Software-Qualitätsmanagement

Kernfach Angewandte Informatik

Sommersemester 2004

Prof. Dr. Hans-Gert Gräbe

Nachtrag: Klassifikation testender Verfahren

Begriff des Programms

- Programm = schrittweise Transformation einer Menge von Eingabedaten in eine Menge von Ausgabedaten nach einem vorgegebenen Algorithmus
- Black-Box-Betrachtung: f: X -> 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

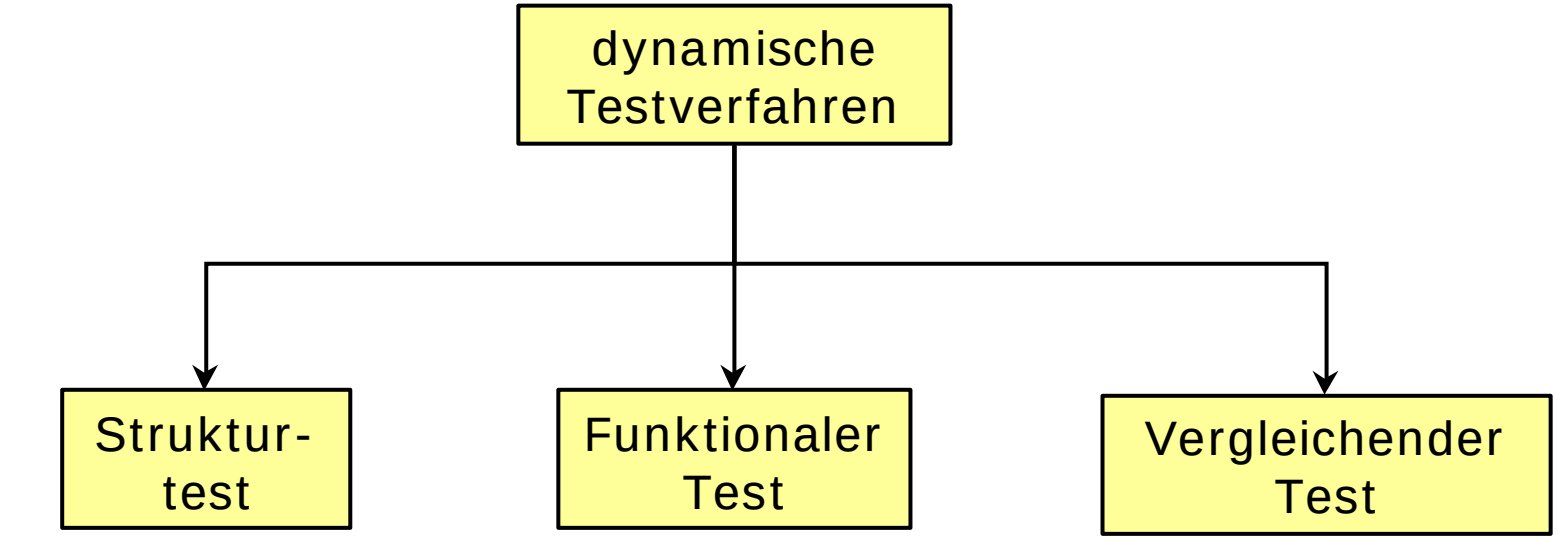
Nachtrag: Klassifikation testender Verfahren

Klassifikation testender Verfahren

- 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, testfallorientiertes Durchgehen
  - typische Verfahren: manuelle Prüfmethoden

Nachtrag: Klassifikation testender Verfahren

Klassifikation testender Verfahren



kontrollfluss- oder datenflussorientiert

benötigt den Quellcode

Testwertauswahl nach funktionalen Gesichtspunkten

benötigt die Spezifikation

Vergleich der Ergebnisse verschiedener Programmversionen, etwa durch Einfügen künstlicher Fehler

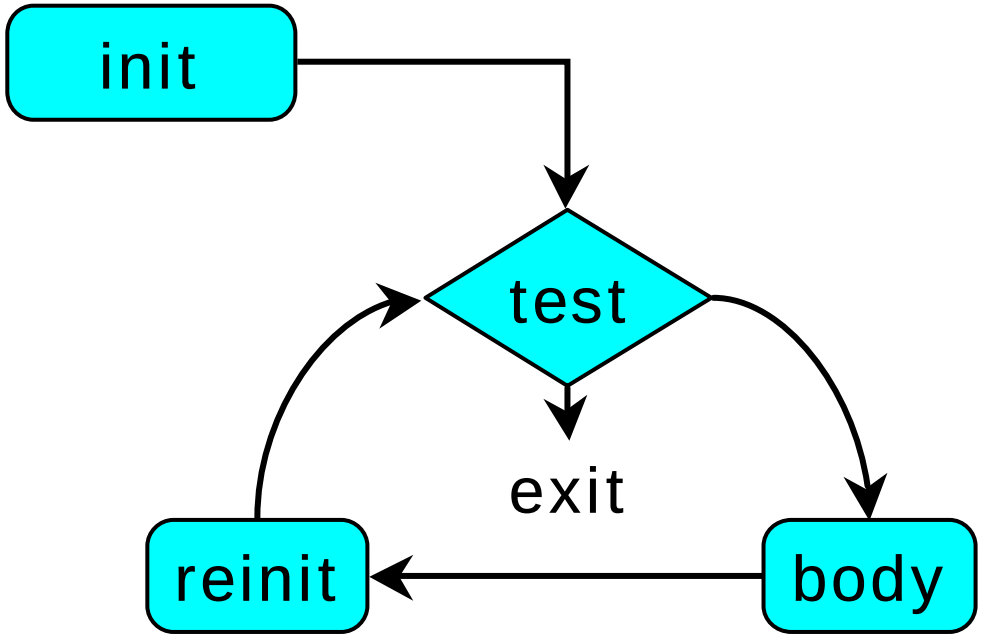
Nachtrag: Klassifikation testender Verfahren

Klassifikation dynamischer 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

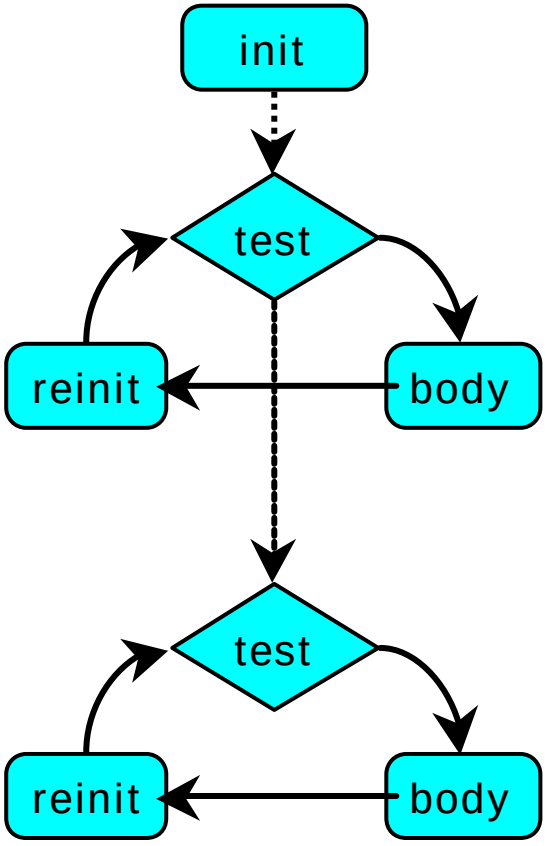
Nachtrag: Pfadüberdeckungstests

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



Nachtrag: Pfadüberdeckungstests

Problem des exponentiellen Wachstums der Anzahl der Pfade



Vernünftige Verfahren überdecken Initialisierung und Reinitialisierung, also jede Schleife zweimal.

Bedingungsüberdeckung

In den Testknoten wird genauer hineingeschaut

Technik: Wahrheitswert-Tabelle

Nachtrag: Auswahl geeigneter Testverfahren

Auswahl geeigneter kontrollflussorientierter Testverfahren

- Liegt Programm im Quellcode vor?
  - Nein: Kein Strukturtestverfahren möglich
- Besteht Programm nur aus Anweisungen?
  - Anweisungsüberdeckung sinnvoll
- .. nur Anweisungen und Verzweigungen mit atomaren Testbedingungen
  - Zweigüberdeckung sinnvoll
- .. Anweisungen, Verzweigungen und Schleifen mit atomaren Testbedingungen
  - Pfadüberdeckung, je nach Komplexität der Schleifensemantik doppelte oder mehrfache Schleifenüberdeckung
- ... komplexe Testbedingungen
  - Kopplung geeigneter Verfahren mit Bedingungsüberdeckung

Datenflussorientierte Strukturtestverfahren

- Verwenden für Testfallgenerierung Zugriff auf Variablen
  - Analyse der Programmdynamik an Hand der Dynamik der in Variablenwerten gespeicherten Programmzustände
  - Aufstellen des Datenflussgraphen
    - Sichtbarkeit und Lebensdauer von Bezeichnern
    - Gültigkeitskontext, Datenflussgraph und Kontrollfluss
    - lesende und schreibende Zugriffe auf Variablen
    - Unterscheidung lesender Zugriffe in Anweisungen und in Bedingungen
- Eignen sich für Test von Datenobjekt- und Datentypmodulen sowie Klassen.
- Nur wenige Testwerkzeuge vorhanden.
  - Kontrolle über Initialisierung und wirkliche Verwendung definierter Variablen ist Standard in modernen Compilern.

Defs/Uses-Verfahren

- Klassifikation der Variablenzugriffe nach
  - Zuweisung (set-Methode, Definition, *def*)
    - Variablenwert wird an einer solchen Stelle geändert
  - Zugriff zur Berechnung von anderen Werten (*computational-use, c-use*)
    - mglw. globale Auswirkung auf Wert anderer Variablen
  - Zugriff zur Berechnung von Wahrheitswerten in Bedingungen (*predicate-use, p-use*)
    - nur lokale Auswirkung in der aktuellen Testbedinugng

- Zur Datenflussanalyse muss Zusammenhang zwischen Wertzuweisung und -benutzung analysiert werden
  - ein Pfad heißt **definitionsfrei** für eine Variable x, wenn in dem Pfad keine Wertzuweisung an x erfolgt.
  - Eine Variablenbenutzung ist **lokal**, wenn Wertzuweisung und -benutzung in einem Block erfolgen, sonst **global.**

Datenflussorientierte Test-Verfahren

- ***all defs* Kriterium**
  - Testfälle sind so zu wählen, dass jeder Wertzuweisung an eine Variable auch eine Wertbenutzung folgt.
  - Vereinigung der Kontrollflüsse muss in jedem Knoten des DFG über eine abgehende Kante gehen.
  - **Idee**: Jede Variable hat einen „Zweck“, eine Semantik, die wenigstens einmal exemplarisch geprüft wird.
- Spezialfall: Kontrolle, ob alle definierten Variablen auch verwendet werden
  - Charakterisiert durch fehlende abgehende Pfeile im Datenflussgraphen.
  - Definitionen, die nicht benutzt werden, weisen auf Programmfehler hin.

- ***all p-uses* Kriterium**
  - Testfälle sind so zu wählen, dass jede Kombination einer Wertzuweisung mit jeder prädikativen Nutzung dieses Variablenwerts überdeckt wird
  - Vereinigung der Kontrollflüsse muss in jedem Knoten des DFG über alle zu Bedingungsknoten abgehende Kanten gehen.
  - **Idee**: Prüfen der Auswirkung einer Wertzuweisung an eine Variable auf alle relevanten Bedingungsknoten
    - Fokus auf die bedingungsrelevanten Variablen
  - beinhaltet Zweigüberdeckung

- ***all c-uses* Kriterium**
  - Testfälle sind so zu wählen, dass jede Kombination einer Wertzuweisung mit jeder berechnenden Nutzung dieses Variablenwerts überdeckt wird
  - Vereinigung der Kontrollflüsse muss in jedem Knoten des DFG über alle zu Berechnungsknoten abgehende Kanten gehen.
    - Beachte: Die Menge der kontrollflussbelegten zu Berechnungsknoten abgehenden Kanten kann leer sein (Variablenwert wird dann nur bedingungsrelevant benötigt)
  - **Idee**: Prüfen der Auswirkung einer Wertzuweisung an eine Variable auf alle kausal davon abhängenden Ausdrücke.
    - Fokus auf die berechnungsrelevanten Variablen

- ***all c-uses / some p-uses* Kriterium**
  - Testfälle sind so zu wählen, dass jede Kombination einer Wertzuweisung mit jeder berechnenden Nutzung dieses Variablenwerts überdeckt wird.
  - Falls kein berechnender Zugriff existiert, so muss der Wert in mindestens einem Prädikat benutzt werden.
  - Vereinigung der Kontrollflüsse muss in jedem Knoten des DFG über alle zu Berechnungsknoten abgehende Kante gehen und die Menge der kontrollflussbelegten abgehenden Kanten darf nicht leer sein
  - **Idee**: Prüfen der Auswirkung einer Wertzuweisung an eine Variable auf alle kausal davon abhängenden Ausdrücke und exemplarischer Test von nur bedingungsrelevanten Variablen

- ***all p-uses / some c-uses* Kriterium**
  - Testfälle sind so zu wählen, dass jede Kombination einer Wertzuweisung mit jeder prädikativen Nutzung dieses Variablenwerts überdeckt wird.
  - Falls kein prädikativer Zugriff existiert, so muss der Wert in mindestens einer Berechnung benutzt werden.
  - Vereinigung der Kontrollflüsse muss in jedem Knoten des DFG über alle zu Bedingungsknoten abgehende Kante gehen und die Menge der kontrollflussbelegten abgehenden Kanten darf nicht leer sein
  - **Idee**: Prüfen der Auswirkung einer Wertzuweisung an eine Variable auf alle davon abhängenden Bedingungen und exemplarischer Test von nur berechnungsrelevanten Variablen

5. Datenflussorientierte Strukturtests

- ***all uses* Kriterium**
  - Testfälle sind so zu wählen, dass jede Kombination einer Wertzuweisung mit jeder Nutzung dieses Variablenwerts überdeckt wird.
  - komplettester und damit aufwändigster datenflussorientierter Test

Leistungsfähigkeit nach Studie [Girgis, Woodward 86]

Vergleich *all defs, all p-/c-uses:*

- *all c-uses:* 48% der Fehler, insbesondere Berechnungsfehler,
- *all p-uses:* 34% und entdeckt Kontrollflussfehler,
- *all defs:* 24% der Fehler, aber keine Kontrollflussfehler

Weitere Verfahren

- **Idee**: Überdeckung längerer Sequenzen aus Zuweisung und Nutzung
  - *Required k-Tuples Test*
- **Idee**: Orientierung nicht an abgehenden, sondern an ankommenden Pfeilen im DFG
  - *Datenkontextüberdeckung*: jede mögliche Herkunft eines Werts wird überdeckt.
  - *geordnete Datenkontextüberdeckung*: zusätzliche Beachtung der Zuweisungsreihenfolge

Überblick

- **Idee**: Testfälle werden aus den Programmspezifikationen abgeleitet.
  - Quellcode wird nicht benötigt, deshalb auch „Black-Box-Verfahren“ genannt
  - Strukturtest = Test der inneren Programmlogik
  - Funktionaltest = Test der äußeren Programmsemantik
- **Ziel**: möglichst umfassende, aber redundanzarme Prüfung der spezifizierten Funktionalität
  - analog der strukturellen spricht man von funktioneller Überdeckung
  - Umfang oft im Pflichtenheft als Qualitätsprüfung vereinbart
- **Problem**: Bereich der möglichen Eingabewerte ist sehr groß oder sogar unendlich groß

Funktionale Äquivalenzklassenbildung

- **Idee:** Einteilung des Definitionsbereichs in endliche Anzahl von Klassen „ähnlicher“ Werte und Prüfung an je einem exemplarischen Vertreter pro Klasse
  - Klasseneinteilung längs „typischen“ Programmverhaltens
    - muss aber nicht mit innerer Programmstruktur zusammenhängen
  - Korrektheit auf einem typischen Vertreter lässt Korrektheit auf der ganzen Klasse erwarten
    - **Natürlich kein Beweis der Korrektheit**!
  - Nicht unbedingt Äquivalenzklassen im streng mathematischen Sinn, da sich Klassen überschneiden können.

- **Bewertung:**
  - geeignet zur Herleitung repräsentativer Testfälle
  - Betrachtung von einzelnen Werten, dadurch werden keine Wechselwirkungen oder Abhängigkeiten getestet

Regeln zur Bildung von Klassen

Ist der Eingabebereich

1. ein zusammenhängender Wertebereich $a \leq x \leq b$
   - drei Bereiche (einer gültig, zwei ungültig)
2. eine Menge von n Werten, welche unterschiedlich behandelt werden
   - für jeden gültigen Wert eine Klasse sowie eine Klasse für alle ungültigen Werte
3. eine Bedingung, die zwingend erfüllt sein muss
   - eine Klasse der Werte, für welche die Bedingung erfüllt ist und deren Komplement

Oft ergibt sich die Einteilung des Eingabebereichs als Vereinigung von Urbildmengen, d.h. Eingaben, welche dieselbe Ausgabe erzeugen, werden in eine Klasse zusammengefasst.

Grenzwertanalyse

Idee:

- Basiert auf der funktionalen Äquivalenzklassenbildung,
- Nutzt jedoch nicht irgendwelche Elemente aus den Klassen, sondern Werte, die am Rand der Klasse liegen.
  - Erfahrung besagt, dass durch Grenzwerte Fehler besonders effektiv entdeckt werden.
- Setzt sinnvollen Grenzbegriff (Topologie, Ordnung) auf der Menge der Eingabewerte voraus

Bewertung:

- Sinnvolle Erweiterung und Verbesserung der funktionalen Äquivalenzklassenbildung.

Ähnlich: Test spezieller Werte (Null-Tests, Nullpointer-Tests etc.), Zufallstest (Auswahl zufälliger Repräsentanten der Klassen)

Test von Zustandsautomaten

- **Idee:** Technische Software ist oft als Menge von Zuständen und Übergängen modelliert und als Zustandsdiagramm spezifiziert. Testfälle sind an dieses Modell angepasst auszuwählen.
  - minimale Teststrategie: Jeder Zustandsübergang ist mindestens einmal abzudecken
- Überdeckung aller Zustandsübergänge garantiert keinen vollständigen Test (analog Zweigüberdeckung)

- **Bewertung**
  - Gut geeignetes Testverfahren, falls die Spezifikation schon als Zustandsautomat vorliegt.
  - Gut geeignet für den Test von Klassen, wenn der Objektlebenszyklus vorliegt.

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