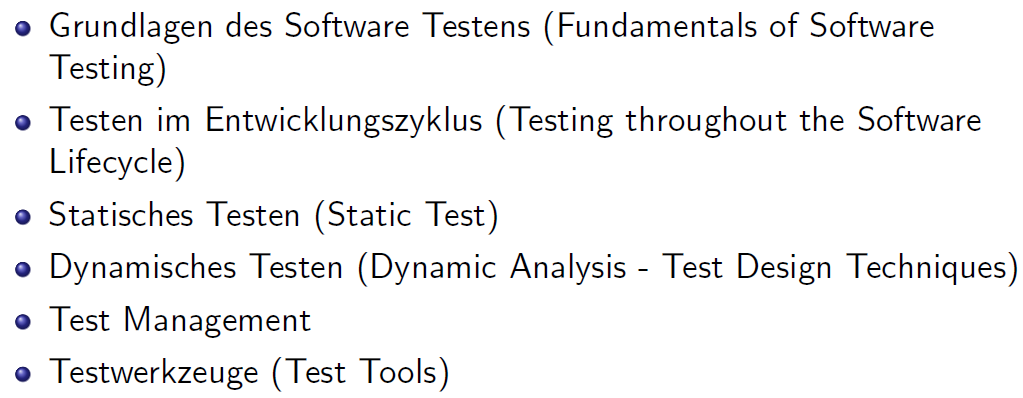
Zusammenfassung Certified Tester

# Übersicht



# Grundlagen des Software Testens

**Anomalie**: Jede Abweichung von der Spezifikation

**Fehler** (1) / **Problem** / **Bug**: Problem im System/Komponente, das die Funktion beeinträchtigt

**Fehler** (2): Jedes Ereignis, das beim Testen auftritt und eine Untersuchung erfordert

**Abweichung**: Unterschied zwischen charakteristischem Wert und dem entsprechenden Referenzwert / Lieferwert / Service

**Debugging**: Lokalisierung und Behebung eines Fehlers

**Testen**: Ausführung eines Testobjektes zu seiner Untersuchung

**Annahmen zur Korrektheit**:

* Kein komplexes System ist frei von Fehlern
* Vollständige Korrektheit kann durch Tests nicht erreicht werden

**Nutzungsqualität**:

* Effektivität, Effizienz, Nutzungszufriedenheit, Risikofreiheit, Kontextabdeckung

**Produktqualität**:

* Funktionelle Eignung, Leistungseffizienz, Kompatibilität, Benutzungsfreundlichkeit, Zuverlässigkeit, Sicherheit, Wartbarkeit, Portabilität

**Testaufwand**

* „Wie viele Tests sind nötig?“
* Daumenregeln:
  + Testaufwand zwischen 25% und 50% des Gesamtaufwands
  + Je kritischer das Produkt, desto höher der erwartete Testaufwand
* Der **Testmanager** ist verantwortlich für die geplante, effiziente Teststrategie

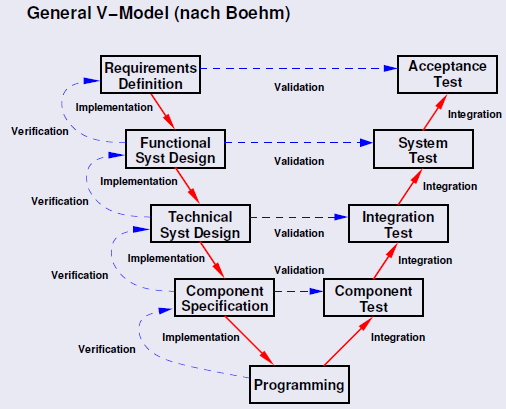
**Ursachenkette für Fehler**

* Fehlerwirkung (failure): Nach Außen sichtbar, Abweichungen von IST- und SOLL-Werten
* Fehlerzustand (defect): Mangel in einem Ergebnis, Nichterfüllen einer Spezifikation
* Fehlhandlung (error): Menschliche Handlung, die zu falschem Ergebnis führt
* Fehlermaskierung:

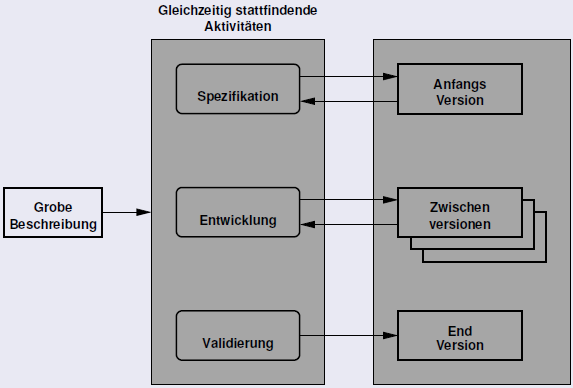
**Validierung und Verifizierung**

* Validierung: „Haben wir das **richtige** System realisiert?“, Nachweis auf korrekte Erfüllung von Anforderungen für einen **spezifischen** Gebrauch
* Verifizierung: „Haben wir das System **richtig** realisiert?“, Nachweis auf korrekte Erfüllung **festgelegter** Anforderungen

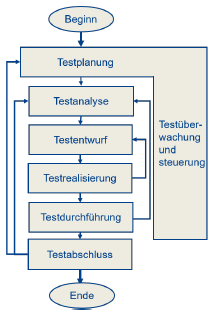
**Generelles V-Modell**



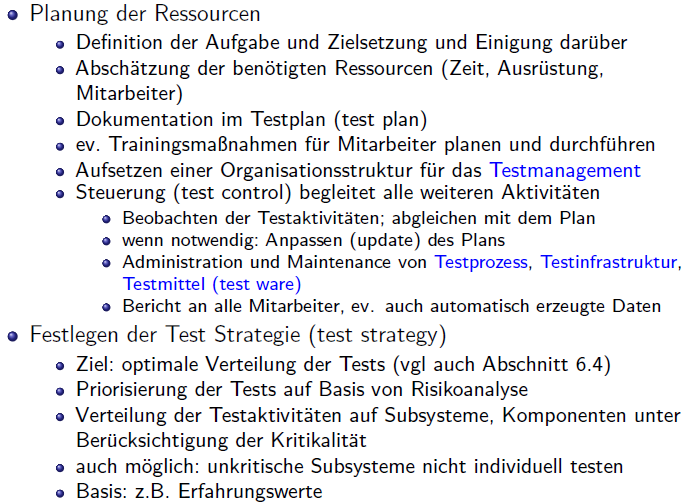
**Inkrementelle Software-Einheit**

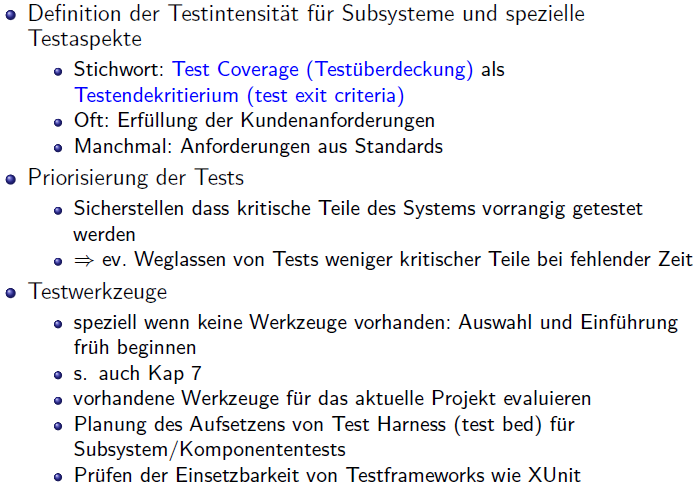


**Fundamentaler Testprozess**

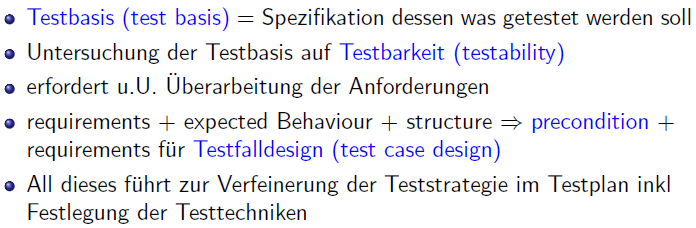


**Testplanung**

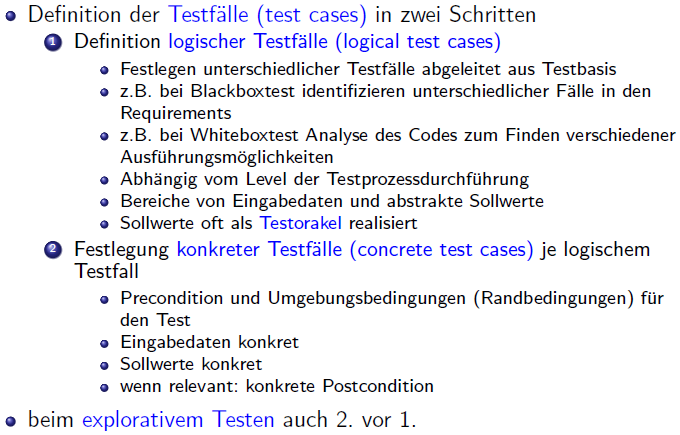




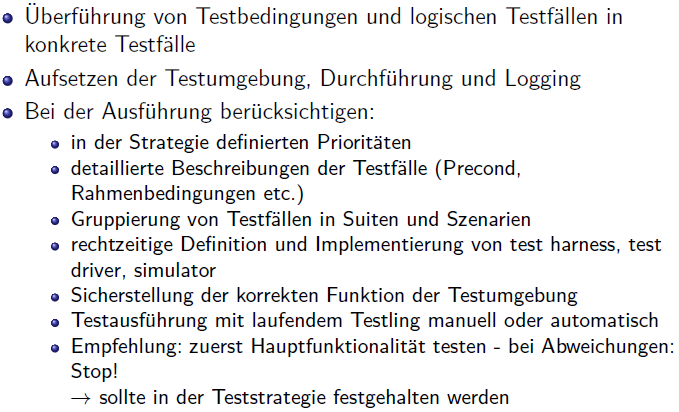
**Testanalyse**



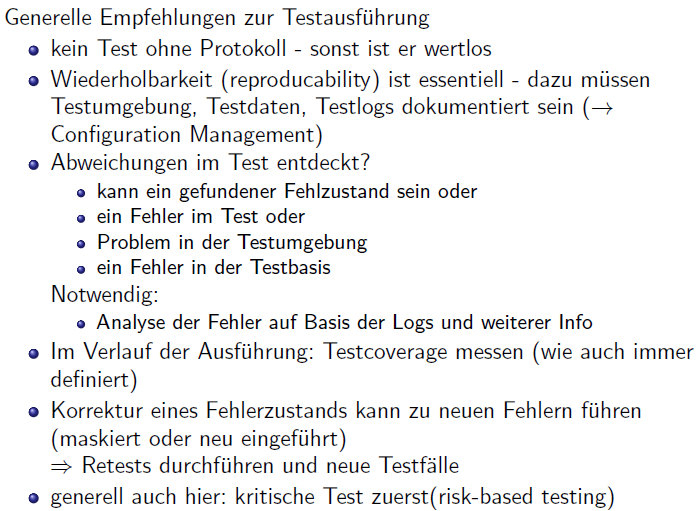
**Testentwurf**



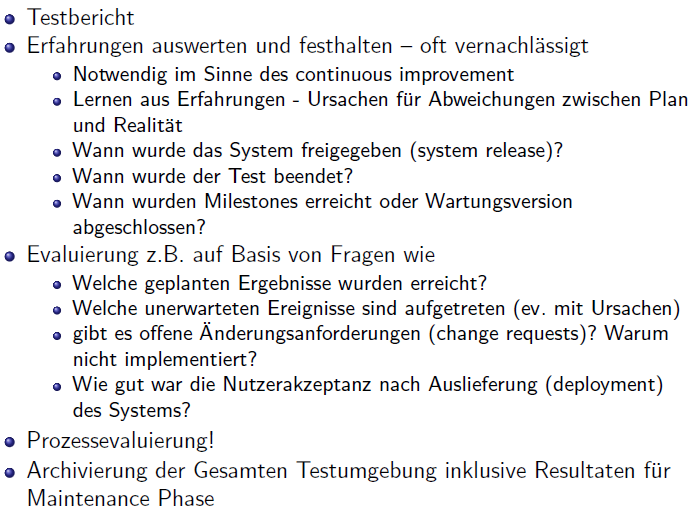
**Testrealisierung**



**Testdurchführung**



**Testabschluss**



**7 Prinzipien des Testens**

* Tests zeigen die Anwesenheit von Fehlern, nicht deren Abwesenheit
* Vollständigen testen bei nicht-trivialen Testobjekten ist nicht möglich
* Testing sollte so früh wie möglich beginnen, später entdeckte Fehler sind teuer
* Fehler sind nicht gleichmäßig über Software verteilt
* Tests anpassen an Einsatzumgebung und Anwendungsrandbedingungen
* Keine Fehler, bedeutet nicht direkt ein nützliches System

# Testen im Entwicklungszyklus (im V-Modell)

**Komponententest:**

Testobjekte:

* Einzelne Funktionen/Prozeduren/Methoden
* Klassen
* Zusammengesetzte Komponenten aus mehreren Objekten mit definierten Schnittstellen

Ziele:

* Sicherstellen der korrekten und vollständigen Umsetzung der Funktionalität
* Input-/Outputverhalten
* Robustheit
* Ressourcennutzung
* Codequalität

Strategie

* Black Box Tests (Verlust von Coverage)
* Automatisieren (z.B. JUnit, Coveragemessung)

**Integrationstest:**

Testobjekte:

* Bereits lokal getestete Komponenten, die zu einer strukturierten Einheit zusammengeführt werden
* Zusätzlich nötig: „monitors“ als Werkzeuge, die Datentransfer zwischen Komponenten lesen

Ziele:

* Korrekte Zusammenarbeit der Komponenten
* Interface-Probleme aufdecken

Strategie:

* Bottom-Up: Schrittweise Integration
* Top-Down: Erst Top-Level Komponenten, dann schrittweise untere Ebenen
* Big-Bang: Alles einzeln testen, dann zusammenführen
* Backbone: Skelett erstellen, dann nach und nach Komponenten hinzufügen
* Notwendig: Dynamischer Test beim Untersuchen auf Interface-Probleme
* Außerdem: Testen nicht-funktionaler Eigenschaften (Performanz, etc.)
* Schwierigkeit: Komponenten werden unterschiedlich schnell fertig,   
  daher notwendige Randbedingungen: Systemarchitektur, Projektplan, Testplan

**Systemtest:**

Testobjekte:

* Voll integriertes System
* Reale Systemumgebung (keine Stubs oder Treiber) auf der Testplattform

Ziele:

* Überprüfen, ob das System alle funktionalen und nicht-funktionalen Eigenschaften erfüllt
* Finden nicht dokumentierter Eigenschaften, die das System aufweist

Strategie:

* Priorisierung einzelner Anforderungen im Testplan

**Akzeptanztest:**

Testobjekte:

* Gesamtsystem in Anwesenheit des Kunden
* Alphaversion, Betaversion

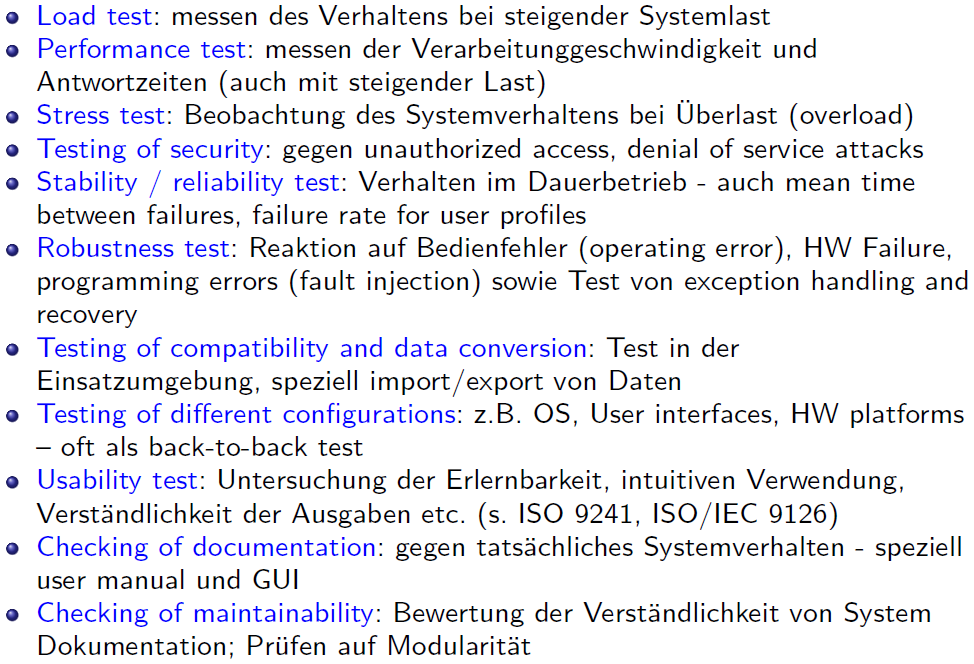
Ziele:

* Erfüllen aller im Vertrag festgehaltenen Dinge
* Nachweise aller erfüllten Eigenschaften
* Review durch den Kunden (speziell durch Systemadministratoren)
* Wenn Kunde nicht Endnutzer ist: User acceptance tests:
  + Endnutzer durch Prototypen testen lassen
  + Feedback einholen

Strategie:

* Kunden-/Endnutzerumgebung schaffen
* Feldtests (Alpha-/Betaversion) von Prototypen

**Nichtfunktionales Testen** (nach Myers)**:**



# Statisches Testen

**Motivation:**

* Testen ohne Ausführung des Testobjekts
* Finden von Abweichungen in der Dokumentation, Standards, Projektplan

Ansätze bezgl.

* Intensität
* Formalität
* Ressourcen
* Ziele

**Strukturierte Gruppenprüfungen**

Review

Verlauf:

* Planung
* Einführung
* Vorbereitung
* Treffen
* Überarbeitung
* Nachbearbeitung

Rollen:

* Manager
  + Auswählen der Review-Objekte
  + Teamzusammenstellung
  + Analyse der Review-Ergebnisse bzgl. Der Konsequenzen
  + Nimmt nicht aktiv am Review teil
* Moderator
  + Verwaltung (Planung, Vorbereitung)
  + Durchführung des Reviews (ordnungsgemäß)
  + Diskussionsleitung
  + Keine eigene Meinung (neutral)
  + Verteilen von Review-Aufgaben
* Autor
  + Ersteller des Review-Objekts
  + Führt notwendige Änderungen durch
* Gutachter
  + Individuelle Prüfung des Review-Objekts nach bestimmten Kriterien
  + Protokollieren von Fehlern
  + Soll auch gute Teile des Review-Objekts ausweisen
* Protokollführer
  + Dokumentation

Typen:

* walkthrough
  + Ziel: finden von Defekten, Mehrdeutigkeiten, Probleme in der Dokumentation
  + Keine Vorbereitung, nur Meeting
  + Autor stellt Objekt anhand von Use Cases/Szenarien vor
  + Reviewer stellen Fragen
  + Kleines Team (<= 10 Personen)
* Inspection
  + Ziel: Bewertung des Objektes
  + Klarer Zeitplan und Ablauf
  + Detaillierte Dokumentation
  + Datensammlung für die Bewertung des Review-Prozesses an sich
* technical Review
  + Ziel: Überprüfung des Objektes gegenüber der Spezifikation, Eignung für den geplanten Einsatz, Standartkonformität
  + Team aus technischen Experten
  + Aufwändige Vorbereitung
  + Review-Meeting ohne Autor
* informal Review
  + Ziel: Unterstützung des Autors bei Entscheidungen, Lokalisation von Fehlern
  + Meist vom Autor einberufen
  + Varianten
    - Pair Programming
    - Buddy testing
    - Code swapping
  + Hohe Akzeptanz bei Entwicklern und Management

Stand: Folie 131, slides.pdf, Statisches Testen

**Statische Analyse:**

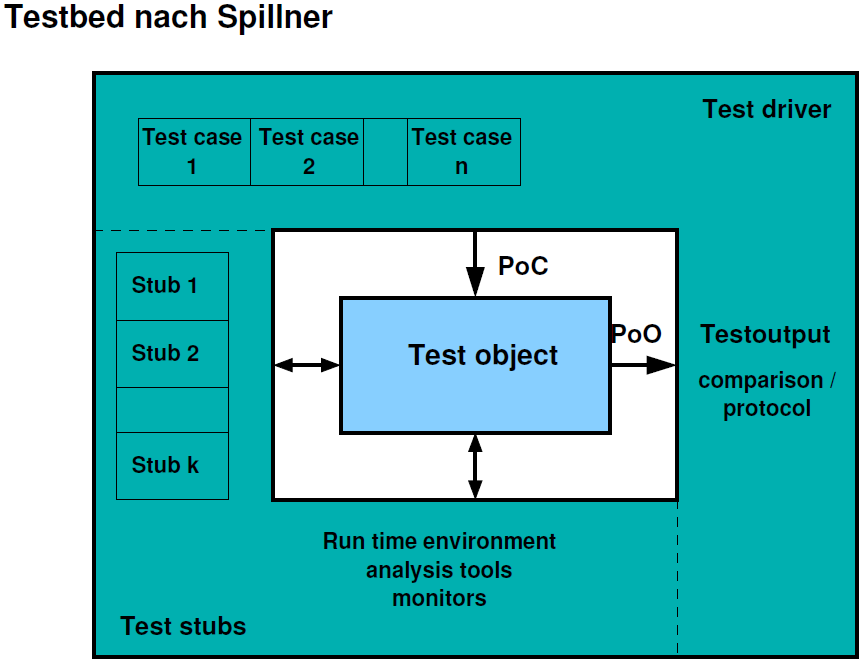
* Werkzeugbasierte Analyse (Viele Werkzeuge durch den Compiler)
* Fehlerklassen
  + Syntax-Verletzungen
  + Abweichungen von Code-Konventionen, Kontrollfluss, Datenfluss
  + Sicherheitsprobleme

# Dynamisches Testen

**Motivation:** Prüfung bei/durch Ausführung des Testobjekts

**Ansätze:**

* Überwiegend Systemtests (anforderungsbasiert)
* Partitionstests (Testen verschiedener Eingaben, die sich unterschiedlich verhalten, Bsp. Äquivalenzklassentest)
* Strukturelle Tests (basiert auf der Struktur des Programms, meist Komponententests, Bsp. Pfadüberdeckung, Anweisungsüberdeckung)



*PoC: Point of Control; PoO: Point of Observation*

Methoden für Testfall-Spezifikation

* Black-Box-Verfahren: Äquivalenzklassenbildung, Grenzwertanalyse, Zustandsbezogener Test
* White-Box-Verfahren: Anweisungsüberdeckung, Zweigüberdeckung, Testen von Bedingungen, Pfadüberdeckung
* Intuitive Testfallermittlung

## Black Box Tests

**Vorteile**

* Prüfung der Funktionalität unabhängig von der Implementierung

**Nachteile**

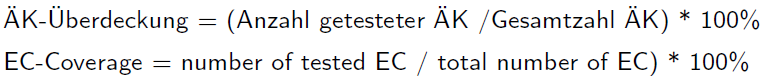
* Kein Erkennen fehlerhafter Spezifikationen
* Nicht geforderte Funktionalität wird nicht erkannt („toter Code“)

**Anwendung**

* Abnahmetests, Schnittstellentest: Überall wo die äußere Funktion relevant ist

**Äquivalenzklassenbildung**:

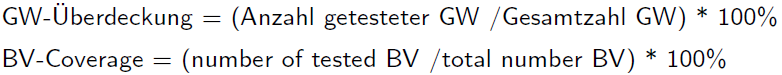
* Äquivalenzklasse: Menge von Werten (Eingabe- oder Ausgabewerte des Testobjekts), die ein gleichartiges Sollverhalten zeigt
* Vorgehen:
  + Ermitteln des Definitionsbereichs für jede Eingabevariable („zulässige Werte“)
  + Festlegen der erwarteten Ergebnisse aus der Spezifikation
* Speziell: Nutzen „unzulässiger“ Eingaben und Ausgaben
* Grenzwerte der Äquivalenzklassen decken oft Missverständnisse und Ungenauigkeiten auf
* Keine Berücksichtigung von Abhängigkeiten und Wechselwirkungen zwischen Parametern
* Wirkungsvoll in Kombination mit Grenzwertanalyse



*Achtung: Ein hoher Überdeckungsgrad kann bei fehlenden Äquivalenzklassen irreführen!*

**Grenzwertanalyse**

* Motivation: Fehler treten häufig an den Grenzbereichen der Äquivalenzklassen auf



**Zustandsbasierte Tests**

* Basierend auf Automaten -> Übergangsbäume
* Startzustand, Folgezustände, Zustandsübergänge
* Bsp. Konformitätstest, Robustnesstest
* Testendekriterien:
  + Jeder Zustand mind. einmal erreicht
  + Jeder Zustandsübergang mind. einmal durchgeführt
  + Alle zustandsverletzenden Zustandsübergänge geprüft
  + Zusatz: Alle Kombinationen von Zustandsübergängen (in jeder Reihenfolge/ mehrfach hintereinander)

**Use Case Testing**

* Motivation: Use-Cases als Requirementspec
* Externe Sicht des Systemverhaltens
* Relevant für System und Abnahmetest
* Bsp. User Stories, Gebrauchsszenarien

**Weitere Black-Box-Tests**

* Syntextest
* Zufallstest
* Smoke-Test (Ausprobieren des Testobjektes)

## White Box Tests

**Vorteile**

* Geeignet für untere Testebenen (Modul, Komponente)
* Guter Support durch Werkzeuge

**Nachteile**

* Nicht umgesetzte Teile der Spezifikation werden u.U. nicht erkannt
* Instrumentierter Code (Bsp. Messen von Überdeckung), evtl. anderes Verhalten ohne diesen zusätzlichen Code (zumindest Laufzeit)

**Anwendung**

**Anweisungsüberdeckung**

* Eine festgelegte Quote der Anweisungen (Testendekriterium) soll mind. einmal ausgeführt werden (C0-Maß)
* Basiert auf Kontrollflussgraph
* Testdurchlauf beinhaltet Nachweis welche Anweisungen ausgeführt wurden
* Ziel: Potentiell unerreichbaren Code identifizieren (Leere IF-Zweigen werden u.U. nicht erkannt

**Zweigüberdeckung**

* Eine festgelegte Quote der Bedingungen soll mind. einmal ausgeführt werden (C1-Maß)
* Basiert auf Kontrollflussgraph
* Testdurchlauf beinhaltet Nachweis welche Bedingungen ausgeführt wurden
* Ziel: Unerreichbaren Code identifizieren, Leere IF-Zweige erkennen

**Bedingungsüberdeckung**

* Alle Möglichkeiten der Bedingungsauswertung berücksichtigen
* Einfache Bedingungsüberdeckung: Jede atomare Teilbedingung wird mit TRUE und FALSE berücksichtigt (Branch Condition Testing)
* Mehrfache Bedingungsüberdeckung: Jede Kombination der atomaren Bedingungswerte soll berücksichtigt werden (Branch Condition Combination Testing)
* Minimale Mehrfachbedingungsüberdeckung: Nur Varianten von Kombinationen atomarere Bedinungswerte, die sich unterscheiden, werden berücksichtigt (Modified Branch Condition Decision Testing)
* Testendekriterium: Festgelegte Quote

**Weitere White-Box-Verfahren**

* Linear Code Sequence and Jump (LCASJ): Folgen von Anweisungen im Vordergrund
* Datenbasierte Verfahren: Basiert auf lesenden und schreibenden Zugriff auf Variablen

# Test Management

**Organisation**

* Motivation: Koordination und Kooperation
* Unabhängige Tester?
  + Vorteile
    - Unvoreingenommene Tester
    - Tester verifizieren implizite Annahmen der Entwickler
  + Nachteile
    - Häufig isolierte Tester, Mangel an Kommunikation
    - Tester werden oft Bottleneck, wenn Ressourcen nicht gut geplant sind
    - Entwickler verlieren Sinn für Verantwortung für das Produkt
  + Lösung: Kombination, Grad an Unabhängigkeit
* Rollen
  + Testmanager: Testplanung, Testkontrolle
  + Test Designer: Spezifikation
  + Test Automator: Automatisierung, Tools
  + Test Administrator: Testsystemumgebung
  + Tester: Testausführung, Fehlererkennung
* Soziale Kompetenzen
  + Teamfähigkeit
  + Diplomatie
  + Hinterfragen
  + Sicheres Auftreten
  + Genauigkeit
  + Kreativität

**Planung**

* Ziele und Strategie definieren
* Testumgebung definieren und dokumentieren
* Testlevel, Kooperation, Integration, Koordination planen
* Dokumentation aufsetzen
* Testplan verfassen
* Ressourcenplanung

**Priorisierung**

* Motivation: Sinnvolles Nutzen von Ressourcen
* Jedem Testfall wird eine Priorisierung zugewiesen
* Kriterien (absteigend)
  + Verwendungshäufigkeit
  + Fehlerwahrscheinlichkeit
  + Risiko
  + Sichtbarkeit eines Fehlers für den Endnutzer
  + Priorität der Anforderung
  + Wichtigkeit der Nichtfunktionalen Anforderung
  + Priorität der Komponente aus Architektursicht
  + Komplexität der Komponente
  + Risikoabschätzung für das Gesamtsystem

**Testendekriterien**

* Festlegen für Gesamttest oder einzelne Testlevel
* Minimierung der Risiken für das Projekt
* Kriterien
  + Testabdeckung (Coverage)
  + Produktqualität
  + Restrisiko
  + Kosten, Projektrisiko, Liefertermine, Marktchancen

**Testzyklen**

* Planen - Aspekte:
  + Entwicklungsstatus
  + Testergebnisse
  + Ressourcen
* Überwachen
  + Messen und Überwachen der Testresultate
  + Verwendung zuverlässiger Metriken
  + Zusammenfassen als Bericht
    - Testobjekt
    - Testverlauf
    - Status
    - Risiken
    - Ausblick auf den nächsten Zyklus
    - Subjektive Bewertung
* Steuern
  + Änderungen müssen jederzeit möglich sein
  + Maßnahmen bei Abweichungen vom Plan oder kritischen Fehlern
    - Neuplanung von Ressourcen
    - Plan ändern
    - Verlängern von Testphasen
  + Detaillierte Dokumentation von Ursachen und Maßnahmen

**Fehlermanagement**

* Protokoll
  + Analyse nach jedem Testlauf, spätestens nach einem Zyklus
  + Abweichungen prüfen
  + Detaillierte Dokumentation
* Bericht
  + Datenbank mit Berichten pflegen -> Einheitliches Format
  + Basis für Dokumentation mit Entwickler
  + Basis für Fehlerreproduzierung
* Klassifikation
  + Schwere: Mild -> Moderat -> Ernst -> Sehr Ernst -> Fatal
  + Priorität: Offen -> Bei Gelegenheit -> Nächster Release -> Sofort
* Status
  + Schema mit Verantwortlichen
    - NEW: Tester, Nutzer
    - OPEN: Testmanager
    - REJECTED: Testmanager
    - ANALYSIS: Entwickler
    - OBSERVATION: Entwickler
    - CORRECTION: Projektmanager
    - TEST: Entwickler
    - CLOSED: Tester
    - FAILED: Tester

# Testwerkzeuge

**Typen**

* Unterstützung von
  + Erfassen, Katalogisieren, Administration von Testfällen
  + Status-Verfolgung (passed/failed?)
  + Ressourcen- und Zyklusplanung
* Automatisiertes Erzeugen von Berichten und Dokumentation

**Spezifikation**

* Abhängig von der Testbasis
  + Datenbank-basiert: Ableitung von Testfällen aus dem Datenbankschema
  + Code-basiert: Testfälle generiert aus dem Sourcecode
  + Schnittstellen-basiert: Analyse der Schnittstellen, Bildung von Äquivalenzklassen
  + Spezifikations-basiert: Voraussetzung -> Spezifikation formal genug (Bsp. UML)

**Statische Tests**

* Review Support
  + Unterstützung für Planung, Durchführung, Auswertung
  + Zusammentragen von Information und Dokumentation von Review-Sitzungen
  + Bsp. Online-Checklisten
* Statische Analyse
  + Metrik-Tools
  + Bsp. Komplexität

**Dynamische Tests**

* Plattform und Sprachabhängig
* Ansätze
  + Debugger
  + Testtreiber
  + Simulator
  + Bots
  + Vergleicher (Comparator)
  + Dynamische Analyse
  + Abdeckungsanalyse (Coverage)

**Nichtfunktionale Tests**

* Loadtests, Performancetests
* Monitore
* Sicherheitstest (Security)

**Auswählen der Tools**

* Schritte
  + Anforderungen identifizieren
  + Marktrecherche (verschiedene Anbieter)
  + Demo
  + Evaluierung
  + Auswahl des Tools
* Kriterien
  + Qualität
  + Vorkenntnisse
  + Integrierbarkeit
  + Lauffähige Plattform
  + Hersteller Service
  + Lizenzbedingungen, Preis, Wartungsaufwand
* Einführung eines neuen Tools

1. Pilotprojekt
2. Evaluierung
3. Regeln für den Gebrauch aufstellen
4. User Training
5. Schrittweise Einführung
6. Begleitendes Coaching