# 1 Qualitätssicherung und Testen von Software

## Inhalt



## Abbildungen

## 1.1 Was versteht man unter Qualität

**Qualität** (quality) – der Grad, in dem ein Satz inhärenter Merkmale Anforderungen erfüllt. (ISO 9000:2000)

**Anforderung** (Requirements) – ein Erfordernis oder eine Erwartung, das oder die festgelegt, üblicherweise vorausgesetzt oder verpflichtend ist.

I**nhärentes Merkmal** (inherent characteristic) – eine kennzeichnende Eigenschaft einer Einheit (Produkt, Dienstleistung, Prozess, System, Person, Organisation, etc.), welche diese aus sich selbst heraus hat und die ihr nicht explizit zugeordnet ist.

- Qualität ist Zielerfüllung. Die Ziele (Anforderungen) können explizit festgelegt oder implizit durch gemeinsame Vorstellungen der Beteiligten gegeben sein.
- Eine Auffassung von Qualität als reine Zweckeignung oder Kundenzufriedenheit greift zu kurz. Sie erfasst den Qualitätsbegriff nicht in seiner Gesamtheit.
- Qualität ist kein absolutes Mass für die Güte einer Einheit.
- Qualität entsteht nicht von selbst. Sie muss definiert und geschaffen werden.
- Qualität bezieht sich sowohl auf Produkte als auch auf Prozesse und Projekte zur Herstellung dieser Produkte.

**Qualitätsmanagement** (quality management) – aufeinander abgestimmte Tätigkeiten zum Leiten und Lenken einer Organisation bezüglich Qualität. Leiten und Lenken bezüglich Qualität umfassen üblicherweise das Festlegen der Qualitätspolitik und der Qualitätsziele, die Qualitätsplanung, die Qualitätslenkung, die Qualitätssicherung und die Qualitätsverbesserung . (ISO 9000:2000)

## 1.2 Konstruktives Quality Management

Bei den konstruktiven QM-Massnahmen wird gewährleistet, dass es dokumentierte Vorgehensweisen für den Software-Entwicklungsprozess gibt. Dies umfasst zum Beispiel die Verwendung von Programmierrichtlinien oder die Durchführung von Reviews und Code-Inspektionen.

Ein *Review* ist ein formal geplanter und strukturierter Analyse- und Bewertungsprozess, in dem die Ergebnisse einer Aktivität einem Team präsentiert und von diesem kommentiert oder genehmigt werden.

Bei einer *Code-Inspektion* wird der erstellte Programmcode durch einen anderen Entwickler bewertet. Dabei wird zum einen auf die Einhaltung der Programmierrichtlinien und zum anderen auf potentielle Fehler und mögliche Performanceverbesserungen geachtet.

Konstruktive Quality Management Massnahmen sorgen von vorneherein für ein gewisses Mass an Qualität :

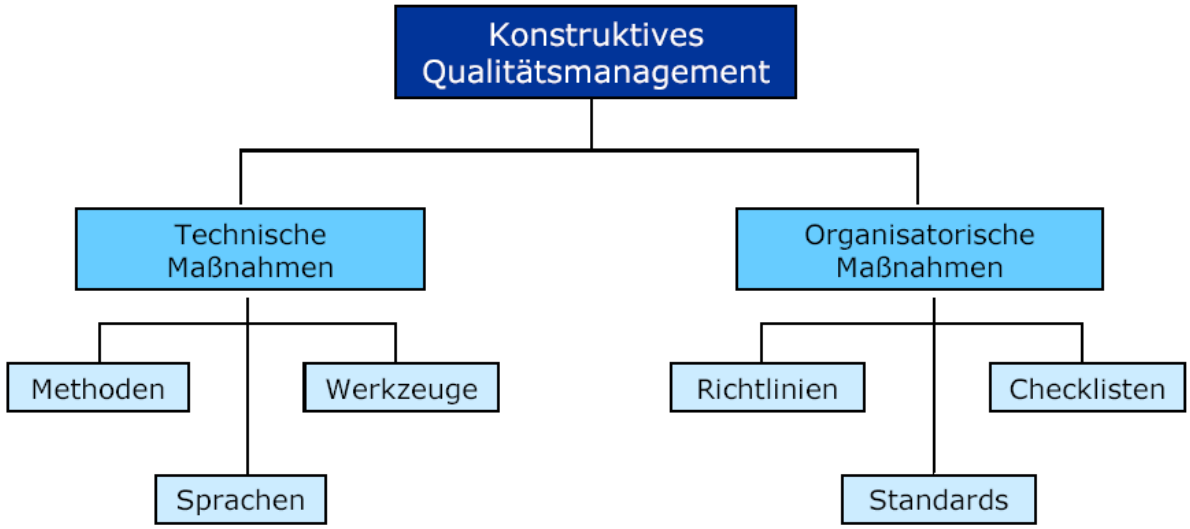


***Abbildung 1-1: Konstruktives Qualitätsmanagement***

## 1.3 Analytisches Qualitätsmanagement

Die analytische Qualitätssicherung umfasst das Testen von System-Bestandteilen und des Gesamtsystems (d.h. Unit-Test, Systemtest, Integrationstest, User-Acceptance-Test). Es ist darauf zu achten, dass die Erfüllung aller Kundenanforderungen durch die Tests abgedeckt wird und dass es Möglichkeiten von einfachen Regressionstests gibt (d.h. funktioniert das System nach einer Änderung noch mindestens in dem Umfang, wie es vor der Änderung funktioniert hat).

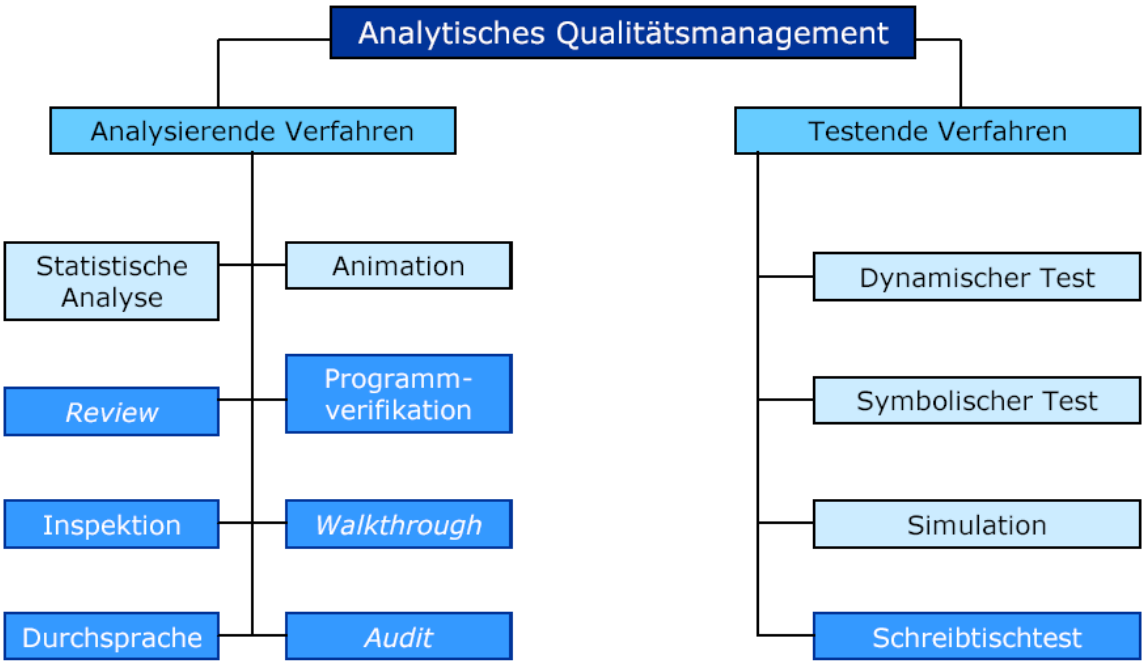


***Abbildung 1-2: Analytisches Qualitätsmanagement - Testpunkte***

## 1.4 Qualitätsmerkmale von Software

Nachfolgende Abbildung zeigt die Punkte, an welchen die Qualität einer Software im Allgemeinen gemessen wird :

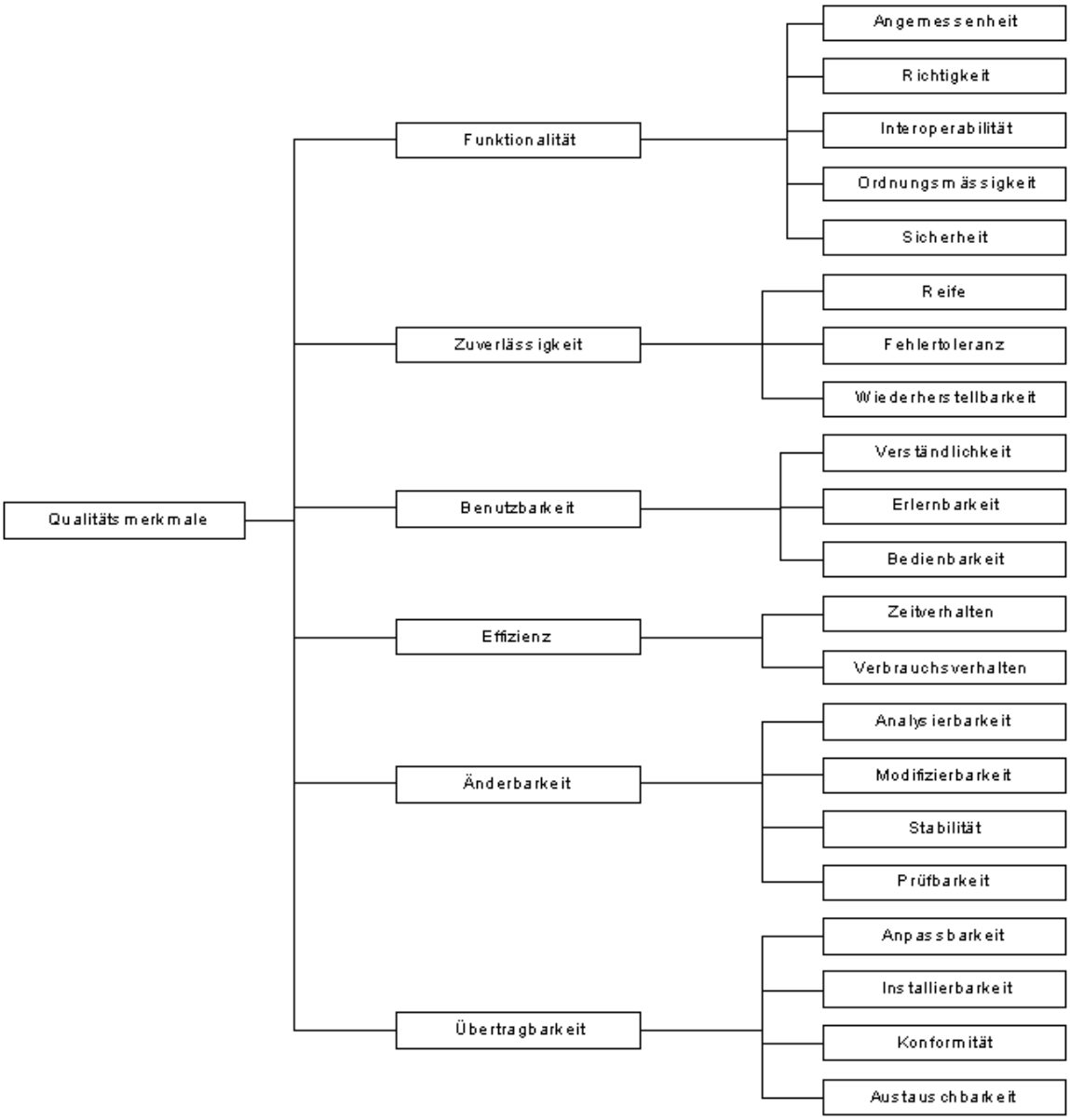


***Abbildung 1-3: Qualitätsmerkmale von Software***

## 1.5 Qualitätsmanagementkosten

Laut DIN 55350 versteht man unter Qualitätskosten alle Kosten, die durch Tätigkeiten der Fehlerverhütung, der planmässigen Qualitätsprüfung sowie durch intern oder extern festgestellte Fehler verursacht werden. Hierzu zählen vorwiegend:

- Organisationskosten
- Qualitätsplanung
- Qualitätscontrolling
- Qualitätsförderung
- Qualitätsverbesserung

- Prüfkosten
- Eingangprüfung
- Dokumentation
- Ausgangprüfung
- Evaluation
- Kosten der Qualitätsabweichung
- Nacharbeit
- Haftung
- Entgangener Markterfolg.

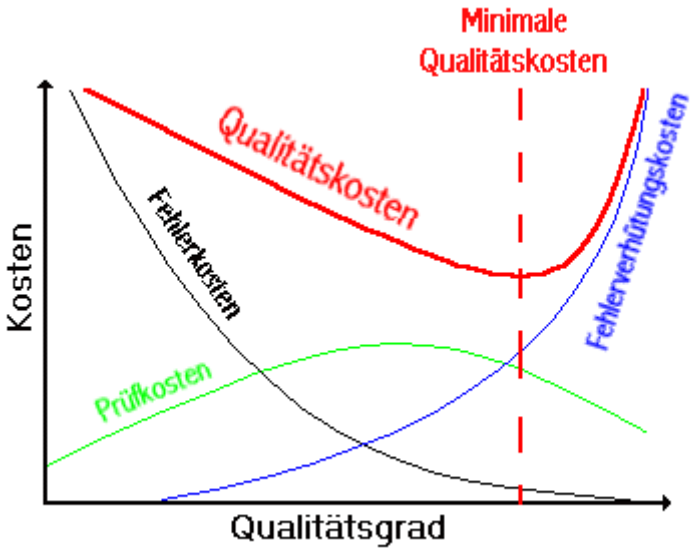


***Abbildung 1-4: Qualitätskosten***

Mit steigendem Qualitätsgrad sinken die Fehlerkosten. Dabei steigen allerdings die Fehlerverhütungskosten an. Die Prüfkosten steigen erst an, ab einem gewissen Punkt ist aber keine genauere, aufwendigere und kostenintensivere Prüfung mehr möglich und dieser Kostenanteil nimmt anteilmässig ab. Dadurch ergibt sich für die Qualitätskosten eine erst abfallende, dann aber wieder ansteigende Kurve. Am tiefsten Punkt dieser Kurve hat man demnach die niedrigsten Qualitätskosten.

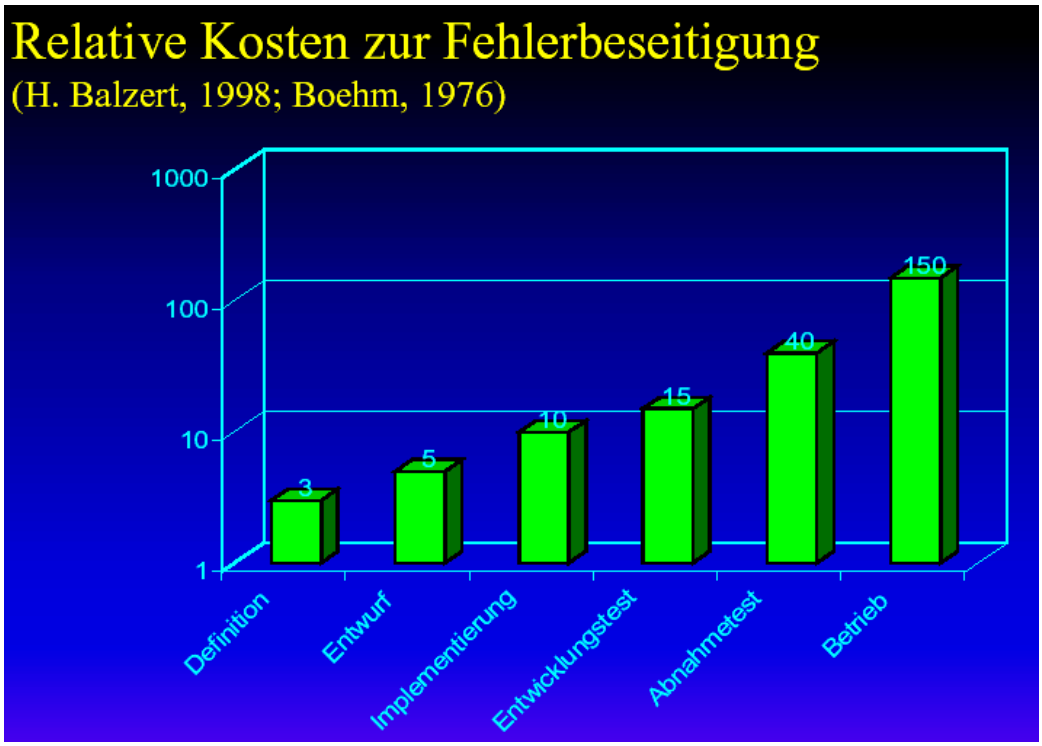


***Abbildung 1-5: Relative Kosten zur Fehlerbeseitigung***

### 1.5.1 Fehlerverhütungskosten

Definition: Fehlerverhütungskosten sind Kosten, die zur Fehlerverhütung oder anderen vorbeugenden Massnahmen der Qualitätssicherung aufgewendet werden.

**Qualitätsplanung**
Vor und während der Entwicklung von Qualitätsprogrammen müssen Qualitätsmerkmale extrahiert, klassifiziert und gewichtet werden. Im Bereich spezifischer Qualitätssicherungsmassnahmen sind die Kosten überschaubar und relativ leicht zu ermitteln, schwieriger dagegen in Bereichen wie Entwicklung, Forschung und Arbeitsvorbereitung zu Qualitätsprogrammen.

**Qualitätsfähigkeitsuntersuchung**
Überprüfung der Eignung von Ressourcen (Mittel, Personal) für die vorgesehene Qualitätssicherung.

**Prüfplanung**
Ermittlung der Kosten, die zur Erstellung der Prüfanweisungen für die Qualitätsprüfungen erforderlich sind (z.B. Festlegung der Prüfverfahren und Stichprobenumfang).

**Qualitätslenkung**
Über vorhandene Qualitätsinformationen werden Massnahmen zur Korrektur eingeleitet und überwacht.

**Qualitätsaudit**
Das Qualitätsaudit beinhaltet Outcome-Audit (Prüfung des Ergebnisses), Verfahrensaudit (Prüfung der Behandlung) und Systemaudit (Prüfung des Qualitätssicherungssystems gemäss Vorgaben wie z.B. DIN-Norm).

**Qualitätsförderung**
Das Streben nach Qualität muss für alle Mitarbeiter unterstützt und gefördert werden. Hierbei entstehen Kosten für Aus-, Fort- und Weiterbildung, Qualitätszirkel, und Massnahmen zur Motivation des Personals.

**Sonstige**
Kosten der Fehlerverhütung (z.B. spezielle, über das normale Mass herausgehende Laboruntersuchungen im Einzelfalle) ausserhalb der o.a. Kategorien.

### 1.5.2 Qualitätsprüfkosten

Definition: Qualitätsprüfkosten sind alle diejenigen Personal- und Sachkosten für Qualitätsprüfungen innerhalb und ausserhalb des Qualitätswesens. Bei Prüfvorgängen, die in anderen Tätigkeiten enthalten sind, muss der Prüfanteil angesetzt werden.

**Prüfmittel**
Beschaffungs- und Betriebskosten unter Berücksichtigung der kalkulatorischen Abschreibungen bei Gerätschaften.

**Prüfkosten**
Kosten für qualitative und quantitative Prüfungen wie Präzisionskontrollen, Richtigkeitsprüfungen (z.B. im Labor und in der Radiologie).

**Dokumentation**
Kosten zur Verwahrung und Verwaltung der Prüfdaten von Qualitätsprüfungen im Hinblick auf Sicherheit und Haftung.

**Instandhaltungskosten**
Kosten für die Instandhaltung der Prüfmittel. Hierzu zählen im technischen Bereich beispielsweise der Kalibrier- und Eichdienst.

**Sonstige**
Alle Kosten, die den bislang erwähnten Bereichen nicht zugeführt werden können, unter anderem Vernetzungs- und Kommunikationssysteme.

### 1.5.3 Interne Fehlerkosten

Definition: Interne Fehlerkosten sind jene Kosten, die bei der internen Aufdeckung von Fehlern aufgewandt werden, um Mängel zu beseitigen.

**Nacharbeit**
Kosten für Mehr- oder Nacharbeit, damit z.B. eine Diagnostik oder Therapie die in sie gesetzten Qualitätsanforderungen erfüllt.

**Wiederholungsprüfung**
Bei mehrfachem Auftreten von Qualitätsmängeln innerhalb eines bestimmten Zeitraumes sind nochmalige, umfangreiche Qualitätsprüfungen indiziert. Dies sind Kosten für Wiederholungsprüfungen.

**Untersuchung zur Fehlerursachenfindung**
Zur nachdrücklichen Problemlösung bei sporadisch auftretenden Mängeln sind ausführliche Untersuchungen zur Auffindung der Fehlerursache erforderlich, z.B. Gutachten.

**Sonstige**
Interne Fehlerkosten, die keinem der oben angegebenen Punkte zugeordnet werden können.

### 1.5.4 Externe Fehlerkosten

Definition: Externe Fehlerkosten sind Kosten für die Mängelbeseitigung von ausserhalb des Krankenhauses entdeckten Verfahrensfehlern.

**Gewährleistung**
Hierzu zählen die Kosten für Gewährleistung.

**Nachbehandlung**
Kosten für die Nachbehandlung

**Haftung**
Entstehen Sach-, Personen- oder Vermögensschäden durch unsachgerechte Behandlung, so sind die hierfür erforderlichen Aufwendungen den Haftungskosten zuzurechnen. Dies gilt auch für die Kosten der Haftpflichtversicherung.

**Sonstige**
Sonstige Kosten treten auf, wenn beispielsweise persönliche Entschuldigungen erforderlich werden. Reisekosten wären in diesem Zusammenhang z.B. "Sonstige Kosten".

# 2 Was versteht man unter Softwareprüfung?

Ein Vorgang zur Kontrolle der Qualitätskriterien einer Software, die in ihrer Spezifikation formuliert, und durch einen definierten Entwicklungsprozess realisiert worden ist. Er dient nicht der Erhöhung der Qualität, da der eigentliche Entwicklungsprozess bereits abgeschlossen ist, sondern der Sicherung der Qualität.
Softwarequalität kann nur durch systematisches Vorgehen in der Entwicklung erreicht werden.

Es stellt sich nun die Frage: Was kann durch eine Prüfung erreicht werden?

Durch eine Prüfung werden die kollektiven und individuellen Schwächen des Prüflings aufgedeckt und dem Entwickler als Hilfestellung für die Verbesserung der als fehlerhaft entdeckten Komponenten zur Verfügung gestellt.

Durch eine zu Beginn der Entwicklung angekündigte Prüfung im Anschluss an den Entwicklungsprozess kann das Verhalten des Entwicklers dahingehend beeinflusst werden, dass er verstärkt bemüht ist, fehlerfreie Software zu erstellen und somit die festgelegte Qualität bei der Entwicklung zu erhöhen (indirekter Einfluss auf die Qualität).

Eine Prüfung stellt einen analytischen Ansatz zur Software--Qualitätssicherung dar. Organisatorische bzw. konstruktive Massnahmen sollen hier nicht betrachtet werden.
Mit einer Prüfung kann die Qualität eines Softwareproduktes nicht direkt erhöht werden, da der eigentliche Entwicklungsprozess bereits abgeschlossen ist. Das Hauptaugenmerk liegt auf der Sicherung der Qualität.

Bei der Softwareprüfung unterscheidet man zwischen folgenden Ansätzen:

- dynamische Prüfung durch Benutzung der Software
    - durch Tests
    - durch Leistungsmessungen
- statische Prüfung durch Untersuchung des Quellcodes / der Dokumentation
    - mit Hilfe von Rechnern
        - Prüfung gegen Regeln, ob gewisse Normen und Vorschriften eingehalten werden
        - Konsistenzprüfungen
        - Prüfung, ob alle Programmkomponenten verwendet werden, ob alle Variablen besetzt sind (mechanisch)
        - Quantitative Untersuchung
    - mechanische Untersuchung z.B. durch Zählen; Auswertung erfolgt durch den Menschen ohne Hilfe von Rechnern
- Review

## 2.1 Grundprinzipen des Softwaretest

**Menschen sehen ihre eigenen Fehler nicht**
Die Leute sind zielorientiert. Der Entwickler versucht zu zeigen, dass "es läuft", der Prüfer soll zeigen, dass es "nicht läuft". Daher ist es besser, dass nicht der Entwickler testet.

**Ein guter Test kann ein schlechtes Programm nicht retten**
Ein Test ist nie 100% effektiv. In der Praxis, ist die Zahl der gefundenen Fehler ziemlich konstant. D.h. wenn ein Programm zu Beginn viele Fehler enthält, so wird es auch nach dem Testen noch viele enthalten. Es ist also sehr schwierig schlechte Programme eingehend zu testen. Zu viele Inputvarianten und - kombinationen müssten berücksichtigt werden. Also ein Test, der viele Fehler entdeckt, sollte die Alarmglocke schellen lassen.

**Prüfen/Testen ist ein destruktiver Prozess**
Testen kann nie beweisen, dass ein Programm richtig ist. Aber der Versuch zu zeigen, dass das Programm nicht läuft, verbessert seine Verlässlichkeit.

**Prüfen/Testen sollte sich auf Spezialfälle konzentrieren (können)**
Schwierige Situationen, falscher Input etc. sollte versucht werden, da die Normalfälle (in der Regel) von den Entwicklern getestet werden.

**Selbst unter Zeitdruck sollte die Qualität gemessen werden**
Es sollten mind. so viele Tests gemacht werden, dass der/die Qualitätsverantwortliche entscheiden kann ob die verlangte Qualität erreicht ist.

**Prüfen/Testen braucht Wissen und Können**
Ein Prüfer muss wissen, was das Programm tun muss. Eigentlich könnte der Entwickler auf eine Art der beste Prüfer sein, aber leider ist er blind. Testen ist nicht einfach. Es bedarf einer systematischen und strukturierten Arbeitsweise. Die Testmethoden sind immer rudimentär und so müssen wir unsere Kenntnisse über die häufigsten Fehler benützen, um Tests zu entwickeln.

## 2.2 Dynamische Softwareprüfung durch Test

Unter Testen versteht man die Ausführung der Software mit dem Ziel, Fehler zu entdecken. Die durch Testen erkennbaren Fehler beschränken sich auf bestimmte Fehler in den Eigenschaften der Software, z.B. Fehler in der Funktionalität.

Bei Prüfungen wird nur festgestellt, ob Fehler vorhanden sind und wie sie sich gegebenenfalls zeigen. Es wird keine Lokalisierung der Fehler vorgenommen.
Allgemein gilt bei der Testdurchführung: Das Testen einer Software sollte von einer dritten (unabhängigen) Person durchgeführt werden, nicht vom Entwickler selbst, denn

- er kennt das Programm.
- er kennt Variablenbesetzung und die Eingabemodalitäten.
- er ist vorbelastet; er möchte in seinem Programm keine Fehler finden, da er davon überzeugt ist, keine Fehler gemacht zu haben.
- durch Termindruck ist oft nur oberflächliches Testen möglich, um verspätete Auslieferung zu vermeiden.

## 2.3 Testvorbereitung

### 2.3.1 Testvorbereitung

Voraussetzung beim Testen ist eine sorgfältige Planung: Festlegen, was und in welchem Umfang getestet wird. Überlegung über das Ziel des Tests und über das Vorgehen zur Erreichung des Ziels (Art und Umfang der durchzuführenden Tests festlegen) werden angestrengt. Zielsetzung des Testens ist das Auffinden von Fehlern; ein Test gilt als erfolglos, wenn kein Fehler gefunden wird. Die Testvorbereitung ist die arbeitsintensivste Phase. Sie stellt die Weichen für den gesamten Test. Die hier festgelegten Arbeitsschritte umfassen:

- Spezifikation und Auswahl der Testfälle
- die Bereitstellung der Testumgebung und der Testdaten
- Vorgehen für die Testausführung (Testvorschrift und Aufbau des Testprotokolls)
- Regeln für die Testauswertung

### 2.3.2 Spezifikation der Testfälle

Enthält folgende Angaben:

- Anfangszustand des Prüflings
- Werte aller Eingabedaten
- notwendige Bedienung
- erwartete Ausgabe

Hier kann man zwei Ansätze wählen:

- man orientiert sich an den Funktionen (Black--Box--Test)
- man orientiert sich an der inneren Struktur des zu prüfenden Programms (White--Box--Test, Strukturtest)

#### 2.3.2.1 Black-Box-Test:

Ausgangspunkt sind hier die funktionalen Anforderungen an das Programm. Die Erstellung der Testfälle richtet sich nach Eingaben, Ausgaben und funktionellen Verknüpfungen. Es gilt zu berücksichtigen:

- jede Funktion wird mindestens einmal ausgeführt,
- jedes Eingabedatum wird in mindestens einem Testfall verwendet,
- jedes Ausgabedatum wird in mindestens einem Testfall erzeugt (d.h. die verschiedenen Ausgabearten, nicht alle möglichen Werte).

Alle möglichen Eingabedaten überdecken einen Gültigkeitsbereich und die Funktion muss für den gesamten Gültigkeitsbereich das richtige Resultat liefern.

Erschöpfendes Testen ist nicht möglich. Die Güte des Tests, d.h. die Zahl der festgestellten Fehler, hängt ab von der Auswahl der Testfälle. Ziel ist es, mit einer möglichst kleinen Menge von Testfällen möglichst viele Fehler zu finden.

Wie wählt man aus der Menge der möglichen Testfälle eine sinnvolle Stichprobe aus? Man bildet Äquivalenzklassen, d.h. Gruppen/Bereiche von möglichen Eingabedaten, deren Werte zu einer gleichwertigen (äquivalenten) Klasse von Fehlern führen. Es wird immer nur ein Repräsentant pro Äquivalenzklasse als Testfall ausgewählt. Die Testfälle werden so gewählt,

dass die Eingabedaten Werte innerhalb und ausserhalb des spezifizierten Gültigkeitsbereichs annehmen.

Eine Funktion muss nicht nur einzeln, sondern auch in Kombination mit verschiedenen vorher und nachher ausgeführten Funktionen richtig sein.

#### 2.3.2.1.1 Äquivalenzklassentest

Ziel der Äquivalenzklassenbildung ist es, durch Bildung von Äquivalenzklassen eine hohe Fehlerentdeckungswahrscheinlichkeit mit einer minimalen Anzahl von Testfällen zu erreichen. Das Prinzip der Äquivalenzklassenbildung besteht darin, die gesamten Eingabedaten eines Programms in eine endliche Anzahl von Äquivalenzklassen zu unterteilen, so dass man annehmen kann, dass mit jedem beliebigen Repräsentanten einer Klasse die gleichen Fehler wie mit jedem anderen Repräsentanten dieser Klasse gefunden werden. Die Definition von Testfällen mit Hilfe der Äquivalenzklassenbildung erfolgt in folgenden Schritten:

- Analyse der Dateneingabeanforderungen, der Datenausgabeanforderungen und der Bedingungen gemäss den Spezifikationen
- Bestimmung der Äquivalenzklassen durch Einteilung der Wertebereiche für Ein- und Ausgabegrössen
- Bestimmung der Testfälle durch Werteauswahl für jede Klasse

Bei der Festlegung der Äquivalenzklassen werden zwei Gruppen von Äquivalenzklassen unterschieden:

- gültige Äquivalenzklassen
- ungültige Äquivalenzklassen

Bei gültigen Äquivalenzklassen werden erlaubte Eingabedaten, bei ungültigen Äquivalenzklassen fehlerhafte Eingabedaten ausgewählt. Die Bestimmung der Äquivalenzklassen ist bei Vorgabe der Spezifikationen hauptsächlich ein heuristischer Prozess.

Ein Beispiel für Grenzwert Analyse:

Die gültigen Werte eines Parameters liegen zwischen 5 und 5000. Es wird nun ein Testfall definiert, welcher verifiziert, dass:

- der Wert 4 als ungültig zurückgewiesen wird,
- der Wert 5 und 5000 als gültig akzeptiert, und
- der Wert 5001 als ungültig wieder zurückgewiesen wird.

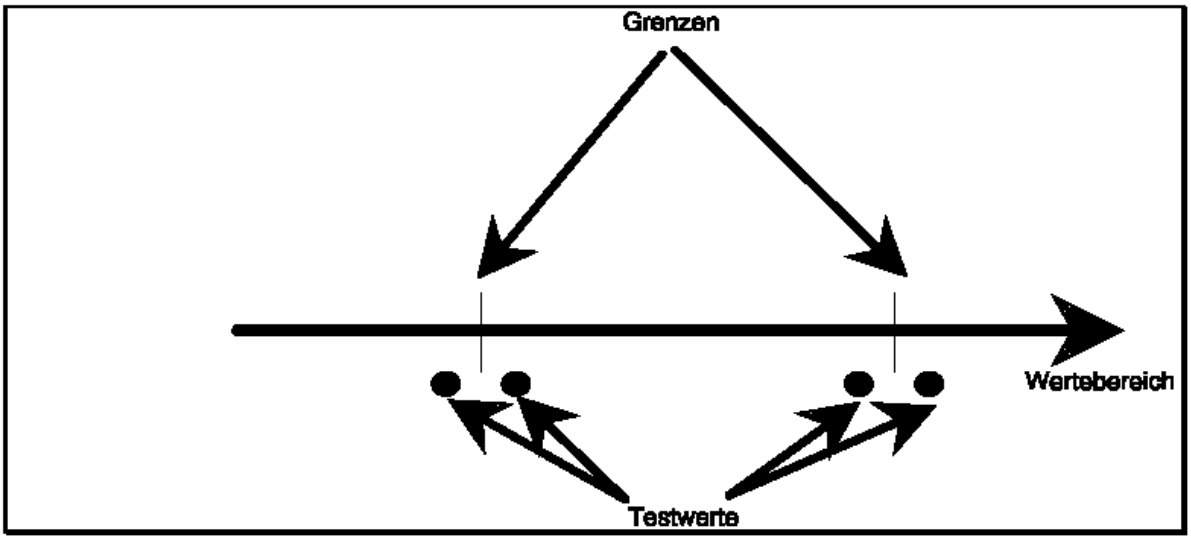


***Abbildung 2-1: Äquivalenzklassen Analyse***

#### 2.3.2.1.2 Grenzwertanalyse

Ziel der Grenzwertanalyse ist es, Testfälle zu definieren, mit denen Fehler im Zusammenhang mit der Behandlung der Grenzen von Wertebereichen aufgedeckt werden können.

Das Prinzip der Grenzwertanalyse besteht darin, die Grenzen von Wertebereichen bei der Definition von Testfällen zu berücksichtigen. Ausgangspunkt sind die mittels Äquivalenzklassenbildung ermittelten Äquivalenzklassen. Im Unterschied zur Äquivalenzklassenbildung wird kein beliebiger Repräsentant der Klasse als Testfall ausgewählt, sondern Repräsentanten an den Grenzen der Klassen. Die Grenzwertanalyse stellt somit eine Ergänzung des Testfallentwurfs gemäss Äquivalenzklassenbildung dar.

#### 2.3.2.1.3 Regression Testing

Das ist ein wiederholtes Testen, nachdem etwas korrigiert wurde. Sein Ziel- Überprüfung ob ein Bugfix:

- „fixed the bug“
- „fixed the bug (or not) and had a side effect (broke something previously working)“

Hier gilt: Benutze immer vollautomatisierten Regressionstest Die Regression Tools dominieren das Markt für automatisiertes Testen. Sie sparen Zeit, minimieren den Aufwand, verbessern die Qualität.

#### 2.3.2.2 White-Box-Test:

Voraussetzung hier ist die Kenntnis der inneren Struktur des Programms. Die Auswahl der Testfälle richtet sich nach Ablaufmöglichkeit des Programms. Die Bildung der Äquivalenzklassen geschieht so, dass die Daten einer Klasse die Ausführung gleicher Folgen von Befehlen bewirken. Hierzu ist die Ablaufstruktur sehr hilfreich: Knoten bilden die Anweisungen, die Kanten stellen die Übergänge dar. Bei der Definition der Äquivalenzklassen muss folgendes berücksichtigt werden:

- Anweisungsüberdeckung: Jeder Knoten des Graphs wird in mindestens einem Test durchlaufen.
- Zweigüberdeckung: Jeder Zweig (Verbindung zwischen zwei Knoten) muss mindestens einmal durchlaufen werden.
- Pfadüberdeckung: Jeder mögliche Pfad (Weg von Anfang bis Ende) muss in mindestens einem Test durchlaufen werden.

Die optimale Auswahl der Testfälle erfolgt unter Berücksichtigung der verschiedenen Äquivalenzklassen.

White-Box Test kann weiter in

- Statische und
- dynamische Tests unterteilt werden.

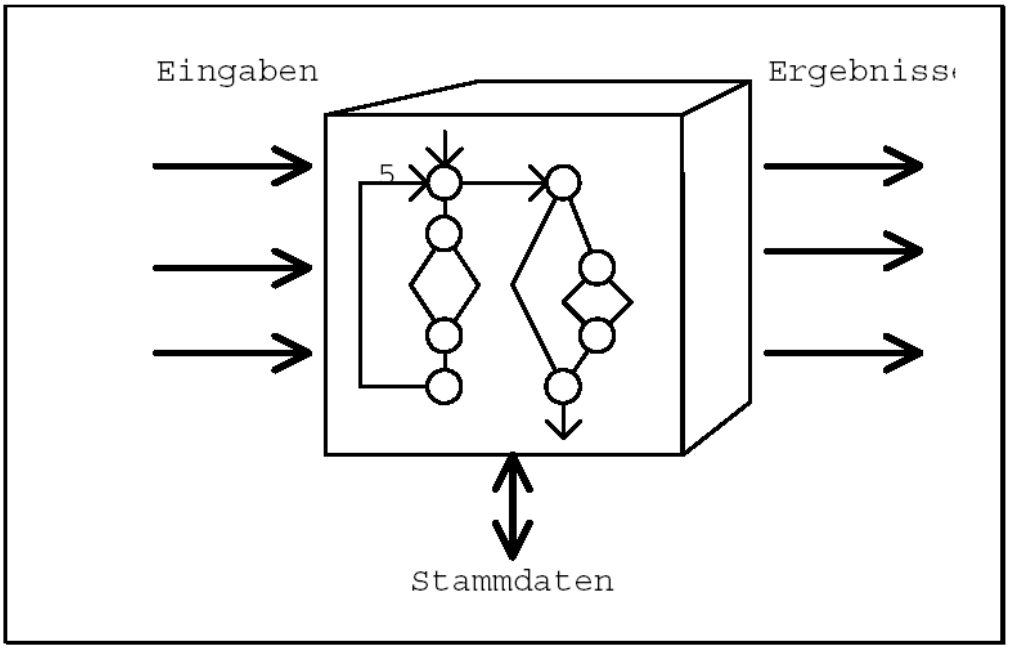


***Abbildung 2-2: White-Box-Test (Glass-Box)***

##### 2.3.2.2.1 Statische Tests

Bei statischen Tests wird die Struktur des Programms hinsichtlich Einhaltung von Programmier-Richtlinien und Komplexität geprüft. Das Programm wird nicht durchgeführt.

#### 2.3.2.2.2 Dynamische Tests

Bei dynamischen Tests richtet sich die Auswahl der Prüffälle nach den Ablaufmöglichkeiten des Programms. Prüffälle und -daten werden mit der Vorgabe, die bekannte interne Struktur zu testen, entwickelt. Das Programm kommt zur Durchführung. Prüffälle und -daten werden von den Designspezifikationen abgeleitet.

#### 2.3.2.2.3 Integrationstest

Um ein Mass der Produktqualität bestimmen zu können ist es zwingend notwendig, eine Metrik aufzustellen. Dies ist, wenn man zu sinnvollen Ergebnissen kommen will, ein beliebig komplexes Thema und soll hier nur am Rande erwähnt werden. Eine triviale Metrik wäre z.B. auf die Anzahl der Codezeilen (lines of code, LOC) bezogen ("Anzahl der Probleme auf 100 Zeilen Code"). Allerdings sind auch einfache Metriken vorstellbar, die durchhaus hilfreich sein können.

An dieser Stelle soll eine Übersicht über eine mögliche (und standardmässig in der Literatur beschriebene) Testorganisation gegeben werden.

Voraussetzung sei, dass alle Systemkomponenten einem Komponententest (Entwicklertest) unterzogen wurden und diesen erfolgreich durchlaufen haben. Die Überprüfung der Produktqualität erfolgt dann, zeitlich nacheinander, in drei Teststufen:

- Integrationstest
- Systemtest
- Abnahmetest

Den Integrationstest kann man auch als "White Box" Test bezeichnen, da alle Systemkomponenten und deren Beziehungen untereinander während des Tests sichtbar sind. Im Gegensatz dazu steht der "Black Box" Test, bei dem das System nur über die nach aussen geführten Schnittstellen, z.B. das Userinterface, betrachtet wird. Aufgabe des Integrationstests ist es, das fehlerfreie Zusammenwirken von Systemkomponenten und Teilmodulen zu überprüfen. Voraussetzung für den Integrationstest ist, dass jede einzelne Systemkomponente in einem Komponententest (Entwicklertest) für sich bereits geprüft ist. Stück für Stück werden die Komponenten zusammengeführt (integriert) und das Zusammenwirken getestet. Komponenten, die im Test noch nicht integriert sind, werden durch Testtreiber bzw. Platzhalter ersetzt. Die Integration erfolgt nach so genannten Integrationsstrategien, die im folgenden anhand weniger einfacher Beispiele kurz angerissen werden.

#### 2.3.2.2.3.1 inkrementelle Strategie

Die Systemkomponenten werden einzeln oder in sehr kleinen Gruppen integriert.

Vorteile:

- Integration bei Fertigstellung der Komponente möglich
- Testfälle können vergleichsweise einfach konstruiert werden, Testüberdeckung kann gewährleistet werden

Nachteil:

- Gegebenenfalls viele Testtreiber/Platzhalter nötig.

#### 2.3.2.2.3.2 testzielorientierte Strategie

Je nach vorher definierten Testzielen werden einzelne Komponenten zur Überprüfung dieser Ziele integriert und getestet. Z.B. können bei dem Testziel "performante Datenübertragung über das Intranet" zunächst die dafür vorgesehenen Komponenten integriert werden. Das restliche System gruppiert sich dann im Anschluss drumherum.

#### 2.3.2.2.3.3 geschäftsprozessorientierte Strategie

Komponenten werden gemäss ihrer Zugehörigkeit zu einzelnen Geschäftsprozessen (Abrechnung, Lager, usw.) integriert.

#### 2.3.2.2.3.4 funktionsorientierte Strategie

Die Testfälle werden so spezifiziert, dass sie auf funktionalen Merkmalen beruhen. Die Komponenten werden dementsprechend integriert und getestet.

#### 2.3.2.2.3.5 Big Bang Integration

Alle Komponenten werden gleichzeitig oder in grossen Gruppen integriert. Es werden zwar keine Testtreiber/Platzhalter benötigt, allerdings läuft der Test dabei unter Umständen unstrukturiert und unsystematisch ab. Fehlersuche und Konstruktion von Testfällen werden dadurch schwierig oder sogar unmöglich. Oftmals ist diese Strategie aber trotzdem das Mittel der Wahl, da in der Realität z.B. strikte Trennungen zwischen System- und Integrationstest aus organisatorischen Gründen nicht vorgenommen werden kann.

#### 2.3.2.2.3.6 Top Down Integration

Die im Schichtmodell bzw. in der Baumhierarchie am weitesten oben stehenden Komponenten werden zuerst integriert. Weiter unten stehende Komponenten werden durch Platzhalter simuliert.

Vorteile:

- aus Sicht des Benutzers entsteht eine frühe Verfügbarkeit des Systems, die Verzahnung von Entwurf und Implementierung ist möglich, d.h. erst wird die oberste Schicht definiert, implementiert und getestet, während Definition und Implementierung der unterlagerten Komponenten währenddessen weiterlaufen kann.

Nachteile:

- schwierige Implementierung der Platzhalter
- mit zunehmender Integrationstiefe ergeben sich Schwierigkeiten bei der Konstruktion von Testfällen für tiefer liegende Komponenten.
- Das Zusammenspiel der Systemsoftware, Hardware und Anwendungsebene wird erst sehr spät getestet.

#### 2.3.2.2.3.7 Bottom Up Integration

Es werden zunächst die Systemkomponenten integriert (benötigen keine Dienste anderer Komponenten). Neue Komponenten werden erst dann hinzugefügt, wenn alle dafür benötigten Komponenten bereits vorhanden und getestet sind.

Vorteile:

- keine Platzhalter nötig
- Testbedingungen leicht erstellbar
- Testergebnisse einfach zu interpretieren

Nachteile:

- lauffähiges Gesamtsystem ist erst sehr spät vorhanden
- Fehler in der Produktdefinition werden erst spät erkannt und können zu umfangreichen Änderungen führen.

Eine Zusammenfassung der beiden letztgenannten Strategien bieten die Verfahren "Outside-In" und "Inside-Out", die aus dem Versuch entstanden sind, die Vorteile der beiden Strategien zu vereinen und die Nachteile zu vermeiden.

### 2.3.2.2.4 Weitere Integrationsverfahren

Die im folgenden genannten Verfahren dienen der Überprüfung der Schnittstellen und des Zusammenspiels der einzelnen am System beteiligten Komponenten über ihre Schnittstellen. Es ist nicht Ziel, die Funktionen (das Funktionieren) einzelner Komponenten zu testen.

#### 2.3.2.2.4.1 dynamischer Integrationstest

Beim dynamischen Integrationstest werden die zu testenden Systeme mit konkreten Eingabewerten versehen und ausgeführt. Die Testfälle werden so ausgewählt, dass nur in der Integrationsphase nachweisbare Fehler aufgezeigt werden. Bei diesem Verfahren werden nur "Stichproben" - Tests gemacht, d.h. dieses Verfahren muss in Kombination mit anderen angewandt werden.

#### 2.3.2.2.4.2 kontrollflussorientierter Integrationstest

Der kontrollflussorientierte Integrationstest betrachtet die Aufrufbeziehungen der einzelnen Komponenten untereinander. D.h. alle Funktionen, die eine Komponente dem System zur Verfügung stellt (exportiert), müssen auch aufgerufen werden und jeder Aufruf muss mindestens einmal durchlaufen werden.

#### 2.3.2.2.4.3 datenflussorientierter Integrationstest

Bei diesem Verfahren werden die Übergabeparameter und die globalen Variablen betrachtet. Es ist der Datenfluss zu überprüfen:

- über globale Variablen und Parameter vor Aufruf einer importierten Operation und nach Eintritt in diese Operation
- vor Austritt aus der importierten Operation und nach Rückkehr zum Aufrufer

In der aufgerufenen Komponente muss jeder Parameter mindestens einmal lesend verwendet werden.

#### 2.3.2.2.4.4 funktionaler Integrationstest

Hier wird die spezifizierte Funktionalität auf korrektes Zusammenspiel geprüft. Eine Abweichung liegt vor, wenn:

- zu wenig Funktionalität geliefert wird, d.h. der Aufrufer eine Teilfunktion erwartet, die nicht vorhanden oder erreichbar ist.
- zu viel Funktionalität geliefert wird, d.h. eine unerwartete Teilfunktion ausgeführt wird, oder
- eine falsche Funktionalität geliefert wird.

#### 2.3.2.2.4.5 wertbezogener Integrationstest

Die Schnittstellen werden mit Extremwerten beaufschlagt, die ähnlich der Testfallermittlung in der Grenzwertanalyse gewonnen werden.

#### 2.3.2.2.4.6 strukturorientierter Integrationstest

Ein weiteres Verfahren ist der strukturorientierte Integrationstest, bei dem der Kontrollfluss bzw. der Datenfluss die Grundlage für Testfälle bieten können.

### 2.3.3 Weitere Testmöglichkeiten

#### 2.3.3.1 Schreibtischtest

Beim Schreibtischtest wird der Programmtext auf dem Papier durchgegangen und der Programmablauf nachvollzogen. Da der Implementierer oft Fehler in seinem eigenen Programm übersieht, sollte zu solch einem walk-through eine zweite Person hinzugezogen werden.

#### 2.3.3.2 Code Review

Eine Variante des Schreibtischtest ist ein Code-Review. Dieses wird vom Implementierer vorbereitet, indem die relevanten Quelltextteile für alle Teilnehmer (Grössenordnung etwa 5 bis 10) des Reviews ausgedruckt werden. Nachdem alle Teilnehmer den Programmtext gelesen haben, wird das Design und mögliche Alternativen diskutiert.

### 2.3.4 Testvorschrift

Es enthält alle für die Durchführung des Tests benötigten Angaben (u.a. die ausgewählten Testfälle, die zu Testsequenzen gruppiert werden.) Ein Inhaltsverzeichnis einer möglichen Testvorschrift könnte folgendermassen aussehen:

1. Einleitung
   1.1 Zweck des Tests
   1.2 Testumfang
   1.3 Referenzierte Unterlagen
2. Testumgebung
   2.1 Überblick
   2.2 Test Software/Hardware
   2.3 Testdaten, Testdatenbank
   2.4 Personalbedarf
3. Abnahmekriterien
   3.1 Kriterien für Erfolg und Abbruch
   3.2 Kriterien für eine Unterbrechung
   3.3 Voraussetzung für Wiederaufnahme
4. Testabschnitt 1
   4.1 Einleitung
   4.1.1 Zweck, Referenz zur Spezifikation
   4.1.2 Getestete Software--Einheiten
   4.1.3 Vorbereitungsarbeiten für Testabschnitt
   4.1.4 Aufräumarbeiten nach Testabschnitt
   4.2 Testsequenz 1-1
   4.2.1 Testfall 1-1-1: Eingabe, Anweisung, Soll-- und Istausgabe, Befund
   4.2.2 Testfall 1-1-2: Eingabe, Anweisung, Soll-- und Istausgabe, Befund
   ...
   4.3 Testsequenz 1-2
   4.n Ergebnis des Abschnitts 1
5. Testabschnitt 2
   ...

## 2.4 Testausführung

Alle in der Testvorschrift spezifizierten Testfälle werden ausgeführt und alle Ergebnisse in dem Testprotokoll aufgezeichnet. Die Durchführung der Tests sollte, falls keine allzu gravierenden Fehler auftreten, nicht unterbrochen werden. Festgestellte Fehler sollten nur notiert, aber nicht behoben werden. Das Ergebnis der Testausführung ist im Testprotokoll festzuhalten. Es sollte folgende Angaben enthalten:

- Bezeichnung Prüfling und Version
- verwendetes Testbed
- welche definierten Testfälle wurden ausgeführt
- Ergebnis der Prüfung

Das Testprotokoll kann mit der Testvorschrift kombiniert sein.

## 2.5 Testauswertung

Im Testprotokoll sind die Ergebnisse der Testausführung festgehalten. Diese Ergebnisse werden mit den in der Spezifikation definierten erwarteten Ergebnisse verglichen.
Gründliche Untersuchung der Testergebnisse: Es wird untersucht, ob das Programm genau das tut, was es soll, und nichts anderes. Der letzte Punkt und nichts anderes ist oft sehr schwer oder gar nicht prüfbar. Das Ergebnis der Testauswertung ist der Testbericht. Er enthält

- administrative Angaben,
- Verweise auf alle den Test betreffenden Dokumente,
- die Testzusammenfassung; präzise Identifizierung des Prüflings, des Testbeds, benutzte Testvorschrift, alle Anlagen und Teilnehmer; die Bewertung des Testteams muss von allen Teilnehmern unterschrieben sein.
- festgestellte Abweichungen (Problemmeldungen); diese dienen später zur Planung und Kontrolle der Fehlerbehandlung
- das Testprotokoll; Angaben, ob Ist-- und Soll--Resultate sich entsprechen; es kann zusammen mit der Testvorschrift realisiert sein, muss aber das Testergebnis enthalten,
- die Schlussbewertung

## 2.6 Statische Softwareprüfung: durch Reviews

Ein Review ist die Prüfung eines Dokuments oder eines Programmes gegenüber Vorgaben und gültigen Richtlinien mit dem Ziel, Fehler und Schwächen des Prüflings aufzuzeigen, aber auch positive Merkmale zu würdigen. Der Prüfling eines Reviews ist der abgeschlossene, für Menschen lesbare Teil von Software (ein einzelnes Dokument, ein Codemodul, ein Datenmodul). Diese Überprüfung wird von Gutachtern ohne Rechnerunterstützung durchgeführt, geleitet durch konkrete Fragestellungen. Die Gutachter erhalten zur Vorbereitung einen Fragenkatalog. Zur Prüfung werden Referenzunterlagen benötigt:

- Vorgabe (oder Spezifikation): Ausgangspunkt für Erstellung
- relevanten Richtlinien bei Erstellung
- zusätzlich: Fragenkatalog, der beim Review beantwortet werden soll.

Anhand der Entwurfsbeschreibung wird überprüft, ob im Lösungsansatz alle Anforderungsspezifikationen entsprechend den Richtlinien des Entwurfs umgesetzt worden sind.

### 2.6.1 Ablauf eines Reviews

1. Planung
   Bereits bei der Zusammenstellung des Arbeitspakets wird ein Review eingeplant; die zukünftigen Gutachter und der Leiter des Reviews werden benannt.
2. Initialisierung
   Nach Abschluss des Entwicklungsprozesses wird die Stabilität des Prüflings untersucht und gegebenenfalls die Einleitung eines Reviews veranlasst. Der Leiter des Reviews versorgt die Gutachter mit den notwendigen Informationen. Falls es für die Gutachter nötig ist, erfolgt vorab eine Präsentation des Prüflings, so dass sich die Gutachter mit ihm vertraut machen können. Der Leiter nimmt eine Aufgabenteilung unter den Gutachtern vor, wobei ein zu prüfender Aspekt von mindestens zwei Gutachtern geprüft werden muss. Es ist darauf zu achten, dass bei den Gutachtern nicht bei allen Aspekten dieselbe Gruppeneinteilung erfolgt.
3. Vorbereitung
   Der Leiter des Reviews stellt den Gutachtern die Rahmenbedingungen (Unterlagen) zur Verfügung. Die Gutachter durchleuchten den Prüfling nach den ihnen zugeteilten Aspekten. Sie notieren alle gefundenen Mängel (Fehler).

   In den Dokumenten werden Fehler der folgenden Art markiert:

   - Abweichungen von Richtlinien bzgl. Identifikation, Layout und Inhalt
   - Tipp-- und Rechtschreibfehler
   - fehlende Begriffe in Index und Abkürzungsverzeichnis
   - falsche Referenzen zu anderen Dokumenten oder innerhalb des Prüflings
   - falsche Verweise auf Bilder oder Tabellen

   In Programmen betreffen Markierungen folgende Fehlerarten:

   - Abweichung von Richtlinien bzgl. Identifikation, Layout, Kommentaren oder Verwendung von Sprachelementen
   - Abweichungen von Konventionen für Benennung
   - Tipp-- und Rechtschreibfehler in Kommentaren

   Der Autor muss den Gutachtern während der Vorbereitung zur Beantwortung von Fragen zur Verfügung stehen.

4. Review--Sitzung
   Eine Review--Sitzung ist auf zwei Stunden zu begrenzen (falls nötig mehrere Sitzungen abhalten). Der Prüfling ist in Bezug auf Vorgaben und Richtlinien zu bewerten. Die Meinung der Mehrheit der Gutacher entscheidet.

   Die Gutachter geben an, wie viel Zeit sie in die Vorbereitung investiert haben. Sie werden nach dem Gesamteindruck des Prüflings gefragt. Die Befunde der Gutachter werden erfasst (durch seitenweises Durchgehen des Dokuments).

   Die Gutachter sprechen lediglich eine Empfehlung aus, dass der Prüfling

   - akzeptiert werden soll.
   - akzeptiert werden soll, aber eine Überarbeitung notwendig ist.
   - nach einer Überarbeitung nochmals durch ein Folgereview geprüft werden soll.

   Beim Review wird nur eine Bestandsaufnahme gemacht, keine Lösungsmöglichkeiten diskutiert oder erarbeitet.

5. Review--Bericht
   Er besteht aus :

   1. einer Zusammenfassung; sie enthält Angaben wie Ort, Zeit, Datum der Review--Sitzung, Liste der verwendeten Unterlagen, Beschreibung des Prüflings, Name der Gutachter, etc., und
   2. einer Liste der Befunde; bezogen auf den Prüfling selbst (positve und negative) und den Referenzunterlagen (negative).

   An die Review--Sitzung kann sich eine sog. dritte Stunde anschliessen, in der die Gutachter über Lösungsideen der gefundenen Probleme diskutieren, oder eine Art Manöverkritik abhalten (was ist gut/schlecht gelaufen).

6. Nacharbeit
   Anhand des Reviewberichts entscheidet der Manager über die Freigabe des Prüflings oder über eine anzuordnende Überarbeitung. Sollte aufgrund einer Überarbeitung ein Folgereview durchzuführen sein, ist darauf zu achten, dass dieselbe Gutachterrunde beteiligt ist. Neue Gutachter müssen sich erst in die Materie einarbeiten, haben andere Ansichten etc., was die Durchführung eines Folgereviews verteuern und verzögern könnte.

### 2.6.2 Aufwand für Review

| | Dokumente | Code |
|---|---|---|
| Maximaler Umfang für *Review* | 50 Seiten | 20 Seiten |
| Zahl der Gutachter | 5 (+ Moderator + Autor) | 3 (+ Moderator + Autor) |
| *Review*-Vorbereitung relativ | 10 Seiten/Std. | 5 Seiten/Std. |
| Aufwand *Review*-Vorbereitung | 25 Stunden | 12 Stunden |
| Aufwand *Review*-Sitzung absolut | 14 Stunden | 10 Stunden |
| Summe *Review*-Aufwand | 5 Personentage | 3 Personentage |
| Erstellungsaufwand relativ | 2 Seiten/Tag | 1 Seite/Tag |
| Erstellungsaufwand absolut | 25 Personentage | 20 Personentage |
| *Review* zu Erstellungsaufwand | 20% | 15% |

***Abbildung 2-3: Review Aufwand tabellarisch***

# 3 Testebenen

| Test-ebene | Beschreibung | Objekt | Test-verantw. | Abnehmer |
|---|---|---|---|---|
| Unit test | Unit ist die Bezeichnung für die kleinste, in sich abgeschlossene Einheit. In der Regel ist das ein einzelnes Programm, eine Prozedur oder eine Klasse. Unit Tests sollen primär die interne Logik dieser Einheit überprüfen. Unit Tests werden in der Entwicklungsumgebung durchgeführt. | ein Modul | Entwickler und Projekt-Test-verant-wortliche(r) | Projekt-Test-verant-wortliche(r) |
| Ketten-test | Testen des Zusammenspiels der einzelnen Units (Programme, Klassen), die in einem Release (Spiralmodell) abgeliefert werden. Getestet werden die Schnittstellen zwischen diesen Objekten, externe Schnittstellen sowie Datenzugriffe sowie die Konsistenz. Die Tests sind auf das Austesten der Schnittstellen ausgerichtet und sollen die Funktionalität des entstehenden Softwaresystems überprüfen. | einzelne bis mehrere Module, Programme | Projekt-Testverant-wortliche(r) | Projekt-leiter |
| Integrati-onstest | Auf dieser Ebene werden keine einzelne Units getestet, sondern die fachlichen Aspekte der ganzen Anwendung. Getestet werden die Funktionen (GVG, Arbeitsablauf) einer Anwendung, ob sie den Spezifikationen bzw. dem Prototypen entsprechen. | An-wen-dung, Projekt | Projekt-Testverant-wortliche(r) (mit Fach-bereich) | Benutzer |
| System-test | Beim Systemtest handelt es sich um technische Tests, bei denen geprüft wird, wie sich die Anwendungen in einer produktionsnahen Umgebung verhalten. Getestet werden vor allem die Schnittstellen zwischen den Applikationen. Es werden auch Performance- und Volumentests durchgeführt. | Gesamt-system | System-testgruppe (mit Fach-bereich) | Benutzer und Produktion |

***Abbildung 3-1: Testebenen***

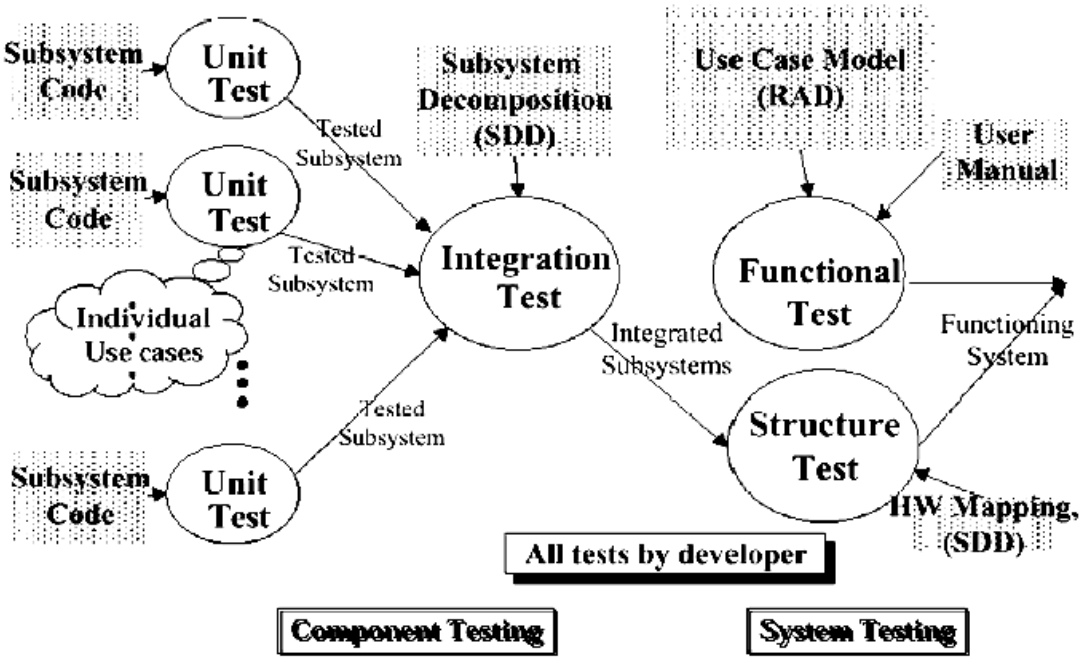


*Abbildung 3-2: Welche Testart für welche Ebene*

# 4 Testdokumentation nach IEEE 829

Testen ist ein normaler Bestandteil des Software-Entwicklungsprozesses. Während die Programmierer Sourcecode (und hoffentlich Dokumentation) erstellen, entwickeln die Tester die erforderlichen Tests und deren Dokumentation.

Falls im Projekt keine ausreichende Dokumentation der Software vorhanden ist, sind die Tester die ersten, die diesen Mangel zu spüren bekommen. Meistens beklagen sich die Tester auch ausgiebig über dieses Manko (nützt aber in der Regel nicht viel), daher sollte das Testteam seinerseits eine gute Dokumentation vorlegen, sonst macht man sich unglaubwürdig.

Viele Projekte haben einige gemeinsame Eigenschaften:

- Zeitdruck
- Probleme mit peripheren Systemen (Schnittstellen, Datenbanken)
- Probleme durch noch nicht testfertige Software
- Probleme durch sich ändernde Spezifikationen

Eine professionelle Testdokumentation unterstützt primär das Testteam, aber letztlich das gesamte Projekt.

- Zeitdruck: Murphy`s Gesetzen zufolge wird es immer letzte Änderungen unmittelbar vor dem Liefertermin geben. Man erwartet vom Testteam, dass es einen kompletten Regressionstest in einem unmöglich kurzen Zeitraum durchführen kann. Mit einer guten Testdokumentation kann der Testmanager jedoch feststellen, welche der zahlreichen Testfälle die wichtigsten sind, welche in dem zur Verfügung stehenden Zeitraum noch abgearbeitet werden können und welche nicht. Allerdings muss die Testdokumentation bereits fertig sein, bevor der Zeitdruck eintritt.

- Behinderungen im Test: Durch Probleme mit der gelieferten zu testenden Software oder mit peripheren Systemen kann es dazu kommen, dass Testfälle nur mit grosser Verzögerung oder gar nicht abgearbeitet werden können. Wenn man diese Aufwände dokumentiert, dann hat man genügend Fakten in der Hand, um Verbesserungen einzufordern. Dies würde dann immerhin dem nächsten Projekt zugute kommen.

## 4.1 Bestandteile der Testdokumentation

Der Standard IEEE/ANSI 829 definiert vier Arten von Testdokumenten sowie vier Arten von Berichten :

Die Testdokumente:

- Testplan
- Testspezifikation
- Testskript
- Testfall

und die Berichtsdokumente:

- Software-Übergabe
- Testprotokoll
- Problemmeldung
- Abschlussbericht

### 4.1.1 Testplan

Übersicht über alle Kapitel eines Testplans:

- Testplan ID
- Einführung
- Zu testende Komponenten
- Zu testende Funktionen
- Nicht zu testende Funktionen
- Vorgehensweise
- Pass / Fail Kriterien
- Produkte
- Testtätigkeiten
- Testumgebung
- Zuständigkeiten
- Personal
- Zeitplan
- Risiken und Risikomanagement
- Genehmigungen

**Testplan ID**

Jedes Dokument erhält eine eindeutige Bezeichnung, so dass man es später immer wieder finden kann.

**Einführung**

Hier ist eine kurze Zusammenfassung der zu testenden Hard- und Software und der zu testenden Funktionen anzugeben.

Die nachfolgend genannten Dokumente sind ebenfalls aufzuführen, wenn sie existieren:

- Projektautorisierung,
- Projektplan,
- Qualitätssicherungsplan,
- Konfigurationsmanagementplan,
- relevante Richtlinien,
- relevante Normen oder Standards.

Falls es mehrere Testpläne gibt, die hierarchisch geschachtelt sind, dann muss jeder untergeordnete Testplan seinen übergeordneten Testplan referenzieren.

**Zu testende Komponenten**

Hier sind sämtliche zu testenden Objekte einschliesslich der Versionsnummer bzw. Revisionsnummer aufzuführen. Ebenso ist anzugeben, auf welchem Medium die Software vorliegt, ob dies einen Einfluss auf Hardwareanforderungen hat und ob die Software vor Testbeginn in irgendeiner Weise transformiert werden muss (z.B. ob sie von Tape auf Diskette kopiert werden muss).

Falls die folgenden Dokumente vorhanden sind, sollten sie aufgeführt werden:

- Spezifikationsdokumente,
- Designdokumente,
- Anwenderhandbuch,
- Betriebshandbuch,
- Installationsunterlagen.

Bereits bekannte Fehler der zu testenden Objekte sind ebenfalls zu dokumentieren. Objekte, die ausdrücklich nicht getestet werden sollen, können auch aufgeführt werden.

**Zu testende Funktionen**

Alle zu testenden Funktionen und zu testende Funktionskombinationen sind aufzuführen.

Für jede zu testende Funktion bzw. Funktionskombination ist die dazugehörige Testspezifikation anzugeben.

**Nicht zu testende Funktionen**

Hier alle Funktionen und wichtige Funktionskombinationen angeben, die nicht getestet werden sollen.

**Vorgehen**

Es ist hier das allgemeine Testvorgehen zu beschreiben.

Für jede wichtige Gruppe von Funktionen oder Funktionskombinationen ist das Verfahren anzugeben, mit dem sichergestellt werden kann, dass diese Gruppen adäquat getestet werden.

Alle Aktivitäten, Techniken und Werkzeuge sind zu spezifizieren, mit denen die einzelnen Gruppen getestet werden sollen. Das Vorgehen sollte so detailliert beschrieben werden, dass es möglich ist, die Haupt-Testaufgaben zu identifizieren und eine Aufwandsschätzung abzugeben.

Die gewünschte minimale Güte ist zu definieren. Es sind die Techniken aufzuführen, mit denen die Güte der Tests bewertet werden soll (z.B. ein Code Coverage Prozentsatz). Alle sonstigen Ende-Kriterien sind zu definieren (z.B. eine Fehlerhäufigkeit). Man sollte auch angeben, mit welchen Techniken die Einhaltung der Spezifikationen durchgeführt wird.

Weiterhin sind alle bedeutenden Einschränkungen des Testvorgangs aufzuführen, wie Verfügbarkeit der Testobjekte, Ressourcen und Deadlines.

**Kriterien für erfolgreiche bzw. fehlgeschlagene Testläufe**

Das Kriterium angeben, mit dem man feststellt, ob ein Testlauf erfolgreich war oder nicht.

**Kriterien für Testunterbrechungen und -wiederaufnahme**

Es sind die Kriterien zu definieren, die eine Unterbrechung aller oder eines Teils der Aktivitäten dieses Testplans bedingen. Ausserdem ist zu definieren, welche Aktivitäten wiederholt werden müssen, wenn das Testen wiederaufgenommen wird.

**Produkte**

Hier sind die zu erstellenden Dokumente aufgeführt.

Die folgenden Dokumente sollten enthalten sein:

- Testplan,
- Testspezifikationen,
- Testskripte,
- Testfälle,
- Installationsberichte der Software,
- Testlogs,
- Testergebnisse,
- Gesamtbericht.

**Testtätigkeiten**

Alle Tätigkeiten sind zu spezifizieren, die zur Vorbereitung und zur Durchführung der Tests nötig sind.

Alle Abhängigkeiten zwischen den Tätigkeiten und alle erforderlichen Spezialkenntnisse sind zu identifizieren.

**Umgebung**

Es sind sowohl die notwendigen als auch die wünschenswerten Eigenschaften der Testumgebung anzugeben.

Die Spezifikation sollte enthalten: die physischen Charakteristiken einschliesslich Hardware, Kommunikations- und Systemsoftware, Betriebsart (z.B. stand-alone) und jede weitere Software oder Betriebsmittel, um die Tests zu unterstützen.

Weiterhin ist anzugeben, welche Sicherheitsstufen für die Testumgebung, Systemsoftware und proprietäre Komponenten wie Software, Daten und Hardware bereitzustellen ist. Wenn spezielle Test-Tools benötigt werden, sind diese aufzulisten. Alle weiteren Anliegen, wie z.B. Büroraum, Fachliteratur ebenso. Für alle Materialien, die im Moment noch nicht zur Verfügung stehen, ist eine Quelle anzugeben.

**Verantwortlichkeiten**

Es sind die verantwortlichen Gruppen für Management, Design, Vorbereitung, Ausführung, Beglaubigung, Überprüfung und Problemlösung zu benennen.

**Personal**

Hier ist der Personalbedarf für die Testaufgabe nach erforderlichen Skills geordnet anzugeben. Ggf. kann auch optionales Training vorgesehen werden.

**Zeitplan**

Im Zeitplan müssen sowohl die Test Milestones aus der Software Projektplanung als auch die Lieferungen der zu testenden Objekte festgehalten werden. Auch zusätzliche Milestones können aufgeführt werden. Für jede Testaufgabe ist der Zeitbedarf zu schätzen sowie ein Zeitplan anzugeben. Für jede Ressource (Geräte, Tools, Personal) sollte man den Betriebs- bzw. Bedarfszeitrahmen erstellen.

**Risiken und Risikomanagement**

Es sind alle mit hohem Risiko behafteten Annahmen des Testplans aufzuführen.

Für jede Annahme ist eine Notfallmassnahme zu definieren (z.B. verspätete Lieferung von zu testenden Objekten könnte Nachtschichten erfordern).

**Genehmigungen**

Namen und Titel aller Personen, die diesen Testplan genehmigen müssen. Man sollte nicht darauf verzichten, eine gedruckte Kopie des Testplans unterschreiben zu lassen.

### 4.1.2 Die Testspezifikation

Dieses Dokument enthält allgemeine Überlegungen, wie eine Komponente getestet werden sollte. Es eignet sich gut für eine Prüfung durch den Endanwender, um zu ermitteln, ob alle relevanten Bereiche durch Tests abgedeckt werden und ob die Tests fachlich korrekt geplant werden. Man sollte die Testspezifikationen auch den Entwicklern vorlegen und deren Anmerkungen einarbeiten.

Eine Testspezifikation sollte die folgenden Abschnitte enthalten:

- Testspezifikation ID
- Zu testende Funktionen
- Testverfahren
- Testskripte und Testfälle
- Pass / Fail Kriterien

**Testspezifikation ID**

Hier eine eindeutige Bezeichnung für die Testspezifikation angeben.

Falls diese Testspezifikation aus einem Testplan referenziert wird, ist die eindeutige Bezeichnung des Testplans ebenfalls anzugeben.

**Zu testende Funktionen**

Es sind hier die zu testenden Objekte zu identifizieren.

Ausserdem sollen alle Funktionen und Kombinationen von Funktionen aufgeführt werden, um die es in dieser Testspezifikation geht.

Für jede Funktion oder Kombination von Funktionen sollte es eine Referenz zu der dazugehörigen Anforderung in den Spezifikations- oder Designunterlagen geben.

**Testverfahren**

Hier sind Erweiterungen des im Testplan definierten allgemeinen Testverfahrens zu dokumentieren. Falls spezielle Test-Techniken verwendet werden sollen, sind diese hier aufzulisten. Auch die Art und Weise, wie die Testergebnisse analysiert werden sollen, sollte gewählt werden (z.B. diff-Programme oder visuelle Inspektion).

Falls es eine Untersuchung gegeben hat, welche Testfälle man auswählen sollte, dann ist das Ergebnis hier zu dokumentieren. Beispielsweise könnte man Bedingungen angeben, aus denen man eine Fehlertoleranz bestimmen kann (um z.B. gültige von ungültigen Eingaben unterscheiden zu können).

Alle gemeinsamen Attribute aller Testfälle sind hier zusammenzufassen. Das könnte Beschränkungen des Inputs beinhalten, die für alle Testfälle gelten, gleiche erforderliche Umgebungsparameter, gleiche Behandlung, gleiche Abhängigkeiten von anderen Testfällen.

**Testskripte und Testfälle**

Alle Bezeichnungen und eine kurze Beschreibung aller zu dieser Testspezifikation gehörenden Testfälle ist tabellarisch aufzulisten. Ein Testfall kann auch in mehreren Testspezifikationen aufgeführt werden.

Ausserdem sind noch alle Testskripte zu nennen, die zu dieser Testspezifikation gehören.

**Pass / Fail Kriterien**

Hier sind die Kriterien zu nennen, anhand derer man entscheiden kann, ob eine Funktion oder Kombination von Funktionen einen Test bestanden hat oder nicht.

### 4.1.3 Das Testskript

Das Testskript definiert, wie Testfälle auszuführen sind. Es enthält detaillierte Informationen, welche Voraussetzungen vor dem Start des Testfalls geschaffen werden müssen und wie der Testfall manuell vom Tester auszuführen ist.

Ein Testskript sollte so detailliert geschrieben werden, das man es auch ohne allzu viel Hintergrundwissen verwenden kann. Man kann dann in einer heissen Projektphase leichter weitere Tester hinzunehmen. Auch der normale Testbetrieb profitiert von detaillierten Testskripts, da die Wahrscheinlichkeit von Bedienungsfehlern abnimmt.

Es ist meistens für mehrere Testfälle gültig, die den gleichen Ablauf mit unterschiedlichen

Inputdaten ausführen. Deswegen enthält das Testskript auch nicht die Eingabedaten selbst, sondern verweist auf die entsprechenden Testfälle.

In den Testfällen wiederum wird nicht auf die Ausführung eingegangen - diese wäre ja für alle Testfälle gleich und man würde sich unnötig wiederholen -, sondern lediglich auf das dazugehörige Testskript verwiesen.

Ein Testskript hat folgende Abschnitte:

- Testskript ID
- Testziel
- Spezielle Anforderungen
- Einzelschritte

**Testskript ID**

Jedes Testskript besitzt eine eindeutige Referenz.

Man sollte ebenfalls die Bezeichnung für die dazugehörige Testspezifikation angeben.

**Testziel**

Hier ist Sinn und Zweck dieses Testskripts zu beschreiben.

Eine Liste aller Testfälle, die zu diesem Testskript gehören, ist an dieser Stelle sehr nützlich, ebenso eine Referenz auf die Dokumentation des Testobjekts.

**Spezielle Anforderungen**

Hier sind alle besonderen Anforderungen zu beschreiben, die für die Ausführung dieses Testskripts erforderlich sind.

Das können bestimmte Vorbereitungen sein, besondere Kenntnisse der Tester oder besondere Anforderungen an die Testumgebung.

**Einzelschritte**

Auf den ersten Blick scheint dieser Abschnitt viel zu detailliert, insbesondere die Angaben über Anhalten, Neustart usw. des Testfalls. Dies hat aber einen sehr praktischen Hintergrund:

Bestimmte Testfälle (z.B. Last- oder Massentests) sind sehr aufwendig und mit hohen Kosten verbunden. Aber ebenso wie ein Stück Programmcode hat auch ein Testfall eine hohe Wahrscheinlichkeit, beim ersten Aufruf nicht zu funktionieren. Für solche Testfälle ist es daher durchaus sinnvoll, bereits im Vorfeld zu klären, was zu tun ist, falls irgendwas schief geht.

Andere Testfälle sind mit hohen Risiken verbunden, z.B. Steuerung von Industrieanlagen. Hier ist es absolut notwendig, jederzeit den Test abbrechen und die Anlage anderweitig steuern zu können. Auch ein Neustart eines Testfalles ist in diesem Zusammenhang nicht so einfach wie beim Test einer Webseite (es gibt hier aus der Industrie diverse Berichte über spektakuläre Unfälle).

**Log**
Logs sind alle speziellen Methoden oder Formate zu beschreiben, um die Ergebnisse der Testläufe, die Zwischenfälle und sonstige wichtige Ereignisse aufzunehmen. Siehe auch

Dokument Testbericht und Fehlermeldung.

**Vorbereitung**
Es ist zu beschreiben, was zu tun ist, um die Ausführung des Testskripts vorzubereiten

**Start**
Es ist zu beschreiben, was zu tun ist, um das Testskript zu starten.

**Ausführung**
Es ist zu beschreiben, was während der Ausführung des Testskripts zu tun ist.

**Beobachtung**
Es ist zu beschreiben, wie Messungen und Testergebnisse gewonnen werden sollen (z.B. die Antwortzeit eines Remote Terminals ist mit einem Netzwerk Simulator zu testen).

**Abbruch**
Es ist zu beschreiben, was zu tun ist, um wegen unvorhergesehenen Ereignissen den Test abzubrechen.

**Neustart**
Es sind alle Startpunkte und das jeweilige Vorgehen zu beschreiben, an denen der Test wieder aufgenommen werden kann.

**Stopp**
Es ist zu beschreiben, was zu tun ist, um das Testskript ordnungsgemäss anzuhalten.

**Aufräumen**
Es sind die Aufräumarbeiten zu beschreiben, um nach den Tests den ursprünglichen Zustand wiederherzustellen.

**Unvorhergesehenes**
Was ist bei unvorhergesehenen Vorkommnissen während der Testläufe zu tun?

### 4.1.4 Der Testfall

Ein Testfall ist eine Kombination von Eingabedaten, Bedingungen und erwarteten Ausgaben, die einem bestimmten Zweck dienen. Man prüft z.B., ob Vorgaben in einem Spezifikationsdokument eingehalten werden oder ob der Programmablauf tatsächlich dem erwarteten Pfad entspricht.

Der Testfall enthält keine Angaben zur Bedienung oder Ausführung des Testfalls. Diese findet man immer im dazugehörigen Testskript, auf das der Testfall verweist.

Die folgenden Abschnitte sind Bestandteil eines Testfalls:

- Testfall ID
- Zu testende Funktionen
- Eingaben
- Ausgaben
- Umgebung
- Besonderheiten
- Abhängigkeiten

**Testfall ID**

Jeder Testfall ist mit einer eindeutigen Referenz zu versehen.

**Zu testende Funktionen**

Hier sind alle Testobjekte und Funktionen zu beschreiben, die von diesem Testfall ausgeführt werden. Testobjekte können z.B. separate Softwaremodule oder einzelne Webseiten sein. Sofern vorhanden, sollte man die folgenden Dokumente referenzieren:

- Spezifikationsdokumente,
- Designdokumente,
- Benutzerhandbuch,
- Betriebshandbuch,
- Installationshandbuch.

**Eingaben**

Hier sind alle Eingaben zu spezifizieren, die der Tester braucht, um den Testfall auszuführen.

Die Eingaben können sowohl als Wert angegeben werden (ggf. mit Toleranzen) als auch als Name, falls es sich um konstante Tabellen oder um Dateien handelt. Ausserdem sind alle betroffenen Datenbanken, Dateien, Terminal Meldungen, nicht flüchtige Speicherbereiche und vom Betriebssystem übergebene Werte anzugeben.

**Ausgaben**

Hier sind alle Ausgaben anzugeben, die aufgrund der Tests zu erwarten sind.

Für jede erwartete Ausgabe muss der genaue erwartete Wert (ggf. mit Toleranzen) angegeben werden. Falls erforderlich, sind auch weitere erwartete Eigenschaften der Ausgabe (z.B. Antwortzeit) anzugeben.

**Umgebung**

Dieser Abschnitt definiert die benötigte Testumgebung für diesen Testfall.

**Besonderheiten**

Manche Testfälle erfordern spezielle Ausführungsvorschriften, die nur für diesen einen Testfall gelten. In diesem Fall ist es nicht zweckmässig, die Ausführungsvorschrift im Testskript unterzubringen. Stattdessen wird sie an dieser Stelle definiert.

**Abhängigkeiten**

Falls vor diesem Testfall andere Testfälle ausgeführt werden müssen, sind diese Testfälle hier zu referenzieren. Man sollte ausserdem kurz beschreiben, worin die Abhängigkeit besteht.

# 5 Informationen über Dokument

| | |
|---|---|
| *Version* | *1.0* |
| *Letzte Änderung* | *31. Jan. 2003* |
| *Copyright* | *Roland Lenz* |
| *Homepage* | *http://www.2cool4u.ch/* |

| ***Historie*** | ***Version*** | ***Datum*** |
|---|---|---|
| *Dokument erstellt & übernommen* | *1.0* | *31. Jan. 2003* |