**Qualitätssicherungskonzept**

**Underline_SE.jpgfür die Softwareentwicklung**

**Teil 4 - Testverfahren und Testtechniken**

**Autor: Marion Grabenweger**

**12. November 2012**

**Version 0.3**

**Inhaltsverzeichnis**

[**1 Einleitung 1-2**](#_Toc340584479)

[**2 Testverfahren, Testtechniken und Teststufen 2-3**](#_Toc340584480)

[**2.1 Spezifikationsbasierte Testverfahren 2-5**](#_Toc340584481)

[**2.1.1 Anforderungsbasiertes Testen 2-5**](#_Toc340584483)

[**2.1.2 Anwendungsfallbasiertes Testen 2-6**](#_Toc340584484)

[**2.1.3 Äquivalenzklassenbildung 2-8**](#_Toc340584485)

[**2.1.4 Grenzwertanalyse 2-9**](#_Toc340584486)

[**2.1.5 Domain-Testing 2-11**](#_Toc340584487)

[**2.1.6 Paarweises Testen (Orthogonale Arrays, Paartabellen) 2-12**](#_Toc340584488)

[**2.1.7 Klassifikationsbaum-Methode 2-14**](#_Toc340584489)

[**2.1.8 Entscheidungstabellen 2-15**](#_Toc340584490)

[**2.1.9 Ursache-Wirkungsgraph Analyse 2-17**](#_Toc340584491)

[**2.1.10 Zustandsbasiertes Testen 2-19**](#_Toc340584492)

[**2.1.11 Syntaxtest 2-21**](#_Toc340584493)

[**2.2 Strukturorientierte Testverfahren 2-24**](#_Toc340584494)

[**2.2.1 Anweisungstests 2-25**](#_Toc340584495)

[**2.2.2 Zweigtests/Entscheidungstests 2-25**](#_Toc340584499)

[**2.2.3 Bedingungstests 2-26**](#_Toc340584500)

[**2.2.4 Pfadtests 2-27**](#_Toc340584501)

[**2.3 Fehlerbasierte und erfahrungsbasierte Testverfahren 2-28**](#_Toc340584502)

[**2.3.1 Fehler-Taxonomien 2-28**](#_Toc340584503)

[**2.3.2 Ad-hoc-Testen 2-30**](#_Toc340584504)

[**2.3.3 Fehlererwartungsmethode 2-31**](#_Toc340584505)

[**2.3.4 Checklisten-basiertes Testen 2-32**](#_Toc340584506)

[**2.3.5 Exploratives Testen 2-33**](#_Toc340584507)

[**2.3.6 Fehlerangriff 2-34**](#_Toc340584508)

[**3 Testen von Qualitätsmerkmalen 3-35**](#_Toc340584509)

[**3.1 Funktionalität und Interoperabilität 3-35**](#_Toc340584512)

[**3.2 Benutzbarkeit 3-36**](#_Toc340584513)

[**3.3 Barrierefreiheit 3-38**](#_Toc340584514)

[**3.4 Sicherheit 3-39**](#_Toc340584515)

[**3.5 Zuverlässigkeit 3-41**](#_Toc340584516)

[**3.6 Effizienz 3-42**](#_Toc340584517)

[**3.7 Wartbarkeit 3-44**](#_Toc340584518)

[**5.8 Portabilität 3-45**](#_Toc340584519)

[**4 Reviews 4-47**](#_Toc340584520)

[**4.1 Reviewprozess 4-48**](#_Toc340584521)

[**4.2 Rollen und Verantwortlichkeiten 4-49**](#_Toc340584522)

[**4.3 Reviewarten 4-49**](#_Toc340584523)

[**4.4 Erfolgsfaktoren für Reviews 4-51**](#_Toc340584524)

# Einleitung

Das Qualitätssicherungskonzept soll die Grundsätze der Qualitätssicherungsmaßnahmen der Softwareentwicklung regeln.

Dieses Dokument gliedert sich in 4 Teile:

Teil 1 „Grundlagen des Softwaretestens“ beschreibt die Grundlagen des Softwaretestens und inkludiert Begriffsbestimmungen, um ein gemeinsames „Wording“ zu ermöglichen.

Im 2. Teil wird ein generischer Testprozess beschrieben. Durch die Gliederung in Teilprozesse soll sichergestellt werden, dass das Testvorgehen für alle Projekte anwendbar ist, wobei Teile des Prozesses von anderen Bereichen übernommen werden können.

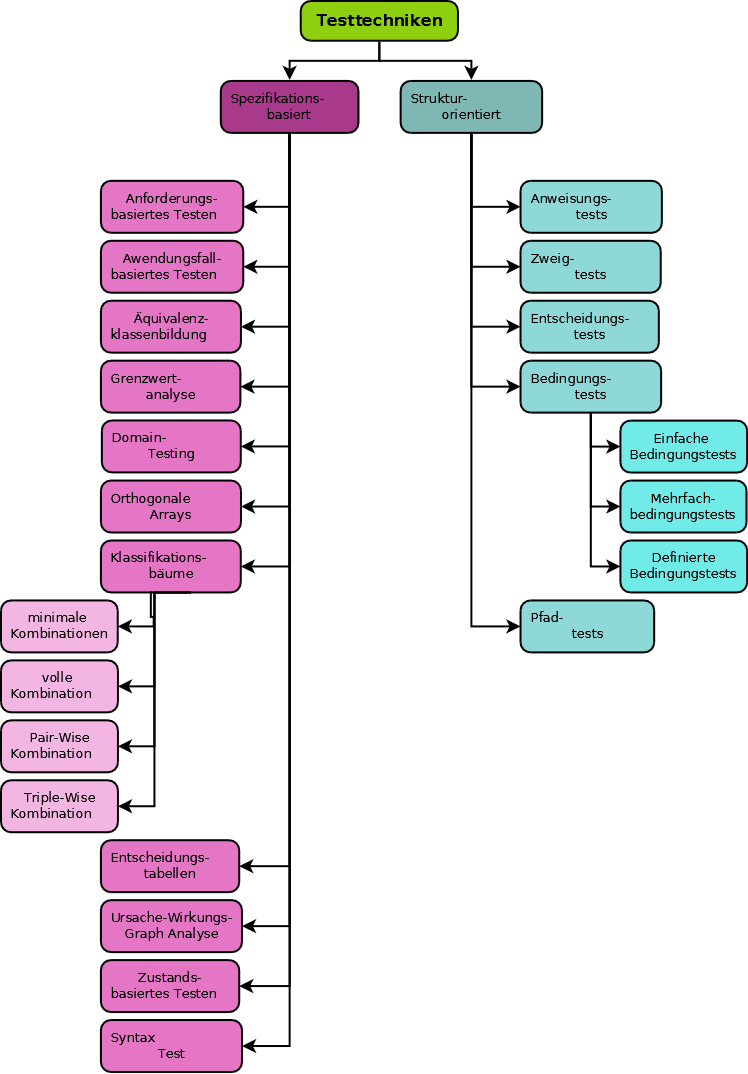
Der 3. Teil beschäftigt sich mit der Dokumentation, die während der jeweiligen Testteilprozesse erstellt werden kann und liefert Templates und Beispiele.

Im 4. Teil werden die Testverfahren und Testtechniken erklärt, die zum Einsatz kommen können um eine systematische Testfallableitung zu gewährleisten.

Alle 4 Teile haben zum Ziel einen systematischen und strukturierten Testprozess einzuführen um die Qualität der zu liefernden Softwareprodukte zu steigern.

# Testverfahren, Testtechniken und Teststufen

In welchem Maße das Testen zur Herstellung eines hochwertigen Endprodukts beiträgt, hängt davon ab, welche Testverfahren gewählt und wie effektiv diese eingesetzt werden.



## Spezifikationsbasierte Testverfahren

Die Testbedingungen und Testfälle spezifikationsorientierter Testverfahren sind von den Spezifikationen der **System- oder Softwareanforderungen** abgeleitet. Die Eingabedokumente, auf deren Grundlage der Code entworfen und entwickelt wird, werden daher mit dem Begriff **Testbasis** zusammengefasst. Die für das Testen verwendeten Spezifikationen erläutern, was von der Software erwartet wird und wie sie diese Erwartungen erfüllen wird. Die **Testüberdeckung** ergibt sich aus dem Prozentsatz der spezifizierten Elemente, die von den entworfenen Tests abgedeckt wurden. Wenn alle spezifizierten Elemente abgedeckt wurden, ist zwar nicht zwangsläufig eine vollständige Testüberdeckung erreicht, aber es zeigt, dass die Software die **Anforderungen** in den Spezifikationen erfüllt. Um eine höhere Testüberdeckung zu erreichen, können zusätzliche Informationen notwendig sein.

* 1. 1. Anforderungsbasiertes Testen

Beim **anforderungsbasierten Testen** bildet das Anforderungsdokument die Grundlage zum Herleiten der Testfälle. Die Anforderungsdefinition enthält **funktionale** und **nicht funktionale** **Anforderungen** an das gesamte System. Daraus werden funktionale und nichtfunktionale Tests abgeleitet. In der funktionalen Spezifikation eines Teilsystems wird die Umsetzung der funktionalen Anforderungen bezüglich dieses Teilsystems detailliert beschrieben (innerer Aufbau, Schnittstellen zu anderen Teilsystemen). **Anforderungsbasiertes Testen** ist Teil des **Requirements Engineering**. Diese Methode hat eine Konsequente Verfolgung der Anforderungen durch den gesamten Entwicklungsprozess zum Ziel.

* **Rückverfolgbarkeit von Anforderungen:**
* Zu jeder Produktanforderung existiert mindestens eine Anforderung an die Anwendung.
* Zu jeder Anwendungsanforderung sollte mindestens ein Test existieren.
* Die Umsetzung der Anforderungen in den verschiedenen Phasen des Softwareentwicklungsprozesses ist nachvollziehbar.
* **Anforderungsbasiertes Testen** prüft, ob das System die Anforderungen des Endanwenders löst. Es kann auch dann eingesetzt werden, wenn (noch) keine Funktionale Spezifikation existiert.
* **Anforderungsbasiertes Testen** hilft außerdem
* Fehler in den Anforderungen zu finden (z.B. Testbarkeit von Anforderungen),
* im Gegenzug Anforderungen klar und testbar zu formulieren.

Aus diesem Grund ist es besonders wichtig, dass die Anforderungen gewissen Qualitätskriterien entsprechen. Dies gilt nicht nur für Anforderungen sondern auch für Change-Requests.

* **Qualitätskriterien von Anforderungen:**
* **identifizierbar**
* **atomar**
* **verständlich**
* **nachweisbar**
* **eindeutig**
* **realisierbar**
* **widerspruchsfrei**
* **redundanzfrei**
* **korrekt abgeleitet**
* **vollständig**
* **rückverfolgbar**
* Das Formulieren **anforderungsbasierter Testfälle** sollte deshalb so früh wie möglich beginnen.

**Steckbrief: Anforderungsbasiertes Testen**

**Grundidee:** Zwischen Anforderungen und Testfällen wird systematisch eine Beziehung hergestellt.

**Klasse:** Spezifikationsbasierte Testverfahren

**Sucht nach:** Fehlern, die entstehen, weil festgelegte Anforderungen bei der Umsetzung übersehen oder missverstanden werden.

**Qualitätsmerkmale:** Alle

**Überdeckungsmaße:** Anzahl getesteter Anforderungen zu Gesamtzahl der Anforderungen.

**Wann:** Am besten immer.

**Verwandt mit:** Anwendungsfallbasiertem Testen, Geschäftsprozessbasiertem Testen

**Kombinierbar mit:** allen anderen Verfahren.

* + 1. Anwendungsfallbasiertes Testen

Tests können aus Anwendungsfällen (**use cases**) abgeleitet werden. Ein Anwendungsfall beschreibt die Interaktionen zwischen den Akteuren (Anwender oder Systeme), die ein aus Sicht des Anwenders oder Kunden gewünschtes und wahrnehmbares Ergebnis zur Folge haben.

* **Anwendungsfälle** können
  + auf einer **abstrakten Ebene** (fachlicher Anwendungsvorfall, technologiefrei, Geschäftsprozessebene)
  + oder auf einer **Systemebene** (Systemanwendungsfall auf Ebene der Systemfunktionalität) beschrieben werden.
* Jeder Anwendungsfall hat **Vorbedingungen**, die erfüllt sein müssen, damit der Anwendungsfall erfolgreich durchgeführt werden kann.
* Jeder Anwendungsfall endet mit **Nachbedingungen**, den beobachtbaren Ergebnissen und dem Endzustand des Systems, wenn der Anwendungsfall vollständig abgewickelt wurde.
* Ein Anwendungsfall hat üblicherweise ein **Hauptszenario** (das wahrscheinlichste Szenario) und **alternative Szenarien**.
* Anwendungsfälle beschreiben die „**Prozessabläufe**” durch das System auf Grundlage seiner voraussichtlich tatsächlichen Verwendung. Daher sind von Anwendungsfällen abgeleitete Testfälle bestens geeignet, während des Praxiseinsatzes des Systems Fehlerzustände in den Prozessabläufen aufzudecken.
* Anwendungsfälle sind für den Entwurf von **Abnahmetests** mit Kunden-/Anwenderbeteiligung sehr hilfreich.
* Indem das Zusammenwirken und die gegenseitige Beeinflussung unterschiedlicher Komponenten betrachtet werden, können sie auch Fehlerzustände im Umfeld der **Integration** aufdecken, die durch den Test der einzelnen Komponenten nicht gefunden werden könnten.
* Das Entwerfen von Testfällen auf Basis von Anwendungsfällen kann mit anderen **spezifikationsorientierten Testentwurfsverfahren** kombiniert werden.

|  |  |  |
| --- | --- | --- |
| Name des Anwendungsfalles: | **MPOS suchen/anzeigen** | ID**: MPOSSUCHEN.AF01** |
| **Ziel** | Überprüfung, ob über die Suchkriterien korrekte Ergebnisse in der Trefferliste angezeigt werden | |
| **Gültigkeitsbereich** | Metapositionen suchen über alle Bereiche (Screenname) |  |
| **Referenzierte Anwendungsfälle** | Login (ID) | |
| **Vorbedingungen** | Der Benutzer ist eingeloggt (Login ID) | |
| **Nachbedingungen** | Die Trefferliste kann verwendet werden | |
| **Akteure** | Anwender (A)  System (S) | |

**Hauptszenario**

|  |  |
| --- | --- |
| Schritte | Aktion |
| **1** | (A) Bereich auswählen |
| **2** | (S) Auswahlliste Metaleistungsgruppe wird befüllt  (A) Metaleistungsgruppe auswählen |
| **3** | … |
| **4** | (A) Laufzeitbeginn eingeben |
| **5** |  |

**Nebenszenarien**

|  |  |
| --- | --- |
| Schritte | Aktion |
| **1a** | keinen Bereich auswählen |
| **2** | (A)Leistungsgruppe auswählen |
| **3** | … |
| **4a** | (A)Laufzeitbeginn aus Kalender wählen |
| **4b** | (A)ungültigen Laufzeitbeginn eingeben  (S) Fehlermeldung |

**Steckbrief: Anwendungsfallbasiertes Testen**

**Grundidee:** Die Anwendungsfälle werden gezielt als Muster für Testfälle genutzt.

**Klasse:** Spezifikationsbasierte Testverfahren

**Sucht nach:** Problemen bei den häufigsten gebrauchten Abläufen.

**Qualitätsmerkmale:** Funktional

**Überdeckungsmaße:** Mindestens ein Testfall pro Anwendungsfall.

**Wann:** Fast immer möglich; ideal, wenn schon Use Cases vorliegen.

**Verwandt mit:** Anforderungsbasiertem Testen, Geschäftsprozessbasiertem Testen

**Kombinierbar mit:** allen anderen Verfahren.

* + 1. Äquivalenzklassenbildung

Bei der **Äquivalenzklassenbildung** werden Testbedingungen gesucht, die gleich behandelt werden. Diese werden zu einer Klasse zusammengefasst. Man kann davon ausgehen, dass jeder Wert oder jede Bedingung innerhalb einer Klasse gleichwertig behandelt wird.

Neben den Äquivalenzklassen, die gültige Eingaben umfassen, sind auch solche für ungültige Eingaben zu berücksichtigen.

* **Testfallableitung:**

Für jede zu testende **Eingabevariable** (z.B.: Funktions-/Methodenparameter beim Komponententest oder Maskenfeld beim Systemtest) wird der **Definitionsbereich** ermittelt. Dieser Definitionsbereich ist die Äquivalenzklasse aller zulässigen bzw. erlaubten Eingabewerte. Die Werte außerhalb des Definitionsbereichs werden als Äquivalenzklassen mit unzulässigen Werten betrachtet. Auch für diese Werte ist zu prüfen, wie das Testobjekt sich verhält.

Als nächster Schritt sind die Äquivalenzklassen zu verfeinern. **Äquivalenzklassenelemente**, die vom Testobjekt laut Spezifikation unterschiedlich verarbeitet werden, sind einer neuen (Unter-) Äquivalenzklasse zuzuordnen. Die Äquivalenzklassen werden so lange aufgeteilt, bis sich alle unterschiedlichen Anforderungen mit den jeweiligen Äquivalenzklassen decken. Für jede Äquivalenzklasse ist dann ein **Repräsentant** für einen Testfall auszuwählen.

* **Richtlinien zum Ermitteln der Äquivalenzklassen:**
* Ermitteln der spezifizierten **Einschränkungen** und **Bedingungen** aus der Spezifikation sowohl für die Eingaben als auch für die Ausgaben.
* Für jede Einschränkung bzw. Bedingung ist die **Äquivalenklassenbildung** vorzunehmen:
  + Ist ein **zusammenhängender Wertebereich** spezifiziert, dann sind eine gültige und zwei ungültige Äquivalenzklassen zu berücksichtigen.
  + Ist spezifiziert, dass eine **Anzahl von Werten** einzugeben ist, sind eine gültige und zwei ungültige (Unter- und Überschreitung der gültigen Anzahl) Äquivalenzklassen zu bilden.
  + Ist eine **Menge von Werten** spezifiziert, die möglicherweise unterschiedlich zu behandeln sind, so ist für jeden Wert der Menge eine gültige (bestehend aus diesem Wert) Äquivalenzklasse vorzusehen und eine zusätzliche ungültige Äquivalenzklasse.
  + Falls die Einschränkung oder Bedingung eine **Situation** beschreibt, die zwingend erfüllt werden muss, ist jeweils eine gültige und ungültige Äquivalenzklasse zu berücksichtigen.
* Bestehen Zweifel an der **Gleichbehandlung von Werten** innerhalb einer Äquivalenzklasse, sollte die Äquivalenzklasse weiter unterteilt werden.
* Die **Repräsentanten** aller gültigen Äquivalenzklassen sind zu Testfällen zu kombinieren, d.h. alle möglichen Kombinationen der jeweiligen Repräsentanten sind vorzusehen. Jede dieser **Kombinationen** bildet einen gültigen Testfall.
* Der Repräsentant einer ungültigen Äquivalenzklasse ist nur mit Repräsentanten von anderen gültigen Äquivalenzklassen zu Kombinieren. Für jede ungültige Äquivalenzklasse ist somit ein extra **„Negativ“-Testfall** zu spezifizieren.
* **Reduzierung der Testfallmenge:**
* Die **Testfälle** aus allen Repräsentanten kombinieren und nach Benutzungsrelevanz (häufig vorkommende Kombinationen oder Benutzungen) priorisieren.
* Es werden Testfälle bevorzugt, die **Grenzwerte** oder Grenzwertkombinationen enthalten.
* Sicherstellen, dass jeder Repräsentant einer Äquivalenzklasse mit jedem Repräsentanten der anderen Äquivalenzklasse in einem Testfall zur Ausführung kommt (d.h. **paarweise Kombination** statt vollständiger Kombination).
* Als **Minimalkriterium** sicherstellen, dass jeder Repräsentant einer Äquivalenzklasse in mindestens einem Testfall vorkommt.
* Repräsentanten ungültiger Äquivalenzklassen nicht mit Repräsentanten anderer ungültiger Äquivalenzklassen kombinieren.
* **Ungültige Werte separat testen:** Ein ungültiger Wert soll nur mit gültigen kombiniert werden, da ein ungültiger Parameter eine Ausnahmebehandlung auslöst. Kombiniert ei9n Testfall mehrere ungültige Werte, kann es zu einer gegenseitigen Fehlermaskierung kommen und es ist beim Auftreten einer Fehlerwirkung nicht klar, welcher ungültige Wert die Wirkung ausgelöst hat.

**Steckbrief: Äquivalenzklassenbildung**

**Grundidee:** Auf Basis der Spezifikation gibt es gleichartige Daten; für diese verhält sich das System falsch oder richtig.

**Klasse:** Spezifikationsbasierte Testverfahren

**Sucht nach:** Falschem Verhalten des Systems bei bestimmten Datenwerten.

**Qualitätsmerkmale:** Funktional

**Überdeckungsmaße:** Mindestens ein Testfall pro Äquivalenzklasse.

Weitere Testfälle je nach gewählter Kombinatorik.

**Wann:** Immer, wenn im Rahmen eines Testfalles Eingabedaten gebraucht werden.

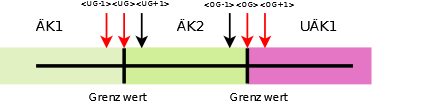
**Verwandt mit:** Domain-Testing, Klassifikationsbaum-Methode

**Kombinierbar mit:** z.B. Anwendungsfallbasiertem Test, Grundlage für Grenzwertanalyse.

* + 1. Grenzwertanalyse

Die **Grenzwertanalyse** ist eine Verfeinerung der Äquivalenzklassenbildung. Grenzwerte sind die Werte oder Bedingungen, die an den äußeren Grenzen der Klassen vorkommen. Ein Test mit Grenzwerten deckt oft Fehlerwirkungen auf, da in Programmen häufig Fehler bei der Behandlung von Grenzwerten und benachbarten Werten auftreten. Diese Methode lässt sich nur anwenden, wenn die Menge der Daten, die in eine Äquivalenzklasse fallen, geordnet ist und sich Grenzen identifizieren lassen.

* **Testfallableitung:**
* Zu jeder Äquivalenzklasse sind die **Werte auf den Grenzen** der Äquivalenzklasse und die beiden benachbarten Werte als **Repräsentanten** zu bestimmen (die Grenzwerte innerhalb der Äquivalenzklasse werden nicht immer getestet, da sie derselben Äquivalenzklasse angehören, wie die Grenzwerte selbst).
* Grenzwerte außerhalb der Äquivalenzklasse sind gleichzeitig Repräsentanten der benachbarten Äquivalenzklassen.
* Bei „offenen“ Zahlenmengen kann die rechnerabhängige Grenze der darstellbaren Zahlen als Grenzwert gewählt werden.
* Ist bei den **Ausgabewerten** die erlaubte **Anzahl** entscheidend, ist in Analogie zur Anzahl bei den Eingabewerten zu verfahren.
* Bei **geordneten Mengen** sind das erste und das letzte Element für den Test von besonderem Interesse.
* Sind als Ein- oder Ausgaben **komplexe Datenstrukturen** gegeben, so können beispielsweise eine leere Liste oder die Nullmatrix als Grenzwerte angesehen und für den Test berücksichtigt werden.
* Zusätzlich sollten Testfälle mit sehr großen Datenstrukturen, Listen und Tabellen usw. durchgeführt werden, um beispielsweise Puffer-, Datei- oder Speichergrenzen zu überschreiten und somit das Verhalten des Testobjekts in Extremfällen zu prüfen.
* Bei Listen und Tabellen sind leere und volle Listen und das erste und das letzte Element von Interesse.



**Steckbrief: Grenzwertanalyse**

**Grundidee:** Bei Wertebereichen, die Ränder haben, werden gezielt die Werte an den Rändern im Test berücksichtigt.

**Klasse:** Spezifikationsbasierte Testverfahren

**Sucht nach:** Fehlern, die bei der Umsetzung von Grenzen zwischen Wertebereichen gemacht werden.

**Qualitätsmerkmale:** Funktional

**Überdeckungsmaße:** Anteil der im Test verwendeten Grenzwerte zur Gesamtzahl der Grenzwerte - von diesen gibt es je nach Testtiefe 2 oder 4 je Intervall (2 oder 3 Grenzwerte je Grenze).

**Wann:** Immer, wenn ein Wertebereich sortierbar ist und Grenzen enthält.

**Verwandt mit:** Domain-Testing, Äquivalenzklassenbildung

**Kombinierbar mit:** Äquivalenzklassenbildung, zustandsbasiertem Testen.

* + 1. Domain-Testing

Domain-Testing setzt die Grundideen aus Äquivalenzklassen- und Grenzwertanalyse gemeinsam um. Domain-Testing berücksichtigt dabei mehrdimensionale Klassen (mehrere, fachlich zusammenhängende Parameter) und testet damit falsch definierte Bereichsgrenzen. Beim Domain-Testing werden zwei Methoden unterschieden:

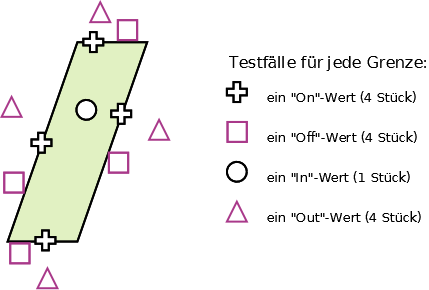
* **Invariant Boundaries** (unveränderliche Grenzen)
* **Pfadbereichstest**
* **Testfallableitung - Invariant Boundaries:**
* Für alle Eingabevariablen werden die Grenzen bestimmt. Dabei muss zwischen einer offenen und einer geschlossenen Grenze unterschieden werden:

**Open Boundary:**

* enthält keine Gleichheitsbedingung
* **On-Point** liegt auf der Grenze🡺 Grenzbedingung ist verletzt
* **Off-Point** liegt innerhalb des Grenzbereichs🡺 Grenzbedingung wird nicht verletzt

**Closed Boundary:**

* enthält Gleichheitsbedingung
* **On-Point** liegt auf der Grenze🡺 Grenzbedingung ist erfüllt
* **Off-Point** liegt außerhalb des Grenzbereichs🡺 Grenzbedingung wird verletzt
* Für jede Variable, die innerhalb der Grenzbereiche liegt, werden Testwerte ausgewählt
* Für jede Variable, die außerhalb der Grenzbereiche liegtwerden Testwerte ausgewählt
* Für jede Grenze wird folgendes gefordert:
* einen **„On“-Point** - Wert auf der Bereichsgrenze
* einen **„Off“-Point** - Wert nicht auf der Bereichsgrenze
* einen **„In“-Point** - Wert, der die Eingabebedingung(en) erfüllt
* einen **„Out“-Point** - Wert, der die Eingabebedingung(en) nicht erfüllt

****

* **Testfallableitung - Pfadbereichstest:**

Das Ziel des **Pfadbereichstests** ist die Entdeckung der Bereichsfehler.

Es können zwei Fehlerklassen gefunden werden:

* **Bereichsfehler:** Ausführung eines falschen Programmpfads
* **Berechnungsfehler:** korrekter Pfad mit falscher Berechnung

Jedem ausführbaren Pfad kann ein Teilbereich der Eingabewerte und die von dem Pfad kalkulierte Funktion zugeordnet werden.

Ein Programm kann in Form von - möglicherweise unendlich vielen - Paaren (Teilbereich, Funktion) dargestellt werden.

Vor dem Testen werden zuerst alle möglichen **Pfade** des Programms bestimmt, d.h. auf wie viele unterschiedliche Arten kann das Programm durch laufen werden. Danach wird bestimmt, bei welchem Eingabewertebereich dieser Pfad durch laufen wird.

Abschließend wird dann die erwartete Berechnung für den angegebenen Pfad bestimmt, bzw. welche Ergebnisse zu erwarten sind, wenn der richtige Pfad mit den zuvor definierten Werten durchläuft.

Die kritischen Stellen sind wieder die **Bereichsgrenzen**. Es werden daher nur Testfälle aus dem Grenzbereich erstellt. Dies geschieht wie beim **Invariant Boundary** durch Setzen der **ON/OFF-Points**.

**Steckbrief: Domain-Testing**

**Grundidee:** Analyse der Domains mit anschließender Auswahl mindestens eines „On“, „Off“, „In“ und „Out“

Wertes pro Domain.

**Klasse:** Spezifikationsbasierte Testverfahren

**Sucht nach:** Fehlern bei der Behandlung der Grenzen zwischen Ergebnisklassen/Subdomains

**Qualitätsmerkmale:** Funktional

**Überdeckungsmaße:** Anteil der im Test abgedeckten „On“, „Off“, „In“ und „Out“ Werte aller Domains zur Gesamtzahl dieser Werte.

**Wann:** Wenn mehrere Parameter nur gemeinsam betrachtet werden können.

**Verwandt mit:** Äquivalenzklassenbildung, Grenzwerttest

**Kombinierbar mit:** z.B. Anwendungsfallbasiertem Test

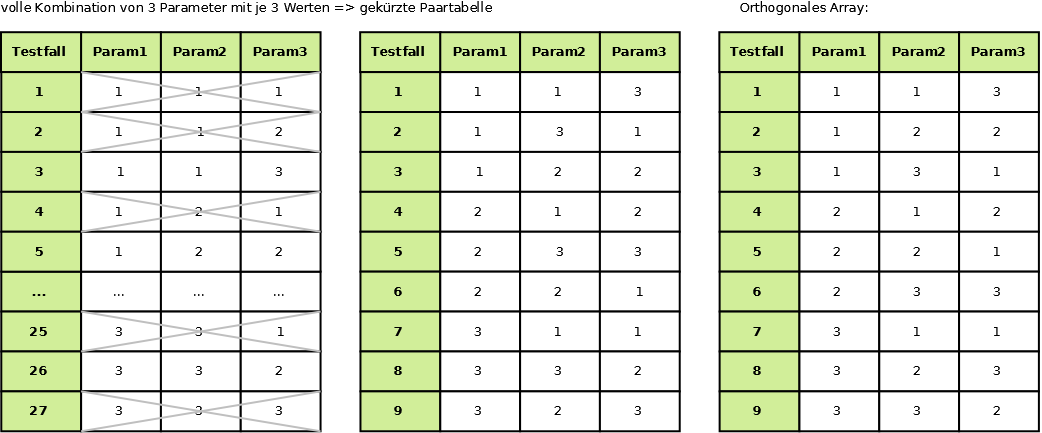
* + 1. Paarweises Testen (Orthogonale Arrays, Paartabellen)

Mithilfe dieser Testverfahren wird festgelegt, welche repräsentativen **Kombinationen** getestet werden, um die Anzahl der Tests von Kombinationen möglicher Belegungen mehrerer Eingabefelder gering zu halten. Das **paarweise Testen** und **orthogonale Arrays** sollten jedoch nur für Werte angewendet werden, die sich gegenseitig nicht beeinflussen. Bei beiden Verfahren werden die Optionen paarweise betrachtet, es unterscheidet sich lediglich die Paarbildung.

Das **paarweise Testen** basiert auf der Annahme, dass sich nicht alle Objekte gegenseitig beeinflussen, die man erhalten würde, wenn man alle möglichen Kombinationen in Betracht zöge.

Beim paarweisen Testen nimmt man sich die Optionen paarweise vor, streicht die unmöglichen oder unwahrscheinlichen Kombinationen und testet alle realistischen Paarkombinationen.

Ein **orthogonales Array** ist ein zweidimensionales Array mit speziellen mathematischen Eigenschaften, bei dem jede Kombination von zwei Spalten alle Kombinationen der Werte enthält. Orthogonale Arrays findet man im Internet. Die Arrays müssen eventuell noch angepasst werden.



**Testfallableitung:**

Um von einer Tabelle mit allen Kombinationsmöglichkeiten zu einer Tabelle nach der **Pairwise-Methode** zu gelangen, kann man folgendes Verfahren einsetzen:

Man betrachtet nacheinander die Zeile der Tabelle von unten nach oben. Gibt es jede der in der betrachteten Zeile vorkommenden Kombinationen von jeweils zwei Werten nochmals in einer der anderen Zeilen, kann man die betrachtete Zeile aus der Tabelle löschen. Dies setzt man so lange fort, bis keine Zeile mehr gelöscht werden kann.

**Steckbrief: Paarweises Testen (Orthogonale Arrays, Paartabellen)**

**Grundidee:** Die Repräsentanten aller Parameter werden zueinander in definiertem Umfang

in Beziehung gesetzt.

**Klasse:** Spezifikationsbasierte Testverfahren

**Sucht nach:** Fehlern, die nur beim Zusammenspiel bestimmter Werte oder Wertetypen ausgelöst werden.

**Qualitätsmerkmale:** Funktional

**Überdeckungsmaße:** Anteil aller getesteten Paarungen an den gesamt vorhandenen Paarungen festgelegter Ordnung und Anzahl

**Wann:** Wenn es auf eine gezielte Kombinatorik der Repräsentanten ankommt, auf mathematischer Grundlage und werkzeugunabhängig

**Verwandt mit:** Klassifikationsbaum-Methode

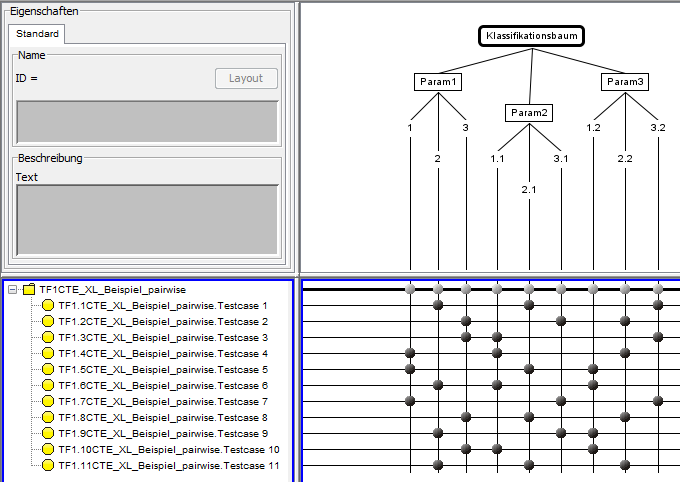
**Kombinierbar mit:** Äquivalenzklassen oder Grenzwerten

* + 1. Klassifikationsbaum-Methode

**Klassifikationsbäume** liefern eine grafische Darstellung der zu testenden Bedingungskombinationen. Die zu testenden Elemente werden in Form von **Klasse** und **Klassifizierungen** innerhalb der Klassen angelegt. Die Erstellung der Klassifikationsbäume kann mithilfe eines der kostenlos verfügbaren Werkzeuge erfolgen(z.B. CTE XL). Der Eingabedatenraum wird mit Hilfe eines Klassifikationsbaumes erstellt. Die Repräsentanten der Klassen werden nach vorher festgelegten Kombinationskriterien miteinander kombiniert und als Eingabewerte für die Testfälle bestimmt.

**Testfallableitung:**

* **Klassifikationsbaum erstellen:**
* Zerlegung der Menge der Eingabebedingungen nach testrelevanten Gesichtspunkten 🡺 Klassifikation
* Vollständige Zerlegung der Klassifikation in disjunkte Teilmengen🡺 (Äquivalenz-)Klassen
* rekursive Anwendung der Klassifikation auf die Klassen 🡺 Klassifikationsbaum
* **Testfälle ermitteln:**
* Blätter des Klassifikationsbaumes sind Kopfzeile einer Kombinationstabelle
* Testfälle ergeben sich aus der Kombination von Repräsentanten unterschiedlicher Klassen, wobei aus jeder Klassifikation nur eine Klasse berücksichtigt wird.
* Kombination nach vorher festgelegten Kombinationskriterien
  + **minimale Kombination**🡺 alle Klassen müssen wenigstens einmal vorkommen
  + **volle Kombination** 🡺 Alle Klassen werden miteinander kombiniert
  + **paarweise Kombination**🡺 Für Eine Teilmenge der Klassifikationen werden alle möglichen paarweisen Kombinationen der Klassifikationen gebildet und alle darin enthaltenen Klassen miteinander voll kombiniert.
  + **tripelweise Kombination**🡺 Für eine Teilmenge der Klassifikationen werden alle möglichen Tripel der Klassifikationen gebildet und alle darin enthaltenen Klassen miteinander voll kombiniert.



**Steckbrief: Klassifikationsbaum-Methode**

**Grundidee:** Aus einer Baumdarstellung abgeleitete Klassen von Werten oder Repräsentanten werden entsprechend ihrer gegenseitigen Beeinflussung kombiniert.

**Klasse:** Spezifikationsbasierte Testverfahren

**Sucht nach:** Fehlern, die nur beim Zusammenspiel bestimmter Werte oder Wertetypen ausgelöst werden.

**Qualitätsmerkmale:** Funktional

**Überdeckungsmaße:** Anteil der durch die Testfälle abgedeckten Kombinationen von Repräsentanten an der Gesamtzahl der Kombinationen, die sich anhand festgelegter Kombinationsregeln ergeben

**Wann:** Wenn es auf eine gezielte, jedoch effiziente Kombinatorik der Repräsentanten ankommt.

**Verwandt mit:** Orthogonale Arrays

**Kombinierbar mit:** Äquivalenzklassen oder Grenzwerten

* + 1. Entscheidungstabellen

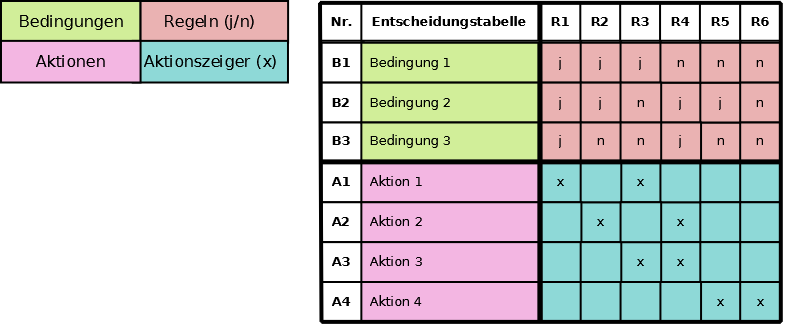
**Entscheidungstabellen** sind eine gute Möglichkeit, um Systemanforderungen zu erfassen, die **logische Bedingungen** enthalten und um den internen Systementwurf zu dokumentieren. Sie können zur Erfassung komplexer, von einem System umzusetzenden Regeln in Geschäftsprozessen verwendet werden.

Beim Erstellen einer Entscheidungstabelle wird die Spezifikation analysiert, und die Bedingungen und Aktionen des Systems werden ermittelt.

Die Eingabebedingungen und Aktionen werden meist so festgesetzt, dass sie entweder „wahr“ oder „falsch“ sein müssen (Boolesche Werte).

Die Entscheidungstabelle enthält die auslösenden **Bedingungen**, oft Kombinationen von „wahr“ und „falsch“ für alle Eingabebedingungen und die daraus resultierenden **Aktionen** für jede Kombination der Bedingungen. Die Stärke des Entscheidungstabellentests ist, dass er Kombinationen von Bedingungen ableitet, die andernfalls beim Test möglicherweise nicht ausgeführt worden wären. Er kann in allen Situationen angewandt werden, in denen die Abläufe der Software von mehreren logischen Entscheidungen abhängen.

* **Testfallableitung:**
* Eine **Entscheidungstabelle** besteht aus vier Bereichen
* Einer Auflistung der zu berücksichtigenden **Bedingungen**
* Einer Auflistung der möglichen **Aktionen**
* Einem Bereich, in dem die möglichen **Bedingungskombinationen** zusammengestellt sind
* Einem Bereich, in dem jeder Bedingungskombination die jeweils durchzuführenden **Aktivitäten** zugeordnet sind
* Jede Spalte der Tabelle entspricht einer Regel im Geschäftsprozess, die eine eindeutige Kombination der Bedingungen definiert, die wiederum die Ausführung der mit dieser Regel verbundenen Aktionen nach sich zieht.
* Der üblicherweise bei Entscheidungstabellentest verwendete Standardüberdeckungsgrad besagt, dass wenigstens ein Testfall pro Spalte in der Tabelle benötigt wird, was in der Regel die Abdeckung aller Kombinationen der auslösenden Bedingungen umfasst.



**Steckbrief: Entscheidungstabellentest**

**Grundidee:** Vollständige Darstellung der Möglichkeiten, der Auwand kann über Zusammenfassung reduziert werden.

**Klasse:** Spezifikationsbasierte Testverfahren

**Sucht nach:** Fehlern aufgrund übersehener, falscher, redundanter oder

widersprüchlich behandelter Bedingungskombinationen.

**Qualitätsmerkmale:** Funktional

**Überdeckungsmaße:** Anteil der getesteten Kombinationen an Gesamtzahl möglicher Kombinationen der Bedingungen, ggf. reduziert durch Zusammenfassung von Bedingungskombinationen mit gleichen Aktionszeigern

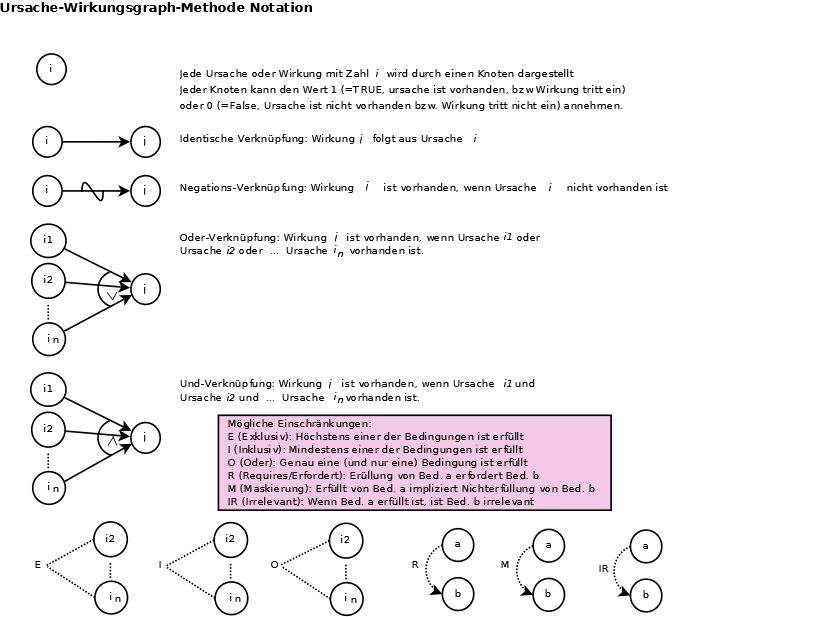
**Wann:** Wenn komplexes Zusammenspiel von Bedingungen vorliegt.

**Verwandt mit:** Ursache-Wirkungsgraph

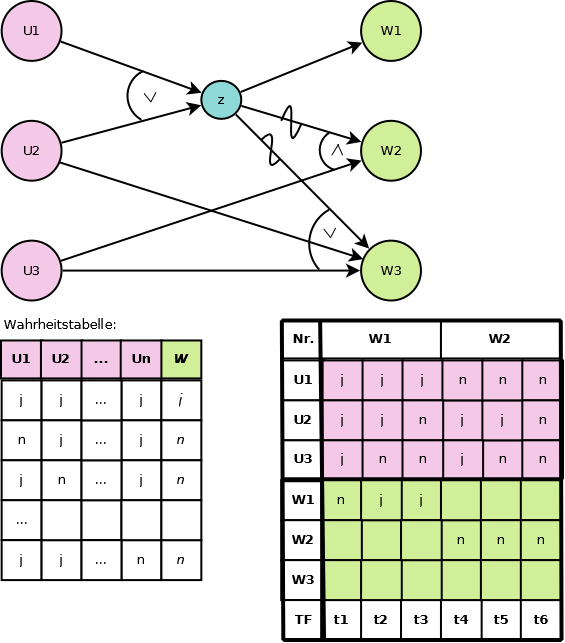
**Kombinierbar mit:** z.B. Anwendungsfallbasiertem Test

* + 1. Ursache-Wirkungsgraph Analyse

Die **Ursache-Wirkungsgraph-Methode** ist ein funktionsorientiertes, dynamisches Testverfahren, das die Spezifikation in eine grafische Form - den **Ursache-Wirkungs-Graphen** - umsetzt, aus dem über Zwischenschritte Testfälle abgeleitet werden.



* **Testfallableitung:**
* Spezifikation in handliche, bearbeitbare Stücke unterteilen.
* **Ursachen** und **Wirkung** des Programms werden anhand der Spezifikation festgelegt:
  + **Ursache** ist eine Eingabebedingung oder Äquivalenzklasse von Eingabebedingungen
  + **Wirkung** ist eine Ausgabebedingung oder Systemtransformation
* Ursachen und Wirkungen werden durch Zahlen eindeutig gekennzeichnet
* Die logischen Beziehungen zwischen Ursachen und Wirkungen werden als gerichteter Boolescher Graph, dem **Ursachen/Wirkungsgraph**, dargestellt.
* Der Ursachen/Wirkungsgraph kann zusätzlich mit Einschränkungen versehen werden, welche unmögliche Kombinationen von Eingangsbedingungen beschreiben. Diese Kombinationen bleiben später beim Ableiten der Testfälle unberücksichtigt.
* Der Ursachen/Wirkungsgraph wird in eine **Entscheidungstabelle** umgesetzt. Zeilen sind die Ursachen und Wirkungen, die Spalten repräsentieren Testfälle. Tabellenelemente sind bestimmte Belegungen der Ursachen und Wirkungen mit Wahrheitswerten. Ziel dieses Schrittes ist die Ermittlung der **Fehlersensitiven Testfälle**.
* Für die ermittelten Kombinationen von Ursachewerten müssen noch Eingangswerte ermittelt werden. Spalten mit Kombinationen von Ursachewerten, die sich nicht widersprechen bzw. redundant sind, werden zusammengefasst. Die Wirkungen ergeben sich anhand der **Ursache-Wirkungsbeziehungen** aus einem bestimmten Zusammenspiel von Ursachen automatisch.



**Steckbrief: Ursache-Wirkungsgraph Analyse**

**Grundidee:** Formalisierung von Anforderungen durch graphische Darstellung, danach mechanische Ableitung eines minimalen Satzes von (logischen) Testfällen bis hin zu Testdaten.

**Klasse:** Spezifikationsbasierte Testverfahren

**Sucht nach:** Fehlern, die aus unklaren Anforderungen im komplexen Zusammenspiel der Ursachen und Wirkungen folgen.

**Qualitätsmerkmale:** Funktional

**Überdeckungsmaße:** Jeder aus der Analyse abgeleitete fehlersensitive Testfall wurde durchgeführt.

**Wann:** Wenn komplexes Zusammenspiel von Bedingungen vorliegt.

**Verwandt mit:** Entscheidungstabellentest

**Kombinierbar mit:** z.B. Äquivalenzklassen oder Grenzwerten.

* + 1. Zustandsbasiertes Testen

Ein System kann in Abhängigkeit von aktuellen Gegebenheiten oder von seiner Vorgeschichte (seinem Zustand) unterschiedliche Reaktionen zeigen. In diesem Fall kann dieser Aspekt des Systems mit einem **Zustandsdiagramm** dargestellt werden. Es ermöglicht dem Tester, die Software darzustellen im Bezug auf ihre Zustände, Übergänge zwischen den Zuständen, Eingaben oder Ereignisse, die die **Zustandsübergänge** (transitions) auslösen, und Aktionen, die aus den Übergängen folgen können.

Die Zustände des Systems oder Testobjekts sind einzeln unterscheidbar, eindeutig identifizierbar und endlich in ihrer Anzahl.

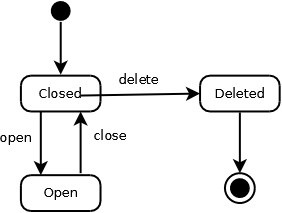
Eine **Zustandsübergangstabelle** stellt den Zusammenhang zwischen Zuständen und Eingaben dar und kann mögliche ungültige Übergänge aufzeigen.

Tests können so gestaltet sein, dass sie jeden Zustand oder eine typische Sequenz von Zuständen abdecken, jeden Übergang oder bestimmte Sequenzen von Übergängen ausführen, oder dass sie ungültige Übergänge überprüfen.

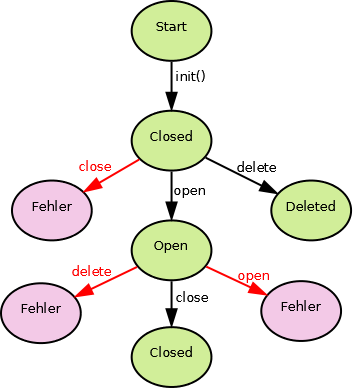
**Zustandsbasierte Tests** werden häufig für die Modellierung von Geschäftsobjekten, die verschiedene Zustände besitzen, oder zum Test von dialogbasierten Abläufen (z.B. für Internet-Anwendungen oder Geschäftsszenarios) eingesetzt.

**Testfallableitung:**

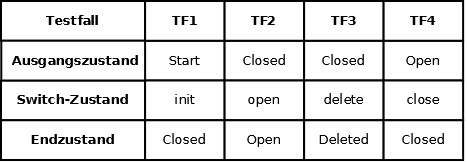
* **Zustandsübergangsdiagramm erstellen:**
  + Beginnend mit dem **Startzustand** nimmt das System unterschiedliche Zustände an
  + Übergänge zwischen den Zuständen werden durch **Ereignisse** (events) hervorgerufen.
  + Zustandsänderungen können Aktionen (actions) auslösen.
  + Ein weiterer spezieller Zustand ist der **Endzustand**.



* **Testkriterium festlegen:**
  + Alle Zustände sollen einmal erreicht werden.
  + Für jeden Zustand sollen alle Ereignisse, die für die von diesem Zustand ausgehenden Übergänge spezifizert sind, eintreten.
  + Alle möglichen Folgen von *n* Zustandsübergängen sollen durchgeführt werden (Chow´s Kriterium: *n*-1-switsch coverage)
* **Robustheitstest durchführen:**
  + Annahme: Für alle Zustände werden alle möglichen Ereignisse eintreten.
  + Wie reagiert das System auf Ereignisse, die für den Zustand, in dem es sich befindet, nicht vorgesehen sind? Das Sollverhalten muss vor Ausführung des Tests spezifiziert werden.
  + Zur Ermittlung der Testfälle wird evtl. ein künstlicher Zustand (z.B. Fehler) hinzugefügt.
  + Robustheitstest ist unabhängig vom gewählten Testkriterium.
* **oder/und Testfälle mit Hilfe eines Übergangsbaumes ermitteln:**
* Zustände werden durch Knoten dargestellt, Kanten repräsentieren Ereignisse.
* Startzustand wird die Wurzel des Baumes.
* Für jeden Folgezustand wird eine Verzweigung eingefügt, solange bis entweder
  + Knoten ein Endzustand ist, oder
  + der Zustand auf dem Weg von der Wurzel schon einmal vorgekommen ist.
* Jeder Pfad von der Wurzel zu einem Blatt stellt einen Testfall dar.
* Für jeden Zustand werden alle für diesen Zustand spezifizierten Funktionen durchgeführt.
* Beim Robustheitstest werden zusätzlich alle möglichen Funktionen durchgeführt.
* Man erhält eine Menge von Sequenzen von Zustandsübergängen ohne Zyklen.



* **oder/und Testfälle mit Hilfe einer Übergangstabelle ermitteln:**
  + Für eine Folge von *n* Zustandsübergängen entwickelt man eine Tabelle mit (n\*3)+1 Zeilen, wobei sich die Zeilen Zustand, Ereignis, Aktion wiederholen.
  + Die Spalten enthalten alle möglichen Folgen von *n* spezifizierten bzw. beim Robustheitstest allen möglichen Zustandsübergängen (Chow´s Kriterium), ausgehend von einem beliebigen Zustand.



* **Testfall spezifizieren:**
  + Nachdem die Zustandsübergänge, die durchexerziert werden sollen, festgelegt sind, ist zu spezifizieren:

**Vorbedingungen:** Anfangszustand, von dem die ermittelten Transitionen ausgehen

**Prüfanweisung:** für *n* Zustandsübergänge *n*-mal

**Eingabewert:** Ereignis

**Sollwert:** erwartet Aktion und Zustand, in den das System übergehen soll

**Nachbedingung:** Endzustand

**Steckbrief: Zustandsbasierter Test**

**Grundidee:** Vorgesehene und nicht vorgesehene Ereignisse werden in verschiedenen Sequenzen durchgeführt.

**Klasse:** Spezifikationsbasierte Testverfahren

**Sucht nach:** Fehlern, die in Abhängigkeit von der Reihenfolge der Ereignisse auftreten.

**Qualitätsmerkmale:** Funktional

**Überdeckungsmaße:** Anteil der erreichten Zustände, Zustandsübergänge oder Sequenzen festgelegter Länge von Zustandsübergängen zur jeweiligen Gesamtzahl

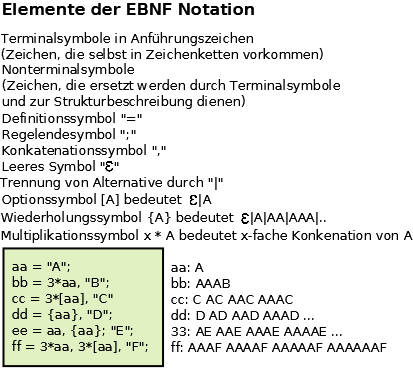
**Wann:** Bei Testobjekt „mit Gedächtnis“ bzw. Historie-/Zeit-/Zustands-abhängigem Systemverhalten.

**Verwandt mit:** Syntaxtest

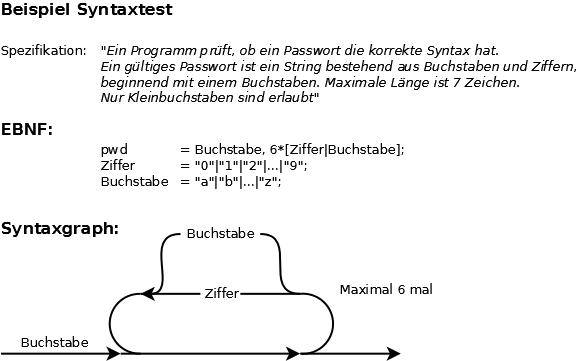
**Kombinierbar mit**: z.B. Äquivalenzklassen oder Grenzwerten.

* + 1. Syntaxtest

Mit Syntaxtest wird ein Verfahren zur Ermittlung der Testfälle bezeichnet, das bei Vorliegen einer formalen Spezifikation für die **Syntax der Eingaben** angewendet werden kann. Die Regeln der syntaktischen Beschreibung werden genutzt, um Testfälle zu spezifizieren, die sowohl die Einhaltung als auch die Verletzung der syntaktischen Regeln für die Eingaben berücksichtigen.



* **Testfallableitung:**
* **Kommandos in EBNF spezifizieren:**
  + Beginnend mit Elementen, die in vielen Eingabestrings erscheinen oder Schlüsselwörter, wird die EBNF Notation entwickelt.
  + Eingabestrings werden nach aufsteigender Komplexität (Anzahl der Elemente) und ähnlichem Muster sortiert.



* **Testfälle ermitteln:**
  + jede **EBNF Notation** eines Eingabestrings kann in einem **Syntaxgraphen** dargestellt werden.
  + Beim **Clean Syntaxtest** werden für gültige Eingabestrings Testfälle nach verschiedenen Testkriterien entwickelt.
  + Beim **Dirty Syntaxtest** wird die Reaktion auf ungültige Eingabestrings getestet, d.h. für die Elemente der Eingabestrings werden Werte eingegeben, die Regelverletzungen darstellen.
* **Testkriterien für Clean Syntaxtest:**
  + Alle Zweige eines **Syntaxgraphen**, der nur **Terminalsymbole** enthält, sollen durch einen Testfall abgedeckt werden
  + Alle Zweige des **Syntaxgraphen**, der auch **Nichtterminalsymbole** enthält, sollen durch einen Testfall abgedeckt werden. Das Nichtterminalsymbol wird im Testfall durch ein Terminalsymbol ersetzt.
  + Enthält der Syntaxgraph **Schleifen**, dann soll auch eine beschränkte Anzahl an Schleifenwiederholungen getestet werden.
* **Testkriterien für Dirty Syntaxtest:**
  + EBNF Notation ist hierarchisch aufgebaut. Für jede Ebene wird für jeweils ein Element der EBNF ein **Dirty Syntaxtest** durchgeführt. Alle anderen Elemente des Eingabestrings sollten gültige Werte erhalten, um eine Maskierung zu vermeiden.
  + Für jedes Element werden verschiedene Fehlerfälle durchexerziert: ein Element fehlt/ist zu viel, Element ist zu kurz/lang, Element enthält ungültiges Zeichen, …

**Steckbrief:** **Syntaxtest**

**Grundidee:** Alle denkbaren Fälle von Verletzungen der Regeln werden durchgespielt, sowie alle denkbaren Varianten gültiger Werte

**Klasse:** Spezifikationsbasierte Testverfahren

**Sucht nach:** Fehlerhafte Zurückweisung von Varianten zulässiger Eingaben, oder fehlerhafte Annahme unzulässiger Eingaben

**Qualitätsmerkmale:** Funktional

**Überdeckungsmaße:** Für Clean Syntaxtest wurden alle terminalen bzw. nicht-terminalen Symbole mit festgelegten Schleifenwiederholungen abgedeckt;

Für Dirty-Syntaxtest alle denkbaren Verletzungen der Syntax im Test durchgeführt.

**Wann:** Wenn Datenformate eingehalten werden müssen, z.B. Plausibilitätsprüfung

**Verwandt mit:** Zustandsbasiertem Test

**Kombinierbar mit:** z.B. Anwendungsfallbasiertem Test

Strukturorientierte Testverfahren

Strukturorientierte Testentwurfsverfahren heißen auch **White-Box- oder codebasierte Testverfahren**.

Dabei werden Code, Daten, Architektur oder Systemfluss als Grundlage des Testentwurfs verwendet und die Tests systematisch aus der Struktur abgeleitet. Der strukturorientierte Test baut auf der vorgefundenen Struktur der Software oder des Systems auf, wie aus folgenden Beispielen ersichtlich ist:

* **Komponentenebene:** Die Struktur der Softwarekomponente, d.h. Anweisungen, Entscheidungen, Zweige oder sogar einzelne Pfade
* **Integrationsebene:** Die Struktur kann ein Aufrufgraph sein (ein Diagramm, das zeigt, welche Module andere Module aufrufen)
* **Systemebene:** Die Struktur kann die Menüstruktur sein, Geschäftsprozesse oder die Struktur einer Webseite

Welche Elemente der Struktur berücksichtigt werden, hängt vom Verfahren ab. Das Verfahren liefert auch Kriterien für den **Überdeckungsgrad**, der entscheidet, wann die Ableitung von Tests abgeschlossen werden kann.

Wenn der Überdeckungsgrad vollständig ist, heißt das nicht, dass die Menge von Tests vollständig ist.

Es bedeutet vielmehr, dass die untersuchte **Struktur** für das vorliegende Testverfahren keine weiteren nützlichen Tests bietet. Der Überdeckungsgrad muss gemessen und mit dem Ziel verglichen werden, das für Projekt oder Unternehmen definiert wurde.

Gemeinsame Merkmale der strukturorientierten Testverfahren sind

* Die Testfälle werden aus Informationen darüber abgeleitet, wie die Software aufgebaut ist, beispielsweise dem Code und der Struktur.
* Der Überdeckungsgrad der Software lässt sich für die bestehenden Testfälle messen.

Zusätzliche Testfälle lassen sich systematisch ableiten, um die Überdeckung zu erhöhen.

Grundlage für die codebasierte Testfallableitung ist ein **Kontrollflussgraph**, der den Aufbau eines Programms auf graphische Weise repräsentiert.

* **Kontrollflussgraph:**
  + Der Kontrollflussgraph ist ein gerichteter Graph mit je einem Start- und Endknoten.
  + Die Befehle (Anweisungen) des Programms werden durch Knoten gekennzeichnet.
  + Eine Folge von Anweisungen ohne Verzweigung kann in einem Knoten zusammengefasst werden.
  + Die Pfade zwischen den Anweisungen werden durch Kanten (Kontrollflüsse) dargestellt.
  + Start- und Endpunkt des Graphen werden bei der Erstellung oft ausgelassen.

Die Instrumentierung, die Auswertung der Testläufe und die Ermittlung der erreichten Überdeckungen sind nicht von Hand vorzunehmen, da diese zu viele Ressourcen benötigt und eine manuelle Instrumentierung fehleranfällig wäre. Es gibt eine große Zahl von Werkzeugen, die diese Aufgaben übernehmen.

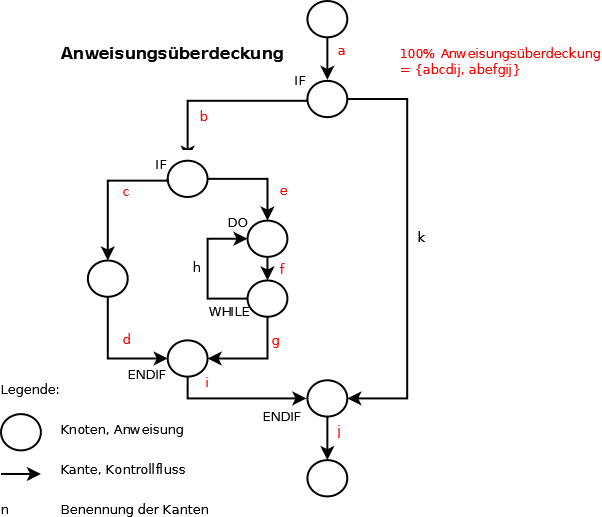
* + 1. Anweisungstests



Im **Komponententest** steht **Anweisungsüberdeckung** für die Messung des prozentualen Anteils von allen Anweisungen einer Komponente, welche durch eine Testsuite ausgeführt wurden. Das **Anweisungstestverfahren** leitet Testfälle so ab, dass bestimmte Anweisungen ausgeführt werden, in der Regel mit dem Ziel die Anweisungsüberdeckung zu erhöhen.

Anweisungsüberdeckung ist bestimmt durch die Anzahl ausführbarer Anweisungen, die durch entworfene oder ausgeführte Testfälle überdeckt sind, dividiert durch die Anzahl aller ausführbaren Anweisungen

des Programmcodes im Test.



**Steckbrief: Anweisungstests**

**Ziel:** Mindestens einmalige Ausführung jeder Anweisung

**Maß:** Anzahl durchlaufene Anweisungen zu Gesamtzahl Anweisungen

Einfach zu messendes, aber zu schwaches Kriterien 🡺 Leere Zweige/Kanten müssen nicht betreten werden, falls dort eine Anweisung „fehlt“, wird dies nicht entdeckt.

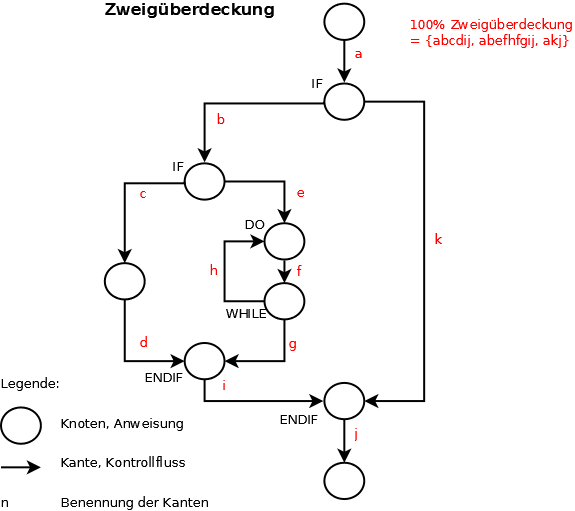
**Vorteil:** Toolunterstützung! Auffinden von „Dead-Code“

* + 1. Zweigtests/Entscheidungstests

Ein weitergehendes Kriterium im Whitebox-Test ist die Überdeckung der **Zweige** des Kontrollflussgraphen.

Nicht die Ausführung der einzelnen Anweisungen wird betrachtet, sondern die Auswertung einer Bedingung.

Die Zweigüberdeckung verlangt, dass bei einer **Verzweigung des Kontrollflusses** alle Möglichkeiten und bei Schleifen neben dem Schleifenkörper auch die Umgehung bzw. ein Rücksprung zum Schleifenanfang zu berücksichtigen sind.

****

Die **Entscheidungsüberdeckung**, die mit dem Zweigtest verwandt ist, ist die Messung des prozentualen Anteils eines Entscheidungsergebnisses (z.B. „wahr“ und „falsch“ bei einer IF-Anweisung), welche durch eine Testsuite ausgeführt wurden.

Beim Entscheidungstestverfahren werden Testfälle abgeleitet, um spezifische Entscheidungen zu durchlaufen. Zweige nehmen ihren Anfang in Entscheidungspunkten des Programmcodes und zeigen die Übertragung der Steuerung zu verschiedenen Stellen im Code.

Die Entscheidungsüberdeckung ist bestimmt durch die Anzahl aller **Entscheidungsausgänge**, die durch entworfene oder ausgeführte Testfälle überdeckt sind, dividiert durch die Anzahl aller Entscheidungsausgänge des Programmcodes im Test.

**Steckbrief: Zweigtests/Entscheidungstests**

**Ziel:** Ausführungaller Zweige = aller Kanten des Kontrollflussgraphen

**Maß:** Durchlaufene Zweige zu Gesamtzahl Zweige

Einfach zu messendes minimales Kriterium, denn Einfluss von Zweigreihenfolge, Schleifen und Bedingungen bleibt unbeachtet.

**Vorteil:** Toolunterstützung! Auffinden von „**Bottlenecks**“

* + 1. Bedingungstests
* **einfache Bedingungsüberdeckung:**

Ziel der einfachen **Bedingungsüberdeckung** ist, dass jede atomare Teilbedingung im Test sowohl den Wert *wahr* als auch den Wert *falsch* angenommen hat. Eine atomare Teilbedingung ist eine Bedingung, die keine logischen Operatoren wie AND, OR oder NOT, sondern höchstens Relationssymbole wie „>“ oder „=“ enthält. Eine Bedingung im Programmtext des Testobjekts kann aus mehreren Teilbedingungen zusammengesetzt sein.

Die einfache Bedingungsüberdeckung ist somit ein schwächeres Kriterium als die Anweisungs- oder auch Zweigüberdeckung, da nicht verlangt wird, dass unterschiedliche Wahrheitswerte bei der Auswertung der Gesamtbedingung im Test zu berücksichtigen sind.

**Steckbrief einfache Bedingungsüberdeckung**

**Ziel:** Alle atomaren Bedingungen in Schleifen und Auswahlanweisungen mindestens einmal als war und einmal als falsch ausführen.

**Maß:** ist nicht unbedingt ein umfassenderes Kriterium als die Zweigüberdeckung

* **Mehrfachbedingungsüberdeckung:**

Bei der **Mehrfachbedingungsüberdeckung** wird gefordert, dass Kombinationen der Wahrheitswerte der atomaren Teilbedingungen berücksichtigt werden. Es sollen möglichst alle Variationen gebildet werden.

Die Mehrfachbedingungsüberdeckung erfüllt auch die Kriterien der Anweisungs- und Zweigüberdeckung. Sie ist ein umfassenderes Kriterium, da sie auch die Komplexität bei zusammengesetzten Bedingungen berücksichtigt.

**Steckbrief: Mehrfachbedingungsüberdeckung**

**Ziel:** Alle atomaren Bedingungen sowie alle Bedingungskombinationen in Schleifen und Auswahlanweisungen mindestens einmal als wahr und einmal als falsch ausführen.

**Maß:** Erfüllt die Kriterien der Anweisungs- und Zweigüberdeckung

* **Minimale Mehrfachbedingungsüberdeckung:**

Die minimale **Mehrfachbedingungsüberdeckung** berücksichtigt jede Kombination von Wahrheitswerten, bei denen die Änderung des Wahrheitswertes einer atomaren Bedingung den Wahrheitswert der logischen Verknüpfung ändern kann.

* + 1. Pfadtests

Die **Pfadüberdeckung** fordert die Ausführung aller unterschiedlichen Pfade durch ein Testobjekt. Ein Pfad beschreibt die mögliche Abfolge von einzelnen Programmteilen in einem Programmstück.

Im Gegensatz dazu werden **Zweige** unabhängig voneinander betrachtet, jeder für sich. Die **Pfade** berücksichtigen Abhängigkeiten zwischen den Zweigen, wie zum Beispiel Schleifen, bei denen ein Zweig an den Anfang eines anderen Zweiges zurückführt. Kommen im Programmstück noch Schleifen hinzu, so zählt jede mögliche Anzahl von Schleifenwiederholungen als ein möglicher Pfad durch das Programmstück.

Eine 100%ige Überdeckung aller Pfade in einem Programm während der Testphase wird nicht erreichbar sein, sobald das Programm nicht trivial ist.

**Steckbrief: Pfadtests**

**Ziel:** Ausführung aller unterschiedlichen Pfade

**Maß:** Durchlaufene unterschiedliche Pfade zu Gesamtzahl Pfade; Einfluss von Zweigreihenfolge, Schleifen und Entscheidungsausgängen wird beachtet.

**Vorteil:** Wegen der sehr langen bzw. potentiell unendlich langen Pfade unpraktikabel.

Fehlerbasierte und erfahrungsbasierte Testverfahren

* **Ausgangsbasis bei fehlerbasierten Tests:**

Bei fehlerbasierten Testverfahren dient die Kategorie des gesuchten Fehlers als Ausgangsbasis für den Testentwurf. Die Tests werden systematisch auf Basis dessen abgeleitet, was über angenommene oder tatsächliche Fehler bekannt ist.

* **Überdeckungsgrad bei fehlerbasierten Tests:**

Vollständigkeit ergibt sich daraus, dass aufgrund der Informationen über die bekannten Fehler keine sinnvollen Testfälle mehr abgeleitet werden können (was nicht bedeutet, dass die so gefundenen Testfälle ausreichend sind).

* **Ausgangsbasis bei erfahrungsbasierten Tests:**

Bei erfahrungsbasierten Testverfahren können auch Fehlerhistorien eine Rolle spielen, der Schwerpunkt liegt aber auf den persönlichen Erfahrungen der Tester, auf ihrer fachlichen Kompetenz und Intuition.

Die Tests sind effizient zum Aufspüren von Fehlern, bieten aber wenig Information im Bezug auf Freigabeentscheidungen.

* **Überdeckungsgrad bei erfahrungsbasierten Tests:**

Systematische Überdeckungsmaße werden nicht oder nur selten formal erfasst.

* + 1. Fehler-Taxonomien

**Taxonomien** können Fehlerursache, Fehlerzustand und Fehlerwirkung beinhalten.

Daneben können folgende Aspekte eingehen:

* Art der zugrundeliegenden Fehlhandlung: wurde etwas übersehen, missverstanden, nicht geglaubt oder ignoriert
* Wie kann das Problem behoben werden: Etwas hinzufügen, löschen, ändern …
* Wie viele Personen sind an der Entstehung des Problems beteiligt: eine, wie, mehrere
* In welcher Art von Element ist der Fehler enthalten: boolescher Ausdruck, Objektname, Name der Operation, Liste, Einheit, Menge, String …

Es gibt veröffentlichte Taxonomien (Kategorien) von Fehlern, anhand derer mögliche Bereiche für das **fehlerbasierte Testen** identifiziert werden können. Der Genauigkeitsgrad von Taxonomien reicht von der sehr breit gefächerten Kategorie der Fehler in der Benutzeroberfläche bis hin zur detaillierten Liste, die in der Norm IEEE 1044 **(Standard Classification for Software Anomalies)** zur Verfügung gestellt wird.

In dieser Liste werden Eingabeprobleme in folgende Kategorien unterteilt:

* Korrekt Eingabe nicht akzeptiert
* Falsche Eingabe akzeptiert
* Beschreibung ist falsch oder fehlt
* Parameter sind unvollständig oder fehlen

Taxonomien können verwendet werden, um gefundene Fehler zu kategorisieren sowie um die Fehlerarten festzulegen, die gesucht werden sollen.

Je detaillierter die verwendete Taxonomie, desto genauer das Testen. Wie bei allen Checklisten sollte man jedoch nicht zu sehr auf die Liste fixiert sein und dabei Punkte übersehen, die nicht auf der Liste enthalten sind.

|  |  |
| --- | --- |
| **Ebene 1** | **Ebene 2** |
| Anforderungen  *(8,12 % der Fehler)* | Anforderungen falsch |
| Anforderungslogik falsch |
| Anforderungen unvollständig |
| Anforderungen nicht überprüfbar |
| Präsentation & Dokumentation |
| Änderungen in den Anforderungen |
| Eigenschaften & Funktionalität  *(16,19 % der Fehler)* | Eigenschaft oder Funktion falsch |
| Eigenschaft unvollständig |
| Anwendungsfall nicht vollständig |
| Fachliche Fehler |
| Anwendermitteilungen und Diagnostik |
| Falsche Behandlung der Ausnahmebedingungen |
| Strukturelle Fehler  *(25,18 % der Fehler)* | Fehler in Kontrollfluss und Struktur |
| Fehler in Verarbeitung |
| Daten  *(22,44 %der Fehler)* | Datendefinition und -struktur |
| Datenzugang und -behandlung |
| Implementierung & Codierung  *(9,88 % der Fehler)* | Codierungs- und Tippfehler |
| Missachtung von Richtlinien & Normen |
| Schlechte Dokumentation |
| Integration  *(8,98 % der Fehler)* | Interne Schnittstellen |
| Externe Schnittstellen, zeitliche Planung, Durchsatz |
| System- und Softwarearchitektur  *(1,74 % der Fehler)* | Aufrufe und Einsatz des Betriebssystems |
| Softwarearchitektur |
| Wiederherstellung und Verantwortlichkeit |
| Performanz |
| Falsche Diagnostik & Erwartungen |
| Klassen & Overlays |
| Umgebung |
| Testdefinition und - ausführung  *(2,76 % der Fehler)* | Fehler im Testentwurf |
| Fehler in der Testausführung |
| Schlechte Testdokumentation |
| Unvollständige Testfälle |

**Beispiel:** Die ersten beiden Ebenen von Beizers Fehlertaxonomie

**Steckbrief: Fehler-Taxonomien**

**Grundidee:** Testfälle werden gezielt im Hinblick auf bekannte, immer wieder auftretende Fehlermuster hin gestaltet

**Klasse:** Fehler- und Erfahrungsbasiert

**Unterklasse:** Fehlerbasiert

**Sucht nach:** Fehlern, die bekannten Fehlermustern entsprechen.

**Qualitätsmerkmale:** Alle

**Überdeckungsmaße:** Passende fehlerhafte Daten und Typen von Fehlern

**Wann:** Geeignete Taxonomien sind vorhanden

**Verwandt mit:** Checklisten-basiertem Test

**Kombinierbar mit:** Fehlererwartungsmethode

* + 1. Ad-hoc-Testen

Testverfahren, bei dem Fehler durch spontanes ungeplantes Ausprobieren der Software gefunden werden, ohne dass eine anerkannte Technik zur Testfallableitung verwendet wird.

* **Kein Plan/Konzept, keine Spezifikation werden erstellt:**

**Vorteile:**

* Keine zusätzlichen Kosten/Zeit
* Keine Vorbereitungszeit nötig, es kann sofort begonnen werden
* Schneller, intuitiver Eindruck über Zustand des Systems

**Nachteile:**

* Kein definiertes Testziel oder Testthema oder vielleicht nur im Kopf des zuständigen Testers
* Keine Grundlage für ein Review/Kommunikation mit Entwicklern
* **Testfälle werden nicht aufgezeichnet/automatisiert:**

**Vorteile:**

* Kein Zeitaufwand für Testfallimplementierung und Automatisierung

**Nachteile:**

* Wiederholbarkeit der Testfälle nicht sicher, Fehler sind evtl. nicht reproduzierbar
* Regressionstests nicht/schwer möglich
* Keine Protokollierung, keine Aussage über die Qualität des Testobjekts oder Fortschritt bei Korrekturmaßnahmen
* Keine Testendebewertung

**Steckbrief: Ad-hoc-Test**

**Grundidee:** Testen ohne Overhead, basierend auf Intuition und Erfahrung der Tester.

**Klasse:** Fehler- und Erfahrungsbasiert

**Unterklasse:** Erfahrungsbasiert

**Sucht nach:** Fehlern, die den Testern aufgrund ihrer Erfahrung auffallen.

**Qualitätsmerkmale:** Alle

**Überdeckungsmaße:** keine

**Wann:** Als Ergänzung zu systematischen Verfahren, wenn gute Tester mit tiefem Anwendungs-Knowhow verfügbar sind

**Verwandt mit:** Exploratives Testen

**Kombinierbar mit:** beliebigen systematischen Methoden

* + 1. Fehlererwartungsmethode

Testverfahren, das Erfahrung und Intuition des Testers ausnutzt, um Testfälle herzuleiten.

* **Vorgehensweise:**
* Tester listet alle möglichen Fehler oder fehlerträchtige Situationen auf, die er aus seiner Praxiserfahrung kennt, z.B.:
* spezielle Werte: Division durch Null, leere Listen, Wurzel aus negativen Werten, …
* fehlerhafte Abfragen in der Programmlogik
* Bekannte Produktrisiken

Falls möglich, können hierzu auch Fehlerdatenbanken von vorhergehenden Versionen oder ähnlichen Projekten analysiert werden.

* Tester versucht die Überlegungen des Entwicklers beim Lesen der Spezifikation nachzuvollziehen:
* Gibt es vergessene Punkte, die für den Designer offensichtlich waren, aber nicht für den Entwickler (Dokumentationsfehler)
* Zusammenhänge, die vom Entwickler nicht erkannt wurden (Schnittstellenfehler, Synchronisationsfehler, Konfigurationsfehler)

Der Überblick und die besonders gute Kenntnis des Gesamtsystem des Testers wird genutzt.

**Vorteile:**

* Praxiserfahrung des Testers wird genutzt
* Erfahrung aus ähnlichen Vorgängerprojekten wird genutzt
* Besondere Kenntnisse der Tester werden genutzt

**Nachteile:**

* Nur mit entsprechend qualifiziertem Personal möglich

**Steckbrief: Fehlererwartungsmethode**

**Grundidee:** Hineinversetzen in die Entwickler: was für Fehler sind wahrscheinlich?

**Klasse:** Fehler- und Erfahrungsbasiert

**Unterklasse:** Erfahrungsbasiert

**Sucht nach:** Bekannten häufigen Fehlern und nach aufgrund der Informationsbasis zu erwartenden Fehlern

**Qualitätsmerkmale:** Alle

**Überdeckungsmaße:** Im Zusammenhang mit einer Taxonomie: Passende fehlerhafte Daten und Typen von Fehlern; ohne Taxonomie: abhängig von der Erfahrung des Testers und der verfügbaren Zeit, jedoch basierend auf den identifizierten möglichen Fehlern

**Wann:** Als Ergänzung zu systematischen Verfahren

**Verwandt mit:** AD-Hoc-Testen, Fehler-Taxonomien

**Kombinierbar mit:** Fehler-Taxonomien

* + 1. Checklisten-basiertes Testen

Erfahrene Tester nutzen abstrakte Checklisten von

* Dingen, die zu erfassen sind;
* Dingen, die zu prüfen sind;
* Dingen, die man nicht vergessen darf;
* Regeln und Kriterien, gegen die ein Produkt zu prüfen ist

Checklisten werden auf Grundlage von Standards, Erfahrungen und weiteren Erwägungen gestaltet.

Einsatzmöglichkeiten von Checklisten:

* Smoke-Tests bei der Übergabe von Software von einer Phase in die andere
* Zur Formalisierung, z.B. im Rahmen von explorativen Tests
* Im Rahmen der Fehlererwartungsmethode
* Zur Referenzierung aus Testspezifikationen
* Bei der Erstellung von Testspezifikationen
* Bei Reviews, abhängig vom Reviewobjekt

**Steckbrief: Checklisten-basiertes Testen**

**Grundidee:** Vollständigkeit der Prüfung wird dadurch unterstützt, dass Checklisten Prüfkriterien vorgeben.

**Klasse:** Fehler- und Erfahrungsbasiert

**Unterklasse:** Erfahrungsbasiert

**Sucht nach:** Abweichungen gegenüber der Checkliste

**Qualitätsmerkmale:** Alle

**Überdeckungsmaße:** Jedes Element der Checkliste wurde überprüft

**Wann:** Wenn Checklisten vorliegen

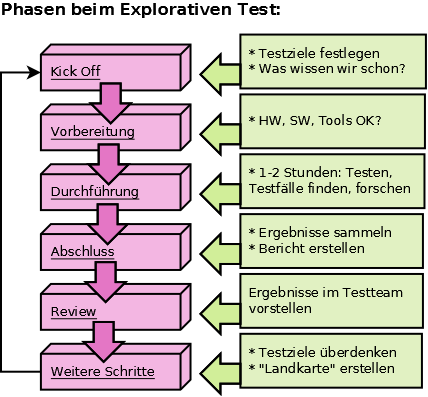
**Verwandt mit:** Fehler-Taxonomien, Fehlererwartungsmethode

**Kombinierbar mit:** z.B. Usability Test, Explorativem Testen

* + 1. Exploratives Testen

Von explorativem Testen spricht man dann, wenn der Tester die Tests gleichzeitig plant, entwirft und durchführt und das Produkt während der Durchfühung des Tests kennenlernt. Im Verlauf des Testens bestimmt der Tester basierend auf den bisherigen Ergebnissen, was als Nächstes getestet wird. Explorative Tests werden geplant und halten sich im Normalfall an eine **Test-Charta**, die eine allgemeine Beschreibung der Testziele enthält.

* **Test-Charta:**
* Besteht aus der Mission oder Aufgabe (Testziele), die von den Testern zu verfolgen ist.
* Und im Idealfall zusätzlich aus den Strategien, die sie bei der Durchführung der Mission zum Erfolg bringen sollen.
* Grundlegende externe Attribute bei Explorativem Testen:
* Begrenzte **Zeit**
* Möglichst qualifizierter **Tester**
* **Produkt**, das dem Test unterliegt
* **Mission** oder Aufgabe, die der Tester zu erfüllen hat
* **Bericht** oder Aufzeichnung, Protokoll über die Durchführung, einschließlich Testschritte und Ergebnisse



**Steckbrief: Exploratives Testen**

**Grundidee:** Das Testobjekt wird erforscht, Thesen über das Sollverhalten gebildet, erprobt und dokumentiert

**Klasse:** Fehler- und Erfahrungsbasiert

**Unterklasse:** Erfahrungsbasiert

**Sucht nach:** Verhalten des Testobjekts, das von den Erwartungen abweicht

**Qualitätsmerkmale:** Alle

**Überdeckungsmaße:** Die Test-Charta kann im Bezug auf eine Sitzung Ziele, Schwerpunkte und Betrachtungsumfang festlegen. Zusätzlich können Erfahrungswerte über Fehler und (mangelnde) Qualität einbezogen werden

**Wann:** Wenn kaum Informationen über das Testobjekt vorliegen

**Verwandt mit:** AD-Hoc-Testen

**Kombinierbar mit:** z.B. Checklistenbasiertem Testen

* + 1. Fehlerangriff

‚Fehlerangriff‘ ist ein gezielter Versuch, die Qualität, insbesondere die Zuverlässigkeit, eines Testobjektes zu untersuchen, indem versucht wird, bestimmte Fehler zum Auftreten zu bringen.

Schnittstellen - oder „User“ - die angreifbar sind:

* Der Mensch - über das GUI oder MMI (Man-Machine-Interface)
* Das File System (Dateiformate, Berechtigungen…)
* Das Betriebssystem (Ressourcenanforderungen wie Speicher, Datei-Handle …)
* Andere Software (Datenbanken, vor- und nachgelagerte Systeme …)

**Beiepiele:**

* Fehlerangriff Standard-Werte erzwingen:
* In Masken eingetragene Werte löschen und dann ohne Eintrag akzeptieren
* direkt nach dem Öffnen ohne Änderung akzeptieren
* Fehlerangriff Dateiinhalte ändern/beschädigen:
* Dateien mit anderen Mitteln als dem Testobjekt bearbeiten
* Schlüsselworte ändern
* Fehlerangriff Speicheranforderungen verweigern:
* Mit Werkzeugen Speicher künstlich begrenzen

**Steckbrief: Fehlerangriff**

**Grundidee:** Alle Schnittstellen werden daraufhin untersucht, ob über sie unerwünschtes Verhalten des Testobjekts auslösbar ist

**Klasse:** Fehler- und Erfahrungsbasiert

**Unterklasse:** Erfahrungsbasiert

**Sucht nach:** Verwundbarkeit des Systems

**Qualitätsmerkmale:** Zuverlässigkeit

**Überdeckungsmaße:** Die verschiedenen Schnittstellen des Systems, das getestet wird, wie z.B. das GUI, zum Betriebssystem oder Dateisystem

**Wann:** Wenn Robustheit einen hohen Stellenwert hat

**Verwandt mit:** Fehlererwartungsmethode, Checklisten-basiert

**Kombinierbar mit:** Fehler-Taxonomien

# Testen von Qualitätsmerkmalen



## Funktionalität und Interoperabilität

Unter Funktionalität versteht man die Fähigkeit der Software, beim Einsatz unter spezifizierten Bedingungen

* Funktionen zu liefern, die festgelegte und vorausgesetzte Erfordernisse erfüllen
* die richtigen oder vereinbarten Ergebnisse oder Wirkungen mit dem benötigten Grad an Genauigkeit zu liefern
* für spezifizierte Aufgaben und Anwenderziele einen geeigneten Satz Funktionen zu liefern

Das funktionale Testen ist der Grundstein jeder Testanstrengung. Der Umfang des funktionalen Testens ändert sich abhängig von der jeweiligen Stufe im Entwicklungszyklus.

Werden **Unittests** durchgeführt, liegt der Blickpunkt auf der Funktionalität der einzelnen Komponenten.

Bei **Integrationstests** werden die verschiedenen Schnittstellen funktional getestet, um zu überprüfen, ob die Software erfolgreich integriert wurde.

Bei **Systemtests** wird überprüft, ob innerhalb des Systems End-to-End Funktionalität vorliegt (bzw. bei Multisystemen wird die End-to-End Funktionalität über die einzelnen Systeme hinaus überprüft).

**Steckbrief: Test der Funktionalität**

**Zeitpunkt:** Alle Teststufen

**Typische Techniken:**

* Reviews
* Statische Analyse
* Black-Box-Testentwurf
* White-Box-Testentwurf

**Fehler, die gefunden werden:** Funktionale Fehler

Mit **Interoperabilitätstests** wird überprüft, ob die zu testende Software in allen Umgebungen korrekt funktioniert.

Eine Software mit guter Interoperabilität lässt sich leicht mit anderen Systemen integrieren, ohne dass größere Änderungen notwendig sind. Bestenfalls sind lediglich Konfigurationsparameter und Eigenschaftsdateien zu ändern. Die Anzahl und die Art der manuell notwendigen Änderungen bestimmen den Grad der Interoperabilität einer Software.

Da bei Interoperabilitätstests die Integration eine zentrale Rolle spielt, werden die häufig im Zusammenhang mit **Systemintegrationstests** durchgeführt. Diese sollten sich mit **Performanz-/Belastungstests** ergänzen.

**Steckbrief: Interoperabilitätstest**

**Zeitpunkt:** Systemintegrationstest

**Typische Techniken:**

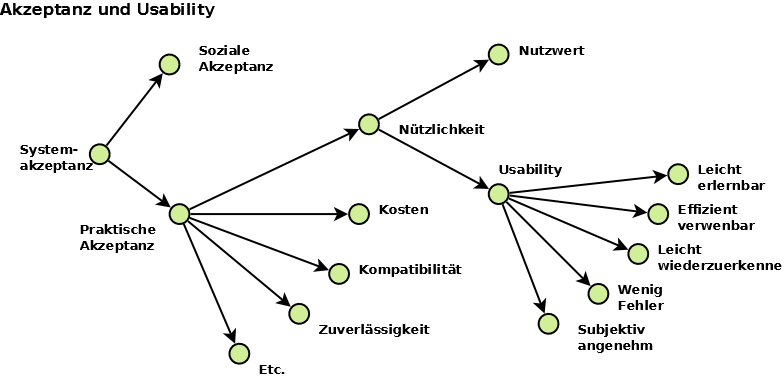
* Testentwurf, v.a. Black-Box
* Metriken, Reviews, Statische Analyse (z.B. Datenflussanalyse)

**Fehler, die gefunden werden:**

* Funktionale Fehler im Zusammenspiel zwischen Systemen
* Nichteinhaltung von Datenformaten, Kommunikationsstandards oder Protokollen
* Fehlerhafte Schnittstellendefinitionen
* Fehler bei der Datenübertragung

Benutzbarkeit

Usability bezeichnet ein Maß dafür, inwiefern eine Problemlösung für den Anwender verwendbar ist. Hierzu werden die Aspekte Erlernbarkeit, Effizienz, Wiedererkennbarkeit, enthaltene Fehler und subjektive Zufriedenheit in Bezug auf den Anwender betrachtet.



Teststrategien für die Benutzbarkeit konzentrieren sich entweder auf das Finden und Entfernen tatsächlicher Benutzbarkeitsfehler (formative Bewertung) oder auf das Testen anhand von Benutzbarkeitsanforderungen (summative Bewertung).

Um gute **Benutzbarkeitstests** durchführen zu können, muss die Software von mehreren verschiedenen Blickwinkeln aus betrachtet werden. Folgende Methoden eignen sich zur Beurteilung der Benutzbarkeit:

* **Inspektion und Review:** Die Anforderungsdokumente, Modelle, Flussdiagramme und Anwendungsfälle werden nach möglichen Benutzbarkeiteproblemen untersucht
* **Heuristische Evaluation:** Systematische Inspektion des Designs einer Benutzeroberfläche. Die 10 Kernpunkte der Usability Heuristik sind:
  + Sichtbarkeit des Systemzustandes
  + Übereinstimmung zwischen dem System und der Wirklichkeit
  + Kontrolle und Freiheit für den Benutzer
  + Konsistenz und Standards
  + Fehler vermeiden
  + Erkennen statt Erinnern
  + Flexibilität und Effizienz der Anwendung
  + Ästhetisches und minimalistisches Design
  + Hilfe und Dokumentation
  + Den Anwendern helfen, Fehlersituationen zu erkennen, zu verstehen und Lösungswege zu finden
* **Standardisierte Checklisten:** Kommerzielle, standardisierte Checklisten/Fragebögen mit umfangreichen zugrundeliegenden Datenbanken erlauben einen Abgleich der Ergebnisse mit dem Stand der Technik, können zudem auch als Abnahmekriterium dienen:
  + **SUMI** (**S**oftware **U**sability **M**easurement **I**nventory)
  + **WAMMI** (**W**ebsite **A**nalysis and **M**easure**M**ent **I**nventory)
  + **MUMMS** (**M**easuring the **U**sability of **M**ulti-**M**edia-**S**ystems)
* **Interaktionsmodelle:** Hierzu werden Anwender als Testpersonen eingesetzt und bei nachfolgenenden Interaktionen beobachtet
  + **Laut denken (Thinking aloud):** Die Testperson wird ersucht alle Gedanken laut zu äußern
  + **Konstruktive Interaktion (constructive Interaction):** Zwei Testpersonen lösen die Aufgabe gemeinsam im Dialog
  + **Rückblickendes Testen (Retrospective Testing):** Nach dem Test wird der Ablauf mit der Testperson, ggf. anhand einer Videoaufzeichnung, rückblickend besprochen
  + **Coaching-Methode (Coaching Method):** Der Testperson wird ein Coach zur Seite gestellt, der erklärt was zu tun ist.

**Steckbrief: Benutzbarkeitstest (Usability-Test)**

**Zeitpunkt:** Akzeptanztest (und möglichst früh)

**Typische Techniken:**

* Heuristische Evaluation
* Checklisten
* Reviews und Statische Analyse
* Interaktionsmodelle
* Beobachtung des Echteinsatzes

**Fehler, die gefunden werden:**

* Fehler und Mängel bzgl. Verständlichkeit, Erlern- und Bedienbarkeit
* Fehler im Zeitverhalten
* Nichteinhaltung von Standards und Styleguides

Barrierefreiheit

Barrierefreiheit ist ein Maß dafür, inwieweit behinderte Benutzer die Software verwenden können.

Anforderungen an **barrierefreie Software:**

* Die Software soll auch ohne Maus, nur mit der Tastatur bedienbar sein (für Screenreader, der ein GUI-Alternative für Sehbehinderte ist und für Benutzer mit motorischen Störungen)
* Soweit möglich, sollen Standardschnittstellen, Standard-Betriebssystemdienste und Standard-Dialogelemente verwendet werden (für Screenreader)
* Jedes sich öffnende Fenster soll das aktive Fenster sein
* Dropdown-Menüs oder Farbkombinationen, die der Benutzerführung dienen, stellen eine Barriere dar!
* Alternativtexte für Graphiken und Bilder sowie Alternativen für Frames, Tabellen, Applets, Plugins und Audio- und Multimedia-Elemente bereitstellen
* Text nicht in Form von Bitmaps darstellen. Blinkende Texte vermeiden. Skalierbare Schriftgrößen wählen
* HTML-Standards einhalten
* Die Webseite übersichtlich, leicht verständlich und einfach handhabbar gestalten (für Benutzer mit kognitiven Behinderungen).

Ideen für den **Barrieretest**:

* Checklisten von Anforderungen verwenden
* Mit unterschiedlicher Bildschirmauflösung und Farbwahl testen, Webseiten auch mit abgeschalteten Grafiken und Sound.
* Spezialsoftware wie Screenreader oder textbasierte Browser (z.B.: Lynx) im Test einsetzen.
* Werkzeuge für die statische Analyse von Webseiten auf Barrierefreiheit einsetzen (z.B.: Wave, A-Prompt, Bobby). Diese prüfen wahlweise nach den Richtlinien der verschiedenen Standards und Verordnungen.

**Steckbrief: Barrieretest (Zugänglichkeitstest)**

**Zeitpunkt:** Systemtest

**Typische Techniken:**

* Testentwurf, auch struktureller
* Checklisten
* Reviews und Statische Analyse

**Fehler, die gefunden werden:**

* Barrieren für bestimmte Benutzergruppen
* Fehler und Mängel in besonderen Zielumgebungen
* Nichteinhaltung von Standards, Schnittstellendefinitionen und Anforderungen

Sicherheit

Die Sicherheit bezeichnet Eigenschaften der Software, die sich auf ihre Fähigkeit beziehen, sowohl beabsichtigten als auch unbeabsichtigten unautorisierten Zugriff auf Programme oder Daten zu verhindern.

In **Sicherheitstests** wird die Anfälligkeit eines Systems gegenüber unterschiedlichen Bedrohungen untersucht.

Sicherheitstests unterscheiden sich von anderen Formen fachlicher oder technischer Tests in zwei

wichtigen Aspekten:

1. Standardverfahren zur Auswahl der Testeingangsdaten können wichtige Sicherheitsthemen übersehen.

2. Die Symptome von Sicherheitsfehlern unterscheiden sich grundlegend von Symptomen bei anderen Testverfahren.

Sicherheits-Schwachstellen gibt es an Stellen, wo die Software nicht nur funktioniert wie im Entwurf

vorgesehen, sondern wo sie zusätzlich weitere, nicht beabsichtigte Aktionen ausführt. Solche Nebeneffekte sind eine der größten Bedrohungen für die Softwaresicherheit gegenüber Angriffen.

Wichtigste Aufgabe der Softwaresicherheit ist es, eine nicht beabsichtigte Nutzung von Informationen durch nicht autorisierte Personen zu verhindern. Sicherheitstests versuchen, die Sicherheitsrichtlinien eines Systems außer Kraft zu setzen, indem sie seine Schwachstellen für Bedrohungen durch verschiedene Maßnahmen ausnutzen:

* Nicht autorisiertes Kopieren von Anwendungen oder Daten (beispielsweise Softwarepiraterie)
* Nicht autorisierter Zugriff (Aktionen auszuführen, für die der Nutzer keine Zugriffsberechtigung hat)
* Überlauf des Eingabebereichs (Pufferüberlauf), der beispielsweise durch Eingabe extrem langer Zeichenketten über ein Eingabefeld der Benutzerschnittstelle ausgelöst werden kann. Nach diesem Überlauf lässt sich möglicherweise bösartiger Code ausführen.
* Denial of Service-Angriff, sodass die Anwendung nicht mehr genutzt werden kann (beispielsweise indem ein Webserver von Übermengen von Störanfragen überschwemmt wird)
* Abhören von Datenübertragungen in Netzwerken, um an vertrauliche Informationen zu gelangen (beispielsweise über Kreditkarten-Transaktionen)
* Verschlüsselungscodes knacken, die vertrauliche Daten schützen
* Logische Fallen (logic bombs), die in bösartiger Absicht in den Code eingeschleust und nur unter bestimmten Bedingungen aktiviert werden (beispielsweise an einem bestimmten Datum). Werden diese logischen Fallen aktiviert, so lösen sie Schadaktionen aus, wie das Löschen von Dateien oder Formatieren von Festplatten.

Sicherheitsbelange lassen sich folgendermaßen einteilen:

* Die Benutzerschnittstelle betreffend
* nicht autorisierter Zugriff
* bösartige Eingaben
* das Dateisystem betreffend
* Zugriff auf vertrauliche Daten in Dateien oder Repositories
* das Betriebssystem betreffend
* unverschlüsseltes Ablegen von sicherheitskritischen Informationen, wie Passwörtern. Wird das System durch bösartige Eingaben zum Absturz gebracht, können die Informationen zugänglich werden.
* Externe Software betreffend
* Interaktionen zwischen externen Komponenten, die das System nutzt. Sie können auf Netzwerkebene stattfinden (wenn beispielsweise inkorrekte Datenpakete oder Meldungen übertragen werden) oder auf Ebene der Softwarekomponenten (wenn beispielsweise eine Softwarekomponente ausfällt, die vom System benötigt wird).

Auf der Suche nach potentiellen Schwachstellen berücksichtigt man insbesondere:

* Alle Schnittstellen zwischen Teilsystemen
* Mögliche Ansatzpunkte für parallele Attacken
* Konsequenzen durch Ausfälle von Teilsystemen

Die Ableitung von Sicherheitstests erfolgt in drei Schritten:

1. Informationssammlung
2. Anfälligkeitsuntersuchrung
3. Planen von Testaktionen, die darauf abzielen, den Sicherheitsgrundsatz des Systems zu durchdringen (Softwareattacken)

**Sicherheitstests** müssen bei Wartungstests berücksichtigt werden, wenn Teile des Multi-Systems geändert oder ersetzt werden.

**Steckbrief: Sicherheitstest**

**Zeitpunkt:** Systemtest, Wartungstest

**Typische Techniken:**

* Syntaxtest
* Statische Analyse
* Softwareattacken

**Fehler, die gefunden werden:**

* Anfälligkeit eines Systems gegenüber unterschiedlichen Bedrohungen

Zuverlässigkeit

**Zuverlässigkeit** bezeichnet eine Menge von Merkmalen, die sich auf die Fähigkeit der Software beziehen, ihr Leistungsniveau unter festgelegten Bedingungen über einen festgelegten Zeitraum oder über eine festgelegte Anzahl von Transaktionen zu bewahren.

Teilmerkmale der Zuverlässigkeit sind.

* **Reife** - Die Fähigkeit eines Software-Produkts Fehlerwirkungen auf Grundlage von Fehlerzuständen zu vermeiden.
* **Fehlertoleranz (Robustheit)** - Die Fähigkeit eines Software-Produkts, ein spezifiziertes Leistungsniveau auch bei Fehlfunktion oder trotz Fehleingeben aufrecht zu erhalten.
* **Wiederherstellbarkeit** - Die Fähigkeit eines Software-Produkts, bei einem Versagen das spezifizierte Leistungsniveau des Systems wiederherzustellen und die direkt betroffenen Daten wiederzugewinnen.

Um den Grad der Zuverlässigkeit messen zu können, müssen diverse unterschiedliche Testverfahren angewendet werden:

* **Robustheitstests:** Getestet wird die Toleranz eines Softwaresystems bezüglich der Probleme, die mit äußeren Einflüssen bzw. der Umgebung zusammenhängen, in der das System läuft. Beispielsweise
  + Fehlermeldungen des Betriebssystems (vollgeschriebene Festplatte, nicht genügend Speicher vorhanden, Datei nicht gefunden, Dienst nicht gestartet, etc.)
  + Von Sensoren gelieferte Ergebnisse wie Temperaturen, Geschwindigkeiten, Entfernungen
  + Fehler an Daten, die bei Datenübertragungen entstehen können
* **Wiederherstellbarkeitstests:** Das Softwaresystem muss nach einem Hardware- oder Software-Fehler in einer vorbestimmten Art und Wiese wieder hergestellt werden können. Im Anschluss muss der normale Betrieb wieder aufgenommen werden können. Um dies zu gewährleisten müssen entsprechende Maßnahmen eingesetzt werden wie beispielsweise
  + **Failover:** Die Fähigkeit eines Systems, bei Ausfall automatisch auf ein Reserve-System umzuschalten.
  + **Backup & Restore:** Daten- Sicherungen verringern die durch Ausfälle oder Fehlbedienung entstehenden Schäden, da die Möglichkeit besteht die Daten ganz oder teilweise wieder herzustellen.

Es ergeben sich somit nachfolgende Testziele bei den **Failover-** und **Backup & Restoretests:**

* Testfälle müssen so spezifiziert werden, dass kritische Pfade im Betriebsablauf überdeckt werden
* Failovertests überprüfen die durch den Einsatz redundanter Systeme erreichte Zuverlässigkeit
* Backup & Restoretests evaluieren die Erfolgsaussichten vorgesehener Maßnahmen zur Wiederherstellung von Daten im Falle des Datenverlusts oder der Datenkorruption
* Testfälle können z.B. mit Hilfe der Ereignisbaum-Methode oder der Fehlerbaum-Methode entworfen werden

**Steckbrief: Zuverlässigkeitstests**

**Zeitpunkt:** Systemtest, Akzeptanztest, Wartungstest

**Typische Techniken:**

* SRET, Nutzungsprofil-orientiertes Testen
* Ereignis- und Fehlerbaum-Methode
* Negativtests

**Fehler, die gefunden werden:**

* Fehler in der Funktionalität der Software deren Auftreten profil-abhängig ist
* Laufzeitfehler (z.B. Speicherlecks)
* Fehler bei der Sicherung und Wiederherstellung von Daten
* Nicht abgefangene Fehleingaben

Effizienz

Das Qualitätsmerkmal **Effizienz** wird mit Tests zu Zeit- und Ressourcenverhalten bewertet. Die

folgenden Abschnitte behandeln Effizienztests bezogen auf das Zeitverhalten unter den Aspekten

**Performanz-, Last-, Stress- und Skalierbarkeitstests**.

* **Performanztests:** Es gibt unterschiedliche Arten von Performanztests, je nach betrachteten nicht-funktionalen Anforderungen. Zu diesen Testarten gehören Performanz-, Last-, Stress- und Skalierbarkeitstests.

Performanztests untersuchen die Fähigkeit einer Kompone oder eines Systems, auf Eingaben des Benutzers oder Systems innerhalb einer definierten Zeit und unter den spezifizierten Bedingungen zu reagieren. Dabei variieren die Performanzmessungen je nach Zielsetzung des Tests.

* so kann die Performanzmessung einer einzelnen Softwarekomponente die CPU-Zyklen berechnen
* bei Client-basierten Systemen lässt sich dagegen die Zeit messen, die das System benötigt, um auf eine bestimmte Nutzeranfrage zu reagieren
* bei Systemarchitekturen, die aus mehreren Komponenten bestehen (beispielsweise Clients, Servern, Datenbanken) wird die Performanz zwischen den einzelnen Systemkomponenten gemessen, um Engpässe zu identifizieren
* **Lasttests:** Lasttests messen die Fähigkeit eines Systems, ansteigende Grade erwarteter, realistischer Systemlasten zu bewältigen, die eine Anzahl paralleler Benutzer als Transaktionsanforderungen generiert. Die durchschnittlichen Antwortzeiten der Benutzer werden in typischen Nutzungsszenarien gemessen und analysiert.

Es gibt zwei Untergruppen von Lasttests:

* solche mit einer realistischen Nutzeranzahl
* und solche, wo ein hohes Datenvolumen im Vordergrund steht

Lasttests messen sowohl die Antwortzeiten als auch Netzwerkdurchsatz.

* **Stresstests:** Stresstests untersuchen die Fähigkeit eines Systems, Spitzenlasten an Kapazitätsgrenzen zu bewältigen. Mit steigender Überbelastung sollte die Systemleistung allmählich, vorhersehbar und ohne Ausfall abnehmen. Vor allem sollte die funktionale Integrität des Systems unter Spitzenlast getestet werden, um mögliche Fehlerzustände bei der funktionalen Verarbeitung oder Dateninkonsistenzen aufzudecken.

Mögliches Ziel von Stresstests kann sein,

* die Grenze zu erreichen, an der das System tatsächlich ausfällt, um so das schwächste Glied in der Kette zu identifizieren
* In Spitzentests (spike tests) werden Bedingungen simuliert, die zu plötzlichen extremen Systemlasten führen.
* Bei Bounce-Tests wechseln solche Spitzenlasten mit Zeiten geringer Nutzung ab. Die Tests untersuchen, wie gut das System mit diesen Laständerungen fertig wird, und ob es nach Bedarf Ressourcen in Anspruch nimmt und wieder freigibt.

Bei Stresstests ist es erlaubt, dem System rechtzeitig zusätzliche Komponenten hinzuzufügen (beispielsweise Speicher, CPU, Datenbankspeicher).

* **Skalierbarkeitstests:** Skalierbarkeitstests untersuchen die Fähigkeit eines Systems, zukünftige Effizienzanforderungen zu erfüllen, die über den gegenwärtigen liegen können. Ziel dieses Tests ist es zu beurteilen, ob das System wachsen kann (beispielsweise mit mehr Benutzern oder größeren Mengen von gespeicherten Daten), ohne die vereinbarten Grenzen zu überschreiten oder zu versagen. Sind diese Grenzen bekannt, lassen sich Schwellenwerte definieren und in der Produktion überwachen, sodass bei bevorstehenden Problemen eine Warnung erfolgen kann.
* **Prüfung der Ressourcennutzung:** Effizienztests der Ressourcennutzung bewerten die Verwendung von Systemressourcen (beispielsweise Hauptspeicher- und Festplattenkapazitäten, Bandbreiten im Netz). Die Verwendung dieser Ressourcen wird sowohl bei normaler Systemauslastung gemessen als auch in Stresssituationen durch beispielsweise hohe Transaktionsraten oder große Datenmengen.
* **Spezifikationen von Effizientests:**

Testspezifikationender Testarten **Performanz-, Last- und Stresstests** basieren auf definierten

Nutzungsprofilen mit verschiedenen Formen des Nutzungsverhaltens bei der Interaktion mit der

Anwendung. Für eine Anwendung kann es mehrere Nutzungsprofile geben.

Die Anzahl der Nutzer pro Nutzungsprofil lässt sich mit Monitorwerkzeugen, oder durch eine

Voraussage. Voraussagen können auf Algorithmen basieren oder von der Betriebsorganisation

stammen, sie sind besonders wichtig für die Spezifizierung der Nutzungsprofile bei Skalierbarkeitstests. Nutzungsprofile sind Grundlage der Testfälle und werden in der Regel mit Testwerkzeugen generiert.

**Steckbrief: Effizienztest**

**Zeitpunkt:** während des System- oder Regressionstests

**Typische Techniken:**

* Performanztests
* Lasttests
* Stresstests

**Fehler, die gefunden werden:**

* Ungenügende nichtfunktionale Eigenschaften des Zeit- und Ressoucenverhaltens

Wartbarkeit

Wartbarkeitstests untersuchen, wie einfach die Software analysiert, geändert und getestet werden kann. Geeignete Verfahren für Wartbarkeitstests sind statische Analysen und Checklisten.

* **Dynamische Wartbarkeitstests:** Dynamische Wartbarkeitstests untersuchen vorrangig die dokumentierten Verfahren, die für die Wartung einer Anwendung entwickelt wurden (beispielsweise für ein Update der Software. Als Testfälle dienen Wartungsszenarien, um sicherzustellen, dass die geforderten Service Levels mit den dokumentierten Verfahren erreicht werden können.

Diese Art des Testens ist besonders wichtig bei komplexen Infrastrukturen, bei denen mehrere Abteilungen/Organisationen an den Supportverfahren beteiligt sind. Die Tests können Teil der betrieblichen Abnahmetests sein.

* **Analysierbarkeit (korrektive Wartung):** Bei dieser Art von Wartbarkeitstests wird die Zeit gemessen, die für Diagnose und Behebung von im System erkannten Problemen nötig ist. Eine einfache Metrik kann der durchschnittliche Zeitaufwand für Diagnose und Korrektur des Fehlers sein.
* **Änderbarkeit, Stabilität und Testbarkeit (adaptive Wartung):** Die Wartbarkeit eines Systems lässt sich auch auf den Aufwand bezogen messen, der für Änderungen nötig ist (beispielsweise Programmänderungen). Da der benötigte Aufwand von mehreren Faktoren abhängt, wie der verwendeten Softwaredesign- Methodik (beispielsweise objektorientiert), von Programmierstandards usw., sind auch Analysen oder Reviews für diese Wartbarkeitstests geeignet. Die Testbarkeit hängt in erster Linie vom Aufwand für das Testen der Änderungen ab. Die Stabilität hängt in erster Linie von der Reaktion des Systems auf Änderungen ab. Systeme mit niedriger Stabilität zeigen nach jeder Änderung eine große Anzahl an Folgeproblemen (ripple effect).

**Steckbrief: Wartbarkeitstest**

**Zeitpunkt:** Früh (während Analyse/Design bzw. nach der Codierung)

**Typische Techniken:**

* Statische Analyse
* Checklisten
* Reviews

**Fehler, die gefunden werden:**

* Mangelende Wartbarkeit
* Mangelnde Strukturierung oder Modularisierung der Software
* Nicheinhaltung von Programmierrichtlinien
* Fehlende oder unvollständige Dokumentation

5.8 Portabilität

Portabilitätstests untersuchen, wie einfach es ist, eine Software in ihre vorgesehene Umgebung zu übertragen, entweder bei der Erstinstallation oder aus einer bestehenden Umgebung. Bei Portabilitätstests werden Installierbarkeit, Koexistenz/Kompatibilität, Anpassbarkeit und Austauschbarkeit getestet.

* **Installationstests:** Installationstests testen Software für die Software-Installation in einer Zielumgebung. Das kann eine Software sein, mit der ein Betriebssystem auf einem Prozessor installiert wird, oder ein Installationsprogramm (Wizard), mit dem ein Produkt auf einem Client-PC verwendet wird. Ziele von Installationstests sind
* Validieren, dass die Software erfolgreich installiert werden kann, indem man die Anweisungen des Installationshandbuchs befolgt (einschließlich der Ausführung von Installationsskripten) oder einen Installationswizard nutzt. Dabei sind auch die unterschiedlichen Optionen für mögliche HW-/SW-Konfigurationen sowie für verschiedene Installationsstufen (ganz oder teilweise) zu testen.
* Testen, ob die Installationssoftware richtig mit Fehlerwirkungen umgeht, die bei der Installation auftreten (beispielsweise DLLs nicht geladen werden), ohne das System in einem undefinierten Zustand zu lassen (beispielsweise mit nur teilweise installierter Software oder inkorrekten Systemkonfigurationen).
* Testen, ob sich eine nur teilweise Installation/Deinstallation der Software abschließen lässt.
* Testen, ob ein Installationswizard eine ungültige Hardware-Plattform oder Betriebssystem-Konfigurationen erfolgreich identifizieren kann.
* Messen, ob der Installationsvorgang im spezifizierten Zeitrahmen oder mit weniger als der spezifizierten Anzahl von Schritten abgeschlossen werden kann.
* Validieren, dass sich eine frühere Version installieren oder die Software sich deinstallieren lässt.

Nach dem Installationstest wird normalerweise die Funktionalität getestet, um Fehler aufzudecken, die durch die Installation eingeschleust worden sein können (beispielsweise inkorrekte Konfigurationen, nicht mehr verfügbare Funktionen). Parallel zu den Installationstests laufen üblicherweise Benutzbarkeitstest (beispielsweise zur Validierung, dass die Anwender bei der Installation verständliche Anweisungen und Fehler-/Meldungen erhalten.

* **Koexistenz:** Computersysteme, die nicht miteinander interagieren, sind dann kompatibel, wenn sie in derselben Umgebung (beispielsweise auf derselben Hardware) laufen können, ohne sich gegenseitig zu beeinflussen (beispielsweise durch Ressourcenkonflikte). Kompatibilitätstests können dann durchgeführt werden, wenn eine neue Software oder eine Aktualisierung in einer neuen Umgebung getestet wurde und Inkompatibilitäten nicht erkennbar waren, wenn diese Anwendung dann in einer anderen, beispielsweise der Produktionsumgebung, installiert wird, in der bereits andere Anwendungen installiert sind.

Typische Ziele von Kompatibilitätstests sind

* Bewertung möglicher negativer Auswirkungen auf die Funktionalität, wenn Anwendungen in derselben Umgebung geladen werden (beispielsweise Konflikte in der Ressourcennutzung, wenn mehrere Anwendungen auf demselben Server laufen),
* Bewertung der Auswirkungen auf jede Anwendung, die sich aus Modifikationen oder dem Installieren einer aktuelleren Betriebssystemversion ergeben.
* Kompatibilitätstests werden normalerweise nach dem erfolgreichen Abschluss der System und Anwenderabnahmetests durchgeführt.
* **Anpassbarkeitstests:** Anpassbarkeitstests sollen zeigen, ob eine bestimmte Anwendung in allen geplanten Zielumgebungen korrekt funktioniert (Hardware, Software, Middleware, Betriebssystem usw.) Für die Spezifikation der Anpassbarkeitstests müssen Kombinationen der beabsichtigten Zielumgebungen identifiziert, konfiguriert und dem Testteam zur Verfügung gestellt werden. Diese Umgebungen werden dann mit ausgewählten funktionalen Testfällen getestet, bei denen die verschiedenen Komponenten der Testumgebung geprüft werden.

Anpassbarkeit kann sich darauf beziehen, dass sich die Software durch Ausführung eines definierten Verfahrens auf verschiedene spezifizierte Umgebungen portieren lässt. Beim Testen kann das Verfahren bewertet werden.

Anpassbarkeitstests können zusammen mit Installierbarkeitstests durchgeführt werden. Normalerweise finden anschließend funktionale Tests statt, um Fehler aufzudecken, die durch das Anpassen der Software an die andere Umgebung entstanden sein können

.

* **Austauschbarkeitstests:** Austauschbarkeit drückt aus, ob sich Softwarekomponenten eines Systems gegen andere Komponenten austauschen lassen. Das ist besonders relevant bei Systemen, die kommerzielle Standardsoftwareprodukte für bestimmte Softwarekomponenten verwenden.

Austauschbarkeitstests können parallel zu den funktionalen Integrationstests durchgeführt werden, wenn mehr als eine Komponente alternativ für die Integration in das Gesamtsystem verfügbar ist. Die Austauschbarkeit lässt sich in technischen Reviews oder Inspektionen bewerten, bei denen Gewicht auf eine klare Definition der Schnittstellen zu möglichen Austauschkomponenten gelegt wird.

**Steckbrief: Portabilitätstest**

**Zeitpunkt:** Systemtest

**Typische Techniken:**

* Testentwurf, v.a. Black-Box

**Fehler, die gefunden werden:**

* Funktionale Fehler in der Installationssoftware
* Nichtfunktionale Fehler in der Installationssoftware (mangelnde Robustheit, Effizienz und Benutzbarkeit)

# Reviews

Anders als das dynamische Testen, welches die Ausführung der Software voraussetzt, beruhen **statische Prüftechniken** auf der „manuellen“ Überprüfung (Reviews) oder automatisierten Analysen (statische Analyse) des Codes oder anderer Projektdokumentation, ohne den Programmcode auszuführen.

**Reviews** sind eine Möglichkeit, Arbeitsergebnisse der Softwareentwicklung (einschließlich Code) zu

prüfen und können problemlos bereits lange vor der dynamischen Testdurchführung durchgeführt

werden. Fehlerzustände, die durch Reviews in den frühen Phasen des Softwarelebenszyklus entdeckt werden (z.B. Fehlerzustände in den Anforderungen), sind häufig bedeutend kostengünstiger zu beheben, als solche, die erst während der Testdurchführung bei Ausführung des Programmcodes gefunden werden.

Ein Review kann komplett als manuelle Aktivität durchgeführt, aber ebenso durch Werkzeuge unterstützt werden. Die wichtigste manuelle Tätigkeit ist die Prüfung und Kommentierung des Arbeitsergebnisses. **Jedes Arbeitsergebnis** der Softwareentwicklung kann einem Review unterzogen werden, einschließlich:

* Anforderungsspezifikationen,
* Designspezifikationen,
* Quellcode
* Testkonzepte
* Testspezifikationen
* Testfälle
* Testskripte
* Anwenderhandbücher
* Web-Seiten

**Vorteile** von Reviews sind:

* frühe Aufdeckung und Korrektur von Fehlerzuständen
* Verbesserung der Softwareentwicklungsproduktivität
* reduzierte Entwicklungsdauer
* reduzierte Testkosten und -dauer
* Reduzierung der Kosten während der Lebensdauer
* weniger Fehlerzustände und verbesserte Kommunikation.

Reviews können beispielsweise Auslassungen in Anforderungen aufdecken (z.B. fehlende Funktionen), die durch einen dynamischen Test vermutlich nicht gefunden würden.

Reviews, statische Analyse und dynamischer Test haben das gleiche Ziel, nämlich Fehlerzustände zu identifizieren. Sie ergänzen sich: Die verschiedenen Methoden können verschiedene Arten von

Fehlern wirksam und effizient aufdecken. Verglichen mit dem dynamischen Test finden statische Prüftechniken eher Ursachen der Fehlerwirkungen (Fehlerzustände) als Fehlerwirkungen selbst.

Zu den **typische Fehlerzuständen**, die effektiver und effizienter durch Reviews als durch dynamische Tests zu finden sind, gehören:

* Abweichungen von Standards,
* Fehlerzustände in Anforderungen,
* Fehlerzustände im Design,
* unzureichende Wartbarkeit,
* fehlerhafte Schnittstellenspezifikationen.

## Reviewprozess

Die verschiedenen Arten von Reviews variieren zwischen **informell**, charakterisiert durch fehlende

schriftliche Vorgaben für die Gutachter, und **systematisch**, charakterisiert durch die Einbindung eines Reviewteams, dokumentierte Reviewergebnisse und dokumentierte Abläufe zur Durchführung von Reviews.

Der **Formalismus eines Reviewprozesses** ist abhängig von Faktoren wie

* Reife des Entwicklungsprozesses
* gesetzlichen oder regulatorischen Anforderungen
* oder der Notwendigkeit eines Prüfnachweises

Die **Art und Weise, wie ein Review durchgeführt wird**, ist abhängig von den festgelegten Zielen des Reviews, beispielsweise

* Finden von Fehlerzuständen
* Erwerben von Verständnis
* Ausbildung von Testern und neuen Teammitgliedern
* oder einer Diskussion und Entscheidungsfindung im Konsens

**Aktivitäten eines formalen Reviews:**

Ein typisches formales Review besteht aus folgenden Hauptaktivitäten:

**1. Planen**

* Festlegen von Review-/Prüfkriterien
* Auswahl der beteiligten Personen
* Besetzen der Rollen
* Festlegen der Eingangs- und Ausgangskriterien bei mehr formalen Reviewarten (z.B. Inspektion)
* Auswahl der zu prüfenden Dokumententeile
* Prüfen der Eingangskriterien (bei formaleren Reviewarten)

**2. Kick-off**

* Verteilen der Dokumente
* Erläutern der Ziele, des Prozesses und der Dokumente den Teilnehmern gegenüber

**3. Individuelle Vorbereitung**

* Vorbereiten der Reviewsitzung durch Prüfung des/der Dokuments/e
* Notieren von potenziellen Fehlerzuständen, Fragen und Kommentaren

**4. Prüfen/Bewerten/Festhalten der Ergebnisse (Reviewsitzung)**

* Diskussion oder Protokollierung, mit dokumentierten Ergebnissen oder Protokollen (bei
* formaleren Reviewarten)
* Festhalten von Fehlerzuständen, Empfehlungen zum Umgang mit ihnen oder Entscheidungen über die Fehlerzustände
* Prüfen/Bewerten und Protokollieren von Problemen während eines physischen Treffens
* oder Nachverfolgen von elektronischer Gruppenkommunikation

**5. Überarbeiten**

* Beheben der gefundenen Fehlerzustände typischerweise durch den Autor
* Protokollieren des aktualisierten Fehlerstatus (in formalen Reviews)

**6. Nachbereiten**

* Prüfen, ob Fehlerzustände zugewiesen wurden
* Sammeln von Metriken
* Prüfen von Ausgangskriterien (bei formaleren Reviewarten)

Rollen und Verantwortlichkeiten

Bei einem typischen formalen Review findet man folgende Rollen:

* **Manager:** Die Person, die über die Durchführung von Reviews entscheidet, Zeit im Projektplan zur Verfügung stellt und überprüft, ob die Reviewziele erfüllt sind.
* **Moderator:** Die Person, die das Review eines Dokuments bzw. von einigen zusammengehörenden Dokumenten leitet, einschließlich der Reviewplanung, der Leitung der Sitzung und der Nachbereitung nach der Sitzung. Falls nötig, kann der Moderator zwischen den verschiedenen Standpunkten vermitteln. Er ist häufig die Person, von der der Erfolg des Reviews abhängt.
* **Autor:** Der Verfasser oder die Person, die für das/die zu prüfende/n Dokument/e hauptverantwortlich ist.
* **Gutachter:** Personen mit einem spezifischen technischen oder fachlichen Hintergrund (auch

Prüfer oder Inspektoren genannt), die nach der nötigen Vorbereitung im Prüfobjekt Befunde

identifizieren und beschreiben (z.B. Fehlerzustände). Gutachter sollten so gewählt werden,

dass verschiedene Sichten und Rollen im Reviewprozess vertreten sind. Sie sollten an allen

Reviewsitzungen teilnehmen können.

* **Protokollant:** Die Person, die alle Ergebnisse, Probleme und offenen Punkte dokumentiert,

die im Verlauf der Sitzung identifiziert werden.

Softwareprodukte oder darauf bezogene Arbeitsergebnisse aus verschiedenen Perspektiven zu betrachten und Checklisten zu nutzen, kann Reviews wirksamer und effizienter machen. So kann eine Checkliste helfen, bisher unentdeckte Probleme aufzudecken, wenn sie typische Anforderungsprobleme enthält und unterschiedliche Perspektiven einnimmt, beispielsweise vom Benutzer, Wartungspersonal, Tester oder Operator.

Reviewarten

Ein einzelnes Softwareprodukt oder ein darauf bezogenes Arbeitsergebnis kann Gegenstand von

mehr als einem Review sein. Falls mehr als nur eine Reviewart eingesetzt wird, kann die Reihenfolge variieren. Ein informelles Review beispielsweise könnte vor einem technischen Review durchgeführt werden oder eine Inspektion einer Anforderungsspezifikation kann vor einem Walkthrough mit Kunden durchgeführt werden.

Die Hauptcharakteristika, optionale Bestandteile und Zwecke allgemeiner **Reviewarten** sind:

* **Informelles Review**
* kein formaler Prozess
* kann in Form des Programmierens in Paaren (pair programming) durchgeführt werden oder

ein technischer Experte unterzieht Entwurf und Quellcode einem Review

* Ergebnisse können dokumentiert werden
* Nutzen variiert abhängig von den Gutachtern
* Hauptzweck: Günstiger Weg eine Verbesserung zu erreichen
* **Walkthrough**
* Sitzung geleitet durch den Autor
* kann in Form von Szenarien, Probeläufen oder im Kreis gleichgestellter Mitarbeiter (Peer Review) stattfinden
* Open-End-Sitzungen
* wahlweise der Sitzung vorausgehende Vorbereitung der Gutachter
* wahlweise Vorbereitung eines Reviewberichts, der eine Liste der Befunde enthält
* wahlweise Protokollant (der aber nicht der Autor ist)
* kann in der Praxis von informell bis sehr formal variieren
* Hauptzweck: Lernen, Verständnis erzielen, Fehlerzustände finden
* **Technisches Review**
* dokumentierter und definierter Fehlerfindungsprozess, der gleichgestellte Mitarbeiter und

technische Experten sowie optional Personen aus dem Management einschließt

* kann als Peer Review ohne Teilnahme des Managements ausgeführt werden
* idealerweise durch einen geschulten Moderator geleitet (nicht der Autor)
* Vorbereitung vor der Sitzung durch Gutachter
* wahlweise Nutzung von Checklisten
* Vorbereitung eines Reviewberichts, der folgende Punkte enthält:
* die Liste der Befunde
* eine Gesamtbewertung, inwieweit das Softwareprodukt die Anforderungen erfüllt,
* und Empfehlungen in Bezug auf die Befunde, wo angebracht
* kann in der Praxis von informell bis sehr formal variieren
* Hauptzweck: Diskussion, Entscheidungen treffen, Alternativen bewerten, Fehlerzustände finden, technische Probleme lösen und prüfen, ob Übereinstimmung mit Spezifikationen, Plänen, Bestimmungen und Standards existiert
* **Inspektion**
* geleitet durch einen geschulten Moderator (nicht der Autor)
* gewöhnlich durchgeführt als Prüfung durch gleichgestellte Mitarbeiter
* definierte Rollen
* schließt das Sammeln von Metriken ein
* formaler Prozess basierend auf Regeln und Checklisten
* spezifizierte Eingangs- und Ausgangskriterien für die Abnahme des Softwareprodukts
* Vorbereitung vor der Sitzung
* Inspektionsbericht mit Liste der Befunde
* formaler Prozess für Folgeaktivitäten (optional mit Komponenten zur Prozessverbesserung)
* wahlweise Vorleser
* Hauptzweck: Fehlerzustände finden
* **Management-Review**
* wird durchgeführt von Managern bzw. für Manager mit direkter Verantwortung für das Projekt oder System
* außerdem wird es durchgeführt von bzw. für einen Betroffenen oder Entscheidungsträger, beispielsweise gehobenes Management oder Direktor
* es wird geprüft, ob Pläne konsistent eingehalten werden oder es Abweichungen gibt, und es wird festgestellt, ob die installierten Managementverfahren adäquat sind
* enthält die Bewertung von Projektrisiken Ergebnis nennt die durchzuführenden Maßnahmen und zu lösenden Probleme
* es wird erwartet, dass sich die Teilnehmer auf die Sitzungen vorbereiten
* Entscheidungen werden dokumentiert
* Hauptzweck: Fortschritt überwachen, Status beurteilen, Entscheidungen über zukünftige Maßnahmen treffen
* **Audit**
  + der Leiter des Audits ist für das Audit verantwortlich und übernimmt die Moderation
* die Prüfer sammeln Beweise für die Konformität durch Interviews, Beobachtungen und die

Prüfung von Dokumenten.

* + zu den Ergebnissen des Audits gehören Beobachtungen, Empfehlungen, Abhilfemaßnahmen

und eine abschließende Beurteilung (bestanden/nicht bestanden).

* Hauptzweck: unabhängige Bewertung der Konformität mit bestimmten Verfahren, Vorschriften, Standards usw.

Walkthroughs, technische Reviews und Inspektionen können in einer Gruppe von gleichgestellten

Kollegen (Peers), d.h. Kollegen aus der gleichen organisatorischen Ebene, durchgeführt werden. Diese Art von Review wird auch „Peer Review“ genannt.

Testmanager sollten an den Management-Reviews teilnehmen bzw. Management-Reviews zum Testfortschritt ansetzen.

Audits sind sehr formal und werden in der Regel dann durchgeführt, wenn Konformität mit bestimmten Erwartungen nachgewiesen werden soll, beispielsweise Konformität mit einem geltenden Standard oder einer vertraglichen Verpflichtung.

Audits sind daher die am wenigsten effektive Methode zum Aufdecken von Fehlerzuständen.

Erfolgsfaktoren für Reviews

Erfolgsfaktoren für Reviews beinhalten:

* Jedes Review hat klar definierte Ziele.
* Abhängig von den Reviewzielen werden geeignete Personen ausgewählt.
* Tester sind geschätzte Gutachter, die einen Beitrag zum Review leisten. Weiters lernen sie

auch das Produkt kennen, was sie befähigt Tests früher vorzubereiten.

* Gefundene Fehlerzustände werden positiv aufgenommen und werden objektiv zur Sprache

gebracht.

* Menschliche und psychologische Aspekte werden beachtet, um das Review beispielsweise als eine positive Erfahrung für den Autor zu gestalten.
* Das Review wird in einer Atmosphäre des Vertrauens durchgeführt, das Ergebnis dient nicht

zur Beurteilung der Teilnehmer.

* Es werden die Reviewtechniken angewendet, die zur Erreichung der Reviewziele, für Art und

Stufe von Arbeitsergebnissen der Softwareentwicklung und für die Gutachter geeignet sind.

* Wenn sie geeignet sind, die Effektivität der Fehleridentifikation zu steigern, werden Checklisten oder Rollen verwendet.
* Es finden Schulungen in Reviewtechniken statt, besonders für die formaleren Methoden wie

Inspektionen.

* Das Management unterstützt einen guten Reviewprozess, indem es beispielsweise angemessene Zeit für Reviewaktivitäten im Projektplan einräumt.
* Es liegt eine Betonung auf Lernen und Prozessverbesserung.

**Quellen:**

* **ISTQB\_/GTB Standardglossar der Testbegriffe**

**Deutsch/Englisch**

**Herausgeber: German Testing Board e.V.**

**Dr. Matthias Hamburg, Dr. Uwe Hehn**

[**http://www.software-tester.ch/PDF-Files/CT\_Glossar\_DE\_EN\_V21.pdf**](http://www.software-tester.ch/PDF-Files/CT_Glossar_DE_EN_V21.pdf)

* **ISTQB Lehrplan**
* **Andreas Spillner/Tilo Linz  
  Basiswissen Softwaretest  
  Aus- und Weiterbildung zum Certified Tester   
  Foundation Level nach ISTQB-Standard**
* **Graham Bath / Judy McKay**

**Praxiswissen Softwaretest – Test Analyst und Technical Test Analyst**

**Aus- und Weiterbildung zum Certified Tester   
Advanced Level nach ISTQB-Standard**

* **Andreas Spillner/Thomas Roßner/Mario Winter/Tilo Linz**

**Praxiswissen Softwaretest – Testmanagement**

**Aus- und Weiterbildung zum Certified Tester   
Advanced Level nach ISTQB-Standard**

* **Wikipedia**

[**http://de.wikipedia.org**](http://de.wikipedia.org)

[**http://de.wikiversity.org/wiki/Kurs:Software-Test**](http://de.wikiversity.org/wiki/Kurs:Software-Test)