

Eine Software zur automatisierten Erstellung von IEEE829-Testdokumentationen

Thorsten Liese

16. April 2003

Diplomarbeit an der
Technischen Universität Darmstadt
Fachgebiet Theoretische Informatik
Kryptographie und Computeralgebra
Prof. Dr. Johannes Buchmann
Dipl. Inf. Alexander Wiesmaier

Thorsten Liese, Rhönring 75, 64289 Darmstadt, thorsten.liese@dreicsystems.de

Hiermit versichere ich, die vorliegende Diplomarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt 17. April 2003

Thorsten Liese

Die in diesem Dokument erwähnten Soft- und Hardwarebezeichnungen sind in den meisten Fällen auch eingetragene Warenzeichen und unterliegen als solche den gesetzlichen Bestimmungen.

Inhaltsverzeichnis

1	Einleitung	7
1.1	Motivation	7
1.2	Heranführung	8
1.3	Aufbau des Dokuments	11
2	Der IEEE829-Standard	12
2.1	Warum IEEE829?	12
2.2	Einleitung	13
2.3	Die Dokumente	14
2.3.1	Testplan	14
2.3.2	Testdesignspezifikation	16
2.3.3	Testfallspezifikation	17
2.3.4	Testablaufspezifikation	18
2.3.5	Testgegenstandsauslieferungsbericht	19
2.3.6	Testprotokoll	20
2.3.7	Testvorfallbericht	20
2.3.8	Testergebnisbericht	21
2.4	Zusammenfassung	22
2.5	Erweiterungen	24
2.5.1	Testgegenstände	24
2.5.2	Parametrisierte Testfälle	25

<i>INHALTSVERZEICHNIS</i>	2
3 Formen der Automatisierung	26
3.1 Einführung	26
3.2 Definition der Dokumente	27
3.2.1 XML und DTD	27
3.2.2 Testplan	29
3.2.3 Testdesignspezifikation	30
3.2.4 Testfallspezifikation	31
3.2.5 Testablaufspezifikation	31
3.2.6 Testgegenstandsauslieferungsbericht	32
3.2.7 Testprotokoll	33
3.2.8 Testvorfallbericht	34
3.2.9 Testergebnisbericht	34
3.2.10 Erweiterung: Testgegenstand	35
3.3 Automatisierte Dokumenterstellung	36
3.3.1 Allgemeine Dokumentstruktur	36
3.3.2 DTD	38
4 Die Software	40
4.1 Einführung	40
4.2 Modellierung	42
4.2.1 Verzeichnisstruktur	42
4.2.2 Package <code>ieee829.app.core</code>	43
4.2.3 Package <code>ieee829.app.editors</code>	46
4.3 Dokumentbeschreibungsdateien	60
4.3.1 Übergreifende Beschreibungsdatei	62
4.3.2 Testplan-Beschreibungsdatei	63
4.3.3 Testdesignspezifikation-Beschreibungsdatei	64
4.3.4 Testfallspezifikation-Beschreibungsdatei	65
4.3.5 Testablaufspezifikation-Beschreibungsdatei	65
4.3.6 Testgegenstandsauslieferungsbericht-Beschreibungsdatei	66

<i>INHALTSVERZEICHNIS</i>	3
4.3.7 Testprotokoll-Beschreibungsdatei	67
4.3.8 Testvorfallbericht-Beschreibungsdatei	67
4.3.9 Testergebnisbericht-Beschreibungsdatei	68
4.3.10 Testgegenstand-Beschreibungsdatei als Erweiterung . .	69
4.3.11 Testgegenstand-Beschreibungsdatei für den Testplan . .	69
4.3.12 Beschreibungsdatei für zu testende Funktionen	70
4.3.13 Beschreibungsdatei für nicht zu testende Funktionen . .	70
4.3.14 Meilenstein-Beschreibungsdatei	71
4.3.15 Ressourcen-Beschreibungsdatei	71
4.3.16 Testaktivitäten-Beschreibungsdatei	72
4.3.17 Protokolleintrag-Beschreibungsdatei	72
4.3.18 Testgegenstandseigenschaft-Beschreibungsdatei	73
4.4 Speicherformat der Software	73
5 Erweiterungen	74
5.1 Einleitung	74
5.2 Dokumenten-Export mit XSL	74
5.3 Testplanung und -durchführung	77
5.4 Beliebige Dokumenttypen	78
5.5 Fazit	78

Abbildungsverzeichnis

2.1	Abhängigkeiten der Dokumente	23
4.1	Screenshot der Software	41
4.2	Basisklassen	43
4.3	Package <code>ieee829.app.core.control</code>	44
4.4	Package <code>ieee829.app.core.model</code>	45
4.5	Package <code>ieee829.app.core.view</code>	46
4.6	Klasse <code>ieee829.app.editors.Editor</code>	47
4.7	Package <code>ieee829.app.editors</code>	49
4.8	Dokumentbeschreibungsdateien	61

Listings

3.1	formattedtext.dtd	28
3.2	testplan.dtd	29
3.3	testdesignspecification.dtd	30
3.4	testcasespecification.dtd	31
3.5	testprocedurespecification.dtd	31
3.6	testitemtransmittalreport.dtd	32
3.7	testlog.dtd	33
3.8	testincidentreport.dtd	34
3.9	testsummaryreport.dtd	34
3.10	testitem.dtd	35
3.11	autodoc.dtd	38
4.1	Editor.java	50
4.2	BasicEditor.java	55
4.3	documents.xml	62
4.4	testplan.xml	63
4.5	testdesign.xml	64
4.6	testcase.xml	65
4.7	testprocedure.xml	65
4.8	testitemtransmittal.xml	66
4.9	testlog.xml	67
4.10	testincident.xml	67
4.11	testsummary.xml	68

<i>LISTINGS</i>	6
4.12 testitem.xml	69
4.13 testplantestitem.xml	69
4.14 featuretest.xml	70
4.15 featurenottest.xml	70
4.16 milestone.xml	71
4.17 resource.xml	71
4.18 testingtask.xml	72
4.19 logentry.xml	72
4.20 itemfeature.xml	73
4.21 project.dtd	73
5.1 testplan.xsl	74

Kapitel 1

Einleitung

1.1 Motivation

Die Entwicklung komplexer Software ist kosten- und zeitaufwändig. Um Entwicklungszeiten und Kosten kalkulierbar zu machen, kommen im Softwareentwicklungsprozess eine Reihe standardisierter Verfahren und Instrumente¹ zum Einsatz, die auch zu einer höheren Qualität der Software beitragen. Eines der wichtigsten Instrumente der Qualitätssicherung ist das Testen.

Da Softwareprojekte meist unter massivem Zeitdruck durchgeführt werden müssen, werden Softwaretests häufig vernachlässigt bzw. nur unzureichend und unstrukturiert am Ende der Implementierungsphase durchgeführt. Wünschenswert wäre aber, die Testaktivitäten von Beginn an klar zu definieren und in den Entwicklungsprozess einzubeziehen. Dies setzt eine sorgfältige Planung und Spezifikation der Testaktivitäten voraus. Ferner sollten Art, Anzahl und Eingabewerte der einzelnen durchzuführenden Tests genau festgelegt werden.

Grundlage für dies ist die Erstellung einer umfassenden Testdokumentation. Sie enthält neben der Planung der Testaktivitäten und der Spezifikation der Testfälle auch die Ergebnisse der durchgeführten Tests. Form und Inhalt einer solchen Testdokumentation werden in dem IEEE-Standard² 829 [IEEE829] definiert. Durch diese Standardisierung ist die Dokumentation nicht nur Teil der internen Organisation, sondern sie kann auch als Referenz nach außen dienen³. Eine vollständige Testdokumentation mit Ergebnisanalyse und -auswertung ist außerdem ein hervorragendes Qualitätsmerkmal und kann als Gütesiegel der erstellten Software betrachtet werden.

¹Siehe hierzu: Softwarekrise [SWKRISE] und Software Engineering [SE].

²IEEE: *Institute of Electrical and Electronics Engineers*

³beispielsweise als Verständigungsgrundlage zwischen Entwickler und Kunde

Das Erstellen der Testdokumentation kann jedoch viel Zeit in Anspruch nehmen. Daher wäre es schön, wenn die Erstellung durch eine Software automatisiert werden könnte. Gerade im Bereich der Spezifikation der Testfälle und der Eingabeparameter sowie der Ergebniserfassung könnte der Erstellungsprozess dadurch erheblich beschleunigt werden. Die Möglichkeiten einer solchen Software zur automatisierten Erstellung von IEEE829-Testdokumentationen sollen hier untersucht werden und sind Thema dieser Diplomarbeit.

1.2 Heranführung

Bei den in dieser Diplomarbeit vorgestellten Lösungsansätzen wurde sehr großer Wert auf die Praxistauglichkeit gelegt. Wenn man sich fragt, warum die Erstellung von Testdokumentationen oft vernachlässigt wird, dann liegt dies meist an den folgenden praktischen Problemen:

- Zunächst ist unklar, auf welcher Software-Basis die Dokumentation überhaupt erstellt werden soll. Da die Dokumente auch extern präsentiert werden müssen, wird man auf etablierte und bekannte Textverarbeitungsprogramme zurückgreifen müssen.
- Da eine umfassende Testdokumentation aus vielen Einzeldokumenten besteht, die teilweise in mehrfachen Beziehungen zueinander stehen, wird man schnell feststellen, dass die üblichen Textverarbeitungssysteme ungeeignet sind. Es fehlt die intelligente Logik, die die Konsistenz der Gesamtdokumentation wahrt.
- Die Spezifikation von Testfällen gestaltet sich schwierig. Häufig müssen Systeme mit einer Vielzahl unterschiedlicher Eingabeparameter getestet werden, die in vielen Kombinationsmöglichkeiten zu berücksichtigen sind. Eine manuelle Erstellung der Spezifikationsdokumente für alle gewünschten Einzelfälle ist sehr aufwändig.
- Die Erfassung der Testergebnisse ist ein ungelöstes Problem. Dass der Softwaretechniker zur Erfassung der Ergebnisse seiner Testdurchläufe beispielsweise ein Word-Dokument erstellen oder editieren muss, ist unpraktikabel. Ein speziell zugeschnittenes Eingabeformular wäre wünschenswert.
- Bei der Vielzahl der durchgeführten oder durchzuführenden Testläufe ist es sehr schwer möglich, den Überblick über die Testaktivitäten und Dokumente zu behalten. Auch hier fehlt eine intelligente Software, die Testaktivitäten und Dokumentation miteinander verbindet.

Es ergibt sich daraus, dass die Erstellung einer vollständigen Testdokumentation aus praktischen Gründen sehr komplex und zeitaufwändig ist. Ich selbst habe diese Erfahrungen gesammelt, als ich mit der Erstellung einer Testdokumentation für ein Softwareprojekt beauftragt war. Ich denke, dass eine Testdokumentation häufig nicht erstellt wird, weil der Aufwand aus den genannten Gründen weit höher ist als der Nutzen. Gerade aber in der kommerziellen Software-Entwicklung muss man davon ausgehen, dass ein Auftraggeber früher oder später eine der folgenden Fragen stellt:

- Wie können Sie uns die Qualität und Zuverlässigkeit Ihrer Software dokumentieren?
- Wie, wann und wie oft testen Sie Ihre Software und wie erfassen Sie die Testergebnisse?
- Können Sie uns die Ergebnisse Ihrer Software-Tests präsentieren?

Insbesondere wenn ein Auftraggeber einen eigenen Qualitätssicherer besitzt, was zunehmend der Fall ist, wird man nicht umhin kommen, die Testaktivitäten hinreichend nach außen zu dokumentieren. Es sollte daher eine Lösung gefunden werden, die es erlaubt, eine Testdokumentation für beliebige Software-Projekte mit geringem Aufwand zu erstellen. Dies soll mit Hilfe der in diesem Dokument vorgestellten Lösung erreicht werden. Schwerpunkt ist dabei die Erstellung der Dokumentation im Hinblick auf deren Verwendung als umfassende Referenz nach außen - also beispielsweise zur Weitergabe an einen Auftraggeber - und nach innen, also als Grundlage der internen Projektorganisation. Die entwickelte Software kann zwar auch die internen Testaktivitäten unterstützen⁴, sie kann aber professionelle Testwerkzeuge zur Zeit- und Ressourcenplanung und zur Testdurchführung nicht ersetzen. In erster Linie soll sie der Erstellung der Dokumentation dienen, nicht der Planung und Durchführung der Testaktivitäten.

Aus dieser Motivation heraus liegen dieser Arbeit folgende Aufgabenstellungen und Zielsetzungen zu Grunde:

1. Die erstellte Dokumentation soll dem anerkannten IEEE-Standard 829 genügen.
2. Die einzelnen Dokumente des Standards müssen genau analysiert werden, um Formen der Automatisierung und Vereinfachung im Hinblick auf eine spätere Software gestützte Erstellung zu erkennen.

⁴beispielsweise durch Erfassung der durchgeführten Testläufe und deren Ergebnisse

3. Eine Software soll entwickelt werden, die es erlaubt, die Dokumente auf einfache Art und Weise zu erstellen. Sie soll die Logik implementieren, die erforderlich ist, um der Struktur und der Konsistenz der einzelnen Dokumente gerecht zu werden und diese zu verwalten und zu verknüpfen.
4. Die Software sollte für beliebige Dokumente konfigurierbar sein, so dass sie für Veränderungen der Dokumentstrukturen offen ist oder gar für völlig andere Zwecke und Dokumenttypen eingesetzt werden kann.
5. Die Definition und das Ausgabeformat der elektronischen Dokumente soll strukturiert und systemunabhängig sein. Hierfür soll XML⁵ zum Einsatz kommen.
6. Die Software soll ein hohes Maß an Bedienerfreundlichkeit aufweisen und den Anwender Schritt für Schritt begleiten, ohne dass genaue Kenntnisse des IEEE-Standards 829 erforderlich sind.
7. Die entwickelte Software soll die genannten Punkte prototypisch implementieren und einen Export ins HTML⁶-Format ermöglichen.

⁵XML: *eXtensible Markup Language* [XML]

⁶HTML: *HyperText Markup Language*

1.3 Aufbau des Dokuments

Dieses Dokument ist in folgende Abschnitte gegliedert:

Kapitel 1	Dieses Kapitel.
Kapitel 2	Zunächst soll dargestellt werden, was eine Testdokumentation eigentlich beinhaltet. Dazu werden in diesem Kapitel der IEEE829-Standard und seine Dokumente vorgestellt. Es sollen hier auch bereits die wesentlichen Strukturen innerhalb der Dokumente und Beziehungen der Dokumente untereinander verdeutlicht werden.
Kapitel 3	Nach der Betrachtung des Standards werden in diesem Kapitel die formalen Mittel erarbeitet, die eine automatisierte Erstellung der Dokumente ermöglichen. Dazu werden die Strukturen der Dokumente soweit aufgebrochen, dass eine Erstellung von <i>Document Type Definitions</i> (DTD) vorgenommen werden kann.
Kapitel 4	Dieses Kapitel stellt den Software-Prototypen vor. Es werden das Software-Modell, die wesentlichen Implementierungsformen sowie Installation und Bedienung der Software beschrieben.
Kapitel 5	Dieses Kapitel geht auf mögliche Erweiterungen der Software ein. Es wird beschrieben, wie sich die Software für beliebige Anwendungsfälle einsetzen lässt und welche weiterführenden Implementierungen vorgenommen werden können.

Kapitel 2

Der IEEE829-Standard

2.1 Warum IEEE829?

Der IEEE-Standard 829 [IEEE829] ist der bekannteste und einzige Standard für Software-Testdokumentationen, der weltweit etabliert ist und eingesetzt wird. Zum Zeitpunkt der Erstellung dieser Arbeit waren unter den Suchbegriffen "*standard software test documentation*" keine nennenswerten Dokumente im World-Wide-Web zu finden, die einen Standard für Testdokumentationen beschrieben oder erwähnten, der nicht auf dem IEEE-Standard basiert. Es kann daher von einer breiten Akzeptanz dieses Standards ausgegangen werden.

Die Standardisierungsgremien des *IEEE Standards Association (SA) Standards Board* [IEEE-SA] setzen sich aus freiwilligen Mitgliedern zusammen, die ehrenamtlich tätig sind und auch nicht notwendigerweise Mitglieder des Instituts [IEEE] (sprich: *ai-trippel-i*) sind. Der verabschiedete Standard stellt einen Konsens der Mitglieder der Gremien und aller Gruppen außerhalb der Gremien, die Interesse an der Entwicklung des Standards bekundet haben, dar. Den breitgefächerten Einfluss verschiedener Experten auf diesem Gebiet kann man sich durch die Verwendung dieses Standards zu Nutze machen.

Alle IEEE-Standards werden mindestens alle fünf Jahre zur Revision vorgelegt und falls notwendig an die aktuellen Entwicklungen angepasst oder erweitert. Die aktuelle Revision des hier verwendeten Standards 829 wurde am 16. September 1998 verabschiedet. Diese Revision ist Grundlage dieser Diplomarbeit. Obwohl eine Wiedervorlage des Standards in naher Zukunft bevorsteht (voraussichtlich im Herbst 2003), ist nicht zu erwarten, dass es wesentliche Änderungen geben wird, die die Gültigkeit oder Aktualität dieser Arbeit beeinflus-

sen könnten, zumal sich die erarbeitete Lösung flexibel an mögliche Änderungen des Standards anpassen lässt.

Die Dokumente der IEEE-Standards unterliegen dem US-Copyright und können gegen Entrichtung einer Gebühr über die IEEE-Standards-Website [IEEE-SA] bezogen werden. Das Einzeldokument des Standards 829 (in elektronischer Form als PDF-Dokument) kostete zum Zeitpunkt der Verfassung dieser Diplomarbeit 98 US-Dollar.

2.2 Einleitung

In diesem Kapitel wird der IEEE-Standard 829 vorgestellt. Es geht hier nicht darum, jeden im Standard beschriebenen Dokumententyp bis in die kleinsten Unterabschnitte im Detail vorzustellen. Stattdessen wurde Wert darauf gelegt, dass die Struktur und die wichtigsten Eigenschaften und Inhalte der Dokumente dargestellt werden. Dadurch soll vor allen Dingen deutlich werden, inwiefern sich die Erstellung der Dokumente später automatisieren lassen kann.

Der IEEE-Standard 829 definiert eine Menge verschiedener Testdokumenttypen und beschreibt deren Inhalte. Er legt nicht fest, welche Dokumente im konkreten Fall Verwendung finden sollen, das heißt, eine Testdokumentation muss nicht notwendigerweise alle Dokumenttypen beinhalten und es obliegt dem Anwender des Standards, für seinen Fall zu entscheiden, welche Dokumente er in wievielfacher Ausführung und zu welchem Zeitpunkt erstellen möchte. Der Standard definiert ebenfalls nicht, welche Dokumente im Rahmen der Testaktivitäten zu welchem Zeitpunkt und an welchen Orten zum Einsatz kommen sollen. Er gibt auch keine Vorgaben zu Art und Umfang der eingesetzten Hilfsmittel, Werkzeuge, Vorgehensweisen und Methodiken für die Dokumentationsverwaltung und Testaktivitäten. Dies ist Aufgabe einer übergeordneten Projektorganisation und eines Qualitätsmanagements. Der Standard beschreibt lediglich den Zweck, die Gliederung und den Inhalt der einzelnen Test-Dokumententypen.

Für alle im Standard definierten Dokumente gilt grundsätzlich:

- Die Inhalte der einzelnen Abschnitte können je nach Anwendungsfall und Testphase zurecht geschnitten werden, das heißt, Abschnitte können den Anforderungen entsprechend verkürzt oder um zusätzliche Inhalte erweitert werden.
- Abschnitte können, falls erforderlich, in zusätzliche Unterabschnitte unterteilt werden.

- Die Inhalte einzelner Abschnitte können teilweise oder vollständig in eigene Dokumente ausgelagert werden, die dann zu referenzieren sind.

Im Folgenden werden die acht Dokumente des Standards vorgestellt.

Hinweis: In diesem Kapitel soll bei der Kurzübersicht über die Dokumente bereits erkenntlich werden, welche Teile für den Automatisierungsprozess und die Erstellung der Software von struktureller Bedeutung sein können. Zu diesem Zweck wurden diese Inhalte, die über reine Prosa hinausgehen und gewisse Strukturinformationen darstellen, durch **Fettschrift** hervorgehoben.

2.3 Die Dokumente

2.3.1 Testplan

Der Testplan ist das zentrale Dokument jeder Testdokumentation und sollte als erstes erstellt werden. Er beschreibt Umfang, Vorgehensweise und Terminplan der Testaktivitäten. Der Testplan identifiziert außerdem die Testgegenstände und deren zu testende Eigenschaften bzw. Funktionen. Ferner werden die durchzuführenden Maßnahmen und die dafür verantwortlichen Personen sowie die allgemeinen Risiken definiert.

Im Einzelnen enthält der Testplan die folgenden Punkte:

- **Identifikator:** Eindeutige Kennung des Testplans
- **Einführung:** Kurze **Beschreibung** des Projektumfelds und der Testgegenstände. In diesem Abschnitt können **Referenzen zu externen Dokumenten**, die nicht zur Testdokumentation gehören, eingefügt werden, beispielsweise Referenzen zu Qualitätssicherungs- und Projektplänen. Ferner wird im Standard an dieser Stelle erwähnt, dass im Falle von mehrstufigen Testplänen eine **Referenz auf den übergeordneten Testplan** angegeben werden soll. Es können also mehrere Testpläne existieren, die in einer hierarchischen Beziehung zueinander stehen.
- **Testgegenstände:** Genaue Definition der Testgegenstände. Zur Definition eines Testgegenstand gehören: **Name und Versionsnummer, Referenzen auf Dokumentationen** des Testgegenstands (**Anforderungsdokument, Designspezifikation, Benutzer- bzw. Bedienungs-handbuch, Installationshandbuch**). Außerdem sind Besonderheiten der Übertragungsmedien in Bezug auf Anforderungen des

Testsystems anzugeben¹. Ferner sind **Referenzen auf alle Testvorfallberichte** (siehe unten) anzugeben.

- **Zu testende Funktionen:** Liste aller Eigenschaften bzw. **Funktionen und deren Kombinationen**, die zu testen sind. Für jede Funktion und Funktionskombination ist die Referenz auf die entsprechende **Testdesignspezifikation** (siehe unten) anzugeben.
- **Nicht zu testende Funktionen:** Lister aller Eigenschaften bzw. **Funktionen und deren Kombinationen**, die nicht zu testen sind und die **Gründe** dazu.
- **Vorgehensweise:** Beschreibung der generellen **Vorgehensweisen für die einzelnen zu testenden Funktionen und Funktionskombinationen**. Hierzu sind jeweils die eingesetzten Techniken und Hilfswerkzeuge anzugeben. Die Beschreibung sollte detailliert genug sein, um die Hauptaktivitäten und deren Zeitbedarf abschätzen zu können. Die Vorgehensweisen werden jedoch in den oben referenzierten Testdesignspezifikationen verfeinert. Es sollten auch der gewünschte Umfang der Testaktivitäten angegeben werden und eventuelle Nebenbedingungen und Abhängigkeiten (von bestimmten Ressourcen) beschrieben werden. Es müssen Kriterien angegeben werden, die es erlauben, im Nachhinein den Grad des Testumfangs zu bestimmen (beispielsweise Anzahl der Testfälle, die mindestens einmal ausgeführt wurden).
- **Pass/Fail-Kriterien der Testgegenstände:** Definition der **Kriterien**, anhand derer zu bestimmen ist, ob das Testen eines **Gegenstands** gescheitert oder erfolgreich ist.
- **Aussetzungs- und Wiederaufnahmeanforderungen:** Definition der Bedingungen, die zur teilweisen oder vollständigen Aussetzung der Testaktivitäten führen sollten, und Definition der Testaktivitäten, die zu wiederholen sind, wenn das Testen fortgeführt wird.
- **Auslieferungsgegenstände:** Auflistung der auszuliefernden Dokumente. Dazu gehören **Testplan, Testdesignspezifikationen, Testfallspezifikationen, Testablaufspezifikationen, Testgegenstandsauslieferungsberichte, Testprotokolle, Testvorfallberichte, Testergebnisberichte**.
- **Testaktivitäten:** Beschreibung der **Aktivitäten** zur Vorbereitung und Durchführung der Tests.

¹beispielsweise die Notwendigkeit, eine Software auf CD zu brennen, um sie auf das Testsystem zu übertragen

- **Umgebungsanforderungen:** Beschreibung der technischen Voraussetzungen für die Testumgebung. Dies umfasst Hard- und Softwareanforderungen, Sicherheitsanforderungen und sonstige Voraussetzungen der Testumgebung.
- **Verantwortlichkeiten:** Es sind alle Personen und Personengruppen zu identifizieren, die für die Verwaltung, das Design, die Vorbereitungen, Durchführungen, Bezeugungen, Überprüfungen und Entscheidungen zuständig sind. Ferner sind die Personen aufzuführen, die für die Bereitstellung der Testgegenstände und der Testumgebung verantwortlich sind.
- **Personal- und Ausbildungsanforderungen:** Beschreibung der Anforderungen an Personal und Ausbildungsniveau.
- **Zeitplan:** Es sind die **Meilensteine** der Projektplanung und der Auslieferungsgegenstände aufzuführen. Diese sind um zusätzliche Test-Meilensteine zu ergänzen. Die geschätzte **Zeitdauer** und ein **Zeitplan** für jede einzelne **Testaktivität** ist anzugeben. Für jede **Testressource** (z.B. Einrichtungen, Werkzeuge und Personal) ist der **Zeitraum der Nutzung** anzugeben.
- **Risiken und Eventualitäten:** Alle Risiken und Eventualitäten des Testplans sind anzugeben. Pläne für die Reaktion auf diese Probleme sind anzugeben.
- **Genehmigungen:** Die **Namen und Titel aller Personen** sind hier aufzulisten, die diesem Testplan zustimmen müssen. Hierzu gehört Raum für **Datum und Unterschriften**.

2.3.2 Testdesignspezifikation

Dieser Dokumenttyp gehört mit den beiden folgenden Dokumenttypen *Testfallspezifikation* und *Testablaufspezifikation* zur Gruppe der Testspezifikation. Die Testdesignspezifikation ist ebenfalls ein wichtiges Element jeder Testdokumentation und hierarchisch unterhalb des Testplans angeordnet. Im Testplan wurden, wie oben dargestellt, die einzelnen Testgegenstände und deren zu testende Funktionen angegeben. Zu den zu testenden Funktionen und Funktionskombinationen wurden im Testplan Referenzen auf diesen jetzt beschriebenen Dokumententyp eingefügt. Die Testdesignspezifikation verfeinert die im Testplan genannten Vorgehensweisen im Hinblick auf die ihr zugewiesenen Funktionen und Testgegenstände.

Im Einzelnen enthält sie die folgenden Punkte:

- **Identifikator:** Eindeutige Kennung der Testdesignspezifikation. Hier muss auch eine **Referenz auf den Testplan** angegeben werden.
- **Zu testende Funktionen:** Liste der **Testgegenstände** und der **Funktionen** bzw. **Funktionskombinationen**, die Gegenstand dieser Testdesignspezifikation sind. **Referenzen auf die zutreffenden Inhalte im Anforderungsdokument oder in der Designspezifikation des jeweiligen Testgegenstands** sollten angegeben werden.
- **Verfeinerung der Vorgehensweise:** Die Beschreibungen des Testplans sind im Hinblick auf die zu testenden Funktionen zu verfeinern. Es sollten die Methoden der Testdurchführung und der Ergebnisanalyse sowie die Gründe für die Auswahl bestimmter Testfälle angegeben werden. In diesem Abschnitt sollten auch alle **Eigenschaften und Randbedingungen** angegeben werden, die **allen** (im nächsten Abschnitt genannten) **Testfällen** **gemein** sind.
- **Testidentifikation:** Liste der Referenzen aller **Testfallspezifikationen**, die mit dieser Testdesignspezifikation verknüpft sind. Zu jeder Referenz sollte eine **kurze Beschreibung der damit abgedeckten Funktionen** angegeben werden.
- **Test/Fail-Kriterien der Funktionen:** Definition der **Kriterien**, anhand derer zu bestimmen ist, ob das Testen einer **Funktion** gescheitert oder erfolgreich ist.

2.3.3 Testfallspezifikation

Dieser Dokumenttyp beschreibt einen konkreten Testfall, das heißt einen "Test" im engeren Sinne. Eine Testfallspezifikation legt alle Umgebungsbedingungen, Eingaben und Ausgaben genau fest. Testfallspezifikationen sind eigenständige Dokumente und nicht Teil einer Testdesignspezifikation, damit sie auch in anderen Situationen verwendet werden können (sie können also auch von verschiedenen Testdesignspezifikationen referenziert werden). Testfallspezifikationen sollten deshalb hinreichend detailliert sein und genügend Informationen beinhalten, um sie später und in anderen Situationen wiederverwenden zu können.

Im Einzelnen enthält eine Testfallspezifikation die folgenden Punkte:

- **Identifikator:** Eindeutige Kennung der Testfallspezifikation.

- **Testgegenstände:** Liste der **Testgegenstände und Funktionen** bzw. Eigenschaften, die von diesem Testfall überprüft werden. **Referenzen auf die Dokumentationen** der Testgegenstände sind anzugeben.
- **Spezifikation der Eingaben:** Detaillierte Angabe aller **Eingaben** (wie Werte, Namen, Tabellen, Dateien etc.) **und deren Beziehungen** (zum Beispiel Zeitfolge).
- **Spezifikation der Ausgaben:** Auflistung aller **Ausgaben und Eigenschaften** (wie z.B. Antwortzeiten), die zu erfassen sind und deren **Werte bzw. Wertebereiche**.
- **Umgebungsanforderungen:** Detaillierte Beschreibung der **Hardware- und Softwareanforderungen und anderer Anforderungen** (z.B. spezielles Personal).
- **Spezielle Ablaufanforderungen:** Definition aller speziellen Beschränkungen oder Eigenheiten, die von eventuellen Testablaufspezifikationen (siehe Abschnitt 2.3.4), die diesen Testfall ausführen, abweichen.
- **Wechselbeziehungen zu anderen Testfällen:** Auflistung aller **Testfälle**, die vor diesem Testfall auszuführen sind und Angabe der Art der Wechselbeziehungen.

2.3.4 Testablaufspezifikation

Eine Testablaufspezifikation definiert Schritt für Schritt, wie die Testfälle, mit denen sie verknüpft ist, auszuführen sind, um den Vorgaben der Testdesignspezifikation zu genügen. Testablaufspezifikationen sind eigenständige Dokumente und nicht Teil der Testdesignspezifikation, weil sie eine schrittweise Ausführungsanleitung darstellen und daher keine für die technische Durchführung irrelevanten Informationen beinhalten sollen.

Im Einzelnen enthält eine Testablaufspezifikation die folgenden Punkte:

- **Identifikator:** Eindeutige Kennung der Testablaufspezifikation und **Referenz auf die Testdesignspezifikation**.
- **Verwendungszweck:** Beschreibung des Verwendungszwecks dieser Testablaufspezifikation. Es sind **Referenzen auf die Testfälle** anzugeben, die von dieser Testablaufspezifikation ausgeführt werden. **Relevante Teile der Testgegenstandsdokumentationen** sollten ebenfalls aufgeführt werden.

- **Spezielle Anforderungen:** Auflistung aller speziellen Voraussetzungen (beispielsweise für Testumgebung oder Personal) für die Durchführung der Testfälle gemäß dieser Testablaufspezifikation.
- **Ausführungsschritte:** Beschreibungen für die folgenden Schritte sind (je nach Bedarf) anzugeben: **Protokollerfassung, Vorbereitungen, Start, Ablauf, Messungen, Unterbrechung, Neustart, Ende, Nachbereitung, Besonderheitenbehandlung.**

2.3.5 Testgegenstandsauslieferungsbericht

Dieser Dokumenttyp beschreibt die Testgegenstände für den Fall, dass separate Entwicklungs- und Testteams bestehen oder ein formaler Beginn der Testdurchführungen gewünscht ist. In diesem Fall werden im Testgegenstandsauslieferungsbericht die verantwortlichen Personen, der physikalische Aufbewahrungsort sowie der Status der Testgegenstände festgehalten und die Randbedingungen der Übergabe dieser Gegenstände definiert.

Im Einzelnen enthält ein Testgegenstandsauslieferungsbericht die folgenden Punkte:

- **Identifikator:** Eindeutige Kennung des Testgegenstandsauslieferungsbericht.
- **Übergebene Gegenstände:** Auflistung der Testgegenstände, deren Versionsnummern und Dokumentationen sowie der für jeden Gegenstand verantwortlichen Personen.
- **Ort:** Zu jedem übergebenen Gegenstand ist eine Ortsangabe und eine Medienbezeichnung anzugeben.
- **Status:** Beschreibung des Status der übergebenen Gegenstände und eventueller Abweichungen von deren Dokumentationen, von früheren Übergaben oder vom Testplan. Liste der **Testvorfallberichte**, die durch diese Auslieferung behoben sein sollten.
- **Genehmigungen:** Die **Namen und Titel aller Personen** sind hier aufzulisten, die diesem Testgegenstandsauslieferungsbericht zustimmen müssen. Hierzu gehört Raum für **Datum und Unterschriften**.

2.3.6 Testprotokoll

Ein Testprotokoll stellt eine chronologische Auflistung aller relevanten Details einer Testausführung dar und beinhaltet im Einzelnen die folgenden Punkte:

- **Identifikator:** Eindeutige Kennung des Testprotokolls.
- **Beschreibung:** Hier sollten alle Informationen aufgeführt werden, die auf alle (folgenden) Einträge des Protokolls zutreffen. Hierzu gehören: Angabe der **Testgegenstände** und ihrer **Versionsnummern** sowie, falls vorhanden, der dazugehörigen **Testgegenstandsauslieferungsberichte** und Auflistung aller Details zu den **Eigenschaften der Testumgebung**.
- **Ereigniseinträge:** Für jeden Ereigniseintrag ist **Datum, Uhrzeit und Autor** zu vermerken. Beginn und Ende der einzelnen durchgeführten Aktivitäten sind ebenfalls als Ereignisse zu vermerken. Desweiteren sind folgende Informationen **für jeden Eintrag** zu berücksichtigen:
 - **Ausführungsbeschreibungen:** Identifikator und **Referenz auf die Testablaufspezifikation, Liste der anwesenden Personen und deren Funktion**.
 - **Ablaufergebnisse:** Inhalt und Ort der sichtbaren Ausgaben und Angabe, ob Ausführung **erfolgreich** war **oder gescheitert** ist.
 - **Umgebungsinformation:** Spezielle Umgebungsbedingungen für diesen Eintrag (beispielsweise Hardwareänderung).
 - **Ungewöhnliche Ereignisse:** Geschehnisse vor und nach dem Ereignis
 - **Vorfallberichte:** Referenz auf **Testvorfallbericht**, wenn ein solcher generiert wurde.

2.3.7 Testvorfallbericht

In einem Testvorfallbericht werden die Ereignisse festgehalten, die während der Durchführung von Tests aufgetreten sind und die eine weitere Untersuchung erforderlich machen. Im Einzelnen enthält ein Testvorfallbericht die folgenden Punkte:

- **Identifikator:** Eindeutige Kennung des Testvorfallberichts.

- **Zusammenfassung:** Zusammenfassende Beschreibung des Vorfalls. Die **Testgegenstände**, ihre **Versionsnummern**, Referenzen zur **Testablaufspezifikation**, **Testfallspezifikation** und **Testprotokoll** sollten aufgeführt werden.
- **Beschreibung des Vorfalls:** Die Beschreibung sollte die folgenden Punkte beinhalten: **Eingaben**, **Erwartete Ausgaben**, **Tatsächliche Ausgaben**, **Anomalien**, **Datum und Uhrzeit**, **Ablaufschritt**, **Umgebung**, **Wiederholversuche**, Namen der **Tester** und **Beobachter** und alle weiteren Angaben, die zur Klärung der Ursache und zur Beiseitigung des aufgetretenen Fehlers beitragen können.
- **Auswirkungen:** Es ist abzuschätzen, welche Auswirkungen der Vorfall auf Testpläne, Testdesignspezifikationen, Testablaufspezifikationen oder Testfallspezifikationen hat.

2.3.8 Testergebnisbericht

Ein Testergebnisbericht fasst die Ergebnisse aller Testaktivitäten einer oder mehrerer Testdesignspezifikationen zusammen und wertet diese Ergebnisse aus. Er enthält die folgenden Punkte:

- **Identifikator:** Eindeutige Kennung des Testergebnisberichts.
- **Zusammenfassung:** Zusammenfassende Betrachtung der Untersuchungen. Es sind die **Testgegenstände**, deren **Versionsnummern** und die **Testumgebung** zu beschreiben. **Für jeden Testgegenstand** sind Referenzen auf die folgenden Dokumente anzugeben, wenn diese vorhanden sind: **Testplan**, **Testdesignspezifikationen**, **Testablaufspezifikationen**, **Testgegenstandsauslieferungsberichte**, **Testprotokolle** und **Testvorfallberichte**.
- **Abweichungen:** Es sind alle Abweichungen der Testgegenstände von ihren Designspezifikationen sowie Abweichungen von Testplan, Testdesignspezifikationen oder Testablaufspezifikationen und die Gründe für die Abweichungen anzugeben.
- **Umfangseinschätzung:** Abschätzung des Grades des Testumfangs nach den Kriterien des Testplans (siehe Testplan, Kapitel 2.3.1, Abschnitt *Vorgehensweise*, Seite 15). **Funktionen und Funktionskombinationen**, die noch nicht hinreichend getestet wurden, sind aufzuführen und die Gründe anzugeben.

- **Ergebniszusammenfassung:** Zusammenfassung der Testergebnisse. Alle **Lösungen von Vorfällen** (siehe Testvorfallbericht, Kapitel 2.3.7) sind aufzuführen und zu beschreiben. **Ungelöste Vorfälle** sind anzugeben.
- **Auswertung:** Für jeden Testgegenstand ist eine umfassende Beurteilung anzugeben, die auf den Testergebnissen basiert. Eine Abschätzung des Ausfallrisikos für einen Testgegenstand kann hier angegeben werden.
- **Zusammenfassung der Aktivitäten:** Zusammenfassende Beschreibung der Hauptaktivitäten und -ereignisse. Für jede Testaktivität sollte ein Ressourcenverbrauch angegeben werden (beispielsweise Personalaufwand, Maschinenlaufzeiten etc.).
- **Genehmigungen:** Die **Namen und Titel aller Personen** sind hier aufzulisten, die diesem Testergebnisbericht zustimmen müssen. Hierzu gehört Raum für **Datum und Unterschriften**.

2.4 Zusammenfassung

In den vorangegangenen Abschnitten wurden die acht Dokumente des Standards vorgestellt. Es sollte insbesondere gezeigt werden, dass die Dokumente eine Vielzahl von gegenseitigen Referenzen enthalten und viele Informationen redundant vorhanden sind. Diese Redundanz ist oft bewusst vorhanden, um die Eigenständigkeit einzelner Dokumente zu ermöglichen. Im Hinblick auf eine automatisierte Erstellung der Dokumente ergeben sich hieraus bereits eine Reihe interessanter Vereinfachungsmöglichkeiten.

Zunächst soll die Struktur der Gesamtdokumentation gezeigt werden. Ein erster Blick auf die Dokumente des Standards macht es schwer, die komplexen Beziehungen der Dokumente untereinander zu erkennen. Daher soll die Grafik 2.1 auf Seite 23 weiterhelfen. Sie soll die gegenseitigen Referenzen deutlich machen und gibt auch die Kardinalitäten dieser Referenzen an.

Zur Grafik sind die folgenden Anmerkungen zu machen: Der Testplan referenziert zusätzlich zu den in der Grafik dargestellten Beziehungen alle anderen Dokumente in einer 1-zu-N-Beziehung, da er alle auszuliefernden Dokumente erwähnt (siehe Abschnitt *Auslieferungsgegenstände* im Testplan, Seite 15). Der Testergebnisbericht referenziert die Dokumente Testdesignspezifikation, Testablaufspezifikation, Testgegenstandsauslieferungsbericht und Testprotokoll in einer N-zu-M- und den Testplan in einer N-zu-1-Beziehung. Um die Grafik übersichtlich zu halten und die wesentliche Struktur zu verdeutlichen, wurden diese Beziehungen nur angedeutet.

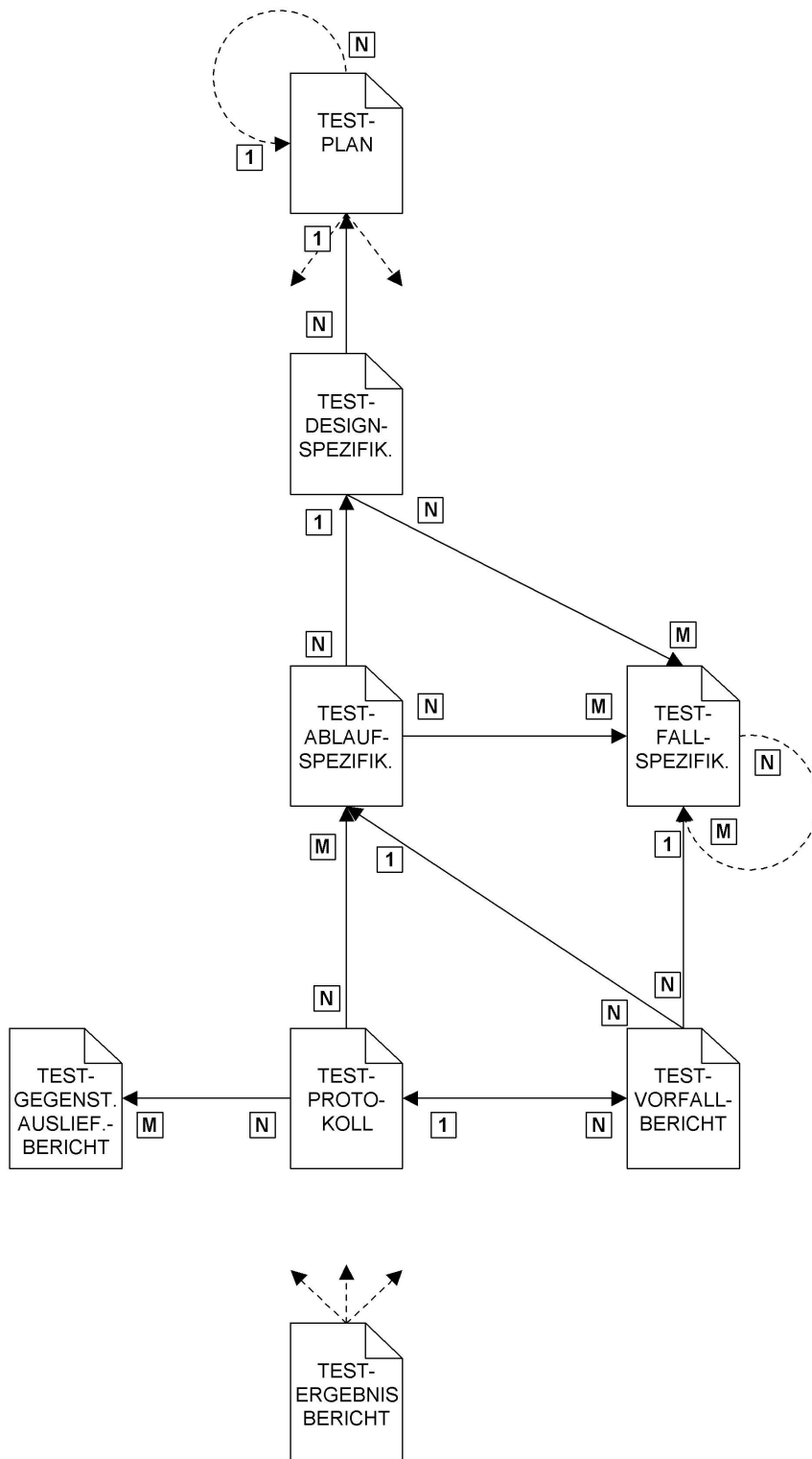


Abbildung 2.1: Abhängigkeiten der Dokumente

2.5 Erweiterungen

Es wurde gezeigt, dass eine Vielzahl von Referenzen unter den einzelnen Dokumenten besteht. Eine Software kann hier bereits erhebliche Vereinfachungen herbeiführen. Anstatt solche Referenzen manuell eingeben zu müssen, kann die Software diese Referenzen dem Anwender auf bequeme Art und Weise in einer Dokumentenliste zur Auswahl anbieten. Ferner kann sie für die Konsistenz der vielfachen Beziehungen sorgen, indem sie die Gültigkeit der einzelnen Referenzen überprüft. Im Hinblick auf eine automatisierte Erstellung der Dokumente sollen in diesem Abschnitt jedoch noch weitere Vereinfachungen betrachtet werden, die durch Erweiterungen der vorliegenden Dokumentenstruktur erreicht werden kann.

Es geht nicht etwa darum, neue Dokumenttypen zu definieren. Es sollen nur zusätzliche Strukturen erarbeitet werden, die in der späteren Software zum Einsatz kommen und die Erstellung der im Standard definierten Dokumente vereinfachen. In den folgenden Abschnitten werden diese Strukturen vorgestellt.

2.5.1 Testgegenstände

Bei der Betrachtung der Dokumente fällt auf, dass viele Informationen in den einzelnen Dokumenten redundant vorhanden sind. Insbesondere trifft dies auf die Angaben zu den Testgegenständen zu. In allen Dokumenten sind Angaben zu den Testgegenständen, zu deren Funktionen und Dokumentationen vorhanden. Um diese Angaben nicht in jedem Dokument manuell und wiederholt eingeben zu müssen, wäre es wünschenswert, wenn die Testgegenstände gesondert definiert werden können. Die Definition eines Testgegenstands sollte dabei folgende Angaben umfassen:

- **Bezeichnung:** Eindeutige Bezeichnung des Testgegenstands
- **Beschreibung:** Kurze Beschreibung des Testgegenstands
- **Dokumentation:** Referenz auf die Dokumentationen des Testgegenstands und deren Aufbewahrungsorte. Dazu gehören **Anforderungsdokument, Designspezifikation, Benutzer- bzw. Bedienungshandbuch** und **Installationshandbuch**.
- **Funktionen und Eigenschaften:** Liste der Funktionen und Eigenschaften des Testgegenstands. Zu jeder Funktion bzw. zu jeder Eigenschaft sind anzugeben:

- **Bezeichnung:** Eindeutige Bezeichnung der Funktion bzw. Eigenschaft
- **Beschreibung:** Ausführliche Beschreibung der Funktion bzw. Eigenschaft. Es ist anzugeben, wie diese Funktionen und Eigenschaften zu testen sind und welche **Rückgabewerte** sie besitzen, anhand derer sich Scheitern und Erfolg messen lassen.

Die separate Erfassung der Testgegenstände an zentraler Stelle vereinfacht das Erstellen der Testdokumente und trägt dazu bei, die Konsistenz der Gesamtdokumentation zu wahren.

2.5.2 Parametrisierte Testfälle

Es kommt in der Praxis sehr häufig vor, dass ein Testgegenstand mit einer Vielzahl verschiedener Umgebungseinstellungen und Eingabewerten getestet werden muss. So müssen die Funktionen einer Software nicht nur mit verschiedenen Eingaben durchgespielt werden, sondern es müssen auch häufig Umgebungseinstellungen variiert werden. Beispielsweise muss eine Software auf verschiedenen Betriebssystemen oder mit verschiedener Hilfssoftware oder Systemeinstellungen getestet werden.

Eine Testfallspezifikation (siehe Kapitel 2.3.3, Seite 17) definiert alle Einstellungen, Parameter und Eingaben - sowohl für den Testgegenstand als auch die Umgebung. Dies bedeutet, dass eine Testfallspezifikation für jeden einzelnen Testfall erstellt werden muss. Da sich diese Dokumente aber nur in wenigen Details unterscheiden, ist es hilfreich, wenn man so genannte *Parametrisierte Testfälle* einführt. Dies erlaubt es, Testfallspezifikationen mit Platzhaltern zu erstellen, die dann durch vorher definierte Wertekombinationen ersetzt werden können. Dies bringt folgende Vorteile:

- Alle fixen Bestandteile einer Testfallspezifikation können vorab einmalig definiert werden und müssen nicht doppelt eingegeben werden. Alle dynamischen (zu variierenden) Bestandteile des Testfallspezifikation werden durch benannte Platzhalter ersetzt.
- Die Platzhalter stellen Parameter einer Testfallspezifikation dar. Indem die zu testenden Werte für jeden Parameter vorab festgelegt werden, können Testspezifikationen für alle Wertekombinationen sehr einfach erstellt werden.

Kapitel 3

Formen der Automatisierung

3.1 Einführung

Auf der Basis der Betrachtungen des vorhergehenden Kapitels soll nun erarbeitet werden, wie eine Automatisierung der Dokumentationserstellung erfolgen kann. In diesem Kapitel werden die Grundlagen für die im nächsten Kapitel vorgestellte Software geschaffen. Zwei Dinge sollen hier definiert werden:

1. Die Definitionen der Dokumente, wie sie im Standard beschrieben werden, müssen in eine maschinenverarbeitbare Form gebracht werden. Da die Dokumente nicht nur automatisch generiert, sondern auch verarbeitet, gespeichert und wieder eingelesen werden müssen, und zwar so, dass eine Software damit sinnvoll umgehen kann, ist genau zu definieren, welche strukturellen Komponenten ein Dokument besitzen muss oder besitzen kann. Dies kann mit Hilfe einer so genannten *Document Type Definition* (DTD)¹ erreicht werden.
2. Es muss eine allgemeine Möglichkeit gefunden werden, die es erlaubt, ein beliebiges Dokument mit Hilfe einer Software zu erstellen. Es würde wenig Sinn machen, die Software in der Art und Weise zu programmieren, dass sie die Strukturen der Dokumente fest verdrahtet beinhaltet. Es sollte vielmehr möglich sein, die einzelnen Abschnitte der Dokumente

¹Eine DTD (Document Type Definition) legt fest, aus welchen Elementen eine XML-Datei bestehen darf. Der DTD-Standard ist vielerorts schon durch seinen leistungsfähigeren Nachfolger XSD (XML Schema Definition) ersetzt worden. Da aber die zusätzlichen Möglichkeiten von XSD im Rahmen dieser Arbeit keine Verwendung finden und eine DTD einfacher zu lesen ist und auch kürzer gehalten werden kann, wird in dieser Arbeit auf die Anwendung von XSD verzichtet. (Siehe hierzu [XML, DTD, XSD, W3C, W3SCHOOLS])

und die Editoren für ihre Bearbeitung mit Hilfe einer Konfigurationsdatei zu definieren. Dadurch ist die Software flexibel einsetzbar und kann als intelligentes Erstellungswerkzeug für beliebige Dokumente dienen.

3.2 Definition der Dokumente

3.2.1 XML und DTD

Als Ausgabeformat der erstellten Dokumente eignet sich die *eXtensible Markup Language* (siehe [XML]). XML ist eine einfache und sehr flexible Text-Formatierungssprache. Vereinfacht gesagt werden bei XML die Teile eines Dokuments mit öffnenden und schließenden *Tags* umklammert, die der entsprechend umklammerten Information eine strukturelle Bedeutung geben. Eine Postleitzahl 61250 wird beispielsweise folgendermaßen umklammert: `<PLZ>61250</PLZ>`. Dabei ist `<PLZ>` das öffnende und `</PLZ>` das schließende Tag. Eine vollständige Adresse kann folgendermaßen ausgedrückt werden:

```
<Adresse>
  <Name>Thorsten Liese</Name>
  <Straße>Rhönring 75</Straße>
  <PLZ>64289</PLZ>
  <Ort>Darmstadt</Ort>
</Adresse>
```

Diese Form der Speicherung macht Dokumente maschinell verarbeitbar. XML-Dokumente lassen sich später in beliebige Anzeigeformate wie beispielsweise HTML² oder PDF³ umwandeln.

Mit Hilfe von XML können also die erstellten Dokumente formatiert werden. Allerdings muss noch definiert werden, aus welchen Bestandteilen (aus welchen Tags) ein bestimmtes Dokument überhaupt bestehen darf oder muss. Um beim obigen Beispiel zu bleiben: Es muss festgelegt sein, dass eine Adresse eben genau aus einem Namen, einer Straße, einer Postleitzahl und einem Ort besteht, vielleicht optional noch aus einer beliebigen Menge von Telefonnummern. Dies kann mit Hilfe einer DTD erreicht werden. Eine DTD für das Adresenbeispiel sieht so aus:

²Die HyperText Markup Language (HTML) kommt als Anzeigeformat im World-Wide-Web (WWW) zum Einsatz.

³Portable Document Format (PDF) von Acrobat.

```

<!ELEMENT Adresse (Name, Straße, PLZ, Ort, Telefonnummer*)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Straße (#PCDATA)>
<!ELEMENT PLZ (#PCDATA)>
<!ELEMENT Ort (#PCDATA)>
<!ELEMENT Telefonnummer (#PCDATA)>

```

Zeile 1 besagt, dass das Adresse-Element genau die in der runden Klammer aufgeführten Elemente enthalten muss, und zwar in der angegebenen Reihenfolge. Das Sternchen hinter `Telefonnummer` bedeutet, dass dieses Element entweder gar nicht oder beliebig oft wiederholt angegeben werden darf (0 bis n-mal). Ein Pluszeichen hinter einem Elementnamen drückt aus, dass das Element ein- bis n-mal vorkommen darf, ein Fragezeichen bedeutet null- oder einmal. Diese kurze Einleitung sollte die in den folgenden Abschnitten vorgestellten DTDs verständlich machen. Für weiterführende Informationen zu XML und DTD sei auf die entsprechende Literatur verwiesen ([DTD, XML, W3C, W3SCHOOLS]).

Man kann nun die Dokumente des IEEE829-Standards mit Hilfe von DTDs beschreiben. Die im vorherigen Kapitel durch **Fettschrift** hervorgehobenen Inhalte eines Dokuments werden dazu in entsprechende DTD-Elemente umgewandelt.

Bevor die DTDs der einzelnen Dokumente vorgestellt werden, soll hier noch ein allgemeiner Bestandteil aller Dokumente definiert werden. Alle Dokumente beinhalten Abschnitte mit normalem Prosatext. In diesen Abschnitten sollte es dem Anwender möglich sein, einfache Formatierungen wie Aufzählungen, Fett- und Kursivschrift vorzunehmen. Solche Abschnitte, die als `formattedtext` bezeichnet sind, haben folgende Struktur:

```

<?xml version="1.0" encoding="UTF-8"?>

<!--
  Document : formattedtext.dtd
  Author   : Thorsten Liese
  Description:
    DTD for formatted text
-->

<!ELEMENT formattedtext (#PCDATA|documentref|p|br|i|b|itemize|enumerate)*>
<!ELEMENT documentref (#PCDATA)>
<!ELEMENT p EMPTY>
<!ELEMENT br EMPTY>
<!ELEMENT itemize (item+)>
<!ELEMENT enumerate (item+)>
<!ELEMENT item (#PCDATA|documentref|p|br|i|b|itemize|enumerate)*>
<!ELEMENT i (#PCDATA)>
<!ELEMENT b (#PCDATA)>
<!ATTLIST documentref ref CDATA #REQUIRED>
<!ATTLIST documentref type (external|internal) internal>

```

Listing 3.1: formattedtext.dtd

Das Element `documentref` bietet die Möglichkeit, eine Referenz auf ein anderes Dokument einzufügen. Hierbei wird zwischen External-Referenzen und Internal-Referenzen unterschieden. Eine Internal-Referenz ist eine Referenz auf ein anderes Dokument des IEEE829-Standards, welches dem System daher bekannt ist. Eine solche Internal-Referenz kann beispielsweise später durch einen Hyperlink im HTML-Format ersetzt werden. Eine External-Referenz verweist auf ein standardfremdes Dokument, welches nur durch einen Dateinamen identifiziert wird, aber nicht verlinkt werden kann.

3.2.2 Testplan

DTD für einen IEEE829-Testplan⁴:

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
  Document : testplan.dtd
  Author   : Thorsten Liese
  Description:
    DTD for an IEEE829 Test Plan
-->

<!ELEMENT testplan (identifier,
  introduction,
  testitems,
  featurstobetested,
  featurstnottobetested?,
  approach,
  itepassfailcriteria,
  suspensioncriteria?,
  testdeliverables,
  testingtasks,
  environmentalneeds,
  responsibilities,
  staffingandtraining?,
  schedule,
  risksandcontingencies,
  approvals)>
<!ELEMENT identifier EMPTY>
<!ELEMENT introduction (formattedtext)>
<!ELEMENT testitems (testitem*)>
<!ELEMENT testitem (name,
  description,
  itemversion?,
  itemtransmittalinfo?,
  itemdocumentation,
  incidentreports)>
<!ELEMENT description (formattedtext)>
<!ELEMENT version (formattedtext)>
<!ELEMENT transmittalinfo (formattedtext)>
<!ELEMENT itemdocumentation (documentref*)>
<!ELEMENT incidentreports (documentref*)>
<!ELEMENT featurstobetested (featuretest*)>
<!ELEMENT featuretest (featureset,testdesignref)>
<!ELEMENT testdesignref (documentref)>
<!ELEMENT featureset (feature*)>
<!ELEMENT feature (name,description?)>
<!ELEMENT featurstnottobetested (featurenottest*)>
<!ELEMENT featurenottest (featureset,description?)>
<!ELEMENT approach (formattedtext)>
<!ELEMENT itepassfailcriteria (formattedtext)>
<!ELEMENT suspensioncriteria (formattedtext)>
<!ELEMENT testdeliverables (documentref*)>
<!ELEMENT testingtasks (testingtask*)>
```

⁴Originalname: Test Plan

```

<!ELEMENT testingtask (identifier,description)>
<!ELEMENT environmentalneeds (description,resources)>
<!ELEMENT resources (resource*)>
<!ELEMENT resource (identifier,description)>
<!ELEMENT responsibilities (formattedtext)>
<!ELEMENT staffingandtraining (formattedtext)>
<!ELEMENT schedule (description,milestones)>
<!ELEMENT milestones (milestone*)>
<!ELEMENT milestone (identifier,description)>
<!ELEMENT risksandcontingencies (formattedtext)>
<!ELEMENT approvals (formattedtext)>

<!ATTLIST identifier name CDATA #REQUIRED>

<!ENTITY % formattedtext SYSTEM "../formattedtext.dtd">
%formattedtext;

```

Listing 3.2: testplan.dtd

Anmerkungen: Das Hauptelement `testplan` enthält zunächst genau die Abschnitte, wie sie im Standard definiert sind. Abschnitte, die optional sind, sind durch ein Fragezeichen entsprechend der DTD-Syntax markiert. Die Elemente, die reinen Prosa-Text enthalten (beispielsweise `description`), sind als `formattedtext` definiert, welches einige Grundformen der Formatierung (wie Fett- und Kursivschrift) bietet.

3.2.3 Testdesignspezifikation

DTD für eine IEEE829-Testdesignspezifikation⁵:

```

<?xml version="1.0" encoding="UTF-8"?>

<!--
  Document : testdesignspecification.dtd
  Author   : Thorsten Liese
  Description:
    DTD for an IEEE829 Test Design Specification
-->

<!ELEMENT testdesignspecification (identifier,
                                   featurestobetested,
                                   approachrefine,
                                   testidentification,
                                   featurepassfailcriteria)>

<!ELEMENT identifier (testplanref?)>
<!ELEMENT testplanref (documentref)>
<!ELEMENT featurestobetested (testitem+,featuretest+)>
<!ELEMENT featuretest (featureset,additionalinfo?)>
<!ELEMENT featureset (feature+)>
<!ELEMENT feature (name,description?)>
<!ELEMENT testitem name>
<!ELEMENT name (#PCDATA)>
<!ELEMENT additionalinfo (formattedtext)>
<!ELEMENT approachrefine (formattedtext)>
<!ELEMENT testidentification (testcases,testprocedures)>
<!ELEMENT testcases (descriptedtestcaseref*)>
<!ELEMENT descriptedtestcaseref (testcaseref,description?)>
<!ELEMENT testcaseref (documentref)>
<!ELEMENT description (formattedtext)>
<!ELEMENT testprocedures (descriptedprocedureref*)>
<!ELEMENT descriptedprocedureref (procedureref,description?)>
<!ELEMENT procedureref (documentref)>
<!ELEMENT featurepassfailcriteria (formattedtext)>

```

⁵Originalname: Test Design Specification

```
<!ATTLIST identifier name CDATA #REQUIRED>
<!ENTITY % formattedtext SYSTEM "../formattedtext.dtd">
%formattedtext;
```

Listing 3.3: testdesignspecification.dtd

3.2.4 Testfallspezifikation

DTD für eine IEEE829-Testfallspezifikation⁶:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Document : testcasespecification.dtd
  Author   : Thorsten Liese
  Description:
    DTD for an IEEE829 Test Case Specification
-->
<!ELEMENT testcasespecification (identifier,
    testitems,
    inputspecification,
    outputspecification,
    environmentalneeds,
    specialproceduralrequirements?,
    intercasedependencies?)>
<!ELEMENT identifier EMPTY>
<!ELEMENT testitems (testitem+)>
<!ELEMENT testitem (name,
    description,
    itemdocumentation)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT itemdocumentation (documentref*)>
<!ELEMENT inputspecification (setting*)>
<!ELEMENT outputspecification (namedvalue)>
<!ELEMENT environmentalneeds (hardware,software,other)>
<!ELEMENT hardware (setting*)>
<!ELEMENT software (setting*)>
<!ELEMENT other (setting*)>
<!ELEMENT setting (#PCDATA)>
<!ELEMENT specialproceduralrequirements (formattedtext)>
<!ELEMENT intercasedependencies (formattedtext)>
<!ATTLIST setting name CDATA #REQUIRED>
<!ATTLIST identifier name CDATA #REQUIRED>
<!ENTITY % formattedtext SYSTEM "../formattedtext.dtd">
%formattedtext;
```

Listing 3.4: testcasespecification.dtd

3.2.5 Testablaufspezifikation

DTD für eine IEEE829-Testablaufspezifikation⁷:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Document : testprocedurespecification.dtd
  Author   : Thorsten Liese
  Description:
```

⁶Originalname: Test Case Specification

⁷Originalname: Test Procedure Specification

```

        DTD for an IEEE829 Test Procedure Specification
-->
<!ELEMENT testprocedurespecification (identifier,
        purpose,
        specialrequirements?,
        proceduresteps)
<!ELEMENT identifier (testdesignspecificationref?)>
<!ELEMENT testdesignspecificationref (documentref)>
<!ELEMENT purpose (formattedtext,testcasepecificationref*)>
<!ELEMENT testcasepecificationref (documentref)>
<!ELEMENT specialrequirements (formattedtext)>
<!ELEMENT proceduresteps (log,setup,start,proceed,measure,shutdown,
        restart,stop,wrapup,contingencies)>
<!ELEMENT log (formattedtext)>
<!ELEMENT setup (formattedtext)>
<!ELEMENT start (formattedtext)>
<!ELEMENT proceed (formattedtext)>
<!ELEMENT measure (formattedtext)>
<!ELEMENT shutdown (formattedtext)>
<!ELEMENT restart (formattedtext)>
<!ELEMENT stop (formattedtext)>
<!ELEMENT wrapup (formattedtext)>
<!ELEMENT contingencies (formattedtext)>

<!ATTLIST identifier name CDATA #REQUIRED>

<!ENTITY % formattedtext SYSTEM "../formattedtext.dtd">
%formattedtext;

```

Listing 3.5: testprocedurespecification.dtd

3.2.6 Testgegenstandsauslieferungsbericht

DTD für einen IEEE829-Testgegenstandsauslieferungsbericht⁸:

```

<?xml version="1.0" encoding="UTF-8"?>
<!--
    Document : testitemtransmittalreport.dtd
    Author   : Thorsten Liese
    Description:
        DTD for an IEEE829 Test Item Transmittal Report
-->
<!ELEMENT testitemtransmittalreport (identifier,
        transmitteditems,
        location,
        status,
        approvals)
<!ELEMENT identifier EMPTY>
<!ELEMENT transmitteditems (testitems,testplanref)>
<!ELEMENT testplanref (documentref)>
<!ELEMENT location (formattedtext)>
<!ELEMENT status (formattedtext,incidentreportref*)>
<!ELEMENT incidentreportref (documentref)>
<!ELEMENT approvals (person*,signaturespace)>
<!ELEMENT signaturespace EMPTY>
<!ELEMENT person (name,title,postion)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT position (#PCDATA)>

<!ELEMENT testitems (testitem+)>
<!ELEMENT testitem (name,
        description,
        version,
        itemdocumentation,
        person)>

```

⁸Originalname: Test Item Transmittal Report

```
<!ATTLIST identifier name CDATA #REQUIRED>
<!ENTITY % formattedtext SYSTEM "../formattedtext.dtd">
%formattedtext;
```

Listing 3.6: testitemtransmittalreport.dtd

3.2.7 Testprotokoll

DTD für ein IEEE829-Testprotokoll⁹:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Document : testlog.dtd
  Author   : Thorsten Liese
  Description:
    DTD for an IEEE829 Test Log
-->
<!ELEMENT testlog (identifier,
                  description,
                  activityandevententries)>
<!ELEMENT identifier EMPTY>
<!ATTLIST identifier name CDATA #REQUIRED>
<!ELEMENT description (formattedtext,testitems,environment)>
<!ELEMENT testitems (testitem+)>
<!ELEMENT testitem (name,
                   version,
                   transmittalreportref?)>
<!ELEMENT transmittalreportref (documentref)>
<!ELEMENT environment (formattedtext)>
<!ELEMENT activityandevententries (logentry*)>
<!ELEMENT logentry (executiondescription,
                  procedureresults,
                  environmentalinformation?,
                  anomalousevents?,
                  incidentreports?)>
<!ELEMENT executiondescription (testprocedureref,person+)>
<!ELEMENT testprocedureref (documentref)>
<!ELEMENT person (name,function)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT function (#PCDATA)>
<!ELEMENT procedureresults (formattedtext)>
<!ELEMENT environmentalinformation (formattedtext)>
<!ELEMENT anomalousevents (formattedtext)>
<!ELEMENT incidentreports (documentref*)>
<!ATTLIST logentry timemillis CDATA #REQUIRED>
<!ATTLIST logentry author CDATA #REQUIRED>
<!ATTLIST procedureresults success (false|true) #REQUIRED>
<!ENTITY % formattedtext SYSTEM "../formattedtext.dtd">
%formattedtext;
```

Listing 3.7: testlog.dtd

⁹Originalname: Test Log

3.2.8 Testvorfallbericht

DTD für einen IEEE829-Testvorfallbericht¹⁰:

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
  Document : testincidentreport.dtd
  Author   : Thorsten Liese
  Description:
    DTD for an IEEE829 Test Incident Report
-->

<!ELEMENT testincidentreport (identifier,
                               summary,
                               incidentdescription,
                               impact)>

<!ELEMENT identifier EMPTY>
<!-- ATTLLIST identifier name CDATA #REQUIRED -->

<!ELEMENT summary (description, testitems, testprocedureref, testcaseref, testlogref)>
<!ELEMENT testitems (testitem+)>
<!ELEMENT testitem (name, version)>
<!ELEMENT testprocedureref (documentref)>
<!ELEMENT testcaseref (documentref)>
<!ELEMENT testlogref (documentref)>
<!ELEMENT description (formattedtext)>

<!ELEMENT incidentdescription (formattedtext)>
<!ELEMENT impact (formattedtext)>

<!ENTITY % formattedtext SYSTEM "../formattedtext.dtd">
%formattedtext;
```

Listing 3.8: testincidentreport.dtd

3.2.9 Testergebnisbericht

DTD für einen IEEE829-Testergebnisbericht¹¹:

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
  Document : testsummaryreport.dtd
  Author   : Thorsten Liese
  Description:
    DTD for an IEEE829 Test Summary Report
-->

<!ELEMENT testsummaryreport (identifer,
                              summary,
                              variances?,
                              comprehensivenessassessment,
                              summaryofresults,
                              evaluation,
                              summaryofactivities,
                              approvals)>

<!ELEMENT identifier EMPTY>
<!-- ATTLLIST identifier name CDATA #REQUIRED -->
<!ELEMENT testitems (testitem+)>
<!ELEMENT testitem (name,
                    version,
                    testplanref,
                    testdesignrefs,
                    testprocedurerefs,
```

¹⁰Originalname: Test Incident Report

¹¹Originalname: Test Summary Report

```

        testitemtransmittalrefs,
        testlogrefs,
        testincidentrefs)>
<!ELEMENT testplanref (documentref?)>
<!ELEMENT testdesignrefs (documentref*)>
<!ELEMENT testprocedurerefs (documentref*)>
<!ELEMENT testitemtransmittalrefs (documentref*)>
<!ELEMENT testlogrefs (documentref*)>
<!ELEMENT testincidentrefs (documentref*)>

<!ELEMENT summary (formattedtext, testitems)>
<!ELEMENT variances (formattedtext)>
<!ELEMENT comprehensivenessassessment (formattedtext,featureset*)>
<!ELEMENT featureset (feature+)>
<!ELEMENT feature name>

<!ELEMENT summaryofresults (formattedtext,solvedincidents,unsolvedincidents)>
<!ELEMENT solvedincidents (testincidentref*)>
<!ELEMENT unsolvedincidents (testincidentref*)>
<!ELEMENT testincidentref (documentref)>

<!ELEMENT evaluation (formattedtext)>
<!ELEMENT summaryofactivities (formattedtext)>

<!ELEMENT approvals (person*,signaturespace)>
<!ELEMENT person (name,title,postion)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT position (#PCDATA)>
<!ELEMENT signaturespace EMPTY>

<!ENTITY % formattedtext SYSTEM "../formattedtext.dtd">
%formattedtext;

```

Listing 3.9: testsummaryreport.dtd

3.2.10 Erweiterung: Testgegenstand

Für die Definition von Testgegenständen wurde eine Erweiterung eingeführt, die es erlaubt, Testgegenstände gesondert von den IEEE829-Standarddokumenten zu erstellen. Da diese Testgegenstände auch gesondert gespeichert werden müssen, um sie später wieder in die Anwendung laden zu können, wurde für einen Testgegenstand die folgende DTD definiert.

```

<?xml version="1.0" encoding="UTF-8"?>

<!--
  Document : testitem.dtd
  Author   : Thorsten Liese
  Description:
    DTD for a test item specification document
-->

<!ELEMENT testitem (identifier,
                    description,
                    itemdocumentation,
                    features)>

<!ELEMENT identifier EMPTY>
<!ATTLIST identifier name CDATA #REQUIRED>
<!ELEMENT description (formattedtext)>
<!ELEMENT itemdocumentation (documentref*)>
<!ELEMENT features (feature*)>
<!ELEMENT feature (identifier,description?)>

<!ENTITY % formattedtext SYSTEM "../formattedtext.dtd">
%formattedtext;

```

Listing 3.10: testitem.dtd

3.3 Automatisierte Dokumenterstellung

Es muss nun ein System entwickelt werden, das es erlaubt, die verschiedenen Dokumente des Standards mit Hilfe einer Software zu erstellen. Ein möglicher Ansatz wäre, die Software starr für die vorgegebenen Dokumente zu entwerfen. Die Strukturen der einzelnen Dokumente wären dann fest in der Software verankert. Dies hat folgende Nachteile:

- Jeder Dokumenttyp mit all seinen Abschnitten und Unterabschnitten müsste im Software-Code nachgebildet werden. Dies allein ist schon mit größerem Aufwand verbunden.
- Anpassungen der Dokumenttypen (also der DTDs) - beispielsweise durch Hinzufügen von Abschnitten oder Unterabschnitten, wie es nach dem IEEE829-Standard ausdrücklich erlaubt ist - würde eine Änderung der Software erforderlich machen.
- Die Software wäre nicht mehr universal für beliebige Dokumente einsetzbar.

Ein besserer Ansatz ist es, die Dokumente mit Hilfe von XML-Konfigurationsdateien zu definieren. Eine solche Lösung soll nun erarbeitet werden.

3.3.1 Allgemeine Dokumentstruktur

Allgemein betrachtet besteht jedes Dokument aus einer Menge von Abschnitten (engl.: *sections*). Jeder Abschnitt kann wiederum aus einer Menge von Unterabschnitten und diese wiederum aus einer Menge von Unterunterabschnitten und so weiter bestehen. Jeder Abschnitt hat einen eindeutigen Namen als Attribut. Eine Beschreibungsdatei für ein Dokument sieht demnach beispielsweise so aus:

```
<section name='Mein Dokument'>
  <section name='Einleitung'>
  </section>
  <section name='Hauptteil'>
    <section name='Unterabschnitt 1'>
    </section>
    <section name='Unterabschnitt 2'>
```

```

        </section>
    </section>
</section>

```

Eine Software kann nun diese Beschreibungsdatei einlesen und den Anwender zu Erstellung eines solchen Dokuments auffordern. Dazu können die Abschnitte und Unterabschnitte übersichtlich in einem Baum dargestellt werden. Der Anwender kann durch Auswahl eines bestimmten Abschnitts im Baum den dazugehörigen Inhalt eingeben.

Nun muss noch eine Editiermöglichkeit für jeden Abschnitt eingeführt werden. Schließlich will der Anwender ja nicht die Inhalte der einzelnen Abschnitte per Hand in ein schlichtes Textfeld eingeben, wenn die Software in dabei unterstützen kann. Zu diesem Zweck wird zu jedem Abschnitt ein Editor eingeführt, d.h. man definiert zu jeder *section* einen so genannten *editor*, der speziell zur Erstellung eines Abschnitts geeignet ist. Ein Editor kann dabei eine beliebige Softwarekomponente darstellen, wie später gezeigt wird. Die Verwendung von Editoren soll nun anhand eines Beispielles verdeutlicht werden.

Angenommen, es soll ein Dokument erstellt werden, welches folgender DTD genügt:

```

<!ELEMENT simpledocument (title,authors,body)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT authors (author+)>
<!ELEMENT author (name,title)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT body (formattedtext)>

```

Zur Erstellung eines solchen Dokuments produziert man die folgende Beschreibungsdatei. Man beachte, dass zusätzlich zum Attribut *name* nun auch ein Attribut *editor* eingeführt wurde.

```

<section name='simpledocument' editor='DocumentEditor'>
    <section name='title' editor='TitleEditor'>
    </section>
    <section name='authors' editor='AuthorsEditor'>
    </section>
    <section name='body' editor='FormattedTextEditor'>
    </section>
</section>

```

Die verschiedenen Editoren sind speziell geeignet, die jeweiligen Inhalte zu erzeugen. Beispielsweise könnte der `AuthorsEditor` dem Anwender spezielle Eingabefelder für die Eingabe der Autoren-Informationen wie Name und Titel anbieten, oder es könnte einfach eine Liste bekannter Autoren zur Auswahl angezeigt werden. Der `TitleEditor` stellt dem Anwender spezielle Eingabe- und Formatierungsmöglichkeiten für Titel zur Verfügung, und der `FormattedTextEditor` dient der Eingabe von formatierten Textblöcken.

3.3.2 DTD

Der Vollständigkeit halber soll hier auch die DTD für die oben hergeleitete Beschreibungsdatei für Dokumente vorgestellt werden.

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
  Document : autodoc.dtd
  Author   : Thorsten Liese
  Description:
    Defines the DTD for XML files which define
    documents for automatic creation in an application
-->

<!ELEMENT section (property*,section*)>
<!ELEMENT property (#PCDATA)>

<!ATTLIST section name CDATA #REQUIRED>
<!ATTLIST section editor CDATA #REQUIRED>
<!ATTLIST section icon (root|section|document|documentgroup) "section">
<!ATTLIST section resourcebundle CDATA #IMPLIED>

<!ATTLIST property name CDATA #REQUIRED>
```

Listing 3.11: autodoc.dtd

Anmerkungen: Zusätzlich zu den bisher erwähnten Bestandteilen können die Section-Elemente so genannte Property-Tags enthalten, über die sich beliebige Eigenschaften eines Editors konfigurieren lassen. Diese Konfiguration macht es möglich, die selben Editoren für mehrere Abschnitte zu verwenden, so dass geringfügige Abweichungen nicht das Programmieren eines eigenen Editors erforderlich machen.

Das Attribut *icon* erlaubt die Auswahl eines von vier Bildchen, das im Dokumentenbaum für den entsprechenden Abschnitt dargestellt wird. Das Attribut *resourcebundle* gibt den Namen eines *PropertyResourceBundle*¹² (also einer Menge von `.properties`-Dateien) an, das die (eventuell mehrsprachigen) Texte für die Darstellung der Abschnitte und die Hilfetexte in der Applikation enthält. Jeder Editor kann auf diese Art und Weise ein eigenes *ResourceBundle* besitzen. Wird das Attribut *resourcebundle* nicht angegeben, so wird automatisch das *ResourceBundle* der nächsten übergeordneten *section* verwendet, das ein

¹²siehe `java.util.PropertyResourceBundle` [J-API]

ResourceBundle definiert. Das Attribut *resourcebundle* vererbt sich also gewissermaßen auf alle Unterabschnitte, wenn diese kein eigenes ResourceBundle definieren.

Auf dieser Grundlage soll nun im folgenden Kapitel die Software vorgestellt werden.

Kapitel 4

Die Software

4.1 Einführung

Die erstellte Software soll zeigen, wie eine automatisierte Erstellung von IEEE829-Testdokumentationen prinzipiell aussehen kann. Im Rahmen dieser Diplomarbeit kann die Software selbstverständlich nur als Prototyp implementiert werden, der noch viel Spielraum für zukünftige Erweiterungen und Ergänzungen lässt. Ich habe vor, die Entwicklung der Software auch nach Abgabe dieser Diplomarbeit fortzusetzen.

Die Software wurde vollständig in der Programmiersprache Java [JAVA] entwickelt, um die Plattformunabhängigkeit und die hervorragenden XML-Fähigkeiten dieser Sprache auszunutzen. Im Einzelnen wurden folgende Werkzeuge zur Erstellung der Software eingesetzt:

- Java 2 Software Development Kit Standard Edition (Java 2 SE) [J2SE]
- Together 6.0 zur UML-Softwaremodellierung [TOGETHER]
- Netbeans IDE 3.4 als Entwicklungsumgebung [NETBEANS]
- CVS für die Versionsverwaltung [CVS]
- Jakarta-Ant 1.5.1 als Build-Tool. [ANT]

In der Abbildung 4.1 auf Seite 41 ist die Oberfläche der Applikation zu sehen. Die Oberfläche enthält vier Hauptbestandteile. Links ist der Dokumentenbaum zu sehen. Jeder Eintrag in diesem Baum repräsentiert einen Abschnitt, also eine *section*, wie sie in einer Dokumenten-Beschreibungsdatei (siehe Abschnitt 3.3.1

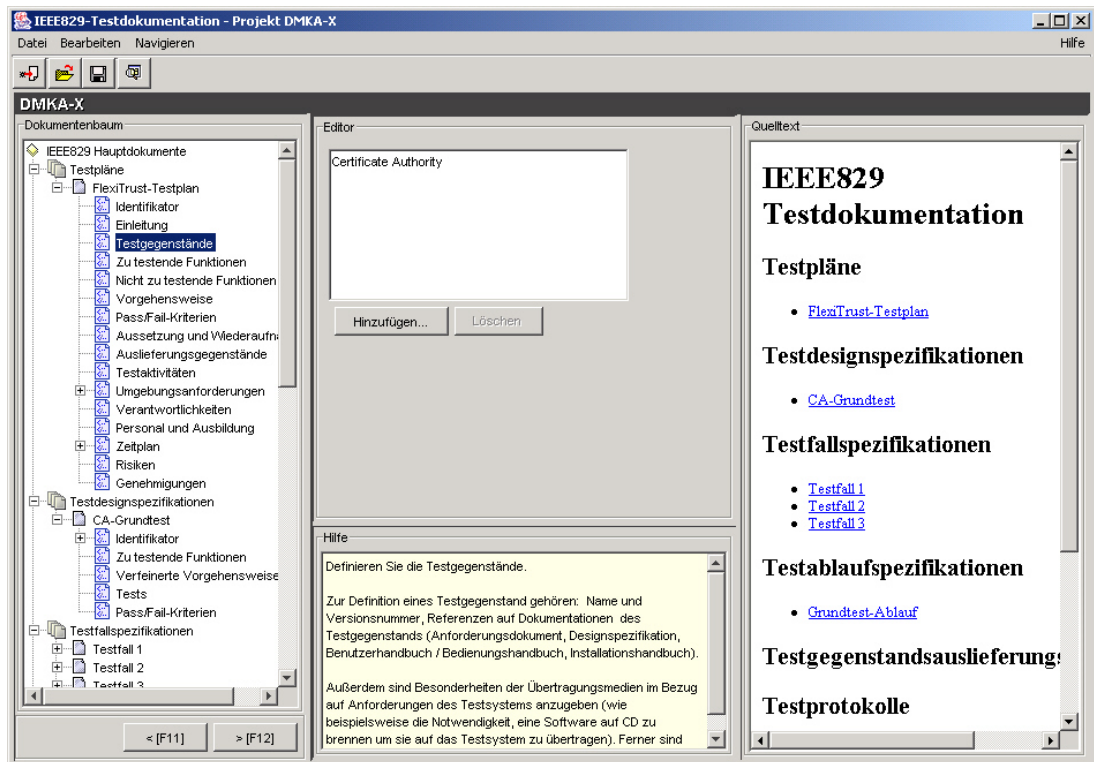


Abbildung 4.1: Screenshot der Software

auf Seite 36) definiert wurde. In der Mitte ist der jeweilige Editor zu sehen, der zu dem im Baum ausgewählten Abschnitt gehört. Darunter befindet sich ein Hilfenfenster mit Informationen zum ausgewählten Abschnitt. Am rechten Bildrand befindet sich ein Web-Browser¹, der entweder den XML-Quellentext des gewählten Abschnitts zeigt oder zur Ansicht des HTML-Exports der Dokumente dient.

Mit Hilfe der F11- und F12-Taste (oder mit den beiden Schaltknöpfen unterhalb des Dokumentenbaums) kann auf einfache Weise durch die Abschnitte des Baum vor und zurück navigiert werden. Der Tastaturfokus landet dabei stets auf dem primären Eingabefeld des Editors, so dass eine schnelle Eingabe der Dokumentinhalte per Tastatur möglich ist.

¹Die Anzeige des Web-Browsers ist derzeit nur auf Windows-Rechnern möglich. Die Software bindet dazu den Microsoft Internet Explorer in die Oberfläche ein. Für die korrekte Darstellung der XML- und XSL-Inhalte wird die Installation des Internet Explorers Version 6.0 oder höher empfohlen.

4.2 Modellierung

4.2.1 Verzeichnisstruktur

Die Dateien der Applikation sind in der folgenden Verzeichnisstruktur angelegt:

<i>Verzeichnis</i>	<i>Inhalt</i>
app	Wurzelverzeichnis der Applikation. Enthält die Ordner lib und src.
app/lib	Enthält externe Bibliotheken, die zum Übersetzen und Starten der Anwendung erforderlich sind.
app/src	Wurzelverzeichnis der Java-Quellen. In diesem Verzeichnis befindet sich auch das zentrale Build-File build.xml zum Übersetzen und Starten der Applikation.
app/src/images	Enthält Bilder und Icons
app/src/lang	Enthält die Sprachdateien
app/src/xml	Enthält die XML-Dokumentbeschreibungsdateien
app/src/xml/dtd	Enthält die allgemeinen DTDs für Dokumentbeschreibungsdateien und für formatierten Text etc.
app/src/xml/dtd/ieee829	Enthält die DTDs der Dokumente des IEEE829-Standards.
app/src/xml/dtd/ext	Enthält die DTDs für erweiterte Dokumenttypen (Testgegenstände)
app/src/xml/xsl	Enthält die eXtended-Stylesheet-Language-Dateien [XSL] zur Konvertierung der Ausgabedokumente nach HTML.
app/src/ieee829	Basispackage der Java-Klassen. Enthält nur den Ordner app.
app/src/ieee829/app	Enthält die startbare Klassen Main.java. Alle weiteren Klassen sind in den Unter-Packages core, editors und ext untergebracht.
app/src/ieee829/app/core	Enthält die Klassen der Kernapplikation in den Unter-Packages model, view und control.

<code>app/src/ieee829/app/core/model</code>	Enthält alle Klassen der Datenhaltung und Datenmanipulation.
<code>app/src/ieee829/app/core/view</code>	Enthält alle Klassen der Swing-Oberflächen.
<code>app/src/ieee829/app/core/control</code>	Enthält alle Klassen, die die eigentlich Programmlogik implementieren.
<code>app/src/ieee829/app/editors</code>	Enthält den abstrakten Basis-Editor <code>Editor.java</code> und alle allgemein verwendbaren Implementierungen von Editoren.
<code>app/src/ieee829/app/editors/ieee829</code>	Enthält Editoren, die spezifisch für die IEEE829-Dokumente sind.
<code>app/src/ieee829/ext</code>	Enthält Erweiterungen der Applikation
<code>app/src/ieee829/ext/testitems</code>	Enthält die Erweiterungs-Editoren für die gesonderte Definition von Testgegenständen.

4.2.2 Package `ieee829.app.core`

Dieses Package enthält die Kernbestandteile der Anwendung. Diese sind nach *Model*, *View* und *Control*² getrennt, also nach Datenmodell, Anzeige und Steuerung. Die Klassen im `control`-Package implementieren demnach die Programmlogik und -steuerung, die Klassen im `model`-Package implementieren die Datenhaltung und die Klassen im `view`-Package implementieren die Java-Benutzeroberfläche. Es sollen hier nur ausgewählte Basisklassen vorgestellt werden, die zum Verständnis der Applikation beitragen. In der Grafik 4.2 sind diese ausgewählten Klassen in ihren Packages dargestellt. Die einzelnen Klassen werden im Folgenden vorgestellt.

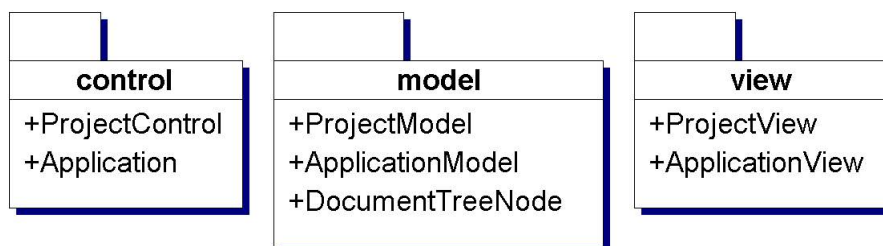
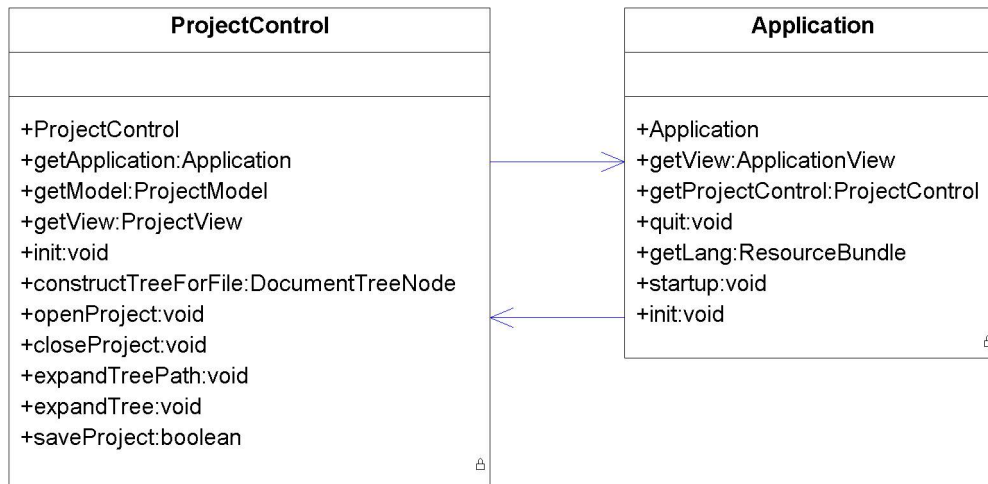


Abbildung 4.2: Basisklassen

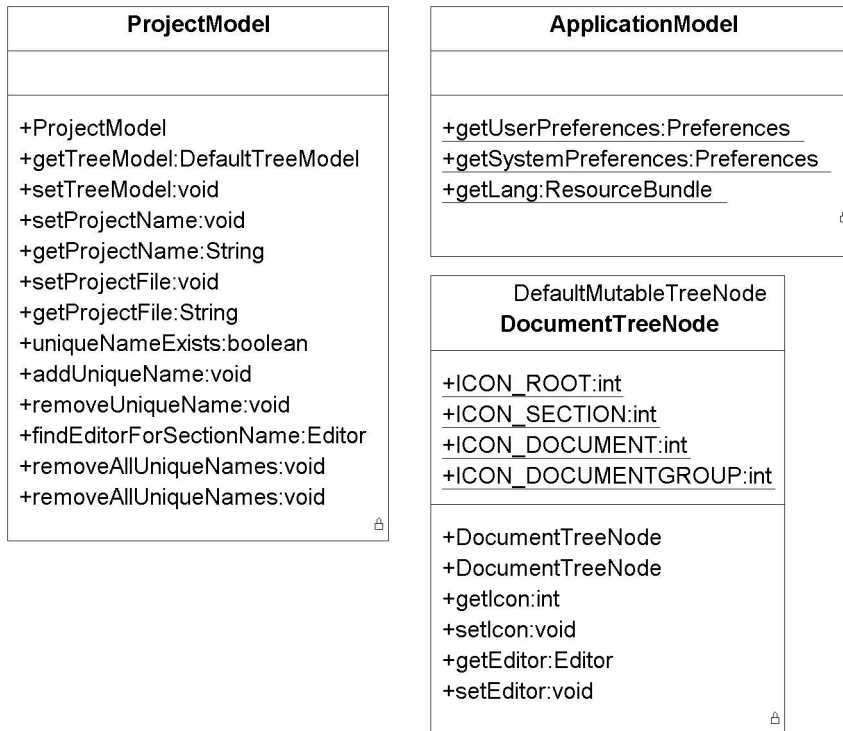
²in Anlehnung an das bekannte MVC-Paradigma (*Model-View-Controller*), das zum Zwecke der höheren Flexibilität und Wiederverwendbarkeit von Software die Trennung von Datenmodell (Model), Bildschirmrepräsentation (View) und Steuerung (Control) vorsieht

4.2.2.1 Package `ieee829.app.core.control`Abbildung 4.3: Package `ieee829.app.core.control`**Klasse `ieee829.app.core.control.Application`**

Die Klasse `Application` implementiert die allgemeine Programmlogik der Anwendung. Zur allgemeinen Programmlogik gehören das Starten, Initialisieren und Beenden der Applikation. Sie übernimmt auch das Laden und Speichern von Benutzereinstellungen. Derzeit kann die Applikation nur ein Projekt zur selben Zeit öffnen, weshalb die Klasse `Application` genau ein `ProjectControl` enthält.

Klasse `ieee829.app.core.control.ProjectControl`

Die Klasse `ProjectControl` enthält sämtliche Logik, die in Zusammenhang mit einem Projekt steht. Dazu gehören das Laden und Speichern von Projekten, das Öffnen und Einlesen der Dokumentbeschreibungsdateien und das Anlegen neuer Dokumente.

4.2.2.2 Package `ieee829.app.core.model`Abbildung 4.4: Package `ieee829.app.core.model`**Klasse `ieee829.app.core.model.ApplicationModel`**

Die Klasse `ApplicationModel` beinhaltet die Benutzer- und Systemeinstellungen und die Textressourcen der Applikation.

Klasse `ieee829.app.core.model.ProjectModel`

Diese zentrale Klasse enthält alle Daten eines Projekts. Dazu gehören das Tree-Model für den JTree und die Speicherung von eindeutigen Dokumentbezeichnern sowie Methoden zur Datensuche.

Klasse `ieee829.app.core.model.DocumentTreeNode`

Diese Erweiterung eines `DefaultMutableTreeNode` (siehe `javax.swing.tree.DefaultMutableTreeNode` [J-API]) stellt einen Abschnitt eines Dokuments dar und enthält neben einem Icon-Typ auch eine Referenz auf den jeweiligen Editor, der diesem Abschnitt zugewiesen ist. Bei Selektion eines Tree-Node im Baum kann die Anwendung durch Aufruf von `getEditor()` eine Referenz auf den entsprechenden Editor erhalten und diesen darstellen.

4.2.2.3 Package `ieee829.app.core.view`

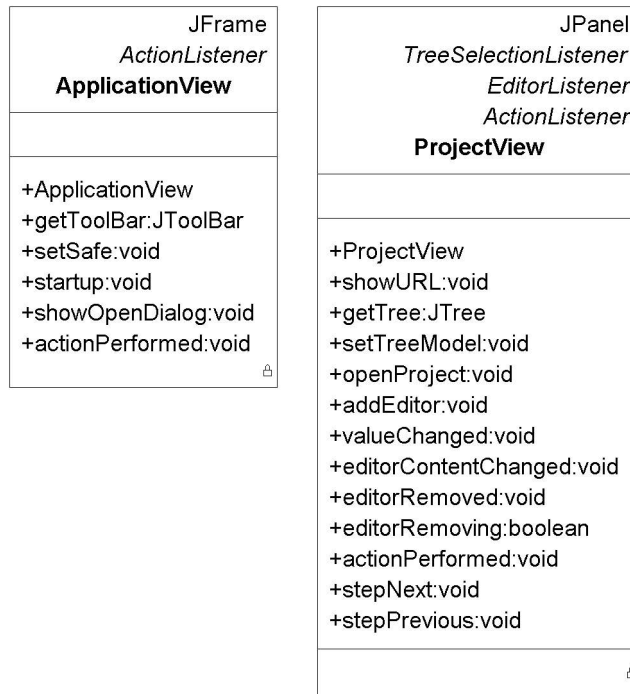


Abbildung 4.5: Package `ieee829.app.core.view`

Klasse `ieee829.app.core.view.ApplicationView`

Diese Klasse implementiert das Applikationsfenster, die Werkzeugleiste und das Hauptmenü.

Klasse `ieee829.app.core.view.ProjectView`

Die Klasse `ProjectView` implementiert die Benutzeroberfläche für die Bearbeitung eines Projekts. Dazu gehört der Dokumentenbaum und die Ansteuerung der Editorkomponenten, der Webbrowser und das Hilfefenster (siehe Abbildung 4.1 auf Seite 41).

4.2.3 Package `ieee829.app.editors`

Dieses Package enthält die Editoren, die für die Bearbeitung von Dokumentabschnitten zur Verfügung stehen. In diesem Package sind nur allgemein verwendbare Editoren enthalten, die unabhängig von den speziellen IEEE829-Testdokumenten sind. Diese Trennung erlaubt einen vielfältigen Einsatz der Software auch für andere Dokumenttypen. Die speziellen Editoren, die für

die Bearbeitung der IEEE829-Dokumente entwickelt wurden, befinden sich im Subpackage `ieee829`.

4.2.3.1 Klasse `ieee829.app.editors.Editor`

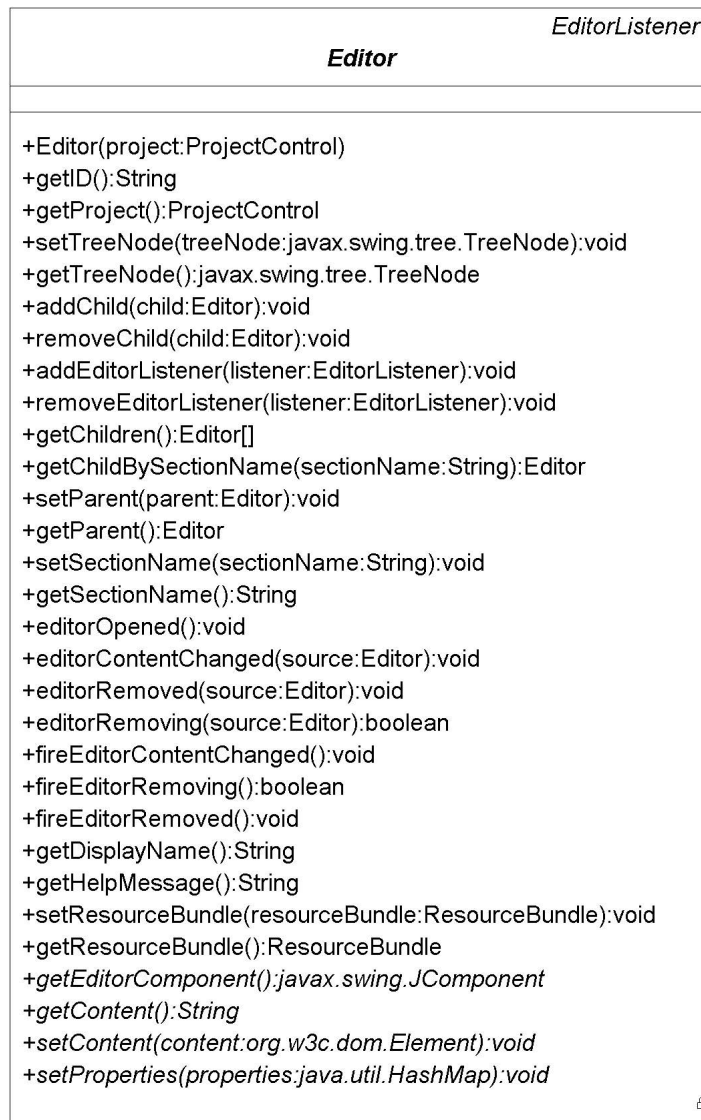


Abbildung 4.6: Klasse `ieee829.app.editors.Editor`

Das Package `editors` enthält die für spätere Erweiterungen wichtigste Klasse `Editor`. Es handelt sich um eine abstrakte Klasse. Alle Implementierungen eines Editors (wie sie in den Dokumentbeschreibungsdateien angegeben wer-

den, siehe Abschnitt 3.3.1 auf Seite 36) müssen von dieser Klasse erben und die als abstrakt definierten Methoden implementieren.

Eine mögliche Implementierung der abstrakten Methoden bietet die Klasse `BasicEditor`, von der alle anderen Editoren in diesem Package erben. Einen Überblick über die Klassen des Packages gibt die Abbildung 4.7 auf Seite 49.

Da die Klasse `Editor` von großer Wichtigkeit ist, soll sie nun genauer vorgestellt werden. Sie hat die in der Abbildung 4.6 auf Seite 47 dargestellten Methoden, von denen die untersten vier (`getContent():String`, `setContent(org.w3c.dom.Element)`, `getEditorComponent():javax.swing.JComponent` und `setProperties(java.util.HashMap)`) abstrakte Methoden sind. Diese sind von Subklassen entsprechend zu implementieren.

Die Methode `getContent():String` sollte ein *well-formed*³ XML-Fragment erzeugen - je nach Inhalt des Editors. Die Methode `setContent(org.w3c.dom.Element)` ist das Gegenstück dazu und sollte die Editor-Inhalte entsprechend dem übergebenen XML-Fragment zurücksetzen. Dieser Vorgang ist beim Laden von abgespeicherten Dokumenten in die Anwendung erforderlich. Der Grund für die auf den ersten Blick merkwürdig erscheinende Tatsache, dass einmal ein *String* und einmal ein *org.w3c.dom.Element* übergeben wird, ist der folgende:

Bei der Erstellung des Rückgabewertes für die `getContent()`-Methode ist die Verwendung des XML-APIs ungeeignet, da einzelne XML-Elemente nicht unabhängig von einem Dokument erstellt werden können. Eine Erzeugung der Inhalte als *String* ist hier wesentlich praktikabler. Für die Übergabe des XML-Inhalts in der `setContent(...)`-Methode allerdings wäre ein *String* unpraktikabel, weil hier Teile des Dokuments mitunter mehrfach und in jedem Abschnitt geparkt werden müssten. Durch Übergabe eines *org.w3c.dom.Element* kann das Dokument einmalig am Anfang geparkt werden und an die entsprechenden Unterabschnitte aufgeteilt werden.

Über die Methode `setProperties(java.util.HashMap)` werden dem Editor die in der Dokumentbeschreibungsdatei definierten Eigenschaften übergeben. Es ist zu beachten, dass diese Methode nur aufgerufen wird, wenn mindestens eine Eigenschaft definiert ist.

Die Methode `getEditorComponent():javax.swing.JComponent` sollte die Benutzerschnittstelle des Editors zurückgeben. Der Rückgabewert darf auch *null* sein.

³Ein wohlgeformtes (*well-formed*) XML-Dokument muss syntaktisch korrekt sein (siehe [XML]).

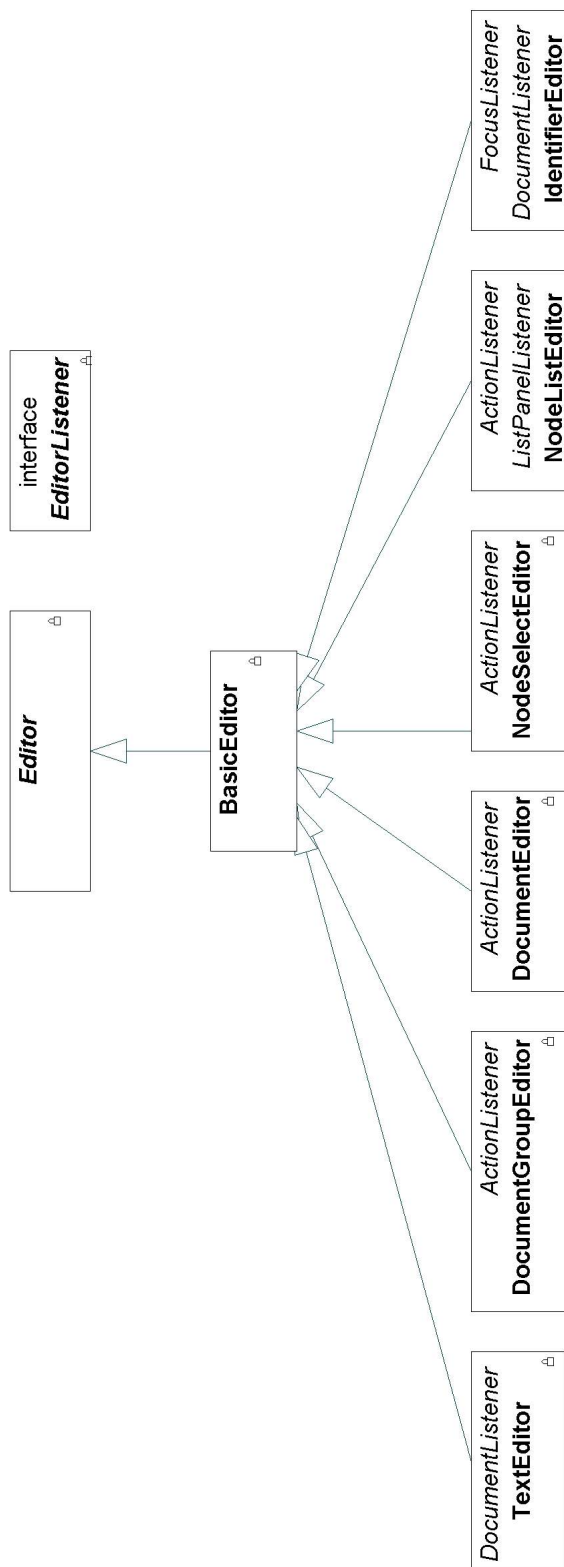


Abbildung 4.7: Package ieee829.app.editors

Die Klasse `Editor` ist folgendermaßen implementiert:

```

/*
 * $Id: Editor.java,v 1.8 2003/04/09 01:32:12 liese Exp $
 * @author: Thorsten Liese - thorsten.liese@dreicsystems.de
 *
 */
package ieee829.app.editors;

import com.dreic.utils.Lib;
import ieee829.app.core.control.ProjectControl;
import java.util.ResourceBundle;

/**
 * Abstract definition for a page editor. A page editor
 * has a gui component for editing a certain part of a document
 * and it provides method for getting and setting the XML content.
 */
public abstract class Editor implements EditorListener
{
    /**
     * Static counter counts editor objects
     */
    private static int idCounter = 0;

    /**
     * Holds my id
     */
    private String id = null;

    /**
     * Holds this editor's tree node
     */
    private javax.swing.tree.TreeNode treeNode = null;

    /**
     * Holds my children editors
     */
    private java.util.ArrayList children = null;

    /**
     * Holds my parent editor
     */
    private Editor parent = null;

    /**
     * Holds the section name
     */
    private String sectionName = null;

    /**
     * Holds my listeners
     */
    private java.util.ArrayList listeners = null;

    /**
     * Hold the current project
     */
    private ProjectControl project = null;

    /**
     * Holds the resource bundle
     */
    private ResourceBundle resourceBundle = null;

    /**
     * Creates a new editor for the given project
     * @param project the project
     */
    public Editor(ProjectControl project)
    {
        this.project = project;
        this.id = "editor"+idCounter;
        idCounter++;

        this.children = new java.util.ArrayList();
        this.listeners = new java.util.ArrayList();
    }
}

```

```
/**
 * Get a unique identifier for this editor object
 */
public final String getID()
{
    return id;
}

/**
 * Get this editor's project
 */
public ProjectControl getProject()
{
    return project;
}

/**
 * Sets this editor's tree node
 */
public void setTreeNode(javax.swing.tree.TreeNode treeNode)
{
    this.treeNode = treeNode;
}

/**
 * Gets this editor's tree node
 */
public javax.swing.tree.TreeNode getTreeNode()
{
    return treeNode;
}

/**
 * Adds a child editor to this editor. We register ourselves as a listener
 * in the child editor so we can refire content change events when the
 * content of a child changes.
 * This also fires a content change event since a new child means new content.
 *
 * @param child an editor
 */
public void addChild(Editor child)
{
    children.add(child);
    child.addEditorListener(this);
    fireEditorContentChanged();
}

/**
 * Removes a child editor from this editor.
 * This fires a content change event since a removed child means new content.
 *
 * @param child an editor
 */
public void removeChild(Editor child)
{
    children.remove(child);
    child.removeEditorListener(this);
    fireEditorContentChanged();
}

/**
 * Registers a listener to this editor.
 * @param listener an EditorListener
 */
public void addEditorListener(EditorListener listener)
{
    listeners.add(listener);
}

/**
 * Removes a listener from this editor.
 * @param listener an EditorListener
 */
public void removeEditorListener(EditorListener listener)
{
    listeners.remove(listener);
}
```

```

/**
 * Get the child editors for this editor
 */
public Editor[] getChildren()
{
    Editor[] eds = new Editor[children.size()];
    for (int i=0; i<eds.length; i++)
        eds[i] = (Editor)children.get(i);

    return eds;
}

/**
 * Convenience method for retrieving a child editor by its section name.
 * @param sectionName a section name
 */
public Editor getChildBySectionName(String sectionName)
{
    for (int i=0; i<children.size(); i++)
    {
        Editor child = (Editor)children.get(i);
        if (child.getSectionName().equals(sectionName)) return child;
    }
    return null;
}

/**
 * Sets the parent editor for this editor
 * @param parent an editor
 */
public void setParent(Editor parent)
{
    this.parent = parent;
}

/**
 * Gets the parent editor for this editor
 */
public Editor getParent()
{
    return parent;
}

/**
 * Sets this editor's section name
 * @param parent an editor
 */
public void setSectionName(String sectionName)
{
    this.sectionName = sectionName;
}

/**
 * Gets this editor's section name
 */
public String getSectionName()
{
    return sectionName;
}

/**
 * Invoked when the editor was made visible thru a selection change in
 * the document tree. The default implementation does not do anything.
 * Override when needed.
 */
public void editorOpened()
{
}

/**
 * Invoked when the content of a child editor has changed. This refires
 * the event to our listeners.
 */
public void editorContentChanged(Editor source)
{
    fireEditorContentChanged();
}

```

```

/**
 * Invoked when a child editor has been removed from the document tree.
 */
public void editorRemoved(Editor source)
{
}

/**
 * Invoked when a child editor is about to be removed from the document tree.
 * Returns true by default.
 */
public boolean editorRemoving(Editor source)
{
    return true;
}

/**
 * Fires a content change event to all listeners that registered with this
 * editor.
 */
public void fireEditorContentChanged()
{
    for (int i=0; i<listeners.size(); i++)
        ((EditorListener)listeners.get(i)).editorContentChanged(this);
}

/**
 * Fires a removing event to all listeners that registered with this
 * editor.
 * Note that this fires a removing event for all child editors first.
 * @return true if all listeners agreed on removing this editor and all of
 *         its child editors.
 */
public boolean fireEditorRemoving()
{
    boolean result = true;

    for (int i=0; i<children.size(); i++)
        result = result && ((Editor)children.get(i)).fireEditorRemoving();
    for (int i=0; i<listeners.size(); i++)
        result = result && ((EditorListener)listeners.get(i)).editorRemoving(this);

    return result;
}

/**
 * Fires a removed event to all listeners that registered with this
 * editor.
 * Note that this fires a removed event for all child editors first.
 */
public void fireEditorRemoved()
{
    for (int i=0; i<children.size(); i++)
        ((Editor)children.get(i)).fireEditorRemoved();
    for (int i=0; i<listeners.size(); i++)
        ((EditorListener)listeners.get(i)).editorRemoved(this);
}

/**
 * Get the display name for this editor to be shown in the document tree.
 * The default implementation looks up the name in the resource
 * bundle you specified in the xml document definition. If the resource bundle
 * is null, the section name is returned.
 * Override this method if you want to change the default behaviour.
 */
public String getDisplayName()
{
    if (resourceBundle!=null)
        return Lib.getResourceString(resourceBundle, sectionName);
    return sectionName;
}

/**
 * Returns a help message that is associated with this editor. It is shown
 * when the editor is opened in the application.
 * The default implementation looks up the help message in the

```

```

    * resource bundle you specified. The help message is looked up under the
    * entry equivalent to the section name appended by '.help'.
    * If the resource bundle is null or the entry does not exist, null is
    * returned (which will make the application say: 'no help available').
    * Override this method if you want to change the default behaviour.
    */
    public String getHelpMessage()
    {
        if (resourceBundle!=null)
            try {
                return resourceBundle.getString(sectionName+".help");
            } catch (Exception e) {
                return null;
            }
        return null;
    }

    /**
     * Set the language resource bundle for this editor.
     * @param resourceBundle a resource bundle.
     */
    public void setResourceBundle(ResourceBundle resourceBundle)
    {
        this.resourceBundle = resourceBundle;
    }

    /**
     * Get this editor's language resource bundle.
     */
    public ResourceBundle getResourceBundle()
    {
        return resourceBundle;
    }

    /**
     * Get the GUI component to display in the application
     */
    public abstract javax.swing.JComponent getEditorComponent();

    /**
     * Produce and return the XML content for this editor.
     */
    public abstract String getContent();

    /**
     * Restores the content of this editor
     * @param content XML content element
     */
    public abstract void setContent(org.w3c.dom.Element content);

    /**
     * Sets properties for this editor. This provides a general
     * method to set attributes for an editor
     * @param properties a hash map
     */
    public abstract void setProperties(java.util.HashMap properties);
}

```

Listing 4.1: Editor.java

Der Konstruktor eines Editors muss eine Instanz eines ProjectControls entgegennehmen können. Subklassen müssen einen entsprechenden Konstruktor vorsehen und darin einen Aufruf von *super(:ProjectControl)* vornehmen. Die Klasse Editor generiert einen eindeutigen ID-String für alle Editor-Instanzen. Dieser wird für die Ansteuerungen im ProjectView benötigt. Die entsprechende Methode *getID()* ist final, damit sie nicht fälschlicherweise überschrieben werden kann und die Eindeutigkeit der IDs gewahrt bleibt.

Wie bereits erwähnt muss jeder Editor die abstrakten Methoden *getContent()*

und `setContent(String)` implementieren, die die entsprechenden XML-Inhalte generieren bzw. aufnehmen müssen. Außerdem besitzt jeder Editor eine Menge von Editor-Kindern. Diese Editorhierarchie spiegelt die Struktur in der Dokumentbeschreibungsdatei (also die Struktur des Dokumentenbaums) wieder. Ein übergeordneter Editor kann so einen Aufruf von `getContent()` an seine Kinder weiterdelegieren und die Teildokumente der Subeditoren zusammenbauen und zurückgeben. Ein Aufruf von `getContent()` auf dem obersten Knoten eines Dokuments gibt also den Inhalt des gesamten Dokuments zurück.

Die Verwaltung der Kinder ist bereits implementiert, so dass sich eine Subklasse nicht mehr darum kümmern muss. Ebenso implementiert ist die Verwaltung der `EditorListener`, die Ereignisse eines Editors erhalten. Die Bedeutung der übrigen Methoden lässt sich aus den Quellcode-Kommentaren ersehen.

Um zu verdeutlichen, wie eine gültige Implementierung aussehen kann, wird nun die Klasse `BasicEditor` vorgestellt, von der alle anderen Klassen im Package `editors` erben.

4.2.3.2 Klasse `ieee829.app.editors.BasicEditor`

Dieser Editor stellt zwar keine Benutzeroberfläche zur Verfügung (die Methode `getEditorComponent()` gibt null zurück) - ermöglicht also keine Benutzereingaben, er implementiert aber die wesentlich Funktionalitäten der `setContent(...)`- und `getContent()`-Methoden. Die `getContent()`-Methode konkateniert die Inhalte aller Editor-Kinder und umfasst das Ergebnis mit einem XML-Tag, das dem Namen der `section` entspricht. Die Klasse ist folgendermaßen implementiert:

```

/*
 * $Id: BasicEditor.java,v 1.4 2003/04/06 04:37:13 liese Exp $
 * @author: Thorsten Liese - thorsten.liese@dreicsystems.de
 *
 */
package ieee829.app.editors;

import com.dreic.utils.Lib;
import ieee829.app.core.control.ProjectControl;
import java.util.HashMap;
import java.util.PropertyResourceBundle;
import java.util.ResourceBundle;
import org.w3c.dom.*;

/**
 * The basic editor is a default implementation of an editor. It does not
 * provide a user interface and the getContent methods simply return the
 * concatenated contents of all childs embraced by a tag that is equivalent
 * to the section name.
 */
public class BasicEditor extends Editor
{
    /**
     * Holds properties
     */
    private HashMap properties = null;

```

```

/**
 * Creates a new instance of BasicEditor
 */
public BasicEditor(ProjectControl project)
{
    super(project);
}

/**
 * Produce and return the XML content for this editor.
 */
public String getContent()
{
    StringBuffer basicStb = new StringBuffer();
    Editor[] children = getChildren();
    for (int i=0; i<children.length; i++)
        basicStb.append(children[i].getContent());

    if (properties!=null)
    {
        String omit = (String)getProperties().get("omitWhenEmpty");
        if (omit!=null && omit.equals("true") && basicStb.length()==0) return "";
    }

    StringBuffer stb = new StringBuffer("<");
    stb.append(getSectionName());
    stb.append(">\n");
    stb.append(basicStb.toString());
    stb.append("</");
    stb.append(getSectionName());
    stb.append(">\n");

    return stb.toString();
}

/**
 * Get the GUI component to display in the application
 */
public javax.swing.JComponent getEditorComponent()
{
    return null;
}

/**
 * Restores the content of this editor
 * @param content XML content
 */
public void setContent(Element content)
{
    NodeList elems = content.getChildNodes();
    Editor[] children = getChildren();
    for (int i=0; i<elems.getLength(); i++)
    {
        if (!(elems.item(i) instanceof Element)) continue;
        Element elem = (Element)elems.item(i);
        for (int j=0; j<children.length; j++)
        {
            BasicEditor basicEditor = (BasicEditor)children[j];
            if (basicEditor.getSectionName().equals(elem.getTagName()))
                basicEditor.setContent(elem);
        }
    }
}

/**
 * Sets properties for this editor. This provides a general
 * method to set attributes for an editor
 * @param properties a hash map
 */
public void setProperties(HashMap properties)
{
    this.properties = properties;
}

/**
 * Return the properties
 */

```

```

    public HashMap getProperties()
    {
        return properties;
    }
}

```

Listing 4.2: BasicEditor.java

Folgende Eigenschaften (properties) können für den BasicEditor gesetzt werden:

Name	Wertebereich	Default	Beschreibung
omitWhenEmpty	boolean	'false'	Wenn dieser Wert auf 'true' gesetzt wird, gibt der Editor einen Leerstring als Inhalt zurück, wenn alle Subeditoren ebenfalls keinen Inhalt erzeugen.

4.2.3.3 Klasse `ieee829.app.editors.TextEditor`

Dieser Editor stellt ein Textfenster zur Eingabe eines beliebigen Textblocks zur Verfügung. Der Rückgabewert ist der Inhalt des Textfensters, umklammert mit einem `<formattedtext>`-Element, welches wiederum durch ein Element umklammert ist, das dem Namen der *section* entspricht. Folgende Eigenschaften können gesetzt werden:

Name	Wertebereich	Default	Beschreibung
omitWhenEmpty	boolean	'false'	Wenn dieser Wert auf 'true' gesetzt wird, gibt der Editor einen Leerstring als Inhalt zurück, wenn das Textfenster leer ist.

4.2.3.4 Klasse `ieee829.app.editors.DocumentGroupEditor`

Dieser Editor besitzt die Fähigkeit, neue Dokumentbeschreibungsdateien zu laden und einen entsprechenden Dokumenten-Teilbaum sich selbst als Kind hinzuzufügen. Er bietet dazu einen Schaltknopf 'Neues Dokument' an. Ein `DocumentGroupEditor` sollte daher üblicherweise als Blatt (ohne Kinder) in einer Dokumentbeschreibungsdatei verwendet werden. Der oberste Editor des neu zu ladenden Dokuments muss ein `ieee829.app.editors.DocumentEditor` sein. Folgende Eigenschaften können gesetzt werden.

Name	Wertebereich	Default	Beschreibung
<code>omitWhenEmpty</code>	<code>boolean</code>	<code>'false'</code>	Wenn dieser Wert auf <code>'true'</code> gesetzt wird, gibt der Editor einen Leerstring als Inhalt zurück, wenn alle Subeditoren ebenfalls keinen Inhalt erzeugen.
<code>documentType</code>	<code>string</code>	<code>null</code>	Eine Bezeichnung für den zu erzeugenden Dokumenttyp. Dies dient lediglich der Darstellung. Der Name wird durch das Resource-Bundle aufgelöst.
<code>documentSpecFile</code>	<code>string</code>	<code>null</code>	Der Dateiname der Dokumentbeschreibungdatei für das neue Dokument.
<code>nameSpace</code>	<code>string</code>	<code>null</code>	Bezeichnung eines Namensraums, in dem der Name des erzeugten Dokuments eindeutig sein muss.

Der Parameter `documentSpecFile` ist zwingend erforderlich.

4.2.3.5 Klasse `ieee829.app.editors.DocumentEditor`

Dieser Editor muss der oberste Editor einer Dokumentbeschreibungdatei sein, die durch einen `ieee829.app.editors.DocumentGroupEditor` geladen wird. Der Editor stellt einen Schaltknopf zur Entfernung des aktuellen Dokuments zur Verfügung und er stellt statt des Abschnittsnamens den Namen des Dokuments im Dokumentenbaum dar.

Name	Wertebereich	Default	Beschreibung
<code>omitWhenEmpty</code>	<code>boolean</code>	<code>'false'</code>	Wenn dieser Wert auf <code>'true'</code> gesetzt wird, gibt der Editor einen Leerstring als Inhalt zurück, wenn alle Subeditoren ebenfalls keinen Inhalt erzeugen.
<code>nameSpace</code>	<code>string</code>	<code>null</code>	Bezeichnung eines Namensraums, in dem der Name des erzeugten Dokuments eindeutig sein muss.

4.2.3.6 Klasse `ieee829.app.editors.IdentifierEditor`

Dies ist ein spezieller Editor, der nur als Kind eines `ieee829.app.editors.DocumentEditor` verwendet werden kann. Er erlaubt das Editieren des Dokumentnamens.

Name	Wertebereich	Default	Beschreibung
<code>nameSpace</code>	<code>string</code>	<code>null</code>	Bezeichnung eines Namensraums, in dem der Name des erzeugten Dokuments eindeutig sein muss.

4.2.3.7 Klasse `ieee829.app.editors.NodeSelectEditor`

Dieser Editor erlaubt die Auswahl eines Wertes aus einer Menge zur Verfügung stehender Werte. Die zur Verfügung stehenden Werte sind dabei die Kinder eines als Eigenschaft `nodeName` angegebenen Abschnittsnamens.

Name	Wertebereich	Default	Beschreibung
<code>omitWhenEmpty</code>	<code>boolean</code>	<code>'false'</code>	Wenn dieser Wert auf <code>'true'</code> gesetzt wird, gibt der Editor einen Leerstring als Inhalt zurück, wenn kein Wert ausgewählt wurde.
<code>nodeName</code>	<code>string</code>	<code>null</code>	Bezeichnung eines Abschnittsnamens (<i>section</i>) im Dokumentenbaum. Da Abschnitte mit dem selben Namen unter Umständen mehrfach im Dokumentenbaum vorkommen können, ist zu beachten, dass der erste Knoten verwendet wird, der bei einer Breitensuche über dem Baum gefunden wird.

4.2.3.8 Klasse `ieee829.app.editors.NodeListEditor`

Dieser Editor erlaubt die Auswahl einer Liste von Werten aus einer Menge zur Verfügung stehender Werte. Die zur Verfügung stehenden Werte sind dabei wie beim `ieee829.app.editors.NodeSelectEditor` die Kinder eines als Eigenschaft `nodeName` angegebenen Abschnittsnamens.

Name	Wertebereich	Default	Beschreibung
<code>omitWhenEmpty</code>	boolean	'false'	Wenn dieser Wert auf 'true' gesetzt wird, gibt der Editor einen Leerstring als Inhalt zurück, wenn die Auswahl-liste leer ist.
<code>nodeName</code>	string	null	Bezeichnung eines Abschnittsnamens (<i>section</i>) im Dokumentenbaum. Da Abschnitte mit dem selben Namen unter Umständen mehrfach im Dokumentenbaum vorkommen können, ist zu beachten, dass der erste Knoten verwendet wird, der bei einer Breitensuche über dem Baum gefunden wird.

4.3 Dokumentbeschreibungsdateien

Damit die Ausgaben den für die Dokumente des IEEE829-Standard definierten DTDs genügen, müssen nun Dokumentbeschreibungsdateien für den IEEE829-Standard erstellt werden, die entsprechend gültige Dokumente erzeugen. Nachfolgend werden diese Dokumentbeschreibungsdateien vorgestellt. Diese Dateien befinden sich im Verzeichnis *xml*. Die Abbildung 4.8 auf Seite 61 gibt einen Überblick über die Abhängigkeiten dieser Dateien.

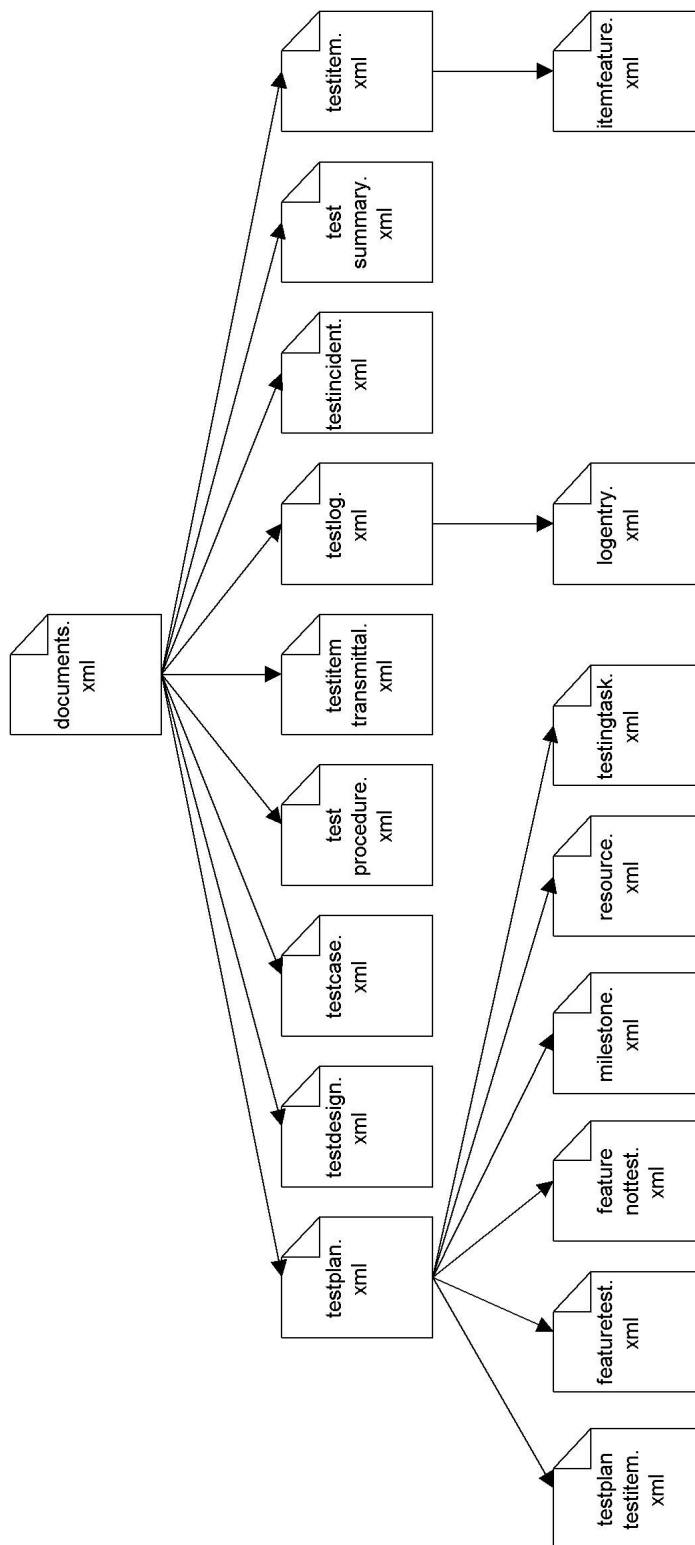


Abbildung 4.8: Dokumentbeschreibungsdateien

4.3.1 Übergreifende Beschreibungsdatei

Diese Datei definiert das Rumpfdokument, das sichtbar ist, wenn die Applikation gestartet wird. Dieses beinhaltet Einträge für die verschiedenen Dokumenttypen des IEEE829-Standard, unter denen der Anwender jeweils neue Dokumente anlegen kann.

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
  Document : documents.xml
  Author   : Thorsten Liese
  Description:
    Defines the different IEEE829 documents for the application
-->

<!DOCTYPE section SYSTEM "dtd/autodoc.dtd">

<section name="ieee829-project"
  editor="ieee829.app.editors.BasicEditor"
  icon="root"
  resourcebundle="lang/ieee829">

  <section name="ieee829-core"
    editor="ieee829.app.editors.ieee829.ExportEditor"
    icon="root">

    <section name="testplans"
      editor="ieee829.app.editors.DocumentGroupEditor"
      icon="documentgroup">
      <property name="nameSpace">testplan</property>
      <property name="documentType">testplan</property>
      <property name="documentSpecFile">xml/testplan.xml</property>
    </section>
    <section name="testdesignspecifications"
      editor="ieee829.app.editors.DocumentGroupEditor"
      icon="documentgroup">
      <property name="nameSpace">testdesign</property>
      <property name="documentType">testdesign</property>
      <property name="documentSpecFile">xml/testdesign.xml</property>
    </section>
    <section name="testcasespecifications"
      editor="ieee829.app.editors.DocumentGroupEditor"
      icon="documentgroup">
      <property name="nameSpace">testcase</property>
      <property name="documentType">testcase</property>
      <property name="documentSpecFile">xml/testcase.xml</property>
    </section>
    <section name="testprocedurespecifications"
      editor="ieee829.app.editors.DocumentGroupEditor"
      icon="documentgroup">
      <property name="nameSpace">testprocedure</property>
      <property name="documentType">testprocedure</property>
      <property name="documentSpecFile">xml/testprocedure.xml</property>
    </section>
    <section name="testitemtransmittalreports"
      editor="ieee829.app.editors.DocumentGroupEditor"
      icon="documentgroup">
      <property name="nameSpace">testitemtransmittal</property>
      <property name="documentType">testitemtransmittal</property>
      <property name="documentSpecFile">xml/testitemtransmittal.xml</property>
    </section>
    <section name="testlogs"
      editor="ieee829.app.editors.DocumentGroupEditor"
      icon="documentgroup">
      <property name="nameSpace">testlog</property>
      <property name="documentType">testlog</property>
      <property name="documentSpecFile">xml/testlog.xml</property>
    </section>
    <section name="testincidentreports"
      editor="ieee829.app.editors.DocumentGroupEditor"
      icon="documentgroup">
      <property name="nameSpace">testincident</property>
      <property name="documentType">testincident</property>
    </section>
  </section>
</section>
```

```

        <property name="documentSpecFile">xml/testincident.xml</property>
    </section>
    <section name="testsummaryreports"
        editor="ieee829.app.editors.DocumentGroupEditor"
        icon="documentgroup">
        <property name="nameSpace">testsummary</property>
        <property name="documentType">testsummary</property>
        <property name="documentSpecFile">xml/testsummary.xml</property>
    </section>
</section>

<section name="ieee829-ext" editor="ieee829.app.editors.BasicEditor"
    icon="root">

    <section name="testitems"
        editor="ieee829.app.editors.DocumentGroupEditor"
        icon="documentgroup">
        <property name="nameSpace">testitem</property>
        <property name="documentType">testitem</property>
        <property name="documentSpecFile">xml/testitem.xml</property>
    </section>
</section>
</section>

```

Listing 4.3: documents.xml

4.3.2 Testplan-Beschreibungsdatei

```

<?xml version="1.0" encoding="UTF-8"?>
<!--
  Document : testplan.xml
  Author   : Thorsten Liese
  Description:
    XML file for creating test plans.
-->
<!DOCTYPE section SYSTEM "dtd/autodoc.dtd">
<section name="testplan" editor="ieee829.app.editors.DocumentEditor" icon="document"
    resourcebundle="lang/ieee829">

    <property name="nameSpace">testplan</property>

    <section name="identifier" editor="ieee829.app.editors.IdentifierEditor">
        <property name="nameSpace">testplan</property>
    </section>
    <section name="introduction" editor="ieee829.app.editors.TextEditor">
    </section>
    <section name="testitems" editor="ieee829.app.ext.testitems.ItemSelectEditor">
        <property name="documentSpecFile">xml/testplantestitem.xml</property>
    </section>
    <section name="featurestobetested" editor="ieee829.app.editors.DocumentGroupEditor">
        <property name="documentType">featureset</property>
        <property name="documentSpecFile">xml/featuretest.xml</property>
    </section>
    <section name="featuresnottobetested" editor="ieee829.app.editors.DocumentGroupEditor">
        <property name="omitWhenEmpty">true</property>
        <property name="documentType">featureset</property>
        <property name="documentSpecFile">xml/featurenottest.xml</property>
    </section>
    <section name="approach" editor="ieee829.app.editors.TextEditor">
    </section>
    <section name="itempassfailcriteria" editor="ieee829.app.editors.TextEditor">
    </section>
    <section name="suspensioncriteria" editor="ieee829.app.editors.TextEditor">
        <property name="omitWhenEmpty">true</property>
    </section>
    <section name="testdeliverables" editor="ieee829.app.editors.ieee829.TestDeliverablesEditor">
    </section>
    <section name="testingtasks" editor="ieee829.app.editors.DocumentGroupEditor">
        <property name="nameSpace">testingtask</property>
        <property name="documentType">testingtask</property>
    </section>

```

```

    <property name="documentSpecFile">xml/testingtask.xml</property>
  </section>
  <section name="environmentalneeds" editor="ieee829.app.editors.BasicEditor">
    <section name="description" editor="ieee829.app.editors.TextEditor">
    </section>
    <section name="resources" editor="ieee829.app.editors.DocumentGroupEditor">
      <property name="nameSpace">resource</property>
      <property name="documentType">resource</property>
      <property name="documentSpecFile">xml/resource.xml</property>
    </section>
  </section>
  <section name="responsibilities" editor="ieee829.app.editors.TextEditor">
  </section>
  <section name="staffingandtraining" editor="ieee829.app.editors.TextEditor">
    <property name="omitWhenEmpty">true</property>
  </section>
  <section name="schedule" editor="ieee829.app.editors.BasicEditor">
    <section name="description" editor="ieee829.app.editors.TextEditor">
    </section>
    <section name="milestones" editor="ieee829.app.editors.DocumentGroupEditor">
      <property name="nameSpace">milestone</property>
      <property name="documentType">milestone</property>
      <property name="documentSpecFile">xml/milestone.xml</property>
    </section>
  </section>
  <section name="risksandcontingencies" editor="ieee829.app.editors.TextEditor">
  </section>
  <section name="approvals" editor="ieee829.app.editors.TextEditor">
  </section>
</section>

```

Listing 4.4: testplan.xml

4.3.3 Testdesignspezifikation-Beschreibungsdatei

```

<?xml version="1.0" encoding="UTF-8"?>
<!--
  Document : testdesign.xml
  Author   : Thorsten Liese
  Description:
    XML document for creating test design specifications
-->
<!DOCTYPE section SYSTEM "dtd/autodoc.dtd">
<section name="testdesignspecification" editor="ieee829.app.editors.DocumentEditor"
  icon="document"
  resourcebundle="lang/ieee829">
  <property name="nameSpace">testdesign</property>
  <section name="identifier" editor="ieee829.app.editors.IdentifierEditor">
    <property name="nameSpace">testdesign</property>
    <section name="testplanref"
      editor="ieee829.app.editors.ieee829.IEEE829DocumentSelectEditor">
      <property name="nodeName">testplans</property>
    </section>
  </section>
  <section name="featurestobetested" editor="ieee829.app.editors.TextEditor">
  </section>
  <section name="approachrefine" editor="ieee829.app.editors.TextEditor">
  </section>
  <section name="testidentification" editor="ieee829.app.editors.TextEditor">
  </section>
  <section name="featurepassfailcriteria" editor="ieee829.app.editors.TextEditor">
  </section>
</section>

```

Listing 4.5: testdesign.xml

4.3.4 Testfallspezifikation-Beschreibungsdatei

```

<?xml version="1.0" encoding="UTF-8"?>
<!--
  Document : testcase
  Author   : Thorsten Liese
  Description:
    XML document for creating test case specifications
-->
<!DOCTYPE section SYSTEM "dtd/autodoc.dtd">
<section name="testcasespecification" editor="ieee829.app.editors.DocumentEditor"
  icon="document"
  resourcebundle="lang/ieee829">
  <property name="nameSpace">testcase</property>
  <section name="identifier" editor="ieee829.app.editors.IdentifierEditor">
    <property name="nameSpace">testcase</property>
  </section>
  <section name="testitems" editor="ieee829.app.editors.TextEditor">
  </section>
  <section name="inputspecification" editor="ieee829.app.editors.TextEditor">
  </section>
  <section name="outputspecification" editor="ieee829.app.editors.TextEditor">
  </section>
  <section name="environmentalneeds" editor="ieee829.app.editors.TextEditor">
    <section name="hardware" editor="ieee829.app.editors.TextEditor">
    </section>
    <section name="software" editor="ieee829.app.editors.TextEditor">
    </section>
    <section name="other" editor="ieee829.app.editors.TextEditor">
    </section>
  </section>
  <section name="specialproceduralrequirements" editor="ieee829.app.editors.TextEditor">
  </section>
  <section name="intercasedependencies" editor="ieee829.app.editors.TextEditor">
  </section>
</section>

```

Listing 4.6: testcase.xml

4.3.5 Testablaufspezifikation-Beschreibungsdatei

```

<?xml version="1.0" encoding="UTF-8"?>
<!--
  Document : testprocedure.xml
  Author   : Thorsten Liese
  Description:
    XML document for creating test procedure specifications
-->
<!DOCTYPE section SYSTEM "dtd/autodoc.dtd">
<section name="testprocedurespecification" editor="ieee829.app.editors.DocumentEditor"
  icon="document"
  resourcebundle="lang/ieee829">
  <property name="nameSpace">testprocedure</property>
  <section name="identifier" editor="ieee829.app.editors.IdentifierEditor">
    <property name="nameSpace">testprocedure</property>
  </section>
  <section name="purpose" editor="ieee829.app.editors.TextEditor">
  </section>
  <section name="specialrequirements" editor="ieee829.app.editors.TextEditor">
  </section>
  <section name="proceduresteps" editor="ieee829.app.editors.TextEditor">
  </section>

```

```

<section name="log" editor="ieee829.app.editors.TextEditor">
</section>
<section name="setup" editor="ieee829.app.editors.TextEditor">
</section>
<section name="start" editor="ieee829.app.editors.TextEditor">
</section>
<section name="proceed" editor="ieee829.app.editors.TextEditor">
</section>
<section name="measure" editor="ieee829.app.editors.TextEditor">
</section>
<section name="shutdown" editor="ieee829.app.editors.TextEditor">
</section>
<section name="restart" editor="ieee829.app.editors.TextEditor">
</section>
<section name="stop" editor="ieee829.app.editors.TextEditor">
</section>
<section name="wrapup" editor="ieee829.app.editors.TextEditor">
</section>
<section name="contingencies" editor="ieee829.app.editors.TextEditor">
</section>
</section>
</section>

```

Listing 4.7: testprocedure.xml

4.3.6 Testgegenstandsauslieferungsbericht- Beschreibungsdatei

```

<?xml version="1.0" encoding="UTF-8"?>
<!--
  Document : testitemtransmittal.xml
  Author   : Thorsten Liese
  Description:
    XML document for creating test item transmittal reports
-->
<!DOCTYPE section SYSTEM "dtd/autodoc.dtd">
<section name="testitemtransmittalreport" editor="ieee829.app.editors.DocumentEditor"
  icon="document"
  resourcebundle="lang/ieee829">
  <property name="nameSpace">testitemtransmittal</property>
  <section name="identifier" editor="ieee829.app.editors.IdentifierEditor">
    <property name="nameSpace">testitemtransmittal</property>
  </section>
  <section name="transmitteditems" editor="ieee829.app.editors.TextEditor">
  </section>
  <section name="location" editor="ieee829.app.editors.TextEditor">
  </section>
  <section name="status" editor="ieee829.app.editors.TextEditor">
  </section>
  <section name="approvals" editor="ieee829.app.editors.TextEditor">
  </section>
</section>

```

Listing 4.8: testitemtransmittal.xml

4.3.7 Testprotokoll-Beschreibungsdatei

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Document : testlog.xml
  Author   : Thorsten Liese
  Description:
    XML document for creating test logs
-->
<!DOCTYPE section SYSTEM "dtd/autodoc.dtd">
<section name="testlog" editor="ieee829.app.editors.DocumentEditor" icon="document"
  resourcebundle="lang/ieee829">
  <property name="nameSpace">testlog</property>
  <section name="identifier" editor="ieee829.app.editors.IdentifierEditor">
    <property name="nameSpace">testlog</property>
  </section>
  <section name="description" editor="ieee829.app.editors.TextEditor">
  </section>
  <section name="activityandevententries" editor="ieee829.app.editors.DocumentGroupEditor">
    <property name="documentType">logentry</property>
    <property name="documentSpecFile">xml/logentry.xml</property>
  </section>
</section>
```

Listing 4.9: testlog.xml

4.3.8 Testvorfallbericht-Beschreibungsdatei

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Document : testincident.xml
  Author   : Thorsten Liese
  Description:
    XML document for creating test incident reports
-->
<!DOCTYPE section SYSTEM "dtd/autodoc.dtd">
<section name="testincidentreport" editor="ieee829.app.editors.DocumentEditor"
  icon="document"
  resourcebundle="lang/ieee829">
  <property name="nameSpace">testincident</property>
  <section name="identifier" editor="ieee829.app.editors.IdentifierEditor">
    <property name="nameSpace">testincident</property>
  </section>
  <section name="summary" editor="ieee829.app.editors.TextEditor">
  </section>
  <section name="incidentdescription" editor="ieee829.app.editors.TextEditor">
  </section>
  <section name="impact" editor="ieee829.app.editors.TextEditor">
  </section>
</section>
```

Listing 4.10: testincident.xml

4.3.9 Testergebnisbericht-Beschreibungsdatei

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Document : testsummary.xml
  Author   : Thorsten Liese
  Description:
    XML document for creating test summary reports
-->
<!DOCTYPE section SYSTEM "dtd/autodoc.dtd">
<section name="testsummaryreport" editor="ieee829.app.editors.DocumentEditor"
  icon="document"
  resourcebundle="lang/ieee829">
  <property name="nameSpace">testsummary</property>
  <section name="identifier" editor="ieee829.app.editors.IdentifierEditor">
    <property name="nameSpace">testsummary</property>
  </section>
  <section name="summary" editor="ieee829.app.editors.TextEditor">
  </section>
  <section name="variances" editor="ieee829.app.editors.TextEditor">
  </section>
  <section name="comprehensivenessassessment" editor="ieee829.app.editors.TextEditor">
  </section>
  <section name="summaryofresults" editor="ieee829.app.editors.TextEditor">
  </section>
  <section name="evaluation" editor="ieee829.app.editors.TextEditor">
  </section>
  <section name="summaryofactivities" editor="ieee829.app.editors.TextEditor">
  </section>
  <section name="approvals" editor="ieee829.app.editors.TextEditor">
  </section>
</section>
```

Listing 4.11: testsummary.xml

4.3.10 Testgegenstand-Beschreibungsdatei als Erweiterung

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Document : testitem.xml
  Author   : Thorsten Liese
  Description:
    XML document for creating test items
-->
<!DOCTYPE section SYSTEM "dtd/autodoc.dtd">
<section name="testitem" editor="ieee829.app.editors.DocumentEditor" icon="document"
  resourcebundle="lang/ieee829">
  <property name="nameSpace">testitem</property>
  <section name="identifier" editor="ieee829.app.editors.IdentifierEditor">
    <property name="nameSpace">testitem</property>
  </section>
  <section name="description" editor="ieee829.app.editors.TextEditor">
  </section>
  <section name="itemdocumentation" editor="ieee829.app.ext.testitems.ItemDocumentationEditor">
  </section>
  <section name="features" editor="ieee829.app.editors.DocumentGroupEditor">
    <property name="nameSpace">itemfeature</property>
    <property name="documentType">itemfeature</property>
    <property name="documentSpecFile">xml/itemfeature.xml</property>
  </section>
</section>
```

Listing 4.12: testitem.xml

4.3.11 Testgegenstand-Beschreibungsdatei für den Testplan

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Document : testplantestitem.xml
  Author   : Thorsten Liese
  Description:
    XML document defines a test item in a test plan
-->
<!DOCTYPE section SYSTEM 'dtd/autodoc.dtd'>
<section name="testitem" editor="ieee829.app.ext.testitems.ItemEditor"
  resourcebundle="lang/ieee829">
  <section name="iteminfo" editor="ieee829.app.ext.testitems.ItemInfoEditor">
  </section>
  <section name="itemversion" editor="ieee829.app.editors.TextEditor">
  </section>
  <section name="itemtransmittalinfo" editor="ieee829.app.editors.TextEditor">
  </section>
</section>
```

Listing 4.13: testplantestitem.xml

4.3.12 Beschreibungsdatei für zu testende Funktionen

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Document : featuretest.xml
  Author   : Thorsten Liese
  Description:
    To define a featureset in a test plan
-->
<!DOCTYPE section SYSTEM 'dtd/autodoc.dtd'>
<section name="featuretest" editor="ieee829.app.editors.DocumentEditor"
  resourcebundle="lang/ieee829">
  <section name="featureset" editor="ieee829.app.ext.testitems.FeatureSelectEditor">
    <property name="depth">2</property>
  </section>
  <section name="testdesignref"
    editor="ieee829.app.editors.ieee829.IEEE829DocumentSelectEditor">
    <property name="nodeName">testdesignspecifications</property>
  </section>
</section>
```

Listing 4.14: featuretest.xml

4.3.13 Beschreibungsdatei für nicht zu testende Funktionen

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Document : featurenottest.xml
  Author   : Thorsten Liese
  Description:
    To define a featureset in a test plan
-->
<!DOCTYPE section SYSTEM 'dtd/autodoc.dtd'>
<section name="featurenottest" editor="ieee829.app.editors.DocumentEditor"
  resourcebundle="lang/ieee829">
  <property name="omitWhenEmpty">true</property>
  <section name="featureset" editor="ieee829.app.ext.testitems.FeatureSelectEditor">
    <property name="omitWhenEmpty">true</property>
    <property name="depth">2</property>
  </section>
  <section name="description" editor="ieee829.app.editors.TextEditor">
    <property name="omitWhenEmpty">true</property>
  </section>
</section>
```

Listing 4.15: featurenottest.xml

4.3.14 Meilenstein-Beschreibungsdatei

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Document : milestone.xml
  Author   : Thorsten Liese
  Description:
    To define a milestone
-->
<!DOCTYPE section SYSTEM 'dtd/autodoc.dtd'>
<section name="milestone" editor="ieee829.app.editors.DocumentEditor"
  resourcebundle="lang/ieee829">
  <property name="nameSpace">milestone</property>
  <section name="identifier" editor="ieee829.app.editors.IdentifierEditor">
    <property name="nameSpace">milestone</property>
  </section>
  <section name="description" editor="ieee829.app.editors.TextEditor">
  </section>
</section>
```

Listing 4.16: milestone.xml

4.3.15 Ressourcen-Beschreibungsdatei

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Document : resource.xml
  Author   : Thorsten Liese
  Description:
    To define a resource
-->
<!DOCTYPE section SYSTEM 'dtd/autodoc.dtd'>
<section name="resource" editor="ieee829.app.editors.DocumentEditor"
  resourcebundle="lang/ieee829">
  <property name="nameSpace">resource</property>
  <section name="identifier" editor="ieee829.app.editors.IdentifierEditor">
    <property name="nameSpace">resource</property>
  </section>
  <section name="description" editor="ieee829.app.editors.TextEditor">
  </section>
</section>
```

Listing 4.17: resource.xml

4.3.16 Testaktivitäten-Beschreibungsdatei

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Document : testingtask.xml
  Author   : Thorsten Liese
  Description:
    To define a testing task
-->
<!DOCTYPE section SYSTEM 'dtd/autodoc.dtd'>
<section name="testingtask" editor="ieee829.app.editors.DocumentEditor"
  resourcebundle="lang/ieee829">
  <property name="nameSpace">testingtask</property>
  <section name="identifier" editor="ieee829.app.editors.IdentifierEditor">
    <property name="nameSpace">testingtask</property>
  </section>
  <section name="description" editor="ieee829.app.editors.TextEditor">
  </section>
</section>
```

Listing 4.18: testingtask.xml

4.3.17 Protokolleintrag-Beschreibungsdatei

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Document : logentry.xml
  Created on : 2. April 2003, 03:44
  Author   : Administrator
  Description:
    Purpose of the document follows.
-->
<!DOCTYPE section SYSTEM "dtd/autodoc.dtd">
<section name="logentry" editor="ieee829.app.editors.DocumentEditor"
  resourcebundle="lang/ieee829">
  <section name="executiondescription" editor="ieee829.app.editors.TextEditor">
  </section>
  <section name="procedureresults" editor="ieee829.app.editors.TextEditor">
  </section>
  <section name="environmentalinformation" editor="ieee829.app.editors.TextEditor">
  </section>
  <section name="anomalousevents" editor="ieee829.app.editors.TextEditor">
  </section>
  <section name="incidentreports" editor="ieee829.app.editors.TextEditor">
  </section>
</section>
```

Listing 4.19: logentry.xml

4.3.18 Testgegenstandseigenschaft-Beschreibungsdatei

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Document : itemfeature.xml
  Author   : Thorsten Liese
  Description:
    XML file for creating item features
-->
<!DOCTYPE section SYSTEM 'dtd/autodoc.dtd'>
<section name="feature" editor="ieee829.app.editors.DocumentEditor"
  resourcebundle="lang/ieee829">
  <property name="nameSpace">itemfeature</property>
  <section name="identifier" editor="ieee829.app.editors.IdentifierEditor">
    <property name="nameSpace">itemfeature</property>
  </section>
  <section name="description" editor="ieee829.app.editors.TextEditor">
  </section>
</section>
```

Listing 4.20: itemfeature.xml

4.4 Speicherformat der Software

Die Software speichert Projekte in einem eigenen Format ab. Dazu werden die Eigenschaften eines Projekts (derzeit nur Projektname) zusammen mit einer Referenz auf die Hauptdokumentdatei in einer Datei *project.xml* gespeichert. Die Hauptdokumentdatei ergibt sich durch Aufruf von *getContent()* auf dem Wurzelknoten des Dokumentenbaums. Diese wird zusammen mit der Datei *project.xml* in ein JAR-Archiv gepackt und in eine Datei mit der Endung *xjr* (Xml documentation JaR) gespeichert.

Die DTD für die Datei *project.xml* sieht folgendermaßen aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Document : project.dtd
  Author   : Thorsten Liese
  Description:
    Defines DTD for project.xml
-->
<!ELEMENT project (docref*)>
<!ELEMENT docref #PCDATA>
<!ATTLIST project name CDATA #REQUIRED>
```

Listing 4.21: project.dtd

Kapitel 5

Erweiterungen

5.1 Einleitung

Die vorgestellte Software bietet eine Vielzahl von Erweiterungsmöglichkeiten, die in diesem Kapitel kurz vorgestellt werden sollen. Einige der Erweiterungen sind bereits ansatzweise im entwickelten Prototyp enthalten.

5.2 Dokumenten-Export mit XSL

Die Möglichkeit, die erstellten Dokumente in Anzeige- bzw. Druckformate zu exportieren, ist von zentraler Bedeutung. Die von der Software erzeugten XML-Dokumente müssen nämlich in andere Formate konvertiert werden, bevor sie auf dem Bildschirm angezeigt oder ausgedruckt werden können.

Um XML in beliebige andere Formate zu überführen, verwendet man die *eXtended Stylesheet Language* (siehe [XSL]). Eine XSL-Datei enthält Vorschriften, wie die einzelnen Teile einer XML-Quelldatei in ein Ergebnisformat zu überführen sind.

Um einen IEEE829-Testplan in ein HTML-Dokument zu konvertieren, kann die folgende XSL-Datei verwendet werden. Die Datei *testplan.xsl* befindet sich im Verzeichnis *app/src/xml/xsl*.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--
  Document : testplan.xml
  Author   : Thorsten Liese
  Description:
           To convert an XML Test Plan to XHTML
-->
```

```

<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format">

  <!-- template rule matching source root element -->
  <xsl:template match="/">
    <html>
      <body>
        <h1><xsl:value-of select="testplan/identifizier/@name"/></h1>
        <ol>
          <xsl:apply-templates/>
        </ol>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="testplan/introduction">
    <h2><li>Einleitung</li></h2>
    <xsl:apply-templates select="formattedtext"/>
  </xsl:template>
  <xsl:template match="testplan/testitems">
    <h2><li>Testgegenstaende</li></h2>
    <ol>
      <xsl:for-each select="testitem">
        <h3><li><xsl:value-of select="name"/></li></h3>
        Version: <xsl:value-of select="itemversion"/><p>
        <xsl:apply-templates select="description/formattedtext"/>
      </xsl:for-each>
    </ol>
  </xsl:template>
  <xsl:template match="testplan/featurestobetested">
    <h2><li>Zu testende Funktionen</li></h2>
    <ul>
      <xsl:for-each select="featuretest">
        <h3><li><xsl:apply-templates select="testdesignref"/></li></h3>
        Getestete Funktionen:
        <ol>
          <xsl:for-each select="featureset/feature">
            <h4><li><xsl:value-of select="name"/></li></h4>
            <xsl:apply-templates select="description/formattedtext"/>
          </xsl:for-each>
        </ol>
      </xsl:for-each>
    </ul>
    <xsl:apply-templates select="formattedtext"/>
  </xsl:template>
  <xsl:template match="testplan/featuresnottobetested">
    <h2><li>Nicht zu testende Funktionen</li></h2>
    Die folgenden Funktionskombinationen sind nicht zu testen:
    <xsl:for-each select="featurenottest">
      <p>
      <ul>
        <xsl:for-each select="featureset/feature">
          <li><xsl:value-of select="name"/></li>
        </xsl:for-each>
      </ul>
      <xsl:apply-templates select="description/formattedtext"/>
    </xsl:for-each>
  </xsl:template>
  <xsl:template match="testplan/approach">
    <h2><li>Vorgehensweise</li></h2>
    <xsl:apply-templates select="formattedtext"/>
  </xsl:template>
  <xsl:template match="testplan/itempassfailcriteria">
    <h2><li>Pass/Fail-Kriterien der Testgegenstaende</li></h2>
    <xsl:apply-templates select="formattedtext"/>
  </xsl:template>
  <xsl:template match="testplan/suspensioncriteria">
    <h2><li>Aussetzungs- und Wiederaufnahmeanforderungen</li></h2>
    <xsl:apply-templates select="formattedtext"/>
  </xsl:template>
  <xsl:template match="testplan/testdeliverables">
    <h2><li>Auslieferungsgegenstaende</li></h2>
    <ul>
      <xsl:for-each select="documentref">
        <li><xsl:apply-templates select="."/></li>
      </xsl:for-each>
    </ul>
  </xsl:template>

```

```

    </ul>
</xsl:template>
<xsl:template match="testplan/testingtasks">
  <h2><li>Testaktivitaeten</li></h2>
  <ol>
    <xsl:for-each select="testingtask">
      <h3><li><xsl:value-of select="identifier/@name"/></li></h3>
      <xsl:apply-templates select="description/formattedtext"/>
      <p/>
    </xsl:for-each>
  </ol>
</xsl:template>
<xsl:template match="testplan/environmentalneeds">
  <h2><li>Umgebungsanforderungen</li></h2>
  <xsl:apply-templates select="description/formattedtext"/>
  <p/>
  <h3>Ressourcen</h3>
  <ul>
    <xsl:for-each select="resources/resource">
      <h4><li><xsl:value-of select="identifier/@name"/></li></h4>
      <xsl:apply-templates select="description/formattedtext"/>
    </xsl:for-each>
  </ul>
</xsl:template>
<xsl:template match="testplan/responsibilities">
  <h2><li>Verantwortlichkeiten</li></h2>
  <xsl:apply-templates select="formattedtext"/>
</xsl:template>
<xsl:template match="testplan/staffingandtraining">
  <h2><li>Personal- und Ausbildungsanforderungen</li></h2>
  <xsl:apply-templates select="formattedtext"/>
</xsl:template>
<xsl:template match="testplan/schedule">
  <h2><li>Zeitplan</li></h2>
  <xsl:apply-templates select="description/formattedtext"/>
  <p/>
  <h3>Meilensteine</h3>
  <ol>
    <xsl:for-each select="milestones/milestone">
      <h4><li><xsl:value-of select="identifier/@name"/></li></h4>
      <xsl:apply-templates select="description/formattedtext"/>
    </xsl:for-each>
  </ol>
</xsl:template>
<xsl:template match="testplan/risksandcontingencies">
  <h2><li>Risiken und Eventualitaeten</li></h2>
  <xsl:apply-templates select="formattedtext"/>
</xsl:template>
<xsl:template match="testplan/approvals">
  <h2><li>Genehmigungen</li></h2>
  <xsl:apply-templates select="formattedtext"/>
</xsl:template>

<xsl:template match="text()">
  <xsl:value-of select="."/>
</xsl:template>
<xsl:template match="i">
  <i><xsl:value-of select="."/></i>
</xsl:template>
<xsl:template match="b">
  <b><xsl:value-of select="."/></b>
</xsl:template>
<xsl:template match="br">
  <br/>
</xsl:template>
<xsl:template match="p">
  <p/>
</xsl:template>
<xsl:template match="itemize">
  <ul>
    <xsl:for-each select="item">
      <li><xsl:apply-templates/></li>
    </xsl:for-each>
  </ul>
</xsl:template>
<xsl:template match="enumerate">
  <ol>
    <xsl:for-each select="item">

```

```
        <li><xsl:apply-templates/></li>
    </xsl:for-each>
</ol>
</xsl:template>
<xsl:template match="paragraph">
    <p/>
</xsl:template>
<xsl:template match="documentref">
    <a href="{@ref}"><xsl:value-of select="."/></a>
</xsl:template>
</xsl:stylesheet>
```

Listing 5.1: testplan.xsl

5.3 Testplanung und -durchführung

Die vorliegende Software unterstützt derzeit lediglich den Erstellungsvorgang der Dokumente. Die Software kann jedoch dahingehend erweitert werden, dass auch Planung und Durchführung der Testaktivitäten mit Hilfe der Software koordiniert werden. Dies könnte beispielsweise für die folgenden Punkte geschehen:

- Die Erstellung des Zeitplans im Testplan kann durch eine leistungsfähigere Aktivitäts- und Ressourcenplanung erweitert werden. Eine grafische Zeitleistendarstellung wäre hier vorstellbar. Außerdem könnten die Testaktivitäten mit logischen Verknüpfungen versehen werden, beispielsweise mit einer Referenz auf auszuführende Testfälle, auf deren Durchführung dann automatisch hingewiesen wird.
- Eine zusätzliche Oberfläche zur Durchführung von Tests wäre eine sinnvolle Erweiterung. Auf dieser könnte man auf alle bereits durchgeführten Tests und deren Ergebnisse in übersichtlicher Form zugreifen und neue, noch auszuführende Tests einsehen. Die Informationen über noch auszuführende Tests könnten dabei direkt aus der Zeit- und Aktivitätsplanung des Testplans stammen.
- Als mögliche Erweiterung könnte die Ausführung von Tests von der Software aus gesteuert werden. Dazu sind konkrete Angaben der Ausführungskommandos (beispielsweise der Name eines Programms und der Parameter) für die Testfälle erforderlich. Die Aufrufparameter könnten sich aus den Parametern einer Testfallspezifikation zusammensetzen, so dass ganze Testreihen automatisch von der Software durchgeführt werden könnten.

5.4 Beliebige Dokumenttypen

Wie bereits erwähnt wurde, lässt sich die Software nicht nur für die Erstellung von IEEE829-Testdokumentationen einsetzen. Sie kann als allgemeines Erstellungswerkzeug für beliebige Dokumente dienen, indem zunächst entsprechende Dokumentbeschreibungsdateien angelegt werden (siehe Abschnitt 3.3.1 auf Seite 36). Insbesondere sehr stark formgebundene Dokumente (beispielsweise Formschriften wie Rechnungen, Mahnungen etc.) können mit Hilfe der Software erstellt werden.

Die folgenden Schritte sind auszuführen, um die Software für einen beliebigen Dokumenttyp anzupassen:

1. Definition des Ausgabeformats mit Hilfe einer DTD oder eines XML-Schemas. Das erzeugte Dokument muss dieser Definition genügen. Die Software selbst führt allerdings (noch) keine direkte Validierung des Ausgabedokuments durch.
2. Erzeugen der Dokumentbeschreibungsdatei, in welcher die Abschnitte und die Editoren festgelegt werden. Die derzeitige Software erwartet diese Datei unter dem Namen *app/src/xml/documents.xml*. In einer späteren Softwareversion sollte dieser Dateiname in einer geeigneten Form konfigurierbar sein.
3. Implementierung der Editoren wie in Abschnitt 4.2.3 beschrieben.

5.5 Fazit

Es wurde deutlich gemacht, dass eine vollständige Testdokumentation nach dem IEEE829-Standard sehr umfangreich sein kann. Die komplexen Beziehungen der Dokumente zueinander, aber auch die redundanten und logisch voneinander abhängigen Inhalte in verschiedenen Dokumenten erfordern bei manueller Erstellung einen hohen Erstellungs- und Kontrollaufwand, wenn man die Konsistenz der Gesamtdokumentation wahren möchte. Es wurde gezeigt, dass die Erstellung mit Hilfe einer Software deutlich vereinfacht werden kann. Dazu wurden die Dokumente des Standards weiter aufgebrochen und in ihre strukturellen Komponenten zerlegt. Dies hat es erlaubt, eine maschinenverarbeitbare Aufbauvorschrift (DTD) der Dokumente zu definieren.

Die erstellte Software kann mit Hilfe von Beschreibungsdateien Dokumente erzeugen, die einer bestimmten Form entsprechen. Für jeden Abschnitt ei-

nes Dokuments kann ein beliebiger Editor zum Einsatz kommen, so dass jede Art von Erstellungslogik und somit jede Form von Vereinfachung bei der Erstellung eines Dokuments realisiert werden kann. Die Software lässt sich damit auch für beliebige andere Dokumenttypen - nicht nur für IEEE829-Testdokumentationen - einsetzen. Die in diesem letzten Kapitel vorgestellten Erweiterungen machen deutlich, dass die Software zudem viele Möglichkeiten der Weiterentwicklung und Verbesserung bietet.

Literaturverzeichnis

- [IEEE] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC., *Organization Website*, <http://www.ieee.org>
- [IEEE-SA] IEEE STANDARDS ASSOCIATION, *Organization Website*, <http://standards.ieee.org>
- [IEEE829] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC., *Standard for Software Test Documentation*, Standard 829, Revision 1998 (IEEE Std. 829-1998)
- [W3C] WORLD WIDE WEB CONSORTIUM (W3C), *Organization Website*, <http://www.w3.org>
- [XML] WORLD WIDE WEB CONSORTIUM (W3C), *eXtensible Markup Language (XML) Website*, <http://www.w3.org/XML>
- [DTD] WORLD WIDE WEB CONSORTIUM SCHOOLS (W3SCHOOLS), *Document Type Definitions (DTD) Tutorial Website*, <http://www.w3schools.com/dtd/>
- [XSD] WORLD WIDE WEB CONSORTIUM (W3C), *XML Schema Definitions (XSD) Website*, <http://www.w3.org/XML/Schema>
- [XSL] WORLD WIDE WEB CONSORTIUM (W3C), *The eXtensible Stylesheet Language (XSL) Website*, <http://www.w3.org/Style/XSL/>
- [W3SCHOOLS] WORLD WIDE WEB CONSORTIUM SCHOOLS (W3SCHOOLS), *Organization Website*, <http://www.w3schools.com>
- [SUN] SUN MICROSYSTEMS, INC., *Organization Website*, <http://www.sun.com>
- [CVS] CONCURRENT VERSIONS SYSTEM, *Open Standard Website*, <http://www.cvshome.org>

- [JAVA] JAVA PROGRAMMING LANGUAGE, *Technology Website*, <http://java.sun.com>
- [J2SE] JAVA PROGRAMMING LANGUAGE, *Java 2 Software Development Kit Standard Edition Website*, <http://java.sun.com/j2se>
- [J-API] JAVA PROGRAMMING LANGUAGE, *Java 2 Documentation Website*, <http://java.sun.com/docs/>
- [NETBEANS] NETBEANS COMMUNITY, *Netbeans IDE Website*, <http://www.netbeans.org>
- [ANT] THE APACHE SOFTWARE FOUNDATION, *The Apache Ant Project Website*, <http://ant.apache.org>
- [TOGETHER] BORLAND SOFTWARE CORPORATION, *Togethersoft Website*, <http://www.togethersoft.com>
- [SWKRISE] BLOHBERGER, PANAGIOTIDIS, KARADAG, *Die Software-Krise*, Ausarbeitung an der Technischen Universität Berlin, Sommersemester 2000
- [SE] IAN SOMMERVILLE, *Software Engineering*, Addison-Wesley Pub Co, Sechste Auflage, August 2000, ISBN 020139815X