LEHRSTUHL 14
SOFTWARE ENGINEERING

Vorlesung (WS 2013/14)

Softwarekonstruktion

Prof. Dr. Jan Jürjens

TU Dortmund, Fakultät Informatik, Lehrstuhl XIV

Teil 3.4: Black-Box-Test

v. 25.01.2014

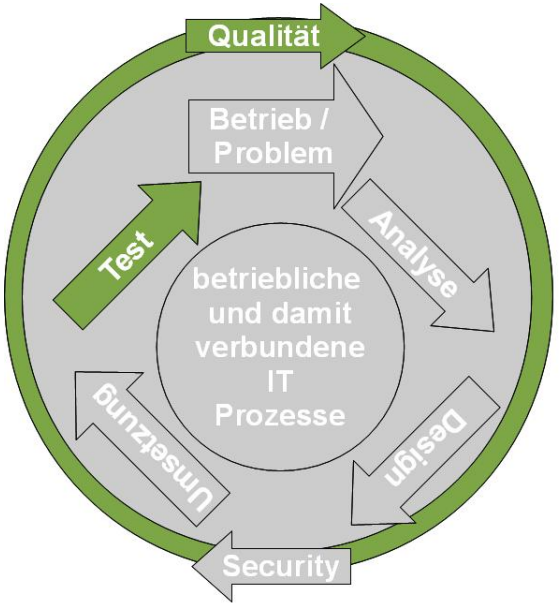
[Basierend auf dem Foliensatz „Basiswissen Softwaretest - Certified Tester“ des „German Testing Board“ (nach Certified Tester Foundation Level Syllabus, deutschsprachige Ausgabe, Version 2011)]

Literatur:

- Andreas Spillner, Tilo Linz: Basiswissen Softwaretest, Aus- und Weiterbildung zum Certified Tester Foundation Level nach ISTQB-Standard.
  - **Kapitel 5.**
- Eike Riedemann: Testmethoden für sequentielle und nebenläufige Software-Systeme.

Einordnung
Black-Box-Test

- Modellgetriebene SW-Entwicklung
- Qualitätsmanagement
- Testen
  - Grundlagen Softwareverifikation
  - White-Box-Test
  - Softwaremetriken
  - Algebraische Spezifikationen
  - Black-Box-Test
  - Testen im Softwarelebenszyklus



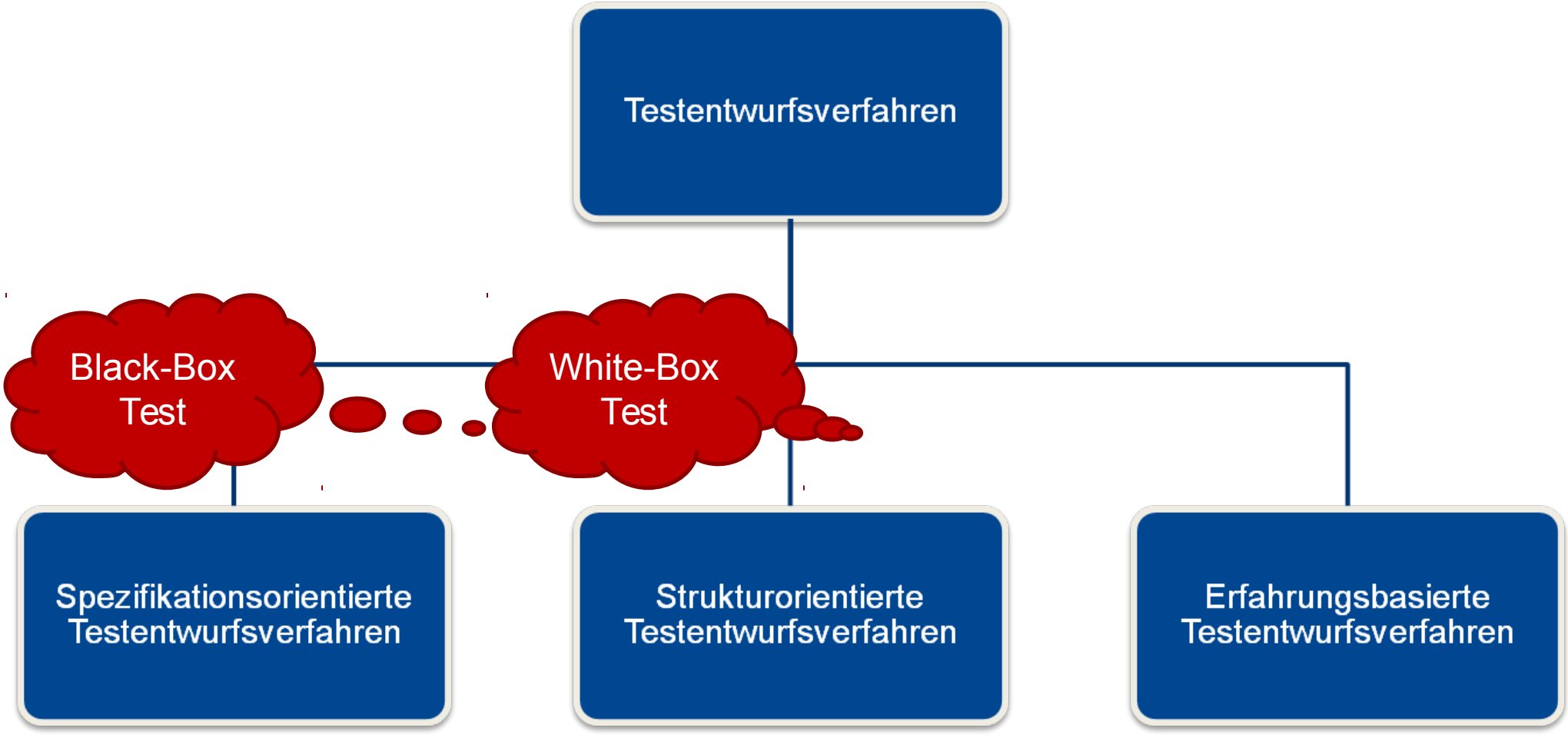
LEHRSTUHL 14
SOFTWARE ENGINEERING

3.4 Black-Box-Test

3.4 Black-Box-Test

- Dynamischer Test – Grundlagen
- Idee der Black-Box-Testentwurfsverfahren
- Äquivalenzklassenbildung
- Grenzwertanalyse
- Zustandsbasierter Test
- Entscheidungstabellentest
- Test mittels Ursache-Wirkungs-Graphen

Testentwurfsverfahren



Black-Box Test vs. White-Box Test

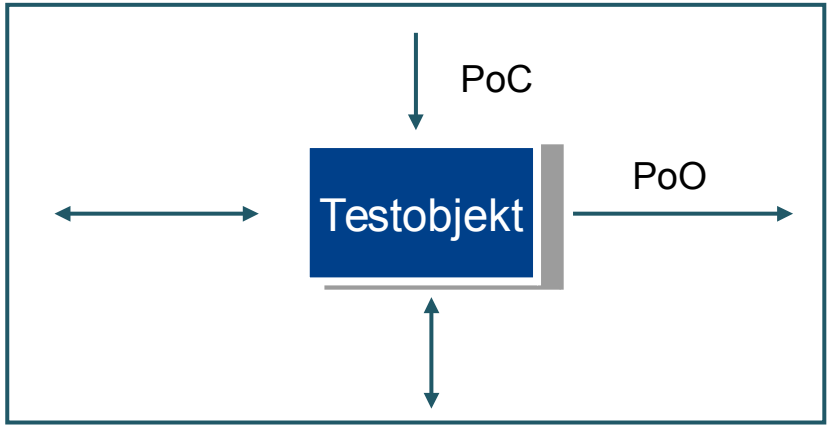
Black-Box Test

Eingabewerte

Ohne Kenntnis der Programmlogik abgeleitet

Testobjekt

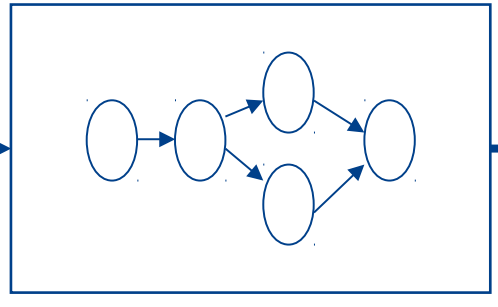
Istergebnis



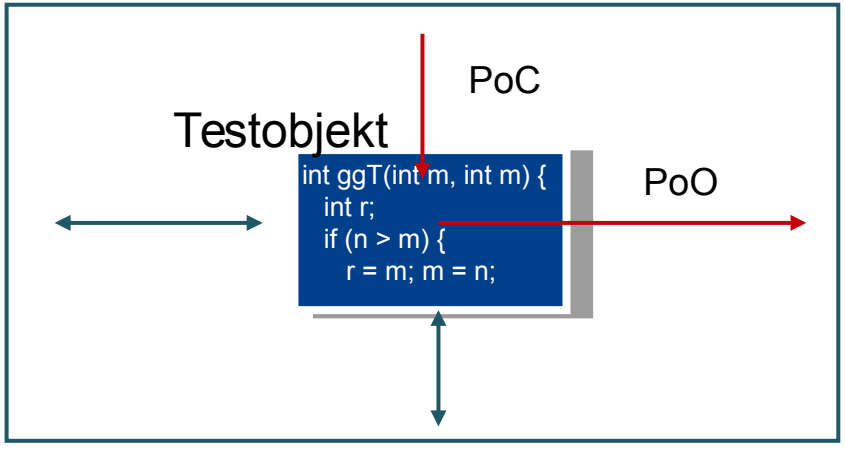
White-Box Test

Eingabewerte

Mit Kenntnis der Programmlogik abgeleitet



Istergebnis



Spezifikationsorientierter / Black-Box-Test: Begriffe

Funktionaler Test

- Dynamischer Test:
  - **Herleitung der Testfälle** unter Verwendung funktionaler Spezifikation des Testobjekts.
  - **Bewertung** von Vollständigkeit der Prüfung anhand funktionale Spezifikation.

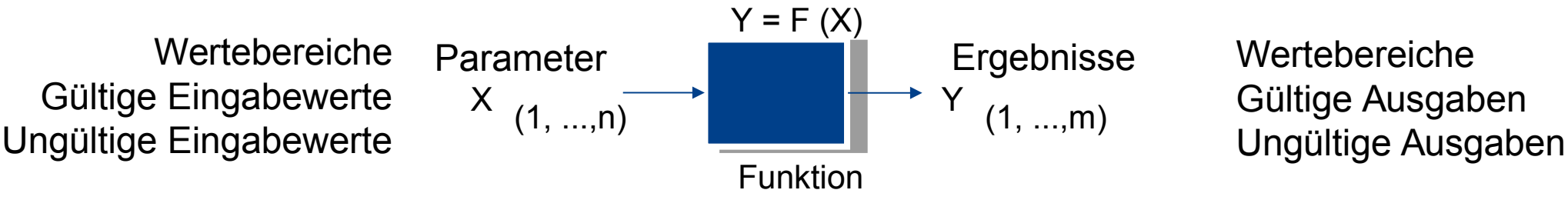
Funktionalität

- Beim Einsatz unter spezifizierten Bedingungen **Funktionen liefern**.
  → Erfüllung festgelegter und vorausgesetzter Erfordernisse [ISO 9126].

  Untermerkmale der Funktionalität nach ISO 9126:
  Angemessenheit, Richtigkeit, Interoperabilität, Sicherheit, Konformität.

ISO/IEC 9126:200x: Bewerten von Softwareprodukten, Qualitätsmerkmale und Leitfaden zu ihrer Verwendung.

Spezifikationsorientierte Testfall- und Testdatenermittlung



Äquivalenzklassenbildung

- Repräsentative Eingaben
- Gültige Dateneingaben
- Ungültige Dateneingaben
- Erreichen der gültigen Ausgaben

Grenzwertanalyse

- Wertebereiche
- Wertebereichsgrenzen

Zustandsbasierter Test

- Komplexe (innere) Zustände und Zustandsübergänge

Entscheidungstabellentest

- Bedingungen und Aktionen

Anwendungsfallbasierter Test

- Szenarien der Systemnutzung

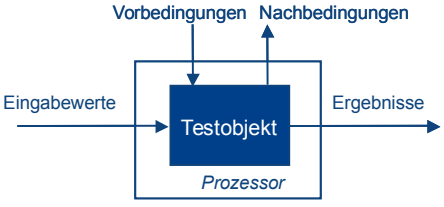
3.4 Black-Box-Test

3.4 Black-Box-Test

- Dynamischer Test – Grundlagen
- Idee der Black-Box-Testentwurfsverfahren
- **Äquivalenzklassenbildung**
- Grenzwertanalyse
- Zustandsbasierter Test
- Entscheidungstabellentest
- Test mittels Ursache-Wirkungs-Graphen

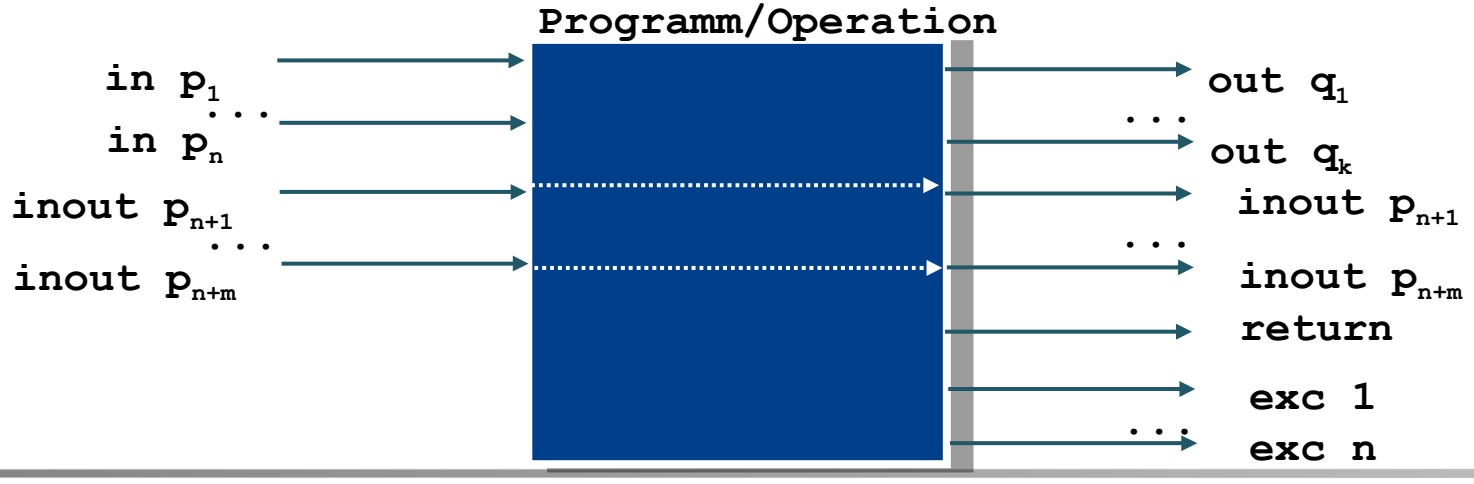
Programme und Operationen

Vereinfachende **Annahme:** Programme / Operationen berechnen Ausgaben aus Eingaben (zustandslos).

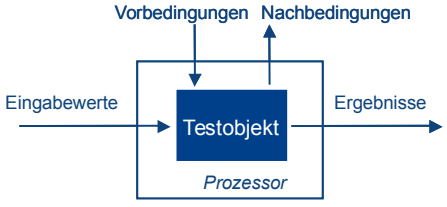


Signatur einer Operation:

- Operationsname, Parametertypen, Rückgabetyp
- Parameter als **`in`**, **`inout`**, **`out`** gekennzeichnet
- Ggf. ein **`out`**-Parameter als Rückgabewert (Funktion)
- Ggf. return oder Exceptions



Operationen in Java (»Methoden«)



- Mehrere **Eingabeparameter:**
  - Atomare Typen: nur call-by-value (in)
  - Klassen bzw. Objekte: call-by-reference (inout)
- Ein **Rückgabewert:**
  - Atomarer Typ (out)
  - Klasse bzw. Objekt (out, inout)
- Ggfs. mehrere **Exceptions.**
- **Typen** spezifizieren Definitionsbereiche.

Beispiel: Funktion ggT

Bestimmung des größten **gemeinsamen Teilers (ggT)** zweier ganzer Zahlen m und n:

ggT(4,8)=4; ggT(5,8)=1; ggT(15,35)=5; ggT(0,0)=0 [per Konvention]

Logische Spezifikation:

$ggT: \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{N}$

$ggT(0,0) = 0 \wedge$

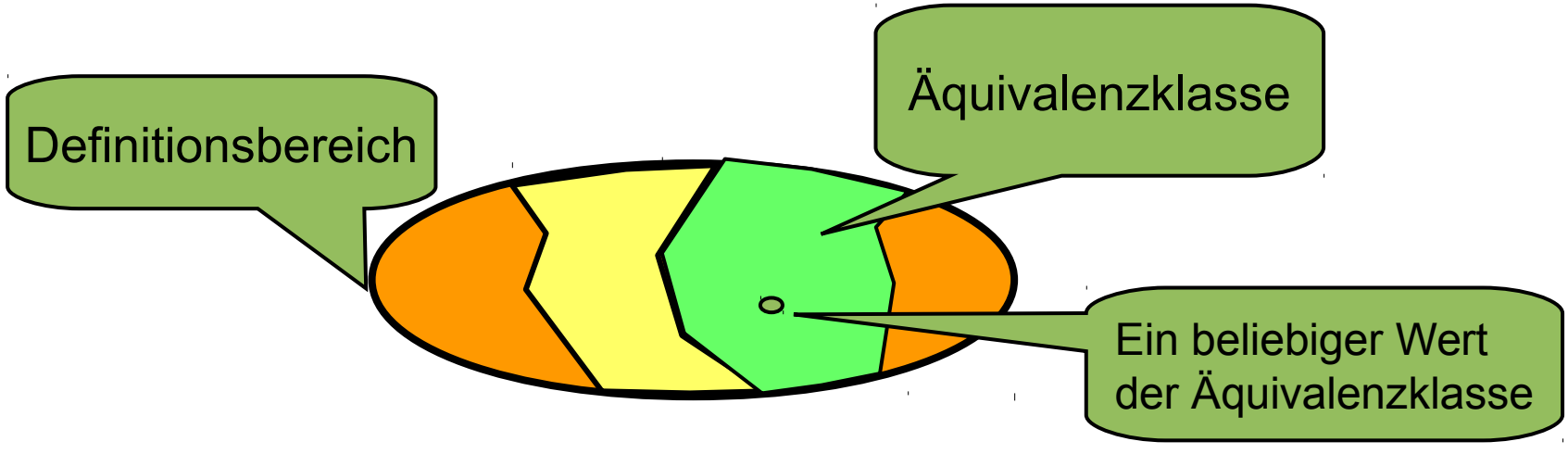
$[m<>0 \vee n<>0 \Rightarrow ggT(m,n)|m \wedge ggT(m,n)|n \wedge \forall o \in \mathbb{N}: o>ggT(m,n) \Rightarrow (\neg(o|m) \vee \neg(o|n))]$

Spezifikation in UML / Java:

```
public int ggt(int m, int n) {
// pre:  m<>0 or n<>0
// post: m@pre.mod(return) = 0 and
//     n@pre.mod(return) = 0 and
//     forall(i : int | i > return implies
//       (m@pre.mod(i) > 0 or n@pre.mod(i) > 0)
… )
```

Grundidee der Äquivalenzbildung

- Zerlegung der Definitionsbereiche der Ein- und Ausgaben in **Äquivalenzklassen (ÄK):** Werte einer Klasse = Äquivalentes Verhalten des Prüflings.
- Wahl Testwertes pro ÄK: **Sinnvolle Stichprobe**.
- Wenn Wert der ÄK **Fehler**
  - **aufdeckt.** → Alle Werte der ÄK sollen diesen Fehler aufdecken.
  - **nicht aufdeckt.** → Kein Wert der ÄK soll einen Fehler aufdecken.



Äquivalenzklassen für ggT: Erster Schritt

Definitionsbereiche der Ein- und Ausgaben:

- Eingabeparameter: `int`
- Rückgabewert: `int`

Gültige von ungültigen Teilbereichen der Java-Implementierung unterscheiden:

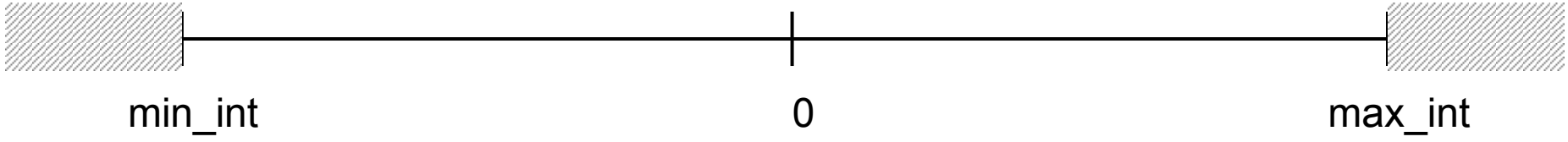
?

Äquivalenzklassen für ggT: Erster Schritt

Definitionsbereiche der Ein- und Ausgaben

- Eingabeparameter: `int`
- Rückgabewert: `int`

Gültige von ungültigen Teilbereichen der Java-Implementierung unterscheiden:



Äquivalenzklassenbildung
Vorgehensweise

Aufstellen der Definitionsbereiche aus Spezifikation.

Äquivalenzklassenbildung für jede Beschränkung:

- Beschränkung spezifiziert **Wertebereich**: Eine gültige und zwei ungültige ÄK.
- Beschränkung spezifiziert **minimale und maximale** Anzahl von Werten: Eine gültige und zwei ungültige ÄK.
- Beschränkung spezifiziert **Menge von Werten**, die unterschiedlich behandelt werden: Für jeden Wert dieser Menge eigene gültige ÄK und zusätzlich eine ungültige ÄK.
- Beschränkung spezifiziert **Situation**, die erfüllt sein muss: Eine gültige und eine ungültige ÄK.
- Werte einer ÄK **nicht gleichwertig** behandelt: Aufspaltung der ÄK in kleinere ÄK.

Äquivalenzklassenbildung
Beispiel Situation

Beschränkung spezifiziert **Situation**, die erfüllt sein muss: Eine gültige und eine ungültige ÄK.

Beispiel

Laut **Spezifikation** erhält jedes Mitglied im Sportverein eine eindeutige Mitgliedsnummer. Diese beginnt mit dem ersten Buchstaben des Familiennamens des Mitglieds.

Gültige Äquivalenzklasse: ?

Ungültige Äquivalenzklasse: ?

Äquivalenzklassenbildung
Beispiel Situation

Beschränkung spezifiziert **Situation**, die erfüllt sein muss: Eine gültige und eine ungültige ÄK.

Beispiel

Laut **Spezifikation** erhält jedes Mitglied im Sportverein eine eindeutige Mitgliedsnummer. Diese beginnt mit dem ersten Buchstaben des Familiennamens des Mitglieds.

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

Diskussionsfrage

Gegeben ist eine Funktion zur Bestimmung der Anzahl der Tage eines Monats mit den Übergaben Monat und Jahr.

- ZahlTageMonat(int Monat, int Jahr)

Wie sehen die **Äquivalenzklassen** dazu aus ?

Diskussionsfrage

Gegeben ist eine Funktion zur Bestimmung der Anzahl der Tage eines Monats mit den Übergaben Monat und Jahr.

- ZahlTageMonat(int Monat, int Jahr)

Wie sehen die **Äquivalenzklassen** dazu aus ?

Klassen für **Monat:**

- Gültig:
  - Monate mit 30 Tagen
  - Monate mit 31 Tagen
  - Februar
- Ungültig:
  - > 12
  - < 1

Klassen für **Jahr:**

- Gültig:
  - Schaltjahre
  - Normaljahre

Testfälle für jeden Parameter tabellarisch notieren

Eindeutige Kennzeichnung jeder Äquivalenzklasse (gÄKn, uÄKn):

	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 m_i **Äquivalenzklassen** (i=1..n) gibt es:

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

Anzahl der Testfälle minimieren: Heuristiken

- **Testfälle** aus Repräsentanten kombinieren und nach »Häufigkeit« sortieren (»**Benutzungsprofile**«).
  - Testfälle in dieser Reihenfolge priorisieren.
  - Mit »benutzungsrelevanten« Testfällen testen.
  - Testfälle, die Grenzwerte oder Grenzwert-Kombinationen enthalten, bevorzugen.
- **Ausführung**: Jeder Repräsentant einer Äquivalenzklasse mit jedem Repräsentanten anderer Äquivalenzklassen in einem Testfall.

  → Paarweise statt vollständiger Kombination.
- **Minimalkriterium:** Min. ein Repräsentant jeder Äquivalenzklasse in min. einem Testfall.
- Repräsentanten **ungültiger Äquivalenzklassen** nicht miteinander kombinieren.

Äquivalenzklassenbildung: Fehlermaskierung

Gleichzeitige Behandlung verschiedener ungültiger ÄK: Bestimmte Fehler evtl. unentdeckt !

Beispiel:

Eingabebereich

```
1 <= wert <= 99; farbe IN (rot, gruen, gelb)
```

Äquivalenzklassen

```
wert_gÄK1:  ?
wert_uÄK1:  ?
wert_uÄK2:  ?
farbe_gÄK1: ?
farbe_uÄK1: ?
```

Testdaten:

`wert_uÄK1` und `farbe_uÄK1:` z.B. `wert=?, farbe=?`

→ Welche Fehler werden evt. übersehen ?

Gleichzeitige Behandlung verschiedener ungültiger ÄK: Bestimmte Fehler evtl. unentdeckt !

Beispiel:

Eingabebereich

```
1 <= wert <= 99; farbe IN (rot, gruen, gelb)
```

Äquivalenzklassen

```
wert_gÄK1:  1 <= wert <= 99
wert_uÄK1:  wert < 1
wert_uÄK2:  wert > 99
farbe_gÄK1: farbe IN (rot, gruen, gelb)
farbe_uÄK1: NOT farbe IN (rot, gruen, gelb)
```

Testdaten

`wert_uÄK1` und `farbe_uÄK1`: z.B. `wert=0, farbe=schwarz`

→ Fehlerhafte Behandlung von `farbe=schwarz` bei `wert_gÄK1` ggf. unentdeckt (und umgekehrt).

Äquivalenzklassenbildung
Ausgangskriterium (Testendekriterium)

- **Spezifisches Ausgangskriterium** festlegen:
  Nach Äquivalenzklassenbildung anhand durchgeführte Tests der Repräsentanten der jeweiligen Äquivalenzklassen im Verhältnis zur Gesamtzahl aller definierten Äquivalenzklassen:

ÄK-Überdeckungsgrad = (Anzahl getestete ÄK / Gesamtzahl ÄK)

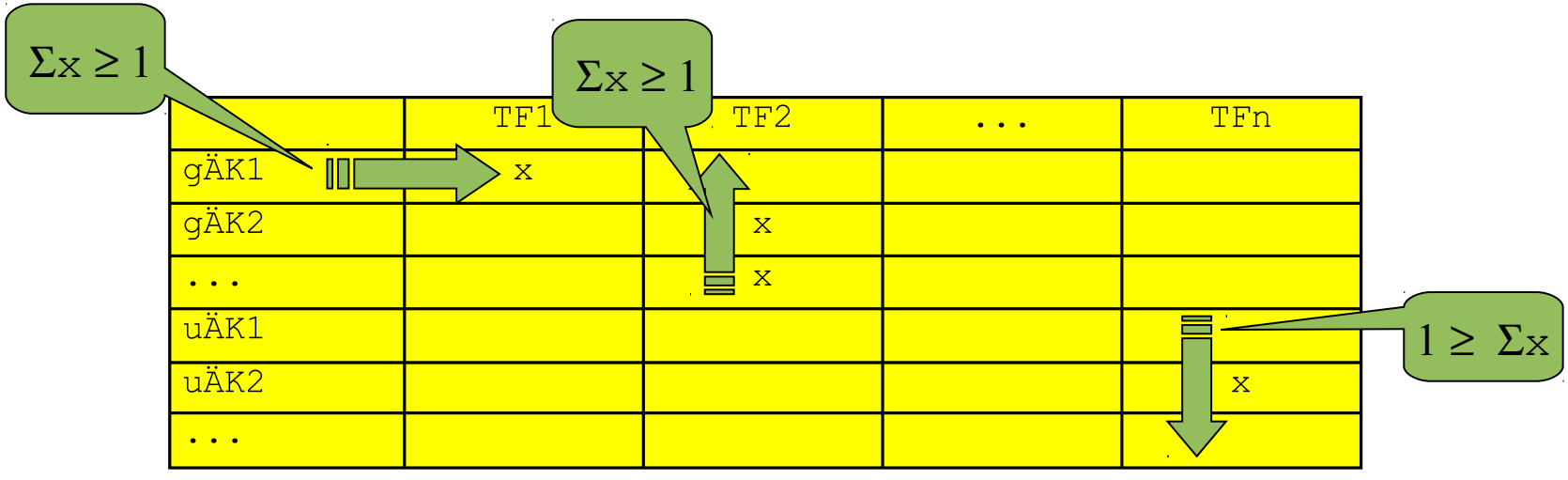
- **Beispiel:** Ermittlung von 18 Äquivalenzklassen aus Anforderungen für ein Eingabedatum und Testen von 15 von 18 Testfällen.
  → Erreichen von ca. 83 % **Äquivalenzklassen-Überdeckung**:
- ÄK-Überdeckung = 15 / 18 ≈ 83 %

Auswahl der Testfälle
Minimalkriterium

Alle ÄK‘s durch mindestens einen Testfall abdecken

Dabei pro Testfall:

- **Mehrere gültige Äquivalenzklassen** - für verschiedene Beschränkungen - abdecken, **oder**
- **Genau eine ungültige Äquivalenzklasse**
  → Einzelne Prüfung notwendig wegen Fehlermaskierung !



Äquivalenzklassen, Testfälle und Testdaten für ggT

```
public int ggT(int m, int n)
```

Äquivalenzklassen für Eingabeparameter n, m *(analog)*: `int`

- `gÄKx_1 : ?`
- `gÄKx_2 : ?`
- `gÄKx_3 : ?`
- `uÄKx_1 : ?`
- `uÄKx_2 : ?`

Testfälle:

TF1 : {n = ?, m = ?; ggT = ?}
TF2 : {n = ?, m = ?; ggT = ?}
TF3 : {n = ?, m = ?; ggT = ?}
TF4 : {n = ?, m = ?; ggT = ?}
TF5 : {n = ?, m = ?; ggT = ?}
TF6 : {n = ?, m = ?; ggT = ?}
TF7 : {n = ?, m = ?; ggT = ?}

Äquivalenzklassen, Testfälle und Testdaten für ggT

```
public int ggT(int m, int n)
```

Äquivalenzklassen für Eingabeparameter n, m *(analog)*: `int`

- gÄKx_1 : min_int ≤ n < 0
- gÄKx_2 : n = 0
- gÄKx_3 : 0 < n ≤ max_int
- uÄKx_1 : n < min_int
- uÄKx_2 : n > max_int

Testfälle:

TF1 : {n = -1, m = -1; ggT = 1}
TF2 : {n = 0, m = 0; ggT = 0}
TF3 : {n = 1, m = 1; ggT = 1}
TF4 : {n = min_int-1, m = -1; error}
TF5 : {n = max_int+1, m = -1; error}
TF6 : {n = -1, m = min_int-1; error}
TF7 : {n = -1, m = max_int+1; error}

	TF1	TF2	TF3	TF4	TF5	TF6	TF7
gÄK1_1	x					x	x
gÄK1_2		x					
gÄK1_3			x				
uÄK1_1				x			
UÄK1_2					x		
gÄK2_1	x			x	x		
gÄK2_2		x					
gÄK2_3			x				
uÄK2_1						x	
UÄK2_2							x

Äquivalenzklassenbildung
Vor- und Nachteile

Vorteile:

- Anzahl 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.

3.4 Black-Box-Test

3.4 Black-Box-Test

- Dynamischer Test – Grundlagen
- Idee der Black-Box-Testentwurfsverfahren
- Äquivalenzklassenbildung
- **Grenzwertanalyse**
- Zustandsbasierter Test
- Entscheidungstabellentest
- Weitere Black-Box-Testentwurfsverfahren

Grenzwertanalyse
Idee

- **Idee**: **Grenzbereiche** in **Verzweigungs- und Schleifenbedingungen**, für die die Bedingung gerade noch zutrifft.
  - Solche Fallunterscheidungen: **fehlerträchtig**.
  - Testdaten, die solche Grenzwerte prüfen, decken Fehlerwirkungen mit höherer Wahrscheinlichkeit.
- Beste Erfolge bei Kombination mit anderen Verfahren.
- Bei Kombination mit **Äquivalenzklassenbildung**:
  - **Grenzen der ÄK** testen.
  - Vorkommen jeder »Rand« einer ÄK in einer Testdatenkombination.

Grenzwertanalyse
Ziel

Ziel: Auswahl von Werten aus Äquivalenzklasse bestehend aus geordneter Menge.

Vorgehen:

- Schritt 1: **Auswahl von Testwerten:** Direkt oder neben beide Grenzen einer Eingabeäquivalenzklasse.
  - Äquivalenzklasse ist **Wertebereich:** Nimm größten und kleinsten Wert.
  - Äquivalenzklasse ist **Anzahl von Werten:** Nimm größte und kleinste gültige Anzahl.
- Schritt 2: Auswahl von Testwerten: Direkt oder neben beide Grenzen einer Ausgabeäquivalenzklasse.

Grenzwertanalyse
Anwendung

In Regel: Grenzwert und Werte über bzw. unter dem Grenzwert testen.

Atomare (geordnete) Bereiche:

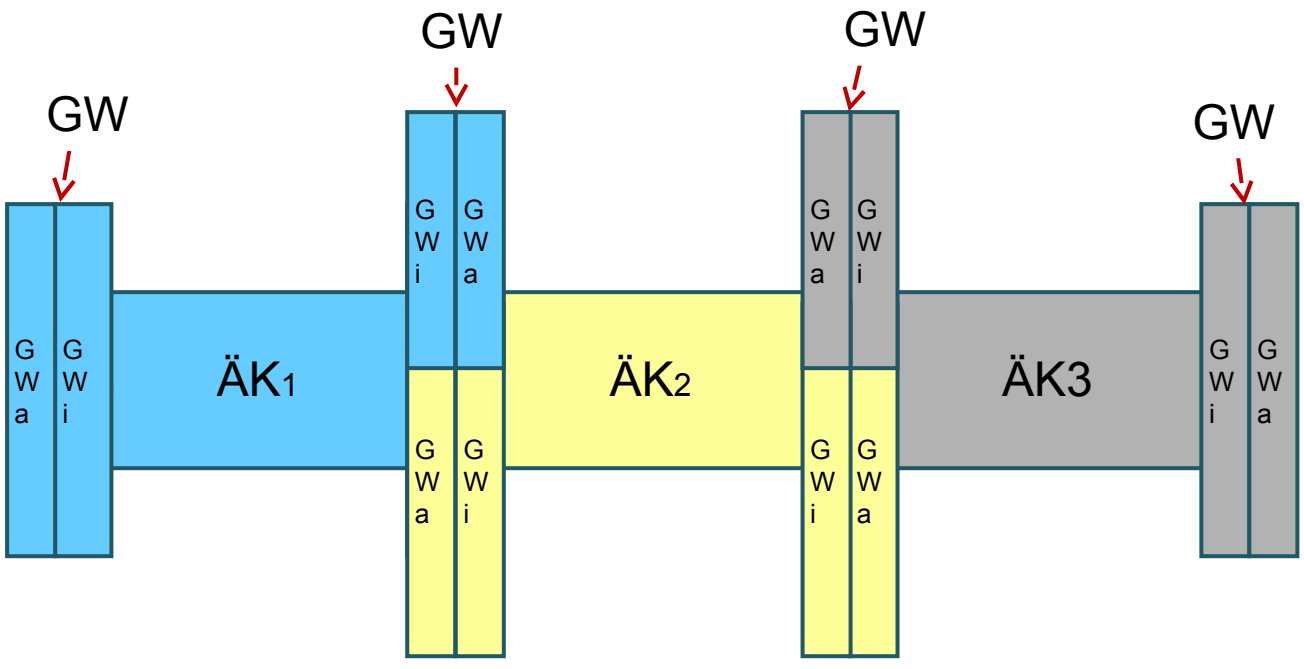
- Werte auf den Grenzen,
- Werte »rechts bzw. links neben« den Grenzen.

Mengenwertige Bereiche:

- Kleinste und größte gültige Anzahl,
- Zweitkleinste und zweitgrößte gültige Anzahl,
- Kleinste und größte ungültige Anzahl.

Fallen bei Äquivalenzklassen für geordnete Bereiche obere und untere Grenze zweier ÄK zusammen, dann auch die entsprechenden Testfälle.

Zusammenfallen der Grenzwerte **benachbarter Äquivalenzklassen**:



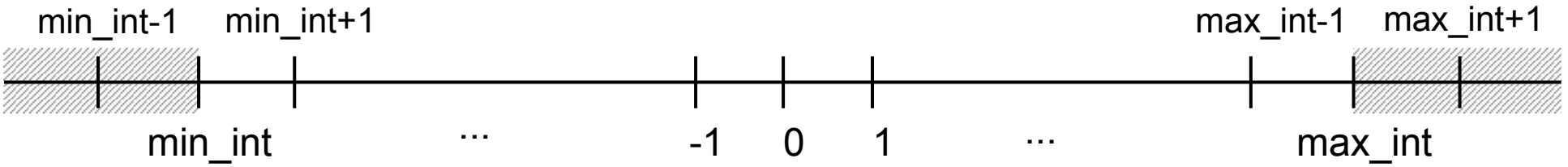
An Grenzen immer zwei Tests, egal ob auf bzw. nur vor oder nach Grenze testen.

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

Beispiele zur Grenzwertanalyse

Datentyp	Grenzen	Größer	Kleiner
integer	?	?	?
char[5]	?	?	?
double	?	?	?

Beispiele zur Grenzwertanalyse



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

Grenzwertanalyse
Ausgangskriterium (Testendekriterium)

Analog zum Ausgangskriterium der Äquivalenzklassenbildung:
Festlegung einer anzustrebenden **Überdeckung der Grenzwerte (GW)** vorab und Berechnung nach Durchführung der Tests:

GW-Überdeckungsgrad = Anzahl getestete GW / Gesamtzahl GW

Grenzwertanalyse
Vorgehen

- **Grenzen des Eingabebereichs**, z.B.:
  - 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**, z.B.:
  - Eingabedatei mit 1 bis 365 Sätzen; Testfälle 0, 1, 365, 366 (2, 364) Sätze
- **Grenzen des Ausgabebereichs**, z.B.:
  - Programm errechnet Beitrag, der zwischen 0,00 EUR und 600 EUR liegt; Testfälle: 0; 600 EUR; Beiträge < 0; (knapp >0); und für > 600; (knapp < 600)
- **Grenzen der erlaubten Anzahl von Ausgabewerten**, z.B.:
  - Ausgabe von 1 bis 4 Daten; Testfälle: Für 0, 1, 4 und 5 (2, 3) Ausgabewerte
- **Erstes und letztes Element bei geordneten Mengen beachten**.
- **Komplexe Datenstrukturen: leere Mengen testen**.
- **Bei numerischen Berechnungen**: Wahl von eng zusammen und weit auseinander liegender Werte.

Grenzwertanalyse: Beispiel (s. Äquivalenzklassentest)

- Grenzwerte für Eingaben:

Äquivalenzklasse	Gültige Grenzwerte	Ungültige Grenzwerte
(1) Anzahl Parameter	2	1, 3
(2) Dateiname (Länge)	1, 6	0, 7
(6) Zeilenanzahl (Ziffern)	1, 3	0, 4
(7) Zeilenanzahl	1, 999	0, 1000

Keine Untersuchung der Äquivalenzklassen 3-5, da keine geordnete Mengen!

Testdaten für **gültige Eingaben**:

Äquivalenzklasse	Testdaten	Ausgewählt bei Äquivalenzklassentest
(1)	PRINT abc1 22	ja
(2)	PRINT a 100	eventuell
(2)	PRINT abcdef 200	eventuell
(6)	PRINT abc 8	eventuell
(6)	PRINT abc 345	eventuell
(7)	PRINT abc 1	eventuell
(7)	PRINT abc 999	eventuell

Testdaten für **ungültige Eingaben:**

Äquivalenzklasse	Testdaten	Ausgewählt bei Äquivalenzklassentest
(1b)	PRINT abc	ja
(1c)	PRINT abc 20 300	eventuell
(2a)	PRINT 20	ja
(2b)	PRINT abcdefg 20	eventuell
(6a)	PRINT abc 4568	eventuell
(7a)	PRINT abc 0	eventuell
(7b)	PRINT abc 1000	eventuell

Grenzwerte für Ausgaben:

- Es können X Seiten gedruckt werden, $1 \leq X \leq 20$
- Letzte Seite enthält Y Zeilen, $1 \leq Y \leq 45$
- Ersten X-1 Seiten enthalten jeweils 45 Zeilen

Äquivalenzklasse	Gültige Grenzwerte	Ungültige Grenzwerte
(1) Anzahl Seiten	1, 20	0, 21
(2) Anzahl Zeilen pro Seite	1, 45	0, 46

Grenzwertanalyse
Beispiel

- Testdaten für **gültige Ausgaben**:
  - PRINT abc 45 (X=1, Y=45)
  - PRINT abc 900 (X=20, Y=45)
  - PRINT abc 46 (X=2, Y=1)
- Testdaten für **ungültige Ausgaben:**
  - PRINT abc 0 (X=0, Y=0)
  - PRINT abc 901 (X=21, Y=1)
  - PRINT abc 46 (X=1, Y=46)

Grenzwertanalyse
Vor- und Nachteile

Vorteile:

- An Grenzen von Äquivalenzklassen: **Häufiger Fehler zu finden** als innerhalb dieser Klassen.
- »Grenzwertanalyse: Eine der **nützlichsten Methoden** für Testfallentwurf bei richtiger Anwendung.«

  Myers, Glenford J.: Methodisches Testen von Programmen, Oldenbourg, 2001 (7. Auflage)
- Effiziente Kombination mit anderen Verfahren: **Freiheitsgrade bei Wahl** der Testdaten.

Nachteile:

- Rezepte für Auswahl von Testdaten schwierig anzugeben.
- Bestimmung aller relevanten Grenzwerte schwierig.
- Kreativität zur Findung erfolgreicher Testdaten gefordert.
- **Anwendung nicht effizient** genug, da einfache Erscheinung.

3.4 Black-Box-Test

3.4 Black-Box-Test

- Dynamischer Test – Grundlagen
- Idee der Black-Box-Testentwurfsverfahren
- Äquivalenzklassenbildung
- Grenzwertanalyse
- **Zustandsbasierter Test**
- Entscheidungstabellentest
- Test mittels Ursache-Wirkungs-Graphen

Zustandsbasierter Test

Bei vielen Systemen: **Einfluss** des bisherigen Ablaufs des Systems auf Berechnung der Ausgaben.

- Endlicher Automat besteht aus endlicher Anzahl von internen Konfigurationen – **Zustände**.
- Zustand eines Systems beinhaltet implizit **Informationen**.

  → Ergibt sich aus bisherigen Eingaben.

  → Nötig um Reaktion des Systems auf folgende Eingaben zu bestimmen.

System: Annahme von unterschiedlichen Zuständen beginnend vom Startzustand.

H. Balzert: Lehrbuch der Softwaretechnik, Bd. I, Spektrum, 2002

- **Auslösung von Zustandsänderungen** oder –übergänge durch Ereignisse, z.B. Funktionsaufrufe.
- Bei Zustandsänderungen Aktionen durchführbar.
- **Spezieller Zustand:** Startzustand und Endzustand.

Zustandsbasierter Test: Begriffe

Zustandsautomat: Berechnungsmodell, bestehend aus einer endlichen Anzahl von Zuständen und Zustandsübergängen, ggf. mit begleitenden Aktionen. [IEEE 610].

Zustandsübergang: Übergang zwi. zwei Zuständen einer Komponente oder eines Systems.

Zustandsdiagramm: Diagramm, das Zustände beschreibt, die System oder Komponente annehmen kann, und Ereignisse zeigt, die Zustandswechsel verursachen und/oder ergeben [IEEE 610].

Zustandsübergangstabelle: Tabelle für Darstellung resultierender Übergänge für jeden Zustand in Verbindung mit jedem möglichen Ereignis. → Gültige oder ungültige Übergänge.

Zustandsbasierter Test: Black-Box-Testentwurfsverfahren, zur Entwurf von Testfällen, um gültige und ungültige Zustandsübergänge zu prüfen.

Beispiel zur Zustandsmodellierung: Stapel (Stack)

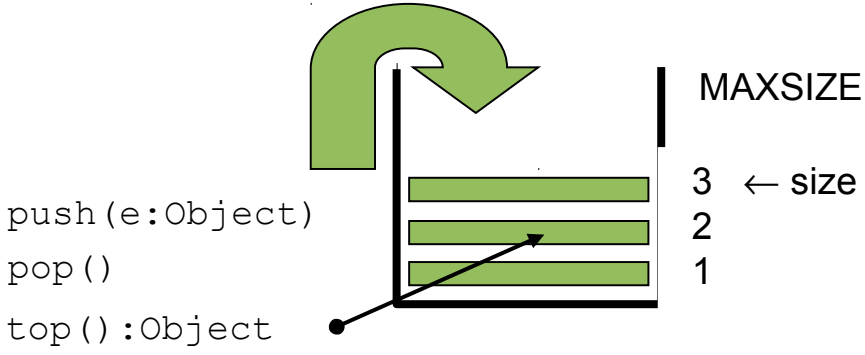
Klasse `Stapel`

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();
```

Ziele des zustandsbasierten Tests

- Nachweis der Konformität des Testobjekts zum Zustandsdiagramm (**Zustands-Konformanztest**).
- Zusätzlich Test unter nicht-konformanten Benutzungen (**Zustands-Robustheitstest**).

Arbeitsschritte des zustandsbasierten Tests

1. Erstellung des **Zustandsdiagrammes**
2. Prüfung auf **Vollständigkeit**
3. Ableiten des **Übergangsbaumes** für den **Zustands-Konformanztest**
4. Erweitern des Übergangsbaumes für den **Zustands-Robustheitstest**
5. Generieren der **Botschaftssequenzen** und Ergänzen der Botschaftsparameter
6. **Ausführen der Tests** und **Überdeckungsmessung**

1. Erstellung des Zustandsdiagrammes

Drei Zustände:

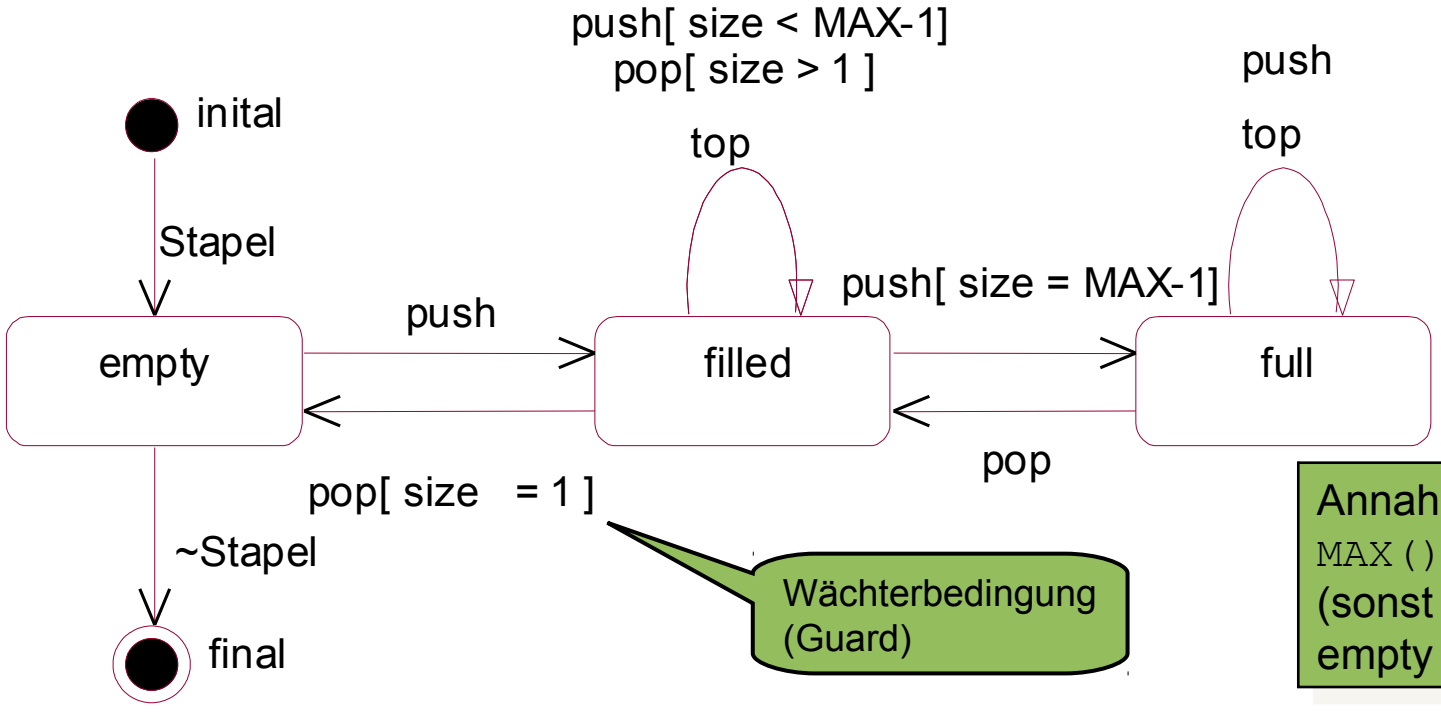
```
empty:  size() = 0;
filled: 0 < size() < MAX();
full:   size() = MAX();
```

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!)
```

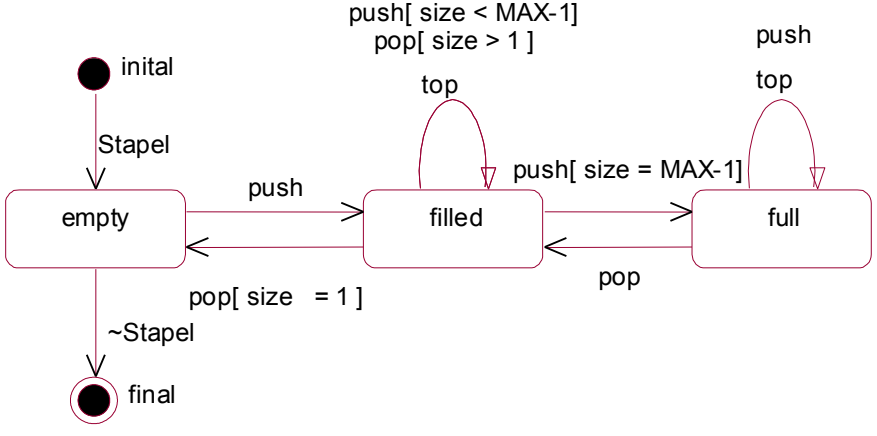


Annahme zur Vereinfachung:
`MAX()>1`
(sonst noch Transition „push“ von empty nach full benötigt)

2. Prüfung auf Vollständigkeit

Zustandsdiagramm hinsichtlich der »**Vollständigkeit**« untersuchen:

- **Zustandsübergangstabelle** anlegen.
- **Wächterbedingungen** bez. eines Ereignisses auf »Vollständigkeit« und Konsistenz prüfen.
- Nicht spezifizierte Zustands/Ereignis-Paare hinterfragen.



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

3. Aufbau des Übergangsbaumes: Zustands-Konformanztest

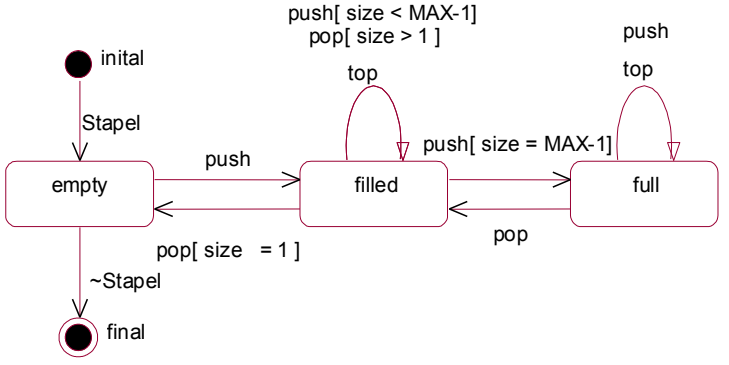
1. Anfangszustand: **Wurzel** des Baumes.
2. Für jeden möglichen **Übergang** vom Anfangszustand zum Folgezustand im Zustandsdiagramm:

   - Übergangsbaum erhält von Wurzel aus **Zweig** zum **Knoten: Nachfolgezustand**.

   - Notieren: Ereignis und Wächterbedingung am Zweig.
3. Schritt 2 für jedes Blatt des Übergangsbaums wiederholen, bis eine der **Endbedingungen** eintritt:
   - Dem Blatt entsprechender Zustand: Auf »höherer Ebene« einmal im Baum enthalten.
   - Dem Blatt entsprechender Zustand: Endzustand und hat keine weitere Übergänge zu berücksichtigen.

→ Jedes Blatt unabhängig von davor liegender Historie betrachten.

Anm.: **Zyklen** werden dadurch höchstens einmal durchlaufen, garantiert endlichen Übergangsbaum, endliche Länge von Testfolgen und somit endliche Menge von Testfolgen.



?

3. Aufbau des Übergangsbaumes: Zustands-Konformanztest

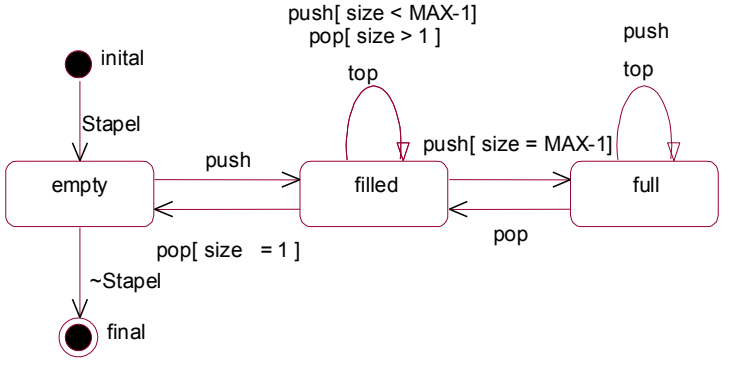
1. Anfangszustand: **Wurzel** des Baumes.
2. Für jeden möglichen **Übergang** vom Anfangszustand zum Folgezustand im Zustandsdiagramm:

   - Übergangsbaum erhält von Wurzel aus **Zweig** zum **Knoten: Nachfolgezustand**.

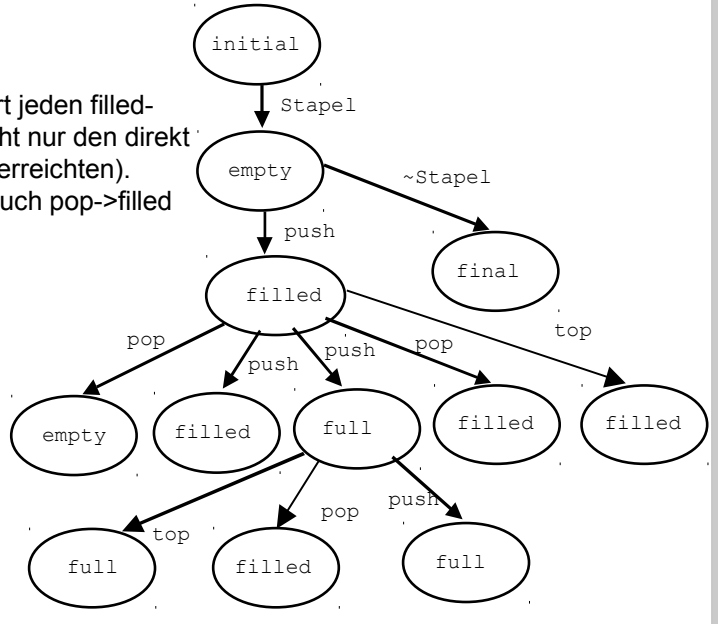
   - Notieren: Ereignis und Wächterbedingung am Zweig.
3. Schritt 2 für jedes Blatt des Übergangsbaums wiederholen, bis eine der **Endbedingungen** eintritt:
   - Dem Blatt entsprechender Zustand: Auf »höherer Ebene« einmal im Baum enthalten.
   - Dem Blatt entsprechender Zustand: Endzustand und hat keine weitere Übergänge zu berücksichtigen.

→ Jedes Blatt unabhängig von davor liegender Historie betrachten.

Anm.: **Zyklen** werden dadurch höchstens einmal durchlaufen, garantiert endlichen Übergangsbaum, endliche Länge von Testfolgen und somit endliche Menge von Testfolgen.



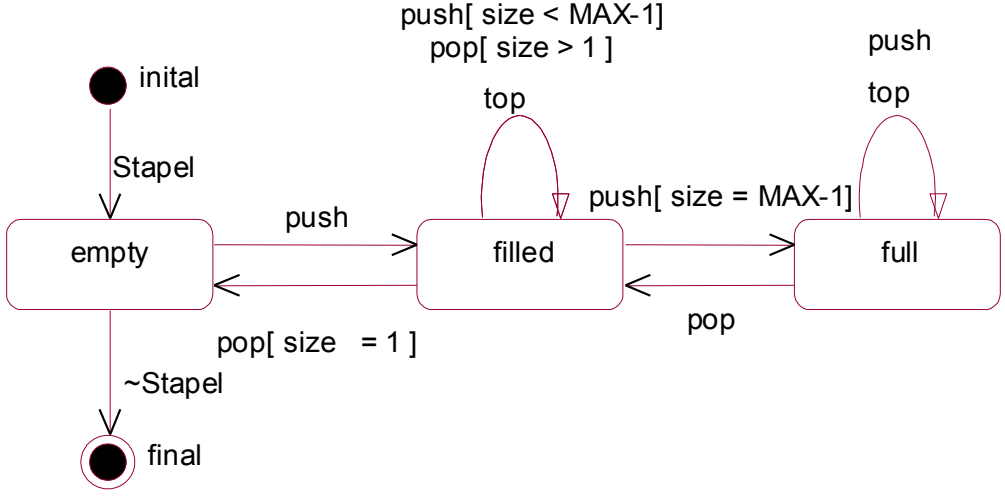
Repräsentiert jeden filled-Zustand (nicht nur den direkt nach empty erreichten). Deswegen auch pop->filled

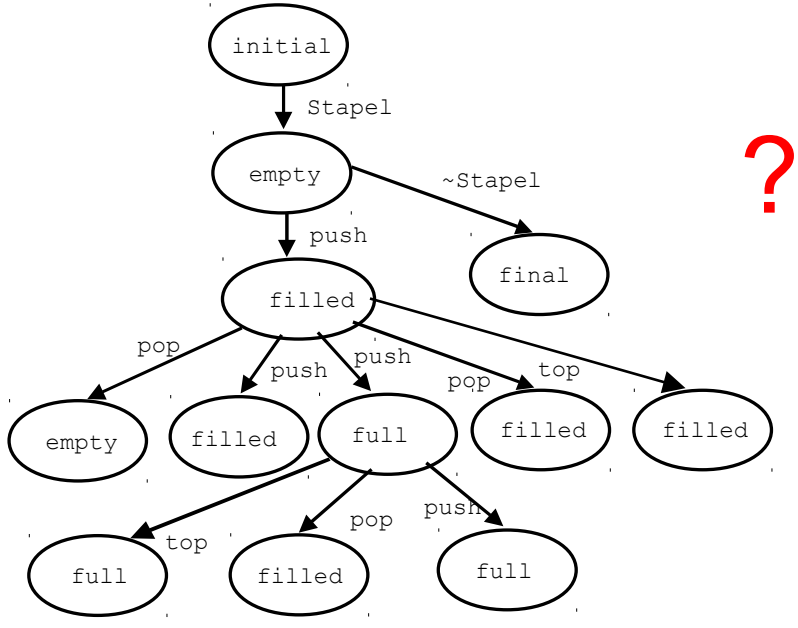


(Wächterbedingungen hier nicht dargestellt)

4. Erweitern des Übergangsbaumes: Zustands-Robustheitstest

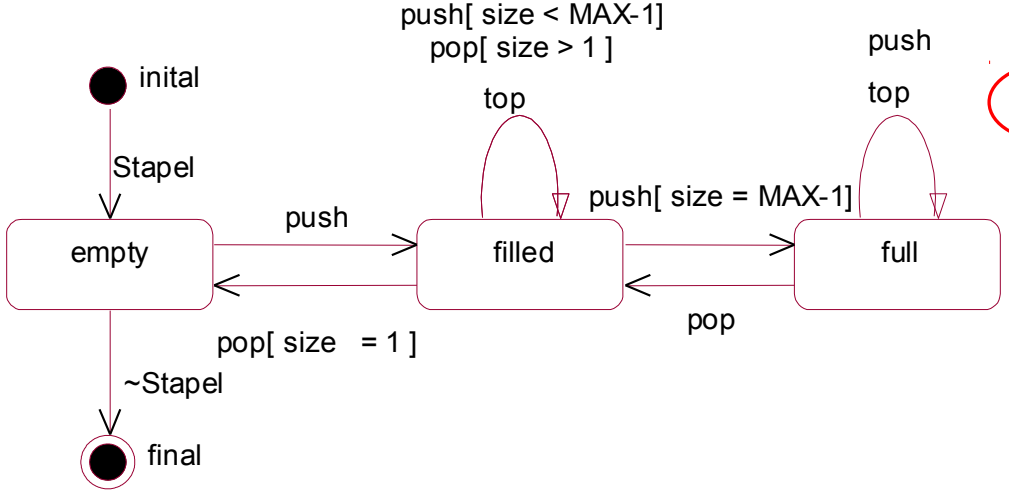
- **Robustheit** unter spezifikationsverletzenden Benutzungen prüfen.
- Für Botschaften, für die aus betrachtetem Knoten kein Übergang spezifiziert ist, **Übergangsbaum** um neuen »Fehler«-Zustand erweitern.

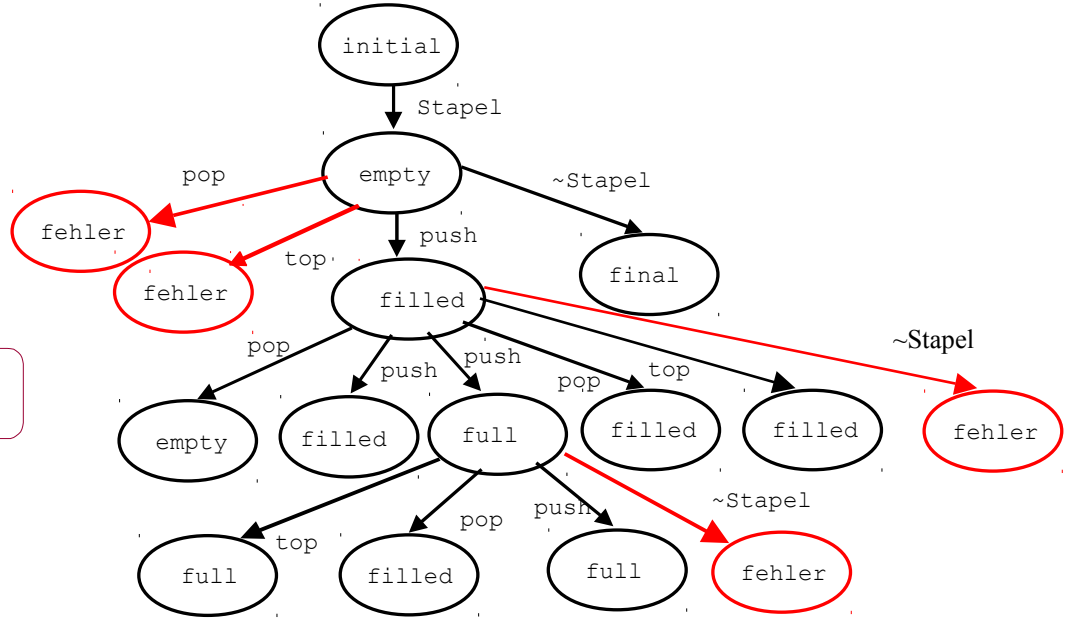




4. Erweitern des Übergangsbaumes: Zustands-Robustheitstest

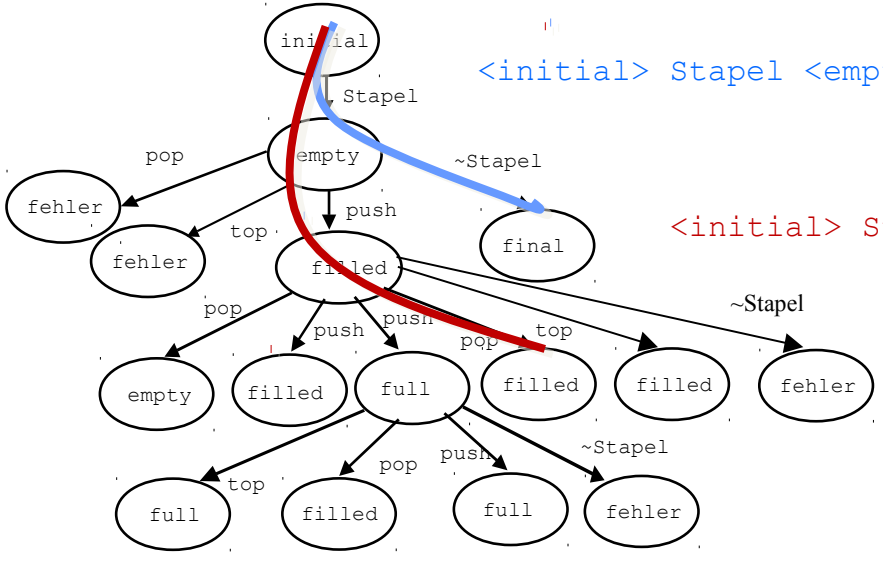
- **Robustheit** unter spezifikationsverletzenden Benutzungen prüfen.
- Für Botschaften, für die aus betrachtetem Knoten kein Übergang spezifiziert ist, **Übergangsbaum** um neuen »Fehler«-Zustand erweitern.





5. Generieren der Testfälle (1)

- Pfade von Wurzel zu Blättern im erweiterten Übergangsbaum: **Funktions-Sequenzen**.
- **Stimulierung** des Testobjekts mit entsprechenden Funktionsaufrufen deckt alle Zustände und Zustandsübergänge im Zustandsdiagramm ab.
- Nicht unbedingt alle möglichen Variablenbelegungen → **Nicht kompletter** theoretisch möglicher **Zustandsraum**.
- Parameter ergänzen.



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

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

5. Generieren der Testfälle (2)

- **Stimulierung** des Testobjekts mit entsprechenden Funktionsaufrufen deckt alle Zustände und Zustandsübergänge im Zustandsdiagramm ab.
- Nicht unbedingt alle möglichen Variablenbelegungen → **Nicht kompletter** theoretisch möglicher **Zustandsraum**.
- Für Konformanztests: Wächterbedingungen beachten !

Zustands-Konformanztest:

```
K1 = <initial> new Stapel() <empty> ~Stapel() <final>
K2 = <initial> new Stapel() <empty> push() <filled> pop() <empty>
K3 = <initial> new Stapel() <empty> push() <filled> push() <filled>
K4 = <initial> new Stapel() <empty> push() <filled> pop() <filled>
...
K8 = <initial> new Stapel() <empty> push() <filled> push() <full> push() <full>
```

Welche Folge verletzt Wächter-bedingung ?

Zustands-Robustheitstest:

```
R1 = <initial> new Stapel() <empty> pop() <fehler>
R2 = <initial> new Stapel() <empty> top() <fehler>
R3 = <initial> new Stapel() <empty> push() <filled> ~Stapel() <fehler>
R4 = <initial> new Stapel() <empty> push() <filled> push() <full> ~Stapel() <fehler>
```

5. Generieren der Testfälle (2)

- **Stimulierung** des Testobjekts mit entsprechenden Funktionsaufrufen deckt alle Zustände und Zustandsübergänge im Zustandsdiagramm ab
- Nicht unbedingt alle möglichen Variablenbelegungen → **Nicht kompletter** theoretisch möglicher **Zustandsraum**.
- Für Konformanztests: Wächterbedingungen beachten !
- Konformanztests, die Wächterbedingungen verletzen, **sinnvoll für Robustheitstests**.

Zustands-Konformanztest:

```
K1 = <initial> new Stapel() <empty> ~Stapel() <final>
K2 = <initial> new Stapel() <empty> push() <filled> pop
K3 = <initial> new Stapel() <empty> push() <filled> push() <filled>
K4 = <initial> new Stapel() <empty> push() <filled> pop() <filled>
...
K8 = <initial> new Stapel() <empty> push() <filled> push() <full> push() <full>
```

Wächterbedingung:
size() > 1 !!

push() <filled> pop() <filled>

Zustands-Robustheitstest:

```
R1 = <initial> new Stapel() <empty> pop() <fehler>
R2 = <initial> new Stapel() <empty> top() <fehler>
R3 = <initial> new Stapel() <empty> push() <filled> ~Stapel() <fehler>
R4 = <initial> new Stapel() <empty> push() <filled> push() <full> ~Stapel() <fehler>
```

Zustandsbasierte Testfälle

Vollständiger **zustandsbasierter Testfall** umfasst:

- Anfangszustand des Testobjektes (Komponente oder System)
- Eingaben für das Testobjekt
- Erwartete Ausgaben bzw. das erwartete Verhalten
- Erwarteter Endzustand

Für jeden **im Testfall erwarteten Zustandsübergang**

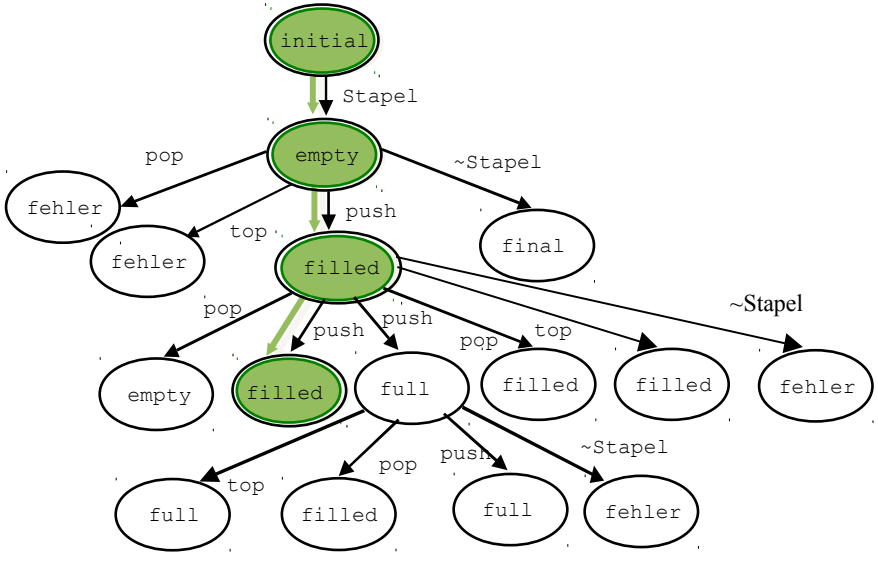
- Zustand vor dem Übergang
- Auslösendes Ereignis, das den Übergang bewirkt
- Erwartete Reaktion, ausgelöst durch den Übergang
- Nächster erwarteter Zustand

festlegen.

Ausführen der Tests

- Testfälle in **Testskript** verkapseln.
- Unter Benutzung **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;
```



Minimalkriterium: Jeder Zustand mindestens einmal eingenommen.

Z-Überdeckungsgrad = Anzahl getestete Z / Gesamtzahl Z

Weitere Kriterien:

- Jeder Zustandsübergang mindestens einmal ausgeführt.

  ZÜ-Überdeckungsgrad = Anzahl getestete ZÜ / Gesamtzahl ZÜ
- Alle spezifikationsverletzenden Zustandsübergänge angeregt.
- Jede Funktion mindestens einmal ausgeführt.

Bei **hoch kritischen Anwendungen:**

- Alle Zustandsübergänge und »Zyklen« im Zustandsdiagramm.
- Alle Zustandsübergänge in jeder beliebigen Reihenfolge mit allen möglichen Zuständen, auch mehrfach hintereinander.

Tipps für den zustandsbasierten Test

- **Zustandsdiagramm**:
  - bei Spezifikation unter **Testgesichtspunkten** bewerten,
  - bei hoher Anzahl von Zuständen und Übergängen auf erhöhten Testaufwand hinweisen,
  - Soweit möglich **auf Vereinfachung dringen**.
- Bei **Spezifikation** achten:
  - **Unterschiedliche Zustände** leicht ermittelbar.
  - **Keine vielfältige Kombination** von Werten von unterschiedlichen Variablen.

  Ggf. **Werkzeuge** zum **Model-Checking** verwenden:
  - **Erreichbarkeit** von Zuständen.
  - Bei interagierenden Testobjekten: **Deadlock, Livelock**, ...

Bewertung des zustandsbasierten Tests

Zustand: Konstellation unterschiedlicher Werte der Variablen:

- Aufgespannter **Zustandsraum:** Sehr **komplex**.
  [Folge der Systemkomplexität und nicht des Testansatzes.]
- **Überprüfung** einzelner Testfälle: **Aufwändig**.

Zustandsbasierte Tests dort, wo **Funktionalität** durch jeweiligen **Zustand** des Testobjektes unterschiedlich **beeinflusst** wird.

- Keine Berücksichtigung anderer vorgestellter Testentwurfsverfahren.
  → Gehen nicht auf Abhängigkeit des Verhaltens der Funktionen vom Zustand ein.

Geeignet zum Test **objektorientierter Systeme**:

- Objekte können unterschiedliche Zustände annehmen.
- Jeweilige **Methoden** zur Manipulation der Objekte: Entsprechend auf unterschiedliche **Zustände** reagieren.
- **Beim objektorientierten Testen:** Zustandsbasierter Test hat herausgehobene Bedeutung.

3.4 Black-Box-Test

3.4 Black-Box-Test

- Dynamischer Test – Grundlagen
- Idee der Black-Box-Testentwurfsverfahren
- Äquivalenzklassenbildung
- Grenzwertanalyse
- Zustandsbasierter Test
- **Entscheidungstabellentest**
- Test mittels Ursache-Wirkungs-Graphen

Entscheidungstabellentest

- Anwendbar bei Systemanforderungen mit
  - **logischen Bedingungen** und
  - komplexen, vom System umzusetzende **Regeln** in Geschäftsprozessen.
- Spezifikation untersuchen und Eingabebedingungen und Aktionen des Systems ermitteln und festsetzen (»wahr« oder »falsch«).
- **Entscheidungstabelle** enthält Kombinationen von »wahr« und »falsch« für alle Eingabebedingungen und daraus resultierende Aktionen.
- Jede **Spalte** der Tabelle: **Regel im Geschäftsprozess.**

  → Definiert eindeutige Kombination der Bedingungen.

  → Zieht Ausführung mit dieser Regel verbundene Aktionen mit sich.
- Bei Entscheidungstabellentest verwendeter **Standardüberdeckungsgrad**: Wenigstens **ein Testfall pro Spalte**.

Entscheidungstabellentest

Entscheidungstabellentest – Die vier Quadranten einer Entscheidungstabelle

Bedingungen	Regeln
Aktionen	Aktionszeiger

- Bedingungen:
  - Mögliche Zustände von Objekten
- Regeln:
  - Kombinationen von Bedingungswerten
- Aktionen:
  - Aktivitäten, die abhängig von den Regeln auszuführen sind
- Aktionszeiger:
  - Belegungen der Bedingungen mit Aktionen

Entscheidungstabellentest: Beispiel

Geschäftsregeln im Warenwirtschaftssystem:

- **Bestellmenge** muss größer als Null sein.
- **Teil-Lieferungen** nicht erlaubt.
- Bei Annahme einer Bestellung muss **Lagermenge** entsprechend reduzieren.
- Beim Unterstritten von Mindestmenge eines Lagerartikels: **Nachbestellung.**

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 „–“)

Analyse von Entscheidungstabellen

- ET ***vollständig***, wenn bei n Bedingungen alle 2^n Kombinationen enthalten (Spalten im oberen Teil).
- ET ***redundanzfrei***, wenn verschiedene Bedingungen zu anderen Aktionen führen.
- ET ***widerspruchsfrei***, wenn logische Beziehungen zwischen Bedingungen zu konsistenten Aktionen führen.

Vollständig, redundanzfrei, widerspruchsfrei ?

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 > 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	

Analyse von Entscheidungstabellen

- ET ***vollständig***, wenn bei n Bedingungen alle 2^n Kombinationen enthalten (Spalten im oberen Teil).
- ET ***redundanzfrei***, wenn verschiedene Bedingungen zu anderen Aktionen führen.
- ET ***widerspruchsfrei***, wenn logische Beziehungen zwischen Bedingungen zu konsistenten Aktionen führen.

Vollständig, redundanzfrei, widerspruchsfrei.

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			

Redundanzfrei, widerspruchsfrei.

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	

Entscheidungstabellentest: Testfälle und -daten

Jede Spalte (Regel): Ein Testfall.

- Voraussetzungen pro Tabelle gleich.
- Bedingungen beziehen sich auf Eingaben.
- Aktionen spiegeln vorausgesagtes Ergebnis wider.

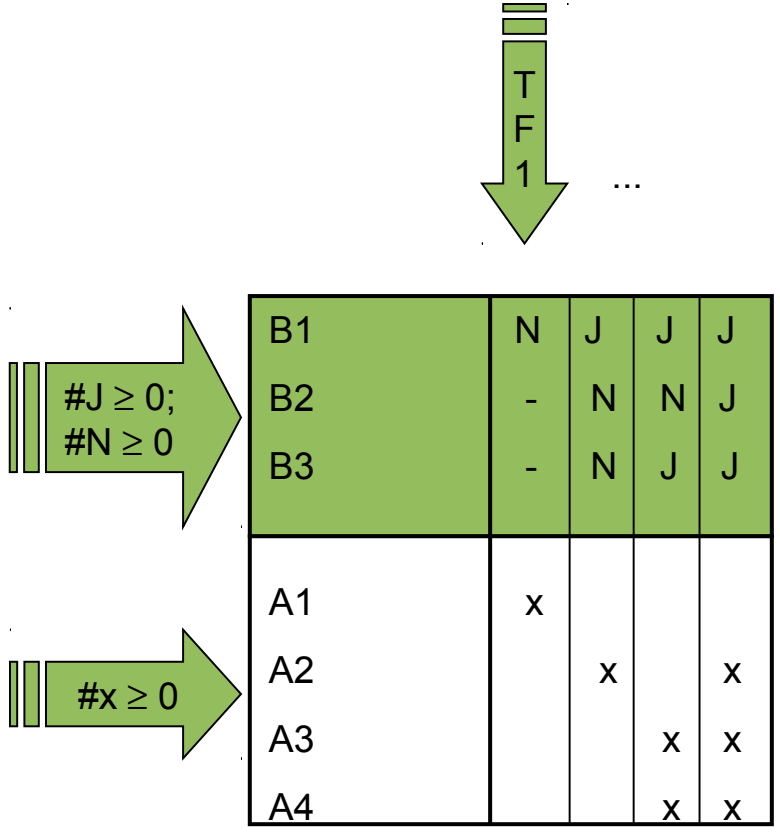
Überdeckungskriterien:

- Alle Bedingungen mindestens einmal „N“ bzw. „J“.
- Alle Aktionen mindestens einmal „x“.
- Alle Bedingungskombinationen.

Konkrete Testdaten aus Wertebereichen ableiten:

- Äquivalenzklassenbildung
- Grenzwertanalyse
- ...

Testfall pro Regel:



Bewertung des Entscheidungstabellentests

Vorteile:

- Stärke des **Entscheidungstabellentests:** Ableitung Kombinationen von Bedingungen, die andernfalls nicht getestet werden.
- Entscheidungstabellen-Technik zur **Problemlösung** anwendbar, wenn Abläufe von mehreren logischen Entscheidungen abhängen.
- **Logische Zusammenhänge** systematisch formulierbar.
- Entscheidungstabellen auf Redundanz, Widerspruchsfreiheit und Vollständigkeit prüfbar.
- Zwingen nicht zur Strukturierung eines Ablaufs.
- Anwendbar auch bei einfacheren zustandsabhängigen Problemen.

Nachteile:

- **Unübersichtlich** bei zu vielen Bedingungen.
- **Zusammenhänge** zwischen einzelnen Bedingungen nur **implizit ausdrückbar.**

3.4 Black-Box-Test

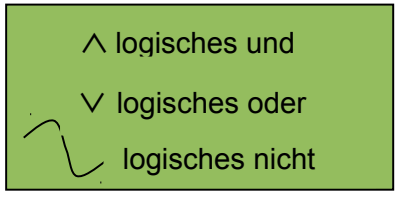
3.4 Black-Box-Test

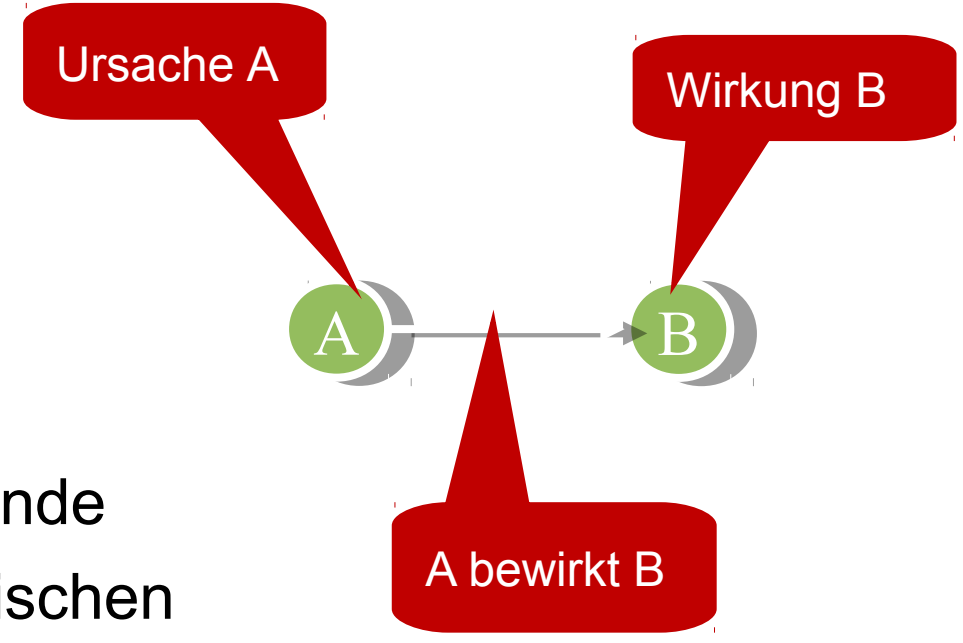
- Dynamischer Test – Grundlagen
- Idee der Black-Box-Testentwurfsverfahren
- Äquivalenzklassenbildung
- Grenzwertanalyse
- Zustandsbasierter Test
- Entscheidungstabellentest
- **Test mittels Ursache-Wirkungs-Graphen**

Ursache-Wirkungsgraphen
Definition

Graphische Beschreibung von **logischen Wirkzusammenhängen**.

- **Ursachen:**
  - Eingaben
  - Dateiinhalte / Datenbanken
  - Initiale Systemzustände
- **Wirkungen:**
  - Ausgaben
  - Resultierende Systemzustände
- **Logische Verknüpfungen** zwischen Wirkungen:





Ursache-Wirkungsgraphen
Vorgehen (1)

Bei Grenzwertanalyse **unberücksichtigt:** Test von Kombinationen von Eingabe- bzw. Ausgabewerten.

- Beispiel: Drucken von 20 Seiten, 45 Zeilen auf letzter Seite
- 10 Eingabebedingungen mit jeweils 4 zu testenden Werten: 410 = 1.048.576 zu testende Kombinationen!

Beschränkung zu testender Kombinationen durch Ursache- oder Wirkungsgraphen.

Vorgehen beim **Ursache-/Wirkungsgraphen:**

- **Schritt 1:** Spezifikation in bearbeitbare Teile zerlegen.
- **Schritt 2:** Ursachen und Wirkungen des Programms anhand Spezifikation identifizieren.
  - **Ursache:** Eingabebedingung, denen Boolesche Wahrheitswerte zugeordnet werden können.
  - **Wirkung:** Ausgabebedingung oder Systemtransformation.

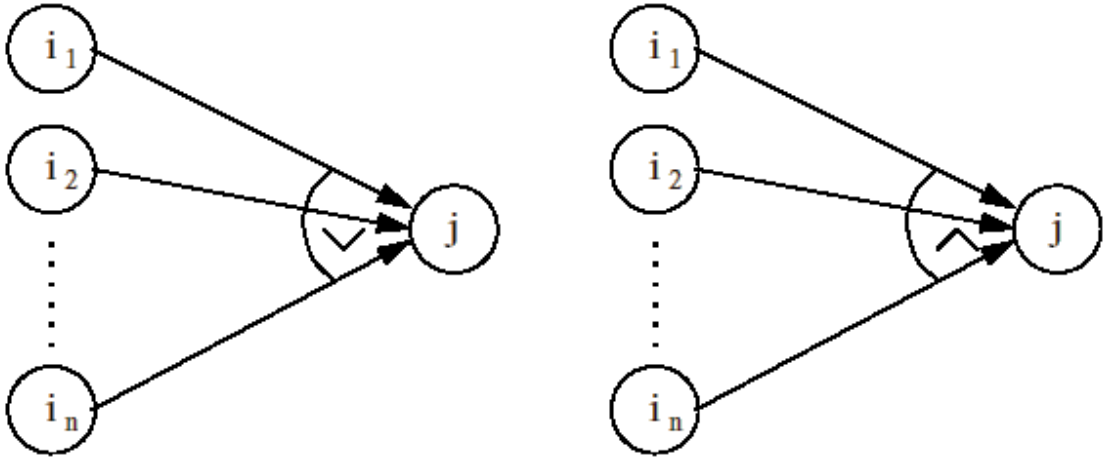
Schritt 3: Logische Beziehungen zwischen Ursachen und Wirkungen als gerichteten Boolschen Graphen darstellen.

- **Boolescher Graph:** Graph, dessen Knoten logische Verknüpfungen zugeordnet werden.
- Jede Ursache/Wirkung mit Zahl i durch einen Knoten darstellbar.
- **Identische Verknüpfung:** Wirkung j gleich Ursache i.
- **Negations-Verknüpfung:** Wirkung j vorhanden, wenn Ursache i nicht vorhanden.

Identität Negation

Schritt 3 ff.:

- **Oder-Verknüpfung:** Wirkung j vorhanden, wenn Ursache i1 oder i2 oder ... in vorhanden.
- **Und-Verknüpfung:** Wirkung j vorhanden, wenn Ursache i1 und i2 und ... in vorhanden.

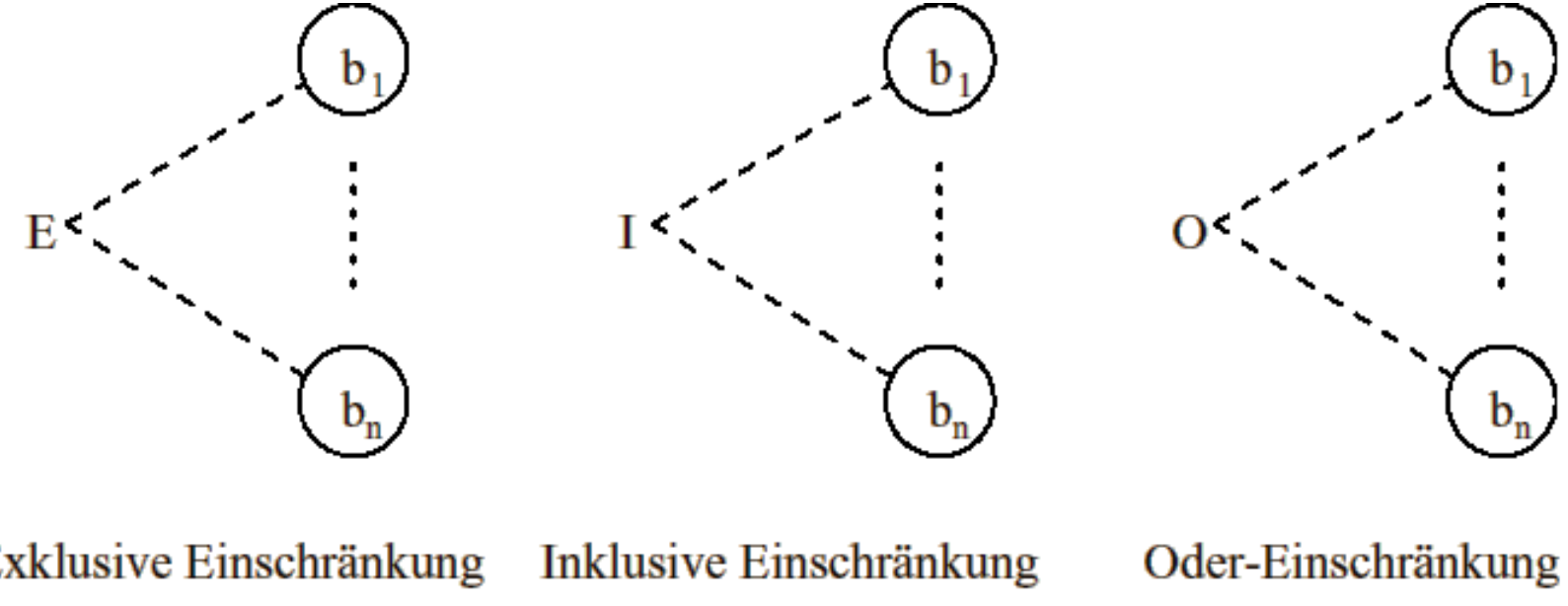


Oder - Verknüpfung Und - Verknüpfung

Ursache-Wirkungsgraphen
Vorgehen (4)

Schritt 4: UWG mit Einschränkungen versehen: Ausschließen vieler Testfälle.

- Einschränkung E (exklusiv): Höchstens eine der Bedingungen b1, ..., bn erfüllt.
- Einschränkung I (inklusiv): Mindestens eine der Bedingungen b1, ..., bn erfüllt.
- Einschränkung O (oder): Eine und nur eine der Bedingungen b1, ..., bn erfüllt.



Exklusive Einschränkung Inklusive Einschränkung Oder-Einschränkung

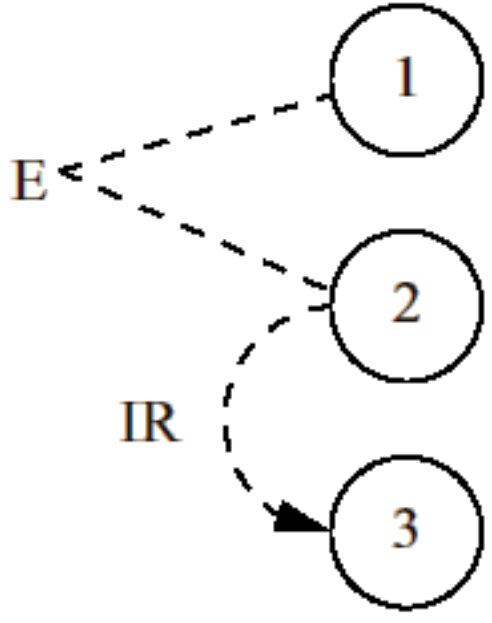
Ursache-Wirkungsgraphen
Vorgehen (5)

- **Schritt 4 ff.**
  - **Einschränkung R (requires):** Erfüllung von Bedingung a erfordert Erfüllung von Bedingung b.
  - **Einschränkung M (maskiert):** Erfüllung von Bedingung a impliziert Nichterfüllung von Bedingung b.
  - **Einschränkung IR (irrelevant):** Erfüllung von Bedingung a impliziert: Bedingung b irrelevant.
- **Schritt 5:** Aus UWG Wertetabelle ableiten mit Ursachen und Wirkungen als Zeilen und Kombinationen von Ursachen als Spalten.
- **Schritt 6:**
  - Kombinationen von Ursachen und im fehlerfreien Fall zu erwartende Wirkungen in Wertetabelle eintragen.
  - Kombinationen für nicht widersprechende verschiedene Ursachen zusammenfassen.
- **Schritt 7:** Für jede Spalte der Wertetabelle Eingabedatum ermitteln.

Ursache-Wirkungsgraphen
Beispiel

Ursache-/Wirkungsgraph: **Beispiel 1:**

- Druckbefehl, dessen erster Operand angibt, ob ab Dateibeginn („START“) oder an einer bestimmten Zeile („Startzeile“) zu drucken ist.
- **Ursachen (d.h. Eingabebedingungen):**
  - 1. Erster Operand: „Startzeile“.
  - 2. Erster Operand: „START“.
  - 3. Operand „Startzeile“ enthält 1 bis 6 Zeichen.



Ursache-/Wirkungsgraph: **Beispiel 2**

- Aufgabe des Textformatierers: Text bestehend aus Wörtern getrennt durch Blank (BL) oder Newline (NL) in Zeilen formatieren, dass keine Zeile mehr als **MAXPOS Zeichen** hat:
  - Zeilenumbrüche nur an BL- oder NL-Zeichen auftreten.
  - Anzahl Wörter je Zeile maximal.
  - **Wort:** Maximale Zeichenfolge, die nicht Zeichen BL und NL enthält.
  - NL-Zeichen als letztes Zeichen in auszugebender Zeile bei Berechung der Zeichenanzahl und dem Vergleich mit Maximalzahl MAXPOS nicht mitzählen.
  - Wenn Eingabe Wort mit mehr als MAXPOS Zeichen enthält: **Textformatierung abbrechen.**
- **Schritt 1 (Spezifikation in bearbeitbare Teile zerlegen): OK.**

Schritt 2 (Ursachen und Wirkungen identifizieren):

Ursachen:

- B1: ϵ {BL, NL, EOF}
- B2: Länge(w) ≤ MAXPOS
- B3: Länge(w) ≤ MAXPOS - f -1
- B4: f > 0
  - z aktuell eingelesenes Zeichen, w aktuell aufgebautes Wort, f Anzahl der Zeichen in der aktuellen Ausgabezeile, MAXPOS maximale Länge der Ausgabezeile

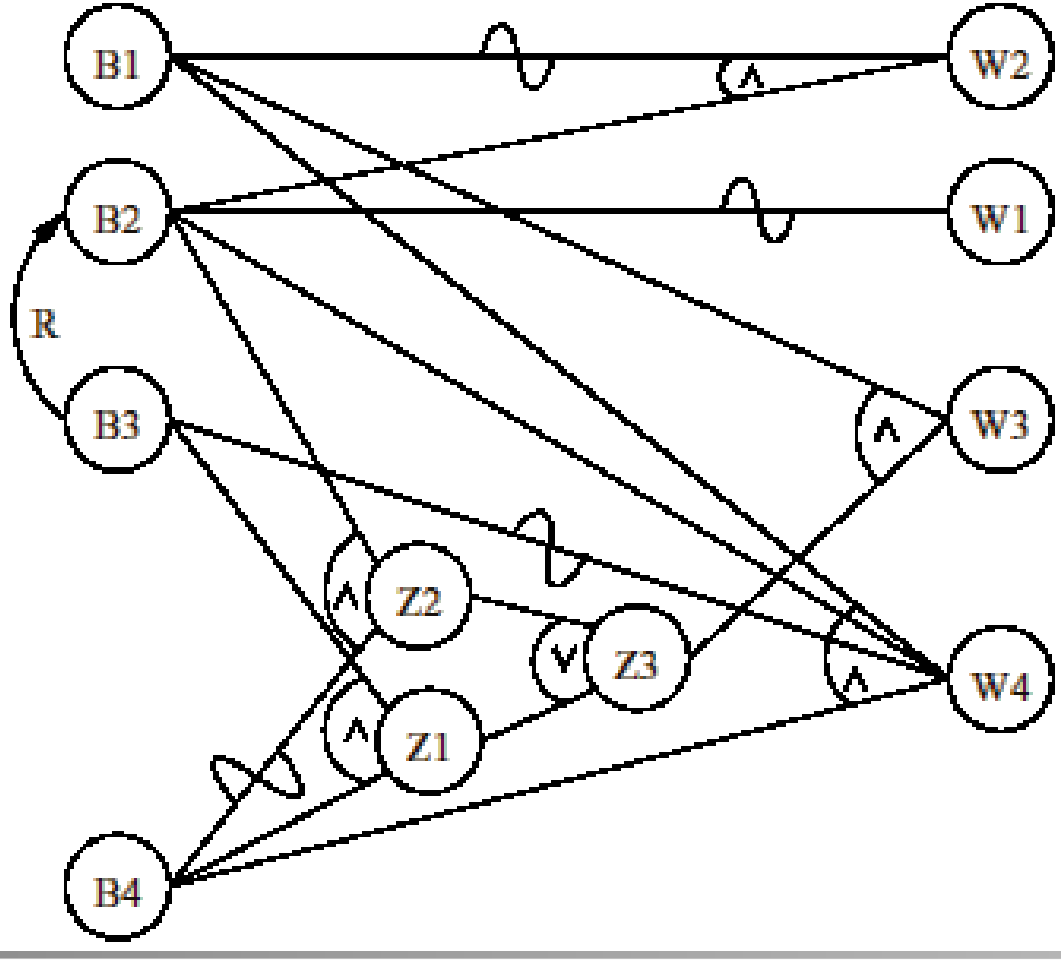
Wirkungen:

- W1: Abbruch, da Wort zu lang.
- W2: Wort im Puffer aufbauen.
- W3: Wort in aktuelle Ausgabezeile einfügen.
- W4: Wort in neue Ausgabezeile einfügen.

Schritt 3 (Logische Beziehungen zwi. Ursachen und Wirkungen darstellen):

- W1 = ¬B2
- W2 = ¬B1 ^ B2
- W3 = B1 ^ [(B3 ^ B4) ˇ (B2 ^ ¬B4)]
- W4 = B1 ^ B2 ^ ¬B3 ^ B4

Schritt 4 (UWG mit Einschränkungen versehen):



Ursache-Wirkungsgraphen
Beispiel 2 – Schritt 5

Schritt 5 (Aus UWG Wertetabelle ableiten):

- Nur Fehler bei Eingabewerten betrachten.

Testfälle pro betrachteter Wirkung W1 bis W4															
	W1		W2			W3 und W4									
	t_1	t_2	t_3	t_4	t_5	t_{6a}	t_{6b}	t_{7a}	t_{7b}	t_8	t_{9a}	t_{9b}	t_{10a}	t_{10b}	t_{11}
B1			0	0	1	0	0	1	1	1	1	1	1	1	0
B2	0	1	1	0	1	1	1	1	1	0	1	1	0	1	1
B3						-	1	-	1	0	0	1	0	0	0
B4						0	1	0	1	0	0	0	1	1	1
W1	1	0													
W2			1	0	0										
W3						0	0	1	1	0	1	1	0	0	
W4									0		0		0	1	0

- W1 = ¬B2
- Damit **mögliche Fehler bei B2** auffallen: B2 beide Wahrheitswerte annehmen.

Schritt 5 (Aus UWG Wertetabelle ableiten) ff.:

Testfälle pro betrachteter Wirkung W1 bis W4														
	W1		W2			W3 und W4								
	t_1	t_2	t_3	t_4	t_5	t_{6a}	t_{6b}	t_{7a}	t_{7b}	t_8	t_{9a}	t_{9b}	t_{10a}	t_{10b}	t_{11}
B1			0	0	1	0	0	1	1	1	1	1	1	1	0
B2	0	1	1	0	1	1	1	1	1	0	1	1	0	1	1
B3						-	1	-	1	0	0	1	0	0	0
B4						0	1	0	1	0	0	0	1	1	1
W1	1	0													
W2			1	0	0										
W3						0	0	1	1	0	1	1	0	0	
W4									0		0		0	1	0

- W2 = ¬B1 ^ B2
- Damit **mögliche Fehler bei B1** auffallen: B2=1 und B1 beide Wahrheitswerte annehmen.
- Damit **mögliche Fehler bei B2** auffallen: B1=0 und B2 beide Wahrheitswerte annehmen.

Schritt 5 (Aus UWG Wertetabelle ableiten) ff.:

- W3 = B1 ^ Z3, Z3 = Z1 ˇ Z2, Z1 = B3 ^ B4, Z2 = B2 ^ ¬B4

Jeweils 2 Alternativen

	$W3 = B1 \wedge Z3$		$Z3 = Z1 \vee Z2$		$Z1 = B3 \wedge B4$		$Z2 = B2 \wedge \neg B4$	
Testfall	$B1$	$Z3$	$Z1$	$Z2$	$B3$	$B4$	$B2$	$\neg\ B4$
t_{6a} $T_{1.0}$	0	1	0	1	-	0	1	1
t_{6b}			1	0	1	1	1	0
t_{7a} $T_{1.1}$	1	1	0	1	-	0	1	1
t_{7b}			1	0	1	1	1	0

- Damit **mögliche Fehler bei B1** auffallen: Z3=1.
- Kombinationen von Z1, Z2 und B2, B3, B4 entsprechend bilden.

Schritt 5 (Aus UWG Wertetabelle ableiten) ff.:

- W3 = B1 ^ Z3, Z3 = Z1 ˇ Z2, Z1 = B3 ^ B4, Z2 = B2 ^ ¬B4

2 Alternativen →

		$Z2 = B2 \wedge \neg B4$		obligatorisch		$Z1 = B3 \wedge B4$	
Testfall		$B2$	$\neg B4$	$Z1$	$B1$	$B3$	$B4$
t_8	$T_{2.0}$	0	1	0	1	0	0
t_{9a}	$T_{2.1}$	1	1	0	1	0	0
t_{9b}						1	0

- Damit **mögliche Fehler bei B2** auffallen: ¬B4=1.
- Z1=0 und B1=1, um **mögliche Fehler nicht zu maskieren**.

Schritt 5 (Aus UWG Wertetabelle ableiten) ff.:

- W3 = B1 ^ Z3, Z3 = Z1 ˇ Z2, Z1 = B3 ^ B4, Z2 = B2 ^ ¬B4

2 Alternativen ->

	$Z1 = B3 \wedge B4$		*obligatorisch*		$Z2 = B2 \wedge \neg B4$	
Testfall	*B3*	*B4*	*Z2*	*B1*	*B2*	*¬ B4*
t_{10a} $T_{3,0}$	*0*	*1*	*0*	*1*	*0*	*0*
t_{10b}					*1*	*0*
t_{7b} $T_{3,1}$	*1*	*1*	*0*	*1*	*1*	*0*

- Damit **mögliche Fehler bei B3** auffallen: B4=1.
- Z2=0 und B1=1, um mögliche Fehler nicht zu maskieren.

Schritt 5 (Aus UWG Wertetabelle ableiten) ff.:

- W3 = B1 ^ Z3, Z3 = Z1 ˇ Z2, Z1 = B3 ^ B4, Z2 = B2 ^ ¬B4

	$Z2 = B2 \wedge \neg B4$		*obligatorisch*		$Z1 = B3 \wedge B4$	
Testfall	$\neg B4$	*B2*	*Z1*	*B1*	*B3*	*B4*
t_{10b} $T_{4,0,Z2}$	*0*	*1*	*0*	*1*	*0*	*1*
t_{9a} $T_{4,1,Z2}$	*1*	*1*	*0*	*1*	*0*	*0*

- Damit **mögliche Fehler bei B4** auffallen: B2=1 bzw. B3=1.
- Z1=0 bzw. Z2=0 und B1=1, um mögliche Fehler nicht zu maskieren.

Ursache-Wirkungsgraphen
Beispiel 2 – Schritt 5

Schritt 5 (Aus UWG Wertetabelle ableiten) ff.:

- W4 = B1 ^ B2 ^ ¬B3 ^ B4

Bereits bekannte Testfälle

	Testfall	B1	B2	¬B3	B4	B3
t_{10b}	$T_{W4.1}$	1	1	1	1	0
t_{11}	$T_{W4.0.1}$	0	1	1	1	0
t_{10a}	$T_{W4.0.2}$	1	0	1	1	0
t_{7b}	$T_{W4.0.3}$	1	1	0	1	1
t_{9a}	$T_{W4.0.4}$	1	1	1	0	0

Neuer Testfall

- B1 ^ B2 ^ ¬B3 ^ B4 = 1 **entdeckt alle Fehler**, bei denen einer der Eingänge von 1 auf 0 wechselt.
- Vier Kombinationen, bei denen Eingang auf 0 und alle anderen aus 1 gesetzt werden, **entdecken Wechsel** von 0 nach 1.

Schritt 5 (Aus UWG Wertetabelle ableiten) ff.:

- Gesamte Wertetabelle im Überblick

Testfälle pro betrachteter Wirkung W1 bis W4														
	W1		W2			W3 und W4								
	t_1	t_2	t_3	t_4	t_5	t_{6a}	t_{6b}	t_{7a}	t_{7b}	t_8	t_{9a}	t_{9b}	t_{10a}	t_{10b}	t_{11}
B1			0	0	1	0	0	1	1	1	1	1	1	1	0
B2	0	1	1	0	1	1	1	1	1	0	1	1	0	1	1
B3						-	1	-	1	0	0	1	0	0	0
B4						0	1	0	1	0	0	0	1	1	1
W1	1	0													
W2			1	0	0										
W3						0	0	1	1	0	1	1	0	0	
W4									0		0		0	1	0

Ursache-Wirkungsgraphen
Beispiel 2 – Schritt 6

Schritt 6 (Kombinationen für verschiedene Ursachen, die sich nicht widersprechen, zusammenfassen):

Testfälle pro betrachteter Wirkung W1 bis W4														
	W1		W2			W3 und W4								
	t_1	t_2	t_3	t_4	t_5	t_{6a}	t_{6b}	t_{7a}	t_{7b}	t_8	t_{9a}	t_{9b}	t_{10a}	t_{10b}	t_{11}
B1			0	0	1	0	0	1	1	1	1	1	1	1	0
B2	0	1	1	0	1	1	1	1	1	0	1	1	0	1	1
B3						-	1	-	1	0	0	1	0	0	0
B4						0	1	0	1	0	0	0	1	1	1
W1	1	0													
W2			1	0	0										
W3						0	0	1	1	0	1	1	0	0	
W4									0		0		0	1	0

- Testfälle, die **durch andere Testfälle abgedeckt** sind und entfallen können
  - t1 abgedeckt durch t8.
  - t2 abgedeckt durch t7b.
  - t3 abgedeckt durch t11.
  - t5 abgedeckt durch t10b.

Schritt 6 (Kombinationen für verschiedene Ursachen, die sich nicht widersprechen, zusammenfassen) ff.:

Testfälle pro betrachteter Wirkung W1 bis W4															
	W1		W2			W3 und W4									
	t_1	t_2	t_3	t_4	t_5	t_{6a}	t_{6b}	t_{7a}	t_{7b}	t_8	t_{9a}	t_{9b}	t_{10a}	t_{10b}	t_{11}
B1			0	0	1	0	0	1	1	1	1	1	1	1	0
B2	0	1	1	0	1	1	1	1	1	0	1	1	0	1	1
B3						-	1	-	1	0	0	1	0	0	0
B4						0	1	0	1	0	0	0	1	1	1
W1	1	0													
W2			1	0	0										
W3						0	0	1	1	0	1	1	0	0	
W4									0		0		0	1	0

- Testfälle, die entfallen können, da **ihre Alternativen für W4 Musstestfälle** sind
  - t7a (t7a,t7b Alternativen für W3, aber T7b Muss für W4).
  - t9b (t9a,t9b Alternativen für W3, aber T9a Muss für W4).

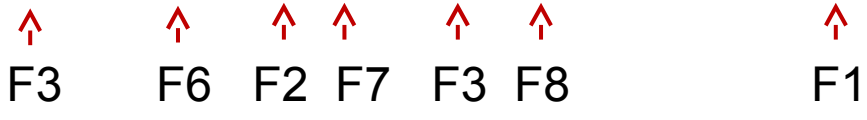
Schritt 6 (Kombinationen für verschiedene Ursachen, die sich nicht widersprechen, zusammenfassen) ff.:

- Optimierte Wertetabelle

Testfall	F1	F2	F3	F4	F5	F6	F7	F8
deckt ab	t_4	t_3, t_{11}	t_{6a}/t_{6b}	t_1, t_8	t_{10a}	t_{9a}	t_5, t_{10b}	t_2, t_{7b}
B1	0	0	0 0	1	1	1	1	1
B2	0	1	1 1	0	0	1	1	1
B3	0	0	- 1	0	0	0	0	1
B4	-	1	0 1	0	1	0	1	1
W1	1	0	0	1	1	0	0	0
W2	0	1	1	0	0	0	0	0
W3	0	0	0	0	0	1	0	1
W4	0	0	0	0	0	0	1	0

Schritt 7 (Für jede Spalte der Wertetabelle Eingabedatum ermitteln):

- _ sei das Blankzeichen und MAXPOS=9
- BEISPIELE_SIND_DIES_POTZBLITZ!



Testfälle T4, T5 nicht testbar, da vorher stets Abbruch durch F1 erfolgt.

- F3: Wort wird im Puffer aufgebaut.
- F6: Trennzeichen. → Wort in Ausgabezeile ausgeben.
- F2: Wort wird im Puffer aufgebaut.
- F7: Trennzeichen. → Wort in neue Ausgabezeile ausgeben.
- F3: Wort wird im Puffer aufgebaut.
- F8: Trennzeichen. → Wort in Ausgabezeile ausgeben.
- F1: Abbruch, da Wort zu lang.

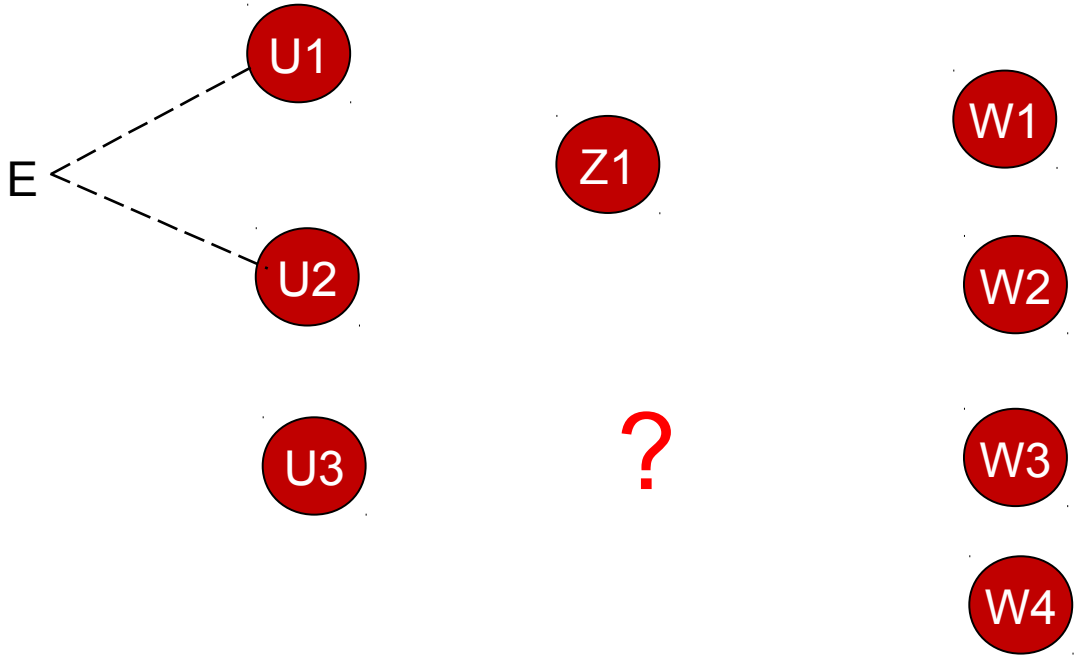
Ursache-Wirkungsgraphen
Beispiel 3

`zaehle(gesamtzahl, vokalAnzahl: int)`
- Liest solange Zeichen, bis Zeichen erkannt wird, das kein Großbuchstabe ist, oder `gesamtzahl` größten `int`-Wert erreicht hat.
- Ist gelesenes Zeichen großer Konsonant oder Vokal, `gesamtzahl` um 1 erhöhen.
- Ist Großbuchstabe Vokal, `vokalAnzahl` um 1 erhöhen.
- Bei Beendigung `gesamtzahl` und `vokalAnzahl` zurückgeben.

U1: Großer Konsonant
U2: Großer Vokal
U3: gesamtzahl < MaxCardinal
Z1: Zeichen gelesen

W1: Gesamtzahl erhöhen
W2: Vokalanzahl erhöhen
W3: Zeichen lesen
W4: Programmende

∧ logisches und
∨ logisches oder
¬ logisches nicht



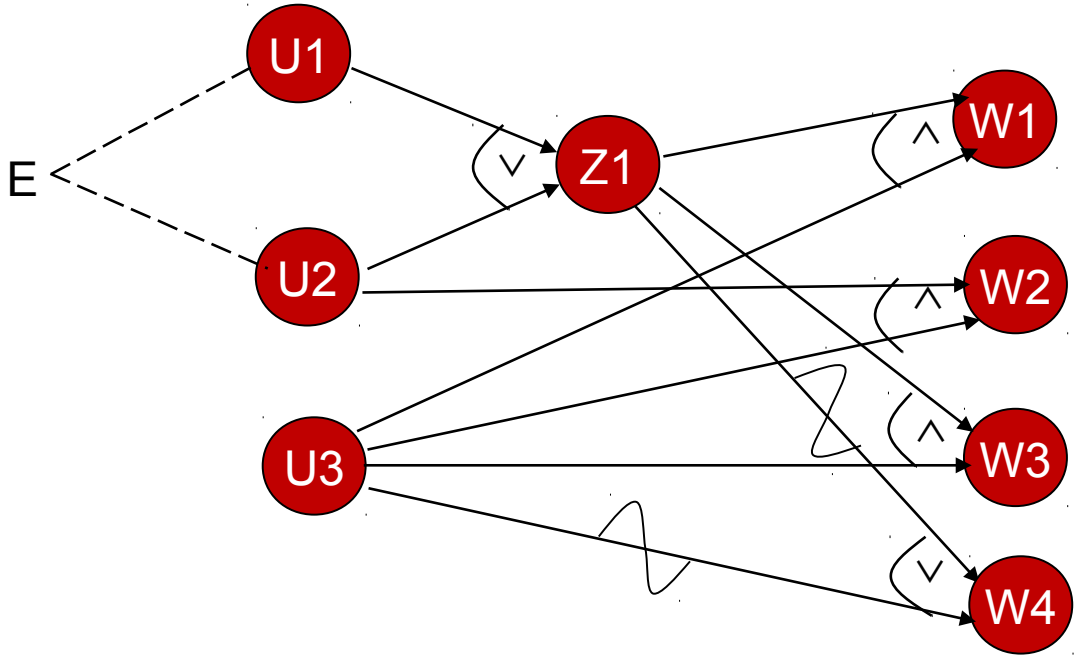
Ursache-Wirkungsgraphen
Beispiel 3

`zaehle(gesamtzahl, vokalAnzahl: int)`
- Liest solange Zeichen, bis Zeichen erkannt wird, das kein Großbuchstabe ist, oder `gesamtzahl` größten `int`-Wert erreicht hat.
- Ist gelesenes Zeichen großer Konsonant oder Vokal, `gesamtzahl` um 1 erhöhen.
- Ist Großbuchstabe Vokal, `vokalAnzahl` um 1 erhöhen.
- Bei Beendigung `gesamtzahl` und `vokalAnzahl` zurückgeben.

U1: Großer Konsonant
U2: Großer Vokal
U3: gesamtzahl < MaxCardinal
Z1: Zeichen gelesen

W1: Gesamtzahl erhöhen
W2: Vokalanzahl erhöhen
W3: Zeichen lesen
W4: Programmende

∧ logisches und
∨ logisches oder
~ logisches nicht



Vergleich: Grenzwertanalyse, Äquivalenzklassenbildung, UW-Graphen: Aufgedeckte Fehler

Aufgedeckte Fehler pro Testkriterium:

- **Angabe konkreter Zahlen** fragwürdig: Studien unterschiedlicher Art mit unterschiedlichen Ansätzen.
- Im Vergleich gilt:
  - **Äquivalenzklassenmethode besser** als Zufallstest.
    - (Nur) geringe Überlegenheit resultiert aus Problem, dass keine algorithmisch-konstruktiven Verfahren zum Erzeugen von vollständig homogenen Äquivalenzklassen gibt.
      - **Homogen:** Für jedes Eingabedatum einer Klasse tritt ein/kein Fehler auf.
  - **Grenzwertanalyse und UWG-Methode besser** als Äquivalenzklassenmethode.

  → Konzentration auf **fehlerträchtige Eingaben** (Grenzwerte, fehlersensitive Ursachenkombinationen).

Bewertung der Black-Box-Testentwurfsverfahren

Grundlagen: **Anforderungen** und **Spezifikation** des Systems und ihr Zusammenwirken.

- **Fehlerhafte Anforderungen** oder **Spezifikationen nicht erkannt**.
- Testobjekt verhält sich nach Forderung der Spezifikation, auch falls fehlerhaft.

Nicht geforderte Funktionalität nicht erkannt:

- **Zusätzliche Funktionen nicht spezifiziert**.
- Testfälle, die diese zusätzlichen Funktionen zur Ausführung bringen, werden nur zufällig durchgeführt.
- Überdeckungskriterien auf Grundlage der Anforderungen.
- **Mutationstesten** kann hier Abhilfe schaffen.

Im Mittelpunkt: **Prüfung der Funktionalität** des Testobjektes.

Korrektes Funktionieren eines Softwaresystems hat höchste Priorität.
→ Black-Box-Testentwurfsverfahren einsetzen.

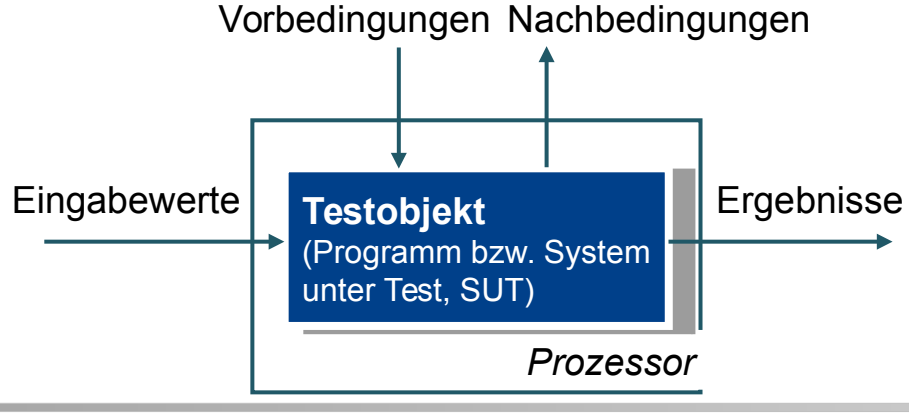
Zusammenfassung

- Dynamische Tests führen Testobjekt aus.
- Spezieller Testrahmen oft notwendig.
- Auswahl der Testfälle als (gute) Stichprobe.
- Black-Box-Testentwurfsverfahren benötigen zur Auswahl der Testfälle keine Kenntnis der Programmlogik.
- Funktionale Tests leiten Testfälle anhand Spezifikation des Testobjekts ab.
- Äquivalenzklassenbildung in Kombination mit Grenzwertanalyse zur Erstellung der Testfälle einsetzen.
- Zustandsbasierte Tests mit Übergangsbaum und Zustands-Konformanztest sowie erweitertem Übergangsbaum und Zustands-Robustheitstest.
- Entscheidungstabellentest bei regelbasierten Anforderungen.

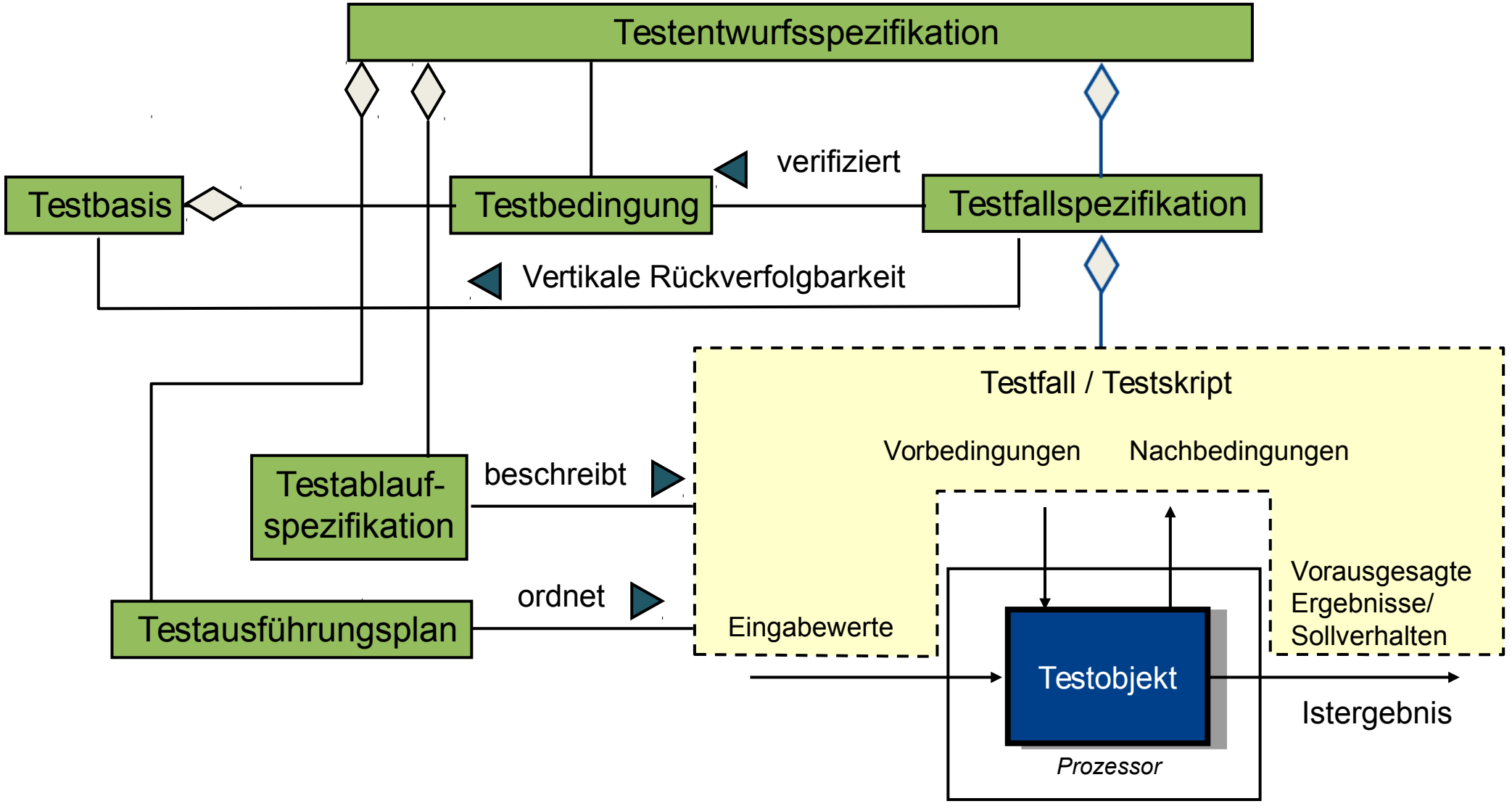
Anhang: weitere Informationen und Beispiele zum Nacharbeiten

Statischer Test vs. dynamischer Test

- **Programme:** Statische Beschreibungen von dynamischen Prozessen.
- **Statische Tests** prüfen Testobjekte:
  - Artefakte des Entwicklungsprozesses, z.B. informelle Texte, Modelle, formale Texte, Programmcode, ...
- **Dynamische Tests** prüfen die durch »Interpretation« einer Beschreibung resultierenden Prozesse.
- Ausführung des Testobjekts im dynamischen Test, also auf einem Prozessor:
  - Bereitstellen von Eingabewerten.
  - Beobachten der Ergebnisse.



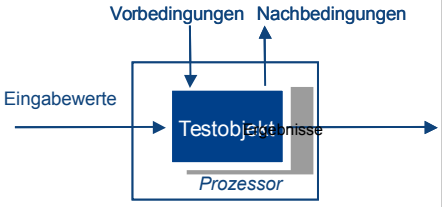
Dynamischer Test
Zusammenhänge



Dynamischer Test
Vorgehensweise

1. **Entwerfen von Tests durch Ermittlung von Testbedingungen:**
   - Analyse der dem Test zugrunde liegenden Dokumente, um Testbedingungen festzulegen.
   - Testbedingungen auf Anforderungen zurückverfolgbar, um Auswirkungen von Änderungen an Anforderungen auf Test und Abdeckung von Anforderungen durch Tests ermitteln.
2. **Spezifizieren der Testfälle:**
   - Entwicklung und detaillierte Beschreibung der Testfälle unter Verwendung von Testentwurfsverfahren.
   - Vorausgesagte Ergebnisse beinhalten Ausgaben, Änderungen von Daten und Zuständen sowie alle anderen Folgen des Tests – falls nicht definierbar, plausibles, aber fehlerhaftes Ergebnis als richtig ansehbar. → Vor Testdurchführung festlegen !
3. **Spezifizieren der Testablaufspezifikationen:**
   - Testfälle in ausführbare Reihenfolge bringen. Vor- und Nachbedingungen beachten.
4. **Testausführungsplanung**
   - Reihenfolge angeben,wann und von wem verschiedene Testablaufspezifikationen ausgeführt werden.
   - Regressionstests, Priorisierung und logische Abhängigkeiten zwi. Testfällen berücksichtigen.

Dynamischer Test in »unteren« Teststufen



Ziele:

- Stichprobenhafte Programmläufe: Nachweis Erfüllung festgelegter Anforderungen durch Testobjekt.
- **Aufdeckung von** eventuellen Abweichungen und **Fehlerwirkungen**.

  → Mit wenig Aufwand viele Fehler nachweisen.

Ausführung dynamischer Tests erfordert **Ausführbarkeit des Testobjekts**.

- Testdurchführung: Vorlage ablauffähigen Programms!

Einzelne Klassen, Komponenten und Teilsysteme **nicht alleine lauffähig:**

- Dienste für andere Softwareeinheiten.
- Kein »Hauptprogramm« zur Koordination des Ablaufes.
- Abstützen auf Dienste anderer Softwareeinheiten.

Klassen-/Modultest, Komponententest und Integrationstest: **Einbettung des Testobjekts** in entsprechenden »Testrahmen«.

Dynamischer Test – Begriffe (1)

Dynamischer Test: Prüfung des Testobjekts durch Ausführung auf einem Rechner.

Testbasis: Alle Dokumente, aus denen die Anforderungen ersichtlich werden, die an Komponente / System gestellt werden (bzw. Dokumentation für Herleitung / Auswahl der Testfälle).

Testbedingung: Einheit oder Ereignis (z.B. Funktion, Transaktion), Feature, Qualitätsmerkmal oder strukturelles Element einer Komponente oder eines Systems, welche bzw. welches durch einen oder mehrere Testfälle verifiziert werden kann.

Testentwurfsspezifikation: Ergebnisdokument, das Testbedingungen für Testobjekt, detaillierte Testvorgehensweise und zugeordnete logische Testfälle identifiziert [nach IEEE 829].

Testfallspezifikation: Dokument, das eine Menge von Testfällen für ein Testobjekt spezifiziert (inkl. Testdaten und Vor-/Nachbedingung), bei dem Testfälle jeweils Ziele, Eingaben, Testaktionen, vorausgesagte Ergebnisse und Vorbedingungen für Ausführung enthalten [nach IEEE 829].

Testsuite: Zusammenstellung mehrerer Testfälle für Test einer Komponente, bei der Nachbedingungen eines Tests als Vorbedingungen des folgenden Tests genutzt werden.

Dynamischer Test – Begriffe (2)

Testfall: Umfasst folgende Angaben:
• die für die Ausführung notwendigen **Vorbedingungen**,
• die Menge der **Eingabewerte** (ein Eingabewert je Parameter des Testobjekts),
• die Menge der **vorausgesagten Ergebnisse**, sowie
• die **erwarteten Nachbedingungen**.
Testfälle werden entwickelt im Hinblick auf bestimmtes Ziel bzw. auf Testbedingung, wie z.B. bestimm-ten Programmpfad auszuführen oder Übereinstimmung mit spezifischen Anforderungen zu prüfen (wie Eingaben an das Testobjekt zu übergeben und Sollwerte abzulesen sind) [nach IEEE 610].

Vorbedingung: Bedingungen an Zustand des Testobjekts und seiner Umgebung, die vor Durchführung eines Testfalls oder Testablaufs erfüllt sein müssen.

Nachbedingung: Zustand des Testobjekts, in dem sich Testobjekt nach Ausführung eines Testfalls/ Testlaufs befinden muss.

Testablaufspezifikation: Dokument, das eine Folge von Schritten zur Testausführung festlegt. Bekannt als Testskript oder Testdrehbuch [nach IEEE 829].

Testskript: Bezeichnet Testablaufspezifikation, insbesondere eine automatisierte.

Dynamischer Test – Begriffe (3)

Testausführungsplan: Plan für Ausführung von Testskripten. Testskripte sind im Testausführungsplan mit ihrem Kontext und in auszuführendeR Reihenfolge festgelegt.

Testlauf: Ausführung eines / mehrerer Testfälle mit bestimmter Version des Testobjekts.

Rückverfolgbarkeit: Fähigkeit, zusammengehörige Teile von Dokumentation und Software zu identifizieren, insbesondere Anforderungen mit dazu gehörigen Testfällen.
Vertikale Rückverfolgbarkeit: Rückverfolgung von Anforderungen durch Ebenen der Entwicklungsdokumentation bis zu Komponenten.
Horizontale Rückverfolgbarkeit: Verfolgen von Anforderungen einer Teststufe über Ebenen der Testdokumentation (z.B. Testkonzept, Testentwurfsspezifikation, Testfallspezifikation, Testablaufspezifikation oder Testskripte).

Rückverfolgbarkeit erlaubt sowohl Analyse der Auswirkungen (impact analysis) aufgrund von geänderten Anforderungen, als auch Bestimmung einer Anforderungsüberdeckung, bezogen auf bestimmte Menge von Tests.

Dynamischer Test – Begriffe (4)

LEHRSTUHL 14
SOFTWARE ENGINEERING

Testrahmen: Testumgebung, die aus für die Testausführung benötigten Treibern und Platzhaltern besteht.

Testumgebung: Umgebung, die benötigt wird, um Tests auszuführen. Umfasst Hardware, Instru-mentierung, Simulatoren, Softwarewerkzeuge u.a. unterstützende Hilfsmittel [nach IEEE 610].

Platzhalter: Spezielle Implementierung der Softwarekomponente, um nicht implementierte Komponente zu simulieren [nach IEEE 610].

Dummy: Platzhalter mit rudimentärer Funktionalität.

Mock: Platzhalter mit erweiterter Funktionalität für Testzwecke.

Simulator: Werkzeug, mit dem Produktivumgebung nachgebildet wird.

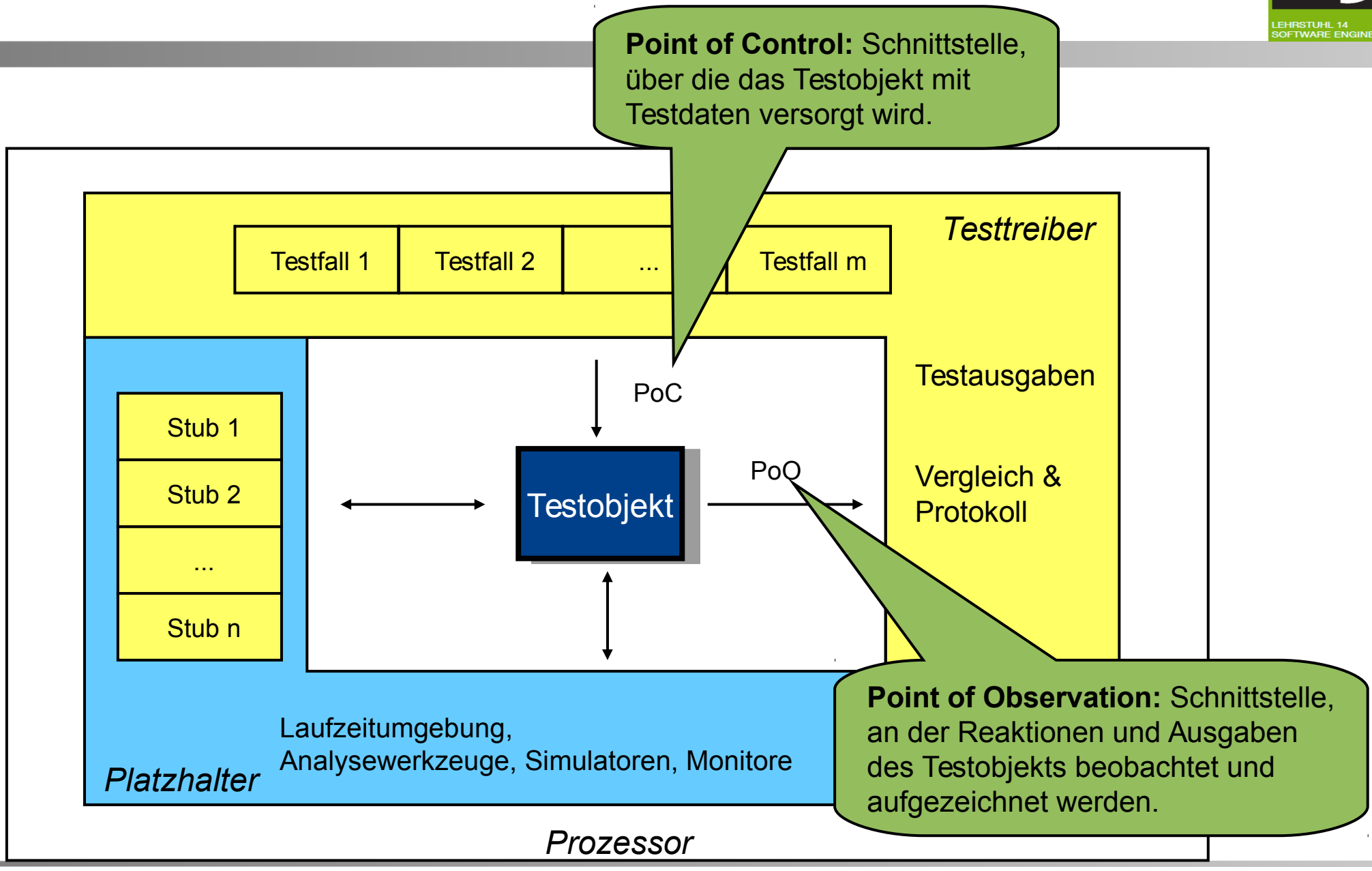
Testentwurfsverfahren: Vorgehensweise, nach der Testfälle abgeleitet oder ausgewählt werden.

Eingangskriterien: Menge generischer und spezifischer Bedingungen, die es in Prozess ermöglichen, mit bestimmter Aktivität fortzuschreiten. Zweck: Durchführung der Aktivität verhindern, wenn dafür Mehraufwand benötigt, als für Schaffung der Eingangskriterien [Gilb und Graham].

Ausgangskriterien: Menge abgestimmter generischer und spezifischer Bedingungen, die von allen Beteiligten für Abschluss des Prozesses akzeptiert wurden. Ausgangsbedingungen für eine Aktivität verhindern die Aktivität als abgeschlossen zu betrachtet, obwohl Teile noch nicht fertig sind [nach Gilb und Graham].

Ausgangskriterien in Berichten referenzieren und zur Planung der Beendigung des Testens verwenden.

Aufbau eines Testrahmens



Black-Box-Testentwurfsverfahren
(Testobjekt »undurchsichtig«)

Synonym: **Spezifikationsorientierte Testentwurfsverfahren.**

Keine Informationen über Programmtext und innerer Aufbau.

- Verhalten des Testobjekts von außen beobachten (PoO - Point of Observation liegt außerhalb des Testobjekts).
- **Steuerung des Ablaufs** des Testobjektes durch Wahl der Eingabetestdaten (PoC - Point of Control liegt außerhalb des Testobjektes).
- **Anforderungen an Testobjekt** (formal/nicht formal) zur Spezifikation des zu lösenden Problems heranziehen.
- Ableitung **systematischer Testfälle** durch diese Modelle.

White-Box-Testentwurfsverfahren
(Testobjekt »durchsichtig«)

Synonym: **Strukturorientierte Testentwurfsverfahren.**

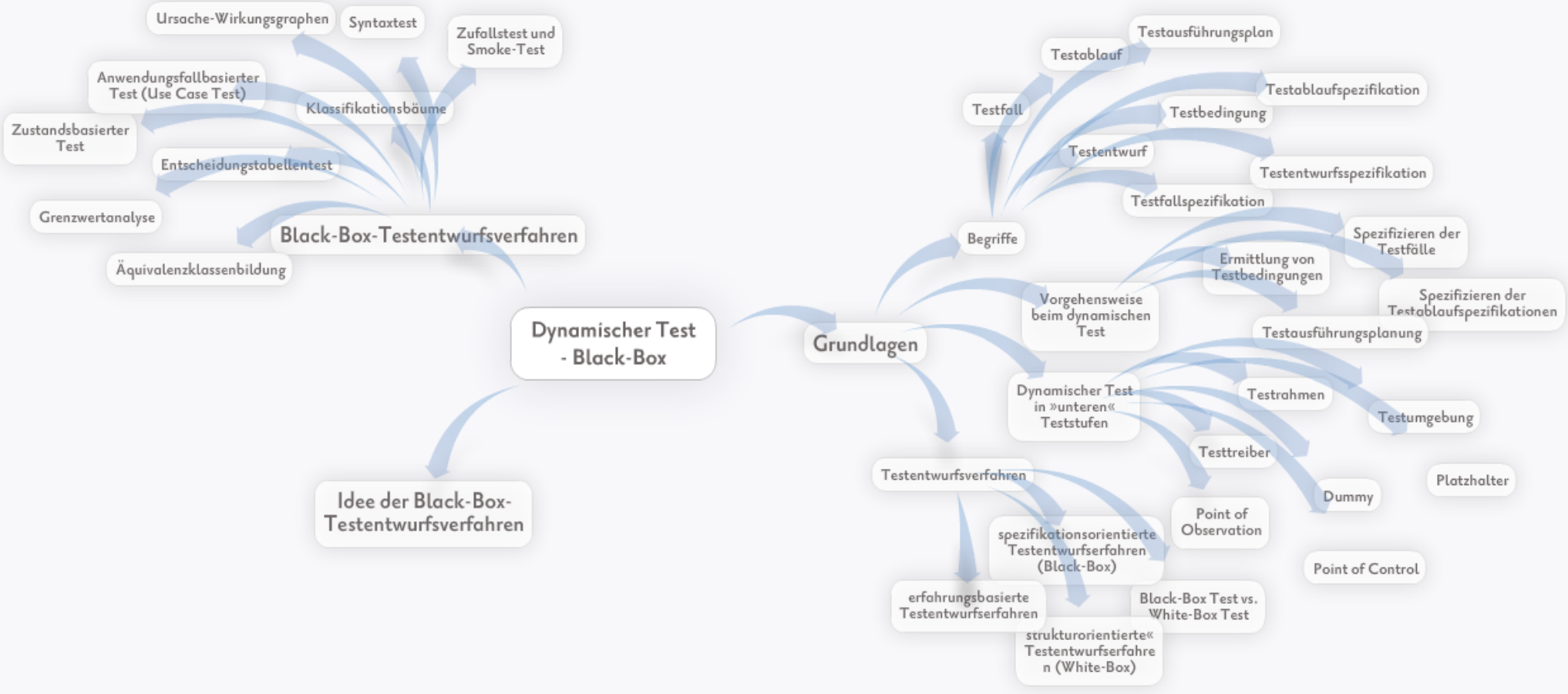
Testfälle auf Grund Programmstruktur des Testobjekts erhaltbar.

- Verwendung der **Informationen** über Aufbau der Software für Ableitung von Testfällen.
- Überdeckungsgrad des Codes für vorhandene Testfälle messbar.
- Weitere Testfälle zur **Erhöhung des Überdeckungsgrades** systematisch ableitbar.
- **Analyse des inneren Ablaufs** im Testobjekt während Ausführung der Testfälle.
- Negativtests, die zu provozierende Fehlbedienung über Komponentenschnittstelle führen, nicht auslösbar.
  → **Eingriff im Ablauf** im Testobjekt.

Erfahrungsbasierte Testentwurfserfahren

- **Ableitung der Testfälle:** Nutzen von Wissen und Erfahrung von Menschen.
- **Wissen von Testern**, Entwicklern, Anwendern und Betroffenen über Software, ihre Verwendung und ihre Umgebung.
- Wissen über **wahrscheinliche Fehler** und ihre Verteilung.

Dynamischer Test - Black-Box



Äquivalenzklassenbildung
Beispiel Wertebereich

Beschränkung spezifiziert **einen Wertebereich**: Eine gültige und zwei ungültige ÄK.

Beispiel

In der **Spezifikation** des Testobjekts ist festgelegt, dass ganzzahlige Eingabewerte zwischen 1 und 100 möglich sind.

Wertebereich der Eingabe	$1 \leq x \leq 100$
Gültige Äquivalenzklasse	$1 \leq x \leq 100$
Ungültige Äquivalenzklasse	$x < 1$, $x>100$ und NaN (not a Number)

Äquivalenzklassenbildung
Beispiel minimal / maximal

Beschränkung spezifiziert **minimale und maximale** Anzahl von Werten: Eine gültige und zwei ungültige ÄK.

Beispiel

Laut **Spezifikation** muss sich ein Mitglied eines Sportvereins mindestens einer Sportart zuordnen. Jedes Mitglied kann an maximal drei Sportarten aktiv teilnehmen.

Gültige Äquivalenzklasse $1 \leq x \leq 3$ (1 bis 3 Sportarten)

Ungültige Äquivalenzklasse $x=0$ und $x > 3$
(keine bzw. mehr als 3 Sportarten zugeordnet)

Äquivalenzklassenbildung
Beispiel Menge von Werten

Beschränkung spezifiziert **Menge von Werten**, die unterschiedlich behandelt werden:
Für jeden Wert dieser Menge eigene gültige ÄK und zusätzlich eine ungültige ÄK.

Beispiel

Laut **Spezifikation** gibt es im Sportverein folgende Sportarten: Fußball, Hockey, Handball, Basketball und Volleyball

- **Gültige** Äquivalenzklasse: Fußball, Hockey, Handball, Basketball und Volleyball
- **Ungültige** Äquivalenzklasse: alles andere z.B Badminton

Äquivalenzklassen für ggT

```
public int ggT(int m, int n)
```

Eingabeparameter m: `int`

gÄK1_1	min_int ≤ m < 0
gÄK1_2	m = 0
gÄK1_3	0 < m ≤ max_int
uÄK1_1	m < min_int
uÄK1_2	m > max_int

Eingabeparameter n: `int`

gÄK2_1	min_int ≤ n < 0
gÄK2_2	n = 0
gÄK2_3	0 < n ≤ max_int
uÄK2_1	n < min_int
uÄK2_2	n > max_int

Rückgabewert ggT: `int`

gÄK3_1	1 < ggT ≤ max_int
gÄK3_2	ggT = 1
uÄK3_1	ggT = 0
uÄK3_2	ggT < 0
uÄK3_3	ggT > max_int

Diskussionsfrage

- Welche Aussagen sind unten angegebenen Bereichen einer Grenzwertanalyse zuzuordnen?

Atomare Bereiche

- Werte auf den Grenzen
- Werte rechts bzw. links neben den Grenzen

Mengenwertige Bereiche

- Kleinste und größte gültige Anzahl
- Zweitkleinste und -größte gültige Anzahl
- Kleinste und größte ungültige Anzahl

Diskussionsfrage

- Wie ist die Methode der **Grenzwertanalyse zu bewerten?**
- Antwort:
  - Grenzwertanalyse **in Verbindung mit Äquivalenzklassenbildung** durchführen.
    → An Grenzen der Äquivalenzklassen häufiger Fehlerwirkungen nachzuweisen als innerhalb der Klassen.

Diskussionsfrage

- Was sind die **Vor- und Nachteile** des zustandsbasierten Tests?
- **Antwort:**

  Vorteile:

  - Beim objektorientierten Testen hat zustandsbasierter Test **herausgehobene Bedeutung.**
  - Prüfung auf **Vollständigkeit.**

  Nachteile:

  - **Überprüfung** einzelner Testfälle sehr **aufwändig.**
  - Keine Berücksichtigung anderer vorgestellter Testentwurfsverfahren.

Diskussionsfrage

- Wie ist die Methode des **Entscheidungstabellen-basierten Testens** zu bewerten ?
- **Antwort:**
  - Durch systematisches und formales Vorgehen beim Erstellen der Entscheidungstabelle mit allen möglichen Kombinationen können **Kombinationen auftreten**, die andere Methoden zur Testfallermittlung **nicht berücksichtigen** oder bisher **übersehen wurden.**

Diskussionsfrage

- Wie ist die **Ursache-Wirkungs** Methode zu **bewerten**?
- **Antwort:**
  - Durch systematisches Vorgehen beim Erstellen der Entscheidungstabelle mit allen möglichen Kombinationen können **übersehene Kombinationen auftreten.**
  - Beim Erstellen optimierter Entscheidungstabelle können **Fehler auftreten,** wenn zu berücksichtigende Kombinationen nicht in die Tabelle übernommen werden.
  - Tabelle kann schnell an **Umfang zunehmen** und an **Übersichtlichkeit verlieren.**

Vergleich: Grenzwertanalyse, Äquivalenzklassenbildung, UW-Graphen: Anzahl Testfälle

Anzahl Testdaten pro Testkriterium: **Äquivalenzklassen.**

- TGÄ ≥ max{Anzahl gültiger Klassen für Eingabebedingung b|b ∈ EB}.
- TGÄ ≤ Anzahl aller gültiger Klassen:
  - TGÄ Testdaten für gültige Äquivalenzklassen; EB Menge aller Eingabebedingungen b.
- TUÄ = KUÄ.
- TÄ = TGÄ + TUÄ.
  - TUÄ Testdaten für ungültige Äquivalenzklassen; KUÄ Anzahl aller ungültigen Äquivalenzklassen; TÄ Testdaten für alle Äquivalenzklassen.
- 1 ≤ TGÄ ≤ be:
  - Gilt, falls jeweils **nur eine gültige Klasse** pro Eingabebedingung gibt, bei Anzahl der Eingabebedingungen.

Vergleich: Grenzwertanalyse, Äquivalenzklassenbildung, UW-Graphen: Anzahl Testfälle

Anzahl der Testdaten pro Testkriterium: **Grenzwertanalyse.**

$$\sum_{b \in B} g(b) \leq TG \leq \prod_{b \in B} g(b) = \prod_{b \in EB} g(b) * \prod_{b \in AB} g(b)$$

- EB Menge der Eingabebedingungen; AB Menge der Ausgabebedingungen; B = EB u AB Menge aller Bedingungen; TG Anzahl der Tests, wenn alle möglichen Kombinationen von Grenzwerten getestet werden; g(B) Anzahl Grenzwerte pro Eingabe- oder Ausgabebedingung.

$$TG \geq \sum_{b \in B} g(b) \geq \sum_{b \in EB} g(b) \geq KEÄ \geq TGÄ + TUÄ = TÄ$$

- Folgt aus Überlegung, dass für geordnete Wertebereiche mindestens ein Grenzwert pro Äquivalenzklasse existiert, KEÄ Anzahl aller Eingabeäquivalenzklassen

  → Grenzwertanalyse erfordert min. so viele Tests wie Äquivalenzklassenbildung.

Vergleich: Grenzwertanalyse, Äquivalenzklassenbildung, UW-Graphen: Anzahl Testfälle

Anzahl Testdaten pro Testkriterium: **Ursache-Wirkungs-Graph:**

$$TUWG = \frac{TG}{4} + 1 \qquad TUWG = \frac{TÄ}{2} + \frac{1}{2}$$

- TUWG Tests bei UWG-Methode; TG Tests bei Grenzwertanalyse; TÄ Tests bei Äquivalenzklassenbildung.
- Annahmen s. [Rie97]

  → UWG-Methode benötigt nur **Hälfte** bzw. **ein Viertel der Tests**.

Nach diesem Abschnitt sollten Sie...

- Dynamischen Test charakterisieren und von anderen Testarten abgrenzen können.
- Funktionen eines Testrahmens für dynamischen Test erklären können.
- White-Box- und Black-Box-Testentwurfsverfahren zur Ermittlung von Testfällen charakterisieren und voneinander abgrenzen können.
- Grundidee der Black-Box-Testentwurfsverfahren erläutern können.
- Äquivalenzklassenbildung und Grenzwertanalyse kennen und auf einfache Beispiele anwenden können.
- Zustandsbasierten Test und Entscheidungstabellentest kennen und auf einfache Beispiele anwenden können.
- Anwendungsfallbasierten Test kennen und weitere Black-Box-Testentwurfsverfahren nennen können.

Folgende Fragen sollten Sie jetzt beantworten können

LEHRSTUHL 14
SOFTWARE ENGINEERING

- Welches ist wesentlicher Unterschied zwischen dynamischen und statischen Tests ?
- Welche grundlegenden Funktionen und Eigenschaften muss Testrahmen haben ?
- Welches ist wesentlicher Unterschied zwischen Black-Box-Testentwurfsverfahren und White-Box Verfahren zur Testfallermittlung ?
- Worauf basieren funktionale Tests ?
- Was versteht man unter Äquivalenzklassenbildung ?
- Wann ist Grenzwertanalyse einsetzbar ?
- Wann ist zustandsbasierter Test einsetzbar ?
- Wie ist Entscheidungstabelle aufgebaut ?
- Welche weiteren Black-Box-Testentwurfsverfahren gibt es noch ?

Diese Begriffe sollten Sie kennen ...