Kluge, René
Händelstraße 14
99610 Sömmerda
Tel.: 0179/7900913
E-Mail: R_Ke@gmx.net
Studiengang: Wirtschaftsinformatik
Matrikel-Nr.: 26083

Automatisierung der Dokumentation im Bereich der Softwareentwicklung

Studienarbeit
am Institut für Theoretische und Technische Informatik
der Technischen Universität Ilmenau

Fachbereich Prozessinformatik
Betreuender Hochschullehrer: Prof. Dr.-Ing. habil. Ilka Philippow
Weiterer Betreuer: Dipl.-Inf. Detlef Streitferdt
Starttermin: 01. Mai 2002
Abgabetermin: 01. August 2002

Kurzfassung

In der vorliegenden Arbeit wird das aktuelle Umfeld der Softwareentwicklung vorgestellt. Vor allem dieses ist momentan gekennzeichnet durch sich ändernde Rahmenbedingungen bei gleichzeitig steigenden Anforderungen. Im Rahmen der Arbeit wird dieses zunehmend schwierige Umfeld am Beispiel des Bereichs Dokumentation beschrieben.

Innerhalb eines Softwareentwicklungsprozesses fallen Dokumente an oder müssen erstellt werden. Eine Betrachtung erfolgt dahingehend unter zwei Gesichtspunkten. Erstens sorgen Dokumente, die während der Softwareentwicklung anfallen für eine Dokumentation der Softwareerstellung und sind somit ein Ansatz zur Analyse und Verbesserung dieses Prozesses. Zweitens müssen diese Dokumente in einer Art und Weise aufbereitet werden, dass eine sinnvolle Nutzung dieser möglich ist.

Im Mittelpunkt der Arbeit stehen deshalb die Anforderungen an die Dokumentation und wie diese erfüllt werden können. Die Betrachtung erfolgt in Hinblick auf einer Entlastung der Softwareentwickler. Eine häufig in deren Verantwortungsbereich liegende Dokumentation muss Anforderungen genügen, welche durch diese nicht immer gewährleistet werden können. Dazu werden zu Beginn der Arbeit Anforderungen im Zusammenhang mit aktuellen Normen, Standards und Richtlinien im Bereich der Dokumentation vorgestellt.

Unter dem Aspekt der Entlastung der Softwareentwickler erfolgt darauf aufbauend eine Vorstellung aktueller Technologien und Datenformate. Dabei stehen Potenziale und Grenzen eines Quellcodeparsers und dessen Fähigkeiten zur Informationsanalyse und -aufbereitung im Vordergrund. Eine Betrachtung erfolgt

dabei im Hinblick auf eine möglichst automatisierte Datenaufbereitung von Informationen im Quellcode hin zu sinnvoll strukturierten, verwertbaren Dokumenten. Dazu werden Möglichkeiten der Datenübertragung und -bearbeitung mit Hilfe von Auszeichnungssprachen wie XML aufgezeigt. Anschließend wird untersucht, inwieweit eine Bearbeitung der Daten durch verschiedene Textsysteme erfolgen soll.

Im darauf aufbauenden Lösungsansatz werden in Bezug auf die vorgestellten Anforderungen und Besonderheiten im Bereich der Dokumentation die Einsatzmöglichkeiten moderner Technologien vorgestellt. Es wird gezeigt, inwieweit Informationen aus dem Quellcode strukturiert aufbereitet werden können und wie mit Hilfe von XML eine Integration der Daten in den betrieblichen Dokumentationsprozess erfolgen kann. Hinsichtlich einer automatisierten Dokumentationserstellung wird weiterhin der Weg von in XML vorliegenden Daten hin zu einer sinnvollen Überführung in ein Textverarbeitungssystem aufgezeigt. Fragen, wie die Daten aufbereitet und anschließend bearbeitet werden können, finden dabei Berücksichtigung.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Im Folgenden wird sich zunächst mit dem Umfeld der Dokumentation im Rahmen eines Softwareentwicklungsprozesses auseinandergesetzt. Ansätze, welche die gewachsene Bedeutung und damit einhergehende Probleme dieses Bereiches aufzeigen, werden dabei vorgestellt. Daraufhin wird eine Zielformulierung vorgenommen, welche eine Verbesserung der Dokumentation und eine betriebliche Entlastung zum Gegenstand hat.

1.1 Umfeld: Dokumentation in der Softwareentwicklung

Durch einen raschen technischen Fortschritt und einen sich zunehmend verschärfenden Wettbewerb entwickelte sich die Softwareentwicklung zu einer umfangreichen und komplexen Tätigkeit. Daher erlangt eine Dokumentation des Softwareentwicklungsprozesses und der Ergebnisse einzelner Arbeitsschritte eine immer wichtiger werdende Bedeutung. Dabei ist die Dokumentation ein bisher eher vernachlässigter Bereich der Softwareentwicklung. Durch einen fehlenden direkt ableitbaren Nutzen, rücken Aspekte einer Erfassung und Aufbereitung von während der Softwareentwicklung entstehender Dokumente in den Hintergrund. Auch lassen sich Fehler aufgrund mangelhafter Dokumentation nicht sofort und in manchen Fällen gar nicht nachvollziehen. Die wachsende Bedeutung der Dokumentation resultiert letztendlich auch aus der Notwendigkeit, dem Softwareprodukt einen qualitativen Mehrwert durch eine hochwertige Benutzerdokumentation u.ä. zu geben. Dokumentation kann somit helfen, eine Grundlage für eine sinnvolle

Betrachtung und Nachvollziehbarkeit des Softwareentwicklungsprozesses zu schaffen.

1.2 Zielstellung

Die wachsende Bedeutung der Dokumentation ist mit wachsenden Anforderungen und steigenden Aufwänden verbunden. Der dabei auf Entwicklerseite entstehende Aufwand stellt vor allem in kleinen bis mittleren Unternehmen mit knappen humanen Ressourcen oder in zeitlich knappen Softwareprojekten ein Problem dar. Die Idee der Arbeit ist somit, Ansätze zu entwickeln, welche zeigen, inwieweit die Dokumentation automatisch erstellt werden kann. Dafür muss eine Betrachtung einzelner Bestandteile der Dokumentation, sowie eine Analyse aktueller Technologien und Werkzeuge erfolgen. Dabei spielen neben den im Softwareentwicklungsprozess entstehenden Dokumenten auch Möglichkeiten der Verarbeitung und der Überführung von Dokumenten- bzw. Datenformaten eine wichtige Rolle. Ein Ansatzpunkt ist beispielsweise der von den Entwicklern erstellte Programmcode. Programmschnittstellen, d.h. Funktionen oder Methoden, zusammen mit ihren Signaturen sind eine wichtige Voraussetzung für das Verständnis des zugrunde liegenden Softwaresystems. Aus diesen Grundelementen vorliegend in einer bestimmten Programmiersprache entstehen letztendlich die Strukturen des Systems. Die Arbeit beschäftigt sich somit vorrangig mit Ansätzen einer automatisierten Dokumentation hinsichtlich der Fragestellungen:

- Welche Anforderungen spielen im Bereich der Dokumentation eine Rolle?
- Wie und in welchem Umfang können Arbeitsschritte im Bereich der Dokumentation automatisiert werden?
- Welche existierenden Ansätze zur Automatisierung der Dokumentation gibt es bereits?
- Welchen Beitrag kann eine sinnvolle Strukturierung in Rahmen einer automatisiert erstellten Dokumentation leisten?

1.3 Übersicht

Zunächst erfolgt eine Einordnung der Dokumentation in den Softwareentwicklungsprozess. Dabei wird auf aktuelle Standards und Werkzeuge in der Dokumentation eingegangen. In diesem Zusammenhang werden aktuelle Datenformate im Hinblick auf eine automatisierte Erstellung mit Hilfe von XML vorgestellt. Darauf aufbauend wird ein Lösungsweg entwickelt und bewertet, worauf im Anschluss die konkrete Umsetzung aufgezeigt wird.

2 Das aktuelle Umfeld der Dokumentation

Dokumentation entwickelt sich immer mehr zu einem zentralen Bestandteil der Softwareentwicklung. Aufgrund gewachsener Anforderungen auf der Nutzerseite und steigenden Differenzierungsdrücken[1] in den Produkten, entstand die Forderung nach qualitativ höheren Programm- und Produktdokumentationen. Einen Überblick über das folgende Kapitel gibt Abbildung 2.1:

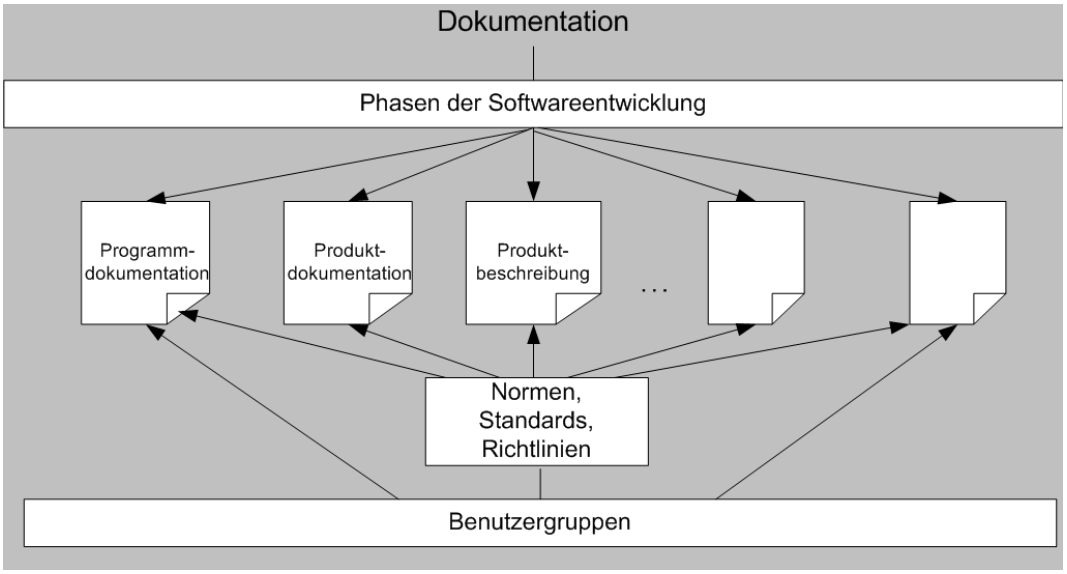

Abbildung 2.1: Überblick über Kapitel 2

[1] vgl. beispielsweise [Mey]
sowie : [Eis]

Die neuen vorhandenen Möglichkeiten von Dokumentationstechnologien, -Formaten und -Standards leisten hier einen entscheidenden Beitrag[2] und werden im Folgenden vorgestellt. Zunächst wird jedoch der Begriff der Dokumentation in den Softwareentwicklungsprozess eingeordnet und anschließend im Hinblick auf aktuelle Entwicklungen und damit einhergehenden Problemen vorgestellt[3].

2.1 Begriffsklärung

Der Begriff der Dokumentation wird von vielen Autoren unter verschiedenen Blickwinkeln betrachtet. Dies liegt hauptsächlich an den vielfältigen Einsatzmöglichkeiten der bei der Dokumentationserstellung anfallenden Dokumente. So muss beispielsweise der Sinn der Nutzung einer Dokumentation berücksichtigt werden. Allein eine recht grobe Untergliederung in *Projektdokumentation*, *Entwicklungsdokumentation*, *Produktdokumentation* und *Benutzerdokumentation* zeigt die Schwierigkeit in der Abgrenzung des Begriffs „Dokumentation". Die Frage, welche sich jedoch stellt, ist vielmehr: Aus welchen Bestandteilen besteht eine Dokumentation und wie entsteht diese im Rahmen einer Softwareentwicklung? Auch besteht das Problem, ob „die Dokumentation" oder jeweils einzelne Dokumente betrachtet werden müssen.

Gemeinsam haben die in der Literatur aufgeführten Definitionen eine Beschreibung als eine Sammlung und Darstellung von Informationen für verschiedene Zwecke, wobei der Prozess „*Dokumentieren*" und das Produkt „*Dokumentation*" unterschiedlich gewertet werden[4]. Als Prozess wird die Erstellung verschiedener Dokumente im Verlauf der Softwareentwicklung verstanden. Das Ergebnis dieses Prozesses sind Dokumente im Sinne eines Produktes, welches dabei einen Bestandteil der gesamten Dokumentation darstellt.

Die folgenden Ausführungen beziehen sich auf die Definition von *Hoffmann/-Schlummer (1990)*. Diese verwenden den Begriff der „*technischen Dokumentation*" zur Verdeutlichung des rein sachlichen Zusammenhangs von Prozess und Produkt. *Technische Dokumentation* wird definiert als „*die strukturierte Sammlung*

[2] vgl. [Bul]
[3] vgl. [Leh94] S.67ff.,S.79ff.
[4] vgl. [Som92] S.571ff.

aller notwendigen und zweckorientierten Informationen über ein auf technischem Wege hergestelltes Produkt und seiner Verwendung". Mit dieser Definition werden somit verschiedene Dokumente zugelassen[5]. Jedoch bleibt vor allem der Prozess wie eine derartige *Sammlung strukturierter Informationen* entsteht, unerwähnt. Dies zeigt inwieweit die Dokumentation einen zunehmenden Bedeutungswandel unterworfen ist. Im Verlauf der Arbeit wird sich zeigen, dass der Prozess der Dokumentationserstellung immer mehr in den Mittelpunkt der Betrachtung rückt und der Begriff der Dokumentation im Sinne eines Produktes wie ihn *Hoffmann/-Schlummer* verwendeten das Ergebnis dieses Prozesses darstellt.

2.2 Einordnung der Dokumentation in die Phasen eines Softwareprojektes

Neben einem notwendigen und innerbetrieblich formal festgelegten Aufbau und einer damit einhergehenden Abgrenzung der mit der Dokumentation betrauten Aufgabenträger, erfolgt eine zunehmend prozessorientierte Betrachtung der Dokumentation. Dokumente fallen im Softwareentwicklungsprozess an, fließen zusammen oder ergänzen in übergeordneten Dokumenten einander. Für eine gezielte Betrachtung eines solchen Softwareprojektes bestehen verschiedene Phasenkonzepte und Vorgehensmodelle. Diese unterscheiden sich vor allem hinsichtlich:

- der zeitlichen Abfolge der Bearbeitung von Aufgaben,
- der Annahmen über die Form der Arbeitsteilung, der Abstimmung von Teilaufgaben und des Aufbaus der Organisation,
- sowie der Empfehlungen zu Dokumentation, Erzeugung von Zwischenprodukten und bestimmter Hilfsmittel.

Das Grundkonzept ist meist eine sequentielle Abfolge der Phasen mit klar zuordenbaren Aufgaben und Ergebnissen. Somit sind auch bestimmte Dokumente das Ergebnis dieser Phasen. Die Anwendung derartiger Konzepte soll für eine bessere Planbarkeit, Organisation und Kontrolle von Softwareprojekten sorgen[6]. In der Realität ist eine strikte Einhaltung eines solchen Phasenkonzepts in der

[5] vgl. [Leh94] S.3ff
[6] vgl. [Leh94] S.74ff.

Regel nicht möglich. Dadurch existieren zahlreiche Modellvarianten, welche die strenge sequentielle Abfolge aufweichen. Im Folgenden wird eine beispielhafte Zuordnung von Dokumenten zu einzelnen Phasen vorgenommen. Diese Zuordnung ist keinesfalls starr, sondern muss für das jeweils zugrunde liegende Konzept angepasst werden. Auch schwankt die Anzahl der in einem Softwareprojekt anfallenden Dokumente je nach Projektgröße und den innerbetrieblichen Vorschriften zur Dokumentation.

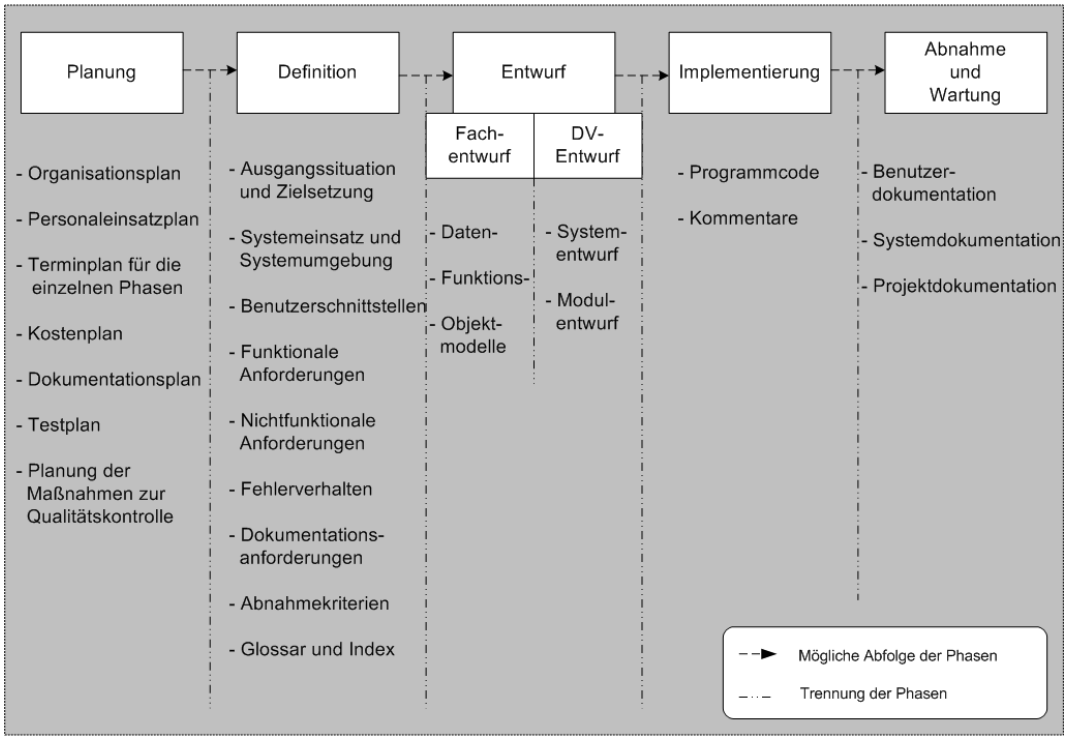

Abbildung 2.2: Phasen und dabei anfallende Dokumente

Die Planungsphase beinhaltet vorrangig die Projektidee und notwendige Voruntersuchungen hinsichtlich technischer Machbarkeit und Wirtschaftlichkeit. Ausgangspunkt für die Weiterentwicklung ist die in dieser Phase entstehende Entwicklungsgenehmigung[7]. Der Organisationsplan legt dabei die für die Entwicklung notwendigen organisationellen Einheiten fest, wobei die einzelnen Aufgabenträger im Personaleinsatzplan aufgeführt werden. Der Terminplan beschreibt eine genau zeitliche Strukturierung der einzelnen Phasen und legt fest mit welchem Ereignis bzw. Ergebnis eine Phase beendet ist. Eine Vorausplanung entstehender Aufwände stellt der Kostenplan dar. Vor allem die darin betrachteten wirtschaftlichen Aspekte sind die Grundlage für die resultierende Entwicklungsgenehmigung. Der Dokumentationsplan hält fest, wann und in welchem Umfang Dokumente im Rahmen der Softwareentwicklung anzufertigen sind. Im Testplan wird beschrieben,

[7] vgl. [Mer00] S.145-149

wann und in welchem Umfang Funktionalitäten der Software getestet werden sollen. Der Qualitätsplan beinhaltet die Maßnahmen, welche zur Erreichung einer bestimmten Qualität notwendig sind. Dies kann sowohl Einfluss auf das Produkt und die damit verbundenen Anforderungen als auch Einfluss auf den Entwicklungsprozess im Sinn einer zeitlichen Verzögerung oder Beschleunigung haben. Je nachdem, wann ein bestimmtes qualitatives Niveau des Produktes erreicht wurde, kann in Abstimmung mit dem Zeitplan eine neue Phase begonnen werden.

In der Phase der Definition werden beispielsweise die Punkte Ist-Analyse, Soll-Konzept, Spezifikation der funktionalen Anforderungen und Vorgaben für den Entwicklungsprozess behandelt. Das dabei entstehende Ergebnis ist das Pflichtenheft. Dies umfasst eine Beschreibung der Ausgangssituation und eine mit dieser verbundenen Zielsetzung. Aufgrund der in der Ausgangssituation vorliegenden Probleme und Anforderungen müssen Systemeinsatz und Systemumgebung festgelegt werden. Auch die notwendige Benutzerschnittstelle, das Fehlerverhalten und die Abnahmekriterien im Sinn der Zielstellung müssen dabei Berücksichtigung finden.

Die Entwurfsphase ist zunächst in den Fachentwurf und datenverarbeitungstechnischen Entwurf zu untergliedern. Der Fachentwurf beinhaltet die Beschreibung von Funktionen und Daten und konkretisiert die in der Definitionsphase spezifizierten Anforderungen. Das Resultat sind Daten-, Funktions- und Objektmodelle. Ein Datenmodell beinhaltet die statische Struktur der Daten, sowie deren Beziehungen untereinander. Funktionsmodelle beinhalten für das zu entwickelnde System relevante Funktionen, die ein Verständis des Gesamtsystems aus funktionaler Sicht ermöglichen. Sie beschreiben vorrangig die Inhalte der Funktionen und deren Zusammenhänge. Objektmodelle beinhalten letztendlich Daten und Funktionen welche durch Objekte zusammengefasst werden. Ein Objekt definiert sich letztendlich aus seinen Daten (Attributen) und Funktionen (Methoden). Der datenverarbeitungstechnische Entwurf baut auf dem Fachentwurf auf und berücksichtigt Hard- und Software- sowie Entwicklungsumgebung. Ergebnisse sind dabei beispielsweise erste Programmkomponenten und deren Beziehungen, Module inklusive Funktionen und Beziehungen sowie deren Aufrufreihenfolge.

Die Phase der Implementierung umfasst die Detaillierung des Systementwurfs und die Umsetzung mit Hilfe der Entwicklungsumgebung (z.B. Codierung). Es

entstehen die Programmabläufe, Benutzeroberflächen sowie implementierte und integrierte Komponenten, welche auch in dieser Phase getestet werden.

Die Phase der Abnahme und Wartung baut zunächst auf einer Überprüfung der Anforderungen des Pflichtenheftes auf und umfasst eine technische bzw. organisatorische Integration des Produktes. Je nach Umfang der einzuführenden Software werden beispielsweise Schulung und Betreuung der Benutzer notwendig. Neben dem eingeführten System entstehen beispielsweise das Abnahmeprotokoll und Benutzerunterlagen. Auch findet die Dokumentation des Softwareprojektes hier ihren Abschluss. Der Begriff der Wartung beinhaltet Aufgaben der Fehlerbeseitigung, Programmänderung, Programmerweiterung und Programmanpassung.

Die beschriebenen Dokumente fallen während des Entwicklungsprozesses an und werden in elektronischer- oder Papierform hinterlegt. Eine strenge Zuordnung auf einzelne Phasen ist dabei häufig nicht gegeben. Oft sind Änderungen und Überarbeitungen aufgrund wechselnder Anforderungen an die zu entwickelnde Software notwendig. Abhängig von den im Personaleinsatzplan festgelegten Verantwortlichkeiten kann und muss auf die einzelnen Dokumente Einfluss genommen werden. Darin liegt jedoch eine häufige Fehlerursache innerhalb der Dokumentation begründet. Bei sich ändernden Anforderungen wird in der Regel eine Anpassung sichtbarer Bestandteile des endgültigen Produktes vorgenommen. Dies umfasst Bestandteile, wie etwa Funktionalitäten der Software. Eine Berücksichtigung nicht direkt sichtbarer Elemente, wie sie die Dokumentation darstellt, wird dabei häufig vernachlässigt. Deshalb liegt eine Wahrung der Konsistenz zwischen Bestandteilen der Software und der zugehörigen Dokumentation häufig in der Verantwortung der Softwareentwickler.

2.3 Dokumente und Benutzergruppen

Die endgültige Dokumentation entsteht keineswegs stets am Ende eines Softwareentwicklungsprozesses[8]. Vielmehr ist in den letzten Jahren eine Überprüfung der Qualität eines Produktes am Ende der Entwicklung einer verfeinert phasenbezogenen Zwischenüberprüfung gewichen[9]. Somit entwickelt sich die Dokumentation

[8] vgl. [Sne91] Software Wartung Kap. 6
[9] vgl. [Sta99] S.324f. sowie Kap. 2.1

aufgrund der in der Softwareentwicklung anfallenden Dokumente selbst zu einem parallel ablaufenden Prozess. Dokumente müssen abhängig von der Phase ihrer Entstehung für verschiedene Einsatzmöglichkeiten erstellt, aufbereitet, verwaltet und gegebenenfalls überarbeitet werden[10]. Vor allem verkürzte "time to market" - Prozesse sorgen für den Druck, Kunden frühzeitig über Softwareprodukte zu informieren und auf dem aktuellen Stand der Entwicklung zu halten[11]. Dies betrifft vor allem Werbe- und Kommunikationsmöglichkeiten zu potentiellen Kunden in bereits frühen Phasen der Softwareentwicklung. Eine Berücksichtigung aller Dokumente und deren Widerspruchsfreiheit, sowie ein nachvollziehbarer Bezug zu vorhergehenden Veröffentlichungen, ist entscheidend. Weiterhin ist zu berücksichtigen, dass ein Großteil der Dokumente für betriebsinterne Zwecke genutzt wird. Neben Dokumenten im Bereich der Programmierung spielen vor allem Dokumente, welche den Entwicklungsprozess als solchen dokumentieren, einen Erfahrungsschatz für das Unternehmen dar. Eine sinnvolle Verwaltung dieser und ein zugrunde liegendes betriebsinternes Wissensmanagement gewinnen zunehmend an Bedeutung und sind ein entscheidender qualitativer und zeitlicher Wettbewerbsfaktor[12]. Auch sind Chancen/Risiken-Abwägungen und Aufwandsabschätzungen anhand abrufbarer Erfahrungen ein wichtiger Aspekt bei der Optimierung betriebsinterner Prozesse[13]. Somit muss der Prozess und das Produkt „*Dokumentation*" unterschieden werden. Um einen Softwareentwicklungsprozess transparent zu gestalten und Prozesse zu optimieren, sind zahlreiche Dokumente notwendig, welche letztendlich in das Produkt „*Dokumentation*" einfließen. Derartige Dokumente sind beispielhaft in Abbildung 2.3 skizziert[14]:

[10] siehe: [Schb]
[11] vgl. [Cor]
[12] vgl. [Tha01]
sowie: [Gru]
[13] vgl. [MH99]
[14] vgl. [Hit99] S. 20ff.

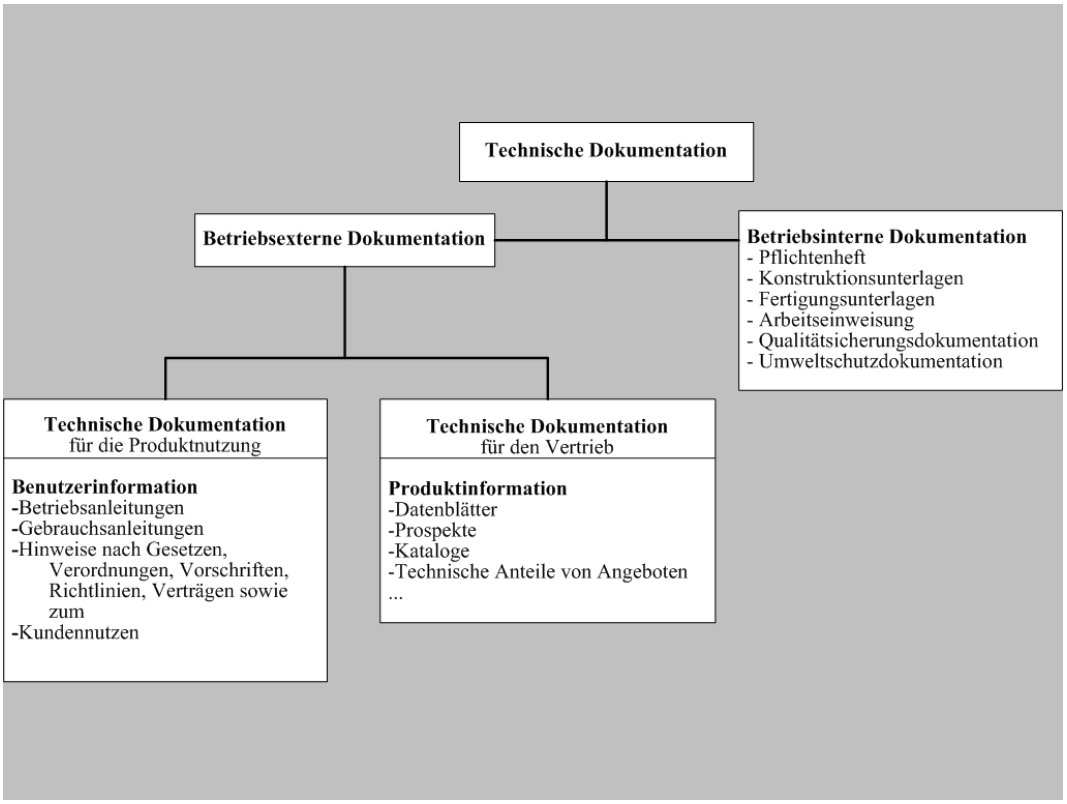

Abbildung 2.3: Gliederung anfallender Dokumente

Eine derartige Untergliederung macht in dem Zusammenhang Sinn, da im Softwareentwicklungsprozess anfallende Dokumente gezielt zugeordnet und hinterlegt werden können, was wiederum die Grundlage für eine fundierte Handhabung und Archivierung aller Dokumente ist. In der Realität überschneiden sich jedoch zahlreiche Bereiche in denen die Dokumente letztendlich anfallen. Auch liegt das Hauptaugenmerk der erstellten Dokumente vorrangig im Bereich der betriebsexternen Dokumentation, wobei die meisten Dokumente auch für betriebsinterne Verwendungen herangezogen werden. Folgende Ausführungen beschäftigen sich deshalb mit den zentralen Bestandteilen einer Dokumentation.

1. *Beschreibung des Produktes*

   In diesem Bereich steht vor allem eine allgemeine Vorstellung des Produktes im Vordergrund. Die Idee, welche hinter der Software steht und welche Funktionalitäten nutzbar sind, werden hier beschrieben. Neben einer Auf-

bereitung im externen betrieblichen Umfeld werden diese Dokumente häufig betriebsintern zur Erklärung für nicht an der Entwicklung beteiligter Mitglieder genutzt. Häufige Bestandteile einer Produktbeschreibung sind:

- *Überblicksartige Beschreibung*: Darin werden Ausgangssituation und Lösungsideen behandelt. Ein- und Ausgabeverhalten der Software sowie Analyse- und Entwurfsdokumente und Verzeichnisse der einzelnen Softwarebestandteile sind weitere Bestandteile.
- *Anwendungsfallbeschreibung*: Anhand eines einfachen Beispiels werden Funktionalitäten der Software vorgestellt.

2. *Dokumentation des Programms*

   Diese dient vorrangig dem Verständnis von Systeminterna. Einzelne Programmkomponenten werden hier aufgelistet und genau beschrieben. Dazu gehören alle Unterlagen, welche für den Betrieb und die Nutzung des Systems notwendig sind und die das System selbst dokumentieren. Die Programmdokumentation ist im betriebsexternen Umfeld eine wichtige Grundlage zur Überprüfung einzelner Anforderungen an die Software und für deren Einführung. Jedoch stellt sie im betriebsinternen Umfeld einen wichtigen Aspekt in der Weiterentwicklung und Wartung der Software dar. Bestandteile sind:

   - *Modelle*: Damit sind vor allem Programmablaufpläne und Unterlagen, welche Zusammenhänge grafisch abbilden, gemeint.
   - *Referenz-Handbuch*: Dieses beinhaltet alle Bestandteile des Programmquellcodes und beschreibt die Zusammenhänge zwischen den Bestandteilen der einzelnen Elementen.
   - *Systemumgebung*: In dieser werden die Voraussetzungen zum fehlerfreien Betrieb der Software beschrieben, beispielsweise: benötigte Hardware, erforderliche Daten, Abhängigkeiten von Betriebssystemen u.ä..

3. *Dokumentation des Produkts*

   Innerhalb dieser werden vorrangig Aspekte im fehlerfreien Umgang im alltäglichen Betrieb der Software beschrieben. Hier stehen ausführliche und mit Erläuterungen versehene Anwendungsbeispiele für alle Funktionen der

Software im Mittelpunkt. Die Produktdokumentation wird deshalb vorrangig für das betriebsexterne Umfeld erstellt. Bestandteile sind:

- *Benutzerdokumentation*: Hier werden organisationelle Voraussetzungen oder notwendige Umgebungen für den Einsatz der Software beschrieben.
- *Benutzerhandbuch*: In diesem wird der Zugriff auf die einzelnen Funktionalitäten der Software und die korrekte Nutzung einzelner Programmelemente, sowie die Handhabung der Daten beschrieben.
- *Sonstige Hinweise*: Dieser nicht zwingende Bestandteil beschreibt beispielsweise Bedeutungen von Fehlermeldungen und die entsprechenden Möglichkeiten der Reaktion auf diese.

Den aufgeführten Dokumenten kommen dabei im Rahmen des Softwareprozesses verschiedene Rollen und Bedeutungen zu[15]:

- Kommunikationsmittel zwischen „Stakeholdern" (von an der Entwicklung beteiligter Mitglieder),
- Systemspezifisches Informationsarchiv für die Wartung,
- Hilfsmittel für das Management zur Planung der Finanzierung und des Zeitmanagements,
- Anleitungen (Benutzeranleitung, Programmbeschreibungen, u.ä.),
- Grundlage für die Modellierung und Implementierung.

Die aufgeführten Rollen und Bedeutungen sind für den betrieblichen Dokumentationsprozess wichtige Kriterien. Es sollten derartige während der Softwareentwicklung anfallende Dokumente ihrer Rollen entsprechend genutzt werden, angefangen von informalen Arbeitsblättern bis zu aufbereiteten Dokumentationen. Häufig werden diese Aufgaben von den Softwareentwicklern selbst erfüllt, jedoch geht die Entwicklung hin zu einer professionellen Unterstützung dieser, beispielsweise durch das Berufsfeld des „*Technischen Redakteurs*".

Wie bereits anhand der zahlreich zu erstellenden Dokumente erkennbar, ist eine zielgruppenbezogene Abstimmung der Dokumentation notwendig[16]. Abbildung

[15] vgl. [Som92] S. 572ff.
[16] vgl. [Leh94]S. 38ff

2.4 stützt sich zunächst auf die bereits aufgezeigte Untergliederung in betriebsinterne und betriebsexterne Dokumentation:

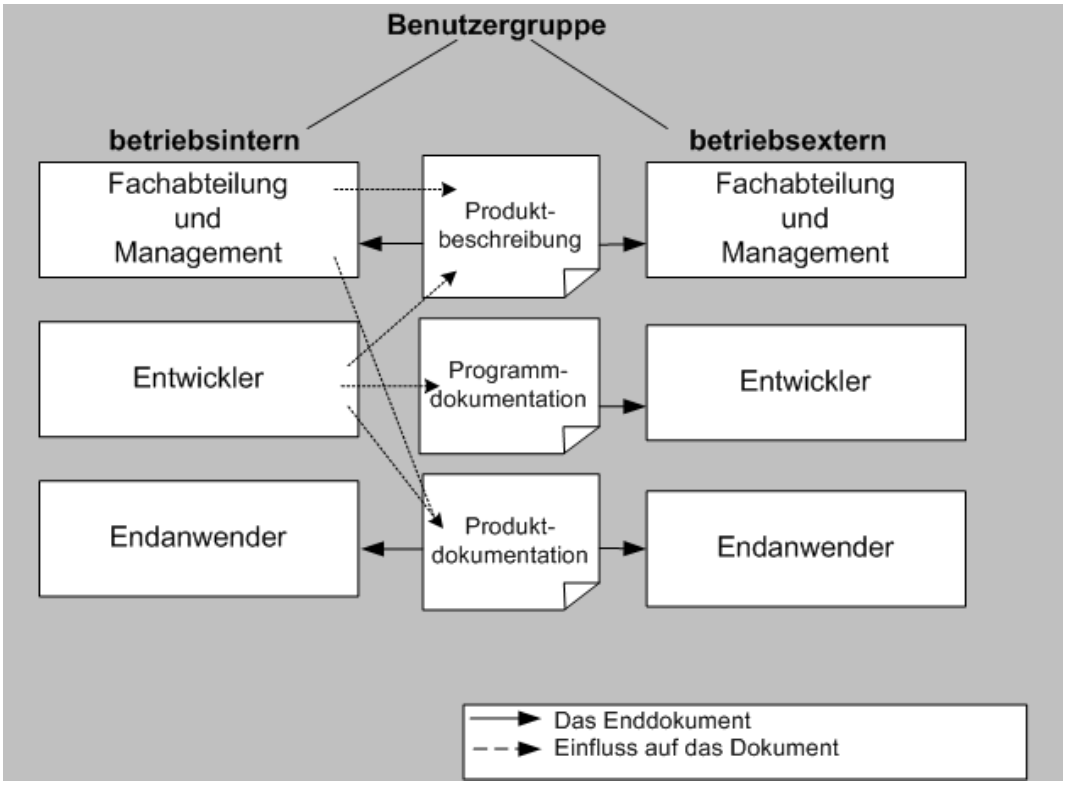

Abbildung 2.4: Benutzergruppen

Im innerbetrieblichen Bereich leistet eine Dokumentation einen wertvollen Beitrag für die Weiterentwicklung und Wartung der Systeme. Jedoch besteht auch hier die Notwendigkeit einer differenzierten Betrachtung einzelner Zielgruppen. Während für die Systementwickler das technische Verständnis von Systeminterna im Vordergrund steht, bestehen für Fachabteilungen ganz andere Anforderungen an die Dokumentation. Es müssen Dokumente erstellt werden, welche den Fachabteilungen die Möglichkeit bieten wesentliche Eigenschaften der Software zu verstehen. Erst dadurch werden Informationen für diese nutzbar, welche beispielsweise für die Vermarktung und den Kundenkontakt aufbereitet werden müssen.

Auch im betriebsexternen Umfeld besteht die Notwendigkeit einer differenzierten

Betrachtungsweise unterschiedlicher Benutzer. So müssen beispielsweise Funktionsweise und Systeminterna für andere Entwickler oder Administratoren klar und deutlich aufbereitet werden. Insbesondere Informationen für die Einführung der Software sind dabei betroffen. Dies steht im Gegensatz zu Inhalten für Fachabteilungen und Management. Hier spielen vor allem die Vorteile und Einsatzmöglichkeiten des Produkts und damit einhergehende Anforderungen und Aufwände eine Rolle. Zudem müssen Aufbereitungen für die Anwender der Produkte erstellt werden, da hier die ordnungsgemäße Nutzung der Software im Vordergrund steht.

Somit unterscheidet die Benutzergruppe nicht nur den Detaillierungsgrad, sondern auch die Ausführlichkeit der Dokumentation, wobei beide Begriffe durchaus unterschiedliche Informationen einschließen. Je nachdem, für welche Zielpersonen die Dokumentation zu entwickeln ist, müssen andere Informationen dargestellt und beschrieben werden. Der Detaillierungsgrad beinhaltet dabei, inwieweit und in welcher Tiefe Informationen dargestellt werden. Es muss dem jeweiligen Benutzer entsprechend entschieden werden, welche Informationen relevant sind und deshalb in das jeweilige Dokument übernommen werden müssen. Der Begriff der Ausführlichkeit umfasst den Umfang der Beschreibung von Informationen. Oft findet die Unterscheidung in Benutzergruppen auch eine Berücksichtigung im Schreibstil. In das System unterweisende Anleitungen bedürfen einer anderen Aufbereitung als beispielsweise die Beschreibung von Systeminterna. Qualitative Anforderungen an ein Dokumentation leiten sich somit aus den spezifischen Anforderungen der Benutzergruppe ab.

Es existieren also nicht nur technische und systemspezifische Anforderungen, sondern auch qualitative Aspekte zusammen mit unterschiedlichen Sichtweisen innerhalb der Dokumentation. Dies wird vor allem mit dem Entstehen neuer Berufe deutlich, wie beispielsweise dem des „*Technischen Redakteurs*" und den zahlreichen Dienstleistungsangeboten im Bereich der Dokumentation[17]. So existieren beispielsweise zahlreiche Angebote verschiedener Unternehmen, eine Dokumentation gemäß den Anforderungen der *DIN ISO9001*[18] zu erstellen.

[17] vgl. [Leh94] S.102ff.
sowie: [Scha]
[18] vgl. Kap.2.4.3

2.4 Anforderungen an Dokumentationen

Die Anforderungen an die Dokumentation ergeben sich zum einen aus wachsenden Ansprüchen im betrieblichen Umfeld und zum anderen aus den qualitativen Forderungen nationaler und internationaler Standards im Bereich der Dokumentation. Im Folgenden wird zunächst das in den letzten Jahren stetigen Veränderungen unterworfene Feld der Dokumentation und damit einhergehende Anforderungen vorgestellt, worauf aktuelle Entwicklungen im Bereich nationaler und internationaler Standards aufgezeigt werden.

2.4.1 Allgemeine Anforderungen

Die Entwicklung der letzten Jahre geht von einer vorrangigen Betrachtung der reinen Programmierproduktivität, hin zu einer ganzheitlichen Betrachtung des Softwareproduktes. Somit wachsen die Ansprüche an Produkte, die vormals als Nebenprodukte der Softwareentwicklung anfielen. Immer häufiger werden qualitative Ansprüche an derartige Nebenprodukte Bestandteil von Softwareverträgen[19]. Diese Entwicklung zwingt nun zu einer differenzierteren Betrachtung des Softwareprozesses[20]. Der in der Abbildung 2.5 aufgezeigte Überblick zeigt den Umfang und die gewachsene Bedeutung der Dokumentation. Ohne die entscheidenden Informationen im Umgang mit der Software an sich, sind deren Funktionalitäten und damit zusammenhängende Qualitäten entweder schwer zugänglich oder nicht nutzbar.

[19] vgl. beispielsweise: [Loh]
[20] vgl. Kap. 2.1

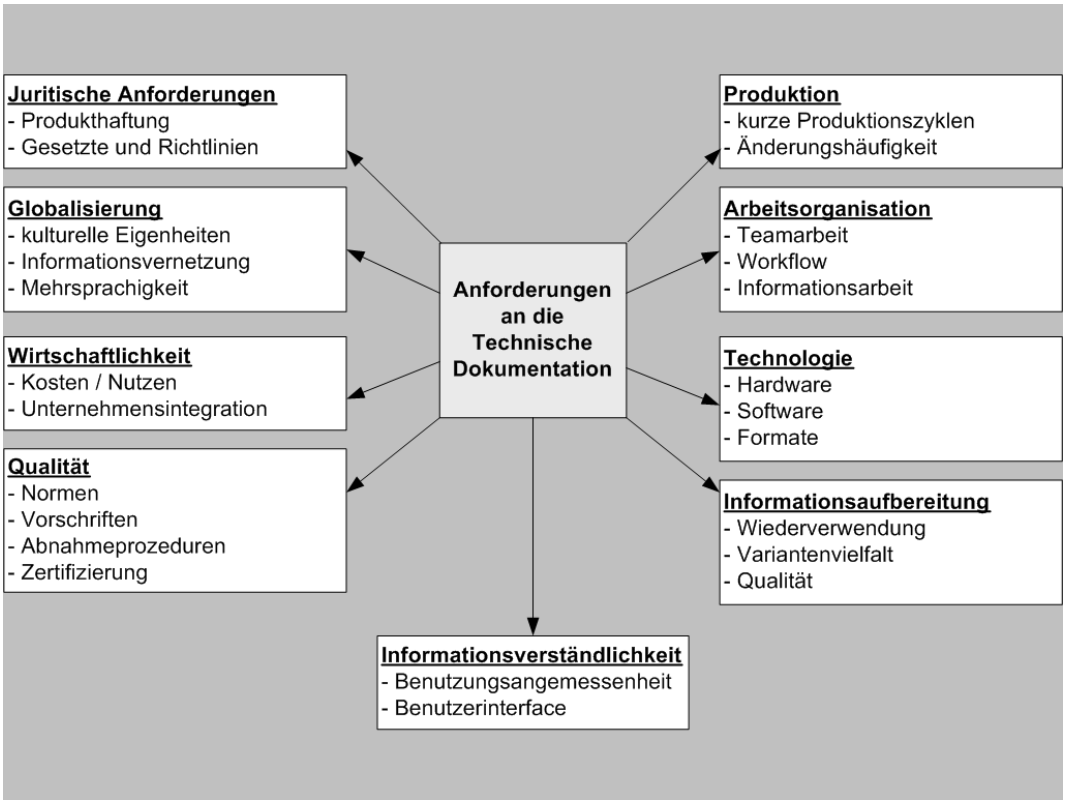

Abbildung 2.5: Anforderungen an technische Dokumentation : Überblick

Die juristischen Anforderungen beinhalten Grundsätze, welche in die Softwareentwicklung einfließen müssen. Insbesondere relevant wird dies, sobald derartige Anforderungen Bestandteil von Hard- und Softwareverträgen werden. Dadurch wird ein verantwortungsvoller Umgang mit den Informationen und in der Aufbereitung dieser notwendig.

Der Begriff der Globalisierung beinhaltet in dem hier dargestellten Zusammenhang eine Vergrößerung der Benutzergruppen. Mit der Ausweitung der Geschäftsaktivitäten auf ein internationales Umfeld muss die Dokumentation den jeweiligen nationalen Ansprüchen genügen. Dies beinhaltet eine Anpassung hinsichtlich Sprache, Layout und kulturellen Eigenheiten der Benutzergruppe.

Die Wirtschaftlichkeit umfasst Aspekte einer gezielten Kostenanalyse unter dem aktuell wachsenden Konkurrenzdruck im Bereich der Softwareentwicklung. In deren Rahmen muss abgewogen werden, in welchem Umfang und in welcher Qualität

eine Dokumentation notwendig ist und inwieweit Unternehmensbereiche, welche über die reine Softwareentwicklung hinausgehen, an der Dokumentation beteiligt sein müssen.

Der Begriff der Qualität beschreibt Anforderungen, welche Aufgrund aktueller Normen, Standards und Richtlinien in die Dokumentation einfließen sollten[21]. Hier muss entschieden werden, welche Aufwände notwendig sind um den qualitativen Ansprüchen zu genügen.

Die Anforderung der Produktion befasst sich mit dem zunehmend schwierigen Umfeld der Softwareentwicklung. Die damit zusammenhängenden verkürzten Produktionszyklen und häufig notwendigen Verbesserungen und Anpassungen der Produkte betreffen auch die Dokumentation. Dokumentation muss unter diesem Aspekt möglichst zügig und dabei jedoch den hier aufgeführten Anforderungen genügend erfolgen.

Eine weiterer Aspekt ist die organisationelle Gestaltung der an der Dokumentation beteiligten Unternehmensbereiche. Dies ist insbesondere mit den bereits aufgeführten Anforderungen von Bedeutung.

Weiterhin ist bei der Erstellung der Dokumentation auf die zugrunde liegende Technologie zu achten. Dokumentation von Software bedarf beispielsweise einer anderen Aufbereitung als die Dokumentation von Hardware. Auch muss die steigende Verfügbarkeit digitaler und multimedialer Dokumentationstechnologien berücksichtigt werden. So wird beispielsweise einer „Online"-Dokumentation die neben der Software selbst verfügbar ist, eine zunehmende Bedeutung zugerechnet.

Die Anforderungen an die Informationsaufbereitung resultieren zum einen aus den bereits aufgeführten Anforderungen und zum anderen aus einer möglichst effizienten Verwendung bereits angefallener Dokumente. Hier werden Fragen nach Wiederverwendbarkeit sich in der betrieblichen Praxis bewährter- und nach der Anzahl notwendiger Varianten der Dokumente gestellt.

Die Informationsverständlichkeit resultiert letztendlich aus einer Analyse und Berücksichtigung der Benutzergruppen. Je nachdem, für welche Gruppe die Dokumentation entwickelt wurde, müssen verschiedene Aspekte in das Produkt ein-

[21] vgl. Kap.2.4.3

fließen[22].

Wie zum Teil beschrieben, ist eine alleinige Betrachtung einzelner Anforderungen nur schwer möglich. Vielmehr scheint eine Berücksichtigung aller und eine entsprechende Ableitung in betriebsinternen Richtlinien sinnvoll. Um den gezeigten Anforderungen zu genügen, sind zudem nicht unerhebliche Investitionen in qualifiziertes Personal und eine Neuorientierung der organisationellen Ausrichtung der Dokumentenverwaltung und des für die Softwareentwicklung verantwortlichen Mitarbeiterstabes notwendig. Die Probleme, welche im Zusammenhang mit der Erfüllung dieser Anforderungen entstehen, sind dabei keinesfalls typische Probleme allein im Bereich der Dokumentation. Vielmehr sind diese Anforderungen auch auf andere Bereiche übertragbar und eher branchen- als bereichstypisch. Jedoch kann mit Hilfe wachsender Qualitäten, in den bei der Softwareentwicklung anfallenden Dokumenten eine wertvolle Hilfe für spätere Projekte gesehen werden. Schätzungen möglicher Aufwände und eine qualitative Weiterentwicklung ist nur anhand der Nachvollziehbarkeit vergangener Projekte möglich[23]. Ein mögliches Resultat wäre dabei eine Wiederverwendung bereits bewährter Verfahren zur Dokumentationserstellung bzw. bereits erstellter Dokumente, welche den gezeigten Anforderungen genügen. Somit stellen die gezeigten Anforderungen aufgrund ihrer Komplexität nicht sofort realisierbare, jedoch klar definierte Zielkriterien dar[24].

2.4.2 Qualitative Anforderungen an die Dokumente

Eine sehr häufig verwendete Interpretation des Qualitätsbegriffes oder des Qualitätsverständnisses lautet, dass unter Qualität die umfassende Erfüllung der Kundenanforderungen zu verstehen ist[25]. Werden diese Forderungen oder Erwartungen erfüllt, die sowohl subjektiver als auch objektiver Natur sind, wird der Kunde mit der erbrachten Unternehmensleistung zufrieden sein. Der aus dieser Formulierung abgeleitete Umkehrschluss erlaubt, dass auftretende Fehler oder Qualitätsmängel als „*Nichterfüllung einer festgelegten Forderung*" interpretiert werden können. In diesem Sinne ist auch in der *DIN EN ISO 8402* die Fehlerdefinition

[22] vgl. Kap.2.3
[23] vgl. [Tha01]
[24] vgl. [Leh94] S. 48ff., S.81ff.
[25] vgl. [Tha01] Kap.1

festgelegt[26]. Im Bereich der Dokumentation betrifft dies vor allem das endgültige Produkt „*Dokumentation*". Qualitativ hochwertige Dokumentation ist auch ein entscheidender Bestandteil qualitativ hochwertiger Programme[27]. Mögliche Kriterien für eine Überprüfung qualitativer Ansprüche in endgültige Dokumente wären:

- *Verfügbarkeit*: Technische Dokumentation soll zur richtigen Zeit zu angemessenen Kosten bereitgestellt werden.
- *Eignung*: Technische Dokumentation soll mit den Zielen und Einsatzmöglichkeiten, für die sie gedacht ist, übereinstimmen. (Zielgruppenanalyse).
- *Dokumentzugänglichkeit*: Technische Dokumentation soll für die Anwender lesefreundlich organisiert sein, so dass sie schnell die Informationen finden, die sie aktuell benötigen.
- *Lesbarkeit/Verständlichkeit*: Technische Dokumentation soll für die jeweiligen Adressaten flüssig, einfach zu lesen und leicht verständlich sein.
- *Vollständigkeit*: Bei den meisten Dokumentationen vom Typ „Nachschlageanleitung" muss die Vollständigkeit der Informationen gewährleistet sein, bei Kurz- und Lernanleitungen hingegen nicht.
- *Richtigkeit*: Sollte eigentlich eine Selbstverständlichkeit in allen Dokumentationsformen sein.
- *Sicherheit der Anwender bei der Nutzung* eines Produktes ist oberstes Ziel der einschlägigen gesetzlichen und normativen Bestimmungen zur technischen Dokumentation. Sicherheitshinweise und Warnungen sind ein wichtiger Beitrag dazu[28].

2.4.3 Anforderungen anhand von Standards, Normen und Richtlinien

Im Bereich der Dokumentation existieren zahlreiche Normen, Standards und Richtlinien. Im Folgenden wird der Begriff Standard als „*ein technisches Do-*

[26] siehe: [Bin]
[27] vgl. [Som92] S.577
[28] siehe: [Schb]

kument, das der Öffentlichkeit zur Verfügung steht" verstanden. „*Es wird unter der Beteiligung aller interessierter Parteien entwickelt und findet deren Zustimmung. Der Standard basiert auf den Ergebnissen aus Wissenschaft und Technik, bezieht Experimente mit ein und zielt darauf ab, das Gemeinwohl zu fördern*" [29].Die folgenden Ausführungen erheben aufgrund der Fülle verschiedenster nationaler bzw. internationaler und branchenspezifischer bzw. branchenübergreifender Standards keinen Anspruch auf Vollständigkeit. Vielmehr sollen aktuelle Entwicklungen in diesem Bereich vorgestellt werden. Die Relevanz ergibt sich sowohl aus der Verwendung für betriebsinterne Qualitätsrichtlinien, als auch durch die Integration der Normen, Standards und Richtlinien in Hard- und Softwareverträge. In diesem Bereich haben sich Normungsinstitute, wie *DIN*, *ISO* und *IEEE* verdient gemacht. Hier stehen vor allem branchenübergreifende Vorgaben im Mittelpunkt. Daneben existieren zahlreiche branchenspezifische Richtlinien, wie beispielsweise die der *Gesellschaft für technische Kommunikation* (tekom), sowie Normen und Richtlinien der Europäischen Union. Aufgrund der rasanten technischen Entwicklung unterliegen diese Normen einem permanenten Änderungsprozess, so dass es nötig ist, bei der Nutzung solcher Normen, Standards und Richtlinien stets auf Aktualität zu achten.

Die DIN 66230

Einen grundlegenden Standard im Bereich der Softwaredokumentation stellt die DIN-Norm *66230* dar. Hauptaugenmerk legt die Norm auf eine ordnungsgemäße Programmdokumentation im Sinne einer Konsistenz zwischen Programmcode und Dokumentation. Dies geschieht in Hinblick darauf, dass durch die Dokumentation sichergestellt sein muss, dass die Software in vollem Funktionsumfang genutzt werden kann. Dazu werden verschiedene Anforderungen an die Dokumentation gestellt:

- vollständige und fehlerfreie Information
- sinnvolle Strukturierung
- ansprechende Gestaltung

[29] vgl. [Tha01] S. 35 - Definition des „British Standards Institute"

Fragen wie diese Anforderungen überprüft werden können, bleiben innerhalb der Norm unberücksichtigt. Lediglich die im Folgenden aufgeführten Dokumente zeigen eine Möglichkeit, diesen gerecht zu werden.

Hintergrund der Norm ist eine phasenorientierte Programmentwicklung[30]. Die Dokumentation wird in die Phase der Systemeinführung neben den Phasen Planung, Realisierung und Test eingeordnet. Wie Abbildung 2.6 zeigt, nimmt die Norm eine Untergliederung der Programmdokumentation in ein Anwendungshandbuch und ein datenverarbeitungstechnisches Handbuch vor[31]:

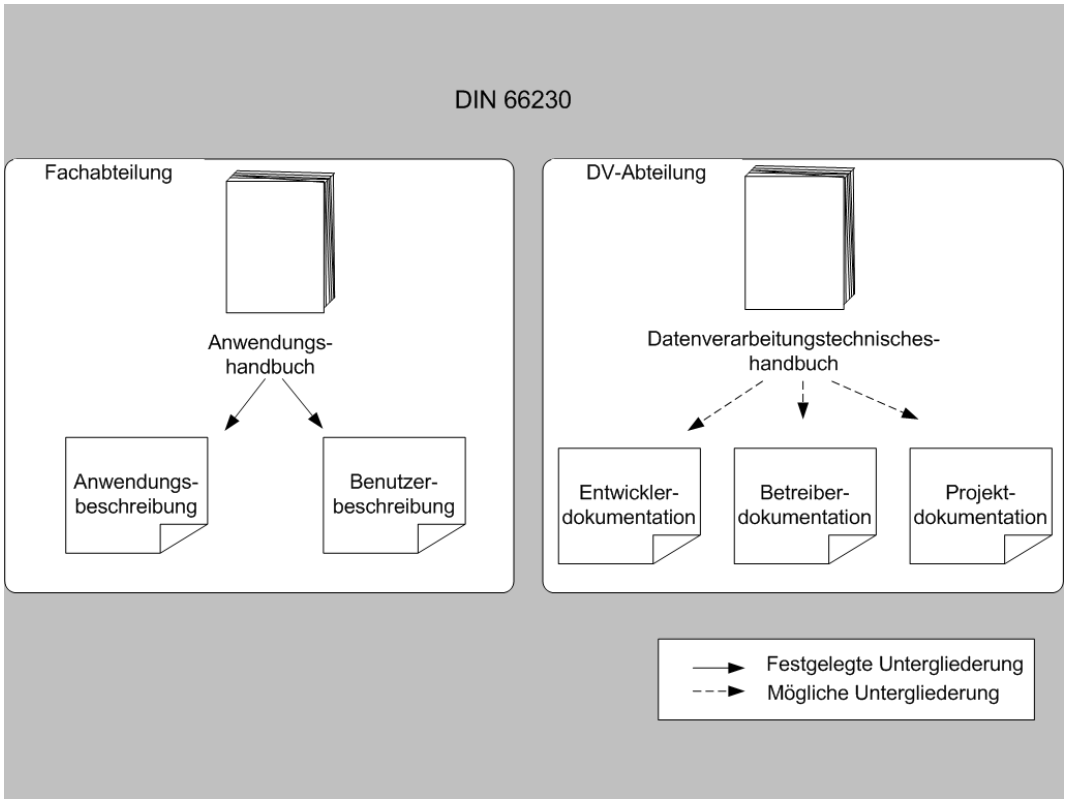

Abbildung 2.6: Überblick über die DIN 66230

Der Oberbegriff des *Anwendungshandbuchs* ist vor allem für Fachabteilungen und Management konzipiert und setzt sich aus einer *Anwendungs-* sowie *Benut-*

[30] vgl. beispielsweise Kap. 2.2
[31] vgl. [Deu96] S. 38 ff.

zerbeschreibung zusammen[32]. Die *Anwendungsbeschreibung* beinhaltet den Leistungsumfang der Software, während die *Benutzerbeschreibung* eine Bedienungsanleitung zum Umgang mit der Software darstellt. Das *datenverarbeitungstechnische Handbuch* enthält Informationen zur Installation, zum Betrieb und zur Pflege des Programms und ist für informationsverarbeitende Abteilungen konzipiert. Darauf aufbauend und unter Einbeziehung der auf der DIN-Norm *66230* folgenden Normen, wird häufig eine weitere Untergliederung vorgenommen. Das geschieht auch im Zusammenhang mit möglichen Weiterentwicklungen in der Softwareentwicklung. Eine derartige Berücksichtigung erfolgt in zahlreichen Veröffentlichungen der Literatur, hauptsächlich unter dem Aspekt verschiedener Benutzergruppen. Eine Variante stellt die Untergliederung des *datenverarbeitungstechnischen Handbuches* in *Entwickler-*, *Betreiber-* und *Projektdokumentation* dar[33]. Die *Entwicklerdokumentation* dient vorrangig der Beschreibung von Systeminterna und Programmcode und enthält alle während der Entwicklung entstandenen Dokumente. Die *Betreiberdokumentation* dient der Anleitung von ausführenden Stellen im Sinne des Betriebes eines Rechenzentrums. Die *Projektdokumentation* dient der Beschreibung des Softwareentwicklungsprozesses und enthält Angaben zur Projektorganisation. Dies nutzt hauptsächlich der Aufwands- und Risokoabschätzung in späteren Projekten. Die Projektdokumentation enthält damit wichtige Erfahrungswerte zur Verbesserung betriebsinterner Prozesse.

Inzwischen wurde der zeitlichen Diskrepanz der *DIN-66230* durch das Deutsche Institut für Normung e.V. Rechnung getragen. Im November 2001 wurde die *DIN EN 62079 „Erstellen von Anleitungen - Gliederung, Inhalt und Darstellung"* veröffentlicht[34]. Diese Norm wurde zwar hauptsächlich für technische Dokumentationen im Sinne von Produkten und Anlagen entwickelt, eine Einschränkung auf einen bestimmten Produktbereich ist jedoch nicht gegeben, so dass in einigen Veröffentlichungen auch eine Anwendung der Norm für die Softwareentwicklung in Betracht gezogen wird[35]. Ähnlich der *DIN 66230* wird eine Struktur vorgeschlagen und Anleitungen zur Darstellung unterbreitet. Vor allem Checklisten zur Selbstbewertung und enthaltene Leitfäden sollen das neue Berufsfeld des „*Tech-*

[32] die aufgeführten Angaben beruhen auf dem Stand Jan. 1981
[33] vgl. [Sta99] S.324f.
[34] vgl. [dVG]
[35] vgl. [DKE]

nischen Redakteurs" in der Dokumentationserstellung unterstützen.

Die DIN EN ISO 9001 - Qualitätsmanagementsysteme : Anforderungen

Als wohl bedeutendste internationale, branchenunabhängige Norm ist die im Dezember 2000 veröffentlichte Weiterentwicklung der DIN ISO 9000 - Familie die *DIN ISO 9001:2000* zu nennen. Ihre Bedeutung erschließt sich vor allem durch die Anzahl der sich dieser Norm verpflichtenden, zertifizierten Unternehmen. Somit stellt die *ISO* eine Dachorganisation bzw. einen nationalen Normungsgremien übergeordneten Zusammenschluss dar. Die Norm ist, wie Abbildung 2.7 zeigt, an eine prozessorientierte Softwareentwicklung angelehnt:

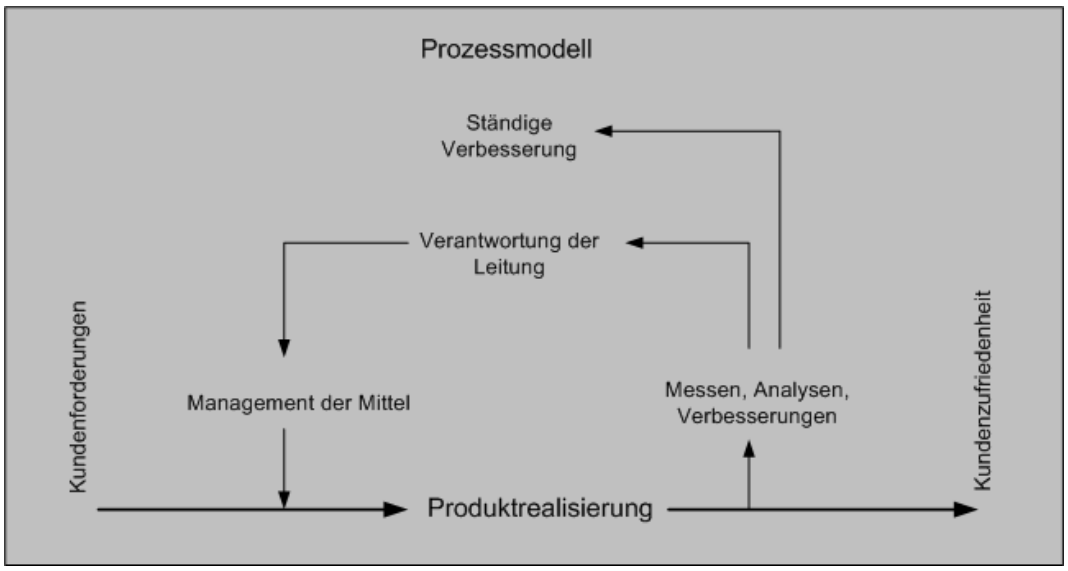

Abbildung 2.7: Überblick : ISO 9001:2000

Die Begriff der Qualität wird hier durch die starke Kundenorientierung definiert, wobei eine kontinuierliche Verbesserung unternehmensinterner Prozesse in den Mittelpunkt der Betrachtung rückt. Aufbauend auf den Kundenanforderungen muss der Prozess der Produktrealisierung stets analysiert und verbessert werden, so dass eine möglichst hohe Kundenzufriedenheit erreicht wird. Somit liegt der Sinn dieser Norm in der Überarbeitung, Definition und Bewertung un-

ternehmensinterner Prozesse und darauf aufbauend in der Leistungssteigerung der gesamten an der Softwareentwicklung beteiligten Organisation. Dabei wird vor allem die „*Verantwortung der Leitung*" und eine durchgängige Qualitätspolitik betont. Die im Anhang A aufgezeigten Abbildungen zeigen beispielhaft eine Umsetzung im Bereich der Dokumentation der in der *ISO 9001:2000* gemachten Anforderungen in Form von Musterdokumenten.

Weiterhin existieren zahlreiche Beispiel- und Musterdokumente, welche Anforderungen betriebsspezifisch konkretisieren. Dies betrifft im Bereich der Dokumentation zusätzliche, in der Norm unerwähnte Aspekte, wie etwa strukturelle und visuelle Anforderungen an die Dokumentation. Zudem bestehen in diesem Anwendungsfeld zahlreiche Unternehmen, welche die betriebsspezifische Umsetzung der *ISO 9001* anbieten. Ein *ISO 9001* - Zertifikat stellt für zahlreiche Unternehmen einen wertvollen Bestandteil des Marktauftritts dar. Dies steht zwar im Gegensatz zu den eigentlichen Absichten der Norm, welche vor allem in der Prozessverbesserung und im Kundennutzen liegen, ist aber auch ein Aspekt für die Durchsetzung und breite Akzeptanz der Norm[36].

Die IEEE 1063

Im Gegensatz zur *ISO 9001:2000* - Norm stellt diese Norm eine spezifischere Vorgabe dar. Die Relevanz des Standards ergibt sich zum einen aus der Aktualität dessen und einer internationalen Zusammenarbeit im Entwicklungs- und Entstehungsprozess. Das *Institute of Electrical and Electronic Engineers* veröffentlichte am 20.12.2001 den internationalen Standard „*IEEE 1063 : Software-Benutzerdokumentation*". Dieser gibt Empfehlungen für Struktur, Inhalt und Form von Software-Benutzerdokumentationen[37]. Dabei finden in der aktuellen Version erstmals Formen elektronischer Dokumentation und „Online"-Hilfen Berücksichtigung. Hauptaugenmerk der Norm ist eine Trennung zwischen einer Anleitung und einer Referenzdokumentation. Eine benutzerspezifische Untergliederung ist somit durch aufgabenorientierte Informationsaufbereitung in der Anleitung und dem

[36] vgl. [Tha01] S. 220,297,318
sowie: [Eur]

[37] vgl. Die im Folgenden getroffen Aussagen beruhen auf: [Grü]

wahlfreien Informationszugriff in der Referenzdokumentation gegeben. Konkrete Anforderungen an Inhalt und Form finden sich in den abgebildeten Kriterien der Abbildung 2.8:

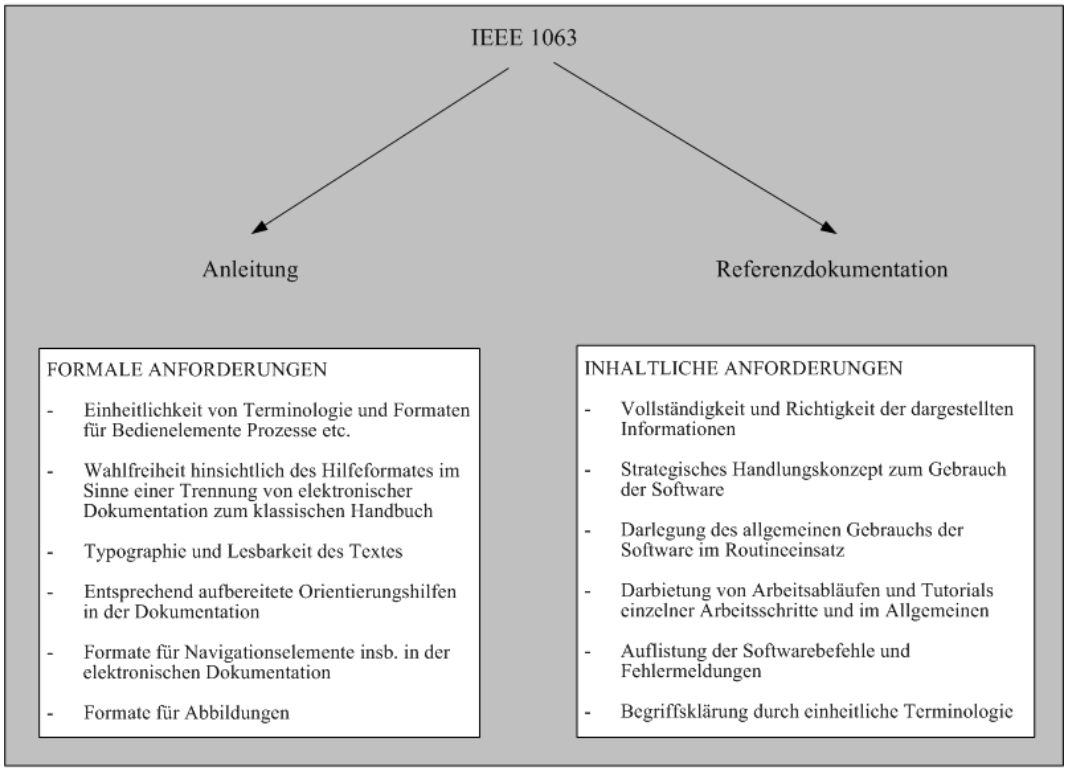

Abbildung 2.8: IEEE 1063 - Anforderungskriterien

Vor allem diese Kriterien stellen aufgrund der strikten Anforderungen eine klare Vorgabe und Orientierungshilfe für die Erstellung einer Dokumentation dar. Somit steht in der *IEEE 1063* vor allem das Produkt der Dokumentation und damit einhergehende qualitative Aspekte im Mittelpunkt der Betrachtung. Nicht der Prozess der Dokumentation, welchem wie in der *ISO 9001* eine softwareentwicklungsbegleitende Bedeutung zukommt, sondern die konkreten Anforderungen an die am Ende der Softwareentwicklung notwendigen Dokumente, finden hier Berücksichtigung. Somit kann die *IEEE 1063* eine ideale Ergänzung zu den prozessorientierten und prozessoptimierenden Betrachtungsweisen der *ISO 9001* darstellen und somit mit in die erwähnten Musterdokumente der Dokumenta-

tion einfließen. Aufgrund der Aktualität der Norm ist eine Verfügbarkeit von Beispielen und Mustern zur Durchsetzung der beschriebenen Anforderungen eingeschränkt. Zukünftige Entwicklungen werden zeigen, inwieweit die *IEEE 1063* Verwendung finden wird.

Volere

Im Folgenden wird als Beispiel für eine gängige, betriebliche Praxis im Zusammenhang mit Standards, Normen und Richtlinien „*Volere*" vorgestellt. *Volere* ist das Ergebnis aus langjähriger Praxis, Beratung und Forschung auf dem Gebiet der Anforderungsanalyse. Die Richtlinie wurde von Suzanne und James Robertson, beides Mitglieder der „*Atlantic System Guild*", entwickelt und öffentlich zur Verfügung gestellt[38].

Volere stellt eine Gliederungsmöglichkeit für Anforderungen dar, welche im Rahmen der Softwareentwicklung entstehen, wobei eine Untergliederung in Form von „Schubladen" bzw. Kategorien erfolgt. In diesen können verschiedenste Arten von Anforderungen erfasst und aufgeführt werden, so dass ein strukturierter Katalog aller Wünsche und Forderungen von Kunden und Nutzern zusammengetragen werden kann. Ein Überblick über diese Kategorisierung und eines Beispieldokumentes zur korrekten Erfassung der Anforderungen sind im Anhang abgebildet[39].

Für die korrekte Erfassung und Untergliederung existieren 26 Kategorien zusammengefasst in 4 Bereichen. Die Bedeutung *Voleres* erschließt sich dabei aus der formalisierten und dokumentenorientierten Hinterlegung von Anforderungen, welche im Rahmen der Softwareentwicklung entstehen. Jede Anforderung kann somit einer Kategorie zugeordnet werden. In dieser erfolgt letztendlich eine genaue Beschreibung der Ziele, Verantwortlichkeiten und ähnliches. Somit ist neben der Orientierung auf die einzelnen Dokumente, eine spätere Analyse des Softwareentwicklungsprozesses möglich, wobei einzelne Fehler und Schwachstellen im Prozess gezielt nachvollzogen werden können.

Im Folgenden soll als Beispiel für die Kategorisierung *Voleres* die als Kategorie

[38] vgl. [Rob99]
[39] siehe Anhang A

25 im Bereich: „Projektrandbedingungen" ausgewiesene Benutzerdokumentation beschrieben werden. Der Plan zur Bildung dieser beinhaltet die Liste der notwendigen Dokumentationen als Teil des Softwaresystems, sowie die zukünftig beabsichtigten Schulungen. Der Zweck in der Ausweisung der Dokumentation in einer eigenen Kategorie ist ein Setzen der zukünftigen Verantwortlichkeiten und eine Feststellung der Erwartungen an die Benutzerdokumentation. Dabei stehen zahlreiche Überlegungen im Vordergrund, welche im Folgenden aufgeführt werden:

- Welches Niveau muss die Dokumentation erfüllen?
- Werden die Benutzer bei der Erstellung beteiligt?
- Wer ist verantwortlich für eine stets aktuelle Version für die Dokumentation?
- In welcher Form soll die Dokumentation erstellt werden?
- Welche Schulungsmaßnahmen werden notwendig sein?
- Wer wird verantwortlich für die Gestaltung und Planung der Schulungsmaßnahmen?
- Wer ist verantwortlich für die Schulung?

Volere lässt für verschiedene Projekte Verfeinerungen der einzelnen Kategorien zu, so dass die aufgeführten Überlegungen der Kategorie „Benutzerdokumentation" nach Bedarf erweitert werden können. Auch werden im Rahmen der vorliegenden Arbeit bereits erwähnte, allgemeine Anforderungen durch die weiteren Kategorien *Voleres* berücksichtigt. So legt beispielsweise der Bereich „Rahmenbedingungen für das System" die Grundlage für eine zielgruppenbezogene Ausrichtung der Dokumentation.

Insgesamt stellt *Volere* durch seine sinnvolle Untergliederung der im Rahmen der Softwareentwicklung entstehenden Anforderungen, sowie seiner praxisbewährten Entstehung und freien Verfügbarkeit, einen häufig verwendeten Ansatz im Sinne einer Richtlinie dar.

2.5 Zwischenfazit

Das zurückliegende Kapitel zeigte die wachsende Bedeutung der Dokumentation. Produkte und damit einhergehende Dokumente müssen einem immer größer werdenden Kundenkreis vermittelt werden. Durch eine zunehmende Globalisierung und einen weltweiten Wettbewerb wachsen die Anforderungen im Hinblick auf Effizienz, Qualität und Wirtschaftlichkeit der Produktionsprozesse. Dies betrifft auch einen steigenden Leistungsdruck und eine immer schneller, immer besser und immer preiswertere Erstellung der Dokumentation. Zudem entstehen ständig neue Technologien und Werkzeuge, welche in diesem Zusammenhang berücksichtigt werden müssen.

Bisher waren jedoch meist die Entwickler der Software für die Erstellung und somit für die Qualität der Dokumentation verantwortlich. Diese müssten in dem dargestellten Umfeld neben ihrer reinen Programmiertätigkeit jedoch Kenntnisse über die gezeigten Anforderungen, sowie Normen, Standards und Richtlinien haben. Auch müssten abhängig von der Phase der Softwareerstellung, neben der reinen Softwareentwicklung, Dokumente durch die Entwickler erstellt werden. Das Entstehen von Berufen, wie die des „*Technischen Redakteurs*" zeigen jedoch, dass eine Dokumentationserstellung durch diese häufig die qualitativen Ansprüche nicht erfüllt. Aber ist ein Hinzuziehen eines *Technischen Redakteurs* in kleinen bis mittleren Softwareprojekten überhaupt wirtschaftlich betrachtet sinnvoll? Eine Entlastung der Entwickler steht somit im Vordergrund. Dabei stellt sich die Frage: Wenn die Entlastung der Entwickler durch eine Automatisierung von Arbeitsschritten erfolgen soll, welche Teile der Dokumentation kommen dafür in Betracht und welche Werkzeuge können dabei genutzt werden? Diese Frage wird im Folgenden durch die Vorstellung aktueller Technologien und Datenformate untersucht und im darauf folgenden Lösungsansatz beantwortet.

3 Aktuelle Technologien und Datenformate im Bereich der Dokumentation

Software ist ein Produkt, welches einer ständigen Veränderung unterworfen ist. Wachsende Anforderungen in einmal entwickelte Projekte, sowie sich ändernde Rahmenbedingungen, stellen hohe Anforderungen an die Wartung und Weiterentwicklung. Damit einhergehend muss auch die Dokumentation stets angepasst werden. Eine Wahrung der Konsistenz vorhandener und neu hinzukommender Dokumente muss gewährleistet sein. Neben Anforderungen im Bereich des Dokumentenmanagements und der Versionsverwaltung entstehen vor allem Ansprüche an die qualitativen Anforderungen der Softwaredokumentation. Hilfsmittel, welche die Verwaltung der Dokumente und eine Konsistenz derer untereinander gewährleisten, sind deshalb notwendig. Zur Entlastung der Softwareentwickler steht dabei eine sinnvolle Aufbereitung des Programmquellcodes im Mittelpunkt der Betrachtung. Im Folgenden werden Werkzeuge und Datenformate vorgestellt und untersucht, welche die Grundlage für einem im darauf folgenden Kapitel behandelten Lösungsansatz darstellen[1].

[1] vgl. [Rom]

3.1 Quellcodeaufbereitung am Beispiel des Parsers „doxygen"

Eine Möglichkeit der Informationsgewinnung aus dem Programmcode ist die Anwendung eines „*Parsers*". Der Begriff „*parsen*" bezeichnet dabei eine Analyse der vorhandenen Daten und eine Segmentierung in verarbeitbare Bestandteile [2]. Am Beispiel des Werkzeugs „*doxygen*" geschieht dies zum einen durch die gezielte Suche nach Schlüsselworten in bestimmten Kontexten und zum anderen durch Kommentare im Programmcode[3]. Während die Schlüsselworte ausgewertet werden können, müssen die Kommentare ein bestimmtes Schema aufweisen, um vom Parser korrekt erkannt zu werden. *Doxygen* ist für die Programmiersprachen *Java*, *C*, *C++* und *IDL* entwickelt worden. Die durch das Programm gewonnenen Daten werden anschließend aufbereitet. Der Vorteil *doxygens* sind die zahlreich generierbaren Formate. Diese sind: *HTTP*, *XML*, *RTF*, *TEX*, *PostScript*, *PDF* sowie *UNIX Man Pages*. Dabei muss jedoch erwähnt werden, dass *doxygen* soweit möglich ein eigenes Layouts in die Endformate integriert. Zudem können Abhängigkeitsgraphen, Vererbungs- und Kollaborationsdiagramme erzeugt werden.

Abbildung 3.1 zeigt beispielhaft eine in der Programmiersprache „C" vorliegende Funktion und deren Kommentierung im Quellcode, sowie eine mögliche Aufbereitung durch *doxygen*:

[2] vgl Quelle : wissen.de und babylon.com
[3] vgl. [Hee]

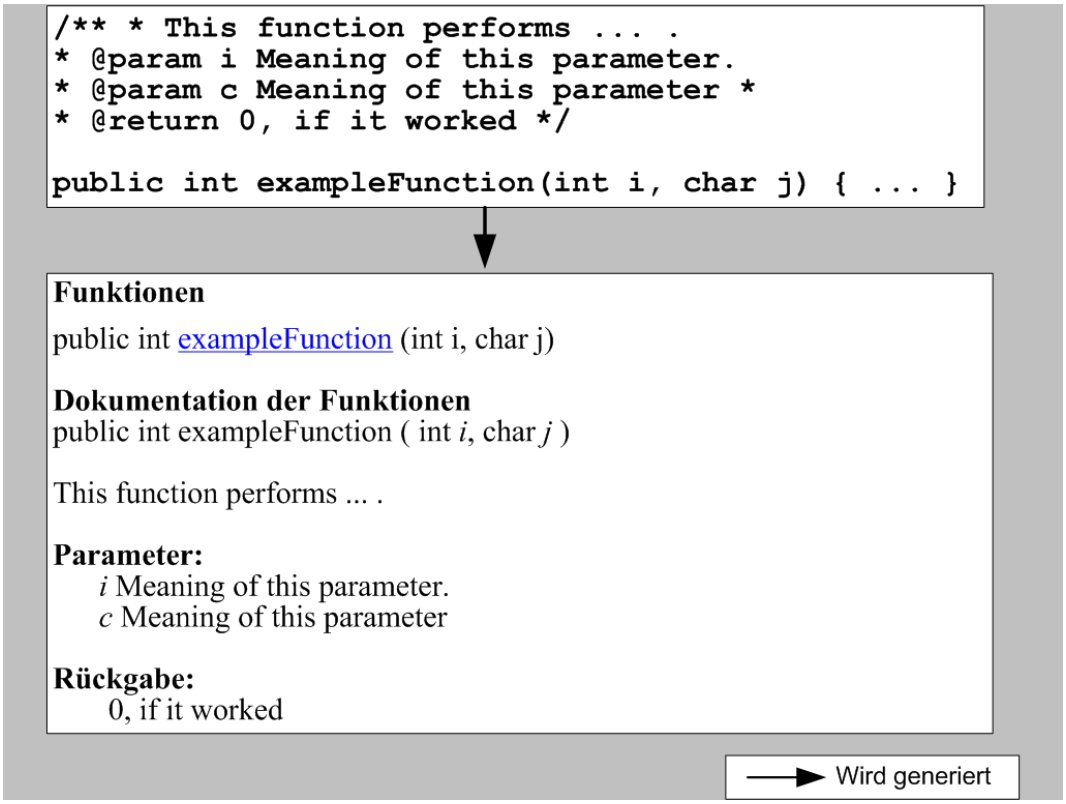

Abbildung 3.1: Umsetzung des Quellcodes in HTML

Es ist nun jedoch zu unterscheiden, welche Programmbestandteile selbständig erkannt werden und welche Informationen zusätzlich durch Kommentare hinzugefügt werden müssen. Der Einsatz von *doxygen* zeigt, dass grundlegende Strukturen wie „*class*", „*struct*", „*union*", „*enum*", „*typedef*", *Funktionen* und mit den jeweiligen Strukturen einhergehende *Paramter* korrekt erkannt werden. Auch Zusammenhänge der Programmbestandteile, wie beispielsweise Funktionsaufrufe, werden richtig aufgeführt. Welche Bedeutung jedoch hinter den Funktionen und Methoden steht, welchen Zweck sie erfüllen bzw. welches Ziel mit dem Softwareprojekt erreicht werden soll, erschließt sich nur durch die zusätzlichen Kommentare im Programmquellcode. Die fehlenden Informationen müssen somit durch parser-spezifische Anweisungen innerhalb der Kommentare hinzugefügt werden. Das abgebildete Beispiel zeigt, dass beispielsweise die Bedeutung eines Parameters durch die Anweisung„`@param [Paramter] [Beschreibung des Parameters]`"

erklärt werden muss. Dabei stellt sich die Frage, inwieweit eine Einarbeitung in diese parser-spezifischen Anweisungen lohnt oder ob die selbständig erkannten Strukturen ausreichen? Diese Frage lässt sich sicherlich nicht einheitlich beantworten. So muss je nach Projektumfang entschieden werden, ob eine Dokumentation innerhalb des Quellcodes sinnvoll ist oder ob eine Überführung der Daten in Textsysteme und eine dortige Bearbeitung erfolgen soll. Für letzteren Fall und um die durch *doxygen* erhaltenen Informationen in einen eigenen Verarbeitungsprozess zu überführen, bietet sich *XML* als Datenformat bzw. Datenaustauschformat an.

3.2 Die eXtensible Markup Language

Die *eXtensible Markup Language* (XML) wird als eine Auszeichnungssprache verstanden. Dieser Begriff ist als eine Sprache, mit deren Hilfe eine optische und inhaltliche Strukturierung von Informationen möglich ist, definiert. Die Anforderungen und Bedingungen, welche an eine Auszeichnungssprache geknüpft werden, sind[4] :

- *Flexible Datenstruktur*: Der Aufbau der Daten muss an unterschiedliche Bedürfnisse anpassbar und eine möglichst leichte Konvertierung der Daten in andere Formate muss möglich sein.
- *Plattformunabhängigkeit*: Der Austausch der Daten muss unabhängig von den zugrunde liegenden Plattformen erfolgen können.
- *Textorientierung*: Die Textorientierung resultiert letztendlich aus der Plattformunabhängigkeit. Die Kennzeichnung der Daten soll dabei über Textmarken erfolgen. Eine Trennung von Text und Format ist demnach gegeben.

Somit liegen Anwendungen von *XML* in einer rein inhaltlichen Definition von Daten. In diesem Zusammenhang wird häufig von logischer Auszeichnung gesprochen, welche eine logische Zuordnung von Gliederungsstruktur und Daten beinhaltet. Dies betrifft den in *XML* baumartigen, hierarchisch gegliederten Aufbau der Daten. Ein weiterer Begriff, welcher in diesem Zusammenhang genannt werden muss, ist der des „*Semantischen Markups*[5]". Dieser umfasst, dass auf-

[4] vgl. [PW00] Kap.2
[5] vgl.[PW00] S.29

grund der Bezeichnung der Gliederungselemente, welche „*tags*" genannt werden, auf deren Inhalt geschlossen werden kann.

Neben der eigentlichen Grundstruktur existieren weitere Merkmale im Umfeld von XML. Mit Hilfe einer, der jeweiligen XML-Datei zugeordneten „*Document Type Definition*" (DTD), lassen sich der strukturelle Aufbau und die logischen Elemente von Dokumenten definieren[6]. Der in diesem Zusammenhang häufig verwendete Begriff der „*Metafähigkeit*" meint dabei, die Möglichkeit der Erstellung eigener Sprachen zur Dokumentenbeschreibung.

Die letztendlich in *XML*-Form vorliegenden, strukturierten Daten können anschließend weiterverarbeitet und aufbereitet werden. Aufgrund der vorgestellten Eigenschaften ist es somit möglich einen eigenen spezifisch ausgerichteten, strukturellen Aufbau zur Beschreibung der Daten zu erstellen. Somit kann *XML* eine Basis für eine umfassende Datenquelle darstellen, welche sowohl plattformübergreifend, als auch von der jeweiligen Anwendung unabhängig ist[7]. Welche Anwendungsidee zugrunde liegt (Dokumente, Datensätze, Transaktionsdaten), spielt aus technischer Sicht keine Rolle. Daten können ihrem Verwendungszweck entsprechend generiert und aufbereitet werden[8]. Somit bestehen mögliche Anwendungen im Bereich von XML, zum einen in der Übernahme und Beschreibung von Datenstrukturen verschiedenster zugrunde liegender Formate[9] und zum anderen in dem Datenaustausch verschiedener Anwendungen. Ein Beispiel für Ersteres bildet die „*DocBook-DTD*", welche im folgenden als Beispiel für zahlreiche Entwicklungen und Standardisierungsbemühungen in diesem Umfeld, vorgestellt wird.

3.2.1 XML-Format am Beispiel „DocBook"

Dokumentationen sind meist in Buchform verfasst. Durch den Aufbau eines Buches ist somit dem zugrunde liegenden Dokument eine bestimmte strukturelle Gliederung vorgegeben. Dieser strukturelle Aufbau wird durch *DocBook* beschrieben.

[6] vgl.[PW00] S.53
[7] vgl. [PW00] S.47ff.
[8] vgl. [Beh99]
[9] vgl. [PW00] S.20 und S.49

Die definierte Struktur in der *DTD* von *DocBook*, welche aktuell in der Version 4.2 vorliegt, ist im Umfeld der technischen Dokumentation angesiedelt[10]. *DocBook*, welches vor rund 10 Jahren entstand, wurde aus der Notwendigkeit heraus entwickelt, UNIX-Dokumentation auszutauschen und zu erschließen[11]. Die dabei entstehenden typographischen Konventionen sorgen für eine klare Definition der endgültigen Dokumentenstruktur, so dass eine wachsende Anzahl von Autoren auf dieses Format zurück greift. *DocBook* ist zwar vorrangig nach dem Modell eines Buches gestaltet, es existieren aber auch Elemente für Projekte größeren bzw. kleineren Umfangs. Aufgrund dieser über 350 vorliegenden, logischen Gliederungselemente, lassen sich Dokumentationen und ähnliche Publikationen strukturiert erstellen. Zudem erleichtern zahlreich veröffentlichte Stylesheets die Umsetzung in verschiedene Präsentationsformate[12]. *DocBook* kann grob in folgende Kategorien untergliedert werden:

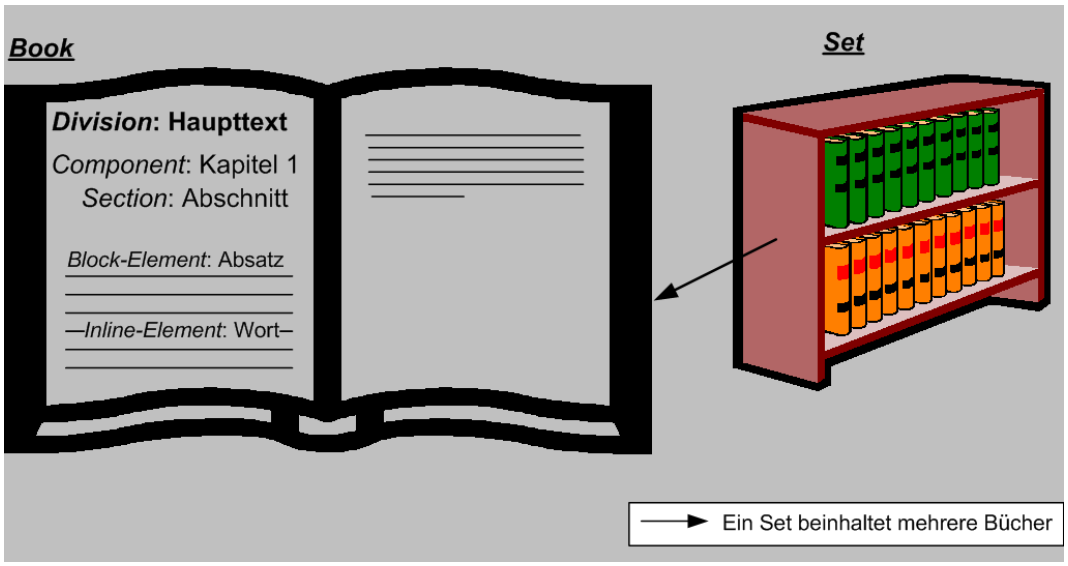

Abbildung 3.2: Grundlegende Elemente in DocBook

Innerhalb dieser Kategorien existieren weitere Elemente, welche die Strukturmerkmale eines Buches weiter spezifizieren. Dadurch ist eine gewisse Einarbeitungszeit in den strukturellen Aufbau notwendig. Jedoch existieren und entstehen

[10] vgl. [Wal]
[11] vgl. [Huh01]
[12] vgl. Kapitel 3.3

erste Editoren, welche diesem Umstand Rechnung tragen, so dass *DocBook* eine sinnvolle Entwicklung in Richtung eines *XML*-Standards im Bereich von Publikationen darstellen kann.

3.3 Die Möglichkeiten der Weiterverarbeitung anhand von XSL

Während *XML* der logischen bzw. semantischen Auszeichnung der Daten dient, ist die *eXtensible Stylesheet Language* (XSL) für die physische Auszeichnung der in *XML* vorliegenden Daten verantwortlich. *XSL* stellt die Möglichkeit einer gezielten, visuellen Datenaufbereitung von zunächst in reiner Textform vorliegender Daten dar[13]. Der dabei häufig verwendete Begriff des „*Cross Media Publishing*", also eine Konvertierung von *XML* in verschiedenste Präsentationsformate, wird häufig in diesem Zusammenhang verwendet. *XSL* untergliedert sich in 3 Teile[14]:

- „*XSL Transformation*" (XSLT): dient der Transformation von XML in verschiedene Formate,
- „*XML Path Language*" (Xpath): dient dem Zugriff auf Elemente in XML,
- „*XSL Formatting Objects*" (xsl-fo): dient der Festlegung des Layouts.

Die Verwendung der einzelnen Bestandteile ist in Abbildung 3.3 skizziert:

[13] vgl. [Kay01] S.11ff.
[14] vgl. [W3C]

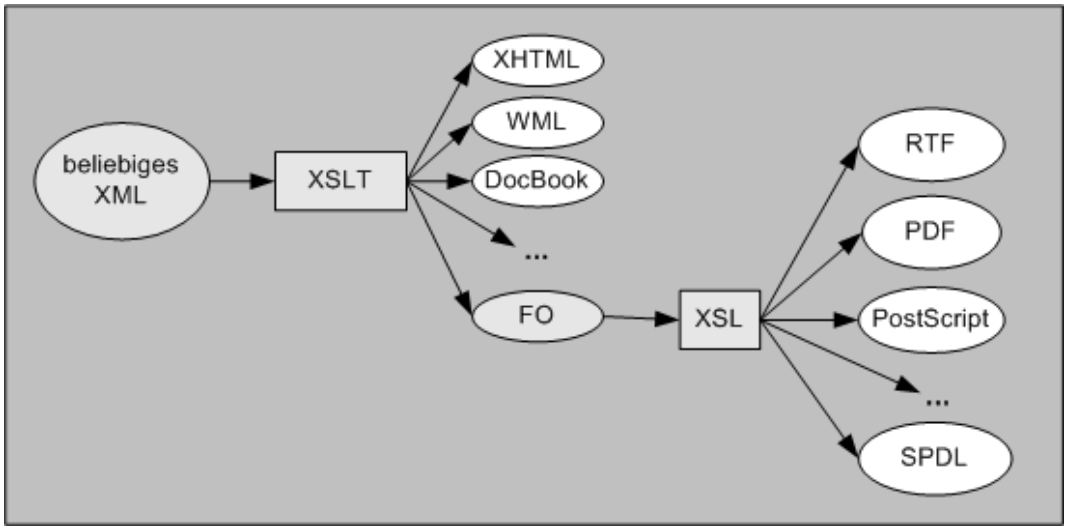

Abbildung 3.3: XSLT und XSL

Während *XSLT* für die Transformation vorhandener in *XML* vorliegender Daten in neue Strukturen oder Texte vorgesehen ist, stellen *xsl-fo* eine Möglichkeit zur visuellen Aufbereitung der Daten dar. Innerhalb von *xsl-fo* wird der Seitenaufbau und das Layout beschrieben. Dies geschieht mit den über 50 *xsl-fo*-Elementen, welche aufgrund des *XML*-Ursprungs baumartig angeordnet sind. Mit Hilfe von *XSLT* wird somit aus der vorliegenden *XML*-Hierarchie ein Ergebnisbaum generiert, wozu ein spezieller *XSLT*-Prozessor notwendig ist. Wie folgende Abbildung verdeutlicht, wird in einem zweiten Schritt dieser Struktur die notwendige Formatierungssemantik hinzugefügt[15].

[15] vgl. [BM00] S.148
sowie: [Dat]

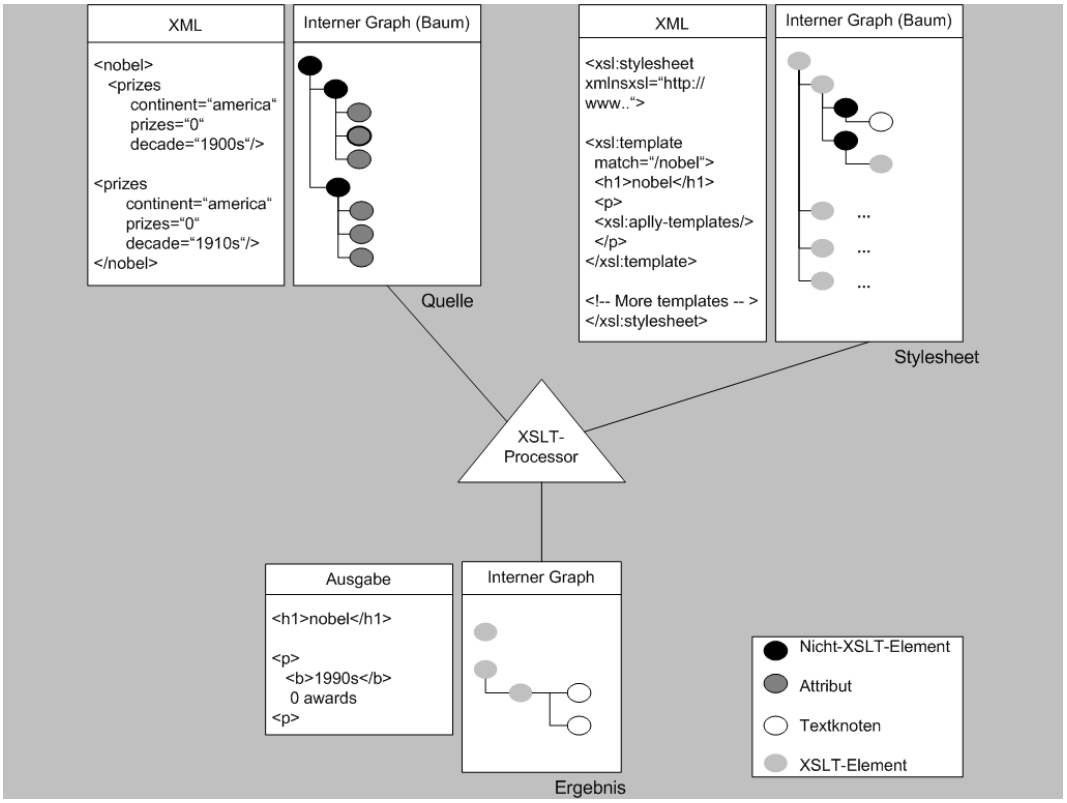

Abbildung 3.4: XSL-Prozessor

Somit sind mit Hilfe von Stylesheets, Transformationen in beliebige Formate möglich, wobei aber die zugrunde liegende *XML*-Struktur erhalten bleibt.

Ein Beispiel für eine derartige Anwendung stellt der „*Formatting Objects Processor*" des „*Apache XML Projects*" dar. Dieser wurde mit dem Ziel entwickelt, aus der beschriebenen Baumstruktur und den Layout-Informationen der *xsl-fo*-Elemente, sofort publizierbare Dokumente wie etwa *PDF* zu generieren. Diese Umwandlung vollzieht sich dabei in zwei Schritten. Zunächst wird mittels der *xsl-fo*-Anweisung ein Ausgabebaum gebildet, welcher in in einem zweiten Schritt vom „*formatting objects processor*" in ein gewünschtes Ausgabeformat überführt wird. Der javabasierte Prozessor hat den Vorteil, dass zwar Kenntnisse im Umgang mit *XML* und *XSL* notwendig sind, jedoch keine Kenntnisse über Umgang und Aufbau des *PDF*. Das sich noch am Anfang der Entwicklung befindliche Projekt ist frei erhältlich und vereint zunehmend die Anforderungen des aktuellen

XML- und *XSL*-Standards.

3.4 Manuelle Dokumentation

Zwar gelten *TeX* und *LaTeX* nun seit mehr als 15 Jahren als Standard im Bereich umfangreicher, wissenschaftlicher Publikationen, jedoch entstehen auch in diesem Bereich Alternativen. Neben den weitgehend kostenlos erhältlichen *TeX*- und *LaTeX*-Veröffentlichungen, existieren mittlerweile einige kommerzielle Produkte, welche sich vor allem durch eine einfachere Zugänglichkeit und Benutzerfreundlichkeit auszeichnen. Im Folgenden wird zunächst TeX, die weit verbreitete Textverarbeitung „*Word*" und „*FrameMaker*", ein professionelles Werkzeug für Publikationen, vorgestellt.

3.4.1 TEX und LATEX im Bereich umfangreicher technischer Dokumentation

TeX ist ein Textsatzprogramm, welches als Ergebnis einen Ausdruck in Buchdruckqualität ermöglicht[16]. Zunächst wird der Text geschrieben und in einem zweiten Schritt erfolgt dessen Formatierung durch ein Formatierungsprogramm, welches *TeX* letztendlich darstellt. Die Formatierungsanweisungen sind dabei in der Form von Programmbefehlen bzw. -anweisungen im Text enthalten. Die Ausgabe von *TeX* ist eine „*DeVice-Independent*"-Datei (DVI), was bedeutet, dass der interpretierte Inhalt dieser Datei, drucker- bzw. geräteunabhängig ausgegeben werden kann. Einen weiteren Vorteil stellt die standardmäßige Fähigkeit von *TeX* dar, Formeln und grafische Elemente darzustellen, wodurch es die Grundlage für zahlreiche wissenschaftliche Publikationen ist. Zudem stellen die programmiertechnische Möglichkeiten in *TeX* in vielen Fällen eine Erleichterung im Umgang mit Struktur und Layout dar. Dies sind aber auch die häufig genannten Schwachpunkte im Umgang mit *TeX*. Durch die Mächtigkeit der Bearbeitung ist eine umfassende Einarbeitung in die über 900 Befehle notwendig. Weiterhin wird die Möglichkeit, den formatierten Text erst jeweils am Ende einer gewissen Arbeitsperiode im endgültigen Layout sehen zu können, oft als Nachteil gewer-

[16] vgl. [Kop00]

tet. „*Ein wirklich erfolgreicher und zufrieden stellender Einsatz von TEX setzt somit Programmierkenntnisse, Fertigkeiten der Satztechnik und grafische Kreativität, verbunden mit dem Wissen über deren psychologische Wirkung beim Leser voraus*[17]".

LaTeX ist als ein Werkzeug anzusehen, welches die logische Struktur eines Textes in *TeX* übersetzt. Es entstand vorrangig aus der Notwendigkeit heraus, die Mächtigkeit von *TeX* in eine einfachere Handhabbarkeit zu überführen. Durch zahlreiche Makro-Pakete wurden die vielfältig möglichen Anweisungen zu einfacheren Befehlen kombiniert. Kenntnisse über Satztechniken und bestimmter Layout-Stile sind somit nicht mehr in dem Umfang notwendig als dies bei *TeX* mit seinen über 900 Befehlen, der Fall ist. Die Merkmale von *LaTeX* sind :

- Durch die breite Akzeptanz und die jahrelangen Erfahrungen im Umgang mit Latex existieren zahlreiche Formatvorlagen, bzw. Layouts, welche von wissenschaftlichen Verbänden definiert wurden.
- Dadurch ist eine Auseinandersetzung mit dem Layout nicht notwendig, jedoch eine Auseinandersetzung mit den Befehlen.
- Zwar existieren auch kommerzielle Angebote im Bereich der Editoren für *LaTeX*, trotzdem sind die zur Arbeit notwendigen Pakete frei verfügbar.
- Das setzen von Formeln u.ä. ist durch Funktionalitäten gängiger Editoren auf einfache Weise möglich.
- *LaTeX*-Dateien sind im ASCII-Format und können dadurch mit beliebigen Editoren bearbeitet werden.

Zahlreich vorhandene Editoren erleichtern zudem die Einarbeitung in die für die Formatierung des Textes notwendigen Anweisungen. Umfangreiche Texte mit einer über das gesamte Dokument konsistenten Struktur und den erwähnten qualitativen Eigenschaften des Ergebnisses, sind somit realisierbar. Die Entwicklung von *LaTeX* geht dabei in Richtung einer plattformübergreifenden Verbreitung und Nutzung der Dokumente.

Die Bedeutung von *TeX* und *LaTeX*, im Zusammenhang mit *XML*, erschließt sich aus der Übernahme logischer Strukturen. So ist der Aufbau des vorgestellten

[17] vgl. [Cot]

DocBook-Formates, dem von *TeX* ähnlich. Beispielsweise entspricht das Element „`<section/>`" im *DocBook*-Format genau der Anweisung „`\section`" in *TeX*, wobei beide eine Kapitelunterglіederung beschreiben. Auch sind beide aus ähnlichen Gründen entwickelt worden, so dass eine Konvertierung von *DocBook* oder anderen denkbaren *XML*-Formaten in das *TeX*-Format, beispielsweise mittels Stylesheets realisierbar ist. Jedoch sind im Falle einer derartigen Überführung weiterhin Kenntnisse über die *TeX*-Anweisungen zur Formatierung und Layout des Textes notwendig, da letztendlich durch *XML* nur der strukturelle Aufbau der Informationen beschrieben wird.

3.4.2 Textverarbeitung „Word"

Im Gegensatz zu *TeX* und *LaTeX* ist Word ein *WYSIWYG*-Editor[18]. Das bedeutet, dass Formatierungen und Änderungen im Layout direkt beim Schreiben des Textes vorgenommen werden können, womit der Text dem Betrachter genau so erscheint, wie er später auch ausgedruckt wird. Die Vorteile liegen hauptsächlich in der starken Verbreitung von *Word*-Anwendungen und der Integration dieser in komplette Softwarepakete. Dadurch kann beispielsweise über eine *Word*-Anwendung auf Daten anderer Anwendungen des Softwarepaketes zugegriffen werden. Dies eröffnet wiederum die Möglichkeit, beispielsweise Daten aus Anwendungen für Tabellenkalkulation oder Präsentationen in eigene *Word*-Dokumente zu integrieren. Vor allem durch die Konzeption der Programme für eine breite Nutzerschicht, weisen diese leicht zugängliche, intuitiv bedienbare Komponenten auf. Der Umgang bzw. die Programmierung erfolgt im Gegensatz zu *TeX* und *LaTeX* ausschließlich über grafische Hilfsmittel und modernen Interaktionsformen, wie Menüs oder Dialogboxen. Darin liegen jedoch auch die meisten Kritikpunkte. Professionell aufbereitete Texte mit fester und konsistenter Struktur und den darauf aufbauenden Regeln der Typographie, liegen in der Verantwortung des Nutzers. Das setzt bei diesem einen entsprechenden Umgang mit dem Text voraus, welcher jedoch nicht immer gegeben ist. Einsatzmöglichkeiten für wissenschaftlich fundierte Publikationen sind mangels eines umfangreichen Formelsatzes und fachspezifischer Formatvorlagen schlecht gegeben. Auch wird in Vergleichstests häufig die Fehleranfälligkeit im Umgang mit den Programmen bemängelt, was

[18] „what you see is what you get": wird auch als „Echtbild-Formatierung" bezeichnet

vor allem Neustrukturierungen und nachträgliche Änderungen eines Dokumentes betrifft[19].

3.4.3 Textsystem „Adobe Framemaker"

„Framemaker" wurde speziell zur Erstellung wissenschaftlicher bzw. technischer Berichte entwickelt. Ähnlich wie in *TeX* bzw. *LaTeX* werden zunächst Struktur und Layout für das endgültige Dokument festgelegt, wodurch auch hier eine konsistenter Aufbau des Dokumentes von Anfang an gewährleistet ist. In der Anwendung vereint *Framemaker* sowohl die Vorteile eines *WYSIWYG*-Textsystems, als auch die eines Textsatzsystems. Formatierungen des Textes können damit jederzeit vorgenommen werden und sind sofort sichtbar, dies jedoch in einem konsistent strukturiertem Layout. Weitere Vorteile stellen der integrierte Formelsatz, leicht zugängliche Verknüpfungen des Textes und die einfache Einbindung von Grafiken dar. Somit sind Befehlskenntnisse, wie sie in *TeX* und *LaTeX* notwendig sind, nicht mehr erforderlich. Durch die beschriebene Mächtigkeit von *Framemaker* ist aber auch hier erst eine gewisse Einarbeitungszeit von Nöten, um die beschriebenen Funktionalitäten in vollem Umfang nutzen zu können. Zudem wird der relativ hohe Preis des Textsystems häufig als Kritikpunkt gegenüber anderen Produkten im Bereich professioneller Dokumentation gewertet.

3.5 PDF: Publikations- und Austauschformat

Für alle vorgestellten Textsysteme besteht die Möglichkeit eigene Formate in das „*Portable Document Format*-Format" zu konvertieren. Zwar wurde dieses Format vorrangig für die Ausgabe am Drucker entwickelt, jedoch sprechen dessen Weiterentwicklungen für die Anwendung des Begriffes „Publikationsformat". Die Merkmale dafür sind :

- Unabhängigkeit von einem bestimmten Betriebssystem (Windows, UNIX, LINUX, OS/2, Mac usw.,)
- Unabhängigkeit von einem bestimmten Textverarbeitungsformat,

[19] vgl. [Ehr00]
sowie: [Dr.99]

- Keine Möglichkeit der Änderungen von Schriftarten, Seitenumbrüchen und dergleichen beim Austausch mit anderen Rechnern,
- Das Dokument entspricht beim Leser genau der vom Autor autorisierten Form,
- Die Möglichkeit der Wort- bzw. Zeichensuche im Dokument,
- Bewahrung der Dokumentenstruktur[20].

Verknüpfungen und interaktive Grafiken[21] in den Texten sind dabei ebenso möglich, wie komfortable Navigationen im Text selbst. Vor allem die Plattformunabhängigkeit und die zahlreichen Möglichkeiten zur Erstellung des Formates, eröffnen Wege eines raschen Dokumentenaustausches, zumal für unterschiedlichste Plattformen kostenlose Betrachtungsprogramme erhältlich sind. Zudem sorgt die Offenlegung des Formates für zahlreiche Bemühungen *PDF* in eigene Anwendungen einzubinden bzw. eigene Formate in *PDF* zu überführen[22].

3.6 Konkrete Zielstellung

Wie die zurück liegenden Ausführungen zeigen, sind Aufbereitungen des Quellcodes mit aktuellen Werkzeugen oft von unzureichender Qualität. Vor allem Bezug nehmend auf die gezeigten Anforderungen an die Dokumentation und die Ziele aufgezeigter Normen, Standards und Richtlinien wird deutlich, dass eine Einflussnahme auf die Daten gegeben sein muss. Ein Rückgriff auf Textsysteme und eine dortige Aufbereitung der Daten ist deshalb notwendig. Das Ziel dabei ist jedoch, möglichst viele Informationen, welche der Quellcode bietet in ein Textformat zu überführen. Um eine Entlastung der mit der Dokumentation betrauten Mitarbeiter zu erreichen, ist es zudem sinnvoll, die Daten in einer sinnvoll strukturierten Form aufzubereiten, so dass lediglich fehlende Informationen ergänzt werden müssen.

Bei dieser Herangehensweise wird jedoch auch deutlich, dass eine Aufbereitung des Quellcodes in die Benutzergruppe der Entwickler einzuordnen ist. Deshalb

[20] vgl. [Arg]
[21] auch unter dem Begriff „Image Maps" gebraucht
[22] Quelle : XML praxis und referenz S. 23

stehen vorrangig qualitativ hochwertige Programmdokumentationen im Hinblick auf einem Referenzhandbuch[23] im Mittelpunkt der Betrachtung. Die entstehenden Dokumente dienen dem Gesamtverständnis des Systems und sind somit die Grundlage für darauf aufbauende Produktbeschreibungen und -dokumentationen. Dementsprechend wird der Aspekt einer Einordnung dieser Dokumente in den betrieblichen Dokumentationsprozess im Folgenden berücksichtigt werden.

Zunächst gilt es, die Eigenschaften eines Quellcodeparsers zu nutzen und die erhaltenen Daten in ein Datenübertragungsformat zu überführen. Dafür bietet sich die Auszeichnungssprache *XML* aufgrund der aufgezeigten Eigenschaften zur Strukturierung der Daten an. In der weiteren Verwendung ist eine Auseinandersetzung mit *XSL* notwendig. Darauf aufbauend soll eine Überführung der in XML vorliegenden Strukturen in das *TeX*-Format als Beispiel für ein Textsystem erfolgen. Dieses bietet sich insoweit an, da die Daten lediglich in eine andere ASCII-Form überführt werden müssen. Der Aufwand in der Überführung der Daten hält sich somit weitestgehend in Grenzen. Eine sinnvolle Bearbeitung in *TeX* ist jedoch nur dann möglich, wenn eigene, dem Verwendungszweck entsprechende Anweisungen und Umgebungen erstellt werden, welche der aufbereiteten Struktur das Layout hinzufügen.

Die folgenden Ausführungen beschäftigen sich vor allem mit der Analyse nutzbarer Formate, um eine fundierte Herangehensweise an die Problemstellung zu ermöglichen. Dies geschieht im Hinblick auf einer möglichst schnellen und einfachen Umsetzung von programmspezifischen Elementen des Quellcodes zu bearbeitbaren bzw. publizierbaren Dokumenten. Es wird untersucht, inwieweit Quellcode-Parser wie beispielsweise „*doxygen*" genutzt werden können und in welcher Art und Weise die generierten Formate zu verwenden sind. Auch die Voraussetzungen, welche derartige Parser an den Quellcode stellen, werden im Folgenden Berücksichtigung finden.

Dabei bilden die im folgenden Kapitel aufgeführten Aussagen die Grundlage für einen Prototyp, welcher zeigen soll, welche Bestandteile des Quellcodes und inwieweit diese korrekt erkannt und ausgewertet werden können.

[23] vgl. Kap. 2.3

4 Lösungsansatz

Das folgende Kapitel beschreibt den Weg von im Quellcode vorliegenden Informationen hin zu einem Format, in dem eine sinnvolle Bearbeitung dieser erfolgen kann. Neben den Rückgriff auf Funktionalitäten eines Quellcodeparsers werden die nutzbaren Datenformate betrachtet:

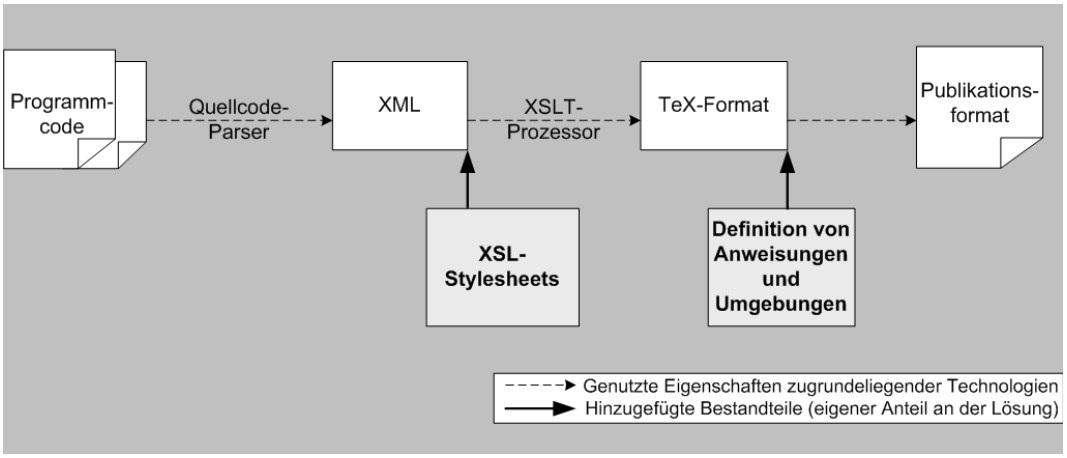

Abbildung 4.1: Lösungsweg

Zur Analyse und Informationsgewinnung wird der Quellcodeparser *doxygen* eingesetzt. Grundlage sind zunächst typische Programmbestandteile, welche in der Programmiersprache *C* vorliegen. Vor allem die Möglichkeit des Parsers, Informationen mittels *XML* zu strukturieren, werden innerhalb des Lösungsansatzes betrachtet.

In einem nächsten Schritt wird untersucht, mittels welcher Technologie die in *XML* vorliegenden Daten in ein Textsystem überführt werden sollen. Aufgrund

der im vorherigen Kapitel aufgezeigten Eigenschaften aktueller Entwicklungen wird die Erstellung eigener Lösungen mittels *XSL* angestrebt. Auch wird sich zeigen, dass die Entscheidung zur Erstellung von XML hinsichtlich einer betrieblichen Dokumentenverwaltung sinnvoll erscheint.

Neben einer Einarbeitung in die durch *doxygen* generierte *XML* wird eine Auseinandersetzung mit einer gezielten Informationsaufbereitung innerhalb der Stylesheets notwendig. Die Erfassung und Darstellung der Informationen muss in entsprechend strukturierter Art und Weise erfolgen. Bei der Bearbeitung der Daten mit Hilfe eines Textsystems wird deshalb Wert darauf gelegt, dass eine möglichst kurze Einarbeitungszeit in die aufbereitete Struktur notwendig ist und dass Kenntnisse über Darstellung der Informationen nicht notwendig sind.

Für eine gezielte Strukturierung und Überführung in ein Textformat werden innerhalb der Stylesheets die Grundlagen für das *TeX*-Format gebildet. Die Informationen werden bereits hier mit Anweisungen und Umgebungen des Textsatzsystems verknüpft. Das Ergebnis der Verbindung von XML und den erstellten Stylesheets ist eine Struktur, welche sich auf die generierten Daten stützt, diese aber bereits in Anweisungen und Umgebungen des *TeX*-Formates integriert.

Die Aufbereitung und die letztendliche Darstellung der Informationen bezieht sich auf die zu erstellenden Anweisungen und Umgebungen. Diese werden als *TeX*-Makro's implementiert, also als gespeicherte Befehlsfolge zum wiederholen von Befehlskombinationen. Dabei können jedoch die Daten diesen Anweisungen und Umgebungen als Parameter übergeben werden, so dass für bestimmte Daten gezielte Anweisungen und Umgebungen erstellt werden können. Es entsteht dadurch ein durchgängiges, den jeweiligen Anforderungen entsprechendes Layout. Somit ist lediglich das Hinzufügen fehlender Informationen innerhalb der Bearbeitung des Textes durch den Autor notwendig. Kenntnisse über Struktur und Layout bzw. über die dafür notwendigen Befehle bzw. Anweisungen des Textsatzsystems sind nicht erforderlich.

Im Folgenden werden die in Abbildung 4.1 aufgezeigten Schritte und die dabei benutzten Werkzeuge vorgestellt.

4.1 Automatisierung der Informationsgewinnung am Beispiel des Quellcodes

Eine Möglichkeit der automatisierten Datenauswertung stellen Anwendungen dar, welche ursprüngliche Daten in Zielformate konvertieren. Anhand des Aufbaus und der Strukturmerkmale der vorliegenden Formate, lassen sich Inhalte in andere Formate überführen. Im Folgenden werden Einsatzmöglichkeiten, Potenziale und Grenzen der automatisiert erstellten Dokumentation auf Basis des Quellcodes vorgestellt und analysiert. Mittels eines Parsers werden typische Strukturen im Quellcode erkannt und aufbereitet. Der erste Schritt des Lösungsansatzes ist somit die Nutzung eines solchen Parsers. Die im Folgenden beschriebene Herangehensweise ist in Abbildung 4.2 skizziert:

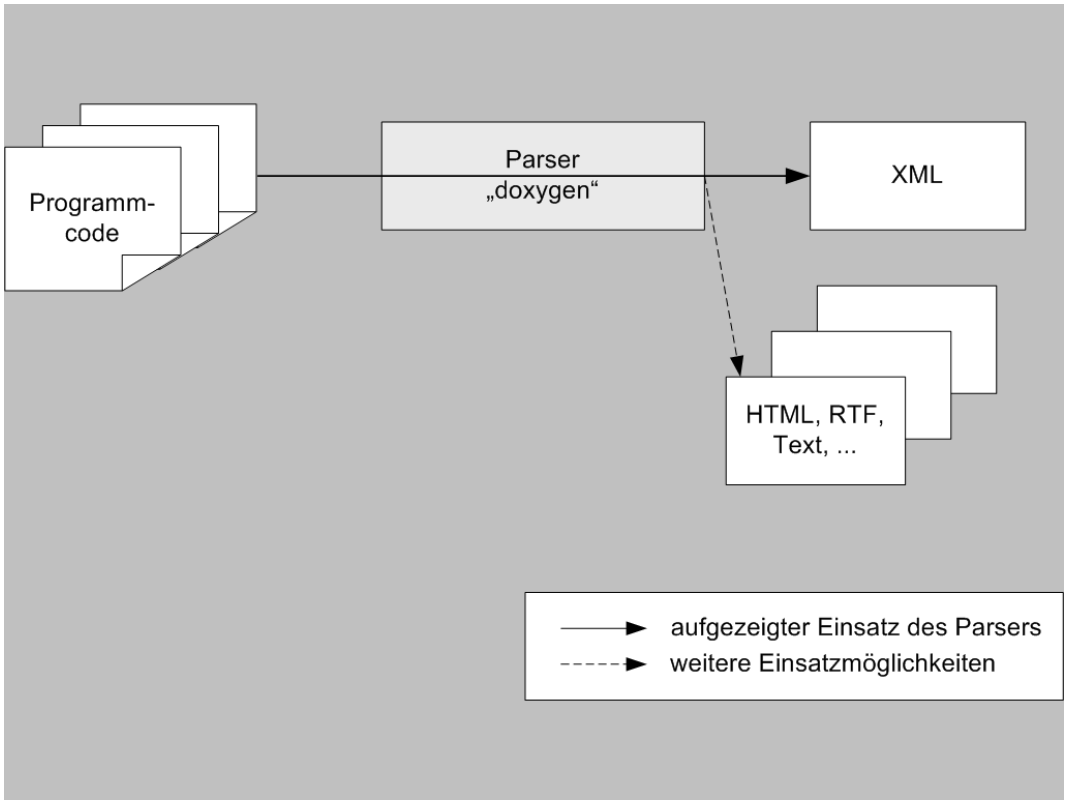

Abbildung 4.2: Überführung des Quellcodes in ein verwertbares Format

Zur Verwendung kommt hier der Quellcodeparser „*doxygen*[1]". Der Einsatz und der im Folgenden beschriebene Aspekt einer konsistenten Dokumentenverwaltung macht dabei nur Sinn, wenn die Möglichkeiten zur Generierung verschiedenster Ausgabeformate vernachlässigt werden. Vielmehr interessant ist die Möglichkeit der Generierung von XML, da dieses leicht und automatisiert mittels Stylesheets[2] in firmeninterne Strukturen transformiert werden kann. Grundlage für den Quellcode bilden hier die Programmiersprachen *C* und *C++*.

Beim Einsatz des Parsers ist zunächst zwischen Strukturen, die selbständig erkannt werden und dem Quellcode zugefügten Kommentaren bzw. Schlüsselwörtern, zu unterscheiden. Vor allem die Fähigkeiten, Programmstrukturen und darauf aufbauende Zusammenhänge richtig zu erkennen, sind dabei interessant. Die Nutzung dieser Eigenschaften macht im Hinblick auf eine automatisierte Dokumentenerstellung nur dann Sinn, wenn tatsächlich eine personelle Entlastung erreicht werden kann.

Die Basis für eine automatisierte Weiterverarbeitung ist somit durch die Überführung der Informationen aus dem Quellcode hin zu *XML* gegeben.

4.2 Der Einsatz von XML

Der Grund für den Einsatz der Auszeichnungssprache sind dessen Eigenschaften[3] zur Strukturierung von Informationen, welche eine Basis für eine Dokumentenverwaltung bilden. Grundlage für den Aufbau und die Umsetzung einer über den Entwicklungsprozess erfolgenden Dokumentation, bildet dafür eine Analyse zugrunde liegender und genutzter Datenformate. Durch die Anzahl unterschiedlicher, innerbetrieblich genutzter Anwendungsprogramme stellt dies einen nicht zu unterschätzenden Aufwand dar. Ein Lösungsansatz für den Aufbau eines Systems zur Dokumentenverwaltung stellt *XML* deshalb dar, da die Auszeichnungssprache wie beschrieben[4] sowohl als Datenquelle, als auch als Austauchformat genutzt werden kann. Die Vorteile einer einheitlichen Struktur, einhergehend mit

[1] vgl. Kap. 3.1
[2] vgl. Kap. 3.3
[3] vgl. Kap.3.2
[4] vgl.Kap.3.2

den Möglichkeiten verschiedener Aufbereitungen sprechen daher für den Einsatz. Jedoch ist mit der Verwendung von *XML* ein erheblicher Aufwand der Erfassung und Umwandlung sowie eine genaue Planung notwendiger Strukturen und Auszeichnungselementen verbunden. Die für eine einheitliche Strukturierung zwingend notwendige *DTD*, stellt dabei die Grundlage dar. Durch diese lassen sich vor allem qualitative Anforderungen an Dokumente, wie sie zahlreiche Normen, Standards und Richtlinien vorgeben[5], im gesamten betrieblichen Umfeld der Dokumentation durchsetzen.

Beim Einsatz von *XML* ist zunächst zwischen datenorientiertem und dokumentenzentriertem Einsatz der Auszeichnungssprache zu unterscheiden[6]. Je nach späteren Verwendungszweck ist zwischen einer strengen Datenverknüpfung und einer losen, jedoch dokumentenübergreifenden Struktur[7], zu wählen. Somit bietet sich bei der Integration verschiedenster betriebsinterner Dokumente in ein Anwendungssystem, eine dokumentenzentrierte Betrachtung an. Dies bewirkt zum einen eine sinnvolle und vollständige Erfassung aller Dokumente und zum anderen, aufgrund der Eigenschaften der Auszeichnungssprache, eine schnellere und gezielte Weiterverarbeitung. Fallspezifische Dokumente sollten jedoch einer strengen datenorientierten Strukur folgen. Eine Transformation vorhandener, allgemeiner *XML* in spezifische Strukturen ist jedoch durch gesonderte Stylesheets möglich.

4.3 Die Umsetzung der Struktur

Zur weiteren Nutzung der durch *doxygen* generierten *XML*-Struktur ist zunächst eine Analyse derer notwendig. Zunächst sollte darauf aufbauend eine Überführung in das eigene firmeninterne *XML*-Format erfolgen. Der bisher dargestellte Weg vollzieht sich somit in der in Abbildung 4.3 dargestellten Art und Weise:

[5] vgl. Kap. 2.4.3
[6] vgl. [Arc01] S.28; S.382ff.
[7] bsp. DocBook

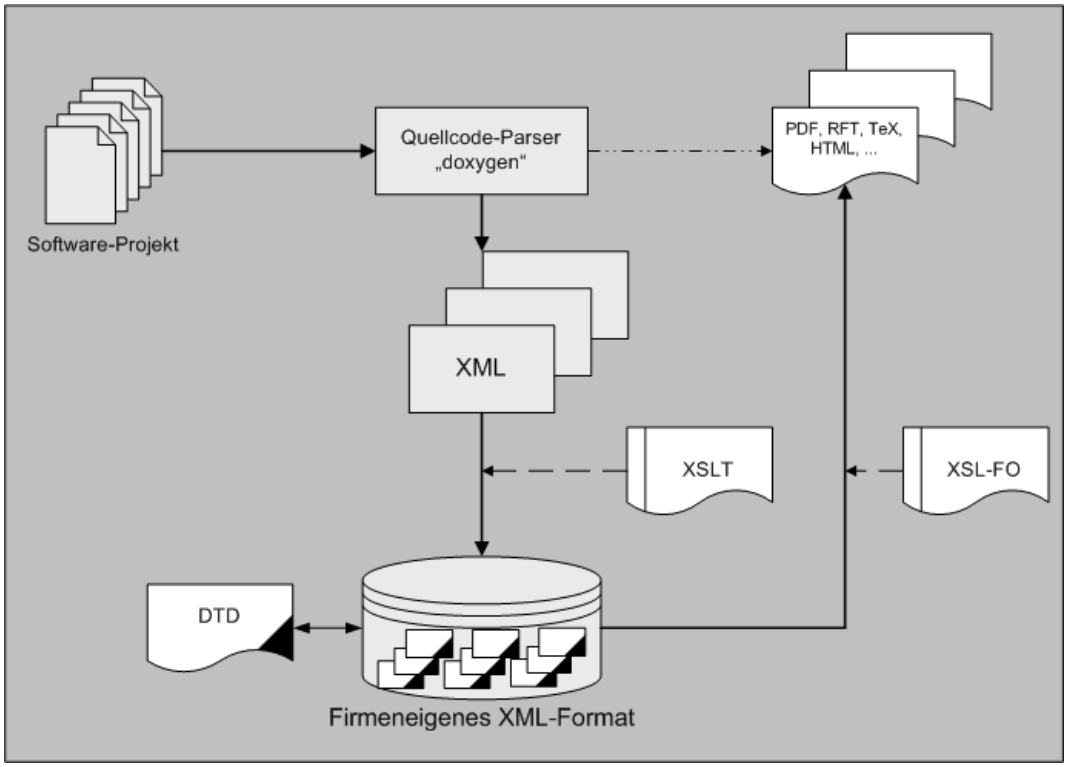

Abbildung 4.3: Lösungsansatz : Überblick

Zusätzlich können bei der Überführung in das firmeneigene Format Anforderungen an die Dokumentenverwaltung Berücksichtigung finden. Da in der Regel mehrere *XML*-Dateien vorliegen, kann beispielsweise eine Indizierung oder eine Versionskennung, gemäß der Regeln einer vorliegenden *DTD*, in den Dateien selbst erfolgen. Auch kann hier durch eine sinnvolle Hinterlegung der Dateien der Grundaufbau der Dateiverwaltung festgelegt werden.

In einem weiteren Schritt ist zu analysieren, in welcher Form eine Weiterverarbeitung der *XML*-Daten erfolgen soll. Zwar wurde eine einfache Überführung firmeninterner *XML* in verschiedenste Formate durch „xsl-fo" dargestellt[8], jedoch existieren zusätzliche Möglichkeiten, die vom jeweiligen Einsatzzweck abhängen und dementsprechend divergieren. Je nachdem inwieweit eine Einflussnahme auf die Daten notwendig ist, können den aktuellen Entwicklungen entsprechend, fol-

[8] vgl. Abb.4.3

gende Alternativen gewählt werden :

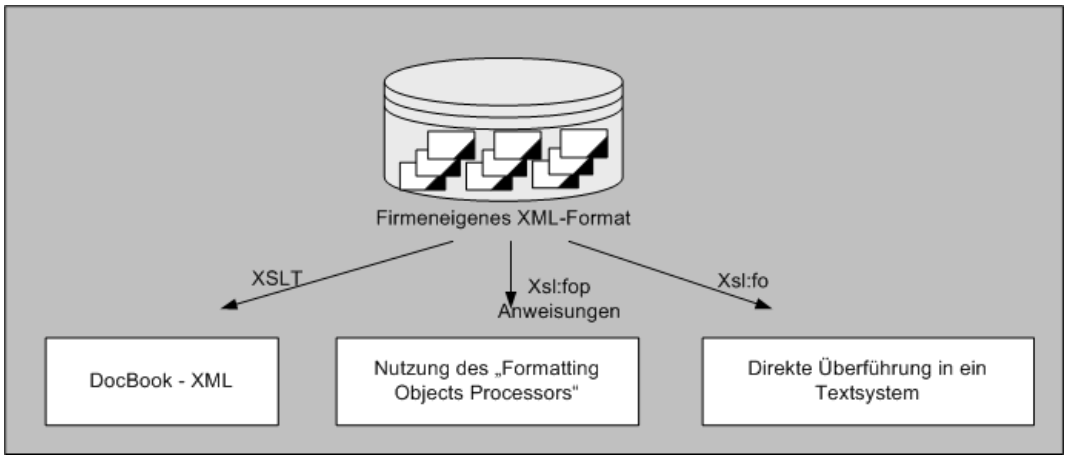

Abbildung 4.4: Lösungsansatz : Datenformate

4.3.1 DocBook

Das *DocBook*-Format ist dahingehend von Interesse, da die qualitativen Ansprüche an die Dokumentation durch diesen bewährten Standard mit eigener *DTD* erfüllt werden können. Somit könnte *DocBook* auch die Grundlage für ein firmeninternes *XML*-Format darstellen. Vor allem der hohe Aufwand zur Erstellung einer eigenen *DTD* entfällt. Weiterhin können durch die für *DocBook* vorhandenen Stylesheets weitgehend beliebige Formate erzeugt werden. Kenntnisse über Bearbeitungsmöglichkeiten von *XML* sind somit nicht nötig. Der eigentliche Anteil läge somit lediglich in der Erstellung eines *DocBook*-konformen Formates aus der von *doxygen* generierten *XML*. Jedoch ist der Aufwand der Einarbeitung in das umfangreiche *DocBook*-Format nicht zu unterschätzen. Auch existieren für die bisher angebotenen Stylesheets keinerlei Standards und die Ergebnisse der Umwandlungen in die Endformate schwanken stark, so dass ein zuverlässiger Einsatz momentan nicht gegeben ist.

4.3.2 Formatting Objects Processor (FOP)

Einen anderen Weg schlägt der „*Formatting Objects Processor*" ein. Hierbei geht es lediglich um die Integration eigener Anweisungen in ein anzufertigendes Stylesheet, welche der Prozessor in Gliederung und Layout umsetzt. Der Aufwand des Aufbaus einer eigenen firmeninternen *DTD*- und *XML*-Struktur entfällt somit nicht. Auch sind Kenntnisse der *XSL* notwendig. Zusätzlich sind die derzeitigen Ergebnisse der Umwandlung in verschiedenste Formate nicht zufrieden stellend. Jedoch lohnt der sich noch am Anfang der Entwicklung befindliche Prozessor einer weiteren Beobachtung, da vor allem hinsichtlich qualitativer Fortschritte, Weiterentwicklungen zu erwarten sind.

4.3.3 Direkter Zugriff auf die Daten

Im Hinblick auf die qualitativen Defizite der Ergebnisse, welche derzeitige Hilfsmittel aufweisen, liegt es nahe, eigene Entwicklungen voranzutreiben. Die im Folgenden beschriebene Herangehensweise soll beispielhaft zeigen inwieweit Strukturmerkmale sinnvoll verarbeitet und aufbereitet werden können.

Wie Abbildung 4.5 zeigt, werden die von *doxygen* erzeugten, vorliegenden *XML*-Daten zunächst mittels eines Stylesheets ausgelesen und entsprechend einer späteren Weiterverwendung angeordnet[9]:

[9] vgl. [Kay01] S.55ff.

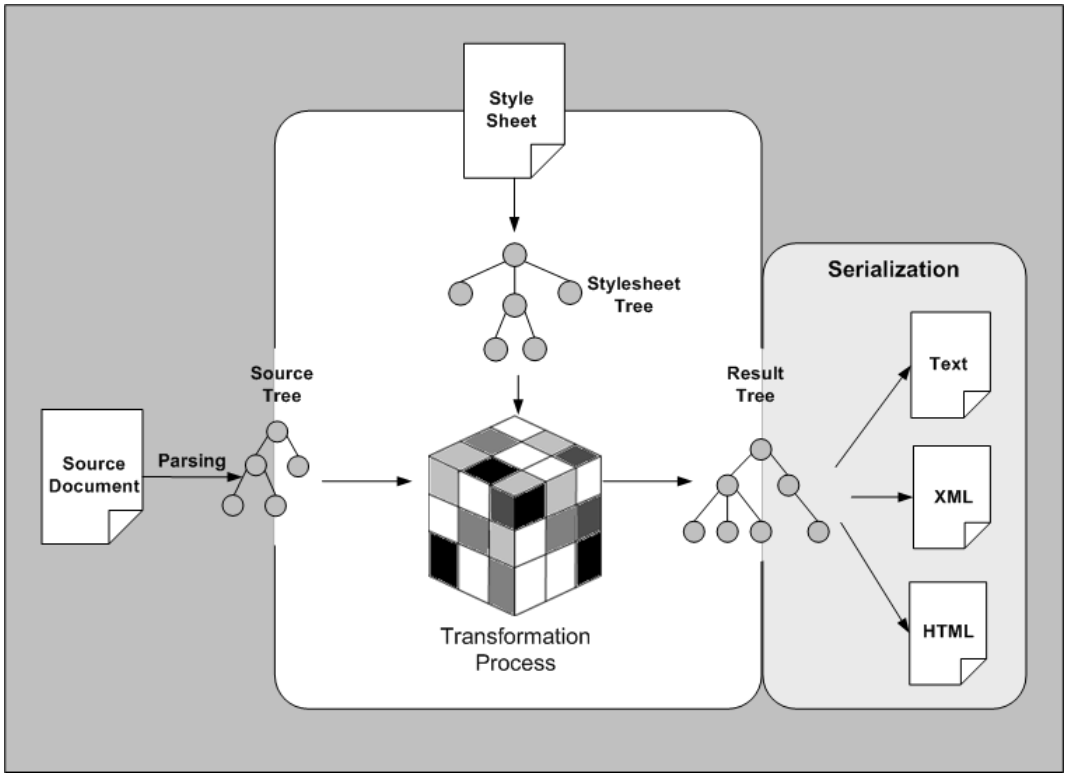

Abbildung 4.5: konkreter Lösungsansatz

Im Mittelpunkt steht dabei eine gezielte Aufbereitung, so dass das generierte Format selbst Strukturmerkmale aufweist, welche für unterschiedliche Zwecke möglichst einfach unterschiedlich aufbereitet werden können. Somit liegt die Überführung der Strukturen in das *TeX*-Format nahe.

4.4 Bearbeitung der Daten in einem Textsystem

Aufgrund der Eigenschaften des Textsatzsystems *TeX* ist die Erstellung und Definition eigener Umgebungen und Anweisungen möglich[10]. In der Bearbeitung der Daten ständen dadurch nur die Strukturanweisungen und der Text selbst im Vordergrund, da die Anweisungen zu Gliederung und Layout in den Strukturanweisungen bzw. -befehlen enthalten sind. Somit könnten für unterschiedliche Zwecke

[10] vgl. Kap 3.4.1

unterschiedliche Dateien entwickelt werden, welche die Formatierungsanweisungen beinhalten und je nach Bedarf eingebunden werden. Diese Verfahrensweise hätte den Vorteil, dass sich zum einen der Autor nur mit dem Text an sich beschäftigen muss und zusätzlich den Komfort eines Editors zur Bearbeitung des Textes nutzen kann. Weiterhin sind von Autorenseite keine Kenntnisse des *TeX*-Formates notwendig und der Aufwand zur Erstellung der Strukturanweisungen wäre für verschiedene Einsatzgebiete einmalig und hält sich dadurch weitestgehend in Grenzen.

Die Definition einer Umgebung bewirkt, dass der innerhalb der Umgebung stehende Text gemäß den darin definierten Umgebungsparametern behandelt wird:

```
vorangehender Text
\begin{quote}
Text innerhalb der Umgebung
\end{quote}
nachfolgender Text
```

Die quote-Umgebung bewirkt ein Einrücken des Textes, welcher zwischen den *begin* und *end*-Anweisungen steht. Das Ergebnis würde dementsprechend wie folgt aussehen:

```
vorangehender Text

    Text innerhalb der Umgebung

nachfolgender Text
```

Bei der Erstellung eigener Umgebungen ist eine Übernahme bereits vorhandener Umgebungen und Anweisungen möglich. Eine sinnvolle Integration dieser zur Aufbereitung des Textes steht somit im Vordergrund.

Eine prototypische Umsetzung dieses Lösungsansatzes und eine konkrete Darlegung der genutzten Stylesheets und Tex-Makros, zeigt das folgende Kapitel.

5 Der Prototyp

Im Folgenden werden die im Lösungsansatz besprochenen Punkte in eine konkrete Umsetzung überführt. Diese soll beispielhaft die Tauglichkeit aktueller Technologien zur Automatisierung der Dokumentation zeigen und eine Unterstützung bei der Entscheidung zum Einsatz einer derartigen Anwendung sein. Die Vorstellung der innerhalb des Prototyps verwendeten Funktionalitäten erfolgt anhand einer einfachen Funktion als Bestandteil des Quellcodes. Die Umsetzung des Prototypes wird anhand eines Beispiels am Ende des Kapitels beschrieben.

5.1 Vom Quellcode zu XML

Zunächst wird auf die Funktionalitäten des Parsers „*doxygen*" zurückgegriffen. Zwar ist es mit diesem möglich unterschiedlichste Endformate zu erzeugen, jedoch ist nur mit *XML* die Grundlage für den Aufbau einer Datenbasis, welche die Entwicklung des Softwareprojektes dokumentiert und eines universellen Austauschformates, welche die Daten für unterschiedlichste Zwecke nutzbar macht, gegeben. Weiterhin besteht bei der direkten Generierung der Endformate das Problem, dass das Layout der Dokumente von *doxygen* erzeugt wird und somit eine ungewollte Einflussnahme auf die Darstellung der Dokumentation erfolgt.

Der Einsatz *doxygens* ist dabei auf zwei Arten möglich. Zum einen können über einen „Wizard[1]" alle notwendigen Einstellung hinsichtlich Ausgabeformat, Ge-

[1]Quelle : http://www.xipolis.net : Assistent (Wizard) = kleines Hilfprogramm in Anwendungen. Der Assistent führt den Anwender Schritt für Schritt durch mehrere Menüs, in denen verschiedene Angaben in Bezug auf Zweck, Erscheinung usw gemacht werden können. Am Ende erzeugt der Assistent basierend auf den Angaben des Anwenders das entsprechende Dokument.

nauigkeit der Informationssuche und ähnliches vorgenommen werden und zum anderen kann ein Zugriff auf die für den Einsatz notwendige Initialisierungsdatei erfolgen. Die folgende Abbildung zeigt zunächst die Benutzeroberfläche des *Wizards*, welcher in *doxygen* enthalten ist:

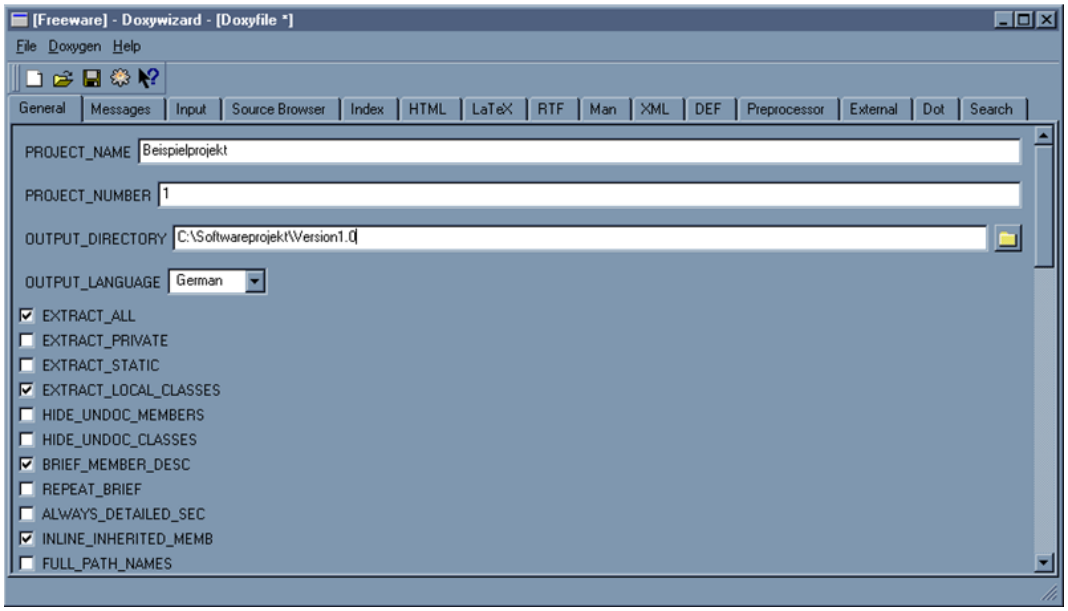

Abbildung 5.1: doxygen-Wizard

Die Einstellung „`EXTRACT_ALL`" sollte beim Einsatz aktiviert sein, damit auch unkommentierte Programmbestandteile berücksichtigt werden. Weiterhin empfiehlt es sich, in dem hier dargestellten Fall, den „`OPTIMIZE_OUTPUT_FOR_C`" zu aktivieren. Weiterhin notwendige, projektspezifische Einstellungen können über den *Wizard* komfortabel vorgenommen werden.

Die weitere, bereits angesprochene Möglichkeit ist, direkt auf die Initialisierungsdatei des Parsers zurückzugreifen. Das hätte zum Vorteil, dass auf die einfache Textdatei über eigene Anwendungen zugegriffen und nur auf für die Anwendung notwendigen Einstellungen Einfluss genommen werden muss. Der Nachteil ist dabei jedoch, dass durch den Ausschluss bestimmter Funktionalitäten eine spätere Änderung der Anwendung notwendig werden kann. Für die Untersuchung der Einsatzmöglichkeiten und auch im Hinblick darauf, dass *doxygen* selbst noch wei-

terentwickelt wird und somit auch neue Funktionalitäten entstehen, reicht eine Nutzung des Parsers über den *Wizard* aus.

Im Folgenden wird zunächst der Quellcode und dessen Umsetzung in *XML* aufgezeigt.

```
/**
* This function performs ... .
* @param i Meaning of this parameter.
* @param j Meaning of this parameter
*
* @return 0, if it worked
*/
public int exampleFunction(int i, char j ) { ... }
```

In diesem Beispiel handelt es sich um eine einfache Funktion und deren Beschreibung über Kommentare im Quellcode. Die Funktion „*exampleFunction*" besitzt zwei Parameter und ist vom Typ „*int*". Alle weiteren Informationen befinden sich in der Kommentierung, welche sich direkt über der Funktion befindet, wodurch *doxygen* einen direkten Bezug zwischen Funktion und Kommentar herstellt. Eine weitere Möglichkeit diesen Bezug herzustellen ist die Nutzung einer Referenz im Kommentar auf den zu beschreibenden Programmbestandteil.

Die Beschreibung innerhalb der Kommentare erfolgt, wie im Quellcode des Beispiels zu erkennen, über Schlüsselwörter. In diesem Fall sind es „`@param`" und „`@return`". Diese besitzen eine durch den Parser definierte Syntax und müssen dementsprechend gehandhabt werden[2].

Eine Überführung der Funktion und deren Beschreibung in *XML* durch *doxygen* ist im Folgenden aufgezeigt, wobei lediglich für das verwendete Beispiel relevante Ausschnitte dargestellt werden.

[2] vgl. Kap.3.1

```
<doxygen version="1.2.16">
  <compounddef id="beispiel_8c" kind="file">
    <compoundname>beispiel.c</compoundname>
      <sectiondef kind="func">
      <memberdef kind="function" id="beispiel_8c_1a0" prot="public">
        <type>public int</type>
        <name>exampleFunction</name>
        <param>
          <type>int</type>
          <declname>i</declname>
        </param>
        <param>
          <type>char</type>
          <declname>j</declname>
        </param>
      <detaileddescription>
        <para>This function performs ... .
            <parameterlist kind="param">
            <title>Parameter:</title>
                <parametername>i</parametername>
                <parameterdescription>
                <para>Meaning of this parameter. </para>
                </parameterdescription>
                <parametername>c</parametername>
                <parameterdescription>
                <para>Meaning of this parameter</para>
                </parameterdescription>
            </parameterlist>
        </para>
        <simplesect kind="return">
            <title>üRckgabe: </title>
            <para>0, if it  worked</para>
        </simplesect>
    </detaileddescription>
   </memberdef>
   </sectiondef>
  </compounddef>
</doxygen>
```

Zunächst sind die vom Parser selbständig erkannten Programmbestandteile von Interesse. In dem hier verwendeten Beispiel betrifft dies die Auszeichnungen ausserhalb des Elementes „`<detaileddescription/>`". Das Element „`<sectiondef/>`" beinhaltet zunächst alle in der Quelldatei vorkommenden Funktionen, welche durch die Auszeichnung „`<memberdef/>`" einzeln strukturiert werden. Die Elemente „`<type/>`", „`<name/>`" und „`<param/>`" zeigen, dass grundsätzliche Zusammenhänge der Programmbestandteile korrekt vom Parser erkannt werden.

Die Kommentare und die damit verbundenen Schlüsselworte im Quellcode, wer-

den innerhalb der Funktionsbeschreibung mit Hilfe des Elements „`<detailed-description/>`" ausgewiesen. Die Schlüsselworte dienen somit der korrekten Zuordnung notwendiger Beschreibungen und sind eine Ergänzung für von *doxygen* nicht erkannte Programmelemente oder Programmbestandteilen, welche einer zusätzlichen Erklärung bedürfen. In dem hier aufgeführten Beispiel betrifft das den, durch das Schlüsselwort „`@return`" beschriebenen, Rückgabewert der Funktion. Wäre eine entsprechende Kommentierung im Quellcode nicht vorgenommen worden, so wäre diese Information verloren gegangen.

Im Folgenden stellt sich die Frage, wie die vorliegenden Strukturen verwertet werden sollen. Es muss eine sinnvolle Verbindung zwischen diesen hergestellt werden und eine Aufbereitung für ein Zielformat erfolgen.

5.2 Die Bearbeitung der Daten mit Hilfe von XSL

Im Folgenden wird die Entwicklung der zur Bearbeitung, der in *XML* vorliegenden Daten, notwendigen Stylesheets vorgestellt. Hierbei bietet die eXtensible Stylesheet Language zwei Möglichkeiten. Zum einen kann aufgrund der hierarchischen Baumstruktur von *XML* eine Abhandlung der einzelnen Hierarchie-Ebenen durch Schleifenstrukturen vorgenommen werden, zum anderen kann ein direkter Zugriff auf die einzelnen Ebenen erfolgen. Eine Abarbeitung über Schleifenstrukturen ist in der folgenden Abbildung einer Abarbeitung durch einen direkten Zugriff gegenübergestellt.

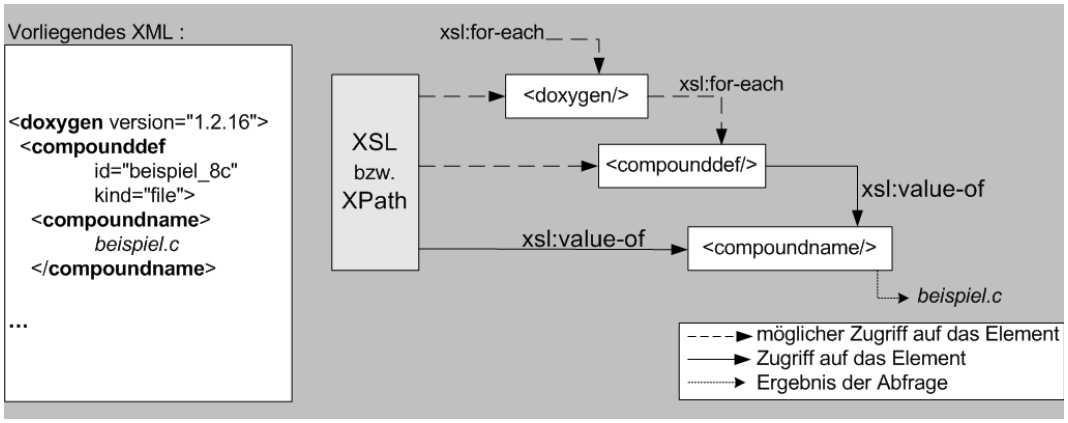

Abbildung 5.2: Schleifenstrukturen und direkter Zugriff

Die Nutzung von Schleifenstrukturen ist deshalb notwendig, da über das Beispiel hinausgehende *XML*-Strukturen komplex verzweigt sind und mehrere gleiche Auszeichnungen beinhalten. So ist es auf den Fall bezogen denkbar, dass mehrere Dateien in einer *XML*-Datei beschrieben werden. Somit gäbe es auch mehrere Auszeichnungen von „`<compoundname/>`" die sich jedoch im Inhalt unterscheiden. Die Schleifenanweisungen „`<xsl:for-each../>`" sorgen für den Wechsel zunächst in die Ebene des Elementes „`<compounddef/>`" und dann in die Ebene von „`<compoundname/>`". Erst dann ist der direkte Zugriff auf die Elemente mit Anweisungen wie beispielsweise „`<xsl:value-of select=""/>`" innerhalb der aktuellen Ebene möglich.

Eine andere Möglichkeit auf einzelne Elemente zuzugreifen bietet XPath an. Dies ist in dem Zusammenhang von Interesse, da eine einfache Bearbeitung einzelner Hierarchiestufen durch den beschriebenen teils notwendigen Rückgriff auf Schleifenstrukturen, dem im Folgenden beschriebenen Grenzen unterworfen ist. Diese werden deutlich, wenn man sich die generierte *XML*-Datei betrachtet. Die Parameter der Funktion werden korrekt erkannt und zugeordnet. Das Problem was sich aber ergibt ist, inwieweit kann innerhalb einer hierarchisch orientierten Schleifenstruktur ein Zugriff auf andere Hierarchieebenen erfolgen? Auf den Fall bezogen eröffnet sich somit die Fragestellung, wie soll der Zugriff auf die Parameterbeschreibung vollzogen werden, wenn die Schleifenstruktur sich in einer anderen Ebene befindet? Die Lösung ist die Nutzung sogenannter „templates".

Diesen können Paramter übergeben werden und in diesen genutzt werden.

Auf das Beispiel bezogen bedeutet dies, dass der Parametername, beschrieben durch die Auszeichnung „`<declname/>`" einem Template als Zeichenkette übergeben wird. Dort findet nach der Feststellung der Namensgleichheit der erkannten und der beschriebenen Struktur ein direkter Zugriff auf die Baumstruktur, unabhängig von der Bezeichnung des zugrunde liegenden Elements, statt.

...
<xsl:apply−templates **select**=”following−sibling::parameterdescription[position()=1]”/>
...

Diese Verfahrensweise ist nur möglich, weil das Element „`<declname/>`" in der selbständig erkannten Auszeichnung und das Element „`<parametername/>`" den gleichen Inhalt haben. Somit wird der Paramtername aus der selbständig erkannten Struktur übernommen, während die Beschreibung dessen aus dem `<parametername/>` folgenden Element `<parameterdescription/>` genutzt wird. Die hier beschriebene, umständlich anmutende Vorgehensweise hat jedoch den Vorteil, dass zunächst selbständig erkannte Programmbestandteile genutzt und aufbereitet werden können und zwar unabhängig davon, ob und inwieweit eine Kommentierung dieser im Quellcode erfolgt ist.

Die letztendliche Übergabe der aufbereiteten *XML*-Inhalte erfolgt über zusätzliche Anweisungen innerhalb der Stylsheets. Für die Überführung in einzelne Endformate ist eine Anweisung innerhalb der Kopfdeklaration von *XSL* notwendig. Zwar sind Endformate wie HTML und ähnliches möglich, jedoch soll wie bereits erwähnt, eine gezielte Aufbereitung der Informationen möglich sein. Deshalb wurde als zu generierendes Endformat *LaTeX* gewählt. Die Ausgabeanweisung am Anfang des Stylsheets lautet somit : „`<xsl:output method="text"/>`". Inhalte, die beispielsweise über „`<xsl:value-of select=""/>`" ausgelesen werden, sind somit in Textform verfügbar. Zudem macht die hierarchisch orientierte Abarbeitung der *XML*-Struktur durch Schleifen in dem Zusammenhang Sinn, da wie im Folgenden beschrieben, auf diesen Strukturen aufbauend eigene *LaTeX* -orientierte Strukturen gebildet werden können.

Mit Hilfe eines gängigen *XSLT*-Prozessors wird die *XSL* und *XML* verknüpft und in das gewünschte Endformat übertragen. Innerhalb des Prototypes kommt dabei

aufgrund seiner leichten Handhabung der Prozessor „*xsltproc*" [3] zum Einsatz.

5.3 Die Nutzung der Eigenschaften von TeX

Zunächst wäre es denkbar, die mittels der Stylesheets erlangten Informationen einfach in das gewünschtes Format zu übertragen, um dort eine Bearbeitung hinsichtlich Gliederung und Layout vorzunehmen. Dies hätte aber zum einen keine gleichbleibende Qualität zur Folge und zum anderen wären Kenntnisse im Umgang mit dem generierten Format notwendig. Die zur Formatierung des Textes notwendigen Anweisungen müssten daher nach genauen Richtlinien angewendet werden und der Aufwand zur Aufbereitung der Informationen wäre zu groß, als dass die bisher zur Automatisierung des Dokumentationsprozesses getätigten Schritte sinnvoll erscheinen. Der Lösungsansatz zur Erstellung eigener Strukturen in *LaTeX* wurde deshalb mit Hilfe der Definition spezifischer Umgebungen und Anweisungen umgesetzt.

Diese Definition erfolgt in *LaTeX* mittels der Anweisungen „`newenvironment`" und „`newcommand`". Die folgenden Auszüge zeigen beispielhaft eine derartige Umsetzung :

```
\newcommand{\file} [1] {\section*{{\tt{#1}}}{\label{sec:#1}}}
```

Hier wird eine neue Anweisung namens „`file`" definiert. Dieser wird ein Parameter übergeben, welcher dann in einer gesonderten Schrift und als Überschrift ausgewiesen wird und eine Markierung (label) erhält. Die folgende Ausführung soll den Einsatz der erstellten Anweisung und den Sinn dieser im Vergleich zur bisherigen Handhabung verdeutlichen.

```
\file {beispiel.c}
```

und die folgende, bisherige Abfolge erzeugen dabei das gleiche Ergbnis in der letztendlichen Ausgabe.

```
{\tt
\section {beispiel.c}}
\label{sec:beispiel.c}
```

[3] Quelle...

Somit können die notwendigen Anweisungen bereits mittels *XSL* gebildet werden, während auf die dafür notwendigen Definitionen in *LaTeX* zurückgegriffen wird. Innerhalb der Stylesheets erfolgt eine derartige Verfahrensweise demnach wie folgend:

```
...
<xsl:for-each select="doxygen">
  <xsl:for-each select="compounddef">
```

\file{<xsl:value-of select="compoundname"/> }

```
  </xsl:for-each>
</xsl:for-each>
...
```

Ähnlich verhält es sich mit der Definition eigener Umgebungen. Diese werden über „*begin*" und „*end*" - Strukturen gebildet. Die in der Umgebungsdefinition enthaltenen Layout-Anweisungen betreffen dann den gesamten Text innerhalb dieser Struktur. Eine Definition erfolgt wie im Folgenden dargestellt.

```
\newenvironment {Includes}
               {{\bf{Included files:}}\begin{itemize}}{\end{itemize}}
```

Der auf dieser Definition beruhende Aufruf von:

```
\begin{Includes}
\item beispiel.h
\end{Includes}
```

bewirkt als Ausgabe:

Included files:
• beispiel.h

Mittels herkömmlicher Anweisungen wäre die gleiche Ausgabe mit mehr Aufwand verbunden, wie folgende Ausführung zeigt.

```
{\bf{Included files:}}
\begin{itemize}
\item beispiel.h
\end{itemize}
```

Der Vorteil einer selbst definierten Umgebung wäre somit, dass allein aufgrund deren Bezeichnung eine bestimmte Strukturierung erkennbar ist. Bei der Bearbeitung des Dokumentes ist eine Auseinandersetzung mit der Darstellung des Textes seitens des Autors nicht mehr notwendig. Lediglich die Vollständigkeit und Richtigkeit der Informationen steht im Mittelpunkt. Sind dahingehend Ergänzungen notwendig, können diese einfach innerhalb der Grenzen der definierten Umgebungen hinzugefügt werden. Für den Autor einer Dokumentation ist deshalb lediglich das Wissen über die Bezeichnungen der Strukturierungselemente notwendig.

Die Definitionen der Umgebungen und Anweisungen können in externen Dateien ausgelagert werden. Dadurch können verschiedenen fallspezifische Dateien gebildet werden, welche die gleichen Anweisungs- und Umgebungsbezeichnungen, jedoch andere Layoutbestandteile, beinhalten.

Die nachfolgende Umwandlung des *LaTeX*-Formates in das gewünschte Endformat *PDF* ist aufgrund gängiger Werkzeuge, welche auch in zahlreichen Editoren für das Textsatzsystem enthalten sind auf komfortable und schnelle Art und Weise möglich.

5.4 Beispiel eines Softwareprojektes

Ein Softwareprojekt besteht in der Regel aus mehreren Quelldateien, welche miteinander verbunden sind. Sowohl *doxygen* als auch der darauf aufbauende Prototyp berücksichtigt diesen Aspekt. Die folgende Abbildung zeigt zunächst die Benutzeroberfläche des Prototyps, worauf im Folgenden die zugänglichen Funktionalitäten formal beschrieben werden.

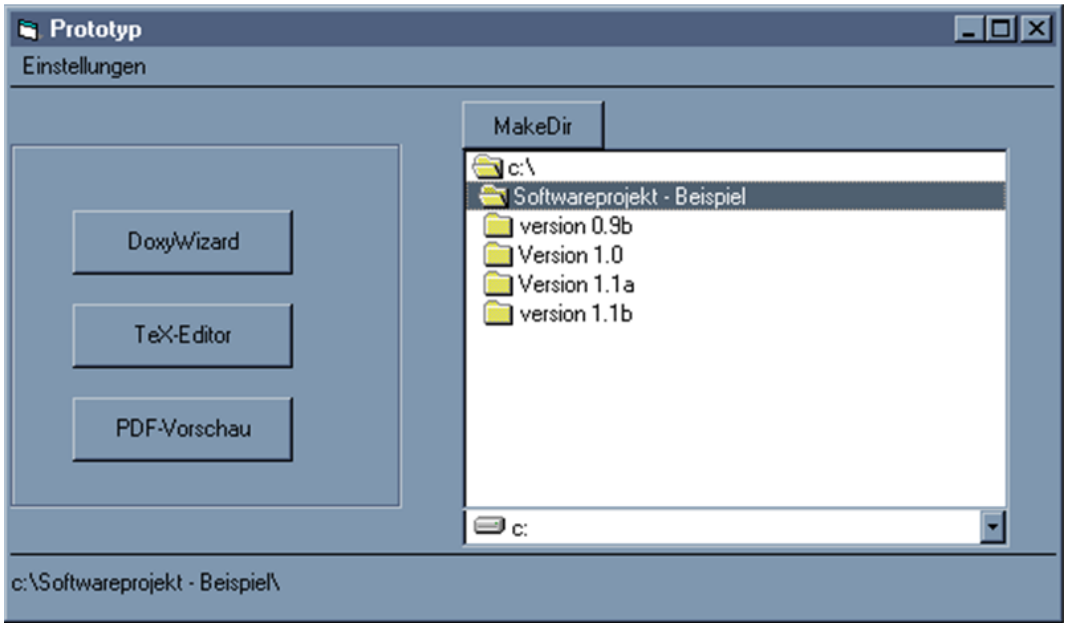

Abbildung 5.3: Benutzeroberfläche des Prototyps

Der entscheidende Ansatz sind die Einstellungen, welche innerhalb des Parsers vorgenommen werden. Je nachdem, welche Dateien und in welchem Umfang diese in das *XML*-Format überführt werden sollen, sind gewisse Einstellungen innerhalb des *Wizards* notwendig.

Liegen die *XML*-Dateien vor, erfolgt in einem nächsten Schritt, welche über die Schaltfläche „*TeX-Editor*" des Prototypes eingeleitet wird eine Verwendung der entwickelten Stylesheets. Mit Hilfe des *XSL*-Prozessors „*xsltproc*" werden alle innerhalb des Projektpfades vorliegenden *XML*-Dateien in das *TeX*-Format überführt. Somit liegt für jede Quelldatei eine *XML*- und eine *TeX*-Datei vor. In der Bearbeitung mit *XSL* wäre es zwar denkbar alle Dateien zu einer *XML*-Datei zu verbinden. Dies macht jedoch aufgrund der Fülle der Informationen welche dabei zusammengefasst werden, keinen Sinn, zumal in der Phase der Umwandlung der endgültige und vielfältige Verwendungszweck noch nicht klar ist. Die Berücksichtigung der Anzahl der Dateien findet jedoch über eine dynamische Verbindung dieser durch „Links[4]" statt. Dies geschieht über das Hinzufügen spezieller Anwei-

[4]Quelle: http://www.xipolis.net - Als Link wird der Verweis auf eine andere Stelle in einem

sungen in die definierten Anweisungen und Umgebungen. Die folgende Abbildung zeigt beispielhaft eine generierte *LaTeX*-Datei.

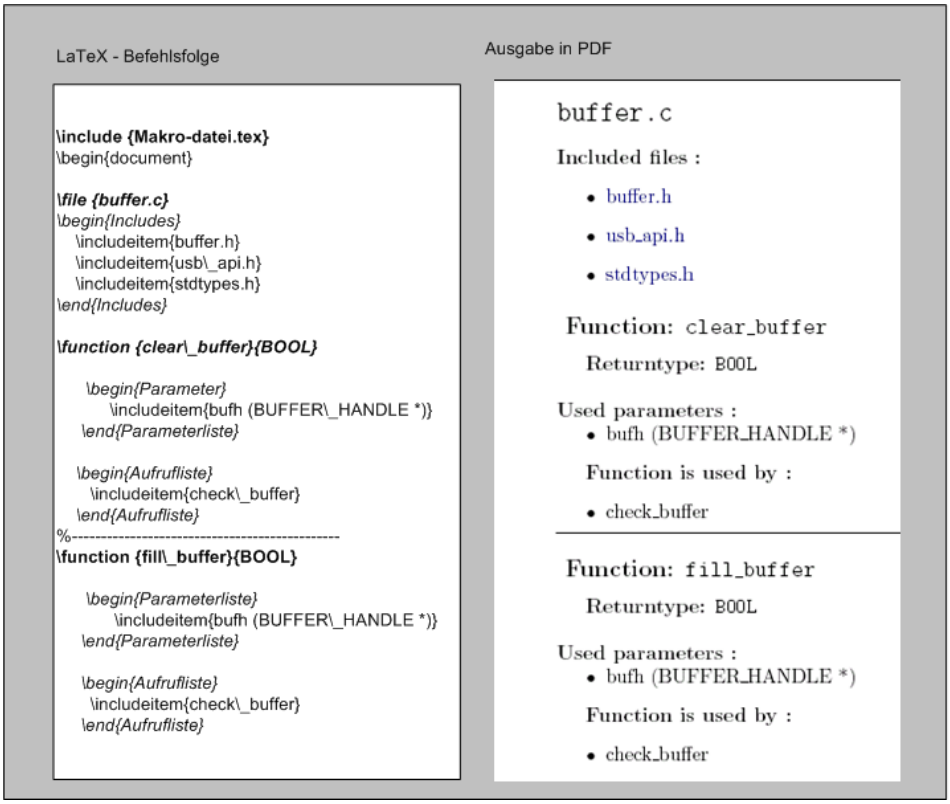

Abbildung 5.4: Die Umwandlung von *TeX* zu *PDF*

Dabei ist es notwendig, dass die Datei, welche die Definition der selbst erstellten Umgebungen und Anweisungen beinhaltet, in das Dokument durch eine „`\include`" Anweisung eingebunden wird. In der Umwandlung der *LaTeX*-Dateien in das *PDF*-Format, werden diese Layout-Anweisungen berücksichtigt, so dass eine gewünschte Präsentation entsteht. Somit ist mit Hilfe dieser eingebundenen Dateien eine einheitliche Präsentation, sowie eine zweckbezogene Berücksichtigung der Zielgruppe gegeben.

Hypertext-Dokument oder auf ein anderes Hypertext-Dokument bezeichnet

6 Zusammenfassung und Bewertung

Die Arbeit belegt das breite Spektrum des Themengebietes Dokumentation. Es wurde gezeigt, dass aufgrund der in einem Softwareentwicklungsprozess zahlreich anfallenden Dokumente und damit einhergehender qualitativer Anforderungen an diese, eine ganzheitliche Betrachtung des Prozesses Dokumentation notwendig wird. Prozessorientierte Standards, wie beispielweise die DIN ISO 9001:2000 leisten in diesem Umfeld einen wertvollen Beitrag, wobei produktspezfische Vorgaben wie die der IEEE 1063 eine sinnvolle produktorientierte Unterstützung darstellen.

Weiterhin wurde untersucht inwieweit sich die zeitintensiven und aufwendigen Erstellungsprozesse von Dokumentationen erleichtern lassen. Der Einsatz aktueller Technologien und Entwicklungen, wie etwa der Quellcodeparser „*doxygen*" und in dem Zusammenhang erstellbare Formate wurden im Hinblick auf Einsatzmöglichkeiten analysiert. Das Ergebnis und den notwendigen Zugriff auf die Daten ist in Abbildung 6.1 aufgeführt:

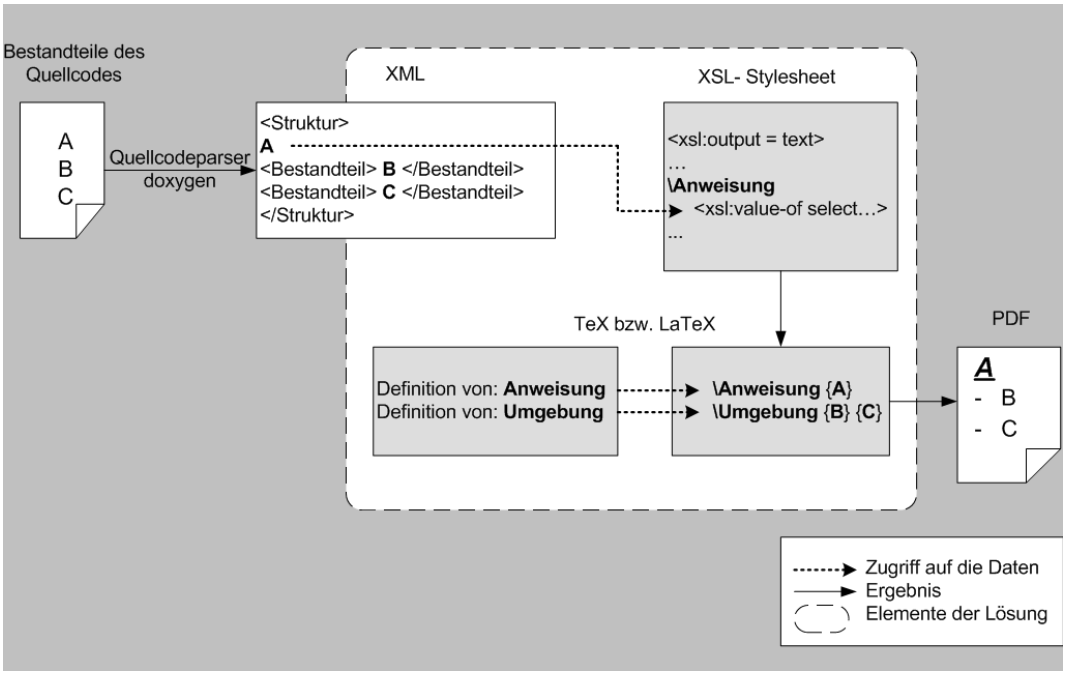

Abbildung 6.1: Überblick des Lösungsansatzes

Es zeigte sich, dass bereits sinnvolle Ansätze existieren, die vor allem durch das selbständige Erkennen vorhandener Strukturen und eine Überführung dieser in *XML* einen Ansatzpunkt für eine Automatisierung darstellen. Es zeigte sich aber auch, dass beim aktuellen Stand der Technik eine Automatisierung des Quellcodes gewissen Grenzen unterworfen ist. Ein selbständiges Erkennen von Programmbestandteilen, reicht für eine qualitativ hochwertige Dokumentation nicht aus. Zusammenhänge zwischen Programmbestandteilen und die Ziele welche mit dem Programm erreicht werden sollen, müssen mit Hilfe eines Textsystems hinzugefügt werden. Ansätze hinsichtlich Referenzdokumentationen, welche vorrangig eine Auflistung der Programmbestandteile zum Inhalt haben sind jedoch gegeben.

Zudem wurde auf die *XML* eingegangen und deren Einsatzmöglichkeiten innerhalb der Problemstellung geprüft. Dabei zeigte sich, dass *XML* ideal als Austauschformat und Grundlage für ein innerbetriebliches Dokumentenmanagement ist, dass aber auch zahlreiche Entwicklungen und Standardisierungsbemühungen innerhalb des Formates für die Lösung nicht geeignet sind. Somit wurde eine

Eigenentwicklung angestrebt. Dazu wurden zunächst die für die Verarbeitung der vorliegenden *XML*-Dateien notwendigen Stylesheets mit Hilfe von *XSL* entwickelt.

Bei der Entwicklung der Stylesheets stellte sich anschließend die Frage, in welches Datenformat die vorliegenden Informationen übertragen werden sollten. Aufgrund der Fehleranfälligkeit der Automatisierung und der Gefahr eines Informationsverlustes lag die Überführung in ein Format nahe, in dem eine abschließende Einflussnahme auf die Informationen möglich ist. Aufgrund der Strukturierungsmöglichkeiten des Textsatzsystems *TeX* wurde sich für dieses als bearbeitbares Zwischenformat entschieden. Die notwendigen Strukturen wurden mit Hilfe der Definition eigener Umgebungen und Anweisungen entwickelt. Diese Herangehensweise scheint in dem Zusammenhang sinnvoll, da zum einen nur die Strukturierungsanweisungen und der Text sichtbar sind und zum anderen diese Anweisungen für den jeweiligen Verwendungszweck, durch das Verändern der in den Anweisung enthaltenen Layoutbefehlen, anpassbar sind.

Abschließend wurde aus dem *TeX*-Format das *PDF*-Format erzeugt, welches aufgrund seiner vorgestellten Eigenschaften, als Publikationsformat geeignet ist.

Letztendlich zeigte sich, dass mit Hilfe der Nutzung aktueller Technologien eine Arbeitserleichterung im komplexen Umfeld der Dokumentation erreicht werden kann. Die oft aufwendige, verantwortungsvolle und deshalb häufig unbeliebte Aufgabe der Dokumentenerstellung lässt sich somit weitestgehend abnehmen. Vor allem für Softwareentwickler, welche vorrangig in kleinen bis mittleren Unternehmen mit derartigen Aufgaben betraut sind, werden somit entlastet, was auch zu einer qualitativen Verbesserung anderer Aufgabenbereiche führen kann. Für größere Softwareprojekte bedeutet die Automatisierung eine Entlastung, vor allem in den Bereichen Kommunikation und Fehleranfälligkeit. Autoren wie beispielsweise „*Technische Redakteure*" sind nicht mehr gezwungen sich in die Gebiete der Softwareprogrammierung einzuarbeiten, wodurch zum Beispiel die Rückfragehäufigkeit an die Softwareentwicklern sinkt.

Durch die Möglichkeit eigene Informationen in die Dokumentation einfließen zu lassen, ist die Gefahr einer unverständlichen Dokumentation eingeschränkt. Den häufig vorrangig auf die Programmierung bezogenen Programmbestandteilen kann somit eine sinnvolle Erklärung zugefügt werden. Weiterhin ist bei der

dargestellten Verfahrensweise eine Konsistenz zwischen Programmquellcode und Dokumentation stets gewährleistet.

7 Ausblick

Zunächst stellt der Prototyp eine Umsetzung der in Kapitel 4 genannten Lösungsansätze zur Automatisierung aufwendiger Arbeitsschritte dar. Jedoch soll die Verwendung eines *Projektfensters* innerhalb des Prototyps schon einen wichtigen Ansatz für eine mögliche Weiterentwicklung zeigen. Im Hinblick darauf, dass in der Regel mehrere Softwareentwickler an einem Projekt arbeiten, macht eine gezielte Verwaltung der Dokumente Sinn. Durch die Verwendung der im Prototyp genutzten Funktionalitäten, könnte sich jeder Entwickler schnell einen Überblick über das bisherige Projekt verschaffen, ohne sich in teilweise unkommentierten Quellcode einarbeiten zu müssen. Inwieweit die Archivierung und Verwaltung der Dokumente dabei erfolgen soll, ist auf der vorliegenden Arbeit aufbauend zu untersuchen. Durch eine projektorientierte Dokumentenverwaltung wäre zusätzlich die Nachvollziehbarkeit des Softwareentwicklungsprozesses anhand der angefallenen Dokumente, gewährleistet. Somit wären Ansätze, wie in den Normen, Standards und Richtlinien beschrieben zur Produkt- und Prozessverbesserung gegeben und müssten in der Weiterentwicklung sinnvoll ausgebaut werden.

Weiterhin stellt sich die Frage, wie die durch *doxygen* erkannten Informationen, weiter genutzt oder ausgebaut werden können. Es kann untersucht werden, inwieweit sich aus bestimmten Programmstrukturen zusätzliche Informationen ableiten lassen. So existieren beispielsweise Ansätze unter dem Begriff „Literate Programming". Dabei werden insbesondere die Kommentare im Quellcode intensiv genutzt. Die Idee dabei ist, dass ein Programmquellcode als eine Anleitung anzusehen ist, welche in eine maschinenabarbeitbare Form übersetzt werden kann[1].

Auch lassen sich aufgrund der dokumentierten Programm- und Projektstruktur

[1] vgl. [Mal]

Ansatzpunkte in der Verbesserung des Quellcodes, im Sinne eines „Refactoring" finden. Refactoring beschreibt dabei den Prozess, ein Softwaresystem so zu verändern, dass das externe Verhalten nicht geändert wird, der Code aber eine bessere interne Struktur erhält[2].

Eine Ausweitung der beschriebenen Funktionalitäten auf das breite Spektrum der Dokumentation und eine Integration in bisherige Hilfsmittel der Softwareentwicklung wäre somit denkbar.

[2]vgl. [Fow00] - Kapitel 1

A Anhang

- Beispiel für Musterdokumente der *DIN EN ISO9001*

Verfahrensanweisung VA-12: Erstellung und Pflege von Dokumenten
ZWECK:
Kontrolle von Dokumenten
Verantwortlich:
Alle operativen Einheiten von SOFTCRAFT
Verfahren:
Für die Erstellung, Behandlung und den Umgang mit Dokumenten gelten die folgenden Punkte:

1. Zur Erstellung von Dokumenten gibt es Produktmuster, die einzusetzen sind. Falls für ein Dokument kein Produktmuster vorliegt, ist das Produktmuster eines ähnlichen Produkts zu verwenden und anzupassen.
2. Für alle Dokumente ist ein einheitliches Deckblatt zu verwenden. Ein Muster dazu findet sich im Anhang.
3. Alle Dokumente werden überprüft, bevor sie angenommen und unter Konfigurations-kontrolle gestellt werden. Eine Methode dafür ist zum Beispiel die Fagan Inspection. Es kommt auch die Prüfung am Schreibtisch in Betracht.
4. Dokumente, die unter Konfigurationskontrolle stehen, werden nur nach dem dokumentierten formellen Verfahren via SCCB geändert. Damit wird die Verfolgbarkeit von Änderungen gesichert.
5. Mitarbeiter im Projekt, die bestimmte Dokumente für ihre Arbeit benötigen, erhalten diese als Kopie vom Konfigurationsmanagement. Das Original verbleibt beim Konfigurations-management.
6. Für Dokumente externer Herkunft gibt es ein separates Verfahren.
7. Veraltete Dokumente werden eingezogen und vom Konfigurationsmanagement noch zehn Jahre aufbewahrt. Diese Frist bezieht sich auf das Ausgabedatum des Dokuments.
8. Nach dieser Frist werden die Dokumente vernichtet.

Verfahrensanweisung VA-51: Erstellung von Dokumenten
ZWECK:
Vorgehen beim Erstellen von Dokumenten
Verantwortlich:
Software-Entwicklung, andere Einheiten der SOFTCRAFT
Verfahren:

- Erstellen Sie das Dokument unter Berücksichtigung eines Produktmusters, falls eines vorliegt.
- Überprüfen Sie, ob ein ähnliches Produktmuster verwendet werden kann, falls kein spezifisches Produktmuster für das Dokument verfügbar ist.
- Passen Sie die Struktur des Dokuments an, falls notwendig. Stützen Sie sich notfalls auf ein Dokument aus einem anderen Projekt und passen Sie den Text an.
- Verwenden Sie als Deckblatt ein Muster.
- Ziehen Sie verwandte Dokumente bei der Erstellung des Software-Dokuments heran, um Widersprüche zu vermeiden.
- Stimmen Sie den Inhalt des Dokuments mit den Entwicklern ab, die Schnittstellen bearbeiten.
- Legen Sie das Dokument dem Qualitätsmanagement vor und bringen Sie es anschließend unter Konfigurationskontrolle.

- Volere - Kategorien von Anforderungen

RANDBEDINGUNGEN FÜR DAS SYSTEM
- Der Zweck des Produkts
- Kunden und andere Beteiligte/Betroffene
- Nutzer des Produkts
- Randbedingungen für das Projekt
- Namenskonventionen und Definitionen
- Relevante Fakten
- Annahmen

FUNKTIONALE ANFORDERUNGEN
- Die Abgrenzung des Systems
- Anforderungen an Funktionen und Daten des Systems

NICHT-FUNKTIONALE ANFORDERUNGEN
- Oberflächenanforderungen
- Benutzbarkeitsanforderungen
- Performance/Durchsatz/Sicherheit
- Operationelle Anforderungen
- Wartungs- und Portierungsanforderungen
- Zugriffsschutzanforderungen
- Politische Anforderungen
- Gesetzliche Anforderungen

PROJEKTRANDBEDINGUNGEN
- Offene Punkte
- Fertiglösungen
- Neue Probleme
- Aufgaben
- Inbetriebnahme und Migration
- Risiken
- Kosten
- Benutzerdokumentation
- ”Warteraum”

- Volere - Requirement Shell

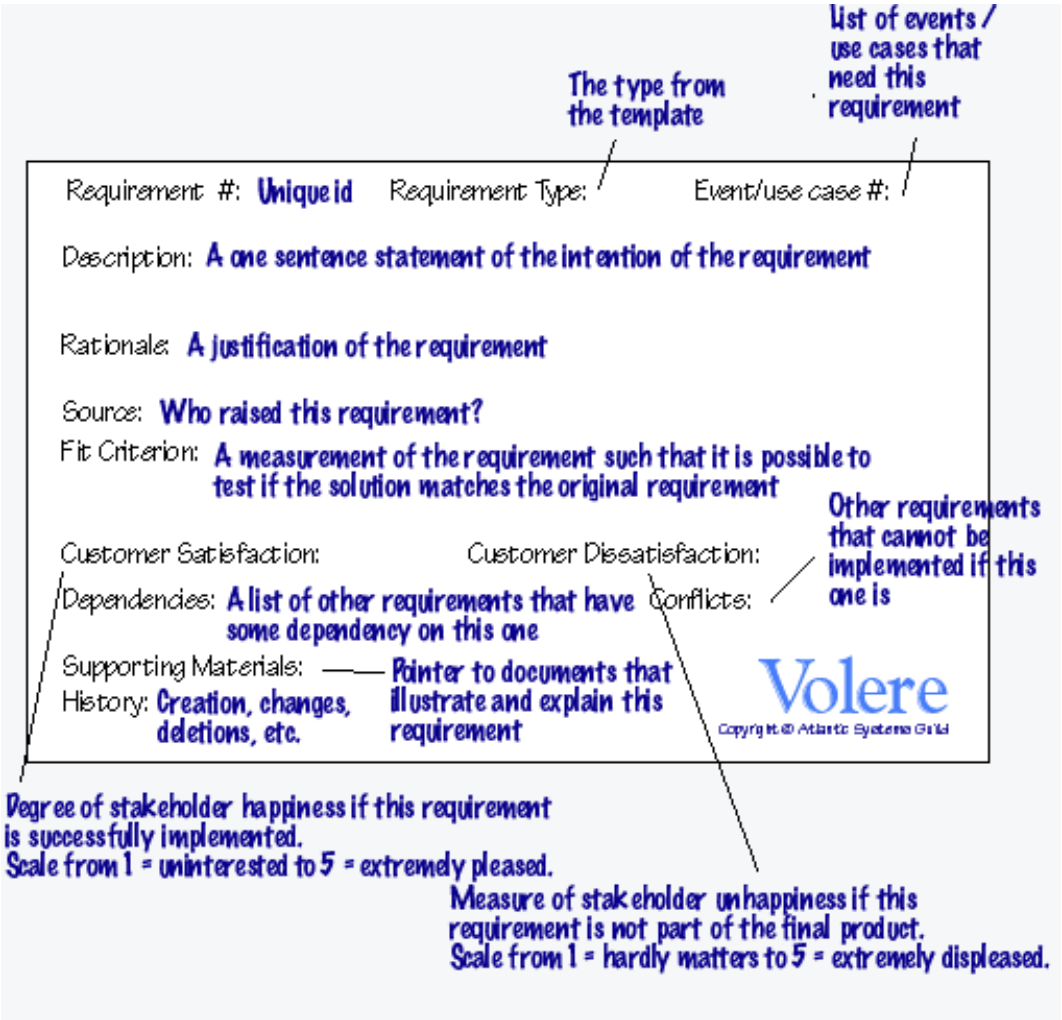

Abbildung A.1: Dokument zur Erfassung von Anforderungen

Literaturverzeichnis

[Arc01] Fabio Arciniegas. *XML Developer's Guide*. Franzis Verlag, Poing, 2001.

[Arg] Thomas Argast. *Was sind die Vorteile von PDF?* Universitätsbibliothek Freiburg, http://www.freidok.uni-freiburg.de/freidok/tutorial/pdf-vorteil.html.

[Beh99] Henning Behme. WWW:XML kommt. *iX - Magazin für profesionelle Informationstechnik*, 02:36ff, 1999.

[Bin] Hartmut F. Binner. *Prozessorientiertes Qualitätsmanagement und DIN EN ISO 9000:2000 - Auszug*. Symposion Publishing GmbH, http://www.symposion.de/qt/qt_04.htm.

[BM00] H. Behme and S. Mintert. *XML in der Praxis*. Addison-Wesley, München, 2000.

[Bul] Arno; Krieger Michael; Rohrbach Matthias Bullinger, Hans-Jörg Prof. Dr.-Ing. habil.; Hitzges. *Erfolgsfaktor technische Dokumentation*. tekom, http://www.tekom.de/PDF/Kurzfass.pdf.

[Cor] Microsoft Corporation. *Microsoft's Investment Strategy*. Microsoft Corporation, http://www.microsoft.com/msft/investphil.htm.

[Cot] Allin Cottrell. *Word Processors: Stupid and Inefficient*. Department of Economics at Wake Forest University, http://ricardo.ecn.wfu.edu/ cottrell/wp.html.

[Dat] Procon Data. *Effiziente Erzeugung von Druckoutputs basierend auf XML - Standard*. PROCON DATA Ges.m.b.H. Project Consulting, http://www.procon.co.at/Technologie_FO.htm.

[Deu96] Deutsches Institut für Normung. *Publikation und Dokumentation: Normen.* Deutsches Institut für Normung: Beuth, Berlin u.a., 1981-1996.

[DKE] DKE. *DIN EN 62079 (VDE 0039) Erstellen von Anleitungen; Gliederung, Inhalt und Darstellung.* Deutsche Kommission Elektrotechnik Elektronik Informationstechnik im DIN und VDE, http://www.dke.de/de/facharbeit/mitteilungen/62079check.htm.

[Dr.99] Dr. Loviscach,Jörn; Brors,Dieter; Himmelein,Gerald. Das Office-2000-Problem. *c't - Magazin für Computer Technik*, 15:122ff., 1999.

[dVG] doculine Verlags GmbH. *DIN EN 62079: neue Standardnorm für technische Redakteure.* doculine Verlags GmbH, http://www.doculine.com/news/2001/1201/din-62079.shtml.

[Ehr00] Ehrmann, Stefan; Brors, Dieter. Fünfkämpfer: Office-Pakete im Praxistest. *c't - Magazin für Computer Technik*, 23:182ff., 2000.

[Eis] Ulrich W. Eisenecker. *Generative Programmierung: Motivation für ein neues Paradigma der Softwaretechnik.* Vortrag am 06. Januar 1998 an der TU Ilmenau, http://home.t-online.de/home/Ulrich.Eisenecker/motiv.htm.

[Eur] Europanozert. *Die neue QM-Normenreihe.* Europanozert Zertifizierungen und Schulungen GmbH, http://www.europanozert.de/ISO9000-2000.pdf.

[Fow00] Martin Fowler. *Refactoring - Wie Sie das Design vorhandener Software verbessern.* Addison-Wesley, München, 2000.

[Grü] Gertrud Grünwied. *Unterstützung für Online-Hilfe-Redaktion durch die IEEE-Norm.* Wilken GmbH, Ulm, http://www.tekom.de/pdf/ceb02_02k.pdf.

[Gru] Matthias Grundmann. *Supply Chain Management - eine Schlüsselkompetenz im Wettbewerb.* HMD Heft 219 | Supply Chain Management - eine Schlüsselkompetenz im Wettbewerb, http://hmd.dpunkt.de/219/05.html.

[Hee] Dimitri van Heesch. *doxygen.* Dimitri van Heesch, http://www.stack.nl/ dimitri/doxygen/.

[Hit99] Arno Hitzges. *Referenzmodell für die Technische Dokumentation.* Jost-Jetter Verlag, Heimsheim, 1999.

[Huh01] Jochem Huhmann. XML: mit DocBook und Emacs arbeiten. *iX - Magazin für profesionelle Informationstechnik*, 09:134ff, 2001.

[Kay01] Michael Kay. *XSLT Programmer's Reference.* Wrox Press, Birmingham, 2nd edition, 2001.

[Kop00] Helmut Kopka. *LaTeX Band 1-3.* Addison-Wesley, München, 2000.

[Leh94] Franz Lehner. *Software-Dokumentation und Messung der Dokumentationsqualität.* Hanser, München; Wien, 1994.

[Loh] Christoph Lohnert, Roland; Jeloschek. *Ein (neues) REcht der Dienstleistung jenseits von Werk- und Dienstvertrag.* Universität Freiburg, Institut für Ausländisches und Internationales Privatrecht - GESELLSCHAFT JUNGER ZIVILRECHTSWISSENSCHAFTLER e.V., http://www.jura.uni-freiburg.de/Junge.Zivilrechtswissenschaftler/Freiburg2001/Tagungsband/JeloschekLohnert.

[Mal] Daniel Mall. *Literate Programming.* www.literateprogramming.com, http://www.literateprogramming.com/.

[Mer00] Freimut Mertens, Peter; Bodendorf. *Grundzüge der Wirtschaftsinformatik. 6. Aufl.* Springer, Berlin, Heidelberg, 2000.

[Mey] Christian Meyer, Anton; Blümelhuber. Dienstleistungen zur differenzierung bei zunehmender produkthomogenität. http://hmd.dpunkt.de/187/02.html.

[MH99] T. Tierney M.T. Hansen, N. Nohira. What´s your Strategy for Managing Knowledge? *Havard Business Review*, 2:106–116, 1999.

[PW00] O. Pott and G. Wielange. *xml-praxis und referenz.* Markt und Technik Verlag, München, 2000.

[Rob99] Suzanne; Robertson James Robertson. *Mastering the requirements process.* Addison-Wesley, Harlow, 1999.

[Rom] Felix A. Romberg, Markus; Ried. *Mit strukturierter Information zur flexibel wiederverwendbaren Dokumentation.* doculine Verlags

GmbH, http://www.doculine.com/news/2002/0402/redaktionssystem-grips.shtml.

[Scha] Rolf Schwermer. *Informationen zum Studiengang Technische Redaktion.* Fachhochschule Hannover - fachbereich für Informations- und Kommunikationswesen, http://www.ik.fh-hannover.de/tr/studiengang/.

[Schb] Rolf Schwermer. *Redaktionsleitfäden Ein unersetzliches Hilfsmittel in der Technikredaktion.* doculine Verlags GmbH, http://www.doculine.com/news/1998/04_98/Redaktionsleitfaden.htm.

[Sne91] Harry M. Sneed. *Softwarewartung und -wiederverwendung.* Rudolf Müller, Köln, 1991.

[Som92] Ian Sommerville. *Software engineering - 4th ed.* Addison-Wesley Publishers Ltd., München, 1992.

[Sta99] Peter; Ulrich Hasenkamp Stahlknecht. *Einführung in die Wirtschaftsinformatik - 9., vollst. überab. Aufl.* Springer, Berlin, Heidelberg, 1999.

[Tha01] Georg Erwin Thaller. *Iso9001:2000: Software-Entwicklungen in der Praxis - 3. aktualisierte Aufl.* Heise, Hannover, 2001.

[W3C] W3C. *The Extensible Stylesheet Language.* W3C World WideWeb Consortium, http://www.w3.org/Style/XSL/.

[Wal] Leonard Walsh, Norman; Muellner. *DocBook: The Definitive Guide.* O'Reilly and Associates, Inc., http://www.docbook.org/tdg/en/.