Willkommen zur Vorlesung
Softwarekonstruktion
im Wintersemester 2012 / 2013

Prof. Dr. Jan Jürjens

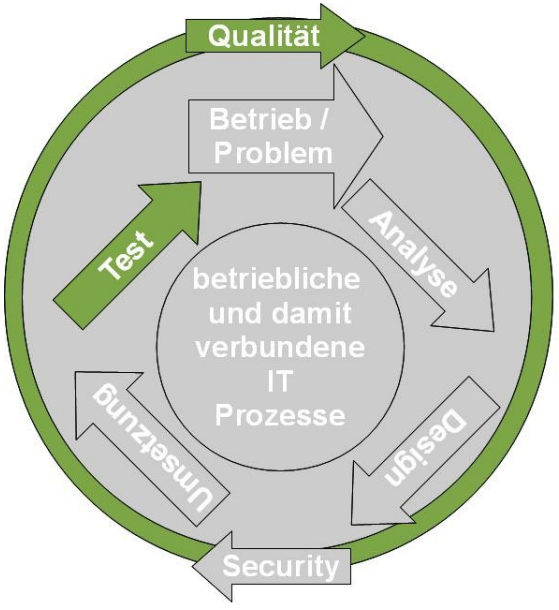
TU Dortmund, Fakultät Informatik, Lehrstuhl XIV

2.4 Black-Box-Test

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

Einordnung
Black-Box-Test

- Qualitätsmanagement
- Testen
  - Einführung
  - Grundlagen Softwaretesten
  - Testen im Softwarelebenszyklus
  - Statischer Test
  - Black-Box-Test
  - White-Box-Test
  - Test-Management
  - Testwerkzeuge
- Modellgetriebene SW-Entwicklung

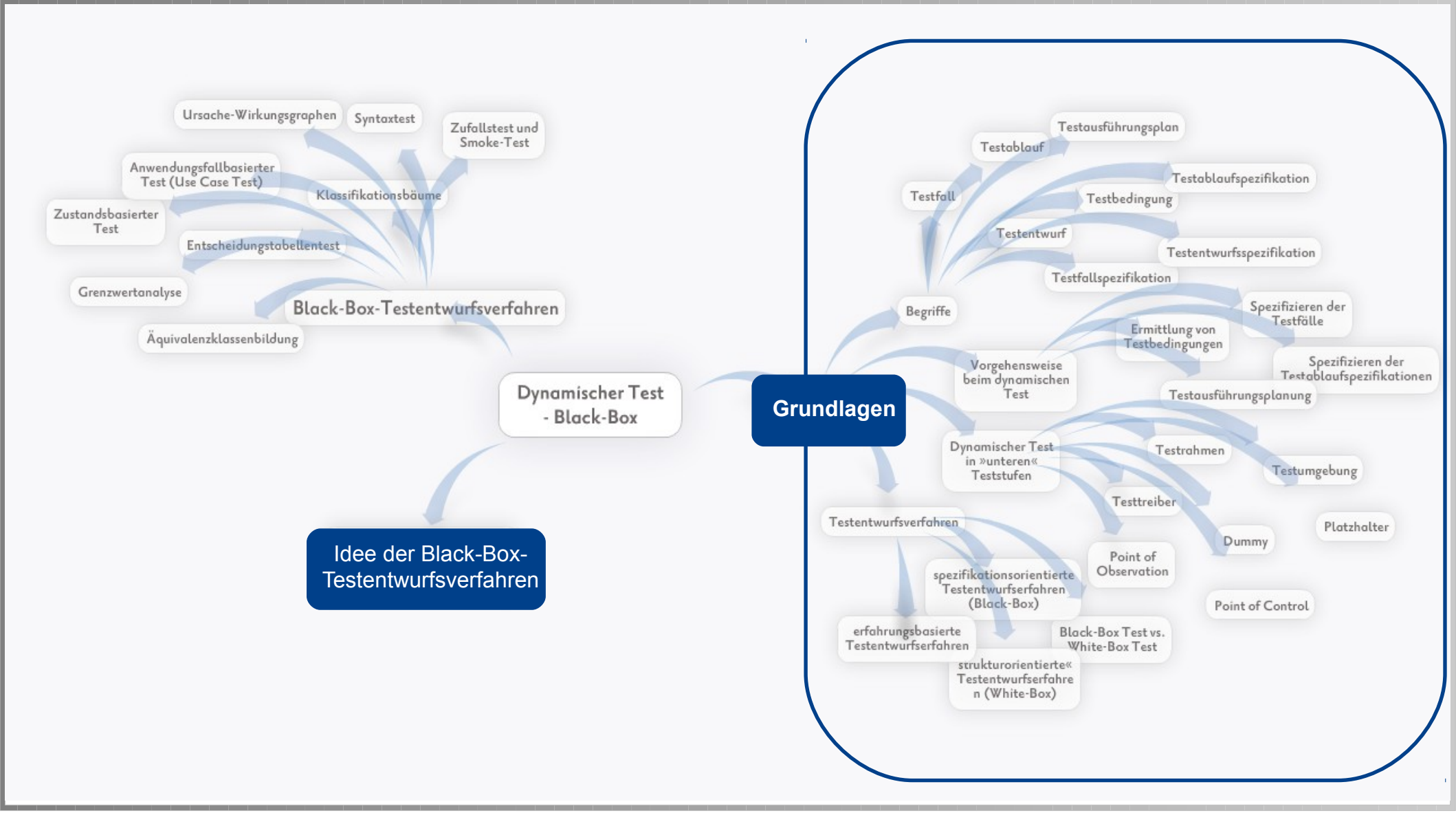


2.4 Black-Box-Test

2.4 Black-Box-Test

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

Dynamischer Test – Black Box



Statischer Test vs. dynamischer Test

Programme sind statische Beschreibungen von dynamischen Prozessen (Berechnungen).

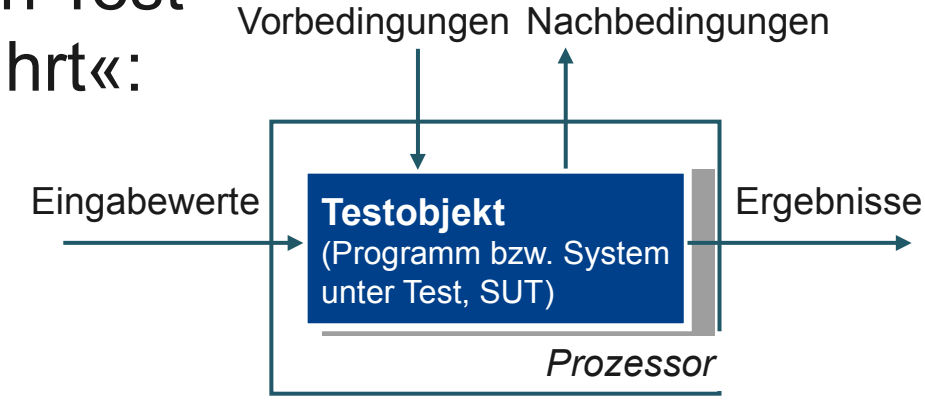
Statische Tests prüfen die Testobjekte »an sich«:

- Artefakte des Entwicklungsprozesses, z.B. informelle Texte, Modelle, formale Texte, Programmcode, ...

Dynamische Tests prüfen die durch »Interpretation« einer Beschreibung (Testobjekt) resultierenden Prozesse.

Das Testobjekt wird im dynamischen Test also auf einem Prozessor »ausgeführt«:

- Bereitstellen von Eingabewerten
- Beobachten der Ergebnisse.



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

Testsuite: Zusammenstellung (Aggregation) mehrerer Testfälle für Test einer Komponente / Systems, 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. bestimmten 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 den Zustand des Testobjekts und seiner Umgebung, die vor der Durchführung eines Testfalls oder Testablaufs erfüllt sein müssen.

Nachbedingung: Zustand des Testobjekts (und/oder der Umgebung), in dem sich Test-objekt (oder Umgebung) nach Ausführung eines Testfalls / Testlaufs befinden muss.

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

Testskript: Bezeichnet üblicherweise eine Testablaufspezifikation, insbesondere eine automatisierte.

Dynamischer Test – Begriffe (3)

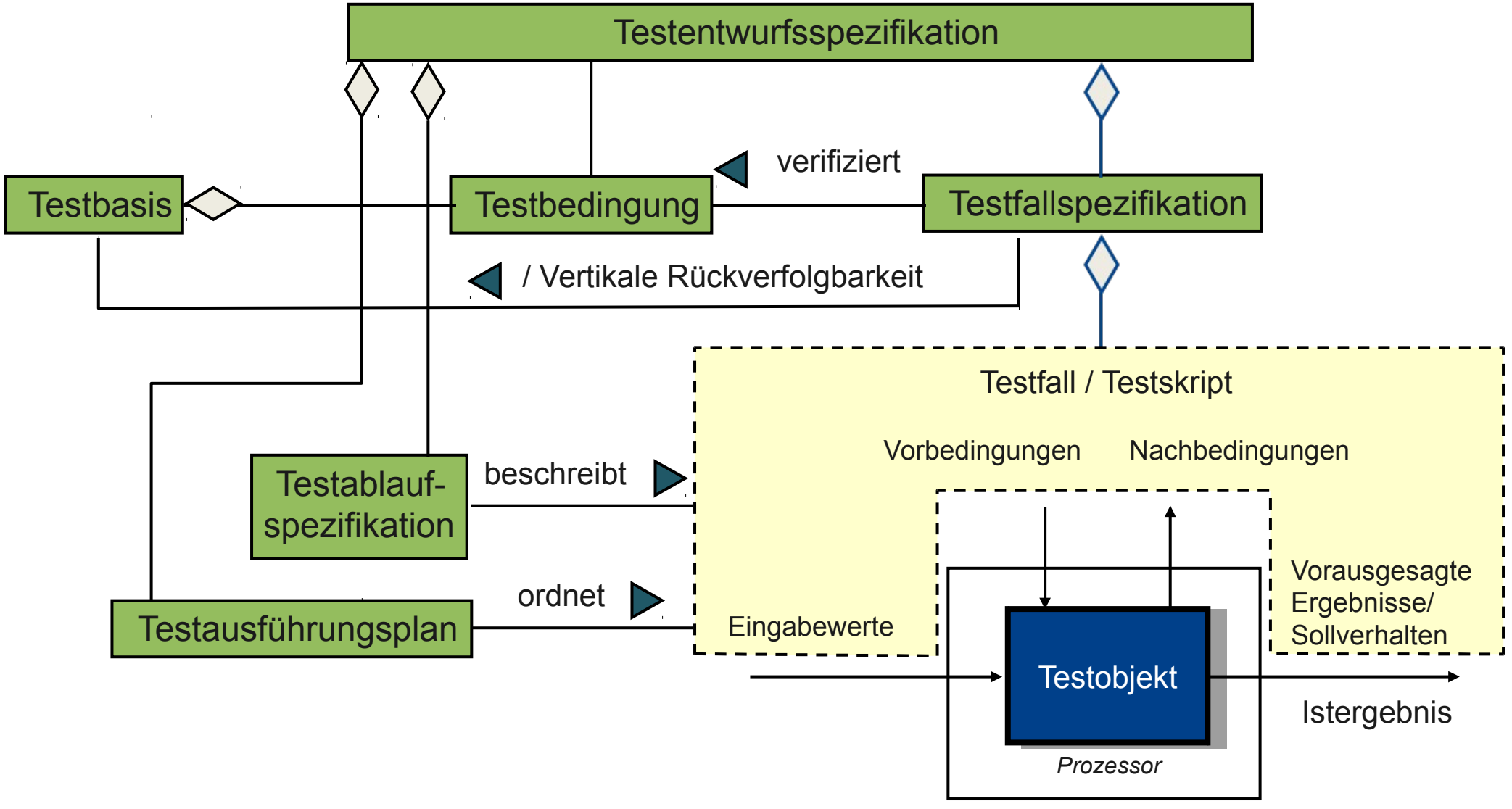
Testausführungsplan: Ein Plan für die Ausführung von Testskripten. Testskripte sind im Testausführungsplan mit ihrem Kontext und in der auszuführenden Reihenfolge festgelegt.

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

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

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

Dynamischer Test - Zusammenhänge

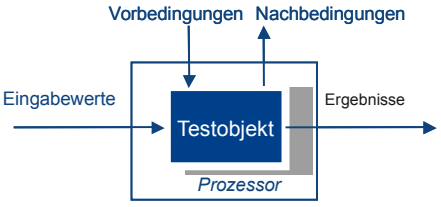


Dynamischer Test – Vorgehensweise

1. **Entwerfen von Tests durch Ermittlung von Testbedingungen**
   - Analyse der dem Test zugrunde liegenden Dokumente, um festzustellen, was getestet werden muss, um also die Testbedingungen festzulegen.
   - Testbedingungen müssen auf Spezifikationen / Anforderungen zurückverfolgbar sein, um Auswirkungen von Änderungen an Spezifikationen / Anforderungen auf Tests (impact analysis) und die Abdeckung von Spezifikationen bzw. Anforderungen durch Tests (Überdeckung) zu ermitteln.
2. **Spezifizieren der Testfälle**
   - Entwicklung und detaillierte Beschreibung der Testfälle und Testdaten unter Verwendung von Testentwurfsverfahren.
   - Vorausgesagte (erwartete) Ergebnisse beinhalten Ausgaben, Änderungen von Daten und Zuständen sowie alle anderen Folgen des Tests – falls diese nicht definiert werden, könnte plausibles, aber fehlerhaftes Ergebnis als richtig angesehen werden (=> vor der Testdurchführung festlegen !).
3. **Spezifizieren der Testablaufspezifikationen**
   - Testfälle in eine ausführbare Reihenfolge bringen. Vor- und Nachbedingungen der Testfälle beachten.
4. **Testausführungsplanung**
   - Reihenfolge angeben, in der die verschiedenen Testablaufspezifikationen bzw. Testskripte (automatisch) ausgeführt werden, und wann und von wem sie auszuführen sind.
   - Faktoren wie Regressionstests (wiederholte Testausführung), Priorisierung und logische Abhängigkeiten zwischen Testfällen bzw. Testskripten berücksichtigen.

Dynamischer Test in »unteren« Teststufen

Ziele des dynamischen Tests:



- Durch stichprobenhafte Programmläufe Nachweis der Erfüllung der festgelegten Anforderungen durch das Testobjekt.
- Aufdeckung von eventuellen Abweichungen und Fehlerwirkungen.
- Dabei mit möglichst wenig Aufwand möglichst viele Anforderungen überprüfen bzw. Fehler nachweisen.

Die Ausführung dynamischer Tests erfordert die Ausführbarkeit des Testobjekts.

- Zur Testdurchführung muss ein ablauffähiges Programm vorliegen !

Einzelne Klassen, Module/Komponenten und Teilsysteme sind jedoch in der Regel für sich alleine nicht lauffähig:

- Bieten oft Operationen/Dienste für andere Softwareeinheiten an.
- Haben kein »Hauptprogramm« zur Koordination des Ablaufes.
- Stützen sich oft auf Operationen/Dienste anderer Softwareeinheiten ab.

In den »unteren« Teststufen Klassen-/Modultest, Komponententest und Integrationstest muss das Testobjekt also in in einen entsprechenden »Testrahmen« eingebettet werden.

Dynamischer Test – Begriffe (4)

Testrahmen (test harness, test bed): Eine Testumgebung, die aus den für die Testausführung benötigten Treibern und Platzhaltern besteht.

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

Platzhalter (stub): Rudimentäre oder spezielle Implementierung einer Softwarekomponente, um eine noch nicht implementierte Komponente zu ersetzen bzw. zu simulieren [nach IEEE 610].

Dummy: Platzhalter mit rudimentärer Funktionalität

Mock: Platzhalter mit erweiterter Funktionalität für Testzwecke (Logging, Plausibilitätsprüfung)

Simulator: Werkzeug, mit dem die Einsatz- bzw. Produktivumgebung nachgebildet wird.

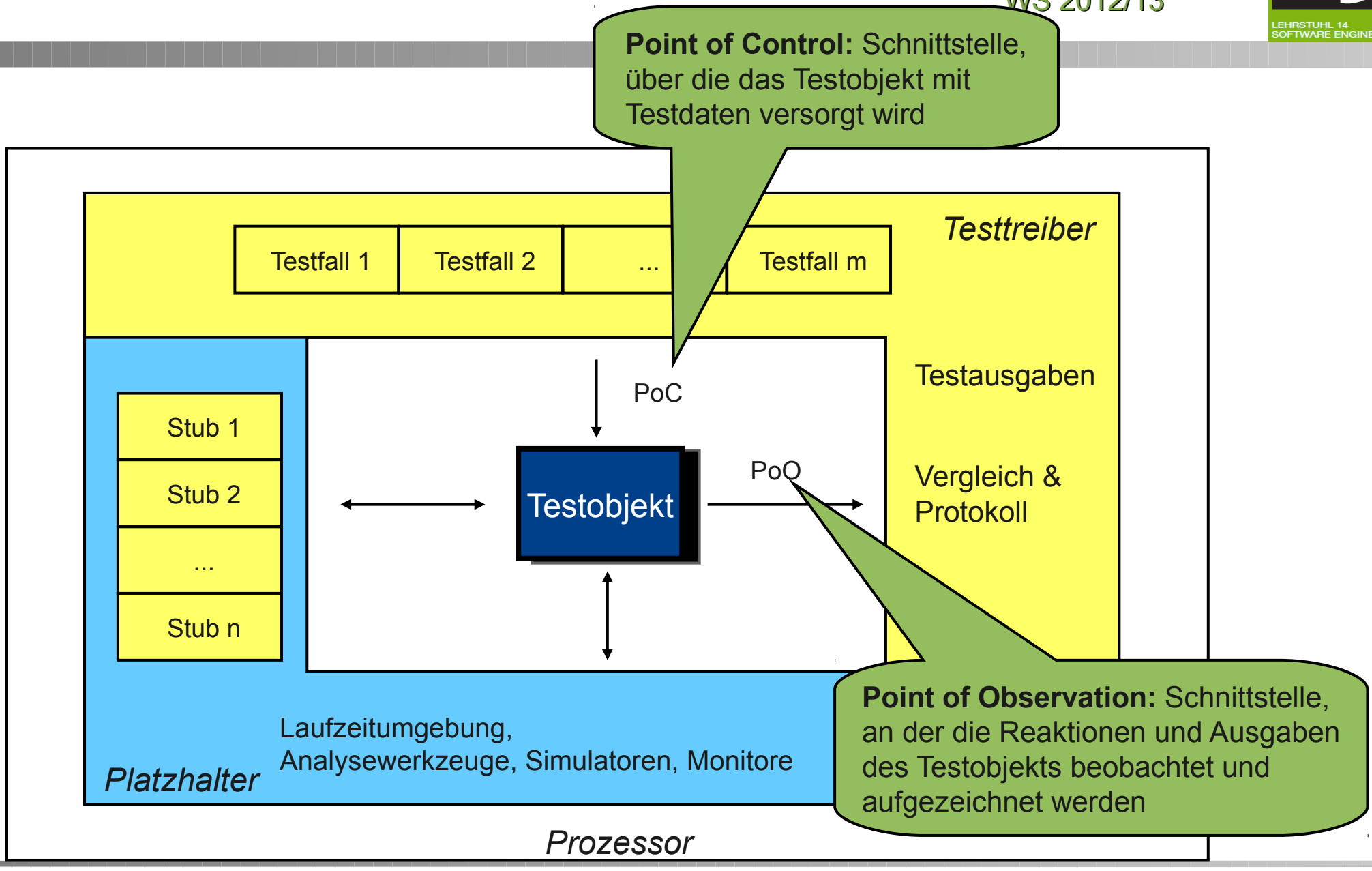
Testentwurfsverfahren: Vorgehensweise, nach der Testfälle abgeleitet oder ausgewählt werden.

Eingangskriterien: Menge der generischen und spezifischen Bedingungen, die es in Prozess ermöglichen, mit bestimmter Aktivität (z.B. Testphase) fortzuschreiten. Zweck ist es, die Durchführung der Aktivität zu verhindern, wenn dafür ein höherer Mehraufwand benötigt (verschwendet) wird als für die Schaffung der Eingangskriterien [Gilb und Graham].

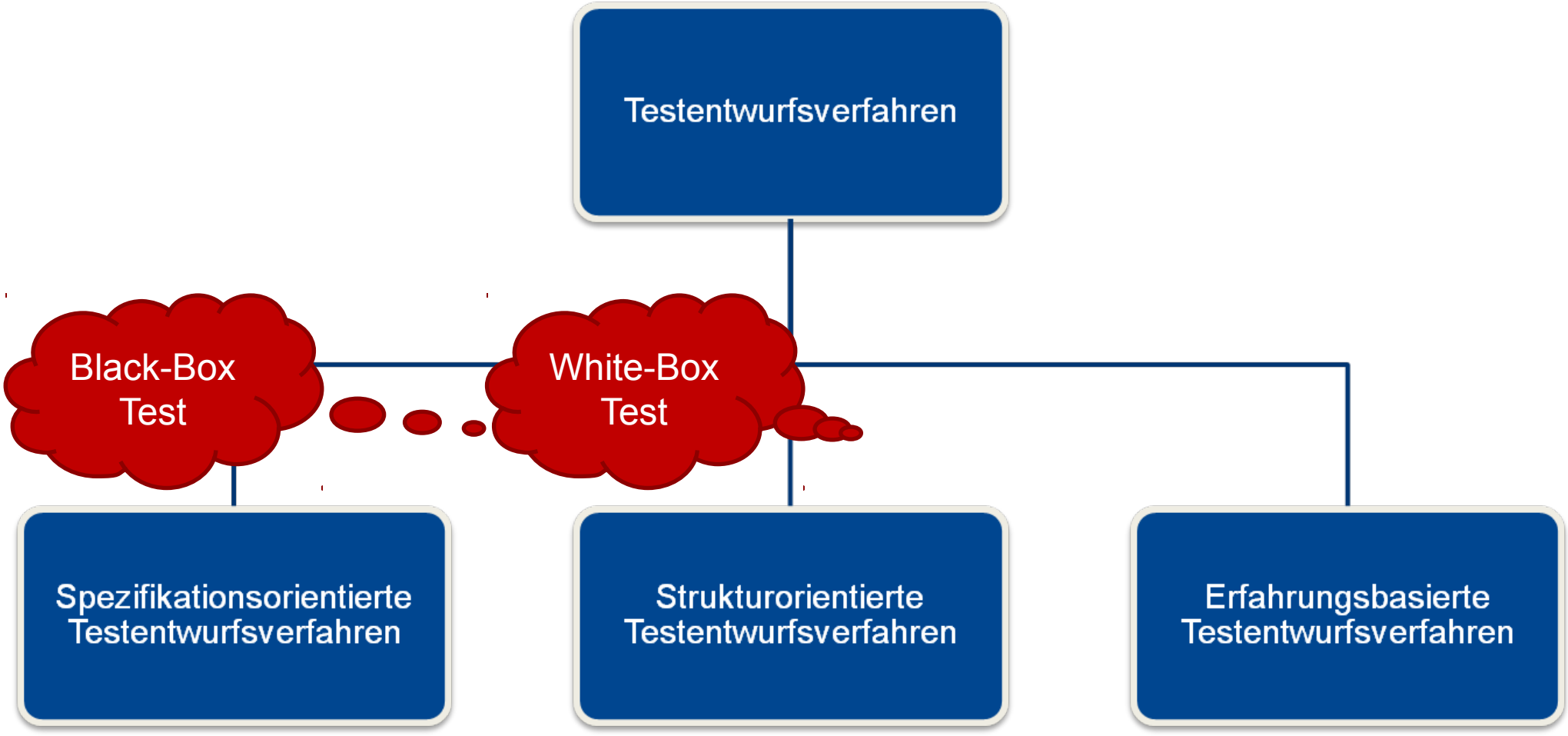
Ausgangskriterien: Die Menge der abgestimmten generischen und spezifischen Bedingungen, die von allen Beteiligten für den Abschluss eines Prozesses akzeptiert wurden. Ausgangsbedingungen für eine Aktivität verhindern es, dass die Aktivität als abgeschlossen betrachtet wird, obwohl Teile noch nicht fertig sind [nach Gilb und Graham].

Ausgangskriterien werden in Berichten referenziert und zur Planung der Beendigung des Testens verwendet.

Aufbau eines Testrahmens



Testentwurfsverfahren



Black-Box Test vs. White-Box Test

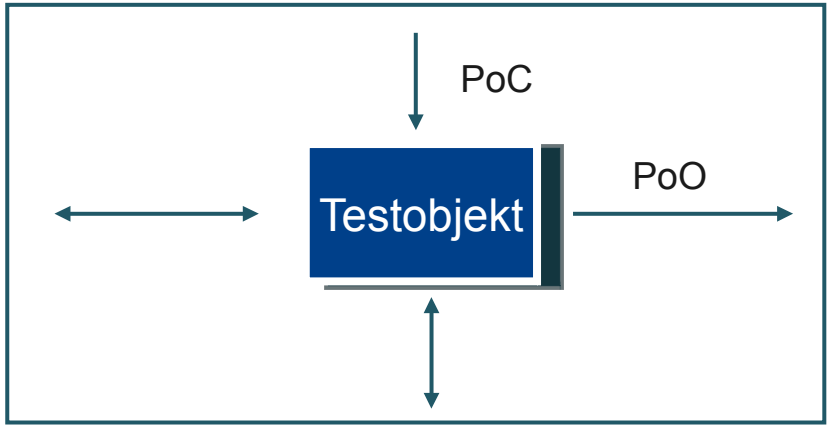
Black-Box Test

Eingabewerte

Ohne Kenntnis der Programmlogik abgeleitet

Testobjekt

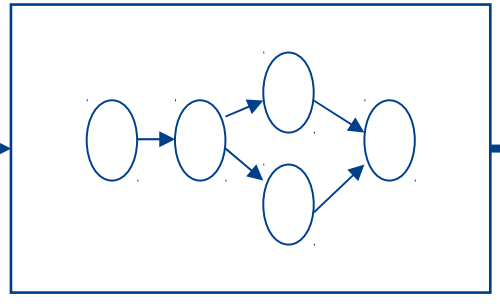
Istergebnis



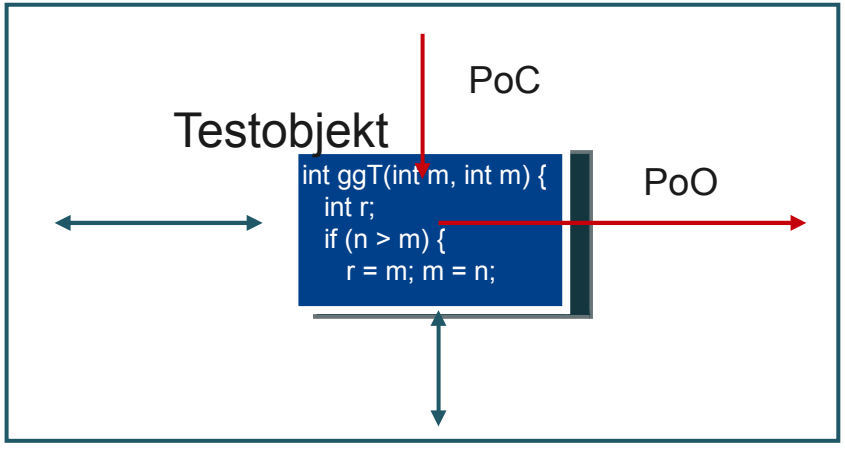
White-Box Test

Eingabewerte

Mit Kenntnis der Programmlogik abgeleitet



Istergebnis



Black-Box-Testentwurfsverfahren (Testobjekt »undurchsichtig«)

Synonym: **spezifikationsorientierte Testentwurfsverfahren.**

Keine Informationen über den Programmtext und den inneren Aufbau.

- Beobachtet wird das Verhalten des Testobjekts von außen (PoO - Point of Observation liegt außerhalb des Testobjekts).
- Steuerung des Ablaufs des Testobjektes nur durch die Wahl der Eingabetestdaten (PoC - Point of Control liegt außerhalb des Testobjektes).
- Modelle bzw. Anforderungen an das Testobjekt, ob formal oder nicht formal, werden zur Spezifikation des zu lösenden Problems, der Software oder ihrer Komponente herangezogen.
- Von diesen Modellen können systematisch Testfälle abgeleitet werden.

White-Box-Testentwurfsverfahren (Testobjekt »durchsichtig«)

Synonym: **strukturorientierte Testentwurfsverfahren.**

Testfälle können auf Grund der Programmstruktur des Testobjektes gewonnen werden.

- Informationen über den Aufbau der Software werden für die Ableitung von Testfällen verwendet, beispielsweise der Code und der Algorithmus.
- Überdeckungsgrad des Codes kann für vorhandene Testfälle gemessen werden.
- Weitere Testfälle können zur Erhöhung des Überdeckungsgrades systematisch abgeleitet werden.
- Während der Ausführung der Testfälle wird der innere Ablauf im Testobjekt analysiert (Point of Observation liegt innerhalb des Testobjekts).
- Eingriff in den Ablauf im Testobjekt möglich, z.B. wenn für Negativtests die zu provozierende Fehlbedienung über die Komponentenschnittstelle nicht auslösbar ist (Point of Control kann innerhalb des Testobjekts liegen).

Erfahrungsbasierte Testentwurfserfahren

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

Spezifikationsorientierter / Black-Box-Test: Begriffe

Funktionaler Test

- Dynamischer Test, bei dem die Testfälle unter Verwendung der funktionalen Spezifikation des Testobjekts hergeleitet werden und die Vollständigkeit der Prüfung (Überdeckungsgrad) anhand der funktionalen Spezifikation bewertet wird.

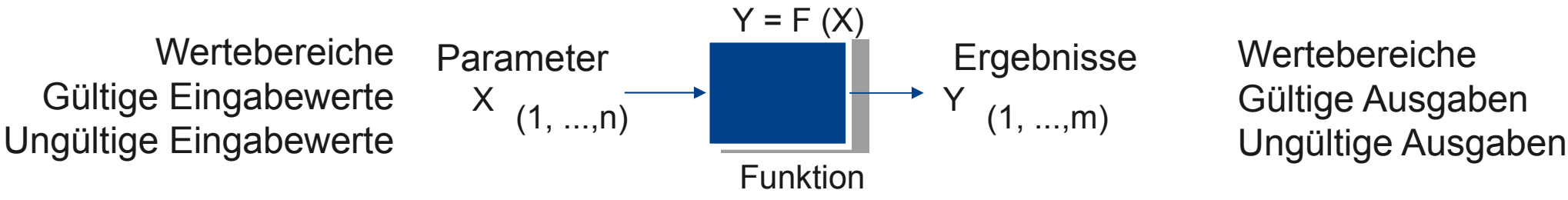
Funktionalität

- Die Fähigkeit eines Softwareprodukts beim Einsatz unter spezifizierten Bedingungen Funktionen zu liefern, die festgelegte und vorausgesetzte Erfordernisse erfüllen [ISO 9126].

  Untermerkmale der Funktionalität nach ISO 9126 sind:
  Angemessenheit, Richtigkeit, Interoperabilität, Sicherheit und 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

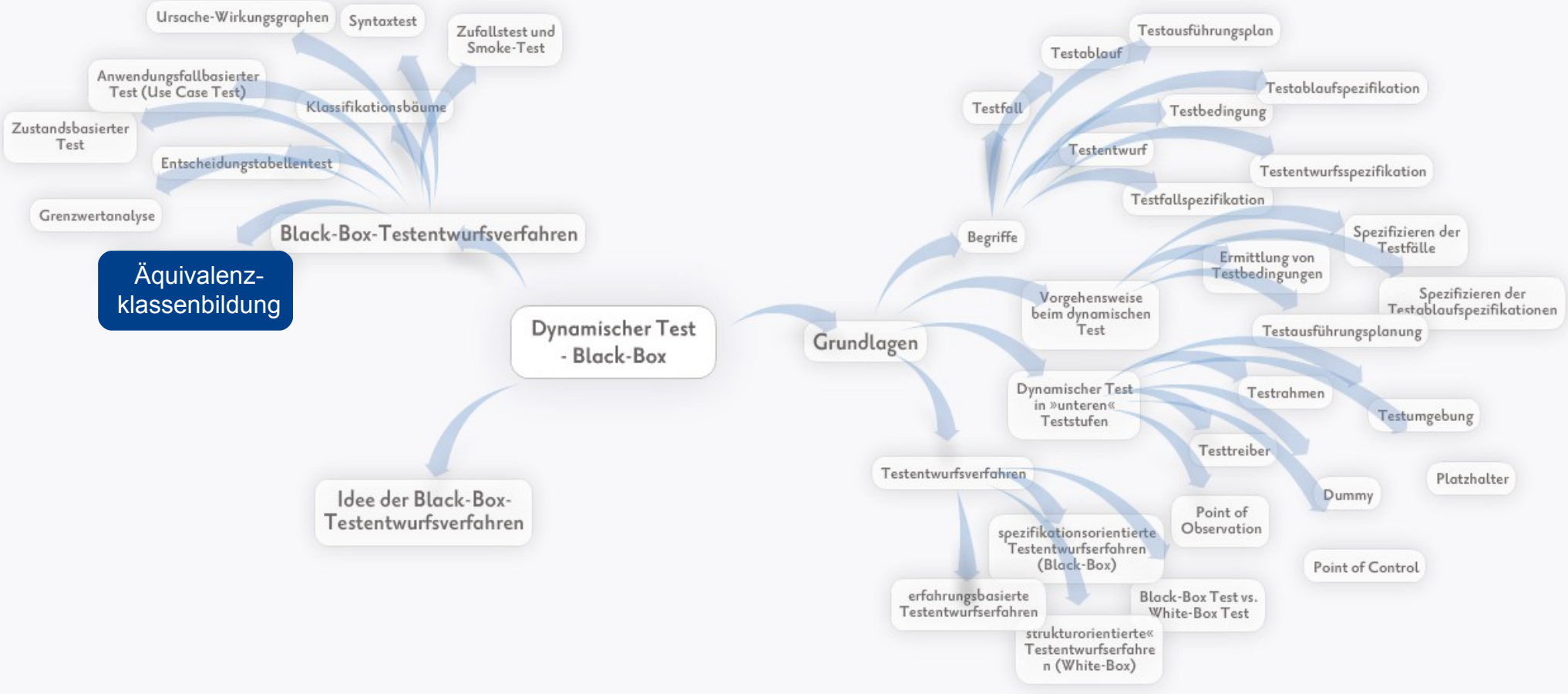
- Szenarien der Systemnutzung

2.4 Black-Box-Test

2.4 Black-Box-Test

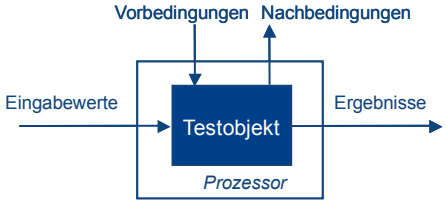
- Dynamischer Test – Grundlagen
- Idee der Black-Box-Testentwurfsverfahren
- **Äquivalenzklassenbildung**
- Grenzwertanalyse
- Zustandsbasierter Test
- Entscheidungstabellentest
- Weitere Black-Box-Testentwurfsverfahren

Dynamischer Test - Black-Box



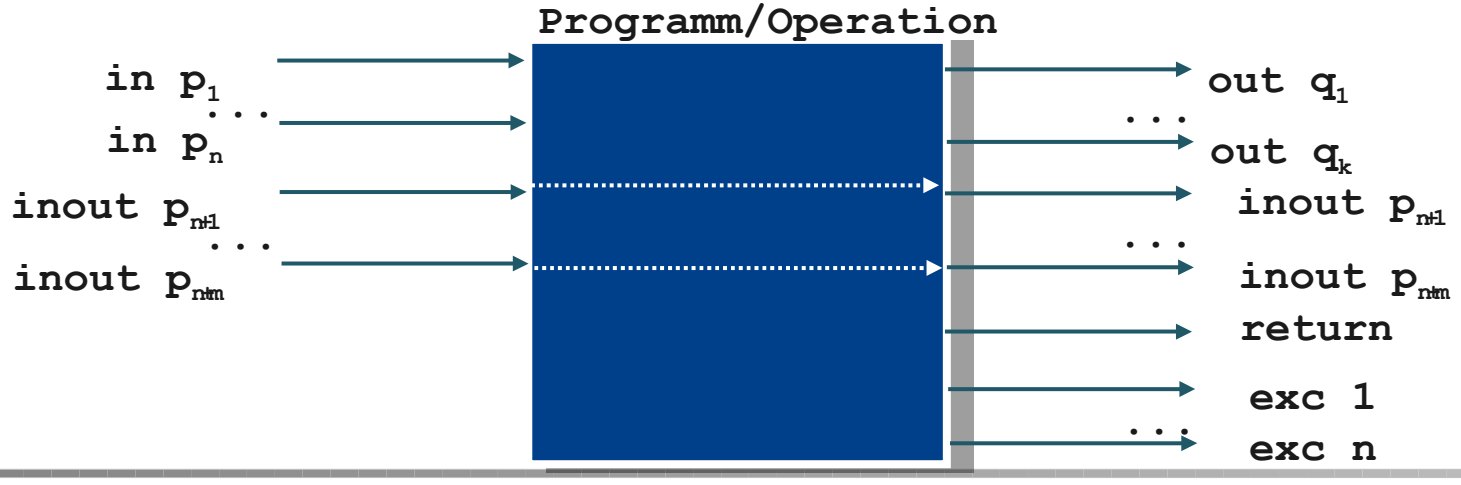
Programme und Operationen

Vereinfachende Annahme: Programme / Operationen berechnen Ausgaben aus Eingaben (zustandslos – frühere Ausführungen spielen keine Rolle).



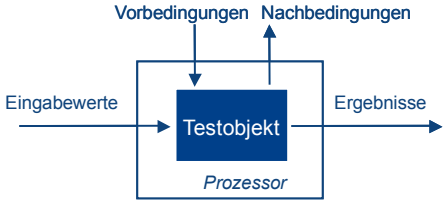
Signatur einer Operation:

- 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 (*greatest common divisor, gcd*):

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

Logische Spezifikation:

$ggT: Z \times Z \rightarrow IN$

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

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

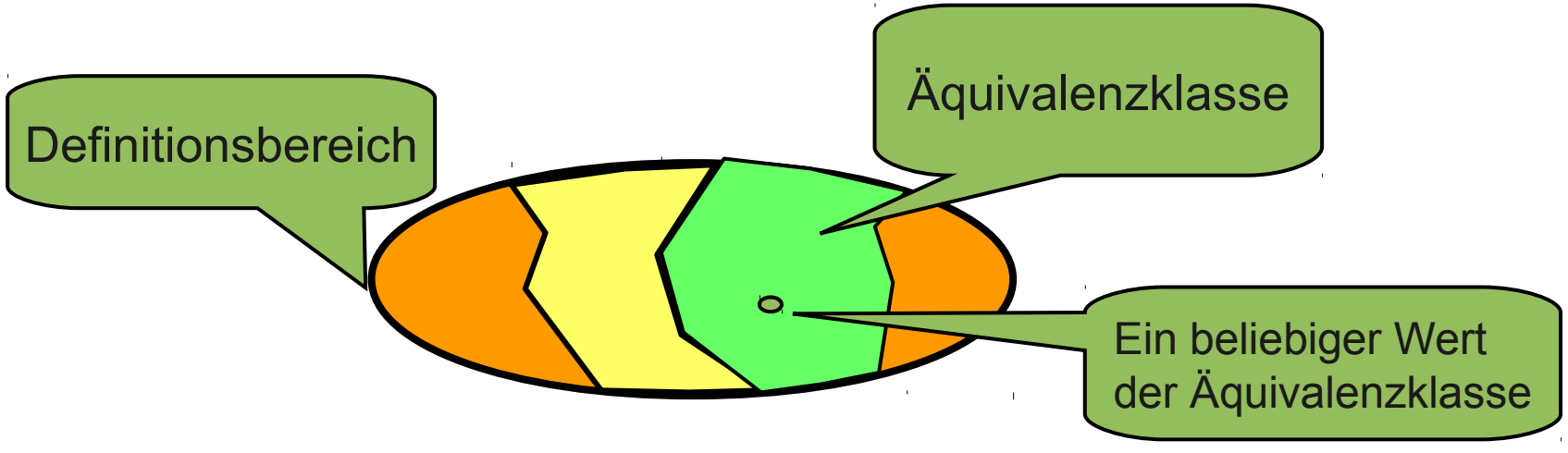
Spezifikation in UML / Java:

```
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

Die Definitionsbereiche der Ein- und Ausgaben werden so in **Äquivalenzklassen (ÄK)** zerlegt (partitioniert), dass alle Werte einer Klasse äquivalentes Verhalten des Prüflings ergeben sollten. Die Wahl nur eines Testwertes pro ÄK (Repräsentant) ergibt dann eine sinnvolle Stichprobe.

- Wenn ein Wert der ÄK einen Fehler aufdeckt, wird erhofft dass auch jeder andere Wert der ÄK diesen Fehler aufdeckt.
- Wenn ein Wert der ÄK keinen Fehler aufdeckt, wird erhofft, dass auch kein anderer Wert der ÄK einen Fehler aufdeckt.



Äquivalenzklassen für ggT: Erster Schritt

Definitionsbereiche der Ein- und Ausgaben

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

Gültige von ungültigen Teilbereichen der Java-Implementierung unterscheiden:

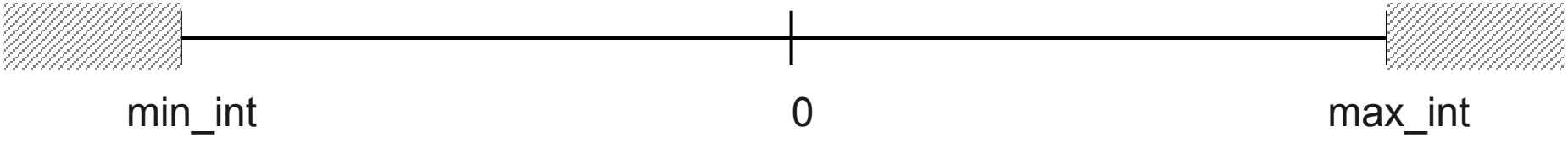
?

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 der Spezifikation (Ein- und Ausgaben !).

Äquivalenzklassenbildung für jede weitere Beschränkung:

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

Äquivalenzklassenbildung: Beispiel Wertebereich

Falls eine Beschränkung **einen Wertebereich** (Intervall) spezifiziert: 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 ?
Gültige Äquivalenzklasse ?
Ungültige Äquivalenzklasse ?

Äquivalenzklassenbildung: Beispiel Wertebereich

Falls eine Beschränkung **einen Wertebereich** (Intervall) spezifiziert: 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

Falls eine Beschränkung eine **minimale und maximale** Anzahl von Werten spezifiziert: 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 ?

Ungültige Äquivalenzklasse ?

Äquivalenzklassenbildung: Beispiel minimal / maximal

Falls eine Beschränkung eine **minimale und maximale** Anzahl von Werten spezifiziert: 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

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

Beispiel

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

Gültige Äquivalenzklasse: ?

Ungültige Äquivalenzklasse: ?

Äquivalenzklassenbildung: Beispiel Menge von Werten

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

Äquivalenzklassenbildung: Beispiel Situation

Falls eine Beschränkung **eine Situation** spezifiziert, die zwingend 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

Falls eine Beschränkung **eine Situation** spezifiziert, die zwingend 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).

Frage

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 ?

Frage

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:

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

Anzahl der Testfälle minimieren: Heuristiken

Testfälle aus allen Repräsentanten kombinieren und anschließend nach »Häufigkeit« sortieren (»**Benutzungsprofile**«).

- Testfälle dann in dieser Reihenfolge priorisieren.
- Nur mit »benutzungsrelevanten« Testfällen testen.
- Testfälle bevorzugen, die Grenzwerte oder Grenzwert-Kombinationen enthalten.

Sicherstellen, dass jeder Repräsentant einer Äquivalenzklasse mit jedem Repräsentanten jeder anderen Äquivalenzklasse in einem Testfall zur Ausführung kommt.

- D.h. paarweise Kombination statt vollständiger Kombination.

Minimalkriterium: Mindestens ein Repräsentant jeder Äquivalenzklasse in mindestens einem Testfall.

Repräsentanten ungültiger Äquivalenzklassen nicht mit Repräsentanten anderer ungültiger Äquivalenzklassen kombinieren.

Äquivalenzklassenbildung: Fehlermaskierung

Bei gleichzeitiger Behandlung verschiedener ungültiger ÄK bleiben 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 ?

Äquivalenzklassenbildung: Fehlermaskierung

Bei gleichzeitiger Behandlung verschiedener ungültiger ÄK bleiben 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)

Ein spezifisches Ausgangskriterium für den Test nach der Äquivalenzklassenbildung lässt sich anhand der durchgeführten Tests der Repräsentanten der jeweiligen Äquivalenzklassen im Verhältnis zur Gesamtzahl aller definierten Äquivalenzklassen festlegen:

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

Beispiel: Sind 18 Äquivalenzklassen aus den Anforderungen bzw. der Spezifikation für ein Eingabedatum ermittelt worden und sind von diesen 18 nur 15 in den Testfällen getestet worden, so ist eine **Äquivalenzklassen-Überdeckung** von ca. 83 % erreicht:

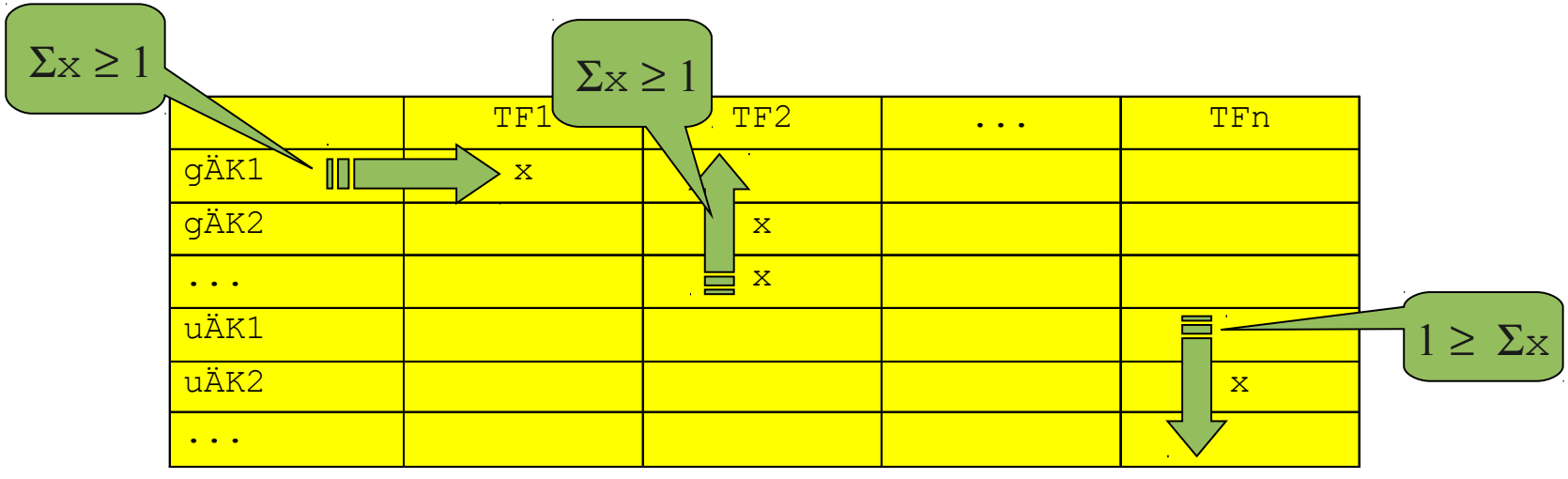
- ÄK-Überdeckung = 15 / 18 ≈ 83 %

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		
gÄK 2_1	x			x	x		
gÄK2 _2		x					
gÄK2 _3			x				
uÄK2 _1						x	
UÄK2 _2							x

Äquivalenzklassenbildung: Vor- und Nachteile

Vorteile:

- Anzahl der Testfälle kleiner als bei unsystematischer Fehlersuche.
- Geeignet für Programme mit vielen verschiedenen Ein- und Ausgabebedingungen.

Nachteile:

- Betrachtet Bedingungen für einzelne Ein- oder Ausgabeparameter.
- Beachtung von Wechselwirkungen und Abhängigkeiten von Bedingungen sehr aufwändig.

Empfehlung:

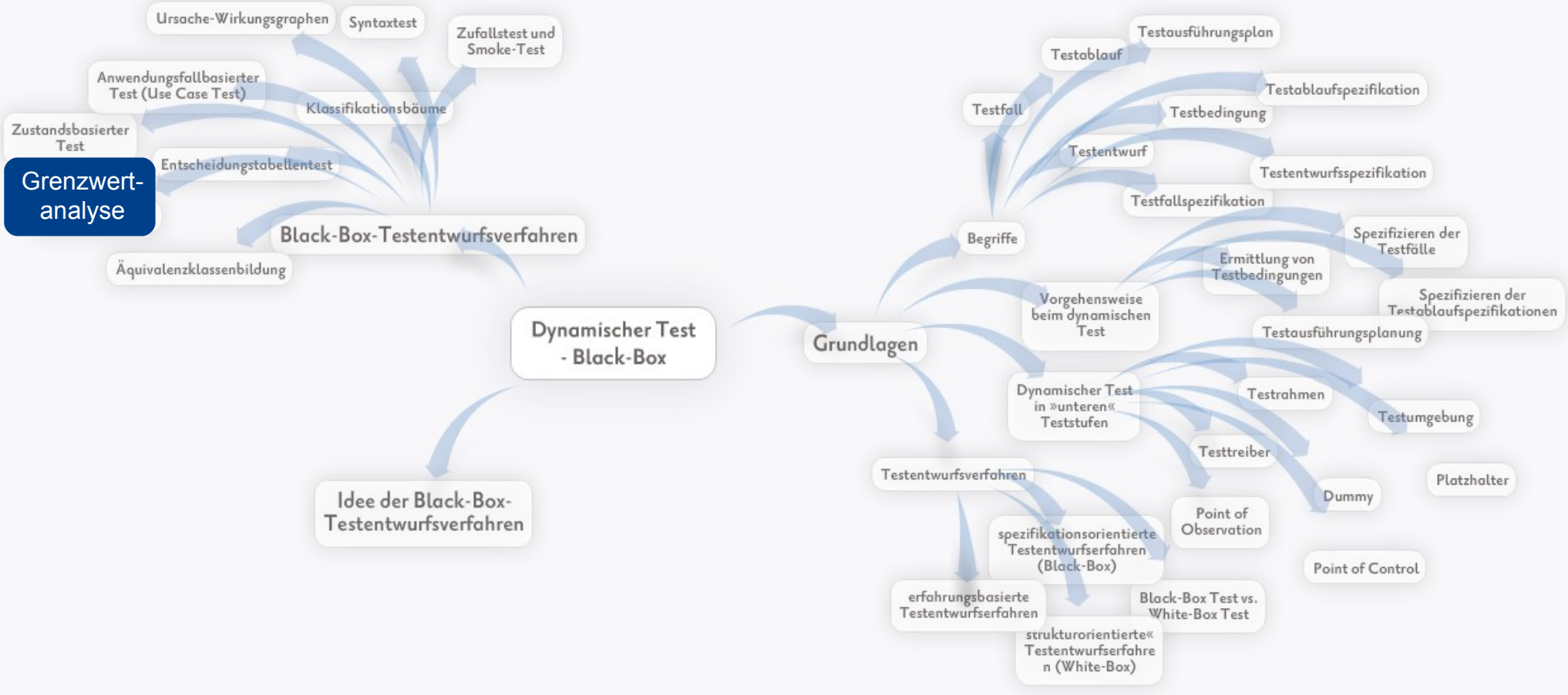
- Zur Auswahl wirkungsvoller Testdaten: Kombination der ÄK-Bildung mit fehlerorientierten Verfahren, z.B. Grenzwertanalyse.

2.4 Black-Box-Test

2.4 Black-Box-Test

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

Dynamischer Test - Black-Box



Grenzwertanalyse: Idee

Idee: In **Verzweigungs- und Schleifenbedingungen** gibt es oft **Grenzbereiche**, für welche die Bedingung gerade noch zutrifft (oder gerade nicht mehr).

- Solche Fallunterscheidungen sind **fehlerträchtig** (*off by one*).
- Testdaten, die solche Grenzwerte prüfen, decken Fehlerwirkungen mit höherer Wahrscheinlichkeit auf als Testdaten, die dies nicht tun.

Beste Erfolge bei Kombination mit anderen Verfahren.

Bei Kombination mit der **Äquivalenzklassenbildung**:

- **Grenzen der ÄK** (größte und kleinste Werte) testen.
- Jeder »Rand« einer ÄK muss in einer Testdatenkombination vorkommen.

Grenzwertanalyse: Ziel

Ziel: Auswahl von Werten aus einer Äquivalenzklasse bestehend aus einer geordneten Menge

Vorgehen:

- Schritt 1: Auswahl von Testwerten, die sich direkt oder neben den beiden Grenzen einer Eingabeäquivalenzklasse befinden
  - Äquivalenzklasse ist Wertebereich: Nimm größten und kleinsten Wert
  - Äquivalenzklasse ist Anzahl von Werten: Nimm größte und kleinste gültige Anzahl
- Schritt 2: Auswahl von Testwerten, die sich direkt oder neben den beiden Grenzen einer Ausgabeäquivalenzklasse befinden

Grenzwertanalyse: Anwendung

In der Regel werden der Grenzwert selbst sowie die Werte unmittelbar über bzw. unter dem Grenzwert getestet.

Atomare (geordnete) Bereiche (integer, real, char):

- Werte auf den Grenzen,
- Werte »rechts bzw. links neben« den Grenzen (ungültige Werte, kleiner bzw. größer als Grenze).

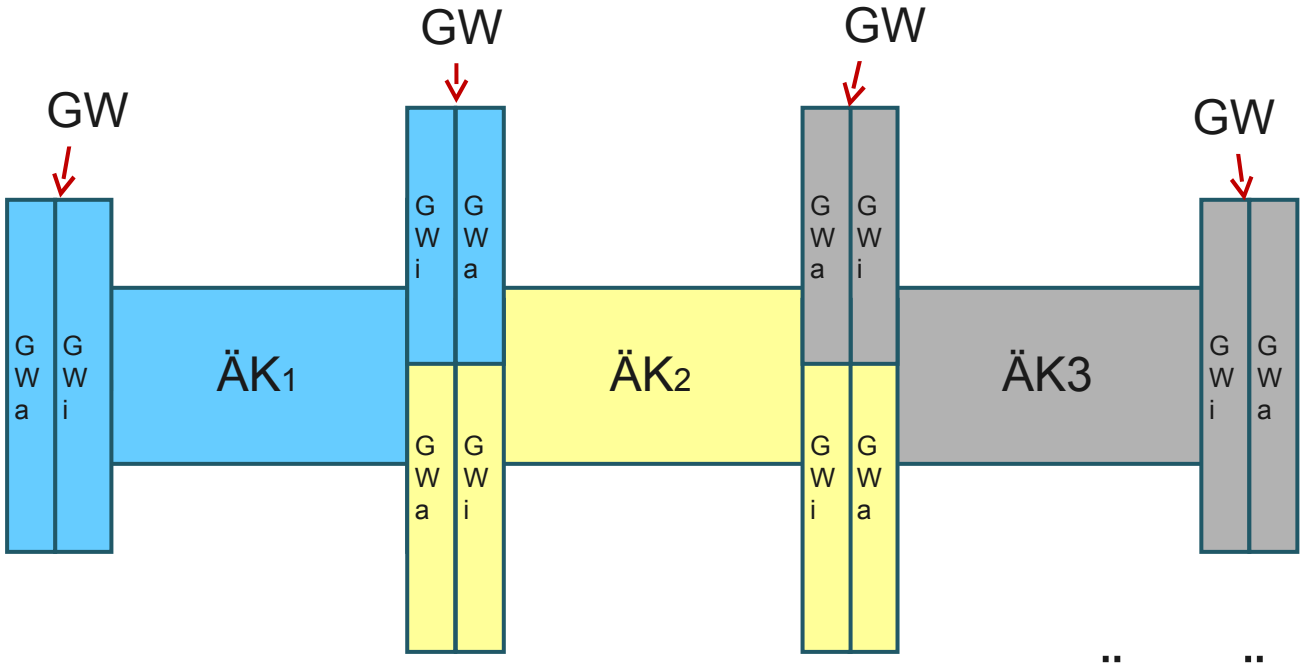
Mengenwertige Bereiche (z.B. bei Datenstrukturen, Beziehungen):

- 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.

Grenzwertanalyse: Innere / äußere Grenzwerte

Zusammenfallen der entsprechenden Grenzwerte **benachbarter Äquivalenzklassen**:



An den Grenzen immer zwei Tests, egal ob auf bzw. nur vor oder nach der Grenze getestet werden kann.

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

Grenzwertanalyse: Ausgangskriterium (Testendekriterium)

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

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** (z.B. sequentielle Datei, lineare Liste, Tabelle).
- **Komplexe Datenstrukturen: leere Mengen testen** (z.B. leere Liste, Null-Matrix).
- **Bei numerischen Berechnungen**: eng zusammen und weit auseinander liegende Werte wählen.

Grenzwertanalyse: Beispiel

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

Die Äquivalenzklassen 3-5 werden nicht untersucht, da es sich hierbei um keine geordneten Mengen handelt!

Grenzwertanalyse: Beispiel

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

Grenzwertanalyse: Beispiel

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

Grenzwertanalyse: Beispiel

Grenzwerte für Ausgaben:

- Es können X Seiten gedruckt werden, $1 \leq X \leq 20$
- Die letzte Seite enthält Y Zeilen, $1 \leq Y \leq 45$
- Die 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 den Grenzen von Äquivalenzklassen sind häufiger Fehler zu finden als innerhalb dieser Klassen.
- »Die Grenzwertanalyse ist bei richtiger Anwendung eine der nützlichsten Methoden für den Testfallentwurf.« Myers, Glenford J.: Methodisches Testen von Programmen, Oldenbourg, 2001 (7. Auflage)
- Effiziente Kombination mit anderen Verfahren, die Freiheitsgrade in der Wahl der Testdaten lassen.

Nachteile:

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