Dynamische Testtechniken

White Box:

- Abdeckung
- Kontrollflusstesten

Dr. Christoph Steindl

Abdeckung

- Abdeckung aller Anweisungen (P_1)
  - alle Anweisungen des Programms ausgeführt
  - eine Mindestanforderung vor der Auslieferung von Software
- Abdeckung aller Verzweigungen (P_2)
  - alle Sprungalternativen (zumindest einmal) ausgeführt
- Abdeckung aller Bedingungen bei Verzweigungen
  - Bei komplexen Verzweigungsbedingungen müssen alle Teilbedingungen einmal wahr und einmal falsch gewesen sein.
- Abdeckung aller Pfade (P_∞)
  - alle möglichen Kontrollflusspfade durch das Programm ausgeführt
  - im allgemeinen unmöglich, unerreichbar
  - 100% Pfadabdeckung ist der (unerfüllbare) Traum
- Werkzeuge messen meist nur Abdeckung auf Anweisungs- bzw. Verzweigungsebene
- Frage: Warum ist P_2 stärker als P_1?

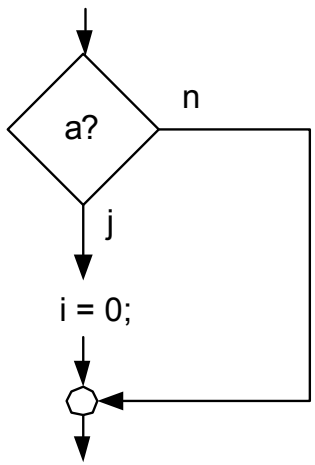
Unterschiede der Abdeckung

- Anweisungsabdeckung ⇔ Verzweigungsabdeckung

```
if (a) {
   i = 0;
}
```

2 Anweisungen

2 Pfade



Fall a == true:

100% der Anweisungen

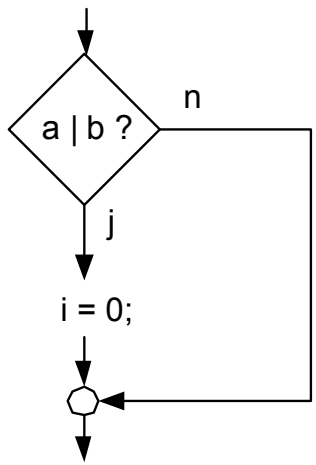
50% der Pfade

Unterschiede der Abdeckung

- Verzweigungsabdeckung ⇔ Bedingungsabdeckung

```
if (a | b) {
   i = 0;
}
```

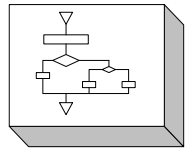
2 Anweisungen
2 Pfade
4 Bedingungen



a	b	a | b
T	T	T
T	F	T
F	T	T
F	F	F

Fall a == true und b == true:
100% der Anweisungen
50% der Pfade
25% der Bedingungen

Anzahl der Pfade



- Jede Verzweigung im Programm verdoppelt die Anzahl der zu testenden Pfade.

```
if (expr) { ...   <= 1. Pfad
} else { ...      <= 2. Pfad
}
```

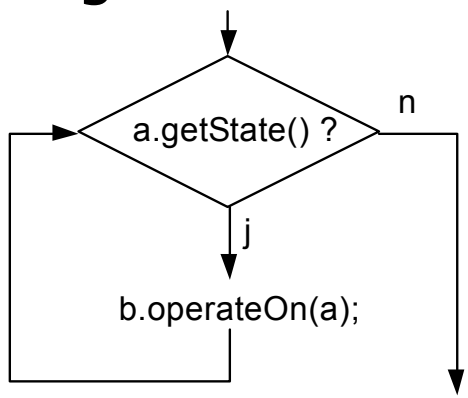
- Jede Schleife multipliziert die Anzahl der zu testenden Pfade mit der Anzahl der Schleifendurchläufe.

```
while (i < 10) {
    ...
    if (i = 5) { ...
    }
}
```

Unterschiede der Abdeckung

- Bedingungsabdeckung ⇔ Pfadabdeckung

```
while (a.getState()) {
  b.operateOn(a);
}
```



2 Anweisungen

unbegrenzte Anzahl von Pfaden

Fall 1 Durchlauf:

100% der Anweisungen

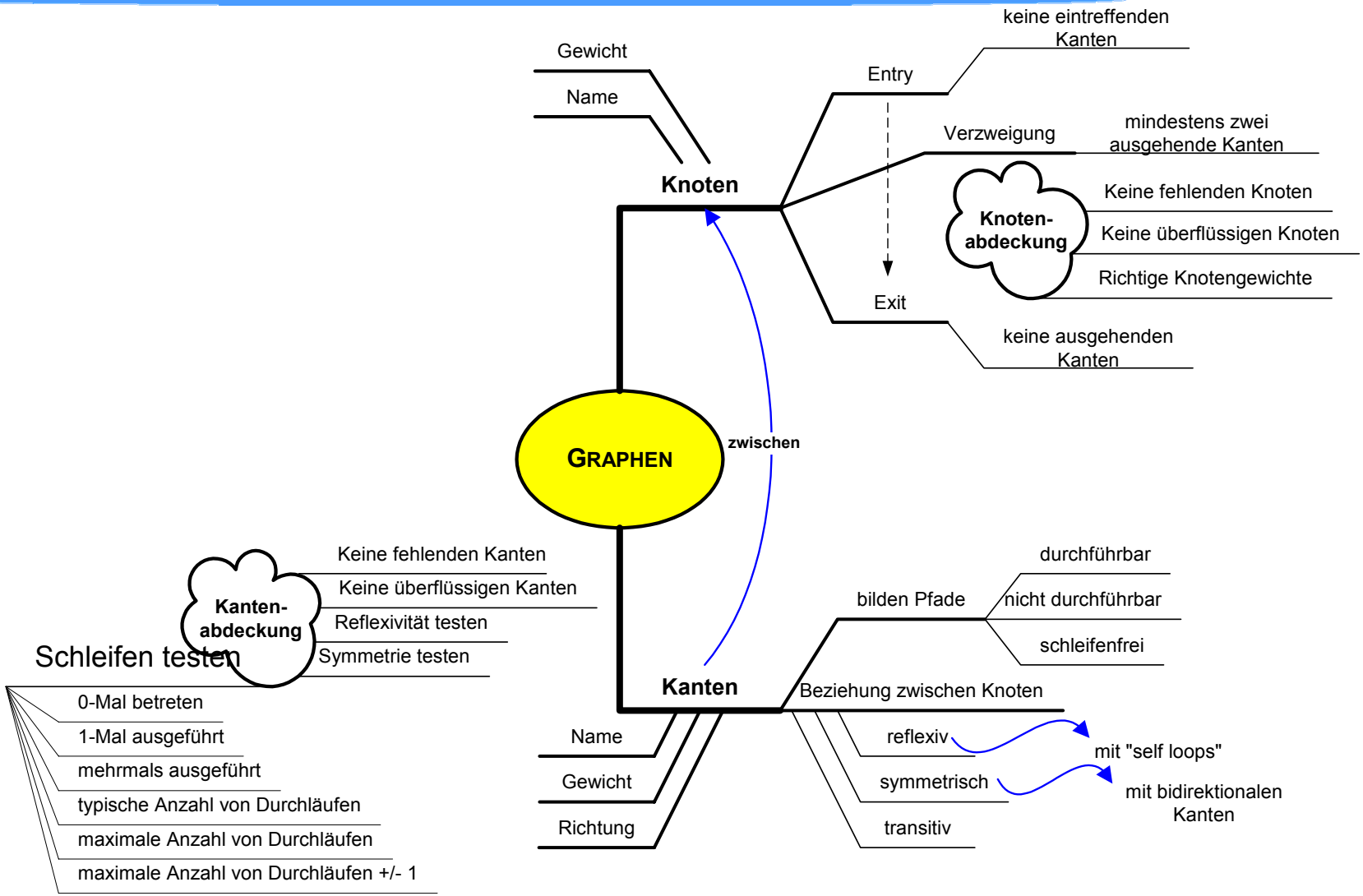
??% der Pfade

100% der Bedingungen

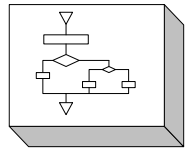
Werkzeuge für die Berechnung der Abdeckung

- Instrumentierend
  - berechnet Anweisungs- und Verzweigungsabdeckung
  - Clover (Cenqua, www.cenqua.com/clover)
  - JCoverage (www.jcoverage.com)
- Programmausführung in spezieller JVM
  - berechnet Anweisungsabdeckung
  - Jtest (ParaSoft, www.parasoft.com)
  - TrueCoverage (Compuware Numega, www.compuware.com)
  - Coverage (JProbe, www.jprobe.com)
  - PureCoverage (Rational, www.ibm.com/software/rational)
- und weitere

Grundlagen über Graphen

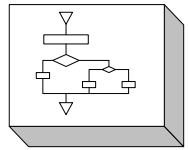


Kontrollflusstesten



- Fehlerhypothese:
  - Kontrollfluss bestimmt, wie Operationen ausgeführt werden mit Verzweigungen und Schleifen
    => Teste den Kontrollfluss sorgfältig!
  - Fehler oft durch falsche Pfade durch das Programm
    => Paradebeispiel „Spaghetti-Code"
    => moderne Programmiersprachen (z.B. Java) und strukturierte Programmierung helfen viel
- Voraussetzung:
  - Wissen über die interne Struktur des Programms (d.h. den Quellcode)
    => White-Box-Testen
- Ziele:
  - mögliche Pfade durch das Programm testen und
  - dabei alle Anweisungen mindestens einmal ausführen (oder andere Strategie der Abdeckung)

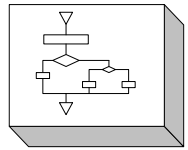
Vorgehensweise



- Vorgehensweise
  1. Ablaufdiagramm erstellen
  2. Testpfade auswählen
  3. Eingabewerte finden, die zu den gewünschten Testpfade führen
  4. Ausgabewerte für jeden Testfall vorhersagen und aufschreiben
  5. Testfall ablaufen lassen
  6. Ergebnisse verifizieren
     - Erwartetes Ergebnis geliefert?
  7. Pfad verifizieren
     - Erwarteter Pfad abgearbeitet?

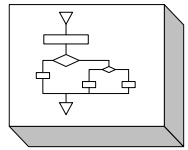
     => verhindert Zufallstreffer

Vorteile und Nachteile



- Kontrollflusstesten ist am erfolgreichsten beim Unit Testing:
  - fängt 65% aller noch nicht entdeckten Fehler
  - typische Teststrategie:
    - Erreichung der Abdeckung auf Anweisungsebene bzw. Verzweigungsebene
    - zusätzlich Endbereichsprüfungen bei Schleifen, um möglichst viele Fehler bei Schleifen zu finden.
- Je größer die Programme werden, desto unbrauchbarer ist Kontrollflusstesten:
  - Ganze Programmsysteme werden so gut wie nie mittels Kontrollflusstesten getestet.
- Fehlende Anforderungen, fehlende Pfade, fehlende/überflüssige Features nicht aufgedeckt, wenn der Programmierer auch die Rolle des Testers hat.
- Zufallstreffer können das Erkennen von Fehlern verhindern.

Beispiel: Tirolerisch

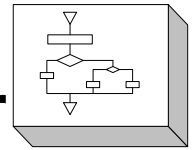


- Schreiben Sie ein Java-Programm, das einen deutschen Text in Tirolerisch umwandelt und dabei folgende Zeichenersetzungen vornimmt:

Zeichenfolge	**ersetzen durch**
"st" am Wortende (danach folgt ein Leerzeichen oder Punkt)	"sch"
"st" am Wortanfang oder in der Wortmitte	"scht"

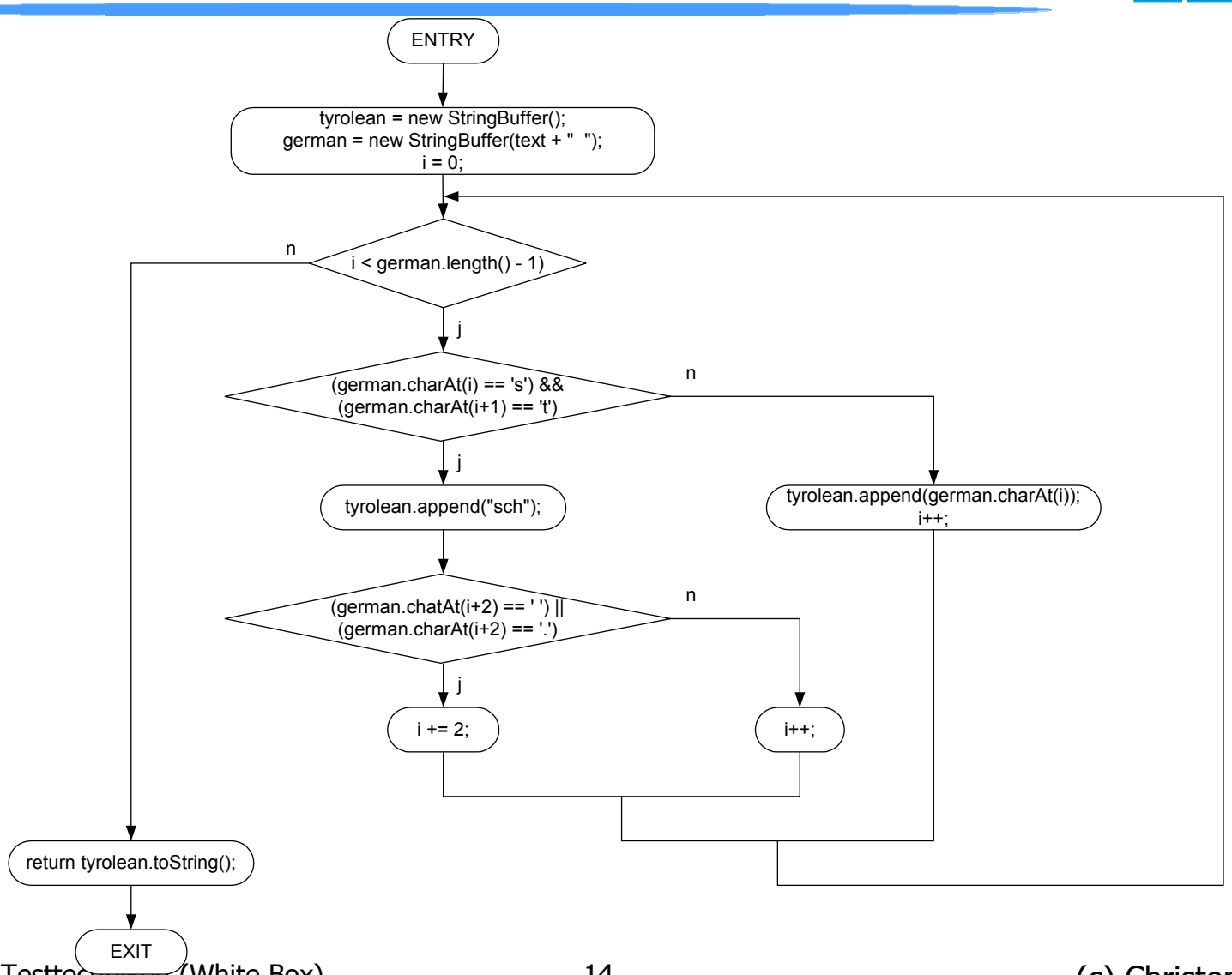
- **Beispiel:**
  - **"Was ist los?" wird zu "Was isch los?"**
  - **"lustig" wird zu "luschtig"**

Beispiellösung von einem Tiroler

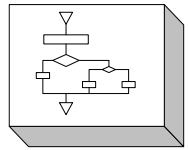



```
static String toTyrolean (String text) {
    StringBuffer tyrolean = new StringBuffer(), german = new StringBuffer(text + "    ");
    int i = 0;
    while (i < german.length() - 1) {
        if ((german.charAt(i) == 's') && (german.charAt(i + 1) == 't')) {
            tyrolean.append("sch");
            if ((german.charAt(i + 2) == ' ') || (german.charAt(i + 2) == '.'))
                i += 2;    // 't' ueberspringen
            else
                i++;    // 't' auch kopieren
        } else {
            tyrolean.append(german.charAt(i));  // kein 'st', also normal kopieren
            i++;
        }
    }
    return tyrolean.toString();
}
```

Beispiel: Ablaufdiagramm

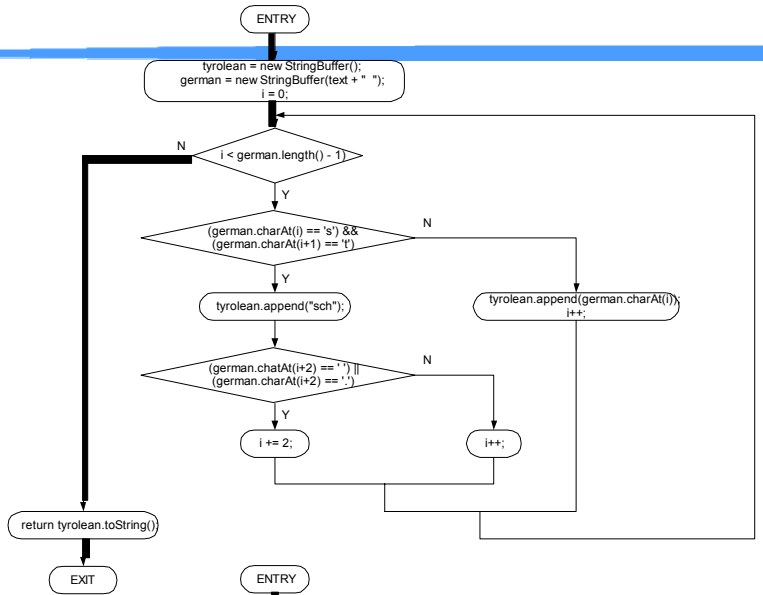


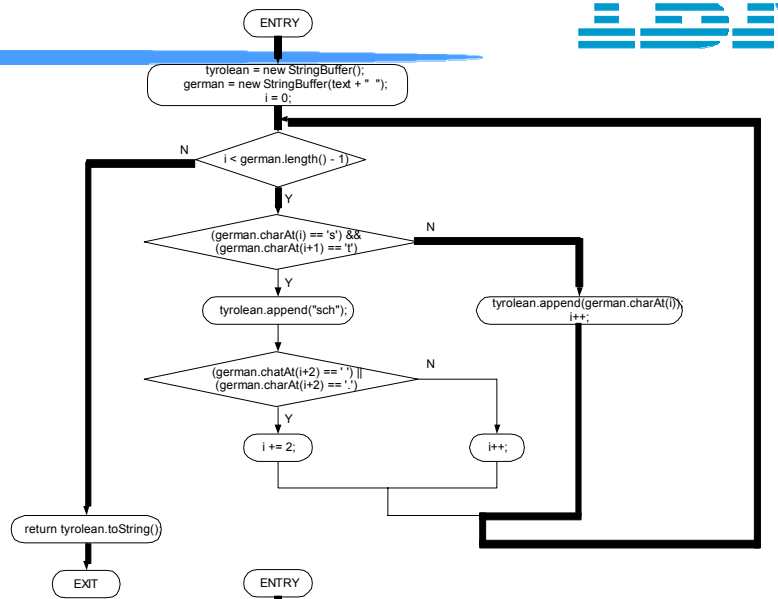
Hinweise für die Pfadauswahl

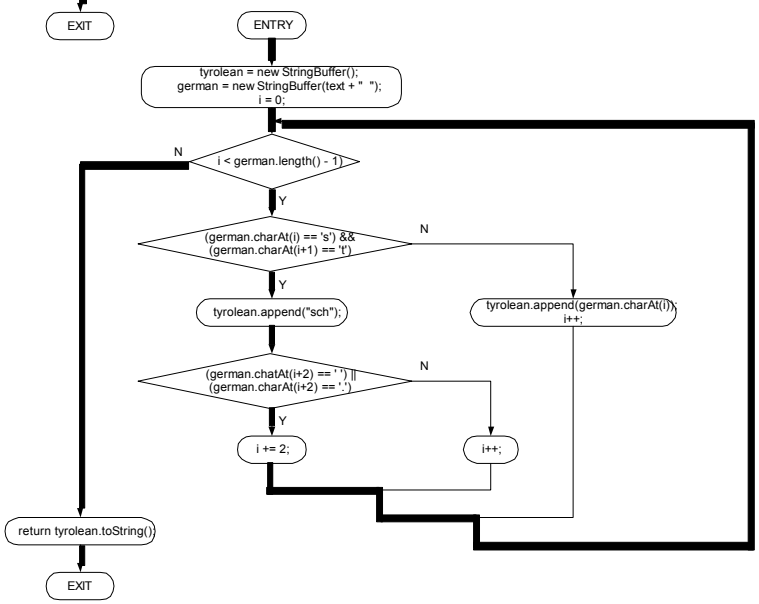


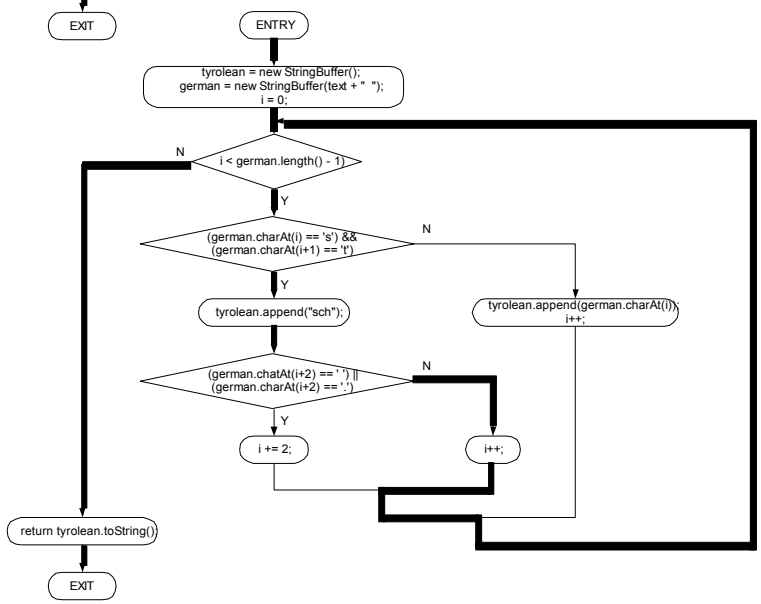
1. Zuerst die einfachsten, sinnvollen Pfade
2. Dann weitere Pfade durch kleine Variationen an den vorigen Pfaden
   - eher Pfade ohne Schleifen
   - eher kurze, einfache und sinnvolle Pfade
3. Weitere Pfade ohne offensichtliche Bedeutung erst, wenn sie zum Erreichen der Kriterien der gewählten Teststrategie (z.B. Abdeckung auf Verzweigungsebene) notwendig sind.
   - Zuerst sollte man sich aber die Frage stellen
     - warum diese Pfade notwendig wären und
     - warum die Kriterien mit den sinnvollen Pfaden nicht erreicht wurde.
4. Extremfälle für Schleifen
   - kein Durchlauf, einer, N-1 und N

- Lieber viele einfache und offensichtliche Tests als wenige komplizierte Tests!



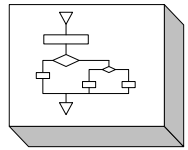






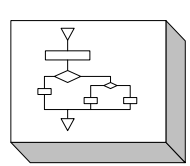
Testpfade auswählen

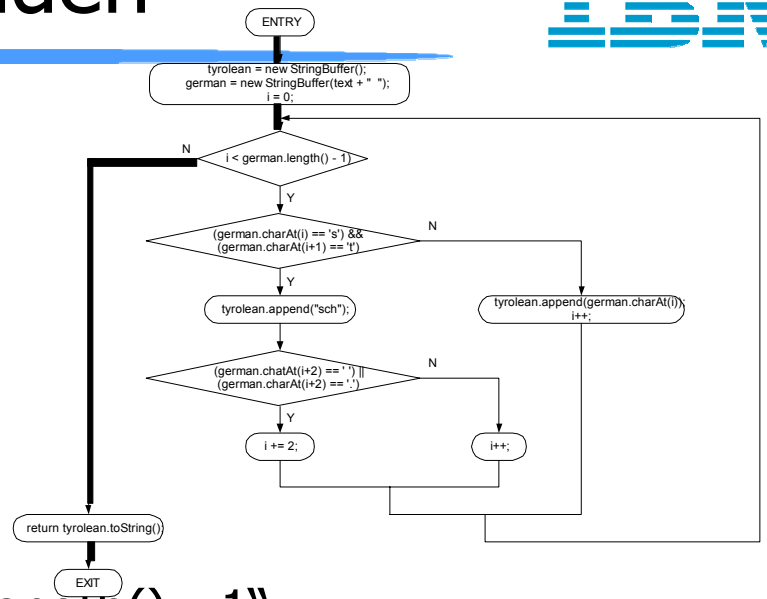
Eingabewerte finden



- Bestimmen der Eingabewerte, die zur Abarbeitung der Pfade führen
  - Bedingungen beachten
    - Hausverstand notwendig!
  - Probleme:
    - nicht-durchführbare Pfade
    - Fehler im Programm und in der Spezifikation
    - korrelierende Bedingungen: (A == true) => (B == true)
      - Wahl des Wahrheitswertes von A legt den Wahrheitswert von B fest
      - Segmente mit korrelierenden Bedingungen zu Pfaden zusammenhängen
      - Pfade mit widersprüchlichen Bedingungen entfernen

Eingabewerte für ersten Testpfad finden

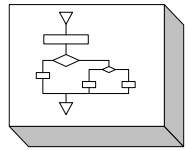




- Erster Testpfad:
  - Schleife nicht betreten

- Mögliche Eingabewerte:
  - Schleifenbedingung „i < german.length() - 1“
    muss false sein, d.h.
    i >= german.length() - 1
    i hat anfangs den Wert 0, d.h.
    german.length() <= 1
    german enthält anfangs zumindest zwei Leerzeichen, d.h.
    german.length() ist mindestens 2
    => Widerspruch => dieser Pfad ist nicht ausführbar.

Anzahl der Pfade im Beispiel



- Schleife
  - bei einer Eingabe der Länge n maximal n+1 Iterationen
  - In der Schleife enthalten ist eine Abfrage, die wiederum eine Abfrage enthält, also 4 Pfade in der Schleife.
- Insgesamt also maximal 4 * (n + 1) Pfade.
- Bei einer Eingabe von 10 Zeichen sind das 44 Pfade, bei 249 Zeichen sind das 1000 Pfade!
- Nicht alle Pfade haben selbe Wahrscheinlichkeit, Fehler aufzudecken
  - bei Schleifen sind 0, 1 und n Durchläufe sehr effektiv