VU Softwarequalitätssicherung WS2006
Testen von Softwaresystemen

Mag. Dipl.-Ing. Denis Frast

denis.frast@qse.ifs.tuwien.ac.at

Institut für Softwaretechnik und Interaktive Systeme

Testen: Häufige Fehleinschätzungen

- **Programmieren erfordert Fachleute, aber Testen kann jeder**

☺

- **Es gibt ohnehin keine systematischen Testmethoden**

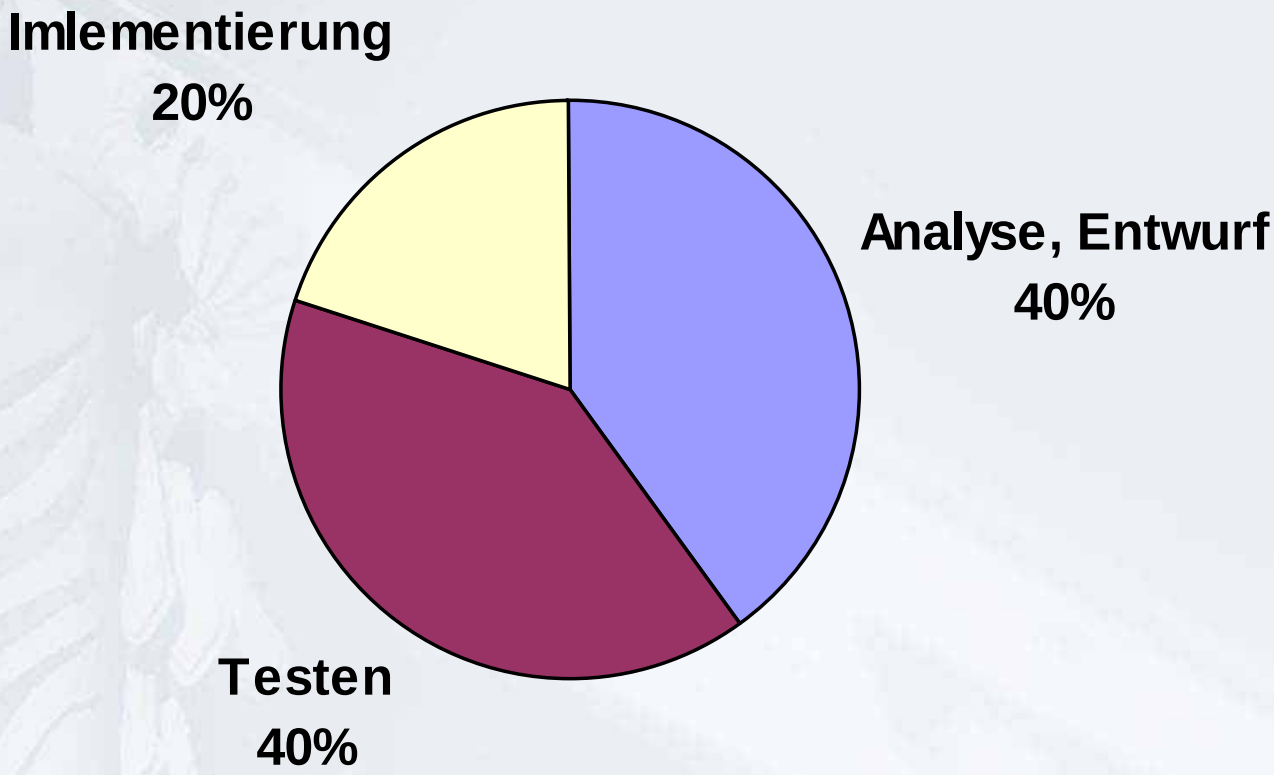
☺

- **Testen erledigen wir zum Schluss**

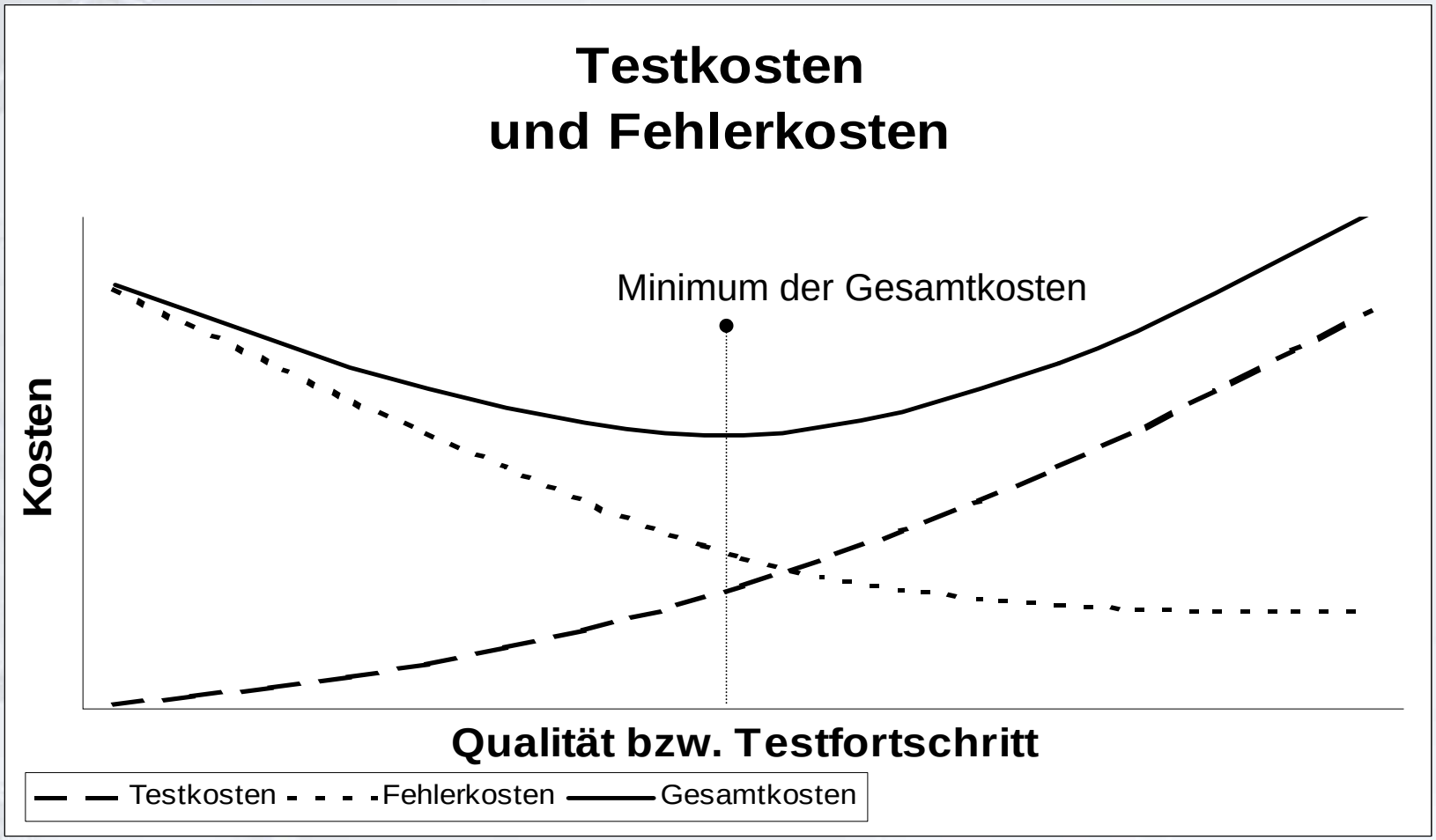
☺

- **Wenn wir sauber entwickeln, ist Testen fast überflüssig**

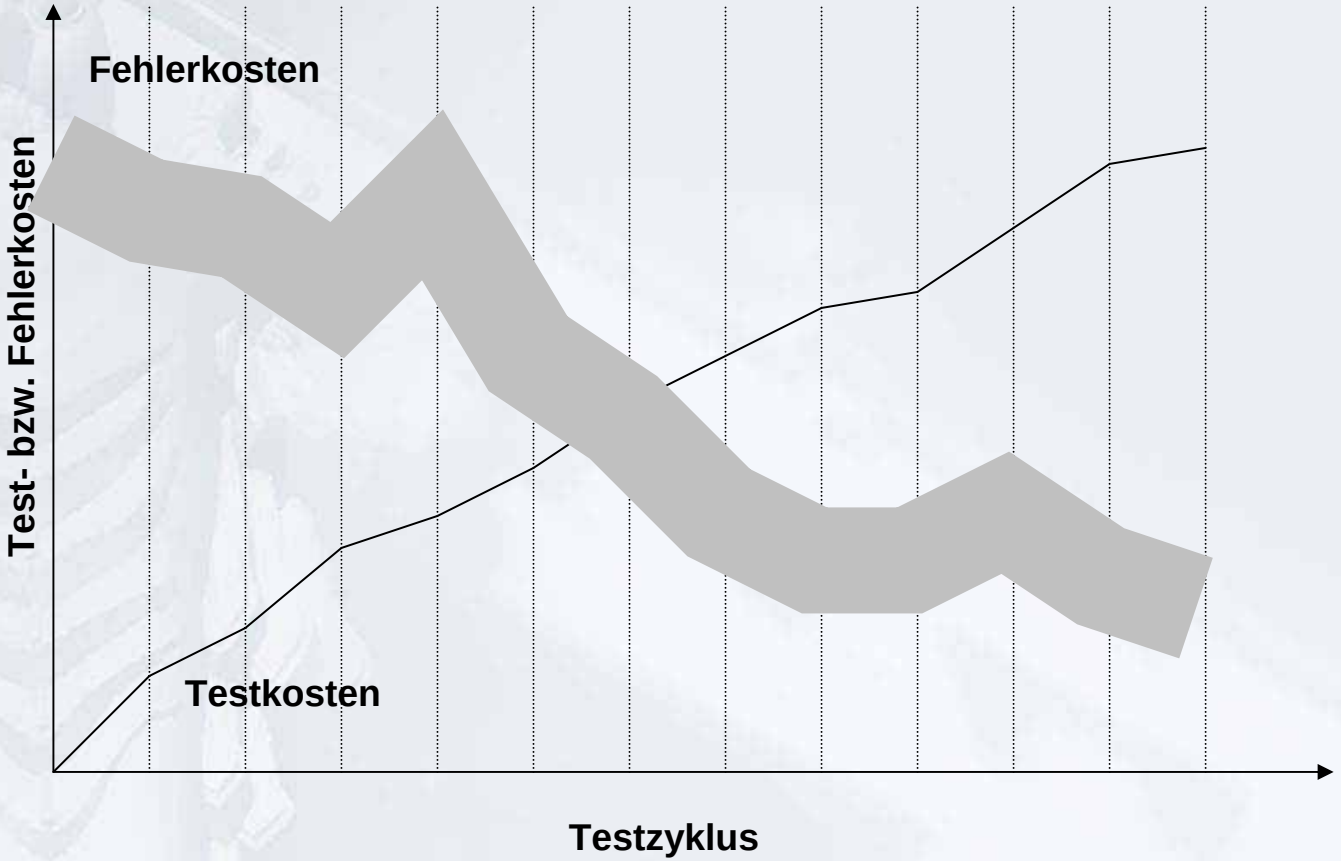
Aufwand: 40-20-40-Regel



Was kostet Qualität?



Testkosten und Qualität in der Praxis



Definition des Begriffs Testen

Testen wird oft falsch definiert:

- **Testen ist der Prozess, der zeigen soll, dass ein Programm keine Fehler enthält**
- **Testen soll zeigen, dass ein Programm seine Spezifikation erfüllt**
- **Testen ist der Prozess, der das Vertrauen erzeugt, dass ein Programm das tut, was es soll**

Was ist Testen?

- **Testen ist…**
  Aufdecken von möglichst vielen Fehlern
- **Testen ist nicht…**
  Fehlerstellen lokalisieren und
  Fehler entfernen (Debuggen)
- **Testen ist nicht…**
  Fehlerfreiheit nachweisen
  (das geht nur mit einem Korrektheitsbeweis!)

Testen von Software

- Was ist Testen?
  - “Testen ist das Ausführen eines Programms, mit der Absicht, Fehler zu finden.“ [Myers 1979]
- Was ist ein Fehler?
  - “Ein Fehler ist eine Abweichung zwischen einem Programm und dessen Spezifikation.“ [Kaner et al., 1999]
- Testen als Teil des Qualitätsmanagements ist ein wichtiger und fortlaufender Bestandteil des Software-Entwicklungsprozesses, nicht bloß eine Pflichtübung am Ende

Psychologie des Testens

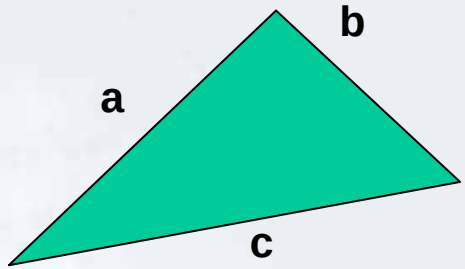
- Testen ist ein destruktiver Prozess!
  - Jedes Programm hat Fehler – Ziel ist es, sie zu finden.
  - Implementierer ist zu wenig destruktiv,
    daher sollte der Tester nicht Implementierer sein.
  - Richtige Einstellung: versuchen, das Programm zu Fall zu bringen.
- Wert des nächsten gefundenen Fehlers:
  - Mit jedem gefundenen Fehler wird das Programm wertvoller!
  - Korrektheitsbeweis ist unmöglich -> Aufgabe, die den Tester frustriert
  - „Noch einen Fehler zu finden“ ist eine kleinere Aufgabe, der sich der Tester gewachsen fühlt.

Testen: Ein Beispiel

Ein Programm liest 3 ganze Zahlen a, b und c und soll feststellen, ob sie die Seiten eines

- **gleichseitigen Dreiecks**
- **gleichschenkligen Dreiecks**
- **rechtwinkeligen Dreiecks**
- **sonstigen gültigen Dreiecks**

bilden.



Mit welchen Eingaben würden Sie es testen?

Beispiel: Testfälle

Gültige Dreiecke

- **Gleichseitig: 3, 3, 3**
- **Gleichschenkelig: 5, 5, 3**
- **Rechtwinkelig: 3, 4, 5**
- **Sonstiges: 3, 5, 7**
- **Permutationen davon**
  (3,5,5), (5,3,5),
  (3,5,4), (4,5,3), (4,3,5), (5,3,4),
  (5,4,3),
  (3,7,5), (5,7,3), (5,3,7), (7,3,5),
  (7,5,3)

Ungültige Dreiecke

- **a+b < c: 3, 3, 7**
- **a+b = c: 3, 4, 7**
- **Negative Seitenlänge: 3, 4, -5**
- **0, 0, 0**
- **Nur 2 Seitenlängen: 3, 4**
- **Permutationen davon**
  (3,7,3), (7,3,3),
  (3,7,4), (7,4,3),...
  (3,-5,4), (-5,3,4)...

- **Haben Sie überall das erwartete Ergebnis dazugeschrieben?**

Testfälle

- Tests bestehen aus einer Menge von Testfällen (Testsuite)
- Ein Testfall besteht aus einer Menge (V, E, A, R)
  - V: Vorbedingungen (z.B. System- oder DB-Zustände)
  - E: Eingabewerte
  - A: Aktionen zur Eingabe der Werte
  - R: erwartete Ergebnisse
- Testerfolg
  - Ein Test ist erfolgreich, wenn er einen Fehler gefunden hat
  - kein Fehler → Information über Produktqualität (Testüberdeckung!)

Qualität von Testfällen

- **Fehlerfindungsrate: Wahrscheinlichkeit, Fehler zu finden**
- **Fehlerlokalisierung: Einschränkung der Fehlerlokation**
- **Testüberdeckung: Wenige gute überdecken mehr als viele schlechte**
- **Komplexität: Allzu komplexe Testfälle sind selten zur Gänze durchführbar! → Kompromiss zwischen Anzahl und Komplexität!**
- **Generalität: so allgemein wie möglich und so genau wie nötig → generische Beschreibung der relevanten Eigenschaften ohne konkrete Werte, z. B. über Äquivalenzklassen; Werte werden erst beim Test inkludiert**

Testfälle: (Negativ-)Beispiel 1 (Dreieck)

- Zwei an sich gute Testfälle testen dasselbe
- TF 1
  - V: -
  - E/A: Start des Programms „Dreieck“, Eingabe der Werte 3, 4, 3 und Start der Auswertung
  - R: Programm liefert „gleichschenkliges Dreieck“, terminiert
- TF 2
  - V: -
  - E/A: Start des Programms „Dreieck“, Eingabe der Werte 3, 5, 3 und Start der Auswertung
  - R: Programm liefert „ gleichschenkliges Dreieck“, terminiert

Testfälle: Negativ-Beispiel 2 (Bibliothek)

- Vermischung von Eingaben und Ergebnissen
- TF 1
  - V: -
  - E/A: Leser möchte Buch x entlehnen
  - R: Entlehnung wird vom System zugelassen
- TF 2
  - V: -
  - E/A: Leser möchte Buch x entlehnen
  - R: Entlehnung wird nicht zugelassen
- vielleicht fehlt aber auch nur die Vorbedingung: Buch x ist entlehnbar / nicht entlehnbar

Testfälle: Negativ-Beispiel 3 (Bibliothek)

- **kein Ergebnis**
- **TF 1**
  - **V: -**
  - **E/A: Leser möchte Buch x entlehnen**
  - **R: -**

Testfälle: Negativ-Beispiel 4 (Bibliothek)

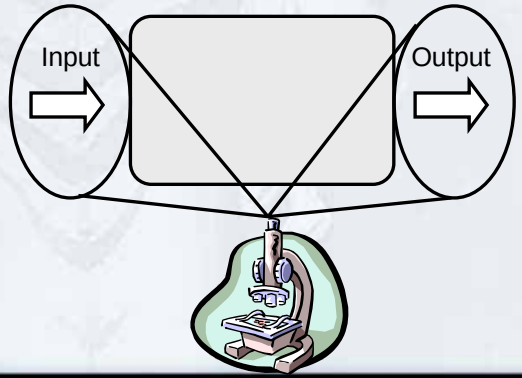
- keine Eingabe an das System bzw. kein vom System beeinflusstes Ergebnis
- TF 1
  - V: -
  - E/A: Person kann sich nicht ausweisen, gibt aber Adresse bekannt. Leser soll angelegt werden.
  - R: Leser wird ohne Vorlage eines Ausweises nicht angelegt. Leserkarte wird nicht ausgegeben

Testfall-Spezifikationsmethoden

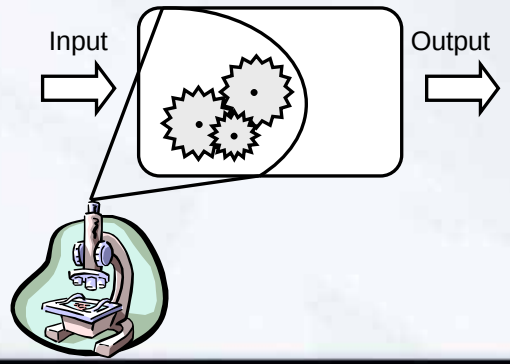
- Strukturierte Anweisungen zur Generierung von Testfällen
- Vorteile gegenüber zufälliger Testfalldefinition
  - Höhere Qualität der Testfälle
  - Machen Qualität und Intensität eines Tests transparent
  - Erreichen gewünschte Abdeckung für jeden Teil der Applikation
  - Effektiver für gegebene Fehlertypen
  - Tests werden nachvollziehbar
  - Testprozess wird unabhängig von Personen
  - Testfall-Spezifikationen werden übertragbar und aktualisierbar
  - Testprozess wird besser plan- und steuerbar
- „Nachteile“
  - Fundiertes Wissen der Tester notwendig

Black Box vs. White Box

- **Black Box**
  - **Basiert auf Spezifikation**
  - **Wissen über innere Struktur wird ignoriert**
  - **Daten-getriebene, Input-Output-getriebene Tests**
  - **Anforderungsüberdeck.**
  - **Partitionierung der Eingabe-Daten**



- **White Box (Glass Box)**
  - **Basiert auf Code(struktur)**
  - **Wissen über innere Struktur notwendig**
  - **Logik-getriebene Tests**
  - **Überdeckung von Pfaden im Kontrollflussgraphen**
  - **Partitionierung von Daten f. Bedingungsauswertung**



Testfall-Ableitungsprinzipien

- Äquivalenzklassen
- Grenzwertanalyse
- Verarbeitungslogik
- CRUD-Matrix
- Ursache/Wirkungs-Graphen
- Zustandsübergänge
- Error Guessing

Testüberdeckung

- **Überdeckungsmaße geben Testintensität an**
  - **Strukturell**
    - **Anweisungs- bzw. Knotenüberdeckung (C0)**
    - **Entscheidungs- bzw. Kanten-Überdeckung (C1)**
    - **Bedingungsüberdeckung**
    - **Anforderungsüberdeckung**
    - **Pathn coverage: Pfade der Länge n**
  - **Funktional**
    - **Abdeckung aller Eingaben, Ausgaben, Funktionen, Wertkombinationen ...**

Äquivalenzklassenzerlegung

- Zerlegung einer Menge von Daten (Input oder Output) in Untermengen (Klassen), welche die selben oder äquivalente Ergebnisse oder Auswirkungen produzieren
- Jede Klasse(nkombination) sollte mindestens einmal getestet werden
- Finde zu testende Funktionen
- Finde Eingabe-Vektor (A, B, C, ...) (= Faktoren) je Funktion
  - explizite: Parameter
  - implizite: globale Variablen, Systemzustand, …
- Finde für jeden Faktor seine un-/gültigen Äquivalenzklassen (Klassen A1, A2, ...; B1, B2, ...; ...) und wähle je 1 Wert aus jeder
- Wahl der Kombinationen: (A1, B1), (A1, B2), ... bestimmt Intensität

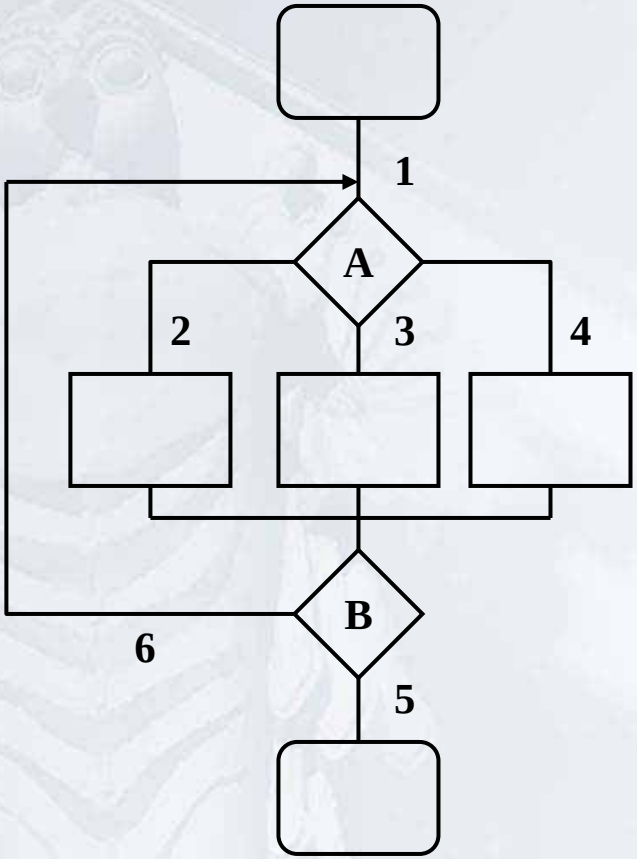
Äquivalenzklassenzerlegung Beispiele

- Anforderung: 18 < Alter ≤ 65 → drei Äquivalenzklassen
  - A1: Alter ≤ 18 (ungültig)
  - A2: 18 < Alter ≤ 65 (gültig)
  - A3: Alter > 65 (ungültig
  - → Drei Testfälle: x ∈ A1, z.B. 10, x ∈ A2, z.B. 35, x ∈ A3, z.B. 70
- Alter über 40, Body Mass Index über 25
  - Alter ≤ 40, BMI = 20
  - Alter > 40, BMI = 25
  - Alter ≤ 40, BMI = 30
  - Alter > 40, BMI = 35
- Äquivalenzklassen beim Beispiel „Dreieck“

Grenzwertanalyse

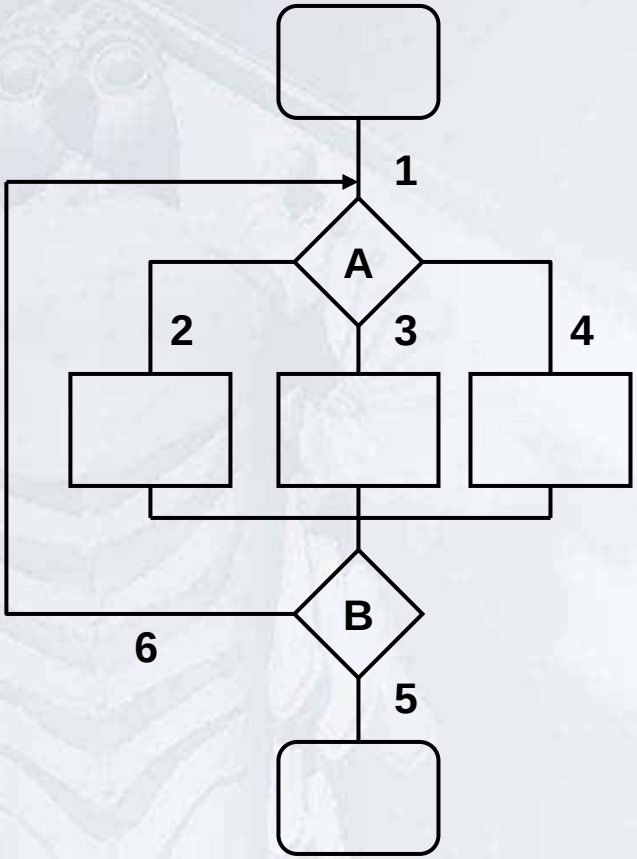
- **Erweiterung der Äquivalenzklassenzerlegung für bessere Überdeckung**
- **Grenzwerte werden als Klassenrepräsentanten gewählt → je Klassengrenze ein Testfall**
- **Anforderung: 18 < Alter ≤ 65**
  - **18 (ungültig), 19 (gültig), 65 (gültig) and 66 (ungültig)**
- **Ev. mehrere Testfälle je Klassengrenze: Alter ≤ 18**
  - **17 (gültig), 18 (gültig) and 19 (ungültig)**
    Bedenke Tippfehler: Alter = 18!

Strukturtest: Testmaß 1



- **Pfade der Länge 1**
  - **-: (1)**
  - **A: (2); (3); (4)**
  - **B: (5); (6)**
- **Testfälle**
  - **(1, 2, 5)**
  - **(1, 3, 6, 4, 5)**

Strukturtest: Testmaß 2



- Pfade der Länge 2
  - A: (1,2); (1,3); (1,4); (6,2); (6,3); (6,4)
  - B: (2,5); (3,5); (4,5); (2,6); (3,6); (4,6)
- Testfälle
  - (1, 2, 5)
  - (1, 3, 5)
  - (1, 4, 5)
  - (1, 2, 6, 2, 5)
  - (1, 3, 6, 4, 6, 3, 5)

Datenzyklustest

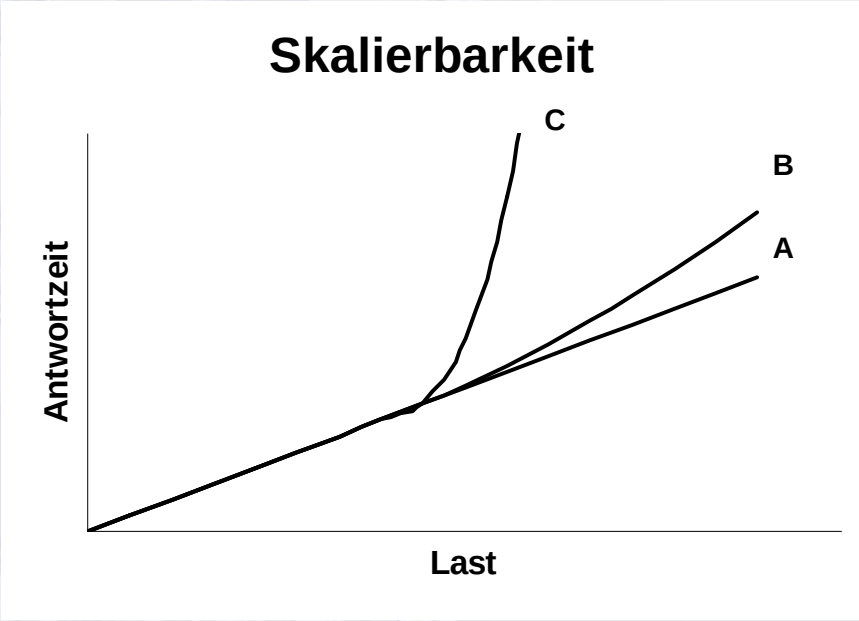
1. Bestimme Entitäten und Funktionen, die darauf zugreifen
   - Create
   - Read
   - Update
   - Delete
2. Bilde CRUD-Matrix

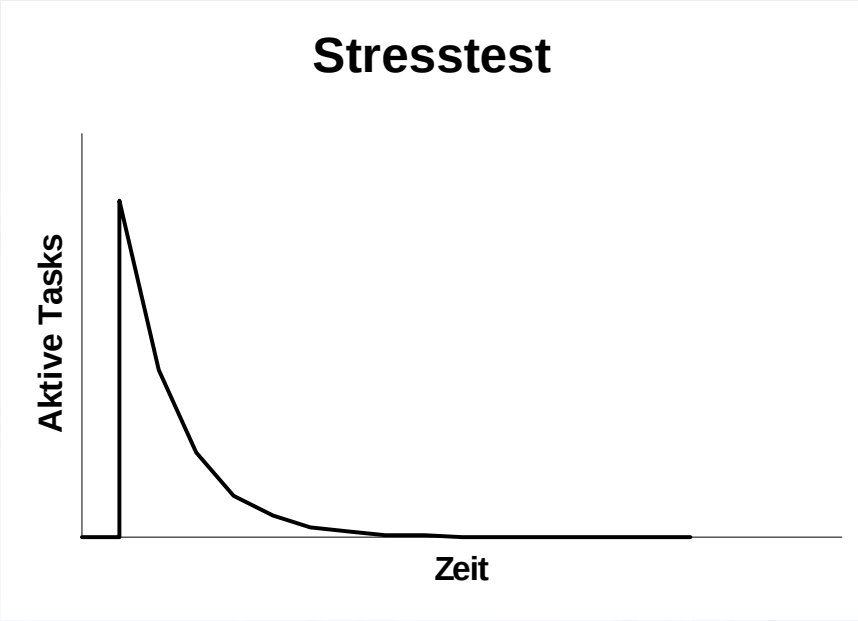
	Entität 1	Entität 2	Entität 3
Funktion 1	R	C, U, D	-
Funktion 2	C	R	-
Funktion 3	C, R, D	-	-

3. Testfälle für jede Entität
   - Create, Read
   - Update, Read
   - Delete, Read

   → mit jeder Funktion!
4. Relationen nicht vergessen!

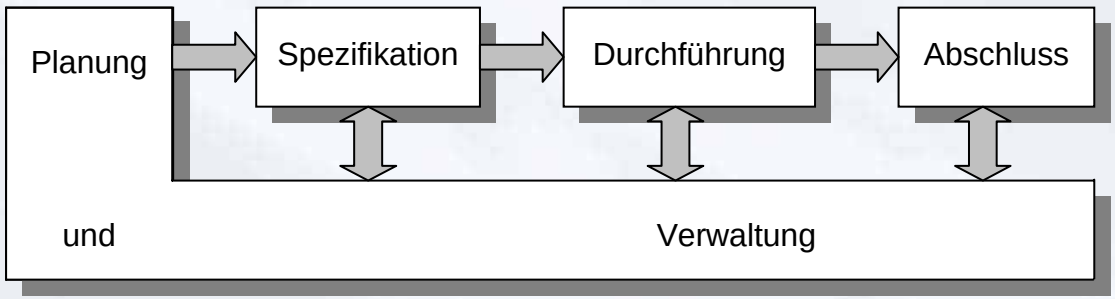
Nichtfunktionale Tests – Beispiel Last/Stress





Testen als Prozess

- **Also was tun Sie wenn Sie die Aufgabe bekommen, ein Programm zu testen?**
  - ***„Unter Testen versteht man den Prozess des Planens, der Vorbereitung und der Messung, mit dem Ziel, die Eigenschaften eines IT-Systems festzustellen und den Unterschied zwischen dem tatsächlichen und dem erforderlichen Zustand aufzuzeigen.“*** **[Pol et al., 2000]**
- **Phasen im Testprozess**
  - **Planung und Verwaltung**
  - **Spezifikation**
  - **Durchführung**
  - **Abschluss**



Zusammenfassung

- Testen hat zum Ziel
  - Fehler zu finden
  - Qualität zu messen
- Testfälle sind Eingaben an das System mit definierten erwarteten Ergebnissen
- Testfall-Spezifikationsmethoden sind strukturierte Anweisungen zur Generierung von qualitativ hochwertigen Testfällen mit vielen Vorteilen gegenüber „zufälligen“ Tests
- Testen ist ein Prozess, der den gesamten Software-Lebenszyklus begleitet

Literaturempfehlungen

- Martin Pol, Tim Koomen, Andreas Spillner: “Management und Optimierung des Testprozesses: ein praktischer Leitfaden für erfolgreiches Testen von Software, mit TPI und TMap”, dpunkt.verlag, 2000, ISBN 3-932588-65-7
- Cem Kaner, Jack Falk, Hung Quoc Nguyen: “Testing Computer Software", Wiley, 1999, ISBN 0-471-35846-0
- DeMarco, Tom: “Der Termin: Ein Roman über Projektmanagement”, Carl Hanser Verlag, 1998, ISBN 3-446-19432-0
- Thaller, "Software-Test", dPunkt, 2002
- Spillner et al., "Basiswissen Software-Test", dPunkt, 2003