Software-Qualitätsmanagement

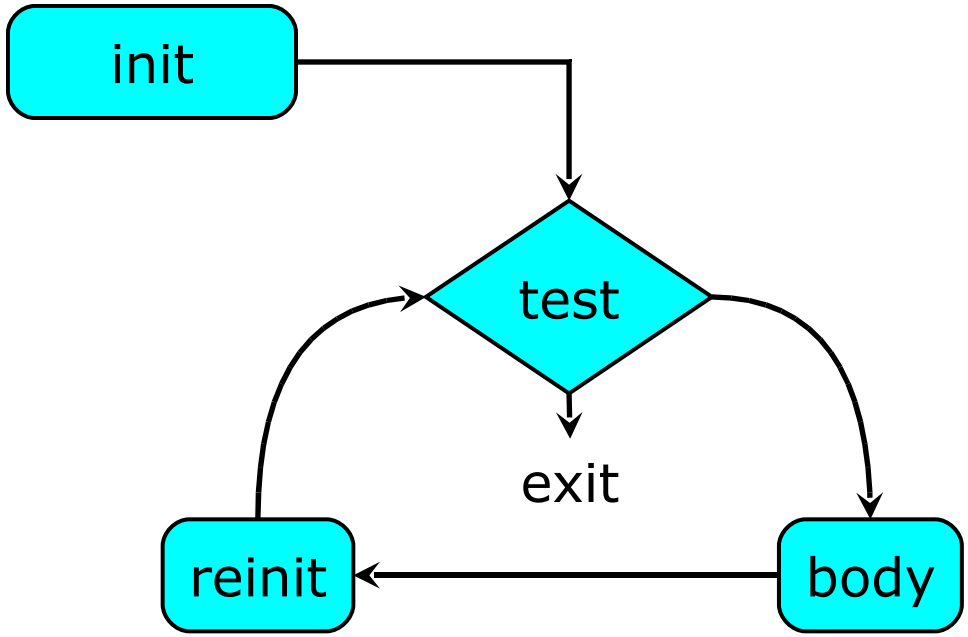
Kernfach Angewandte Informatik und
Vorlesung im Modul 10-202-2319
Software-Management

Sommersemester 2008

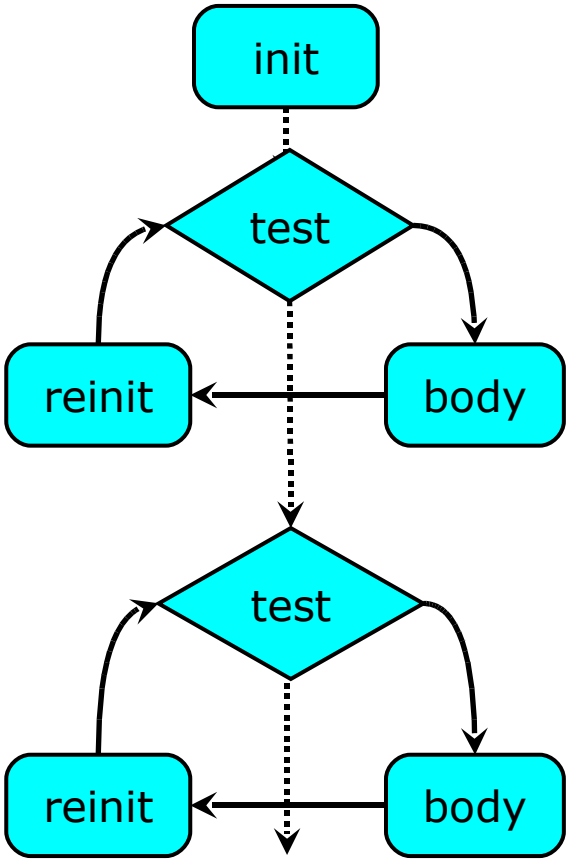
apl. Prof. Dr. Hans-Gert Gräbe
http://bis.informatik.uni-leipzig.de/HansGertGraebe

Testen von Schleifen

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



Problem des Wachstums der Anzahl der Pfade



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

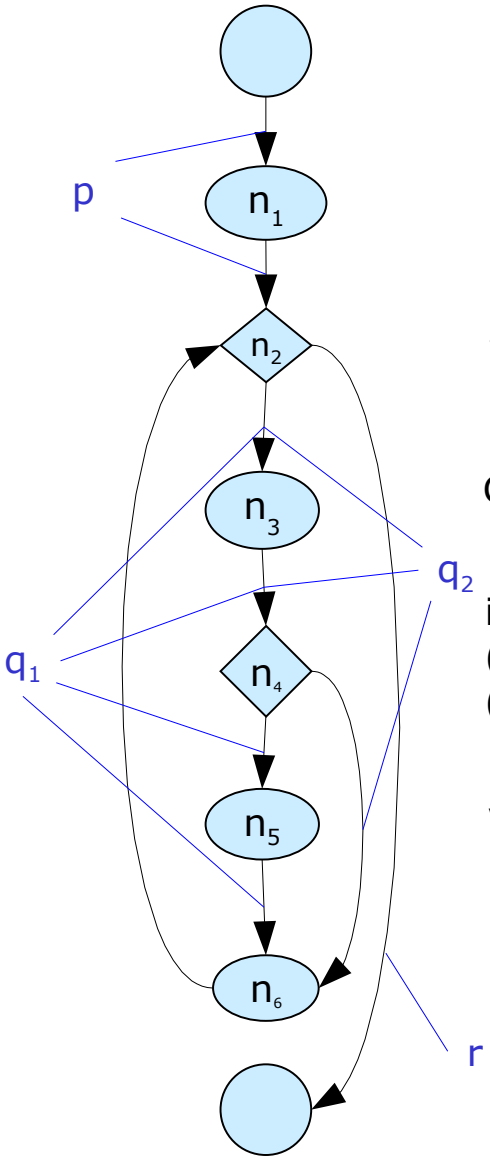
Brute force: Testbeispiele zur Ausführung **aller** unterschiedlichen Pfade im Programm

Beispiel:

Jeder Schleifendurchlauf trägt entweder zu Vokal oder zu Konsonant bei.

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

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



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Allgemeiner Pfadüberdeckungstest

Eigenschaften:

- Pfadanzahl bei unbestimmten Wiederholungen (while, …) nicht beschränkt.
- 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:

- Praktische Bedeutung höchstens für Programmteile ohne Schleifen

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Boundary Interior – Test

- **Idee**: Unterscheide Durchlauf nur keinmal, einmal, mehrmals
- Eingeschränkter Pfadüberdeckungstest, für Programme ohne Schleifen sogar identisch
- Zwei Gruppen von Pfaden für jede Schleife im Programm:
  - Grenztest-Gruppe (*boundary tests*):
    Pfade, welche die Schleife nur einmal durchlaufen
    Testet Kombination init -- body
  - Gruppe zum Test der Reinitialisierung (*interior tests*):
    Pfade, welche die Schleife mindestens einmal wiederholen
    Testet Kombination reinit -- body
- Praktisch anwendbar (im Gegensatz zum allgemeinen Pfadüberdeckungstest)
- **Strukturierter Pfadtest**

  ➔ Verallgemeinerung des *Boundary Interior* Tests

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Bedingungsüberdeckungstest

- **Ziel:** Analysiert und überprüft die Testbedingungen in Schleifen und Verzweigungen aus struktureller Perspektive
- **Ansatz**: Bedingung ist logische Verknüpfung atomarer boolescher Funktionen. Testfälle sollen verschiedene Kombinationen dieser atomaren Werte überdecken. Analyse des Termbaums.
- 3 verschiedene Varianten:

  Einfache Bedingungsüberdeckung:

  Überdeckt alle atomaren Werte einzeln (im Extremfall 2 Testfälle)

  Mehrfach-Bedingungsüberdeckung:

  Überdeckt alle möglichen Kombinationen atomarer Werte (im Extremfall 2^n Testfälle)

  Minimale Mehrfach-Bedingungsüberdeckung:

  Jede Bedingung (ob atomar oder nicht) muss für sich überdeckt sein

Bewertung:

- Einfache Bedingungsüberdeckung:
  - weder Zweig- noch Anweisungsüberdeckung enthalten
  - sehr schwaches Kriterium

- Mehrfach-Bedingungsüberdeckung:
  - Zweigüberdeckung ist enthalten, jedoch sehr aufwändig

- Minimale Mehrfach-Bedingungsüberdeckung:
  - Wie einfache BÜ, aber beachtet die hierarchische Struktur
  - Es müssen nicht nur die Blätter (Atome), sondern auch alle Teilbäume des Ausdrucksbaums überdeckt sein
  - Sinnvolle Weiterentwicklung des Zweigtests (entspricht Überdeckung des Wurzelknotens)

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

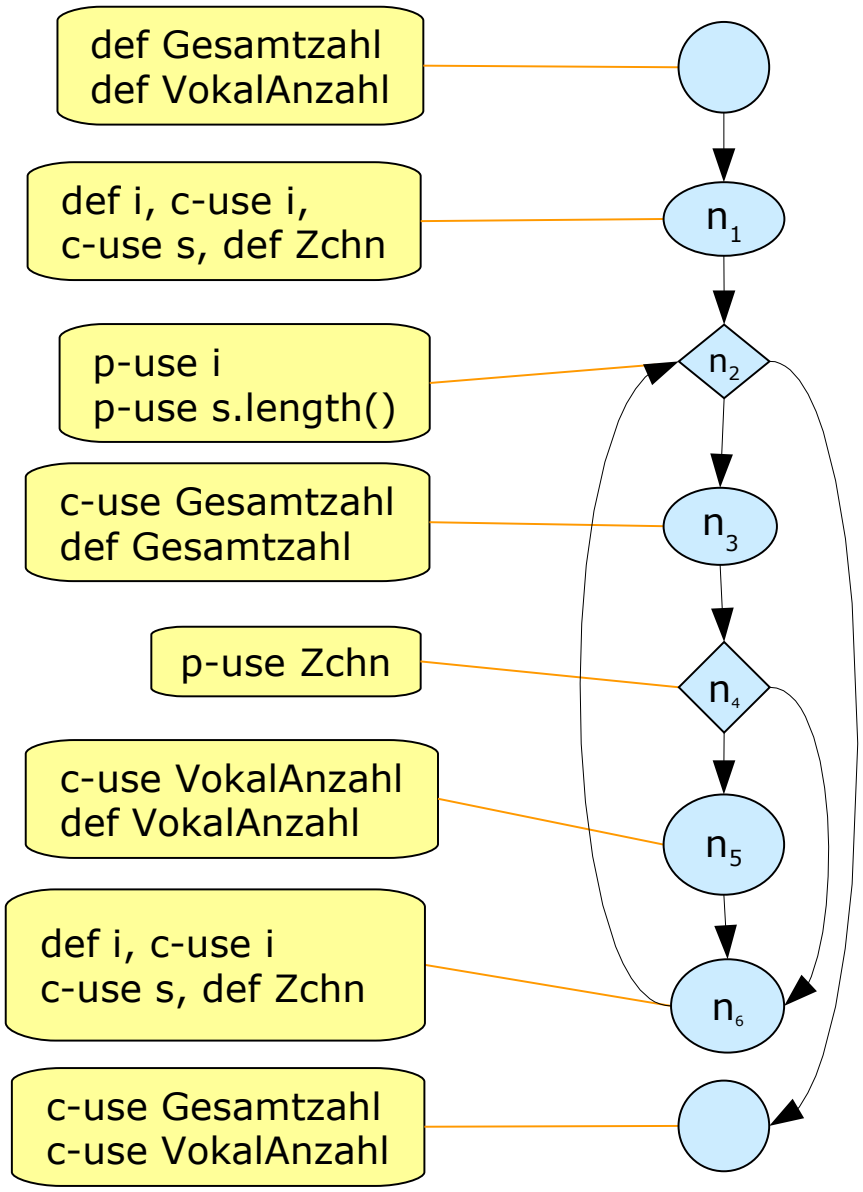
- Ebenfalls dynamisches Strukturtestverfahren
- Im Gegensatz zu kontrollflussorientierten Verfahren werden Datenbenutzungen und damit eher globale Aspekte getestet.
- Analyse der Programmdynamik an Hand der Dynamik der in Variablenwerten gespeicherten Programmzustände
  - Aufstellen des Datenflussdiagramms
    - Sichtbarkeit und Lebensdauer von Bezeichnern
    - Variablenidentitäten: Gültigkeitskontext und Kontrollfluss
    - lesende und schreibende Zugriffe auf Variablen (use, def)
    - Unterscheidung lesender Zugriffe in Anweisungen und Bedingungen (c-use, p-use)
- Eignen sich besonders für den Test von Datenobjekt- und Datentypmodulen sowie Klassen.
- Nur wenige Testwerkzeuge vorhanden.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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*)
  - Auswirkung auf Wert anderer Variablen (Programmzustand)
- Zugriff zur Berechnung von Wahrheitswerten in Bedingungen (*predicate-use, p-use*)
  - Auswirkung auf Wert der Testbedingung (Kontrollfluss)




```
i=0;
Zchn=s.charAt(i++)

while(i<s.length())

Gesamtzahl += 1;

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

VokalAnzahl += 1;

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

Datenflussgraph

- Knoten: jede Verwendung jeder Variablen B im Quelltext:
  (def B) (c-use B) (p-use B)
- Kanten: von (def B) zu allen nachfolgenden (use B), gelegentlich auch eine Kante pro möglichem Kontrollfluss
- Kantenmarken: Angabe des / der möglichen Kontrollflüsse (als Pfade im Kontrollflussgraphen)

Datenflussorientierte Test-Verfahren

- ***all defs* Kriterium**
  - Testfälle sind so zu wählen, dass jeder Wertzuweisung an eine Variable auch eine Wertbenutzung folgt.
  - Zu jeder Variablen gibt es einen Testfall, in welchem die Variable wenigstens einmal geschrieben und gelesen wird.
  - **Idee**: Jede Variable hat einen „Zweck", eine Semantik, die wenigstens einmal exemplarisch geprüft wird.
- Spezialfall: Kontrolle, ob alle definierten Variablen auch verwendet werden (statischer Test, den der Compiler durchführen kann)
  - Charakterisiert durch fehlende abgehende Pfeile im DFD.
  - Variablen, die nicht benutzt werden, weisen auf Programmfehler hin.
  - Problem fällt beim Aufstellen der Testfälle auf.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- ***all p-uses* Kriterium**
  - Testfälle sind so zu wählen, dass jede Kombination einer Wertzuweisung mit jeder prädikativen Nutzung dieses Variablenwerts überdeckt wird
  - Zu jedem Kontrollfluss, in dem eine Variable geschrieben und später in einer Bedingung gelesen wird, gibt es einen Testfall, welcher diesen Kontrollfluss abdeckt.
  - **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
  - Zu jedem Kontrollfluss, in dem eine Variable geschrieben und später in einer Berechnung gelesen wird, gibt es einen Testfall, welcher diesen Kontrollfluss abdeckt.
  - **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.
  - **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.
  - **Idee**: Prüfen der Auswirkung einer Wertzuweisung an eine Variable auf alle davon abhängenden Bedingungen und exemplarischer Test von nur berechnungsrelevanten Variablen

- ***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
  - Nachteil: Betrachtung von einzelnen Werten, dadurch werden keine Wechselwirkungen oder Abhängigkeiten getestet

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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)

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Test von GUI-Komponenten und anderen ereignisbasierten Teilen

Probleme

- Ereigniskonzept ist weitere Kommunikationsebene im Programm.
- Ereigniskonzept modelliert Nebenläufigkeit von Programmteilen.
- Ereignisse werden oft durch manuelle Nutzerinteraktionen ausgelöst und sind so nur bedingt automatisierbar.
- Ereignisse sind kaskadierend auf verschiedenen Abstraktionsebenen implementiert (Bsp.: MouseEvent vs. buttonPressed)

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Test von GUI-Komponenten und anderen ereignisbasierten Teilen

Modellierung und Tests

- Ereignisse werden über Ereignis**kanäle** verteilt, die zur Designzeit als potenzielle Wege der Ereignispropagierung angelegt werden.
- Ereignisse fokussieren gewöhnlich auf einen speziellen **Anwendungsfall** mit eingeschränktem Zugriff auf die Datenlandschaft.
- Entspricht speziellem **Programmzustand** mit eingeschränkten funktionellen Möglichkeiten
  - Aufstellen eines entsprechenden **Zustandsdiagramms** mit entsprechenden **Knoten** sowie **Vor**- und **Nach**bedingungen
  - Das Generieren der Testfälle sollte dieser Strukturierung folgen
  - Oft hierarchischer Zugang sinnvoll.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Beispiel: Planung von GUI-Tests [Memon, Pollack, Lou Soffa, 1999]

- Herausfinden der **primitiven Operationen**, die in 1-1-Korrespondenz mit direkten Mausaktionen stehen.
- Finde unter ihnen die **Expansionsoperatoren** (etwa Pulldown-Menüs)
  - expandieren die Menge der verfügbaren Aktionen/Zustände und beschneiden zugleich andere Expansionsmöglichkeiten
  - andere Operatoren heißen **Interaktionsoperatoren**, da sie mit der unterliegenden Software interagieren.
- Zu jeder Expansionssequenz wird ein **Zwischenoperator** konstruiert.
  - Bsp.: Edit expandiert zu Cut, Paste (primitiv), Edit+Cut, Edit+Paste (Interaktionssequenzen)
  - Ansatz vermeidet Generieren von Testfällen nur für Edit.

- Finde unter diesen die **abstrakten Operatoren**, die nach Aktivieren die GUI-Interaktion monopolisieren (Bsp: Edit+Preferences)
  - Typische Struktur:

  - Ansatz ist selbstähnlich, da Aktion dieselbe Struktur hat.
  - Cluster- und Subclusterstruktur entspricht einer baumartigen Struktur der Aktionen und Teilaktionen

1. Verschiedene Aktionen können gemeinsame Teilaktionen haben.
   - Bsp.:Open, SaveAs, Open.Select, Open.Up, Open.Home, Open.OK, SaveAs.Select, SaveAs.Up, SaveAs.Home, SaveAs.OK
   - Identifiziere **gemeinsame Teilaktionen**, da für diese nur einmal Testfälle zu generieren sind.

- Planung der **Testfälle**: Zunächst Plan für die „höherwertigen" abstrakten Operatoren, schrittweise Verfeinerung
  - Beispiel MS-WordPad: 325 primitive Operatoren, aber nur 32 der obersten Abstraktionsstufe
  - Vor- und Nachbedingungen sind auf dieser Ebene zudem meist einfach zu identifizieren
  - Teilpläne lassen sich daraus werkzeuggestützt generieren
  - abstrahiert von low-level-Details wie Fonts, Farben usw.
  - funktionale Änderungen am GUI können einfach in der Testsuite berücksichtigt werden.

Testen von Klassen

Im [Balzert] wenig systematisch aufbereitet, dort im Kap. 5.13.

- Alle bisherigen Testverfahren waren auf den funktionalen Test von Methoden unter dem imperativen Paradigma ausgerichtet.
- Kommunikation zwischen den Methoden desselben Objekts erfolgt sowohl über die Aufrufparameter als auch den Zustand der Objektattribute (Objekt ist immer implizit ein Parameter).

Kleinste sinnvolle Testeinheit im OO-Bereich ist also die Klasse.

Weitere Besonderheiten von Tests im OO-Bereich

- Wiederverwendbarkeitskonzept
  - Einsatzzweck von Klassen oft nicht genau umrissen
  - Allgemeinheit führt zu vielen möglichen Testfällen
- Vererbung von Attributen und Methoden
  - Redundanz wird eliminiert zu Lasten von zusätzlichen Abhängigkeiten
- Polymorphismus und dynamische Bindung
  - neue Testverfahren nötig
  - Test jeder möglichen Bindung für Polymorphismus nötig
- folgende Arten von Klassen sind zu unterscheiden:
  - normale Klassen
  - abstrakte Klassen
  - parametrisierte Klassen