9. Was ist ISTQB®?

- International Software Testing Qualifications Board (ISTQB)
- Gegründet 2002 in Edinburgh
- Dachverband regionaler Fachgremien wie dem German Testing Board (GTB)

  German Testing Board: http://www.german-testing-board.info

- Verantwortlich für
  - Erstellung der Lehrinhalte und Lehrpläne
  - Akkreditierung der Trainingsanbieter
  - Erstellung der Prüfungsfragen
  - Autorisierung der Prüfstellen (Zertifizierungsstellen)
  - Umsetzung der Verfahren und Prozesse

9. Der ISTQB® Certified Tester

- Weltweit anerkannte Ausbildung
- Weltweit über 200.000 Certified Tester (Stand 03/2012)
- Training nur durch akkreditierte Trainingsanbieter
- Zertifizierung nur durch autorisierte Zertifizierungsstellen

Cert-IT GmbH	www.cert-it.com
gasq Service GmbH	de.gasq.org
International Software Quality Institute GmbH	www.isqi.org

Fehlerbeispiel 1: Ariane 5

Am **4. Juni 1996** startete die Ariane 5 zu ihrem Erstflug.

Nach genau 36,7 Sekunden sprengte sich die Rakete.

Es stellte sich heraus, dass die in Teilen von der Ariane 4 übernommene Software nicht den nötigen Anforderungen entsprach. Die Ariane 5 beschleunigt schneller als die Ariane 4. Dies führte zu einem Überlauf einer Variablen und zu einem Absturz des Lenksystems, das nur noch Statusdaten an den Navigationscomputer sendete. Dieser interpretierte die Daten als echte Fluglage und ließ die Schubdüsen der Booster bis zum Anschlag schwenken. Die Rakete begann auseinanderzubrechen und das bordeigene Neutralisationssystem löste die Selbstzerstörung aus, bevor die Bodenkontrolle eingreifen konnte.

Unglücklich daran war, dass dieser Teil der Software für die Ariane 5 nicht notwendig war und nur zur Beherrschung eines Startabbruchs in letzter Sekunde bei der Ariane 4 diente.

Übung 1: „Fehler“ im täglichen Umfeld

Welche Beispiele für “Fehler“ haben Sie?

Fehlerbegriffe [nach ISTQB_DE]

„Er hat einen ‚Fehler' gemacht!"

Fehlhandlung

„Da ist ein ‚Fehler' im Code!"

Fehlerzustand

„Es ist ein ‚Fehler' aufgetreten!"

Fehlerwirkung

Fehlhandlung (engl. *error, mistake*)
Die menschliche Handlung, die zu einem falschen Ergebnis führt [nach IEEE 610].

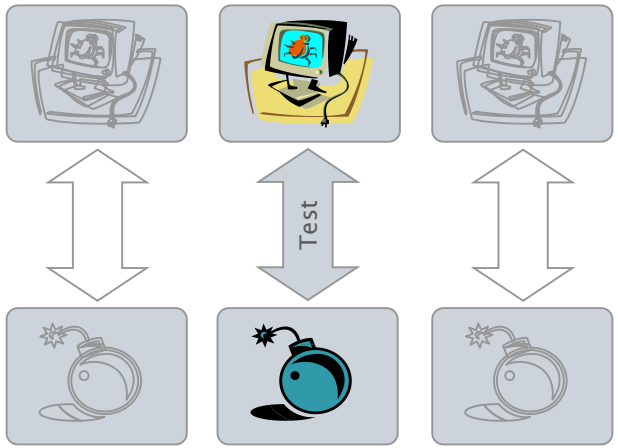
→ Kann einen Fehlerzustand verursachen

Fehlerzustand (innerer Fehlerzustand) (engl. *defect, bug*)
Defekt (innerer Fehlerzustand) in einer Komponente oder einem System, der eine geforderte Funktion des Produkts beeinträchtigen kann

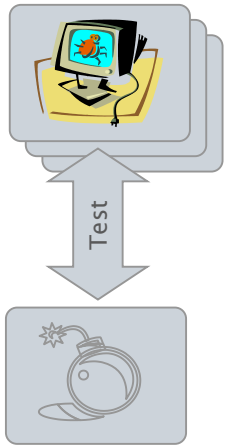
→ Kann eine Fehlerwirkung verursachen

Fehlerwirkung (äußerer Fehler) (engl. *failure*)
Abweichung einer Komponente/eines Systems von der erwarteten Lieferung, Leistung oder dem Ergebnis

Keine Fehlerwirkung bedeutet nicht fehlerfrei!



Unbekannter Fehlerzustand
Fehlerzustand der aufgrund eines spezifischen Tests nicht als Fehlerwirkung erkannt wird



Fehlermaskierung (engl. *defect masking*)
Ein Umstand bei dem ein Fehlerzustand die Aufdeckung eines anderen verhindert [nach IEEE 610]

Ursachen für Fehlerzustände I

- Auch Umgebungsbedingungen können das Produkt beeinflussen:
  - Strahlung, Elektromagnetische Felder
  - Schmutz
  - Temperatur

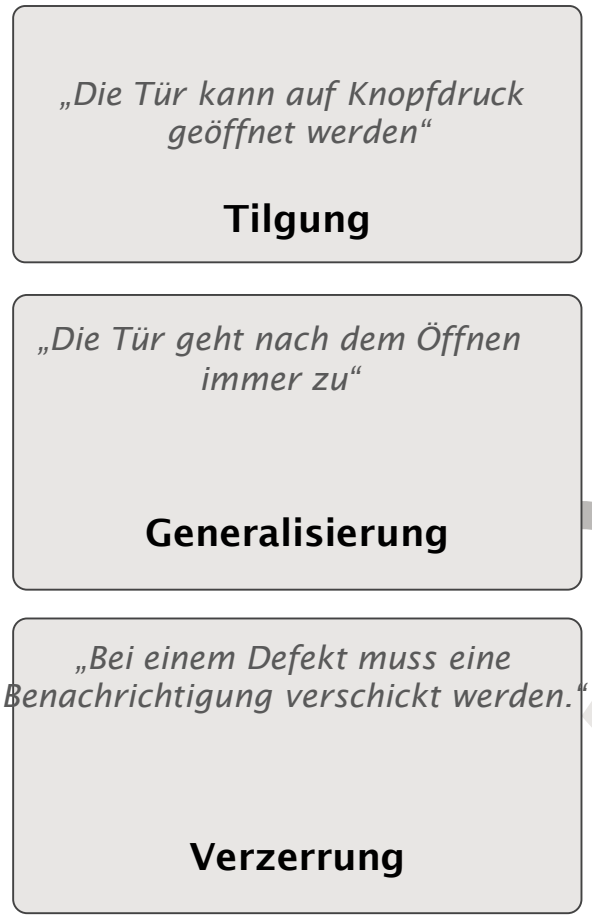
Wenn Umgebungseinflüsse nicht bedacht werden ist auch das eine Fehlhandlung!

- Diese Beeinflussungen können zu Fehlerzuständen im Produkt führen:
  - Datenverlust
  - Kurzschluss
  - Timingverletzungen

→ **Fehlerzustände können auch durch Umgebungsbedingungen hervorgerufen werden!**

1. Einführung - Motivation

Die häufigsten sprachlichen Defekte in Anforderungsspezifikationen. [Rupp],[Bandler]



Tilgung

Informationen bei der Wissensdarstellung werden vernichtet und die Welt wird auf Ausmaße reduziert, mit denen wir umgehen können

→ *Durch wen wird die Tür geöffnet?*

Generalisierung

Die Übertragung gewonnener Erfahrungen auf verwandte Zusammenhänge übertragen

→ *Geht die Tür wirklich immer nach dem Öffnen zu?*

Verzerrung

Tritt häufig in Form von Nominalisierungen auf. Dabei wird ein Prozess (Verb) in ein Ereignis (Substantiv oder Argument) umgeformt

→ *Was verbirgt sich hinter der „Benachrichtigung“?*

Ursachen für Fehlerzustände II

Der Kunde beschreibt seine **Realität** (seine Erwartungen) durch einen **sprachlichen Ausdruck** (z.B. Anforderungen)

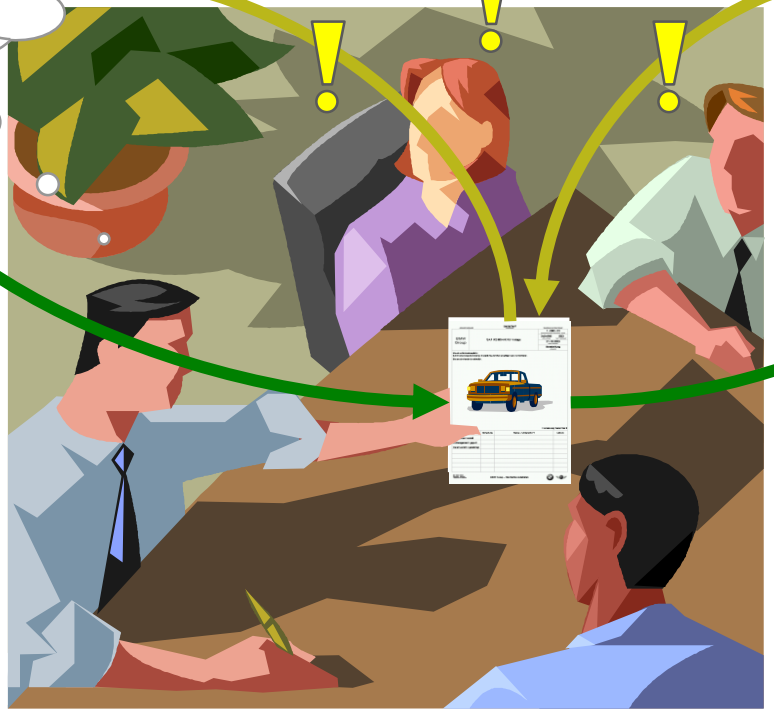
Der Lieferant entwickelt seine **Realität** (das Produkt) anhand eines **sprachlichen Ausdrucks** (z.B. Anforderungen)

Die Anforderung ist nicht die Erwartung!
Das Produkt ist nicht die Anforderung!

Maßnahmen zur Fehlerzustandsidentifikation

Validieren, ob das reale Produkt den **Erwartungen** des Kunden entspricht

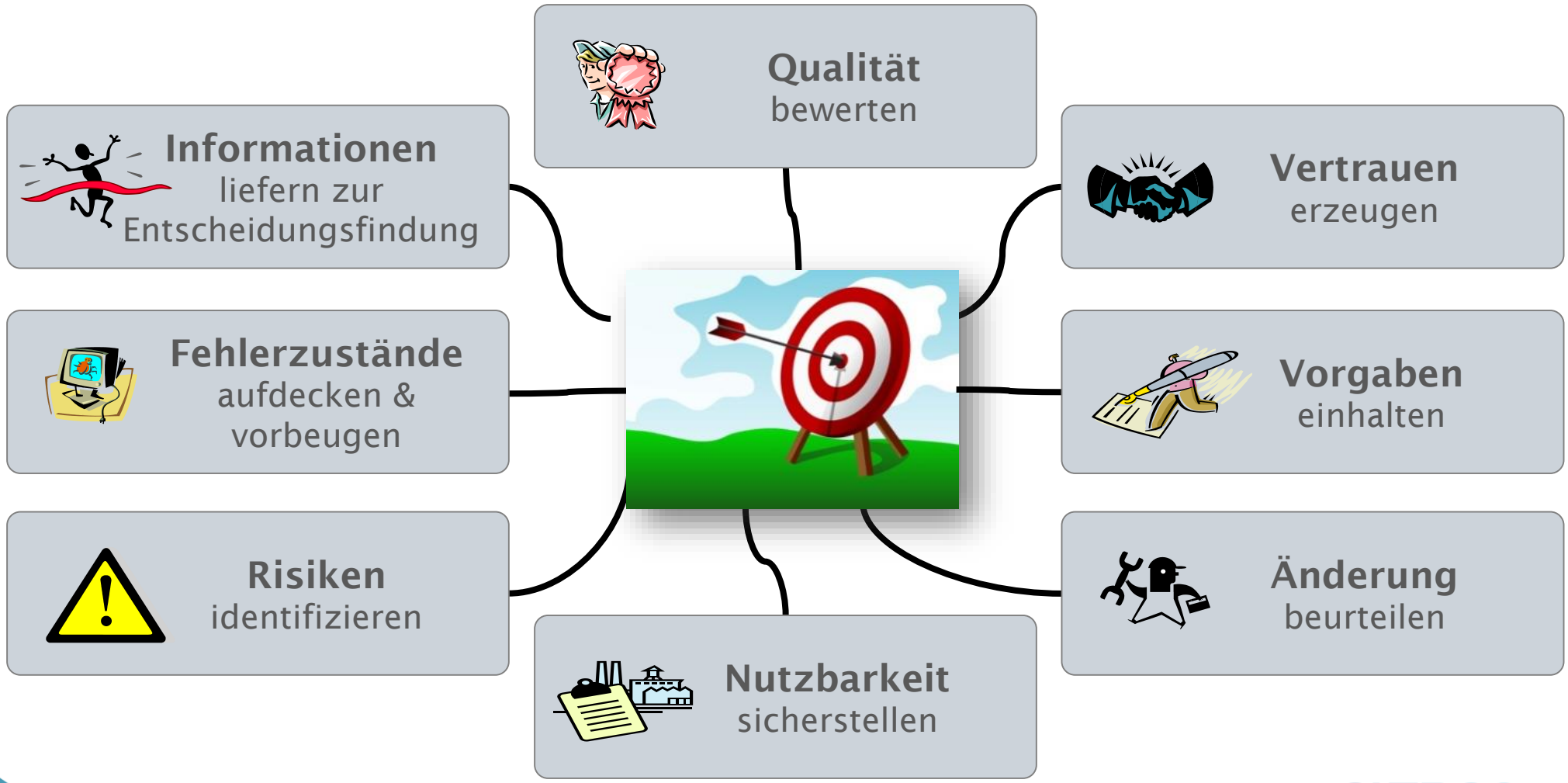
Validieren, ob die Anforderungen die **Erwartungen** des Kunden beschreiben



Verifizieren, ob das reale Produkt die **Anforderungen** erfüllt

Verifizieren: Das Produkt ist richtig entwickelt!
Validieren: Es ist das richtige Produkt entwickelt!

Testziele (Übersicht): Warum Testen?



Qualitätsbegriff

Qualität

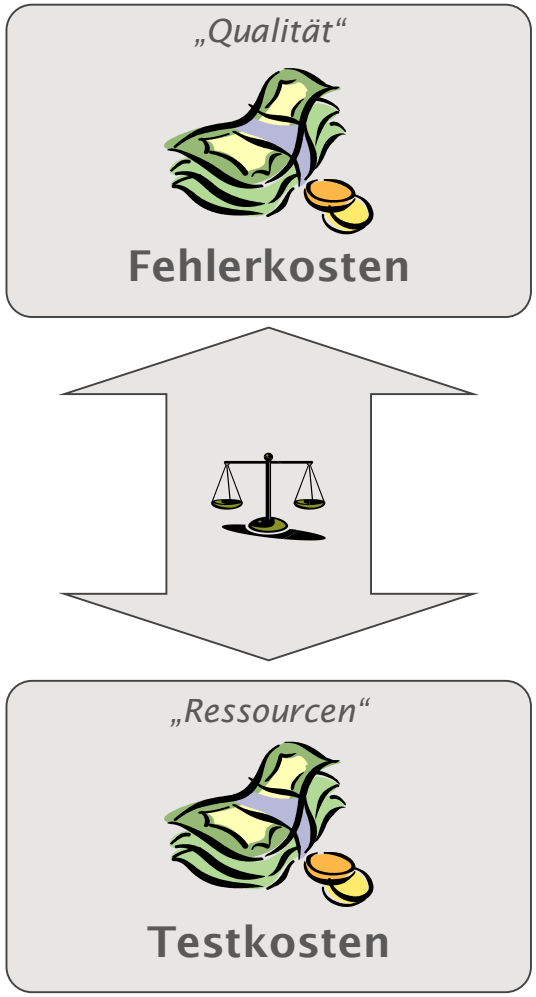
Inhärenten Merkmale (auch: innewohnende Merkmale)
Unter „inhärenten Merkmalen“ sind **objektiv messbare Merkmale** zu verstehen.
Durch Definition von Zielgruppen und Meinungsumfragen kann auch das subjektive Empfinden objektiv messbar gemacht werden!

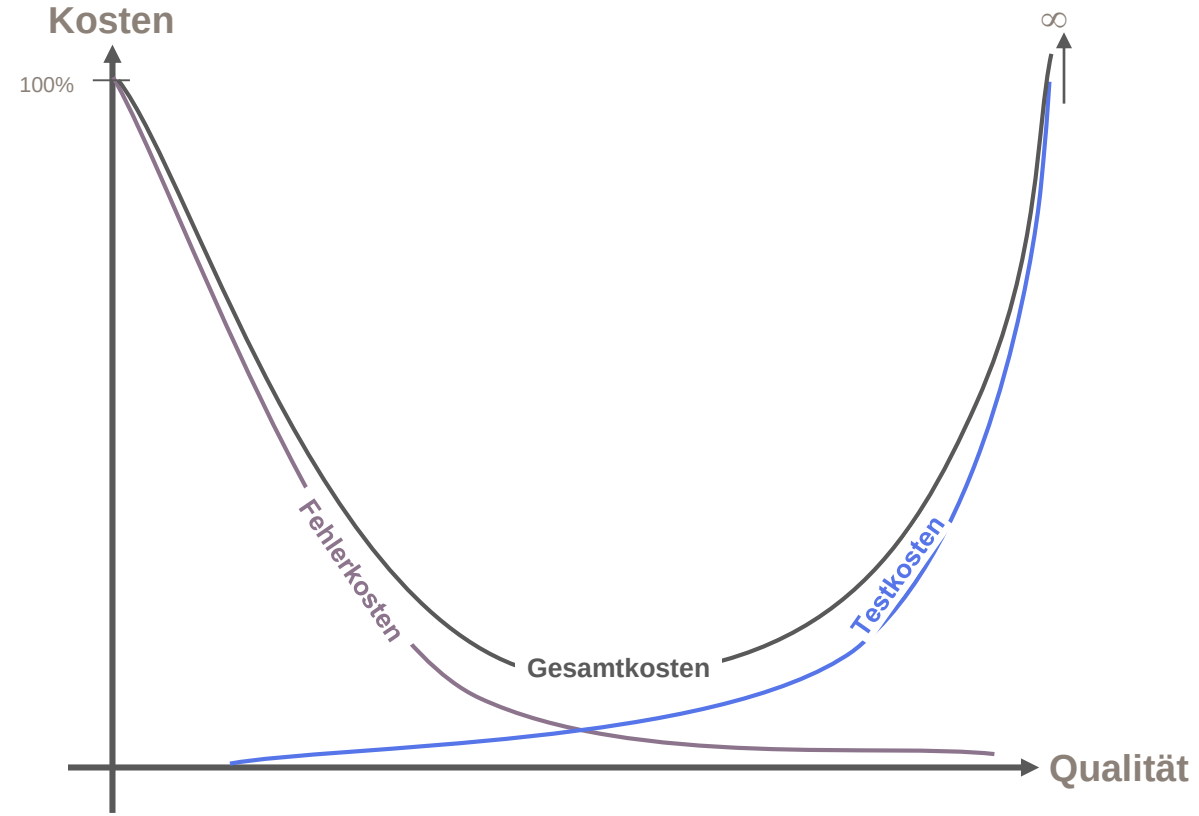
Qualität (engl. quality)

1. „Qualität ist der Grad, in dem ein Satz inhärenter Merkmale Anforderungen erfüllt “
   [ISO 9000]
2. „Der Grad, in dem ein System, eine Komponente oder ein Prozess die Kundenerwartungen und -bedürfnisse erfüllt“
   [nach IEEE 610]
3. „Qualität ist die Gesamtheit von Merkmalen einer Einheit bezüglich ihrer Eignung, festgelegte und vorausgesetzte Erfordernisse zu erfüllen.“
   [ISO 9000]

Qualitätsmerkmale können der [ISO 25000] entnommen werden (→ Kapitel 3.2)

Testen und Kosten





Je höher die Qualitätsziele um so überproportional höher sind die Testkosten!

Wie viel Testen ist genug?

Testen ist dann genug, wenn:

- Ausreichend Informationen vorliegen
  um fundierte Entscheidungen über die Freigabe oder Abnahme des Produktes treffen zu können

→ Es sind Metriken notwendig um die gewonnen Informationen (Testergebnisse) bewerten zu können

→ Aufgrund von Testendekriterien wird das Testen beendet

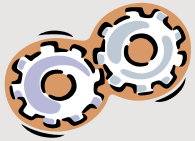
- Vertragliche Vorgaben, gesetzliche Vorgaben bzw. Normen erfüllt sind

Testverfahren

„Review & statische Analyse“

statische Tests

„Device under Test“

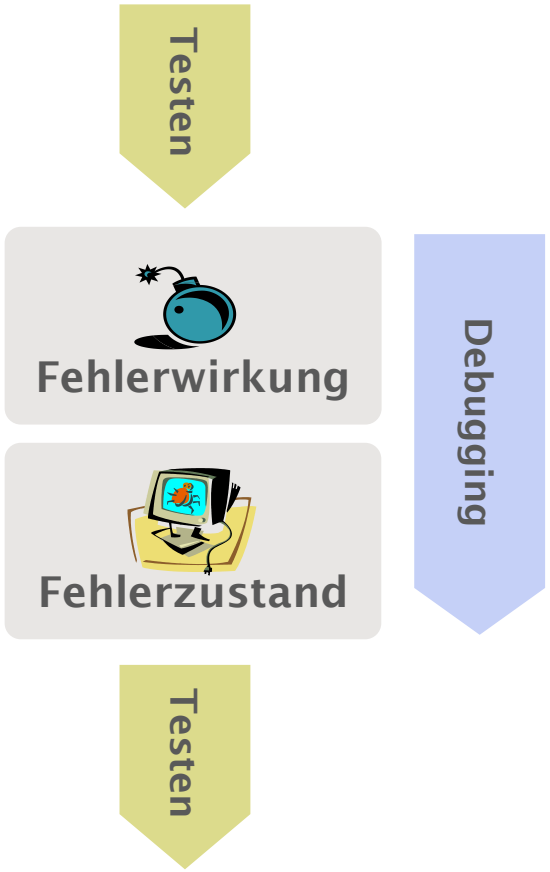


dynamische Tests

- ✓ Testobjekt (z.B. Anforderungen) wird analysiert bzw. **gedanklich ausgeführt**
  - Review/Inspektion
  - Codeanalyse
  - Korrektheitsbeweis (z.B. Durchgängigkeit)
  - Symbolische Tests (mathematischer Beweis)
- ✓ **Reales Ausführen** bzw. Betreiben des Testobjektes (z.B. HW, SW)
  - Blackbox, Whitebox
  - Abnahmetest
  - In-Circuit-Test (ICT)
  - Simulationen

Auch das Prüfen der Testbasis kann Fehlerzustände im Produkt verhindern!

Testen und Debugging



→ Testen **kann** Fehlerwirkungen zeigen, die durch Fehlerzustände bzw. Fehlhandlungen verursacht werden

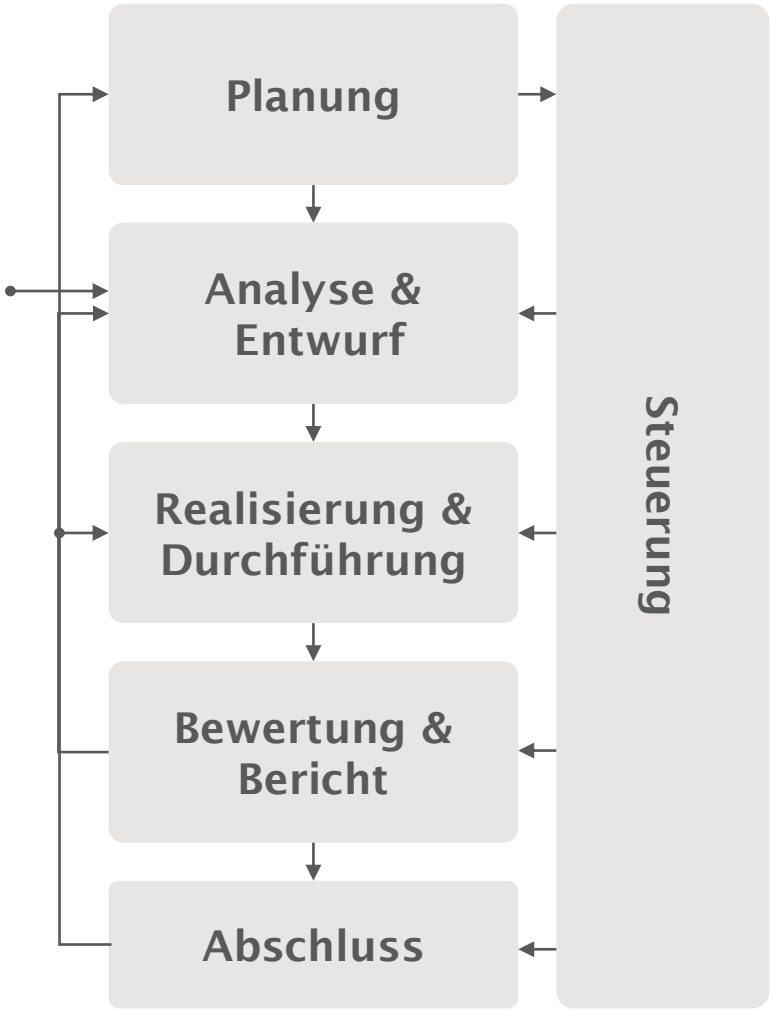
→ Debugging ist eine Entwicklungsaktivität, die

- den Fehlerzustand einer Fehlerwirkung identifiziert
- den Code korrigiert und überprüft, ob der Fehlerzustand korrekt behoben wurde

→ Debugging erfolgt durch den Entwickler!

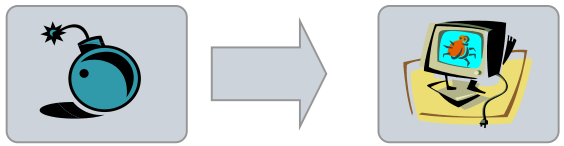
→ Anschließende Fehlernachtests stellen sicher, ob die Korrektur die Fehlerwirkung behoben hat

Was gehört zum Testen?

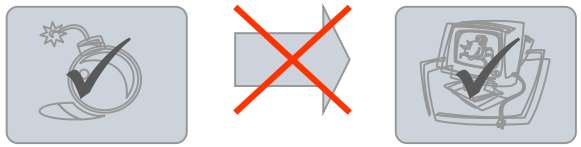


- Planen aller Testaktivitäten und Ressourcen
- Analysieren der Testbasis und Spezifizieren der geplanten Testfälle
- Durchführen der spezifizierten Tests
- Bewertung von Ausgangskriterien und Testabschlussbericht der Testergebnisse
- Abschließen der Testarbeiten
- Steuern der Testaktivitäten

§1 Testen zeigt die Anwesenheit von Fehlerzuständen – Nicht das Fehlen dieser!



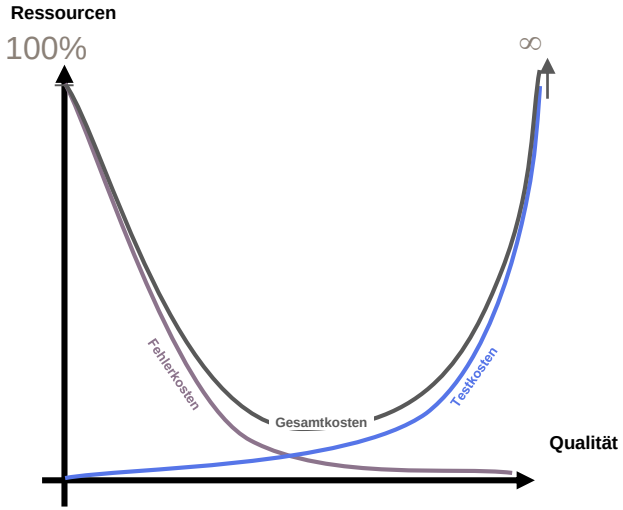
- ✓ „Mit Testen wird das Vorhandensein von Fehlerzuständen nachgewiesen.“
- ✓ „Ausreichendes Testen verringert die Wahrscheinlichkeit, dass noch unentdeckte Fehlerzustände im Testobjekt vorhanden sind.“



- ✘ „Selbst wenn keine Fehlerzustände im Test aufgezeigt wurden, ist dies kein Nachweis für Fehlerfreiheit.“
- ✘ „Mit Testen lässt sich nicht beweisen, dass keine Fehlerzustände im Testobjekt vorhanden sind.“

Aus: ISTQB, Deutsche Ausgabe des GTB, „Certified Tester, Foundation Level Syllabus“, Stand 2010

§2 Erschöpfendes (vollständiges) Testen ist nicht möglich!



✘ „Ein erschöpfender Test, bei dem alle möglichen Eingabewerte und deren Kombinationen unter Berücksichtigung aller unterschiedlichen Vorbedingungen ausgeführt werden, ist nicht durchführbar, mit Ausnahme von sehr trivialen Testobjekten.“

✓ „Tests sind immer nur Stichproben, und der Testaufwand ist entsprechend Risiko und Priorität festzulegen“

Aus: ISTQB, Deutsche Ausgabe des GTB, „Certified Tester, Foundation Level Syllabus“, Stand 2010

§3 Mit dem Testen frühzeitig beginnen!

- ✓ „Um Fehlerzustände frühzeitig zu finden sollen Testaktivitäten im System- oder Softwarelebenszyklus so früh wie möglich beginnen und definierte Ziele verfolgen. “

Aus: ISTQB, Deutsche Ausgabe des GTB, „Certified Tester, Foundation Level Syllabus“, Stand 2010

§4 Häufung von Fehlern!

- ✓ Der Testaufwand soll sich proportional zu der erwarteten und später beobachteten Fehlerdichte auf die Module fokussieren
- ✓ Ein kleiner Teil der Module enthält gewöhnlich die meisten Fehlerzustände, die während der Testphase entdeckt werden oder ist für die meisten Fehlerwirkungen im Betrieb verantwortlich

Aus: ISTQB, Deutsche Ausgabe des GTB, „Certified Tester, Foundation Level Syllabus“, Stand 2010

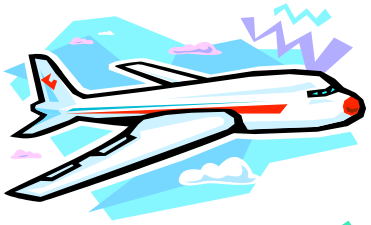
§5 Wiederholungen haben keine Wirksamkeit!

- ✓ Wiederholungen der immer gleichen Testfälle führen zu keinen neuen Erkenntnissen

- → Damit die Effektivität der Tests nicht absinkt, sind die Testfälle regelmäßig zu prüfen und neue oder modifizierte Testfälle zu erstellen
  (Bisher nicht geprüfte Teile der Software oder unberücksichtigte Konstellationen bei der Eingabe werden dann ausgeführt und somit mögliche weitere Fehlerzustände nachgewiesen.)

Aus: ISTQB, Deutsche Ausgabe des GTB, „Certified Tester, Foundation Level Syllabus“, Stand 2010

§6 Testen ist abhängig vom Umfeld!



- ✓ „Je nach Einsatzgebiet und Umfeld des zu prüfenden Systems ist das Testen anzupassen.“
- ✓ „Sicherheitskritische Systeme werden beispielsweise anders getestet als E-Commerce-Systeme.“

Aus: ISTQB, Deutsche Ausgabe des GTB, „Certified Tester, Foundation Level Syllabus“, Stand 2010

§7 Trugschluss: Keine Fehlerzustände = Brauchbares System.

- ✓ „Fehlerzustände zu finden und zu beseitigen hilft nicht, wenn das gebaute System nicht nutzbar ist und den Vorstellungen und Erwartungen der Nutzer nicht entspricht.“

Aus: ISTQB, Deutsche Ausgabe des GTB, „Certified Tester, Foundation Level Syllabus“, Stand 2010

Rollenkonflikt zwischen Tester und Entwickler

Wirksame Tests müssen

- destruktiv sein und
- systematisch ablaufen

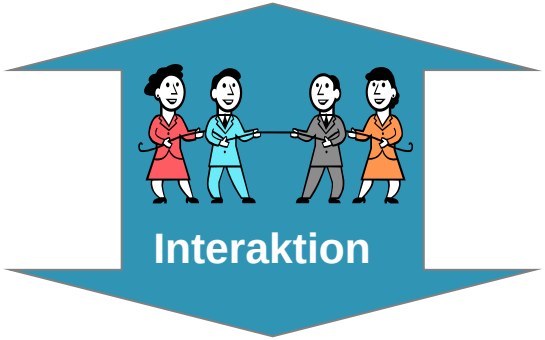
→ **Tester sollen Fehlerwirkungen finden**!

Tests können vom Entwickler als kontraproduktiv wahrgenommen werden!

Entwickler testen mit dem Fokus auf Ihre implementierte Funktion (Betriebsblindheit)

→ **Entwickler sollen die Funktion liefern**!

Aspekte der Gruppendynamik



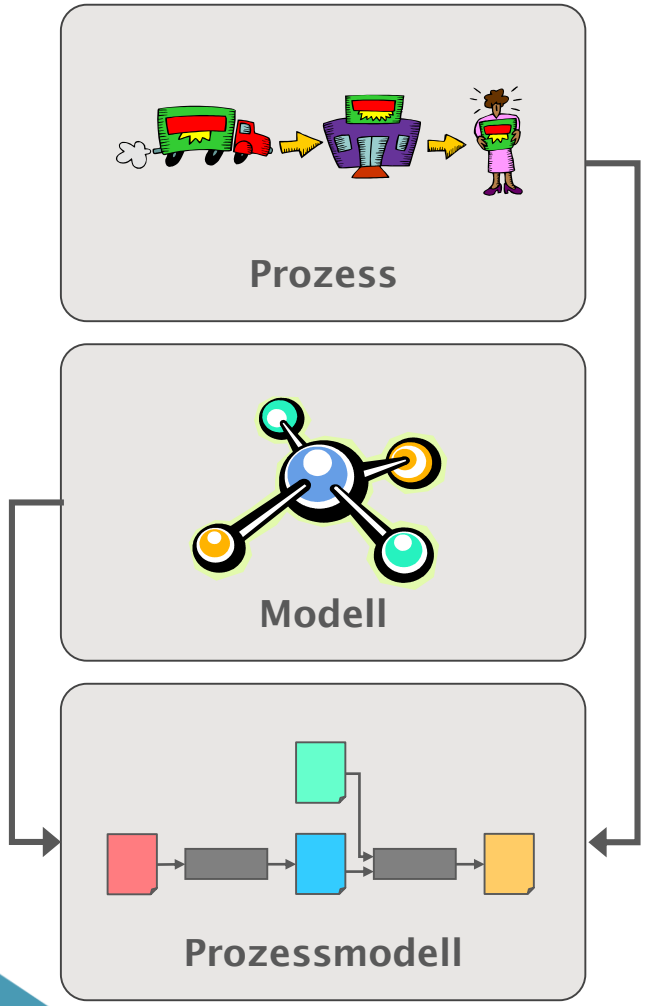
Interaktion

Es entsteht ein Gruppendynamischer Prozess, in dem die Gruppe ihre Struktur bildet und ggf. verändert

Kooperation (Zusammenarbeit)

Es können synergetische Effekte erreicht werden. „Gesamtleistung ist höher als die Summe der Einzelleistung.“

Definitionen

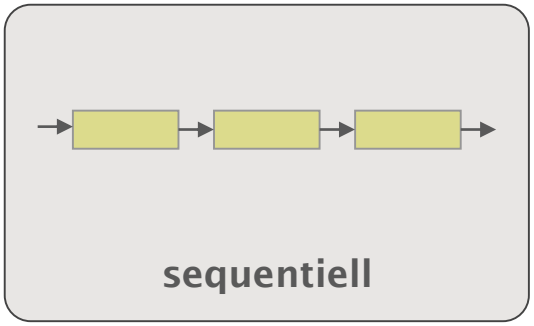


Prozess (Ablauf, Vorgang) (engl. *process*)
Ein Prozess ist „ein Satz von in Wechselbeziehungen stehenden Mitteln und **Tätigkeiten, die Eingaben in Ergebnisse umgestalten**“. [DIN EN ISO 8402]

Modell (Vorbild, Muster) (engl. *model*)
Ein Modell zeichnet sich durch die bewusste Vernachlässigung bestimmter Merkmale aus, um die für den Modellierer oder den Modellierungszweck **wesentlichen Modelleigenschaften** hervorzuheben

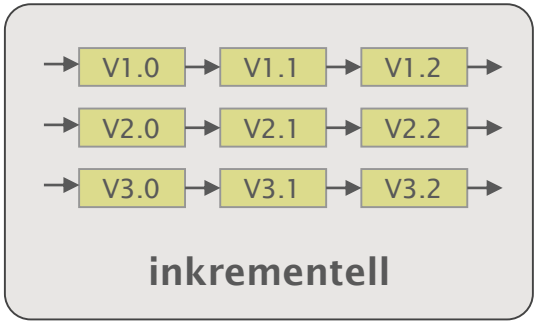
Prozessmodell (Vorgehensmodell) (engl. *process model*)
Aufgabe eines Vorgehensmodells ist es, die allgemein in einem Gestaltungsprozess auftretenden Aufgaben-stellungen und Aktivitäten in einer **sinnfälligen logischen Ordnung darzustellen**

Übersicht über Vorgehensmodellarten I



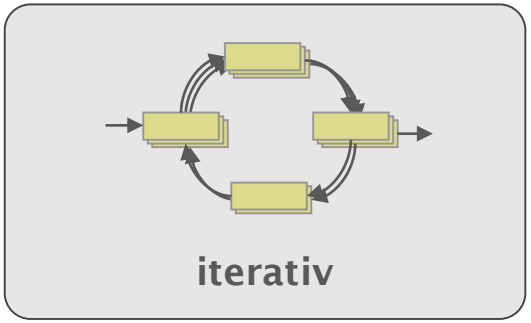
Sequentiell (Aufeinanderfolge, Reihenfolge) (lat. *sequi*)

Das sequentielle Vorgehensmodell beschreibt die lineare Aufreihung von Aktivitäten mit einer ausgewiesenen Richtung, die Vorgänger und Nachfolger kennzeichnet



Inkrementell

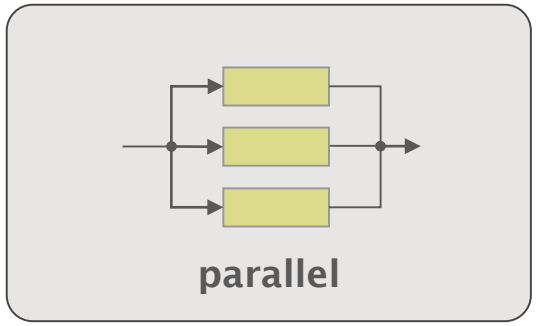
Das inkrementelle Vorgehensmodell beschreibt einen Prozess der kontinuierlichen Verbesserung, welcher häufig in kleinen oder sogar kleinsten Schritten vollzogen wird



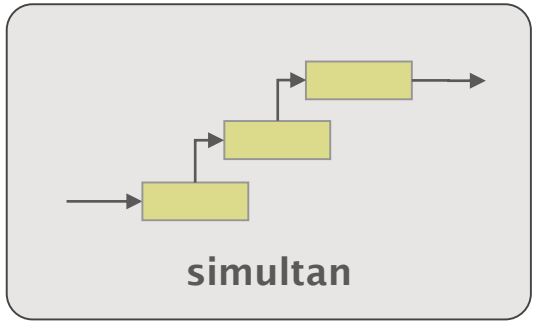
Iterativ (wiederholen) (lat. *iterare*)

Die Ergebnisse einer Iteration werden als Eingangsgrössen der jeweils nächsten Iteration genommen – bis das Ergebnis zufriedenstellt

Übersicht über Vorgehensmodellarten II

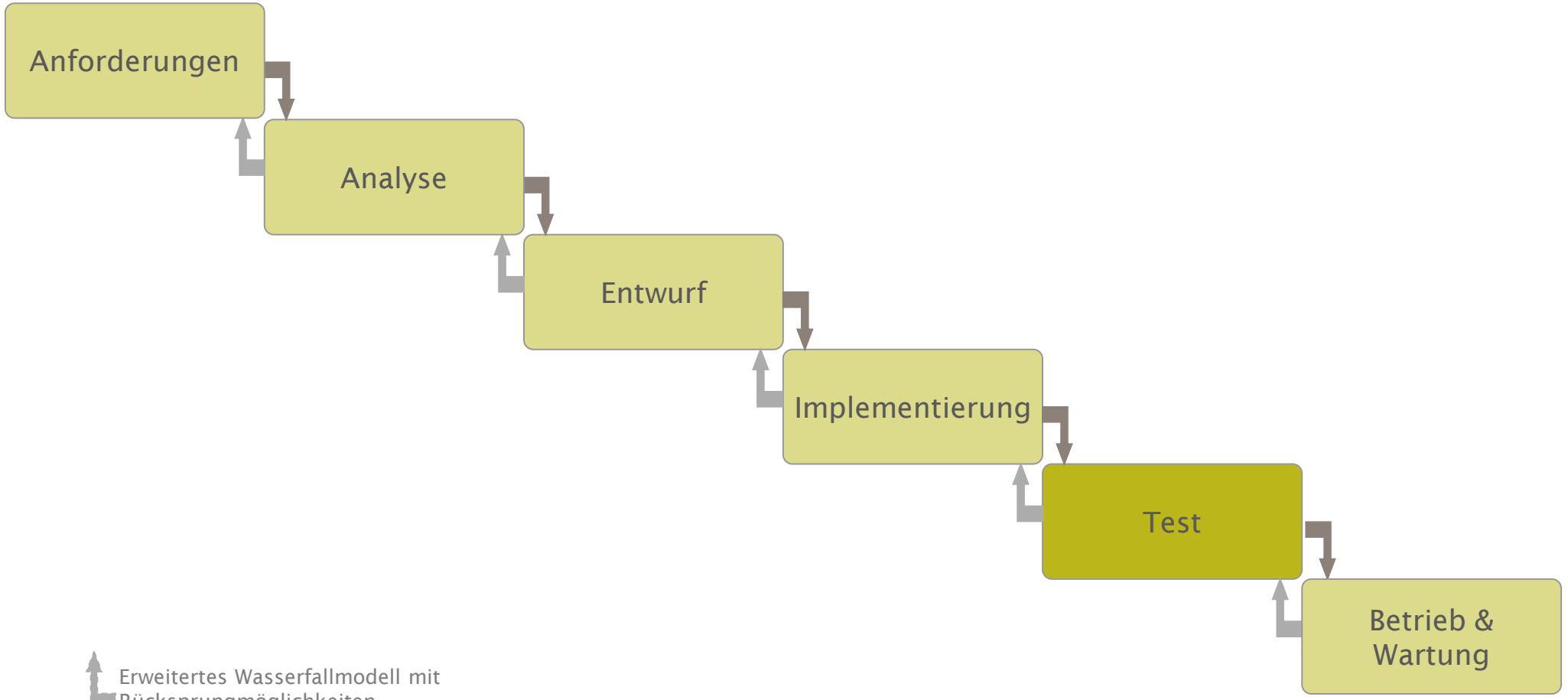


Parallel (neben, gleichzeitig) (griech. *pará*)
Das parallele Vorgehensmodell beschreibt zeitlich gleichzeitig durchgeführte Aktivitäten



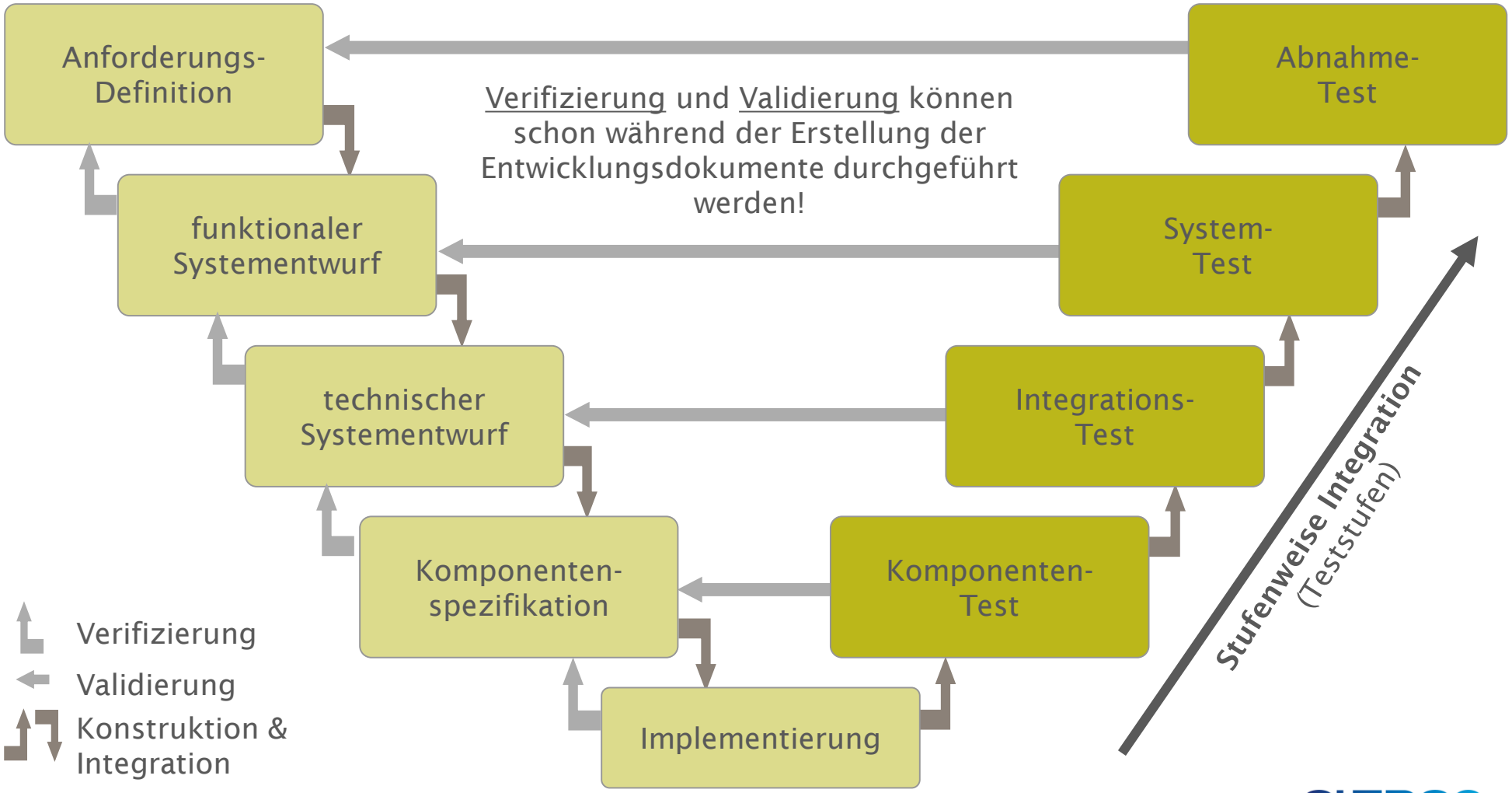
Simultan (zugleich, zusammen) (lat. *simultaneus*)
Das simultane Vorgehensmodell beschreibt zeitlich versetzte, teilparallel durchgeführte Aktivitäten

Wasserfall-Modell (sequentiell) [nach Royce]



Erweitertes Wasserfallmodell mit Rücksprungmöglichkeiten

Allgemeines V-Modell (sequentiell) [nach B. Boehm]

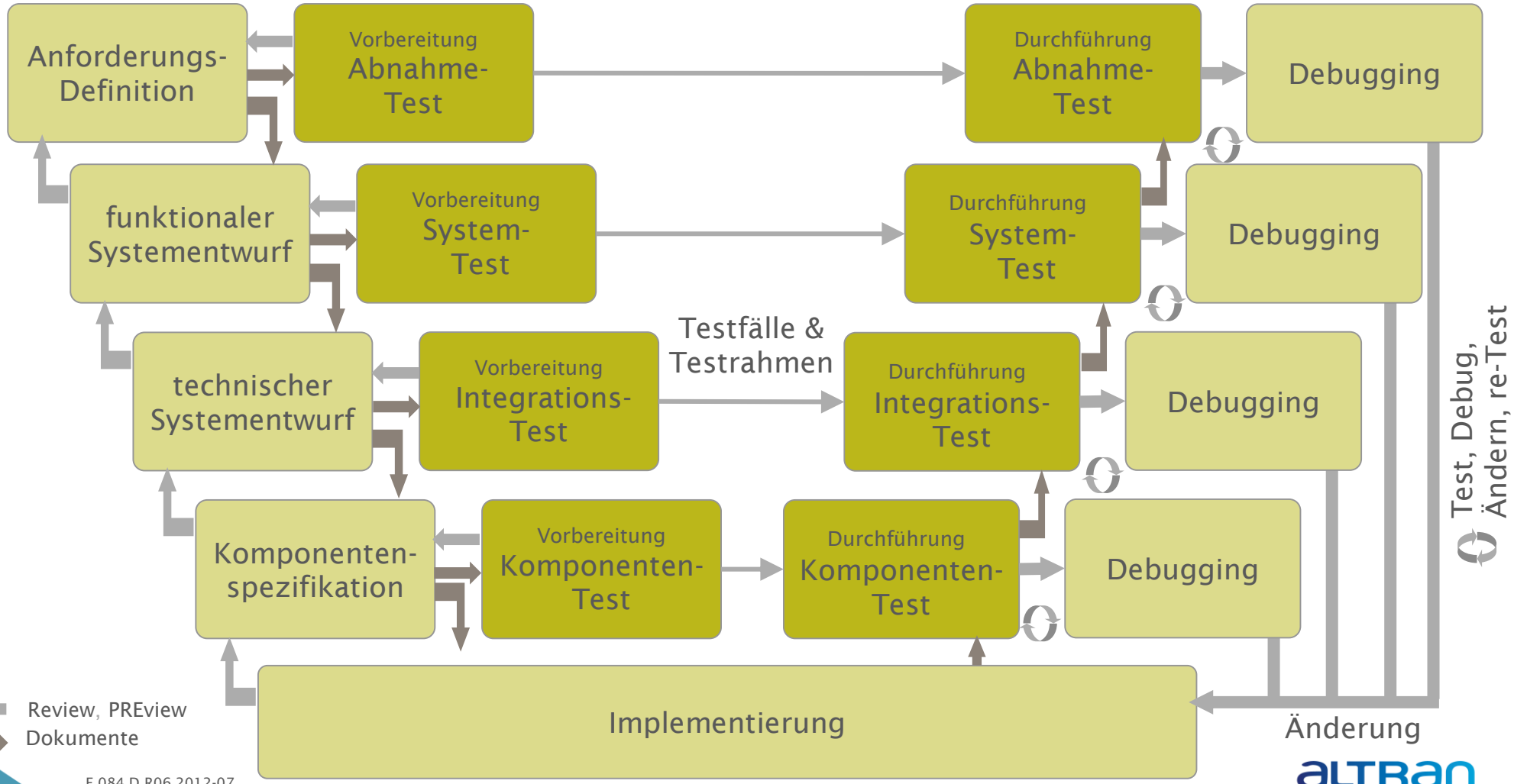


2. Der Testprozess – Testen im PEP

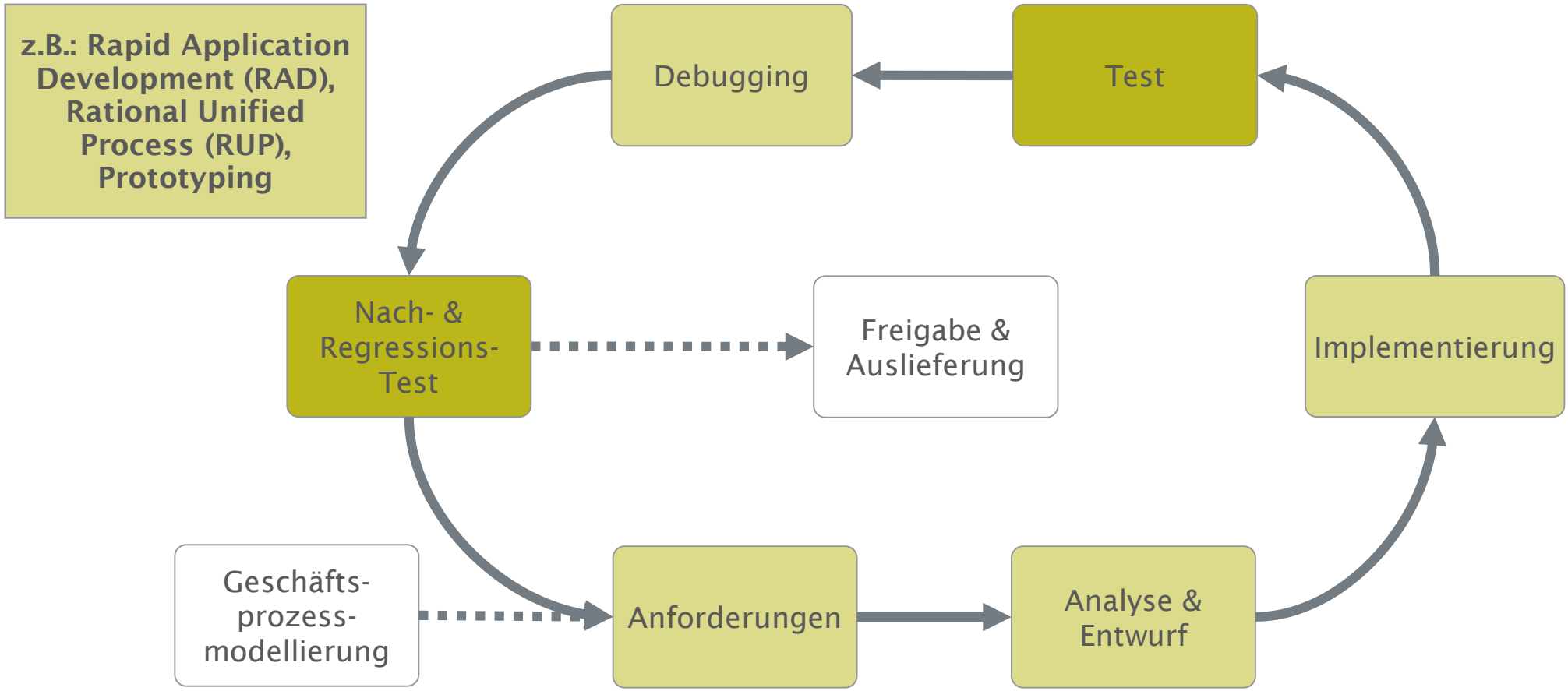
Exkurs

nicht Teil des CTFL!

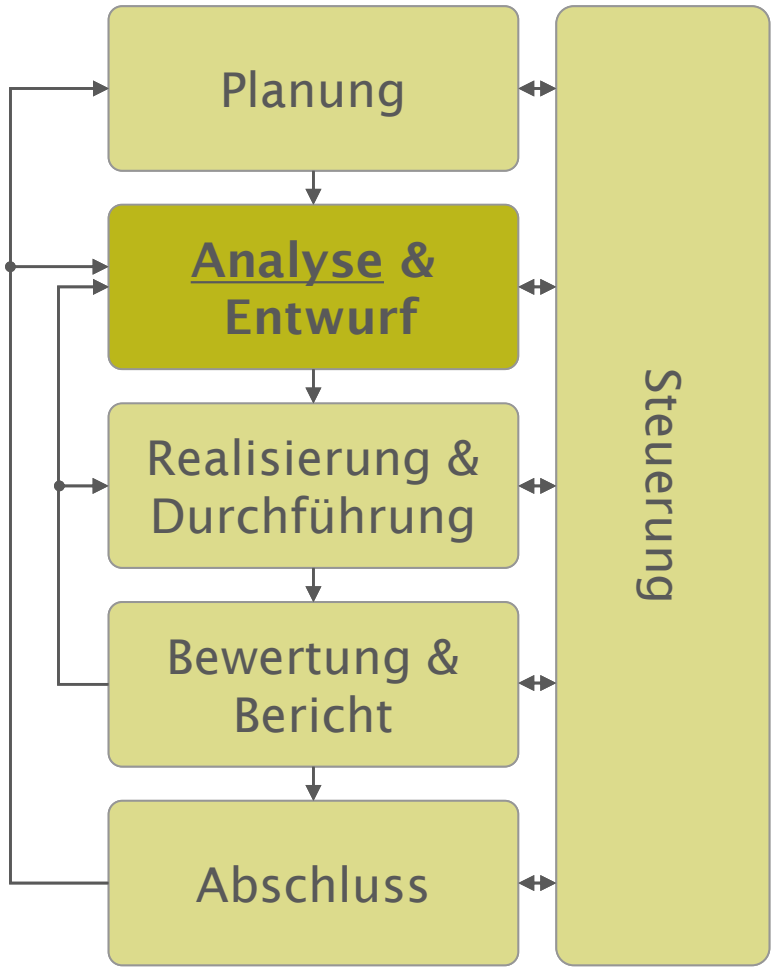
W-Modell (sequentiell-parallel) [nach Spillner_AL]



Iterativ-inkrementelle Entwicklungsmodelle [nach Balzert]



Was gehört zum Testen?



Entwicklungsmodelle geben keine Information darüber, was zum Testen gehört.

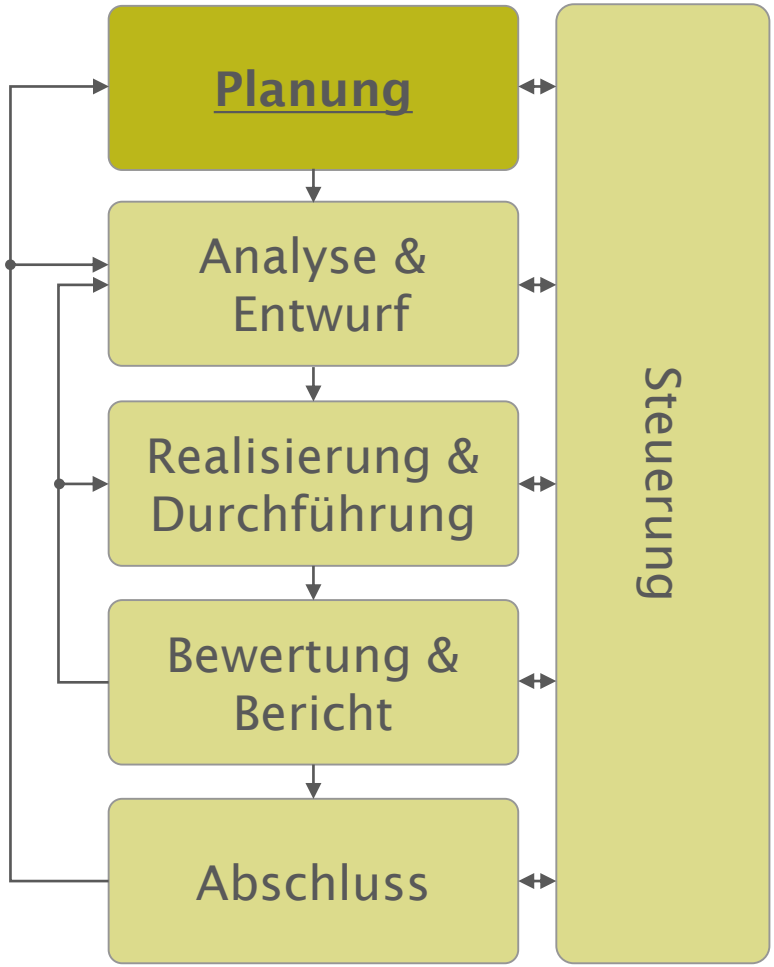
→ Testdurchführung ist der sichtbarste Teil des Testens

→ **Testen ist mehr als nur Testdurchführung!**

Anmerkung:

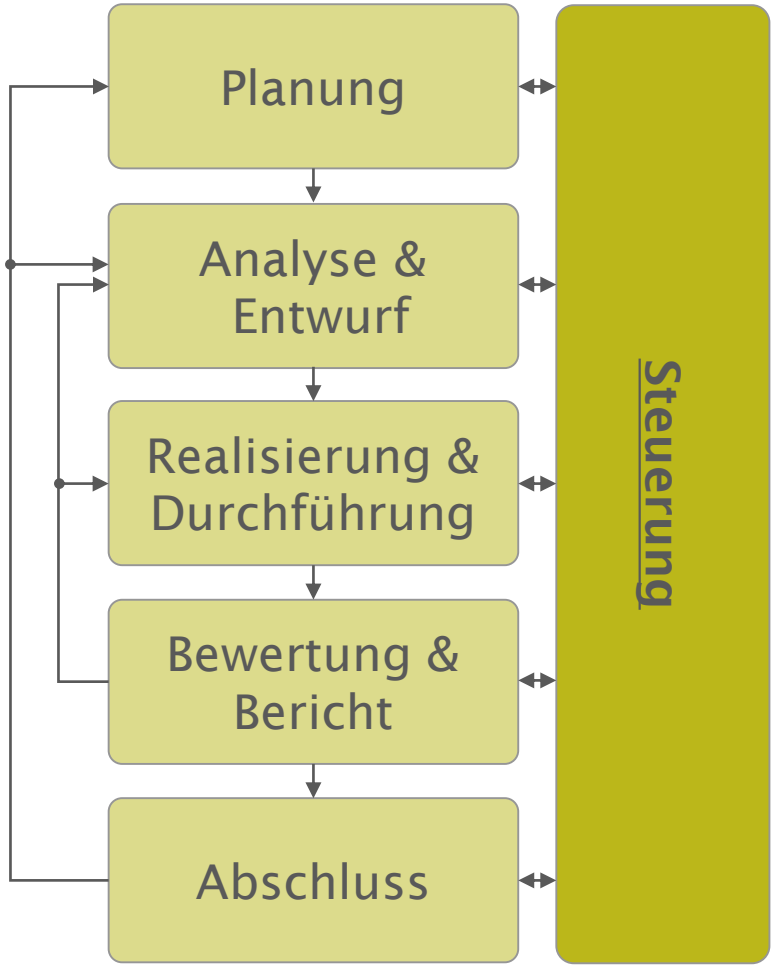
- einzelne Aktivitäten können auch (teilweise) parallel durchlaufen werden
- Alle während des Prozesses erstellten Arbeitsergebnisse sollten gereviewed werden

Testplanung (Testkonzept / Mastertestkonzept)



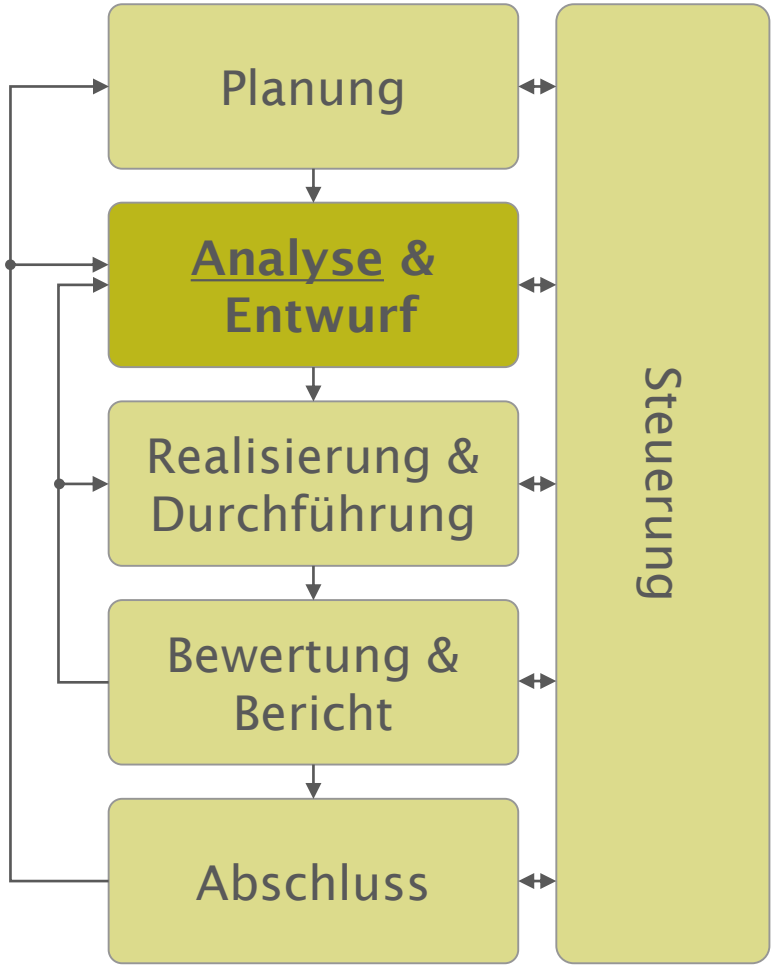
- Umsetzen der übergeordneten Testrichtlinie.
- Dokumentation der Teststrategie im Testkonzept / ggf. Mastertestkonzept:
  - Definition der Testziele (WARUM).
  - Festlegung des Aufgabenumfangs (WAS) des Testens.
  - Auswahl der Testentwurfsverfahren (WIE) und Testtiefe (Ausgangskriterien).
  - Einplanen benötigter Ressourcen (WOMIT).
  - Zeitplanung (WANN) über alle Testphasen.
  - Vereinbaren der Testorganisation (WER).

Teststeuerung



- Messen und Analysieren der Testmetriken (Testfortschritt, Testüberdeckung, Testergebnisse…)
- Überwachen und Dokumentieren des Testfortschritts über alle Testphasen
- Anstoßen von Korrekturmaßnahmen.
- Feedback an die Testplanung

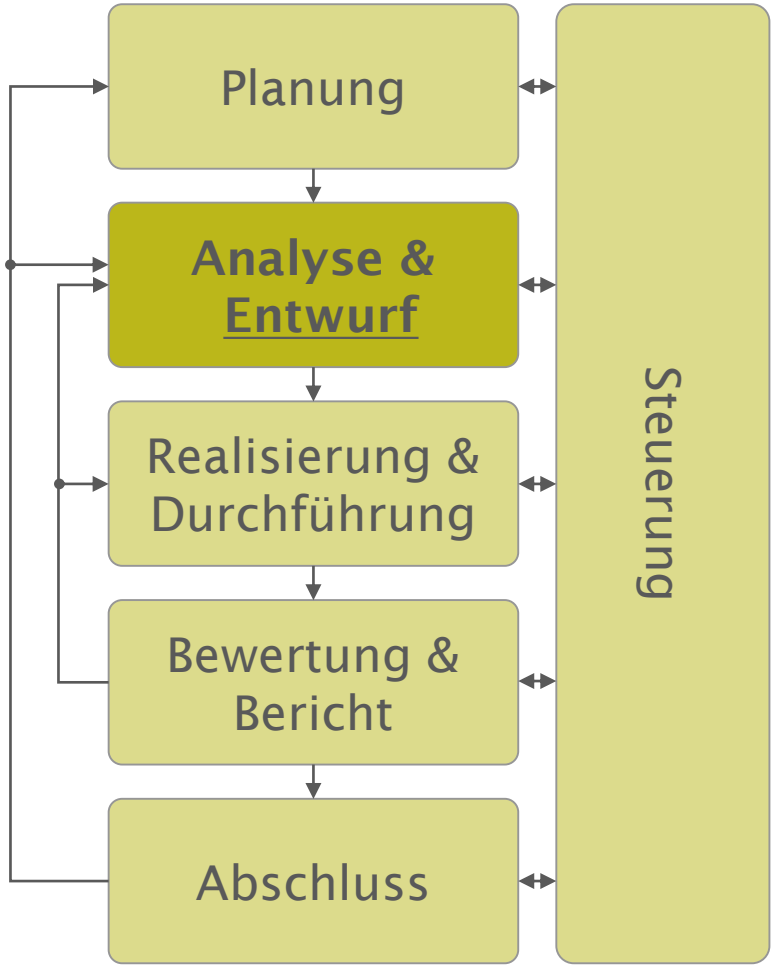
Testanalyse



- Review der Testbasis.
  (z.B. Anforderungen, Software Integrity Level*) (Risikoausmaß), Risikoanalysebericht , Architektur, Entwurf, Schnittstellenspezifikationen)
- Bewertung der Testbarkeit von Testbasis und Testobjekten.
- Identifizierung und Priorisierung der Testbedingungen auf Grundlage
  - der Testobjektanalyse.
  - der Spezifikation.
  - des Verhaltens.
  - der Struktur.

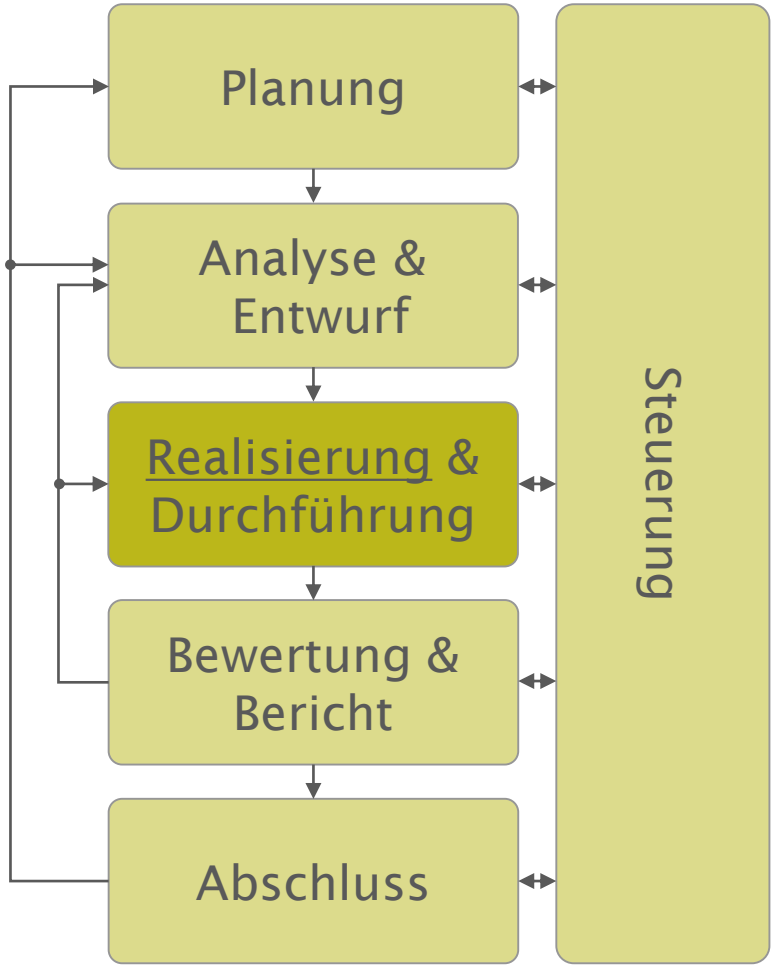
*) Der Erfüllungsgrad einer Menge vom Stakeholder ausgewählter Software- und/oder Software -basierter Merkmale (z.B. Softwarekomplexität, Risikoeinstufung, Sicherheitsstufe Zugriffsschutz) und funktionalen Sicherheit, gewünschte Performanz, Zuverlässigkeit, oder Kosten), die definiert wurden, um die Bedeutung der Software für den Stakeholder zum Ausdruck zu bringen.

Testentwurf



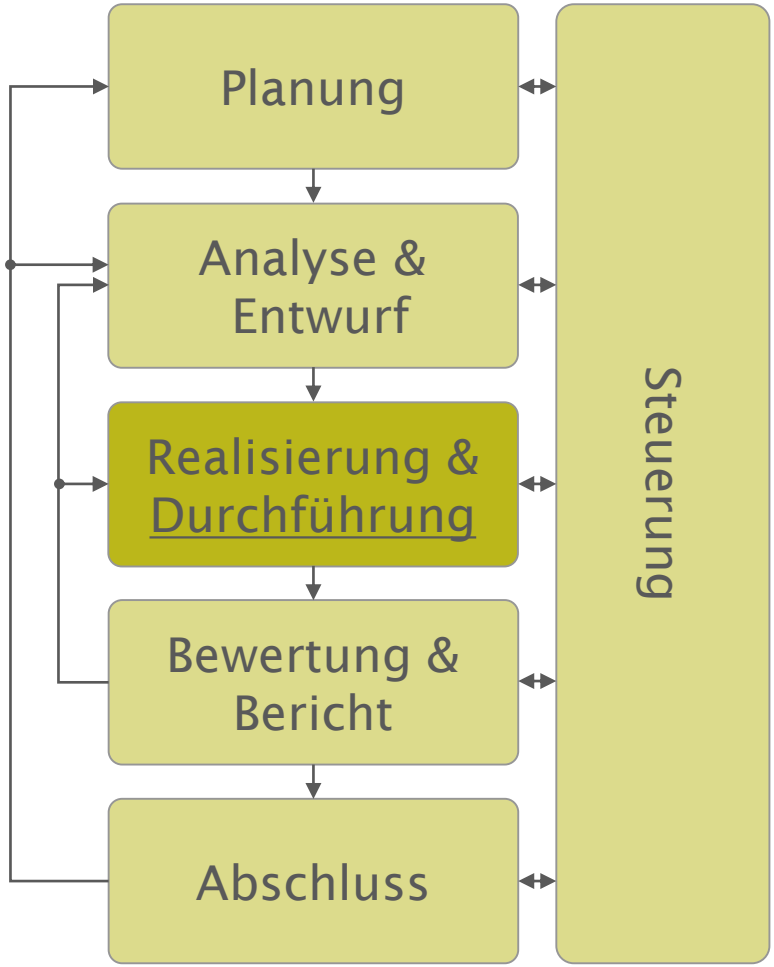
- Testentwurf und Priorisierung von abstrakten Testfällen
- Identifizierung benötigter Testdaten, um Definition von Testbedingungen und Testfällen zu unterstützen
- Entwurf des Testumgebungsaufbaus
- Identifikation der benötigten Infrastruktur und Werkzeuge
- Erzeugen (bzw. sicherstellen) der Rückverfolgbarkeit (Bidirektional zwischen Testbasis und Testfällen)

Testrealisierung



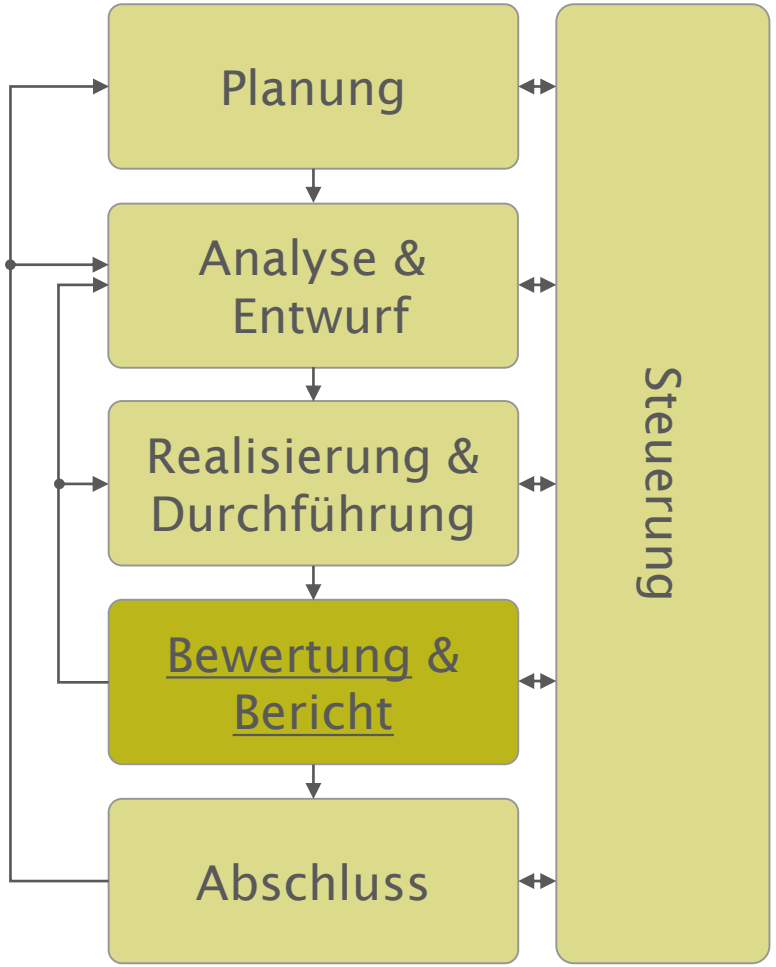
- Realisieren (entwickeln / ableiten) und priorisieren von:
  - konkreten Testfällen
  - Testdaten
  - Testablauf / Testskripte (Schritte zur Testausführung)
  - Testausführungsplänen oder Testsuiten (Zusammenstellung mehrerer Testfälle)
- Kontrolle, ob die Testumgebung korrekt aufgesetzt wurden.
- Überprüfen und aktualisieren der Rückverfolgbarkeit (Bidirektional zwischen Testbasis und Testfällen)

Testdurchführung



- Ausführung von Testabläufen unter Einhaltung des Testkonzept. (Reihenfolge, ggf. Testsuiten etc.)
- Protokollieren der Testergebnisse und der Konfiguration der eingesetzten Testumgebung im Testprotokoll
- Vergleich der Ist-Ergebnisse mit den Soll-Ergebnissen (vorausgesagten Ergebnissen).
- Festhalten und analysieren von gefundenen Fehlerwirkungen oder Abweichungen.
- Bestätigung der Fehlerbehebung durch Fehlernachtest.
- Testwiederholungen (Regressionstest).

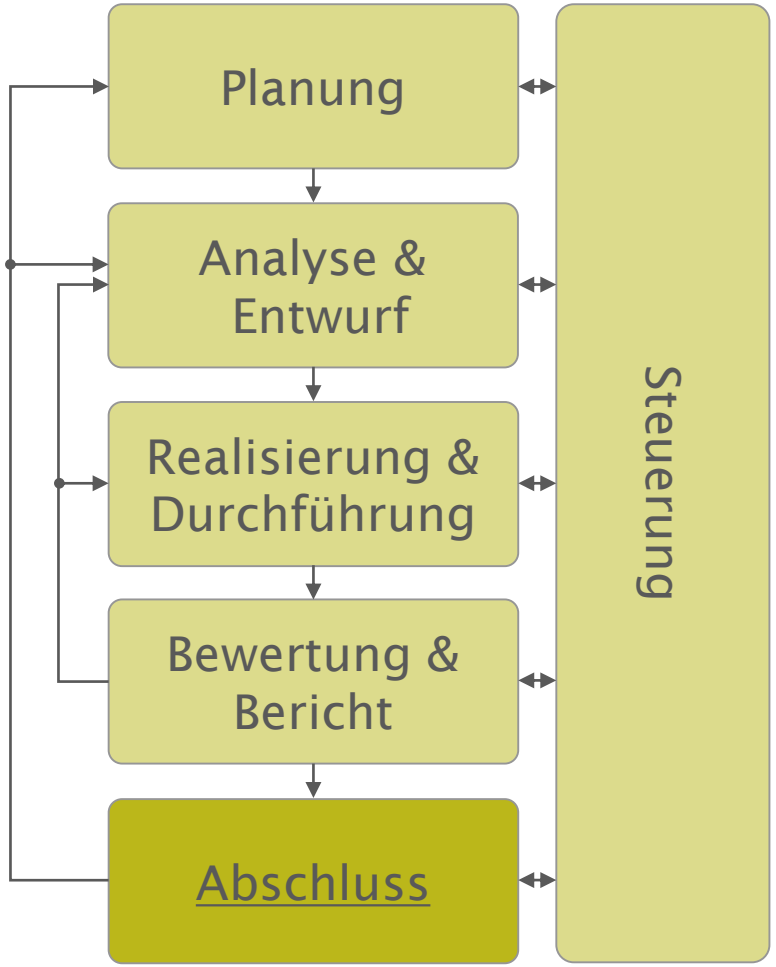
Bewertung von Ausgangskriterien & Testabschlussbericht



Bewerten der Ausgangskriterien durch untersuchen der Testaktivitäten hinsichtlich Ihrer Ziele.

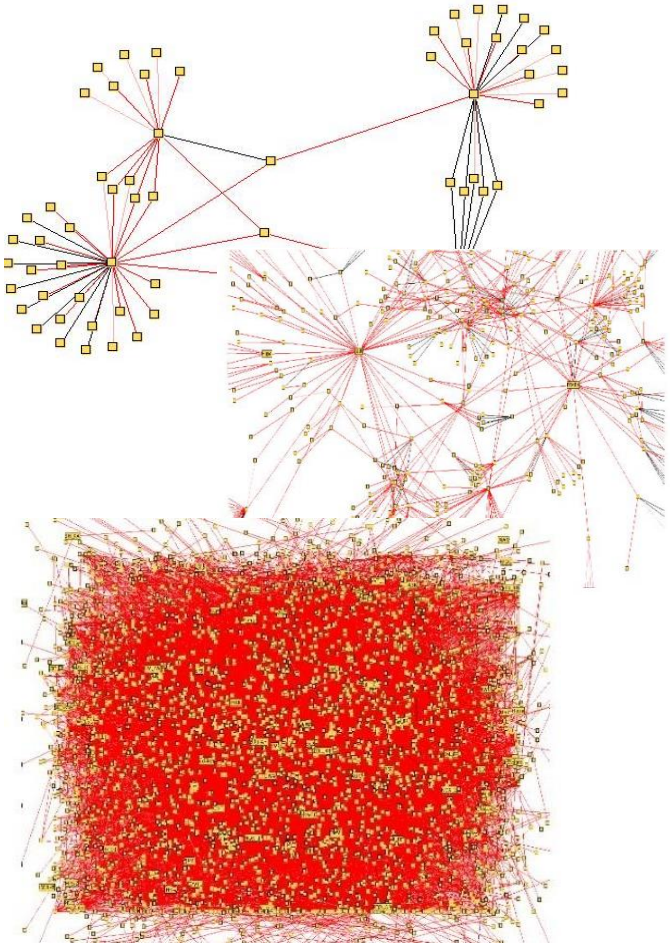
- Auswerten der Testprotokolle in Hinblick auf die im Testkonzept festgelegten Ausgangskriterien.
- Entscheidung:
  - Ob mehr Tests durchgeführt werden müssen.
  - Ob die festgelegten Ausgangskriterien angepasst werden müssen.
- Erstellen des Testabschlussberichts für die Stakeholder

Abschluss der Testaktivitäten



- Kontrolle, welche der geplanten Arbeitsergebnisse geliefert wurden.
- Schließen der Fehler-/Abweichungsmeldungen.
- Erstellen von Änderungsanforderungen. (z.B. für weiter bestehende Fehler/Abweichungen)
- Dokumentation der Abnahme des Systems.
- Dokumentation und Archivierung der Testmittel. (Dokumentation, Testumgebung, etc.)
- Übergabe der Testmittel an die Wartungsorganisation.
- Analyse und Dokumentation von „lessons learned“
- Nutzen der gesammelten Informationen zum verbesserr der Testreife.

Teststufen aufgrund der Vernetzungskomplexität



Mit neuen Produktgenerationen wird die „Intelligenz“ der Systeme und damit die **Komplexität der Produkte** weiter vorangetrieben

→ Die gestiegene Komplexität hat damit auch die Anzahl und **Vernetzung der Komponenten** (funktional & elektrisch) erhöht

→ Deren Beherrschung erfordert eine **stufenweise Integration**

Partitionierung aufgrund der Produktkomplexität

Teile und Herrsche

Komplexe Produkte werden durch eine Vielzahl von Personen entwickelt

→ Eine **Partitionierung** in beherrschbare Einheiten ist notwendig

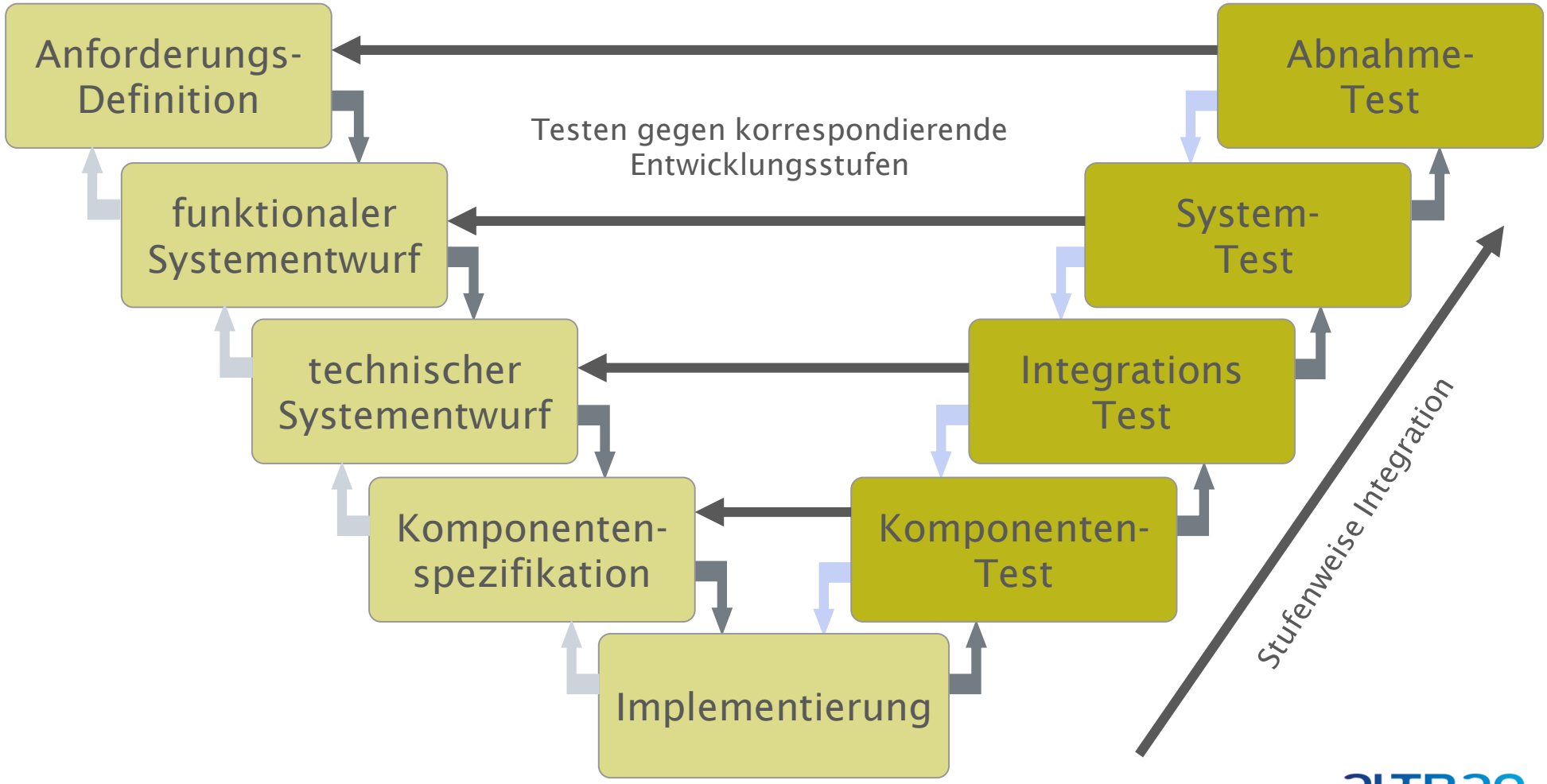
Die Produktentwicklungszeiten werden immer kürzer

→ (teilweise) **paralleles Entwickeln** der einzelnen Einheiten von verschiedenen Teams

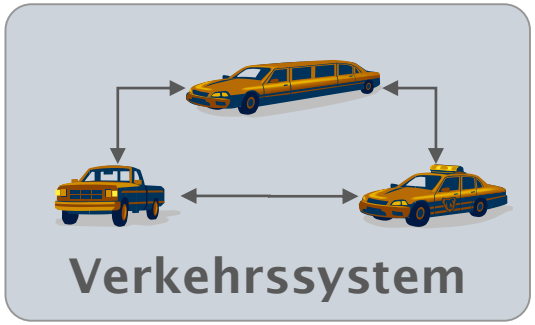
Integration

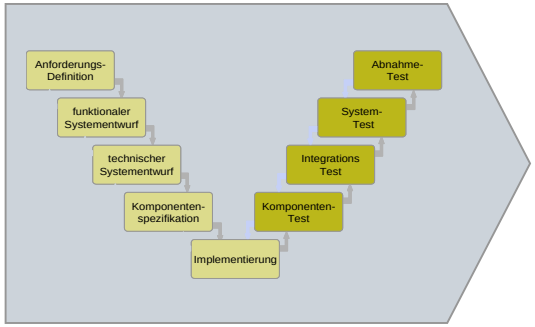
→ **Zusammenfügen** der einzelnen Einheiten am Ende der Produktentwicklung

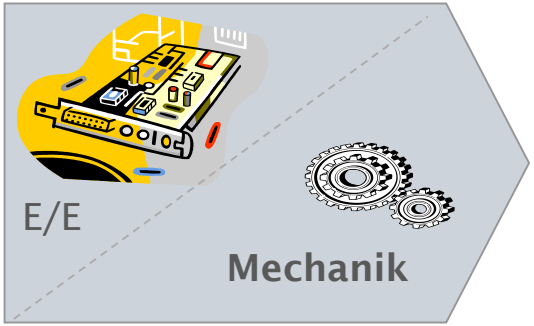
Übersicht über Teststufen

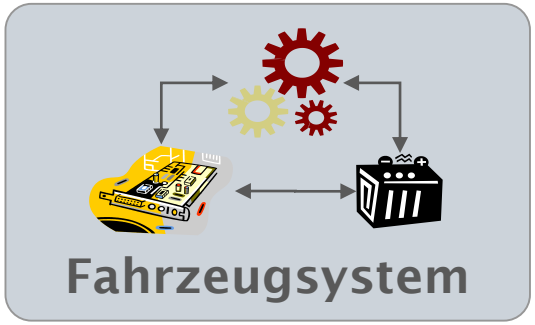


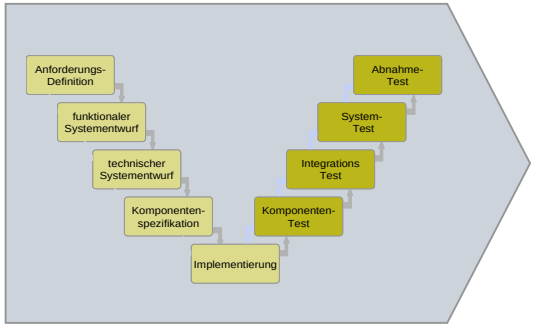
Teststufen auf mehreren Ebenen

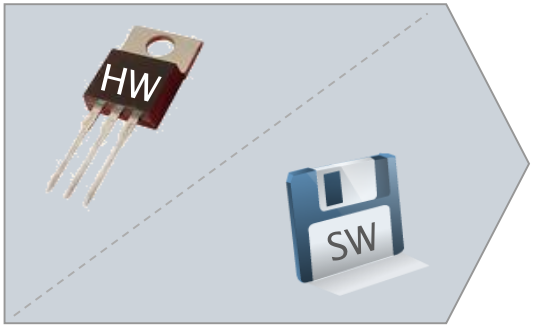


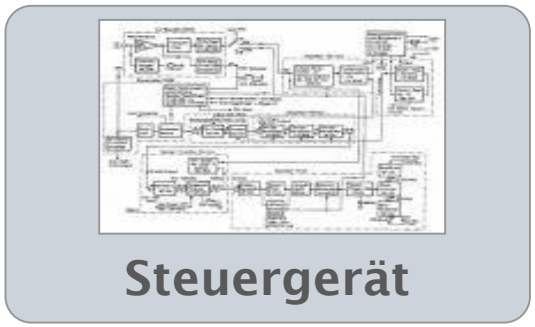


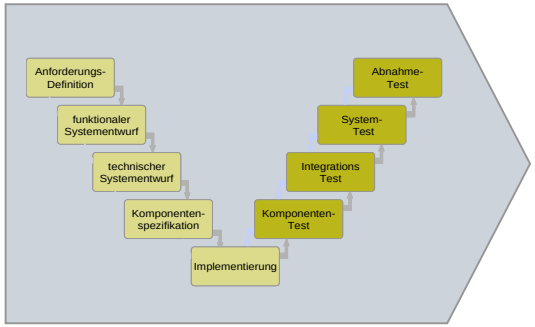


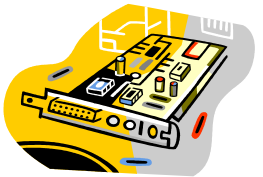




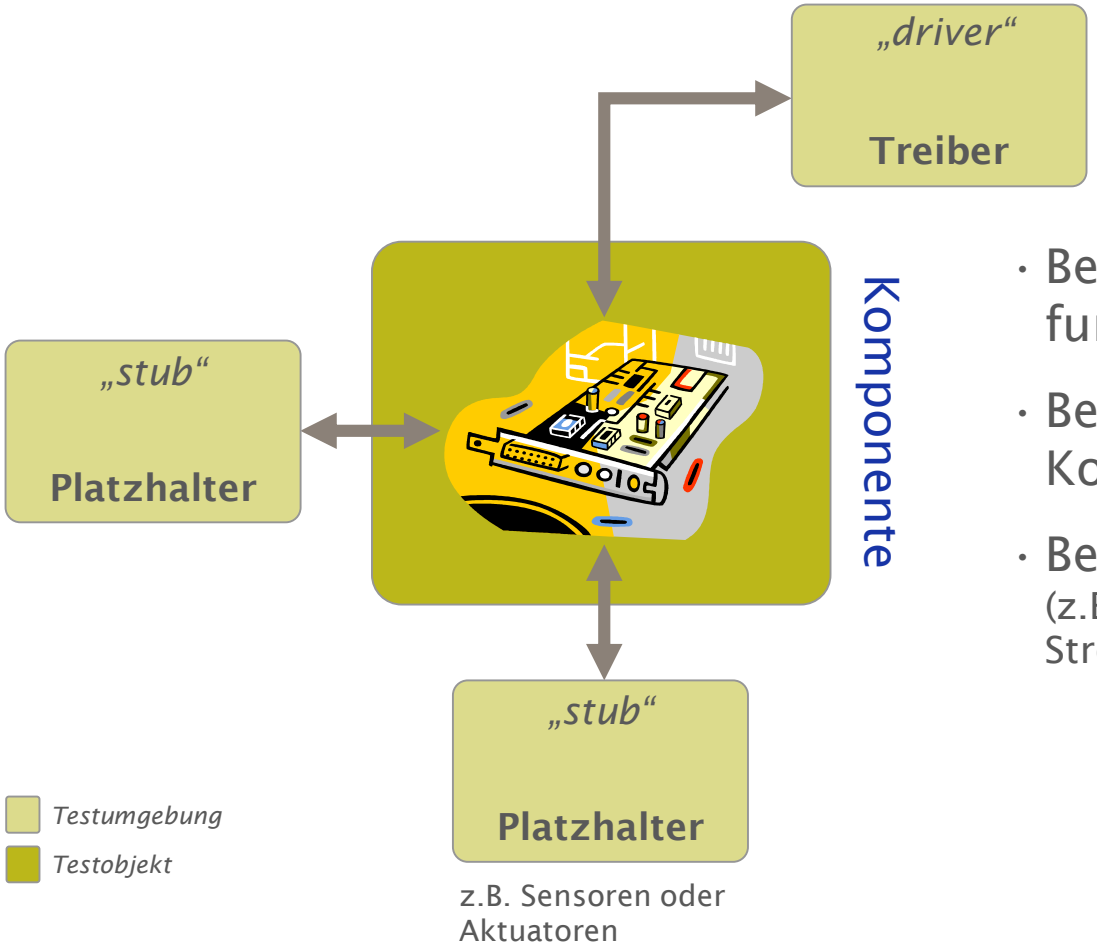






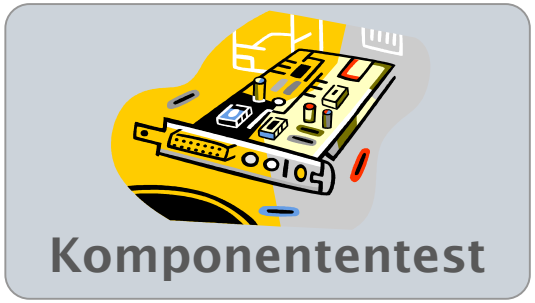


Komponententest



- Bestätigen der Komponenten-funktionen
- Bewerten der nicht-funktionalen Komponenteneigenschaften
- Bewerten des Ressourceneinsatzes (z.B. Prozessor-/Speicherauslastung, Stromverbrauch)

Komponententest - Definition

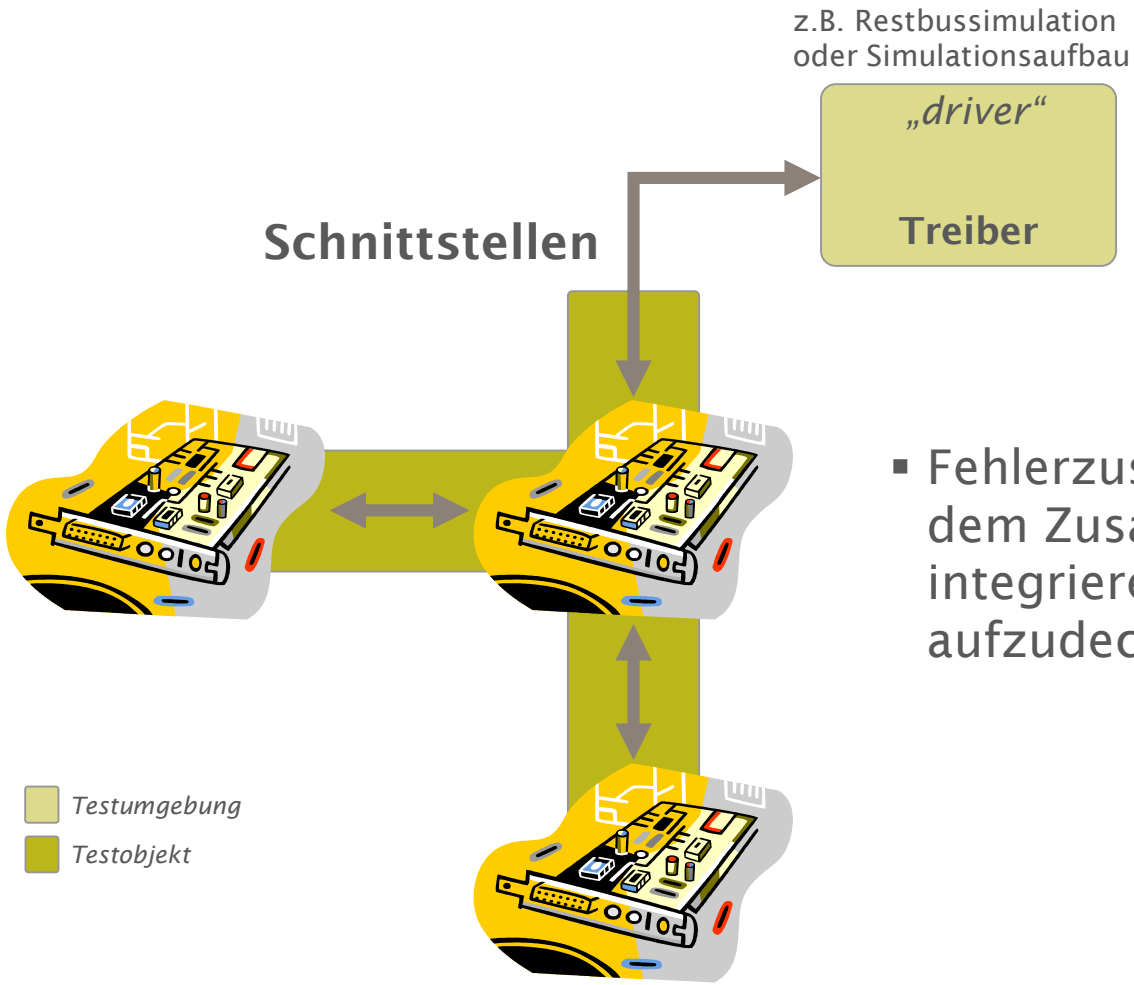


Komponententest (engl. *component testing*)

Test einer (einzelnen) Komponente. [nach IEEE 610][ISTQB_DE]

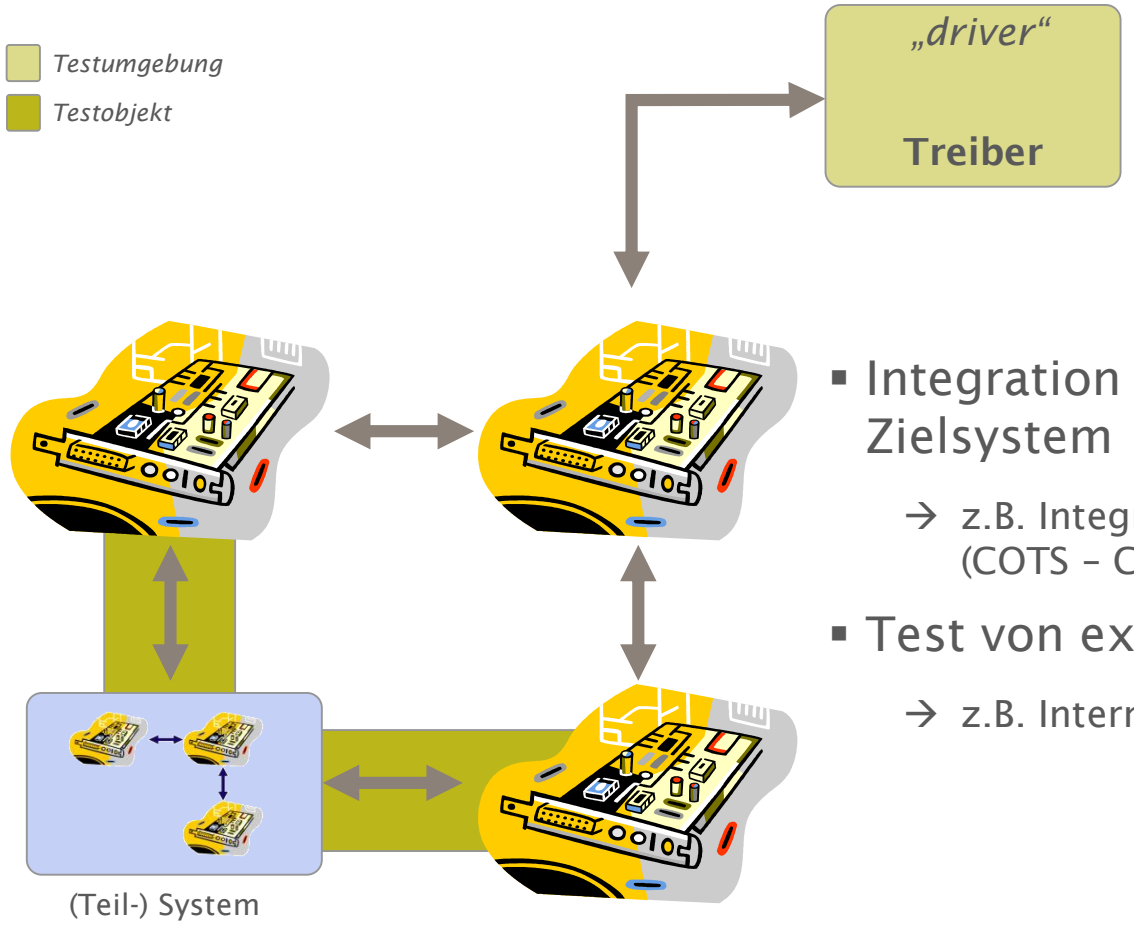
Ziel	• Bestätigung der Komponenten-Funktion • Bewerten des Umgangs mit Ressourcen	• Performance (Stressverhalten, Antwortzeit) • Robustheit (Negativtests, ungültige Werte)
Testbasis	• Anforderungen an die Komponente • detaillierter Entwurf	• Code
Testobjekt	• Komponenten • Programme	• Datenumwandlung/Migrationsprogr. • Datenbankmodule
Typische Fehlerwirkungen	• Funktionale Komponentenfehler • Laufzeitfehler	• (lokale) Performance Probleme • Robustheit
Testumgebung	• Simulatoren (Prozessor/Hardware/Software in the Loop) • Entwicklungswerkz. (Debugger, Unit Test Framework)	• Treiber (driver) • Platzhalter (stubs)
Spezifische Ansätze	• White-Box-Test • Black-Box-Test	• Grey-Box-Test • Testgetriebene Entwicklung
Verantwortlichkeiten	• Häufig der Entwickler selber ***Fehlerzustandwerden häufig direkt behoben und gar nicht erst erfasst!***	

Integrationstest (in realer Umgebung)



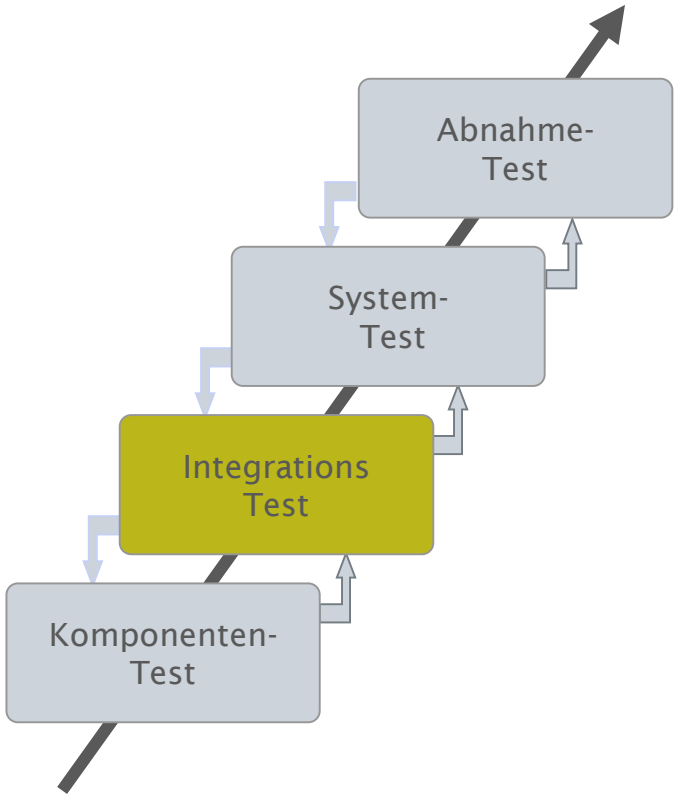
- Fehlerzustand in den Schnittstellen und dem Zusammenwirken der zu integrierenden Komponenten aufzudecken

(System-) Integrationstest



- Integration eines (Teil-) Systems in das Zielsystem
  - → z.B. Integration kommerzieller Standardsoftware (COTS – Commercial off the shelf)
- Test von externen Schnittstellen
  - → z.B. Internet, Datenaustausch

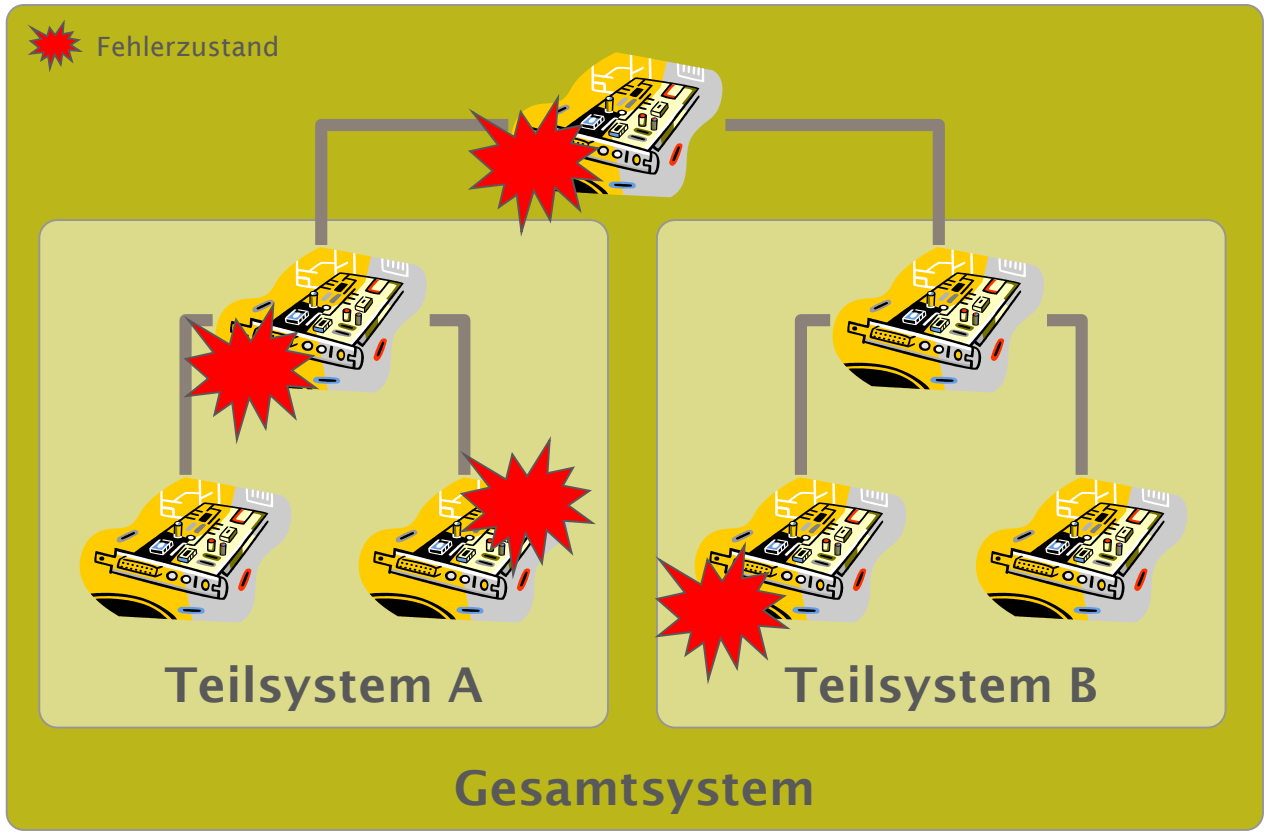
Integrationsstrategien



- „Big Bang“ Integration
  - → Alle Elemente werden auf ein Mal integriert
- „Bottom Up“ Integration
  - → Elemente werden jeweils mit **Treibern** (engl. *driver*) integriert
- „Top Down“ Integration
  - → Elemente werden jeweils mit **Platzhaltern** (engl. *stubs*) integriert
- „Critical First“ Integration
  - → Kritische Elemente werden als erste mit **Platzhaltern und Treibern** integriert
- „Ad-Hoc“ Integration
  - → Elemente werden in der Reihenfolge der Verfügbarkeit mit **Platzhaltern und Treibern** integriert

Platzhalter und Treiber bilden das Testgerüst (engl. *Scaffolder*).

Integrationstest – Big Bang



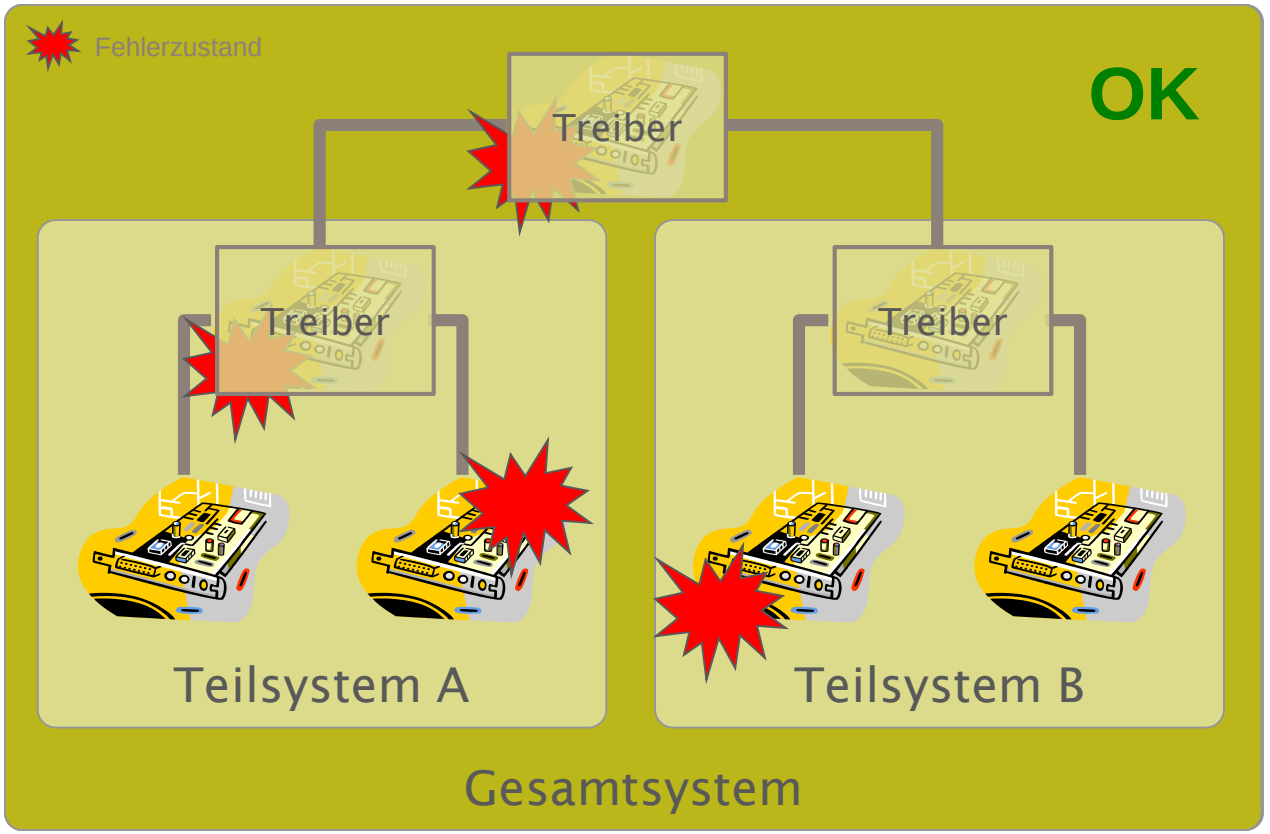
Vorteile

+ Gut geeignet bei wenig Änderungen
+ Benötigt keine Treiber & Platzhalter

Nachteile

– Fehlerzustandschwer lokalisierbar
– Hohe Wartezeit bis zur vollständigen Verfügbarkeit

Integrationstest – Bottom up



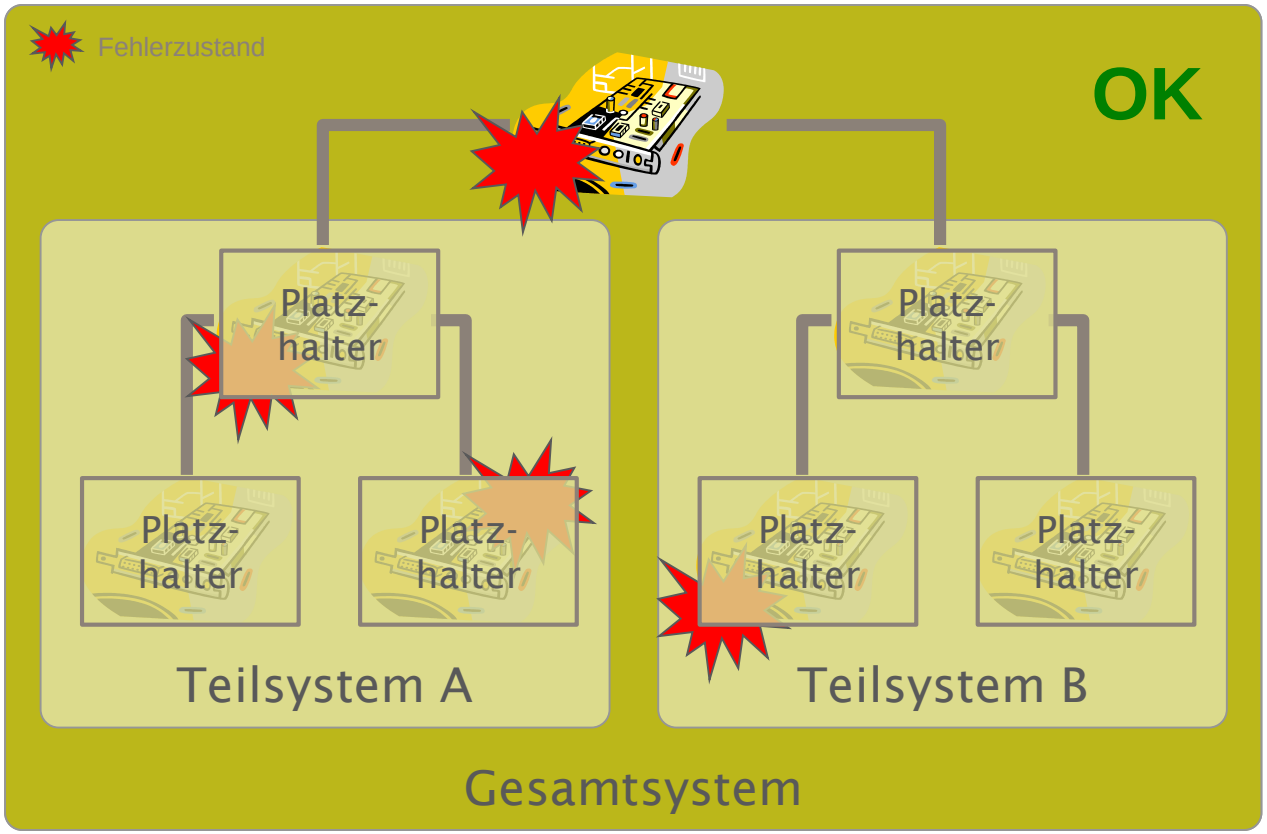
Vorteile

+ Benötigt keine Platzhalter für untergeordnete Komponenten

Nachteile

– Übergeordnete Komponenten müssen durch Treiber simuliert werden

Integrationstest – Top Down



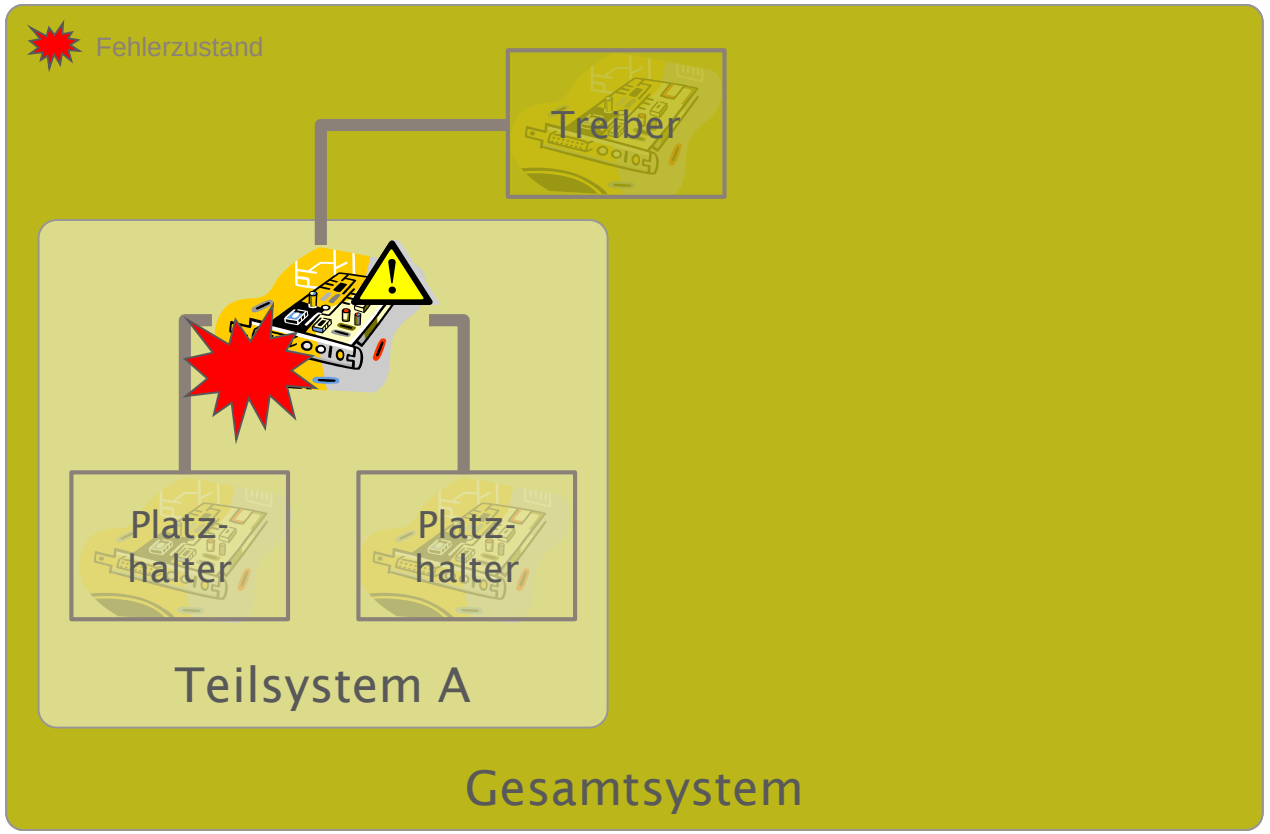
Vorteile

+ Es werden keine bzw. nur sehr einfache Treiber benötigt

Nachteile

– Es werden Platzhalter für untergeordnete Komponenten benötigt
→ ggf. sehr aufwendig

Integrationstest – Critical First



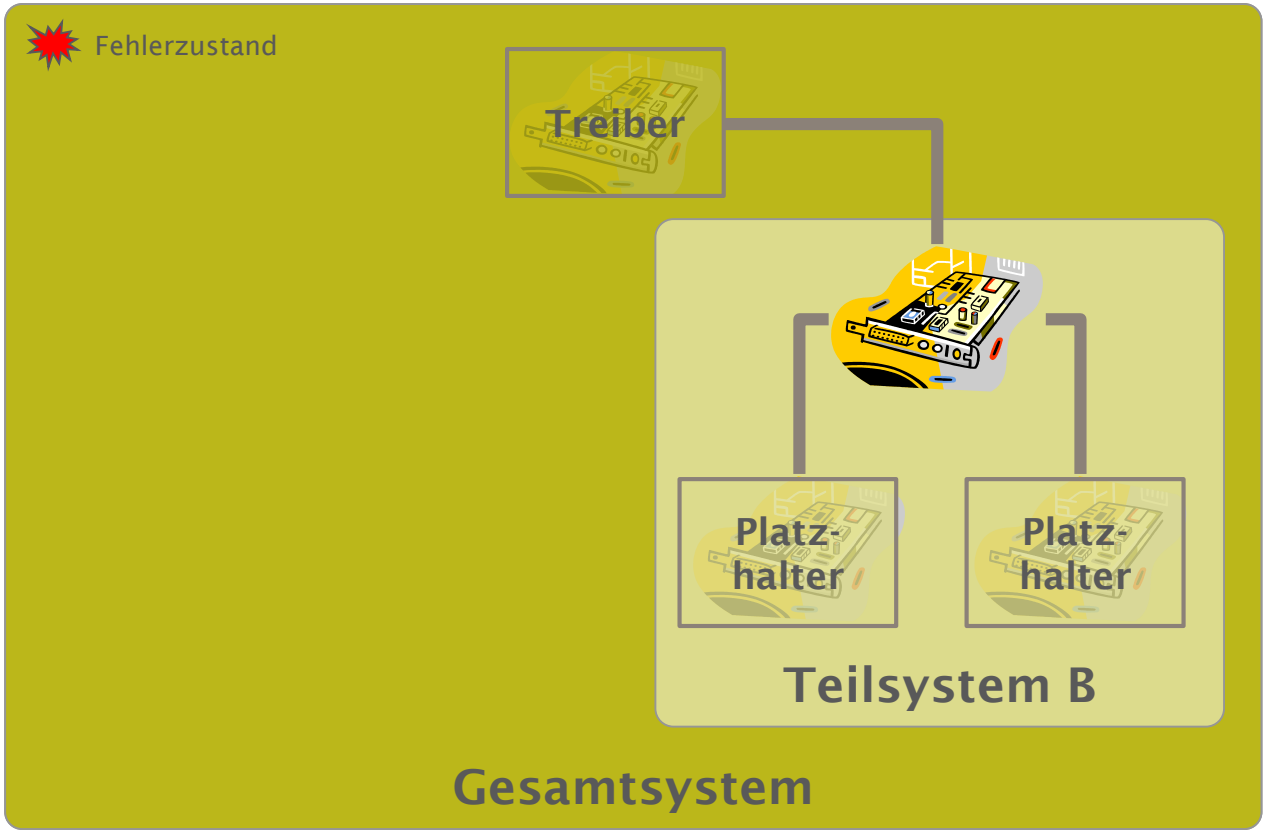
Vorteile

+ Effizienter Ansatz
+ Paart Vorteile der Top-Down und Bottom-Up Integrationsstrategie

Nachteile

– Es werden sowohl Platzhalter und Treiber benötigt
– Vorherige Risikobewertung notwendig

Integrationstest – Ad-Hoc



Vorteile

+ Zeitgewinn, da jeder Baustein frühestmöglich integriert wird

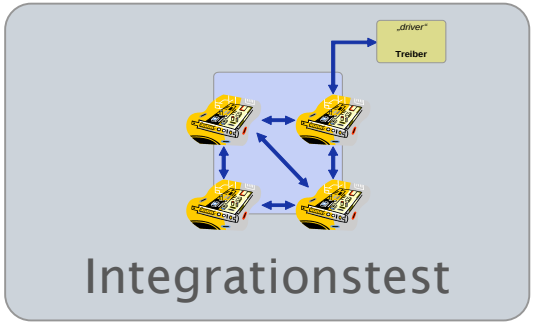
Nachteile

– Es werden sowohl Platzhalter und Testreiber benötigt

Integrationstest - Definition

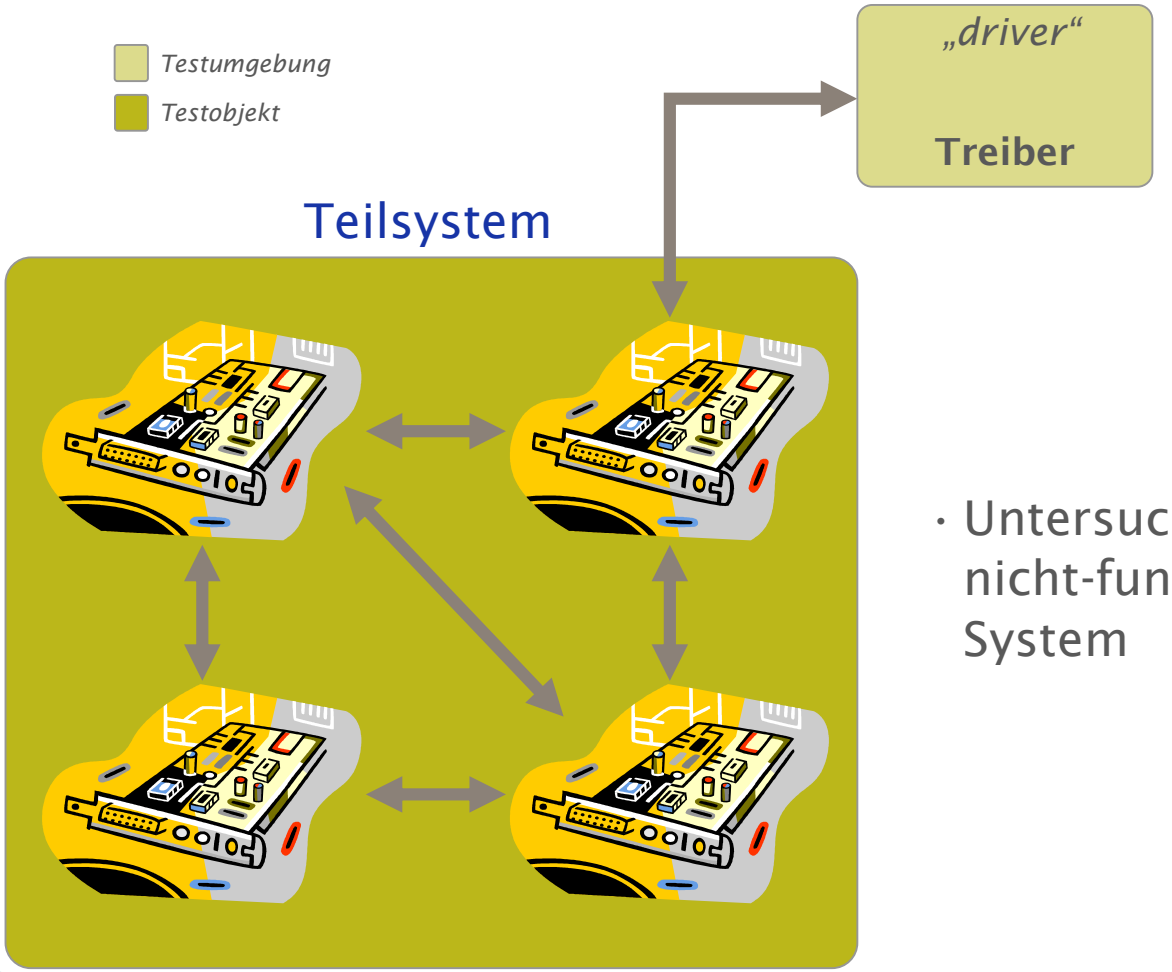
Integrationstest (engl. *integration testing*)

Testen mit dem Ziel, Fehlerzustände in den Schnittstellen und im Zusammenspiel zwischen integrierten Komponenten aufzudecken. [ISTQB_DE]



Grundannahme	• **Die Komponente in sich ist korrekt**	
Ziel	• Integration von (Teil-) Systemen • Fehlerzustände in Schnittstellen aufdecken	• Interaktionen zwischen verschiedenen Teilen eines Systems
Testbasis	• Software- und Systementwurf • Architektur	• Nutzungsabläufe/Workflows • Anwendungsfälle (use cases)
Testobjekt	• Datenbankimplementierungen in Subsystemen • Infrastruktur	• Schnittstellen • Systemkonfiguration und Konfigurationsdaten
Typische Fehlerwirkungen	• Bus-Timing • Kommunikationsfehler	• Fehlerzustandin Schnittstellen
Testumgebung	• Platzhalter (stubs)	• Treiber (driver)
Spezifische Ansätze	• Komponentenintegrationstest • Systemintegrationstest	• Top-Down / Bottom Up / Critical First • Big Bang
Verantwortlichkeiten	• Integrationstester	

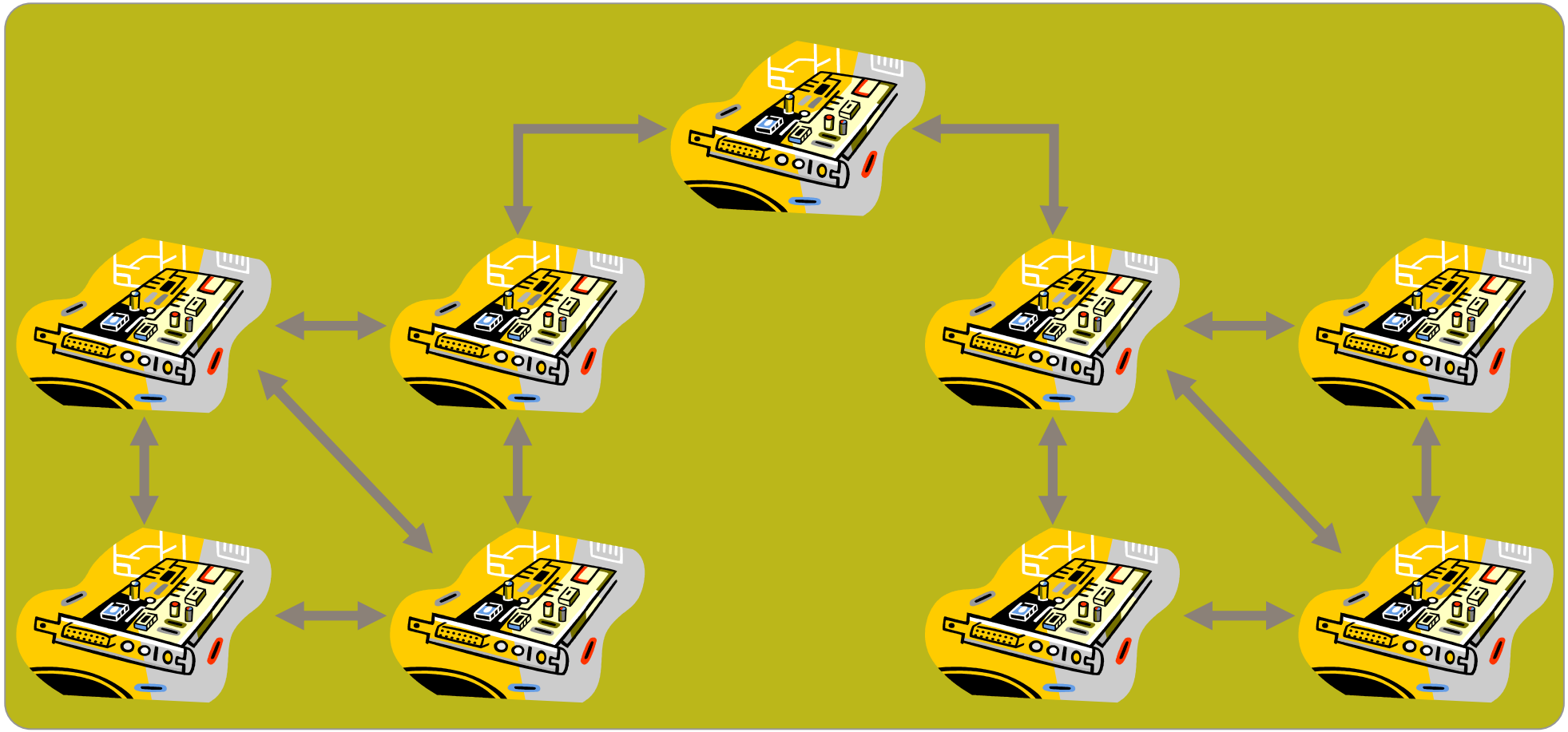
(Teil-) Systemtest



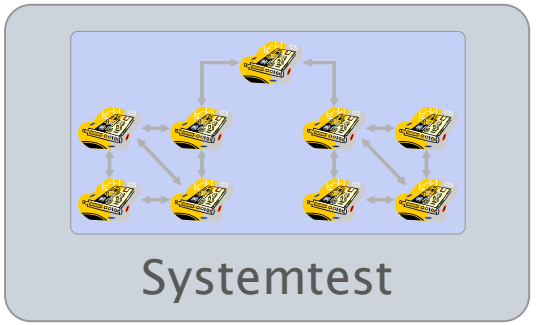
- Untersuchen von funktionalen als auch nicht-funktionalen Anforderungen an das System

Systemtest

System



Systemtest - Definition

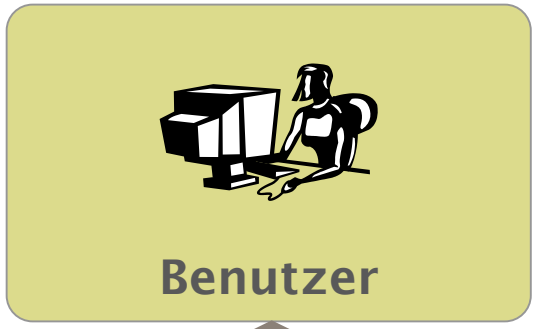


Systemtest (engl. *system testing*)

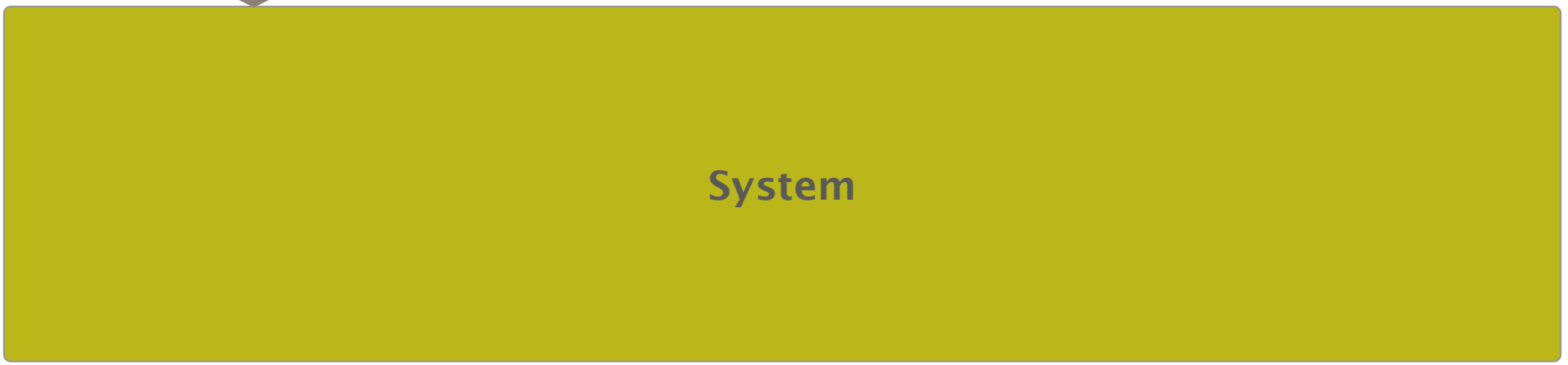
Test eines integrierten Systems, um sicherzustellen, dass es spezifizierte Anforderungen erfüllt.[ISTQB_DE]

Grundannahme	• **Die Komponenten in sich sind korrekt** • **Die Schnittstellen an sich sind korrekt**	
Ziel	• Untersuchen von funktionalen als auch nicht-funktionalen Anforderungen an das System	
Testbasis	• System- und Anforderungsspezifikation • Anwendungsfälle (use cases)	• funktionale Spezifikation • Risikoanalyseberichte
Testobjekt	• System-, Anwender- und Betriebshandbücher	• Systemkonfiguration und Konfigurationsdaten
Typische Fehlerwirkungen	• umgebungsspezifische Fehler • Konzeptionelle Fehler	• unvollständige oder undokumentierte Anforderungen
Testumgebung	• Möglichst Ziel- oder Produktionsumgebung	• Ggf. Platzhalter und Treiber für Teilsystem
Spezifische Ansätze	• White-Box-Test • Black-Box-Test	
Verantwortlichkeiten	• Häufig unabhängige Testteams	

Abnahmetest (Akzeptanztests)



- Vertrauen in das System schaffen
- Bereitschaft eines Systems für den Einsatz und die Nutzung bewerten



Aspekte bei Abnahmetests I

Abnahmetest

- Benutzer-Abnahmetest
  - Prüft die Tauglichkeit eines Systems zum Gebrauch durch Benutzer bzw. Kunden
- Betrieblicher Abnahmetest
  - Die Abnahme des Systems durch den Systemadministrator beinhaltet etwa:
    - Erstellen und Wiedereinspielen von Sicherungskopien (wie z.B. Backup/Restore)
    - Wiederherstellbarkeit nach Ausfällen
    - Benutzermanagement
    - Datenlade- u. Migrationsaufgaben und
    - Periodische Überprüfungen von Sicherheitslücken

Aspekte bei Abnahmetests II

- Alpha-Test
  - Tests kommerzieller Produkte durch potentielle Kunden/Benutzer am Herstellerstandort
- Beta-Test (Feldtest)
  - Tests kommerzieller Produkte durch potentielle Kunden/Benutzer am Kundenstandort
    (z.B. auch COTS, Zukaufteile)

Aspekte bei Abnahmetests III

- Regulatorischer Abnahmetest

  Prüfen gegen alle Gesetze und Standards, denen das System entsprechen muss (z.B. staatliche, gesetzliche oder Sicherheitsbestimmungen)

- Vertraglicher Abnahmetest

  Prüfen gegen vertragliche Abnahmekriterien

  → Abnahmekriterien sollten definiert werden, wenn der Vertrag abgeschlossen wird

Abnahmetest - Definition

Abnahmetest (engl. *acceptance testing*)

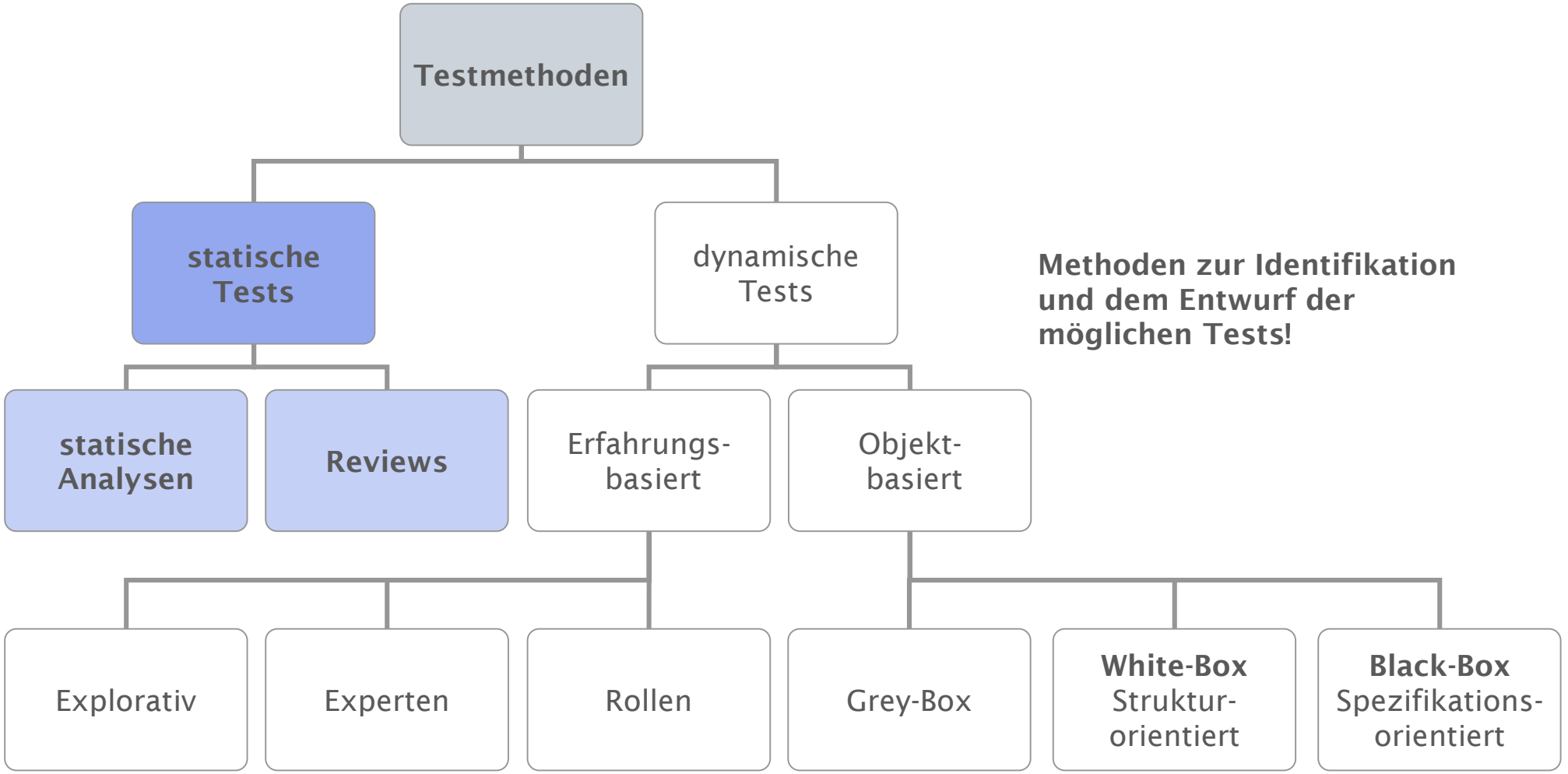
Formales Testen hinsichtlich der Benutzeranforderungen und -bedürfnisse bzw. der Geschäftsprozesse. Es wird durchgeführt, um einem Auftraggeber oder einer bevollmächtigten Instanz die Entscheidung auf Basis der Abnahmekriterien zu ermöglichen, ob ein System anzunehmen ist oder nicht. [nach IEEE 610][ISTQB_DE]

Grundannahme	• **Das System in sich ist korrekt**	
Ziel	• Vertrauen in das System schaffen • Bereitschaft eines Systems für den Einsatz und die Nutzung bewerten	
Testbasis	• System- & Benutzeranforderungen • Anwendungsfälle (use cases)	• Geschäftsprozesse • Risikoanalyseberichte
Testobjekt	• Das voll integrierte Systems • Anwenderverfahren	• Formulare & Berichte • Konfigurationsdaten
Typische Fehlerwirkungen	• Bedienkonzeptfehler • Spezifikationsfehler	
Testumgebung	• Einsatzumgebung beim Kunden	
Spezifische Ansätze	• Benutzer-Abnahmetest • Regul. und vertraglicher Abnahmetest	• Betrieblicher Abnahmetest • Alpha- und Beta-Test (oder Feldtest)
Verantwortlichkeiten	• Kunden oder Benutzer des Systems • Weitere Stakeholder	• Prüfzentren (z.B. TÜV)

Gegenüberstellung der Teststufen

	Komponententest	Integrationstest	Systemtest	Abnahmetest
Ziel	• Bestätigung der Komponenten-Funktion und Eigenschaften	• Integration von Systemen • Fehlerzustandin Schnittstellen • Interaktionen zwischen verschiedenen Teilen	• Untersuchen von funktionalen als auch nicht-funktionalen Anforderungen an das System	• Vertrauen schaffen • Bereitschaft eines Systems für den Einsatz und die Nutzung bewerten
Testbasis	• Anforderungen an die Komponente • detaillierter Entwurf • Code	• Software- und Systementwurf • Architektur • Nutzungsabläufe/Workflows • Anwendungsfälle (use cases)	• System- und Anf.-Spezifikation • Anwendungsfälle (use cases) • funktionale Spezifikation • Risikoanalyseberichte	• System- & Benutzeranf. • Anwendungsfälle (use cases) • Geschäftsprozesse • Risikoanalyseberichte
Testobjekt	• Komponenten • Programme • Datenumwandlung • Migrationsprogramme • Datenbankmodule	• Datenbankimpl. in Subsystemen • Infrastruktur • Schnittstellen • Syst.-Konfiguration und Konfigurationsdaten	• System-, Anwender- und Betriebshandbücher • Systemkonfiguration und Konfigurationsdaten	• Das voll integrierte Systems • Anwenderverfahren • Formulare & Berichte • Konfigurationsdaten
Typische Fehler-wirkungen	• Funktionale Komponentenfehler • Laufzeitfehler • (lokale) Performance Probleme • Robustheit	• Bus-Timing • Kommunikationsfehler • Fehlerzustandin Schnittstellen	• umgebungsspezifische Fehler • Konzeptionelle Fehler • unvollständige oder undokumentierte Anf.	• Bedienkonzeptfehler • Spezifikationsfehler
Testumgebung	• Simulatoren Entwicklungswerkz. • Treiber (driver) • Platzhalter (stubs)	• xIL • Treiber (driver) • Platzhalter (stubs)	• Möglichst Ziel- oder Produktionsumgebung • Ggf. Platzhalter und Treiber für Teilsystem	• Einsatzumgebung beim Kunden
Spezifische Ansätze	• White-Box-Test • Black-Box-Test • Grey-Box-Test • Testgetriebene Entwicklung	• Komponentenintegrationstest • Systemintegrationstest • Top-Down / Bottom Up / Critical First / Big Bang	• White-Box-Test • Black-Box-Test	• Benutzer-Abnahmetest • Regul. und vertraglich • Betrieblicher Abnahmetest • Alpha- und Beta-Test
Verantwort-lichkeiten	• Häufig der Entwickler selber	• Integrationstester	• Häufig unabhängige Testteams	• Kunden oder Benutzer Weitere Stakeholder • Prüfzentren

Übersicht über statische Tests

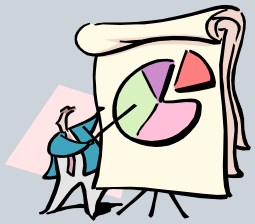


Review vs. Analyse

Review

Review [nach ISTQB_DE] („strukturierte Gruppenprüfungen“)

- Bewertung eines Produktes oder eines Projektstatus
- Diskrepanzen zu geplanten Ergebnissen aufdecken
- Verbesserungspotential identifizieren



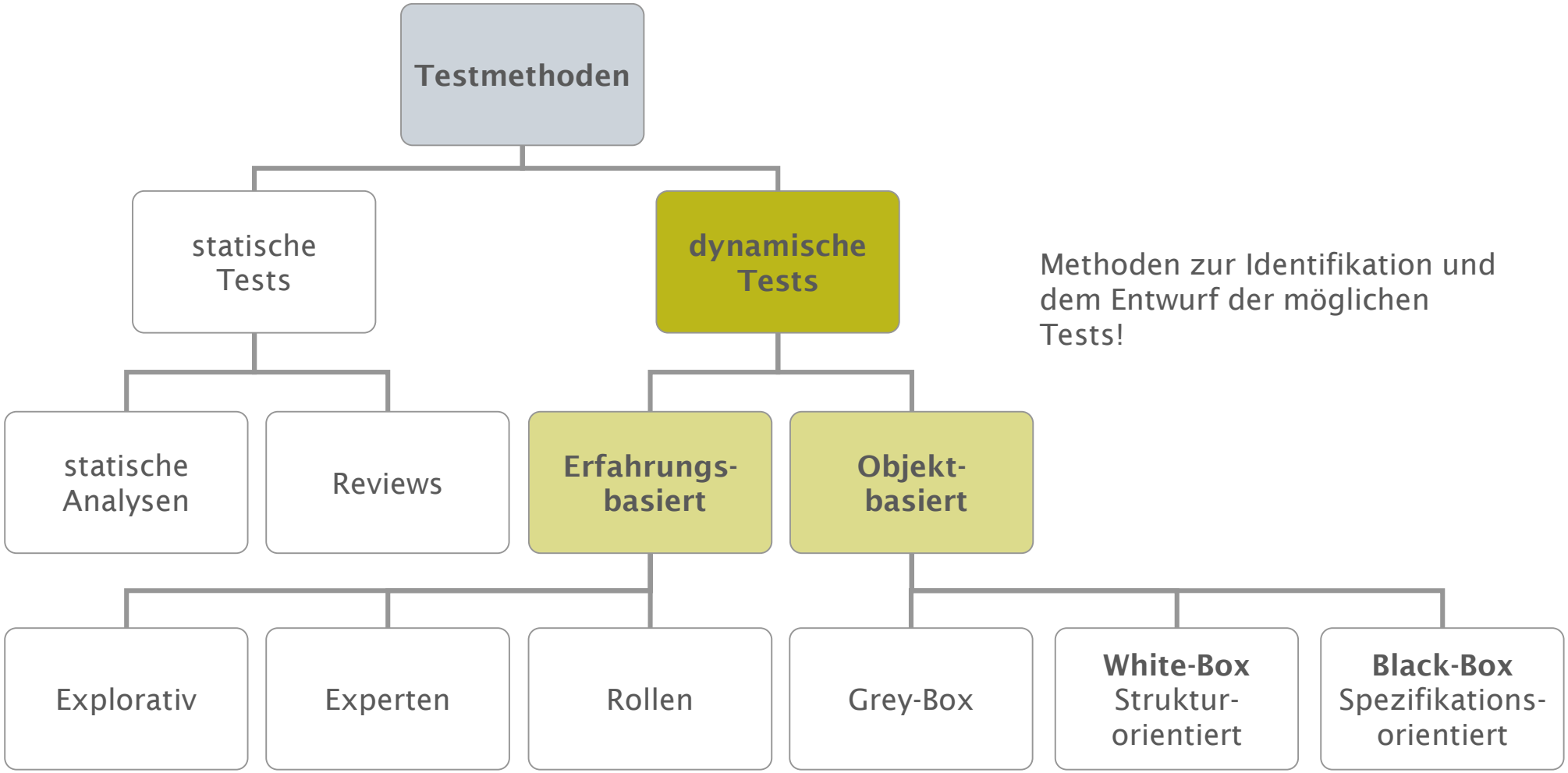
Statische Analyse

Statische Analyse [nach ISTQB_DE]

- Durchführung der Analyse eines Artefaktes (z.B. Anforderung oder Quelltext)
- Erfolgt ohne Ausführen des Programmcodes (bzw. ohne Betreiben des Prüfobjektes)
- Voraussetzung ist eine definierte formale Struktur des Prüfobjektes

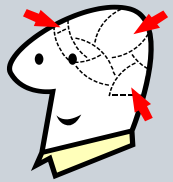
Reviews und Statische Analysen finden Fehlerzustände eher als Fehlerwirkungen!

Übersicht über dynamische Tests



Methoden zur Identifikation und dem Entwurf der möglichen Tests!

Erfahrungs- vs. Objektbasierte Verfahren



Erfahrungsbasierte Verfahren

Erfahrungsbasierte Verfahren [nach ISTQB_DE]

- Testfälle auf Basis von Wissen, Erfahrung und Intuition
  - → Auch Aufdecken von Spezifikationsmängeln!
  - → Finden typischer Fehlerzustände (intuitive Testfallermittlung)

Objektbasierte Verfahren

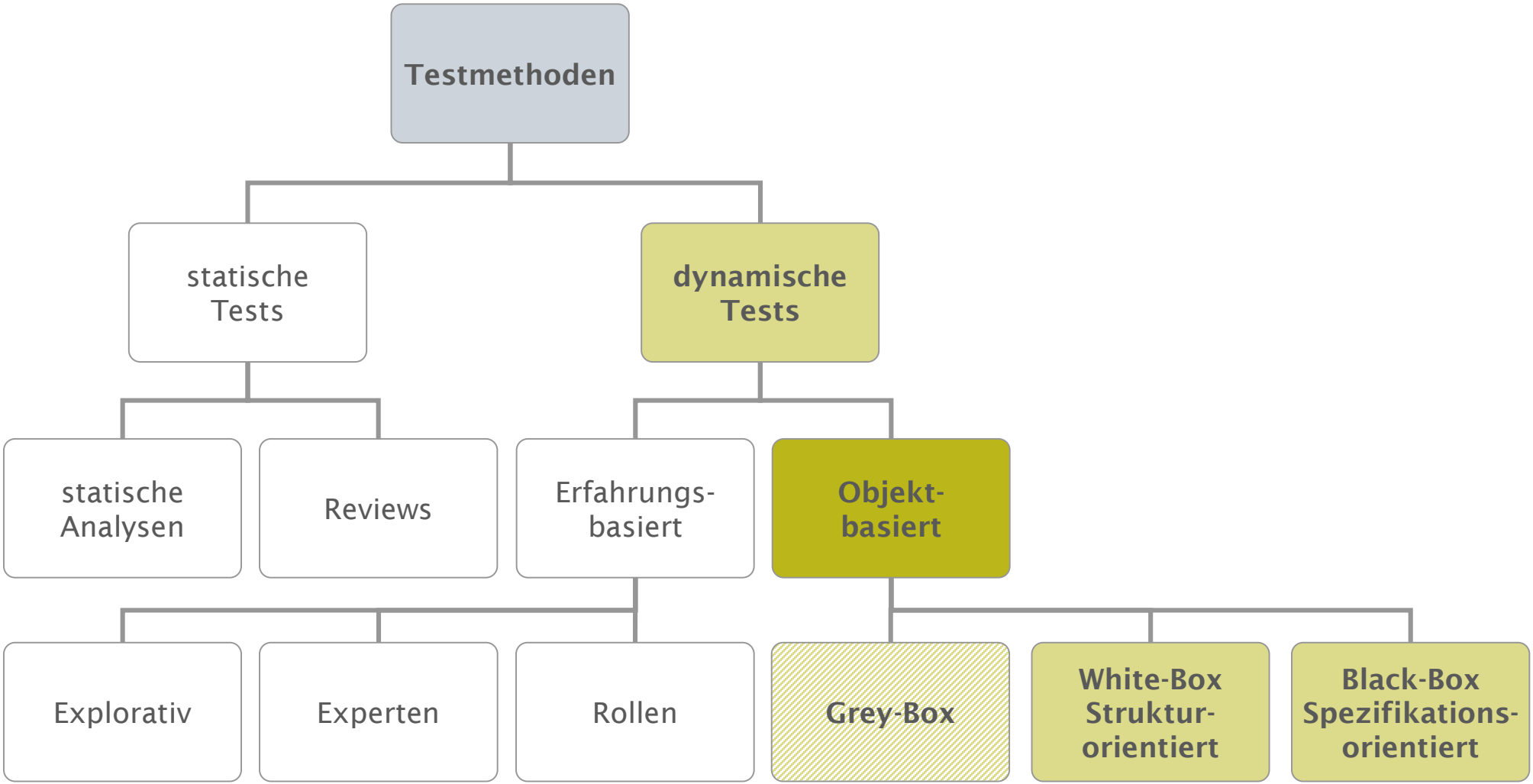
Nicht zu verwechseln mit objektorientiert!

Objektbasierte Verfahren

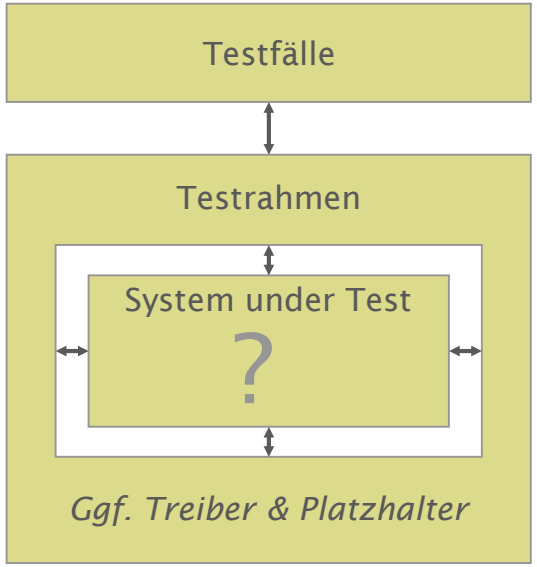
- Systematische Testfallentwicklung
- Testfälle auf Basis von Geschäftsobjekten (wie Spezifikationen, Zeichnungen, Diagrammen etc.)
  - → Aufdecken von Spezifikationsverletzungen!

Dynamische Tests finden Fehlerwirkungen eher als Fehlerzustände!

Black-Box Testentwurfsverfahren



Prinzip der Black-Box-Tests



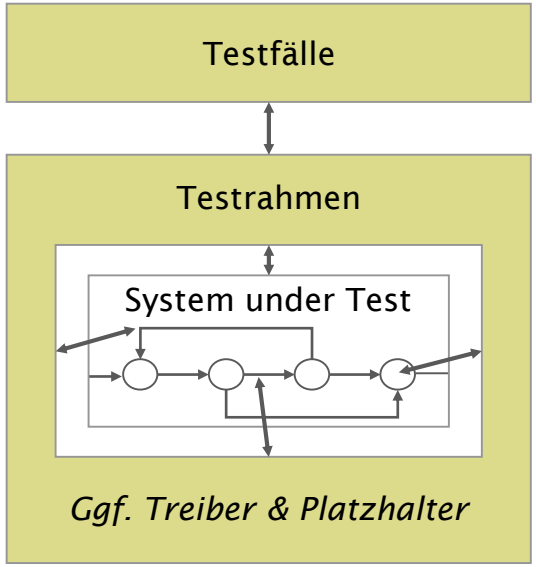
Black-Box-Test (spezifikationsorientierte/-basierte Testentwurfsverfahren) [ISTQB_DE]:

Funktionales oder nicht-funktionales Testen ohne Nutzung von Informationen über Interna eines Systems oder einer Komponente

Merkmale:

- Testfall und Testdaten werden aus Spezifikationen des Testobjekts systematisch abgeleitet
- Testbasis können sein:
  - formal / informell
  - funktionale / nicht-funktionale
  - modellhaft (→ Modellbasiertes Testen)
- Die Struktur („wie“ ist es implementiert) wird nicht ausgewertet. (→ White-Box-Tests)

Prinzip der White-Box-Tests



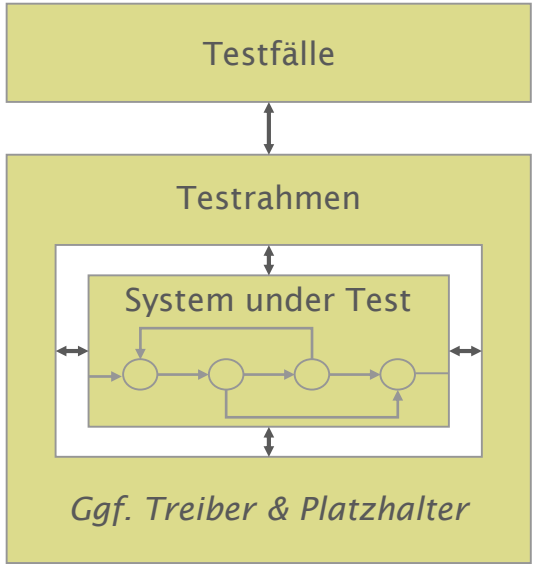
White-Box-Test (strukturorientierte/-basierte Testentwurfsverfahren) [ISTQB_DE]:

Ein Test, der auf der Analyse der internen Struktur einer Komponente oder eines Systems basiert

Merkmale:

- Testfall und Testdaten aus der Struktur des Testobjekts abgeleitet
- Testbasis können sein:
  - SW-Code / SW-Design (Codebasierte Testfallableitung)
  - Signal-/Daten-/Kontroll-Flussdiagramme
- Es kann das Maß der Strukturabdeckung gemessen werden
  → Weitere Testfälle können zur Erhöhung des Überdeckungsgrades systematisch abgeleitet werden!

Prinzip der Grey-Box-Tests



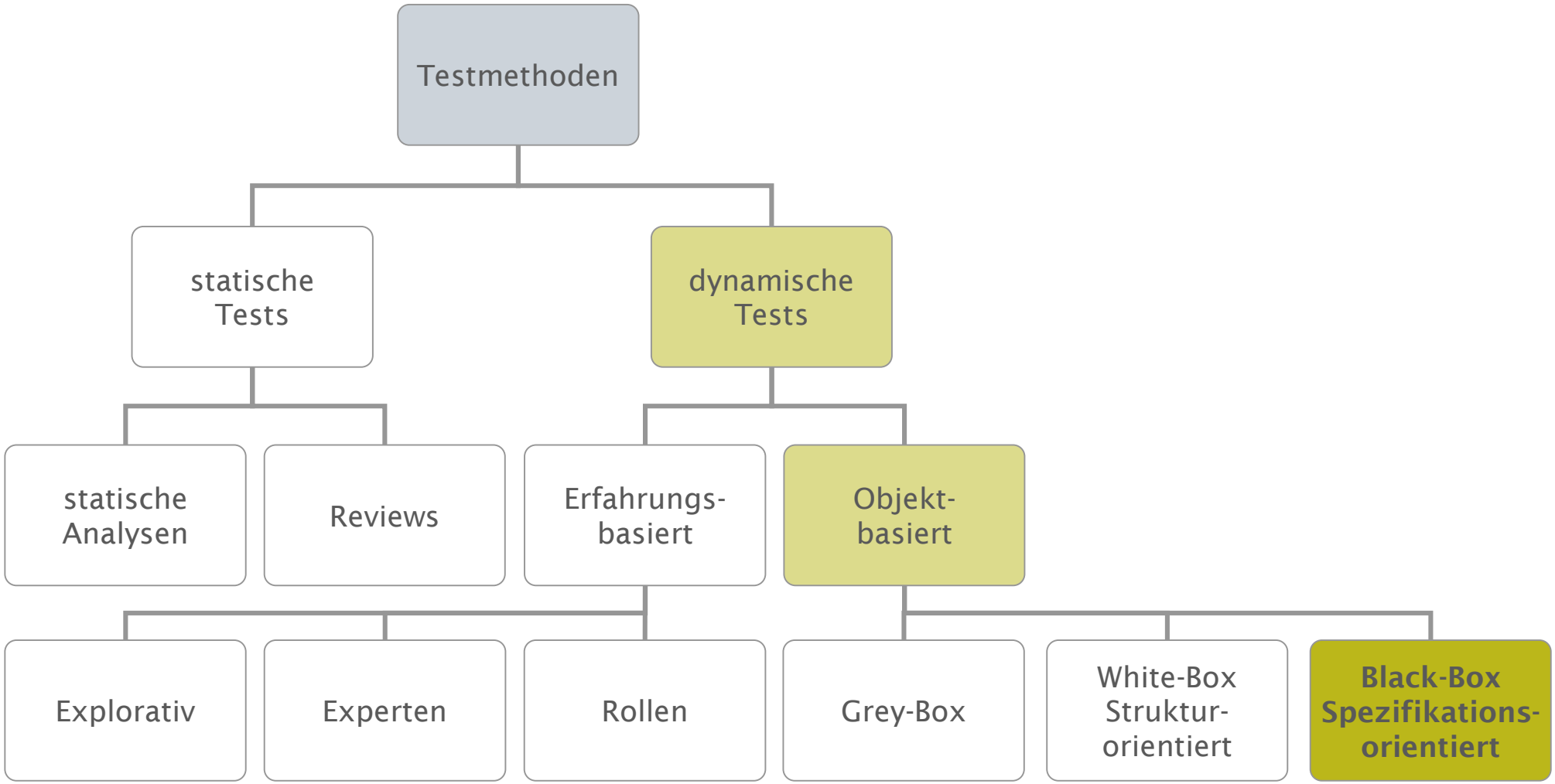
Grey-Box-Test (u.a. Testgetriebene Entwicklung) [nach wiki_de]

- Verbindet Vorteile von Black-Box-Tests und White-Box-Tests
- Kommt in der testgetriebenen Entwicklung zum Einsatz

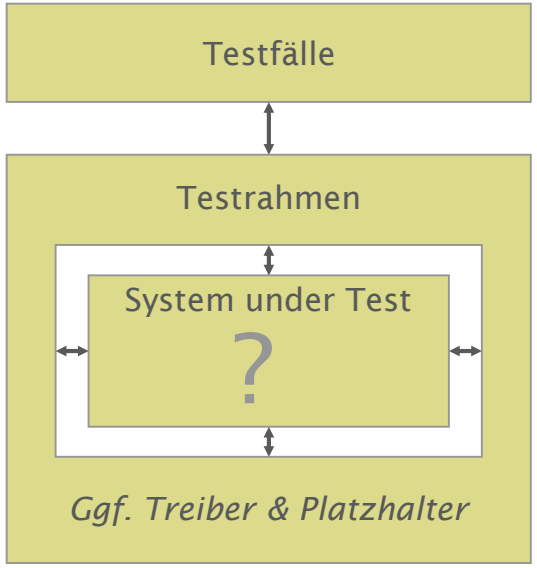
Testgetriebene Entwicklung [nach Spillner_FL]

- Tests werden vor der Entwicklung geschrieben. (z.B. Crash-Tests (System), Extreme Programming (Komponente))
- Das Produkt wird so lange verbessert, bis die Tests fehlerfrei absolviert sind

Black-Box Testentwurfsverfahren



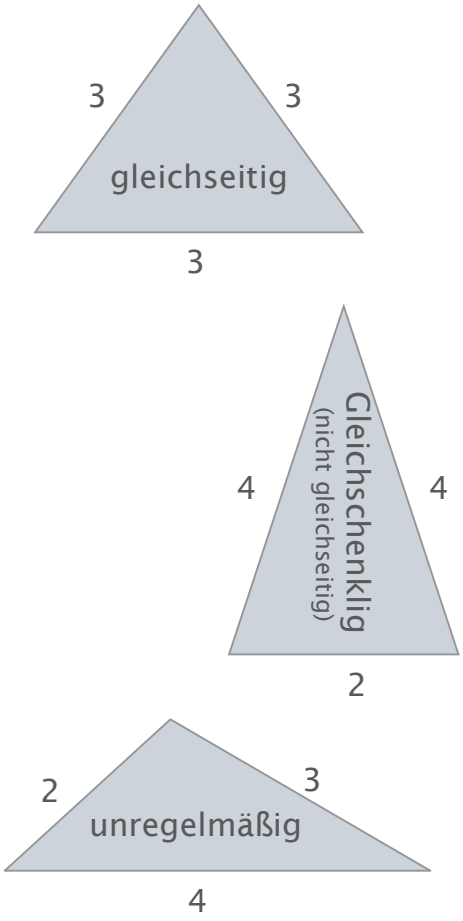
Überblick über Black-Box-Testentwurfsverfahren



Typische Testentwurfsverfahren:

- Äquivalenzklassenbildung
  (Äquivalenzklassenmethode)
- Grenzwertanalyse
- Entscheidungstabellentest
- Zustandsbasierter Test
- Anwendungsfallbasierter Tests

Übung 8: Äquivalenzklassenbildung



Dreieck-Test

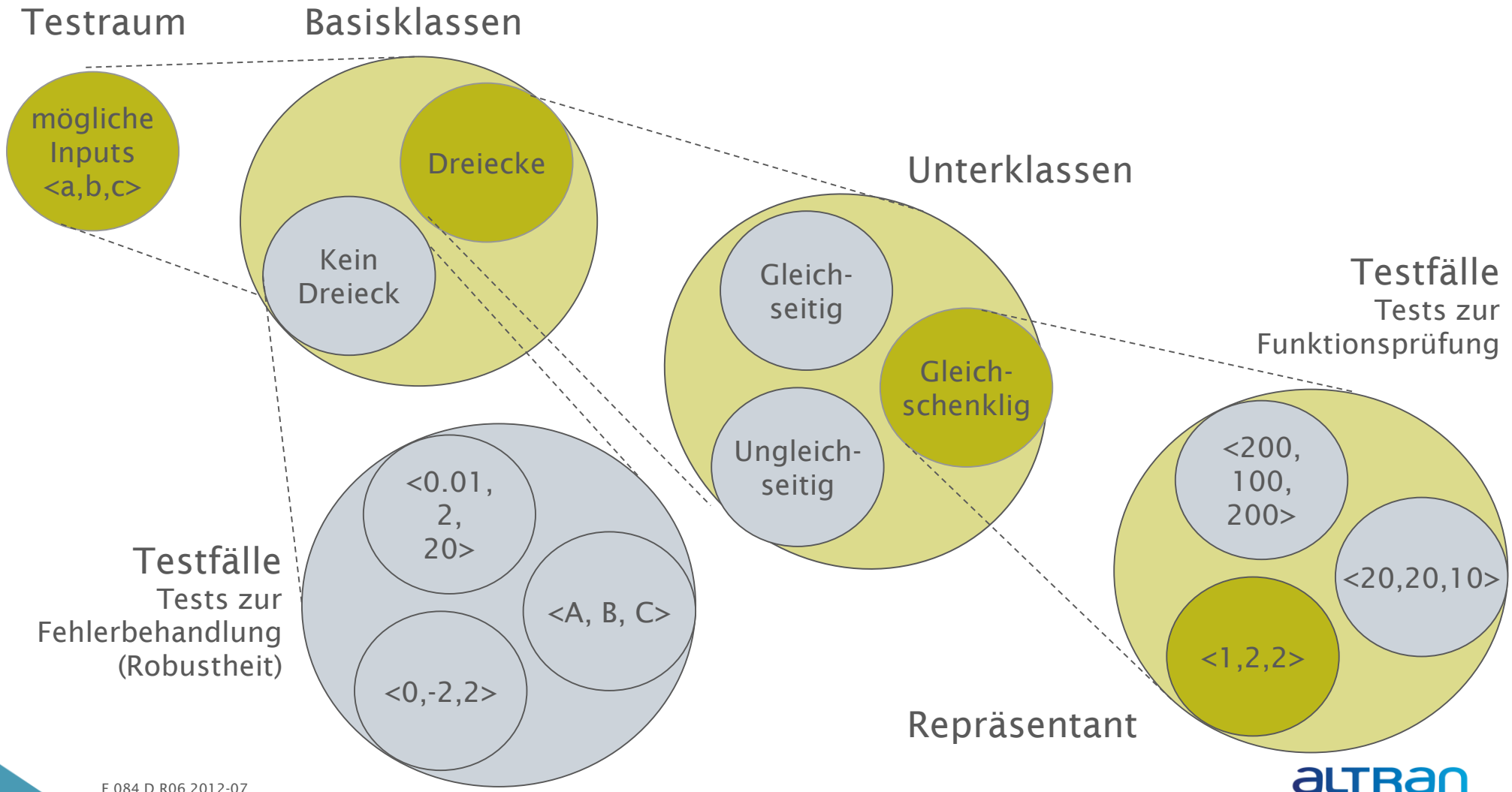
- Das Programm liest 3 ganzzahlige Werte ein
- Die Zahlen werden als Längen von Dreiecksseiten interpretiert
- Das Programm gibt aus welche Art von Dreieck vorliegt:
  - Unregelmäßig
  - Gleichschenklig
  - Gleichseitig

Aus: [Myers]

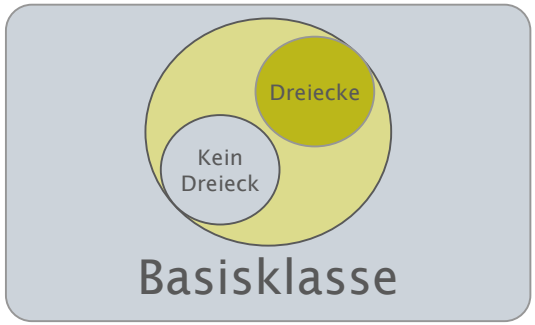
Frage:

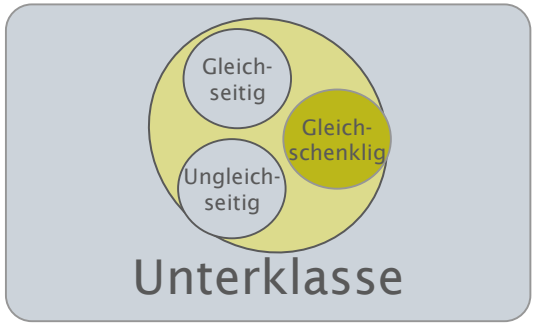
Wie viele Testfälle sind nötig, um das Programm zu testen?

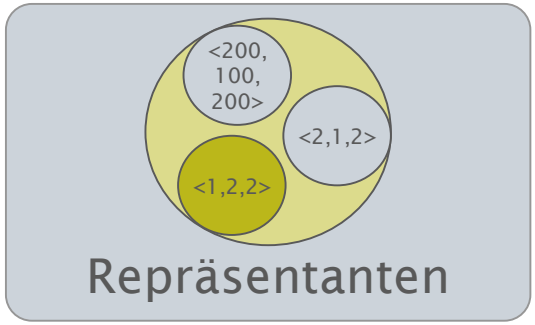
Äquivalenzklassenbildung



Vorgehen zur Äquivalenzklassenbildung







Für alle Eingabewerte/-größen:

- Definitionsbereiche ermitteln, bei denen man von einem ähnlichen Verhalten ausgeht
- Klasse zulässiger Werte bilden (→„Dreiecke“)
- Klasse unzulässiger Werte bilden (→„kein Dreieck“)

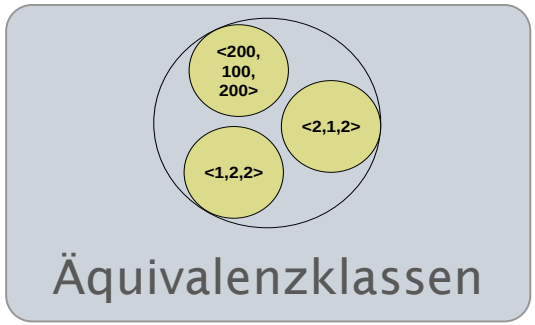
Klassen in Unterklassen verfeinern:

- Klassenelemente mit vermutlich unterschiedlicher Verarbeitung eigenen Unterklassen zuteilen
  (→„Dreiecke“=„gleichseitig“ & „unregelmäßig“ & „gleichschenklig“)

Repräsentanten auswählen:

- Für jede Klasse einen repräsentativen Wert auswählen
  (ggf. unter Berücksichtigung der Risiken, Wahrscheinlichkeit, etc.)

Merkmale der Äquivalenzklassenbildung (-methode)



Ziel der Äquivalenzklassenbildung:

- Strukturiertes Herleiten von Testfällen zur Testabdeckung der Klassen
- hohe Fehlerentdeckungswahrscheinlichkeit bei minimaler Anzahl von Testfällen

Die Klassen können gebildet werden für

- Ausgabewerte
  (z.B. beim Komponenten- & Systemtest)
- interne Werte
  (z.B. beim Komponententest)
- zeitbezogene Werte
  (z.B. vor oder nach einem Ereignis)
- Schnittstellenparameter
  (z.B. beim Integrationstest)

Überdeckungsgrad

- Anzahl aller Äquivalenzklassen zu getesteten.
  (Überdeckungsziele im Bezug auf Eingabe oder Ausgabewerte)

→ *Äquivalenzklassenbildung kann in allen Teststufen angewandt werden.*

Übung 2.2: Äquivalenzklassenbildung (-methode)

Schreiben eines Testfalls unter Verwendung der **Äquivalenzklassenbildung**

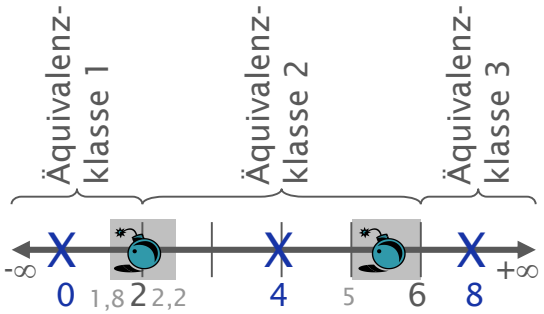
Risiko bei der Wahl der Repräsentanten

Die Wahl der Repräsentanten erfolgt aufgrund der Erfahrung bzw. typischer Nutzungsszenarien und zudem möglichst aus der Mitte heraus.

→ Fehlerhafte Grenzbereiche werden nur unzureichend entdeckt:

- Toleranzüberschreitung
  (hier: $2^{\pm 10\%}$ anstatt $2^{\pm 5\%}$)
- Fehlerhafte Grenzwertparametrisierung
  (hier: 5 anstatt 6)

→ Fehlerzustände treten häufig an den Grenzbereichen der Äquivalenzklassen auf!

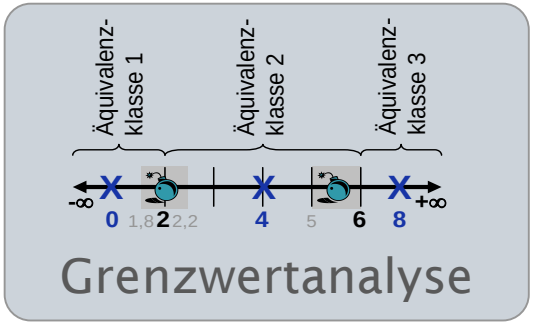


Nach der Äquivalenzklassenanalyse sollte eine Grenzwertanalyse folgen!

Auswahl der Grenzwerte

- Der größte und der kleinste Wert einer Klasse sind deren Grenzwerte
- Für Tests nutzt man (wenn möglich) den exakten Grenzwert und den benachbarten Wert der nächsten Klasse

Merkmale der Grenzwertanalyse



Grenzwertanalyse

Ziel der Grenzwertanalyse:

- Strukturierte Auswahl von Testdaten
- Vergleichsweise einfach anwendbar mit hohem Potential Fehlerzustände aufzudecken

Für die Erstellung sind detaillierte Spezifikationen hilfreich

Die Grenzwertanalyse kann angewendet werden bei:

- Bildung von Äquivalenzklassen
- Kommunikationstests
  (z.B. Timing, Jitter bei Integrationstests)
- Benutzereingaben
  (z.B. Zeitspannen, Timeouts bei Systemtests)

Grenzen von Tabellenbereichen
(z.B. Tabellengröße 256*256 bei Komponententests)

→ *Grenzwertanalyse kann in allen Teststufen angewandt werden.*

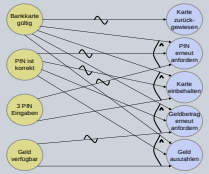
Überdeckungsgrad

$$\text{GW-Überdeckung} = \frac{\text{Anzahl getestete GW}}{\text{Gesamtzahl GW}} * 100\%$$

Übung 2.3: Grenzwertanalyse

Schreiben eines Testfalls unter Verwendung der **Grenzwertanalyse**

Test von Abhängigkeiten zwischen Ursache und Wirkung



Ursachen-Wirkungs-Graph Analyse

Nachteile der Äquivalenzklassenmethode und Grenzwertanalyse:

- Es werden nur mögliche Eingangsbedingungen einzeln herangezogen
- Abhängigkeiten (zwischen Ursache und Wirkung) werden nicht betrachtet

→ Einsatz der Ursache-Wirkungs-Graph Analyse

→ Verfahren, um Abhängigkeiten bei der Testfallerstellung zu berücksichtigen

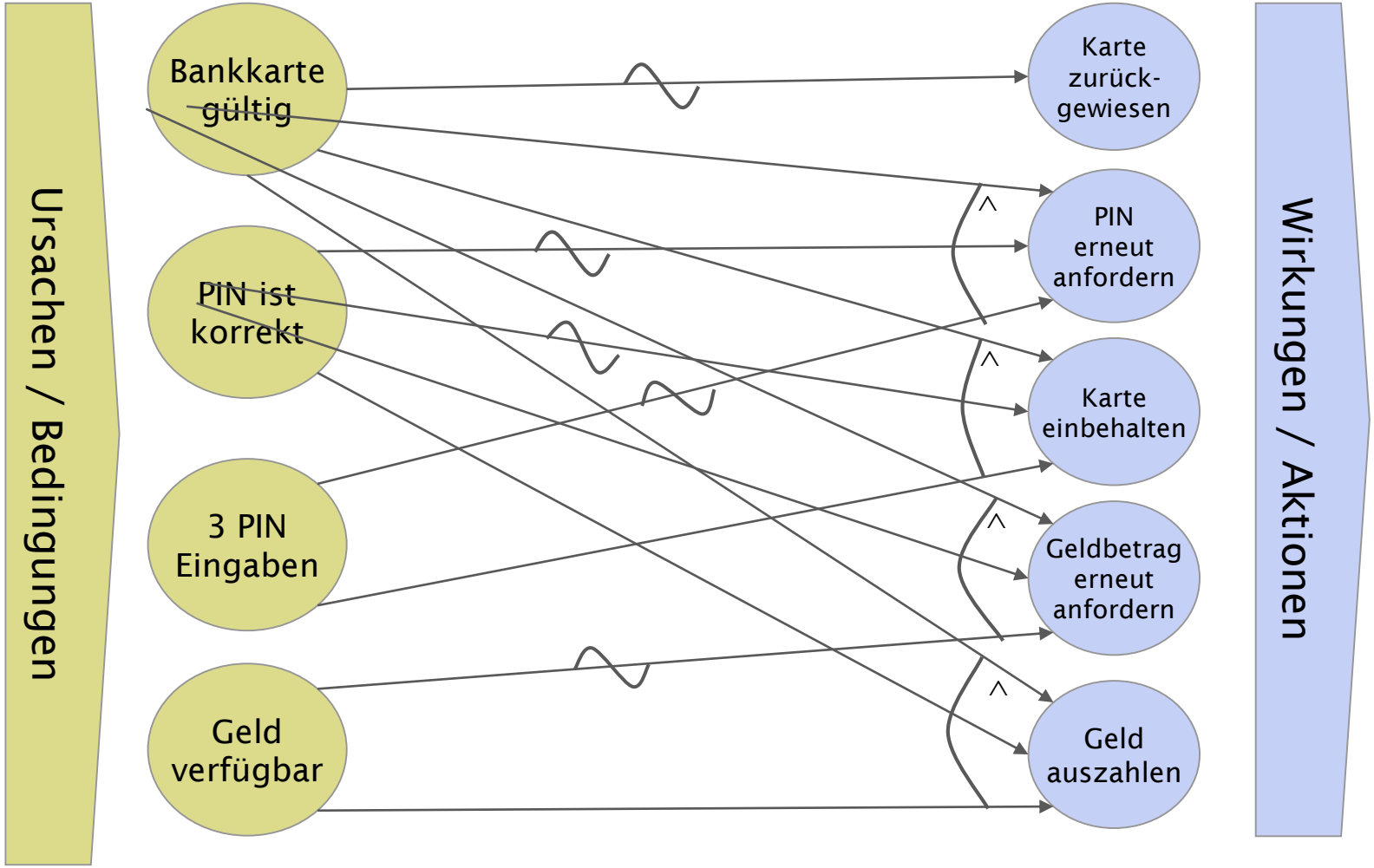
Voraussetzung:

Ursachen und Wirkung müssen aus der Spezifikation ermittelt werden können!

logische UND-Verknüpfung

Negation

Beispiel für Ursache-Wirkungs-Graph Analyse



Aus: Spillner / Linz, Basiswissen Softwaretest, dpunkt Verlag, Heidelberg, 2006

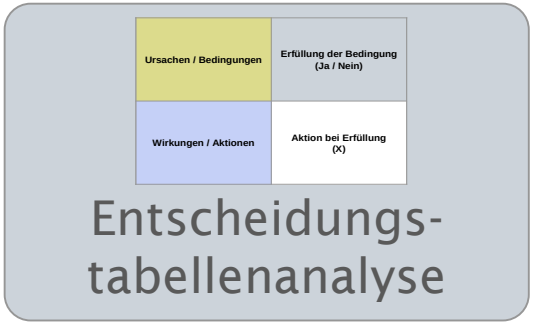
Transfer in eine Entscheidungstabelle

Ursachen / Bedingungen	Erfüllung der Bedingung (Ja / Nein)
Wirkungen / Aktionen	Aktion bei Erfüllung (Ja / Nein)

Beispiel für Entscheidungstabelle

Entscheidungstabelle		Testfall 1 (TF1)	Testfall 2 (TF2)	Testfall 3 (TF3)	Testfall 4 (TF4)	Testfall 5 (TF5)
Bedingung	Bankkarte gültig	Nein	Ja	Ja	Ja	Ja
	PIN ist korrekt	-	Nein	Nein	Ja	Ja
	3 PIN Eingaben	-	Nein	Ja	-	-
	Geld verfügbar	-	-	-	Nein	Ja
Aktion	Karte zurückgewiesen	Ja	-	-	-	-
	PIN erneut anfordern	-	Ja	-	-	-
	Karte einbehalten	-	-	Ja	-	-
	Geldbetrag erneut anfordern	-	-	-	Ja	-
	Geld auszahlen	-	-	-	-	Ja

Merkmale der Entscheidungstabellenanalyse



Stärken:

- Ableiten der Kombinationen von (logischen) Bedingungen, die beim Test möglicherweise nicht ausgeführt worden wären
- Anwendung wenn Abläufe von mehreren logischen Bedingungen abhängen

Eigenschaften:

- Eingabebedingungen und Aktionen werden (meist) „wahr“ oder „falsch“ gesetzt
- Jede Spalte der Tabelle entspricht einer Regel im Geschäftsprozess die getestet werden muss

Überdeckungsgrad

- Standardüberdeckungsgrad: wenigstens ein Testfall pro Spalte

Mögliche Bedingungskombinationen

		TF 1	TF 2	TF 3	TF 4
Bedingung	Bedingung 1	N	J	J	N
	Bedingung 2	N	N Don't care J		J
Aktion	Aktion 1	X	-	-	-
	Aktion 2	-	X	X	-
	Aktion 3	-	-	-	X

Anzahl der Bedingungskombinationen:

- Bei n Bedingungen gibt es genau 2^n Bedingungskombinationen!
  (hier: n = 2 Bedingungen → 2^2 = 4Kombinationen)

Reduzierung der Bedingungskombinationen:

- Einführung von „don't care" Elementen wenn:
  - Die Bedingung irrelevant ist („Don't care")
    (hier: unabhängig von Bedingung 2 führen TF 2 / TF 3 immer zu Aktion 2)

Risiko:

- Auch wenn lt. Definition eine Bedingung keinen Einfluss haben darf („don't care"), kann der Nachweis der Unabhängigkeit notwendig sein!

Prüfen auf Redundanz und Vollständigkeit

		TF 1	TF 2/3	TF 4
Bedingung	Bedingung 1	N	J	N
	Bedingung 2	N	Don't care	J
Zähler		**1**	**2**	**1**
Aktion	Aktion 1	X	-	-
	Aktion 2	-	X	-
	Aktion 3	-	-	X

n: Anzahl der Äquivalenzklassen der Bedingung

Prüfen der Entscheidungstabelle über Checksumme:

- Bedingungen ohne „don't care" zählen „1"
  (hier: TF4 und TF 1)
- Bedingungen mit „don't care" zählen „2"
  (hier: TF2/3)
- Zähler durch Multiplikation aller Bedingungen (J/N = 1, don't care = 2) bilden

Unvollständigkeit:

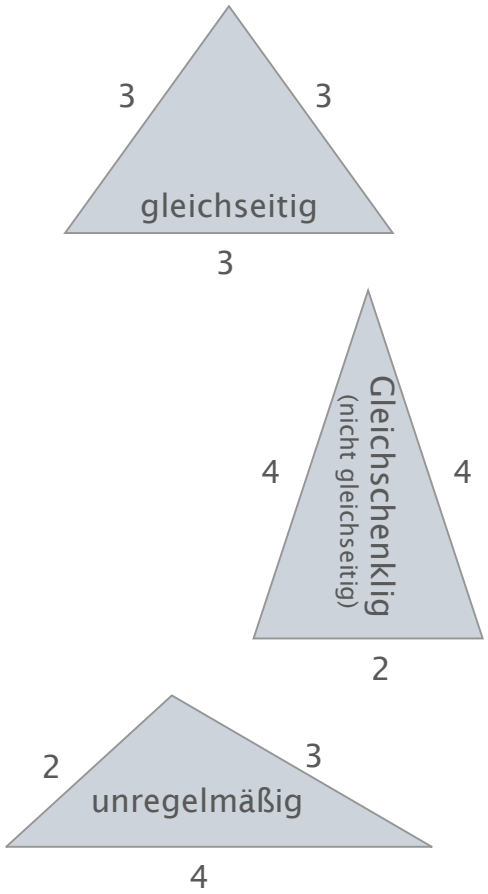
- Wenn die Summe der Zähler $<2^n$ ergibt

Redundanz:

- Wenn die Summe der Zähler $>2^n$ ergibt

hier: *Zähler = (1*1)+(1*2)+(1*1) = 4 == 2^2*
→ Die Entscheidungstabelle ist deterministisch.

Beispiel zur Erstellung einer Entscheidungstabelle



Dreieck-Test

- Das Programm liest 3 ganzzahlige Werte ein
- Die Zahlen werden als Längen von Dreieckseiten interpretiert
- Das Programm gibt aus welche Art von Dreieck vorliegt:
  - Unregelmäßig
  - Gleichschenklig
  - Gleichseitig
  - Kein Dreieck

Frage:

Wie viele Testfälle sind mindestens nach der Entscheidungstabellen-methode notwendig?

1. Vollständige Entscheidungstabelle

	2^6=64 Bedingungen	TF1	TF2	TF3	TF4	TF5	TF6	TF7	TF8	TF9	TF10	TF11
Bedingung	a < b + c	N	J	J	J	J	J	J	J	J	J	J
	b < c + a	-	N	J	J	J	J	J	J	J	J	J
	c < a + b	-	-	N	J	J	J	J	J	J	J	J
	a = b	-	-	-	J	J	J	J	N	N	N	N
	a = c	-	-	-	J	J	N	N	J	J	N	N
	b = c	-	-	-	J	N	J	N	J	N	J	N
Aktion	kein Dreieck	X	X	X								
	gleichseitig				X							
	gleichschenklig							X		X	X	
	ungleichschenklig											X
	unlogisch					X	X		X			

- Vollständige Befüllung der resultierenden Aktionen
- Vollständige Befüllung aller möglichen Bedingungen
- Vervollständigen der „unlogischen“ Bedingungen die sich ausschließen

2. Prüfen auf Vollständigkeit und Inkonsistenz

2^6=64 Bedingungen		TF1	TF2	TF3	TF4	TF5	TF6	TF7	TF8	TF9	TF10	TF11
Bedingung	a < b + c	N	J	J	J	J	J	J	J	J	J	J
	b < c + a	-	N	J	J	J	J	J	J	J	J	J
	c < a + b	-	-	N	J	J	J	J	J	J	J	J
	a = b	-	-	-	J	J	J	J	N	N	N	N
	a = c	-	-	-	J	J	N	N	J	J	N	N
	b = c	-	-	-	J	N	J	N	J	N	J	N
Checksumme = 64		32	16	8	1	1	1	1	1	1	1	1
Aktion	kein Dreieck	X	X	X								
	gleichseitig				X							
	gleichschenklig							X		X	X	
	ungleichschenklig											X
	unlogisch					X	X		X			

- Über Checksumme prüfen auf:
  - Inkonsistenz
  - Redundanzen
  - Vollständigkeit

3. Konsolidieren

2^6=64 Bedingungen		TF1	TF2	TF3	TF4	TF7	TF9	TF10	TF11
Bedingung	a < b + c	N	J	J	J	J	J	J	J
	b < c + a	-	N	J	J	J	J	J	J
	c < a + b	-	-	N	J	J	J	J	J
	a = b	-	-	-	J	J	N	N	N
	a = c	-	-	-	J	N	J	N	N
	b = c	-	-	-	J	N	N	J	N
Aktion	kein Dreieck	X	X	X					
	gleichseitig				X				
	gleichschenklig					X	X	X	
	ungleichschenklig								X

- Löschen „unlogischer“ Konditionen, wenn diese nicht explizit getestet werden sollen

4. Zusammenfassen

2^6=64 Bedingungen		TF1	TF2	TF3	TF4	TF7	TF9	TF10	TF11
Bedingung	a < b + c	N	J	J	J	J	J	J	J
	b < c + a	-	N	J	J	J	J	J	J
	c < a + b	-	-	N	J	J	J	J	J
	a = b	-	-	-	J	J	N	N	N
	a = c	-	-	-	J	N	J	N	N
	b = c	-	-	-	J	N	N	J	N
Aktion	kein Dreieck		ELSE						
	gleichseitig				X				
	gleichschenklig					X	X	X	
	ungleichschenklig								X

- Ggf. Zusammenfassen von Regeln, die durch eine ELSE-Regel zur gleichen Aktion führen.
  (Es darf nur eine ELSE-Regel je Tabelle geben!)

Gegenüberstellung zweier Entwurfsverfahren

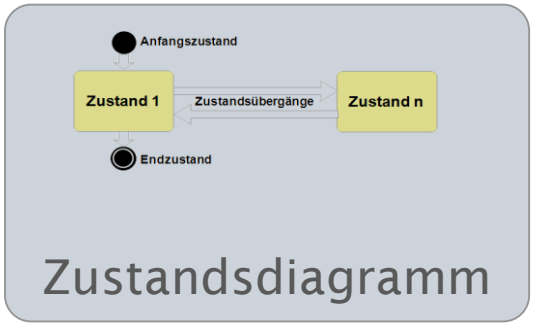
2^6=64 Bedingungen		TF4	TF7	TF9	TF10	TF11	ELSE
Bedingung	a < b + c	J	J	J	J	J	
	b < c + a	J	J	J	J	J	
	c < a + b	J	J	J	J	J	
	a = b	J	J	N	N	N	
	a = c	J	N	J	N	N	
	b = c	J	N	N	J	N	
Aktion	kein Dreieck						X
	gleichseitig	X					
	gleichschenklig		X	X	X		
	ungleichschenklig					X	

- Äquivalenzklassenmethode:
  - → Führt hier zu 6 Testfällen (siehe S.75)
- Entscheidungstabelle:
  - → Führt zu maximal 64 theoretisch möglichen Testfällen
    - → Geeignet für Testautomatisierung
  - → Kann auf ebenfalls 6 Testfälle reduziert werden!
    - → Wechselwirkungen nicht identifizierbar
      (theoretisch weitere 58 Kombinationen)

Übung 2.4: Entscheidungstabellenanalyse

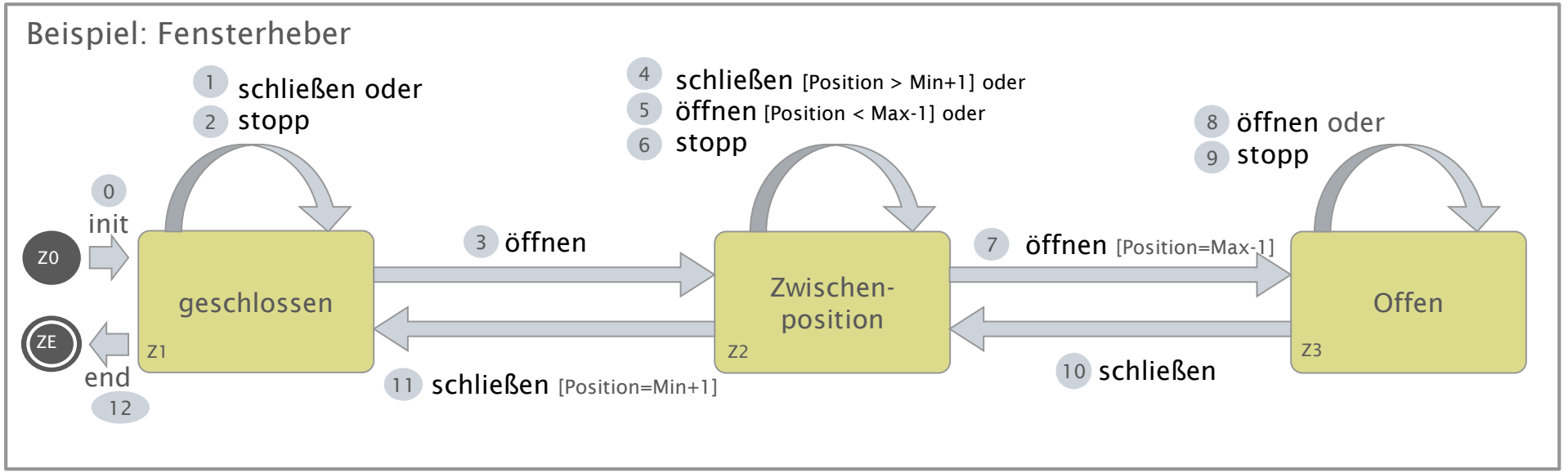
Schreiben eines Testfalls unter
Verwendung der
Entscheidungstabellenanalyse

Zustandsbasierte Tests



Zustandsdiagramm [IEEE 610]

Ein Diagramm, das die Zustände beschreibt, die ein System oder eine Komponente annehmen kann, und die Ereignisse bzw. Umstände zeigt, die einen Zustandswechsel verursachen und/oder ergeben.



Zustandsbasierte Tests

Zustands-
Übergangstabelle

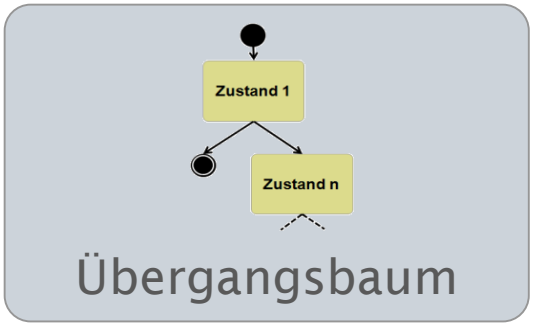
Zustandsübergangstabelle [ISTQB_DE]

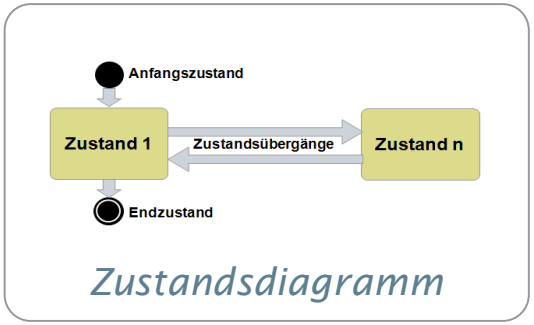
Eine Tabelle, die für jeden Zustand in Verbindung mit jedem möglichen Ereignis die resultierenden Übergänge darstellt.

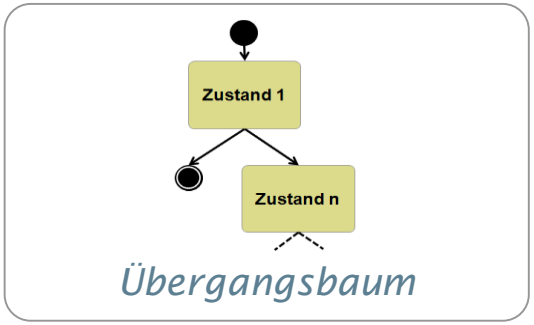
Das können sowohl gültige als auch ungültige Übergänge sein

Zustandsübergangstabelle (Beispiel Fensterheber)			Zustandsübergänge												
			0	1	2	3	4	5	6	7	8	9	10	11	12
Zustände	Z0	Anfangszustand	Z1	-	-	-	-	-	-	-	-	-	-	-	-
	Z1	Geschlossen	-	Z1	Z1	Z2	-	-	-	-	-	-	-	-	ZE
	Z2	Zwischenposition	-	-	-	-	Z2	Z2	Z2	Z3	-	-	-	Z1	-
	Z3	Offen	-	-	-	-	-	-	-	-	Z3	Z3	Z2	-	-
	ZE	Endzustand	-	-	-	-	-	-	-	-	-	-	-	-	-

Zustandsbasierte Tests



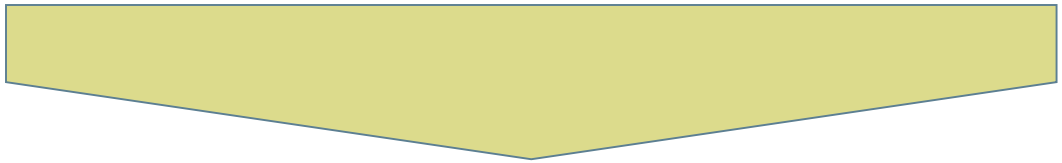




Übergangsbaum [nach Sneed]

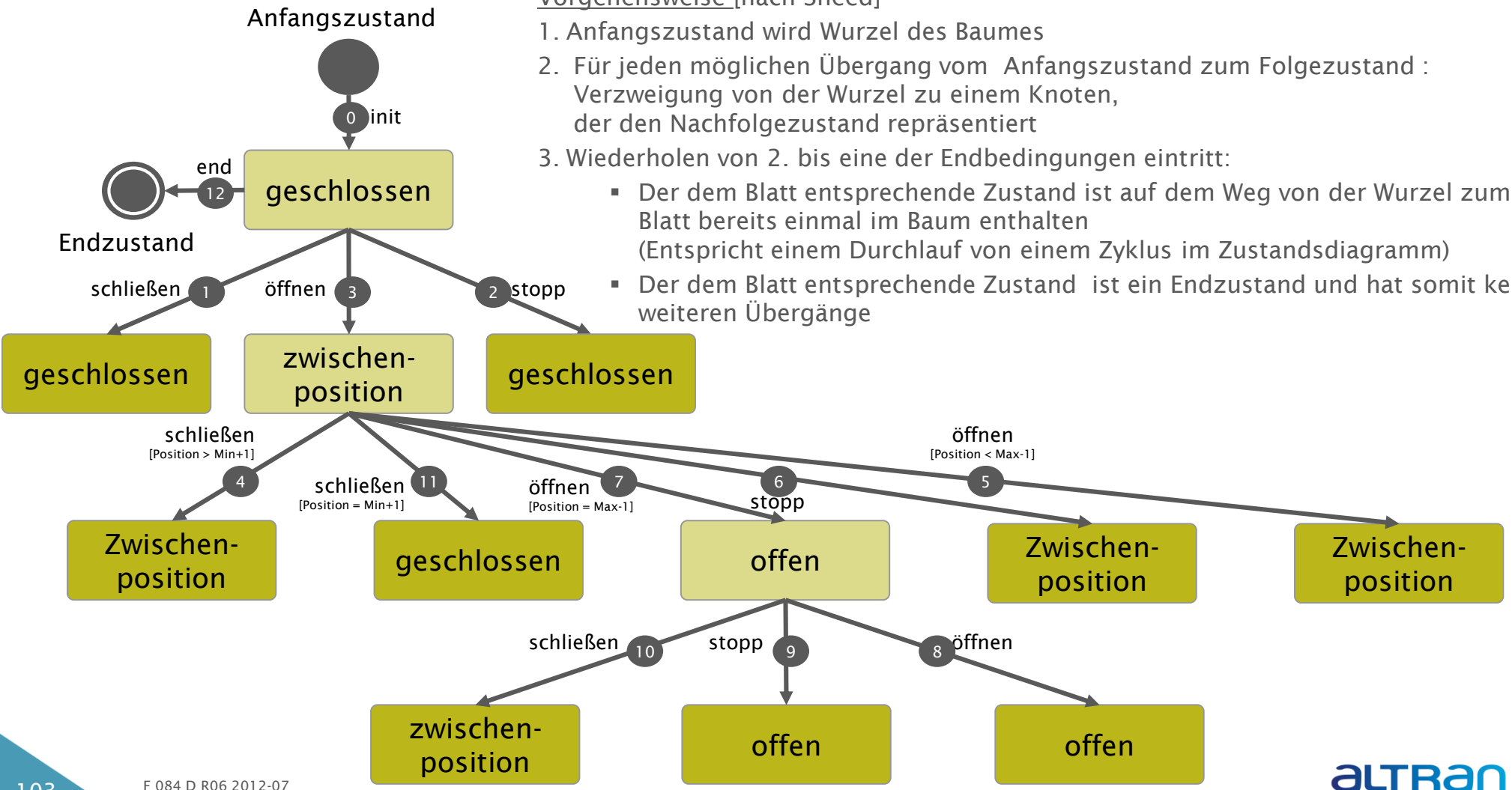
Ermittlung notwendiger Testfälle durch Überführen des Zustandsdiagramms in einen Übergangsbaum

Zyklisches Zustandsdiagramm mit potentiell unendlichen Folgen



Übergangsbaum mit repräsentativer Menge von Zuständen
(ohne Zyklen)

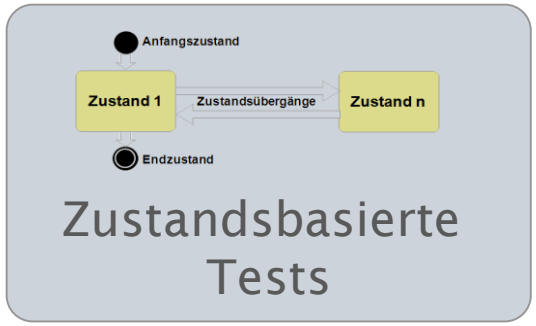
Zustandsbasierte Tests (Beispiel: Übergangsbaum)



Vorgehensweise [nach Sneed]

1. Anfangszustand wird Wurzel des Baumes
2. Für jeden möglichen Übergang vom Anfangszustand zum Folgezustand : Verzweigung von der Wurzel zu einem Knoten, der den Nachfolgezustand repräsentiert
3. Wiederholen von 2. bis eine der Endbedingungen eintritt:
   - Der dem Blatt entsprechende Zustand ist auf dem Weg von der Wurzel zum Blatt bereits einmal im Baum enthalten (Entspricht einem Durchlauf von einem Zyklus im Zustandsdiagramm)
   - Der dem Blatt entsprechende Zustand ist ein Endzustand und hat somit keine weiteren Übergänge

Zustandsbasierte Tests



Zustandsbasierte Tests [ISTQB_DE]

Testentwurfsverfahren, mit dem Testfälle entworfen werden, um gültige und ungültige Zustandsübergänge zu prüfen

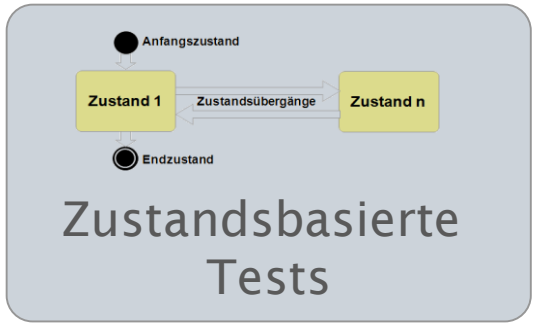
Stärken:

- Berücksichtigt die Vorgeschichte des Systems
- Es können ungültige Übergänge aufgezeigt werden

Anwendungsbereiche:

- Embedded Software
- Automatisierungstechnik
- dialogbasierte Abläufe
  (z.B. für Internet-Anwendungen oder Geschäftsszenarien)

Zustandsbasierte Tests



Überdeckungsgrad

Es gibt verschiedene Kriterien für die Berechnung des Überdeckungsgrades

- Jeder Zustand wird mind. einmal erreicht
- Jeder Zustandsübergang wurde einmal ausgeführt
- Alle Kombinationen von Zustandsübergängen
- Alle Zustandsübergänge in jeder beliebigen Reihenfolge mit allen möglichen Zuständen, auch mehrfach hintereinander

Sehr große Zahl der benötigten Testfälle macht Einschränkung der Anzahl von Kombinationen erforderlich

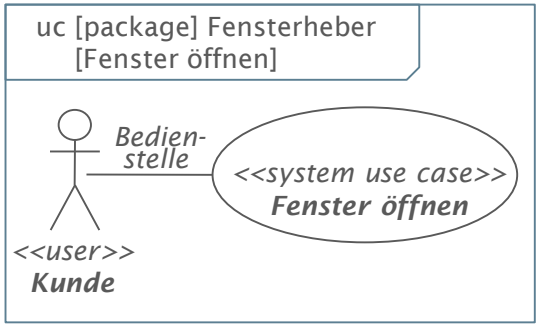
Aus: Spillner / Linz,
Basiswissen Softwaretest,
dpunkt Verlag, Heidelberg, 2006

Übung 2.5: Zustandsbasierter Test

Schreiben eines Testfalls unter Verwendung des **Zustandsbasierten Testentwurfsverfahrens**

Anwendungsfälle (Use-Case)

Anwendungsfälle



Beispiel Fensterheber:

Fenster öffnen *(fachliches Ergebnis)*
wenn der Kunde *(Akteur)* die
Bedienstelle „Fenster öffnen“ betätigt
(fachlicher Auslöser)

Anwendungsfall (engl. *Use-Case*) [ISTQB_DE]

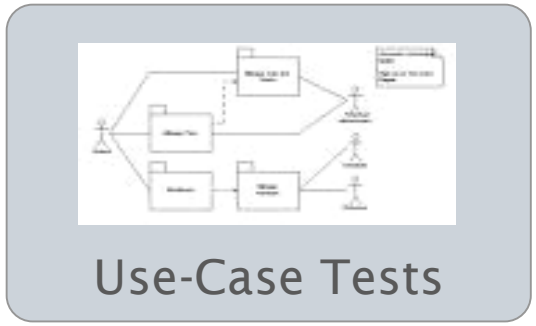
Ein Anwendungsfall beschreibt eine Reihe von Vorgängen in einem Dialog zwischen einem Benutzer und einem System, die zu einem konkreten Ergebnis führen

Eigenschaften (SysML) [nach Weilkiens]

- Mindestens 1 Akteur
- Genau ein fachlicher Auslöser
- Endet mit einem fachlichen Ergebnis
- Ablauf zwischen Auslöser und Ergebnis ist zeitlich zusammenhängend (zeitliche Kohärenz)

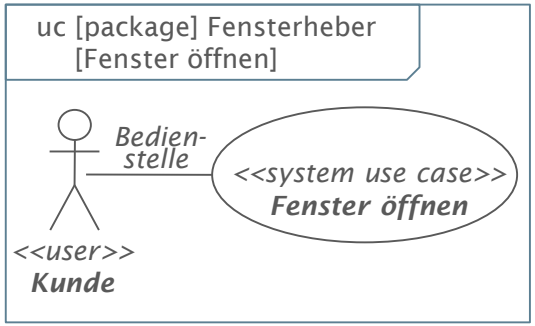
4.4 Tests spezifizieren - Objektbasierte Testentwurfsverfahren

Anwendungsfallbasiertes Testen (Use-Case basiertes Testen)



Merkmale:

- Tests auf Basis von Anwendungsfällen (Use Cases)
- Kann auch zur Bewertung des Prozesses herangezogen werden
- Häufig als Abnahmetests (fachlicher Anwendungsvorfall) oder als Systemtest (Systemanwendungsfall)



Überdeckungsgrad:

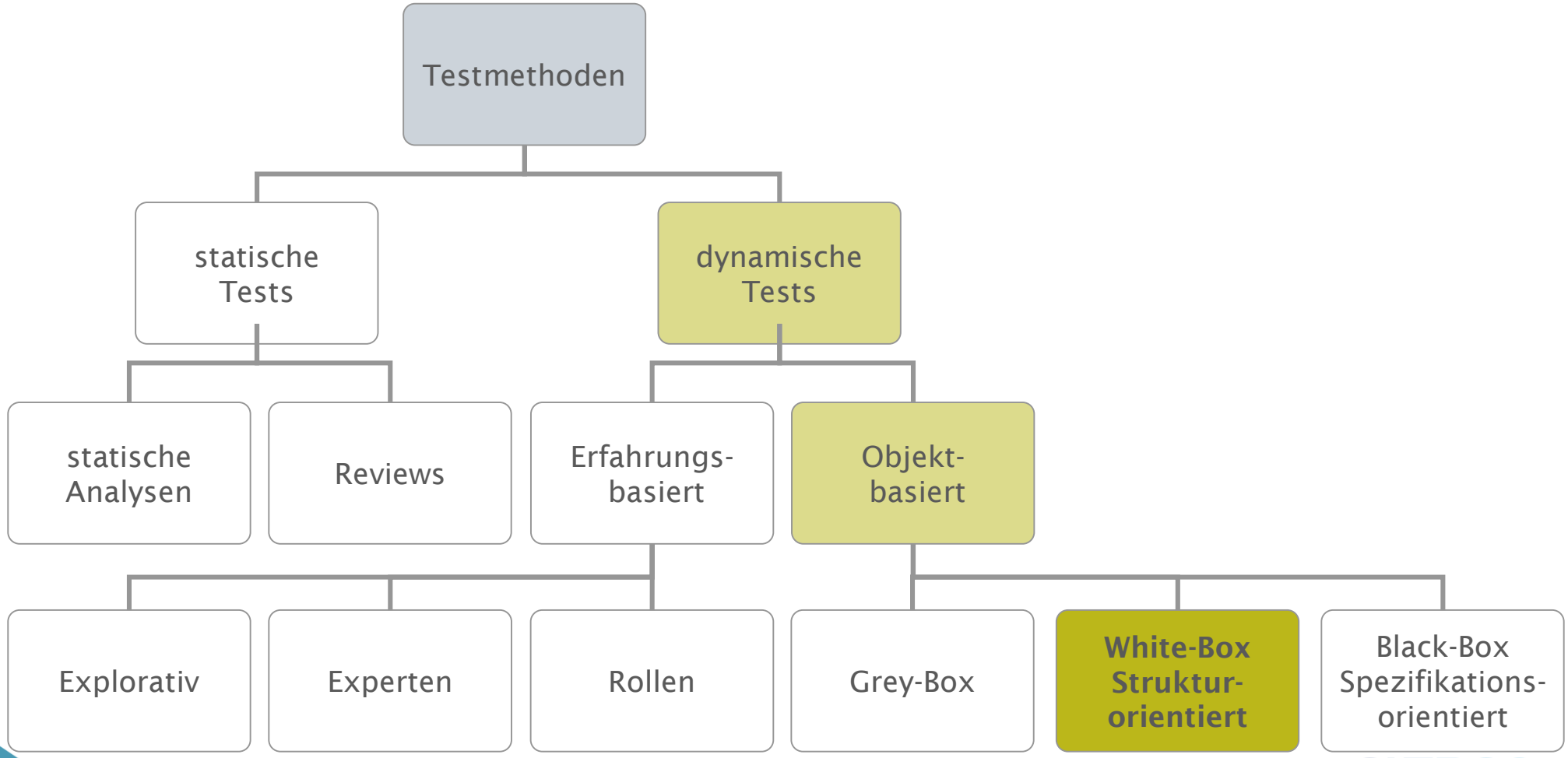
- Anzahl aller Anwendungsfälle zu Testfällen die einen Anwendungsfall testen

Beispiel Fensterheber:

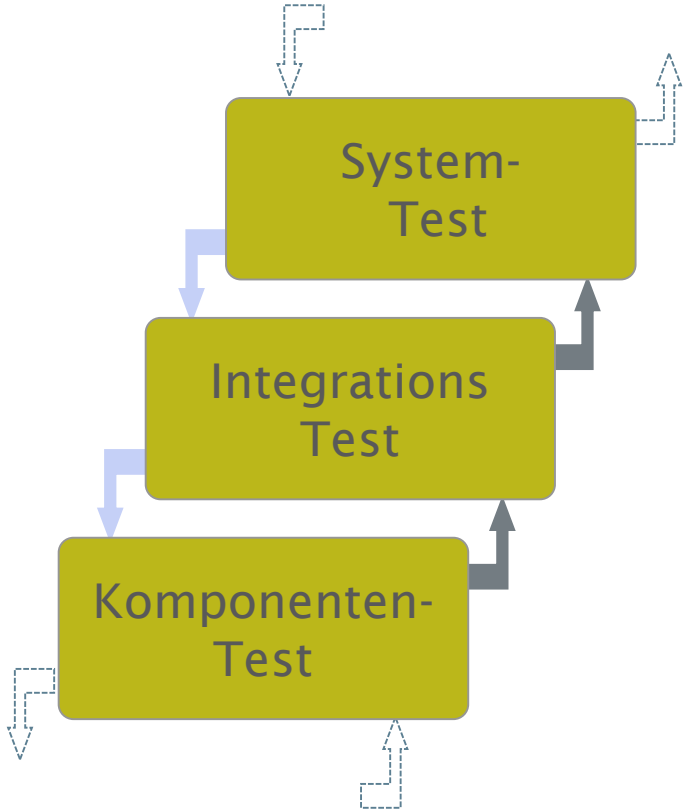
Fenster öffnen *(fachliches Ergebnis)* wenn der Kunde *(Akteur)* die Bedienstelle „Fenster öffnen“ betätigt *(fachlicher Auslöser)*

Schritt	Name	Eingabe (fachlicher Auslöser)	Vorausgesagtes Ergebnis (fachliches Ergebnis)
⓿	Vorbedingung	Das Fenster ist geschlossen	
❶	Aktion 1	Bedienstelle „Fenster öffnen“ betätigen	Fenster öffnet
❷	Nachbedingung	Fenster wieder schließen	

White-Box Testentwurfsverfahren
(strukturorientierte Testentwurfsverfahren)



White-Box Testentwurfsverfahren
(strukturorientierte Testentwurfsverfahren)

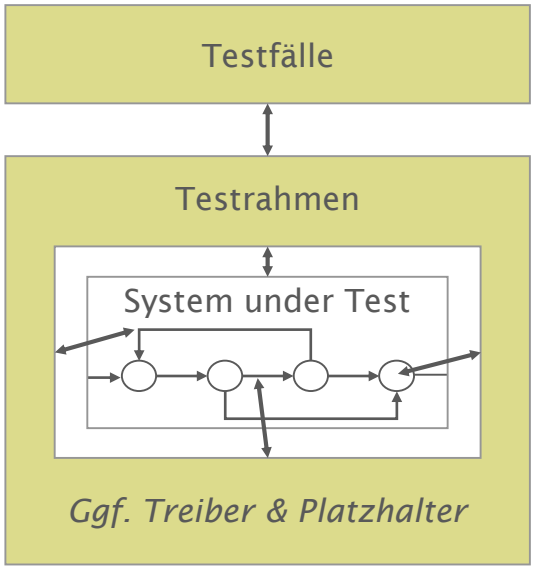


White-Box Testentwurfsverfahren bauen auf der vorgefundenen Struktur auf verschiedenen Teststufen auf :

- Systemebene
  - Die Struktur kann die Menüstruktur sein, Geschäftsprozesse oder die Struktur einer Webseite
- Integrationsebene
  - Die Struktur kann ein Aufruf-Baum sein. (ein Diagramm, das zeigt, welche Module andere Module aufrufen)
- Komponentenebene
  - Die Struktur der Softwarekomponente selbst, also Anweisungen, Entscheidungen bzw. Zweige oder einzelner Pfade

White-Box Testentwurfsverfahren (Überblick)

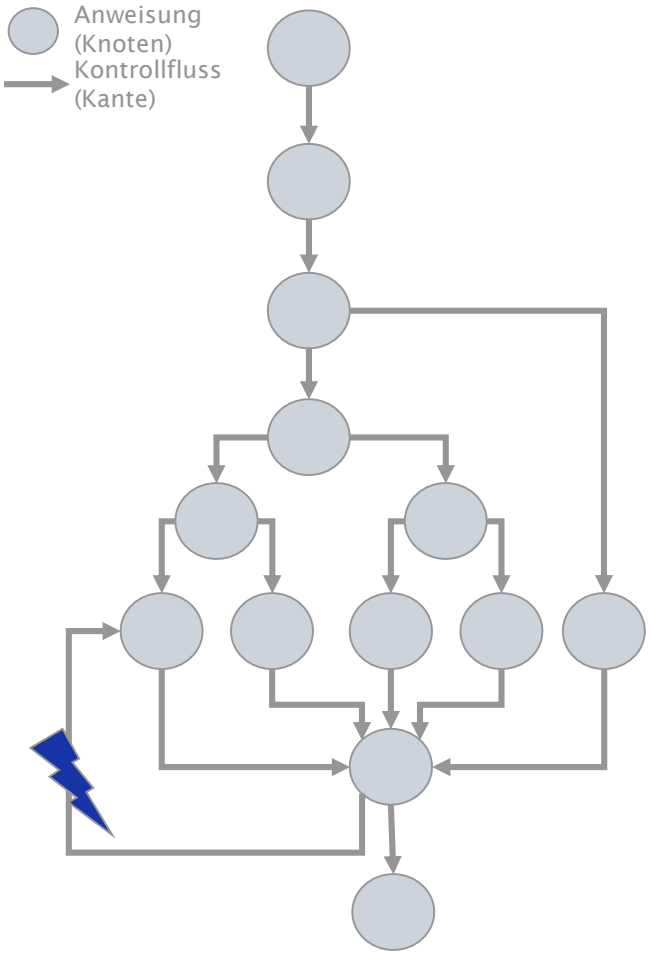
(strukturorientierte Testentwurfsverfahren)



Typische (Codeüberdeckungs-)Testentwurfsverfahren:

- Anweisungsüberdeckung
- Entscheidungsüberdeckung
- Bedingungsüberdeckung
- Bedingungskombinationsüberdeckung
- Pfadüberdeckung
- …

C0: Anweisungsüberdeckung (engl. *statement coverage*)



Ziel:

- Mindestens einmalige Ausführung jeder Anweisung (Überdeckungsgrad >= 100%)

Formel:

$$\text{Überdeckungsgrad} = \frac{Anzahl\ Anweisungen, die\ durch\ Testfälle\ abgedeckt\ sind}{Gesamtzahl\ aller\ ausführbaren\ Anweisungen}$$

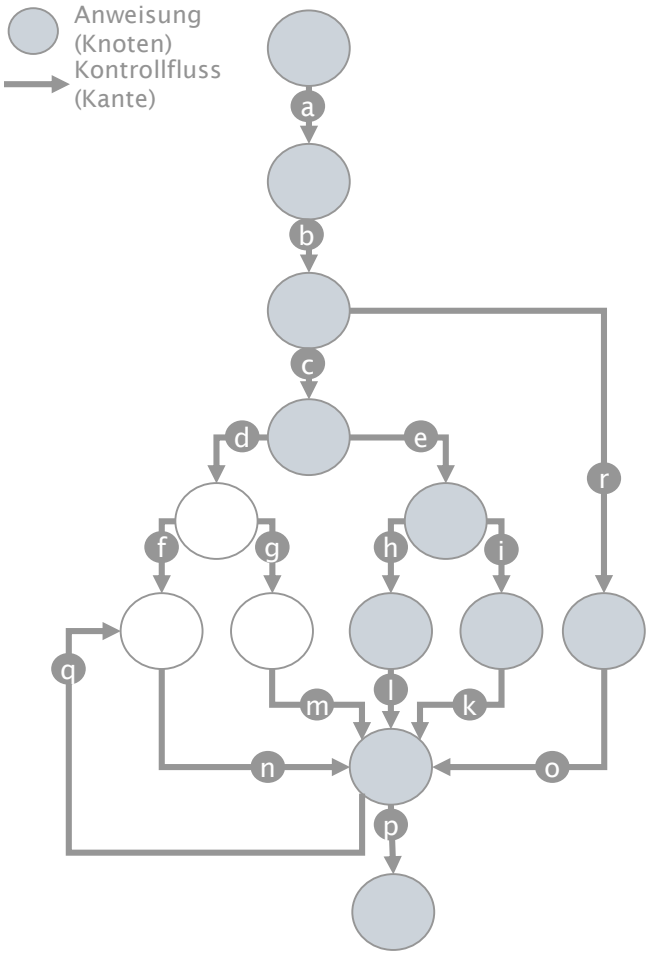
Vorteil:

+ Einfach zu messendes Kriterium
+ Toolunterstützung
+ Auffinden von „Totem Code“

Nachteil:

- Schwaches Kriterium
- Fehlerzustände bei Abzweigungen werden nicht identifiziert

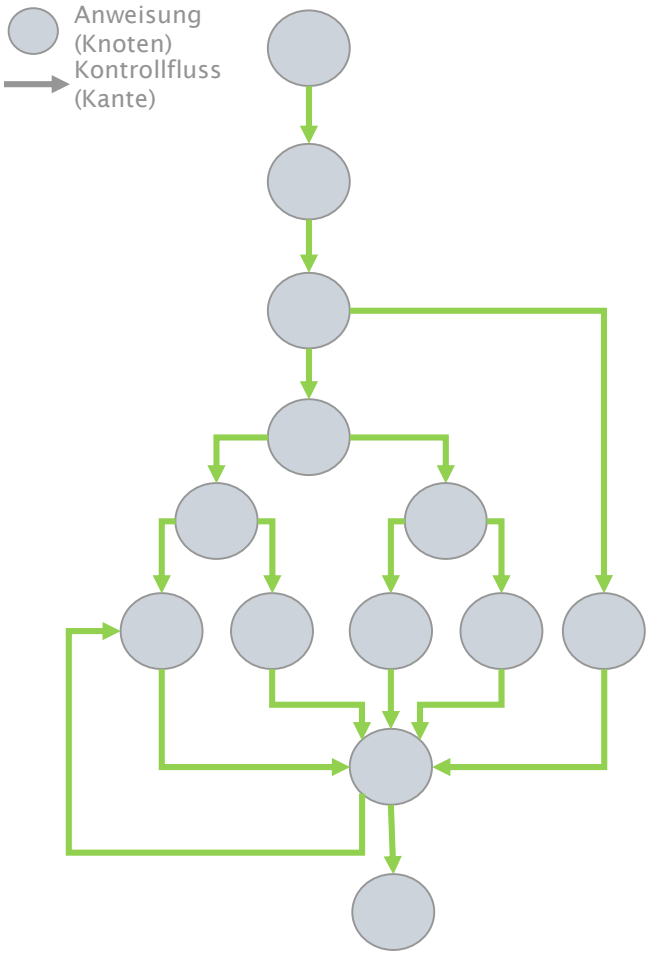
Übung: Anweisungsüberdeckung



Frage:

1. Wie hoch ist im nebenstehendem Beispiel die Anweisungsüberdeckung?
2. Wie viele Testfälle sind für eine 100% Anweisungs-überdeckung notwendig?

C1: Entscheidungsüberdeckung oder Zweigüberdeckung (branch coverage)



Ziel:

- Mindestens einmaliges Durchlaufen aller Entscheidungen. (Überdeckungsgrad >= 100%)

Formel:

- $\text{Überdeckungsgrad} = \frac{Anzahl\ aller\ Entscheidungsausgänge, die\ durch\ Testfälle\ abgedeckt\ sind}{Gesamtzahl\ aller\ Entscheidungsausgänge}$

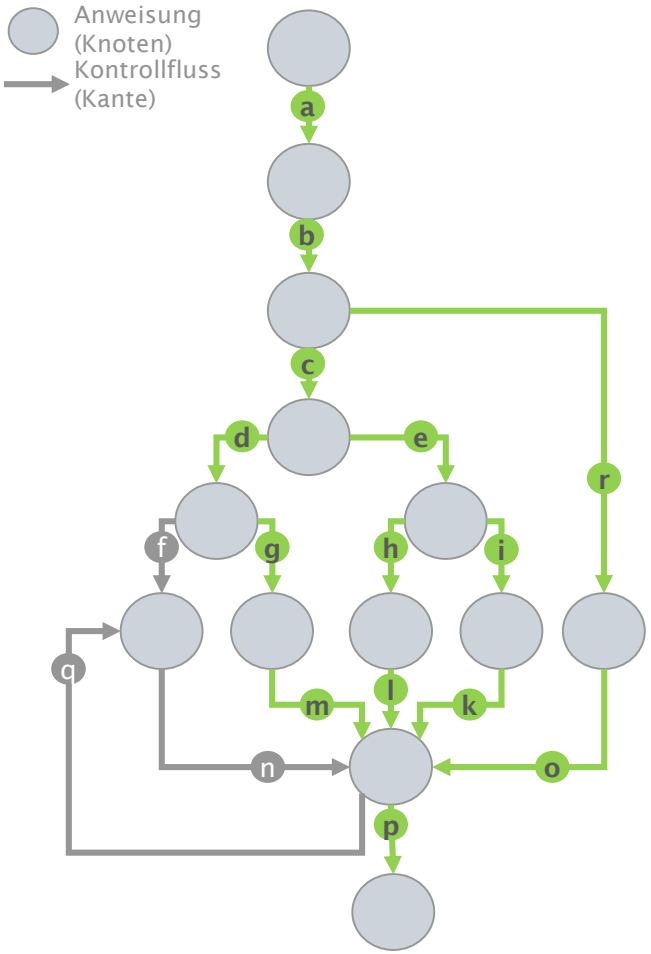
Vorteil:

+ Einfach zu messendes Kriterium
+ Toolunterstützung
+ Auffinden von „Bottlenecks“

Nachteil:

- Minimales Kriterium

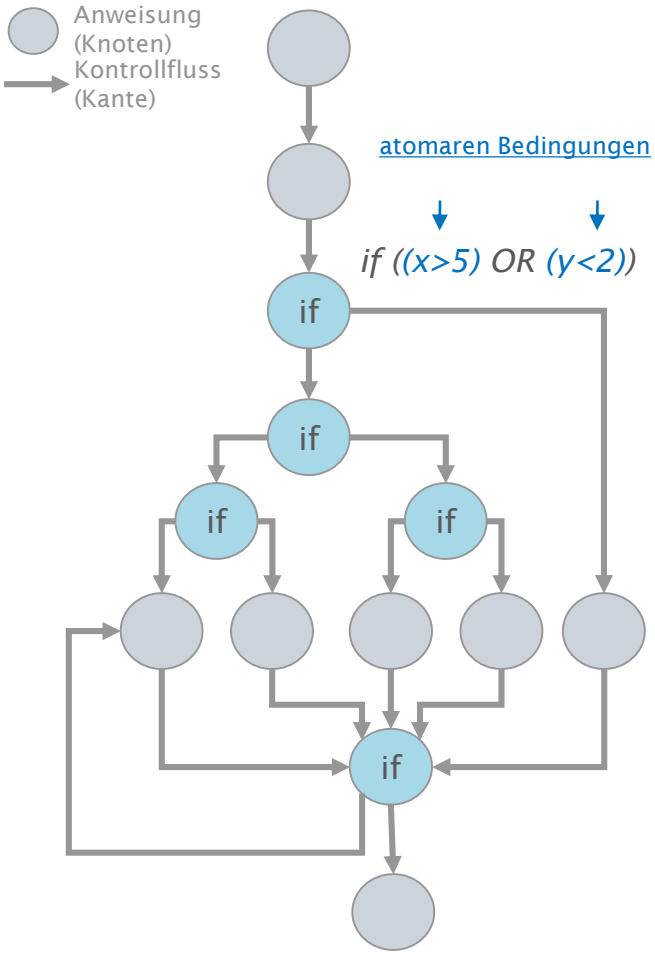
Übung: Entscheidungsüberdeckung



Frage:

1. Wie hoch ist im nebenstehendem Beispiel die Entscheidungsüberdeckung?
2. Wie viele Testfälle sind für eine 100% Entscheidungs-überdeckung notwendig?

C2: (einfache) Bedingungsüberdeckung



Ziel:

- Alle atomaren Bedingungen in Schleifen und Auswahlanweisungen mindestens ein Mal als „wahr“ und „falsch“ durchlaufen

Maß:

- $\text{Überdeckungsgrad} = \frac{Anzahl\ durchlaufener\ Bedingungen}{Gesamtzahl\ aller\ möglichen\ Bedingungen}$

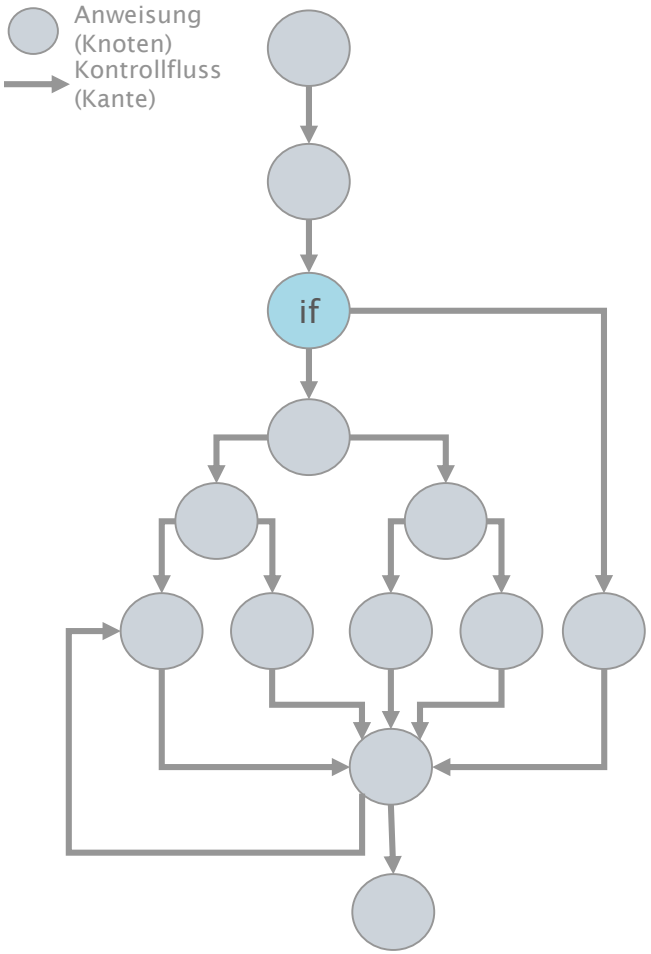
Vorteil:

+ Auch atomare Bedingungen werden erfasst

Nachteil:

- Vergleichbares Kriterium zu C1
- Ggf. werden nicht alle Kanten durchlaufen

Übung: (einfache) Bedingungsüberdeckung



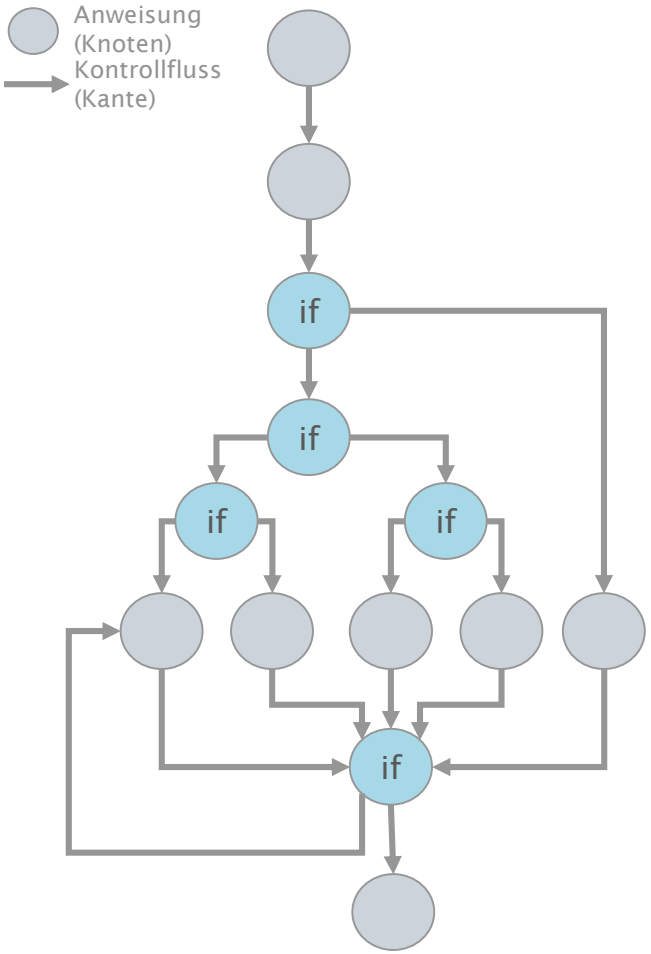
Beispiel:

If ((x>5) OR (y<2))

Frage:

1. Benennen Sie die atomaren Bedingungen
2. Wie viele Testfälle sind für eine 100% Bedingungs-überdeckung notwendig?

C3: Bedingungskombinationsüberdeckung (Mehrfach-)



Ziel:

- Alle atomaren Bedingungen und alle Bedingungskombinationen in Schleifen und Auswahlanweisungen mindestens ein Mal als „wahr“ und „falsch“ durchlaufen

Maß:

- Überdeckungsgrad = [Anzahl durchlaufener Bedingungen Zweige] / [Gesamtzahl der möglichen Bedingungen]

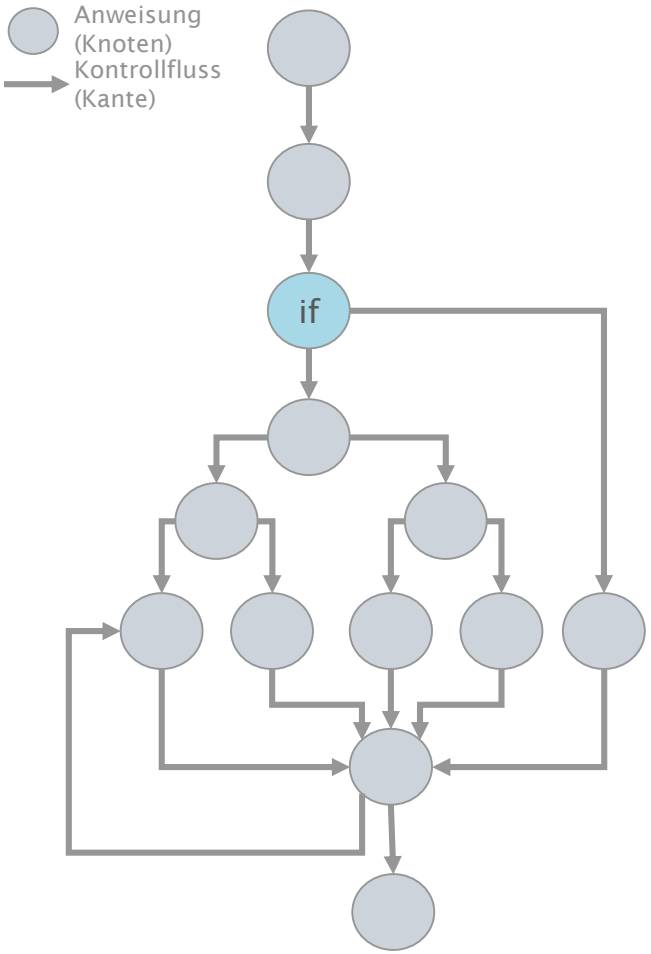
Vorteil:

+ Stellt umfassenderes Kriterium aus C1 und C2 dar

Nachteil:

- Aufwändig

Übung: Bedingungskombinationsüberdeckung (Mehrfach-)



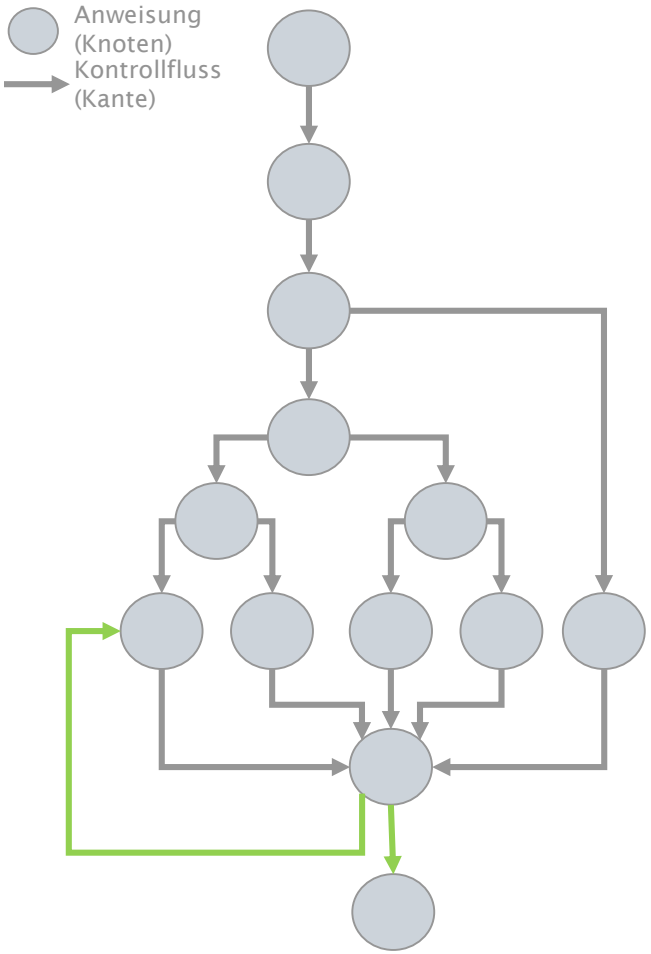
Beispiel:

If ((x>5) OR (y<2))

Frage:

1. Wie viele Testfälle sind für eine 100% Bedingungs-kombinationsüberdeckung notwendig?

C4: Pfadüberdeckung (engl. *path coverage*)



Ziel:

- Ausführung aller möglicher Pfade (Kantenkombinationen)

Maß:

- Überdeckungsgrad = [Anzahl durchlaufener unterschiedlicher Pfade] / [Gesamtzahl der Pfade]

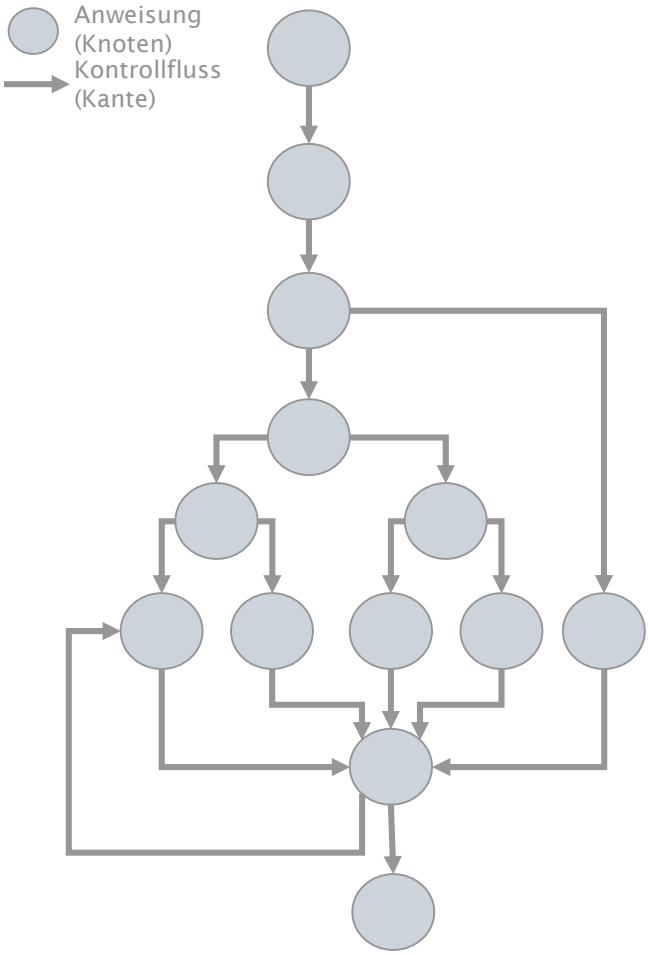
Vorteil:

+ Feststellen von Einflüssen durch Zweigreihenfolgen

Nachteil:

- Sehr lang und theoretisch unendlich lange Pfade
  - → sehr Aufwändig
  - → Nur sinnvoll (wenn überhaupt) bei Einsatz von Testautomatisierung

Übung: Pfadüberdeckung



Frage:

1. Wie viele Testfälle sind für eine 100% Pfadüberdeckung notwendig?

Übung 2.6 zur White-Box-Tests

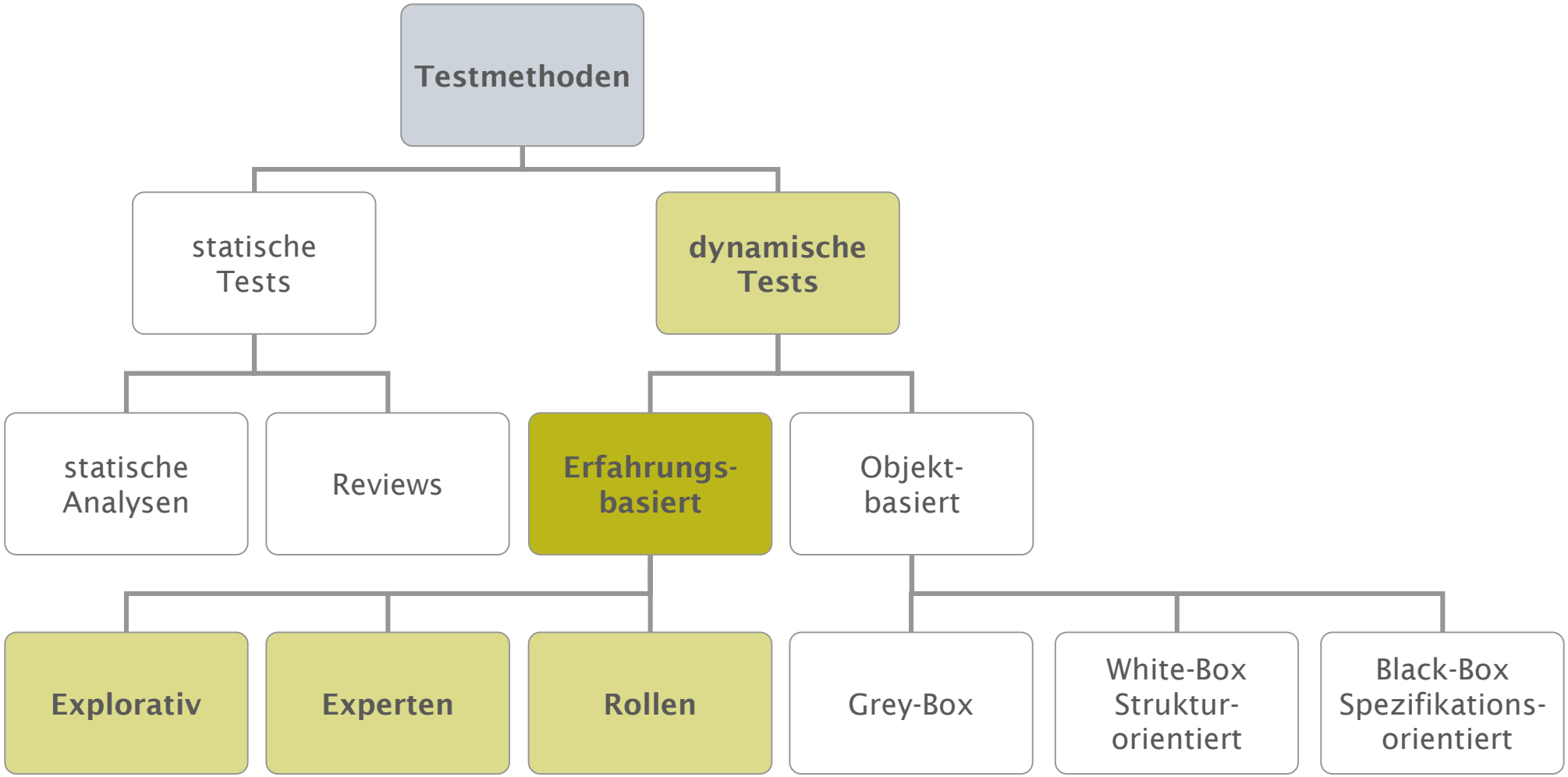
Übung 2.6:
Schreiben von Testfällen zu vorgegebenen Kontrollflüssen unter Verwendung der

- **Anweisungsüberdeckung**
- **Entscheidungsüberdeckung**

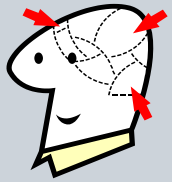
Seminarinhalte

4 Tests spezifizieren

Erfahrungsbasierte Testentwurfsverfahren



Erfahrungsbasierte Testentwurfsverfahren



Erfahrungsbasiertes Testen

Stärken:

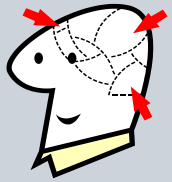
- Ergänzt systematisch ermittelte Testfälle
- Aufdecken von Schwächen in den Spezifikationen und Dokumentationen
- Erfahrung der Tester wird genutzt

Basierend auf:

- Erfahrungen und Erwartungen von Testern über das Testobjekt
  (wie z.B. Erfahrung über frühere Fehlerzustände und Vermutungen über zukünftige Fehler.)
- Benutzerverhalten spezifischer Anwendergruppen

Erfahrungsbasierte Tests sind häufig notwendig, können aber system. Verfahren nicht ersetzen!

Erfahrungsbasierte Testentwurfsverfahren

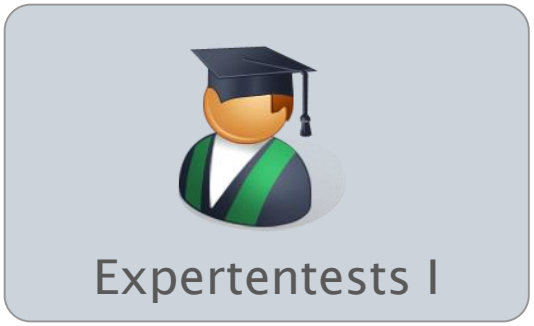


Erfahrungsbasiertes Testen

Gemeinsamkeiten:

- Das Wissen und die Erfahrung von Menschen wird zur Ableitung der Testfälle genutzt
- Das Wissen von Testern, Entwicklern, Anwendern und Betroffenen über die Software, ihre Verwendung und ihre Umgebung ist eine Informationsquelle
- Das Wissen über wahrscheinliche Fehler-zustände und ihre Verteilung ist eine weitere Informationsquelle

Expertentests (Error Guessing)



Merkmale:

- Auch bekannt als intuitive Testfallermittlung *(engl. „Error Guessing")*
- Strukturierter Ansatz > Fehlerangriff *(engl. „Fault Attack" - Testfälle werden auf Basis möglicher und bekannter Fehlerzustände und Fehlerwirkungen erstellt.)*
- Weit verbreitetes, systematisches Verfahren
- Nutzt Intuition, besondere Kenntnisse und Erfahrung der Tester
- Qualifiziertes Personal notwendig
- Effizienz abhängig von der Erfahrung

Vorgehensweise:

- Auflisten möglicher Fehlerzustände oder fehlerträchtiger Situationen. (ggf. aus Fehlerdatenbank!)
  - → Ableiten weiterer Testfälle

Expertentests (erwartungsorientiertes Testen)

Expertentests II

Merkmale:

- Basierend auf Erwartungen an das Produkt
- Häufig bei Parametrisierungsarbeiten
- Nutzt Intuition, besondere Kenntnisse und Erwartungen der Tester
- Auch als Abnahmetest durch den Kunden

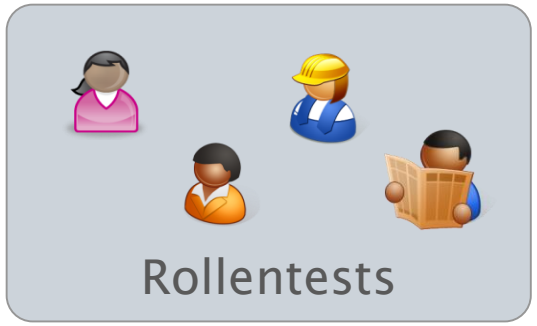
Vorgehensweise:

- Zusammenstellen möglicher Nutzungsszenarien
  - → Ableiten weiterer Testfälle

Exkurs

nicht Teil des CTFL!

Rollentests



Merkmale:

- Nutzt Intuition, besondere Kenntnisse und Erwartungen der Zielgruppen
  (z.B. Kind vs. Manager)
- Auch als Abnahmetest durch den Kunden (z.B. Feldtests) oder durch das Hineinversetzen in den Kunden

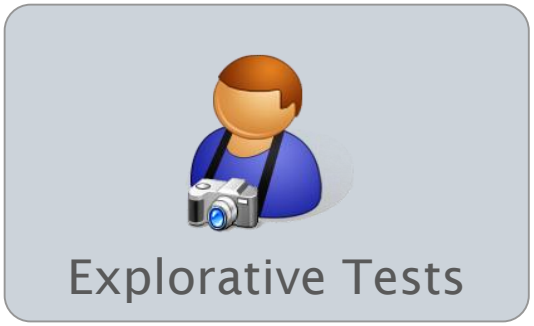
Vorgehensweise:

- Zusammenstellen möglicher Nutzungsszenarien aus Sicht einer speziellen Zielgruppe
  - → Ableiten weiterer Testfälle
- Wichtige Faktoren der Zielgruppe :
  - Was ist ihr wichtig?
  - Was kann sie besonders gut?
  - Was sind typische Verhaltensmuster?

Exkurs

nicht Teil des CTFL!

Explorative Tests



Definition:

Ein informelles Testentwurfsverfahren, bei dem der Tester den Entwurf der Tests aktiv steuert, indem er testet und die Informationen, die er während des Tests erhält für den Entwurf weiterer Tests verwendet.
[nach Bach]

Vorgehensweise:

- Definition der Test-Charta, der die Testziele zu entnehmen sind
- Testentwurf, -ausführung und -protokollierung quasi gleichzeitig

Vorteile:

- Gut eignet, wenn es nur wenige oder ungeeignete Spezifikationen gibt
- Eignet sich auch unter Zeitdruck

Seminarinhalte

4 Tests spezifizieren

Pragmatischer Ansatz zum Testfallentwurf

Testfallentwurf

Basistestfälle herleiten:

- Logische Testfälle anwendungsfallorientiert spezifizieren
- Testdaten für konkrete Testfälle mittels Äquivalenzklasse ermitteln

Testfälle ergänzen:

- Testdaten mittels Grenzwerte ergänzen.
- Testabdeckung durch strukturorientierte Tests erhöhen

Testsequenzen zusammenstellen:

- Permutation der Testfälle (soweit möglich)

Auswahl der Testentwurfsverfahren

Die Auswahl und Kombination der geeigneten Testentwurfsverfahren hängt ab von:

- Art des Systems
  (Software, Mechatronisch…)
- Zu erfüllende Vorgaben
  (Vorschriften, behördliche, Kunden- oder Vertragsanforderungen…)
- Teststufe
  (Komponententest, Abnahmetest…)
- Testziele
  (Funktionalität, Zuverlässigkeit…)
- Verfügbare Dokumentation
  (Modelle, Lastenhefte, Diagramme…)
- Mitarbeiterqualifikation
  (Erfahrung, Fähigkeiten)
- Risikograd / Art des Risikos
  (bisher gefundene Fehlerzustände, neue Technologie, wenig Dokumentation…)
- Projektrahmenbedingungen
  (Zeit und Geld, gewählte Entwicklungsmodelle)

Gegenüberstellung der Testentwurfsverfahren

	Black-Box	White-Box	Erfahrungsbasiert
Auch genannt	• Spezifikationsorientiertes Testen • Spezifikationsbasiertes Testen	Strukturorientiertes Testen	Intuitives Testen
Testfall-ableitung	Tests werden aus Spezifikationen beliebiger Form abgeleitet, ohne die Struktur des Produktes zu kennen.	Tests werden auf Basis der Struktur des Produktes ohne Berücksichtigung der Spezifikationen.	Tests auf Basis der spezifischen Erfahrungen der Tester, ohne Struktur und Spezifikation zu berücksichtigen.
Methoden	• Äquivalenzklassenmethode • Grenzwertanalyse • Entscheidungstabellen • Zustandsbasierter Test • Anwendungsfallbasiertes Testen	• Anweisungsüberdeckung • Entscheidungsüberdeckung • Bedingungsüberdeckung • Pfadüberdeckung	• Fehlererwartung • Exploratives Testen • Rollentests • Expertentest
Teststufen	+ Komponententest + Integrationstest + Systemtest + Abnahmetest		

INNOVATION MAKERS