Test, Revision, Qualitätssicherung

vorgelegt von:

Christian Rolfes

Matrikelnummer 8704050
Diplom Informatik
rolfes.christian@mail.uni-oldenburg.de
Datum: 26.10.2007

Inhaltsverzeichnis

1 Einleitung

In dieser Seminararbeit geht es um Qualitätssicherung, Tests und Review.

1.1 Motivation

Seit vielen Jahren zeigt sich im Informationsbereich eine deutliche Steigerung des Kostenanteils für Software im Vergleich zu dem Anteil an den Kosten durch die Hardware. Des Weiteren haben Softwareprodukte eine immer höhere Lebensdauer. Obwohl viele Produkte noch nicht marktreif sind, werden sie schon ausgeliefert. Dadurch kommt es zu hohen Kosten für Wartungsarbeiten, die nachträglich Fehler korrigieren sollen. Die Fehler sind während der Entwicklung entstanden und oft die Folge von mangelnder Qualität. Sind die fehlerhaften Softwaresysteme dann auch noch schlecht strukturiert, hoch komplex und mangelhaft dokumentiert, steigen die Kosten für die Wartung dieser Fehler noch mal um das vielfache, da die Lokalisierung und die Beseitigung nur langsam vonstatten geht.
Um die zeitintensive Wartungsarbeit und dem damit verbundenen Problem für das Projekt, dass durch die Fehlerbeseitigung Projektmitglieder an anderer Stelle fehlen, zu verhindern, müssen Methoden der Qualitätssicherung genutzt werden. Diese können sowohl konstruktiv als auch analytisch sein. Im Rahmen der konstruktiven Qualitätssicherung werden Methoden verwendet, die vorbeugend agieren sollen. Da aber im Vorfeld nicht alle Fehler vermieden werden können, sollen analytische Methoden entstandene Mängel aufdecken.

1.2 Aufbau der Arbeit

Diese Arbeit ist folgendermaßen aufgebaut: Zu Beginn wird kurz darauf eingegangen, welche Folgen Softwarefehler haben können und was wesentliche Qualitätsmerkmale eines Softwareprodukts sind. Anschließend werden zuerst einige Methoden der konstruktiven Qualitätssicherung vorgestellt, unter anderem Test- und Prozessmanagement. Da die einzelnen Prinzipien des Software Engineerings nicht Gegenstand dieser Ausarbeitung sind, wird auf diese nur kurz eingegangen. Dann werden im Rahmen der analytischen Qualitätssicherung dynamische Verfahren wie Tests und statische Verfahren, insbesondere der Review, erläutert. Im Fazit werden dann die wichtigsten Punkte noch einmal abschließend zusammengefasst.

2 Grundlagen

In diesem Kapitel wird kurz beschrieben welche fatalen Folgen Softwarefehler haben können und was die wesentlichen Qualitätsmerkmale eines Softwareprodukts sind.

2.1 Folgen von Softwarefehlern

In der Geschichte der Softwareentwicklung kam es schon des Öfteren zu Fehlern mit kleineren aber auch schwerwiegenden Folgen.

- 1962 zerstörte sich die Trägerrakete Mariner 1. Die Ursache war eine fehlerhafte Wertzuweisung durch das Setzen eines Punktes statt eines Kommas, daher kam es zu einer Abweichung der Trägerrakete von der eigentlichen Flugbahn, wodurch diese sich selbst zerstörte. [Has04]

- 1988 erhielten Studenten der NC State University falsch addressierte Rechnungen aufgrund des Verteilerprogramms. [Gra04]

- Im Oktober 1987 gab es eine Weltwirtschaftskrise, ausgelöst durch den Zusammenbruch der Börsen. Als Ursache identifizierte man den automatischen Aktienhandel durch Computerprogramme. Viele Banken verzichteten in der Folgezeit auf diese Automatisierung, einige fuhren aber wie bisher fort. [Gra04]

- 1996 sprengte sich die Rakete vom Typ Ariane 5 selbst. Die vier zu transportierenden Cluster-Satelliten wurden ebenfalls zerstört. Der Grund war ein extremer Kurswechsel und die daraus resultierenden aerodynamischen Belastungen. Als Ursache stellte sich heraus, dass Softwarekomponenten des älteren Typs Ariane 4 unverändert übernommen wurden und diese nicht den nötigen Anforderungen entsprachen. Alle neuen Komponenten wurden ausführlich getestet. Aus Kostengründen blieb ein gemeinsamer Test der gesamten Steuerungssysteme aus. Der Schaden belief sich auf ca. 500 Millionen Euro.[Inc07a, Has04]

2.2 Qualitätsmerkmale von Software

Qualitätssicherung soll wie der Name schon sagt, die Qualität eines Softwareprodukts sichern. Diese Qualität lässt sich in verschiedene Punkte unterteilen.
Die externen Qualitätseigenschaften spiegeln die Anforderungen der Benutzer an ein Softwareprodukt wieder:

- Benutzbarkeit: Das Produkt sollte möglichst einfach zu bedienen und der Umgang mit diesem auch gut erlernbar sein. Des Weiteren sollte das Programm auch sehr robust gegenüber Fehleingaben sein.

- Verlässlichkeit: Für die meisten wirtschaftlichen Programme gibt es wahrscheinlich nichts schlimmeres, wenn ein Service nicht mehr funktioniert, nicht verfügbar ist oder aufgrund mangelnder Sicherheit Daten von Unbefugten ausgelesen werden können.

- Brauchbarkeit: Auch sollte ein Programm eine angemessene Performance aufweisen und sich gut pflegen lassen.

Interne Qualitätseigenschaften sind für die Entwickler gedacht, wie man in der folgenden Auflistung sieht, sind diese Eigenschaften für den Umgang während der Entwicklung und der Revision:

- Zuverlässigkeit: Ein Programm sollte korrekt, d.h. fehlerfrei arbeiten.
- Wartbarkeit: Jeder, der schon einmal programmiert hat, kennt das Problem, wenn man sich Quellcode anschaut, den man Jahre vorher einmal geschrieben hat. Es dauert eine gewisse Zeit, bis man sich in den Ablauf des Programms wieder hineingearbeitet hat. Um dies zu vereinfachen, sollte ein Programm gut strukturiert und dokumentiert sein. Es sollte vor allem auch so strukturiert sein, dass es leicht zu ändern, zu erweitern und zu testen ist. Des Weiteren sollte darauf geachtet werden, dass es korrekt arbeitet.
- Effizenz: Obwohl die Kosten für Hardware immer mehr sinken, sollte trotzdem dafür gesorgt werden, dass ein Programm Speicher effizent nutzt. Laufzeiteffizenz.

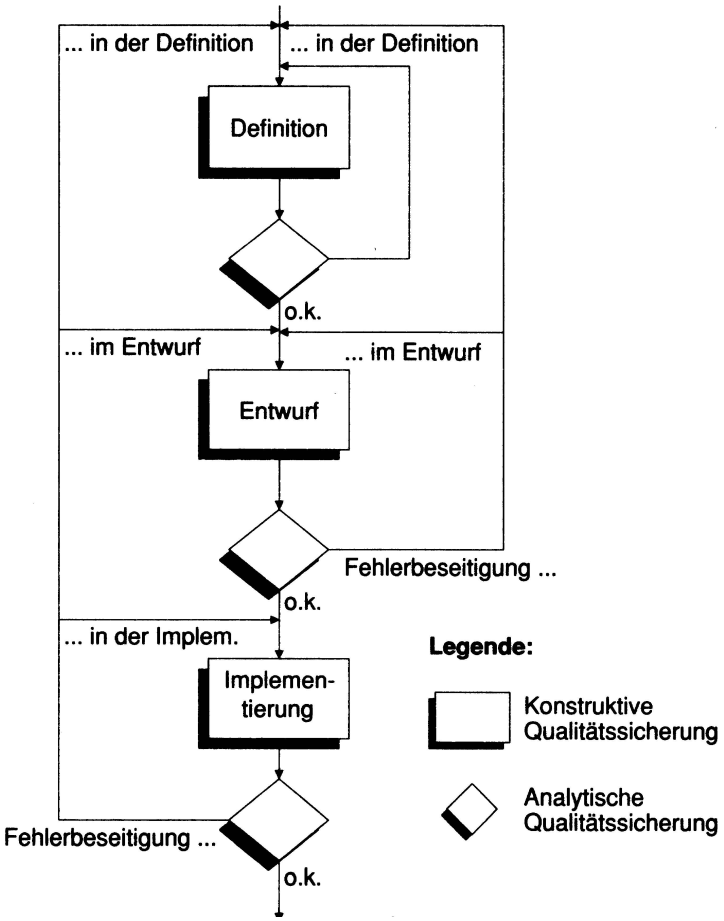

Abbildung 1: konstruktive und analytische Qualitätssicherung [Lig90]

3 Konstruktive Qualitätssicherung

Im Rahmen der konstruktiven Qualitätssicherung sollen vorbeugende (vgl. Abb. 1) Maßnahmen ergriffen werden, um das Entstehen von Fehlern zu verhindern. Dies bedeutet, dass Prinzipien des Software Engineerings bei der Entwicklung durchzuführen sind.

3.1 Allgemeines

Vor dem eigentlichen Beginn eines Projekts festgelegt sollte festgelegt werden, nach welchen Konzepten, Modellen das Projekt durchgeführt wird. Ebenso muss vorher feststehen, welche Technologien und Strategien verwendet werden. Es muss sichergestellt sein, welches Mitglied welche Aufgaben und Kompetenzen hat, welche Ressourcen zur Verfügung stehen, welche Richtlinien, Standards und Checklisten bestehen. Da der Status dieser Bereiche aber nicht immer überprüfbar ist, wurde 1986 im Auftrag des US-Verteidigungsministerium ein Reifegradmodell vom Software Engineering Institut SEI entwickelt. Daher wird im Folgenden näher auf das Reifegradmodell CMM bzw. dessen Erweiterung CMMI eingegangen. [Com07, Pre05, Lig90]

3.2 Capability Maturity Model Integration

Das Capability Maturity Model Integration, CMMI, stellt eine Erweiterung des 1986 entwickelten Capability Maturity Model, CMM, dar. Es kann verwendet werden, um die positiven und negativen Seiten einer Produktentwicklung zu analysieren, diese zu verbessern und sie in eine optimale Reihenfolge zu bringen. Dabei bedient man sich vor allem an der Einteilung des Prozesses in Reifegrade.

1. **Initial:**
   Auf dieser Stufe steht man zu Anfang automatisch, denn an das Projekt sind noch keine Anforderungen gestellt. Es existiert noch keine Organisation und es wurde auch noch keine Vorgehensweise bei der Erstellung und Wartung von Software definiert. Des Weiteren sind noch keine Werkzeuge bereitgestellt oder werden nur selten genutzt. Änderungen, die an der bestehenden Software durchgeführt wurden, werden nicht dokumentiert und werden auch nicht kontrolliert.

2. **Managed:**
   Diese Stufe besagt, dass die grundlegende Projektmanagement-Aufgaben durchgeführt werden. Es besteht eine einfache Projektverwaltung und Termine werden laufend überwacht. So sind die Abläufe von Prozessen ungefähr vorhersagbar.

3. **Defined:**
   Hat man diese Stufe erreicht, ist ein gültiger Software-Prozess definiert und eingeführt worden. Manager und Entwickler haben ihre Rollen erhalten und agieren in diesen und den dazugehörigen Verantwortungsbereichen.

4. **Quantitatively**
   Die Qualität der Produkte und der Prozesse wird erfaßt und gemessen. Aufgrund dieser Informationen werden Entscheidungsträger maßgeblich unterstützt.

5. **Optimizing**
   Innovative Ideen und Technologien werden genutzt. Zudem ist ein kontinuierlicher Verbesserungsprozess zu beobachten.

In Europa wurde auf das CMM aufbauend 1998 ein ähnliches System entwickelt. Dies war das SPICE, Software Process Improvement and Capability Determination, Projekt. Bei diesem sollen die Fähigkeitstufen, die Entwicklungs- und Wartungsmethoden untersucht und analysiert werden.
[Rie97, Inc07b, Inc07d, MF05]

4 Analytische Qualitätssicherung

4.1 dynamische Verfahren, Tests

Dynamische Verfahren sind Prüfverfahren. Diese sind Tests, welche Programme ausführen, mit deren Hilfe Fehlfunktionen im eigentlichen Softwareprodukt aufgedeckt werden sollen.

4.1.1 Allgemeiner Ablauf von Tests

Tests bestehen im Allgemeinen aus drei Phasen, zuerst wird der Test vorbereitet, dann ausgeführt und zum Schluss ausgewertet.
Im Rahmen der Vorbereitung werden anfangs die Testfälle ausgewählt. Dies ist insofern wichtig, da die Qualität des Tests von der Auswahl der Testfälle abhängt. Allerdings wird versucht, mit möglichst wenigen Testfällen möglichst viele Fehler aufzudecken. Bei der Spezifikation eines Testfalls muss entschieden werden, was getestet werden soll. Entweder führt man Black-Box oder White-Box Tests durch. Beim Black-Box-Testen werden die Features eines Programms geprüft. Dies wären unter anderem die Reaktionszeit, die Sicherheit oder die Effizienz der Datenbewältigung. White-Box-Tests dienen dagegen zum Testen der Struktur eines Programms. Diese beiden Verfahren werden später noch näher erläutert. Sind die Testfälle ausgewählt, muss sichergestellt werden, dass die Tests unter den gewollten Bedingungen auch durchgeführt werden können. Dies bedeutet, dass für die Tests alle benötigten Daten und Ressourcen bereitgestellt werden müssen. Anschließend muss noch das Testvorgehen bestimmt werden. Das Ziel ist dabei, dass der gesamte Test möglichst wenig Zeit in Anspruch nimmt. Daher müssen die Testfälle in einer optimalen Reihenfolge eingeplant werden, denn die zeitlichen Aufwendungen, die für das Umstellen der Anfangsbedingungen für die einzelnen Testfälle benötigt werden, sind nicht zu unterschätzen.
Nachdem die Testvorbereitungen abgeschlossen sind, kann mit der Ausführung aller Testfälle begonnen werden. Man sollte aber nicht den Tests abbrechen, sobald ein Fehler gefunden wurde oder ein Teilergebnis von dem erwarteten Ergbnis abweicht. Denn nur der ganze Test ermöglicht eine Bestandsaufnahme über die geprüften Komponenten und somit ein Abschätzen des Aufwands, der noch zu tätigen ist. Wird bei einem Fehler sofort abgebrochen und dieser korrigiert, wird zudem das Auffinden von Entwurfsmängeln erschwert. Des Weiteren ist es auch effizienter, das Ausführen des Tests, die Fehlersuche und die Korrektur zu trennen, da so der Auf- und Abbau der Anfangsbedingungen für die Testfälle reduziert wird. Da die einzelnen Phasen dadurch länger sind, gibt es den Mitarbeitern die Möglichkeit, sich auf die jeweilige Phase besser einzustellen und zu gewöhnen. Sie müssen weniger oft umdenken, was eine Menge Arbeit und somit Zeit erspart.
Ist die Durchführung des Tests beendet, müssen die erwarteten Ergebnisse mit denen des Tests verglichen und ausgewertet werden. Der Test ist erfolgreich, wenn noch nicht entdeckte Fehler aufgedeckt werden. Andererseits ist das Programm erfolgreich, wenn es genau das gemacht hat, was es tun soll, und nichts anderes.[KF91]

4.1.2 Black-Box

Der Black Box Test ist ein Systemtest, welcher die Funktionsweise der Software als „ganzes“ testen soll und daher in der Regel am Ende des Projekts durchgeführt wird. Die Abbildung verdeutlicht bereits, dass der Black Box Test ohne Kenntnis der Implementierung durchgeführt wird und lediglich das Verhalten des Systems auf bestimmte Eingaben untersucht werden soll. Die genaue Beschaffenheit des Programms ist irrelevant und wird als Black

Box behandelt. Für das Finden geeigneter Testfälle wird entsprechend die gewünschte

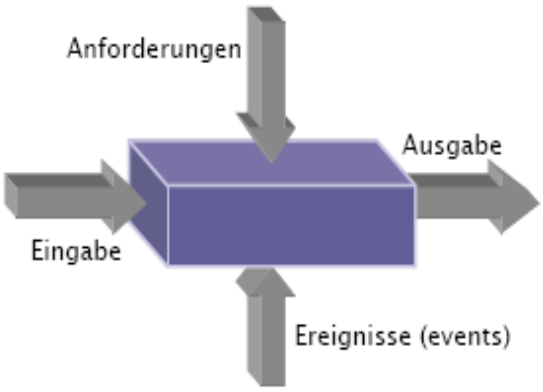

Abbildung 2: Black-Box [Wir03]

Wirkung, eine so genannte Spezifikation, herangezogen. Ziel ist es nun, die Spezifikation mit dem entsprechenden Verhalten des Systems zu überprüfen und damit eine weitgehende Funktionsüberdeckung zu erreichen. Folgende Methoden wurden für dieses Verfahren entwickelt:

- **funktionale Äquivalenzklassenbildung:**
  Die funktionale Äquivalenzklassenbildung nimmt an, dass ein Repräsentant einer Äquivalenzklasse das gleiche Verhalten aufweist, wie alle anderen Vertreter derselben Äquivalenzklasse. Ziel ist es also die Definitionsbereiche der Eingabewerte und die Wertebereiche der Ausgabewerte in Äquivalenzklassen zu zerlegen. Durch die Auswahl eines Repräsentanten aus jeder Äquivalenzklasse kann das System umfassend geprüft werden.(Hasselbrink, 2004)
  Prinzipiell werden zwei Äquivalenzklassentypen unterschieden: Die gültigen und die ungültigen Äquivalenzklassenbildungen. Bei der gültigen Äquivalenzklassenbildung werden zulässige Parameter in Klassen eingeteilt. Bei der ungültigen Äquivalenzklassenbildung werden entsprechend unzulässige Werte gesucht und diese in Klassen eingeteilt. Das bilden von Äquivalenzklassen erfolgt unter logischen Aspekten, indem das Augenmerk in erster Linie auf die Gleichartigkeit von Objekten gerichtet wird.

- **Grenzwertanalyse:**
  Die Grenzwertanalyse soll die Äquivalenzklassenbildung ergänzen. Häufig führen nämlich genau die Grenzfälle der Äquivalenzklassen zu Fehlern, so dass diese gesondert betrachtet werden sollten:

  Außerdem sollten besondere Werte, die durch die Äquivalenzklassenbildung nicht berücksichtigt werden, aufgegriffen werden.

- **Test von Zustandsautomaten:**
  Diese Methode ist für Testobjekte mit starken dynamischen Verhalten interessant.

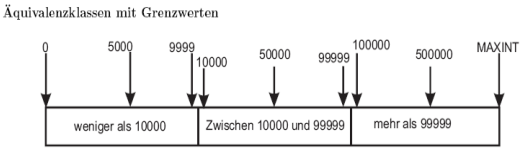

Abbildung 3: Äqulivalenzklassen und Grenzwerte [Has04]

> Voraussetzung der Anwendung dieser Methode ist allerdings, dass ein Zustandsdiagramm bzw. ein Ablaufsdiagramm als Spezifikation des Testobjektes vorliegt. Da ein Testfall die Folge von Ereignissen beschreibt, die auf das Testobjekt angewendet werden, wird nun für jeden Schritt dieser Folge als Soll der nächste erwartete Zustand und die entsprechende Ausgabe festgelegt.

Der Black Box Test ist lediglich ein ergänzender Test und alleine nicht ausreichend. Viele Fehler können aufgrund des hohen Abstraktionsgrades der Äquivalenzklassenbildung bei diesem Testverfahren nicht gefunden werden. Im Allgemeinen kann nicht davon ausgegangen werden, dass das System auf jeden Vertreter einer Äquivalenzklasse das gleiche Verhalten zeigt und die gewünschten Ergebnisse liefert. Ein erfolgreicher Black Box Test ist daher keine Garantie für die Fehlerfreiheit eines Programms.
Um trotzdem eine möglichst umfassende, funktionsüberdeckende Fehleranalyse zu gewährleisten, sollte ein Black Box Test bestenfalls von einem Kollegen durchgeführt werden, der den Quellcode tatsächlich nicht kennt. Dies deckt fehlende Inhalt des Quellcodes und Implementierungsfehler auf, indem im vor hinein verhindert wird, dass unbewusste zusätzliche Annahmen gemacht werden, die außerhalb der Spezifikation liegen, jedoch gewünschte Ergebnisse liefern. In großen Softwareprojekten gibt es daher in der Regel ein separates Testteam.
[Lig90, KF91, Sne83]

4.1.3 White-Box

Ähnlich wie der Black Box Test verläuft das Testverfahren des White Box Tests experimentell, durch erstellen geeigneter Testfälle. Im Gegensatz zum Black Box Test basiert jedoch der White Box Test auf die Analyse der Struktur des Systems und kann daher nur mit Kenntnis des Quellcodes durchgeführt werden. Ziel ist es, eine oder die Testmengen so zu erstellen, dass jeder Teil des Quellcodes, d.h.jede Anweisungen, jede Bedingungen und jeder Pfade mit dieser Testmenge mindestens einmal ausgeführt wird. Um die Übersicht des Testverfahren zu erhalten, sollten die Testmengen dabei stets möglichst minimal gewählt werden.
Der White Box Test besteht aus verschiedenen Hinlänglichkeitskriterien, die mit einer entsprechenden Testmenge erfüllt werden sollten. Damit wird der Quellcode systematisch getestet und die Qualität der Testverfahren sicher gestellt. Um einen besseren Überblick zu haben, welche Teile des Programms von dem jeweiligen Kriterium mit dem entsprechenden Testfall abgedeckt werden, wird zur Hilfestellung in der Regel ein Kontrollflussdiagramm konstruiert, dessen Knoteninhalte aus den Anweisungen bestehen und dessen Kontrollflüsse zwischen den Anweisungen als Kanten dargestellt werden(vgl Beispiel unten).
Der White Box Test besteht in der Regel aus folgenden kontrollflussorientierten Kriterien:

- **Anweisungsüberdeckungskriterium:** Jede Anweisung soll durch durch eine oder

mehrere geeignete Testmengen mindestens einmal ausgeführt werden. Dieses Kriterium bietet die Möglichkeit, nicht erreichte Anweisungen aufzudecken, wenn eine vollständige Überdeckung gefordert ist. Fehlende Anweisungen werden allerdings nicht erkannt.

- **Kantenüberdeckungskriterium:**
  Das Ziel dieses Kriteriums ist es, jede Kante eines Kontrollflussgraphen mindestens einmal zu durchlaufen. Die Kombinationen der zu durchlaufenden Kanten ist dabei unerheblich, ebenso wenig wie die Häufigkeit, wie oft die Kanten durchlaufen werden. Entscheidend ist, ob jede Kante mindestens einmal durchlaufen wird. Für das Kantenüberdeckungskriterium sind in der Regel mehr Testmengen erforderlich als bei dem Anweisungsüberdeckungskriterium. Dafür können mit Hilfe dieses Kriterium fehlende Anweisungen in Kanten erkannt werden, es ist jedoch unzureichend für den Test von Schleifen.

- **Bedingungsüberdeckungskriterium:**
  Dieses Kriterium orientiert sich im Wesentlichen an das Kantenüberdeckungskriterium. Der neue Kontrollflussgraph überdeckt jedoch zusätzlich alle elementaren (Teil-)Bedingungen, d.h. er berücksichtigt die durch logische Operatoren verknüpften Bedingungen und macht diese sichtbar. Prinzipiell werden -je nach Code- drei unterschiedliche Varianten des Bedingungsüberdeckungskriterium differenziert:
  - Einfache Bedingungsüberdeckungskriterium: Für das einfache Bedingungskriterium muss jeder Wert genau einmal den Wert true und genau einmal den Wert false annehmen.
  - Mehrfache Bedingungsüberdeckungskriterium: In dieser Variante müssen alle Kombinationen von true und false berücksichtigt werden.
  - Minimales mehrfach Bedingungsüberdeckungskriterium: Dieses Kriterium erstellt mehr Testmengen als in der ersten Variante, aber weniger als in der zweiten. Es werden hier lediglich die Kombinationen von true und false berücksichtigt, die die gesamte Bedingung auf true oder auf false setzen.

- **Pfadüberdeckungskriterium:**
  Jeder Pfad vom Start- bis zum Endknoten soll durch eine geeignete Testmenge mindestens einmal durchlaufen werden, wobei Schleifenwiederholungen als separate Pfade aufgefasst werden. Insbesondere lassen sich mit Hilfe des Pfadüberdeckungskriteriums Schleifen überprüfen. Eine vollständige Pfadüberdeckung ist wegen der in der Praxis unbestimmten Anzahl an Schleifendurchläufen jedoch nicht realsierbar.

Je nach Struktur des Quellcodes können aber auch andere Kriterien für den White Box Test verwendet werden.
Die Bestimmung der Testfälle sowie die Durchführung des Testverfahrens übernimmt der Programmierer. Dabei werden entweder ganze Module oder aber nur einzelne Methoden getestet. Daher kann insbesondere schon während der Programmierung getestet werden und nicht -wie beim Black Box Test- erst am Ende der Implementierung.
[Sil06, KF91, Wir03, Rie97]

Beispiel in Abb. 4 für das Vorgehen des Kantenüberdeckungskriterium:
Um eine vollständige Kantenüberdeckung zu erreichen, müssen die Testmengen so ge-

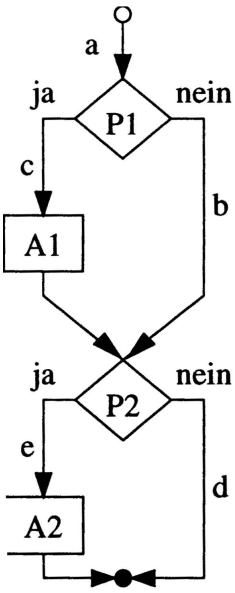

Abbildung 4: Kontrollflussgraph [Rie97]

wählt werden, dass alle Kanten des Kontrollflussgraphen mindestens einmal durchlaufen werden. Folgende Testdaten t und d erfüllen gemeinsamdie Kantenüberdeckung:
t:P1 = false und P2 = true. Hierdurch wird die Segementfolge *abc* ausgeführt.
d:P1 = true und P2 = false. Hierdurch wird die Segmentfolge *acd* ausgeführt. Da damit alle möglichen Kanten des Kontrollflussgraphen durchlaufen wurden, sind keine weiteren Testdaten mehr nötig.

Eine weiterer wichtiger Baustein des White Box Testverfahrens bildet das datenflussorientierte Unit-Testverfahren, das im Folgenden erläutert werden soll.

Vorteile des White Box Tests Mit dem White Box Test lassen sich schon während der Implementierung Prüfungen durchführen, was zu einer frühzeitigen Fehlererkennung führt. Ebenfalls erfolgt durch diesen Test eine Codeoptimierung, da überflüssige Codezeilen erkannt und gelöscht werden können. Außerdem wird die Wiederverwendbarkeit von Code unterstützt. Der White Box Test ist damit eine wichtige Ergänzung zum Black Box Test und deckt mehr Fehler auf als der Black Box Test, allerdings ist der White Box Test oftmals ein sehr aufwändiges Testverfahren.

Unit Test Vor allem in Java ist der Unit Test ein fester Bestandteil im Softwareentwicklungsprozess. Dieser Test ist ein Komponententest, dessen Ziel es ist, einzelne Module schon während der Softwareentwicklung auf ihre Korrektheit zu überprüfen. Der Unit Test ist damit ein wichtiger Baustein in der Qualitätssicherung und bildet die Voraussetzung für einen Systemtest. Das Modul wird in der Regel sehr klein gewählt, ist beispielsweise eine Klasse oder eine komplexere Methode, die getestet werden soll. Abhängigkeiten zwischen den einzelnen Modulen sollen nach Möglichkeit nicht einbezogen werden.
Die Ausführung des Unit Tests erfolgt im Allgemeinen dadurch, dass das zu testende Modul mit Parametern konfrontiert wird und die entsprechende Reaktion mit den erwartenden Verhalten verglichen wird. Stimmt die Reaktion mit den Erwartungen überein, so wird der Test verifiziert. Anderenfalls muss das entsprechende Modul überarbeitet werden und

der Test wiederholt werden. In der Regel müssen dennoch verschiedene Parameter gewählt werden, bevor die Korrektheit des Moduls tatsächlich bestätigt ist. Welche Parameter als testwürdig empfunden werden, hängt vom Entwickler ab. Prinzipiell sollte darauf geachtet werden, dass die Eingabe von den für das Modul typischen Parametern überprüft werden, sowie entsprechende Grenzwerte wie zum Beispiel extrem große oder kleine Werte, oder aber besondere Werte wie der Null- Zeiger. Erst wenn die verschiedenen Fälle verifiziert werden konnten, lässt sich die Korrektheit des Moduls bestätigen.
In der Praxis werden in der Regel zu einzelnen Klassen entsprechende Testklassen geschrieben, mit denen die Klasse automatisch getestet werden kann. Dies hat den Vorteil, dass der Programmierer Methoden und Funktionen möglichst einfach implementiert und die entsprechenden Objektstrukturen möglichst sauber entwirft, um entsprechend einfache Testklassen implementierten zu können. Ein weit verbreitetes Framework zum Schreiben und Ausführen dieser Tests für die Programmiersprache Jave ist JUnit. Die Tests werden direkt in Java programmiert, sind selbstüberprüfend und damit auch nach Änderungen wiederholbar.
Das Unit Testverfahren führt jedoch in der Praxis zu Problemen, wenn Benutzeroberflächen genutzt getestet werden sollen. Hierfür sollte besser das Model- View Konzept verwendet werden. Außerdem können Problem auftreten, wenn die zu testende Komponente mit anderen Komponenten in Abhängigkeit steht, oder wenn entsprechende Bibliotheken genutzt werden. Für solche Fälle sollten so genannte „Dummy Objekte“ in die Testklasse integriert werden, die eine künstliche Schnittstelle zu noch nicht vorhandenen Systemstellen schaffen.
[Mül07, Wes05]

4.2 Statische Verfahren, Review

Bei der Prüfung von Software ist man nicht nur auf das Testen durch Programme beschränkt. Die folgenden Abschnitte beschäftigen sich mit der Analyse eines Programms *von Hand*. Diese Verfahren nennt man auch statische Verfahren. Im Folgenden wird als statisches Verfahren hauptsächlich auf das Review eingegangen. Einige der weniger bedeutenden Verfahren werden am Ende dieses Kapitels kurz vorgestellt.

4.2.1 Grundlagen zu Reviews

Das Review ist eine Prüfung von Anforderungen, Entwürfe, Code, Testfällen oder Dokumentationen mit dem Ziel, Mängel aber auch positive Eigenschaften aufzuzeigen. Dabei wird nach vorher festgelegten Vorgaben und Richtlinien vorgegangen.

Beim Review gibt es eine bestimmte Rollenverteilung, die allerdings in den verschiedenen Kombinationen des Reviews auf verschiedene Arten umgesetzt werden kann. Prinzipiell wird aber folgende Rollenverteilung genutzt:

- **Manager:** Dies ist normalerweise der Vorgesetzte des Projektteams. Er hat den Auftrag zum Review gegeben, nimmt aber normalerweise am Review nicht teil.
- **Autor:** Der Autor kann für ein ganzes Team als Repräsentant stehen oder auch nur für sich alleine. Er ist der Entwickler des zu prüfenden Objekts.
- **Moderator:** Dieser ist ein Projektmitglied, der den Review leitet und für einen ordentlichen Ablauf sorgt.
- **Gutachter:** Gutachter analysieren das zu prüfende Programm und stellen Fachwissen zur Verfügung.

- **Protokollant:** Er ist der Schreiber, der das Protokoll der Review-Sitzung aufnimmt.
- **Review Team:** Zum Review Team gehören alle Mitglieder des Projekts, außer der Manager und der Autor.

Die Rolle des Protokollanten kann von jedem der am Review Beteiligten durchgeführt werden. Der Moderator sollte dies allerdings nicht tun, da er schon mit der Leitung des Reviews ausgelastet sein sollte. Am Ehesten eignet sich dafür der Autor, da dieser während des Reviews nicht viel zu tun hat.

4.2.2 Allgemeiner Ablauf

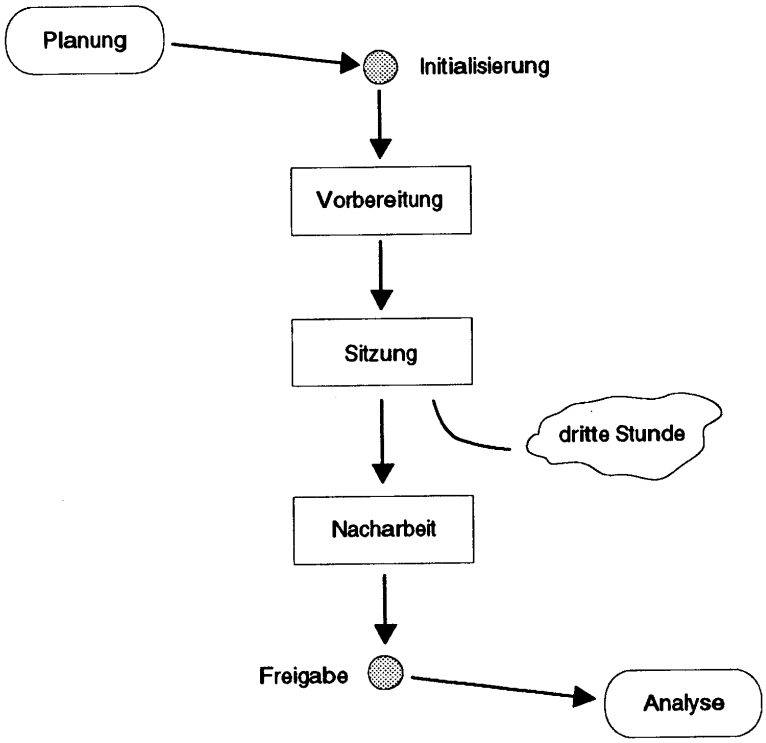

Abbildung 5: Review-Ablauf [KF91]

Der Ablauf des Reviews (vgl. Abb. 5) gliedert sich eigentlich in drei Phasen, Vorbereitung, Sitzung und Nacharbeit. Allerdings finden vorher noch die Phasen Planung und Initialisierung und im Anschluß an den Review die Freigabe und die Analyse statt.

Planung/Initialisierung:
Im Vorfeld des Reviews planen der Mangager und der Moderator zusammen die Richtlinien, nach denen die Arbeit des Autors vom Review Team analysiert werden soll. Die Gutachter werden nach den vorher bestimmten Richtlinien ausgewählt.
Falls der Autor seine Arbeit beendet hat, wird der Manager benachrichtigt. Dieser hat nun die Aufgabe, zu entscheiden, ob das Produkt des Autors in einem Review überprüft werden soll oder nicht. Soll die Arbeit einem Review unterzogen werden, erhält der Moderator im Rahmen der Initialisierung die Aufgabe, die Gutachter mit allen nötigen Informationen zu versorgen. Ist den Gutachtern das Umfeld der Arbeit nur gering bekannt, sollte ihnen in einer kurzen Einführungssitzung die zu prüfende Arbeit vorgestellt werden.

Vorbereitung:
In der Vorbereitungphase wird die Arbeit von den Gutachtern nach den ihnen zugeteilten Aspekten auf Fehler überprüft. Während der Überprüfung werden die gefundenen Mängel dokumentiert, allerdings nicht nur diejenigen, nach denen man gesucht hat, sondern es können auch Mängel aufgezeichnet werden, die nicht zu dem eigenen Aspekt gehören. Kleinere Mängel sollten direkt in der Arbeit markiert werden, da so die Review-Sitzung und der Bericht von Kleinigkeiten entlastet wird. Die ganze Vorbereitungszeit über sollte der Autor den Gutachtern für Fragen zur Verfügung stehen. Dem Moderator kommt dabei die Aufgabe zu, die Gutachter so zu unterstützen, dass sie diese Aufgabe ohne Schwierigkeiten wahrnehmen können. Vor allem muss er dafür Sorge tragen, dass die Gutachter für die Vorbereitung von ihren sonstigen Arbeiten freigestellt sind. Sollte dies nicht der Fall sein, muss er dies dem Vorgesetzten melden, der dann intervenieren sollte. Hat der Moderator den Eindruck, dass die Gutachter sich nicht ausreichend vorbereiten konnten, muss er den Review abbrechen, da dieser sinnlos sein würde.

Sitzung:
Die Sitzung sollte sich auf eine bestimmte, nicht zu lange Zeit beschränken. Schafft man es nicht, die komplette Arbeit zu besprechen, sollte man an einem anderen Tag die Sitzung fortführen. In der Sitzung soll die Arbeit von den Gutachtern beurteilt werden. Dabei soll nicht versucht werden, die Probleme zu lösen oder eventuelle Lösungen zu diskutieren. Das Ziel des Reviews besteht darin, festzustellen, welche Probleme in der Arbeit und im Projekt zu lösen sind.
Zu Beginn der Sitzung muss der Moderator sich über die Vorbereitung der Gutachter erkundigen. Ist er der Meinung, dass sie sich unzureichend vorbereitet haben, oder die Gutachter zu dem Schluss gekommen sind, dass die Qualität der Arbeit unzureichend ist, kann der Moderator die Sitzung schon zu diesem Zeitpunkt abbrechen. Andernfalls können die Gutachter damit beginnen, möglichst präzise ihre Befunde zu berichten. Dabei sollten sie darauf achten, keine Belehrungen auszusprechen, sondern nur ihre Beobachtungen zu schildern. Allerdings sollten nicht nur die negativen Ergebnisse berichtet werden, für die psychologische Einstellung des Autors ist es wichtig, dass auch positive Beobachtungen honoriert werden. Während der Vorstellung der Gutachter ist es die Aufgabe des Protokollanten, sämtliche Befunde aufzuzeichnen. Hierbei ist es wichtig, dass alle Mitglieder des Review-Teams die Aufzeichnung verfolgen können, um sofort eventuelle Missverständnisse aus dem Weg räumen zu können. Der Autor hat im Laufe der Review-Sitzung keine besondere Aufgabe, er sollte sich eigentlich rein passiv verhalten. Sein Beitrag beschränkt sich darauf, Verständnisschwierigkeiten zu seiner Arbeit zu beheben und der Diskussion zuzuhören. Am Ende der Sitzung bekommt er die markierten Kopien seiner Arbeit von den Gutachtern zurück, damit er diese in seine Arbeit einarbeiten kann.
Im Anschluss an die Sitzung wird die Beurteilung der Arbeit erstellt. Der Moderator sollte hierbei versuchen, das Review-Team zu einem Konsens zu bewegen. Diese Beurteilung soll dem Manager bei der späteren Freigabeentscheidung unterstützen.

Die dritte Stunde:
Da in der Sitzung selbst keine Lösungen besprochen werden sollten, es aber in der Natur eines Entwicklers liegt, bei Problemen auch über Lösungen nachzudenken, wäre es Ressourcenverschwendung, diese Lösungen bzw. eventuelle Lösungsansätze nicht zu besprechen. Daher steht es dem Review-Team frei, nach der eigentlichen Sitzung kurz über Lösungen zu diskutieren.

Nacharbeit/Freigabe:
Nachdem die Gutachter ihre Befunde vorgetragen haben, muss der Manager auf Grundlage

der Beurteilung entscheiden, ob die Arbeit freigegeben wird oder überarbeitet werden soll. Im Falle einer Überarbeitung, kann der Autor damit beginnen die festgestellten Mängel zu beseitigen. Auf keinen Fall sollten nichtgeforderte Neuerungen eingearbeitet werden, da dies nicht zum Review gehört und eventuell dazu führen kann, dass sich dadurch ein weiterer Review anschließt.
Hat der Autor seine Nacharbeit abgeschlossen, überprüft der Moderator diese. Wurden in der Sitzung allerdings gravierende Probleme festgestellt, muss dasselbe Review-Team die Prüfung erneut durchführen. In beiden Fällen wird anschließend der Manager bzw. Vorgesetzte benachrichtigt und ihm der Reviewbericht ausgehändigt. Er muss wieder entscheiden, ob diese freigegeben wird oder erneut überarbeitet und eventuell ein weiteres Mal einem Review unterzogen werden soll.

Analyse:
Ein Review hilft dem ganzen Projekt-Team, nicht nur dem Autor oder der Autorengruppe. Durch das Aufzeigen von Problemen und Mängeln kann jeder einzelne von ihnen etwas Neues lernen. Auch können durch diese Art von Prüfung gravierende Fehler, die während der Planung entstanden sind, erkannt werden.
[Inc07c, KF91]

4.2.3 Zu prüfende Dokumente durch Reviews

Damit nicht sämtliche kleinen Meilensteine geprüft werden und die Projektmitglieder nicht von einem Review in den nächsten gehen, was zu viel kostbare Zeit verschwenden würde, stellt sich die Frage, welche Art von Dokumenten sollten einem Review unterzogen werden. Laut [KF91] sind dies

- Dokumente, die dem Anwender bzw. Arbeitgeber präsentiert werden müssen. Dies wären also mindestens die Anforderungsanalyse, die Abnahmetestvorschrift und das Benutzerhandbuch
- Produkte, welche wesentliche Richtlinien enthalten. Dazu gehören auf jeden Fall der Systementwurf und Teilsystementwürfe.
- Alle Dokumente von denen man nicht weiß, wie sie sich auf ein Projekt auswirken. Dies wären technisches Neuland oder Arbeitsresultate neuer Mitarbeiter.
- Zentrale Bestandteile, die von mehreren Teilsystemen genutzt werden.
- Komponenten, die zeit- oder speicherplatzkritisch sind.

4.2.4 Alternativen zum Review

Es gibt noch eine Reihe weiterer statischer Verfahren zum Prüfen von Software. Im folgenden werden vier davon vorgestellt.

- **Schreibtischtest:**
  Schreibtischtests kann jeder für sich alleine durchführen. Hat man ein oder mehrere Ziele seiner Aufgabe erreicht, sollte man den Bildschirm verlassen und das bisher Erreichte analysieren. Etwas Abstand zum prüfenden Programm ist dabei recht sinnvoll.
- **Stellungnahme:**
  Die Stellungnahme ist simpel, sie benötigt keine besondere Ausbildung, keine Planung und kann kurzfristig durchgeführt werden. Bei diesem Verfahren reicht der

Entwickler seine Ergebnisse an ein oder mehrere Kollegen weiter, damit diese seine Arbeit analysieren. Erhält der Autor die Antworten, sollte er die negativen Stellungnahmen analysieren und -falls es in sein Konzept passt- in seine Arbeit integrieren. Das Verfahren scheitert allerdings oft daran, dass die Antworten sehr lange benötigen, bis der Autor sie erhält. Da der Autor normalerweise an seinen Aufgaben weiterarbeitet, kommen sie meistens zu spät, denn oft hat sich der betreffende Teil schon verändert.

- **technisches Review:**
  Bei dem technischem Review wurde versucht, die Nachteile des Verfahrens der Stellungnahme zu reduzieren. Ähnlich wie bei der Stellungnahme fungieren Kollegen als Gutachter, diese haben aber konkrete Aufgaben. Zuerst müssen sie das zu prüfende Dokument nach zugeteilten Gesichtspunkten analysieren. Ist dies abgeschlossen, treffen sie sich in einer Review-Sitzung, in welcher sie gefundene Probleme vorstellen und gemeinsam begutachten und protokollieren. Als Resultat erhält der Autor eine Zusammenstellung der Probleme mit Lösungsansätzen. Im Anschluß muss der Autor die Mängel seiner Arbeit beseitigen.

- **Walkthrough:**
  Beim Walkthrough handelt es sich um ein Verfahren, das eine weichere Form des Reviews darstellt. Hierbei ist der Autor der Moderator und stellt seine Ergebnisse nacheinander vor. Die Gutachter versuchen währenddessen mögliche Probleme ausfindig zu machen, welche gegebenfalls protokolliert werden und dem Autor anschließend zur Nacharbeit übergeben werden.

[KF91]

4.3 Test und Review im Vergleich

Beide Verfahren, sowohl das dynamische als auch das statische, haben unterschiedliche Stärken und Schwächen. Allerdings ergänzen sie sich gegenseitig. Ein Test kann für eine definierte Menge von Testfällen viele Fakten liefern. Ein Review schafft dies nicht, da ein Mensch nicht annähernd an die Leistung eines Programms heranreicht. Dagegen versagt ein Test, wenn der Fehler nicht im zu testenden Programm ist, sondern es ein konzeptioneller Fehler ist, von Entwicklern ist dieser dagegen unter Umständen schnell auszumachen. Des Weiteren erkennt ein Testprogramm nicht, wie aussagekräftig die Dokumentation oder Bezeichner sind. Auch ist der Review nicht nur auf den Quellcode angewiesen, durch einen Review kann man auch Fehler im Design oder in den Anforderungen aufdecken.
[KF91, Pre05, Inc07c]

5 Fazit

5.1 Zusammenfassung

Nachdem uns bewusst geworden ist, welche Verantwortung auf Software-System liegt (vgl. Kapitel 2.1), wurde deutlich gemacht, nach welchen Merkmalen man Qualität unterscheiden kann. Daraufhin wurde festgestellt, dass man durch konstruktive Qualitätssicherung (vgl. Kapitel 3) Fehler ohne großen Aufwand vermeiden kann. Denn durch gutes Prozess- und Projektmanagement, aber auch durch gute Organisation, entstehen nicht so viele Fehler in der Planung und Konzeption wie ohne gute Vorbereitung. Auch gibt es gute Systeme, die Organisationen bei dem Aufbau einer guten Projektstruktur unterstützen. In Kapitel

4 wurde erklärt, wie Tests arbeiten und welche Verfahren es gibt. Zudem wurden durch die verscheidenen Arten des Reviews Wege aufgezeigt, wie Fehler auch ohne Computergestützte Tests aufgedeckt werden können und warum es wichtig ist, dass nicht nur das dynamische oder das statische Verfahren benutzt wird.

5.2 Ausblick

Qualitätssicherung in der Softwareentwicklung bleibt unabdingbar. Sie wird sogar noch wichtiger werden. Denn immer mehr Bereiche der Gesellschaft werden von Softwaresystemen erschlossen und sind von dort nicht mehr wegzudenken. Da viele dieser Systeme eine hohe Lebensdauer haben, ist es wichtig, dass diese Systeme, nachdem sie auf den Markt gekommen sind, möglichst fehlerfrei sind. Zwar ist es anfangs billiger auf Qualitätssicherung zu verzichten oder nur rudimentär umzusetzen, doch steigen bei niedriger Qualität die Kosten für die Fehlerbeseitigung nach der Auslieferung sehr stark an (vgl. Abb. 6).
So sieht man, dass sich Kosten und Zeit, die man für die Qualitätssicherung aufgebracht

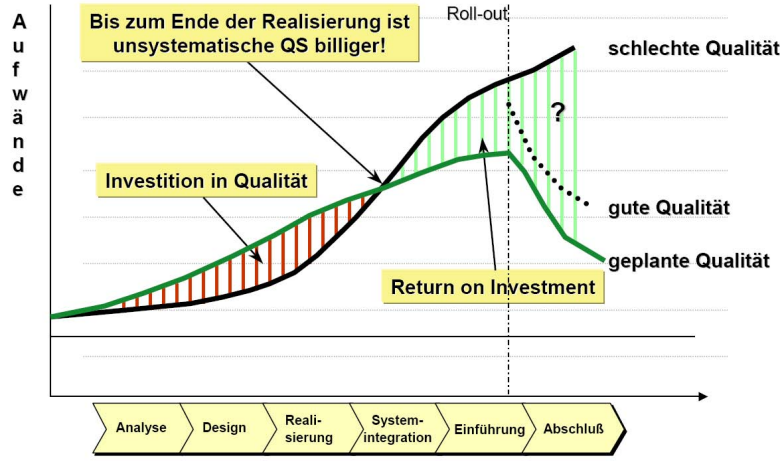

Abbildung 6: Kosten im Zusammenhang mit Tests [Kug01]

hat, am Ende doch bezahlt machen. Denn die Kosten für die Wartung des System bleiben nach dem Abschluss in der Regel eher niedrig. Ein weiterer Grund für die Qualitätssicherung ist, dass der Kunde es einem danken wird, wenn es nicht zu viele Probleme mit der Software gibt.

Literatur

[Com07] Communication, Global N.: *Konstruktive Qualitatssicherung.* Website, 2007. – Online http://www.gnc.at/A_Prod_Dev_009 (2007-10-25)

[Gra04] Grabowski, Jens: *Vorlesung: Methoden und Verfahren für das Testen von Software.* Website, 2004. – Online http://www.swe.informatik.uni-goettingen.de/notes/WS2004/Grabowski/I-Einfuehrung-2-auf-1-SW-Test.pdf (2007-10-24)

[Has04] Hasselbrink, Willhelm: *Software Engineering SS 2004.* Website, 2004. – Online https://elearning.uni-oldenburg.de/sendfile.php?force_download=1&type=0&file_id=304e4a87241c8bc0d2a99f46625c3524&file_name=s1-1on1.pdf (2007-10-24)

[Inc07a] Inc., Wikimedia F.: *Ariane 5.* Website, 2007. – Online http://de.wikipedia.org/wiki/Ariane_5 (2007-10-24)

[Inc07b] Inc., Wikimedia F.: *Capability Maturity Model Integration.* Website, 2007. – Online http://de.wikipedia.org/wiki/Capability_Maturity_Model_Integration (2007-10-25)

[Inc07c] Inc., Wikimedia F.: *Review(Softwareentwicklung.* Website, 2007. – Online http://de.wikipedia.org/wiki/Review_%28Softwareentwicklung%29 (2007-10-25)

[Inc07d] Inc., Wikimedia F.: *Spice (Norm).* Website, 2007. – Online http://de.wikipedia.org/wiki/Spice_%28Norm%29 (2007-10-25)

[KF91] Karol Frühauf, Helmut S. Jochen Ludewig L. Jochen Ludewig: *Software-Prüfung Eine Fibel.* Stuttgart : B.G.Teubner, 1991

[Kug01] Kugel, Thomas: *Präsentation SQS.* Website, 2001. – Online http://www.gi-dresden.de/files/181001.pdf (2007-10-25)

[Lig90] Liggesmeyer, Peter: *Modultest und Modulverifikation.* Mannheim : Bibliographisches Institut und F.A. Brockhaus AG, 1990

[MF05] Malte Foegen, Marion T. Claudia Raak R. Claudia Raak: *Capability Maturity Model Integration, Eine Einführung in CMMI als ein Werkzeug zur Prozessverbesserung.* Website, 2005. – Online http://www.wibas.de/e20/e52/e912/cmmi_de.pdf (2007-10-25)

[Mül07] Müller, Alexander: *Unit-Tests (Komponententests).* Website, 2007. – Online http://www.evocomp.de/softwareentwicklung/unit-tests/unittests.html (2007-10-25)

[Pre05] Prechelt, Lutz: *Softwaretechnik WS2005/2006.* Website, 2005. – Online http://www.inf.fu-berlin.de/inst/ag-se/teaching/V-SWT-2005/41_Analytische-QS.pdf, http://www.inf.fu-berlin.de/inst/ag-se/teaching/V-SWT-2005/42_Analytische-QS2.pdf, http://www.inf.fu-berlin.de/inst/ag-se/teaching/V-SWT-2005/43_Konstruktive-QS.pdf (2007-10-24)

[Rie97] Riedemann, Eike H.: *Testmethoden für sequentielle und nebenläufige Software-Systeme*. Stuttgart : B.G.Teubner, 1997

[Sil06] Silberhorn, Bernhard: *White-Box-Test, Fachhochschule Augsburg*. Website, 2006. – Online `http://www.diplomanach.fh-augsburg.de/informatik/projekte/testen/ss2006/whitebox.pdf` (2007-10-25)

[Sne83] Sneed, Harry M.: *Software-Qualitätssicherung für kommerzielle Anwendungssysteme*. Köln-Braunsfeld : Verlagsgesellschaft Rudolf Müller GmbH, 1983

[Wes05] Westphal, Frank: *Unit Tests mit JUnit*. Website, 2005. – Online `http://www.frankwestphal.de/ftp/UnitTestsmitJUnit.pdf` (2007-10-25)

[Wes06] Westphal, Frank: *Testgetriebene Entwicklung mit JUnit und FIT*. 69115 Heidelberg, Ringstraße 19 : dpunkt.verlag GmbH, 2006

[Wir03] Wirsing, Martin: *Vorlesung Methoden des Software Engineering*. Website, 2003. – Online `http://www.pst.ifi.lmu.de/lehre/WS0304/mse/BlackWhiteBox2p.pdf` (2007-10-25)