Nicht funktionale Prüfung (oo):

Spezifikationstest (auch: Black-Box-Test):
→ Überprüfen der Eingabe-/Ausgabebeziehungen einer Klasse oder Komponente.
→ Die interne Struktur wird vernachlässigt.
→ Erstellung geeigneter Äquivalenzklassen

> **Äquivalenzklasse** – Bezüglich Ein- und Ausgabedaten ähnliche Objekte, wo erwartet wird, dass sie sich gleich verhalten

---

Beispiel: Anwendungsfall "Veranstaltung finden"
Anwendungsfall: Veranstaltung finden | *Primärer Akteur*: Käufer | *Kontext*: Ticket Informationsschalter | *Bedingungen*: keine
Testenszenario:
1. **Benutzer** wählt Genre und einen Datums- und Uhrzeitrahmen aus. Das System überprüft, dass der Zeitrahmen stimmt
2. Das **System** zeigt das Ergebnis der Abfrage.
3. Der **Benutzer** fordert eine detaillierte Ansicht einer ausgewählten Veranstaltung an.
4. Das **System** zeigt die Detailansicht für die ausgewählte Veranstaltung.

Erweiterungen: 2a. Die Abfrage enthält keine Ergebnisse. Zurück zu 1. oder abbrechen.
Konsequenzen: keine.
Anweisungen: Abfrage durch das Webportal des Systems, Kiosk-Schnittstelle, Kassa-Client (unterstützter sekundärer Akteur)

Nr.	Testfall	Erwartetes Ergebn.	Eingabedaten
1	Entering valid name	Search Result	name="cinema word"
2	Entering valid state	Search Result	state="NÖ"
3	Entering empty state	Exception Message	state="
4	Entering invalid state	Empty Search Result	state="bavaria"
5	Entering valid city	Search Result	city="Hürm"

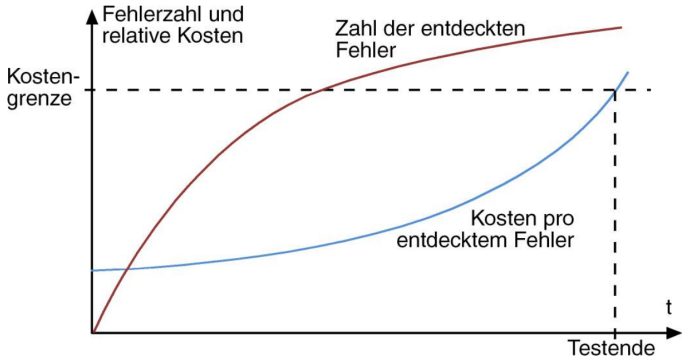
6	**Entering city not in database**	**Empty Search Result**	**city="4444"**

> **Schwellenwert** – Wert, der gerade noch innerhalb oder schon außerhalb eines gültigen Wertebereichs liegt.

→ Falsche Vergleichsoperatoren, z.B. < statt <=, Rundungsfehler,...

Strukturtest (White-Box-Test):
→ Prüfung der *internen Programmstruktur*
→ Struktur des zu untersuchenden Objekts muss bekannt sein
→ Experimentiere: 1. Bestimme alle Ausführungswege und 2. Entdecke mögliche Fehler in einem dieser Wege
→ **Vorteil**: Lage des Fehlers kann sehr genau eingegrenzt werden
→ **Nachteil**: Sehr kosten- und arbeitsintensiv

Grad der Codeabdeckung



C0-Überdeckung oder Knotenüberdeckung: Jede auftretende **Anweisung** im Code muss mindestens **einmal** in einem Testfall durchlaufen werden.

C1-Überdeckung oder Kantenüberdeckung: Jede **Kante** im Kontrollflussgraph muss mindestens **einmal** in einem Testfall durchlaufen werden.

C*infinite*-Überdeckung oder Pfadüberdeckung: Jede (praktisch) mögliche Reihenfolge aller Codeanweisungen müssen von den Testfällen abgedeckt werden.

Bedingungsüberschneidung: Jede Kombinationen der Teilausdrücke logischen Ausdrucks muss in Testfällen überprüft werden.

Kantenüberlappung vs. Zustandsüberlappung IF (A=true) and (B=false) THEN ...
Zwei Testfälle, um eine **Kantenüberlappung** zu erreichen: (A=true) and (B=false) == true | (A=true) and (B=true) == false
Vier Testfälle, um **Bedingungsüberlappung** erreichen: A = true, B = true A = true, B = false A = false, B = true A = false, B = false

---

Revision #3
Created 30 September 2022 11:28:36 by Merith Holtmann
Updated 30 September 2022 11:30:03 by Merith Holtmann