mängeln

Anforderungs-definition	Korrekte Anforderungen	Fehlerhafte Anforderungen		
System-spezifikation	Korrekte Spezifikation	Spezifikations-fehler	Induzierte Fehler aus Anforderungen	
Entwurf	Korrekter Entwurf	Entwurfs-fehler	Induzierte Fehler aus Anforderungen + Spezifikation	
Realisierung	Korrektes Programm	Programm-fehler	Induzierte Fehler aus Anforderungen + Spezifikation + Entwurf	
Test + Integration	Korrektes Verhalten	Korrigierte Fehler	Bekannte unkorrigierte Fehler	Unbekannte Fehler

Wirtschaftlichkeit des Testens



H.-U. Frehr,
Total Quality Management - Unternehmensweite Qualitätsverbesserung, Carl Hanser Verlag, 1994

Einleitende Begriffe

- Qualität
  - „Qualität ist die Beschaffenheit einer Einheit bezüglich ihrer Eignung, festgelegte und vorausgesetzte Erfordernisse zu erfüllen“ (DIN 55350, Teil 11)
- Qualität ist die Übereinstimmung mit den Kundenanforderungen, und zwar die mit den tatsächlichen Anforderungen von den beteiligten Personen.
  - Vertrieb will andere Qualitäten als der Entwickler als der Kunde als der Marketier
  - Die zu erreichenden Qualitäten müssen möglichst klar und verständlich spezifiziert und offensiv kommuniziert werden.
- Qualität entsteht während der Entwicklung und lässt sich nicht (oder nur sehr schwer) später einbauen.

Einleitende Begriffe

- Produktorientiertes Qualitätsmanagement
  - Software-Produkte und Zwischenergebnisse werden auf vorher festgelegte Qualitätsmerkmale überprüft.
  - Bei Anwendungs-Software gehören Gütebedingungen und Prüfbestimmungen dazu.
  - Diese können Gegenstand einer Zertifizierung sein, d.h. einer Bestätigung durch anerkannte (akkreditierte) Stellen, dass bestimmte Normen eingehalten wurden.

- Prozessorientiertes Qualitätsmanagement
  - Bezieht sich auf den Erstellungsprozess der Software.
  - Dazu gehören Methoden, Werkzeuge, Richtlinien und Standards.
  - Alle Mitarbeiter müssen ein Qualitätsbewusstsein entwickeln, das den Produktionsprozess hin zu einer optimalen Qualität verändert.

Einleitende Begriffe

- Projektinternes Qualitätsmanagement
  - Umfasst alle innerhalb des Projektes und durch das Projektpersonal durchgeführten Qualitätssicherungsmaßnahmen.
  - Hierzu zählen die Unterstützung der Entwickler bei Komponenten- und Modultest, oft auch die Unterstützung bei den Integrationstests.
  - Für diese Maßnahmen gibt es in den meisten Projekten ab einer gewissen Größenordnung einen projektinternen Qualitätssicherungsbeauftragten, der dem Projektleiter unterstellt ist (für die Dauer des Projektes) und der ansonsten zur QS-Abteilung gehört.

- Projektexternes Qualitätsmanagement
  - Umfasst alle Qualitätssicherungsmaßnahmen, die von projektunabhängigem Personal durchgeführt werden.
  - Hierzu zählen mindestens der Abnahmetest, oft auch der Systemtest.

Einleitende Begriffe

- Konstruktives Qualitätsmanagement
  - Sind Methoden, Sprachen, Werkzeuge, Richtlinien, Standards und Checklisten, die dafür sorgen, dass das entstehende Produkt bzw. der Erstellungsprozess **à priori** bestimmte Eigenschaften besitzt.

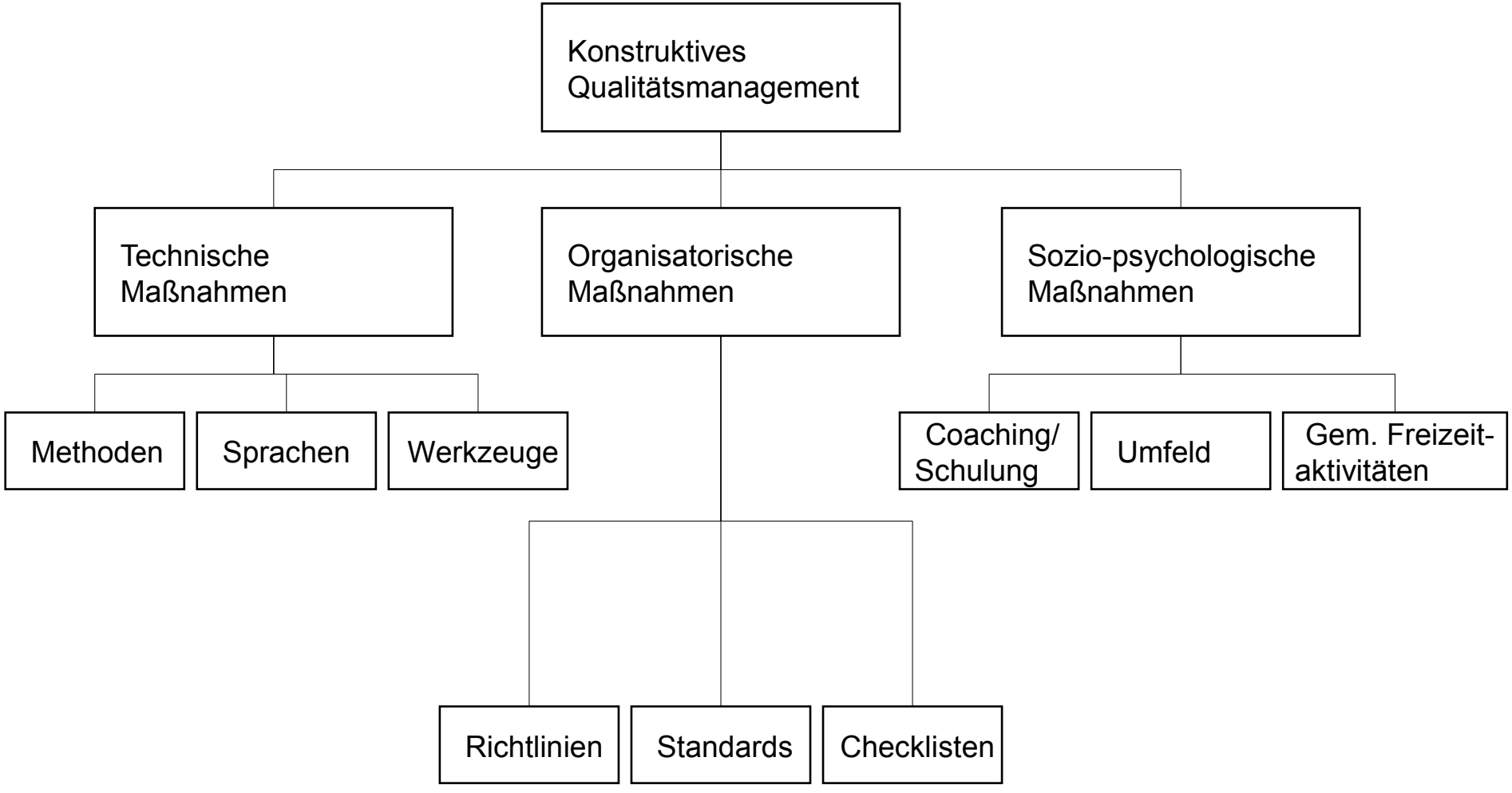
- Analytisches Qualitätsmanagement
  - Sind diagnostische Maßnahmen **ex post**, d.h. sie bringen in das Produkt oder den Entwicklungsprozess keine Qualität per se.
  - Durch analytische Maßnahmen wird das existierende Qualitätsniveau gemessen.
  - Ausmaß und Ort der Defekte können identifiziert werden.
  - Das Ziel ist also die Prüfung und Bewertung der Prüfobjekte.

Akkreditierung und Zertifizierung

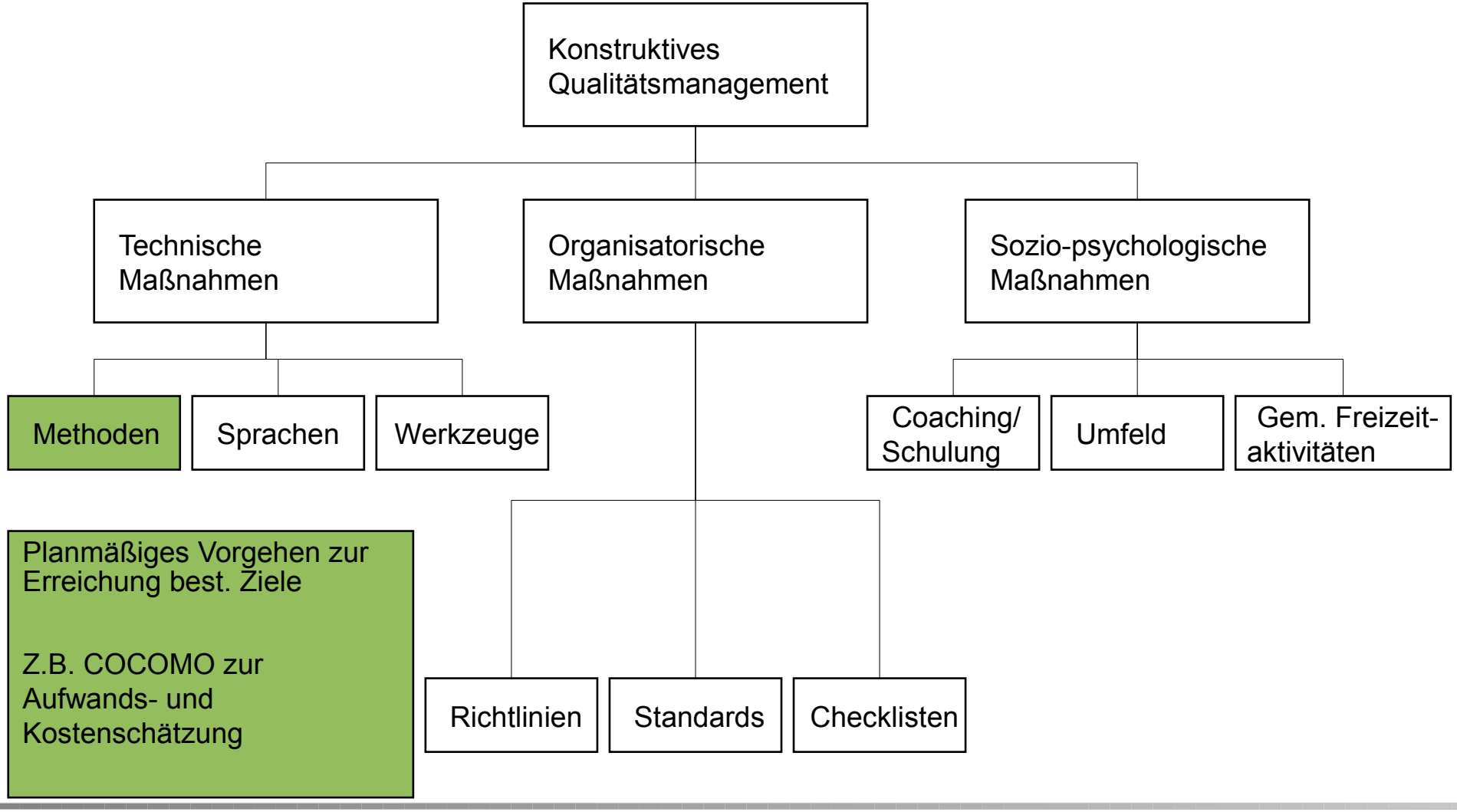
- Gelegentlich irreführende Vielfalt:
  - Was wird getestet: Spezifikation, Produkt oder Produkterstellungsprozess? Oder ein ganz anderer Teil des Unternehmens?

- Negative Beispiele:
  - „Sichere Onlineshops“ mit SQL Injection Lücken.
  - Zertifizierte, (fast nicht) verschlüsselte USB-Sticks („FIPS 140-2 Level 2“)
  - Zertifizierung für Betriebssysteme mit ungewöhnlichen Einschränkungen (Ohne Laufwerk, ohne Netzwerkanschluss...)

Übersicht über das konstruktive Qualitätsmanagement:

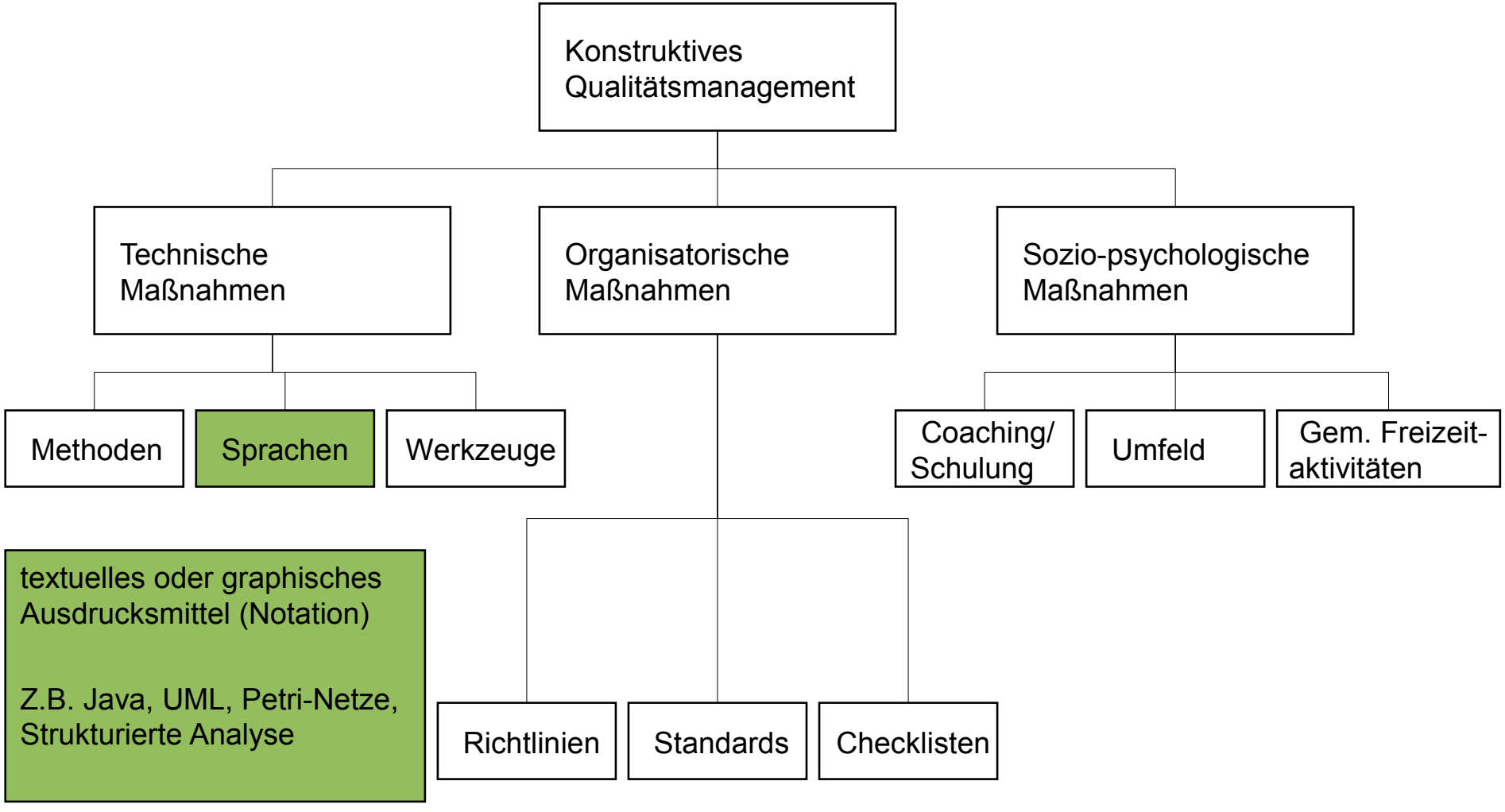


Übersicht über das konstruktive Qualitätsmanagement:

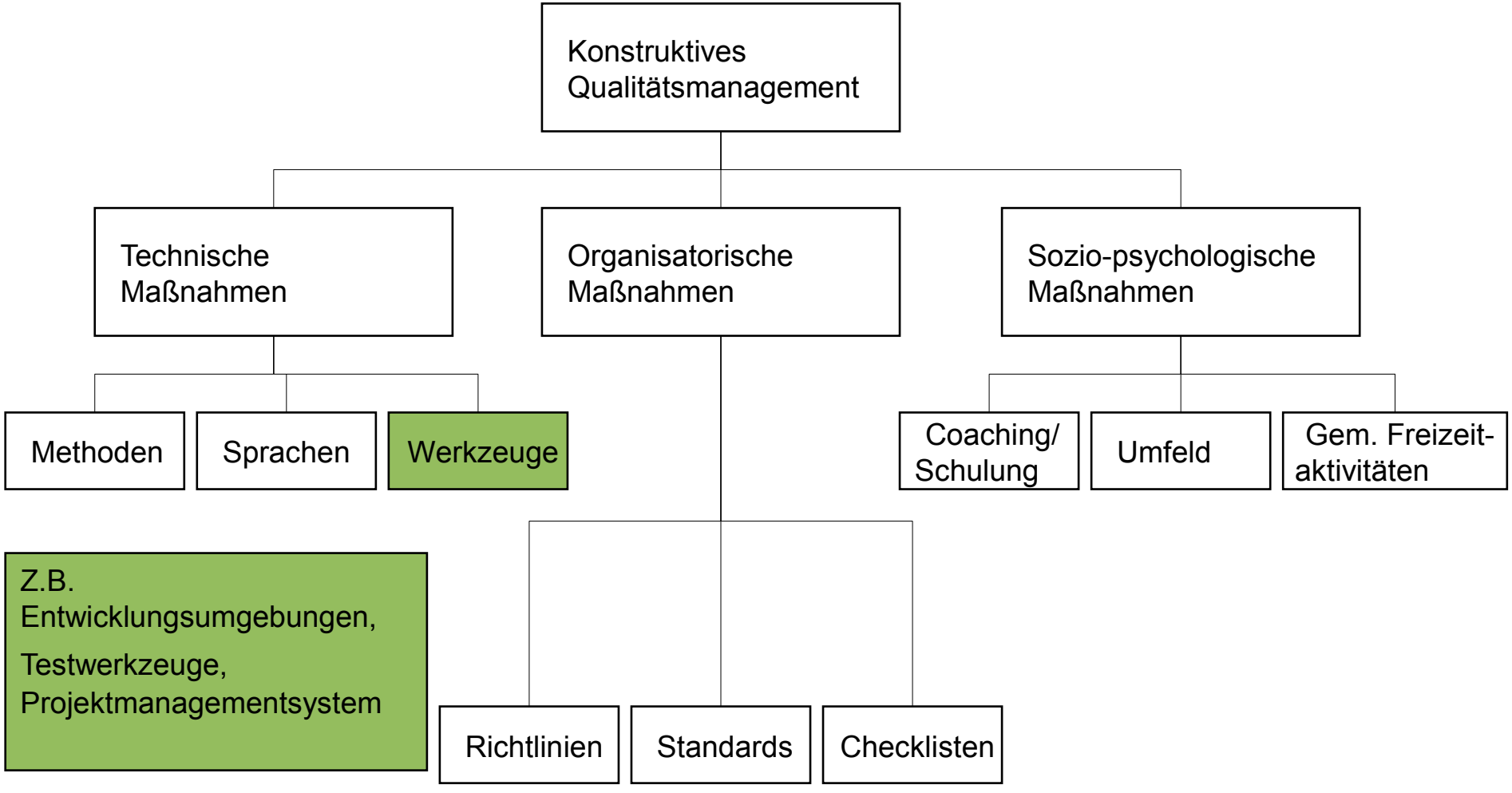


Einleitende Begriffe

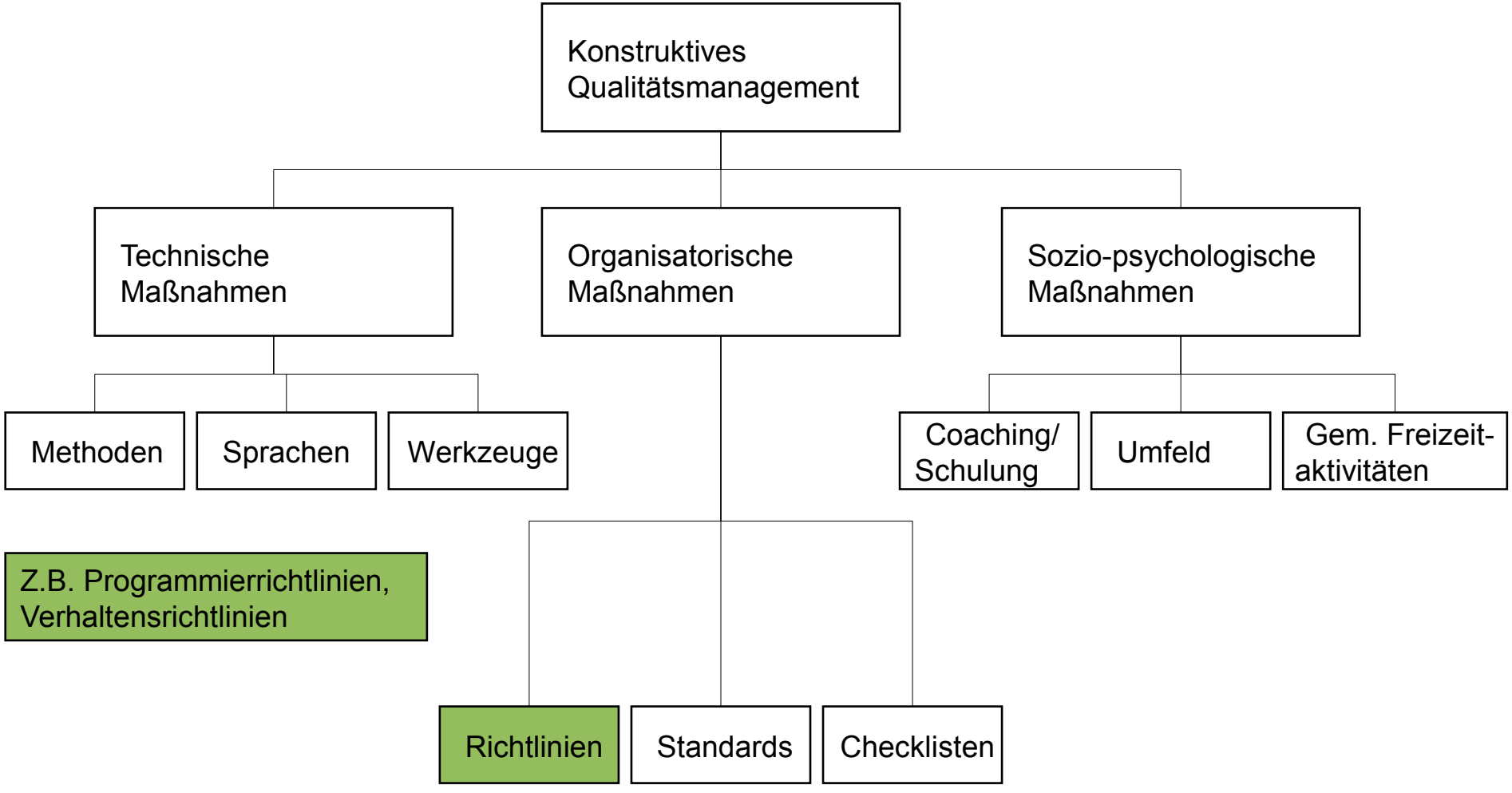
Übersicht über das konstruktive Qualitätsmanagement:



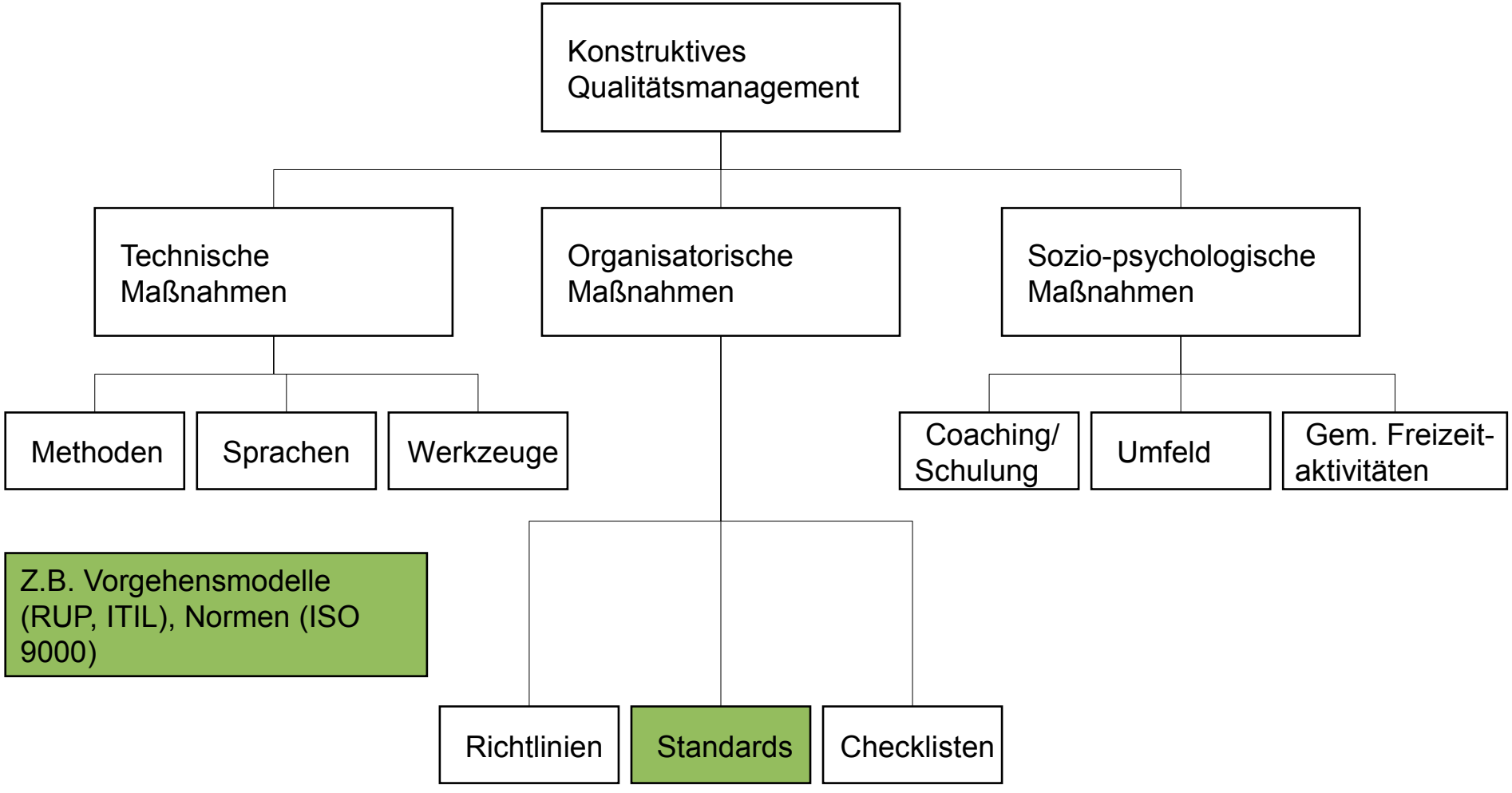
Übersicht über das konstruktive Qualitätsmanagement:



Übersicht über das konstruktive Qualitätsmanagement:

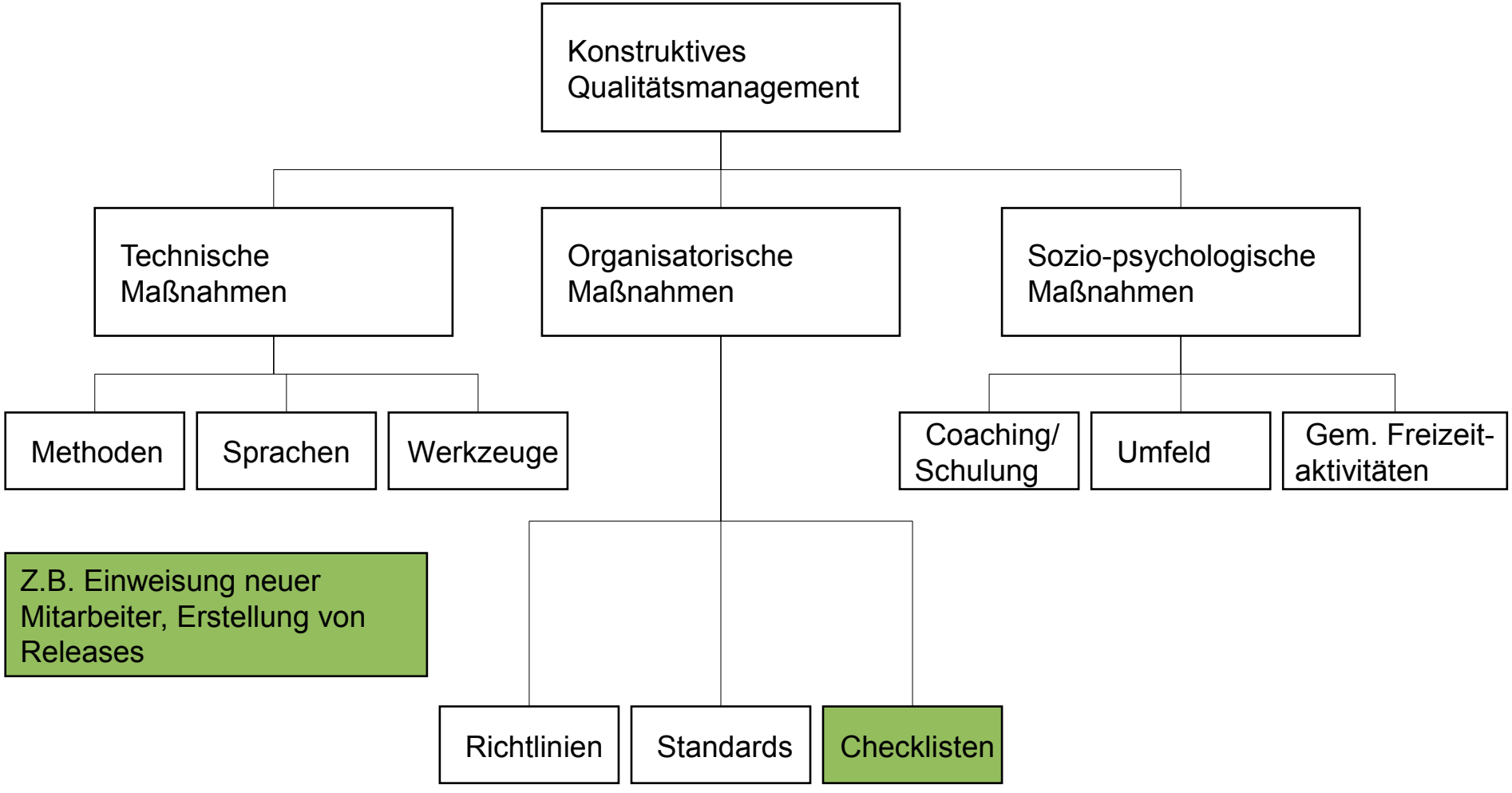


Übersicht über das konstruktive Qualitätsmanagement:



Einleitende Begriffe

Übersicht über das konstruktive Qualitätsmanagement:



Analytisches Qualitätsmanagement

- Ziel
  - Qualität von Software prüfen und bewerten (ex post)
- Voraussetzung
  - Vorherige Definition von Anforderungen zur Ableitung von Prüfzielen (traceability of requirements)

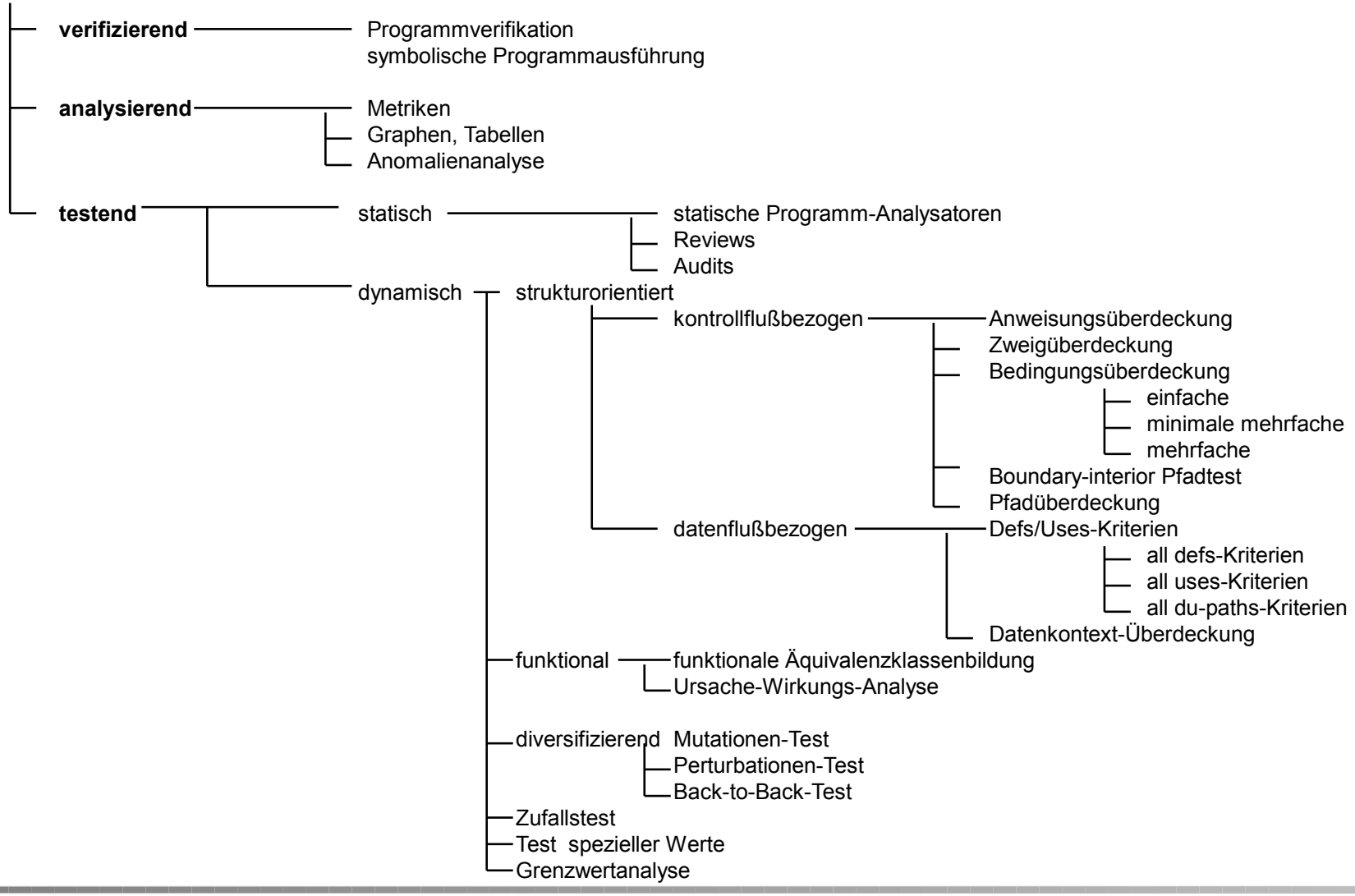
- Drei Klassen von Aktivitäten
  - Verifikation
    - Mathematischer Beweis der Korrektheit eines Programms auf Grundlage seiner formalen Spezifikation
  - Validation
    - Eignung eines Produktes bzgl. seines Einsatzzwecks
  - Tests

    - [IEEE Std 1012-1986: Standard for Software Verification and Validation Plans]

Analytisches Qualitätsmanagement

- Verifikation und Validation dienen dazu
  - Fehler so früh wie möglich zu entdecken und zu beseitigen
  - Projektrisiken und -kosten zu reduzieren
  - Die Produktqualität zu verbessern
  - Transparenz des Entwicklungsprozesses für das Management zu schaffen
  - Änderungen und deren Auswirkung auf Produktqualität, Kosten und Termine abzuschätzen

Übersicht Prüfverfahren

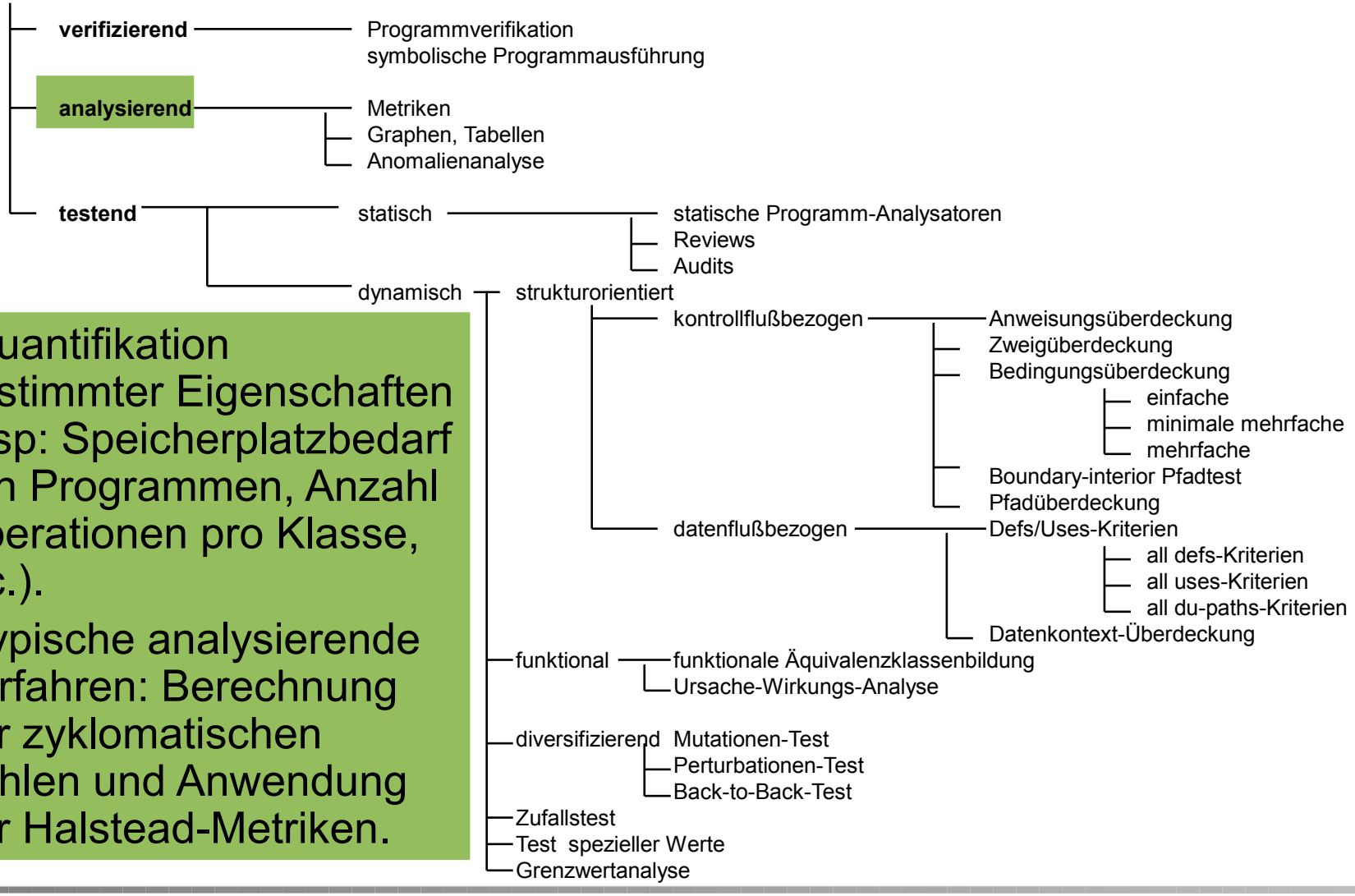


Übersicht Prüfverfahren

- **verifizierend**
  - Programmverifikation
  - symbolische Programmausführung
- **analysierend**
  - Metriken
  - Graphen, Tabellen
  - Anomalienanalyse
- **testend**
  - statisch
    - statische Programm-Analysatoren
    - Reviews
    - Audits
  - dynamisch
    - strukturorientiert
      - kontrollflußbezogen
        - Anweisungsüberdeckung
        - Zweigüberdeckung
        - Bedingungsüberdeckung
          - einfache
          - minimale mehrfache
          - mehrfache
        - Boundary-interior Pfadtest
        - Pfadüberdeckung
      - datenflußbezogen
        - Defs/Uses-Kriterien
          - all defs-Kriterien
          - all uses-Kriterien
          - all du-paths-Kriterien
        - Datenkontext-Überdeckung
    - funktional
      - funktionale Äquivalenzklassenbildung
      - Ursache-Wirkungs-Analyse
    - diversifizierend
      - Mutationen-Test
      - Perturbationen-Test
      - Back-to-Back-Test
    - Zufallstest
    - Test spezieller Werte
    - Grenzwertanalyse

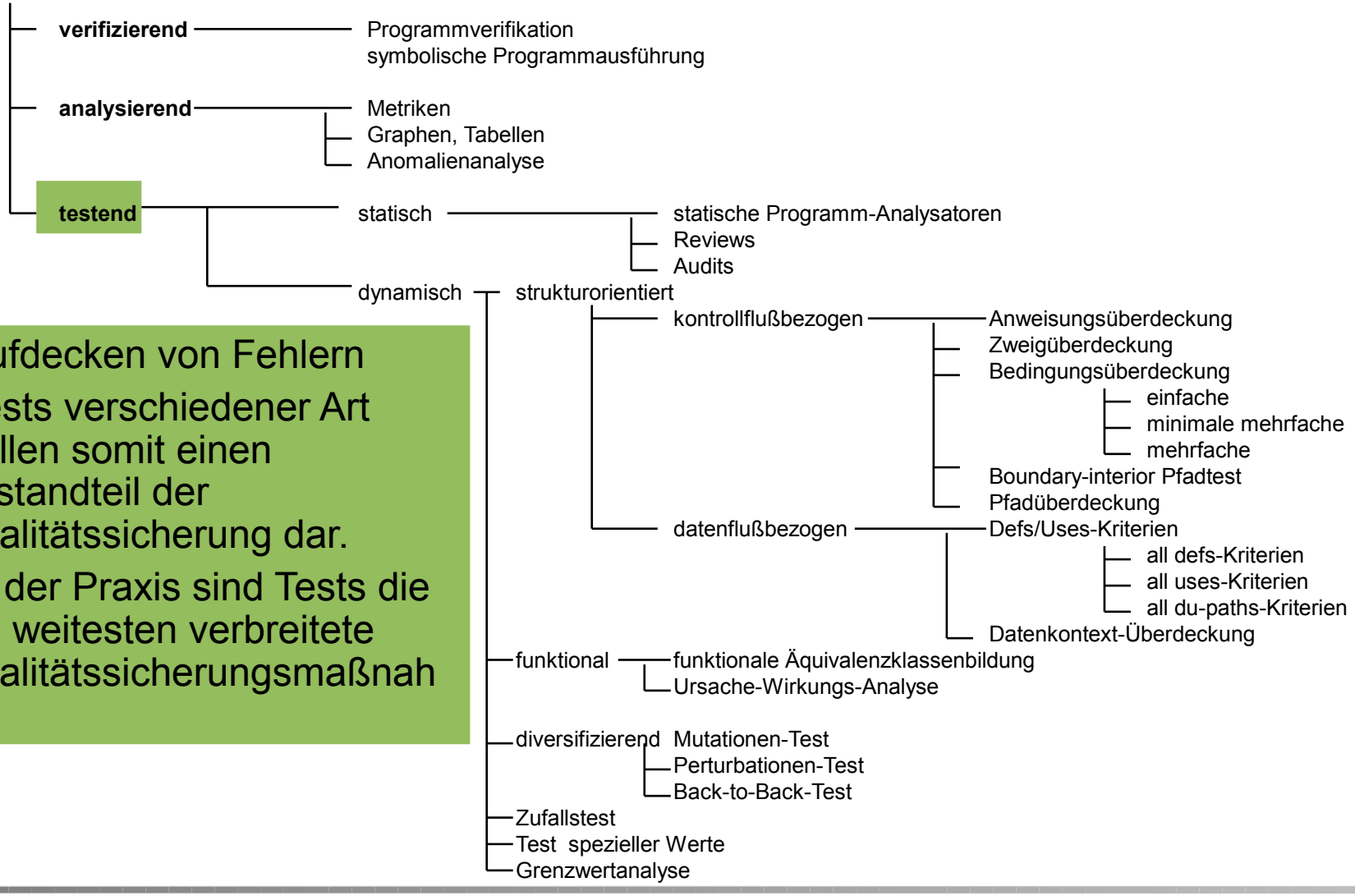
Beweis der Korrektheit eines Programms (Hinweis: Korrektheit wird gegen Anforderungsdokument und Spezifiktion geprüft!)

Übersicht Prüfverfahren



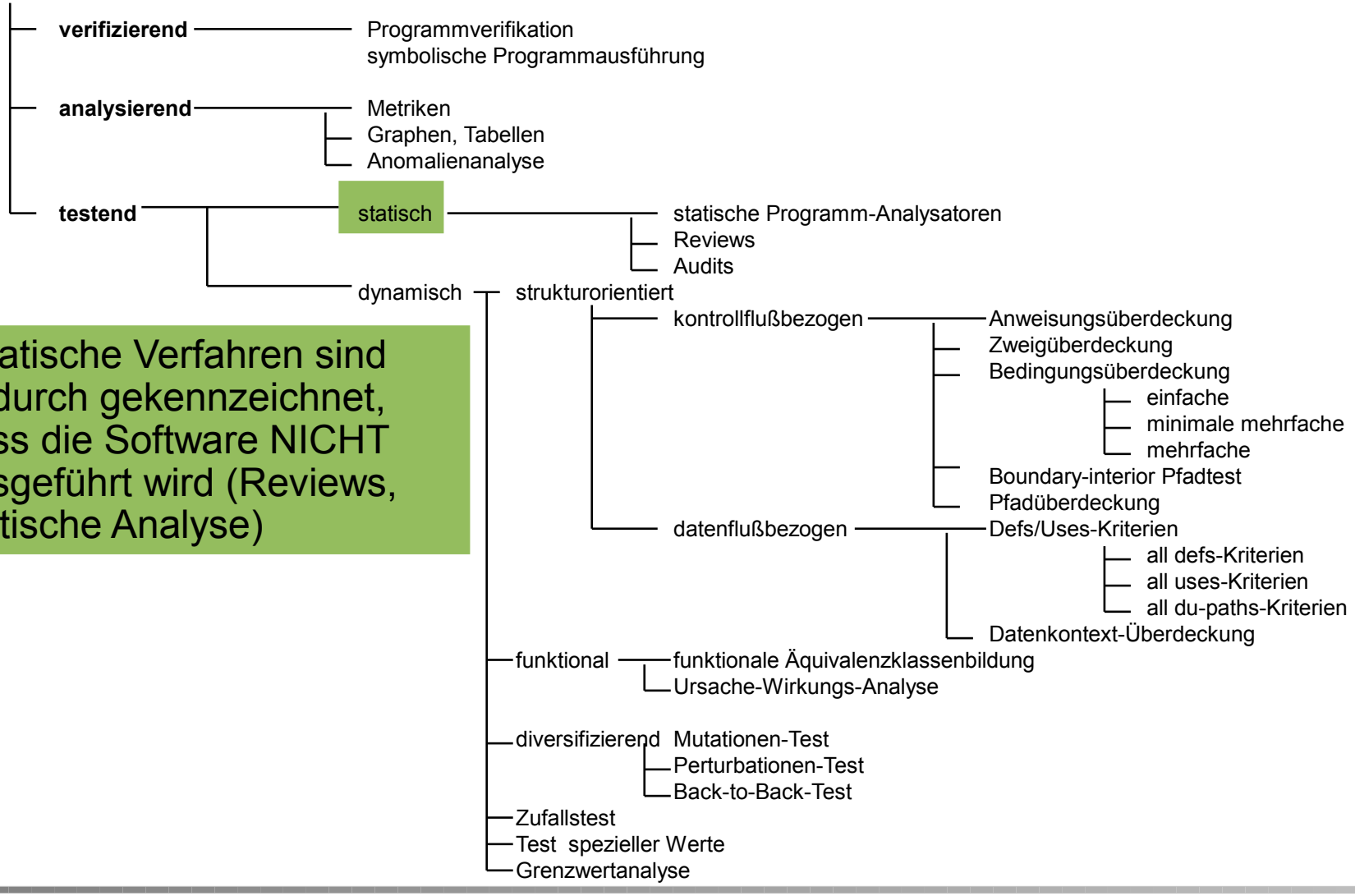
- Quantifikation bestimmter Eigenschaften (Bsp: Speicherplatzbedarf von Programmen, Anzahl Operationen pro Klasse, etc.).
- Typische analysierende Verfahren: Berechnung der zyklomatischen Zahlen und Anwendung der Halstead-Metriken.

Übersicht Prüfverfahren



- Aufdecken von Fehlern
- Tests verschiedener Art stellen somit einen Bestandteil der Qualitätssicherung dar.
- In der Praxis sind Tests die am weitesten verbreitete Qualitätssicherungsmaßnahme

Übersicht Prüfverfahren



- Statische Verfahren sind dadurch gekennzeichnet, dass die Software NICHT ausgeführt wird (Reviews, statische Analyse)

Übersicht Prüfverfahren

- **verifizierend**
  - Programmverifikation
  - symbolische Programmausführung
- **analysierend**
  - Metriken
  - Graphen, Tabellen
  - Anomalienanalyse
- **testend**
  - statisch
    - statische Programm-Analysatoren
    - Reviews
    - Audits
  - dynamisch
    - strukturorientiert
      - kontrollflußbezogen
        - Anweisungsüberdeckung
        - Zweigüberdeckung
        - Bedingungsüberdeckung
          - einfache
          - minimale mehrfache
          - mehrfache
        - Boundary-interior Pfadtest
        - Pfadüberdeckung
      - datenflußbezogen
        - Defs/Uses-Kriterien
          - all defs-Kriterien
          - all uses-Kriterien
          - all du-paths-Kriterien
        - Datenkontext-Überdeckung
    - funktional
      - funktionale Äquivalenzklassenbildung
      - Ursache-Wirkungs-Analyse
    - diversifizierend
      - Mutationen-Test
      - Perturbationen-Test
      - Back-to-Back-Test
    - Zufallstest
    - Test spezieller Werte
    - Grenzwertanalyse

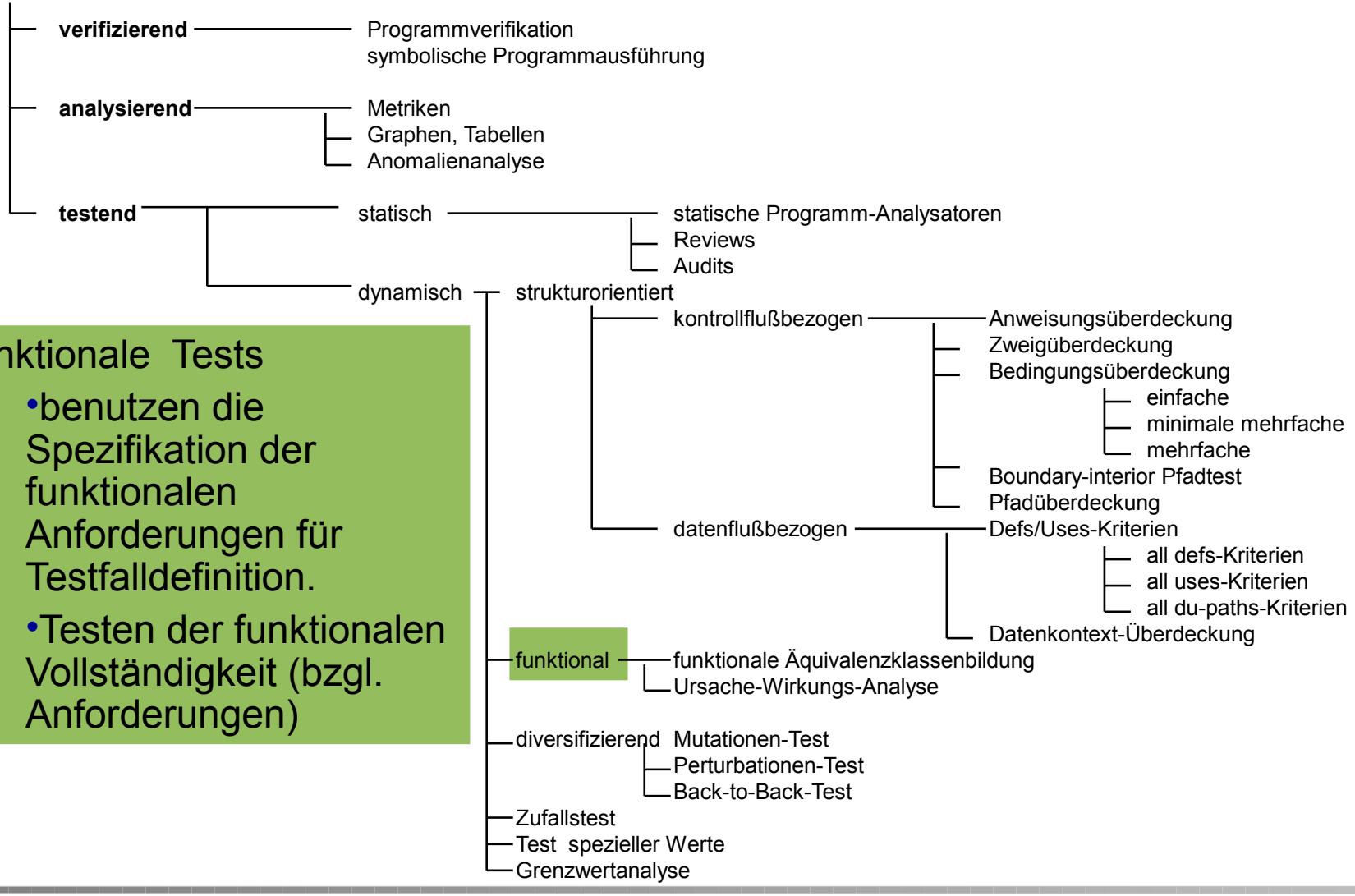
- Dynamische Verfahren setzen zur Fehlererkennung die Ausführbarkeit des Prüflings voraus

Übersicht Prüfverfahren

- **verifizierend**
  - Programmverifikation
  - symbolische Programmausführung
- **analysierend**
  - Metriken
  - Graphen, Tabellen
  - Anomalienanalyse
- **testend**
  - statisch
    - statische Programm-Analysatoren
    - Reviews
    - Audits
  - dynamisch
    - strukturorientiert
      - kontrollflußbezogen
        - Anweisungsüberdeckung
        - Zweigüberdeckung
        - Bedingungsüberdeckung
          - einfache
          - minimale mehrfache
          - mehrfache
        - Boundary-interior Pfadtest
        - Pfadüberdeckung
      - datenflußbezogen
        - Defs/Uses-Kriterien
          - all defs-Kriterien
          - all uses-Kriterien
          - all du-paths-Kriterien
        - Datenkontext-Überdeckung
    - funktional
      - funktionale Äquivalenzklassenbildung
      - Ursache-Wirkungs-Analyse
    - diversifizierend
      - Mutationen-Test
      - Perturbationen-Test
      - Back-to-Back-Test
    - Zufallstest
    - Test spezieller Werte
    - Grenzwertanalyse

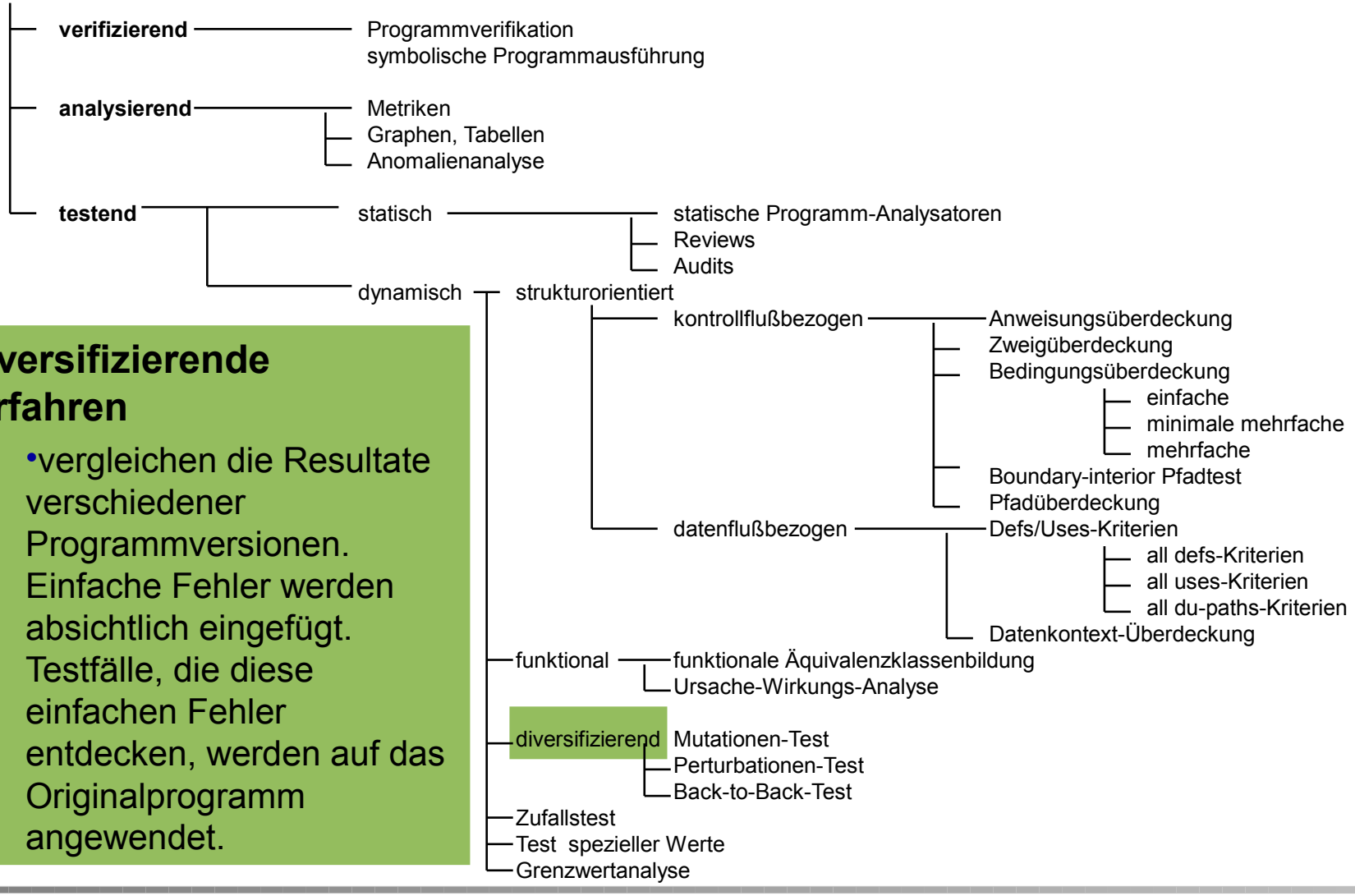
- strukturorientierte Tests
  - basieren auf der Struktur des Prüflings.
  - In Abhängigkeit von Struktur werden Testfälle abgeleitet und Testaktivitäten geplant.
  - Typisch: kontroll- und datenflussbezogene Ableitung der Testfälle

Übersicht Prüfverfahren



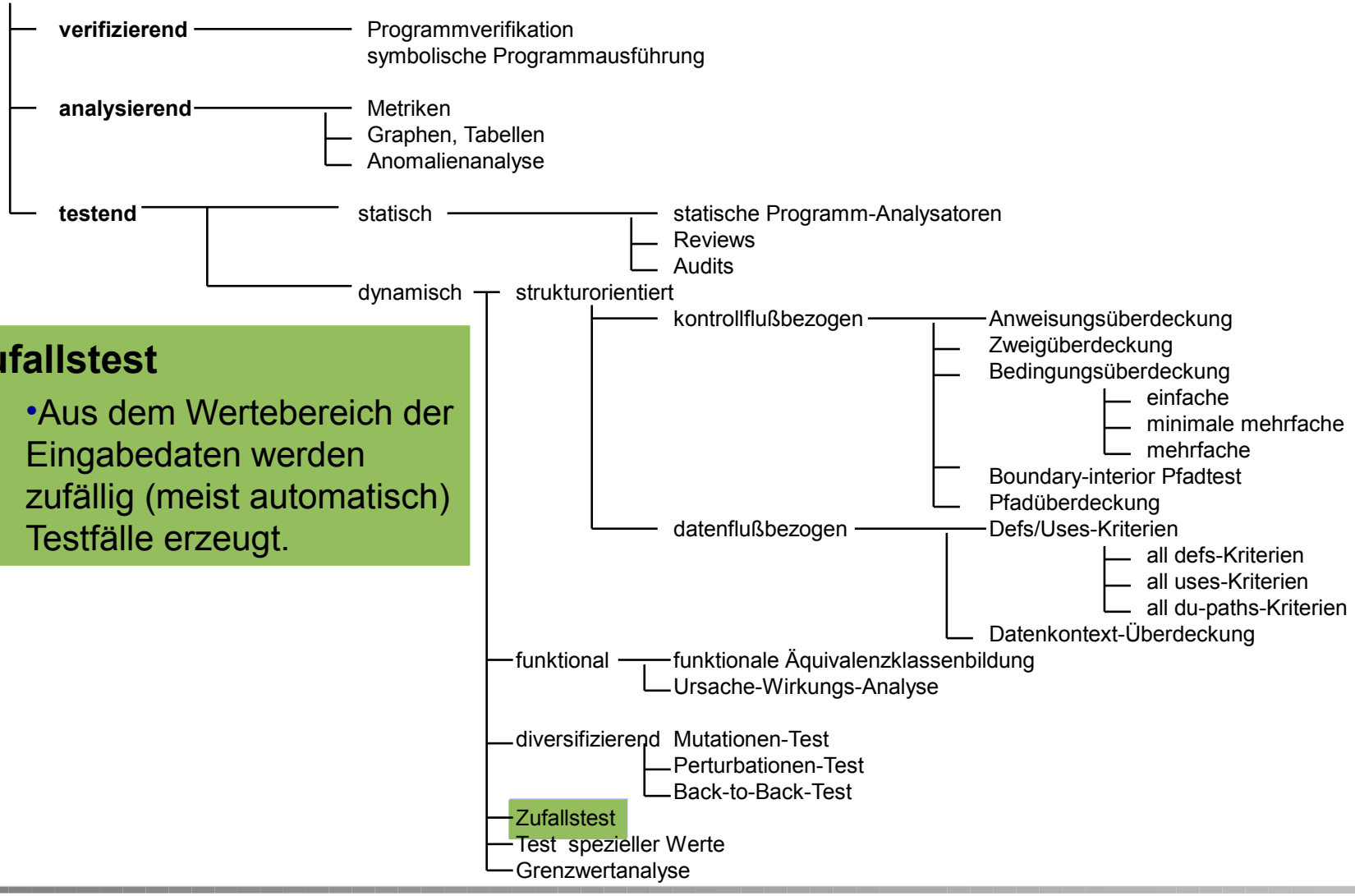
- funktionale Tests
  - benutzen die Spezifikation der funktionalen Anforderungen für Testfalldefinition.
  - Testen der funktionalen Vollständigkeit (bzgl. Anforderungen)

Übersicht Prüfverfahren



- **diversifizierende Verfahren**
  - vergleichen die Resultate verschiedener Programmversionen. Einfache Fehler werden absichtlich eingefügt. Testfälle, die diese einfachen Fehler entdecken, werden auf das Originalprogramm angewendet.

Übersicht Prüfverfahren



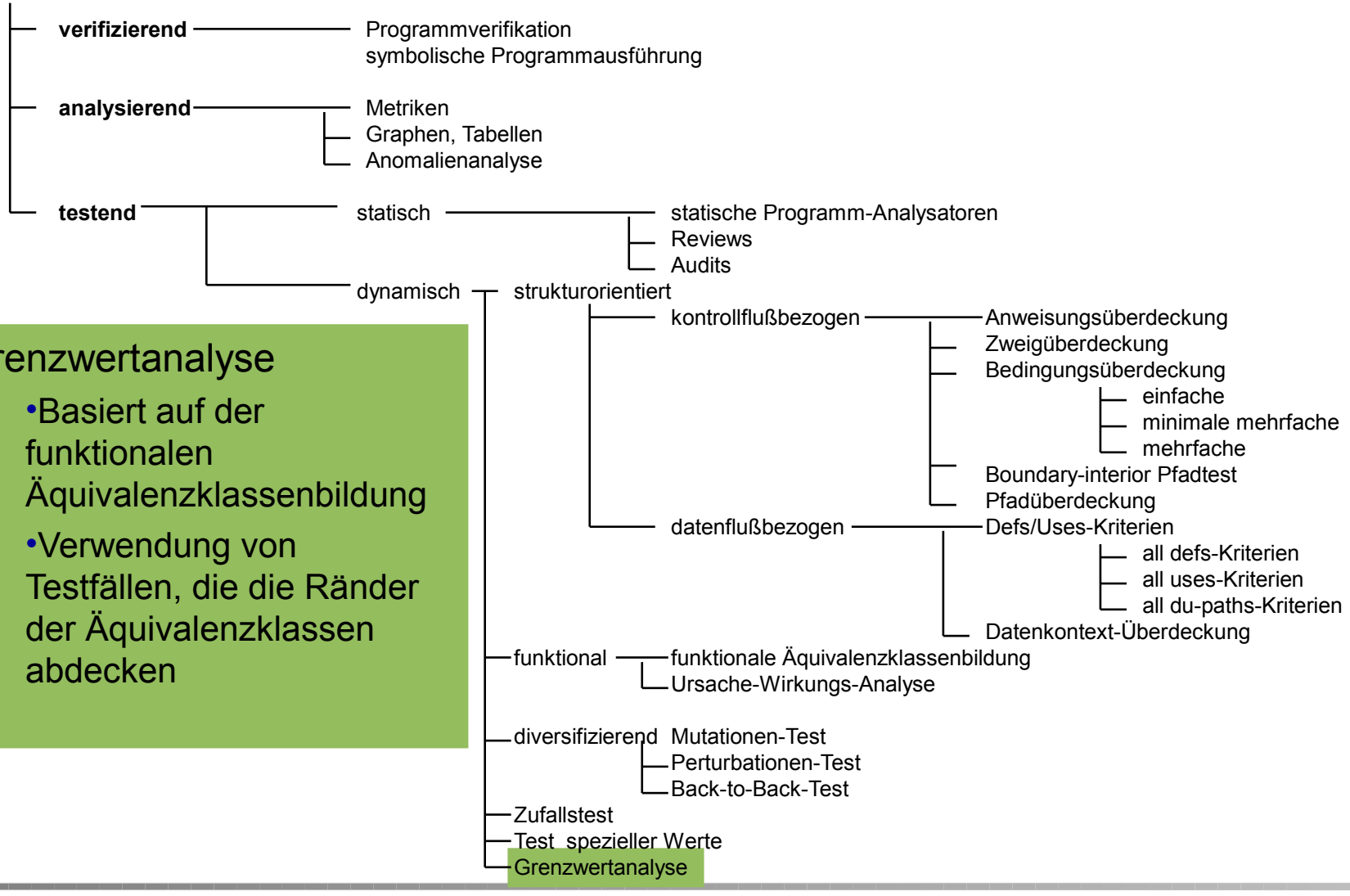
- **Zufallstest**
  - Aus dem Wertebereich der Eingabedaten werden zufällig (meist automatisch) Testfälle erzeugt.

Übersicht Prüfverfahren

- **verifizierend**
  - Programmverifikation
  - symbolische Programmausführung
- **analysierend**
  - Metriken
  - Graphen, Tabellen
  - Anomalienanalyse
- **testend**
  - statisch
    - statische Programm-Analysatoren
    - Reviews
    - Audits
  - dynamisch
    - strukturorientiert
      - kontrollflußbezogen
        - Anweisungsüberdeckung
        - Zweigüberdeckung
        - Bedingungsüberdeckung
          - einfache
          - minimale mehrfache
          - mehrfache
        - Boundary-interior Pfadtest
        - Pfadüberdeckung
      - datenflußbezogen
        - Defs/Uses-Kriterien
          - all defs-Kriterien
          - all uses-Kriterien
          - all du-paths-Kriterien
        - Datenkontext-Überdeckung
    - funktional
      - funktionale Äquivalenzklassenbildung
      - Ursache-Wirkungs-Analyse
    - diversifizierend
      - Mutationen-Test
      - Perturbationen-Test
      - Back-to-Back-Test
    - Zufallstest
    - Test spezieller Werte
    - Grenzwertanalyse

- **Test spezieller Werte**
  - Auswahl der Testdaten auf Basis von Erfahrung

Übersicht Prüfverfahren



- Grenzwertanalyse
  - Basiert auf der funktionalen Äquivalenzklassenbildung
  - Verwendung von Testfällen, die die Ränder der Äquivalenzklassen abdecken

Verifizierende Prüfverfahren

- Korrektheitsbeweise
  - Konsistenz eines Programms mit seiner Spezifikation wird mittels mathematischer Beweisverfahren nachgewiesen
  - Voraussetzung: formale Spezifikation
  - Prinzip: Mathematischer Nachweis, dass sich aus Vorbedingungen einer Operation bestimmte Nachbedingen ableiten lassen
  - Nachteile
    - Manueller Beweis fehleranfällig
    - Definition von assertions (Vor-, Nachbedingungen) oft schwierig

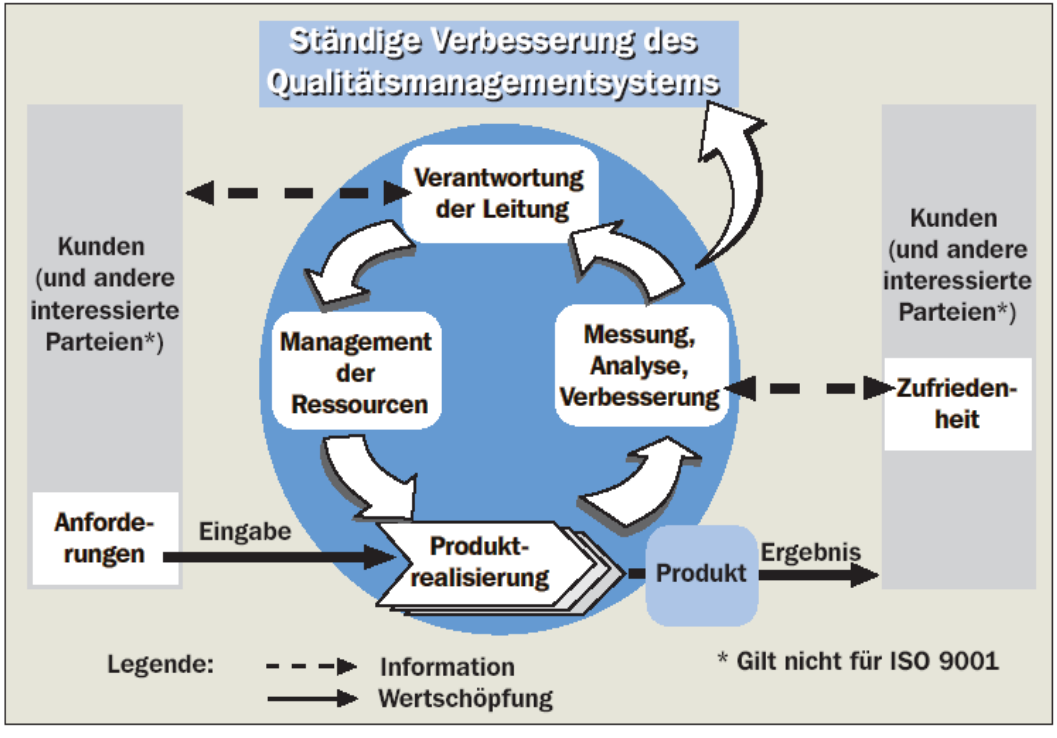
➢ Nur bei kritischen Programmteilen vorteilhaft

Verifizierende Prüfverfahren

- Symbolische Programmausführung
  - Methode, bei der statt aktuellen Datenwerten symbolische Ausdrücke (bestehend aus Variablennamen) zur Ausführung von Programmpfaden verwendet werden
  - Ergebnis: Baum, in dem jedes Blatt einem Pfad durch das Programm entspricht
  - Darstellbar als komplexer Ausdruck
  - Vergleich mit formaler Spezifikation
  - Nachteile:
    - Hoher Berechnungsaufwand

Analysierende Prüfverfahren

- Metriken
  - „To measure is to know“
  - Messen: Erfassung von Werten einer Kenngröße des Entwicklungs-, Pflegeprozesses oder Softwareprodukts mit Hilfe von Werkzeugen
  - Ziel: Kontrolle und Verbesserung von Prozessen und Produkten



Kernfrage: Was soll überhaupt gemessen werden?

- *I can only think of one metric that is worth collecting now and forever: defect count. Any organization that fails to track and type defects is running at less than its optimal level.*
- *There are many other metrics that are worth collecting for a while. Each time you introduce and begin collection a new metric, you need to put in place a mechanism to cease collecting that metric at some time in the future.*
- *Many of the most useful metrics should be collected only on a sampling basis.*

Alle aus [deM95]: T. de Marco, Why does software cost so much, Dorset House Publishing, 1995

Analysierende Prüfverfahren

- *When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager, unsatisfactory kind: it may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced to the stage of science.*

  (Lord Kelvin, 1891)
- *„Don‘t accept practices as „software engineering“ until they have been measured and proven.“,*

  R.B. Grady, Practical Software Metrics For Project Management and Process Improvement, Prentice-Hall, 1992
- *„It has been suggested that the distinction between a craft and an engineering discipline is that craftsmen use qualitative methods whereas engineering is based on quantitative techniques.“*

  Ian Sommerville, Software Engineering - Fourth Edition, Addison-Wesley, 1992

Analysierende Prüfverfahren

- Ein Maß ist eine Zuordnung einer Zahl oder eines Symbols zu einer Entität. Das Maß charakterisiert ein konkretes Attribut der Entität.
  - Unter einer **direkten Messung** versteht man eine Messung eines Attributs, für die keine zusätzlichen Attribute gemessen werden.
    - Bsp: LOC ist ein direktes Maß für die Länge eines Programms.
  - Unter einer **indirekten Messung** versteht man die Messung eines Attributs, das zur Ableitung eines anderen Attributs gebraucht wird.
    - Bsp: Das Messen von fan-in und fan-out für modulare Entwürfe ist indirekt, denn eigentlich wird versucht die Komplexität des Entwurfes zu messen. Mangels direkter Komplexitätsmaße wird fan-in und fan-out gemessen.
- Ein **Softwaremaß** ist die Zuordnung einer Zahl oder eines Symbols zu einem Ergebnis des Softwareprozesses oder zum Softwareprozess selbst.
- Unter einer **Softwaremetrik** verstehen wir ein Softwaremaß zusammen mit Bewertungsregeln.

Analysierende Prüfverfahren

- Konkret können Metriken zum Beispiel eingesetzt werden, um:
  - die schwachen Komponenten gemäß 80/20 Regel zu identifizieren
  - Schulungsmaßnahmen zu identifizieren
  - die sinnvolle Aufteilung von Aufwänden über Projektphasen zu prüfen
  - Risiken in einem bestimmten Projekt zu bewerten.

- Unsinnige Metriken können katastrophale Auswirkungen auf den gesamten Softwareprozess haben!

Analysierende Prüfverfahren

- Beispiele für Metriken im Bereich Testen
  - Anzahl der Fehler
  - Fehlerdichte (z.B. Fehler/Komponente)
  - Fehlerart
  - Fehlerquelle (Verursacher)
  - Fehlerschwere
  - Zeitpunkt des Auftretens

Testende Verfahren

- Testende Verfahren, haben das Ziel, Fehler zu erkennen
- Getestet werden sämtliche Dokumente, die bei der Softwareentwicklung entstehen.
- Prüflinge: Dokumente, die Testverfahren unterworfen werden
  - Dazu gehören die Produktdefinition, die Softwarespezifikation(en), der Quelltext, das ausführbare Programm und die Testdokumente selbst.
- Testen ist eine Kontrollfunktion im Softwareentwicklungsprozess. Sie umfasst nicht die Fehlerkorrektur.
- Debuggen ist ein Prozess, bei dem die Ursache eines Fehlers lokalisiert, dessen Korrektur überlegt, die Folgen der Korrektur geprüft und die Korrektur durchgeführt wird.
  - anschließend: erneutes Testen („Retesting“)

Testende Verfahren

- Phänomene rund um das Testen von Software
  - Testen in der Fertigungsindustrie versus Testen in der Software-Entwicklung
    - Industrie: Partieller Test -> 150 Teile Ausschuss auf 1.000 untersuchte Teile -> Hochrechnung: In 100.000 Teilen befinden sich 15.000 Teile Ausschuss. Aussortieren einer entsprechenden Menge von Ausschussteilen.
    - SW-Entwicklung: Partieller Test -> 150 Fehler -> Beseitigung von 150 Fehlern -> Hoffnung, dass Software fehlerfrei ist
  - Auslassen einzelner Tests ist KEIN Mittel, um slippage aufzuholen!
    - Phänomen des Hoffens auf ein Wunder
    - Verringerung des eigenen Handlungsspielraumes
    - Überstehen einer Deadline gefährdet das gesamte weitere Projekt

Testende Verfahren

- Zur Natur des Testens:
  - Der Entwickler ist nach der Kreativität fordernden Entwicklungsarbeit zunächst in aller Regel von der Korrektheit des erzeugten Programms überzeugt.
  - Der Test wird oft als lästige Pflicht aufgefasst, der unangenehmerweise weitere Arbeit mit sich bringt (Fehlerbeseitigung) und dazu den Glauben an die eigene „Unfehlbarkeit“ erschüttern könnte.
  - Das Finden von Fehlern ist somit ärgerlich und wird unterbewusst so weit wie möglich vermieden.
  - Werden übergeordnete Tests durch andere Personen durchgeführt, so wird es noch ärgerlicher, denn neben der neuen Arbeit sind es nun auch noch andere, die die eigene Fehlbarkeit entdecken.
  - Die allermeisten Versuche, die Fehlerentdeckung als Chance zur Verbesserung aufzufassen, scheitern an der Eitelkeit der betroffenen Personen.

Allgemeines und Prinzipien

- Wesentliche Begriffe:
  - Testfall
    - Testfälle umfassen Eingabewerte und die zu diesen Eingabewerten erwartete Ausgabewerte.
    - Umfasst also eine Menge von Testdaten
    - Testfälle werden während der gesamten Entwicklung entwickelt und vom Tester zusammenfassend überprüft.
    - Testfälle decken sowohl erlaubte als auch unerlaubte Eingaben ab.
    - Protokollierung der Anwendung eines Testfalls (Ergebnis, Plattform, Tester, Version, Datum)
    - Rückbezug von Testfällen zu Anforderungen (welche Anforderungen werden durch Tests überprüft, welche werden erfüllt)
    - Rückbezug von Testfällen zur Spezifikation (Test Driven Modelling)

- Wesentliche Begriffe:
  - Testorakel
    - Ermittelt zu einem Eingabedatum das Solldatum
      - Wird verwendet, wenn die Spezifikation keine passenden Aussagen zu Solldaten macht.

Allgemeines und Prinzipien

- Wesentliche Begriffe
  - Black box testing (auch funktionaler Testansatz genannt)
    - Testfälle und Testdaten werden durch Analyse der Spezifikation erzeugt ohne Berücksichtigung der internen Struktur des Programms
    - Güte der erstellten Testdaten ist abhängig von der semantischen Aussagekraft der Spezifikation und von Verfahren zur Auswahl der Testdaten
  - Bewertung
    + Spezifizierte, aber nicht implementierte Programmteile werden entdeckt
    + Aussagekräftige Spezifikation erlaubt systematisches Erstellen von Testdaten (z.B. zum Testen von formal definierten Integritätsbedingungen)
    + Wiederverwendung von Testdaten für verschiedene Plattformen
    - Implementierte, aber nicht spezifizierte Programmteile werden nicht entdeckt und nicht getestet
    - Wenig aussagekräftige Spezifikation führen zu unrepräsentativen Testdaten

- Wesentliche Begriffe
  - Beispiele für funktionale Testverfahren (für Black Box Testing)
    - Datenbereichsbezogenes Testen
    - Testen von funktionalen Formen (z.B. while-Schleife)
    - Entwurfsorientiertes Testen
    - Testen auf Basis von DFDs, Petri-Netzen, und endl. Automaten
    - Testen von Reihenfolgebedingungen
    - Testen auf Basis von algebraischen Spezifikationen
  - Spezifikationsorientiertes Testen
    - Umfasst funktionales Testen
    - Darüber hinaus: Testen nichtfunktionaler Eigenschaften (z.B. Einhaltung von Zeitbedingungen)

Allgemeines und Prinzipien

- Wesentliche Begriffe
  - White box testing (auch struktureller oder implementationsorientierter Testansatz genannt)
    - Testfälle und Testdaten werden durch strukturelle Analyse des Programms erzeugt
    - Kontroll- und Datenfluss ist in der Regel Informationsbasis
    - Auf diese Flüsse beziehen sich Überdeckungskriterien, die eine relevante Teilmenge aller möglichen Programmpfade definieren
    - Eine hinreichende Testdatenmenge ist dann eine Menge von Eingabendaten, die alle über das Überdeckungskriterium bestimmten Programmpfade ausführen
  - Bewertung
    + Automatische Tests möglich, da Programm Testbasis mit wohldefinierter Syntax und Semantik
    + Programme können ohne Spezifikation getestet werden
    + Implementierte, aber nicht spezifizierte Programmteile werden getestet
    - Spezifizierte, aber nicht implementierte Programmteile werden nicht entdeckt

- Wesentliche Begriffe

  - Beispiele für strukturelle Testverfahren
    - Kontrollflussbezogenes Testen
    - Datenflussbezogenes Testen
    - Ausdrucks-, anweisungs-, datenbezogenes Testen
    - Mutationsanalyse

Allgemeines und Prinzipien

- Wesentliche Begriffe
  - Spezifikations- und implementationsorientierte Testverfahren
    - Empfohlen: Einsatz mehrerer Testverfahren zur Erhöhung der Zuverlässigkeit der Tests
    - Mindestens ein Spezifikations- und ein implementationsorientiertes Testverfahren!
    - Vor- und Nachteile der Spezifikations- und implementationsorientierten Testverfahren können so ausgeglichen werden

- Abgrenzung der in der Übersicht genannten statischen / manuellen Testverfahren

	Inspektion formales Review	**Review**	**Walkthrough** abgeschwächtes Review
Ziele			
	schwere Defekte im Prüfobjekt identifizieren Entwicklungsprozeß verbessern Inspektionsprozeß verbessern Metriken ermitteln	Stärken und Schwächen des Prüfobjekts ermitteln (Entwicklungsprozeß verbessern)	Defekte und Probleme des Prüfobjektes identifizieren Ausbildung von Benutzern und Mitarbeitern
Teilnehmer			
	Moderator Autor Gutachter Protokollführer (Vorleser)	Moderator Autor Gutachter Protokollführer	Autor (Moderator) Gutachter

Legende: () optional

- Abgrenzung der in der Übersicht genannten statischen / manuellen Testverfahren

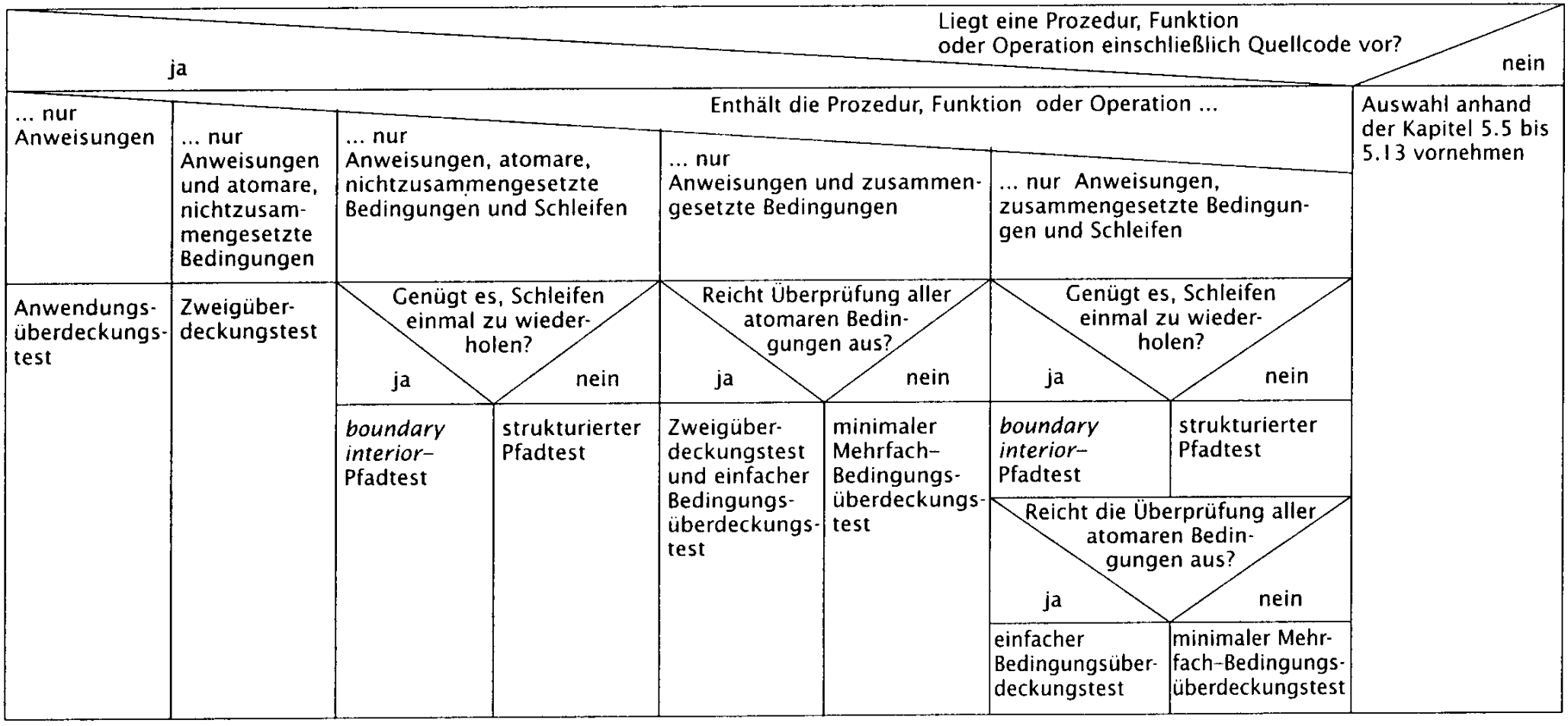
	Inspektion formales Review	**Review**	**Walkthrough** abgeschwächtes Review
Durchführung			
	Eingangsprüfung	(Eingangsprüfung)	
	Planung	Planung	
	(Einführungssitzung)	(Einführungssitzung)	
	indiv. Vorbereitung / Prüfung	inidiv. Vorbereitung, Prüfung	(indiv. Vorbereitung, Prüfung)
	Gruppensitzung	Gruppensitzung	Gruppensitzung
	Überarbeitung	Überarbeitung	
	Nachprüfung	Nachprüfung	
	Freigabe		
Referenzunterlagen			
	Ursprungsprodukt	Ursprungsprodukt	
	Erstellungsregeln	Erstellungsregeln	
	Checklisten	Fragenkataloge	
	Inspektionsregeln		
	Inspektionsplan		

- Abgrenzung der in der Übersicht genannten statischen / manuellen Testverfahren

Inspektion formales Review	**Review**	**Walkthrough** abgeschwächtes Review
Charakterisitika		
ausgebildeter Moderator Prüfobjekt wird vom Vorleser Absatz für Absatz vorgetragen Moderator gibt Freigabe	ausgebildeter Moderator Prüfteam gibt Empfehlung an Manager	Prüfobjekt wird vom Autor aublauforientiert vorgetragen Autor entscheidet

Kontrollflussbezogene Testverfahren

- Auswahl eines geeigneten Testverfahrens



Kontrollflussbezogene Testverfahren

- Kontrollflussbezogene Testverfahren
  - sind white-box-Verfahren, denn sie nutzen die Kenntnis der Programmstruktur aus.
  - Sie sind dynamische Verfahren, denn sie basieren auf der Ausführung des Programms mit verschiedenen Testdaten.
- Ziel
  - Testdaten so wählen, dass möglichst viele Durchläufe durch den Kontrollfluss des Programms getestet werden und dass dafür möglichst wenig Testfälle gebraucht werden.
  - Die Varianten der kontrollflussbezogenen Verfahren differenzieren nach angestrebtem Überdeckungsgrad und nach Art und Weise, in der der Begriff der Überdeckung verstanden wird.
- Analysemittel
  - Kontrollflussgraph: Verdeutlicht den Kontrollfluss in einem Programm

Kontrollflussbezogene Testverfahren

- Der Kontrollflussgraph eines Programms P ist ein gerichteter Graph

  G= (N,E)

  mit 2 ausgezeichneten Knoten *nstart,nfinal*

  - Ein Knoten stellt Anweisungen (oder strikt sequenzielle Anweisungsfolgen dar).
  - Eine Kante aus der Menge der Kanten E $\subseteq$ NxN beschreibt den möglichen Kontrollfluss zwischen zwei Anweisungen. Eine Kante aus E wird auch Zweig genannt.
  - Die beiden ausgezeichneten Knoten stellen die Anfangs- und Endeanweisung eines Programms dar.
  - Beispiel

Kontrollflussbezogene Testverfahren

- Auswertung eines Ziffernstrings und Ermittlung des dargestellten Wertes

```
PROCEDURE WerteZiffernfolgeAus (inZiffernString: Ttext) : REAL;
TYPE TWoBinIch = (VordemKomma, NachdemKomma);
    Tziffern = SET OF [`0`, ..., `9`]
CONST Cfehlercode = -1.0;
     Cziffern = Tziffern {`0`, ..., `9`}
VAR Zchn : CHAR
   Wert, Genauigkeit : REAL;
   Position: CARDINAL ;
   WoBinIch: TWoBinIch ;
   Fehlerfrei : Boolean;
BEGIN
     Wert:=0.0;
     Genauigkeit:=1.0;
     WoBinIch:= VorDemKomma;
     Fehlerfrei:=TRUE;
     Position:=1;
     ...
```

Kontrollflussbezogene Testverfahren

...

```
WHILE (Position <=TextVw.Length(inZiffernString))
        AND Fehlerfrei DO
        Zchn:=TextVW.LiesZeichen (inZiffernString,Position);
        IF Zchn IN Cziffern THEN
                IF WoBinIch=NachDemKomma THEN
                    Genauigkeit:= Genauigkeit / 10.0;
                END; (* IF *)
                Wert:= 10.0 *Wert + Konvertiere (Zchn)
        ELSIF (Zchn=´.´) AND (WoBinIch=VorDemKomma) THEN
                WoBinIch:=NachDemKomma
        ELSE
                Fehlerfrei:=FALSE;
        END; (* IF *)
        INC (Position);
END; (*WHILE*)
...
```

Kontrollflussbezogene Testverfahren

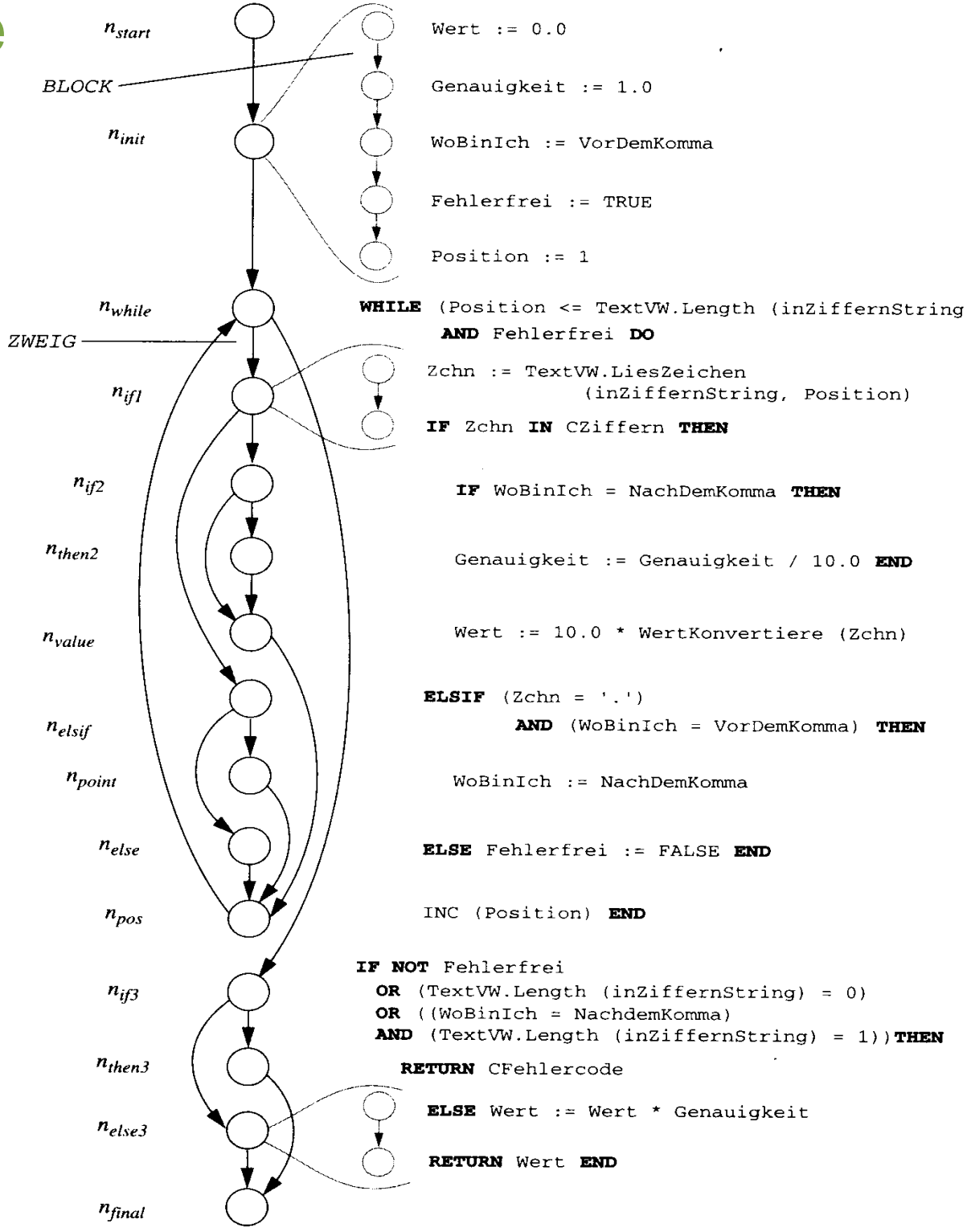
...

```
        IF NOT Fehlerfrei
        OR (TextVW.Length (inZiffernString) =0);
        OR (WoBinIch = NachDemKomma)
        AND (TextVwLenght (inZiffernString) = 1)) THEN
                RETURN Cfehlercode
                (* -- ILLEGALE ZEICHENFOLGE -- *)
        ELSE
                Wert:=Wert * Genauigkeit;
                RETURN Wert
        END (* IF *)
END WerteZiffernFolgeAus;
```

Kontrollflussbezogene Testverfahren

Durchspielen an der Ziffernfolge 28.9

- inZiffernString: 28.9
- Wert:
- Genauigkeit:
- WoBinIch:
- Fehlerfrei:
- Position:
- Zchn:



Kontrollflussbezogene Testverfahren

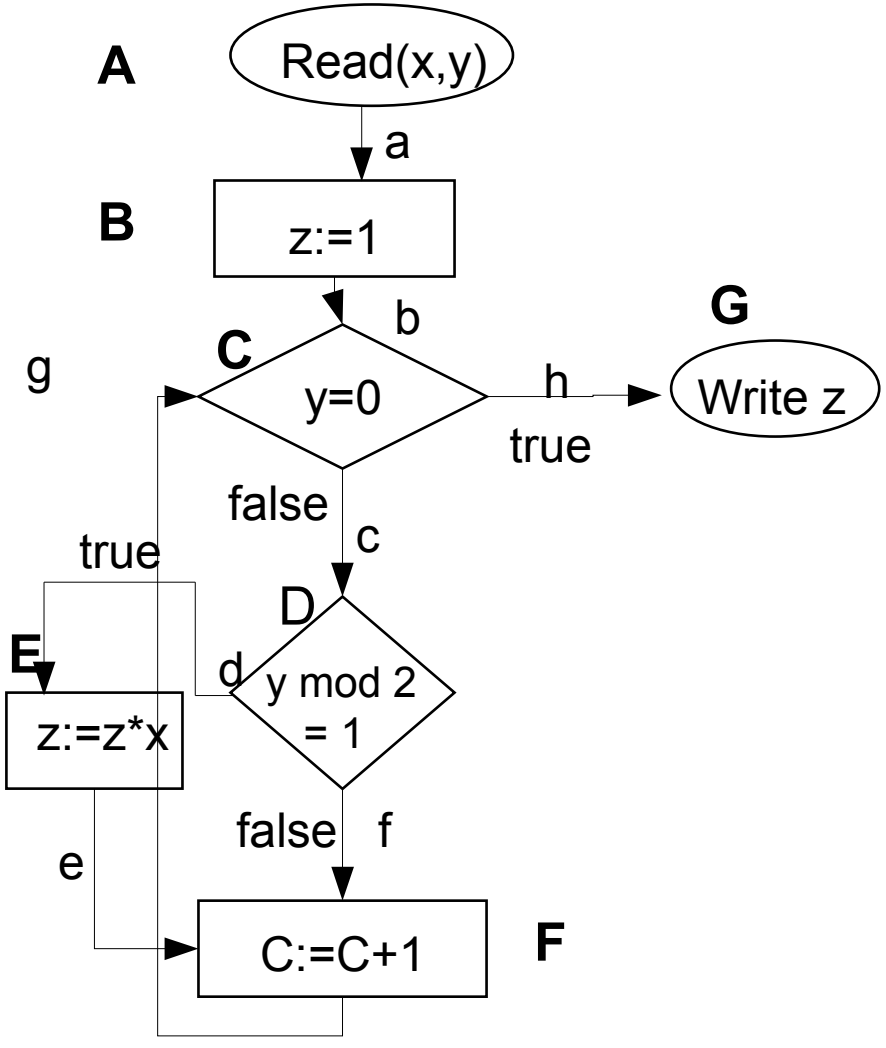
- Der Kontrollflussgraph eines Programms P
  - Unter einem Block verstehen wir eine nichtleere Folge von Knoten,
    - die ausschließlich durch den ersten Knoten betreten werden kann,
    - die - sobald der erste Knoten durchlaufen wird - in der vorgegebenen Reihenfolge deterministisch genau einmal durchlaufen wird,
    - die bezüglich der ersten beiden Eigenschaften maximal ist.
  - Pfad/vollständiger Weg
    - Folge von Knoten und Kanten, die mit dem Startknoten beginnt und mit dem Endeknoten endet
  - Wege (T,P)
    - Ist die Menge der vollständigen, endlichen Wege w des Kontrollflussgraphen (des Programms P), für die es ein Testdatum t aus der Menge der Testdaten T gibt, das den Weg w ausführt.
  - Ein Entscheidungsknoten ist ein Knoten mit mindestens 2 Nachfolgerknoten.

Kontrollflussbezogene Testverfahren

- Der Kontrollflussgraph eines Programms P
  - Ein Zyklus ist ein Weg im Kontrollflussgraphen mit mindestens zwei Knoten, der an demselben Knoten beginnt und endet.
  - Ein einfacher Zyklus ist ein Zyklus, bei dem alle Knoten (außer Anfangs- und Endknoten) verschieden sind.

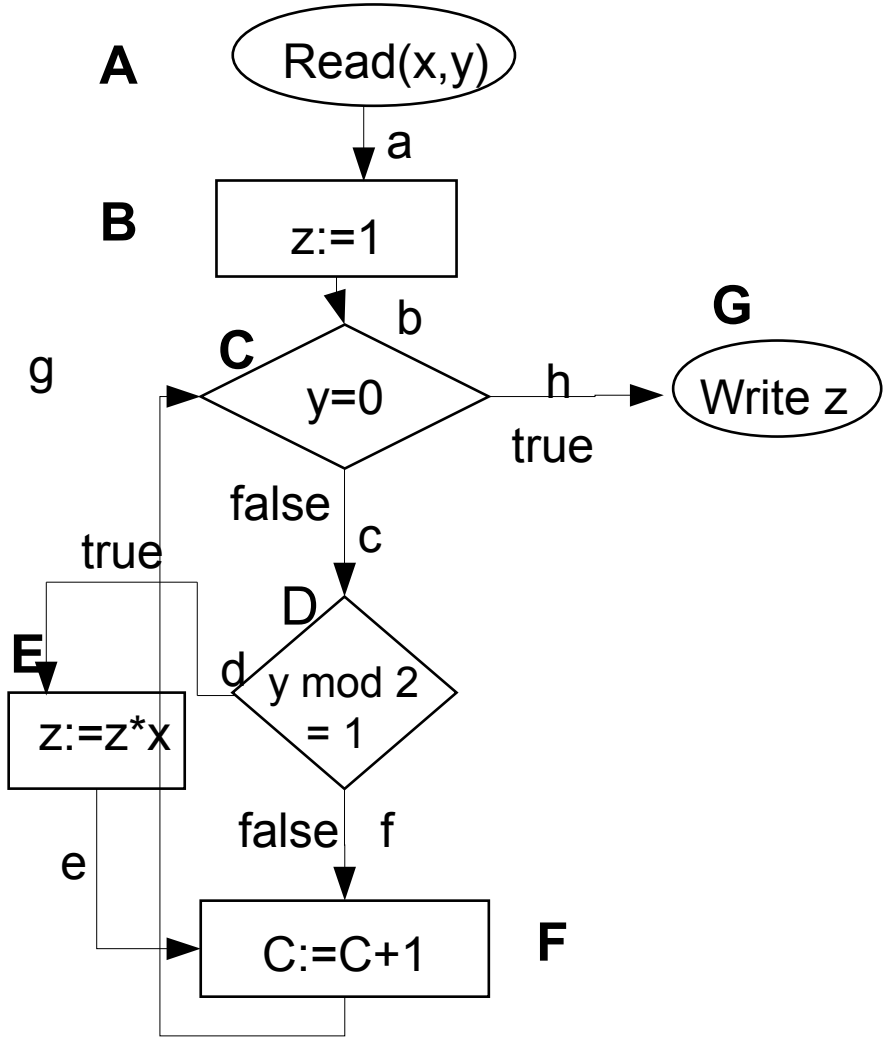
Kontrollflussbezogene Testverfahren

- Ein **Entscheidungs-Entscheidungsweg** ist ein Wegstück, welches bei einem **Entscheidungsknoten** oder dem **Anfangsknoten** beginnt und alle folgenden Knoten und Kanten bis zum nächsten **Entscheidungsknoten** bzw. bis zum **Endknoten** des Kontrollflussgraphen (einschließlich) enthält.
- Hier: {A,B,C}; {C,G}; {C,D}; {D,E,F,C}; {D,F,C}



Kontrollflussbezogene Testverfahren

- Ein **Segment** ist ein Wegstück mit den Eigenschaften
  - Der **erste Knoten** des Wegstücks ist der Anfangsknoten des Kontrollflussgraphen oder ein Entscheidungsknoten oder ein Vereinigungsknoten (mehrere Inputs).
  - Der **letzte Knoten** des Wegstücks ist der Endknoten des Kontrollflussgraphen oder ein Entscheidungsknoten oder ein Vereinigungsknoten
  - Alle **andere Knoten** haben nur eine Eingangs- und eine Ausgangskante
- Hier: {A,B,C}; {C,G}; {C,D}; {D,E,F}; {D,F}; {F,C}



Kontrollflussbezogene Testverfahren

- Fehler, die durch kontrollflussbezogene Verfahren aufgedeckt werden können
  - Berechnungsfehler
    - Sind Fehler, bei denen zwar der richtige Kontrollflussweg im Programm ausgeführt wird, aber mindestens **ein berechneter Variablenwert falsch** ist.
  - Bereichsfehler
    - Sind Fehler, bei denen **nicht der richtige Kontrollflussweg** im Programm ausgeführt wird. Dies liegt daran, dass der Eingabebereich (Menge aller Eingabewerte, die einen Weg ausführen) des vorliegenden Kontrollflussweges nicht mit dem Eingabebereich des korrekten Kontrollflussweges im korrekten Programm übereinstimmt.
  - Unterbereichsfehler
    - Sind (spezielle Bereichs-)Fehler, bei denen **ein Kontrollfluss zu wenig oder zu viel** im Programm vorkommt, d.h. es fehlt eine Abfrage oder es gibt eine zusätzliche, falsche Abfrage.

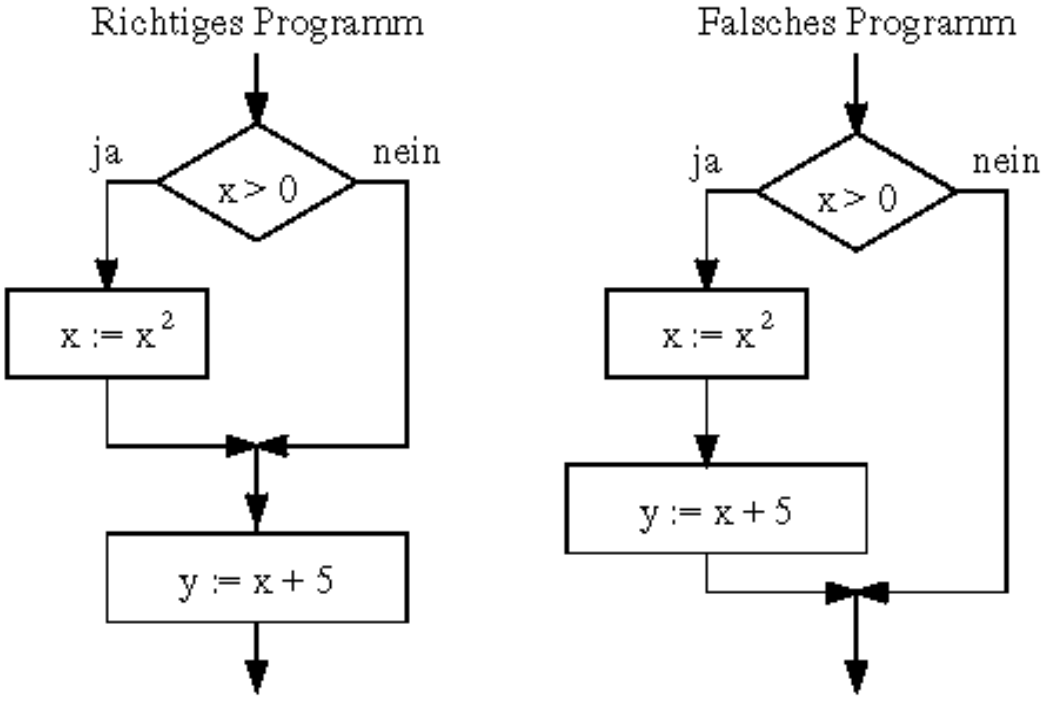
Kontrollflussbezogene Testverfahren

- Anweisungsüberdeckung (C_0-Test) (C für Coverage)
  - Eine Testdatenmenge T erfüllt die C_0-Überdeckung für Programm P genau dann, wenn es für jede Anweisung A des Programms P ein Testdatum t aus T gibt, das die Anweisung A ausführt.
  - Eine Anweisung A wird unter T ausgeführt/überdeckt, genau dann wenn der zur Anweisung A gehörende Knoten k in einem Weg von Wege(T) vorkommt.
  - Als Testwirksamkeitsmaß TWM_0 wird der Anweisungsüberdeckungsgrad definiert (Verhältnis der besuchten Knoten zur Gesamtzahl von Knoten)

$$TWM_0 \stackrel{def}{=} \frac{\text{Zahl der unter T überdeckten Anweisungen}}{\text{Zahl aller Anweisungen}}$$

Kontrollflussbezogene Testverfahren

- Anweisungsüberdeckung (C_0-Test): Beispiel



Testdatum

T1: Eingabedatum: x=5,y=0, Solldatum: x=25, y=30

erfüllt zwar die C_0-Überdeckung, deckt aber nicht den Fehler im falschen Programm auf

- Anweisungsüberdeckung (C_0 -Test)
  - Bewertung
    - Geringe Zahl von Eingabedaten
    - Notwendiges, aber kein hinreichendes Testkriterium
    - Mangelhafte Aussagekraft: nicht-ausführbare Programmteile werden sicher entdeckt, alle anderen Fehler werden nur zufällig entdeckt

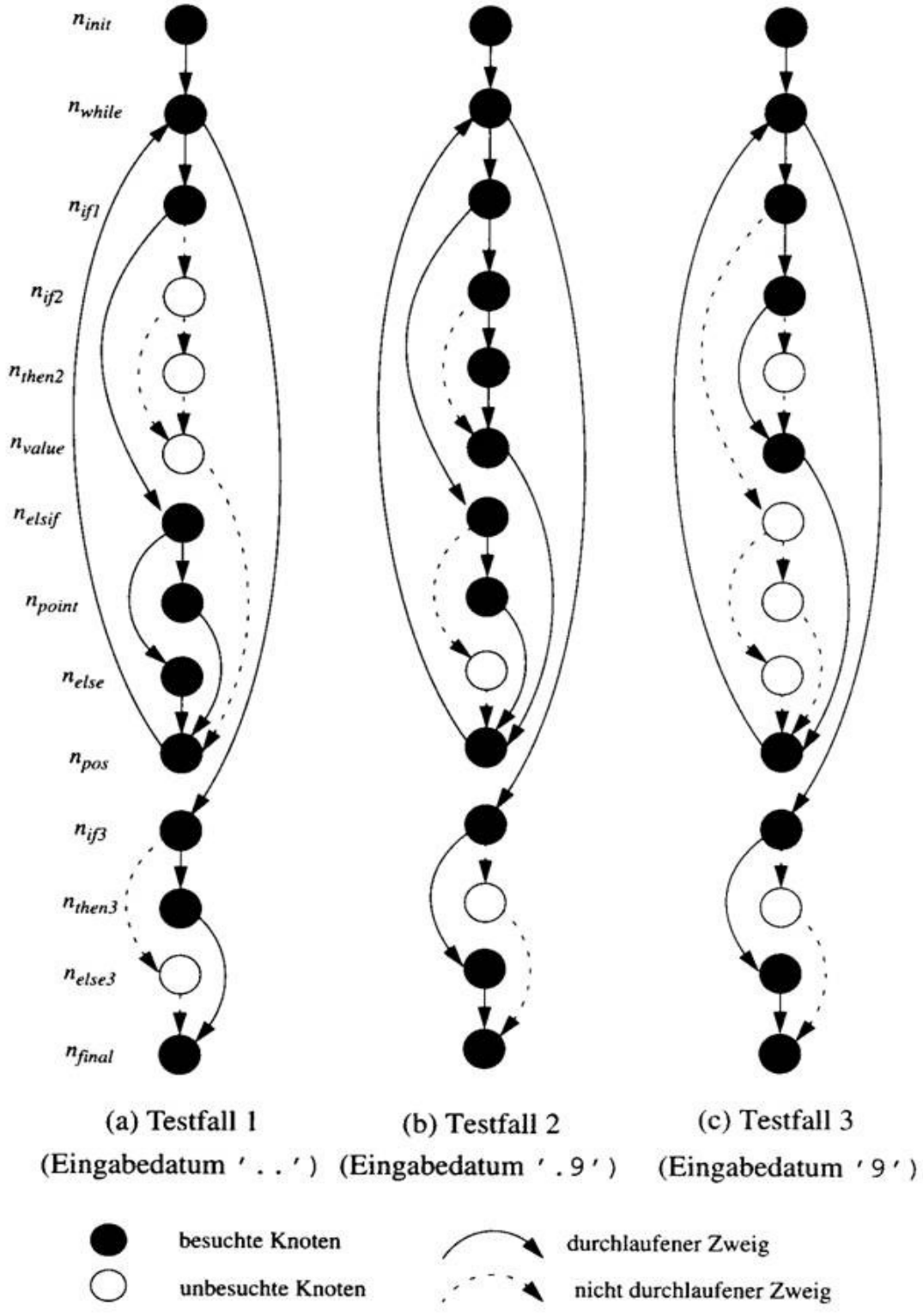
Kontrollflussbezogene Testverfahren

- Zweigüberdeckung (C_1-Test)
  - strenger als Anweisungsüberdeckung, alle durch Selektion oder Iteration bedingten Verzweigungen im Kontrollfluss sind mindestens einmal zu verfolgen.
  - Eine Testdatenmenge T erfüllt die C_1-Überdeckung für Programm P, genau dann wenn es für jede Kante k im Kontrollflussgraphen von P einen Weg in Wege(T,P) gibt, zu dem k gehört
  - Als Testwirksamkeitsmaß C_{Zweig} wird der Zweigüberdeckungsgrad definiert (Verhältnis der besuchten Zweige zur Gesamtzahl von Zweigen)

$$C_{Zweig} \stackrel{def}{=} \frac{\text{Anzahl der besuchten Zweige}}{\text{Gesamtanzahl der Zweige}}$$

  - Betrachtung der beiden Testfälle aus der Anweisungsüberdeckung:
    - nur die Kante (nif2, nvalue) wurde nicht durchlaufen
    - der Pfad (nstart, ninit, nwhile, nif1, nif2, nvalue, npos, nwhile, nif3, nelse3, nfinal) enthält diese Kante
    - verwendbares Eingabedatum: ‚9‘

Kontrollflussbezogene Testverfahren



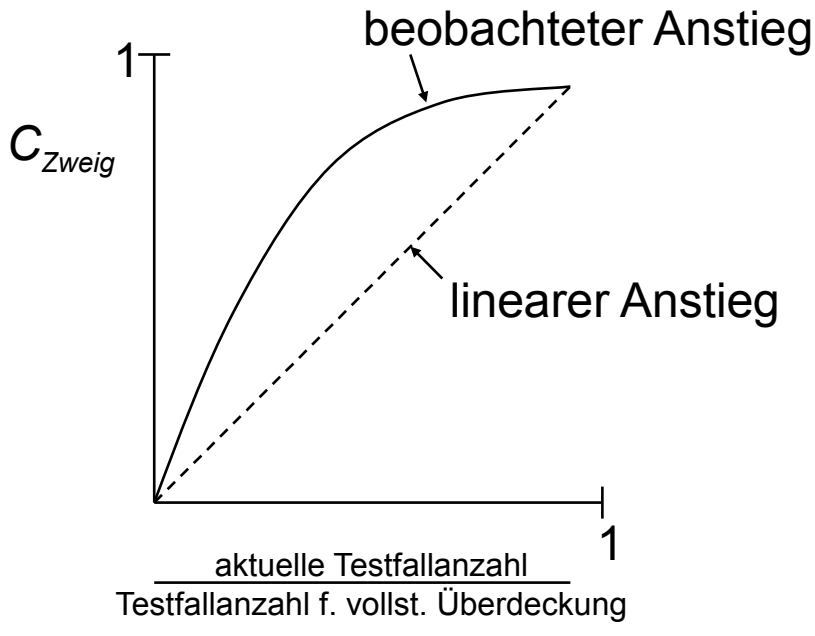
- Zweigüberdeckung (C_1-Test)
  - Bewertung
    - geringe Zahl von Eingabedaten
    - Aussagekraft besser als C_0: nicht-ausführbare Knoten und Zweige werden sicher entdeckt, alle anderen Fehler werden nur zufällig entdeckt
    - Oft durchlaufene Programmteile können erkannt und ggf. optimiert werden
    - Unzureichend für den Test von Schleifen
      - Pfadüberdeckungstest
    - Abhängigkeiten zwischen Zweigen nicht berücksichtigt
      - Pfadüberdeckungstest
    - Kein Test komplexer, zusammengesetzter Bedingungen
      - Bedingungsüberdeckungstest
    - empirische Untersuchungen ergeben, dass auch bei 100%iger Abdeckung nur 25% der Fehler erkannt werden, dennoch: besser als völlig unsystematisches Testen!

Kontrollflussbezogene Testverfahren

- Zweigüberdeckungsrate
  als Funktion der Testfallanzahl [Lig90]
  (gilt in ähnlicher Weise für andere Überdeckungsmaße)

Beobachteter Zusammenhang!

Keine streng mathematische Notwendigkeit



Was bedeutet das?

Kontrollflussbezogene Testverfahren

- Bedingungsüberdeckungen
  - berücksichtigen Bedingungen in den Schleifen und Auswahlkonstrukten des Bausteins zur Definition von Tests
  - Annahme: es reicht nicht, dass zusammengesetzte Bedingungen (sog. Entscheidungsprädikate) nur einmal zu True und einmal zu False ausgewertet werden, sondern es müssen die Teilbedingungen (atomare Prädikatterme) variiert werden
  - Beispiel
    - if A>1 and (B=2 or C>1) and D then ... else ...
    - Entscheidungsprädikat: A>1 and (B=2 or C>1) and D
    - Atomare Prädikatterme: A>1, B=2, C>1, D
  - Anderes Beispiel: Vergleich der Zweigüberdeckung von if (a AND b AND c) {...} einer äquivalenten Konstruktion if a {if b {if c {...}}}

Kontrollflussbezogene Testverfahren

- Bedingungsüberdeckungen
  - Einfache (oder atomare) Bedingungsüberdeckung (C_2-Test):
    - alle atomaren Prädikate einmal True und einmal False auswerten (Rückverweis)
    - Garantiert keine Zweigüberdeckung
  - Mehrfach-Bedingungsüberdeckung (C_3-Test):
    - alle Kombinationen der atomaren Prädikate werden betrachtet.
    - Bei einigermaßen komplizierten Prädikaten ist dies nicht mehr handhabbar
  - minimale Mehrfach-Bedingungsüberdeckung (CMinMehrfach):
    - Kompromiss zwischen C_2 und C_3,
    - jedes Prädikat (egal ob atomar oder zusammengesetzt) wird zu True und False ausgewertet,
    - weniger als die kombinatorische Abdeckung,
    - mehr als die beidseitige Auswertung der zusammengesetzten Prädikate.

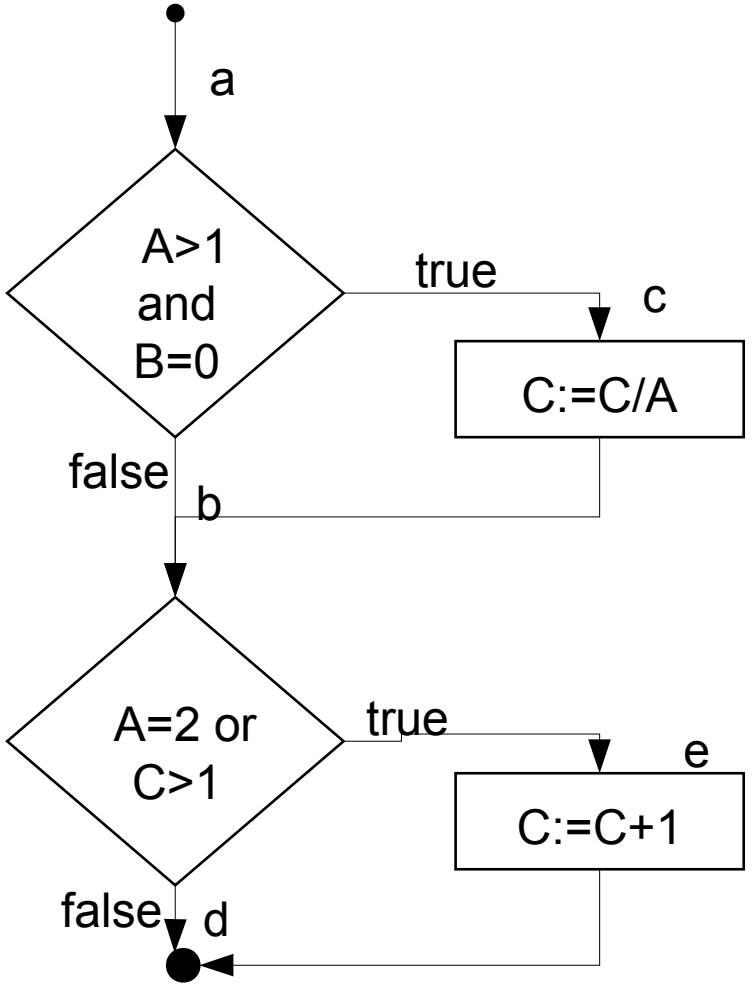
Kontrollflussbezogene Testverfahren

- Einfache Bedingungsüberdeckung (C_2-Test)
  - Eine Testdatenmenge T erfüllt die C_2-Überdeckung genau dann, wenn es für jede Verzweigung im Programm mit zwei Ausgängen und für jeden Prädikatterm p des zur Verzweigung gehörenden Booleschen Ausdrucks zu jedem Wahrheitswert von p ein Testdatum t aus T gibt, bei dessen Ausführung p diesen Wahrheitswert annimmt.
  - Hinweis: Verzweigungen mit mehr als zwei Ausgängen (z.B. case) können stets in ein äquivalentes Konstrukt mit zwei Ausgängen umgeformt werden
  - Garantiert keine Zweigüberdeckung, da Testen beider Wahrheitswerte der kompletten, d.h. der zusammengesetzten Booleschen Ausdrücke nicht garantiert ist

Kontrollflussbezogene Testverfahren

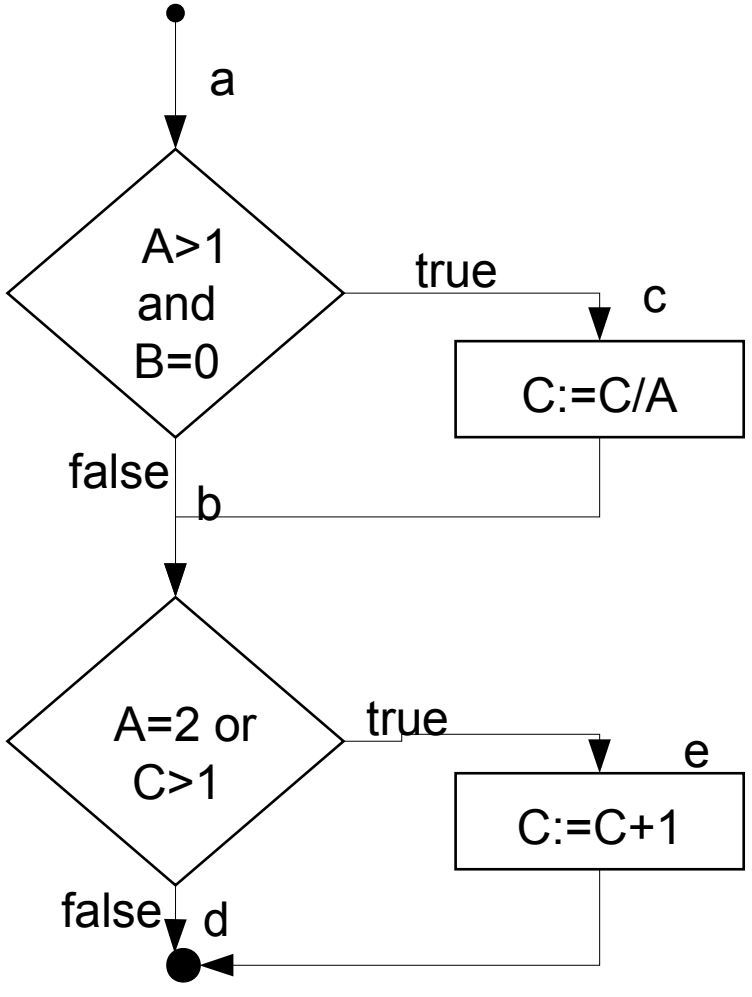
- Einfache Bedingungsüberdeckung (C_2-Test)
  - Beispiel
    - Testdatum A=2,B=1,C=4
      - A>1 true
      - B=0 false
      - A=2 true
      - C>1 true
      - Weg: abe
    - Testdatum A=1,B=0,C=1
      - A>1 false
      - B=0 true
      - A=2 false
      - C>1 false
      - Weg: abd

    ➢ Zweig c wird nicht überdeckt!



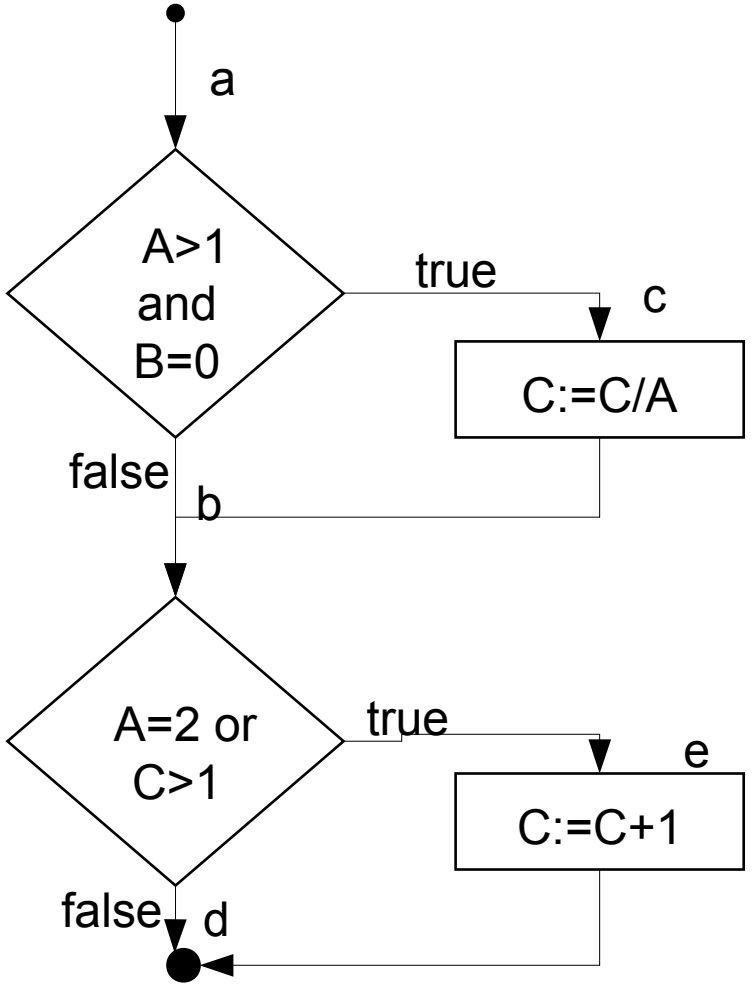
Kontrollflussbezogene Testverfahren

- Zweig-/Bedingungsüberdeckung
  - Eine Testdatenmenge erfüllt die Zweig-/Bedingungsüberdeckung genau dann, wenn sie die C_2-Überdeckung (atomare Bedingungsüberdeckung) und die C_1-Überdeckung (Zweig-überdeckung) erfüllt

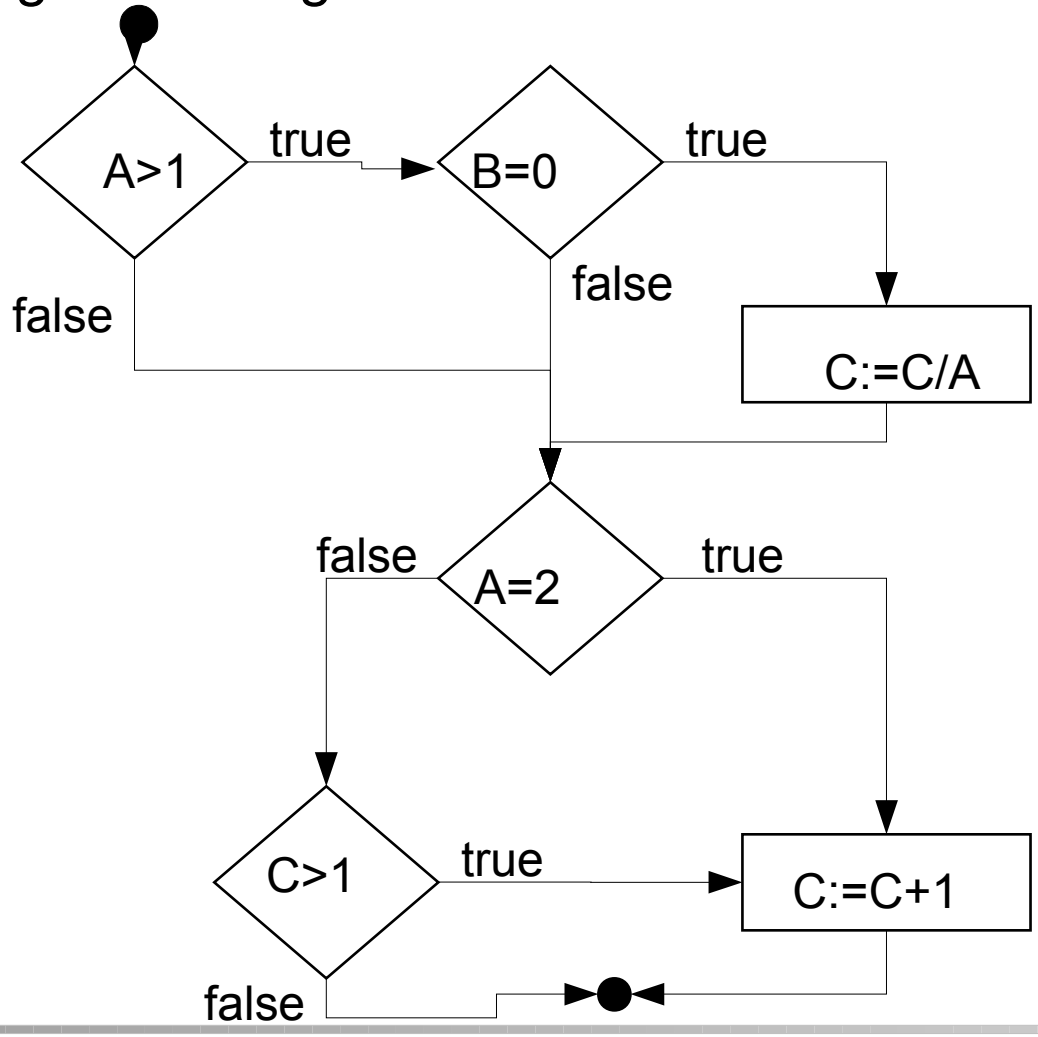


Kontrollflussbezogene Testverfahren

- Zweig-/Bedingungsüberdeckung
  - Beispiel
    - Testdatum A=2,B=0,C=4
      - A>1 true
      - B=0 true
      - A=2 true
      - C>1 true
      - Weg: ace
    - Testdatum A=1,B=1,C=1
      - A>1 false
      - B=0 false
      - A=2 false
      - C>1 false
      - Weg: abd
  - ➢ Alle Zweige nun überdeckt, aber nicht alle Zweigkombinationen

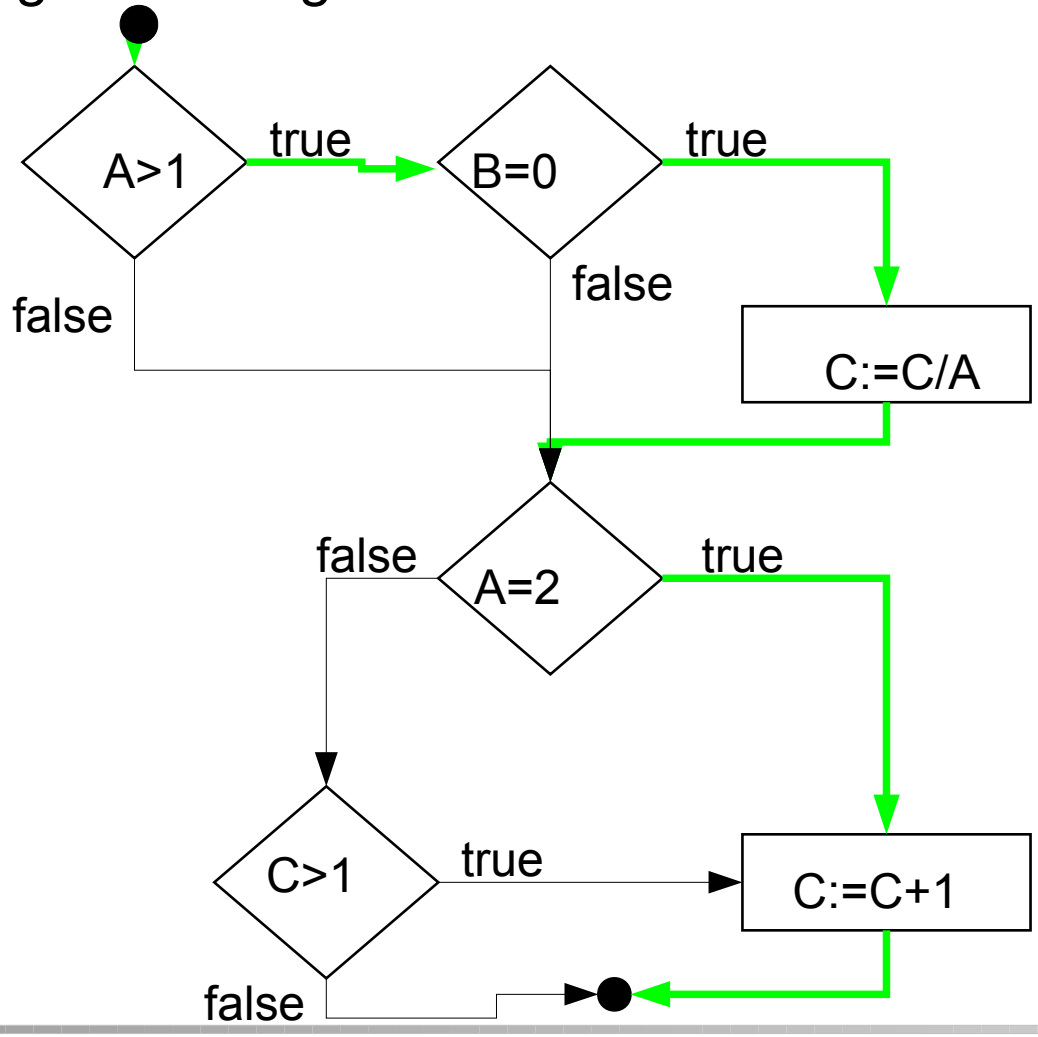


- Zweig-/Bedingungsüberdeckung
  - Beispiel für fehlende Überdeckung von Zweigkombinationen
    - Um dies zu zeigen: Äquivalente Umformung des Kontrollflussgraphen

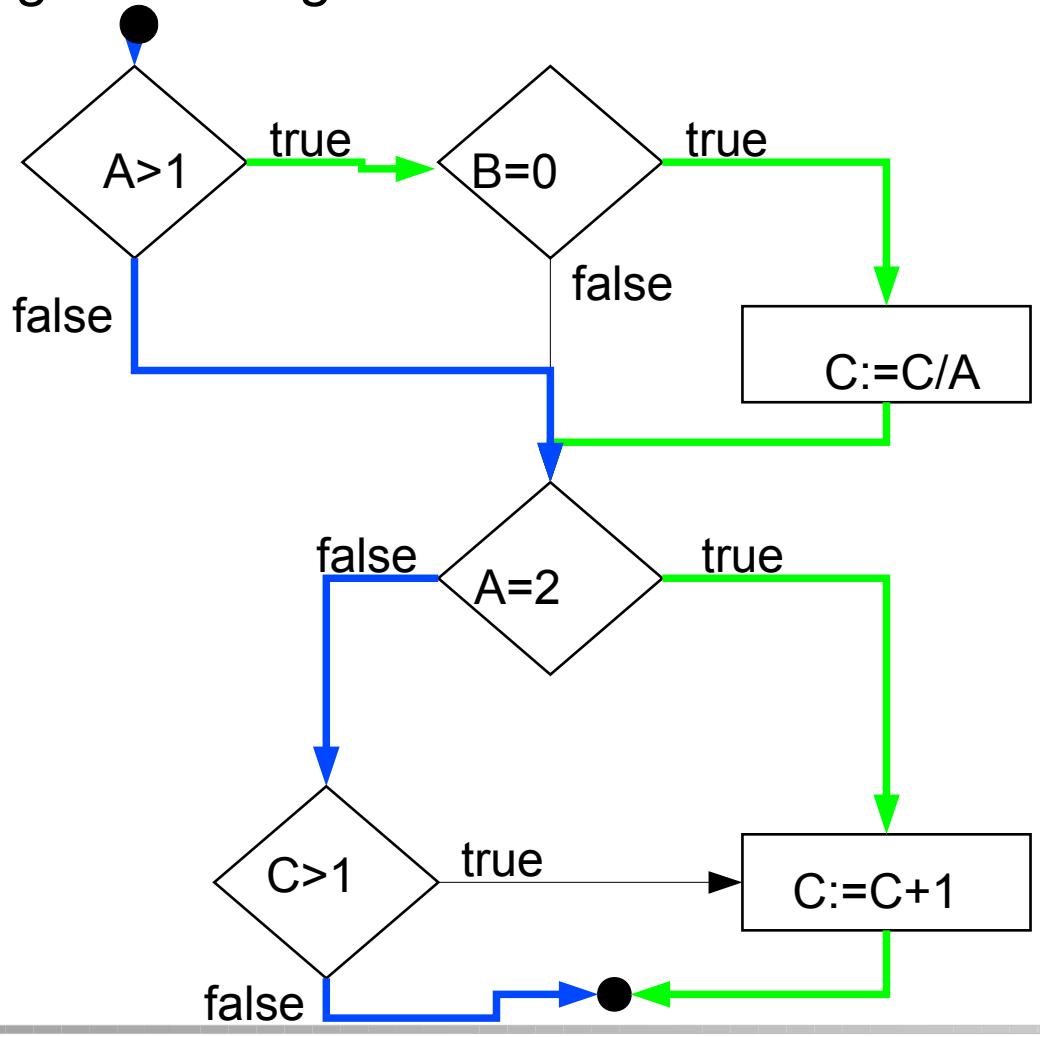


Kontrollflussbezogene Testverfahren

- Zweig-/Bedingungsüberdeckung
  - Beispiel für fehlende Überdeckung von Zweigkombinationen
    - Testdatum A=2,B=0,C=4

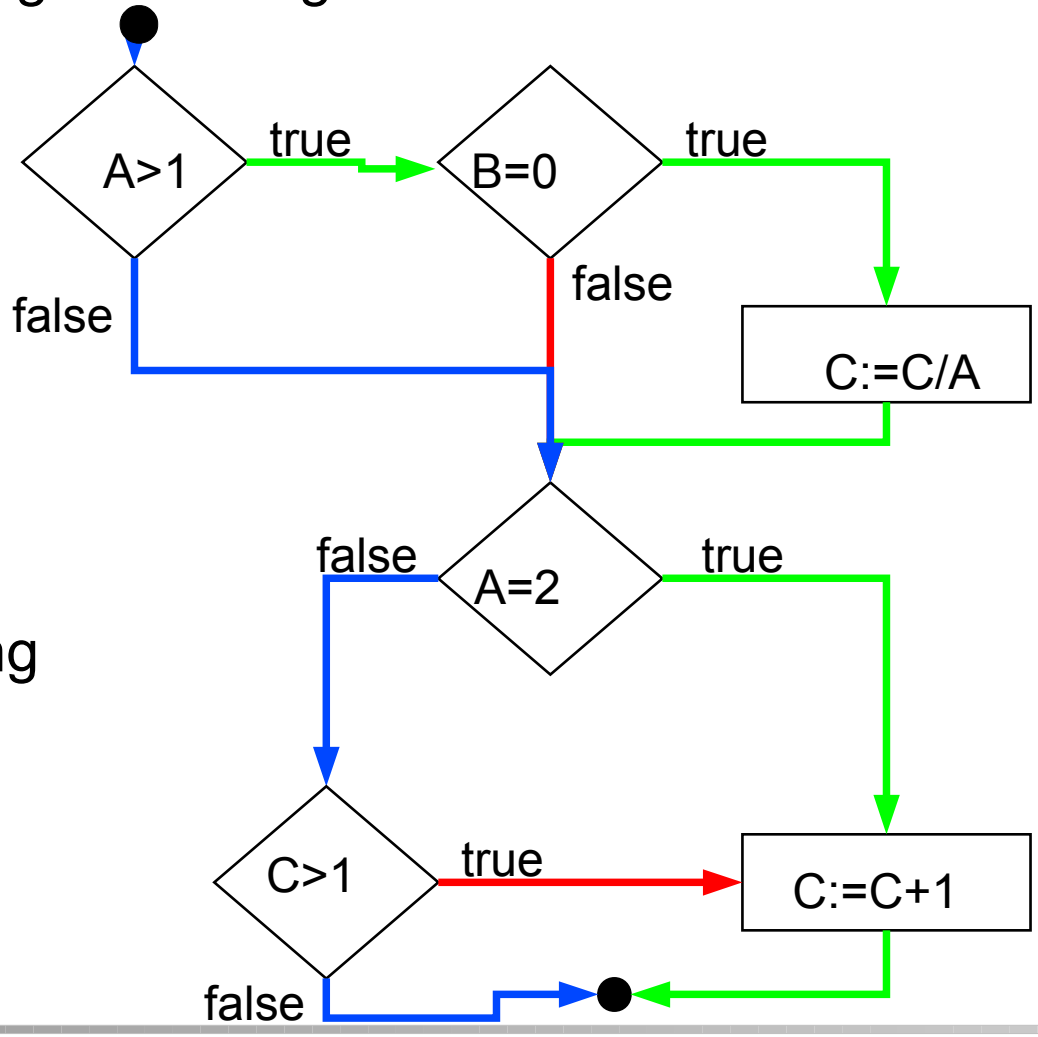


- Zweig-/Bedingungsüberdeckung
  - Beispiel für fehlende Überdeckung von Zweigkombinationen
  - Testdatum A=2,B=0,C=4
  - Testdatum A=1,B=1,C=1



Kontrollflussbezogene Testverfahren

- Zweig-/Bedingungsüberdeckung
  - Beispiel für fehlende Überdeckung von Zweigkombinationen
  - Testdatum A=2,B=0,C=4
  - Testdatum A=1,B=1,C=1
  - 2 Zweige fehlen!



  - Lösung:
    Mehrfachbedingungsüberdeckung

- Mehrfachbedingungsüberdeckung (C_3-Überdeckung)

  - Eine Testdatenmenge erfüllt die Mehrfachbedingungsüberdeckung genau dann wenn

  - für jede Verzweigung im Programm mit zwei Ausgängen und für den zur Verzweigung gehörenden Booleschen Ausdruck jede mögliche Kombination der Wahrheitswerte der atomaren Prädikatterme ausgeführt wird

- Mehrfachbedingungsüberdeckung (C_3-Überdeckung)

X (A>1)	Y (B=0)	X and Y
false	false	false
false	true	false
true	false	false
true	true	true

X (A=2)	Y (C>1)	X or Y
false	false	false
false	true	true
true	false	true
true	true	true

Jeweils 2^n Variationen

Kontrollflussbezogene Testverfahren

- Mehrfachbedingungsüberdeckung (C_3-Überdeckung)
  - Enthält Zweigüberdeckung
  - Aufwändig zu realisieren (2^n Variationsmöglichkeiten pro n atomaren Bedingungen)
  - Nicht immer sind alle Kombinationen durch Testdaten realisierbar
  - Dies muss nicht auf einen Fehler der Programmlogik hinweisen!
  - Testbeurteilung wird erschwert!
  - Besser: minimale Mehrfachbedingungsüberdeckung

Kontrollflussbezogene Testverfahren

- Minimale Mehrfachbedingungsüberdeckung (C_2(mM)-Überdeckung)
  - Eine Testdatenmenge erfüllt die minimale Mehrfach-bedingungsüberdeckung genau dann wenn
  - für jede Verzweigung im Programm mit zwei Ausgängen und für den zur Verzweigung gehörenden Booleschen Ausdruck folgende Kombinationen der Wahrheitswerte der atomaren Prädikatterme, die mit ein- und mehrstelligen logischen Operatoren verknüpft sind, ausgeführt werden:
  - Jede mögliche Kombination von Wahrheitswerten, bei denen die Änderung des Wahrheitswertes eines Terms den Wahrheitswert der logischen Verknüpfung ändern kann.
  - Anzahl der Testfälle geringer als bei der Mehrfachbedingungsüberdeckung, nämlich zwischen n+1 und 2n (n Anzahl der booleschen Operanden)

- Minimale Mehrfachbedingungsüberdeckung (C_2(mM)-Überdeckung)

X (A>1)	Y (B=0)	X and Y
false	false	false
false	true	false
true	false	false
true	true	true

Alle Kombinationen bis auf die erste zu testen, da bei dieser eine Änderung des Wahrheitswertes eines Terms keine Änderung des Wahrheitswertes der Kombination bewirkt

X (A=2)	Y (C>1)	X or Y
false	false	false
false	true	true
true	false	true
true	true	true

Alle Kombinationen bis auf die letzte zu testen, da bei dieser eine Änderung des Wahrheitswertes eines Terms keine Änderung des Wahrheitswertes der Kombination bewirkt

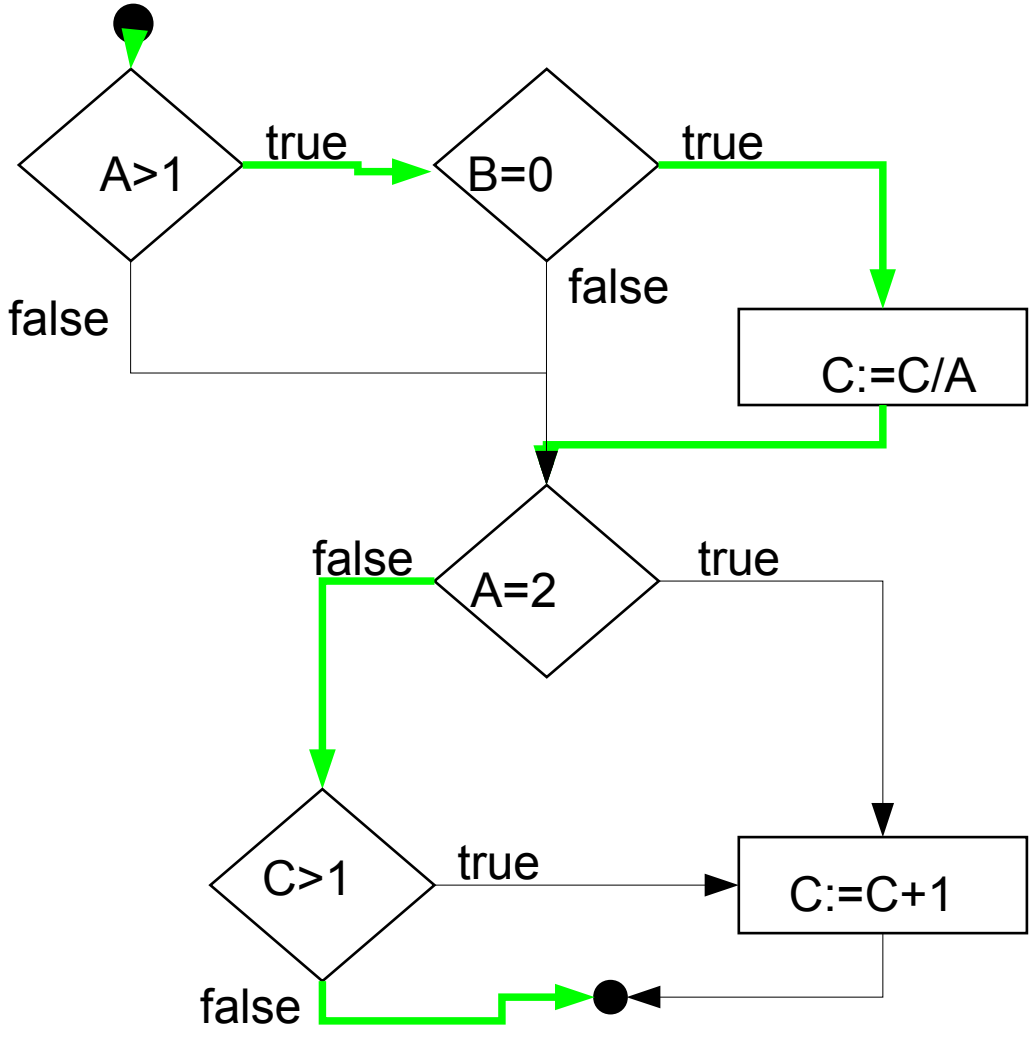
- Minimale Mehrfachbedingungsüberdeckung (C_2(mM)-Überdeckung)

X (A>1)	Y (B=0)	X and Y
false	false	false
false	true	false
true	false	false
true	true	true

X (A=2)	Y (C>1)	X or Y
false	false	false
false	true	true
true	false	true
true	true	true

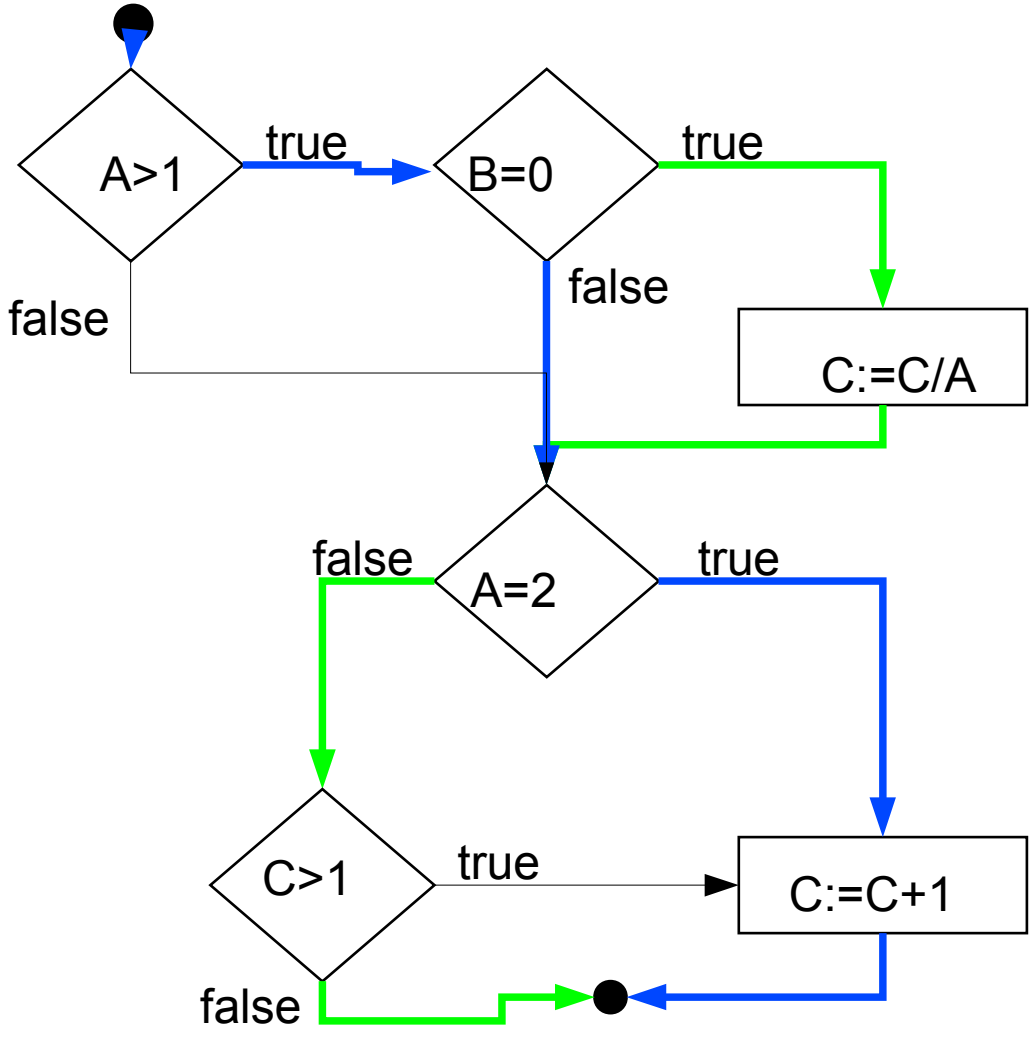
Testdatum A=3,B=0,C=3
Testdatum A=2,B=1,C=0
Testdatum A=1,B=0,C=2

- Minimale Mehrfachbedingungsüberdeckung
- Beispiel
  - Testdatum A=3,B=0,C=3
    - A>1 true
    - B=0 true
    - A=2 false
    - C>1 false (da Auswertung C:=C/A)

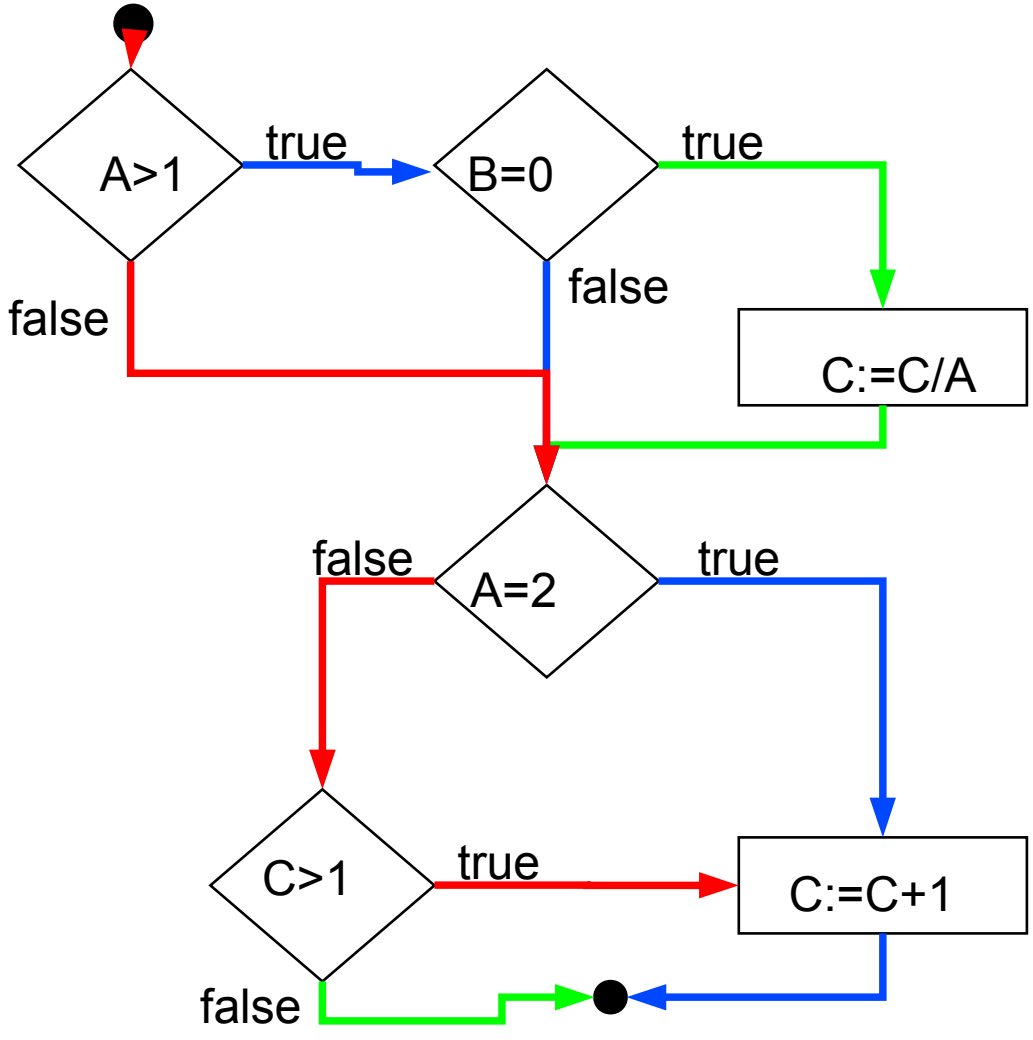


Kontrollflussbezogene Testverfahren

- Minimale Mehrfachbedingungsüberdeckung
- Beispiel
  - Testdatum A=3,B=0,C=3
    - A>1 true
    - B=0 true
    - A=2 false
    - C>1 false (da Auswertung C:=C/A)
  - Testdatum A=2,B=1,C=0
    - A>1 true
    - B=0 false
    - A=2 true
    - C>1 false



- Minimale Mehrfachbedingungsüberdeckung
- Beispiel
  - Testdatum A=3,B=0,C=3
    - A>1 true
    - B=0 true
    - A=2 false
    - C>1 false (da Auswertung C:=C/A)
  - Testdatum A=2,B=1,C=0
    - A>1 true
    - B=0 false
    - A=2 true
    - C>1 false
  - Testdatum A=1,B=0,C=2
    - A>1 false
    - B=0 true
    - A=2 false
    - C>1 true
- Alle Zweige geprüft



- Minimale Mehrfachbedingungsüberdeckung
  - Testwirksamkeitsmaß $C_{MinMehrfach}$

$$C_{MinMehrfach} \overset{def}{=} \frac{\text{Anzahl Prädikate}_{TRUE} + \text{Anzahl Prädikate}_{FALSE}}{2 * \text{Anzahl der Prädikate}}$$

Kontrollflussbezogene Testverfahren

Minimale Mehrfach-bedingungs-überdeckung

- 2. Beispiel: Prozedur WerteZiffernfolgeAus

Prädikate	Eingabedaten					
	'9'	'z'	'.9'	'..'	''	'.'
`Position <= TextVW.Length (inZiffernString)`	T,F					
`Fehlerfrei`	T	T,F				
`WHILE-Bedingung`	T,F					
`Zchn IN CZiffern`	T	F				
`WoBinIch = NachDemKomma`	F	-	T			
`Zchn = '.'`	-	F	T			
`WoBinIch = VorDemKomma`	-	T	T	T,F		
`ELSIF-Bedingung`	-	F	T			
`NOT Fehlerfrei`	F	T				
`TextVW.Length (inZiffernString) = 0`	F	F	F	F	T	
`WoBinIch = NachDemKomma`	F	F	T			
`TextVW.Length (inZiffernString) = 1`	T	T	F			
`IF3/AND-Prädikat`	F	F	F	F	F	T
`IF3-Bedingung`	F	T				

- Vorgehen
  - Erster Testfall:
  - Menge aller Eingaben, die das Prädikat
    Position <= TextVWLenght(inZiffernstring) zu TRUE auswerten

    -> Eingabedatum ´9´
  - Protokollierung der Werte der Prädikate für diesen Testfall
    => erstes und drittes Prädikat wird vollständig abgehandelt

$$C_{MinMehrfach} = \frac{5 + 8}{2 * 14} = \frac{13}{28} = 46\%$$

(5 Prädikate mit T in erster Spalte der Tabelle, 8 mit F; 14 Prädikate insgesamt)

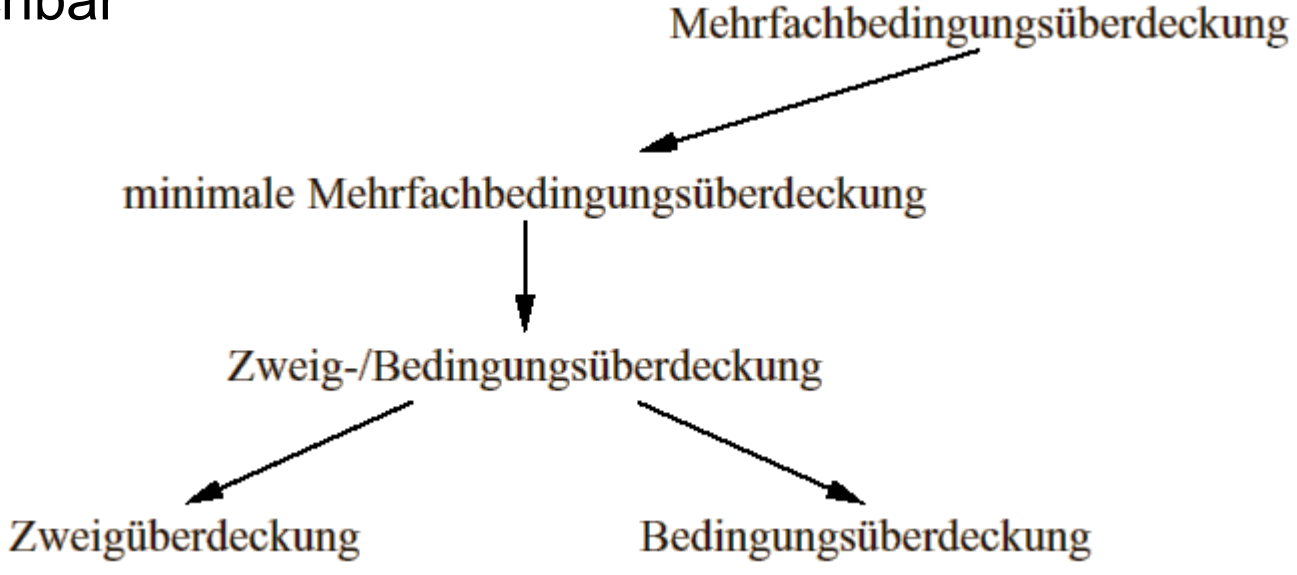
- Vorgehen
  - Zweiter Testfall:
  - Menge aller Eingaben, die Fehlerfrei zu FALSE auswertet

    -> Eingabedatum ´z´
  - Protokollierung der Werte der Prädikate für diesen Testfall

$$C_{MinMehrfach} = \frac{8+12}{2 * 14} = \frac{20}{28} = 71\%$$

Kontrollflussbezogene Testverfahren

- Zusammenhang der kontrollflussbezogenen Verfahren
  - Eine Kante von Kriterium K1 nach K2 bedeutet, dass K1 das Kriterium K2 strikt enthält, d.h. dass K1 Kriterium K2 enthält, aber K2 nicht K1 enthält.
  - K1 enthält K2, genau dann, wenn jede Testdatenmenge, die K1 erfüllt, auch K2 erfüllt
  - Falls kein Weg von K1 nach K2 existiert, sind die Kriterien unvergleichbar



Kontrollflussbezogene Testverfahren

- Anzahl der Tests pro Testkriterium
  - Aufwand nur vergleichbar für Testen einer einzelnen Entscheidung mit n Termen

Kriterium	**Zahl der erforderlichen Tests pro Entscheidung mit n Termen**
Bedingungsüberdeckung	2
Minimale Mehrfachbedingungs-überdeckung	$\leq n+1$
Mehrfachbedingungs-überdeckung	$\leq 2^n$

- Anzahl der Testdaten pro Testkriterium
  - Falls die Anzahl der ausgehenden Kanten pro Entscheidungsknoten und die Anzahl der Variablen pro Segment durch eine Konstante begrenzt sind, dann gilt für ein Programm mit n Segmenten im schlechtesten Fall:
    - Pfadüberdeckung (C_4-Überdeckung; jeder Pfad wird mindestens einmal durchlaufen): keine a-priori-Grenze für die Anzahl von Testdaten (nur vorhanden bei vorhandener Grenze an Anzahl der Schleifendurchläufe)
    - Strukturierte Pfadüberdeckung ($C_P(k)$; jede Schleife wird höchstens k mal ausgeführt): $O(2^n)$ Testdaten
    - Zweig- und Anweisungsüberdeckung: O(n) Testdaten

Kontrollflussbezogene Testverfahren

- Aufgedeckte Fehler pro Testkriterium (vgl. [Rie97])
  - Fehler: inkorrektes Programmverhalten

Kriterium	**Gefundene Fehler**
Anweisungsüberdeckung	18% bis 41%
Zweigüberdeckung	16% bis 92%
Strukturierte Pfadüberdeckung:	das zweifache der Zweigüberdeckung (43% statt 21%*)
Pfadüberdeckung	das dreifache der Zweigüberdeckung (62% bis 64% statt 21%*)
[minimale (Mehrfach-)] Bedingungsüberdeckung	keine Studien vorhanden

  - Unterschiedliche Quoten beruhen auf unterschiedlichen Untersuchungsansätzen

* Die Angabe 21% beruht auf der Messung nach [How78]

Datenflussbezogene Testverfahren

- Datenflussbezogene Testverfahren sind (wie kontrollflussorientierte Verfahren) white-box-Verfahren, denn sie nutzen die Kenntnis der Programmstruktur aus.

- Sie sind dynamische Verfahren, denn sie basieren auf der Ausführung des Programms mit verschiedenen Testdaten.

- Idee: Getestet werden Interaktion zwischen Anweisungen, die den Wert einer Variablen berechnen (definieren), und Anweisungen, die diesen Variablenwert benutzen (referenzieren).

Datenflussbezogene Testverfahren

- Beispiel: Anweisung A=B+C*D liefert ein falsches Ergebnis, wobei die Anweisung korrekt ist, aber die berechneten Werte vorher falsch berechnet wurden

- Datenflussorientierte Verfahren unterscheiden sich darin, ob sie alle Interaktionen oder nur einen Teil davon testen (->Überdeckungsmaße)

- Definition der Überdeckungsmaße orientiert sich am Kontrollflussgraphen, erweitert um zusätzliche Informationen
  - ➢ Datenflussgraph

Datenflussbezogene Testverfahren

- Datenflussgraph
  - Ein Datenflussgraph ist ein Kontrollflussgraph, bei dem zusätzlich zu jedem Knoten k die Mengen DEF(k), UNDEF(k), und REF(k) gehören.
  - DEF(k): Menge der Variablen x, für welche die Anweisungsfolge f, die zum Knoten k gehört, der Variablen x einen Wert zuweist, der nicht anschließend in f undefiniert wird.
  - UNDEF(k): Menge der Variablen x, für welche die Anweisungsfolge f, die zum Knoten k gehört, die Variablen x in einen undefinierten Zustand überführt, ohne x anschließend in f neu zu definieren.

Datenflussbezogene Testverfahren

- Datenflussgraph ff.
  - REF(k): Menge der Variablen x, für welche die Anweisungsfolge f, die zum Knoten k gehört, die Variable x referenziert, ohne dass x vorher in f undefiniert wird.
    - Dabei zu berücksichtigen: Rein lokale Datenflüsse müssen vermieden werden, denn die werden bei den datenflussbezogenen Testkriterien nicht berücksichtigt. Ein Datenfluss heißt rein lokal, wenn intern eine Referenz auf eine Definition folgt.
  - Datenflussschema
    - Entsteht durch Ersetzen der Anweisungen und Entscheidungsprädikate durch formale Bezeichner.

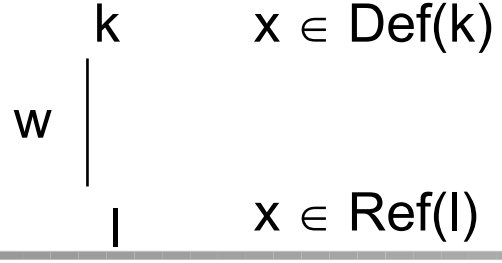
Datenflussbezogene Testverfahren

- Datenflussgraph: Beispiele
  - Buffer(bufpos):=c sei Anweisung des Knotens k
    - DEF(k) = {Buffer}
    - REF(k) = {bufpos,c}
    - UNDEF(k) = {}
  - X:=X+1 sei Anweisung des Knotens k
    - DEF(k) = REF(k) = {X}
    - UNDEF(k) = {}
  - X:=A+B; Y:=C*D; B:=Z, FREE(A); Y:=A+Z
    sei Anweisungsfolge des Knotens k
    - DEF(k) = {X,Y,B}
    - REF(k) = {A,B,C,D,Z}
    - UNDEF(k) = {A}

- Datenflussgraph: Beispiele
  - X:=A+B; Y:=C*D; B:=Y, FREE(A); Y:=A+Z sei Anweisungsfolge des Knotens k
    - Enthält rein lokalen Datenfluss von Y:=C*D nach B:=Y, also auf 2 Knoten aufteilen:
      - X:=A+B; Y:=C*D; sei Anweisungsfolge des Knotens k
        - DEF(k) = {X,Y}
        - REF(k) = {A,B,C,D}
        - UNDEF(k) = {}
      - B:=Y, FREE(A); Y:=A+Z sei Anweisungsfolge des Knotens l
        - DEF(l) = {Y,B}
        - REF(l) = {Y,Z}
        - UNDEF(l) = {A}

Datenflussbezogene Testverfahren

- Datenflussgraph: weitere Definitionen
  - Die Zuordnung von bedingten Anweisungen zu Knoten sei so gewählt, dass diese Knoten (auch Entscheidungsknoten) nur Referenzen von Variablen enthalten, d.h. DEF(K)=UNDEF(K)={}
    - Beispiel: If ((B=C+D)) aufsplitten in B=C+D und if B
  - Man sagt „Eine Definition von x im Knoten k erreicht eine Referenz von x im Knoten l über den Weg w“
    - genau dann, wenn der Weg w im Kontrollflussgraphen von k nach l führt und die Variable x auf dem Weg nicht neu definiert oder undefiniert wird.



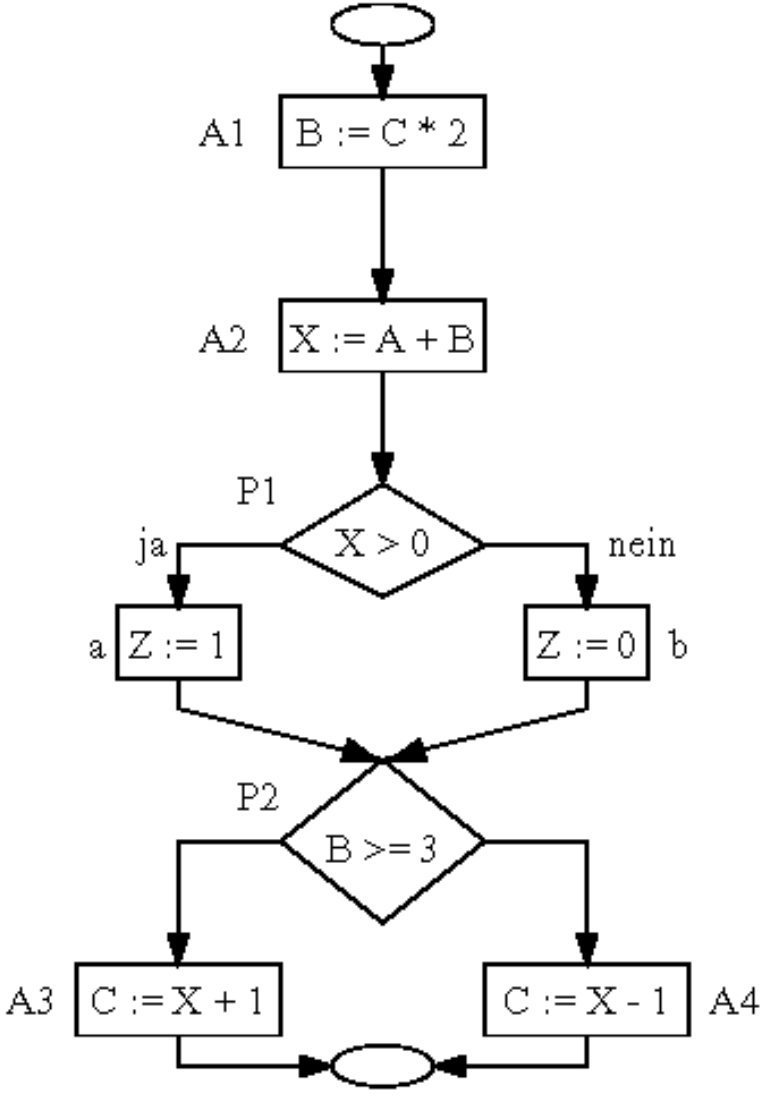
Datenflussbezogene Testverfahren

- Einfache Kriterien zur Überdeckung von Datenflüssen / informale Übersicht
  - Alle Definitionen
    - Das Resultat einer Zuweisung (Definition) wird wenigstens einmal benutzt wird
  - Alle DR-Interaktionen
    - ALLE Paare von Definitionen/Referenzen einer Variablen testen
  - Alle Referenzen
    - Alle ausgehenden Kanten eines Entscheidungsknotens berücksichtigen
  - Alle Entscheidungs-/einige Berechnungs-Referenzen
    - Schwächere Überdeckung des Datenflusses, trotzdem Zweigüberdeckung und Testen „aller Definitionen“
  - Alle Berechnungs-/einige Entscheidungs-Referenzen
    - Keine Zweigüberdeckung, aber Testen „aller Definitionen“ und „aller Referenzen“ in Berechnungsknoten
  - Alle DR-Wege
    - Stärkere Überdeckung unter Kontrollflussaspekten durch Annäherung an Pfadtesten, allerdings ohne Schleifeniterationen

Datenflussbezogene Testverfahren

- Kriterium „alle Definitionen“
  - Kriterium: Das Resultat einer Zuweisung (Definition) wird wenigstens einmal benutzt
  - Eine Testdatenmenge T erfüllt das Kriterium „alle Definitionen“ g.d.w. es
    - für jede Variable x und jede Definition von x mindestens einen Weg in Wege(T) gibt, auf dem die Definition eine Referenz von x erreicht.

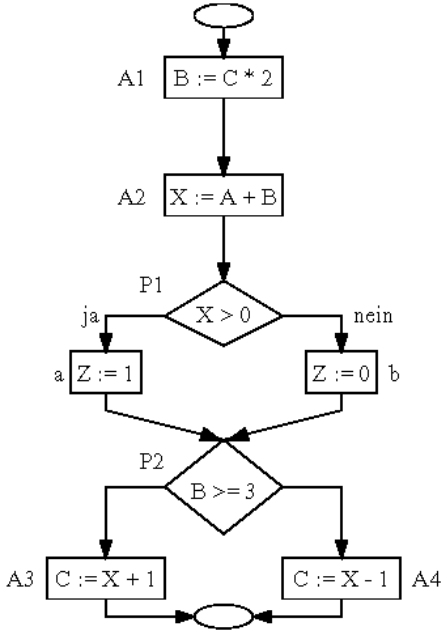
Datenflussbeispiel



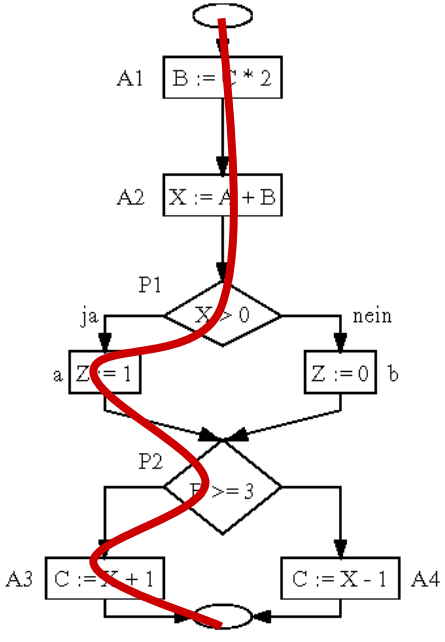
- Kriterium „alle Definitionen“: Beispiel
  - T sei die Testdatenmenge, die den Weg (A1, A2, P1, a, P2, A3) ausführt
  - T erfüllt das Kriterium „alle Definitionen“ für die Variablen X und B
  - Aber: nicht ALLE Paare von Definitionen/Referenzen einer Variablen werden getestet (Definition von X, Referenz von X in A4)

$X \in Ref(P1)$

$B \in Ref(P2)$



- Kriterium „alle Definitionen“: Beispiel
  - T sei die Testdatenmenge, die den Weg (A1, A2, P1, a, P2, A3) ausführt
  - T erfüllt das Kriterium „alle Definitionen“ für die Variablen X und B
  - Aber: nicht ALLE Paare von Definitionen/Referenzen einer Variablen werden getestet (Definition von X, Referenz von X in A4)



Weg (A1, A2, P1, a, P2, A3)

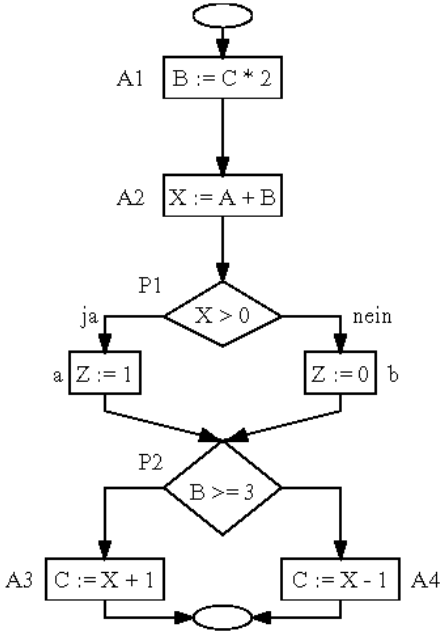
- Kriterium „alle DR-Interaktionen“
  - Ziel: ALLE Paare von Definitionen/Referenzen einer Variablen testen
  - Eine Testdatenmenge T erfüllt das Kriterium „alle DR-Interaktionen“ g.d.w. es
    - für jede Variable x, jede Definition von x und jede Referenz von x, die davon erreicht wird, mindestens einen Weg in Wege(T) gibt, auf dem die Definition die Referenz von x erreicht.

- Kriterium „alle DR-Interaktionen“: Beispiel
  - T sei die Testdatenmenge, die die Wege (A1, A2, P1, a, P2, A3) und (A1, A2, P1, a, P2, A4) ausführt
  - T erfüllt das Kriterium „alle DR-Interaktionen“ für die Variablen X und B
  - Aber: Entscheidungskante b wird nicht ausgeführt

$X \in Ref(P1)$

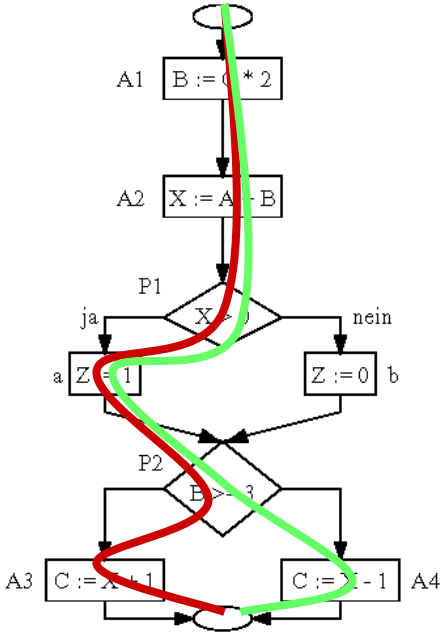
$B \in Ref(P2)$

$X \in Ref(A3)$



$X \in Ref(A4)$

- Kriterium „alle DR-Interaktionen“: Beispiel
  - T sei die Testdatenmenge, die die Wege (A1, A2, P1, a, P2, A3) und (A1, A2, P1, a, P2, A4) ausführt
  - T erfüllt das Kriterium „alle DR-Interaktionen“ für die Variablen X und B
  - Aber: Entscheidungskante b wird nicht ausgeführt



Weg (A1, A2, P1, a, P2, A3)

Weg (A1, A2, P1, a, P2, A4)

Datenflussbezogene Testverfahren

- Motivation: Für die Referenz einer Variablen in einem Entscheidungsknoten ist es von Bedeutung wie der Ausgang der Entscheidung ist.

- Kriterium „alle Referenzen“
  - Ziel: alle ausgehenden Kanten eines Entscheidungsknotens berücksichtigen

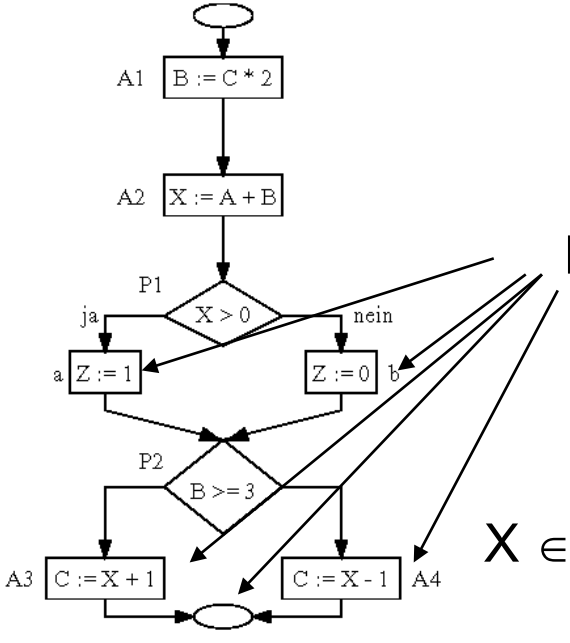
  - Eine Testdatenmenge T erfüllt das Kriterium „alle Referenzen“ g.d.w. es
    - für jede Variable x, jede Definition von x in einem Knoten k, jede Referenz von x in einem Knoten l, die von der Definition in k erreicht wird, und für jeden Nachfolgerknoten m von l die Wegemenge Wege(T) mindestens ein Wegstück u*m enthält, wobei die Definition von x in k die Referenz von x in l über den Weg u erreicht.
    - Hierbei: u*m Knotenfolge mit u={k,..,l} gefolgt von Knoten m, d.h. u*m={k,..,l,m}

- Kriterium „alle Referenzen“
  - T sei die Testdatenmenge, die die Wege (A1, A2, P1, a, P2, A3) und (A1, A2, P1, b, P2, A4) ausführt
  - T erfüllt das Kriterium „alle Referenzen“ für die Variablen X und B
  - Aber: die Wege (A1, A2, P1, a, P2, A4) und (A1, A2, P1, b, P2, A3) werden nicht ausgeführt

$X \in Ref(P1)$

$B \in Ref(P2)$

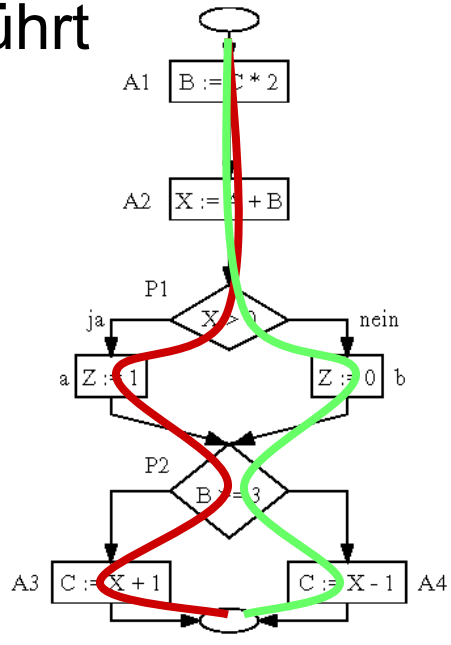
$X \in Ref(A3)$



$X \in Ref(A4)$

Datenflussbezogene Testverfahren

- Kriterium „alle Referenzen“
  - T sei die Testdatenmenge, die die Wege (A1, A2, P1, a, P2, A3) und (A1, A2, P1, b, P2, A4) ausführt
  - T erfüllt das Kriterium „alle Referenzen“ für die Variablen X und B
  - Aber: die Wege (A1, A2, P1, a, P2, A4) und (A1, A2, P1, b, P2, A3) werden nicht ausgeführt



Weg (A1, A2, P1, a, P2, A3)

Weg (A1, A2, P1, b, P2, A4)

Datenflussbezogene Testverfahren

- Einfache Datenflusskriterien
  - „Alle DR-Interaktionen“ und „alle Referenzen“ dienen als Kriterien zum Testen aller Paare von Definitionen und Referenzen, aber jeweils nur auf einem Weg von Definition zur Referenz.
  - Dies ist allerdings unter dem Gesichtspunkt des Datenflusses ausreichend, da zwischen Definition und Referenz keine Änderung der Variablen erfolgen kann.

Datenflussbezogene Testverfahren

- Anzahl der Testdaten pro Testkriterium
  - Falls die Anzahl der ausgehenden Kanten pro Entscheidungsknoten und die Anzahl der Variablen pro Segment durch eine Konstante begrenzt sind, dann gilt für ein Programm mit n Segmenten im schlechtesten Fall:
  - Kriterium „alle DR-Wege“ erfordert $O(2^n)$ Testdaten
  - Kriterien „alle E-/einige B-Referenzen“, „alle B-/einige E-Referenzen“, „alle Referenzen“ und Kontextüberdeckung erfordern $O(n^2)$ Testdaten
  - Kriterium „alle Definitionen“erfordert O(n) Testdaten

Datenflussbezogene Testverfahren

- Aufgedeckte Fehler pro Testkriterium (vgl. [Rie97])

Kriterium	Fehler erkannt
Alle Definitionen	24%
Alle E-/einige B-Referenzen	34%
Alle B-/einige E-Referenzen	48%
Alle DR-Interaktionen	51%

- Für restliche Kriterien keine Studien vorhanden

Zusammenfassung

- Diese Folien beinhalteten:
  - Ursachen von SW-Fehlern
  - Klassifikation von SW-Fehlern
  - Einleitende Begriffe (rund ums Testen)
  - Analytisches Qualitätsmanagement

Literatur

- Balzert, H.: Lehrbuch der Softwaretechnik, Band 2, Spektrum, 1998.
- Beydeda, S.: The Self-Testing COTS Components (STECC) Method, m press, 2004.
- Gao, J. et. Al.: Testing and Quality Assurance for Component-Based Software, Artech House, 2003.
- Meyers, G.: The Art of Software Testing, Wiley&Sons, 1979.
- Riedemann, E.H.: Testmethoden für sequentielle und nebenläufige Software-Systeme, Teubner, 1997
- Sommerville, I.: Software Engineering, Addison-Wesley, 2001.