Testen

Prinzipien und Methoden

ALP 2 SS2002

4.7.2002

Natalie Ardet

Definition

Im folgenden gilt:

Software

=

Programm + Daten + Dokumentation

Motivation

- Software wird immer mehr in Bereichen eingesetzt, wo ein Versagen zu wirtschaftlichen Verlusten führt und Menschenleben gefährdet wird.
- 50 % der Ausfälle im industriellen Sektor sind auf Software-Fehler zurückzuführen

Was sind die Folgen?
=> Zunehmende Qualitätsanforderungen

Beispiel

Das akzeptieren eines 0,1%-Defektniveaus bedeutet:

- Pro Jahr: 20.000 fehlerhafte Medikamente
  300 versagende Herzschrittmacher
- Pro Woche: 500 Fehler bei med. Operationen
- Pro Tag: 16.000 verlorene Briefe in der Post
  18 Flugzeugabstürze
- Pro Stunde: 22.000 Schecks nicht korrekt gebucht

Verifikation vs. Validation

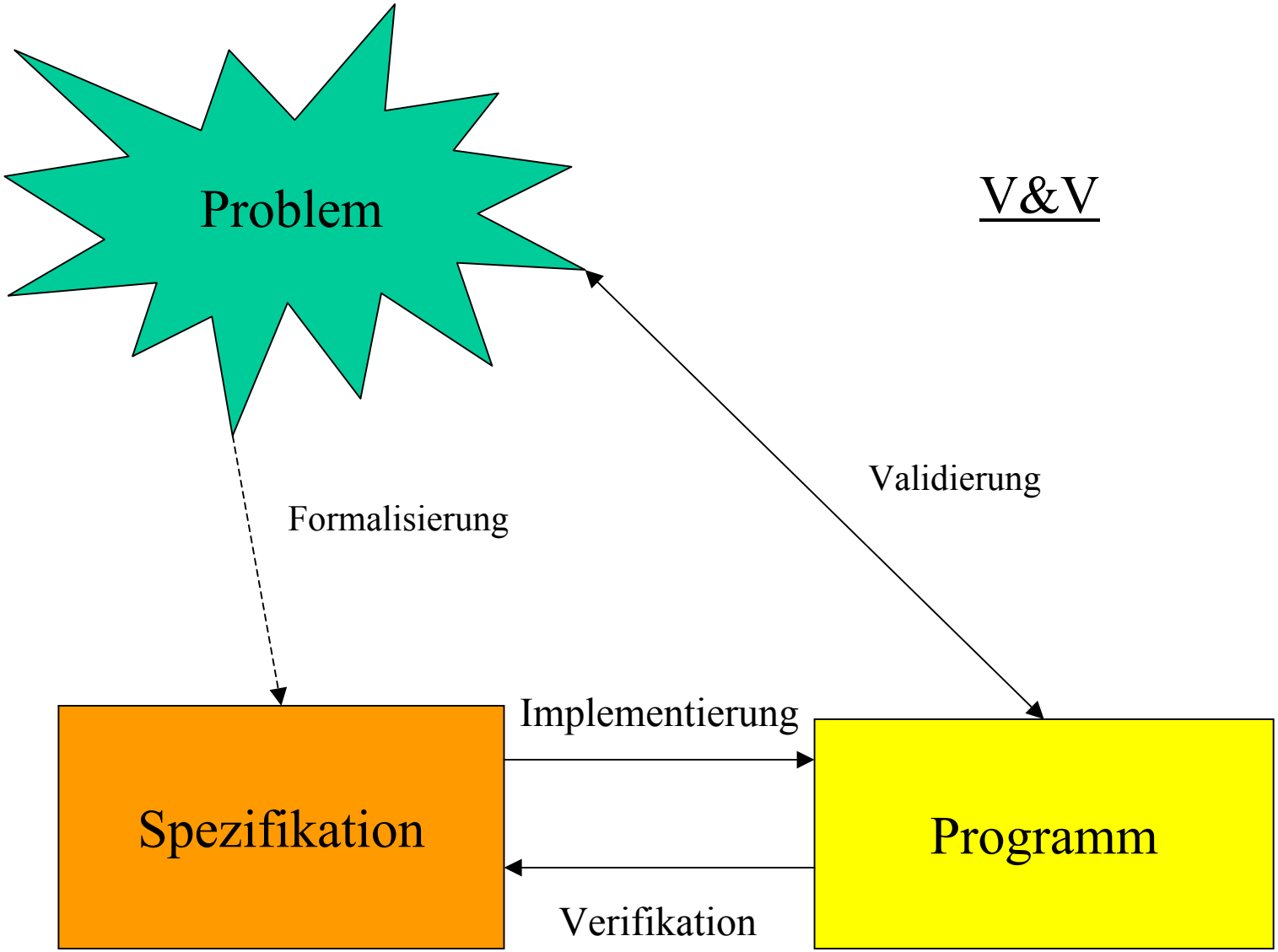
- Mittels Verifikation wird festgestellt ob ein Programm seiner Spezifikation entspricht.

  => z.B. mit dem Hoare-Kalkül.

Die Verifikation beantwortet die Frage:
„Erstellen wir das Produkt richtig?“ Boehm (1979)

Wie beantworte ich die Frage „Erstelle ich das richtige Produkt?“ ? => Validierung

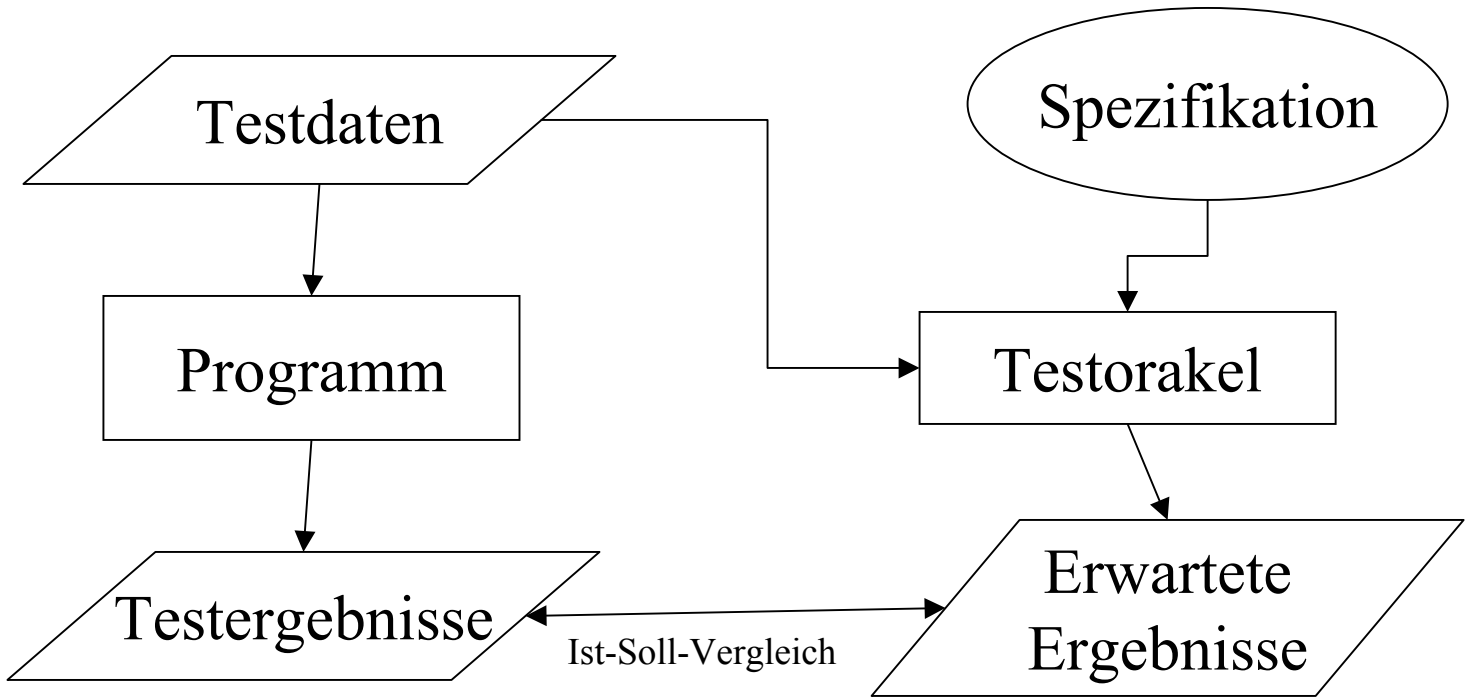
V&V


Problem
Formalisierung
Validierung
Implementierung
Spezifikation
Programm
Verifikation

Das Testen

- Das Testen ist ein Validierungsverfahren, um festzustellen ob ein Programm/Komponente seinen Zweck erfüllt.
- Das Testen ist ein dynamisches Verfahren.
- Als „Nebeneffekt“ kann das Vorkommen von Fehlern in dem Programm festgestellt werden.
- Die Fehlerbehebung ist ein Vorgang, der diese Fehler lokalisiert und korrigiert.

Prinzip


Testdaten
Spezifikation
Programm
Testorakel
Testergebnisse
Ist-Soll-Vergleich
Erwartete
Ergebnisse

Quiz

Ein Programm wird getestet. Die Ausgaben entsprechen den Erwartungen.

Ist das Programm fehlerfrei?

Kernaussage

- Tests zeigen das Vorhandensein, nicht die Abwesenheit von Programmfehlern.

Erläuterung:
Wirklich vollständige Tests, bei denen jeder denkbare Ablauf der Ausführung eines Programms getestet wird, sind praktisch undurchführbar.

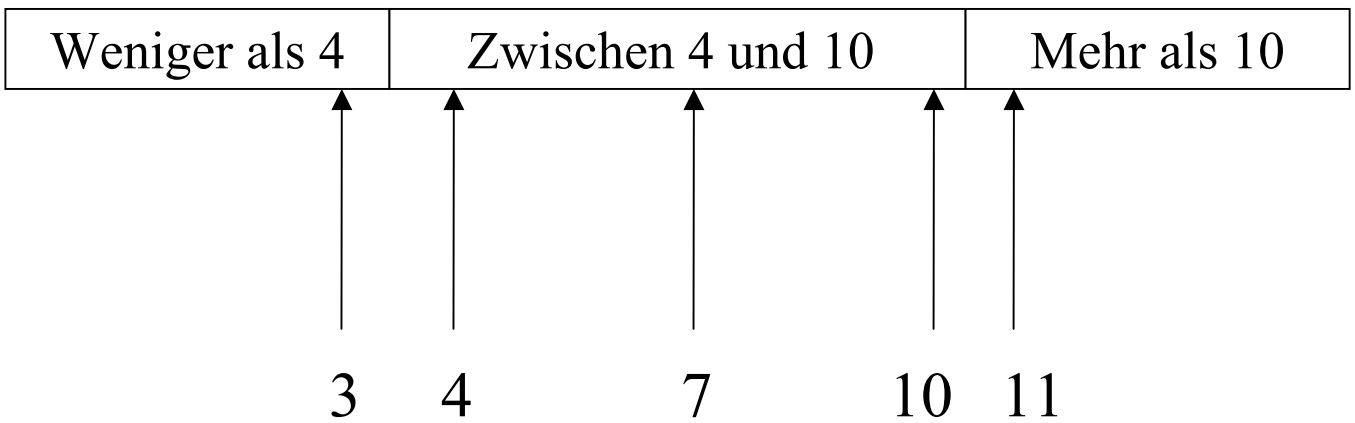
Äquivalenzklassen

- Ein systematischer Ansatz des Testens beruht auf die Partitionierung der Eingabedaten in Klassen.
- Die Programme verhalten sich für alle Mitglieder einer Klasse auf vergleichbare Weise.
- Wegen dieses äquivalentes Verhalten werden diese Klassen Äquivalenzklassen genannt.
- Die Testfälle werden so entworfen, dass die Eingaben mindestens ein Element jeder Klasse beinhalten.

Grenzwerte

- Eine gute Richtlinie, an der man sich bei der Auswahl der Testdaten orientieren kann, besteht darin, zum einen Testdaten an den Grenzen von Klassen und zum anderen solche auszuwählen, die in der Mitte der Klasse liegen.

Beispiel



Testverfahren

- Funktionales Testen
- Strukturelles Testen
  - Kantenüberdeckungstest
  - Pfadüberdeckungstest

Funktionales Testen

- Funktionale Tests stellen einen Testansatz dar, bei dem die Tests von der Programm- oder Komponentenspezifikation abgeleitet werden.
- Das System ist eine „Black Box“, deren Verhalten nur durch die Untersuchung ihrer Eingaben und der dazugehörigen Ausgaben festgestellt werden kann.
- Funktionales Testen beschäftigt sich nur mit der Funktionalität und nicht mit der Implementierung.

Strukturelles Testen

- Testansatz, bei dem die Tests aus der Kenntnis der Implementierung abgeleitet wird.
- Auch „White-Box-Test“, „Glass-Box-Test“ oder „Clear-Box-Test“ genannt.
- Für kleine Programmeinheiten geeignet (z.B. eine Methode)
- Der Code wird analysiert, dadurch wird die Ablaufstruktur bekannt.
- Die Testdaten werden aus der Ablaufstruktur abgeleitet.

Kantenüberdeckungstest

Definition:

Kantenüberdeckungstests sind eine Strategie des strukturellen Testens, deren Ziel darin besteht alle Anweisungen im Programm oder in der Komponente wenigstens **einmal** im Verlauf des Testprozesses auszuführen

Pfadüberdeckungstest

Definition:

Pfadüberdeckungstests sind eine Strategie des strukturellen Testens, deren Ziel in der Erprobung jedes einzelnen unabhängigen Ausführungspfades eines Programms oder einer Komponente besteht.

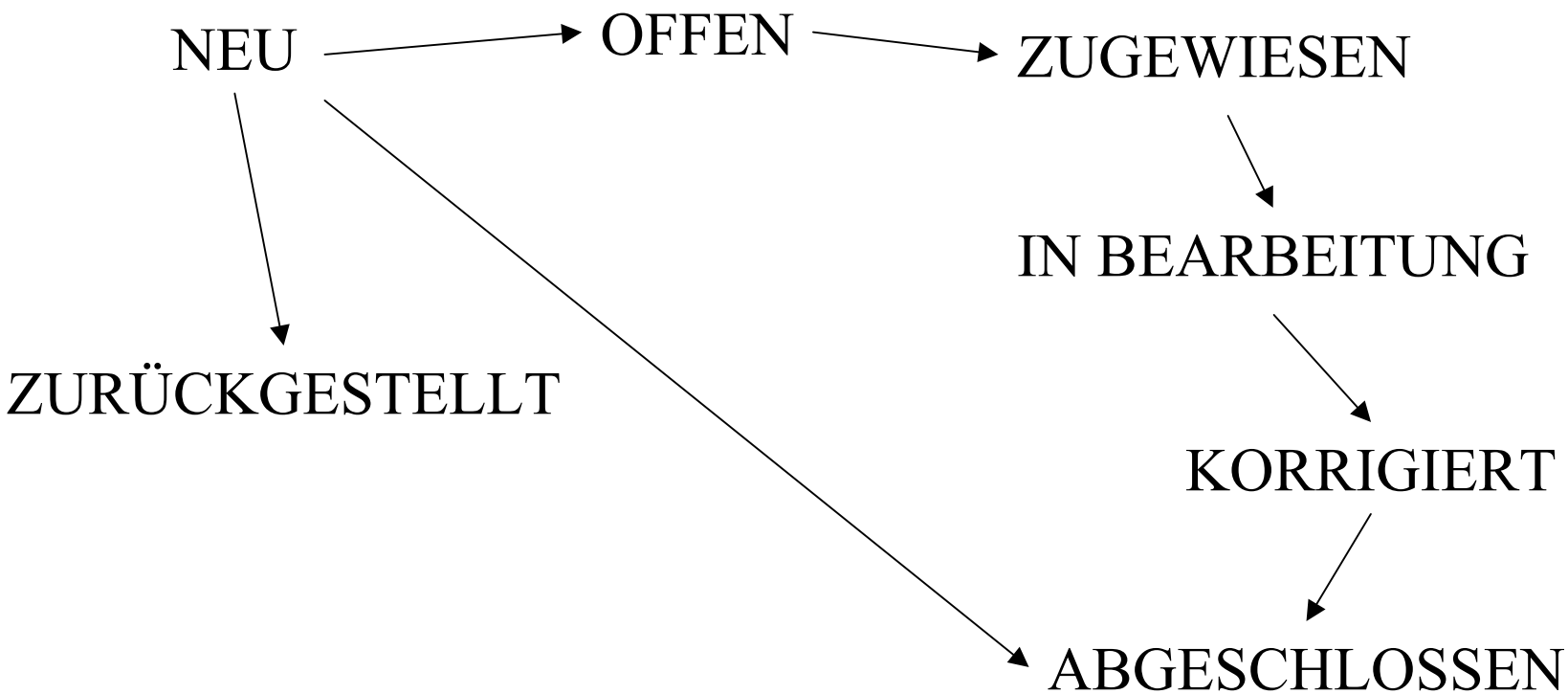
Fehlerbehebung

- Es muss sichergestellt werden, dass der „vermeintlicher“ Fehler ein „tatsächlicher“ Fehler ist.
- Es muss sichergestellt werden, dass die Fehlerbehebung keine neuen Fehler zur Folge hat => Regressionstest

Prioritätsebene eines Fehlers

1. Kritisch: Der Betrieb der Anwendung ist unterbrochen.
2. Hohe Priorität: ein signifikantes Problem, aber das Programm läuft noch.
3. Mittlere Priorität: Das Problem wirkt sich kaum auf den Betrieb der Anwendung aus.
4. Niedrige Priorität: Das Problem hat keinerlei Einfluss auf den Betrieb der Anwendung.

Fehlerlebenszyklusmodell



Dokumentation eines Fehlers

- Die Priorität des Fehlers bestimmen
- Jedem Fehler einen eindeutigen Bezeichner zuordnen
- Testverfahren und Softwareversion dem Fehler zuordnen
- Datum an dem der Fehler berichtet wurde
- Status des Fehlers protokollieren

Testaufwand

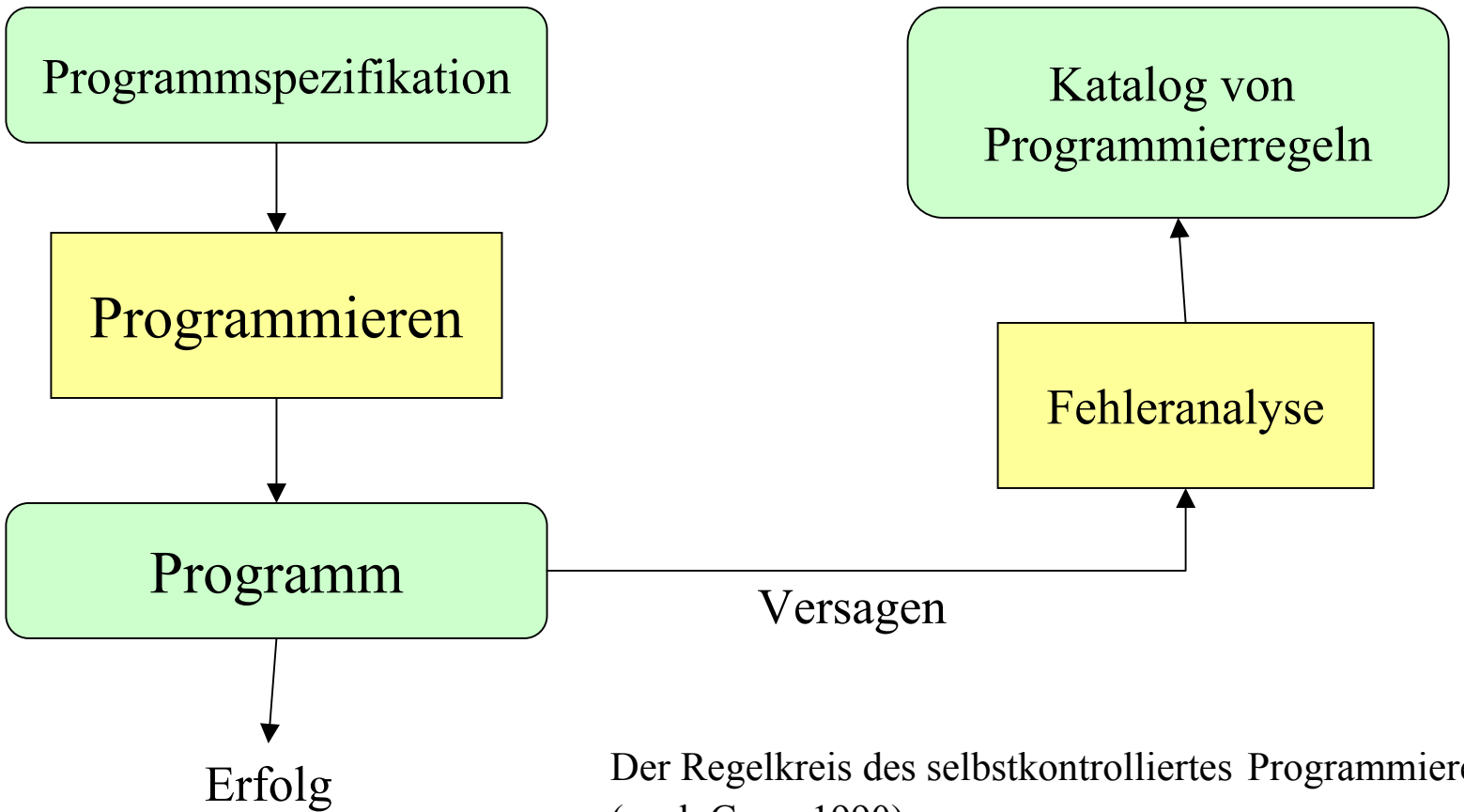
Der Testaufwand lässt sich dadurch verringern, dass man die Fehler mittels Verfahren entdeckt die keine Fehlerlokalisierung benötigen, wie:

- Code-Inspektionen (Vier-Augen-Prinzip)

und die Fehleranzahl reduziert:

- Selbstkontrolliertes Programmieren

Selbstkontrolliertes Programmieren



Der Regelkreis des selbstkontrolliertes Programmierens (nach Gram 1990)

Testphasen

- Komponententest (Entwickler)
- Integrationstest (Entwickler)
- Systemtest (Entwickler, Benutzer)
- Abnahmetest (Benutzer)
  - Alpha-Test (für individuelle Software)
  - Beta-Test (für Software Produkte)

Komponententest

- Eine logische Einheit wird getestet (z.B. eine Klasse).
- Ein Testtreiber muss implementiert werden um das System zu simulieren.

Integrationstest

- Das „Zusammenspiel“ der *getesteten* Komponenten wird getestet.
- Der Integrationstest sollte inkrementell sein: keine „Big-Bang Integration“
- Fehlende Komponenten werden durch Testtreiber oder Stubs simuliert. Diese sind Platzhalter und verfügen über eine geringere Funktionalität als die fehlenden Komponenten.

Beispiel

- A verwendet B und A soll getestet werden
  => es wird ein Stub für B geschrieben, der das Verhalten von B simuliert.
- B verwendet A und A soll getestet
  => es wird ein Testtreiber geschrieben, der die Interaktion mit B simuliert.

Testplanung

Reihenfolge der Testplanung

1. Abnahmetest
2. System & Integrationstest
3. Komponententest

Neue Testformen

- Ursprünglich diente das Testen der Validation einer Software im Ganzen:
  => Validation gegen Benutzeranforderungen
- Heutzutage wird Test-Driven Development betrieben, wo jede Komponente eines Programms intensiv getestet wird.
  => Validation/Verifikation gegen Entwickleranforderungen
  Beispiel: eXtreme Programming & Unit Testing

Literatur/Quellen

- Theorie: Software Engineering (Kap.7) W.Zuser et. Al., Pearson Studium 2001.
- Praxis: Effective Methods for Software Testing, W.E.Perry, Wiley 1999
- Software: Junit, www.junit.org