

Institut für Softwaretechnologie

Abteilung Software Engineering

Universität Stuttgart
Universitätsstraße 38
D - 70569 Stuttgart

Fachstudie Nr. 89

Analyse und Kritik von Anforderungsspezifikationen

Stefan Franke, Christoph Müller, Diana Przybylski

Studiengang: Softwaretechnik

Prüfer: Prof. Dr. rer. nat. Jochen Ludewig

Betreuer: Dipl.-Inf. Holger Röder

begonnen am: 01.05.2008

beendet am: 03.11.2008

CR-Klassifikation: D.2.1 Requirements/Specifications



Universität Stuttgart
Institut für Softwaretechnologie
Universitätsstraße 38
70569 Stuttgart



Abteilung
Software Engineering

Fachstudie

Analyse und Kritik von Anforderungsspezifikationen

Abschlussdokument

Prüfer:	Prof. Dr. rer. nat. Jochen Ludewig
Betreuer:	Dipl.-Inf. Holger Röder
Bearbeiter:	Stefan Franke, Christoph Müller, Diana Przybylski
Status:	Freigegeben
Datum:	3. November 2008
SVN-Revision:	228

Dokumentversionen

Version	Datum	Name	Kommentar
0.1	02.09.2008	Diana Przybylski	Dokument erstellt
0.2	23.09.2008	Christoph Müller	Neue Gliederung; weitere Dokumente eingebunden
0.3	04.10.2008	Diana Przybylski	Einleitung, Vorgehen
0.4	29.10.2008	Diana Przybylski	Vorgehen fertiggestellt
0.5	29.10.2008	Christoph Müller	Bewertungsübersicht
0.6	02.11.2008	Stefan Franke	Ergebnisse der Fachstudie
1.0	03.11.2008	Christoph Müller	Version 1.0 freigegeben

Inhaltsverzeichnis

1	Einleitung	11
1.1	Die Aufgabe	11
1.2	Reflexion der Aufgabenstellung	11
1.3	Abgrenzung der Arbeit	12
1.4	Inhalt des Dokumentes	13
2	Vorgehen	15
2.1	Phasen und Meilensteine	15
2.2	Literaturrecherche	16
2.3	Spezifikationssuche	17
2.4	Kriterienkatalog	19
2.5	Bewertungsphase	20
2.6	Kommentierung einer Spezifikation	24
3	Ergebnisse der Fachstudie	25
A	Literaturrecherche	29
A.1	Einleitung	29
A.2	Anforderung	30
A.2.1	IEEE 610.12	30
A.2.2	IEEE 1233	30
A.2.3	Wikipedia	31
A.2.4	Robertson	31
A.2.5	Ludewig	32
A.3	Spezifikation	33
A.3.1	IEEE 610.12-1990	33
A.3.2	IEEE 830-1998	33
A.3.3	IEEE 1012-2004	34
A.3.4	Pressmann	35
A.3.5	Wikipedia	35
A.3.6	Robertson	35

A.3.7	Ludewig	36
A.4	Spezifikationsvorlagen	37
A.4.1	Ludwig und Lichter	37
A.4.2	IEEE 1233	39
A.4.3	Robertson	40
A.4.4	Klüver	42
A.4.5	Wikipedia	42
A.5	Kriterienkataloge & Bewertungsschemata	44
A.6	Checklisten	45
A.6.1	Drappa	45
B	Kriterienkatalog	47
B.1	Einleitung	47
B.1.1	Zweck des Dokuments	47
B.1.2	Aufbau des Dokuments	47
B.2	Kriterien	49
B.2.1	Bewertungsschema	49
B.2.2	Grundlegende Kriterien	49
B.2.2.1	Dokumenteigenschaften	49
B.2.2.1.1	Einhaltung grundlegender Dokumentanforderungen [K-01]	49
B.2.2.1.2	Deckblatt [K-02]	51
B.2.2.1.3	Abbildungen und Tabellen (Äußere Form) [K-03]	52
B.2.2.1.4	Ausgewogenheit [K-04]	53
B.2.2.1.5	Größe nichtstrukturierter Unterkapitel [K-05]	54
B.2.2.2	Lesbarkeit/Übersichtlichkeit	56
B.2.2.2.1	Grammatik/Satzbau [K-06]	56
B.2.2.2.2	Textsatz [K-07]	57
B.2.2.2.3	Typographische Konventionen [K-08]	58
B.2.2.2.4	Links [K-09]	59
B.2.2.3	Sprache	60
B.2.2.3.1	Präzision [K-10]	60
B.2.2.3.2	Knappheit [K-11]	61

B.2.2.3.3	Fremdwörter [K-12]	63
B.2.2.3.4	Sprachliche Konsistenz [K-13]	65
B.2.2.4	Begriffslexikon	67
B.2.2.4.1	Existenz [K-14]	67
B.2.2.4.2	Begriffsdefinitionen [K-15]	68
B.2.2.4.3	Verwendung [K-16]	69
B.2.2.5	Benutzbarkeit	71
B.2.2.5.1	Versionierung [K-17]	71
B.2.2.5.2	Index [K-18]	72
B.2.2.6	Prozessfreiheit [K-19]	73
B.2.2.7	Abbildungen [K-20]	74
B.2.3	Vollständigkeit der inhaltlichen Aspekte [K-21]	75
B.2.4	Allgemeine Kriterien für Anforderungen	81
B.2.4.1	Eigenschaften der Anforderungen	81
B.2.4.1.1	Identifizierbarkeit [K-22]	81
B.2.4.1.2	Eindeutigkeit [K-23]	83
B.2.4.1.3	Nachverfolgbarkeit [K-24]	84
B.2.4.1.4	Konsistenz (Widerspruchsfreiheit) [K-25] . . .	85
B.2.4.1.5	Zusammengehörigkeit & Ordnung [K-26] . . .	86
B.2.4.1.6	Verifizierbarkeit [K-27]	87
B.2.4.2	Entwurfsoffenheit [K-28]	88
B.2.4.3	Abstraktionsgrad [K-29]	90
B.2.4.4	Beispiele [K-30]	91
B.2.5	Funktionale Anforderungen	93
B.2.5.1	Methodik [K-31]	93
B.2.5.2	Zu treffende Aussagen pro funktionale Anforderung [K-32]	97
B.3	Abgrenzung	99
C	Kommentierte Spezifikation	101
C.1	Einleitung	101
C.1.1	Motivation	101
C.1.2	Verwendungshinweise	103

C.1.3	Anwendung des Kriterienkataloges	104
C.1.4	Allgemeine Hinweise zur Erstellung einer Spezifikation	104
C.2	Spezifikation - CodeCover	106
	Version History	107
	Contents	110
1	Introduction	112
1.1	Project overview	112
1.2	About this document	113
1.3	Addressed audience	113
1.4	Conventions for this document	114
1.5	Authors	114
2	Functional requirements	115
2.1	Test sessions and test cases	115
2.2	Actors	116
2.3	Use case description	117
2.4	Batch interface	145
2.5	Configuration	157
2.6	Report	159
2.7	Instrumentation, types of coverage and measurement	162
2.8	Language support	167
2.9	JUnit integration	167
2.10	ANT integration	169
2.11	Live Test Case Notification	182
3	Graphical User Interface	185
3.1	Package and file states	185
3.2	Instrumentation	186
3.3	Launching	186
3.4	Coverage view	187
3.5	Test sessions view	188
3.6	Import	190
3.7	Export	193
3.8	Source code highlighting	194
3.9	Preferences dialog	199

3.10	Project properties dialog	201
3.11	Correlation Matrix	202
3.12	Live Notification View	204
3.13	Boolean Analyzer	205
3.14	Hot-Path	206
4	Non-functional requirements	207
4.1	Technologies and development environment	207
4.2	Requirements to the working environment	207
4.3	Quantity requirements	208
4.4	Performance requirements	210
4.5	Availability	211
4.6	Security	211
4.7	Robustness and failure behavior	211
4.8	Usability	211
4.9	Portability	212
4.10	Maintainability	212
4.11	Extensibility	212
	List of Figures	214
	Glossary	215
	Literaturverzeichnis	221

1 Einleitung

1.1 Die Aufgabe

Die Aufgabe ist die Analyse und konstruktive Kritik von Anforderungsspezifikationen, die in der Lehre oder auch der Praxis erstellt wurden. Um die Spezifikationen analysieren und kritisieren zu können, war die Erstellung eines Kriterienkataloges und eines zugehörigen Bewertungsschemas für jedes einzelne Kriterium notwendig. Die Kriterien ergaben sich aus der Literaturrecherche. Diese Recherche hatte zur Aufgabe, charakteristische Merkmale für die Qualität einer Spezifikation zu finden. Merkmale, die sich wegen ihrer Formulierung nur indirekt für die Bewertung einer Spezifikation eignen, wurden entsprechend interpretiert. Mit diesem Dokument sollen die aus der Lehre und Industrie gesammelten Spezifikationen bewertet werden. Die Spezifikation mit der besten Bewertung soll mithilfe des erstellten Kriterienkataloges kommentiert werden.

Diese Fachstudie richtet sich also an alle, die an einer kommentierten Spezifikation interessiert sind. Sie ist nicht nur für Studierende geeignet, die eine ihrer ersten Spezifikationen erstellen wollen, sondern auch für die Praxis nützlich. Dabei ist zu beachten, dass die Beispielspezifikation ein Testwerkzeug spezifiziert. Es ist also gut möglich, dass sich nicht alles auf die Erstellung von Spezifikationen anwenden lässt. Dies ist der Fall, wenn das entstehende Produkt zu große Unterschiede zum Produkt der Beispielspezifikation aufweist. Trotzdem vermittelt die Beispielspezifikation eine Vorstellung davon, wie eine Spezifikation aussehen sollte.

1.2 Reflexion der Aufgabenstellung

Die Anforderungen an ein Software-Produkt sind die Grundlage für eine gute Qualität der Software. Im Entwicklungsprozess einer Software werden verschiedene Dokumente erstellt. Eines davon ist die Anforderungsspezifikation (im Folgenden auch nur Spezifikation genannt). In ihr werden alle Anforderungen spezifiziert. Diese Anforderungsspezifikation bildet den Ausgangspunkt für die Entwicklung der Software, aber auch für alle weiteren Dokumente, die während des Entwicklungsprozesses

entstehen. Aufgrund dessen ist eine gute Qualität dieses Dokumentes sehr wichtig.

Trotz dieser Tatsache werden Spezifikationen in der Praxis selten erstellt und vorhandene Spezifikationen unterscheiden sich deutlich in Qualität, Aufbau und Inhalt. Zudem werden Spezifikationen je nach Umfeld auch Pflichtenheft oder Fachkonzept genannt. Eine weitere Schwierigkeit besteht für die erste Erstellung einer Spezifikation darin, dass es nur sehr wenige gute Beispielspezifikationen gibt, die öffentlich zugänglich sind.

Das Problem der Bewertung der Qualität von Spezifikationen sind wir angegangen, indem wir eine Reihe an Bewertungskriterien entwickelt haben, mit denen die Qualität bewertet werden kann. Da die Qualität der Spezifikationen aber nicht nur von greifbaren Aspekten abhängt, gibt es auch Kriterien, die mit dem subjektiven Empfinden bewertet werden müssen. Trotzdem haben wir versucht, alle Kriterien so objektiv wie möglich zu halten. Zusätzlich haben wir als Hilfestellung für die Bewertung Anleitungen zu einzelnen Kriterien geschrieben.

1.3 Abgrenzung der Arbeit

Die Aufgabe zielt nicht darauf ab, ein Review der Spezifikationen durchzuführen. Die Spezifikationen werden ausschließlich anhand des Kriterienkataloges bewertet. Dieser Katalog enthält verschiedene Kriterien, die beispielsweise die Form der Spezifikation oder auch die verwendeten Notationen bewerten. Inhalte, die ein Kenntnis der Anforderungsanalyse voraussetzen, können nicht bewertet werden, weil die Betrachtung der Anforderungsanalyse nicht zur Aufgabe zählt.

Die Erarbeitung einer Beispielgliederung von Spezifikationen gehört auch nicht zu den Aufgaben der Fachstudie. Bei der Recherche und der ersten Durchsicht hat es sich gezeigt, dass es auch nicht möglich wäre, eine Beispielgliederung zu erstellen, da die Eignung einer Gliederung sehr stark von dem beschriebenen Produkt abhängt. Auch die Neuerstellung einer Beispielspezifikation ist nicht Teil der Aufgabe. Die Kommentierung einer bestehenden Spezifikation ist hier sinnvoller, da in ihr sowohl positive, als auch negative Aspekte aufgezeigt werden können. Diese Spezifikation sollte trotzdem nicht als **die** Superspezifikation gesehen werden, da zwar

eine gute Spezifikation zur Kommentierung ausgewählt wurde, diese aber nur eine Lösungsmöglichkeit darstellt.

Die Aufgabe beschränkt sich also ausschließlich auf die Betrachtung von Spezifikationen. Dies bedeutet, dass auch der Prozess, in dem die Spezifikationen erstellt wurden und auch die dazu verwendeten Werkzeuge keine Rolle spielen.

1.4 Inhalt des Dokumentes

Dieses Dokument ist wie folgt aufgebaut:

- Unser Vorgehen

In diesem Kapitel werden zuerst die Phasen und Meilensteine der Fachstudie aufgezeigt. Darauf folgt die Erläuterung der Literaturrecherche. Es wird das konkrete Vorgehen beschrieben. Die Ergebnisse dieser Phase ist im Anhang zu finden. Neben der Literaturrecherche ist auch die Spezifikationssuche beschrieben worden. Auch zu dieser Phase werden das Vorgehen und die Ergebnisse beschrieben. Auf Basis der Ergebnisse der Literaturrecherche entsteht in der nächsten Phase der Kriterienkatalog, der auch beschrieben wird. Der Katalog an sich befindet sich aber nicht an dieser Stelle, sondern ist wieder im Anhang zu finden. Nach all diesen Phasen folgte die Bewertungsphase, in der nun verschiedene Spezifikationen bewertet wurden, um eine geeignete Spezifikation für die Kommentierung zu finden und diese auch zu kommentieren. Dieses Vorgehen wird am Ende des Kapitels ausführlich beschrieben.

- Ergebnisse Hier werden alle Ergebnisse, die aus der Arbeit während der Fachstudie gezogen werden konnten zusammengefasst.

- Anhang Im Anhang befinden sich alle Dokumente, die im Rahmen der Fachstudie erstellt worden sind. Dies sind im Einzelnen:

- Literaturrecherche
- Kriterienkatalog
- Kommentierte Spezifikation

2 Vorgehen

In diesem Kapitel werden zuerst die einzelnen Phasen und die dazugehörigen Meilensteine erläutert. In den Unterkapiteln 2.2 bis 2.6 werden die einzelnen Phasen beschrieben.

2.1 Phasen und Meilensteine

Eine Fachstudie ist auf drei Monate ausgelegt. Sie ist in die Meilensteine Literaturrecherche und Sammeln von Spezifikationen, Kriterienkatalog, Bewertung der Spezifikationen und Kommentierung der Spezifikation aufgeteilt. Jeder von uns wollte auch in jeder Phase mitarbeiten. Bei unserer Fachstudie haben wir bewusst mehr Zeit eingeplant, da wir für unsere Arbeit Spezifikationen sammeln mussten. Wir sahen Probleme darin, in dieser kurzen Zeit genug Spezifikationen zu finden, mit denen wir arbeiten können. Aus diesem Grund ist unsere Fachstudie auf sechs Monate ausgelegt worden.

Während wir die Spezifikationen sammelten, hatten wir noch ausreichend Zeit, die Literaturrecherche durchzuführen. Während dieser Phase mussten wir uns auch darüber klar werden, was unsere Aufgabe genau beinhaltet. In manchen Punkten hatten wir sehr unterschiedliche Vorstellungen, mussten diese diskutieren, um zu einer klaren und einheitlichen Vorstellung zu gelangen. Unser größter Diskussionspunkt war, welche Kriterien wichtig sind, um eine Spezifikation zu bewerten und was diese Kriterien beinhalten sollten. Ein Problem, das sich daraus ergab, war, dass wir immer wieder bei dem Review unseres Kriterienkataloges darauf zu sprechen kamen, wie die gesamte Bewertung durch die einzelnen Kriterien zustande kommen sollte.

Unsere Vorstellungen von Anforderung und Spezifikation sind in den Ergebnissen der Literaturrecherche im Anhang zu finden.

2.2 Literaturrecherche

In der Literaturrecherche haben wir zuerst nach Definitionen für Spezifikation und Anforderung gesucht, um uns klar zu machen, auf welche Einzelheiten wir bei unserer Arbeit Wert legen müssen. Darüber hinaus haben wir uns auf eine Definition einer Spezifikation und einer Anforderung festgelegt (siehe Literaturrecherche).

Wir haben Anhaltspunkte gesucht, die uns einen Weg weisen, wann die Qualität einer Spezifikation gut oder wann sie schlecht ist. Dabei ist uns aufgefallen, dass in der Literatur die Qualitätsmerkmale oft an konkreten Elementen festgemacht werden. Sie werden nur benannt, es gibt keinen Hinweis und auch keine Anleitung, wie man überprüfen könnte, ob bestimmte Kriterien erfüllt sind.

Außerdem haben wir nach Beispielspezifikationen und Bewertungsschemata gesucht. Diese Dokumente hätten uns bei unserer Aufgabe unterstützen können. Die Recherche hatte aber ein ernüchterndes Ergebnis. Wir fanden zwar Definitionen, die uns weiterbrachten, aber keine vorhandenen Bewertungsschemata oder Kriterienkataloge und nur eine Beispielspezifikation. Diese Beispielspezifikation war für uns nützlich, da wir sie als erste Spezifikation bewertet haben. Ihr Aufbau war interessant, die Qualität konnten wir aber nicht richtig bewerten, da diese Spezifikation als Beispiel Anregungen geliefert hat, aber oft nicht vollständig war.

Auch Arbeiten, die unserer ähnlich sind, haben wir nicht gefunden. Darüber hinaus haben wir auch keine Anhaltspunkte gefunden, die etwas über die Eignung einer Spezifikation für bestimmte Softwarekategorien aussagen und auch nicht die Eignung z. B. der Gliederung überhaupt.

Mit diesem Ergebnis der Recherche waren wir also auf uns alleine gestellt und haben alle Ergebnisse des Kriterienkataloges selbst erarbeitet. Als Grundlage dienten uns dazu Anhaltspunkte, die wir während der Recherche in der Literatur als hilfreich und wegweisend empfanden.

Ein weiteres Ergebnis der Literaturrecherche war die Definition des Begriffes „Anforderungsspezifikation“, auf den sich die Fachstudie stützt. Unser Verständnis einer Spezifikation entspricht zusammengefasst der Definition von „software requirements specification“ im Standard IEEE 1012-2004 (IEE, 2005):

Documentation of the essential requirements (functions, performance, design constraints, and attributes) of the software and its external interfaces.

Wir legen zusätzlich einen besonderen Wert darauf, dass Anforderungsspezifikationen dem Entwurf nicht vorgreifen dürfen und nur tatsächliche Entwurfsanforderungen aus der Analysephase dokumentieren sollten. Zusätzlich sind die Beschreibungen innerhalb einer Spezifikation frei von Abhängigkeiten vom vorliegenden Entwicklungsprozess zu halten. Abgabetermine und Prioritäten für Teile des Systems sollten in einer gesonderten Weise dokumentiert sein.

2.3 Spezifikationssuche

Bei der Spezifikationssuche konzentrierten wir uns zuerst auf Software- und Studienprojekte der Universität Stuttgart. In der Industrie war es sehr schwierig, Spezifikationen zu finden. Einer der Gründe dafür ist, dass in der Industrie häufig keine Spezifikationen erstellt werden. Zudem kommt es häufig vor, dass zwar Spezifikationen existieren, aber nicht eingesehen und verwendet werden dürfen. Aus diesen Gründen haben wir die meisten Spezifikationen von studentischen Projekten bekommen. Darunter auch von anderen Universitäten.

Letztlich konnten dennoch einige Industriespezifikationen beschafft werden. Vornehmlich über Kontakte der wissenschaftlichen Mitarbeiter der Abteilung Software Engineering.

Wenn man die gesammelten Spezifikationen betrachtet, fällt auf, dass Spezifikationen von der gleichen Universität sehr ähnlich aufgebaut sind, dass sie sich aber zwischen den Universitäten und erst recht im Vergleich zur Industrie sehr stark unterscheiden. Am Ende der Spezifikationssuche war die Zahl der gesammelten Spezifikationen größer, als wir zu Beginn vermutet hatten.

Im Einzelnen sind dies die Spezifikationen:

- Softwarepraktika der Universität Stuttgart
 - SPT-Manager (Werkzeug, 56 Seiten)

- TimeCop (Werkzeug, 32 Seiten)
- Fred (Werkzeug, 93 Seiten)
- Studienprojekte A und B der Universität Stuttgart
 - Aims (Informationssystem, 68 Seiten)
 - CodeCover (Werkzeug, 115 Seiten)
 - Emmu (Werkzeug, 168 Seiten)
 - EvA (Simulation, 114 Seiten)
 - EvoLab (Framework, 75 Seiten)
 - FOAS (Simulation, 46 Seiten)
 - FOAS 2 (Simulation, 100 Seiten)
 - IMLGen (Werkzeug, 123 Seiten)
 - JUST (Informationssystem, 91 Seiten)
 - Justus (Werkzeug, 65 Seiten)
 - VISdGets (GUI-Toolkit, 158 Seiten)
- Projekte von anderen Universitäten
 - An exemplary requirements specification (Beispielspezifikation, 33 Seiten)
 - Bitart (Informationssystem, 99 Seiten)
 - Black Music (Webseite, 33 Seiten)
 - Gnu (Informationssystem, 123 Seiten)
 - medizinisches Beratungssystem (Werkzeug, 19 Seiten)
 - FIFO (Informationssystem, 17 Seiten)
- Industrie
 - Daimler: GO WIN (Informationssystem, 357 Seiten)
 - Daimler: Empress (Steuergerät, 64 u. 85 Seiten)

- Fraunhofer: Türsteuergerät (Steuergerät, 61 Seiten)
- Fraunhofer: Elektronischer Arztausweis (Schnittstelle, 121 Seiten)
- Praktikum sd&m: Messung der Codeüberdeckung (Werkzeug, 109 Seiten)
- Bonner Akademie: SAM (Informationssystem, 8 Seiten)
- accenture: GLMS (Informationssystem, 163 Seiten)

2.4 Kriterienkatalog

Die Erstellung des Kriterienkataloges fand mit den Ergebnissen der Literaturrecherche statt. Wir haben die Erstellung der Kriterien etappenweise vorgenommen. Zuerst haben wir uns – jeder für sich – den Aufbau eines Kriteriums überlegt. Nachdem wir diese Ergebnisse diskutiert und zusammengeführt hatten, stand fest, dass ein Kriterium aus einer Beschreibung, einem Zweck, einer Ausführungsanleitung und einer Bewertung bestehen soll.

Nun ging es im nächsten Schritt darum, dass jeder ein Kriterium mit diesem Aufbau einmal ausformuliert, damit wir entscheiden konnten, ob der Kriterienaufbau zielführend ist. Die in diesem Schritt erstellten Kriterien fanden später in Teilen auch Einzug in den Katalog. Mit diesen drei Kriterien als Grundlage haben wir in der nachfolgend die Liste an Kriterien aufgestellt. Dazu hat sich zunächst wieder jeder selbst Kriterien überlegt, die wir anschließend zusammengeführt haben.

Nach diesen Schritten war der Katalog in seinen Grundzügen erstellt, d. h. die Liste der Kriterien war erstellt und musste nun sortiert, hierarchisiert und ausformuliert werden. Es schloss sich dann die Ausformulierung aller Kriterien an, die bisher nur als Bezeichnung und zugrundeliegende Idee vorhanden waren. Während der Ausformulierung der Kriterien wurde schnell klar, dass es bei einigen Kriterien nicht möglich ist, eine objektive Bewertung abzugeben. In diesen Fällen mussten wir auf das subjektive Empfinden des Gutachters zurückgreifen.

Der gesamte Kriterienkatalog ist mit einer gesonderten Beschreibung im gleichnamigen Anhang B zu finden.

2.5 Bewertungsphase

Zweck der Bewertungsphase war es, aus allen Spezifikationen eine zu auszuwählen, die sich am besten für die Kommentierung eignet. Außerdem haben wir unseren Kriterienkatalog angewendet und so verbesserungs- oder besprechungswürdige Kriterien identifiziert. Dadurch konnten wir die Praktikabilität validieren und ihn verbessern.

Da wir nicht mit einem solchen Erfolg beim Sammeln der Spezifikationen gerechnet hatten, hatten wir auch nicht so viel Zeit eingeplant, um alle Spezifikationen zu bewerten. Aus diesem Grund haben wir eine Auswahl an Spezifikationen getroffen, die wir bewerten wollten. Hierzu haben wir eine Tabelle erstellt, die den Inhalt, die Art der beschriebenen Software und den Umfang der Spezifikation darstellt.

Danach haben wir erst eine grobe Vorauswahl getroffen. In dieser Vorauswahl wurden die Spezifikationen aus folgenden Gründen aussortiert:

- liegt in Papierform vor:
 - eEnergy2
 - SAM
 - GLMS
- wir sind als Gutachter befangen:
 - Fred
 - Timecop
 - SPT-Manager
 - IMLgen
 - EVA
 - FOAS 2
 - Messung der Codeüberdeckung
 - CodeCover (ist zu gut um hier herauszufallen)

- Steuergeräte, die nicht in unser Bild passen:
 - Empress
 - Türsteuergerät
 - Elektronischer Arztausweis
- Schlechte Qualität erwartet:
 - FIFI
 - Black Music
 - Med. Beratungssystem

Für das Festhalten der Bewertungsergebnisse erstellten wir ein Bewertungstemplate in Microsoft Excel/OpenOffice. Für jedes Kriterium des Kriterienkataloges erstellten wir darin ein Arbeitsblatt, in dem der Gutachter Kommentare und seine letztendliche Bewertung festhalten kann. Unterstützung bei der Berechnung sowie für die automatische Auswahl der Bewertungskategorie (für manche Kriterien) unterstützen den Gutachter zusätzlich. Eine Übersichtsseite zeigt dem Gutachter alle Bewertungen zentral und erlaubt uns einen einfachen Vergleich der Spezifikationen.

Um eine grobe Fehleinschätzung bei diesen Bewertungen zu vermeiden, führten wir zu Beginn eine Bewertung der Beispielspezifikation „An exemplary requirements specification“ durch. Ein Vergleich unserer Ergebnisse sollte uns als Gutachter eichen und dafür sorgen, dass wir alle ein gleiches Verständnis der Kriterien erlangen. Erst danach führten wir die Bewertung der nicht aussortierten Spezifikationen wie folgt durch:

- Christoph: GO (nur Teile bewertet, da das Dokument sehr groß ist)
- Diana: CodeCover, EvoLab
- Stefan: Bitart, JUST, CodeCover

Die Resultate der Bewertungsphase fasst Tabelle 1 zusammen. Die Abkürzungen in den Zellen stehen für:

- v: Kriterium *voll erfüllt*

- ü: Kriterium *überwiegend erfüllt*
- t: Kriterium *teilweise erfüllt*
- n: Kriterium *nicht erfüllt*
- -: Kriterium nicht bewertet oder nicht bewertbar

Kriterium	GO	CodeCover (Stefan)	CodeCover (Diana)	Bitart	JUST	EvoLab
<i>Einhaltung grundlegender Dokumentanforderungen [K-01]</i>	v	v	v	v	v	v
<i>Deckblatt [K-02]</i>	v	ü	ü	t	v	ü
<i>Abbildungen und Tabellen (Äußere Form) [K-03]</i>	ü	v	ü	ü	ü	ü
<i>Ausgewogenheit [K-04]</i>	ü	v	v	ü	t	ü
<i>Größe nichtstrukturierter Unterkapitel [K-05]</i>	ü	v	ü	ü	v	v
<i>Grammatik / Satzbau [K-06]</i>	v	v	v	v	v	v
<i>Textsatz [K-07]</i>	ü	v	v	v	ü	v
<i>Typographische Konventionen [K-08]</i>	n	v	ü	ü	ü	n
<i>Links [K-09]</i>	t	v	ü	v	n	ü
<i>Präzision [K-10]</i>	ü	v	v	v	v	ü
<i>Knappheit [K-11]</i>	v	v	v	v	v	v
<i>Fremdwörter [K-12]</i>	v	v	v	v	v	v
<i>Sprachliche Konsistenz [K-13]</i>	ü	ü	v	v	v	v
<i>Existenz [K-14]</i>	n	v	v	v	t	v
<i>Begriffsdefinitionen [K-15]</i>	-	v	v	t	v	v
<i>Verwendung [K-16]</i>	-	v	v	v	ü	ü
<i>Versionierung [K-17]</i>	ü	ü	ü	ü	n	v
<i>Index [K-18]</i>	n	-	-	v	-	-
<i>Prozessfreiheit [K-19]</i>	t	v	v	v	v	v
<i>Abbildungen [K-20]</i>	v	v	v	v	ü	v
<i>Vollständigkeit der inhaltlichen Aspekte [K-21]</i>	t	ü	v	ü	ü	t
<i>Identifizierbarkeit [K-22]</i>	ü	v	v	v	v	v
<i>Eindeutigkeit [K-23]</i>	v	v	v	ü	v	ü
<i>Nachverfolgbarkeit [K-24]</i>	v	ü	ü	v	ü	v
<i>Konsistenz (Widerspruchsfreiheit) [K-25]</i>	v	v	v	v	v	v

Kriterium	GO	CodeCover (Stefan)	CodeCover (Diana)	Bitart	JUST	EvoLab
<i>Zusammengehörigkeit & Ordnung [K-26]</i>	v	v	v	v	ü	v
<i>Verifizierbarkeit [K-27]</i>	v	v	ü	v	v	v
<i>Entwurfsoffenheit [K-28]</i>	n	v	v	ü	v	ü
<i>Abstraktionsgrad [K-29]</i>	v	v	v	v	v	v
<i>Beispiele [K-30]</i>	ü	v	v	v	v	n
<i>Methodik [K-31]</i>	v	v	v	v	v	v
<i>Zu treffende Aussagen pro funktionale Anforderung [K-32]</i>	ü	ü	t	t	v	t
<i>voll erfüllt</i>	13	25	22	21	20	19
<i>überwiegend erfüllt</i>	10	5	8	8	8	8
<i>teilweise erfüllt</i>	3	0	1	2	2	2
<i>nicht erfüllt</i>	4	0	0	0	1	2

Tabelle 1: Bewertungsübersicht

Die Ergebnisse bei den einzelnen Bewertungsdurchgängen unterscheiden sich miteinander deutlich. Die große Trennung ist zwischen den Ergebnissen der Industriespezifikation GO und den studentischen Spezifikationen zu sehen. GO schneidet insgesamt deshalb schlechter ab, da sich der Kriterienkatalog an der Lehre an der Universität Stuttgart orientiert und in der Industrie schlichtweg andere Prioritäten gesetzt werden. Das lässt sich insbesondere dadurch erkennen, dass in der Industrie der Prozess die entscheidende Rolle spielt, während diese Herangehensweise in der Lehre eher geringe Bedeutung hat. Auch fehlt das in der Lehre als äußerst wichtig deklarierte Begriffslexikon.

Eine andere interessante Erkenntnis aus diesen Daten ist, dass die Spezifikation Bitart, die ebenfalls von Studenten, aber an der Universität Bremen, geschrieben worden ist, ähnlich gut abschneidet wie die Stuttgarter Spezifikationen. Es deutet also anhand der vorhandenen Daten alles darauf hin, dass eine erhebliche Kluft zwischen Theorie und Praxis besteht. Andere Erklärungen für diese Unterschiede könnten auch die kleine Zahl an Gutachtern sein, wodurch die Ergebnisse zu sehr durch subjektives Empfinden der einzelnen Gutachter verschoben werden. Für fundierte Aussagen müssen also weitere Daten erhoben werden.

Nach den Bewertungen wurde die Spezifikation, die für die Kommentierung geeignet ist, ausgesucht, indem die Bewertungen, die sich für jedes Kriterium von voll erfüllt bis nicht erfüllt strecken konnten, gezählt wurden. Die Spezifikation, bei der die meisten Kriterien mit voll erfüllt bewertet wurden, war die Siegerspezifikation. Sie ist im dritten Teil des Abschlussdokumentes zu finden und heißt CodeCover. Diese Spezifikation ist im Rahmen eines Studienprojektes A der Universität Stuttgart entstanden. Sie beschreibt die Entwicklung eines Werkzeuges für die Überdeckungsmessung im Glass-Box-Test.

2.6 Kommentierung einer Spezifikation

Die Kommentierung der Spezifikation CodeCover erfolgte primär unter Zuhilfenahme der beiden Bewertungstemplates. In diesen Templates wurden bereits während der Bewertung zumindest die negativen Punkte, die bei den verschiedenen Kriterien aufgefallen sind, festgehalten. Die ausgewählte Spezifikation ist recht groß, in englisch geschrieben, enthält eine gute Versionierung, eine ausführliche GUI-Betrachtung, verschiedene Methodiken zur Beschreibung funktionaler Anforderungen, lässt sich gut lesen und ist speziell für uns nutzbar, weil uns die \LaTeX -Quellen zugänglich sind, mit denen die Spezifikation erstellt wurde.

Die kommentierte Spezifikation CodeCover ist mit einer gesonderten Beschreibung im gleichnamigen Anhang C zu finden.

3 Ergebnisse der Fachstudie

Während der Fachstudie haben wir uns umfassend mit dem Themenbereich Anforderungsspezifikation auseinandergesetzt. Dabei haben wir in der Literatur nach vorhandenen Erkenntnissen gesucht und anschließend neue Inhalte sowie neue Verknüpfungen von Wissen herausgearbeitet. Besonders hervorzuheben ist in diesem Zusammenhang der Kriterienkatalog. Der darin enthaltene Ansatz für die Bewertung der Qualität von Spezifikationen ist in Wissenschaft und Praxis bisher nicht vorhanden gewesen. Das besondere an den aufgestellten Bewertungskriterien ist, dass Sie sich von dem durch fachliche Fragestellungen getriebenen Review deutlich abgrenzen, in dem die Methodik der Aufbereitung der Anforderungen sowie Fragen der Form und des Dokumentaufbaus in den Mittelpunkt der Betrachtung gestellt werden.

Diese Herangehensweise bei der Bewertung von Spezifikationen kann helfen, allgemeingültige Aussagen zu treffen, welche Art der Anforderungsbeschreibung bei welcher Art von Anforderungen die ergiebigste zu sein scheint. Darüberhinaus bietet der Kriterienkatalog einen Überblick darüber, welche Inhalte eine Spezifikation haben sollte. Allerdings sind die Kriterien während der Fachstudie nur für bestimmte Produktarten eingesetzt worden, nämlich Werkzeuge und Informationssysteme. In folgenden, auf diese Arbeit aufbauenden, Untersuchungen muss die Einsetzbarkeit der Kriterien auf Spezifikationen untersucht werden, die beispielsweise im Bereich Embedded Systems angesiedelt sind.

Insgesamt ist der Kriterienkatalog, wie die Ergebnisse der Bewertungsphase gezeigt haben, deutlich an der Lehre an der Universität Stuttgart orientiert. Dadurch werden zwar auf der einen Seite Spezifikationen von der Studenten dieser Universität besser bewertet, auf der anderen Seite gibt das vor allem Praktikern die Möglichkeit ihre Anforderungsdokumente mit dem aktuellen Stand der Lehre/Wissenschaft zu vergleichen. Insofern bietet allein der Kriterienkatalog bereits wichtige Ansatzpunkte, die für Wissenschaft und Praxis interessant sind. Darüber hinaus wird für die Bewertung der Qualität von Spezifikationen ein neuer Schwerpunkt fokussiert.

Neben dem Kriterienkatalog ist auch die kommentierte Spezifikation ein wesentli-

ches Ergebnis unserer Arbeit. Eine solche Kommentierung einer vorhandenen Spezifikation ist komplett neu. Bisher gab es lediglich Beispielspezifikationen, die aber die wirklich wichtigen Sachverhalte ausgelassen haben. Auch verfügbar waren Inhaltsangaben oder Gliederungsvorschläge. Die vorliegende kommentierte Spezifikation bietet erstens eine vollständige Anforderungsdokumentation eines existierenden Produktes und zweitens Kommentare, in denen wir erklären, warum eine Beschreibung oder Darstellung besonders gelungen ist, was besser hätte gemacht werden können, oder welche anderen Möglichkeiten der Darstellung es gibt.

Damit bieten wir eine umfassende Hilfestellung für einen Einstieg in den Themenkomplex Spezifikationserstellung, -bearbeitung und -bewertung an, den es in dieser Form bisher noch nicht gab. Die Ergebnisse richten sich damit vor allem an Einsteiger – seien es Unternehmen, die erstmals eine Anforderungsdokumentation erstellen möchten, oder Studenten im Grundstudium, die ebenfalls das erste Mal mit dem Thema Spezifikationserstellung in Berührung kommen. Zurückblickend hätte uns selbst eine solche Arbeit im ersten Studienabschnitt zweifelsohne entscheidend geholfen.

Aber nicht nur für den Einsteiger gibt es interessante Ergebnisse. So enthält der Kriterienkatalog oder die kommentierte Spezifikation auch für den routinierten Software-Entwickler neue Ansätze, die er bisher noch nicht beachtet hat oder sich noch nicht bewusst war, dass diese einen Einfluss auf die Qualität haben können.

Zu den Ergebnissen der Fachstudie zählen zusammenfassend neben dem Kriterienkatalog oder der kommentierten Spezifikation auch weitere Dokumente. So kann es für weitere Arbeiten auf diesem Gebiet interessant sein, die Bewertungsergebnisse genauer zu analysieren oder weitere Bewertungen vorzunehmen, auch von mehreren Gutachten. Weiterführende Untersuchungen könnten sich auch mit der Frage beschäftigen, in wie weit die Bewertung einer Spezifikation mit unserem Kriterienkatalog mit der subjektiven Wahrnehmung der Entwickler korreliert, die mit der untersuchten Spezifikation gearbeitet haben. Es gibt darüberhinaus noch weitere Anwendungs- und Weiterführungsmöglichkeiten wie die Verbreiterung der Spezifikationssammlung auf weitere Universitäten, Unternehmen und Produktarten.

Abschließend ist aus unserer Sicht zu sagen, dass mit den Ergebnissen der Fachstu-

die ein guter Beitrag zum Themenkomplex Spezifikationserstellung, -bearbeitung und -bewertung geleistet werden konnte. Wir waren vor und sind nach dieser Arbeit noch mehr davon überzeugt, dass eine gut geschriebene Anforderungsspezifikation einen wesentlichen Faktor für den Erfolg eines Software-Projektes darstellt und somit bei jeder Entwicklung erstellt werden sollte.

A Literaturrecherche

A.1 Einleitung

In den folgenden Kapiteln haben wir zusammengetragen, welche Literatur wir für die Fachstudie gesichtet haben. In kurzen Darstellungen gehen wir darauf ein, welche Literatur Aussagen zu Anforderungen, Spezifikationen und Kriterienkatalogen machen.

Grundsätzlich sollen folgende Fragen geklärt werden:

- Welche Quellen definieren:
 - Anforderung (siehe Kapitel A.2)
 - Spezifikation (siehe Kapitel A.3)
- Was für Vorlagen und Beispielgliederungen gibt es für Spezifikationen? (siehe Kapitel A.4)
- Wie sind die Quellen nach Nützlichkeit zu bewerten?
- Gibt es bereits Beispielspezifikationen?
- Gibt es bereits Bewertungsschemata oder Kriterienkataloge? (siehe Kapitel A.5)
- Gibt es bereits ähnliche Studien?

Durch dieses Vorgehen erhoffen wir uns eine fundierte Basis für den weiteren Verlauf des Projekts. Die Literaturrecherche ermöglicht es uns vor allem den Hauptteil der Fachstudie – die Erstellung eines Bewertungsschemas – anzugehen und Ideen für den Kriterienkatalog zu finden.

A.2 Anforderung

Es gibt einige Definitionen von Anforderungen. Insbesondere wird auf deren Eigenschaften eingegangen.

A.2.1 IEEE 610.12

Der Standard IEEE Std 610.12-1990 (IEE, 1990) definiert eine Anforderung wie folgt:

requirement.

- (1) A condition or capability needed by a user to solve a problem or achieve an objective.
- (2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.
- (3) A documented representation of a condition or capability as in (1) or (2).

See also: design requirement; functional requirement; implementation requirement; interface requirement; performance requirement; physical requirement.

A.2.2 IEEE 1233

Der Standard IEEE 1233 „IEEE guide for developing system requirements specification“ (IEE, 1998a) definiert:

A **well-formed requirement** is a statement of system functionality (a capability) that can be validated, that must be met or possessed by a system to solve a customer problem or to achieve a customer objective, and that is qualified by measurable conditions and bounded by constraints.

A.2.3 Wikipedia

Der Wikipedia-Artikel „Anforderung (Informatik)“ definiert eine Anforderung wie folgt:

In der (Software-)Technik ist eine Anforderung (häufig engl. requirement) eine Aussage über eine zu erfüllende Eigenschaft oder zu erbringende Leistung eines Produktes, Systems oder Prozesses. Anforderungen werden üblicherweise in einem Lastenheft zusammengefasst, können in der Realität aber auch in nahezu beliebigen anderen Dokumenten zu finden sein, oder sind nicht dokumentiert.

Weiterhin werden die Anforderungen in funktionale und nichtfunktionale unterteilt, wobei letztere in verschiedene Arten eingeteilt werden:

- Zuverlässigkeit (Systemreife, Wiederherstellbarkeit, Fehlertoleranz)
- Aussehen und Handhabung (Look and Feel)
- Benutzbarkeit (Verständlichkeit, Erlernbarkeit, Bedienbarkeit)
- Leistung und Effizienz (Antwortzeiten, Ressourcenbedarf)
- Betrieb und Umgebungsbedingungen
- Wartbarkeit, Änderbarkeit (Analysierbarkeit, Stabilität, Prüfbarkeit)
- Portierbarkeit und Übertragbarkeit (Anpassbarkeit, Installierbarkeit, Konformität, Austauschbarkeit)
- Sicherheitsanforderungen (Vertraulichkeit, Datenintegrität, Verfügbarkeit)
- kulturelle und politische Anforderungen
- rechtliche Anforderungen

A.2.4 Robertson

Bei Robertson (Robertson u. Robertson, 1999) geht man ausführlich auf die Anforderungen ein. Es wird beschrieben, was Anforderungen sind, wie man sie findet. Auch auf die funktionalen und nichtfunktionalen Anforderungen wird eingegangen. Anschließend wird betrachtet, wie man sie in die Spezifikation einbringt und wie

man bereits vorhandene Anforderungen von früheren Projekten wiederverwendet. Es wird auch darauf eingegangen, wie man die fehlende Anforderungen findet.

A.2.5 Ludewig

Ludewig (Ludewig u. Lichter, 2007) gibt mehrere Definitionen einer Anforderung. Er beschreibt, welche verschiedenen Anforderungen es gibt (funktional, nicht-funktional, offene und latente, harte und weiche, objektivierbare und vage, natürlichsprachliche). Auf die Formulierung von nichtfunktionnnalen Anforderungen ist auch eingegangen.

A.3 Spezifikation

A.3.1 IEEE 610.12-1990

Der Standard IEEE Std 610.12-1990 „IEEE standard glossary of software engineering terminology“ (IEE, 1990) definiert folgende mit Spezifikation im Zusammenhang stehende Begriffe:

specification. A document that specifies, in a complete, precise, verifiable manner, the requirements, design, behavior, or other characteristics of a system or component, and, often, the procedures for determining whether these provisions have been satisfied.

See also: formal specification; product specification; requirements specification.

requirements specification. A document that specifies the requirements for a system or component. Typically included are functional requirements, performance requirements, interface requirements, design requirements, and development standards.

Contrast with: design description. See also: functional specification; performance specification.

product specification.

- (1) A document that specifies the design that production copies of a system or component must implement. Note: For software, this document describes the as-built version of the software. See also: design description.
- (2) A document that describes the characteristics of a planned or existing product for consideration by potential customers or users.

A.3.2 IEEE 830-1998

Der Standard IEEE 830-1998 „IEEE Recommended Practice for Software Requirements Specifications“ (IEE, 1998b) betrachtet die Umgebung einer Spezifikation und beschreibt die „recommended practice“ bei deren Erstellung. Der Standard fordert u.

a. folgende Eigenschaften einer Spezifikation:

- Correct
- Unambiguous
- Complete
- Consistent
- Ranked for importance and/or stability
- Verifiable
- Modifiable
- Traceable

Zitierungswürdig ist außerdem eine Auflistung von „benefits“, die eine Spezifikation bringt:

- Establish the basis for agreement between the customers and the suppliers on what the software product is to do.
- Reduce the development effort.
- Provide a basis for estimating costs and schedules.
- Provide a baseline for validation and verification.
- Facilitate transfer.
- Serve as a basis for enhancement.

A.3.3 IEEE 1012-2004

Der Standard IEEE Std 1012-2004 „IEEE Standard for Software Verification and Validation“ (IEE, 2005) detailliert noch den Begriff Softwareanforderungsspezifikation:

software requirements specification (SRS). Documentation of the essential requirements (functions, performance, design constraints, and attributes) of the software and its external interfaces.

A.3.4 Pressmann

Roger S. Pressman macht wenige Aussagen zum Thema Spezifikation (Pressman, 2005):

„In the context of computer-based system (and software), the term specification means different things to different people. A specification can be a written document, a set of graphical models, a formal mathematical model, a collection of usage scenarios, a prototype, or any combination of these.“

Pressmann beschreibt in einem eigenen Kapitel das Thema Requirements Engineering. Dieses ist jedoch eher prozesslastig angelegt und sehr UML lastig.

A.3.5 Wikipedia

Der Wikipedia-Hauptartikel „Spezifikation“ (Wikipedia, 2008d) betrachtet eine Spezifikation nur als eine Art Vereinbarung zwischen Auftraggeber und Kunde. Eine Spezifikation dient danach dafür, festzulegen, was vom Kunden bezahlt wird und damit in der Auslieferung enthalten sein muss. Der Artikel selbst ist sehr kurz gehalten und verweist auf Lastenheft und Pflichtenheft.

Der Artikel „Lastenheft“ (Wikipedia, 2008b) definiert dann:

Ein Lastenheft (teils auch auch **Anforderungsspezifikation**, Kundenspezifikation oder Requirements Specification) beschreibt die unmittelbaren Anforderungen durch den Besteller eines Produktes. Im Rahmen eines Werkvertrages oder Werkliefervertrages und der dazu gehörenden formellen Abnahme beschreibt das Lastenheft präzise die nachprüfbaren Leistungen und Lieferungen.

A.3.6 Robertson

In diesem Buch (Robertson u. Robertson, 1999) findet man im Bezug auf Spezifikationen, welche Inhalte sie haben sollte und woraus sich diese Inhalte genau zusammensetzen (In Kapitel Writing the specification). Im Kapitel Whither Requirements ist zur Spezifikation noch geschrieben, wann man sie veröffentlichen sollte, was sie

beinhalten sollte und woher man diese Informationen nimmt. Dieses Buch bietet eine Vorlage für die Spezifikation an.

A.3.7 Ludewig

Ludewig (Ludewig u. Lichter, 2007) gibt mehrere Definitionen einer Spezifikation, erläutert deren Inhalt, gibt eine Standardstruktur vor und die angestrebten Eigenschaften einer Spezifikation. Dazu gibt er Regeln, die zusätzlich erklärt werden vor. Zusätzlich wird auf Normen und Vorlagen anderer Quellen verwiesen. Auf Probleme bei der Abgrenzung und die Schwierigkeiten bei der Formalisierung wird eingegangen. Auch die Bedeutung und der Nutzen einer Spezifikation ist erwähnt.

A.4 Spezifikationsvorlagen

Gibt es bereits Vorlagen für Spezifikationen oder Standardgliederungen?

A.4.1 Ludwig und Lichter

Ludewig und Lichter (Ludewig u. Lichter, 2007) führen eine „Struktur der Spezifikation für eine Software X nach IEEE Std 830 (1998)“ (IEE, 1998b) auf. Dies ist eine Art deutsche Version des IEEE Standards:

1. Einleitung

- a) Zweck: Beschreibt den Zweck und den Leserkreis der Spezifikation.
- b) Einsatzbereich und Ziele: Gibt an, wo X eingesetzt werden soll und welche wesentlichen Funktionen es haben wird. Wo sinnvoll und notwendig, sollte auch definiert werden, was X nicht leisten wird. Beschreibt die mit X verfolgten Ziele.
- c) Definition: Entfällt und wird Begriffslexikon.
- d) Referenzierte Dokumente: Verzeichnet alle Dokumente, auf die in der Spezifikation verwiesen wird.
- e) Überblick über das Dokument: Beschreibt, wie der Rest der Spezifikation aufgebaut ist, insbesondere, wie Kapitel 3 strukturiert ist.

2. Allgemeine Beschreibung

- a) Einbettung: Beschreibt, wie X in seine Umgebung eingebettet ist und wie X mit den umgebenden Komponenten und Systemen zusammenspielt. Dazu werden die Schnittstellen, Kommunikationsprotokolle etc. definiert. Auch die Arbeitsumgebung: welche Hardwareanforderungen.
- b) Benutzerprofile: Charakterisiert die Benutzergruppen von X und die Voraussetzungen, die diese jeweils mitbringen (Ausbildung, Know-how, Sprache, ...).
- c) Einschränkungen: Dokumentiert Einschränkungen, die die Freiheit der

Entwicklung reduzieren (z. B. Ziel-Hardware, Basis-Software oder das anzuwendende Prozessmodell).

- d) Annahmen und Abhängigkeiten: Nennt explizit die Annahmen und externen Voraussetzungen, von denen bei der Spezifikation ausgegangen wurde.

3. Einzelanforderungen

- a) Anforderung i: Beschreiben die Anforderung i, und zwar so genau, dass bei der Verwendung der Spezifikation (für den Entwurf, die Testdaten u. s. w.) keine Rückfragen dazu notwendig sind.

Für das Kapitel 3 listen Ludewig und Lichter Informationen auf, die ihrer Meinung nach enthalten sein müssen:

- Funktionale Anforderungen
- Qualitätsanforderungen
- Leistungsanforderungen
- Einschränkungen des Entwurfs
- Definition der externen Schnittstellen zu anderen Systemen

Der IEEE 830 „IEEE Recommended Practice for Software Requirements Specifications“ (IEE, 1998b), welcher ja die Quelle dieser Gliederung war, beschreibt weiterhin acht mögliche Gliederungsvarianten für das wichtige Kapitel 3:

- organized by mode: Version 1
- organized by mode: Version 2
- organized by user class
- organized by object
- organized by feature
- organized by stimulus
- organized by functional hierarchy
- showing multiple organizations

Außerdem wird erklärt, wie man nach dieser Gliederung konform zum Standard IEEE 1233 „IEEE guide for developing system requirements specification“ (IEEE, 1998a) arbeiten kann.

A.4.2 IEEE 1233

(IEEE, 1998a) stellt folgende Vorlage vor:

- Introduction
 - System purpose
 - System scope
 - Definitions, acronyms, and abbreviations
 - References
 - System overview
- General system description
 - System context
 - System modes and states
 - Major system capabilities
 - Major system conditions
 - Major system constraints
 - User characteristics
 - Assumptions and dependencies
 - Operational scenarios
- System capabilities, conditions, and constraints (System behavior, exception handling, manufacturability, and deployment should be covered under each capability, condition, and constraint.)
 - Physical
 - * Construction

- * Durability
- * Adaptability
- * Environmental conditions
- System performance characteristics
- System security
- Information management
- System operations
 - * System human factors
 - * System maintainability
 - * System reliability
- Policy and regulation
- System life cycle sustainment
- System interfaces

A.4.3 Robertson

Das „Volere Requirements Specification Template“ (Robertson u. Robertson, 2007) betrachtet sich als Basis für andere Spezifikationen. Das Template enthält eine Beispielgliederung. Die Kapitel sind jedoch nicht leer, sondern enthalten Beschreibungen, was in diesen Kapiteln für eine konkrete Spezifikation eingetragen werden müsste. Es entsteht folgende Gliederung:

- Project Drivers
 - 1. The Purpose of the Project
 - 2. The Client, the Customer, and Other Stakeholders
 - 3. Users of the Product
- Project Constraints
 - 4. Mandated Constraints

- 5. Naming Conventions and Definitions
- 6. Relevant Facts and Assumptions
- Functional Requirements
 - 7. The Scope of the Work
 - 8. The Scope of the Product
 - 9. Functional and Data Requirements
- Nonfunctional Requirements
 - 10. Look and Feel Requirements
 - 11. Usability and Humanity Requirements
 - 12. Performance Requirements
 - 13. Operational and Environmental Requirements
 - 14. Maintainability and Support Requirements
 - 15. Security Requirements
 - 16. Cultural and Political Requirements
 - 17. Legal Requirements
- Project Issues
 - 18. Open Issues
 - 19. Off-the-Shelf Solutions
 - 20. New Problems
 - 21. Tasks
 - 22. Migration to the New Product
 - 23. Risks
 - 24. Costs
 - 25. User Documentation and Training
 - 26. Waiting Room

– 27. Ideas for Solutions

A.4.4 Klüver

(Klüver, 2008) hat Spezifikationsvorlagen gesammelt. Diese stellt er durch Auflistung der Gliederungspunkte vor. Zu den Gliederungen gehören:

- Balzert
- Pressman
- Retis (Aarau)
- Industriebeispiel 1
- Industriebeispiel 2 (Prozesssteuerung)
- Pfadler (Molzberger)
- Daniels (Online 3/83)
- Dreßler

A.4.5 Wikipedia

Der Artikel „Lastenheft“ (Wikipedia, 2008b) führt folgende Angaben auf, welche typisch für ein Lastenheft sind:

1. Ausgangssituation und Zielsetzung
2. Produkteinsatz
3. Produktübersicht
4. Funktionale Anforderungen
5. Nicht funktionale Anforderungen
 - Benutzbarkeit
 - Zuverlässigkeit
 - Effizienz

- Änderbarkeit
- Übertragbarkeit
- Wartbarkeit

6. Risikoakzeptanz

7. Skizze des Entwicklungszyklus und der Systemarchitektur oder auch ein Struktogramm

8. Lieferumfang

9. Abnahmekriterien

A.5 Kriterienkataloge & Bewertungsschemata

Vorhandene Kriterienkataloge für die Bewertung von Spezifikation konnten in einer ausgedehnten Recherche nicht gefunden werden. Dementsprechend sind auch auf den Kriterien aufbauende Bewertungsschemata nicht zu finden.

Als Suchanfrage wurden verschiedene Aspekte rund um die Bewertung von Spezifikationen genutzt, die aber mit den gängigen Suchmaschinen im Internet (IEEEExplore, Google Scholar, Google) kaum verwertbare Literatur lieferten. Zu der Reihe an Suchbegriffen zählten neben dem festen Bestandteil *software requirement specification* Wörter wie *criteria*, *good*, *validation* oder *assessment*. Auch eine Suche im deutschen Sprachraum blieb ergebnislos.

Zusammenfassend lässt sich sagen, dass Kriterienkataloge für die Bewertung von Spezifikation und darauf aufbauende Bewertungsschemata schwer auffindbar sind. Es lässt sich daraus ableiten, dass wir mit unserer Arbeit an einen Kriterienkatalog bei nahe null beginnen und dementsprechend mehr Aufwand dafür einplanen müssen.

Dennoch wurden bei der Recherche Dokumente gefunden, die beim Aufbau eines Kriterienkatalog und eines Bewertungsschemas nützlich sein könnten. In sehr vielen Dokumenten, die recherchiert wurden, ist zu erkennen, dass Autoren bei der Frage nach der Güte des Dokumentes Spezifikation auf die Nennung der Qualitätsmerkmale einer Anforderung zurückfallen (siehe A.3.2). Die Autoren scheinen davon auszugehen, dass eine Reihe von qualitativ „guten“ einzelnen Anforderungen auch ein „gutes“ Gesamtdokument ergeben. Ob diese Ansicht berechtigt ist, lässt sich in diesem Stadium des Projekts noch nicht abschließend beantworten, jedoch hat der Autor Zweifel daran.

Eine weitere große Menge an Dokumenten geben Hinweise, wie eine „gute“ Spezifikation erstellt werden oder welche Standardfehler vermieden werden sollten. Meist sind diese Dokumente Vorlesungs- oder Weiterbildungsunterlagen. Darüber hinaus lassen sich noch eine Reihe an beispielhaften Inhaltsverzeichnissen oder Templates finden.

A.6 Checklisten

Manche Quellen stellen Checklisten für Spezifikationen zusammen. Diese Kriterien sind meist für Reviews der Spezifikationen gedacht.

A.6.1 Drappa

(Drappa) hat eine „Checkliste zum Pflichtenheft“ geschrieben es werden folgende Gruppen von Fragen notiert:

- Allgemeine Fragen zur Unterlage
- Spezielle Fragen zum Pflichtenheft
 - Betrachtung des Umfelds
 - Abgleich der Entry-Unterlage Lastenheft
 - Aufgaben und Daten
 - DV-Grobkonzept
 - Qualitätsmerkmale
 - Exit-Kriterien

(Drappa) kommt insgesamt auf 52 Fragen, wovon besonders die Kategorie „Allgemeine Fragen zur Unterlage“ für uns einsetzbar sind. Den Abgleich zum Lastenheft können wir schlecht untersuchen – da fehlt uns das Material.

B Kriterienkatalog

B.1 Einleitung

B.1.1 Zweck des Dokuments

In diesem Dokument werden Kriterien vorgestellt, die bei der Qualitätsbewertung von Anforderungsspezifikationen genutzt werden sollen. Es dient den Bearbeitern der Fachstudie „Analyse und Kritik von Anforderungsspezifikationen“ als Arbeitsdokument für das weitere Vorgehen im Verlauf der Arbeit für die Fachstudie.

B.1.2 Aufbau des Dokuments

An dieses einleitende Kapitel schließt sich die Auflistung der Kriterien an. Diese beginnt mit der Beschreibung des Bewertungsschemas. Nach diesem erfolgt die Bewertung der Kriterien. Darauf folgend werden die Kriterien beschrieben. Zunächst werden grundlegende Kriterien definiert, die Basisanforderungen wie die Struktur, die Lesbarkeit oder die Benutzbarkeit, aber auch sprachliche Aspekte eines Dokuments überprüfen. Solche Basisanforderungen müssen an ein qualitativ hochwertiges Spezifikationsdokument gestellt werden, da der Leser ansonsten vom Inhalt abgelenkt werden könnte.

Auf den grundlegenden Kriterien folgend werden Kriterien definiert, die gezielt die Qualität einer Anforderungsspezifikationen bewerten sollen. Dazu zählt in erster Linie die Vollständigkeit der inhaltlichen Aspekte einer Spezifikation, d. h. eine Aufzählung der Inhalte, die eine gute Spezifikation in einer geeigneten Form beinhalten sollte.

In Spezifikationen werden Anforderungen an ein Softwaresystem festgehalten. Demnach müssen auch Kriterien aufgestellt werden, die eine Qualitätsbewertung für Anforderungen im Allgemeinen und für funktionale Anforderungen im Besonderen zulassen. Darunter sind Kriterien, die den Abstraktionsgrad, Identifizierbarkeit oder Entwurfsoffenheit untersuchen.

Im letzten Kapitel wird abgegrenzt, welche Aspekte einer Spezifikation ohne das Wissen über die Fachlichkeit nicht untersucht werden können.

Für die Erstellung dieses Bewertungskatalogs wurden verschiedene Quellen zur Erstellung der Kriterien genutzt. Zum Zwecke der Übersichtlichkeit ist bei der Beschreibung der Kriterien auf Verweise zur jeweils relevanten Quellen verzichtet worden. Stattdessen sind alle Quellen im Kapitel Literaturverzeichnis aufgelistet.

B.2 Kriterien

B.2.1 Bewertungsschema

Die Bewertung jedes Kriteriums erfolgt durch die Zuordnung in eine von vier Kategorien. Diese sind *voll erfüllt*, *überwiegend erfüllt*, *teilweise erfüllt* und *nicht erfüllt*. Für jedes in diesem Kapitel aufgeführte Kriterium wird unter dem Punkt Bewertung angegeben, wie die Verteilung auf die vier Kategorien erfolgt. Wenn beispielsweise bei einem Kriterium eine Prozentzahl errechnet werden kann, so wird angegeben, bis zu welchem Prozentwert das Kriterium als *voll erfüllt*, als *überwiegend erfüllt* usw. angesehen werden kann. Bei Kriterien, die rein auf dem subjektiven Empfinden des Gutachters basieren, sind für die vier Kategorien Beschreibungen angegeben, die dem Gutachter bei der Einordnung helfen.

Bei der Bewertung einiger Kriterien werden quantifizierende Adjektive wie beispielsweise „viele“, „einige“, „wenige“, „häufig“ oder „selten“ genutzt, um die Einordnung auf die vier Kategorien zu beschreiben. In nahezu allen Fällen sind diese Adjektive in Relation zur Größe des Dokuments zu verstehen.

Viele Kriterien lassen sich nur subjektiv bewerten. Deshalb sollten diese Kriterien von mehreren Gutachtern bewertet werden. Die Gutachter sollten anschließend ihre eigenen Befunde in einer Sitzung vortragen und zu einem Ergebnis zusammentragen.

B.2.2 Grundlegende Kriterien

B.2.2.1 Dokumenteigenschaften

B.2.2.1.1 Einhaltung grundlegender Dokumentanforderungen [K-01]

ID des Kriteriums: K-01

Beschreibung Dieses Kriterium bewertet das Dokument dahingehend, ob grundlegende Dokumenteigenschaften erfüllt sind. Dazu zählen Punkte wie korrekte Seitenzahlen, keine fehlenden Seiten, Übereinstimmung von Inhaltsverzeichnis und Glie-

derung, angebrachte Sprache und Symbolik (Smilies, Comic Sans, Ironie) u. Ä.

Zweck/Motivation Ist einer der oben genannten Punkte nicht erfüllt, kann das Dokument letztlich nicht verwendet werden, da Fehler bei solchen grundlegenden Dokumentanforderungen einerseits implizieren, dass auch beim eigentlichen Inhalt nicht sachgemäß gearbeitet wurde und andererseits kein zweckmäßiges Arbeiten mit dem Dokument möglich sein wird.

Ausführung/Anwendung Das gesamte Dokument wird auf die folgenden Punkte hin überprüft:

- inkorrekte Seitenzahlen – z. B. wenn eine Seitenzahl übersprungen wird
- fehlende Seiten
- Unstimmigkeit von Inhaltsverzeichnis und Gliederung
- ToDos
- offene Punkte oder Überarbeitungskommentare
- nicht angebrachte Sprache oder Symbolik (Smilies, Comic Sans, Ironie)

Ein Auftreten eines solchen Punktes wird in einer Negativliste dokumentiert.

Bewertung Es wird folgende Bewertung durch den Gutachter durchgeführt:

- *voll erfüllt* – Auf der Negativliste sind sehr wenige oder gar keine Punkte festgehalten. Zusätzlich sollte zu keiner Zeit das Gefühl entstehen, dass die wenigen handwerklichen Fehler vom Inhalt ablenken oder das Dokument weniger anwendbar machen.
- *überwiegend erfüllt* – Auf der Negativliste ist eine kleine Menge an verschiedenen Punkten dokumentiert. Für diese Kategorie kann beim Gutachter zeitweise das Gefühl entstehen, dass die handwerklichen Fehler vom Inhalt ablenken oder das Dokument weniger anwendbar machen.
- *teilweise erfüllt* – Auf der Negativliste ist eine große Menge an verschiedenen Punkten dokumentiert. Das Dokument kann durch viele handwerkliche Fehler nur schwerlich als anwendbar betitelt werden. Entsprechend wird der Gutachter sehr oft vom Inhalt abgelenkt.

- *nicht erfüllt* – Die Negativliste ist sehr lang. Der Gutachter hat zu kaum einem Zeitpunkt das Gefühl, dass handwerklich ordentlich gearbeitet wurde. Es gibt auf sehr vielen Seiten Fehler der oben stehenden Art.

B.2.2.1.2 Deckblatt [K-02]

ID des Kriteriums: K-02

Beschreibung Dieses Kriterium bewertet den Informationsgehalt des Deckblattes dahingehend, ob wichtige Informationen über das Dokument vorhanden sind. Zum Beispiel der oder die Autoren, der Titel und Weiteres.

Zweck/Motivation Fehlen solche grundlegenden Informationen, kann das leicht implizieren, dass die Autoren auch bei den inhaltlichen Aspekten Grundlegendes übersehen haben. Insbesondere das Nichtnennen des Autors/der Autoren lässt vermuten, dass dieser/diese nicht hinter dieser Arbeit stehen.

Ausführung/Anwendung Zunächst wird das Deckblatt darauf hin überprüft, ob alle wichtigen Informationen aufgeführt sind:

- Titel des Dokuments
- Projekttitel oder Bezugsrahmen
- Autor oder Autoren
- Version mit Datum
- Status

Ist einer oder mehrere dieser Punkte weder auf dem Deckblatt noch auf den ersten Seiten vorhanden, wird dies auf einer Negativliste vermerkt.

Bewertung Es wird folgende Bewertung durch den Gutachter durchgeführt:

- *voll erfüllt* – Auf der Negativliste sind gar keine Punkte festgehalten.
- *überwiegend erfüllt* – Auf der Negativliste sind nur die Punkte Status und/oder Version enthalten.
- *teilweise erfüllt* – Auf der Negativliste ist nur der Punkt Autor oder Autoren

enthalten. Zusätzlich können noch die Punkte aus *überwiegend erfüllt* gelistet sein.

- *nicht erfüllt* – Auf der Negativliste sind mehr als die in *überwiegend erfüllt* und *teilweise erfüllt* genannten Punkte enthalten.

B.2.2.1.3 Abbildungen und Tabellen (Äußere Form) [K-03]

ID des Kriteriums: K-03

Beschreibung Dieses Kriterium bewertet die Verwendung von Abbildungen und Tabellen. Dabei wird nur die äußere Form geprüft, wie beispielsweise die Beschriftung oder das Zitat im Text. Die Qualität einer Abbildung wird erst im Kriterium Abbildungen (Kapitel B.2.2.7) bewertet.

Zweck/Motivation Bei Abbildungen und Tabellen ist es wichtig, dass sie eindeutig beschriftet sind. Dies macht es erst möglich, sie im Text einzubinden und in ein Verzeichnis einzuordnen. Die Einbindung einer Abbildung oder einer Tabelle und auch die Erläuterung im Text kann das Verständnis eines Abschnitts erhöhen. Außerdem erfährt der Leser an geeigneter Stelle, wann er die Abbildung oder Tabelle sich anschauen sollte.

Ausführung/Anwendung Es werden die Abbildungen und Tabellen im Dokument validiert. Dazu wird folgende Checkliste genutzt:

- Abbildungen und Tabellen haben eindeutige Beschriftung (Name)
- Es gibt eine eindeutige Nummerierung
- Sie werden im Text zitiert (Es reicht auch der Verweis „auf das folgende Bild“ aus.)
- Räumliche Nähe zur Zitatstelle

Abweichungen werden auf einer Negativliste dokumentiert. Dabei werden nichtbeschriftete Abbildungen in jedem Fall in diese Liste aufgenommen, jedoch nicht beschriftete Tabellen nur, wenn der erklärende Text weit entfernt steht. Für die Zitierung im Text muss nicht der Identifikationscode genannt sein, es reicht der Verweis „auf das folgende Bild“ aus.

Bewertung Bei diesem Kriterium ist die Berechnung des Erfüllungsgrades E möglich. Dazu wird die Gesamtheit der Abbildungen und Tabellen gezählt oder im Abbildungs- und Tabellenverzeichnis nachgesehen. Diese Gesamtzahl wird mit vier multipliziert, da je Abbildung bzw. Tabelle vier Eigenschaften überprüft werden. Von der so errechneten Gesamtpunktzahl G wird nun die Anzahl der Punkte auf der Negativliste N abgezogen, durch die Gesamtpunktzahl dividiert.

$$E = \frac{G - N}{G}$$

In der folgenden Tabelle ist die Zuordnung des Erfüllungsgrades zu den vier Bewertungskategorien niedergeschrieben.

Kategorie	E
<i>voll erfüllt</i>	≥ 0.90
<i>überwiegend erfüllt</i>	< 0.90 und ≥ 0.60
<i>teilweise erfüllt</i>	< 0.60 und ≥ 0.30
<i>nicht erfüllt</i>	< 0.30

B.2.2.1.4 Ausgewogenheit [K-04]

ID des Kriteriums: K-04

Beschreibung Dieses Kriterium bewertet die Strukturierung des Dokuments, insbesondere die Größe der Kapitel bzw. Unterkapitel. Diese sollten ähnlich groß sein, wobei dies an den Seitenzahlen festgemacht wird.

Zweck/Motivation Sind in einem Dokument Kapitel oder Unterkapitel vorhanden, die sich deutlich in ihrer Größe unterscheiden, so deutet dies auf das Scheitern des/der Autoren hin, den Inhalt angemessen zu strukturieren. Solche weniger gut gegliederten Dokumente behindern den Leser unnötig, da die Motivation des Lesers abnimmt, wenn er nach der Lektüre eines Kapitels mit wenigen Seiten vor einem Kapitel mit der mehrfachen Größe steht.

Ausführung/Anwendung Es wird das Inhaltsverzeichnis betrachtet, ob es Kapitel gibt, die sich deutlich in ihrer Größe unterscheiden. Dabei wird ein eventuell vor-

handenes einleitendes Kapitel nicht betrachtet, da es in den meisten Fällen deutlich weniger Seiten benötigt, als der Hauptteil des Dokuments.

Ebenso wird nachfolgend mit den Unterkapiteln der einzelnen Kapitel vorgegangen. Dieses Verfahren wird auf alle Ebenen der Gliederung angewendet, die im Inhaltsverzeichnis aufgeführt sind. Der deutliche Größenunterschied sollte an der Gesamtheit an Seitenzahlen festgemacht werden. Bei der Betrachtung der Kapitel bei einem 100-Seiten-Dokument mit fünf Kapiteln sollte beispielsweise jedes Kapitel etwa 20 Seiten stark sein. Beim Betrachten von Unterkapiteln wird als Gesamtheit die Seitenanzahl des übergeordneten Kapitels und die Anzahl an Unterkapiteln herangezogen.

Weicht die Seitenanzahl eines Kapitels oder Unterkapitels stark vom errechneten Wert ab (20 Seiten bei 100-Seiten-Dokument mit fünf Kapiteln), dann wird das Auftreten auf einer Negativliste dokumentiert. Ein starkes Abweichen wäre im oben genannten Beispiel ein Kapitel von unter 10 Seiten, dass kein einleitendes Kapitel ist. Letztlich muss der Gutachter aber subjektiv entscheiden, was er als ausgewogen oder nichtausgewogen betrachtet.

Bewertung Es wird folgende Bewertung durch den Gutachter durchgeführt:

- *voll erfüllt* – Das Dokument ist ausgewogen gegliedert und somit dem Leser sehr zugänglich.
- *überwiegend erfüllt* – Für diese Kategorie ist das Dokument im Großen und Ganzen ausgewogen gegliedert.
- *teilweise erfüllt* – An vielen Stellen ist die Gliederung nicht ausgewogen und sollte überarbeitet werden.
- *nicht erfüllt* – Eine Arbeit mit dem Dokument wird erheblich erschwert.

B.2.2.1.5 Größe nichtstrukturierter Unterkapitel [K-05]

ID des Kriteriums: K-05

Beschreibung Dieses Kriterium bewertet die Granularität der Gliederung. Dabei wird überprüft, ob eine Gliederung zu grob ist, d. h. sehr viele unstrukturierte Unter-

kapitel, oder zu fein gegliedert ist. Letzteres ist beispielsweise gegeben, wenn jeder Gliederungspunkt einer Anwendungsfallbeschreibung als Unterkapitel festgehalten ist.

Zweck/Motivation Die beiden Extreme, die das Kriterium schlecht bewerten, sind zum einen eine zu grobe Gliederung, wodurch Inhalte nur schwer gefunden werden können (langes Durchsuchen unstrukturierter Unterkapitel). Zum anderen ist eine zu feine Gliederung als ungünstig anzusehen, da der Inhalt in sehr kleine Stückchen geteilt wird und so der Leser immer wieder in seinem Lesefluss unterbrochen wird.

Ausführung/Anwendung Auch bei diesem Kriterium wird das Inhaltsverzeichnis zur Hand genommen. Darin werden zunächst Unterkapitel gesucht, die gemessen an der Gesamtseitenanzahl des Dokuments sehr viele Seiten umfassen. Die gefundenen nichtuntergliederten Unterkapitel werden nun untersucht, ob eine weitere Gliederung der Inhalte möglich ist. Wenn das der Fall ist, wird dieses Unterkapitel auf einer Negativliste dokumentiert.

Anschließend werden alle Unterkapitel nochmals daraufhin überprüft, ob mehr als drei Unterkapitel einer Hierarchieebene auf einer Seite stehen. Ist dies der Fall wird die entsprechende Seite im Dokument überprüft, ob die Granularität verringert werden kann. Wenn das der Fall ist, wird dieses Unterkapitel auf einer Negativliste dokumentiert.

Bewertung Es wird folgende Bewertung durch den Gutachter durchgeführt:

- *voll erfüllt* – Die Granularität der Gliederung ist angemessen und somit dem Leser sehr zugänglich.
- *überwiegend erfüllt* – Für diese Kategorie ist die Granularität der Gliederung im Großen und Ganzen angemessen.
- *teilweise erfüllt* – An vielen Stellen ist die Granularität der Gliederung nicht angemessen und sollte überarbeitet werden.
- *nicht erfüllt* – Die Granularität der Gliederung ist gänzlich unangemessen. Eine Arbeit mit dem Dokument wird erheblich erschwert.

B.2.2.2 Lesbarkeit/Übersichtlichkeit

B.2.2.2.1 Grammatik/Satzbau [K-06]

ID des Kriteriums: K-06

Beschreibung Dieses Kriterium bewertet den Satzbau und die Grammatik dahingehend, dass keine komplizierten Konstruktionen verwendet werden. Dazu zählen unter anderem auch zu tief verschachtelte und zu lange Sätze.

Zweck/Motivation In allen Texten der Spezifikation ist es wichtig, dass die Inhalte verständlich beschrieben sind. Dabei sollte korrekte Grammatik benutzt werden, aber keine komplizierten Konstruktionen, die nicht auf Anhieb klar verständlich sind. Das Wichtigste ist und bleibt der Inhalt. Es sollte keine zusätzliche Zeit entstehen, um den Inhalt zu verstehen.

Ausführung/Anwendung Diese Punkte sind nur subjektiv bewertbar. In der Spezifikation sollten keine Sätze vorkommen, die folgende Eigenschaften haben:

- lange Nebensätze
- Schachtelsätze, die eine zu große Tiefe haben
- lange Satzkonstrukte

Bewertung

- *voll erfüllt*, wenn die Spezifikation beim ersten Mal lesen sofort verständlich ist.
- *überwiegend erfüllt*, wenn die Spezifikation bis auf Kleinigkeiten sofort verständlich ist. Es kommen selten lange Satzkonstrukte oder Nebensätze vor.
- *teilweise erfüllt*, wenn die Spezifikation nicht leicht verständlich ist und der Benutzer oft den gleichen Abschnitt lesen muss. Es kommen also oft lange Satzkonstrukte oder Nebensätze vor, die das Verständnis erschweren, werden oft verwendet.
- *nicht erfüllt*, wenn die Spezifikation nicht verständlich formuliert ist. Dies ist der Fall, wenn alle Punkte der oben genannten Liste häufig vorkommen.

B.2.2.2.2 Textsatz [K-07]

ID des Kriteriums: K-07

Beschreibung Dieses Kriterium bewertet die äußere Form des Textes. Sie soll den Leser unterstützen, was unter anderem durch die Größe von Überschriften, aber auch der Schriftgröße und -Art beeinflusst wird.

Zweck/Motivation Der Leser darf nicht durch die Form des Textes irritiert werden und somit beim Lesen des Textes behindert werden. Aus diesem Grund ist es wichtig, dass das Dokument eine Form hat, die den Leser unterstützt. Um dieses zu gewährleisten, gibt es verschiedene Punkte, die dazu gegeben sein müssen.

Ausführung/Anwendung In der Spezifikation sollte Folgendes begutachtet werden:

- Schriftgröße, die zu groß oder zu klein gewählt wurde
- passende Schriftart
- passende Verwendung von Serifen
- Größe der Überschriften
- Verwendung von Kopfzeile und Fußnoten
- Verwendung von Blocksatz

Einige dieser Punkte sind nur subjektiv bewertbar.

Bewertung

- *voll erfüllt*, wenn die Spezifikation ein gutes äußeres Erscheinungsbild hat. Das bedeutet, dass die Schriftgröße ungefähr zwischen 10 und 12pt liegt. Beim Drucken muss das Dokument noch lesbar sein. Die Schrift muss schlicht und gut lesbar sein. Der Text ist in Blocksatz. Auch alle anderen oben genannten Kriterien sind erfüllt.
- *überwiegend erfüllt*, wenn die Spezifikation bis auf Kleinigkeiten ein gutes äußeres Erscheinungsbild hat. Es wurde keine Kopfzeile verwendet. Auch Serifen wurden nicht richtig verwendet (Serifenlos in der Überschrift, der Text mit Serifen).

- *teilweise erfüllt*, wenn die Spezifikation, zusätzlich zu den in *voll erfüllt* und *überwiegend erfüllt* genannten Punkten, eine eher zu große oder zu kleine Schrift hat (bis 9 pt ist sie zu klein, ab 12pt zu groß), nicht im Blocktext steht und eine Schrift verwendet wird, die nicht gut lesbar ist.
- *nicht erfüllt*, wenn die Spezifikation keine der oben genannten Punkte erfüllt.

B.2.2.2.3 Typographische Konventionen [K-08]

ID des Kriteriums: K-08

Beschreibung Dieses Kriterium bewertet, ob typographische Konventionen vorhanden sind. Zu diesen Konventionen zählen die Hervorhebung von:

- Beispielen
- Verlinkungen
- Wörtern des Begriffslexikons
- Code
- Schlüsselwörtern.

Zweck/Motivation In einem Dokument, das sehr groß werden kann, ist eine einheitliche Darstellung von den oben genannten Punkten wichtig. Es erleichtert dem Leser, z. B. Beispiele im Dokument schnell zu erkennen oder zu finden.

Ausführung/Anwendung Es muss ein separates Kapitel geben, in dem die typographischen Konventionen erläutert werden und alle Konventionen auf den ersten Blick ersichtlich werden. Diese Konventionen müssen das gesamte Dokument über eingehalten werden. Um dies zu überprüfen, muss der Gutachter die Spezifikation lesen. Als Erstes muss er überprüfen, ob es ein separates Kapitel gibt, das typographische Konventionen enthält. Ist dies nicht der Fall, wird beim nachfolgenden Schritt darauf geachtet, ob dennoch Konventionen vereinbart sind. Als Nächstes muss überprüft werden, ob die Konventionen eingehalten wurden. Dazu muss der Gutachter aber die ganze Spezifikation lesen. Er muss herausfinden, ob die Konventionen konsequent eingehalten wurden oder nicht.

Bewertung

- *voll erfüllt*, Es gibt ein Kapitel mit typographischen Konventionen. Zudem werden alle oben genannten Punkte erwähnt, erläutert und in der Spezifikation eingehalten.
- *überwiegend erfüllt*, wenn die Konventionen für Beispiele, Code und Wörter aus dem Begriffslexikon erwähnt, erläutert und in der Spezifikation eingehalten werden. Außerdem ist das Kriterium *überwiegend erfüllt*, wenn es kein separates Kapitel über die typographischen Konventionen gibt (sie also nur nicht beschrieben sind), trotzdem aber über das gesamte Dokument eingehalten werden.
- *teilweise erfüllt*, wenn typographische Konventionen zwar erläutert werden, sie aber in der Spezifikation nicht eingehalten werden. Hier gehört auch dazu, wenn Konventionen nur teilweise beschrieben und nur teilweise eingehalten werden. Außerdem fällt hierunter auch, wenn es typographische Konventionen gibt, sie aber nur für Verlinkungen und Schlüsselwörter eingehalten werden.
- *nicht erfüllt*, wenn keine typographischen Konventionen erwähnt oder erläutert werden und auch sonst nicht ersichtlich ist, dass irgendeine Art von Konvention eingehalten wurde.

B.2.2.2.4 Links [K-09]

ID des Kriteriums: K-09

Beschreibung Dieses Kriterium bewertet, ob die Spezifikation leserfreundlich ist. Genauer bedeutet dies, dass der Umgang mit dem Dokument einfach sein muss. Dazu zählt, dass ein Leser nicht lange nach Abschnitten, die er schon im Inhaltsverzeichnis gefunden hat, scrollen oder suchen muss. Es muss Links geben, die den Leser sofort zu der gewünschten Stelle navigieren.

Zweck/Motivation Für einen Leser ist es sehr mühsam, sich Stellen von Hand zu suchen. Das Lesen gestaltet sich einfach, wenn der Leser durch einen einfachen Klick zur gewünschten Stelle kommt.

Ausführung/Anwendung Der Gutachter muss die Spezifikation lesen und dabei überprüfen, ob das Inhaltsverzeichnis, sowie auch alle Verweise im Dokument gelinkt sind. Zusätzlich sollten Bookmarks verwendet werden. Wenn das Dokument nur als PDF-Datei vorhanden ist, dann wird das Kriterium so bewertet wie im nächsten Punkt ausgeführt. Sollte zusätzlich eine Word-Datei vorliegen, so kann beim Nichtvorhandensein von Links, das Word-Dokument geprüft werden.

Bewertung

- *voll erfüllt*, wenn das Inhaltsverzeichnis, sowie auch alle Verweise im Dokument gelinkt sind. Zusätzlich sollten Bookmarks verwendet werden.
- *überwiegend erfüllt*, wenn nur Bookmarks nicht verwendet werden.
- *teilweise erfüllt*, wenn zusätzlich Verweise im Dokument nicht gelinkt sind. Auch wenn einige Links an die falsche Stelle führen, ist diese Kategorie zu wählen.
- *nicht erfüllt*, wenn keine Links verwendet werden.

B.2.2.3 Sprache

B.2.2.3.1 Präzision [K-10]

ID des Kriteriums: K-10

Beschreibung Das Kriterium Präzision untersucht, ob bei der Spezifikation syntaktische Konstruktionen verwendet werden, die schwammig oder unsicher sind. Im Gegensatz zur Eindeutigkeit (Kapitel B.2.4.1.2) bewertet dieses Kriterium nur die Sprache.

Zweck/Motivation Der Leser einer Spezifikation soll den Eindruck bekommen, ein präzises Dokument vor sich zu haben. Findet er Anforderungen in der Möglichkeitsform, haben diese eine geringe Aussagekraft und werten die Spezifikation ab. Nach vagen Formulierungen kann nicht entwickelt werden und der Entwurf von soliden Testfällen wird erschwert.

Viele Autoren schrecken davor zurück, konkrete Aussagen zu machen. Dies zeigt,

dass an dieser Stelle Anforderungen unklar waren. Dieses Kriterium reflektiert also auch Missstände bei der Anforderungsanalyse.

Ausführung/Anwendung Die Anwendung des Kriteriums erfordert das Lesen des Dokuments. Unsichere Phrasen werden markiert und gezählt, um die spätere Bewertung durchzuführen:

- Verwendung der Möglichkeitsform (Konjunktiv) (z. B. sollte, würde, könnte)
- Wörter die einen Wunsch ausdrücken (z. B. „möglichst“, „wenn es geht“, „eventuell“, „es ist wünschenswert“)
- Offene Listenenden (z. B. „etc.“, „usw.“, „u. a.“, „o. ä.“; Bsp.: „Das System soll sich an die Standards A, B, usw. halten.“)

Bewertung Für die Bewertung dieses Kriteriums ist eine rein quantitative Einstufung weniger geeignet, deshalb wird folgende Bewertung durch den Gutachter durchgeführt:

- *voll erfüllt* – Es wurden sehr selten unpräzise Phrasen verwendet. Diese standen nicht im Zusammenhang mit Anforderungen (z. B. in der Einleitung oder einem Begleittext).
- *überwiegend erfüllt* – Es wurden mehr als selten aber wiederholt unpräzise Phrasen verwendet. Ein Teil davon stand im Zusammenhang mit Anforderungen und schränkte somit deren Aussagekraft ein.
- *teilweise erfüllt* – Es werden häufiger unpräzise Phrasen verwendet. Mehrere wichtige Anforderungen werden durch vage Formulierungen verwässert.
- *nicht erfüllt* – Unpräzise Phrasen tauchen ständig auf. Die Autoren haben scheinbar nicht auf deren Vermeidung geachtet und der Großteil der Anforderungen ist durch die Ungenauigkeit nicht als Basis für eine solide Entwicklung geeignet.

B.2.2.3.2 Knappheit [K-11]

ID des Kriteriums: K-11

Beschreibung Dieses Kriterium untersucht die Länge der Spezifikation und die Länge von Abschnitten. Es wird untersucht, ob die Ausführlichkeit, mit der die Spezifikation geschrieben wurde an jeder Stelle berechtigt ist. Dies folgt der Idee: „Was man auf einer Seite spezifizieren kann, sollte man nicht auf drei Seiten ausdehnen.“

Zweck/Motivation Teilweise ist zu beobachten, dass die Spezifikation ihren Anspruch, ein fachliches Dokument zu sein, verloren hat und mehr zu einer Art Belletristikersatz wird. Eine Spezifikation sollte so kurz und knapp geschrieben werden, damit alle Anforderungen abgedeckt sind. Sprachliches Beiwerk und ausschweifende Formulierungen sind bei einer Spezifikation fehl am Platz.

Zu große Dokumente sind schwer zu warten und die Benutzbarkeit leidet. Deshalb rechtfertigen nur Komplexität und Vielzahl der Funktionen eines Systems, dass eine Spezifikation mehrere hundert Seiten hat.

Ausführung/Anwendung Zur Untersuchung dieses Kriteriums ist das Lesen des Dokuments erforderlich. Eigentliches Augenmerk sollte auf den Prosa-Teil der Spezifikation geworfen werden. Dabei wird geschaut, ob:

- Sätze dienlich für die Definition von Anforderungen sind oder keinen Mehrwert enthalten
- Sätze künstlich verlängert werden, um sie interessanter klingen zu lassen
- Absätze auch kürzer hätten formuliert werden können, ohne Inhalt wegzulassen
- klare und nachvollziehbare grafische Darstellungen (Prozessketten, Diagramme) unnötig ausführlich beschrieben worden sind

Dabei ist es nicht die Aufgabe des Gutachters, die Abschnitte gedanklich kürzer zu formulieren. Meist reicht das Empfinden des Gutachters aus, um zu beurteilen, ob der Inhalt in Relation zur Absatzlänge steht.

Bewertung Die Bewertung wird auf dem bekannten Schema durchgeführt:

- *voll erfüllt* – Die Spezifikation formuliert alle Anforderungen knapp und hat keine unnötigen Absätze.

- *überwiegend erfüllt* – An wenigen Stellen sind die Absätze zu lang geraten, ohne das dies durch den Inhalt gerechtfertigt ist. Unnötiges sprachliches Beiwerk ist selten zu finden.
- *teilweise erfüllt* – An mehreren Stellen haben die Autoren unpassende stilistische Konstruktionen verwendet. Die Länge der Abschnitte ist meist ohne Grund zu lang.
- *nicht erfüllt* – Die Autoren haben sich scheinbar in künstlerisches Beiwerk verliebt und zieren damit nahezu jeden Absatz; die Spezifikation wirkt insgesamt viel zu lang, ohne das dies durch die Fachlichkeit gefordert wird.

B.2.2.3.3 Fremdwörter [K-12]

ID des Kriteriums: K-12

Beschreibung Dieses Kriterium untersucht, ob die verwendeten Fremdwörter richtig gebraucht werden und nur auf diese zurückgegriffen wird, die für die Anwendungsdomäne verständlich sind. Es sollen Wortneuschöpfungen vermieden werden, die beispielsweise aus der Zusammensetzung eines deutschen und eines englischen Wortes entstehen.

Zweck/Motivation Fachwörter sind wichtig für eine Anwendungsdomäne. Insbesondere in der Informatik stammen jedoch viele Fachwörter aus dem Englischen und sind damit faktisch Fremdwörter. Der massive Gebrauch dieser Fremdwörter stellt einen schlechten Stil in einer deutschen Spezifikation dar und sollte deshalb vermieden werden. Das Gleiche gilt für Fremdwörter in anderen Sprachen.

Dieses Kriterium soll besonders Wortneuschöpfungen und bedeutungsferne Benutzung von Fremdwörtern bestrafen. Dies dient dem Leser, da Irritationen und Unklarheiten meist vermieden werden.

Ausführung/Anwendung Bei der Untersuchung dieses Kriterium ist erneut das Lesen erforderlich. Fremdwörter tauchen jedoch nicht nur in Prosa-Absätzen auf. Oft sind sie auch Sinnträger in Diagrammen und Prozessbeschreibungen. Für das Kriterium soll untersucht werden:

- Ist ein Fremdwort nötig oder gäbe es eine passende deutsche Entsprechung? (z. B. *herunterladen/übertragen* anstatt *downloaden*, *angeklickt* anstatt *gecheckt*)
- Sind Fremdwörter konsistent mit der gleichen Bedeutung verwendet worden?
- Sind Fremdwörter mit der korrekten Semantik verwendet? (z. B. *Performanz* ist die Bezeichnung für die Sprachverwendung (das Sprechen) und nicht mit *Performance* zu verwechseln)
- Sind Fremdwörter nötig, ist die Orthografie korrekt? (z. B. *Plug-In* wird meist falsch geschrieben.)
- Sind Fremdwörter nötig, wird auf eine Konjugation verzichtet (kein *gedownloadet*)
- Keine Wortneuschöpfungen, die meist aus Anpassung eines Fremdworts an die Dokumentsprache entstehen (z. B. *escapet*).
- Sind Fachwörter als Fremdwörter verwendet, deren Bedeutung unklar sein könne, sollten diese im Begriffslexikon erläutert werden? (z. B. Instrumentieren)

Bewertung Die Bewertung wird auf dem bekannten Schema durchgeführt:

- *voll erfüllt* – Die Spezifikation verzichtet auf unnötige Fremdwörter. Fremdwörter, die als Fachwörter in der Anwendungsdomäne nötig sind, werden im Begriffslexikon erklärt.
- *überwiegend erfüllt* – Die Spezifikation führt nur wenige neue Fremdwörter ein. Einige dieser Fremdwörter hätten auch durch deutsche Wörter ersetzt werden können.
- *teilweise erfüllt* – Viele Fremdwörter werden verwendet, einige davon in falscher oder inkonsistenter Bedeutung.
- *nicht erfüllt* – Die Spezifikation nutzt Fremdwörter und Wortneuschöpfungen ständig. Die Autoren haben z. B. viele englische Verben einfach übernommen und der deutschen Konjugation unterzogen.

B.2.2.3.4 Sprachliche Konsistenz [K-13]

ID des Kriteriums: K-13

Beschreibung Dieses Kriterium bewertet, ob sprachliche und syntaktische Konstruktionen einheitlich verwendet wurden. Die konsistente Verwendung der Begriffe des Begriffslexikons wird im Kriterium B.2.2.4.3 untersucht und bleibt hier unberücksichtigt.

Zweck/Motivation Sprachliche Inkonsistenz macht das Lesen der Spezifikation unnötig schwierig und sorgt oft für Verwirrungen. Ursache ist meist, dass verschiedene Autoren oder ein Autor in verschiedenen Situationen Beiträge für die Spezifikation verfasst haben.

Die Spezifikation sollte wie *ein* Dokument erscheinen und homogen für den Leser sein. So wird die Verständlichkeit erhöht und die Spezifikation gewinnt an Nutzbarkeit.

Ausführung/Anwendung Der Gutachter hat bei der Untersuchung dieses Kriteriums folgende Punkte durchzuarbeiten. Das ausführliche Lesen der Spezifikation ist dafür nötig. Manche Punkte haben nur für Spezifikationen in deutscher Sprache Sinn und sollten deshalb nur bei letzteren untersucht werden.

- Werden Datums- und Zeitangaben immer im gleichen Format angegeben? (z. B. 01.01.1984 08:11)
- Sind Beschreibungen von Anforderungen im gleichen Tempus (also z. B. immer im Präsens)?
- Beschreibungen sollten nur in einer Sprache erfolgen. Die Spezifikation sollte also nicht in Deutsch und Englisch geschrieben sein.
- Verweise auf andere Kapitel sollten immer im gleichen Stil geschehen. (z. B. *siehe Kapitel 3.8.9* oder *siehe Punkt „Besonderheiten“ auf Seite 391*)
- Auch Begriffe, die nicht im Begriffslexikon stehen, werden synonymfrei verwendet. Wortwiederholungen werden zur sprachlichen Konsistenz in Kauf genommen (z. B. keine synonymhafte Verwendung von *Hochschule*, *Universität* und *Uni*).

- Werden Auflistungen mit Anstrichen häufig verwendet, sollten diese im gleichen Stil formuliert sein (z. B. immer in Stichworten und ohne erzwungene Großbuchstaben).
- Anführungszeichen sollten konsistent und korrekt verwendet werden (also z. B. „diese“ im Deutschen).
- Wird konsistent neue deutsche Rechtschreibung verwendet? Etwas eingeschränkt: Werden Wörter, die durch die neue deutsche Rechtschreibung geändert wurden, konsistent neu oder alt geschrieben (Fotograf vs. Photograph; darüberhin- ausgehende vs. darüber hinausgehende Informationen).
- Konsistente Verwendung des Genitiv-„[e]s“ (Dokumentes vs. Dokuments), wo das „e“ optional ist.
- Konsistente Zusammen- bzw. Getrennschreibung bei gleichen Wortpaaren (z. B. Indexgröße vs. Index-Größe).

Ist die sprachliche Konsistenz verletzt, notiert sich dies der Gutachter auf einer Negativliste.

Bewertung Die Bewertung wird auf dem bekannten Schema durchgeführt:

- *voll erfüllt* – Die Autoren haben auf die sprachliche Konsistenz geachtet und es gibt nur sehr wenige Einträge auf der Negativliste.
- *überwiegend erfüllt* – Die Negativliste hat einige Einträge. Verstöße gegen die Konsistenz tauchen jedoch nur über Kapitelgrenzen hinweg auf und stören damit nicht direkt beim Lesen.
- *teilweise erfüllt* – Die Negativliste hat eine Reihe von störenden Einträgen. Diese verwirren den Leser und behindern das Verständnis der Spezifikation teilweise.
- *nicht erfüllt* – Die Autoren haben nicht auf sprachliche Konsistenz geachtet. Auch innerhalb von Absätzen tauchen auffallende Inkonsistenzen auf. Dies behindert den Leser merklich und wird von ihm als störend empfunden.

B.2.2.4 Begriffslexikon

B.2.2.4.1 Existenz [K-14]

ID des Kriteriums: K-14

Beschreibung Dieses Kriterium bewertet, ob es ein Begriffslexikon gibt. Ist dieses Kriterium nicht erfüllt, können die Kriterien B.2.2.4.2 und B.2.2.4.3 nicht untersucht werden.

Zweck/Motivation Ein Begriffslexikon definiert die wichtigsten Begriffe der Anwendungsdomäne. Es hat damit einerseits den Zweck, Unklarheiten zwischen Kunden- und Entwicklerseite zu vermeiden. Andererseits müssen Wörter, welche im Anwendungskontext eine spezielle Bedeutung haben, nur an einer Stelle im Begriffslexikon erklärt werden und nicht an mehreren Stellen in der Spezifikation. Damit werden Inkonsistenzen bei der Bedeutung verhindert und die Spezifikation bleibt griffig, da die definierten Begriffe nicht erst erläutert werden müssen.

Ausführung/Anwendung Für die Prüfung dieses Kriteriums ist lediglich zu untersuchen, ob es ein Begriffslexikon (manchmal auch Glossar genannt) gibt. Zusätzlich sollte untersucht werden, ob dieses Dokument innerhalb der Spezifikation (z. B. als Anhang) zu finden ist oder als eigenes Dokument gepflegt wird. Schlussendlich sollte der Gutachter auf einen Verwendungshinweis innerhalb der Einleitung schauen. Wenn ein Leser nicht weiß, dass es ein Begriffslexikon gibt, nützt es ihm wenig.

Bewertung Die Bewertung wird auf dem bekannten Schema durchgeführt:

- *voll erfüllt* – Es gibt ein Begriffslexikon, welches im Spezifikationsdokument (z. B. als Anhang) zu finden ist. Das Begriffslexikon wird bei den Richtlinien für das Dokument erwähnt.
- *überwiegend erfüllt* – Es gibt ein Begriffslexikon, welches jedoch in einem eigenen Dokument gepflegt wird. Die Existenz und die Quelle für das Begriffslexikon werden in der Einleitung der Spezifikation erwähnt.
- *teilweise erfüllt* – Es gibt ein Begriffslexikon, das nicht erwähnt wird und scheinbar geringe Verwendung genossen hat.

- *nicht erfüllt* – Es gibt kein Begriffslexikon oder ebenbürtiges Dokument.

B.2.2.4.2 Begriffsdefinitionen [K-15]

ID des Kriteriums: K-15

Beschreibung Dieses Kriterium betrachtet die Definitionen im Begriffslexikon und untersucht diese auf Verständlichkeit. Auch soll ausgeschlossen werden, dass Begriffsdefinitionen keine Anforderungen enthalten, die in der Spezifikation festgehalten werden sollten.

Anmerkung: Dieses Kriterium kann selbstverständlich nur bewertet werden, wenn es überhaupt ein Begriffslexikon gibt.

Zweck/Motivation Oftmals wird ein Begriffslexikon als Zwang wahrgenommen und die positiven Effekte eines gut geführten Begriffslexikons vergessen. Dies führt zu dem, dass unsinnige Begriffe im Begriffslexikon definiert werden oder die Definitionen nicht durchdacht sind.

Zum anderen stellt ein Begriffslexikon, dass Kundenseite und Entwickler nicht aneinander vorbeireden, ohne in der Spezifikation ständig alle Fachwörter erklären zu müssen.

Ausführung/Anwendung Der Gutachter hat die folgenden Punkte zu untersuchen. Dafür ist die Sichtung des Begriffslexikons ausreichend.

- Die Definitionen sollten verständlich sein. Der Leser sollte rein mit der Erklärung des Begriffs dessen Bedeutung verstehen.
- Die Definitionen sollten keinen Interpretationsspielraum offen lassen sondern eindeutig sein.
- Die Definitionen sollten keine Wörter aus dem Begriff wiederverwenden (z. B. „Die Arbeitsdauer ist die Dauer, die der Entwickler arbeitet.“).
- Die Definitionen können andere Definitionen voraussetzen. Es sollte jedoch keine zyklischen Erklärungen geben (z. B. „Eine Hochschule ist eine Universität“ und umgekehrt).

- Eine Definition eines Begriffs sollte keine Anforderungen enthalten. Dies trifft besonders für Begriffe der Anwendungsdomäne zu (z. B. „Eine Person hat einen Namen, der maximal 20 Zeichen lang ist.“).
- Das Begriffslexikon (bzw. das Glossar) sollte auch unbekannte Abkürzungen enthalten, wenn diese nicht in einem eigenen Abkürzungsverzeichnis erklärt werden. Neben der Erklärung sollte auch das ausgeschriebene Wort oder die vollständige Wortgruppe angegeben werden.

Bewertung Die Bewertung wird auf dem bekannten Schema durchgeführt:

- *voll erfüllt* – Die Definitionen sind durchgängig verständlich und hilfreich. Das Begriffslexikon ist gut verwendbar.
- *überwiegend erfüllt* – Die Definitionen sind, bis auf wenige Ausnahmen, gut definiert und reichen als Erklärungen aus. Es gibt Abzüge bei der Bewertung nur durch Kleinigkeiten, welche den Einsatz des Begriffslexikons nur in Teilen behindert.
- *teilweise erfüllt* – Das Begriffslexikon enthält mehrere Definitionen, die für einen Leser nicht nachvollziehbar sind. Die Definitionen sind zu detailliert und enthalten Anforderungen oder die Definitionen sind oberflächlich geführt worden.
- *nicht erfüllt* – Das Begriffslexikon ist schlecht geführt und enthält kaum verständliche Definitionen. Die Definitionen können auch rekursiver Natur sein oder sich auf Begriffe stützen, die nicht selbst definiert sind. Das Begriffslexikon ist nicht hilfreich für den Umgang mit der Spezifikation.

B.2.2.4.3 Verwendung [K-16]

ID des Kriteriums: K-16

Beschreibung Dieses Kriterium betrachtet die tatsächliche Verwendung des Begriffslexikons. Es wird untersucht, ob alle wichtigen Begriffe auch definiert sind und im Gegenzug auch alle definierten Begriffe tatsächlich verwendet werden.

Anmerkung: Dieses Kriterium kann selbstverständlich nur bewertet werden, wenn

es überhaupt ein solches gibt.

Zweck/Motivation Das beste Begriffslexikon verliert seinen Zweck, wenn es nicht konsequent benutzt wird. Sind Synonyme anstatt definierter Begriffe verwendet worden, verwirrt dies den Leser. Eine Kennzeichnung definierter Begriffe erleichtert darüberhinaus die Benutzbarkeit der Spezifikation.

Ausführung/Anwendung Für die Überprüfung, ob alle Begriffe auch verwendet werden, ist im Grunde das Suchen der Begriffe des Begriffslexikons in der Spezifikation ausreichend. Für die Beantwortung aller folgenden Punkte, ist jedoch das Lesen der Spezifikation nötig. Zu untersuchen ist, ob:

- alle definierten Begriffe des Begriffslexikons auch in der Spezifikation (oder im Begriffslexikon selbst) verwendet worden sind
- die im Begriffslexikon definierten Begriffe in der Spezifikation besonders hervorgehoben sind
- die im Begriffslexikon definierten Begriffe in der Spezifikation in das Begriffslexikon verlinkt sind (z. B. über PDF-Hyperlinks)
- alle u. U. unklaren oder mehrdeutigen Begriffe, die in der Spezifikation verwendet werden, auch im Begriffslexikon definiert sind
- zusätzlich Definitionen an anderen Stellen in der Spezifikation (z. B. in Fußnoten) festgehalten werden
- keine Synonyme für definierte Begriffe verwendet werden (z. B. replizieren, comitten, hochladen, übertragen)
- keine Definitionen in der Spezifikation zitiert werden oder sogar andere Definitionen in der Spezifikation zu finden sind
- die Begriffe in der Spezifikation entgegen ihrer Definition im Begriffslexikon verwendet werden

Bewertung Die Bewertung wird auf dem bekannten Schema durchgeführt:

- *voll erfüllt* – Die Spezifikation erfüllt alle oben genannten Punkte. Die Begriffe müssen nicht bei jedem Auftauchen gekennzeichnet sein. Dies hat negativen Einfluss auf die Lesbarkeit. Das erste Auftauchen im Kapitel ist ausreichend

für die Erfüllung der Bewertungskategorie *voll erfüllt*.

- *überwiegend erfüllt* – Alle Begriffe werden benutzt. Die Benutzbarkeit des Begriffslexikons ist jedoch eingeschränkt, da die Begriffe nicht hervorgehoben sind und keine Verlinkungen in das Begriffslexikon existieren.
- *teilweise erfüllt* – Es werden öfter Synonyme für die Begriffe des Begriffslexikons benutzt. Dies schränkt die Verwendbarkeit der Definitionen des Begriffslexikons merklich ein. Einige Begriffe werden nicht verwendet, sind jedoch im Begriffslexikon definiert worden.
- *nicht erfüllt* – Die Begriffe des Begriffslexikons werden nur sporadisch verwendet. Synonyme sind eher die Regel als die Ausnahme. Einige Begriffe werden nicht im Sinne ihrer Definition verwendet. Das Begriffslexikon wurde scheinbar nur teilweise bei der Erstellung der Spezifikation genutzt.

B.2.2.5 Benutzbarkeit

B.2.2.5.1 Versionierung [K-17]

ID des Kriteriums: K-17

Beschreibung Dieses Kriterium bewertet den Versionsstand der Spezifikation. Es muss ein Versionsstand auf dem Deckblatt oder vor Beginn des ersten Kapitels vorhanden sein.

Zweck/Motivation Bei einer Spezifikation muss nachvollziehbar sein, wann welche Teile der Spezifikation geändert wurden. Zudem muss gegeben sein, dass das Dokument einen einheitlichen Versionsstand hat.

Ausführung/Anwendung Der Versionsstand muss auf dem Deckblatt oder einer der ersten Seiten vor dem ersten Kapitel stehen. Er kann auch auf jeder Seite der Spezifikation vorkommen, muss aber unbedingt eindeutig sein. Die Spezifikation muss nur oberflächlich angeschaut werden.

Bewertung

- *voll erfüllt*, wenn ein Versionsstand vorhanden ist, der einheitlich ist und auf

jeder Seite der Spezifikation steht.

- *überwiegend erfüllt*, wenn ein einheitlicher Versionsstand auf dem Deckblatt oder noch vor dem ersten Kapitel vorhanden ist.
- *teilweise erfüllt*, wenn ein Versionsstand vorhanden ist, bei dem sofort klar wird, dass er nicht einheitlich ist.
- *nicht erfüllt*, wenn kein Versionsstand vorhanden ist.

B.2.2.5.2 Index [K-18]

ID des Kriteriums: K-18

Beschreibung Dieses Kriterium bewertet den Index (Schlagwortverzeichnis), der in einer Spezifikation enthalten ist. Er erhöht die Benutzbarkeit bei größeren Dokumenten. Ist das Dokument kleiner als 200 Seiten wird dieses Kriterium nicht angewendet.

Zweck/Motivation In großen Dokumenten fällt es dem Leser oft schwer, sich zurecht zu finden. Dies liegt an der hohen Seitenzahl und an der Menge des Inhalts. Die Benutzbarkeit wird erhöht, falls die Möglichkeit gegeben ist, dass der Leser wichtige Wörter nachschlagen kann und so die Stellen schneller findet, an denen der Begriff vorkommt. Im Index stehen auch Wörter, die nicht im Begriffslexikon auftauchen, weil sie eindeutig sind. Es kann aber trotz dessen nützlich sein, die zu finden.

Ausführung/Anwendung Die Spezifikation muss nur oberflächlich angeschaut werden. Es wird schnell ersichtlich, ob ein Index vorhanden ist, oder nicht. Bei Dokumenten, die größer sind als 200 Seiten, muss ein Index vorhanden sein.

Bewertung

- *voll erfüllt*, wenn ein Schlagwortverzeichnis bei einem Dokument über 200 Seiten vorhanden ist, das vollständig erscheint.
- *überwiegend erfüllt*, wenn ein Schlagwortverzeichnis bei einem Dokument über 200 Seiten vorhanden ist, bei dem bei näherer Betrachtung klar wird, dass nicht alle relevanten Begriffe enthalten sind.

- *teilweise erfüllt*, wenn ein Schlagwortverzeichnis bei einem Dokument über 200 Seiten vorhanden ist, bei dem sofort klar wird, dass nicht alle relevanten Begriffe enthalten sind.
- *nicht erfüllt*, wenn kein Schlagwortverzeichnis bei einem Dokument über 200 Seiten vorhanden ist.

B.2.2.6 Prozessfreiheit [K-19]

ID des Kriteriums: K-19

Beschreibung Dieses Kriterium kontrolliert Spezifikationsdokumente dahingehend, dass keine Aussagen über Kosten, Meilensteine, Abgabetermine, Software-Entwicklungsmethoden, Qualitätsprozess, Abnahme-Prozeduren oder Priorisierung enthalten sind.

Zweck/Motivation Die Spezifikation ist ein Dokument, welches darstellt, was für ein Softwaresystem gebaut werden soll. Dabei sind Informationen über den Prozess, also das Vorgehen, wie das System erstellt werden soll, hinderlich. Solche Managementinformationen sollten in extra Dokumenten, wie zum Beispiel dem Projektplan beschrieben werden. Insbesondere das Vermischen von Anforderungen und Managementinformationen soll vermieden werden.

Ausführung/Anwendung Das Dokument wird auf Aussagen über die folgenden Punkte hin überprüft:

- Kosten
- Meilensteine
- Abgabetermine
- Software-Entwicklungsmethoden
- Qualitätsprozess
- Abnahme-Prozeduren
- Priorisierung
- Support und Wartung

Ein Auftreten eines solchen Punktes wird in einer Negativliste dokumentiert. Dabei ist ein bloßer Verweis auf ein anderes Dokument nicht zu berücksichtigen. Lediglich ein Ausführen in der Spezifikation ist auf der Negativliste zu vermerken.

Bewertung Es wird folgende Bewertung durch den Gutachter durchgeführt:

- *voll erfüllt* – Auf der Negativliste sind gar keine Punkte festgehalten. Es sind keine Aussagen über den Prozess enthalten.
- *überwiegend erfüllt* – Die Negativliste ist nicht leer. Diese Bewertung darf jedoch nur vergeben werden, wenn die Prozessinhalte auf eine Stelle im Dokument konzentriert sind und so einfach in ein anderes ausgelagert werden könnten.
- *teilweise erfüllt* – Die Negativliste ist nicht leer und wenige der oben genannten Punkte sind über das gesamte Dokument verstreut, so dass sie sich mit Anforderungen vermischen. Mit einigem Aufwand könnte man diese Vermischung wieder aufheben.
- *nicht erfüllt* – Die Negativliste ist nicht leer. Die oben genannten Punkte sind über das gesamte Dokument verstreut, so dass sie fest mit den Anforderungen verbunden sind.

B.2.2.7 Abbildungen [K-20]

ID des Kriteriums: K-20

Beschreibung Dieses Kriterium untersucht den Einsatz von Abbildungen in einer Spezifikation. Dabei wird insbesondere auf die Eigenschaften der Abbildungen Wert gelegt, wie beispielsweise Lesbarkeit, Verständlichkeit oder Übersichtlichkeit. Außerdem sollten verwendete Notationen und Symboliken in einer Legende oder im Text erläutert werden, wenn sie keinem allgemeingültigen Standard folgen.

Zweck/Motivation Mit diesem Kriterium wird im Gegensatz zum Kriterium Abbildungen und Tabellen (Äußere Form) im Kapitel B.2.2.1.3 die Qualität der Abbildung begutachtet. Wenn die Abbildungen in einem Dokument von der äußeren Form her gut integriert sind, heißt dies noch nicht, dass sie auch von angemessener Güte sind.

Abbildungen, die den Text nicht sinnvoll ergänzen oder handwerklich schlecht gemacht sind, werden die Qualität der Spezifikation wesentlich herabsetzen.

Ausführung/Anwendung Es werden die Abbildungen im Dokument validiert. Dazu wird folgende Checkliste genutzt:

- Abbildungen sind eine sinnvolle Ergänzung des Textes.
- Sie sind lesbar, verständlich und übersichtlich.
- Notationen und Symboliken sind erklärt, wenn nicht allgemein bekannt.
- Sie werden (wenn nötig) im Text weiter erläutert.

Abweichungen werden auf einer Negativliste dokumentiert. Eine Abweichung ist bei diesem Kriterium sehr stark von der subjektiven Meinung des Gutachters abhängig.

Bewertung Bei diesem Kriterium ist die Berechnung des Erfüllungsgrades E möglich. Dazu wird die Gesamtheit der Abbildungen gezählt oder im Abbildungsverzeichnis nachgesehen. Diese Gesamtzahl wird mit drei multipliziert, da je Abbildung drei Eigenschaften überprüft werden. Von der so errechneten Gesamtpunktzahl G wird nun die Anzahl der Punkte auf der Negativliste N abgezogen, durch die Gesamtpunktzahl dividiert.

$$E = \frac{G - N}{G}$$

In der folgenden Tabelle ist die Zuordnung des Erfüllungsgrades zu den vier Bewertungskategorien niedergeschrieben.

Kategorie	E
<i>voll erfüllt</i>	≥ 0.90
<i>überwiegend erfüllt</i>	< 0.90 und ≥ 0.60
<i>teilweise erfüllt</i>	< 0.60 und ≥ 0.30
<i>nicht erfüllt</i>	< 0.30

B.2.3 Vollständigkeit der inhaltlichen Aspekte [K-21]

ID des Kriteriums: K-21

Beschreibung Dieses Kriterium untersucht die Vollständigkeit der Spezifikation nach einer Liste von Inhaltsaspekten. Bewertet wird, ob alle geforderten Inhalte der Liste in der Spezifikation zu finden sind oder ausgelassen wurden. Dabei spielt die genaue Art der Darstellung keine Rolle. Die Bewertung orientiert sich an der Zahl der ausgelassenen Inhaltsaspekte.

Zweck/Motivation Es gibt eine Reihe von möglichen Gliederungen für Spezifikationen. Diese unterscheiden sich nach Anwendungsbereich und Umfeld des Urhebers erheblich. Viele Inhalte findet man in den verschiedenen Gliederungen wieder. Einige durchaus wichtige Inhaltsaspekte sind dagegen eher selten vertreten.

Es ist nachvollziehbar, dass die Spezifikation Aussagen zu allen relevanten Bereichen der spezifizierten Software machen muss. Erst durch die vollständige Abdeckung aller Anforderungen an das Softwaresystem kann die Spezifikation als vollständig gelten. Die geforderten anforderungsfremden Gliederungspunkte sind wichtig für den Umgang mit der Spezifikation und korrelieren teilweise mit anderen Kriterien dieses Katalogs.

Ausführung/Anwendung Bei der Untersuchung dieses Kriteriums reicht die Sichtung des Inhaltsverzeichnisses allein nicht aus. Zur vollständigen Untersuchung müssen manche Inhaltsaspekte evtl. in der Spezifikation gesucht werden. Taucht ein geforderter Inhaltsaspekt in der zu untersuchenden Spezifikation auf, notiert sich der Gutachter einen Punkt dafür. Taucht ein Inhaltsaspekt nicht auf, muss der Gutachter diesen Umstand wie folgt handhaben:

- der Inhaltsaspekt taucht an anderer Stelle auf oder hat einen anderen Namen als auf der Liste → der Inhaltsaspekt zählt als enthalten und erhält einen Punkt
- der Inhaltsaspekt enthält nur eine Überschrift und einen Vermerk wie „Es gibt keine Anforderungen an die Sicherheit des Systems.“; hat der Gutachter keinen begründeten Verdacht, dass dieser Inhaltsaspekt eigentlich hätte beschrieben werden müssen, zählt Letzterer als enthalten und erhält einen Punkt
- der geforderte Inhaltsaspekt taucht nicht in der Spezifikation auf; es ist aber offensichtlich, dass der Inhaltsaspekt in der vorliegenden Spezifikation nicht benötigt wird → der Inhaltsaspekt zählt als enthalten und erhält einen Punkt

- der geforderte Inhaltsaspekt taucht in keiner befriedigenden Form irgendwo in der Spezifikation auf → der Inhaltsaspekt zählt als nicht enthalten und erhält keinen Punkt

Manche Inhaltsaspekte erlauben eine differenziertere Bewertung als 0 Punkte ↔ 1 Punkt. Für diese Inhaltsaspekte kann der Gutachter auch Viertelpunkte (0; 0,25; 0,5; 0,75; 1) vergeben. Der Gutachter kann auf dieses Mittel zurückgreifen, wenn ein Inhaltsaspekt in der Spezifikation enthalten ist, jedoch nicht ausführlich genug erörtert wurde.

Es folgt eine Liste mit Inhaltsaspekten, welche in einer geeigneten Form enthalten sein sollten. Eine kurze Beschreibung sorgt für die Abgrenzung und zeigt evtl. Variationsmöglichkeiten auf. Die aufgeführten Beschreibungen bilden darüberhinaus Anhaltspunkte, um die Objektivität der differenzierten Bewertung zu erhöhen. Die Inhaltsaspekte sind hierarchisiert und die Bewertung wird nur für die Inhaltsaspekte durchgeführt, welche sich selbst nicht weiter unterteilen.

- *Versionsverzeichnis*
Wer hat wann was am Dokument in welcher Version geändert?
- *Inhaltsverzeichnis*
Auflistung der Gliederungspunkte mit Seitenzahlen
- *Abbildungs- und Tabellenverzeichnis*
Gemeinsame oder getrennte Liste der Abbildungen und Tabellen mit Beschriftung und Seitenzahl.
- *Einleitung*
 - *Zweck des Dokuments*
Wozu dient die Spezifikation? Wie ist sie in den Entwicklungsprozess eingebunden? Wer gehört zum Leserkreis?
 - *Einsatzbereich und Ziele*
Grobe Vorstellung der zu entwickelten Software auf bis zu zwei Seiten. Wo soll die Software eingesetzt werden? Welche wesentlichen Funktionen soll sie haben? Evtl. auch eine Beschreibung, was die Software nicht leisten soll. Welche Ziele verfolgt der Kunde mit dieser Software?

- *Referenzierte Dokumente*

Auf welche Dokumente beruft oder bezieht sich die Spezifikation (Begriffslexikon, Standard XY, DIN83, andere Spezifikationen)?

- *Konventionen für das Dokument*

Welche Konventionen gelten für die Spezifikation? Welche typografische Konventionen werden eingesetzt? Gibt es Hinweise an den Leser?

- *Überblick über das Dokument*

Folgt das Dokument einer Standardgliederung? Wie ist das Dokument aufgebaut? Welche Kapitel gibt es und wie sind sie strukturiert? Gibt es einen Anhang? Wenn ja, was enthält dieser?

- *Allgemeine Beschreibung*

- *Systemüberblick*

Kurzer Überblick über das gesamte System. Aus welchen Bestandteilen oder Komponenten besteht das System (im Rahmen der Entwurfsneutralität)? Der Leser kann dadurch die späteren Beschreibungen besser einordnen.

- *Einschränkungen an die Entwicklung*

Dokumentiert Einschränkungen, welche die Freiheit der Entwicklung reduzieren.

- * *Arbeitsumgebung*

Auf welcher Ziel-Hardware wird die Software eingesetzt (für Server und Clients)? Welche Software und Bibliotheken werden benötigt. Welche Betriebssysteme sollen unterstützt werden?

- * *Systemumfeld/Produktumgebung*

Welche anderen Systeme gibt es, die mit dem zu entwickelten System interagieren? Dies kann z. B. durch ein Schema visualisiert werden. Welche Schnittstellen nutzt das System von anderen Systemen (z. B. eine bestimmte Datenbank)?

- * *Schnittstellen*

Welche Schnittstellen bietet das System an? Die Benutzerschnittstelle

wird hier nicht untersucht; dafür aber Hardware-Schnittstellen (z. B. ein bestimmter Bus) und Software-Schnittstellen (z. B. Dateiformate für den Import, unterstützte Protokolle).

– *Benutzerprofile & Akteure*

Welche Benutzerprofile gibt es? Welche Anforderungen werden an die Benutzer der einzelnen Profile gestellt. Gibt es bestimmte Rollen beim Kunden, die diesen Benutzerprofilen entsprechen? Sind sie evtl. identisch?

• *Funktionale Anforderungen*

Hierzu gehören API-Beschreibungen, Use-Case-Diagramme und -Beschreibungen sowie Prosabeschreibungen, welche Anforderungen an die Funktionen des Systems enthalten. Funktionen sind meist durch das Eingabe-Verarbeitung-Ausgabe-Paradigma gekennzeichnet.

• *Benutzerschnittstelle*

Gibt es eine Benutzeroberfläche, sollte diese spezifiziert werden. Für grafische Benutzerschnittstellen, eignen sich meist GUI-Entwürfe mit Begleittexten.

• *Nichtfunktionale Anforderungen*

Die nichtfunktionalen Anforderungen stellen einen Katalog von Anforderungen dar, welche meist für alle funktionalen Anforderungen gelten.

– *Bedienbarkeit*

Sollen bestimmte Bedienkonzepte eingehalten werden? Werden Anforderungen an bestimmte Benutzungs-Paradigmen gefordert. Wird ein Handbuch erstellt?

– *Quantitative Anforderungen*

* *Leistungsanforderungen*

Gibt es Anforderungen an Antwortzeit, Durchsatz, Zeit für das Starten des Programms?

* *Mengengerüst*

Gibt es Anforderungen an die Zahl der Benutzer, Transaktionen, Anfragen pro Zeiteinheit. Sind für bestimmte fachliche Objekte Anforderungen an das Mengenmodell gefordert (z. B. 60 Bestellungen pro

Minute, 3.000 Bestellungen pro Stunde sowie 40.000 Bestellungen pro Tag müssen mindestens verarbeitet werden können)?

* *Eingaben und Längen*

Gibt es Längenunter- oder -obergrenzen für Attribute fachlicher Objekte (z. B. Namen haben maximal 40 Zeichen)?

– *Robustheit/Verhalten bei Störungen*

Gibt es grundsätzliche Verhaltenskonzepte des Systems bei fehlerhaften Benutzereingaben oder falschen externen Aufrufen? Wie verhält sich das System, wenn benutzte Fremdsysteme nicht verfügbar sind? Gibt es Anforderungen an Transaktionsschutz, Recovery nach Fehlern?

– *Verfügbarkeit*

Gibt es Anforderungen an die maximalen Ausfallzeiten, Ausfallraten, Mean Time Between Failures (MTBF)?

– *Sicherheit*

Gibt es Anforderungen an den Schutz von Teilen des Systems vor unbefugtem Zugriff? Sollen Nachrichten und Dateien verschlüsselt werden? Gibt es zu unterstützende Authentifizierungs- und Autorisierungsprotokolle?

– *Portabilität & Kompatibilität*

Gibt es Anforderungen an eine einfache Portierung auf andere Hardware, ein anderes Betriebssystem, andere Fremdsysteme?

– *Internationalisierung*

Welche Sprachen muss die Benutzeroberfläche unterstützen? Wie soll das System die Sprache auswählen. In welchen Sprachen wird das Handbuch angeboten? Müssen bestimmte Zeichensätze beim Import/Export unterstützt werden?

– *Protokollierung*

Sollen bestimmte Ereignisse (z. B. Fehlerfälle) protokolliert werden? In welchem Format sollte dies geschehen? Sollen Protokolle über die Zeit bereinigt werden?

– *Installation*

Wie kann die Software installiert werden. Welche Anforderungen werden an den Benutzer dahingehend gestellt. Wie schnell soll die Installation abgehandelt sein?

– *Erweiterbarkeit*

Sind bereits zum Spezifikationszeitpunkt Erweiterungen geplant? Welche zukünftigen Anforderungen sollen bereits bei der Entwicklung berücksichtigt aber noch nicht umgesetzt werden?

– *Wartbarkeit*

Auf welche Weise soll das System eine Diagnose unterstützen? Gibt es Anforderungen, um die Langzeitwartung zu unterstützen?

– *Gesetzliche Einschränkungen*

Tangiert das System Richtlinien zum Datenschutz? Müssen bestimmte Auflagen beim Betrieb des Systems eingehalten werden?

Bewertung Für die Bewertung dieses Kriteriums werden die erreichten Punkte pro Inhaltsaspekt aufsummiert. Für die insgesamt 28 Inhaltsaspekte können also bei vollständiger Erfüllung des Kriteriums 28 Punkte erreicht werden. Die Einordnung in die Bewertungskategorien geschieht nach den erreichten Punkten wie folgt:

Kategorie	Punkte
<i>voll erfüllt</i>	$\geq 25,0$
<i>überwiegend erfüllt</i>	$< 25,0$ und $\geq 17,0$
<i>teilweise erfüllt</i>	$< 17,0$ und $\geq 8,0$
<i>nicht erfüllt</i>	$< 8,0$

B.2.4 Allgemeine Kriterien für Anforderungen

B.2.4.1 Eigenschaften der Anforderungen

B.2.4.1.1 Identifizierbarkeit [K-22]

ID des Kriteriums: K-22

Beschreibung Dieses Kriterium bewertet, ob jede Anforderungen als solche erkennbar und somit abgrenzbar zur nächsten Anforderung ist. Es wird dabei lediglich bewertet, ob die genannte Abgrenzung im Dokument möglich ist. Um das Kriterium zu erfüllen, ist es nicht notwendig, dass Anforderungen besonders formatiert werden, um sie hervorzuheben.

Zweck/Motivation Die Identifizierbarkeit ist ein entscheidendes Kriterium für die folgenden Kriterien, da bei diesen davon ausgegangen wird, dass eine einzelne Anforderung identifizierbar ist. Darüber hinaus bringt die Abgrenzung zwischen verschiedenen Anforderungen Vorteile während der Umsetzung der Spezifikation mit sich, in dem eine leichtere Zuordnung zu Arbeitspaketen möglich ist. Außerdem werden Techniken wie Traceability durch die Identifizierbarkeit wesentlich unterstützt. Auch für Managementangelegenheiten ist die Abgrenzung zwischen verschiedenen Anforderungen hilfreich, beispielsweise bei der Vertragsgestaltung oder der Planung.

Ausführung/Anwendung Das Dokument muss vollständig gelesen werden. Jede erkannte Anforderung wird im Dokument markiert und dann daraufhin überprüft, ob in der Präzisierung und Beschreibung der Anforderung eine weitere Anforderung genannt ist. Eine solche Verschachtelung von Anforderungen wird auf einer Negativliste dokumentiert. Ist das komplette Dokument durchgearbeitet, kann aus den markierten Anforderungen eine Liste der in der Spezifikation vorhandenen Anforderungen aufgestellt werden. Auf diese Liste wird in den folgenden Kriterien zurückgegriffen.

Bewertung Es wird folgende Bewertung durch den Gutachter durchgeführt:

- *voll erfüllt* – Auf der Negativliste sind sehr wenige oder gar keine Punkte festgehalten. Der Gutachter kann eine vollständige Liste der Anforderungen aus der Spezifikation extrahieren.
- *überwiegend erfüllt* – Auf der Negativliste ist eine kleine Menge an Punkten dokumentiert. Das Erstellen einer Liste der Anforderungen wird durch einige schlecht von einander abgegrenzter Anforderungen erschwert. Insgesamt ist die entstandene Liste der Anforderungen aber gut geeignet, um als Gesamterfassung der in der Spezifikation enthaltenen Anforderungen zu dienen.

- *teilweise erfüllt* – Auf der Negativliste ist eine große Menge an Punkten dokumentiert. Die Liste der Anforderungen lässt sich nur mühsam erstellen. An vielen Stellen im Dokument lassen sich Anforderungen kaum abgrenzen.
- *nicht erfüllt* – Die Negativliste ist sehr lang. Das Aufstellen der Liste der Anforderungen ist kaum möglich.

B.2.4.1.2 Eindeutigkeit [K-23]

ID des Kriteriums: K-23

Beschreibung Dieses Kriterium bewertet, ob alle Aussagen in der Spezifikation eindeutig formuliert sind. Es darf keinen Spielraum für Interpretationen geben.

Zweck/Motivation Um eine gewisse Qualität des entstehenden Produktes gewährleisten zu können, müssen alle Aussagen eindeutig formuliert sein. Ist dies nicht der Fall, kann es durch unterschiedliche Interpretationen zu Missverständnissen und anderen Auffassungen der Leser kommen. Dies wirkt sich unmittelbar negativ auf das entstehende Produkt aus. Es muss nämlich klar sein, was genau entwickelt werden soll.

Ausführung/Anwendung Alle Aussagen müssen eindeutig formuliert sein. Das bedeutet, dass kein Raum zur Interpretation gelassen wird und auch keine Fragen offen bleiben. Darunter fällt, dass beispielsweise keine Passivsätze verwendet werden dürfen, da hier der Akteur unklar bleibt. Dazu wird die Liste aus B.2.4.1.1 zur Hand genommen werden. Geht man diese Liste durch, wird überprüft, ob alle Anforderungen eindeutig sind.

Bewertung Bei diesem Kriterium kann eine Berechnung des Erfüllungsgrades E vorgenommen werden. Dabei wird die Anzahl der Einträge der Negativliste N von der Anzahl der Einträge der Liste G aus B.2.4.1.1 abgezogen, durch G dividiert.

$$E = \frac{G - N}{G}$$

In der folgenden Tabelle ist die Zuordnung des Erfüllungsgrades zu den vier Bewertungskategorien niedergeschrieben.

Kategorie	E
<i>voll erfüllt</i>	≥ 0.90
<i>überwiegend erfüllt</i>	< 0.90 und ≥ 0.60
<i>teilweise erfüllt</i>	< 0.60 und ≥ 0.30
<i>nicht erfüllt</i>	< 0.30

B.2.4.1.3 Nachverfolgbarkeit [K-24]

ID des Kriteriums: K-24

Beschreibung Dieses Kriterium bewertet, ob in der Spezifikation zu den Anforderungen eine Quelle angegeben ist. Außerdem muss jede Anforderung eindeutig identifizierbar sein.

Zweck/Motivation In der Spezifikation ist es wichtig, dass bei jeder Anforderung klar ist, woher sie stammt. Dies ist wichtig, damit man bei Fragen weiß, wer der Ansprechpartner ist. Alle Anforderungen müssen auch eindeutig identifizierbar sein, damit man sie in der Spezifikation an anderen Stellen oder auch in anderen Dokumenten eindeutig ansprechbar ist. So kommt es zu keinen Verwechslungen bei den Anforderungen.

Ausführung/Anwendung Die Spezifikation muss nicht ganz gelesen werden. Es reicht aus, die erstellte Liste von B.2.4.1.1 zu benutzen und zu überprüfen, ob alle Anforderungen eindeutig identifizierbar sind. Zusätzlich muss am Anfang des Dokuments überprüft werden, ob auf andere Dokumente oder Tools hingewiesen wird, die die Quelle der Anforderungen darstellen. Es muss überprüft werden, ob alle Anforderungen eindeutig identifizierbar sind. Dies ist möglich durch:

- Gliederungsnummer
- ID
- Name

Bewertung

- *voll erfüllt*, wenn alle Anforderungen eindeutig identifizierbar durch die oben genannten Punkte gekennzeichnet sind.

- *überwiegend erfüllt*, wenn alle Anforderungen eindeutig identifizierbar sind, dies aber nicht durch eine explizite Kennzeichnung erfolgt ist.
- *teilweise erfüllt*, wenn nicht alle Anforderungen eindeutig identifizierbar sind.
- *nicht erfüllt*, wenn viele Anforderungen nicht eindeutig identifizierbar sind.

B.2.4.1.4 Konsistenz (Widerspruchsfreiheit) [K-25]

ID des Kriteriums: K-25

Beschreibung Dieses Kriterium bewertet, ob alle Anforderungen konsistent sind. Dies bedeutet, dass es keine Widersprüche in einer Anforderung oder zwischen verschiedenen Anforderungen gibt.

Zweck/Motivation Wenn eine Spezifikation von mehreren Verfassern geschrieben wird, kann es passieren, dass an verschiedenen Stellen die gleiche Anforderung unterschiedlich spezifiziert wird. Es muss gewährleistet sein, dass die Aussagen eindeutig und konsistent sind. Außerdem ist es möglich, dass in einer Beschreibung einer Anforderung der selbe Vorgang anders beschrieben wird. Es dürfen auch zwischen verschiedenen Anforderungen keine Inkonsistenzen vorkommen.

Ausführung/Anwendung Die Spezifikation muss gelesen werden. Beim Lesen muss darauf geachtet werden, dass es keinerlei Widersprüche in einer Anforderung oder zwischen verschiedenen Anforderungen gibt.

Bewertung

- *voll erfüllt*, die Spezifikation weist keinerlei Widersprüche im oben genannten Sinn auf.
- *überwiegend erfüllt*, die Spezifikation weist nur kleine, unbedeutende Widersprüche im oben genannten Sinn auf.
- *teilweise erfüllt*, die Spezifikation weist einige Widersprüche im oben genannten Sinn auf. Diese sind gravierend im Hinblick auf die weitere Entwicklung.
- *nicht erfüllt*, die Spezifikation weist (sehr) viele Widersprüche im oben genannten Sinn auf. Diese sind so gravierend im Hinblick auf die weitere Entwicklung,

dass der Erfolg der Projekts in Frage gestellt ist.

B.2.4.1.5 Zusammengehörigkeit & Ordnung [K-26]

ID des Kriteriums: K-26

Beschreibung Das Kriterium bewertet, ob zusammengehörige Teile einer Spezifikation auch räumlich gruppiert sind. Wenn beispielsweise eine Anforderung näher erläutert wird, müssen diese Erläuterungen in einem Kapitel stehen und dürfen nicht über das Dokument verteilt sein. Dazu zählen auch eine nachvollziehbare Ordnung im Dokument und eine Hierarchisierung der Anforderungen, wo nötig.

Zweck/Motivation Dem Benutzer muss der Umgang mit der Spezifikation so einfach wie möglich gemacht werden. Dazu gehört, dass der Benutzer alle zusammengehörenden Teile auch an einer Stelle im Dokument findet. So muss er sich nicht die komplette Darstellung im ganzen Dokument zusammensuchen.

Ausführung/Anwendung Alle Punkte, die zu einem Bereich gehören, müssen an einer Stelle zu finden sein. Dazu muss die Spezifikation gelesen und begutachtet werden, ob alle Anforderungen sinnvoll gruppiert sind.

Bewertung

- *voll erfüllt*, wenn in der Spezifikation alle Anforderungen, die zusammengehören, sinnvoll gruppiert sind.
- *überwiegend erfüllt*, wenn in der Spezifikation bis auf wenige Ausnahmen, die nicht keine erhebliche Auswirkung auf das Verständnis haben, alle Anforderungen, die zusammengehören, sinnvoll gruppiert sind.
- *teilweise erfüllt*, wenn in der Spezifikation alle Anforderungen, die zusammengehören, versucht wurden zu gruppieren, dies aber überwiegend nicht gelungen ist.
- *nicht erfüllt*, wenn in der Spezifikation die Anforderungen, die zusammengehören, nicht sinnvoll gruppiert sind.

B.2.4.1.6 Verifizierbarkeit [K-27]

ID des Kriteriums: K-27

Beschreibung Dieses Kriterium bewertet Anforderungen dahingehend, welches Augenmerk auf die Verifizierbarkeit der Anforderungen gegeben wurde. Damit dieses Kriterium vollständig erfüllt ist, muss jede einzelne Anforderung so beschrieben sein, dass die Erfüllung der Anforderung praktisch geprüft werden kann.

Zweck/Motivation Durch dieses Kriterium werden schwammige Formulierungen als schlecht bewertet, da sie nicht verifizierbar beschrieben sind. Solche schwammigen Aussagen führen in vielen Fällen zur Entwicklung von Software, die der Kunde so nicht gefordert hat und sollten deshalb unterlassen werden. Insbesondere sollten nichtfunktionale Anforderungen mit diesem Kriterium untersucht werden.

Ausführung/Anwendung Die erste Anforderung wird herangezogen und daraufhin überprüft, ob die Erfüllung der Anforderung praktisch geprüft werden kann. Ist dies nicht möglich, so ist die Anforderung in eine Negativliste einzutragen. Dieses Vorgehen wird für alle Anforderungen wiederholt.

Bewertung Bei diesem Kriterium kann eine Berechnung des Erfüllungsgrades E vorgenommen werden. Dabei wird die Anzahl der Einträge der Negativliste N von der Anzahl der Einträge der Liste G aus B.2.4.1.1 abgezogen, durch G dividiert.

$$E = \frac{G - N}{G}$$

In der folgenden Tabelle ist die Zuordnung des Erfüllungsgrades zu den vier Bewertungskategorien niedergeschrieben.

Kategorie	E
<i>voll erfüllt</i>	≥ 0.90
<i>überwiegend erfüllt</i>	< 0.90 und ≥ 0.60
<i>teilweise erfüllt</i>	< 0.60 und ≥ 0.30
<i>nicht erfüllt</i>	< 0.30

B.2.4.2 Entwurfsoffenheit [K-28]

ID des Kriteriums: K-28

Beschreibung Dieses Kriterium untersucht, ob die Spezifikation frei von Entwurfseinschränkungen ist, die nicht durch Anforderungen gerechtfertigt sind. So soll klassischen Entwurfsaufgaben nicht vorgegriffen werden und sich die Spezifikation rein auf das „was“ konzentrieren, dem Entwurf das „wie“ überlassen. Dabei ist zu bemerken, dass diese oft zitierte was-wie-Unterteilung nur begrenzt anwendbar ist. Natürlich muss die Spezifikation erläutern, „wie“ der Benutzer die Software starten und benutzt. Was das System dabei jedoch im Hintergrund tut, ist dem Entwurf und der Realisierung überlassen.

Zweck/Motivation Viele Softwareentwickler haben eine größere Vorliebe für das Programmieren als für das Spezifizieren. Dies drückt sich meist dadurch aus, dass in der Spezifikation begonnen wird, diese scheinbar interessanteren Punkte mit einzubeziehen.

Eine Spezifikation einer Software sollte jedoch lediglich die Anforderungen enthalten, welche sich aus den Wünschen des Kunden ergeben. Die durch die Entwickler stattfindende Aufbereitung dieser Anforderungen gehört ebenfalls dazu. Jedoch sollte die Phase der Spezifikation lediglich mit Spezifikationsaktivität verbracht werden und kein paralleler Entwurf im Spezifikationsdokument stattfinden. Somit bleibt die Spezifikation entwurfsneutral und kann später durch verschiedene Programmiersprachen und Entwurfsparadigmen umgesetzt werden.

Ausführung/Anwendung Zur Analyse dieses Kriteriums hat der Gutachter nach Sätzen, Formulierungen und Grafiken zu suchen, die dem Entwurf zugerechnet werden können:

- Gliederung der Software in Komponenten und Zuweisung von Aufgaben an die Komponenten
- Komponentendiagramme, Verteilungsdiagramme, Paketdiagramme, Klassendiagramme
- Nutzung von Entwurfspatterns

- Modellierung der Klassen der fachlichen Domäne
- Entwurfsnahe Beschreibungen (z. B. „Das System kann über die Klasse Controller von den Fremdsystemen X, Y und Z aufgerufen werden.“)
- Realisierungsnahe Beschreibungen (z. B. „Wenn der Akteur auf den Button klickt, wird ein Listener benachrichtigt und das Programm beendet sich.“)
- Festlegung eines Style-Guides für die Quelltexte
- Festlegung der Struktur der Entwicklungs- und Programmverzeichnisse
- Beschreibung des technischen Deployments (Anforderungen an die Installation sind dagegen erwünscht)
- Angabe zu verwendender Technologien (Datenbank, Bibliotheken), insofern dies nicht durch eine Anforderung festgelegt ist.
- Angabe der zu verwendenden Programmiersprache, insofern diese nicht durch eine Anforderung festgelegt ist.

Bewertung Die Bewertung wird auf dem bekannten Schema durchgeführt:

- *voll erfüllt* – Die Spezifikation macht keinerlei Aussagen, welche den späteren Entwurf oder die Implementierung einschränken. Über die technische Umsetzung der Anforderungen wird nichts spezifiziert.
- *überwiegend erfüllt* – Die Spezifikation schränkt an wenigen Stellen Technologien und Entwurfsfreiheit ein. Meist ist dies im Zuge von Beschreibungen geschehen. Es wird jedoch keine Modellierung oder andere klassische Entwurfstätigkeiten durchgeführt.
- *teilweise erfüllt* – Die Spezifikation schränkt den Entwurf an mehreren Stellen stark ein. Darüberhinaus werden Richtlinien und Hinweise an die Implementierung gestellt, welche nicht durch Anforderungen gerechtfertigt sind. Ein Entwurf auf der grünen Wiese ist nur mit Einschränkungen möglich.
- *nicht erfüllt* – Dem Entwurf wird an vielen Stellen vorgegriffen. Klassische Entwurfsaktivitäten (wie z. B. die Modellierung) wurden bereits in der Spezifikation durchgeführt. Auch die Realisierung wird eingeschränkt und Entwurfspattern für die Lösung bestimmter Anforderungen festgelegt.

B.2.4.3 Abstraktionsgrad [K-29]

ID des Kriteriums: K-29

Beschreibung Dieses Kriterium betrachtet den Abstraktionsgrad, der in abgegrenzten Teilen der Spezifikation vorliegt. Der Abstraktionsgrad gibt an, wie abstrakt oder konkret die Darstellung eines Sachverhaltes durchgeführt wurde. Dabei wird insbesondere betrachtet, ob der Abstraktionsgrad innerhalb einer Abbildung, eines Schemas, eines Anwendungsfalls oder eines Textabschnitts gleich ist.

Zum Beispiel in einer Use-Case-Darstellung für den Internetauftritt eines Warenhauses würde der Anwendungsfall „Artikel in Warenkorb legen“ einen geringeren Abstraktionsgrad haben wie die ebenfalls enthaltenen Anwendungsfälle „Shop anzeigen“, „Impressum anzeigen“ und „Startseite anzeigen“, da es sich beim erst genannten Anwendungsfall um eine Funktion des Shops handelt und nicht um eine Hauptfunktion der Internetseite.

Zweck/Motivation Wird dieses Kriterium nicht erfüllt, dann beinhaltet das geprüfte Dokument einen andauernden Wechsel des Abstraktionsgrades innerhalb der oben genannten Teile einer Spezifikation. Solche Abstraktionsgradwechsel können die Verständlichkeit des Dokuments erheblich beeinträchtigen und somit zu unklaren Anforderungen führen.

Ausführung/Anwendung Das gesamte Dokument muss durchgearbeitet werden. Dabei sollte ein einheitlicher Abstraktionsgrad innerhalb folgender Teile vorhanden sein.

- Abbildungen
- Schemata
- Anwendungsfälle
- Textabschnitte

Ist der einheitliche Abstraktionsgrad nicht gegeben, dann wird das Vorkommen auf einer Negativliste dokumentiert.

Bewertung Es wird folgende Bewertung durch den Gutachter durchgeführt:

- *voll erfüllt* – Auf der Negativliste sind sehr wenige oder gar keine Punkte festgehalten. Der Gutachter kann an kaum einer Stelle eine Vermischung verschiedener Abstraktionsgrade feststellen.
- *überwiegend erfüllt* – Auf der Negativliste ist eine kleine Menge an Punkten dokumentiert. Für diese Kategorie kann der Gutachter einige Vorkommen von verschiedenen Abstraktionsgraden vorfinden. Diese sind jedoch in einer Art, dass sie das Verstehen nicht übermäßig beeinflussen.
- *teilweise erfüllt* – Auf der Negativliste ist eine große Menge an Punkten dokumentiert. Der Gutachter findet sehr viele Vorkommen von verschiedenen Abstraktionsgraden. Zusätzlich erschweren diese das Verständnis der Abbildungen, Texte oder Schemata.
- *nicht erfüllt* – Die Negativliste ist sehr lang. Das Aufstellen der Liste der Anforderungen ist kaum möglich. Das Dokument ist auf Grund ständiger Wechsel des Abstraktionsgrades innerhalb der oben genannten Teile der Spezifikation nur mit unnötig großem Aufwand verständlich.

B.2.4.4 Beispiele [K-30]

ID des Kriteriums: K-30

Beschreibung Dieses Kriterium bewertet zum einen, ob die Spezifikation Beispiele sinnvoll einsetzt und diese damit dienlich für den Leser sind. Zum anderen wird durch dieses Kriterium untersucht, ob weitere Anforderungen durch Beispiele illustriert werden sollten.

Zweck/Motivation Beispiele sind gleichzeitig Fluch und Segen für die Spezifikation. Sind sie sinnvoll eingesetzt, helfen sie dem Leser komplexe Zusammenhänge zu verstehen und können sogar für die Erstellung von Testfällen herangezogen werden. Leider werden Beispiele häufig dazu genutzt, abstrakte Anforderungsbeschreibungen zu umgehen und stattdessen konkrete Sachverhalte zu fordern. Besonders der letzte Fall wird durch dieses Kriterium bestraft, da dies dem Prinzip der allgemeingültigen Spezifikation widerspricht.

Ausführung/Anwendung Zur Bewertung dieses Kriteriums untersucht der Gutach-

ter die folgenden Punkte in der Spezifikation:

- Verwendet die Spezifikation zur Illustration von Anforderungen überhaupt Beispiele?
- Werden die Beispiele/Konkretisierungen auch als solche gekennzeichnet (z. B., bspw., „Konkret könnte das heißen, dass ...“)?
- Sind die Beispiele nachvollziehbar und für den Leser verständlich? Dienen sie also überhaupt ihrem Zweck als Beispiele die Anforderungen zu illustrieren?
- Sind die Beispiele überhaupt nötig oder hätte man manche Anforderungen auch ohne Beispiele verstanden?
- Spielen die Beispiele in der Fachdomäne? Hat sich der Autor also eine tatsächliche Anwendung der Anforderung ausgedacht („Der Planer gibt im Feld Modellname ‚S 65 AMG‘ ein und im Feld Anzahl ‚10‘. Danach wird die Gesamtschätzung für die S-Klasse vom System automatisch auf 365 Stück neu berechnet.“) oder redet er von „Äpfeln und Birnen“.
- Sind die Beispiele präzise genug, um für die Erstellung von Testfällen zu dienen?
- Wird deutlich, zu welchen Anforderungen die Beispiele jeweils gehören?
- Werden die Beispiele anstatt der Anforderungen verwendet? Gehen sie also weit über ihren eigentlichen Zweck hinaus und enthalten selbst Ablaufbeschreibungen, Vorbedingungen usw. und enthalten damit selbst Anforderungen, die sonst nirgendwo spezifiziert worden sind?
- Gibt es komplexe Anforderungen, wo Beispiele oder Konkretisierungen beim Verständnis geholfen hätten aber nicht verwendet worden sind?

Bewertung Die Bewertung wird auf dem bekannten Schema durchgeführt:

- *voll erfüllt* – Die Spezifikation verwendet nur für die komplexen Anforderungen Beispiele und damit keine unnötigen. Die Beispiele verdeutlichen dem Leser, wie im späteren Einsatz die Funktion einer Anforderung aussehen wird. Es wird für alle Beispiele deutlich, dass es sich um solche handelt und zu welchen Anforderungen sie gehören.

- *überwiegend erfüllt* – Die Spezifikation verwendet zu wenige oder zu viele Beispiele. Der Gutachter erkennt jedoch, dass die Beispiele gezielt zur Illustration komplexer Anforderungen eingesetzt werden und damit gemäß ihrem Zweck. Es wird für die meisten Beispiele deutlich, dass es sich um solche handelt und zu welchen Anforderungen sie gehören.
- *teilweise erfüllt* – Die Spezifikation enthält sehr viele unnötige Beispiele oder Beispiele fehlen an für viele Anforderungen. Einige Beispiele sind nicht an Anforderungen geknüpft oder enthalten selbst anforderungsrelevante Informationen, die sonst nirgendwo in der Spezifikation abgebildet sind.
- *nicht erfüllt* – Beispiele werden ziellos verwendet und es ist nicht erkennbar, dass nur komplexe Anforderungen illustriert werden sollten. An vielen Stellen sind die Beispiele mit den Anforderungen stark verknüpft und nicht mehr von letzteren zu trennen. Spezifikationen können auch dieser Klasse zugeordnet werden, wenn sie keine Beispiele enthalten, komplexe Anforderungen dies aber dringend erfordert hätten.

B.2.5 Funktionale Anforderungen

B.2.5.1 Methodik [K-31]

ID des Kriteriums: K-31

Beschreibung Dieses Kriterium soll untersuchen, ob die Methodik, mit der die funktionalen Anforderungen spezifiziert worden sind, angemessen sind. Dabei kann die Vielfalt der Darstellungen ebenso bewertet werden, wie die Konsistenz.

Zweck/Motivation Funktionale Anforderungen sind das Herzstück der Spezifikation. Eben deshalb sollte besonderes Augenmerk darauf geworfen werden. Es gibt diverse Methodiken aus Literatur und Praxis, um funktionale Anforderungen zu erfassen. Doch erst wenn diese Methodiken konsistent umgesetzt werden, können die Funktionen später realisiert und getestet werden.

Eine Use-Case-Analyse ist meist ein guter Anfang, um Funktionen zu entdecken. Doch erst die Use-Case-Beschreibungen enthalten alle Kernaussagen zu den Infor-

mationen. Manchmal müssen komplexe Use-Cases in Aktivitätendiagrammen aufgedröselt werden um das fallweise Verhalten des Systems deutlich zu machen. An anderer Stelle bieten sich vielleicht keine Use-Cases an und die Funktionalität muss über eine reine Schnittstellenbeschreibung erfolgen. Da diese vielschichtige Betrachtung das Inhaltskriterium (siehe Kapitel B.2.3) nicht abbilden konnte, ist dieses Kriterium motiviert worden.

Ausführung/Anwendung Dieses Kriterium kann sehr schwierig über eine Checkliste bewertet werden – primär ist die subjektive Meinung des Gutachters gefragt. Um dem Gutachter eine Hilfestellung zu geben, folgt eine Liste mit Punkten, die zur Bewertung dieses Kriteriums untersucht werden sollten:

- Macht die Spezifikation selbst Aussagen über die verwendete Spezifikationsmethodik und nennt Gründe für deren Einsatz?
- Welche verschiedenen Methodiken werden zur Spezifikation der funktionalen Anforderungen verwendet? Wie viele verschiedene sind dies?
- Gibt es überblickshafte Darstellungen des Prozesses der die Funktionen enthält?
 - Ereignisgesteuerte Prozessketten¹
 - Vorgangskettendiagramme²
 - Swimlane-Diagramme
 - unstandardisierte Schemata mit einer Abfolge der Funktionen
 - Prosabeschreibungen
- Gibt es ein übergeordnetes Paradigma, dass die Funktionen einbindet?
 - Zustandsautomaten
 - Petri-Netze
 - Nicht zu verletzende Bedingungen?
 - Metaphern
- Wie werden die eigentlichen funktionalen Anforderungen dargestellt?

¹http://de.wikipedia.org/wiki/Ereignisgesteuerte_Prozesskette

²<http://de.wikipedia.org/wiki/Vorgangskettendiagramm>

- rein formal
- unstrukturierte Prosa-Beschreibungen
- strukturierte Beschreibungen (z. B. Use-Case-Beschreibungen)
- API-Beschreibungen der Schnittstelle
- Gibt es funktionale Anforderungen, die über präzise Darstellungstechniken beschrieben werden?
 - formale Spezifikation mit Z^3
 - prädikatenlogische Bedingungen
 - zu berechnende mathematische Formeln
- Wie werden komplexe funktionale Anforderungen detailliert?
 - durch Zustandsdiagramme
 - durch Sequenzdiagramme mit fachlichen Komponenten anstatt Objekten
 - formale Ergänzungen

Diese Vorbereitungsfragen kann der Gutachter im Vorfeld bearbeiten. Sie dienen zur Beantwortung der folgenden Fragen, welche einen stärkeren Bewertungscharakter haben und die Basis für die Bewertung dieses Kriteriums darstellen.

- Ist ein guter Überblick über die Funktionalität des Systems möglich?
- Sind die Funktionen sinnvoll hierarchisiert oder gruppiert (z. B. nach Akteur)?
- Sind für ähnliche Funktionen gleiche Darstellungstechniken verwendet worden? Ist der Grund dieser Verwendung nachvollziehbar?
- Werden die Darstellungstechniken konsistent verwendet (z. B. die identische Gliederung bei allen Use-Case-Beschreibungen)?
- Wenn der Gutachter die Darstellungstechnik kennt: Wird sie korrekt verwendet?
- Sind die Darstellungstechniken für die funktionalen Anforderungen angemessen?

³<http://de.wikipedia.org/wiki/Z-Notation>

- Können über die Darstellungstechnik alle relevanten Aussagen der funktionalen Anforderungen angegeben werden?
- Wären andere Darstellungstechniken oder Ergänzungen sinnvoll oder sind sogar nötig?
- Werden komplexe funktionale Anforderungen ausreichend detailliert?

Bewertung Wie bereits angedeutet, ist dieses Kriterium nur sehr subjektiv zu untersuchen – ebenso subjektiv ist dann die Bewertung auf den bekannten Bewertungskategorien:

- *voll erfüllt* – Die Spezifikation enthält eine geeignete Darstellung über den Überblick der Funktionen. Es werden wenige Darstellungstechniken für die funktionalen Anforderungen verwendet, welche alle konsistent angewandt und korrekt ausgeführt worden sind. Ergänzungen zur Formalisierung oder Detaillierung einzelner Funktionen wurden an den nötigen Stellen eingesetzt. Die Auswahl der Darstellungstechniken bewertet der Gutachter als durchgängig angemessen.
- *überwiegend erfüllt* – Es gibt Abstriche zur Bewertungskategorie *voll erfüllt*. Die Darstellungstechniken wurden nicht immer konsistent eingesetzt aber korrekt ausgeführt. An einigen funktionalen Anforderungen fehlt eine Detaillierung oder Präzisierung. Der Überblick über die Funktionen ist geeignet und dienlich.
- *teilweise erfüllt* – Der Gutachter bewertet die eingesetzten Darstellungstechniken nicht immer als angemessen. Oftmals werden funktionale Anforderungen in eine Darstellungstechnik gepresst, ohne dass diese nötig oder dienlich ist. Der Überblick über die Funktionen ist nicht förderlich oder nicht vorhanden.
- *nicht erfüllt* – Die Darstellungstechniken werden wild gemischt und deren Einsatz ist dem Gutachter nicht nachvollziehbar. Der Gutachter bewertet die verwendeten Darstellungstechniken als nicht angemessen und nicht dienlich. Bei der Konsistenz und Korrektheit gibt es zahlreiche Ungereimtheiten. Der Überblick über die Funktionen fehlt völlig. Eine Gruppierung der Anforderungen fand nicht statt.

B.2.5.2 Zu treffende Aussagen pro funktionale Anforderung [K-32]

ID des Kriteriums: K-32

Beschreibung Dieses Kriterium bewertet eine Art der Vollständigkeit einer funktionalen Anforderung. Unter vollständig wird hier verstanden, dass zu jeder funktionalen Anforderung Eingaben, Ausgaben, Fehlerfälle, reguläre und alternative Abläufe, Vor- und Nachbedingungen und der Akteur beschrieben worden ist.

Zweck/Motivation Werden die oben beschriebenen Informationen zu einer funktionalen Anforderung nicht gegeben, so kann es während der Umsetzung der Anforderung zu Verzögerungen kommen, da diese Inhalte nachgearbeitet werden müssen. Im schlechtesten Fall entscheidet der Implementierer selbst, wie er es am sinnvollsten findet. Durch ein solches Vorgehen entstehen jedoch fast immer Produkte, die dem Kundenwunsch nicht entsprechen.

Ausführung/Anwendung Jede Anforderung wird daraufhin überprüft, ob die Punkte auf der folgenden Checkliste ausreichend beschrieben sind.

- Eingaben
- Ausgaben
- Fehlerfälle
- regulärer
- alternative Abläufe
- Vorbedingungen
- Nachbedingungen
- Akteur

Abweichungen werden auf einer Negativliste dokumentiert. Eine Abweichung ist hier nicht zu eng zu sehen. Wenn also bei einer funktionalen Anforderung zum Beispiel die Eingaben klar ersichtlich sind, aber nicht explizit festgehalten, dann sollte dies nicht als Abweichung gewertet werden. Ist aber der Gutachter, oder mehrere, der Meinung, dass das Fehlen einer Aussage Raum für verschiedenartige Implementierungen lässt, dann ist die Abweichung auf der Negativliste festzuhalten.

Bewertung Bei diesem Kriterium ist die Berechnung des Erfüllungsgrades E möglich. Dazu wird die Gesamtheit der funktionalen Anforderungen gezählt. Diese Gesamtzahl wird mit acht multipliziert, da je Anforderung acht Punkte erwartet werden. Von der so errechneten Gesamtpunktzahl G wird nun die Anzahl der Punkte auf der Negativliste N abgezogen, durch die Gesamtpunktzahl dividiert.

$$E = \frac{G - N}{G}$$

In der folgenden Tabelle ist die Zuordnung des Erfüllungsgrades zu den vier Bewertungskategorien niedergeschrieben.

Kategorie	E
<i>voll erfüllt</i>	≥ 0.90
<i>überwiegend erfüllt</i>	< 0.90 und ≥ 0.60
<i>teilweise erfüllt</i>	< 0.60 und ≥ 0.30
<i>nicht erfüllt</i>	< 0.30

B.3 Abgrenzung

Folgende Punkte werden durch den vorliegenden Kriterienkatalog nicht angesprochen. Der Grund liegt darin, dass projektfremde Gutachter nicht die nötige Wissensbasis haben, um die Punkte objektiv genug zu bewerten. Zu diesen Punkten gehört:

- Alle Anforderungen des Kunden sollten in der Spezifikation enthalten sein.
- Alle Anforderungen sollten korrekt in der Spezifikation enthalten sein (so wie der Kunde sie sich vorgestellt hat).
- Es sollten lediglich die Anforderungen enthalten sein, welche die spätere Software erfüllen muss. Also keine Anforderungen, welche sich die Autoren ausgedacht haben.
- Die Spezifikation sollte dem Zweck dienen, die Brücke zwischen Kunden- und Entwickler-Seite zu schlagen.
- Eine Spezifikation sollte mit einem Textverarbeitungsprogramm erstellt und durch diverse Tools bei der Anforderungsverwaltung und beim Konfigurationsmanagement unterstützt werden.
- Die Spezifikation sollte die Basis für alle folgenden Dokumente (Entwurf, Test, etc.) sein und darin ausgiebig genutzt werden.
- Der Umfang und Inhalt der Spezifikation sollte zum Software-Entwicklungs-Prozess passen. Eine sehr detaillierte Spezifikation ist beispielsweise bei einer agilen Entwicklung sehr hinderlich (häufige Änderungen der Anforderungen).
- Der „Degree of stability“ meint, dass zu jeder Anforderung angegeben wird, wie stabil diese Anforderung ist und ob weitreichende Änderungen an dieser Anforderung erwartet werden. Da wir eher von stabilen Spezifikationen ausgehen, ist dieser Punkt von uns nicht bewertbar.
- Die Spezifikation sollte firmen- oder kundenspezifische Anforderungen an die Dokumentgestaltung einhalten.

C Kommentierte Spezifikation

C.1 Einleitung

C.1.1 Motivation

Die Studenten der Softwaretechnik der Universität Stuttgart haben in ihrem Diplom-Grundstudium – und bald auch im Bachelor-Fachstudium – ein Software-Praktikum zu absolvieren. Die in der Softwaretechnik-Vorlesung erlangten Kompetenzen hinsichtlich des Software-Entwicklungsprozesses sollen dabei ganz praktisch umgesetzt werden. Die Spezifikationsphase ist dabei meist eine Hürde, da kein Student zuvor ein ähnliches Dokument verfasst hat.

Bisher haben sich die Studenten damit beholfen, Spezifikationen vergangener Jahre anzuschauen und ähnliche Aspekte – oder sogar ganze Inhalte – zu übernehmen. Dieses Vorgehen war für Studenten sehr zielführend, negative Aspekte konnte jedoch beobachtet werden. Da die Spezifikationen der vorherigen Jahrgänge ebenfalls die ersten ihrer Autoren waren, wurden Anfängerfehler begangen. Viele dieser Fehler wurden in den Folgejahren kopiert, da man sich derer nicht bewusst war. Eine geeignetere Spezifikation als Vorlage wäre für diese Studenten sehr empfehlenswert.

Im Rahmen der Fachstudie „Analyse und Kritik von Anforderungsspezifikationen“ wurden verschiedene Kriterien untersucht, mit denen Software-Spezifikationen bewertet werden können. In der eigentlichen Bewertungsphase wurden dann mehrere Spezifikation dem entwickelten Bewertungsschema unterzogen. Basierend auf diesen Bewertungsergebnissen entstand nun die Idee, eine kommentierte Spezifikation zu erarbeiten, welche den Studenten in zukünftigen Software-Praktika helfen kann. Dafür wurde die Spezifikation des Studien-Projektes CodeCover⁴ ausgewählt. Hierbei handelt es sich um die Beschreibung eines Werkzeuges zur Überdeckungsmessung im Rahmen des Glass-Box-Tests.

Für die Studenten bieten sich folgende Vorteile durch die Verwendung dieses Doku-

⁴<http://www.codecover.org>

menten als Ausgangsbasis für ihre Spezifikationsarbeit:

- Es wird eine Spezifikation aus dem Hauptstudium als Basis verwendet.
- Die Erfahrungen der Autoren der ausgewählten Spezifikation sind größer als die der Studenten des Grundstudiums.
- Anfängerfehler sind in dieser Spezifikation weitestgehend ausgebügelt.
- Ein Reviewprozess hat diese Spezifikation bereits abgesegnet.
- Kommentare – motiviert durch den Kriterienkatalog der Fachstudie – machen auf positive und negative Aspekte aufmerksam.
- Zusätzliche Anmerkungen geben allgemeine und praktische Tipps für die Erstellung einer Spezifikation.

Selbstverständlich sind die Resultate der Fachstudie nicht nur für Teilnehmer der Software-Praktika interessant. Über dem akademischen Horizont hinaus kann dieses Dokument sicherlich auch den ein oder anderen Software-Dienstleister interessieren. Vertreter der Industrie erhalten durch diese kommentierte Spezifikation die Möglichkeit, einen Einblick in eine aktuelle studentische Software-Spezifikationen zu erlangen.

Dies ist zum einen für die Einordnung des eigenen Spezifikationsprozesses dienlich. Es können Ideen generiert werden, wie der Spezifikationsprozess verbessert werden kann oder man fühlt sich bestätigt, dass man ebenfalls auf dem aktuellen Stand der Software-Spezifikation ist.

Zum anderen kann diese Arbeit dazu dienen, zukünftige Mitarbeiter einzuschätzen. Die Spezifikation ist im Rahmen eines realen Projektes von Studenten der Softwaretechnik verfasst worden und spiegelt deren Kompetenzen und Erfahrungen hinsichtlich Anforderungsanalyse und -spezifikation wieder.

Sicherlich sind noch eine Menge weiterer Einsatzmöglichkeiten dieser Arbeit denkbar, deren gedankliche Ergänzung dem Leser überlassen sei.

C.1.2 Verwendungshinweise

Die vorliegende Spezifikation ist im Rahmen eines Software-Praktikums entstanden, dass sich zum Ziel gesetzt hatte, in ein Open-Source-Projekt überzugehen. Sämtliche Projektdokumentation ist deshalb auf Englisch verfasst worden. Hierbei sei angemerkt, dass die Autoren keinen herausragenden Grad ihrer Englischkenntnisse erlangt hatten, weshalb das entstandene Dokument für den deutschen Leser einfach zu verstehen sein sollte.

Die Spezifikation ist ab Seite 106 des Dokumentes zu finden und mit Kommentaren versehen. Diese verdeutlichen dem Leser die Anwendung der Bewertungskriterien der Fachstudie und geben ihm darüberhinaus Tipps bei der Erstellung einer eigenen Spezifikation. Die Kommentierungen sind nach drei Farben kategorisiert:

- **Grün**: ein Kriterium befand diesen Punkt der Spezifikation als gut
- **Rot**: ein Kriterium befand diesen Punkt der Spezifikation als schlecht oder der Punkt führte zur Abwertung
- **Gelb**: dieser Kommentar enthält einen allgemeinen Tipp oder einen hilfreichen Hinweis für die Erstellung einer eigenen Spezifikation

Die vorliegende Spezifikation ist mit 115 Seiten vergleichsweise umfangreich. Interessant für das einsteigende Lesen sind sicherlich die folgenden Kapitel:

- Einleitung – Seite 112
- Einführung in die Use-Case-Beschreibung – Seite 117
- Beispiel Use-Case „select instrumentable items“ – Seite 121
- Beschreibung der Batch-Schnittstelle – Seite 145
- Beschreibung einer GUI-Komponente – Seite 185
- Funktionale Anforderung durch mathematische Formeln spezifiziert – Seite 202
- Beschreibung der nichtfunktionalen Anforderungen – Seite 207
- Glossar – Seite 215

Da es sich bei CodeCover um ein Open-Source-Projekt handelt, sind die Quellen der

Spezifikation frei verfügbar. Zu finden sind sie im SourceForge-SVN-Repository⁵.

Die Spezifikation wurde im PDF-Format in dieses Dokument integriert. Dadurch sind leider alle Verlinkungen innerhalb der Spezifikation verloren gegangen. Zur Unterstützung bei der groben Navigation wurden die Hauptkapitel manuell mit in das Inhaltsverzeichnis dieses Dokumentes übernommen.

C.1.3 Anwendung des Kriterienkataloges

Viele Kommentare, mit welchen die Spezifikation ergänzt wurde, sind durch die Bewertungskriterien des Kriterienkataloges motiviert worden. Eine Verdeutlichung der Hintergründe der Kommentare ist durch Sichtung des jeweils angegebenen Kriteriums möglich. Die Beschreibung eines Kriteriums enthält neben den Anweisungen zur Ausführung auch die Bewertungsmodalitäten.

An dieser Stelle sei angemerkt, dass besonders die Kriterien der Sprache schwer zu bewerten waren, da sich die Ausführungsanweisungen sehr auf die deutsche Sprache stützen. Der Umstand, dass es sich um recht einfaches Englisch handelt, führte auch zu keiner Abwertung hinsichtlich Knappheit, was bei deutschen Schachtelsätzen schon ganz anders aussehen würde. Siehe Kriterium *Knappheit* [K-11] auf Seite 61. Das teilweise umständliche oder falsche Englisch wurde bei der Durchführung der Bewertung ignoriert. Grammatische und orthografische Korrektheit sollten in jedem Fall bei der Erstellung einer Spezifikation beachtet werden. Auf eine kleinliche Untersuchung diesbezüglich hat die Bewertung bewusst verzichtet.

C.1.4 Allgemeine Hinweise zur Erstellung einer Spezifikation

Neben den allgemeinen Kommentaren im Dokument sei an dieser Stelle noch auf übergreifende Punkte hingewiesen, die bei der Erstellung einer (studentischen) Spezifikation beachtet werden müssen. Es hat sich gezeigt, dass besonders Spezifikationsanfänger dankbar für diese Tipps sind

- Es sollte unbedingt ein Textsatzprogramm (L^AT_EX, Word, OpenOffice) verwendet werden. Die Erstellung einer Vorlage für alle Dokumente des Projektes hat

⁵<https://codecover.svn.sourceforge.net/svnroot/codecover/trunk/spec>

sich als aufwendig aber lohnenswert herausgestellt. Dadurch wird ein Corporate Design⁶ geprägt, dass besonders auf Kundenseite auf Zustimmung trifft.

- Die Versionshistorie im Dokument sollte nach jeder größeren Änderungen aktualisiert werden. Dabei sollte die Angabe des Autors und der veränderten Kapitel erfolgen. Im Nachhinein ist es dadurch möglich, direkt im Dokument zu erkennen, wer welche Teile der Spezifikation verfasst hat und wann in einem Kapitel zuletzt Änderungen gemacht wurden.
- Für größere Dokumente sollten Verantwortliche für Teile der Spezifikation ernannt werden. So ist es allen Projektteilnehmern (insbesondere dem Kunden) möglich, den jeweiligen Ansprechpartner bei Unklarheiten zu kontaktieren und sich die beschriebenen Anforderungen und deren Hintergründe erläutern zu lassen.
- Wie alle anderen Dokumente auch, sollte die Spezifikation der Versionsverwaltung (z. B. SVN) unterzogen werden. Die Angabe eines Kommentars für eine neue Version/Revision sollte ebenfalls zur Pflicht gemacht werden.
- Neben der Versionsverwaltung ist auch ein fundiertes Konfigurationsmanagement wichtig. Besonders die Versionen für das Review und für die Abgabe sollten besonders markiert werden (z. B. ein Tag im SVN). Sollte \LaTeX als Textsatzprogramm verwendet werden, empfiehlt es sich, auch die kompilierte PDF auf dem Stand der markierten Version hinzuzufügen. Dadurch kann man sich sehr einfach den Stand des Dokumentes anschauen, der an einem bestimmten Tag zum Kunden geschickt wurde.
- Ein organisierter Review-Prozess, sowie ein Freigabe-Prozess sollten vom Qualitätsbeauftragten für die Spezifikation definiert und umgesetzt werden.

⁶http://de.wikipedia.org/wiki/Corporate_Design

Specification

CodeCover

Glass Box Testing Tool

Student Project A “OST-WeST”
University of Stuttgart

Version: 1.1-dev

Last changed on November 7, 2007 (SVN Revision 2387)

Titel, Bezugsrahmen und Version sollten auf dem Deckblatt vorhanden sein. Zusätzlich wären Autoren- und Kundenangabe hilfreich. Siehe Kriterium *Deckblatt [K-02]* auf Seite 51.

Hier fehlt der Status des Dokumentes, z. B. „in Bearbeitung“, „für Review freigegeben“ oder schlicht „freigegeben“.

Version History

Date	Version	Author	Modifications
11.01.2007	0.1	Stefan Franke	- Chapter files and master document file
16.01.2007	0.2	Stefan Franke	- ui: Eclipse Plug-in and images within
17.01.2007	0.3	Michael Starzmann Christoph Müller	- Headwords following the corresponding chapter in the analysis-notes - nr: Keywords taken over by the analysis notes - fr: The foreword for the functional requirements
18.01.2007	0.4	Stefan Franke	- ui: Rewrite source code highlighting - Reorder chapters - Rename some sections - ui: Added source code highlighting for COBOL
19.01.2007	0.5	Christoph Müller	- Document structure changed - fr: Use case pictures imported - fr: Foreword, fr: actors, fr: general arrangements
20.01.2007	0.6	Johannes Langauf Christoph Müller	- nf: Expand some keywords to complete sentences - nf: Find new NFRs - fr: Configuration - fr: Use case description - fr: Language support
21.01.2007	0.7	Christoph Müller Stefan Franke	- fr: Use case description - ui: Session view, Coverage view and Launching
23.01.2007	0.8	Christoph Müller Stefan Franke Michael Starzmann	- Correction after specification meeting - fr: Use case description of measure coverage - fr: General functional requirements - fr: Reports - ui: Configuration dialogs
24.01.2007	0.9	Michael Starzmann Christoph Müller	- in: Introduction - fr: Use case description - fr: Coverage Criteria
25.01.2007	0.10	Michael Starzmann Stefan Franke	- ui: Package and file selection to ... states - fr: Coverage measurement improved - in: Introduction

Das Versionsverzeichnis ist sehr umfangreich. So kann auch im späteren Projektverlauf schnell eingesehen, wer was geschrieben hat. Siehe Kriterium *Versionierung [K-17]* auf Seite 71.

Eine Versionsangabe fehlt dagegen in der Kopf- oder Fußzeile jeder Seite.

Date	Version	Author	Modifications
26.01.2007	0.11	Christoph Müller Stefan Franke	<ul style="list-style-type: none"> - Correction after specification meeting - fr: New use case instrument instrumentable items - ui: Configuration sections - ui: Source code highlighting
27.01.2007	0.12	Christoph Müller Stefan Franke Michael Starzmann	<ul style="list-style-type: none"> - fr: Use case description of administrate sessions - ui: Instrumentation subsection - ui: Solved todos - ui: first draft for the Batch interface - fr: Release, folders, files
28.01.2007	0.13	Christoph Müller Stefan Franke Johannes Langauf	<ul style="list-style-type: none"> - fr: Batch interface - ui,fr: Correction after internal review - nr: improve and fill out most non-functional requirements
29.01.2007	0.14	Christoph Müller	<ul style="list-style-type: none"> - fr: Solve todos, use case diagrams, folder structure - Correction after QA meeting - small spell check
30.01.2007	0.15	Christoph Müller Stefan Franke Johannes Langauf	<ul style="list-style-type: none"> - fr: New use cases: analyse coverage log, export session - ui: New Context menu - ui: Import, Export, Report - ui: Small adaption at figures - nf: Correction after specification review - nf: Extensibility, performance requirements, program examples
31.01.2007	0.16	Christoph Müller	<ul style="list-style-type: none"> - Correction after Igor's big bang
31.01.2007	1.0	Igor Podolskiy	Declaring version 1.0, ready for review
08.02.2007	1.1-dev-1	Stefan Franke Michael Starzmann	<ul style="list-style-type: none"> - Correction after specification review
09.02.2007	1.1-dev-2	Christoph Müller	<ul style="list-style-type: none"> - Correction after specification review
10.02.2007	1.1-dev-3	Christoph Müller	<ul style="list-style-type: none"> - Correction after specification review: Bugs 47, 90, 52, 57, 56, 58, 60, 62, 63, 64, 65, 39, 40, 41, 42, 44, 46, 50, 51, 52, 54, 34
11.02.2007	1.1-dev-4	Johannes Langauf Christoph Müller Stefan Franke	<ul style="list-style-type: none"> - Correction after specification review: Bugs 48, 35, 32, 91, 16
12.02.2007	1.1-dev-5	Johannes Langauf Christoph Müller	<ul style="list-style-type: none"> - Correction after specification review: Bug 22 - new batch commands

Date	Version	Author	Modifications
13.02.2007	1.1-dev-6	Stefan Franke Christoph Müller	- moved the glossary to specification document - added links to glossary entries - Bug 7: Work flow - Bugs 6, 21, 28, 88, 89, 91
14.02.2007	1.1-dev-7	Stefan Franke Christoph Müller Michael Starzmann	- Bug 45
16.02.2007	1.1-dev-8	Stefan Franke Christoph Müller	- Bug 45
11.05.2007	1.1-dev-9	Stefan Franke Christoph Müller	- Bugs 98, 99, 100
15.06.2007	1.1-dev-10	Christoph Müller	- fr: 2.9 JUnit integration
17.06.2007	1.1-dev-11	Stefan Franke	- ui: 3.13 Boolean Analyzer
18.06.2007	1.1-dev-12	Christoph Müller	- fr: 2.7.6 coverage log file name - fr: 2.4.6 instrument supports a charset - fr: 2.4.7 analyze supports a charset - fr: 2.4.5 Instrumenter-info
19.06.2007	1.1-dev-13	Christoph Müller	- fr: 2.4.6 instrument has --copy-uninstrumented
29.06.2007	1.1-dev-14	Christoph Müller	- fr: 2.4.6 instrument has include, exclude
19.09.2007	1.1-dev-15	Johannes Langauf	- ui: 3.14 Hot-Path: make outstanding decisions, update for configureable colors - fr: remove PDF-Report support
31.10.2007	1.1-dev-16	Tilman Scheller	general update of specification

Die letzte Versionsnummer stimmt nicht mit der auf dem Deckblatt überein. Siehe Kriterium *Versionierung [K-17]* auf Seite 71.

Contents

1	Introduction	7
1.1	Project overview	7
1.2	About this document	8
1.3	Addressed audience	8
1.4	Conventions for this document	9
1.5	Authors	9
2	Functional requirements	10
2.1	Test sessions and test cases	10
2.2	Actors	11
2.3	Use case description	12
2.4	Batch interface	40
2.5	Configuration	52
2.6	Report	54
2.7	Instrumentation, types of coverage and measurement	57
2.8	Language support	62
2.9	JUnit integration	62
2.10	ANT integration	64
2.11	Live Test Case Notification	77
3	Graphical User Interface	80
3.1	Package and file states	80
3.2	Instrumentation	81
3.3	Launching	81
3.4	Coverage view	82
3.5	Test sessions view	83
3.6	Import	85
3.7	Export	88
3.8	Source code highlighting	89
3.9	Preferences dialog	94
3.10	Project properties dialog	96
3.11	Correlation Matrix	97
3.12	Live Notification View	99
3.13	Boolean Analyzer	100
3.14	Hot-Path	101
4	Non-functional requirements	102
4.1	Technologies and development environment	102
4.2	Requirements to the working environment	102
4.3	Quantity requirements	103
4.4	Performance requirements	105

Das Inhaltsverzeichnis sollte automatisch generiert werden, damit die Übereinstimmung mit der tatsächlichen Gliederung sichergestellt ist. Außerdem helfen Verlinkungen im Inhaltsverzeichnis beim Navigieren im Dokument.

Das Kapitel „Functional requirements“ ist verhältnismäßig lang. Generell sollte darauf geachtet werden, dass Kapitel und Unterkapitel ausgewogen gegliedert sind. Bei der Verwendung von Spezifikationsvorlagen oder vorgeschriebenen Gliederungen kann man solche Unausgewogenheiten entschuldigen. Siehe Kriterium *Ausgewogenheit [K-04]* auf Seite 53.

4.5	Availability	106
4.6	Security	106
4.7	Robustness and failure behavior	106
4.8	Usability	106
4.9	Portability	107
4.10	Maintainability	107
4.11	Extensibility	107
List of Figures		109
Glossary		110

1 Introduction

1.1 Project overview

CodeCover stands for **code coverage** tool. It measures the code coverage[✓] of a running program and will be as independent as possible of the programming language of the covered program.

Characteristics of *CodeCover*:

- *CodeCover* runs at least on Linux and Windows,
- *CodeCover* can measure code coverage for programs written in Java and COBOL.
- *CodeCover* is extensible to measure code coverage for further programming languages as well.
- *CodeCover* measures multiple code coverage criteria and is extensible to further ones.
- *CodeCover* provides functionality to create reports of the measured code coverage in HTML[✓]-files.
- *CodeCover* is an Eclipse¹ plug-in with a graphical user interface, but also provides a command line interface for use without Eclipse.

To understand the functional requirements specified in this document, a visual overview of the work flow is shown in figure 1.1.

Several steps and intermediate results exist for the whole of the coverage measurement process. The ellipses stand for processing, the rectangles stand for intermediate results or final results.

The process starts with the instrumentation[✓] of code files[✓]. A MAST[✓] is produced in addition to the instrumented code files. The MAST[✓] is stored with the code files[✓] in a session container[✓]. After the compilation and execution of all the instrumented code files of the SUT[✓], a coverage log[✓] with the raw coverage results is produced.

During the analysis phase, the coverage log is processed to obtain a test session[✓] with test cases[✓]. They contain all the processed coverage results. These information are

In der Kopfzeile sind Seitenzahl, Projekt- und Dokumenttitel ersichtlich. Dadurch ist das Dokument durch jede Seite identifizierbar.

Überschriften werden serifenlos gesetzt, der Text dagegen mit Serifen im Blocksatz. Dies erhöht die Lesbarkeit. Siehe Kriterium *Textsatz [K-07]* auf Seite 57.

Die Beschreibung der Hauptfunktionen ist gut gelungen. Es wird jedoch nicht auf die Ziele des Kunden mit der Software und den Ist-Zustand eingegangen. Eine eventuelle Abgrenzung, was die Software nicht leisten soll, wäre ebenfalls empfehlenswert. Siehe Kriterium *Methodik [K-31]* auf Seite 93.

¹<http://www.eclipse.org/>

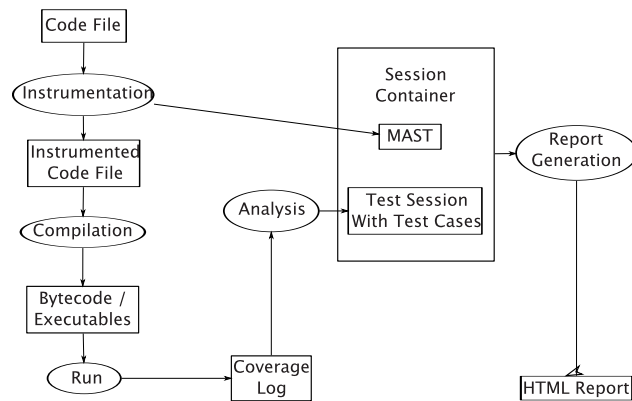


Figure 1.1: Work flow of the software

added to the session container[✓].

Using the information of the MAST[✓] and the test sessions[✓], *CodeCover* can generate a HTML[✓] report.

1.2 About this document

This document specifies all requirements the software has to fulfill and all interfaces to users or other programs. The design of the software will be written based upon this document. This document is the common ground between the customer and the developers[✓]. Therefore, it's important that both, customers and developers, pay attention to the quality of this document and keep it current.

1.3 Addressed audience

This document is addressed to

- the customer who ordered the software
- the project manager controlling the work
- the designers writing the software design

Für den Projektüberblick bietet es sich hier an, eine Darstellung des Prozesses zu wählen, welche den späteren Einsatz der Software illustriert. So wird dem Leser schnell ersichtlich, welche Aktivitäten (Funktionen) die Software leisten können muss und welche Teilprodukte beim Einsatz der Software entstehen.

Die verwendete Notation sollte allgemein bekannt sein, oder – wie in dieser Spezifikation – erklärt werden. So könnte beispielsweise eine Ereignisgesteuerte Prozesskette^a eingesetzt werden. Siehe Kriterium *Methodik [K-31]* auf Seite 93.

^ahttp://de.wikipedia.org/wiki/Ereignisgesteuerte_Prozesskette

Neben dem Zweck des Dokumentes ist auch ein Überblick über den Inhalt angebracht. Der Leser weiß dadurch wie das Dokument strukturiert ist, ob es einen Anhang gibt und ob das Dokument einer Standardgliederung folgt. Siehe Kriterium *Vollständigkeit der inhaltlichen Aspekte [K-21]* auf Seite 75.

- the quality assurance division creating test cases[✓] for the software
- the developers implementing the design
- future developers maintaining and extending the software
- interested users of the software
- students of upcoming student projects

1.4 Conventions for this document

A glossary is shipped together with this specification[✓]. It contains basic definitions and allows clear statements in this document because it prevents ambiguity. Therefore words mentioned in the glossary are used often and are not explicitly defined in this specification but in the glossary.

The term “software” is used for *CodeCover*. Code examples and file names are written in the `typewriter style`. Labels and names of graphical user interface components are written in SMALL CAPS. If necessary, examples are used and placeholders are enclosed by percentage signs: %placeholder%. Furthermore, glossary entries are marked with the symbol ↗, but only at the first occurrence in a section.

1.5 Authors

In the following table the contact persons per section are named.

Section	Author	E-mail
Introduction	Michael Starzmann	*****@studi.informatik.uni-stuttgart.de
Functional requirements (2.1 – 2.5)	Christoph Müller	*****@studi.informatik.uni-stuttgart.de
Functional requirements (2.6 – 2.8)	Michael Starzmann	*****@studi.informatik.uni-stuttgart.de
Graphical user interface	Stefan Franke	*****@studi.informatik.uni-stuttgart.de
Non-functional requirements	Johannes Langauf	*****@studi.informatik.uni-stuttgart.de

Die Einführung und Verwendung von Konventionen für das Dokument ist beispielhaft.

Der Leser wird auf das Begriffslexikon hingewiesen, welches als Anhang enthalten ist. Verlinkte Verweise erlauben das Navigieren zum entsprechenden Begriff. Siehe Kriterium *Existenz [K-14]* auf Seite 67.

Die Autoren sollten auf den ersten Seiten des Dokumentes angegeben werden, z. B. direkt auf dem Deckblatt. Eine Nennung der Verantwortlichkeiten für Teile des Dokumentes ist vorteilhaft, falls Fragen zu bestimmten Anforderungen auftreten. Siehe Kriterium *Deckblatt [K-02]* auf Seite 51.

Es fehlen Verweise auf Dokumente im Projektrahmen, die mit dieser Spezifikation mitgelten oder auf die verwiesen in der Spezifikation wird. Beispiele wären Analysedokumente, der Projektplan sowie die verwendeten Standards. Wichtig ist auch wo diese Dokumente zu finden sind. Siehe Kriterium *Vollständigkeit der inhaltlichen Aspekte [K-21]* auf Seite 75.

2 Functional requirements

2.1 Test sessions and test cases

The software produces test sessions[✓] which contain test cases[✓]. A session container[✓] stores a number of test sessions which each refer to a code base[✓]. Each test session has to have a unique name within a session container. It is not specified how the session containers are stored (XML, internal database, ...) but it should be decided in the software design phase.

Test cases are used to subdivide a test session. Test cases contain the results of the coverage measurement over a period of time during the execution of the SUT[✓]. This can either be the whole of the SUT run, in this case the test session contains only a single test case, or a smaller period of time.

The end of a test case should be explicitly declared, so that it is clear where a test case begins and ends.

The test cases do not overlap. In consequence, the start of a new test case enforces the end of the previous test case. A test case in a test session is uniquely defined by its (test case's) name. If a test case with the same name is started several times, all respective results of the coverage measurement are associated with the same test case.

Test cases will be defined by JUnit or by the user during the SUT execution using a dialog box.

In addition to that, the software provides an integrated test case notification mechanism.

To use the test case notification mechanism in Java, a small JAR file containing a `Protocol` class can be added to the SUT's class path. This class has the following methods the user can call anywhere in the code of the SUT to create test cases:

```
//defining the start of a named and described test case:
public static void startTestCase(String name, String comment)
//Alternatively defining the start of named test case:
public static void startTestCase(String name)
//Defining the end of the last test case:
public static void endTestCase(String name)
```

Die Spezifikation nimmt dem Entwurf die technische Realisierung nicht vorweg, sondern bleibt diesbezüglich neutral. Siehe Kriterium *Entwurfsoffenheit* [K-28] auf Seite 88.

Bevor die Use-Cases beschrieben werden, können allgemeine funktionale Anforderungen spezifiziert werden. Diese gelten beispielsweise über alle funktionalen Anforderungen hinweg und erleichtern deren Verständnis. Außerdem bleiben so die Use-Case-Beschreibungen kompakter, da allgemeine Anforderungen zentral beschrieben werden.

```
//Alternatively defining the end of the last without using the name:  
public static void endTestCase()
```

The name of the `Protocol` class and the names of the methods are not normative. They are examples used to describe the mechanism and can be adapted by the software design at will.

If there is not a valid `Protocol` call in the code files of the SUT, the software will create one anonymous test case with the name `unnamed test case` for the full test session results. But if there are defined test cases, only the coverage occurring during test cases is measured.

Test sessions and test cases are related to a specific version of the code files, the code base. The software can only highlight the results of a coverage run of a code file (see section 3.8) if the code file is equal to the code file used for the instrumentation.

Test sessions depending on the same code base can be merged. This means that all test cases contained in two or more different test sessions are copied into a new test session. If test cases have the same name, they are renamed to `test case name (session name 1)`, `test case name (session name 2)`. Also, two or more test cases of one test session can be merged to a new test case.

2.2 Actors

To describe the use cases in the following section 2.3, actors must be defined. These actors are the users of the software. It is assumed that these users are software developers⁷ or testers and thus have experience with software tools. The actor model is shown in figure 2.1.

As the software partly integrates with the Eclipse IDE, one type of actor is the *Eclipse user*. He has experience in using Eclipse and has worked with plug-ins before. He wants the plug-in interface to be intuitive and expects a similar behavior he is used to from other plug-ins. He is accustomed to control every software feature out of Eclipse and does not want to have to open external programs.

The *shell user* is the user who controls the software using the system shell. He has used Windows or Linux shells before as well as other command line programs and is used to their respective characteristics. He wants a well-written reference manual and on line

Die Akteure sollten zentral beschrieben und danach nur noch verwiesen verwiesen werden. Es sollte deutlich werden, welcher Akteur welche Benutzerschnittstelle der Software nutzt. Zusätzlich sollten die Fähigkeiten und Erfahrungen der Akteure beschrieben werden. Erwartungen an die Software können zusätzlich ergänzt werden. Ein interessanter Ansatz dafür sind sogenannte Personas^a

^a<http://de.wikipedia.org/wiki/Personas>

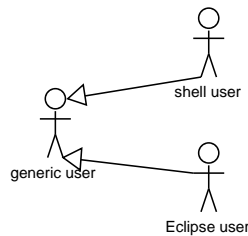


Figure 2.1: Actors

help with subcommand and option listing.

In the following use case models often a *generic user* is used, because Eclipse and shell users participate in the same use cases. In these cases, both are generalized to the *generic user*.

2.3 Use case description

2.3.1 Preface

In the rest of this section the functional specification[✓] is described. To clarify the functional requirements, a use case analysis is used. There is no common understanding of the purpose of use cases. On the one hand, they are used to show the key functions of the specified software – the key functions a customer wants to be implemented. On the other hand, they are used to describe the sequence of actions clearly. For the following use case description both aspects are needed.

For the description of the key functions, key use cases are applied. They summarize smaller use cases and allow an overview. These use cases are defined in section 2.3.3. Besides the key use cases, standard use cases are employed to describe the functions in detail.

Die Use-Case-Analyse wird auf zwei Abstraktionsgraden durchgeführt. Die „Key-Use-Cases“ spiegeln die Geschäftsziele wieder, die der Kunde mit der Software verfolgt. Die „Standard-Use-Cases“ spezifizieren die funktionalen Anforderungen sehr detailliert (Vor- und Nachbedingungen, alternative Abläufe, usw.). Siehe Kriterium *Methodik [K-31]* auf Seite 93.

2.3.2 Predefinitions

2.3.2.1 Use case descriptions

Each standard use case comes with a use case description. The use case description consists of the following items:

- Actor
- Preconditions
- Regular sequence
- Other sequences
- Postconditions
- Possible exceptions

The *actor* is the person performing the use case. The actors are described in section 2.2.

The *preconditions* describe the circumstances needed to start the use case. This can be a state of the software, an open dialog box or the successful termination of another use case.

The *regular sequence* is the description of the normal steps of the use case. It states how the actor interacts with the software, which input is made and which feedback is returned by the software. It is assumed that the sequence is successfully finished without errors.

If there are short cuts or small modifications possible for the regular sequence, they are described in *other sequences* too. Also the cancellation of a use case belongs to this section.

The *postconditions* describe the state of the software and – if affected – data after the regular sequence is successfully finished. If there are other possible sequences, the *postconditions* describe the state of the software after each of these sequences.

The *possible exceptions* are used to specify sequences where errors occur. They can be caused by mistakes of the actor, file system errors or other states which cause the sequences to be aborted.

Beginning with the section 2.3.2.2 general assumptions are described that hold true for every use case. An explicit statement[✓] in the use case description can override these

Für diese Art der funktionalen Anforderungen (Benutzungsabläufe) ist die Use-Case-Analyse gut geeignet. Dieses Verfahren ist standardisiert und enthält alle wichtigen Beschreibungselemente für funktionale Anforderungen.

Sind durch den Kunden keine Vorgaben bezüglich der Spezifikationsmethodik getroffen worden, ist die Use-Case-Analyse meist eine gute Wahl. Siehe Kriterium *Methodik [K-31]* auf Seite 93.

assumptions for a particular use case.

2.3.2.2 General preconditions

Considering the Eclipse user, Eclipse must be started. The plug-in must be installed correctly and must not be disabled. A Eclipse project[✓] must be opened.

2.3.2.3 General other sequences

For the use cases of the Eclipse user it is assumed, that every use case which includes interaction with a dialog, can be stopped immediately by clicking the Cancel button in the dialog.

There are often several ways to open a dialog or to execute a command in Eclipse. In most cases only one way is described for simplicity. Some are listed here, because they are explicitly used:

- the Eclipse dialog PROPERTIES can be opened using the context menu of the dialog or the menu PROJECT
- the context menu of a code file[✓] can be opened in the PACKAGE EXPLORER and in the NAVIGATOR
- the context menu can be opened using a right click or the Context Menu Key on the keyboard

2.3.2.4 General postconditions

If it is stated that something – e.g. a test session[✓] – is *saved* the changes are stored persistently in the related session container[✓].

2.3.2.5 General possible exceptions

The general behavior of the software in abnormal situations is described in the section 4.7.

If a session container could not be updated or created – e.g. due to lack of access permissions or low disk space – an error message is shown to inform the actor.

Abläufe und Bedingungen, die allgemein gültig sind, werden hier vorangestellt und nicht in jedem Use-Case wiederholt.

2.3.3 Key use cases

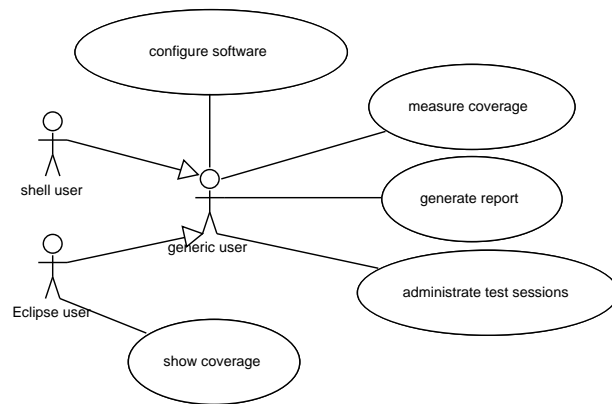


Figure 2.2: Key use cases

This diagram shows the key use cases of the software. As introduced in section 2.2, there is an Eclipse user and a shell user. Both are specialized from the generic user.

The key use cases summarize a block of functionality and can be subdivided into smaller standard use cases. These are:

- Measure coverage (see section 2.3.4)
- Show coverage (see section 2.3.5)
- Administrate test sessions (see section 2.3.6)
- Generate report (see section 2.3.7)
- Configure software (see section 2.5)

2.3.4 Measure coverage

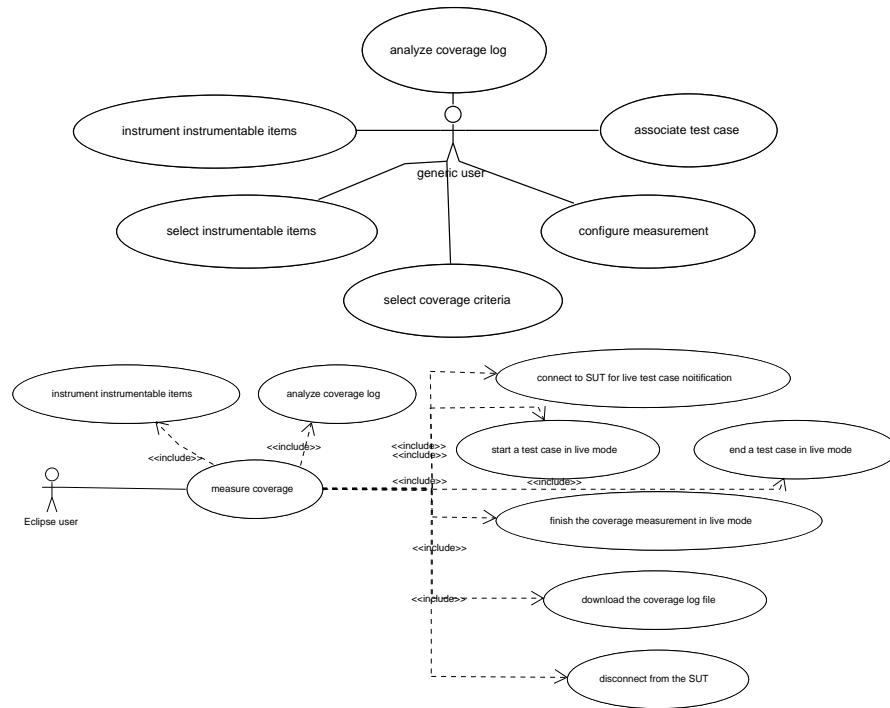


Figure 2.3: Use cases related to measuring coverage

2.3.4.1 Use case: select instrumentable items

2.3.4.1.1 Actor

The actor of this use case is the Eclipse user (see section 2.2).

2.3.4.1.2 Preconditions

The actor has opened an Eclipse project containing at least one instrumentable item[✓].

2.3.4.1.3 Regular sequence

The actor selects one or more instrumentable items – e.g. in the PACKAGE EXPLORER – and clicks on the check box menu item USE FOR COVERAGE MEASUREMENT in the

Use-Case-Diagramme sind gut für den überblickshaften Einstieg in die funktionalen Anforderungen geeignet. Man sieht die Funktionen, deren Abhängigkeiten und beteiligten Akteure. Siehe Kriterium *Methodik [K-31]* auf Seite 93.

Abbildungen sollten, wenn verfügbar, als Vektorgrafiken eingebunden werden. Trotz deren Skalierbarkeit sollte die Abbildung in der Standardgröße lesbar sein. Überlappungen, wie bei dem <<include>> sollten vermieden werden.

Der Leser sollte die Abbildung verstehen können. Eine Erklärung der Notation kann dabei helfen. Sollte es sich um eine gängige Notation oder Modellierungstechnik handeln (wie beispielsweise Use-Cases), ist eine Erklärung nicht erforderlich. Siehe Kriterium *Abbildungen [K-20]* auf Seite 74.

context menu.

To deselect instrumentable items, the actor repeats the described procedure and clicks on the context menu item `USE FOR COVERAGE MEASUREMENT` again.

2.3.4.1.4 Other sequences

There are no other sequences possible for this use case.

2.3.4.1.5 Postconditions

Selecting or deselecting an instrumentable item has an recursive effect on all its sub items: for example, selecting a package causes all its sub packages and types to be selected, too. The same applies for the deselecting. If an instrumentable item had been selected before and a parent item is selected later, the originally selected item remains selected.

In the `PACKAGE EXPLORER`, the icons of the selected instrumentable items change to `USED FOR COVERAGE MEASUREMENT` state.

If the actor has deselected instrumentable items, the icon changes to the normal state. (see section 3.1)

2.3.4.1.6 Possible exceptions

There are no special possible exceptions to be considered for this use case.

2.3.4.2 Use case: select coverage criteria

2.3.4.2.1 Actor

The actor of this use case is the Eclipse user (see section 2.2).

2.3.4.2.2 Preconditions

If the actor has not changed the coverage criteria of a project, all coverage criteria are selected.

2.3.4.2.3 Regular sequence

The actor opens the `PROJECT PROPERTIES` dialog for the particular project. Then he clicks on the item *CodeCover* . Here the actor can select which coverage criteria he wants to measure. To add a criterion for measurement, he activates the related check box. Deactivating a check box means that the corresponding coverage criterion will not be measured. It is not possible to deselect all check boxes and apply the changes.

Durch die feine Gliederung erhält jeder Use-Case eine eindeutige Kapitelnummer. Die dadurch beschriebene funktionale Anforderung ist also eindeutig identifizierbar. Zudem ist der Name jedes Use-Cases eindeutig. Siehe Kriterium *Identifizierbarkeit [K-22]* auf Seite 81.

Die Nummerierung der Aspekte innerhalb eines Use-Cases ist dagegen unnötig oder sogar störend.

After the actor has made his choice, he clicks on the button OK. The dialog PROPERTIES closes.

2.3.4.2.4 Other sequences

There are no other sequences possible for this use case.

2.3.4.2.5 Postconditions

At least one coverage criterion is selected for the edited Eclipse project. The software saves this selection. In addition to that, the software checks whether the already instrumented instrumentable items[✓] must be reinstrumented for the new selection of coverage criteria.

2.3.4.2.6 Possible exceptions

If the actor has deselected all check boxes the dialog prohibits the click on OK.

2.3.4.3 Use case: instrument instrumentable items

2.3.4.3.1 Actor

The actor of this use case is the Eclipse user (see section 2.2).

2.3.4.3.2 Preconditions

There are no special preconditions needed for this use case.

2.3.4.3.3 Regular sequence

The actor uses the menu PROJECT and the menu item INSTRUMENT PROJECT... to explicitly instrument the selected instrumentable items. A dialog opens and asks the actor to enter the target path for instrumented code files[✓]. The actor puts in a valid path he has write access to. With a click on the button INSTRUMENT he starts the instrumentation[✓] process.

While this process is running, a progress bar appears to inform the actor about the progress of the instrumentation process. The Eclipse integrated progress bar is used for this purpose.

2.3.4.3.4 Other sequences

This use case is implicitly started by the use case *measure coverage* (see section 2.3.4.4). In this case, the default target folder of the project for instrumented code files is used

and the dialog is not displayed (see section 2.7.1).

If there are no instrumentable items selected for coverage measurement, the software opens a dialog box to ask the actor, whether he wants to instrument every instrumentable item or wants to cancel.

2.3.4.3.5 Postconditions

If the use case is explicitly started by the user, a new code base[✓] is created having the date and time of the end of the instrumentation process. A MAST[✓] is created of the source files. A new session container[✓] is created, containing the code base and the MAST. The session container is stored. In the TEST SESSIONS view the new code base is selected. It has got no test session. All code files which are USED FOR COVERAGE MEASUREMENT are instrumented and stored at the given target path. All other source files are just copied.

The same procedure is used, if this use case is implicitly started by the use case *measure coverage*. The only exception is made, if no changes were made at the source files of the project since the last start of *measure coverage* for this project and the selection of the code files USED FOR COVERAGE MEASUREMENT has not changed. In this case, the last selected code base can be used again.

2.3.4.3.6 Possible exceptions

If the instrumented code files could not be written – e.g. due to lack of access permissions or low disk space – an error message is shown to inform the actor.

2.3.4.4 Use case: measure coverage

2.3.4.4.1 Actor

The actor of this use case is the Eclipse user (see section 2.2).

2.3.4.4.2 Preconditions

At least one coverage criterion is activated for measurement. There is an entry point[✓] in the current project.

2.3.4.4.3 Regular sequence

The actor navigates to the entry point for which he wants to start the coverage measurement. He clicks on the COVERAGE BUTTON (see figure 3.3) and at the appearing menu on the button COVERAGE AS... , JAVA-APPLICATION.

Da die Benutzerschnittstelle in einem eigenen Kapitel beschrieben wird, sollte an dieser Stelle darauf verwiesen werden (im Originaldokument hilft ein Link bei der Navigation).

If no code file of this project has been changed since the last start of this use case for the same project and the selection of the code files USED FOR COVERAGE MEASUREMENT has not changed, no instrumentation is needed. Otherwise, the software will implicitly start the use case *instrument instrumentable items* (see section 2.3.4.3) and a new code base will be created.

After having instrumented all the required files, the software rebuilds the instrumented code files⁷ and starts the SUT⁷ using the selected entry point.

After the instrumented project has terminated, the software proceeds with the measurement calculation of the covered elements.

2.3.4.4.4 Other sequences

If the actor has not selected any instrumentable item for coverage measurement, the software opens a dialog box to ask the actor, whether he wants to instrument every instrumentable item or wants to cancel.

If the entry point has been used for coverage measurement before, the list of the COVERAGE BUTTON contains this entry so that the actor can use this entry directly instead of using the buttons COVERAGE AS. . . , JAVA-APPLICATION again.

Additionally, the COVERAGE dialog (see section 3.3) contains entries for coverage measurements used in past. The actor can use this dialog to start a coverage measurement too. He selects the entry in the entry list on the left and clicks on the button COVERAGE.

2.3.4.4.5 Postconditions

The result of the coverage measurement run is a coverage log⁷. The use case *analyze coverage log* (see section 2.3.4.6) is implicitly started for this coverage log. This use case produces a test session for the measurement.

The software has saved the results of the coverage measurement in a new test session. This test session has the name *New test session* and a number as suffix if needed for uniqueness. The TEST SESSIONS view (see figure 3.5) contains the new test session which is automatically selected. All test cases of the new test session are shown in the list of test cases. They are all automatically activated.

If there had not been at least one instrumentable item selected for coverage measurement and the software had selected all after request, then the state of them changes to USED FOR COVERAGE MEASUREMENT and the PACKAGE EXPLORER updates their icons.

2.3.4.4.6 Possible exceptions

If there are errors in the process, the process will be canceled and an error message will be shown. Possible errors might be:

- I/O errors while instrumenting
- compile errors
- errors starting the entry point
- access permissions or low disk space when writing the coverage log

2.3.4.5 Use case: associate test case

2.3.4.5.1 Purpose

The actor wants the software to start a named test case, when the control flow passes a specific line in a code file, e.g. the actor has written a test script which calls several methods of several test classes. Test cases should be defined for each of these method calls.

2.3.4.5.2 Actor

The actor of this use case is the Eclipse user (see section 2.2).

2.3.4.5.3 Preconditions

There is a code file in an open Eclipse project which contains the code the actor wants to add test case notifications to.

2.3.4.5.4 Regular sequence

The actor navigates to the code file and positions the cursor before the line of the code file where the test case should start. Then he uses the menu items SOURCE, *CodeCover* TEST CASE NOTIFICATION, START TEST CASE WITH NAME AND COMMENT (see section ??). The software adds an import declaration in the file and adds a new code line at the position of the cursor:

```
Protocol.startTestCase("%NAME%", "%COMMENT%");
```

The actor changes "%NAME%" and "%COMMENT%" to the name and the comment of the test case. The software is ordered to start a new test case when this method is called in the coverage measurement.

To define the end of a test case, the actor uses the menu items SOURCE, *CodeCover* TEST CASE NOTIFICATION, END TEST CASE WITH NAME (see section ??). The software adds a new code line:

```
Protocol.endTestCase("%NAME%");
```

The actor changes the "%NAME%" to the name of the test case started before and saves the file.

2.3.4.5.5 Other sequences

If the actor wants multiple test cases, he uses this procedure at different lines of the code file. He can also associate test cases in other code files.

If the actor only wants to have one test case for the whole test script, he must not add a special statement anywhere. The software then treats the whole measurement as one test case. (see section 2.1)

There are also other test case notification forms possible that have the same effect. These are described in section 2.1. The start of a test case implies the end of the prior test case.

If the actor is more advanced, he can write the statement into the source code on his own. In this case he must add the JAR containing the `Protocol` class to the class path of the related Eclipse project too.

2.3.4.5.6 Post conditions

The software or the actor has added the JAR containing the `Protocol` class to the class path of the Eclipse project. The code file is prepared for measurement with test case association.

2.3.4.5.7 Possible exceptions

There are no special possible exceptions to be considered for this use case.

2.3.4.6 Use case: analyze coverage log

2.3.4.6.1 Actor

The actor of this use case is the Eclipse user (see section 2.2).

Die Verlinkung (??) ist fehlerhaft. Dies sollte vor der Freigabe unbedingt korrigiert werden.

2.3.4.6.2 Preconditions

The actor has used the use case *instrument instrumentable items* and has run the compiled SUT on his own and without Eclipse support. In the consequence there is a coverage log related to an Eclipse project.

If the actor has instrumented the code files out of Eclipse, the actor has to import the session container[✓] with the corresponding code base[✓] first (see section 2.3.6.2).

Anyway, there is a code base loaded in Eclipse and the source files of this code base were compiled and executed. A coverage log[✓] file was created.

2.3.4.6.3 Regular sequence

The actor activates the TEST SESSIONS view (see figure 3.5). In the view, he clicks on the button IMPORT COVERAGE LOG. The import dialog for session containers opens.

In the dialog the actor specifies the coverage log. Moreover he must state a name and can type a comment for the test session that will be created. After that, he clicks on FINISH. The dialog closes. (see figure ??)

2.3.4.6.4 Other sequences

The dialog for importing a coverage log[✓] file can also be opened by using the default Eclipse import dialog. For example this can be opened using the menu FILE and the item IMPORT. . . . There the actor selects the item *CodeCover* COVERAGE LOG in the group OTHER and clicks on NEXT.

2.3.4.6.5 Post conditions

The software processes the given coverage log and creates a new test session with the given name and comment. The test session is assigned to the corresponding code base. The test session is saved in the session container[✓].

In the new test session is selected in the TEST SESSIONS view (see figure 3.5). All its test cases are shown in the list of the test cases and are selected.

2.3.4.6.6 Possible exceptions

If there is no code base[✓] in Eclipse, the new coverage log belongs to, an error message is shown.

If the test session is not related to the Eclipse project of the code base, the code highlighting won't be possible (see section 2.3.5.3) but the coverage results can be examined

(see section 2.3.5.2).

If there are errors processing the coverage log, the process is interrupted and an error message is shown.

2.3.4.7 Use case: configure measurement

The following use cases contains the configuration of the measurement behavior. It is described in section 2.5.

2.3.4.8 Use case: connect to an SUT for live test case notification

2.3.4.8.1 Actor

The actor of this use case is the Eclipse user (see section 2.2).

2.3.4.8.2 Preconditions

The instrumented SUT has been configured to support the live test case notification and is running. The TEST CASE NOTIFICATION view is open.

2.3.4.8.3 Regular sequence

The actor enters the host name and port to connect to and clicks the CONNECT button.

2.3.4.8.4 Other sequences

There are no other sequences possible for this use case.

2.3.4.8.5 Postconditions

The view is connected to the running SUT.

2.3.4.8.6 Possible exceptions

If there was a network error, an error message is shown.

If the CodeCover MBean is not available on the server, the client waits until a CodeCover MBean is registered; all operations except disconnecting will stay disable until a CodeCover MBean is registered on the Server.

2.3.4.9 Use case: start a new test case in live mode

2.3.4.9.1 Actor

The actor of this use case is the Eclipse user (see section 2.2).

2.3.4.9.2 Preconditions

A connection to a running SUT is established (see use case 2.3.4.8). The TEST CASE NOTIFICATION view is open. The coverage measurement in the current SUT run is not finished.

2.3.4.9.3 Regular sequence

The actor enters a test case name in the text field of the view and clicks the START button.

2.3.4.9.4 Other sequences

There are no other sequences possible for this use case.

2.3.4.9.5 Postconditions

The test case has been started.

2.3.4.9.6 Possible exceptions

If there was a network error, an error message is shown.

2.3.4.10 Use case: end the current test case in live mode

2.3.4.10.1 Actor

The actor of this use case is the Eclipse user (see section 2.2).

2.3.4.10.2 Preconditions

A connection to a running SUT is established (see use case 2.3.4.8). The TEST CASE NOTIFICATION view is open. A test case is started.

2.3.4.10.3 Regular sequence

The actor clicks the END button in the TEST CASE NOTIFICATION view.

2.3.4.10.4 Other sequences

There are no other sequences possible for this use case.

2.3.4.10.5 Postconditions

The test case has been ended.

2.3.4.10.6 Possible exceptions

If there was a network error, an error message is shown.

2.3.4.11 Use case: finish coverage measurement in live mode

2.3.4.11.1 Actor

The actor of this use case is the Eclipse user (see section 2.2).

2.3.4.11.2 Preconditions

A connection to a running SUT is established (see use case 2.3.4.8). The TEST CASE NOTIFICATION view is open.

2.3.4.11.3 Regular sequence

The actor clicks the FINISHED button in the TEST CASE NOTIFICATION view.

2.3.4.11.4 Other sequences

There are no other sequences possible for this use case.

2.3.4.11.5 Postconditions

The measurement has been finished.

2.3.4.11.6 Possible exceptions

If there was a network error, an error message is shown.

2.3.4.12 Use case: download the coverage log file

2.3.4.12.1 Actor

The actor of this use case is the Eclipse user (see section 2.2).

2.3.4.12.2 Preconditions

A connection to a running SUT is established (see use case 2.3.4.8). The TEST CASE NOTIFICATION view is open. The coverage measurement has been finished.

2.3.4.12.3 Regular sequence

The actor clicks the DOWNLOAD button in the TEST CASE NOTIFICATION view.

2.3.4.12.4 Other sequences

There are no other sequences possible for this use case.

2.3.4.12.5 Postconditions

The coverage log file is downloaded and has the same base name as the file written on the system running the SUT and the same location as if the execution was local and triggered by *CodeCover* Eclipse plug-in.

2.3.4.12.6 Possible exceptions

If there was a network or file access error, an error message is shown.

2.3.4.13 Use case: disconnect from the SUT in live mode

2.3.4.13.1 Actor

The actor of this use case is the Eclipse user (see section 2.2).

2.3.4.13.2 Preconditions

A connection to a running SUT is established (see use case 2.3.4.8). The TEST CASE NOTIFICATION view is open.

2.3.4.13.3 Regular sequence

The actor clicks the DISCONNECT button in the TEST CASE NOTIFICATION view.

2.3.4.13.4 Other sequences

There are no other sequences possible for this use case.

2.3.4.13.5 Postconditions

The connection to the SUT is closed.

2.3.4.13.6 Possible exceptions

If there was a network error, a warning is shown.

2.3.5 Show coverage

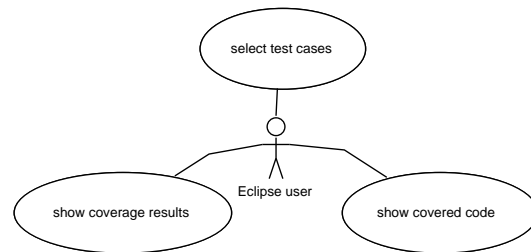


Figure 2.4: Use cases related to showing coverage

Die Use-Cases sind durch die Key-Use-Cases bereits gruppiert. Die Gliederung übernimmt diese Gruppierung, wodurch die Use-Cases fachlich strukturiert sind.

2.3.5.1 Use case: select test cases

2.3.5.1.1 Actor

The actor of this use case is the Eclipse user (see section 2.2).

2.3.5.1.2 Preconditions

There is at least one test session in Eclipse which has at least one test case.

2.3.5.1.3 Regular sequence

The actor opens the TEST SESSIONS view (see figure 3.5). Then he selects the related CODE BASE.

Now the actor can activate all test cases he wants to view the coverage results of, using the ACTIVATED check boxes. The rest of the test cases' check boxes have to be deactivated.

2.3.5.1.4 Other sequences

The actor can select test cases of different test sessions. To select all test cases of a test session, the actor uses the ACTIVATED check box of the specific test session.

There are some other sequences possible to activate test cases – e.g. using the context menu in the TEST SESSIONS view (see section 3.5).

2.3.5.1.5 Postconditions

The source code highlighting and the coverage view are refreshed if needed, based on the results of the selected test cases.

2.3.5.1.6 Possible exceptions

There are no special possible exceptions to be considered for this use case.

2.3.5.2 Use case: show coverage measurement

2.3.5.2.1 Actor

The actor of this use case is the Eclipse user (see section 2.2).

2.3.5.2.2 Preconditions

The actor has selected test cases of test sessions belonging to the same code base (see use case *select test cases*, section 2.3.5.1).

2.3.5.2.3 Regular sequence

The actor activates the COVERAGE view of the plug-in (see figure 3.4). At this view he has an overview of all instrumented instrumentable items in a hierarchical order. He can expand an item of the hierarchy to examine the coverage results of its sub items.

The result columns show the measured results of the coverage by criterion. Only the criteria that are measured are shown in this view.

To order the lines of the tree table ascending or descending, the actor clicks respectively clicks twice on the specific column header. The lines of items are then sorted within their parent item in the tree table.

2.3.5.2.4 Other sequences

There are no other sequences possible for this use case.

2.3.5.2.5 Postconditions

There are no possible post conditions of this use case.

2.3.5.2.6 Possible exceptions

There are no special possible exceptions to be considered for this use case.

Wenn ein Aspekt innerhalb eines Use-Cases keinen Inhalt hat, wird er dennoch mit einem entsprechenden Vermerk aufgenommen. Dies zeigt, dass er nicht vergessen wurde.

2.3.5.3 Use case: show covered code

2.3.5.3.1 Actor

The actor of this use case is the Eclipse user (see section 2.2).

2.3.5.3.2 Preconditions

The actor has selected test cases of test sessions belonging to the same code base (see use case *select test cases*, section 2.3.5.1). The code base of the test sessions selected is still the current one, which means, no code file has been changed since the instrumentation of the code files.

2.3.5.3.3 Regular sequence

The actor navigates to the code file in which the coverage results are to be displayed by source code highlighting. Then he opens the code file in an editor.

2.3.5.3.4 Other sequences

There are no other sequences possible for this use case.

2.3.5.3.5 Postconditions

The software highlights the elements of the code according to results of the measurement of the selected coverage criteria. The highlighting rules are specified in section 3.8 in detail.

2.3.5.3.6 Possible exceptions

If a code file has changed since the coverage run of the test session, the highlighting can not be shown. Therefore the software has to check, if the code file has the same content as the code file used for the related coverage run.

If the actor changes a code file, the highlighting is not possible anymore. If he revokes his changes, the software shows the highlighting again.

2.3.6 Administrate test sessions

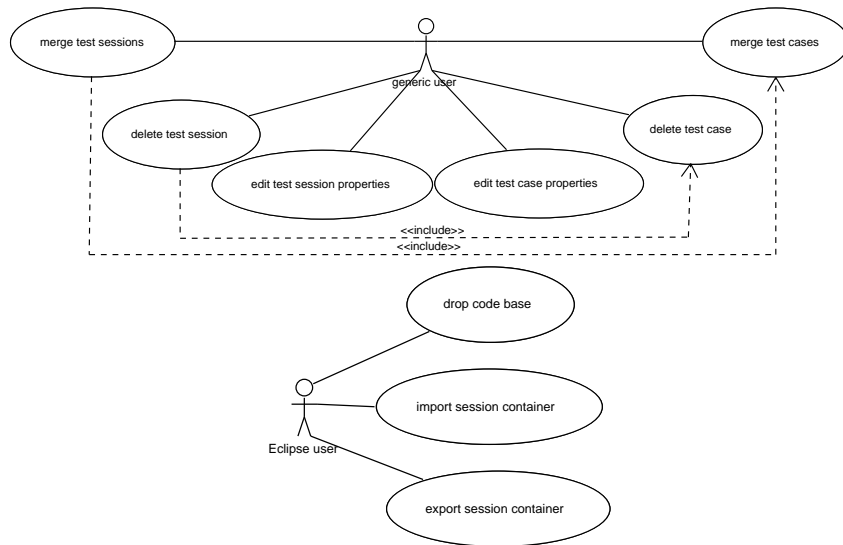


Figure 2.5: Use cases related to administrating test sessions

2.3.6.1 Preface

These use cases are based on other use cases described before. By measuring the coverage, the software creates a test session containing associated test cases. They are the basis of analysis and can be edited in several ways. The test sessions and test cases can be merged and their properties can be altered. Test cases can be deleted from a test session and whole test sessions with all included test cases can be deleted.

To use the Eclipse plug-in and the batch interface side by side, import and export functionality is supported by the Eclipse plug-in. General definitions regarding test sessions and test cases are made in the section 2.1.

2.3.6.2 Use case: import session container

2.3.6.2.1 Actor

The actor of this use case is the Eclipse user (see section 2.2).

2.3.6.2.2 Preconditions

There must be at least one session container[✓] in the file system. This can be either created by a coverage measurement in Eclipse or using the batch mode (see section 2.4).

2.3.6.2.3 Regular sequence

The actor activates the TEST SESSIONS view (see figure 3.5). In the view, he clicks on the button IMPORT TEST SESSION. The import dialog for session containers opens (see figure ??). In this dialog, the actor specifies the path to the session container and selects the related Eclipse project. Finally the actor clicks on the button FINISH. The dialog closes.

2.3.6.2.4 Other sequences

The dialog for importing a session container can also be opened by using the default Eclipse import dialog. For example this can be opened using the menu FILE and the item IMPORT. . . . There the actor selects the item *CodeCover* SESSION CONTAINER in the group OTHER and clicks on NEXT.

If the code base of the session container is not related to an Eclipse project, the actor needn't select a project.

2.3.6.2.5 Postconditions

The code base of the session container is imported into Eclipse.

If the session container has got test session(s), they are imported too. In this case, one of the imported test sessions is selected in the TEST SESSIONS view (see figure 3.5). All its test cases are shown in the list of the test cases and are activated.

All information needed to use the session container in Eclipse for future are saved.

2.3.6.2.6 Possible exceptions

If the code base[✓] of the session container is not related to the specified project, the code highlighting won't be possible (see section 2.3.5.3) but the coverage results can be examined (see section 2.3.5.2).

If there are errors loading the session container, the process is interrupted and an error message is shown.

2.3.6.3 Use case: export session container

2.3.6.3.1 Actor

The actor of this use case is the Eclipse user (see section 2.2).

2.3.6.3.2 Preconditions

There must be at least one test session in Eclipse.

2.3.6.3.3 Regular sequence

The actor selects the code base and its test sessions he wants to export in the TEST SESSIONS view (see figure 3.5) and clicks on the button EXPORT in the tool bar.

The export dialog opens (see figure 3.9). The selected code base and the selected test sessions are preselected in the dialog, but the actor can also select more test sessions. He changes the TYPE to *CodeCover* SESSION CONTAINER, chooses a destination and clicks on FINISH. The dialog closes.

2.3.6.3.4 Other sequences

The dialog for exporting a test session can also be opened by using the default Eclipse export dialog. This dialog can for example be opened using the menu FILE and the item EXPORT. . . . In the selection dialog the actor selects the item *CodeCover* COVERAGE RESULT EXPORT in the group OTHER and clicks on NEXT.

2.3.6.3.5 Postconditions

A session container is created at the specified destination. It contains the code base and all selected test session.

2.3.6.3.6 Possible exceptions

There are no special possible exceptions to be considered for this use case.

2.3.6.4 Use case: drop code base

2.3.6.4.1 Actor

The actor of this use case is the Eclipse user (see section 2.2).

2.3.6.4.2 Preconditions

There must be at least one code base in Eclipse.

2.3.6.4.3 Regular sequence

The actor opens the TEST SESSIONS view (see figure 3.5), selects the code base[^] and clicks on the button DROP CODE BASE. A dialog opens, requesting the actor if he wants to drop the selected code base out of Eclipse or if he wants to delete the related session container[^] too. The actor clicks on the button DROP. The dialog closes.

2.3.6.4.4 Other sequences

If the actor wants to drop the code base out of Eclipse and delete the related code base too, he clicks on DELETE.

2.3.6.4.5 Postconditions

The selected code base and all depending test sessions and test cases are removed from the TEST SESSIONS view.

If the actor has chosen DELETE, the session container of the code base is deleted in the file system too.

2.3.6.4.6 Possible exceptions

There are no special possible exceptions to be considered for this use case.

2.3.6.5 Use case: merge test sessions

2.3.6.5.1 Actor

The actor of this use case is the Eclipse user (see section 2.2).

2.3.6.5.2 Preconditions

There must be at least two test sessions in Eclipse.

2.3.6.5.3 Regular sequence

The actor activates the TEST SESSIONS view (see figure 3.5). In this view, he selects the test session, he want to merge and clicks on the button MERGE. The dialog TEST SESSION PROPERTIES opens (similar to figure 3.6).

The actor puts in the name and the comment of the merged test session and clicks on the button OK. The dialog closes.

2.3.6.5.4 Other sequences

There are no other sequences possible for this use case.

2.3.6.5.5 Postconditions

A new test session with the specified name is created. All test case information from the test sessions selected for merge are copied into the new test session. The new test session is saved.

If some test cases have the same name, they are renamed to `test case name (session name 1)`, `test case name (session name 2)`.

The new test session appears in the list of the TEST SESSIONS view. The new session and all its test cases are activated.

2.3.6.5.6 Possible exceptions

If the actor has selected less than two test sessions, the dialog prohibits the click on the button MERGE.

2.3.6.6 Use case: edit test session properties

2.3.6.6.1 Actor

The actor of this use case is the Eclipse user (see section 2.2).

2.3.6.6.2 Preconditions

There must be at least one test session in Eclipse.

2.3.6.6.3 Regular sequence

The actor activates the TEST SESSIONS view (see figure 3.5), selects a test session and clicks on the button PROPERTIES. The dialog TEST SESSION PROPERTIES opens (similar to figure 3.6).

The actor changes the name and/or the comment of the selected test session. After he has finished editing the properties, he clicks on the button OK. The dialog closes.

2.3.6.6.4 Other sequences

There are no other sequences possible for this use case.

2.3.6.6.5 Postconditions

The new name and the new comment of the test session are saved. The name of the test session changes in the list.

2.3.6.6.6 Possible exceptions

The new name of the test session can not equal to a name of another test session in the session container. The dialog does not allow to save a duplicate name.

2.3.6.7 Use case: delete test session

2.3.6.7.1 Actor

The actor of this use case is the Eclipse user (see section 2.2).

2.3.6.7.2 Preconditions

There must be at least one test session in Eclipse.

2.3.6.7.3 Regular sequence

The actor opens the TEST SESSIONS view (see figure 3.5), selects the related code base and the test session. Then he clicks on the button DELETE.

2.3.6.7.4 Other sequences

If the actor wants to drop more than one test session at once, he selects these test cases and clicks on the button DELETE. A dialog opens, requesting the actor if he really wants to delete the selected test sessions. The actor clicks on the button DELETE. The dialog closes.

2.3.6.7.5 Postconditions

The selected test sessions are removed from the session container and from the TEST SESSIONS view.

2.3.6.7.6 Possible exceptions

There are no special possible exceptions to be considered for this use case.

2.3.6.8 Use case: merge test cases

2.3.6.8.1 Actor

The actor of this use case is the Eclipse user (see section 2.2).

2.3.6.8.2 Preconditions

There must be a test session in Eclipse that contains at least two test cases that fit the merging criteria stated in section 2.1.

2.3.6.8.3 Regular sequence

The actor opens the TEST SESSIONS view (see figure 3.5), selects the related code base[✓], the test session and the test cases. Then he clicks on the button MERGE. The dialog TEST CASE PROPERTIES opens (see figure 3.6).

The actor puts in the name and the comment of the merged test case and clicks on the button OK. The dialog closes.

2.3.6.8.4 Other sequences

The actor can also use the item MERGE in the context menu of the test cases.

2.3.6.8.5 Postconditions

A new test case with the specified name is created. All information from the selected test cases are copied into the new test case. The test case is saved.

The new test case appears in list of the TEST SESSIONS view.

2.3.6.8.6 Possible exceptions

If the actor has selected less than two test cases, the button MERGE TEST CASES is deactivated.

2.3.6.9 Use case: edit test case properties

2.3.6.9.1 Actor

The actor of this use case is the Eclipse user (see section 2.2).

2.3.6.9.2 Preconditions

There must be at least one test case in a test session in Eclipse.

2.3.6.9.3 Regular sequence

The actor opens the TEST SESSIONS view (see figure 3.5), selects the related code base[✓], the test session and the test case. Then he clicks on the button PROPERTIES. The dialog TEST CASE PROPERTIES opens (see figure 3.6).

The actor changes the name and/or the comment of the selected test case. After he has finished editing the properties, he clicks on the button OK. The dialog closes.

2.3.6.9.4 Other sequences

The actor can also use the menu item `PROPERTIES` in the test case's context menu in the `TEST SESSIONS` view.

2.3.6.9.5 Postconditions

The new name and the new comment of the test case are saved. The name of the test case is updated in the list.

2.3.6.9.6 Possible exceptions

If the actor has not selected exactly one test case, the button `TEST CASE PROPERTIES` is deactivated.

The new name of the test case can not equal to a name of another test case in the test session. The dialog does not allow to save a duplicate name.

2.3.6.10 Use case: delete test case

2.3.6.10.1 Actor

The actor of this use case is the Eclipse user (see section 2.2).

2.3.6.10.2 Preconditions

There must be at least one test case in a test session in Eclipse.

2.3.6.10.3 Regular sequence

The actor opens the `TEST SESSIONS` view (see figure 3.5), selects the related code base[✓], the test session and the test case. Then he clicks on the button `DELETE`. A dialog opens, requesting the actor if he really wants to delete the selected test cases. The actor clicks on the button `DELETE`. The dialog closes.

2.3.6.10.4 Other sequences

The actor can also use the menu item `DELETE` in the test case's context menu in the `TEST SESSIONS` view.

2.3.6.10.5 Postconditions

The selected test cases are removed from the list in the `TEST SESSIONS` view. The test cases are removed from the test session in the session container.

2.3.6.10.6 Possible exceptions

There are no special possible exceptions to be considered for this use case.

2.3.7 Use case: generate report

2.3.7.1 Actor

The actor of this use case is the Eclipse user (see section 2.2).

2.3.7.2 Preconditions

There must be at least one test session in Eclipse.

2.3.7.3 Regular sequence

The actor selects a code base[✓] and a set of test sessions in TEST SESSIONS view (see figure 3.5). Then he clicks on the button EXPORT. The EXPORT dialog opens (see figure 3.9).

The current code base and the selected test sessions are preselected in the dialog, but the actor can also select another code base or other test sessions. He changes the TYPE to REPORT, chooses a destination and clicks on NEXT. The report dialog opens (see figure 3.10).

The actor clicks on the button FINISH. The dialog closes.

2.3.7.4 Other sequences

The dialog for generating a report can also be opened by using the default Eclipse export dialog. For example, this can be done using the menu FILE and the item EXPORT. . . . There the actor selects the item *CodeCover* COVERAGE RESULT EXPORT in the group OTHER and clicks on NEXT.

2.3.7.5 Post conditions

A report is generated and stored in the chosen directory. The contents and appearance of the generated report is defined in section 2.6.

2.3.7.6 Possible exceptions

If the report files could not be written – e.g. due to lack of access permissions or low disk space – an error message is shown to inform the actor.

2.4 Batch interface

2.4.1 Preface

To describe the use cases for the actor *shell user*, the shell commands are described. It is recommended that the executable `codecover` is contained in the `PATH` variable of the operating system.

The software can be run in the shell by calling either `codecover %command% [%options%]` or `codecover %option%`. All available commands are described in the rest of this section. Section 2.4.3 contains an overview about all supported commands.

If the actor wants to use `spaces` in arguments – e.g. file names – he must take care that the shell he is using parses the file name as one argument. For example he might enclose the file name using quotation marks: `"my file name.sql"`.

If the actor has called the software with an unsupported command, the software stops immediately and prints an error message:

`Command not supported. Use "codecover --help" for a command overview.`

If the actor has called the software with a supported command but a wrong option or syntactically wrong parameters, the software stops immediately and prints an error message like this:

`Wrong argument usage. Use "codecover help %command%" for options description.`

2.4.2 General options

Using the software without a command some general options are supported. These options can be used by: `codecover %option%`.

Für die Beschreibung der Batch-Schnittstelle eignet sich die Use-Case-Beschreibung weniger, da die Spezifikation der Kommandos und Parameter in Prosa umständlich ist und zu erhöhter Redundanz führt. Außerdem lassen sich die optionalen Parameter nur eingeschränkt in den alternativen Abläufen darstellen.

Zielführender ist daher der Einsatz einer Art Befehlsreferenz in tabellarischer Form. Dieses Vorgehen hat den Vorteil, dass die Übersicht erhöht wird und die Spezifizierung der Funktionen präziser durchgeführt werden kann. Siehe Kriterium *Methodik [K-31]* auf Seite 93.

Option	Explanation
--help -h	A help page containing this command overview. Has the same effect like <code>codecover help</code> .
--version -V	Prints out the version of the software.

Table 2.1: General batch options

2.4.3 Command overview

For almost every command, either a long or a short version can be used. These commands can be used by `codecover %command% [%options%]`.

Command	Description
instrumenter-info ii	Information of all available instrumenters
instrument in	Instrumentation [↗] of code files [↗]
analyze an	Analysis of a coverage run to create a test session
report re	Generating a report from a test session
info	Showing the information of a session container [↗] and contained test sessions and test cases
merge-sessions ms	Merging two or more test sessions
alter-session as	Altering test session information
copy-sessions cs	Copy test sessions from one session container to another
remove-sessions rs	Removing test sessions from a session container
merge-test-cases mc	Merging two or more test cases
alter-test-case at	Editing test case information
remove-test-cases rt	Removing test cases from a session container
help h	A help page containing this command overview or an option and parameter overview a given command.

Table 2.2: Batch command overview

2.4.4 Global command options

For every command a set of options is supported. In addition to specific command options, all commands support some global parameterless options. They are not required but change the behavior of the software when used. These commands can be used by `codecover %command% [%options%]`.

Option	Explanation
--verbose -v	Orders the software to print more information as usual. For example this can be a description of the actions being done. This option is the opposite of --quiet.
--quiet -q	Orders the software not to print information to the shell. This option is the opposite to --verbose.
--pretend -p	Orders the software not to perform any actions affecting the data persistently but to print information about what the software would do instead. Using --pretend the actor can make sure that his command has the correct syntax and would be successfully executed.
--help -h	Prints an option and parameter overview of the given command. Has the same effect like <code>codecover help %command%</code> .

Table 2.3: General batch options

2.4.5 Instrumenter-info

This command allows the actor to get information of the instrumenters, that are available by the software. Using this command, the actor can get to know, which instrumenter fits to his programming language and which additional options are supported.

This command is run by:

```
codecover (instrumenter-info|ii) [options]
```

Option		Parameter and description	Default if omitted
Short	Long		
-l	--language	the name of the programming language	all

Option		Parameter and description	Default if omitted
Short	Long		

Table 2.4: Options for command `instrumenter-info`

2.4.6 Instrument

This command instruments a set of code files[✓]. To select the code files, which should be instrumented, a `root-directory` must be specified. All code files must be located under this directory. For this reason a `default package` would be the best choice.

For a more detailed selection, include patterns can be used. These patterns allow wild-cards and are adopted from the apache ant project² (see the pattern description³). The actor can specify more than one include pattern, to select the source files for instrumentation. The same way exclude patterns can be specified.

A file will be instrumented, if its relative path matches at least one include pattern, if it has the correct extension for the stated programming language and its relative path matches no exclude pattern.

This command is run by:

```
codecover (instrument|in) [options]
```

Option		Parameter and description	Required / Default
Short	Long		
-r	--root-directory	the root directory of the source files	•
-l	--language	the name of the programming language	•
-d	--destination	the destination directory for the instrumented files	•
-c	--container	the new session container	•
-I	--instrumenter	the unique key of the instrumenter to use	
-a	--charset	the character encoding of the source files	system default
-i	--include	a relative include pattern; this argument can occur more than one time	all files

²<http://ant.apache.org/>

³<http://ant.apache.org/manual/dirtasks.html>

Die einzelnen Kommandos werden nach einem einheitlichen Schema beschrieben:

- Kommandobeschreibung
- Syntax
- Optionen und Parameter
- Beispiele
- Besonderheiten

Ein Batch-Kommando, welches eine funktionale Anforderung darstellt, wird damit vollständig spezifiziert.

Option Short	Long	Parameter and description	Required / Default
-f	--includes-file	a file containing a list of relative include patterns separated by new line	
-e	--exclude	a relative exclude pattern; this argument can occur more than one time	
-x	--excludes-file	a file containing a list of relative exclude patterns separated by new line	
-o	--criterion	one of (all , st , br , co , lo); this argument can occur more than one time – once for every criterion	all
-u	--copy-uninstrumented	advices the software to copy all files of the root-directory, that were not instrumented, to the destination	disabled
-D	--directive	arguments of the style key=value to enable special features of the instrumenter; the <code>instrumenter-info</code> command should print out a list of directives, an instrumenter supports	

Table 2.5: Options for command `instrument`

The arguments of option `criteria` stand for:

Criteria abbreviation	Explanation
all	all criteria
st	statement coverage↗
br	branch coverage↗
co	condition coverage↗
lo	loop coverage↗

Table 2.6: Explanation of the criteria abbreviations

An example call of the command `instrument` is:

```
codecover instrument --root-directory "C:\my files\project 1\" --include
```

Bei komplizierten Sachverhalten helfen Beispiele beim Nachvollziehen. Siehe Kriterium *Beispiele [K-30]* auf Seite 91.

```
"de\foo\pak1\*" --include "de\foo\pak2\dot*.java" --exclude "**\*Test.java"
-d "C:\my files\instrumented\" --language java --criterion st --criterion br
--copy-uninstrumented --container session-container-file.xml
```

The option `--copy-uninstrumented` can be used to get a replica of a source directory including resource files like images or configuration files.

If there is more than one instrumenter available for the specified programming language, the software aborts this instrumentation attempt. An error message is printed out like

```
There is more than one instrumenter available for %programming language%.
Please use the command codecover instrumenter-info to get to know the
unique key of the instrumenter you prefer. Than use the option instrumenter
for this command to exactly specify the instrumenter by its unique key.
```

Along with the instrumented code files, the instrumentation process produces a session container[✓] containing the code base[✓] and the MAST[✓]. The new code base[✓] is has the date and time of the end of the instrumentation process. The code base is stored.

2.4.7 Analyze

This command is run by:

```
codecover (analyze|an) [options]
```

This command needs the session container produced by the instrumentation process and the coverage log[✓] produced by the executed instrumented and compiled program.

Option		Parameter and description	Required / Default
Short	Long		
-c	--container	the session container the coverage data should be added to	•
-g	--coverage-log	the coverage log produced by the executed program	•
-n	--name	the name of the new test session containing the coverage results	•
-m	--comment	a comment describing the test session	empty

Option		Parameter and description	Required / Default
Short	Long		
-a	--charset	the character encoding of the coverage log file	system default

Table 2.7: Options for command analyze

If the target session container does not exist, an empty session container is created at the specified target.

2.4.8 Report

This command is run by:

```
codecover (report|re) [options]
```

This command requires a test session, produced by the command **analyze** and a template file for the report generation.

Option		Parameter and description	Required
Short	Long		
-c	--container	the session container to use	•
-s	--session	the name of the test session for the report	•
-p	--template	the template file containing transformation descriptions	•
-d	--destination	the destination for the report	•

Table 2.8: Options for command report

The generated report file will be a HTML[✓] file. The HTML file and a subdirectory containing other sources will be created.

2.4.9 Info

This command is run by:

```
codecover info [options]
```

This command shows information about a session container.

Option		Parameter and description	Required
Short	Long		
-c	--container	the session container	•
-s	--session	the name of a test session	
-T	--test-cases	showing test case information	

Table 2.9: Options for command info

If no options are used, the program puts out a list of all sessions ordered by code base. The output can look like this:

```
user@rechner ~ >codecover info --container main.xml
codecover session container: "main.xml"
```

```
code bases and test sessions:
code base ID | session      | date       | time
-----
12           |              | 21.10.2006 | 17:23:00
              | GUI          | 22.10.2006 | 20:14:03
              | Performance  | 22.10.2006 | 20:14:50
-----
14           |              | 22.10.2006 | 21:12:00
              | Model I      | 22.10.2006 | 22:16:41
-----
15           |              | 23.10.2006 | 08:11:00
              | Model II     | 24.10.2006 | 06:43:00
-----
```

If the argument `--test-cases` is set, additionally to every session all test cases are put out.

If the test session name is set, the output is reduced just for the indicated test session. The output can look like this:


```

user@rechner ~ >codecover info --container main.xml --session "GUI test"
--test-cases
codecover session container: "main.xml"
session name:      GUI test
session comment: some clicks in the menu
session date:      22.10.2006
session time:      20:12:01

test cases:
name          | date          | time
-----
menu file     | 22.10.2006   | 20:14:03
menu edit     | 22.10.2006   | 20:14:50
menu options  | 22.10.2006   | 20:16:41
menu view     | 22.10.2006   | 20:17:13
menu help     | 22.10.2006   | 20:19:37

```

2.4.10 Merge-sessions

This command is run by:

```
codecover (merge-sessions|ms) [options]
```

With this command the actor can merge two or more test sessions in a session container⁴ into a new test session (see section 2.1).

Option		Parameter and description	Required / Default
Short	Long		
-c	--container	the session container to use	•
-s	--session	a name of a test session participating at the merging; this argument can occur more than one time – once for every participant	•
-R	--remove-old-test-sessions	indicates, whether or not the test sessions, that were merged, are removed after merging	

Option		Parameter and description	Required / Default
Short	Long		
-n	--name	the name of the merged test session	•
-m	--comment	a comment describing the merged test session	empty

Table 2.10: Options for command merge-sessions

2.4.11 Alter-session

This command is run by:

```
codecover (alter-session|as) [options]
```

With this command the actor can change the information of a test session.

Option		Parameter and description	Required / Default
Short	Long		
-c	--container	the session container to use	•
-s	--session	the old name of the test session	•
-n	--name	a new name of the test session	name not altered
-m	--comment	a new comment describing the test session	comment not altered

Table 2.11: Options for command alter-session

2.4.12 Copy-sessions

This command is run by:

```
codecover (copy-sessions|cs) [options]
```

With this command the actor can copy one or more test sessions from a session container to another.

Option		Parameter and description	Required
Short	Long		
-c	--container	the source session container	•
-s	--session	a name of a test session participating at the copy; this argument can occur more than one time – once for every participant	•
-d	--destination	the destination session container	•

Table 2.12: Options for command copy-sessions

If the destination session container does not exist, a copy of the source session container containing only the defined sessions is created at the specified destination.

2.4.13 Remove-session

This command is run by:

```
codecover (remove-sessions|rs) [options]
```

With this command the actor can remove one or more test sessions and their test cases from a session container.

Option		Parameter and description	Required
Short	Long		
-c	--container	the session container to remove from	•
-s	--session	the name of the test session to be removed; this argument can occur more than one time – once for every test session	•

Table 2.13: Options for command remove-sessions

2.4.14 Merge-test-cases

```
codecover (merge-test-cases|mt) [options]
```

With this command the actor can merge two or more test cases into one test case. These test cases must be in one test session and must fit the merging criteria stated in section 2.1.

Option		Parameter and description	Required / Default
Short	Long		
-c	--container	the session container to use	•
-s	--session	the name of the test session	•
-t	--test-case	a name of a test case participating at the merging; this argument can occur more than one time – once for every participant	•
-R	--remove-old-test-cases	indicates, whether or not the test cases, that were merged, are removed after merging	
-n	--name	the name of the merged test case	•
-m	--comment	a comment describing the merged test case	empty

Table 2.14: Options for command merge-test-cases

2.4.15 Alter-test-case

```
codecover (alter-test-case|at) [options]
```

With this command the actor can change a test case (see section 2.1).

Option		Parameter and description	Required / Default
Short	Long		
-c	--container	the session container to use	•
-s	--session	the name of the test session	•
-t	--test-case	the old name of the test case	•
-n	--name	the new name of the test case	new name ignored
-m	--comment	a new comment describing the test case	new comment ignored

Table 2.15: Options for command alter-test-case

2.4.16 Remove-test-cases

This command is run by:

```
codecover (remove-test-cases|rt) [options]
```

With this command the actor can remove one or more test cases of a test session from a session container.

Option		Parameter and description	Required
Short	Long		
-c	--container	the session container to use	•
-s	--session	the name of the test session	•
-t	--test-case	the name of the test case to be removed; this argument can occur more than one time – once for every test case	•

Table 2.16: Options for command remove-test-cases

2.4.17 Help

```
codecover (help|h) [%command%]
```

Using the command **help** without using an optional command, the program prints a help page containing the command overview (see section 2.4.3).

If a command is given, the program prints an option and parameter overview of the given command. Has the same effect like `codecover %command% --help`.

2.5 Configuration

2.5.1 Overview

Eclipse creates files containing preferences and other information at the run time. These files are stored in the default folders defined for Eclipse plug-ins. They can be separated into global preferences and project wide properties. Files to store are:

- preferences for the plug-in
- properties for every Eclipse project✓
- stored session containers with test sessions
- instrumented code files✓
- compiled instrumented code files

There are several options which control the behavior and the appearance of the software. For the Eclipse user, there is the dialog **PREFERENCES** for Eclipse wide configuration and **PROPERTIES** for project wide configuration. (see section 2.5.2)

For the batch mode there is a default configuration in the release jar, which can not be altered by the shell user but can be overwritten for each batch run by setting options on the command line.

The report style is configured by template XML files (see section 2.6), which can be processed using the batch mode.

2.5.2 Configure Eclipse plug-in

To configure the behavior and the general appearance of the Eclipse plug-in, a configuration file is stored. The Eclipse default mechanism for storing plug-in preferences is used. The target folder will be a sub folder of the **.metadata** folder in the Eclipse workspace.

The Eclipse user can use the Eclipse dialog **PREFERENCES** to edit the Eclipse-wide plug-in preferences of the software. The Eclipse user clicks on the menu **WINDOW** and the entry **PREFERENCES...** In the Eclipse configuration dialog, there is an entry *CodeCover* – the configuration section of the software. (see section 3.9)

On the corresponding dialog page, the user can configure the following preferences:

Configurable property	Available options
colors of the source code highlighting	for each code element – covered, partly covered and not covered – one color from a color chooser and an enable button

Table 2.17: Configurable properties of Eclipse

To configure project properties, the actor uses the Eclipse dialog **PROPERTIES** (see sec-

tion 3.10). These preferences are also stored using common Eclipse preference methods for plug-ins.

In the dialog PROPERTIES there is a section *CodeCover* where following properties can be configured:

Configurable property	Available options
coverage criteria	a not empty multiple selection out of statement coverage✓, branch coverage✓, condition coverage✓, loop coverage✓

Table 2.18: Configurable properties of an Eclipse project

2.6 Report

2.6.1 HTML

2.6.1.1 Overview

The hierarchic HTML✓ report consists of a set of HTML files placed into a directory tree. The HTML files contain the results of the coverage measurement.

There are three different types of HTML files: code pages, selection pages and title pages. These types are in a hierarchical order: the top-most page is the title page, followed by a number of selection pages. The bottom-most page type is the code page. Depending on the programming language the SUT✓ is written in, the depth of this structure may vary.

Each page type contains a lexicographically ordered list of elements with the coverage results measured for this element. Results for each coverage criterion✓ are always shown in two columns: in the first column, the number of covered items (e.g. branches) and the total number of items is written, separated by a slash, and in the second column the percentage of coverage is written with a colored bar graph visualizing this percentage.

2.6.1.2 Title page

Each report has exactly one title page named `index.html` that shows a summary of the measured coverage of the whole project[✓] (as far as it was instrumented).

This summary is followed by a list of metrics, number of instrumented packages, classes and methods.

The next element of the title page is a list as described in subsection *Overview*. It contains the top-most structural elements the programming language of the SUT provides. Each of these elements is a link to a file on the next deeper level. If the language only has two hierarchical levels, that file is a code page, otherwise it is a selection page.

In the following the title page shows an overview of the test cases[✓]. If JUnit test cases were used, the test case overview is enriched with these information. Here is an example of this overview:

Number of test cases	8
Number of JUnit test cases	6
Number of failures	3
Number of errors	1

Table 2.19: Draft of the test case overview at the report title page

Date	Test case name	Comments
2007-03-13 15:43:02	GUI test 1	
2007-03-13 18:09:18	GUI test 2	
2007-03-13 21:55:57	Black box test	
2007-03-13 23:01:20	tests.MoneyTest.testMoney1	
2007-03-13 23:01:22	tests.MoneyTest.testMoney2	failure AssertionFailedError at MoneyTest.java:23
2007-03-13 23:06:28	tests.PersonTest	tests.PersonTest.testSetName tests.PersonTest.testSetSalary error ArithmeticException at Person.java:45
2007-03-13 23:06:29	tests.DatabaseTest	tests.DatabaseTest.testLoad tests.DatabaseTest.testStore failure AssertionFailedError at DatabaseTest.java:57 tests.DatabaseTest.testCommit

Die Spezifikation des Berichts hält sich mit Formatierungsvorgaben zurück. Sie gibt jedoch detailliert vor, welche Inhalte im Bericht auftauchen müssen.

Alternativ zur Einbettung in das Spezifikationsdokument hätte auch eine eingescannte Von-Hand-Zeichnung im Anhang verwendet werden können.

2007-03-13 23:06:44	tests.FileTest	tests.FileTest.testImport error AssertionFailedError at FileTest.java:21 tests.FileTest.testExport
---------------------	----------------	--

Table 2.20: Draft of the test case table at the report title page

The first three test cases were captured out of JUnit. For the fourth and the fifth test case (tests.MoneyTest.testMoney) test methods of JUnit test cases were used as test cases. The other test cases are equal to the JUnit test cases. To allow a more detailed inspection, their test methods are show too. In the column *Failures and Errors* JUnit failures and errors are listed for JUnit test cases and test methods.

2.6.1.3 Selection page

Each selection page belongs to one element of the SUT at one hierarchical level. In Java, e.g., one selection page could belong to the package *Package A*. A selection page starts with a link to the file belonging to the next-higher level, which can be the title page or another selection page, and the overview of the element's name and coverage results measured for it. The rest of the page is a list as described above with the structural elements on the next deeper level. Each of these elements is linked to a file of the next deeper level. If the level of the current page is last but one, the linked file is a code page, otherwise it is a selection page of the next deeper level.

2.6.1.4 Code page

Each code page starts with a link to the file belonging to the next higher level, which can be the title page or a selection page, the name of the current element and the overview of the coverage results for the current element (e.g. a class in Java).

At the bottom of the page is the source code belonging to this element. If condition coverage[✓] is activated, a table with the covered boolean expressions is written after each condition.

Between the code and the overview stands a list as described above with the deepest structural elements of the programming language provides (e.g. methods in Java). Each item of the list provides a link to an anchor in the corresponding line in the code.

2.6.1.5 Implementation for Java and COBOL

For Java software, the title page lists the packages of the project[✓] and links to selection pages. Each selection page belongs to a package. This selection pages link to code pages. Each code page belongs to a class. In each code page is a list of the methods in the corresponding class.

For COBOL software, the title page lists the sections of the program. There are no selection pages. Each entry of the list on the title page links to one code page. The code pages contain no list, because there are no next-deeper elements than sections.

2.7 Instrumentation, types of coverage and measurement

2.7.1 Instrumentation process

The instrumentation[✓] process uses instrumentable files as input, adds additional counters at all code elements of interest and saves the instrumented code file[✓] in a given target folder. Where these counters are placed and how they are used must be clarified in the software design.

When using Eclipse for the instrumentation process, all instrumented code files of a Eclipse project are persistently saved in a sub folder of the plug-in's properties folder of the project. Moreover the compiled instrumented code files are stored in a bin folder too.

Saving these already instrumented and compiled files allows Eclipse to execute an entry point[✓] of the Eclipse project and using an already created code base[✓]. So not every instrumentation process creates a new code base[✓].

Code files that are not USED FOR COVERAGE MEASUREMENT (see section 2.3.4.1) are compiled into a folder, where the compiled instrumented files are stored as well. Moreover, non-code files of the original source folder are copied too. This is done to emulate the original bin folder when measuring the coverage. If these files are too big, meaning there is insufficient free disk space, the standard message for insufficient disk space is shown in Eclipse (if the Actor uses the Eclipse plug-in) or an error message is written in the console to informs the user about this problem asking him to resolve it.

This can also happen if the space is not enough to compile the instrumented source files or to write the report.

2.7.2 Statement coverage

Statement coverage is defined in the glossary shipped with this specification⁴. Also *basic statement*⁵ is defined there generically.

For Java *basic statement* is according to the Java Grammar⁴:

- StatementExpression ;
- break [Identifier] ;
- continue [Identifier] ;

In COBOL, everything that is (according to the COBOL grammar for JavaCC⁵) matched by `void Statement()` except `void IfStatement()` and `void PerformStatement()` is counted as a statement.

In general, statement coverage is defined as a percentage that is calculated as follows for the instrumented part of the SUT:

$$\frac{\text{number of covered basic statements}}{\text{total number of basic statements}}$$

2.7.3 Branch coverage

Branch coverage is defined in the glossary shipped with this specification.

If the programming language of the SUT supports exception handling, the branches implied by the *possible* exceptions are excluded from the branch coverage⁴ calculations but explicit TRY-CATCH BLOCKS are treated as branches: one for running without an exception and one for every catch statement.

Branch coverage is defined as a percentage that is calculated as follows for the instrumented part of the SUT:

$$\frac{\text{number of covered branches}}{\text{total number of branches}}$$

⁴http://java.sun.com/docs/books/jls/third_edition/html/syntax.html

⁵<http://mapage.noos.fr/~bpinon/cobol.jj>

Eine funktionale Anforderung kann auch aus einer Definition oder einer mathematischen Formel bestehen. Diese Anforderungen gelten dann nicht für einen konkreten Anwendungsfall sondern finden global Beachtung. Durch die mathematische Präzision kann man aus dieser Art von Anforderung auch leicht Testfälle ableiten. Siehe Kriterium *Methodik [K-31]* auf Seite 93.

2.7.4 Condition coverage

2.7.4.1 General view

Condition coverage and *strict condition coverage*¹ are defined in the glossary shipped with this specification.

The software uses the strict condition coverage but it is intended that other condition coverage criteria can be adapted with small effort (see section 4.11).

In general, strict condition coverage is defined as a percentage that is calculated as follows for the instrumented part of the SUT:

$$\frac{\text{number of covered basic boolean terms}}{\text{total number of basic boolean terms}}$$

There are some characteristics handling condition coverage for the Java and COBOL programming language. They are considered in the next sections.

2.7.4.2 Short-circuit operators

Some languages, e.g. Java, provide so called short-circuit boolean operators: the operands are only evaluated as far as they could affect the result of the whole expression. For example, as `||` is the short-circuit logical OR operator in Java, if `A` in `(A || B)` is true, `B` is not evaluated at all. To cover `A`, it must be once false while `B` is false, and once true while the value of `B` does not matter since it is not evaluated.

If the normal logical OR operator `|` was used, `(A | B)`, `B` would be required to stay false in both cases for `A` to be covered.

2.7.4.3 Java ternary operator

There are two different cases in handling the Java ternary operator `(A ? B : C)` in conditional expressions.

If the operator is used as a boolean term in a conditional expression, as in

```
if (x > 5 ? isA() : y == 7) {...},
```

`A`, `B` and `C` are considered *separate* coverable items. The use in the conditional expression implies that `B` and `C` are boolean expressions themselves. The coverage is determined based on the same criteria as described above: the whole expression must change if the

covered basic boolean term is changed, while the other boolean terms stay the same, as far as they are evaluated. That is:

- to cover A, B has to be the opposite of C, otherwise the change of A would not affect the whole expression,
- to cover B, A must stay true, C may have any value since is not evaluated and B must evaluate both to true and false,
- to cover C, A must stay false, B may have any value since it is not evaluated and C must evaluate both to true and false.

In all other cases, the ternary operator does not affect the condition coverage. For example, in

```
if (i == (foo ? 2 : 3)) {...},
```

the expression `i == (foo ? 2 : 3)` is a *single* basic boolean term.

2.7.4.4 COBOL boolean abbreviations

COBOL provides abbreviations in boolean expressions, e.g. `IF A = 3 OR = 7`. These abbreviations are converted to their long form, in the example `IF A = 3 OR A = 7`, and checked the usual way: to achieve full strict condition coverage, `A = 3`, `A = 4` and `A = 7` would be sufficient.

2.7.5 Loop coverage

Loop coverage is defined in the glossary shipped with this specification.

Loop coverage does not consider elements of the source code as coverable items⁷ but the number of times the loop body is entered. The coverable items are:

- loop body is not entered
- loop body is entered once, but not repeated
- loop body is repeated more than one time

Looping statements like do-while cannot be bypassed and have only two possible coverable items.

In general, loop coverage is define as a percentage that is calculated as follows for the

instrumented part of the SUT:

$$\frac{\text{number of covered coverable items}}{\text{total number of coverable items}}$$

2.7.6 Coverage measurement

The process of the coverage measurement needs the instrumented code files[✓]. They are compiled together with the uninstrumented code files. When the instrumented SUT is executed, a coverage log[✓] is produced. This log contains counters for all instrumented statements and code elements.

The name of the log file can have one of the following formats:

`coverage-log.clf`

or

`coverage-log-yyyy-MM-dd-HH-mm-ss-SSS.clf`

The date and time refer to the start of the coverage measurement. An example is `coverage-log-2007-06-07-09-48-12.clf`.

If a file with the given name still exists, it is not overwritten, but the name of the new file is extended by (1), (2) and so on. The instrumenter can make the instrumented SUT to support additional parameters that the tester can specify a path of the coverage log file or enable overwriting – e.g. environment variables or system properties for Java.

2.7.7 Coverage analysis

The coverage log[✓] file is processed in the analysis period. For this purpose, the Eclipse user can use the *analyze coverage log* use case (see section 2.3.4.6) or the complete instrumentation[✓], execution and analysis use case (*measure coverage*, section 2.3.7). The shell user can use the command `codecover analyze` (see section 2.4.7).

The result of the analysis process is a test session, that could be used for report generation (see section 2.3.7) or coverage analysis (see section 2.3.5.2).

2.8 Language support

The software can show all texts used in the Eclipse plug-in as well as in the reports in any language of which all needed characters are part of the Unicode standard. All releases will be delivered in German and English localizations. The default Eclipse language support for plug-ins is used.

The Eclipse plug-in tries to use the language Eclipse uses, taking English if it can't find an appropriate language setting. The language used in reports, on the other hand, results from the template files which not only define the layout but also set every string used in the report outputted by *CodeCover*.

The batch interface is English only.

2.9 JUnit integration

2.9.1 Preface

The term *test case* can be mixed up in this section. For this reason we distinguish the terms *JUnit test case* and *test case*⁶ in the understanding of this software.

The phrase *a test case failed* will be used in the meaning that a test case had an unexpected behaviour or causes an error.

2.9.2 Basic concepts

Till now, only the manually added method calls can set the start and the end of a test case:

```
Protocol.startTestCase("JUnit Test 1")
```

This rudimentary test case notification mechanism for Java will be enhanced. Therefore JUnit⁶ will be integrated in the software. Thereby the software is informed about the start and the end of JUnit test cases while the instrumented SUT is running.

Es ist mitunter sehr schwierig eine Anforderung in funktional bzw. nichtfunktional einzuordnen. Auch die einschlägige Literatur ist dabei keine Hilfe. Man könnte deshalb auch die Anforderungen zur Sprache unter dem Gesichtspunkt der Internationalisierbarkeit betrachten. Somit wäre diese Anforderung nichtfunktional.

Die Hinführung zur Anforderung erinnert eher an einen wissenschaftlichen Artikel als an eine sachliche Spezifikation. Die Spezifikation sollte knapp gehalten werden.

Darüber hinaus wurde die Testfallabgrenzung bereits in Kapitel 2.3.4.5.4 spezifiziert und ist damit an dieser Stelle redundant.

⁶<http://www.junit.org>

It must be configurable whether to use the JUnit test cases or the test methods as test cases in the understanding of the software. This must be decidable for each JUnit test run. The test cases of the SUT need not to be instrumented or changed for this feature.

To support these features, the software must log static information of the JUnit test cases and observe its run. This includes for each test case:

- the name of the related JUnit test case class
- the names of the test methods of the JUnit test case
- whether the test methods of the JUnit test case failed or not
- if a test method failed, which failure respectively error was the reason
- the date and time of the execution
- the belonging code coverage[✓] results

If a JUnit test case is used as a test case, the test case has to store all test methods of the JUnit test case. The related test case is marked as failed if at least one test method has failed.

If the test methods of a JUnit test case are used as test cases, the test case name must have a unique name to identify the test method.

All the data collected must be stored in the coverage log[✓] file, cause this is the only result of the coverage measurement phase.

2.9.3 Compatibility

The approach must be compatible to:

- the JUnit 3.8.x family
- the JUnit 4.x family
- the Eclipse plug-in family: `org.junit_3.8.x`
- the Eclipse plug-in family: `org.junit4_4.x`

This means, that the JUnit integration works with all four families and allows the features described above.

The implementation of the JUnit integration must be compatible to the Java version of the supported JUnit family – e.g. the implementation of the support for JUnit 3.8.x must be compatible to Java 1.4. For this reason the software supports test case execution for older systems, that still rely on Java 1.4.

2.9.4 Report

The additional JUnit test case information are added to the report. See section 2.6.1.2.

2.10 ANT integration

2.10.1 Preface

CodeCover provides an Apache ANT⁷ integration.

The *CodeCover* ANT integration will provide a `codecover` command which will have subcommands as its content, i.e. an example will look like:

```
<target name="foo">
  <codecover>
    <subcommand1 param1="bar" param2="foobar" />
    <subcommand2>
      <someElement param="42" />
    </subcommand2>
  </codecover>
</target>
```

2.10.2 Subcommand overview

The following subcommands are available:

Target	Description
load	load a session container

⁷<http://ant.apache.org/>

Ähnlich wie bei der Beschreibung der Batch-Schnittstelle verhält es sich auch bei der Beschreibung der Ant-Integration. Es werden keine Use-Cases verwendet sondern die Tasks und deren Parameter übersichtlich in tabellarischer Form spezifiziert. Siehe Kriterium *Methodik [K-31]* auf Seite 93.

Target	Description
save	save a session container
createContainer	creates a new container
instrument	Instrumentation of code files
analyze	Analysis of a coverage run to create a test session
report	Generating a report from a test session
mergeSessions	Merging two or more test sessions
alterSession	Altering test session information
copySessions	Copy test sessions from one session container to another
removeSessions	Removing test sessions from a session container
mergeTestCases	Merging two or more test cases
alterTestCase	Editing test case information
removeTestCases	Removing test cases from a session container

2.10.3 load

This subcommand loads a session container. The loaded session container then can be used in future subcommands.

Syntax:

```
<load containerId="..." filename="..." />
```

Attribute	Required	Description
containerId	•	An ID assigned to the loaded container. This ID can later be used to reference this container.
filename	•	The file to load. If filename is a relative filename, it will be interpreted relative to the project's basedir.

2.10.4 save

This subcommand saves a session container.

Syntax:

```
<save containerId="..." filename="..." override="..." />
```

Attribute	Required	Description
containerId	•	The ID of a container which will be saved.
filename	•	The name of the new file. If <code>filename</code> is a relative filename, it will be interpreted relative to the project's basedir.
override		If this attribute is true, an existing file will be overridden. Defaults to true.

2.10.5 createContainer

This subcommand creates a new test session container using the static information from another container.

Syntax:

```
<createContainer oldContainerId="..." newContainerId="..." />
```

Attribute	Required	Description
oldContainerId	•	The ID of a container.
newContainerId	•	An ID assigned to the newly created container. This ID can later be used to reference this container.

2.10.6 instrument

This subcommand instruments source code.

Syntax:

```
<instrument containerId="..." language="..." instrumenter="..."
  destination="..." charset="..." copyUninstrumented="..." override="...">
  <source ...>
    ...
  </source>
```

```

<criteria>
  <criterion name="..." />
  <criterion name="..." />
  ...
</criteria>
</instrument>

```

Attribute	Required	Description
containerId	•	An ID assigned to the newly created container. This ID can later be used to reference this container.
language	•	The name of the programming language.
instrumenter	•	The full name of the instrumenter to use. TODO: THIS APPEARS IN THE BATCH SPECIFICATION; IT HOWEVER IS NOT IMPLEMENTED. WHAT IS THIS FOR? IS THIS OPTION REALLY REQUIRED? YES!
destination	•	The destination directory for the instrumented files. If destination is a relative filename, it will be interpreted relative to the project's basedir.
charset		The character encoding of the source files. If none is given, the system default will be used.
copyUninstrumented		If this attribute is true, all non-instrumented files in the root directory will be copied to the destination. Defaults to false.
override		If this attribute is true, existing files will be overridden. Defaults to true.

Element	Required	Description
source	•	A fileset ⁸ pointing to the files to instrument. The root directory of the fileset has to point to the root directory of the source files.
criteria		A list of criteria to be used for instrumentation. If this element isn't given, all criteria will be used.

⁸<http://ant.apache.org/manual/CoreTypes/fileset.html>

ToDoS sind für die Erstellungsphase einer Spezifikation sehr hilfreich, um offene Punkte oder Hinweise zu einer konkreten Stelle festzuhalten. In einer freigegebenen Version sollten ToDoS jedoch nicht mehr auftauchen. Siehe Kriterium *Einhaltung grundlegender Dokumentanforderungen [K-01]* auf Seite 49.

2.10.7 analyze

This subcommand takes the information from a coverage log and writes it into a session container.

Syntax:

```
<analyze containerId="..." coverageLog="..."
  name="..." comment="..." charset="..." />
```

Attribute	Required	Description
containerId	•	The ID of a container in which the information will be written.
coverageLog	•	The coverage log file. If <code>coverageLog</code> is a relative file-name, it will be interpreted relative to the project's basedir.
name	•	The name of the new test session.
comment		The comment for the new test session. If none is given, the comment is empty.
charset		The character encoding of the coverage log. If none is given, the system default will be used.

2.10.8 report

This command creates a report containing information about test cases.

Syntax:

```
<report containerId="..." destination="..."
  template="..." override="...">
  <testCases>
    <testSession name="...">
      <testCase pattern="*" />
    </testSession>
    <testSession pattern="foo.*bar">
```

```

        <testCase pattern=".*" />
    </testSession>
    <testSession name="...">
        <testCase name="..." />
        <testCase name="..." />
        <testCase name="..." />
    </testSession>
</testCases>
</report>

```

Attribute	Required	Description
containerId	•	The ID of a container in which the information will be written.
destination	•	The destination file for the report. If destination is a relative filename, it will be interpreted relative to the project's basedir.
template	•	The template file. If template is a relative filename, it will be interpreted relative to the project's basedir.
override		If this attribute is true, existing files will be overridden. Defaults to true.

Element	Required	Description
testCases	•	The test cases to use in the report. This element is described in 2.10.17

2.10.9 mergeSessions

This command merges multiple sessions in a session container into one session.

Syntax:

```

<mergeSessions containerId="..." name="..." comment="..."
    removeOldSessions="...">
    <testSessions>

```

```
<testSession name="..." />
<testSession pattern="foo.*bar" />
</testSessions>
</mergeSessions>
```

Attribute	Required	Description
containerId	•	The ID of a container of the used container.
name	•	The name of the new test session.
comment		The comment for the new test session. If none is given, the comment is empty.
removeOldSessions		If this attribute is true, the original test sessions will be removed. Defaults to false.

Element	Required	Description
testSessions	•	The test sessions to merge. This element is described in 2.10.16

2.10.10 alterSession

This command modifies the name and/or the comment of a test session.

Syntax:

```
<alterSession containerId="..." session="..." name="..." comment="..." />
```

Attribute	Required	Description
containerId	•	The ID of a container of the used container.
session	•	The old name of the test session.
name		The new name of the test session. If none is given, the name will not be changed.
comment		The new comment for the test session. If none is given, the comment will not be changed.

Generell sollten im gesamten Dokument keine Objekte über den Rand des Blocksatzes hinausragen.

2.10.11 copySessions

This command will copy session from one session container into another.

Syntax:

```
<copySessions sourceContainerId="..." destinationContainerId="..."
  removeOldSessions="...">
  <testSessions>
    <testSession name="..." />
    <testSession pattern="foo.*bar" />
  </testSessions>
</copySessions>
```

Attribute	Required	Description
sourceContainerId	•	The ID of the container of copy from.
destinationContainerId	•	The ID of the container to copy to.

Element	Required	Description
testSessions	•	The test sessions to copy. This element is described in 2.10.16

2.10.12 removeSessions

This command removes sessions from a session container.

Syntax:

```
<removeSessions containerId="...">
  <testSessions>
    <testSession name="..." />
    <testSession pattern="foo.*bar" />
  </testSessions>
</removeSessions>
```


Attribute	Required	Description
containerId	•	The ID of a container to remove sessions from.

Element	Required	Description
testSessions	•	The test sessions to remove. This element is described in 2.10.16

2.10.13 mergeTestCases

This command merges multiple test cases in a session into one test case.

Syntax:

```
<mergeTestCases containerId="..." name="..." comment="..."
  removeOldTestCases="...">
  <testCases>
    <testSession name="...">
      <testCase pattern="foo.*bar" />
      <testCase name="..." />
      <testCase name="..." />
      <testCase name="..." />
    </testSession>
  </testCases>
</mergeTestCases>
```

Attribute	Required	Description
containerId	•	The ID of a container of the used container.
name	•	The name of the new test case.
comment		The comment for the new test case. If none is given, the comment is empty.
removeOldTestCases		If this attribute is true, the original test cases will be removed. Defaults to false.

Element	Required	Description
testCases	•	The test cases to merge. This element is described in 2.10.17. The list has to contain exactly one test session. This session will also contain the new test case.

2.10.14 alterTestCase

This command modifies the name and/or the comment of a test case.

Syntax:

```
<alterTestCase containerId="..." session="..." testCase="..." name="..."
  comment="..." />
```

Attribute	Required	Description
containerId	•	The ID of a container of the used container.
session	•	The name of the test session.
testCase	•	The old name of the test case.
name		The new name of the test case. If none is given, the name will not be changed.
comment		The new comment for the test case. If none is given, the comment will not be changed.

2.10.15 removeTestCases

This command removes sessions from a session container.

Syntax:

```
<removeTestCases containerId="...">
  <testCases>
    <testSession name="...">
      <testCase pattern=".*" />
    </testSession>
    <testSession pattern="foo.*bar">
```

```
        <testCase pattern=".*" />
    </testSession>
    <testSession name="...">
        <testCase name="..." />
        <testCase name="..." />
        <testCase name="..." />
    </testSession>
</testCases>
</removeTestCases>
```

Attribute	Required	Description
containerId	•	The ID of a container to remove test cases from.

Element	Required	Description
testCases	•	The test cases to remove. This element is described in 2.10.17

2.10.16 List of test sessions

A `testSessions` element contains a list of `testSession` elements. Each `testSession` element has either a `name` attribute, which gives the exact name of the test session it matches, or a `pattern` attribute, which gives a Java regular expression which test sessions names it will match.

So e.g.

```
<testSessions>
    <testSession name="42" />
    <testSession name="23" />
    <testSession pattern="foo.*bar" />
</testSessions>
```

will match the session with the name “42”, the session with the name “23” and any session starting with “foo” and ending with “bar”.

2.10.17 List of test cases

A `testCases` element contains a list of `testSession` elements (as described above), however here each `testSession` element contains a list of `testCase` elements. Each `testCase` element has either a `name` attribute, which gives the exact name of the test case it matches, or a `pattern` attribute, which gives a Java regular expression which test cases names it will match.

So e.g.

```
<testCases>
  <testSession name="foo">
    <testCase pattern=".*" />
  </testSession>
  <testSession pattern="foo.*bar">
    <testCase pattern=".*" />
  </testSession>
  <testSession name="42">
    <testCase name="1" />
    <testCase name="2" />
    <testCase name="3" />
  </testSession>
</testCases>
```

will contain all test cases from the test session “foo”, all test cases from test sessions which names start with “foo” and end with “bar” and the test cases “1”, “2” and “3” from the test session “42”.

2.10.18 Examples

2.10.18.1 Instrumentation

```
<codecover>
  <instrument containerId="container" language="java"
    destination="instrumented" charset="utf-8" copyUninstrumented="true">
    <source dir="src">
```

```
<include name="**/*.java" />
</source>
<criteria>
  <criterion name="st" />
  <criterion name="br" />
</criteria>
</instrument>
<save containerId="container" filename="container.xml" />
</codecover>
```

This will instrument all java files in the directory `src`, write the result into `instrumented` and use statement and branch coverage. The resulting test session container will be written into `container.xml`.

2.10.18.2 Analysis

```
<codecover>
  <load containerId="container" filename="container.xml" />
  <analyze containerId="container" coverageLog="coverage.log"
    name="New Test Session" />
  <save containerId="container" filename="container.xml" />
</codecover>
```

This will write the content of `coverage.log` into `container.xml` into a new test session called “New Test Session”.

2.10.18.3 Reporting

```
<codecover>
  <load containerId="container" filename="container.xml" />
  <report containerId="container" destination="report.html"
    template="HTML_Report_hierarchic.xml">
    <testCases>
      <testSession pattern=".*">
        <testCase pattern=".*" />
      </testSession>
    </testCases>
```

```
</report>
</codecover>
```

This will create a report in `report.html` containing all test cases from the test session container `container.xml` using the template in `HTML_Report_hierarchic.xml`.

2.11 Live Test Case Notification

The live test case notification feature provides a way to manually define test case borders during the execution of the instrumented SUT without manually modifying either the test cases or the SUT before instrumenting or compiling. Additionally, this feature can be used to automatically download the created coverage log files from a remote SUT, for example, a web application.

Live test case notification is available for Java SUTs only.

The communication between the instrumented SUT and *CodeCover* is carried out using the Java Management Extensions (JMX)⁹ remote protocol over Java RMI over TCP. As JMX is generally protocol-independent, support for other protocols can be added in the future; this is, however, beyond the scope of the *CodeCover* project.

2.11.1 Basic principle of operation

JMX uses a client/server model to enable management and monitoring of a Java application. The target Java application (the SUT) acts as a server. As long as the application is being executed, one or more clients can connect to it, query property values and initiate operations exposed by the server. The communication model and infrastructure is defined in the JMX specification and implemented in most major JRE Version 5 or later distributions as well as in both open source and commercial products and libraries. The target application only needs to provide its specific management functionality by creating objects that adhere to a particular interface and naming convention (called *MBeans* in JMX terminology) and registering them with a *MBean Server* which is provided by the JMX implementation.

⁹<http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/>

During the instrumentation process of the SUT, a MBean class is added to the SUT along with the standard measurement classes. When the SUT starts and the coverage logging facilities are initialized, an object of this class is created and registered with the MBean Server. If the SUT was started in a JVM which supports remote JMX, operations and properties of the *CodeCover* MBean are automatically exported to the network interface. At the time of this writing, remote JMX is not enabled by default but can be simply enabled by setting special system properties at startup of the JVM. The procedure of enabling remote JMX on major JREs is to be documented in the user's manual.

The *CodeCover* client, which is a part of the *CodeCover* Eclipse plug-in, connects to the JMX server in the SUT's JVM and executes operations on the MBean as requested by the Eclipse user. These operations trigger test case notification information to be written into the coverage log file.

It is assumed that only one client accesses the instrumented SUT via remote JMX. Concurrent accesses are not supported as they wouldn't have any meaningful semantics.

2.11.2 MBean Interface

The MBean exports the following properties and operations to remote clients:

1. An operation to start a test case with a name and to end a test case. These operations correspond to methods described in section 2.1.
2. An operation which instructs the measurement classes to finish the coverage logging and close the coverage log file.
3. A read-only property that contains the file name of the current coverage log file.
4. An operation which allows the client to download the contents of the current coverage log file.

2.11.3 Application life cycle issues

2.11.3.1 Java SE applications

In standard Java SE applications, the MBean is initialized together with the rest of the coverage logging classes.

2.11.3.2 Web applications

The MBean must be registered with the MBean Server when the web application is initialized and unregistered when the application shuts down. Since the application startup and shutdown times are not necessarily identical with the application container's ones, some interaction with the application container is necessary to get the notifications on startup and shutdown.

This is done by installing a *context listener* into the web application context. Context listeners are part of the servlet specification¹⁰ since version 2.3. *CodeCover* provides a generic context listener class which registers and unregisters the required MBean.

The user is only required to add a generic context listener declaration to the deployment descriptor (`web.xml`) of the SUT. This process is described in the user's manual.





Moreover, as the MBean is initialized with the application and a MBean Server might be initialized together with the container, a connected client is required to listen for MBean registration and deregistration events on the MBean Server and behave accordingly to the current MBean status.

¹⁰<http://java.sun.com/products/servlet/index.jsp>

3 Graphical User Interface

3.1 Package and file states

The instrumentable items[^] (e.g. packages or source code files[^]) are presented in one of two states: *normal* and *used for coverage measurement*. The following table shows the icons for different instrumentable items in these states.

Object	normal	used for coverage measurement
Package		
Java file		

The user can change the state of an instrumentable item by selecting the USE FOR COVERAGE MEASUREMENT menu item in the context menu of an instrumentable item (e.g. in the JDT's PACKAGE EXPLORER or the generic NAVIGATOR). USE FOR COVERAGE MEASUREMENT is a check box menu item so that a second selection of this menu item removes the instrumentable item from coverage measurement.

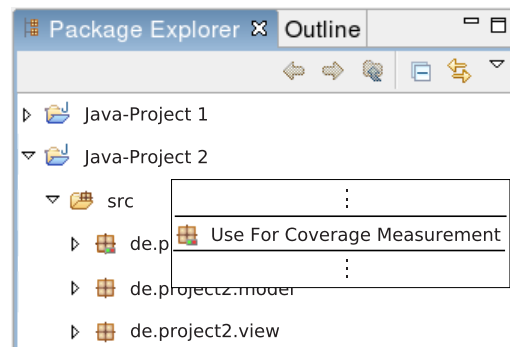


Figure 3.1: Package selection

Die grafische Benutzungsoberfläche kann durchaus in einem eigenen Kapitel beschrieben werden. Dadurch werden die Use-Case-Beschreibungen nicht von der Beschreibung der Oberfläche unterbrochen. Es kann auf Besonderheiten gezielt eingegangen werden und die grundlegende Verhaltensweise der Bedienelemente unabhängig von der Beschreibung der funktionalen Anforderung spezifiziert werden. Allerdings ist darauf zu achten, dass in den Use-Case-Beschreibungen Verweise auf die entsprechenden GUI-Kapitel erfolgen.

Auf die Abbildung wird nicht verwiesen. Der Leser sollte durch einen Verweis angeleitet werden, an der Stelle die Abbildung zu betrachten. Siehe Kriterium *Abbildungen und Tabellen (Äußere Form)* [K-03] auf Seite 52.

3.2 Instrumentation

The INSTRUMENT PROJECT... item is added to the PROJECT menu. This command opens the dialog window which is shown in figure 3.2.

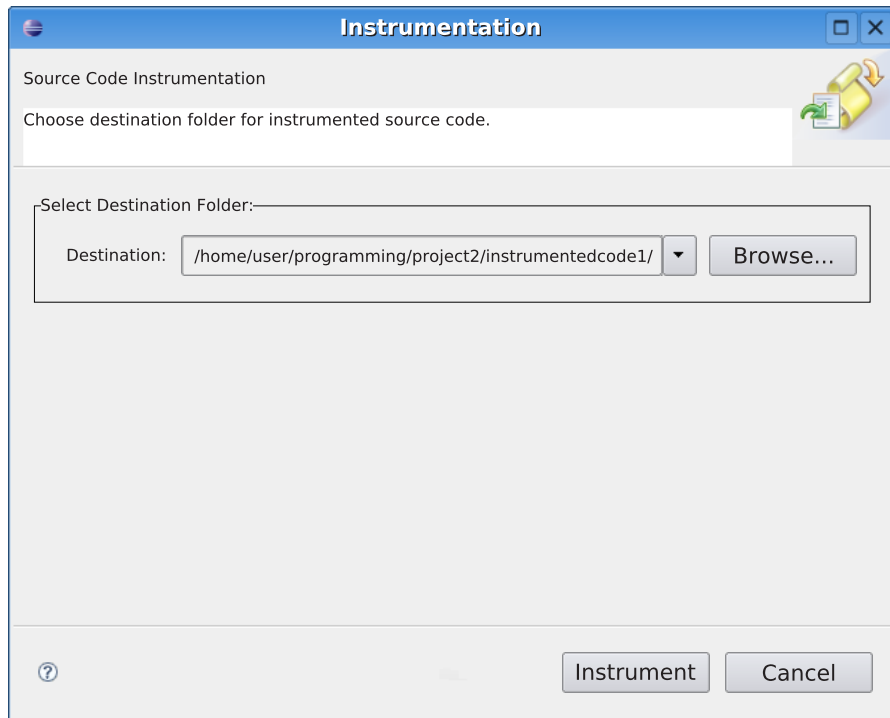


Figure 3.2: Instrumentation dialog

3.3 Launching

The software adds a new launch mode to the Eclipse workbench. This mode is called Coverage mode and works exactly like the existing Run and Debug modes. Figure 3.3 shows the pull down menu of the COVERAGE BUTTON on the tool bar. The menu items COVERAGE HISTORY, COVERAGE AS and COVERAGE... are added to the COVERAGE

menu.

The `COVERAGE...` option opens the `COVERAGE` dialog which is similar to the `Run` dialog, except that the `RUN` button in that dialog is called `COVERAGE`.

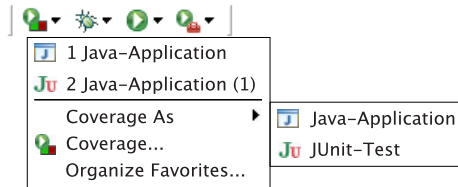


Figure 3.3: Coverage button

3.4 Coverage view

The coverage results are presented in the `COVERAGE` view. It displays the results of statement, branch, condition and loop coverage per project, package, class (including interfaces and enums) and method. This view is shown in figure 3.4.

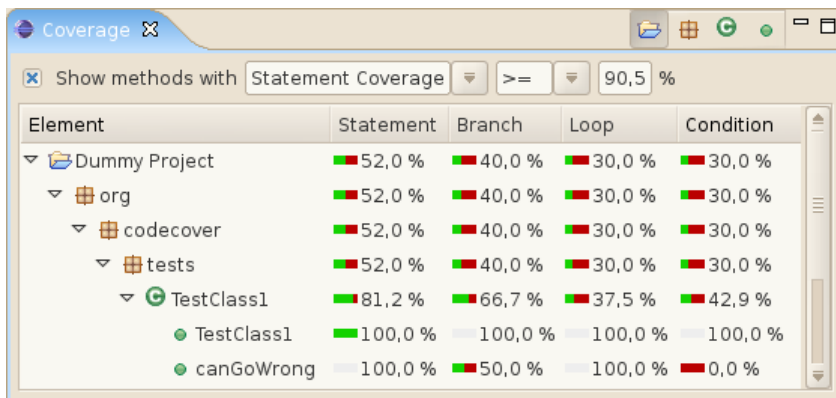


Figure 3.4: Coverage view

A grey bar is shown left of a coverage result if there are no coverable items of the associated coverage criterion in the associated `ELEMENT`.

Es müssen keine Screenshots oder ausgefeilte Grafikentwürfe erstellt werden. In der Regel reichen auch von Hand erstellte und eingescannte Konzeptentwürfe aus. Selbstverständlich müssen diese Entwürfe lesbar, übersichtlich und verständlich sein. Siehe Kriterium *Abbildungen [K-20]* auf Seite 74.

The check box in the upper-left corner of the view enables a simple filter if checked. If the filter is activated only methods with a coverage result smaller, smaller or equal, greater, greater or equal than a given percentage are shown in the coverage view. The coverage criterion to compare with can be selected in the most left combo box in the view. The operator to compare the coverage result with can be selected in the combo box right of the combo box of the coverage criteria. The percentage to compare with can be entered in the field right of the combo box of the comparison operators.

The tool bar items in the upper-right part of the view provide the possibility to select Java elements shown as root entries in the element column. Possible root entries are projects, packages, classes (including interfaces, enums and annotations) and methods.

3.5 Test sessions view

The TEST SESSIONS view lists the test sessions[↗] of the selected session container[↗]. Figure 3.5 shows an example of this view. A session container can be chosen using the combo box. The session container list entries display the date and time along with the associated project[↗].

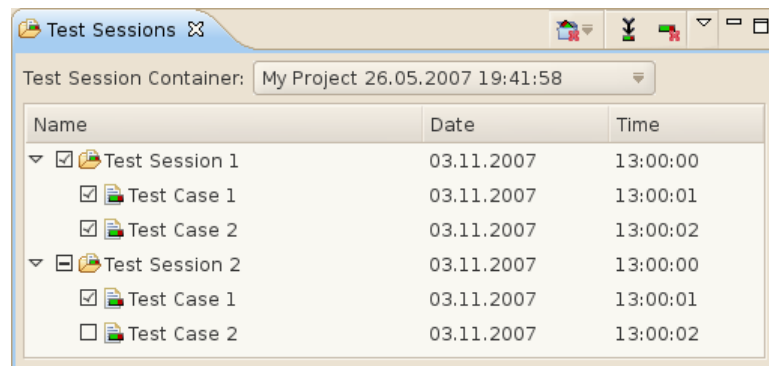


Figure 3.5: Test Sessions view

The check box in front of each row determines whether a TEST ELEMENT is activated. The coverage results of activated TEST ELEMENTS are visualized by the views of the plugin (e.g., COVERAGE view) and the source code highlighting. The visualizations are

automatically refreshed as the user activates or deactivates particular TEST ELEMENTS. By default, all TEST ELEMENTS are activated.

The activation or deactivation of a test session activates or deactivates all test cases of this test session. For test sessions the check box has an additional state, partly activated. This state is visualized by the crossed out check box in figure 3.5 and means that at least one test case of a selected test session is deactivated.

The information about the set of active (visualized) TEST ELEMENTS is stored; that is, selecting a new session container does not change the set of the active TEST ELEMENTS of the previous session container.

The tool bar items represent following commands (from left to right):

- Delete Test Session Container
 - Delete Multiple Containers
- Merge
- Delete

DELETE TEST SESSION CONTAINER deletes the active session container. Prior to the actual deletion the user has to confirm his intent to do so. By using the drop-down menu of the DELETE SESSION CONTAINER item, the user can reveal the item for the deletion of multiple session containers: DELETE MULTIPLE CONTAINERS. It opens a dialog that prompts the user to select the session containers to delete and performs the deletion after the user confirmed the selection. The item MERGE allows the user to merge the selected test elements. On selection of this item a dialog pops up which prompts the user to select the type of test element (test cases or test sessions) to merge and prompts the user to enter a name and comment for the merged test element. Furthermore the user is able to review and change the set of test elements to merge. The DELETE item deletes the selected test elements, whereas the user is asked for confirmation before the deletion is performed.

The TEST ELEMENTS have a context menu which contains following items:

- Select All
- Activate All
- Deactivate All
- Delete

- Properties

The item **SELECT ALL** selects all test elements of the active session container to be able to delete them all at once for example. **ACTIVATE ALL** and **DEACTIVATE ALL** activate or respectively deactivate all test elements of the active session container. The item **DELETE** evokes the same action as described above. The **PROPERTIES** item opens a dialog which allows the user to edit the properties of the selected **TEST ELEMENT**. The **TEST CASE PROPERTIES** dialog is shown in figure 3.6. It is possible to change the name of the selected **TEST ELEMENT**. Furthermore, a multi line comment may be entered.

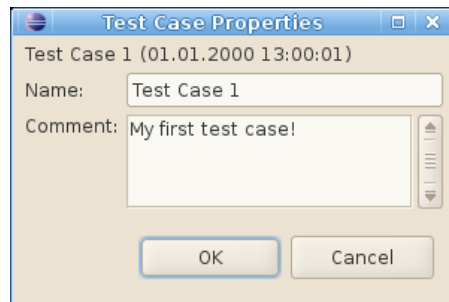


Figure 3.6: Test case properties

3.6 Import

The software extends the standard Eclipse **IMPORT** interface by adding two entries to group *CodeCover*, **TEST SESSION CONTAINER** and *CodeCover COVERAGE LOG*. Figure 3.7 shows the correspondent **IMPORT** wizard page. This dialog allows the user to select a session container[✓] and a project[✓] into which the session container will be imported. The file extensions used in the following dialogs are examples.

Selecting *CodeCover COVERAGE LOG* proceeds with the wizard page shown in figure 3.8. This import operation requires the coverage log[✓] file, created while running the instrumented program. Moreover the user has to select the session container the coverage log will be imported to and enter a name and comment for the new test session that will contain the data of the coverage log.

Hier wird auf Abbildung 3.7 verwiesen. Der Leser kann vorblättern, das Bild anschauen und dann das Lesen fortsetzen. Siehe Kriterium *Abbildungen und Tabellen (Äußere Form)* [K-03] auf Seite 52.

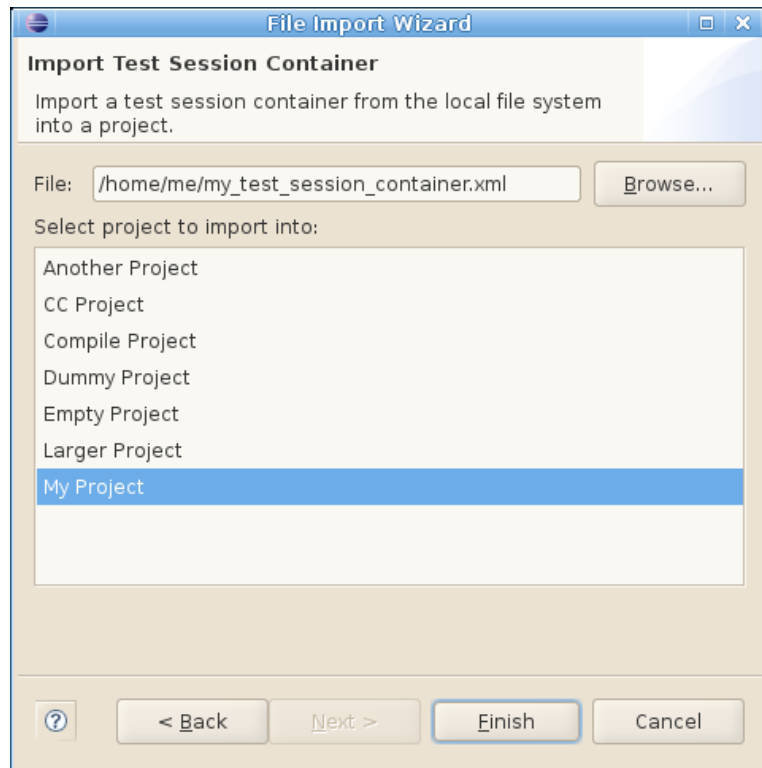


Figure 3.7: Import Test Session Container

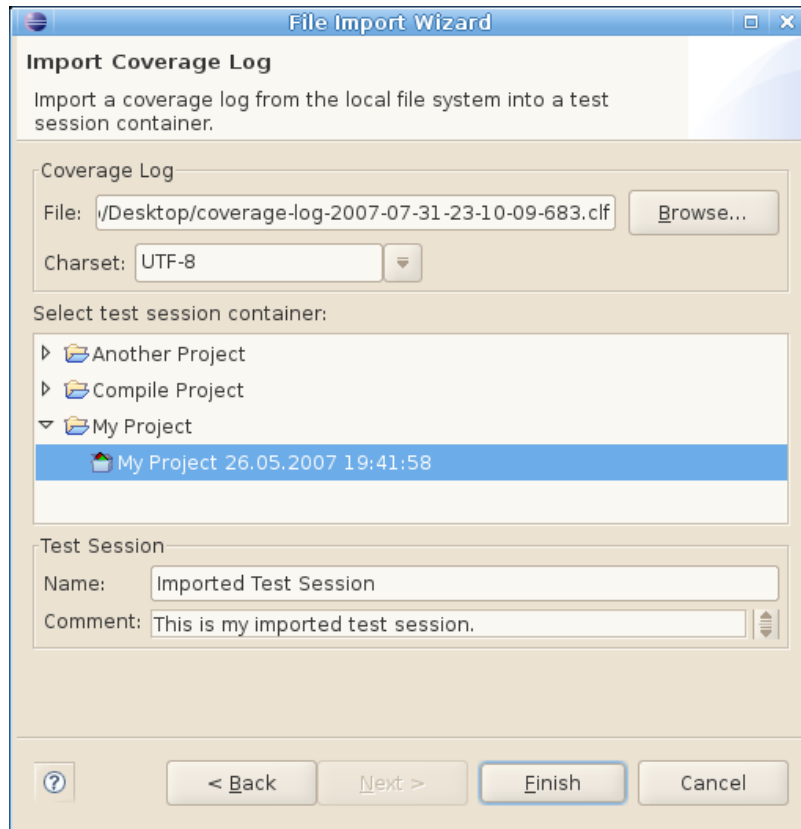


Figure 3.8: Import Coverage Log

3.7 Export

The item *CodeCover* COVERAGE RESULT EXPORT is added to the OTHER group of standard Eclipse EXPORT dialog. The respective wizard page is shown in figure 3.9. The dialog contains the list of all available test sessions⁷ in the selected code base. The list AVAILABLE TEST SESSIONS allows multiple selections. The code base can be chosen from the combo box. By default, the last code base or the code base which is used in the TEST SESSIONS view is preselected.

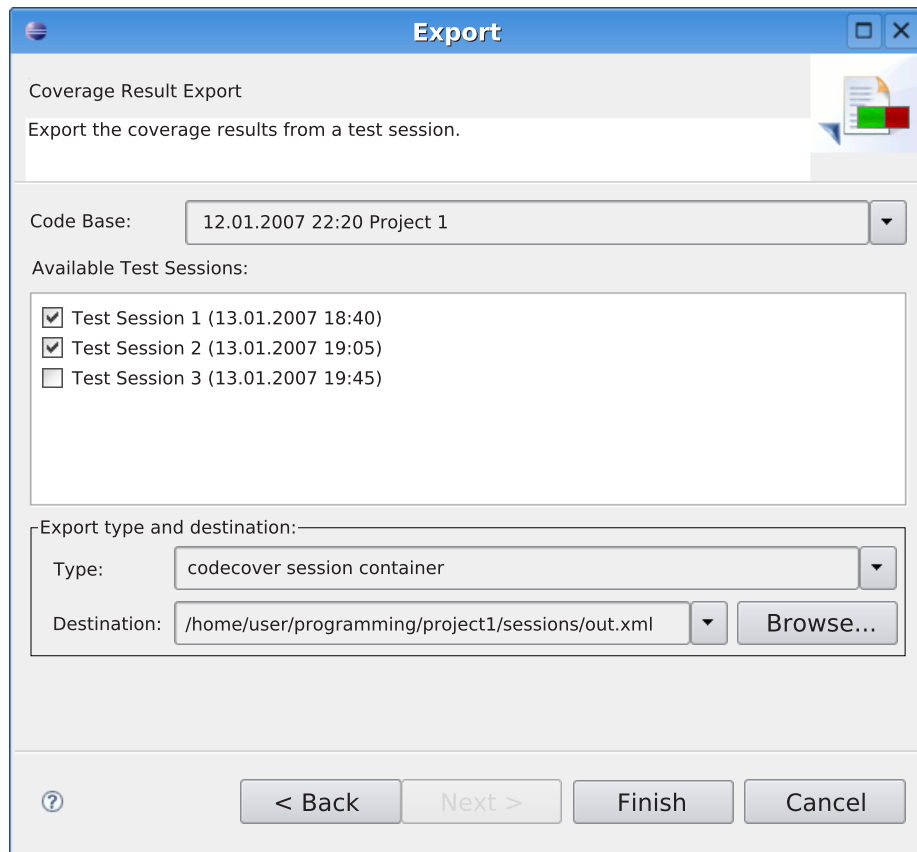


Figure 3.9: Export Test Session

Possible export types are *CodeCover* SESSION CONTAINER and *Report*. The report type has an extra wizard page which allows the user to select a report template (figure 3.10).

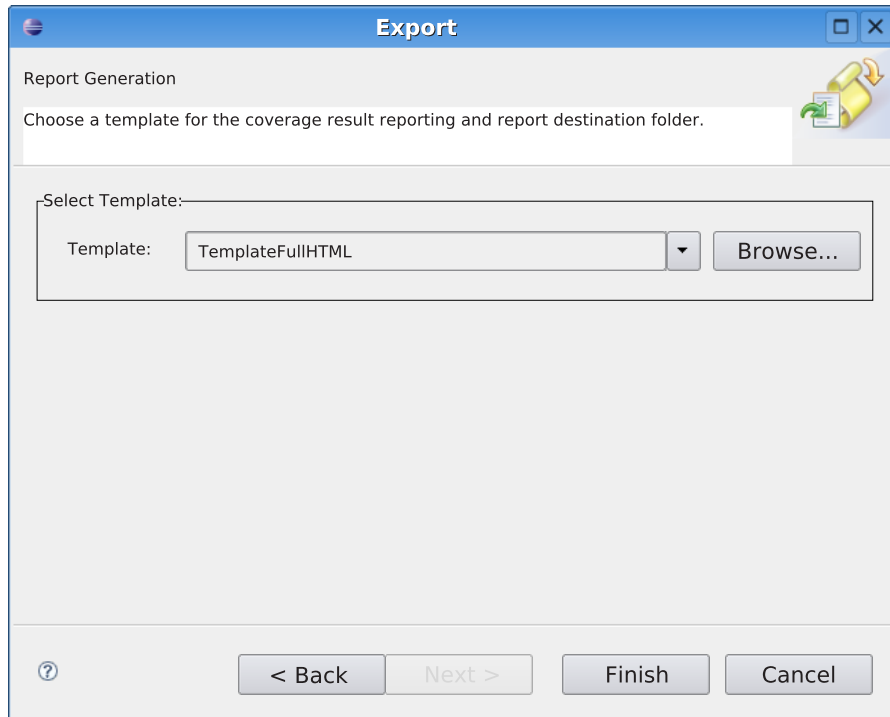


Figure 3.10: Report dialog

3.8 Source code highlighting

3.8.1 General

This section describes the visualization of coverage results by highlighting the source code of the SUT'. There are three different colors for displaying the state of a particular part of code. The default color scheme uses green for "covered", yellow for "partly covered"

and red for “not covered”. All these colors are configurable (see section 3.9). Throughout this section, the default colors are used to explain the details of the highlighting.

Statement coverage is shown by highlighting the statements’. To display branch coverage’, the keywords of conditional statements’ are highlighted. Condition coverage is represented by coloring each basic term of a condition with green or red. For loop coverage’, the keywords of looping statements are highlighted. All examples in this section show source code highlighting with all coverage criteria.

3.8.2 Java

3.8.2.1 Basic statements

Basic statements are completely highlighted either green or red, for covered and not covered statements, respectively.

3.8.2.2 Conditional statements

3.8.2.2.1 If-then-else statements

The following list describes the meaning of the background color of the `if` keyword:

- **Green:** Both branches are covered.
- **Yellow:** Only the then-branch is covered.
- **Red:** Only the else-branch is covered.

Figure 3.11 illustrates the different possibilities of if-then coverage. If the branching statement is not evaluated because it is not executed, it is also highlighted with red color.

```

if ( isA() ) {
doS();
}

if ( isA() ) {
doS();
}

if ( isA() ) {
doS();
}

```

Figure 3.11: If-then statements

The background color of `else` keyword has the following meanings:

- **Green:** The else-branch is covered.
- **Red:** The else-branch is not covered.

```

if ( isA() ) {
  doB();
} else {
  doC();
}

if ( isA() ) {
  doB();
} else {
  doC();
}

if ( isA() ) {
  doB();
} else {
  doC();
}

```

Figure 3.12: If-then-else statements

Example highlighting of if-then-else statements is shown in the figure 3.12. The left picture represents full statement, branch and condition coverage⁷. In the middle picture only the then-branch and in the right one only the else-branch is covered.

An if-then-else statement can be nested in another if-then-else statement. In that case the same color scheme as described above is applied. Figure 3.13 shows some examples for nested if-then-else statements.

```

if ( isA() ) {
  doB();
} else {
  if ( isC() ) {
    doB();
  } else {
    doC();
  }
}

if ( isA() ) {
  doB();
} else {
  if ( isC() ) {
    doB();
  } else {
    doC();
  }
}

if ( isA() ) {
  doB();
} else {
  if ( isC() ) {
    doB();
  } else {
    doC();
  }
}

```

Figure 3.13: Nested if-then-else statements

Abbildungen können manche Anforderungen prägnanter und kürzer spezifizieren, als dies in Prosa möglich wäre. Siehe Kriterium *Abbildungen [K-20]* auf Seite 74.

3.8.2.2.2 Switch statements

The following list describes the highlighting scheme for the **switch** keyword:

- **Green:** All cases are covered.
- **Yellow:** Some cases are covered, but at least one case is not covered.
- **Red:** No case is covered.

If the default section of the **switch** statement is not covered the **switch** keyword is highlighted as partly covered (yellow) as well. The highlighting is independent from the fact that the default section can be omitted. Figure 3.14 shows a sample highlighted **switch** statement with a **default** section.

For every case the keyword **case** and the **constant** are highlighted the following:

- **Green:** The case is covered.
- **Red:** The case is not covered.

If the default section is not omitted, the keyword **default** is highlighted the same way as cases.

```
switch ( iValue ) {
  case 0: doA();
          break;
  case 1: doB();
          break;
  default: doC();
}
```

Figure 3.14: Switch-statement with some cases and a default section

3.8.2.3 Looping statements

3.8.2.3.1 General

The keywords of looping statements (**while**, **for** and **do-while**) are highlighted as specified in the following list. The loop coverage[✓] is defined in section 2.7.5.

- **Green:** Loop body is covered (fulfill all requirements of loop coverage).
- **Yellow:** Loop body is partly covered (at least one requirement of loop coverage, but not all requirements).
- **Red:** Loop body is not covered at all.

3.8.2.3.2 While loops

Figure 3.15 shows full, partly and not covered **while** loops (from left to right).

```
while ( isA() ) {           while ( isA() ) {           while ( isA() ) {
  doB();                     doB();                     doB();
}
```

Figure 3.15: Highlighting of while loops

3.8.2.3.3 Do-while

Do-while loop is represented similar to the **while** loop; only the **while** keyword is highlighted. The colors are the same (see figure 3.16).

3.8.2.3.4 For

The coverage results of **for** loops are visualized in the same way as those for **while** loops. Samples of highlighted **for** loops are shown in the figure 3.17.

```

do {
doB();
} while ( !SA() );

do {
doB();
} while ( !SA() );

```

Figure 3.16: Do-while statements

```

for (int i=0; i<a.length; i++) {
doB();
}

for (int i=0; i<a.length; i++) {
doB();
}

for (int i=0; i<a.length; i++) {
doB();
}

```

Figure 3.17: For statements

3.8.3 COBOL

3.8.3.1 Basic statements

Basic statements like `DISPLAY`, `ACCEPT` or `COMPUTE` are highlighted with green and red for covered and not covered statements, respectively.

3.8.3.2 Conditional statements

3.8.3.2.1 If-then-else statements

The highlighting scheme for if-then-else statements is completely analogous to that of the corresponding statements in Java described above. Figure 3.18 shows the highlighting of an if-then statement in the COBOL programming language:

```

IF A < 5
MOVE A TO B
END-IF.

IF A < 5
MOVE A TO B
END-IF.

IF A < 5
MOVE A TO B
END-IF.

```

Figure 3.18: If-then statements in COBOL

3.8.3.2.2 Evaluate statement

The `EVALUATE` statement is the counterpart of the Java `switch` statement. Therefore analogous highlighting rules are applied for this statement. Figure 3.19 shows an example of the `EVALUATE` statement highlighting:

```

EVALUATE VALUE
  WHEN "0" PERFORM A
  WHEN "1" PERFORM B
  WHEN OTHER PERFORM C
END-EVALUATE.

```

Figure 3.19: Evaluate statement

3.8.3.3 Looping statements

3.8.3.3.1 Perform

The PERFORM statement is overloaded and can be used as a basic statement⁷ to jump to a particular paragraph (e.g. as in figure 3.19) as well as a loop statement. Figure 3.20 shows a while-equivalent statement and figure 3.21 a do-while-equivalent one. The highlighting rules for the while and do-while loops in Java apply accordingly.

```

PERFORM WITH TEST BEFORE UNTIL A = "10"
  DISPLAY A
  ADD 1 TO A
END-PERFORM.

```

Figure 3.20: Perform statement with test before

```

PERFORM WITH TEST AFTER UNTIL A = "10"
  DISPLAY A
  ADD 1 TO A
END-PERFORM.

```

Figure 3.21: Perform statement with test after

3.9 Preferences dialog

The *CodeCover* entry in the PREFERENCES dialog contains Eclipse-wide options for configuring the software. This dialog page is shown in figure 3.22.

TODO: ADOPT THIS FULLY TO STANDARD DIALOG FOR ANNOTATIONS. PICTURE MAY BE MADE WHEN IT'S CODED. The dialog provides a list of annotations to configure. For each of the four metrics there are three annotations: fully covered, partly covered, not covered. For each annotation the color can be selected using the standard mechanisms of Eclipse. A partly covered state is not produced by the default metrics for basic

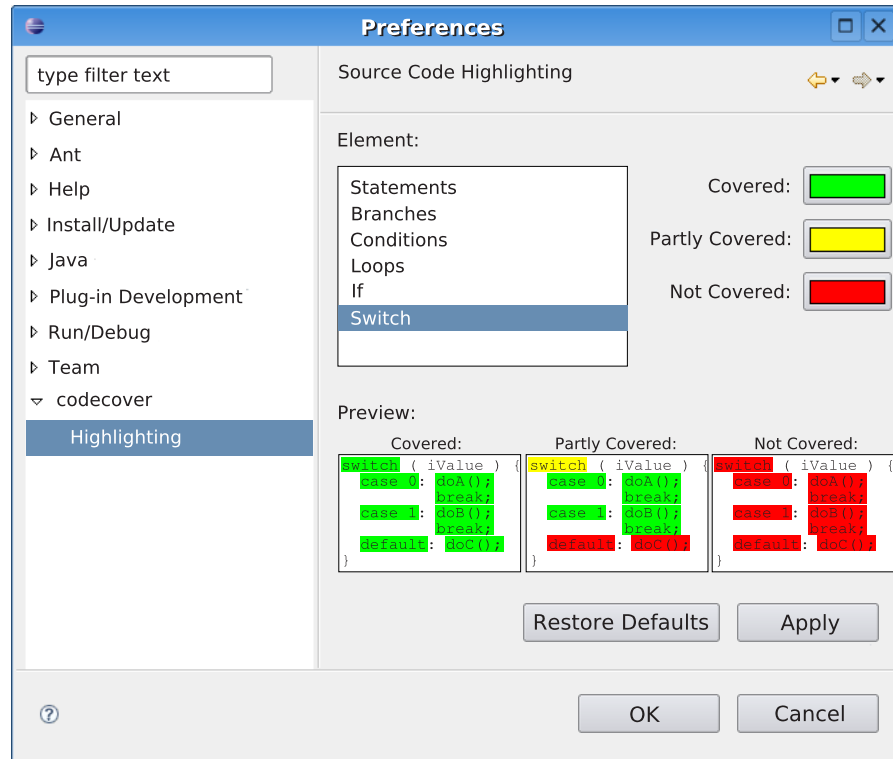


Figure 3.22: Preferences dialog

statements, basic boolean terms and branches. However as these may very well be produced by add on metrics there are also options to configure their color.

3.10 Project properties dialog

In the PROPERTIES dialog of a project a *CodeCover* entry is added. On this page, the user can activate codecover for the project. If codecover is activated the selection of coverage criteria which are to be measured for the project is enabled too. Figure 3.23 shows this dialog page.

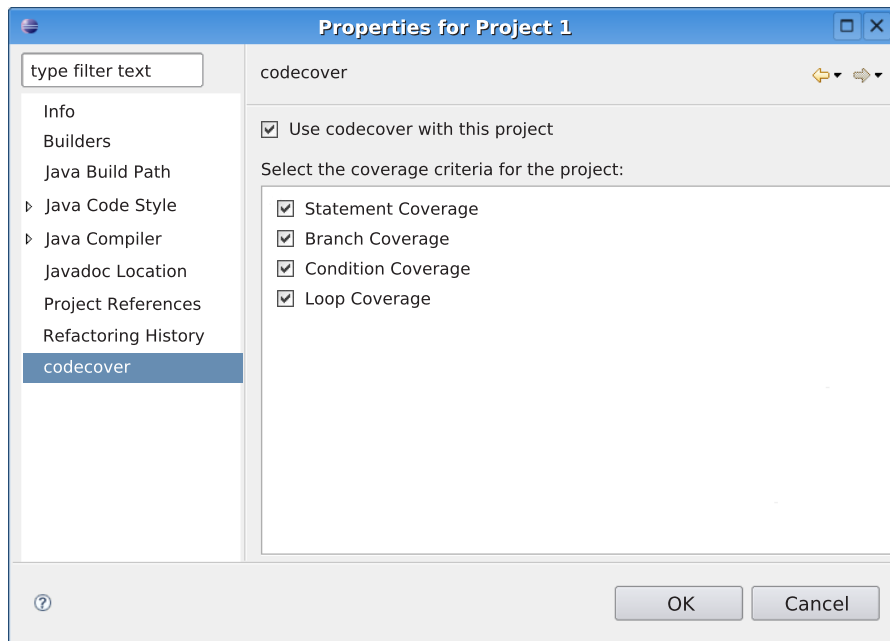


Figure 3.23: Project properties dialog

3.11 Correlation Matrix

3.11.1 Mathematic prelude

The CORRELATION MATRIX shows the correlation between test cases of a single test-session-container. Every test case contains a set of coverable items, which represents those parts of the code, that were covered during an execution of the instrumented system under test. Those sets are defined as follows:

$$C_T := \{x | x \in T \wedge x \in \text{CoverableItems} \wedge x \in \text{covered}\} \quad (1)$$

Correlation between two test cases T_1, T_2 is then defined as:

$$K_{U,V} := \frac{|U \cap V|}{|V|} \quad (2)$$

with $U = C_{T_1}$ and $V = C_{T_2}$.

Using this definition of correlation a *partially ordered set* R can be defined:

$$R = \{(U, V) \in C \times C : K_{U,V} = 1\} \quad (3)$$

In words this means, that two test cases T_1, T_2 are in relation R , if, and only if, T_1 contains all the coverable items T_2 does (or possibly more), which would make T_2 superfluous. This *partially ordered set* then allows to detect and establish subsumption chains, in which one test cases completely contains another and so forth.

Proof that R is a *partially ordered set* requires to proof that it possesses the following three attributes:

1. reflexivity

$$(U, U) \in R \Leftrightarrow \frac{|U \cap U|}{|U|} = 1 \Leftrightarrow \frac{|U|}{|U|} = 1 \quad (4)$$

2. antisymmetry

$$(U, V) \in R \Leftrightarrow \frac{|U \cap V|}{|V|} = 1 \Leftrightarrow V \subseteq U \quad (5)$$

$$(V, U) \in R \Leftrightarrow \frac{|V \cap U|}{|U|} = 1 \Leftrightarrow U \subseteq V \quad (6)$$

$$\Rightarrow V \subseteq U \wedge U \subseteq V \Rightarrow U = V \quad (7)$$

Die mathematische Beschreibung der Korrelation von Testfällen sollte im Kapitel 2 bei den anderen funktionalen Anforderungen stehen.

Die mathematische Komplexität dieser Anforderung erfordert auch eine mathematische Auseinandersetzung mit der Problemstellung. Diese Form ist sehr viel präziser als es in Prosa möglich wäre. Siehe Kriterium *Methodik [K-31]* auf Seite 93.

3. transitivity

$$(U, V) \in R \Leftrightarrow \frac{|U \cap V|}{|V|} = 1 \Leftrightarrow V \subseteq U \quad (8)$$

$$(V, W) \in R \Leftrightarrow \frac{|V \cap W|}{|W|} = 1 \Leftrightarrow W \subseteq V \quad (9)$$

$$\Rightarrow W \subseteq V \wedge V \subseteq U \Rightarrow W \subseteq U \Leftrightarrow \frac{|U \cap W|}{|W|} = 1 \Leftrightarrow (U, W) \in R \quad (10)$$

with T_1 , T_2 and T_3 being three test cases and $U = C_{T_1}$, $V = C_{T_2}$ and $W = C_{T_3}$.



Figure 3.24: Correlation View

3.11.2 Correlation View

Using the definitions from 3.11.1, the view in Eclipse is defined as shown in figure 3.24. A tree is located on the left of the view. This tree shows the subsumption chains of test cases defined in 3.11.1. All the children of a node in the tree are completely covered by the node itself.

The CORRELATION MATRIX is located on the right side of the view. It shows all the test cases, that were used in the calculation of the correlation. The meaning of the colors is

explained in the legend situated to the right of the matrix. The matrix is to be read from the left, e.g [test case left] covers [test case top] by [color value]%. A tooltip shown, when hovering above one entry of the matrix, contains the exact percentage of correlation, as well as the amount of total coverable items and shared coverable items.

The tool bar items represent following commands (from left to right):

- Export the currently displayed matrix data into a .csv file - This command exports the data of the matrix into a comma seperated file.
- Hide top-level tree items with no children - This command hides all the top level entries in the tree, which have no children, meaning they do not subsume any other test case;
- Choose and calculate correlation - This command selects the metric to be used in calculating the correlation, with the pull-down menu of the arrow and calculates the correlation with a push on the button.
- Show Legend - This command shows or hides the legend.
- Automatically calculate correlation - If this command is toggled on, the correlation is automatically recalculated, when the selection of test case is changed. It can be switched off to improve performance.

The test cases, that are to be used in the calculation, are selected using the Test sessions view. If the AUTOMATICALLY CALCULATE CORRELATION command is toggled on, every change in the selection causes the correlation view to update its contents. Since this can potentially be time-consuming, the AUTOMATICALLY CALCULATE CORRELATION command can be toggled off and the CHOOSE AND CALCULATE CORRELATION command can be used, after the desired selection was achieved.

3.12 Live Notification View

The LIVE NOTIFICATION VIEW is used to implement the Live Test Case Notification (2.11). Two textfields are used to enter the hostname and port. The name of the test case can be entered as well. Test cases can be started and stopped with two labeled buttons. The test session can be finished with another button. The log file can be retrieved with the “Download Coverage Log File” button. This also automatically stores the data of log file in the test session container.

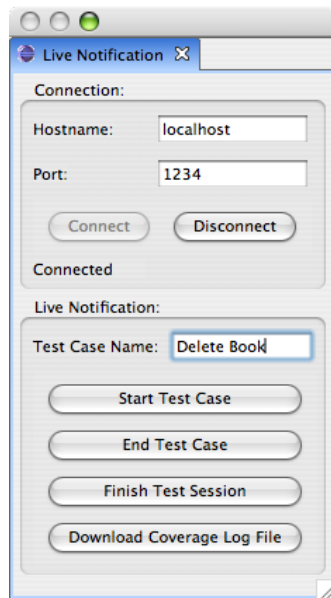


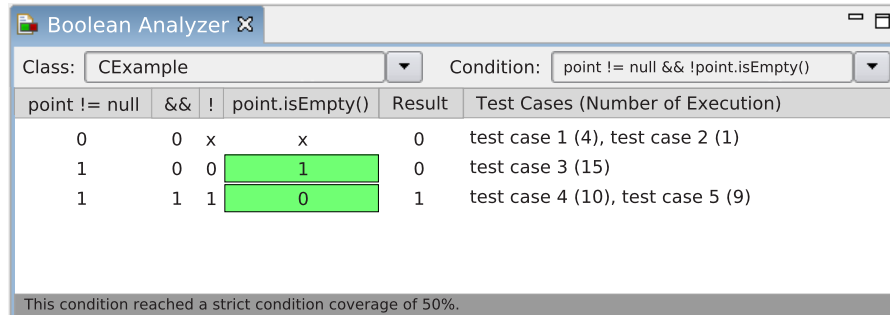
Figure 3.25: Live Notification

3.13 Boolean Analyzer

The `BOOLEAN ANALYZER` shows the boolean value of each basic boolean term, operator term and the root term according to evaluations of the condition which are recorded during the execution of the SUT. This data is presented in a table. The operators, operands and brackets define the columns and the evaluations are shown as rows in the table. In addition, a column for test cases is added. In that column one can see the test cases which covered the evaluation and the number of execution.

Two table values of a column may contain green background. This represents that the basic boolean term of that column is covered according to the strict condition coverage criterion. If a column of a basic boolean term does not contain any colored values then the term is not covered. Figure 3.26 shows the `BOOLEAN ANALYZER`.

There are two ways to select a condition in the `BOOLEAN ANALYZER`. First, there are combo-boxes for classes and conditions. The second way is to select the keyword of the condition in the source code, right-click, and select the "Analyze Term" item in the



The screenshot shows a window titled 'Boolean Analyzer'. It has a 'Class' dropdown set to 'CExample' and a 'Condition' dropdown set to 'point != null && !point.isEmpty()'. Below this is a table with columns: 'point != null', '&&', '!', 'point.isEmpty()', 'Result', and 'Test Cases (Number of Execution)'. The table contains three rows of data. The first row shows '0', '0', 'x', 'x', '0', and 'test case 1 (4), test case 2 (1)'. The second row shows '1', '0', '0', '1', '0', and 'test case 3 (15)'. The third row shows '1', '1', '1', '0', '1', and 'test case 4 (10), test case 5 (9)'. The cells containing '1' and '0' in the 'point.isEmpty()' column are highlighted in green. At the bottom of the window, a status bar indicates 'This condition reached a strict condition coverage of 50%'.

point != null	&&	!	point.isEmpty()	Result	Test Cases (Number of Execution)
0	0	x	x	0	test case 1 (4), test case 2 (1)
1	0	0	1	0	test case 3 (15)
1	1	1	0	1	test case 4 (10), test case 5 (9)

This condition reached a strict condition coverage of 50%.

Figure 3.26: Boolean Analyzer

context menu, which will automatically select the condition in the Boolean Analyzer.

3.14 Hot-Path

The plugin displays line-wise execution counters corresponding to the currently selected test cases. The measured number of executions of lines is added to the VERTICAL RULER of the java text editor via annotations with a color code. When the cursor hovers over such a Hot-Path annotation a tooltip with the execution counter is shown.

If no basic statement is found in a line, then no color is shown for that line in the ruler. Otherwise the color corresponding to the most often executed basic statement of that line is shown.

The color code is a mapping from the execution count of one line (e) and the highest execution count within a source file (e_{max}) to the color to mark that line. It must be encapsulated in a single method to make it easy to change in the source code. The mapping is a linear blending between the colors given in the users preferences. The exact mapping is: $color(e, e_{max}) = color_{max} * (e/e_{max}) + color_{null}(1 - e/e_{max})$.

4 Non-functional requirements

4.1 Technologies and development environment

The following software is used for development:

- Java 5 SE
- Eclipse 3.3.x
- the customer's COBOL-85 grammar ¹¹
- Subversion 1.3.0 on the server for configuration management
- ArgoUML 0.22 for use case diagrams in the specification[✓]
- BOUML 2.21.5 or compatible for UML diagrams

The following technologies are used for development:

- LaTeX as document format
- UTF-8 for encoding text
- XML as the intermediate format for reports
- Java and COBOL-85 example programs

Die genaue Festlegung der Entwicklungsumgebung ist wichtig, damit diese im Wartungsfall nachgestellt werden kann. Prinzipiell kann diese Liste auch nur im Projektplan gepflegt werden, damit keine unnötige Redundanz in der Projektdokumentation entsteht. Siehe Kriterium *Prozessfreiheit* [K-19] auf Seite 73.

4.2 Requirements to the working environment

4.2.1 Software requirements

The following programs are required to use the software:

- Java version 5 or later
- Eclipse 3.3.x for all GUI functions
- PDF[✓] viewer for documentation which supports at least PDF 1.4
- JUnit for automatic test case[✓] recognition

¹¹bruessel.informatik.uni-stuttgart.de:/home/export/bruessel/projects/stupro06/grammars/cobol.jj

- for COBOL support a COBOL-85 preprocessor to prepare code for instrumentation[✓] and a compiler to compile the instrumented code

4.2.2 Hardware requirements

To support a wide installation base moderate hardware should be enough for using the software. Exact minimum requirements must be determined based on the final application.

The minimum hardware required is:

- 512 MiB¹² of RAM
- a CPU as powerful as an AMD Athlon with 1 GHz clock rate
- 100 MiB of free hard disk space for installation with Eclipse already installed, to store working data and for some session containers[✓]

The following hardware is recommended:

- 1 GiB of RAM
- a CPU as powerful as single core AMD Athlon with 2 GHz clock rate
- 60 MiB/s read and write sequential transfer rate measured at file system level
- 10 GiB of free hard disk space to store working data and some coverage results

4.3 Quantity requirements

Quantities that have no defined maximum are only limited by the resources of the PC the software runs on.

¹²1 MiB = 2²⁰ Bytes

4.3.1 Program examples

4.3.1.1 Small program

Fred v1.3.5 RC2¹³ is a small program. All functions of the software that don't work with it are completely useless.

4.3.1.2 Medium program

Tomcat 5.5.20¹⁴ is a medium sized program. All functions of the software must work with it.

4.3.1.3 Large program

Eclipse SDK 3.1.2¹⁵ is a large program. Running functions on the large program is sufficient to show that they work for any project[✓] that must be supported.

4.3.2 Timestamp

Timestamps have a resolution of one minute or finer. The software must use a state of the art representation of timestamps to support a sufficiently wide time period.

4.3.3 Text value

Text values are of arbitrary length and may contain any Unicode characters.

4.3.4 Test case

The name and the comment of a test case[✓] are text values. The start time is a time stamp.

Die Aussage „arbitrary length“ ist nicht präzise und kann keinesfalls in einem Test überprüft werden. Vielmehr müsste für jedes Text-Attribut einer fachlichen Entität die Minimal- und Maximallänge angegeben werden. Sollte diese Festlegung verallgemeinert werden, wäre sicher eine Unterscheidung in Namens- und Kommentarlänge angebracht. Siehe Kriterium *Verifizierbarkeit* [K-27] auf Seite 87.

Die Angabe der Quellen ist sehr sinnvoll, wenn man sich diese Dokumente anschauen möchte. Für eine bessere Übersicht wäre auch eine Auslagerung in ein gesondertes Quellenverzeichnis denkbar.

¹³<http://sourceforge.net/projects/fred>

¹⁴<http://tomcat.apache.org/download-55.cgi>

¹⁵<http://archive.eclipse.org/eclipse/downloads/drops/R-3.1.2-200601181600/index.php>

4.3.5 Test session

An arbitrary number of test sessions[✓] must be supported.

The name and the comment of a test session[✓] are text values. The start time is a time stamp.

4.3.6 Code Base

An arbitrary number of code bases[✓] must be supported. The date and time of the first instrumentation is a time stamp.

4.4 Performance requirements

All performance requirements must be met on the recommended hardware.

4.4.1 Batch processing

Building and instrumenting a large program (4.3.1.3) must be possible within 10 hours.

4.4.2 Eclipse plug-in

The following constraints must hold true for a medium program (4.3.1.2) and should hold true for a large program (4.3.1.3).

The Eclipse plug-in may not slow down Eclipse significantly.

Additionally the following response times must be met. They don't apply to the first call of a function. During the first call initialization routines may add a significant delay.

For simple GUI interaction 0.1 s response time is enough while 0.5 s is too slow. Selecting instrumentable items is such a simple GUI interaction.

For interaction resulting in complicated rebuilds of the UI components the code highlighting 5 s response time is enough while 30 s is too slow. During this time the Eclipse

Die Wortgruppe „may not slow down [...] significantly“ lässt zwar eine unpräzise Anforderung vermuten, diese wird aber in der Folgebeschreibung konkretisiert. So ist es sogar möglich, einen Testfall aus dieser Anforderung zu entwickeln. Die Angabe der Beispiel-Benutzerinteraktivität ist eine zusätzliche Hilfe. Siehe Kriterium *Verifizierbarkeit* [K-27] auf Seite 87.

UI may not respond. An example for such interaction is changing the source code annotations while they are displayed.

For loading and saving files as well as processing tasks like building the correlation matrix and generating a report there are no performance requirements. These tasks may not block the Eclipse UI.

4.5 Availability

There are no special availability requirements.

4.6 Security

There are no special security requirements.

4.7 Robustness and failure behavior

File types are detected using a syntax check for plain text files and at least 64 bit long magic numbers for binary files. That syntax check only has to detect wrong file types.

The software may show arbitrary behavior when it runs on broken hardware.

4.8 Usability

Any string displayed by the Eclipse plug-in is localizable. The default language is English.

A developer[✓] can install the software easily. The installation procedure may only require unzip and Java. The software must be installable within 5 minutes by a developer after he has found the installation instructions and downloaded the necessary files.

All colors used for highlighting can be changed by the user.

Internal procedures must be invisible to the user as long as he doesn't want to change their behavior. Where applicable sensible defaults must be preset.

An English user's manual is required. Online help is not required.

Selbst wenn es zu einer Kategorie keine nichtfunktionalen Anforderungen gibt, wird ein Kapitel mit einem entsprechenden Hinweis angefügt. Würde das Kapitel weggelassen, könnte sich ein Leser nicht sicher sein, ob das Kapitel vergessen wurde, oder ob es keine Anforderungen diesbezüglich gab. Siehe Kriterium *Vollständigkeit der inhaltlichen Aspekte [K-21]* auf Seite 75.

In Kapitel 4.8 sind mehrere nichtfunktionale Anforderungen enthalten. Diese können nicht eindeutig durch die Gliederungsnummer identifiziert werden, wodurch die Nachverfolgbarkeit erschwert wird.

Man könnte die Gliederung an dieser Stelle weiter verfeinern, wodurch jede Anforderung wieder eine eindeutige Kapitelnummer erhält. Ein anderer Ansatz, den Lauesen (2007) vorschlägt, verwendet eindeutige Identifizierungsnummern für jede elementare Anforderung. Siehe Kriterium *Nachverfolgbarkeit [K-24]* auf Seite 84.

The user interface must be intuitive to the point that on line help is not needed. Eclipse User Interface Guidelines version 2.1¹⁶ must be followed for the Eclipse plug-in. Additionally per default the user is asked in a confirm dialog before a file with valuable user data is deleted or overwritten.

4.9 Portability

The software must run on all platforms Eclipse 3.2.x runs on.

The delivered instrumentation[✓] procedure must be compatible with Java 2 version 1.4.x and Java 5 as well as preprocessed COBOL-85. Other languages must be implementable without changing the session container's[✓] format.

4.10 Maintainability

The source code follows a style guide based on Sun's Code Conventions¹⁷. The style guide is described in a separate document.

All technical documents made specifically for creating and verifying the software are released to the public.

4.11 Extensibility

The software is written to be highly extensible and documented on a level easy to understand for their target audience. The documentation of the Eclipse plug-in is written for the Eclipse user, the documentation of the batch interface is written for the shell user (see Actors 2.2) and the documentation to extending the Software is written for maintenance engineers[✓]. Tutorials for maintenance engineers will show how to extend the software to other programming languages than COBOL and Java, how to add the collecting of metrics during the instrumentation[✓] and how to implement further coverage criteria. The tutorials may assume good knowledge of Java, Grammars, JavaCC and the languages that must be supported by the changed software.

Hier wird die Einhaltung einer bekannten Richtlinie gefordert. Das macht die Überprüfung objektiv.

Für die Wartbarkeit wird auf externe Richtlinien verwiesen. Damit ist diese nichtfunktionale Anforderung hier prägnant und überprüfbar formuliert. Vereinbarungen zum Wartungsprozess sind nicht in der Spezifikation sondern im Projektplan festgehalten.

¹⁶<http://www.eclipse.org/articles/Article-UI-Guidelines/Index.html>

¹⁷<http://java.sun.com/docs/codeconv/>

Reports can be customized using templates, which can be selected in Eclipse using a wizard. It must be easy to add new report formats like PDF and DocBook XML, which can be transformed into many formats.

To ease external analysis and report generation all data of a test session, except for the boolean values sampled in conditional statements, must be available to easily implement an export to XML. The design team must decide if the session container already has such a format, or if report generation already contains it as an intermediate step.

Other coverage criteria should be easy to add for maintenance engineers as long as they don't depend on execution order. No change to the instrumentation, test case management and highlighting may be necessary to implement another condition coverage criterion.

It must be easy to add further analysis of a session container. All evaluation results of boolean expressions and counters must be easily accessible.

Based on the TestCaseNotifier class (see 2.1), functions to define test cases must be added: a live mode with a graphical dialog which allows the user to start and stop test cases during the SUT execution and automatic test case recognition of JUnit test cases.

Die Formulierung „easy to add“ ist ungenau und kann nur subjektiv überprüft werden. Eine Angabe des möglichen Aufwandes könnte diese Anforderung präzisieren. Siehe Kriterium *Verifizierbarkeit* [K-27] auf Seite 87.

Zu den nichtfunktionalen Anforderungen zählen darüber hinaus:

- Protokollierung
- Installation
- Wartbarkeit
- Gesetzliche Bestimmungen

Siehe Kriterium *Vollständigkeit der inhaltlichen Aspekte* [K-21] auf Seite 75.

List of Figures

1.1	Work flow of the software	8
2.1	Actors	12
2.2	Key use cases	15
2.3	Use cases related to measuring coverage	16
2.4	Use cases related to showing coverage	28
2.5	Use cases related to administrating test sessions	31
3.1	Package selection	80
3.2	Instrumentation dialog	81
3.3	Coverage button	82
3.4	Coverage view	82
3.5	Test Sessions view	83
3.6	Test case properties	85
3.7	Import Test Session Container	86
3.8	Import Coverage Log	87
3.9	Export Test Session	88
3.10	Report dialog	89
3.11	If-then statements	90
3.12	If-then-else statements	91
3.13	Nested if-then-else statements	91
3.14	Switch-statement with some cases and a default section	92
3.15	Highlighting of while loops	92
3.16	Do-while statements	93
3.17	For statements	93
3.18	If-then statements in COBOL	93
3.19	Evaluate statement	94
3.20	Perform statement with test before	94
3.21	Perform statement with test after	94
3.22	Preferences dialog	95
3.23	Project properties dialog	96
3.24	Correlation View	98
3.25	Live Notification	100
3.26	Boolean Analyzer	101

Glossary

basic statement

is a statement, that is not a looping statement[↗] or a conditional statement[↗]. For Java the statements `return`, `throw`, `assert` are also excluded.

branch coverage

(synonym: decision coverage) is a coverage criterion[↗]. A coverable item is a branch of a conditional statement[↗]. For branch coverage, a coverable item is covered, if it is entered at least once.

code base

contains all the uninstrumented code files[↗] of a specific version of the SUT[↗]. A code base has a date and time of the first instrumentation[↗]. If a code base is instrumented from Eclipse, it has a relation to an Eclipse project.

code coverage

has two meanings:

1. is a measurement needed for a glass box test[↗]. There are different coverage criteria, each defining the coverable items[↗] and how they are covered.
2. depends on a concrete coverage criterion[↗] and is defined—only considering the instrumented part of the SUT—as the quotient of the covered coverable items and the total number of coverable items.

code file

is a file containing the whole or a part of the source code of the SUT. For example a `*.java` file in Java.

condition coverage

is a coverage criterion[↗]. Condition coverage defines the coverable items[↗] as basic boolean terms[↗] used in statements[↗] which require a boolean expression that affects the control flow. There are different definitions of when such a basic boolean term is considered as covered—e.g. strict condition coverage[↗].

Die Begriffserklärung beinhaltet Anforderungen an eine „code base“. Anforderungen sollten im Begriffslexikon nicht vorkommen. Beziehungen zwischen den Begriffen können auch in einem gesonderten Begriffsmodell veranschaulicht werden (Ludewig u. Lichter, 2007, Seite 348). Siehe Kriterium *Begriffsdefinitionen [K-15]* auf Seite 68.

conditional statement

is a statement[↗] of a specific programming language. Conditional statements are statements creating branches in the control flow, e.g. **if** or **switch** in Java. It does not matter, if a *particular* usage of a conditional statement creates a branch (e.g. **if** (**true**)) or if branches of a *particular* usage are equal (e.g. **if** (**a**) { **}**). It is a conditional statement creating two branches nonetheless, because an **if** usually creates two branches.

On the other hand, though the result of some operators depends on a decision, an operator itself creates no branch in the control flow. An example for such an operator would be the **A ? B : C** operator, of which the result is determined by the value of A. Accounted by itself, such an operator only influences data, not the control flow. Therefore, it is no conditional statement.

coverable item

is the smallest unit that can be covered by a coverage criterion, e.g. a **then** branch in branch coverage.

coverage criterion

defines the coverable item[↗] and under which condition they are covered. Some coverage criteria are:

- statement coverage[↗]
- branch coverage[↗]
- condition coverage[↗]
- loop coverage[↗]

coverage log

is a container for raw result data of a coverage run. It contains e.g. counters for all basic statements. This file must be processed afterwards to produce test sessions[↗] and test cases[↗].

developer

is a person who is able to write and compile programs in at least one programming language and can understand well documented programs. He is also experienced in both using his computer, especially with file system interaction, web browsing,

extracting archive files, applying patches and editing plain text.

DocBook XML

is a XML based markup language for technical documentation.

entry point

is the item, that is used to start the SUT[✓]. For Java, it is a class file containing the `main` method. For COBOL, it is the single code file[✓].

HTML

(abbreviation for: Hyper Text Markup Language) is the predominant markup language for the creation of web pages.

instrumentable item

is an item of the SUT[✓]. An instrumentable item is a package, containing other instrumentable items, or a code file.

instrumentation

is the process of adding extra code elements to a code file[✓] in order to get information about the control flow of the running SUT. Instrumentation is used to measure the code coverage[✓] of the code file.

loop coverage

is a coverage criterion. The loop coverage defines several coverable items[✓] for each looping statement[✓]:

- loop body is not entered
- loop body is entered once, but not repeated
- loop body is repeated more than one time

Looping statements like do-while cannot be bypassed and have only two possible coverable items.

maintenance engineer

is a developer[✓] who changes software. He understands technical English. He is able to work out technical details himself, if he is pointed to good documentation.

MAST

(abbreviation for: More Abstract Syntax Tree) The MAST is a model of the source code containing only the elements of the source code which are necessary to calculate coverage criteria[✓] e.g. statements[✓], branches or boolean expressions which have an affect on control flow.

A MAST always refers to a specific code base[✓].

PDF

(abbreviation for: Portable Document Format) is an file format created and controlled by Adobe Systems, for representing two-dimensional documents in a device independent and resolution independent fixed-layout document format.

project

is an Eclipse project that appears e.g. in the Package Explorer of Eclipse.

session container

is a file in the file system. It contains:

- a code base[✓]
- a MAST[✓] of this code base
- a number of test sessions[✓] (≥ 0)

specification

describes all functional and non-functional requirements the software has to fulfill.

statement

is an element in a code file[✓] that is the result of the statement production of the grammar of the corresponding programming language.

statement coverage

is a coverage criterion[✓]. Statement coverage defines the coverable items[✓] as basic statements[✓]. A coverable item is covered, if the basic statement is successfully executed.

strict condition coverage

is a kind of condition coverage[✓]. Strict condition coverage defines a basic boolean

term as covered, if it is evaluated to both true and false and the change from true to false (or false to true) changes the result of the whole condition while every other basic boolean term of the condition remains constant or is not evaluated.

SUT

(abbreviation for: system under test) The system tested with the software.

test case

1. is the description of the input for a test with its expected output according to the specification[^].
2. is an element of a test session[^] containing a part or all of the code coverage[^] results for code files depending on a set of coverage criteria. Additional information which are stored with a test case are:
 - a name
 - date and time of measurement
 - a comment
 - the related test session

If the test case is related to a JUnit test case or test method extra information are needed:

- the names of the test methods of the JUnit test case
- whether the test methods of the JUnit test case failed or not
- if a test method failed, which failure respectively error was the reason

test session

is the result of a coverage measurement of the SUT by the software. It has:

- a name
- a date and a time of measurement
- a comment

and contains:

- a number of test cases[^] (≥ 0)
- the measurement results

- calculated coverage by instrumentable item✓
- a reference to a code base✓
- possibly a reference to a related Eclipse project✓

Literaturverzeichnis

- [IEE 1990] IEEE standard glossary of software engineering terminology. In: *IEEE Std 610.12-1990* (1990)
- [DIN 1997] DIN 69905 – Projektwirtschaft, Projektabwicklung, Begriffe. (1997). – Deutsches Institut für Normung, Normenausschuß Qualitätsmanagement, Statistik und Zertifizierungsgrundlagen (NQSZ)
- [IEE 1998a] IEEE guide for developing system requirements specification. In: *IEEE Std 1233, 1998 Edition* (1998)
- [IEE 1998b] IEEE Recommended Practice for Software Requirements Specifications. In: *IEEE Std 830-1998 (Revision of IEEE Std 830-1993)* (1998)
- [IEE 2005] IEEE Standard for Software Verification and Validation. In: *IEEE Std 1012-2004 (Revision of IEEE Std 1012-1998)* (2005), S. 1–110
- [Drappa] DRAPPA, Anke: *Checkliste zum Pflichtenheft*. <http://www.iste.uni-stuttgart.de/se/links/checklists/download/Pflichtenheft-1.html>. – Stand 04.06.2008 22:00
- [Faulk u. a. 1997] FAULK, S ; THAYER, R. ; DORFMAN, M.: Software Requirements: A Tutorial. In: *Software Requirements Engineering 2nd Edition* (1997). – ISSN 1550–6002
- [Firesmith 2003] FIRESMITH, Donald G.: Specifying Good Requirements. In: *Journal of Object Technology* (2003), Juli/August, S. 77 ff.
- [Hussain u. a. 2007] HUSSAIN, Ishrar ; ORMANDJIEVA, Olga ; KOSSEIM, Leila: Automatic Quality Assessment of SRS Text by Means of a Decision-Tree-Based Text Classifier. In: *Seventh International Conference on Quality Software* (2007), Oktober, S. 209 ff. <http://dx.doi.org/10.1109/QSIC.2007.4385497>. – DOI 10.1109/QSIC.2007.4385497. – ISSN 1550–6002
- [Klüver 2008] KLÜVER, Wolfgang: *Software Engineering: Gliederungen für Pflichtenhefte*. Webseite. <http://www.fh-augsburg.de/informatik/vorlesungen/se1t/script/definition/pfheft.html>. Version:2008. – Stand 08.06.2008 18:45

- [Lauesen 2007] LAUESEN, Søren: *Requirements specification for Electronic Health Record system*. The IT University of Copenhagen, 2007
- [Ludewig u. Lichter 2007] LUDEWIG, Jochen ; LICHTER, Horst: *Software Engineering - Grundlagen, Menschen, Prozesse, Techniken*. 1. Auflage. dpunkt.verlag GmbH, Heidelberg, 2007
- [Pressman 2005] PRESSMAN, Roger S.: *Software Engineering: A Practitioner's Approach*. 6th edition. International Edition. McGraw-Hill, 2005 <http://www.mhhe.com/engcs/pressman/>. – ISBN 0-07-365578-3
- [Robertson u. Robertson 2007] ROBERTSON, James ; ROBERTSON, Suzanne: *Volere Requirements Specification Template*. 13th edition, 2007
- [Robertson u. Robertson 1999] ROBERTSON, Suzanne ; ROBERTSON, James: *Mastering the Requirements Process*. Addison-Wesley Professional, 1999. – ISBN 0201360462
- [Wikipedia 2008a] WIKIPEDIA: *Anforderung (Informatik)* – *Wikipedia, die freie Enzyklopädie*. http://de.wikipedia.org/w/index.php?title=Anforderung_%28Informatik%29&oldid=46512861. Version: 2008. – Stand 27.05.2008 23:08
- [Wikipedia 2008b] WIKIPEDIA: *Lastenheft* – *Wikipedia, die freie Enzyklopädie*. <http://de.wikipedia.org/w/index.php?title=Lastenheft&oldid=46565676>. Version: 2008. – Stand 27.05.2008 23:00
- [Wikipedia 2008c] WIKIPEDIA: *Pflichtenheft* – *Wikipedia, die freie Enzyklopädie*. <http://de.wikipedia.org/w/index.php?title=Pflichtenheft&oldid=46543318>. Version: 2008. – Stand 06.08.2008 16:20
- [Wikipedia 2008d] WIKIPEDIA: *Spezifikation* – *Wikipedia, die freie Enzyklopädie*. <http://de.wikipedia.org/w/index.php?title=Spezifikation&oldid=40884861>. Version: 2008. – Stand 27.05.2008 22:45