Zürcher Hochschule
für Angewandte Wissenschaften

zh aw School of Engineering

Bachelor of Science (BSc) in Informatik

Modul Advanced Software Engineering 2 (ASE2)

LE 09 – Software Testing

5 Dynamischer Test

Institut für Angewandte Informationstechnologie (InIT)
Walter Eich (eicw) / Matthias Bachmann (bacn)
https://www.zhaw.ch/de/engineering/institute-zentren/init/

Agenda

Lernziele nach Syllabus ISTQB CTFL (1/2)

4.1 Kategorien von Testverfahren

FL-4.1.1 (K2) Die Eigenschaften, Gemeinsamkeiten und Unterschiede zwischen Black-Box-Testverfahren, White-Box-Testverfahren und erfahrungsbasierten Testverfahren erklären können

4.2 Black-Box-Testverfahren

FL-4.2.1 (K3) Die Äquivalenzklassenbildung anwenden können, um Testfälle aus vorgegebenen Anforderungen abzuleiten

FL-4.2.2 (K3) Die Grenzwertanalyse anwenden können, um Testfälle aus vorgegebenen Anforderungen abzuleiten

FL-4.2.3 (K3) Entscheidungstabellentests anwenden können, um Testfälle aus vorgegebenen Anforderungen abzuleiten

FL-4.2.4 (K3) Zustandsübergangstests anwenden können, um Testfälle aus vorgegebenen Anforderungen abzuleiten

FL-4.2.5 (K2) Erklären können, wie man Testfälle aus einem Anwendungsfall ableitet

Lernziele nach Syllabus ISTQB CTFL (2/2)

4.3 White-Box-Testverfahren

FL-4.3.1 (K2) Anweisungsüberdeckung erklären können

FL-4.3.2 (K2) Entscheidungsüberdeckung erklären können

FL-4.3.3 (K2) Die Bedeutung von Anweisungs- und Entscheidungsüberdeckung erklären können

4.4 Erfahrungsbasierte Testverfahren

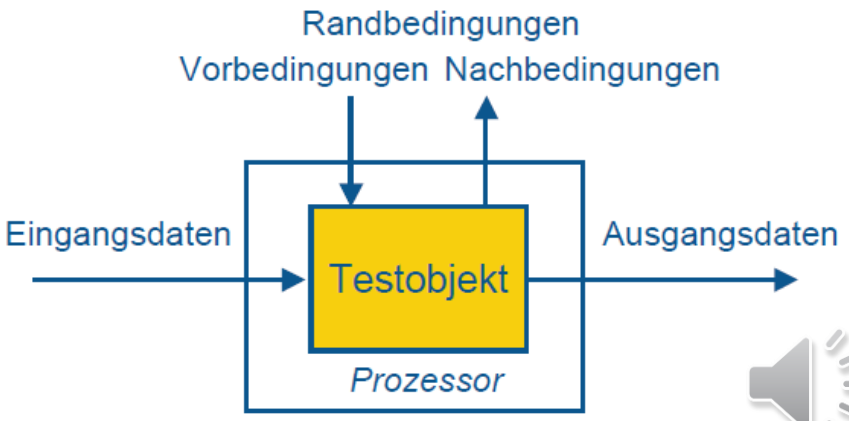
FL-4.4.1 (K2) Die intuitive Testfallermittlung erklären können

FL-4.4.2 (K2) Exploratives Testen erklären können

FL-4.4.3 (K2) Checklistenbasiertes Testen erklären können

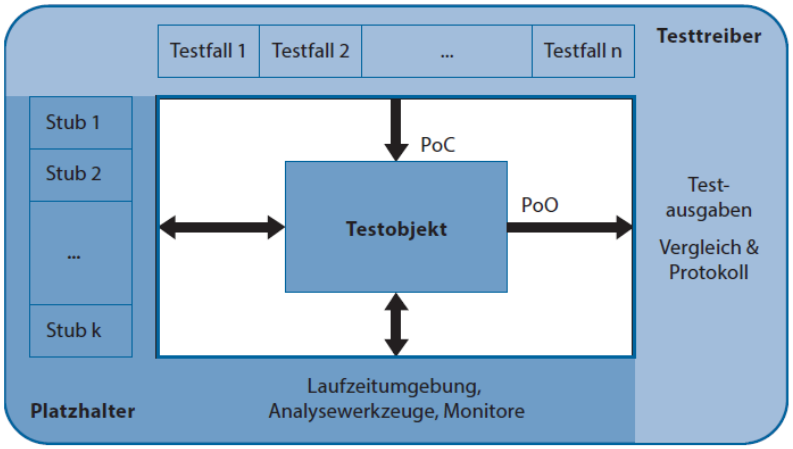
5 Dynamischer Test

- Programme sind statische Beschreibungen von dynamischen Prozessen (Berechnungen).
- Statische Tests prüfen die Testobjekte «an sich».
  - Artefakte des Entwicklungsprozesses, z.B. informelle Texte, Modelle, formale Texte, Programmcode, ...
- Dynamische Tests prüfen die durch «Interpretation» einer Beschreibung (Testobjekt) resultierenden Prozesse.
- Das Testobjekt wird im dynamischen Test also auf einem Prozessor «ausgeführt».
  - Bedingungen und Voraussetzungen für die Tests und die Ziele festlegen
  - Testfälle spezifizieren
    - Bereitstellen von Eingangsdaten
    - Beobachten der Ausgangsdaten
  - Testausführung festlegen

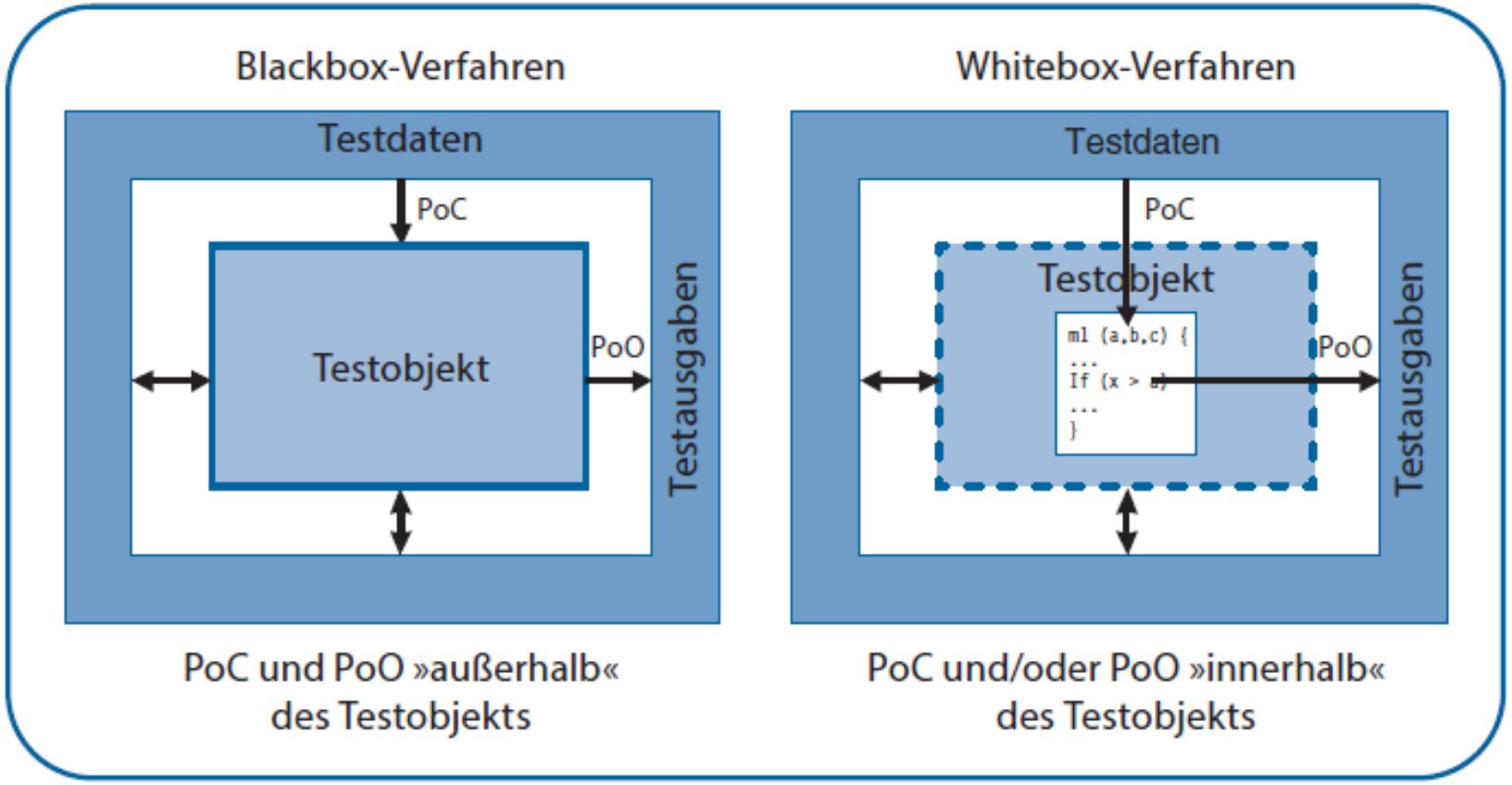


5 Testrahmen

- Das Testobjekt wird meist weitere Programmteile über definierte Schnittstellen aufrufen.
- Diese Programmteile werden durch Platzhalter (Stubs, Mock-Objekte, Stellvertreter) realisiert, wenn sie noch nicht fertig implementiert und damit einsatzbereit sind oder für diesen Test des Testobjekts nur simuliert werden sollen.
- Platzhalter simulieren das Ein-/Ausgabeverhalten des eigentlich aufzurufenden Programmteils.



5 Blackbox- vs. Whitebox-Testverfahren



PoC = Point of Control PoO = Point of Observation

5 Blackbox-Testverfahren (Testobjekt «undurchsichtig»)

- Blackbox-Testverfahren allgemein
  - Keine Informationen über den Programmtext und den inneren Aufbau
  - Beobachtet wird das Verhalten des Testobjekts von aussen (→ PoO - Point of Observation liegt ausserhalb des Testobjekts).
  - Steuerung des Ablaufs des Testobjektes nur durch die Wahl der Eingabetestdaten (→ PoC - Point of Control liegt ausserhalb des Testobjektes)
  - Blackbox-Testverfahren können sowohl auf funktionale als auch auf nicht funktionale Tests angewendet werden.
- Spezifikationsorientierte Verfahren («funktionale» Testverfahren)
  - Modelle bzw. Anforderungen (Use Cases, User Stories etc.) an das Testobjekt, ob formal oder nicht formal, werden zur Spezifikation des zu lösenden Problems, der Software oder ihrer Komponente herangezogen.
  - Von diesen Modellen können systematisch Testfälle abgeleitet werden.

5 Whitebox-Test (Testobjekt «durchsichtig»)

- Testfälle können auf Grund der Programmstruktur des Testobjektes gewonnen werden («strukturorientierte» Testverfahren)
  - Informationen über den Aufbau der Software werden für die Ableitung von Testfällen verwendet, beispielsweise der Code und der Algorithmus.
  - Überdeckungsgrad des Codes kann für vorhandene Testfälle gemessen werden.
  - Weitere Testfälle können zur Erhöhung des Überdeckungsgrades systematisch abgeleitet werden.
  - Während der Ausführung der Testfälle wird der innere Ablauf im Testobjekt analysiert (→ PoO - Point of Observation liegt innerhalb des Testobjekts).
  - Eingriff in den Ablauf im Testobjekt möglich, z.B. wenn für Negativtests die zu provozierende Fehlbedienung über die Komponentenschnittstelle nicht auslösbar ist (→ PoC - Point of Control kann innerhalb des Testobjekts liegen).
  - Die Whitebox-Testverfahren lassen sich auf den unteren Teststufen wie Komponenten- und Integrationstest anwenden.

5 Erfahrungsbasierte Testfallermittlung

- Erfahrungsbasierte Verfahren
  - Nutzen Wissen und die Erfahrung von Menschen zur Ableitung der Testfälle
  - Wissen von Testern, Entwicklern, Anwendern und Betroffenen über die Software, ihre Verwendung und ihre Umgebung
  - Wissen über wahrscheinliche Fehler und ihre Verteilung

Agenda

5 Dynamischer Test

5.1 Blackbox-Testverfahren

5.2 Whitebox-Testverfahren

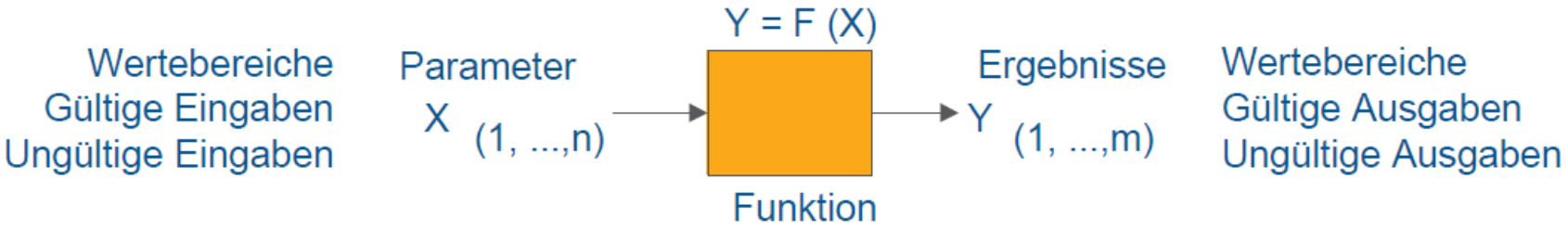
5.3 Erfahrungsbasierte Testfallermittlung

5.4 Auswahl von Testverfahren

5.5 Wrap-up

5.1 Blackbox-Verfahren - Übersicht

Spezifikationsorientierte Testfall- und Testdatenermittlung



- Äquivalenzklassenbildung
  - Repräsentative Eingaben
  - Gültige Dateneingaben
  - Ungültige Dateneingaben
- Grenzwertanalyse
  - Wertebereiche
  - Wertebereichsgrenzen

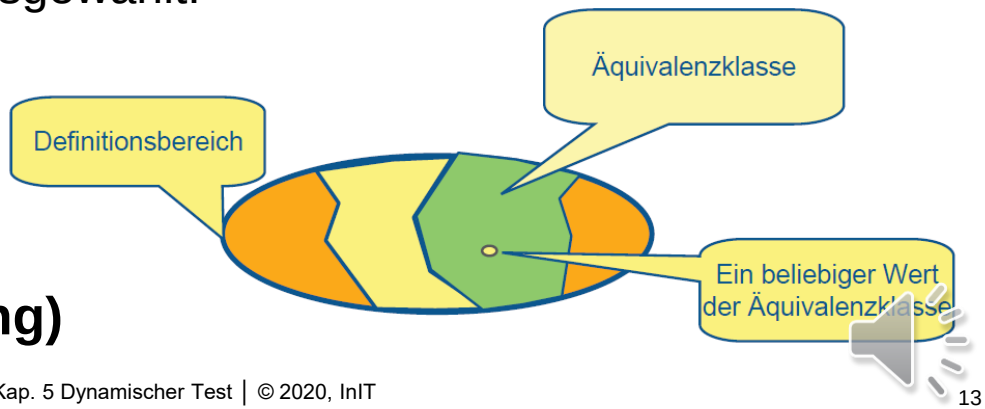
Diese Verfahren sollten bekannt sein ;-)

- Zustandsbezogener Test
  - Komplexe (innere) Zustände und Zustandsübergänge
- Anwendungsfallbasierter Test
  - Prozess-Abläufe und Benutzerinteraktionen
- Entscheidungstabellentest
  - Bedingungen und Aktionen

5.1.1 Äquivalenzklassenbildung

- Die Äquivalenzklassenbildung hat zum Ziel mit möglichst wenig Testfällen möglichst wirkungsvoll zu testen.
- Die Spezifikation wird nach Eingabegrössen und deren Gültigkeitsbereichen durchsucht.
  - Je Gültigkeitsbereich wird eine Äquivalenzklasse definiert
  - Tipp: Wann immer man vermutet, dass Eingabewerte innerhalb einer Äquivalenzklasse nicht gleichartig behandelt werden, sollte entsprechend in mehrere Äquivalenzklassen aufgeteilt werden.
  - Je «Mitte» einer Äquivalenzklasse wird ein beliebiges Element - ein Repräsentant - als Testfall ausgewählt.
- Vollständigkeit: Alle Repräsentanten sind getestet

(Äquivalenzklassenüberdeckung)



5.1.1 Äquivalenzklassenbildung – Beispiel 1

- Ein Sensor meldet an ein Programm die Motoröltemperatur. Ist die Öltemperatur (T) unter 50°C soll eine blaue Kontrolllampe leuchten, bei T über 150 °C soll eine rote Kontrolllampe leuchten. Sonst soll keine der Lampen leuchten.

Äquivalenzklasse		Soll-Resultat	Repräsentant
1	$T < 50°C$	blaue Lampe leuchtet	17°C
2	$50 \leq T \leq 150°C$	keine Lampe leuchtet	97°C
3	$T > 150°C$	rote Lampe leuchtet	159°C

- Wenn ein «Verdacht» besteht, dass z.B. ab Frost mit einem speziellen Verhalten des Sensors oder des Programms zu rechnen ist, so wird die Äquivalenzklasse 1 entsprechend in weitere Äquivalenzklassen aufgeteilt.

5.1.1 Äquivalenzklassenbildung – Beispiel 2

- Falls eine Beschränkung einen Wertebereich spezifiziert: eine gültige und zwei ungültige Äquivalenzklassen:
  - In der Spezifikation des Testobjekts ist festgelegt, dass ganzzahlige Eingabewerte zwischen 1 und 100 möglich sind (Definitionsbereich).

    Wertebereich der Eingabe: 1 <= x <= 100

    Gültige Äquivalenzklasse: 1 <= x <= 100

    Ungültige Äquivalenzklassen: x < 1 und x > 100

5.1.1 Äquivalenzklassenbildung – Beispiel 3

- Falls eine Beschränkung eine Anzahl von Werten spezifiziert: eine gültige und zwei ungültige Äquivalenzklassen

  – Spezifikation: Ein Mitglied eines Sportvereins muss sich mindestens einer Sportart zuordnen. Jedes Mitglied kann an maximal drei Sportarten aktiv teilnehmen.

  Gültige Äquivalenzklasse: 1 <= x <= 3 (1 bis 3 Sportarten)

  Ungültige Äquivalenzklassen:

  0 = x (keine Zuordnung zu einer Sportart)

  x > 3 (mehr als 3 Sportarten zugeordnet)

5.1.1 Äquivalenzklassenbildung – Beispiel 4

- Falls eine Beschränkung eine Menge von Werten spezifiziert, die möglicherweise unterschiedlich behandelt werden: für jeden Wert dieser Menge eine eigene gültige Äquivalenzklasse und zusätzlich insgesamt eine ungültige Äquivalenzklasse

  - Spezifikation: Im Sportverein gibt es folgende Sportarten: Fussball, Hockey, Handball, Basketball und Volleyball.

  - Gültige Äquivalenzklassen:
    - Fussball,
    - Hockey,
    - Handball,
    - Basketball,
    - Volleyball
  - Ungültige Äquivalenzklasse: alles andere, z.B. Badminton

5.1.1 Äquivalenzklassenbildung – Beispiel 5

- Falls eine Beschränkung eine Situation spezifiziert, die zwingend erfüllt sein muss: eine gültige und eine ungültige Äquivalenzklasse

  – Spezifikation: Jedes Mitglied im Sportverein erhält eine eindeutige Mitgliedsnummer. Diese beginnt mit dem ersten Buchstaben des Familiennamens des Mitglieds.

    Gültige Äquivalenzklasse: erstes Zeichen ist ein Buchstabe

    Ungültige Äquivalenzklasse: erstes Zeichen ist kein Buchstabe (z.B. eine Ziffer oder ein Sonderzeichen)

5.1.1 Testfälle mit mehreren Parametern

- Eindeutige Kennzeichnung jeder Äquivalenzklasse
  - gÄKn = gültige Äquivalenzklasse n
  - uÄKn = ungültige Äquivalenzklasse n

	TF1	TF2	...	TFn
gÄK1	x			
gÄK2		x		
...			x	
uÄK1				
uÄK2				x
...				

- Pro Parameter mindestens zwei Äquivalenzklassen
  - Eine mit gültigen Werten
  - Eine mit ungültigen Werten
- Bei n Parametern mit mi Äquivalenzklassen (i=1..n) gibt es

$$\prod_{i=1..n} m_i \text{ unterschiedliche Kombinationen (Testfälle)}$$

5.1.1 Testfälle minimieren

- Testfälle aus allen Repräsentanten kombinieren und anschliessend nach »Häufigkeit« sortieren.
  - Testfälle dann in dieser Reihenfolge priorisieren
  - Nur mit «benutzungsrelevanten» Testfällen testen
  - Testfälle bevorzugen, die Grenzwerte oder Grenzwert-Kombinationen enthalten
- Sicherstellen, dass jeder Repräsentant einer Äquivalenzklasse mit jedem Repräsentanten jeder anderen Äquivalenzklasse in einem Testfall zur Ausführung kommt.
  - d.h. paarweise Kombination statt vollständiger Kombination
- Minimalkriterium: Mindestens ein Repräsentant jeder Äquivalenzklasse in mindestens einem Testfall
- Repräsentanten ungültiger Äquivalenzklassen nicht mit Repräsentanten anderer ungültiger Äquivalenzklassen kombinieren.

5.1.1 Testendekriterium

- Eine spezifische Ausgangsbedingung für den Test nach der Äquivalenzklassenbildung lässt sich anhand der durchgeführten Tests der Repräsentanten der jeweiligen Äquivalenzklassen im Verhältnis zur Gesamtzahl aller definierten Äquivalenzklassen festlegen:

ÄK-Überdeckung = (Anzahl getestete ÄK / Gesamtzahl ÄK) * 100 %

  - Sind zum Beispiel 18 Äquivalenzklassen aus den Anforderungen bzw. der Spezifikation für ein Eingabedatum ermittelt worden und sind von diesen 18 nur 15 in den Testfällen getestet worden, so ist eine Äquivalenzklassen-Überdeckung von 83% erreicht.
  - ÄK-Überdeckung = (15 / 18) * 100% = 83,33 %

5.1.1 Vor- und Nachteile

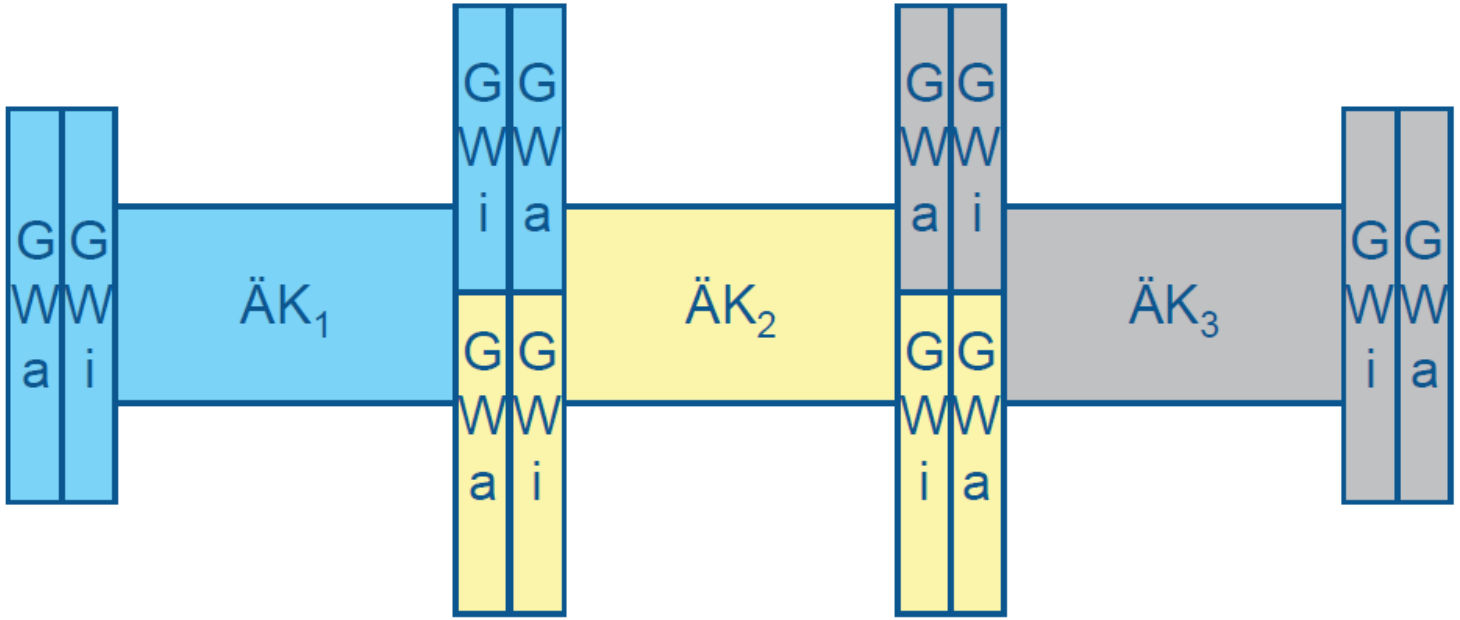
- **Vorteile**
  - Anzahl der Testfälle kleiner als bei unsystematischer Fehlersuche
  - Geeignet für Programme mit vielen verschiedenen Ein- und Ausgabebedingungen
- **Nachteile**
  - Betrachtet Bedingungen für einzelne Ein- oder Ausgabeparameter
  - Beachtung von Wechselwirkungen und Abhängigkeiten von Bedingungen sehr aufwändig
- **Empfehlung**
  - Zur Auswahl wirkungsvoller Testdaten: Kombination der ÄK-Bildung mit fehlerorientierten Verfahren, z.B. Grenzwertanalyse

5.1.2 Grenzwertanalyse

- **Idee:** In Verzweigungs- und Schleifenbedingungen gibt es oft Grenzbereiche, für welche die Bedingung gerade noch zutrifft (oder gerade nicht mehr) – Solche Fallunterscheidungen sind fehlerträchtig (off by one).
  - Testdaten, die solche Grenzwerte prüfen, decken Fehlerwirkungen mit höherer Wahrscheinlichkeit auf als Testdaten, die dies nicht tun
- Beste Erfolge bei Kombination mit anderen Verfahren
- Bei Kombination mit der Äquivalenzklassenbildung:
  - Grenzen der ÄK (grösste und kleinste Werte) testen
  - Jeder «Rand» einer ÄK muss in einer Testdatenkombination vorkommen

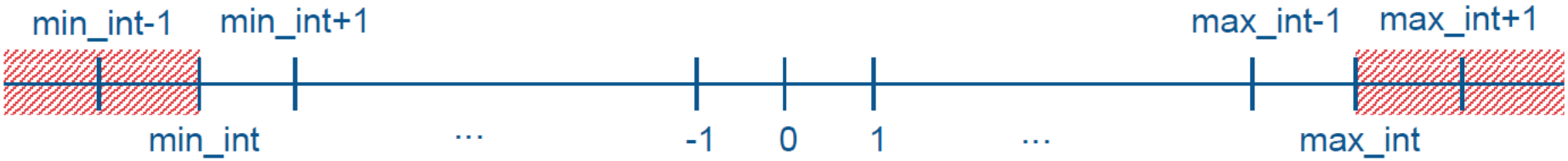
5.1.2 Grenzwertanalyse

- Zusammenfallen der entsprechenden Grenzwerte benachbarter Äquivalenzklassen



ÄK - Äquivalenzklasse
GW – Grenzwert:
i - innerhalb der ÄK
a - außerhalb der ÄK

5.1.2 Beispiel zur Grenzwertanalyse



Datentyp	Grenzen	Größer	Kleiner
integer	0 min_int max_int	1 min_int + 1 max_int + 1	-1 min_int - 1 max_int - 1
char[5]	““ “xxxxx“	“x“ “xxxxxx“	`null` (falls möglich) “xxxx“
double	0.0e0, min_double (-∞) max_double (+∞) NaN (not a number)	+ δ min_double + δ max_double + δ ??	- δ min_double - δ max_double - δ ??

5.1.2 Grenzwertanalyse – Testendekriterium

- In Analogie zum Testendekriterium der Äquivalenzklassenbildung lässt sich auch eine anzustrebende Überdeckung der Grenzwerte (GW) vorab festlegen und nach der Durchführung der Tests berechnen:

GW-Überdeckung = (Anzahl getestete GW / Gesamtzahl GW) * 100 %

5.1.2 Beispiel zur Grenzwertanalyse

- Grenzen des Eingabebereichs
  - Bereich: [-1.0;+1.0]; Testdaten: -1.001; -1.0; +1.0; +1.001 (-0.999; +0.999)
  - Bereich: [-1.0;+1.0]; Testdaten: -1.0; -0.999; +0.999; +1.0 (-1.001; +1.001)
- Grenzen der erlaubten Anzahl von Eingabewerten
  - Eingabedatei mit 1 bis 365 Sätzen; Testfälle 0, 1, 365, 366 (2, 364) Sätze
- Grenzen des Ausgabebereichs
  - Programm errechnet Beitrag, der zwischen 0,00 CHF und 600 CHF liegt;
  - Testfälle: Für 0; 600 CHF und möglichst auch für Beiträge < 0; (knapp >0); und für > 600; (knapp < 600)
- Grenzen der erlaubten Anzahl von Ausgabewerten
  - Ausgabe von 1 bis 4 Daten; Testfälle: Für 0, 1, 4 und 5 (2, 3) Ausgabewerte

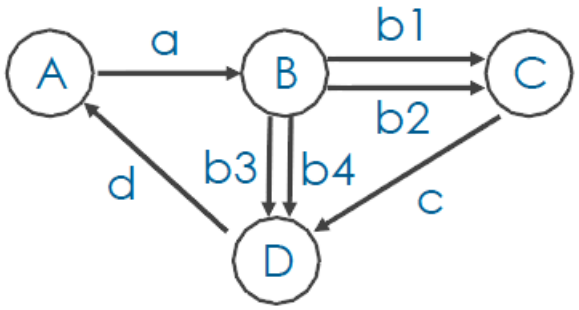
5.1.2 Vor- und Nachteile der Grenzwertanalyse

- **Vorteile**
  - An den Grenzen von Äquivalenzklassen sind häufiger Fehler zu finden als innerhalb dieser Klassen.
  - «Die Grenzwertanalyse ist bei richtiger Anwendung eine der nützlichsten Methoden für den Testfallentwurf».
  - Effiziente Kombination mit anderen Verfahren, die Freiheitsgrade in der Wahl der Testdaten lassen.

- **Nachteile**
  - Rezepte für die Auswahl von Testdaten schwierig anzugeben.
  - Bestimmung aller relevanten Grenzwerte schwierig.
  - Kreativität zur Findung erfolgreicher Testdaten gefordert.
  - Oft nicht effizient genug angewendet, da sie zu einfach erscheint.

5.1.3 Zustandsbasierter Test

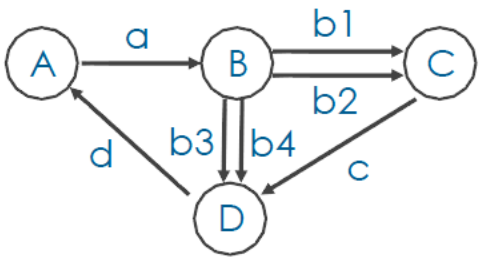
- Ausgangsbasis bildet die Spezifikation des Programms als Zustandsgraph (endlicher Automat, UML-Zustandsdiagramm)
- Zustände und Zustandsübergänge sind so beschrieben
- Beispiel:



- Ein Testfall wird aus
  - dem Ausgangszustand,
  - dem Ereignis (→ Eingabedaten)
  - und dem Soll-Folge-Zustand (→Soll-Resultat)

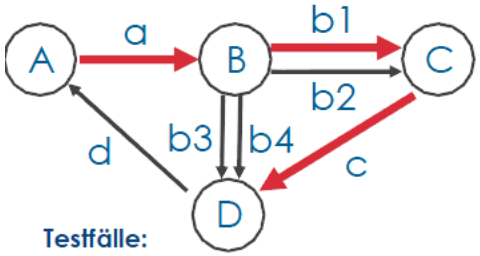
- Testumfang für einen betrachteten Zustand
  - Alle Ereignisse, die zu einem Zustandswechsel führen
  - Alle Ereignisse, die auftreten können, aber ignoriert werden
  - Alle Ereignisse, die auftreten können und eine Fehlerbehandlung erfordern

5.1.3 Zustandsbasierter Test



Zustands-überdeckung

Jeder Zustand muss mindestens einmal erreicht werden

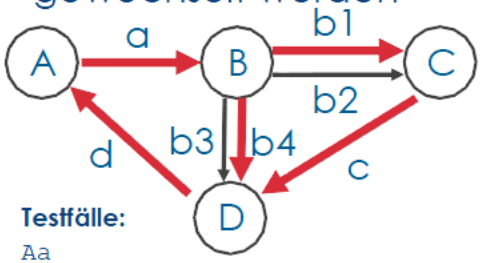


Testfälle:

```
Aa
Bb1
Cc
```

Zustandspaar-überdeckung

Von jedem Zustand muss in jeden möglichen Folgezustand gewechselt werden

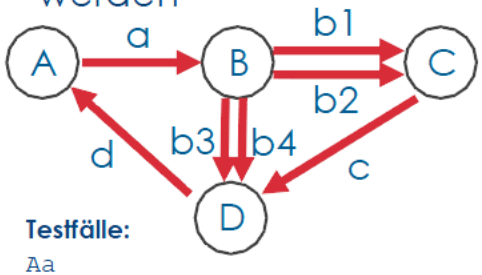


Testfälle:

```
Aa
Bb1
Bb4
Cc
Dd
```

Transitions-überdeckung

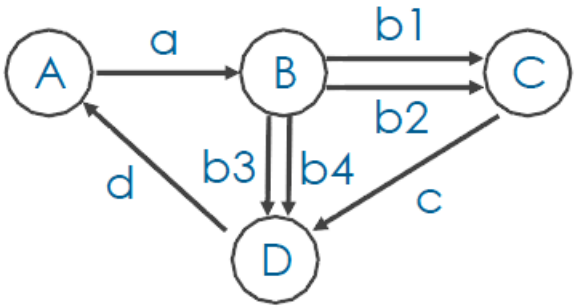
Alle Zustandsübergänge müssen mindestens einmal wirksam werden



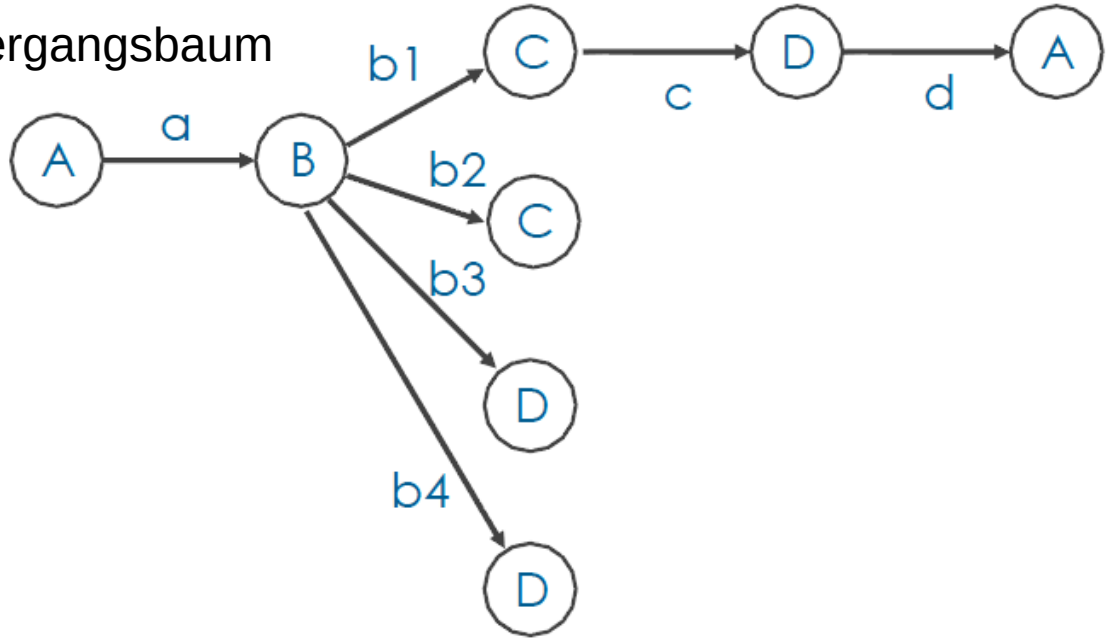
Testfälle:

```
Aa
Bb1     Bb4
Bb2     Cc
Bb3     Dd
```

5.1.3 Roundtrip-Folgen



- Roundtrip-Folgen
  - Beginnend im Startzustand
  - Ended in einem Endezustand oder in einem Zustand, der bereits in dieser oder einer anderen Roundtrip-Folge enthalten war
  - (Zustands-)Übergangsbaum



Testfälle:

```
Aa Bb1 Cc Dd
Aa Bb2
Aa Bb3
Aa Bb4
```

5.1.3 Zustandsbasierter Test - Beispiel

Klasse `Stapel`

Zustandserhaltende Operationen

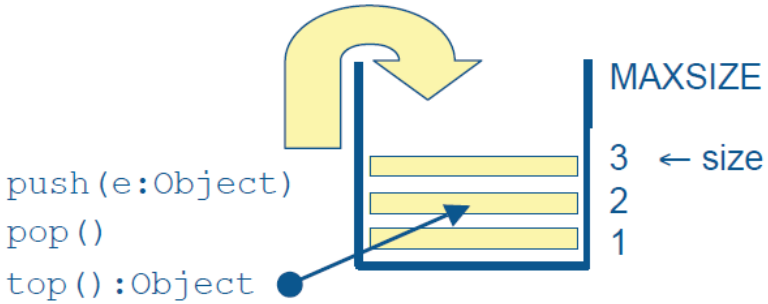
```
size():integer; // Anzahl gestapelter Elemente
MAX():integer;  // Maximale Anzahl
top():Object;   // Zeiger auf oberstes Element
```

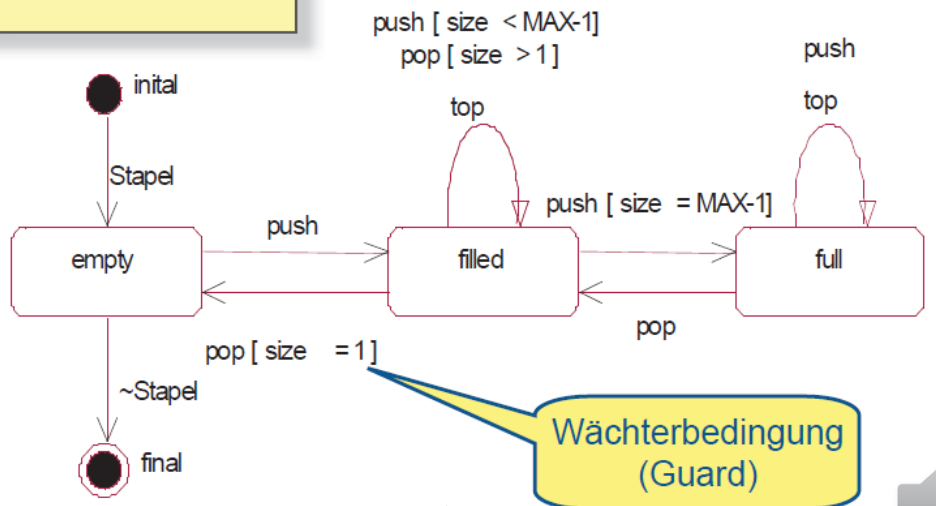
Zustandsverändernde Operationen

```
Stapel(Max:integer);   // Konstruktor
~Stapel();             // Destruktor
push(element:Object);  // Stapelt Element
pop(); // Entfernt oberstes Element
```

Drei Zustände:

```
empty:  size() = 0;
filled: 0 < size() < MAX();
full:   size() = MAX();
```





5.1.3 Zustandsbasierter Test - Arbeitsschritte

1. Analyse des Zustandsdiagramm
2. Prüfung auf Vollständigkeit
3. Ableiten des Übergangsbaumes für den Zustands-Konformanztest
4. Erweitern des Übergangsbaumes für den Zustands-Robustheitstest
5. Generieren der Ereignissequenzen und Ergänzen der Parameter bei Aktionen bzw. Methodenaufrufen
6. Ausführen der Tests und Überdeckungsmessung

5.1.3. Zustandsbasierter Test - Beispiel

Drei Zustände:

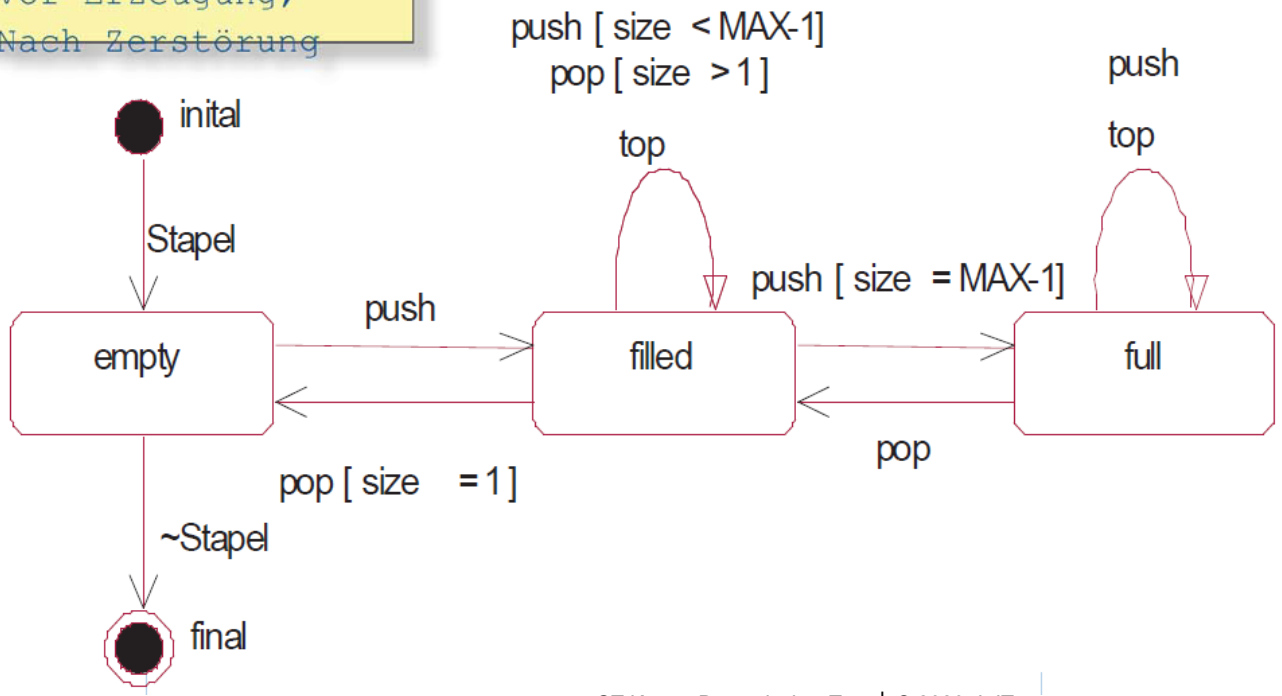
```
empty:  size() = 0;
filled: 0 < size() < MAX();
full:   size() = MAX();
```

Zwei „Pseudo-Zustände“:

```
initial: Vor Erzeugung;
final:   Nach Zerstörung
```

Acht Zustandsübergänge:

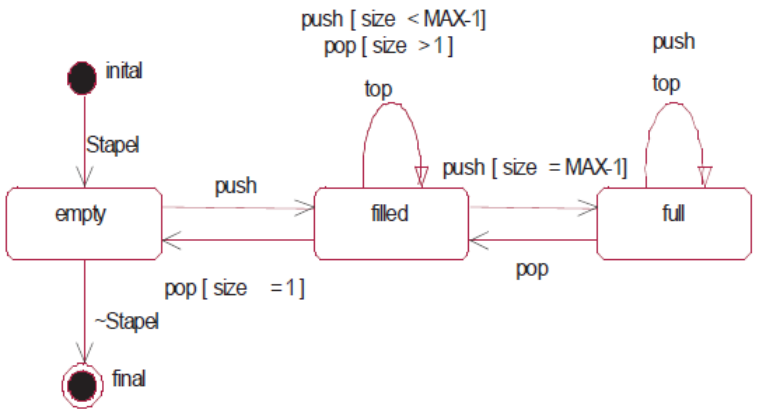
```
initial → empty; empty → final
empty → filled; filled → empty (Zyklus!)
filled → full; full → filled   (Zyklus!)
filled → filled; full → full   (Zyklen!)
```



5.1.3 Zustandsbasierter Test - Beispiel

2. Prüfung auf Vollständigkeit

- Zustandsdiagramm hinsichtlich der «Vollständigkeit» untersuchen

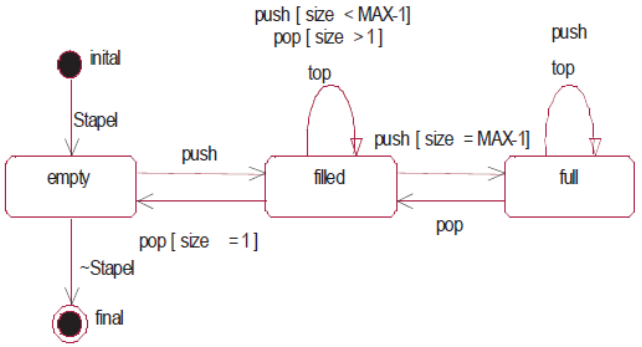


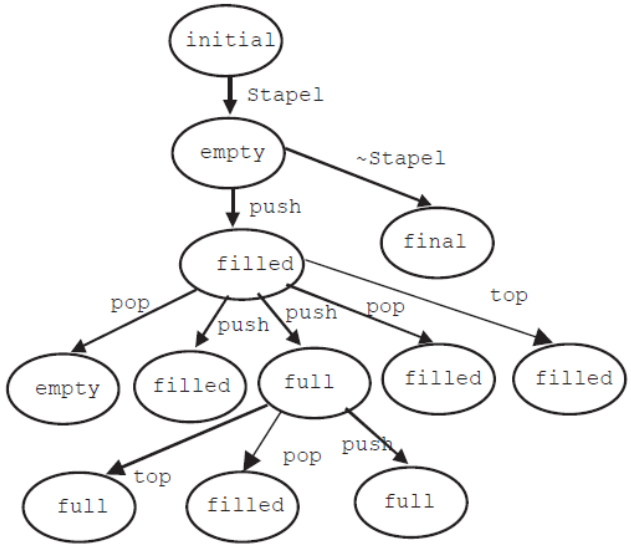
Zustand / Ereignis	initial	empty	filled	full
Stapel()	empty	N/A	N/A	N/A
~Stapel()	N/A	final	?	?
push()	N/A	filled	filled, full	full
pop()	N/A	?	empty, filled	filled
top()	N/A	?	filled	full

5.1.3 Zustandsbezogener Test - Beispiel

3. Übergangsbaum

1. Der Anfangszustand wird die Wurzel des Baumes.
2. Für jeden möglichen Übergang vom Anfangszustand zu einem Folgezustand im Zustandsdiagramm erhält der Übergangsbaum von der Wurzel aus einen Zweig zu einem Knoten, der den Nachfolgezustand repräsentiert. Am Zweig wird das Ereignis (Operation) und ggf. die Wächterbedingung notiert.
3. Der letzte Schritt wird für jedes Blatt des Übergangsbaums so lange wiederholt, bis eine der beiden Endbedingungen eintritt:
   - Der dem Blatt entsprechende Zustand ist auf einer «höheren Ebene» bereits einmal im Baum enthalten.
   - Der dem Blatt entsprechende Zustand ist ein Endzustand und hat somit keine weiteren Übergänge, die zu berücksichtigen wären.



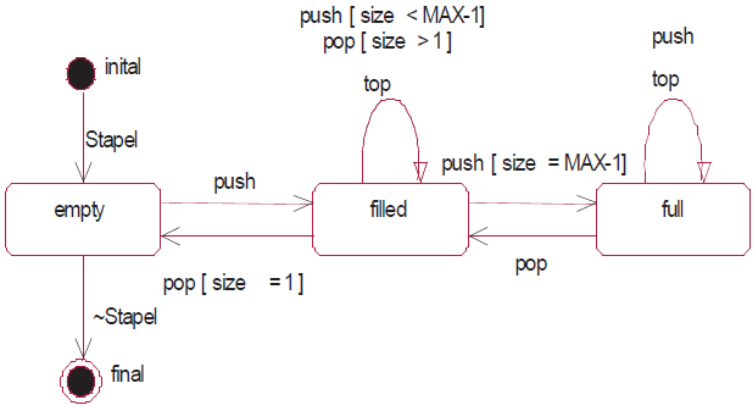


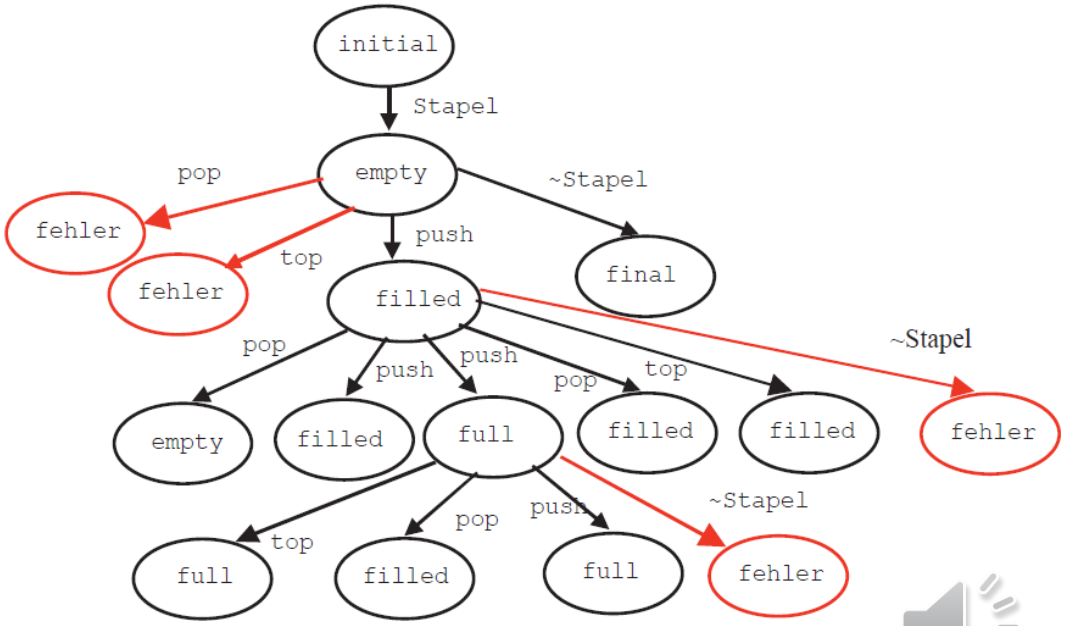
(Wächterbedingungen hier nicht dargestellt)

5.1.3 Zustandsbasierter Test - Beispiel

4. Erweitern des Übergangsbaumes: Zustands-Robustheitstest

- Robustheit unter spezifikationsverletzenden Benutzungen prüfen
- Für Botschaften, für die aus dem betrachteten Knoten kein Übergang spezifiziert ist, den Übergangsbaum um einen neuen «Fehler»-Zustand erweitern

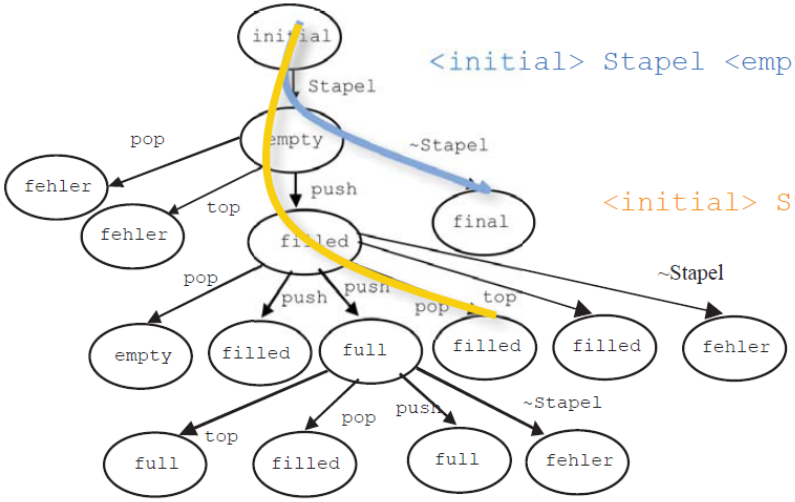




5.1.3 Zustandsbasierter Test - Beispiel

5. Generieren der Testfälle

- Pfade von der Wurzel zu Blättern im erweiterten Übergangsbaum als Funktions-Sequenzen auffassen
- Stimulierung des Testobjekts mit den entsprechenden Funktionsaufrufen deckt alle Zustände und Zustandsübergänge im Zustandsdiagramm ab
- Ergänzen der Funktions-Parameter!



<initial> Stapel <empty> ~Stapel <final>

<initial> Stapel <empty> push <filled> pop <filled>

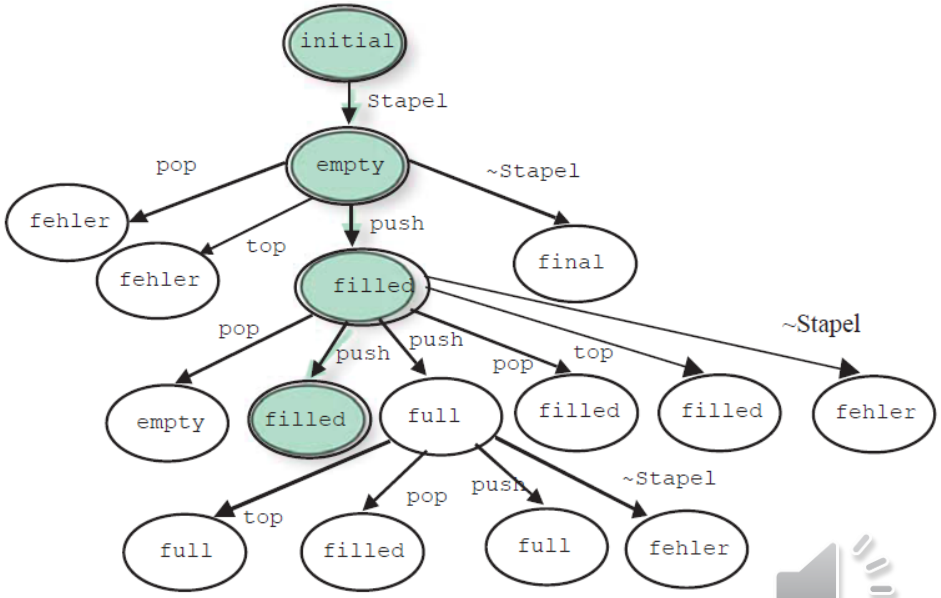
5.1.3 Zustandsbasierter Testfälle

- Ein vollständiger zustandsbezogener Testfall umfasst folgende Informationen:
  - Anfangszustand des Testobjektes (Komponente oder System)
  - Eingaben für das Testobjekt
  - Erwartete Ausgaben bzw. das erwartete Verhalten
  - Erwarteter Endzustand
- Ferner sind für jeden im Testfall erwarteten Zustandsübergang folgende Aspekte festzulegen:
  - Zustand vor dem Übergang
  - Auslösendes Ereignis, das den Übergang bewirkt
  - Erwartete Reaktion, ausgelöst durch den Übergang
  - Nächster erwarteter Zustand

5.1.3 Ausführen der Tests

- Testfälle bzw. Ereignisfolgen in ein Testskript verkapseln.
- Unter Benutzung eines Testtreibers ausführen.
- Zustände über zustandserhaltende Operationen ermitteln und protokollieren.

```
K3' = //<initial>
        Stapel OUT = new Stapel(5)
      //<empty>
        OUT.push(new Object())
      //<filled>
        OUT.push(new Object())
      //<filled>
        if (OUT.size() != 2) then
         throw WrongStateException;
```



5.1.3 Testendekriterien

- Minimalkriterium: Jeder Zustand wurde mindestens einmal eingenommen

Z-Überdeckung = (Anzahl getestete Z / Gesamtzahl Z) * 100 %

- Weitere Kriterien:
  - Jeder Zustandsübergang wurde mindestens einmal ausgeführt

ZÜ-Überdeckung = (Anzahl getestete ZÜ / Gesamtzahl ZÜ) * 100 %

  - Alle spezifikationsverletzenden Zustandsübergänge wurden angeregt
  - Jede Aktion (Funktion) wurde mindestens einmal ausgeführt
- Bei hoch kritischen Anwendungen
  - Alle Zustandsübergänge und alle «Zyklen» im Zustandsdiagramm
  - Alle Zustandsübergänge in jeder beliebigen Reihenfolge mit allen möglichen Zuständen, auch mehrfach hintereinander

5.1.4 Ursache-Wirkungs-Graph (Exkurs)

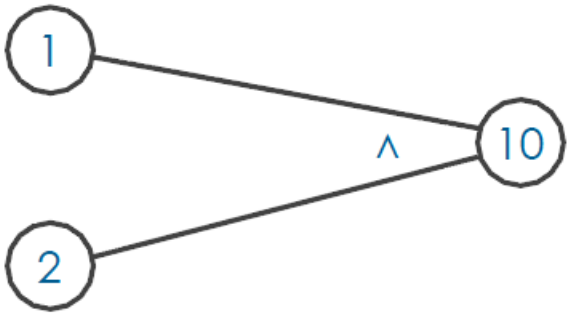
- Grafische Darstellung von Wirkungen zwischen Eingabedaten und Ausgabedaten bzw. Aktionen

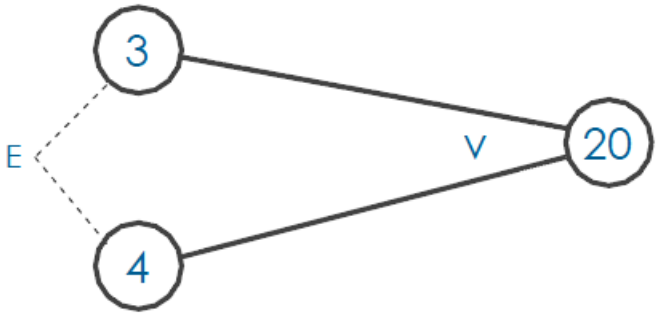
1: Parken im Parkverbot
2: Polizist kommt vorbei

10: Strafzettel erhalten

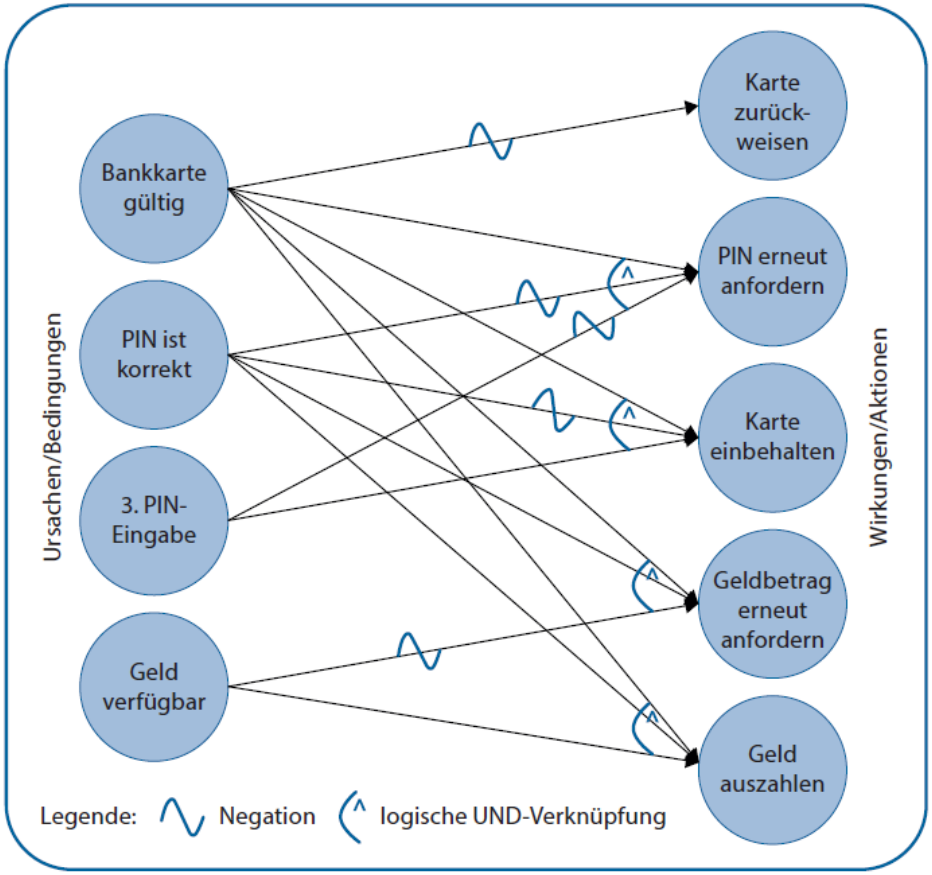
3: Rückwärtsgang eingelegt
4: Langsam vorwärts fahren

20: Abstandswarner aktivieren





5.1.4 Ursache-Wirkungs-Graph - Beispiel



Der Graph ist dann in eine Entscheidungstabelle umzuformen, aus der dann die Testfälle abzuleiten sind.

5.1.4 Ursache-Wirkungs-Graph - Beispiel

Entscheidungstabelle		TF1	TF2	TF3	TF4	TF5
Bedingungen	Bankkarte gültig?	Nein	Ja	Ja	Ja	Ja
	PIN ist korrekt?	–	Nein	Nein	Ja	Ja
	3. PIN-Eingabe?	–	Nein	Ja	–	–
	Geld verfügbar?	–	–	–	Nein	Ja
Aktionen	Karte zurückweisen	X				
	PIN erneut anfordern		X			
	Karte einbehalten			X		
	Geldbetrag erneut anfordern				X	
	Geld auszahlen					X

Optimierte Entscheidungstabelle abgeleitet aus dem Ursache-Wirkungs-Graphen.

5.1.4 Entscheidungstabellentest

- Entscheidungstabellen beschreiben sehr anschaulich (Geschäfts-) Regeln der Art «wenn ... dann ... Sonst».
- Typische Anwendungssituation: Abhängig von mehreren logischen Eingangswerten sollen verschiedene Aktionen ausgeführt werden.
- Entscheidungstabellen unterstützen durch ihren Aufbau die Vollständigkeit des Tests.
- Jeder Tabellenspalte (Regel) entspricht ein Testfall.
- Regelüberdeckung: Je Regel wird mindestens ein Testfall gewählt.
- Das Soll-Resultat ergibt sich durch die entsprechend ausgeführten Aktionen.

5.1.4 Entscheidungstabellentest

	Regeln				
Bedingungen	Regel 1	Regel 2	Regel 3	...	*Regel k*
Bedingung 1	T	F	T	Beding-ungen ver-wenden Eingabe-daten	F
Bedingung 2	-	T	F		F
Bedingung i	F	-	-		-
Aktionen					
Aktion 1		x	x	Aktionen erzeugen Ausgabe-daten	x
Aktion 2	x		x		
Aktion j	x				

Don't care

```
IF (Bedingung 1) AND (Not Bedingung i)
  Aktion 2
  Aktion j
END-IF
```

5.1.4 Entscheidungstabellentest - Beispiel

- In einem Warenwirtschaftssystem gelten folgende Geschäftsregeln:
  - Die Bestellmenge muss grösser als Null sein.
  - Teil-Lieferungen sind nicht erlaubt.
  - Bei der Annahme einer Bestellung muss die Lagermenge entsprechend reduziert werden.
  - Wird die Mindestmenge eines Lagerartikels unterschritten, muss eine Nachbestellung erfolgen.

Textteil	Regelteil			
Bestellmenge > 0	N	J	J	J
Bestellmenge > Art-Lagermenge	-	J	N	N
Art-Lagermenge - Bestellmenge >= Art-Mindestmenge	-	-	N	J
Melde "Bestellmenge ungültig"	X			
Melde "Menge nicht ausreichend"		X		
Reduziere Lagermenge			X	X
Schreibe Nachbestellung			X	

Bedingungsanzeiger:
N = nicht erfüllt
J = erfüllt
- = ohne Bedeutung
= nicht definiert

Aktionsanzeiger:
X = ausführen
= nicht ausführen
(auch „–“)

5.1.4 Entscheidungstabellentest - Beispiel

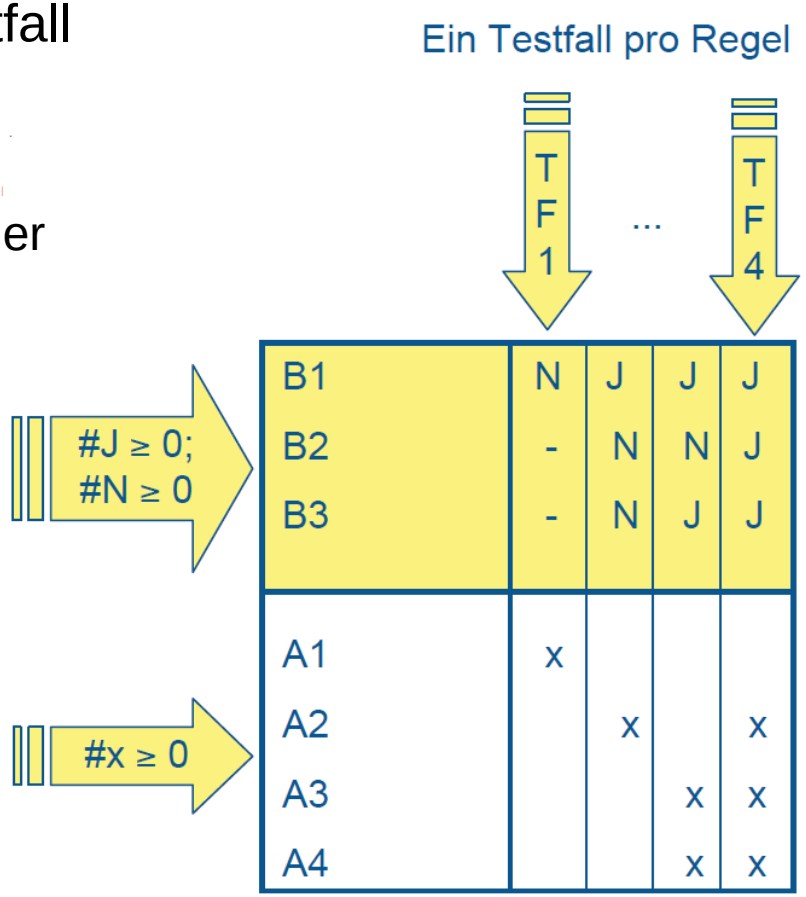
- Analyse von Entscheidungstabellen (ET):
  - ET vollständig, wenn bei n Bedingungen alle 2^n Kombinationen enthalten sind.
    - Spalten im oberen Teil betrachten
  - ET redundanzfrei, wenn speziellere Bedingungen zu anderen Aktionen führen
    - Spalten im oberen und unteren Teil betrachten
  - ET widerspruchsfrei, wenn gleiche Bedingungen zu gleichen Aktionen führen
    - Spalten im oberen und unteren Teil betrachten

Bestellmenge > 0	N	N	N	N	J	J	J	J
Bestellmenge > Art-Lagermenge	N	N	J	J	N	N	J	J
Art-Lagermenge - Bestellmenge >= Art-Mindestmenge	N	J	N	J	N	J	N	J
Melde "Bestellmenge ungültig"	X	X	X	X				
Melde "Menge nicht ausreichend"							X	X
Reduziere Lagermenge					X	X		
Schreibe Nachbestellung					X			

Bestellmenge > 0	N	J	J	J
Bestellmenge > ArtLagermenge	-	J	N	N
Art-Lagermenge - Bestellmenge >= Art-Mindestmenge	-	-	N	J
Melde "Bestellmenge ungültig"	X			
Melde "Menge nicht ausreichend"		X		
Reduziere Lagermenge			X	X
Schreibe Nachbestellung			X	

5.1.4 Entscheidungstabellentest - Testfälle

- Jede Spalte (Regel) entspricht einem Testfall
  - Voraussetzungen pro Tabelle gleich
  - Bedingungen beziehen sich auf Eingaben
  - Aktionen spiegeln erwartetes Ergebnis wider
- Überdeckungskriterien z.B.
  - Alle Bedingungen mindestens einmal N und J
  - Alle Aktionen mindestens einmal x
  - Alle Spalten (alle Bedingungskombinationen)
- Konkrete Testdaten aus Wertebereichen ableiten
  - Äquivalenzklassenbildung
  - Grenzwertanalyse



5.1.5 Kombinatorisches Testen (Exkurs)

- Die Basis bei den bisherigen Testverfahren zur Erstellung der Testfälle waren die Spezifikation und darauf aufbauende Überlegungen.
- Wenn keine Abhängigkeiten zwischen einzelnen Eingaben existieren, diese also frei kombinierbar sind, kann die Mathematik zur Hilfe genommen werden, um Kombinationen auszuwählen und damit systematisch Testfälle zu erstellen.
- Es sollen nicht alle möglichen Kombinationen über alle Parameter beim Testen berücksichtigt werden, sondern nur Kombinationen zwischen zwei, drei oder mehreren Parametern, diese dann aber vollständig.

5.1.6 Anwendungsfallbasierter Test

- Anwendungsfälle (Use Case) oder Geschäftsszenarien beschreiben die Interaktionen zwischen den Akteuren und System, die ein Ergebnis oder einen Wert für den Anwender des Systems zur Folge haben.
  - Jeder Anwendungsfall hat Vorbedingungen, die erfüllt sein müssen, damit der Anwendungsfall erfolgreich durchgeführt werden kann und endet mit Nachbedingungen, den beobachtbaren Ergebnissen und dem Endzustand des Systems, wenn er vollständig abgewickelt wurde.
  - Ein Anwendungsfall hat üblicherweise ein Hauptszenario (das wahrscheinlichste Szenario) und manchmal mehrere alternative Abläufe und Ausnahmefälle (Varianten).
  - Anwendungsfälle beschreiben die «Prozessabläufe» durch das System auf Grundlage seiner voraussichtlich tatsächlichen Verwendung.

5.1.6 Anwendungsfall - Beispiel

Anwendungsfall Auszahlung **In Modell** Bankautomat
Akteure Bankkunde, Zentralrechner
Vorbedingung Kartenleser betriebsbereit **UND** Bedienpult gesperrt
Normaler Ablauf

1. Der Bankkunde meldet sich am Bankautomaten an (include: Anmelden)
2. Der Bankkunde wählt als Transaktion „Auszahlung“
3. Der Bankkunde gibt den abzuhebenden Betrag ein
4. Der Bankautomat prüft den Betrag und meldet ihn an den Zentralrechner
5. Der Bankautomat aktualisiert die Karte und gibt sie aus
6. Der Bankkunde entnimmt die Karte
7. Der Bankautomat gibt das Geld aus
8. Der Bankkunde entnimmt das Geld

Alternativer Ablauf

4.a Der Betrag ist zu hoch und muss neu eingegeben werden
4.b Der Betrag ist nicht in Scheinen auszahlbar und muss neu eingegeben werden

Nachbedingung Saldo des Kontos um Auszahlungsbetrag reduziert
UND Geldvorrat um den Auszahlungsbetrag reduziert
UND Kartenleser betriebsbereit **UND** Bedienpult gesperrt
Ausnahmeablauf

6.a Die Karte wird nicht innerhalb von 60 Sekunden entnommen

Nachbedingung Karte eingezogen **UND** Kartenleser betriebsbereit **UND** Bedienpult gesperrt
Ausnahmeablauf

1.-4. Der Bankkunde bricht den Vorgang ab
5.a Der Bankautomat aktualisiert die Karte und gibt sie aus
6.a Der Bankkunde entnimmt die Karte

Nachbedingung Kartenleser betriebsbereit **UND** Bedienpult gesperrt
END Auszahlung.

5.1.6 Anwendungsfallbasierter Test

- Testfälle so auswählen, dass die geforderte Überdeckung des Anwendungsfalls erzielt wird
  - Normaler Ablauf
  - Alternative Abläufe
  - Ausnahmeabläufe
  - Mögliche Wiederholungen innerhalb der Szenarien
- Ein mögliches Testendekriterium ist, dass jeder Use Case bzw. jede mögliche Abfolge von Use Cases im Diagramm mindestens einmal mit einem Testfall zu überprüfen ist
- Der anwendungsfallbasierte Test ist sehr gut geeignet, um typische Benutzer-System-Interaktionen zu prüfen.
- Sein Einsatz empfiehlt sich daher für den Systemtest und Akzeptanztest.

5.1.6 Weitere Blackbox-Verfahren (Exkurs)

- **Syntaxtest**
  - Verfahren zur Ermittlung der Testfälle, das bei Vorliegen einer formalen Spezifikation der Syntax der Eingaben angewendet werden kann.
  - Die Regeln der syntaktischen Beschreibung werden genutzt, um Testfälle zu spezifizieren, welche sowohl die Einhaltung als auch die Verletzung der syntaktischen Regeln für die Eingaben berücksichtigen.
- **Zufallstest**
  - Wählt aus der Menge der möglichen Werte eines Eingabedatums zufällig Repräsentanten für die Testfälle aus.
  - Ist eine statistische Verteilung der Werte zu vermuten oder gegeben (z.B. eine Normal-Verteilung), so soll diese auch für die Wahl der Repräsentanten herangezogen werden, um möglichst realitätsnahe Testfälle zu erhalten und um Aussagen zur Zuverlässigkeit des Systems zu erhalten.

5.1.6 Weitere Blackbox-Verfahren

- **Smoke-Test**
  - «Ausprobieren» des Testobjektes, das vorwiegend die prinzipielle Lauf- und Testfähigkeit des Testobjekts prüft.
  - Es wird kein Testorakel verwendet und folglich werden auch keine Soll-Ergebnisse erstellt.
  - Beim Smoke-Test wird versucht, einen offensichtlichen Absturz des Testobjektes zu erzeugen.
  - Oft als «Installationstest» und für neue Software-Updates verwendet.

5.1.7 Allgemeine Bewertung der Blackbox-Testverfahren

- Grundlage aller Blackbox-Verfahren sind die Anforderungen sowie die Spezifikation des Systems bzw. der einzelnen Komponenten und ihres Zusammenwirkens.
- Sind fehlerhafte Festlegungen in den Anforderungen getroffen oder war eine fehlerhafte Spezifikation Grundlage für die Implementierung, so können diese Fehler nicht erkannt werden.
- Mit den Blackbox-Verfahren kann auch nicht festgestellt werden, ob das Testobjekt noch weitere Funktionalität bereitstellt, die über die Spezifikation hinausgeht (dies ist oft der Grund für auftretende Sicherheitsprobleme).
- Im Mittelpunkt aller Blackbox-Verfahren steht die Prüfung der Funktionalität des Testobjekts.
- Es ist sicherlich unumstritten, dass das korrekte Funktionieren eines Softwaresystems höchste Priorität hat und somit auch die Blackbox-Testverfahren stets einzusetzen sind.