

- Motivation
- Testklassifikation
- ➔ ■ Black Box Testtechniken
- White Box Testtechniken
- Testmetriken
- Grenzen des Software Tests
- Testautomatisierung



Black Box Testverfahren

Die Verfahren werden auch als **spezifikationsbasierte Testentwurfungsverfahren** bezeichnet, da sie auf der Spezifikation (den Anforderungen) basieren.

Ein Test mit allen möglichen Eingabewerten und deren Kombination wäre ein vollständiger Test. Dies ist aber wegen der großen Zahl von möglichen Eingabewerten und Kombinationen unrealistisch. Eine sinnvolle Auswahl aus den möglichen Testfällen muss getroffen werden.



Warum nicht einfach alles durchtesten?


Bsp: Methode der Berechnung eines Absolutbetrags aus `java.lang.Math`:

```
public static long abs(long a)
```

Testen aller mögliche Eingabewerte:

Long Variable hat 64 Bit → 2^{64} Testfälle



- 
- Äquivalenzklassen Test
 - Grenzwertbetrachtung
 - Zustandsbasierter Software Test
 - Use Case Test
 - Entscheidungstabellen basierter Test
 - Paarweises Testen



Prinzip

1. Bilden von Äquivalenzklassen (Teilmengen der möglichen Eingabewerte, die intern eine identische Verarbeitung erwarten lassen.)
2. Testfallkonstruktion: Für jede Äquivalenzklasse wird ein Vertreter gewählt.
3. Für diesen Vertreter wird ein Testfall konstruiert.



Äquivalenzklassen: Teilmengen der möglichen Eingabewerte, die intern eine identische Verarbeitung erwarten lassen.

Bsp `public static long abs(long a)`

Äquivalenzklassen:

`[minLong, -1]` , `[0]` und `[1, maxLong]`

Im Fall einer Methode, die n Übergabeparameter entgegennimmt, sind n -dimensionale Äquivalenzklassen zu bilden.

➔ Vorgehen:

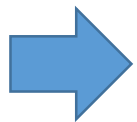
1. Äquivalenzklassenbildung für jeden Eingabeparameter suchen.
2. Zusammengehörige Äquivalenzklassen verschmelzen.

Mehrdimensionale Äquivalenzklassen

Bsp: Methode berechnet einen Preis anhand eines regulären Preises und einem Kundentyp, der den Rabatt regelt.

```
public enum CustomerType {  
    REGULAR,  
    SILVER,  
    PLATIN  
}
```

```
public static float calculatePrice(float regularPrice, CustomerType type)
```



Äquivalenzklassen?

Mehrdimensionale Äquivalenzklassen

```
public static float calculatePrice(float regularPrice, CustomerType type)
```

Äquivalenzklassen für
regularPrice:

- [floatmin,0[
- 0.f
-]0, floatmax]

Äquivalenzklassen für type:

- REGULAR
- SILVER
- PLATIN

Äquivalenzklassen

[floatmin,0[REGULAR	0.f REGULAR]0, floatmax] REGULAR
[floatmin,0[SILVER	0.f SILVER]0, floatmax] SILVER
[floatmin,0[PLATIN	0.f PLATIN]0, floatmax] PLATIN



Bsp. Zur Äquivalenzklassenbildung

Bsp aus: D. Hoffmann, Software-
Qualität

```
package aequivalenzklassen;

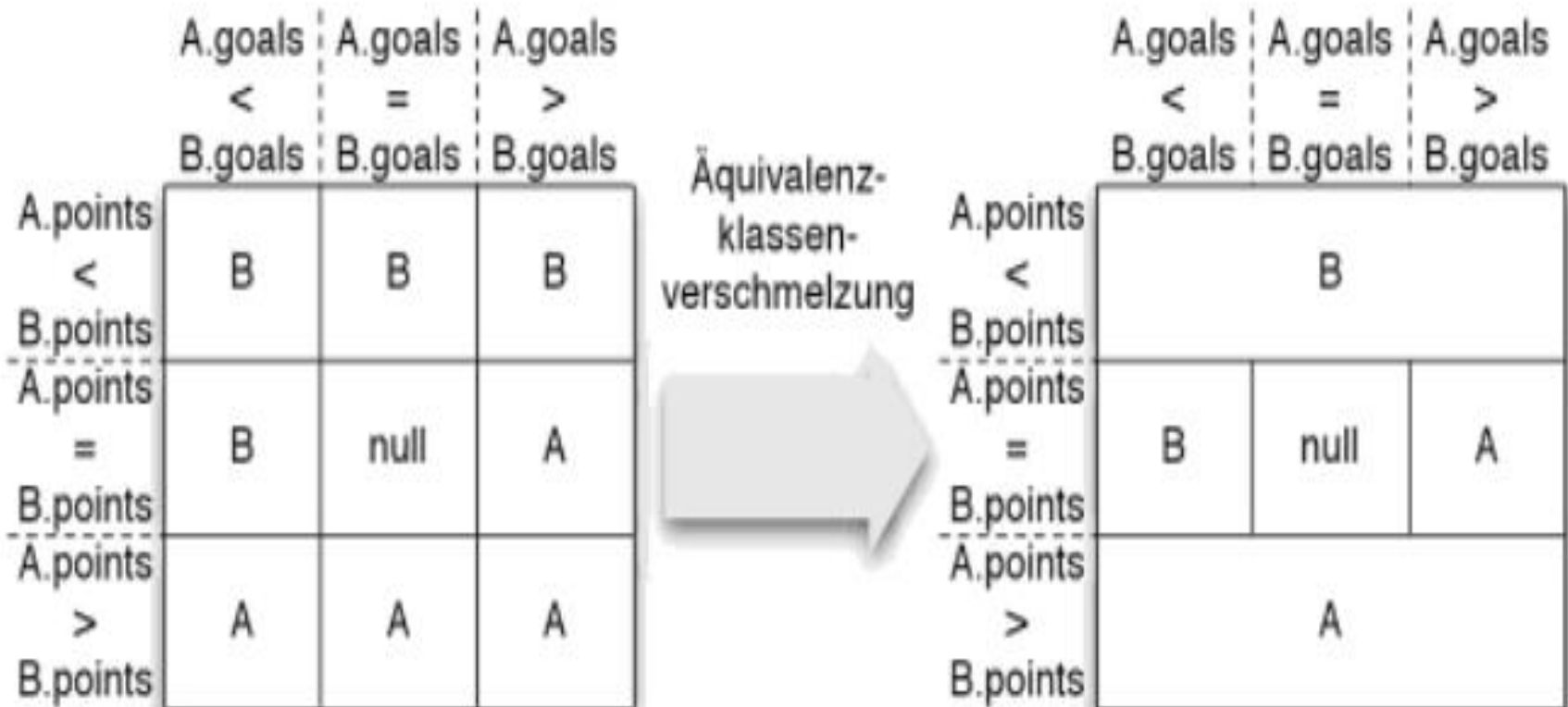
public class Team {

    int points;
    int goals;

    /**
     * Bestimmt den Sieger unter zwei Mannschaften.
     *
     * @param team1 Referenz auf das erste Team
     * @param team2 Referenz auf das zweite Team
     */
    public static Team calculateChampion(Team team1, Team team2){
        ...
    }
}
```



Bsp Äquivalenzklassen



Quelle der Abb: D. Hoffmann, Software-Qualität

Äquivalenzklassen für partiell definierte Funktionen

Was, wenn die erlaubten Eingabewerte eine Teilmenge der durch den Datentyp definierten Eingabewerte sind?

→ Zwei prinzipielle Vorgehensweisen:

- **Partielle Partitionierung**
Äquivalenzklassenbildung unter Ausschluss der ungültigen Eingabewerte.
- **Vollständige Partitionierung**
Äquivalenzklassenbildung bezieht die ungültigen Werte mit ein.

Beschreibung:

Über die Verkaufssoftware kann ein Autohaus seinen Verkäufern Rabattregeln vorgeben. In der Beschreibung der Anforderungen findet sich folgende Textpassage: „Bei einem Kaufpreis von weniger als 15.000 € soll kein Rabatt gewährt werden. Bei einem Preis bis zu 20.000 € sind 5 % Rabatt angemessen. Liegt der Kaufpreis unter 25.000 €, sind 7 % Rabatt möglich, darüber sind 8,5 % Rabatt einzuräumen.“

Bsp aus A. Spillner: Basiswissen Softwaretest




Äquivalenzklassen

Parameter	Äquivalenzklasse	Repräsentant
Verkaufspreis	gÄK1: $0 \leq x < 15000$ gÄK2: $15000 \leq x \leq 20000$ gÄK3: $20000 < x < 25000$ gÄK4: $x \geq 25000$	14500 16500 24750 31800

Äquivalenzklassen für ungültige Werte:

Parameter	Äquivalenzklasse	Repräsentant
Verkaufspreis	uÄK1: $x < 0$ (»negativer« – also falscher – Verkaufspreis) uÄK2: $x > 1000000$ (»unrealistisch hoher« Verkaufspreis ^a)	-4000 1500800

- Äquivalenzklassen Test
-  ▪ Grenzwertbetrachtung
- Zustandsbasierter Software Test
- Use Case Test
- Entscheidungstabellen basierter Test
- Paarweises Testen

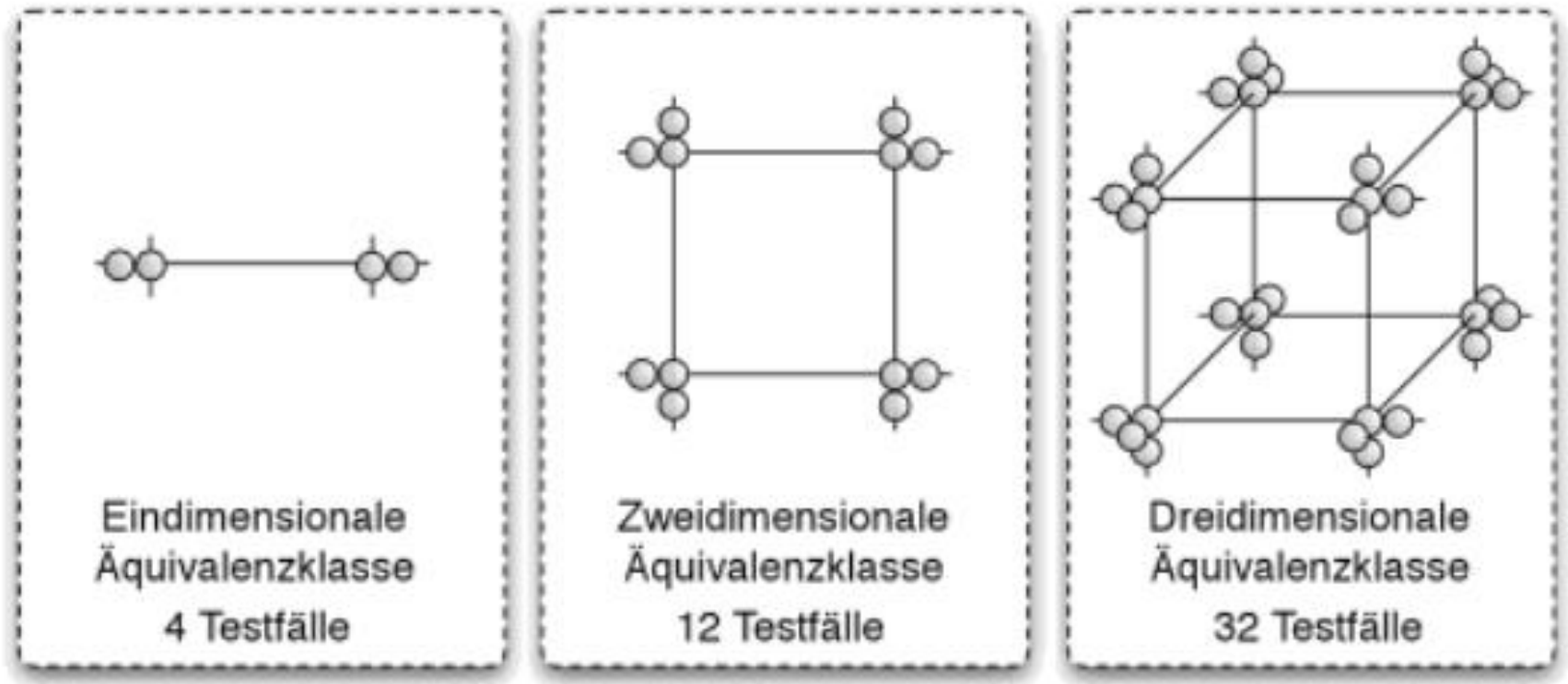


Grenzwertbetrachtung

- Partitionierung identisch wie bei der Äquivalenzklassenbildung
- Testfallauswahl aber nicht beliebig innerhalb einer Äquivalenzklasse, sondern jeweils den Randwert und Tupel, bei denen ein einzelner Wert außerhalb der Äquivalenzklasse liegt.



Grenzwertbetrachtung



Voraussetzung für diese Methode: Eingabewerte sind geordnet.

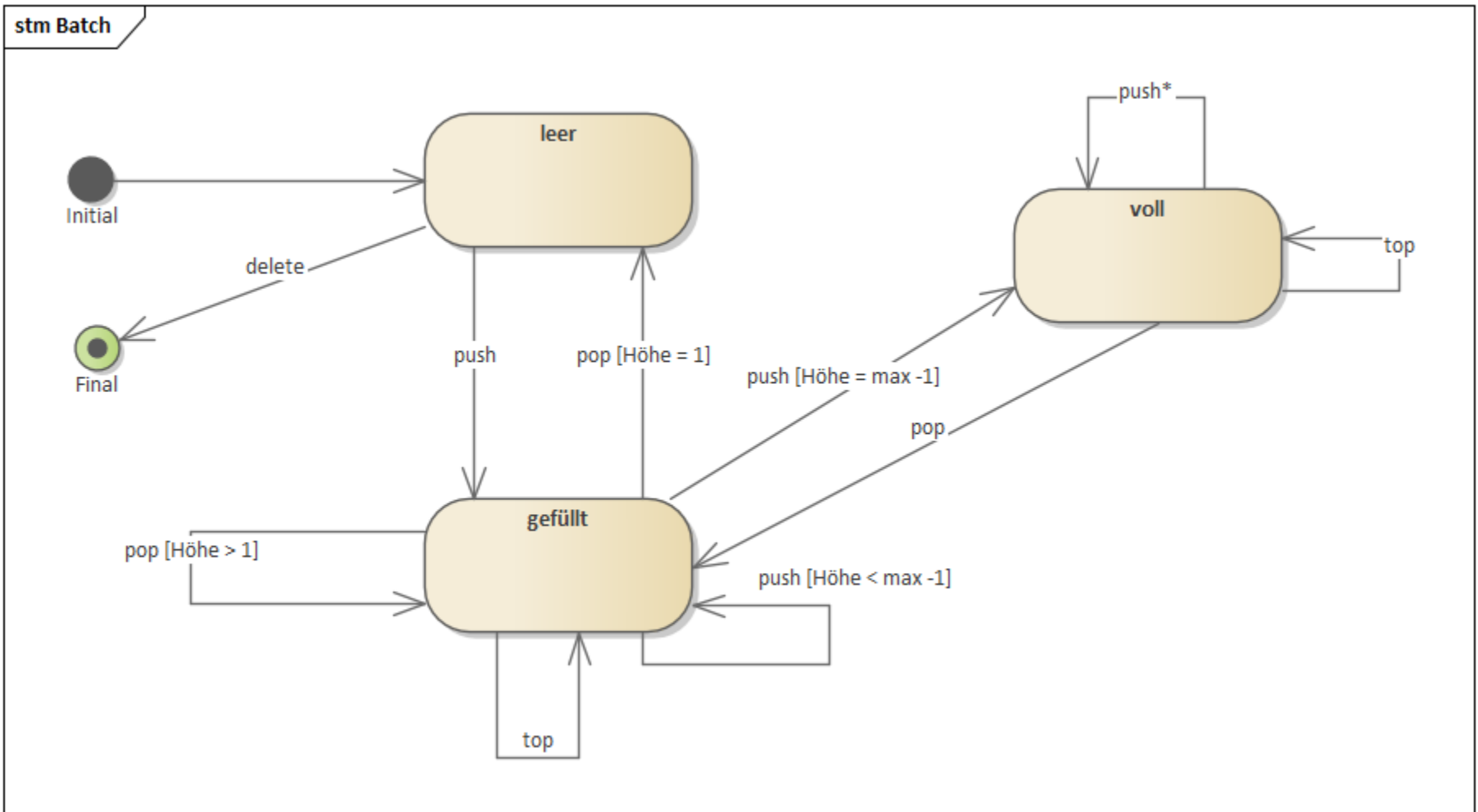
- Äquivalenzklassen Test
- Grenzwertbetrachtung
- ➔ ▪ Zustandsbasierter Software Test
- Use Case Test
- Entscheidungstabellen basierter Test
- Paarweises Testen



- **Bisher:** Ergebniswerte werden ausschließlich durch Eingabewerte bestimmt.
- **Jetzt:** Programmfunktionen mit Gedächtnis (also Zustand)
- **Idee:** Alle möglichen Übergänge zwischen zwei Zuständen werden mit mindestens einem Testfall geprüft.

Zustandsbehafteter Software Test – Bsp.

Bsp: Stapel (z.B. von Tellern) aus A. Spillner, T. Linz: Basiswissen

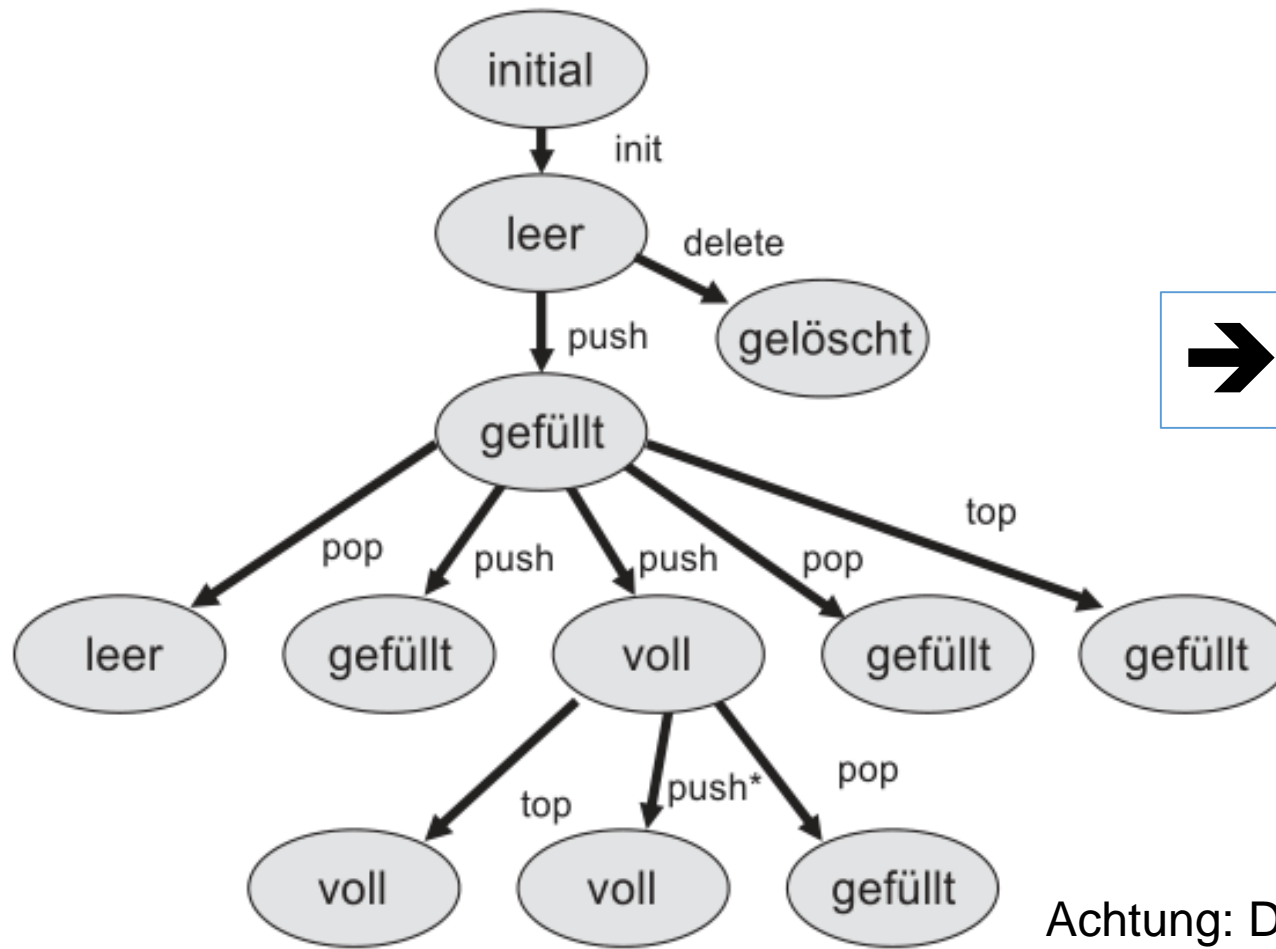


Ermittlung der Testfälle mittels Übergangsbaum

Bildung eines Übergangsbaums:

1. Der Anfangszustand ist die Wurzel des Baums.
2. Für jeden möglichen Übergang vom Anfangszustand zu einem Folgezustand im Zustandsdiagramm erhält der Übergangsbaum von der Wurzel aus eine Verzweigung zu einem Knoten, der den Nachfolgezustand repräsentiert.
3. Der letzte Schritt wird für jedes Blatt des Übergangsbaums solange wiederholt, bis eine der beiden Endbedingungen eintritt:
 - a. Der dem Blatt entsprechende Zustand ist auf dem Weg von der Wurzel zum Blatt bereits einmal im Baum enthalten. Diese Endbedingung entspricht einem Durchlauf von einem Zyklus im Zustandsdiagramm.
 - b. Der dem Blatt entsprechende Zustand ist ein Endzustand und hat somit keine weiteren Übergänge, die zu berücksichtigen wären.

Übergangsbaum für Stapelbeispiel

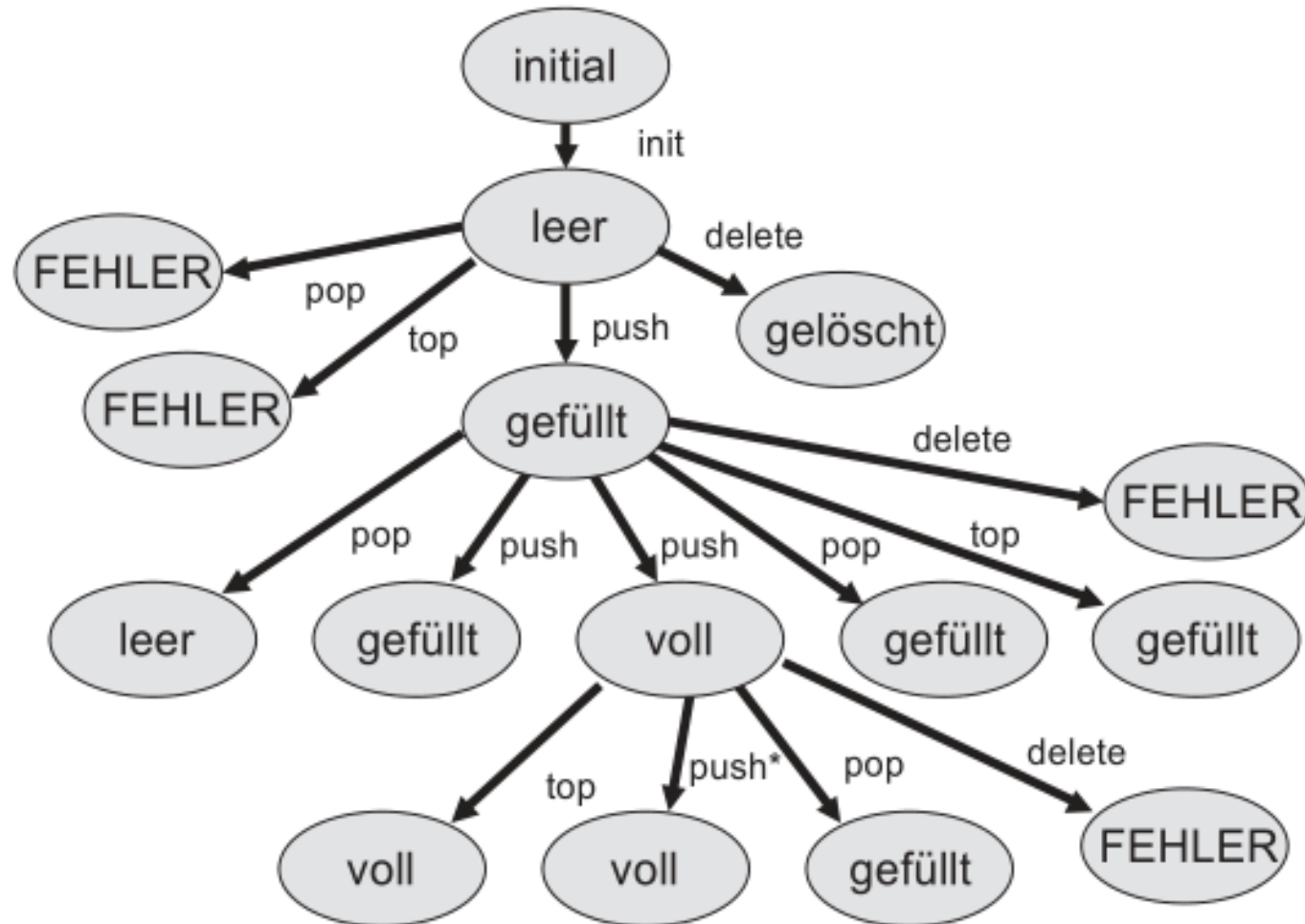


➔ 8 Testfälle

Achtung: Darstellung ohne guards!

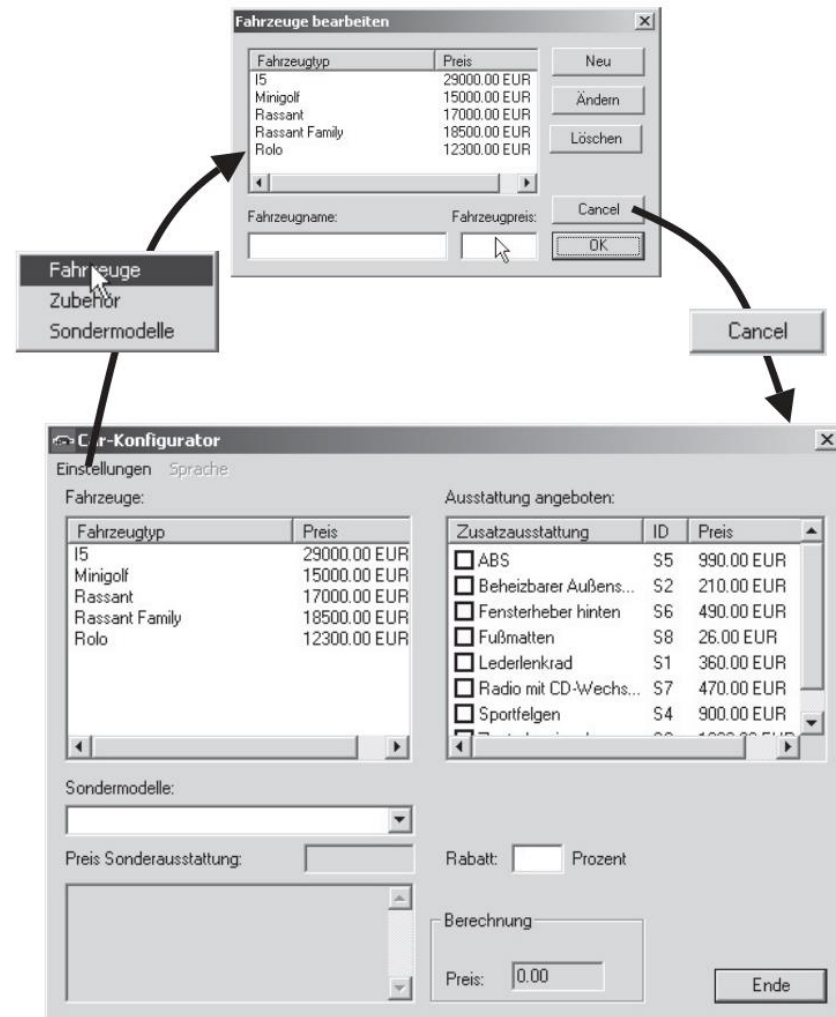


Übergangsbaum für Robustheitstest



Zustandsbasierter Test - Bsp

GUI Masken können als Zustände aufgefasst werden. Daher eignet sich der zustandsbasierte Test für die Testfallerstellung von GUI Abfolgen.



Quelle der Abb.; Spillner, Linz:
Basiswissen Softwaretest


- Äquivalenzklassen Test
- Grenzwertbetrachtung
- Zustandsbasierter Software Test
- ➔ ▪ Use Case Test
- Entscheidungstabellen basierter Test
- Paarweises Testen



Use Case Test

Speziell für Systemebene (Systemtest, Abnahmetest):

Usecase basierter Black Box Test:
Alle möglichen Szenarien eines Usecases werden durch mindestens einen Testfall abgedeckt.

- Äquivalenzklassen Test
- Grenzwertbetrachtung
- Zustandsbasierter Software Test
- Use Case Test
-  ▪ Entscheidungstabellen basierter Test
- Paarweises Testen



- **Verwendet, um Geschäftslogik zu testen (wenn <bedingungen> dann <Aktionen>)**
- **Prinzipielles Vorgehen:**
 - Alle möglichen Bedingungen identifizieren
 - Alle möglichen Reaktionen des Systems identifizieren.
 - Für jede Kombination von Bedingungen in der Tabelle eine Spalte für einen Testfall.
 - Für jeden Testfall definiere die Kombination aus Aktionen, die folgt.



Beispiel Geldautomat

Um Geld aus einem Automaten zu bekommen, sind folgende Bedingungen zu erfüllen :

- Die Bankkarte ist gültig.
- Die PIN ist korrekt eingegeben.
- Es dürfen nur maximal drei PIN-Eingaben erfolgen.
- Geld steht zur Verfügung (im Automat und auf dem Konto).

Als Aktion bzw. Reaktion des Geldautomaten sind folgende Möglichkeiten gegeben:

- Karte zurückweisen
- Aufforderung, erneut die PIN einzugeben
- Karte einbehalten
- Aufforderung, neuen Geldbetrag einzugeben
- Geldbetrag auszahlen



Entscheidungstabelle (komprimiert)

		TC1	TC2	TC3	TC4	TC5
Bedingungen	Bankkarte gültig	nein	Ja	ja	Ja	ja
	PIN korrekt	-	Nein	nein	Ja	ja
	3. PIN Eingabe	-	Nein	ja	-	-
	Geldverfügbar	-	-	-	Nein	ja
Aktionen	Karte zurückweisen	Ja	Nein	Nein	Nein	nein
	Pin erneut anfordern	Nein	Ja	Nein	Nein	nein
	Karte einbehalten	Nein	Nein	Ja	Nein	nein
	Gelbetrag erneut anfordern	Nein	Nein	Nein	Ja	nein
	Geld auszahlen	Nein	Nein	Nein	Nein	ja



- Äquivalenzklassen Test
- Grenzwertbetrachtung
- Zustandsbasierter Software Test
- Use Case Test
- Entscheidungstabellen basierter test
- ➔ ▪ Paarweises Testen



Paarweises Testen

- Testfallkonstruktion in der Absicht, die Anzahl der Fälle aus rein kombinatorischen Überlegungen zu reduzieren.
 - Annahme: Es müssen nicht alle möglichen, sondern nur alle paarweisen Kombinationen getestet werden, um alle Fehler zu finden.
- ➔ **Ansatz:** Sicherstellen, dass jeder Repräsentant einer Äquivalenzklasse mit jedem Repräsentanten der anderen Äquivalenzklassen in einem Testfall zur Ausführung kommt (d. h. paarweise Kombination statt vollständiger Kombination).



Paarweises Testen - Bsp

Webapplikation soll mit gängigen Browsern und gängigen Betriebssystemen kompatibel sein und im online sowie im offline Modus funktionieren.

Betriebssystem	Modus	Browser
Windows	Online	IE
Linux	Offline	Firefox
Mac OS X		Chrome

→ 18 Konfigurationen,
Bei verschiedenen
Versionen
entsprechend
Mehr.



Paarweises Testen

- Pragmatischer Ansatz: Jedes Ausprägungspaar ist durch einen Testfall abgedeckt
- **Nicht:** Alle kombinatorisch möglichen Konfigurationen



- Jeder Web Browser mit jedem Betriebssystem
- Jeder Modus mit jedem Webbrowser
- Jeder Modus mit jedem Betriebssystem



Paarweiser vollständiger Test - Bsp

OS	Modus	Browser
Windows	Online	IE
Windows	Offline	Firefox
Windows	Online	Chrome
Linux	Online	Firefox
Linux	Offline	Chrome
Linux	Offline	IE
Mac OS X	Online	Chrome
Mac OS X	Offline	IE
Mac OS X	Online	Firefox

9 Testfälle anstelle von 18 !

