UNIVERSITÄT KOBLENZ-LANDAU
CAMPUS KOBLENZ, FACHBEREICH 4: INFORMATIK

Entwicklung eines Referenzmodells zur Erfassung und Verwaltung von Anforderungen

MASTERARBEIT

zur Erlangung des Grades eines Master of Science im Studiengang Informationsmanagement

vorgelegt von
Thomas Wehner
Koblenz im September 2006

Erstgutachter:
Prof. Dr. Jürgen Ebert, Universität Koblenz-Landau (Institut für Softwaretechnik)

Zweitgutachter:
Dr. Andreas Winter, Johannes Gutenberg Universität Mainz (Institut für Informatik)

Eidesstattliche Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Mit der Einstellung dieser Arbeit in die Bibliothek der Universität bin ich einverstanden.

Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.

Koblenz, 31. Oktober 2006 _____

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AK	Abnahmekriterium
ASCII	American Standard Code for Information Interchange
ASP	Active Server Pages
CPS	Change Proposal System
CSV	Comma Seperated Values
DOORS	Dynamic Object Oriented Requirements System
DXL	DOORS eXtension Language
HTML	Hypertext Markup Language
IDC	International Data Corporation
IEEE	Institute of Electrical and Electronics Engineers
INCOSE	International Council on Systems Engineering
ISO	International Organization for Standardization
MDA	Model Driven Architecture
MS	Microsoft
OMG	Object Management Group
RETH	Requirements Engineering Through Hypertext
RTF	Rich Text Format
RUP	Rational Unified Process
SQL	Structured Query Language
SR	System Requirement
TSV	Tab Seperated Values
UCM	Unified Change Management
UML	Unified Modelling Language
UR	User Requirement

Abstract

Problems in the analysis of requirements often lead to failures when developing software systems. This problem is nowadays being faced by *requirements engineering*. The early involvement of all kinds of stakeholders in the development of such a system and a structured process to elicitate and analyse requirements have made it a crucial factor as a first step in software development. The increasing complexity of modern softwaresystems though leads to a rising amount of information which has to be dealt with during analysis. Without the *support of appropriate tools* this would be almost impossible to do. Especially in bigger projects, which tend to be spatially distributed, an effective requirements engineering could not be implemented without this kind of support. Today there is a wide range of tools dealing with this matter. They have been in use since some time now and, in their most recent versions, realize the most important aspects of requirements engineering. Within the scope of this thesis some of these tools will be analysed, focussing on both the *major functionalities* concerning the management of requirements and the *repository* of these tools. The results of this analyis will be integrated into a *reference model*.

Zusammenfassung

Fehler in der Anforderungsanalyse führen häufig zu Misserfolgen in der Entwicklung von Softwaresystemen. Seit einiger Zeit wird versucht, diesem Problem durch *Requirements-Engineering* zu begegnen. Durch die frühe Beteiligung aller Stakeholder an der Entwicklung eines Systems sowie eine strukturierte Vorgehensweise zur Ermittlung und Analyse von Anforderungen an ein zu erstellendes System hat es als erster Schritt in der Entwicklung von Softwaresystemen zunehmend an Bedeutung gewonnen. Die steigende Komplexität moderner Softwaresysteme bringt jedoch eine gewaltige Menge an Informationen mit sich, die während der Analyse erfasst und verwaltet werden müssen. Dieser Informationsflut ist ohne *Unterstützung durch entsprechende Software* kaum beizukommen. Vor allem in größeren, räumlich verteilten Projekten wäre ein effektives Requirements-Engineering sonst kaum möglich. Es gibt inzwischen eine Vielzahl von Tools, die das Requirements-Engineering unterstützen. Diese Tools sind bereits seit geraumer Zeit im Einsatz und setzen in ihren aktuellsten Versionen die wichtigsten Konzepte des Requirements-Engineering um. Im Rahmen dieser Arbeit werden einige dieser Tools im Hinblick auf *Funktionsumfang und Verzeichnisstruktur* untersucht. Besonderes Interesse galt der Verwaltung von Anforderungen und deren Abhängigkeiten untereinander. Die gewonnenen Erkenntnisse werden anschließend in ein *Referenzmodell* integriert.

1 Einleitung

1.1 Motivation

Selbst die einfachsten Softwaresysteme sind heutzutage hochkomplex. Diese Komplexität führt häufig zu einer Explosion der beiden kritischen Erfolgsfaktoren Zeit und Kosten bei der Softwareentwicklung und ist damit verantwortlich dafür, dass ein hoher Prozentsatz von Projekten fehlschlägt. Dies zeigt der CHAOS Survey aus dem Jahr 1994 der Standish Group auf [in LW03, S. 6]:

> „In the United States, we spend more than $250 billion each year on IT application developement of approximately 175,000 projects. The average cost of a development project for a large company is $2,322,000; for a medium company, it is $1,331,000, and for a small company; it is $434,000. (...)
>
> The Standish Group research shows a staggering 31% of projects will be cancelled before they ever get completed. Further results indicate 52.7% of projects will cost 189% of their original estimates.
>
> Based on this research, the Standish Group estimates that (...) American companies and government agencies will spend $81 billion for cancelled software projects. These same organizations will pay an additional $59 billion for software projects that will be completed but will exceed their original time estimates.“

Diese Situation hat sich über die Jahre nicht wesentlich gebessert. Aktuelle Untersuchungen der Standish Group haben ergeben, dass inzwischen zwar 29% aller Projekte erfolgreich abgeschlossen werden, aber immer noch 18% der Projekte scheitern und 53% den Zeit- und Kostenrahmen sprengen [vgl. The04, S. 2].

Mindestens ein Drittel aller Projekte scheitert an Umständen, die direkt mit der Erhebung, Dokumentation und Verwaltung von Anforderungen in Beziehung stehen [vgl. LW03, S. 7]:

1. Mangelhafter Einfluss der Anwender: 13% aller Projekte
2. Unvollständige Anforderungen und Spezifikationen: 12% aller Projekte
3. Änderungen an den Anforderungen und Spezifikationen: 12% aller Projekte

Durch eine rechtzeitige Vermeidung bzw. Beseitigung von Fehlern in der Entwicklungsphase liessen sich Folgefehler vermeiden und so die kritischen Faktoren Zeit und Kosten in Grenzen halten. Denn „Fehler, die in einer frühen Phase der Softwareentwicklung gemacht und erst in späteren Phasen behoben werden (...), sorgen für einen Summationseffekt, bei dem sich die Fehler fortpflanzen und potenzieren. Tatsächlich lassen sich ca. 65% der schwerwiegenden Fehler in Programmen auf Unzulänglichkeiten in der Analyse zurückführen“ [vgl. Rup04, S. 10], wie aus Abbildung 1.1 hervorgeht:

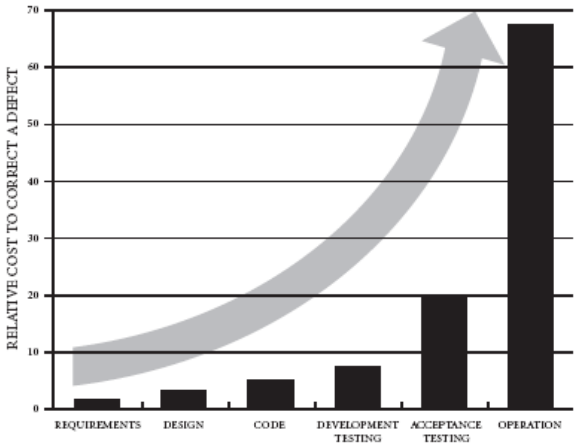


Abbildung 1.1: Exponentielles Wachstum der Kosten im Entwicklungsprozess [vgl. Bor05, S. 4]

Damit gewinnt die Anforderungsanalyse als erster Schritt der Entwicklung solcher Systeme zunehmend an Bedeutung. Dieser als *Requirements-Engineering* bekannte Prozess umfasst die Erhebung, Analyse, Spezifikation, Gültigkeitsprüfung und Dokumentation von Anforderungen. Das Requirements-Engineering beschreibt also einen systematischen Prozess von der Projektidee hin zu einem vollständigen Satz von Anforderungen an ein zu erstellendes System, wobei jede Anforderung eine Aussage über eine zu erfüllende Eigenschaft oder zu erbringende Funktion eines Systems, eines Prozesses oder der am Prozess beteiligten Personen ist [vgl. Rup04, S. 11]. Diesen Vorgang kann man als *Organisation und Dokumentation von Anforderungen bezeichnen.*

In praktisch allen Projekten ändern sich Anforderungen jedoch im Laufe der Entwicklung. Dies kann eine Vielzahl von Gründen haben. So entwickeln z.B. die am Projekt beteiligten Personen mit der Zeit ein besseres Verständnis für das zu erstellende System und präzisieren ihre Vorstellungen. Häufig beeinflussen auch projektexterne Faktoren die Anforderungen, etwa wenn sich Geschäftsprozesse oder Hard- und Softwareumgebung des Kunden ändern. Tatsächlich ergeben Studien eine durchschnittliche Änderungsrate aller Anforderungen von ca. 2% je Monat [vgl. Rup04, S. 42].

Das *Requirements-Management* stellt Techniken und Richtlinien zur Verfügung, um mit diesen Änderungen umzugehen. „Anforderungsänderungen werden erst durch die Einführung von Requirements-Management-Techniken systematisch handhabbar“ [vgl. Rup04, S. 12]. Durch das Requirements-Management sollen Entscheidungen und Abhängigkeiten zwischen Informationen von Projektbeginn bis zu seinem Ende nachvollziehbar gestaltet werden. Das *Verwalten von Anforderungen und deren Abhängigkeiten* zählt damit zu den wichtigsten Aufgaben des Requirements-Management.

Systemanalytiker, die für das Requirements-Engineering und -Management verantwortlich sind, haben also eine Vielzahl von Informationen zu verwalten, die mit steigender Projektgröße zunehmen. Alle Stakeholder müssen zum gleichen Zeitpunkt stets identische Informationen besitzen, da sie ansonsten unter Umständen auf falschen Vorgaben arbeiten. Daher ist es besonders wichtig, Informationen möglichst schnell zu verteilen und die Stakeholder über etwaige Änderungen automatisch zu informieren. [vgl. Rup04, S. 350]

Unterstützung erfahren Systemanalytiker bei diesen Tätigkeiten von computergestützten *Tools*. Ohne diese wäre ein effizientes Requirements-Engineering in räumlich verteilten, komplexen Projekten kaum zu verwirklichen. Dabei sind diese Tools nicht als Ersatz für ein methodisches Vorgehen gedacht. Sie dienen dem Anwender vielmehr als Unterstützung bei der Ausführung seiner Aufgaben. Aus den bereits aufgeführten Punkten ergeben sich folgende (minimale) Anforderungen an solche Tools:

- Organisation der Anforderungen
- Dokumentation der Anforderungen
- Verwalten der Anforderungen
  - Verwalten von Anforderungsabhängigkeiten
  - Verwalten von Anforderungsänderungen

Der grundsätzliche Aufbau solcher Tools ist in vielen Fällen identisch und entspricht weitestgehend der Darstellung in Abbildung 1.2:

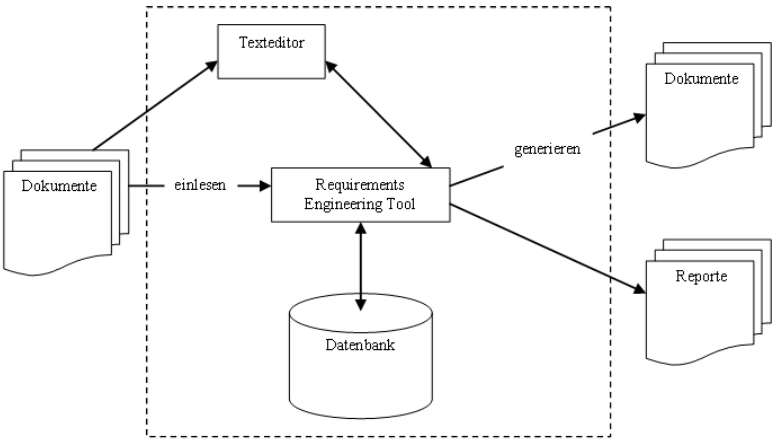


Abbildung 1.2: Grundsätzlicher Aufbau von Requirements-Engineering-Tools [vgl. Rup04, S. 390]

Demnach werden Anforderungen aus Anforderungsdokumenten mit Hilfe des Tools importiert oder über eines integrierten Text-Editors direkt eingegeben. Das Tool verwaltet diese Anforderungen in einer Datenbank und generiert bei Bedarf Anforderungsdokumente oder Analysen dieser Anforderungen in Form von Reporten.

1.2 Aufbau und Ziele der Arbeit

Im Zuge dieser Arbeit soll auf Basis einer Untersuchung gegenwärtiger Requirements-Engineering-Tools ein *Referenzmodell für die Organisation und Verwaltung von Anforderungen* entwickelt werden. Dabei sollen vor allem die Konzepte der *Traceability* (deutsch: Nachverfolgbarkeit) und *Reusability* (deutsch: Wiederverwendbarkeit) von Anforderungen näher untersucht werden.

Im zweiten Kapitel der Arbeit werden basierend auf dem Stand der gegenwärtigen Literatur die *wichtigsten Begriffe* dieser Arbeit erläutert, um eine gemeinsame Grundlage für das weitere Vorgehen zu sichern.

Das dritte Kapitel wird eine *Übersicht über den Markt aktueller Requirements-Engineering-Tools* geben. Mit Hilfe eines zuvor definierten Auswahlprozesses werden dann drei

Tools für die weiteren Untersuchungen ausgewählt. Mit Hilfe eines Beispiels, welches im vierten Kapitel näher beschrieben wird, werden diese Tools dann untersucht.

Im den nachfolgenden drei Kapiteln erfolgt eine *Untersuchung der Tools* auf ihre jeweiligen Funktionalitäten. Besondere Aufmerksamkeit soll dabei der Organisation und Verwaltung der Anforderungen gelten. Im Sinne des *Reverse-Engineering* werden zum Ende der Untersuchung aus den gewonnenen Erkenntnissen Modelle der Tools entwickelt.

Im achten Kapitel schließlich werden die drei Modelle als Basis für die *Entwicklung des Referenzmodelles* herangezogen. Dieses soll die wichtigsten Aspekte der Tools integrieren und so als Referenz für entsprechende Modelle dienen können. Mit Hilfe des Referenzmodells soll so die Grundlage für eine bessere Reusability und Traceability von Anforderungen im Requirements-Engineering geschaffen werden. Daraus ergeben sich die folgenden Punkte als Anforderungen an das Referenzmodell:

- Integration der wichtigsten Aspekte der untersuchten Tools
- Unterstützung von Traceability und Reusability
- Darstellung in einer standardisierten Modellierungssprache

Sämtliche Modelle der Anwendungen und das Referenzmodell werden mit Hilfe der *Unified Modelling Language (UML)* dargestellt. Sie ermöglicht es, die gewonnenen Erkenntnisse und deren Abhängigkeiten in einer standardisierten Sprache zu formulieren und mittels entsprechenden grafischen Notationen (→ Klassendiagramme) zu beschreiben.

In einem Fazit sollen letztlich die *Bedeutung des Referenzmodells* für spezifische Themen wie Traceability oder Reusability herausgearbeitet werden. In einem Ausblick soll das Anwendungspotential des Modells beschrieben werden.

2 Grundlagen des Requirements-Engineering

Das folgende Kapitel setzt sich mit den wesentlichen Konzepten des Requirements-Engineering auseinander. Es soll grundlegende Begriffe definieren und so die Basis für ein gemeinsames Verständnis schaffen.

Zu Beginn wird geklärt, was unter einer *Anforderung* zu verstehen ist und welche verschiedenen Arten von Anforderungen existieren. Anschließend widmet sich das Kapitel dem *Requirements-Engineering*. Im Zusammenhang mit der vorliegenden Arbeit wird dabei vor allem auf die Konzepte der *Traceability* und *Reusability* von Anforderungen genauer eingegangen.

Zum Ende dieses Kapitels wird schließlich der Begriff des *Referenzmodells* eingeführt. Darüber hinaus werden generelle Anforderungen an ein solches Referenzmodell definiert.

2.1 Anforderungen

Die Erhebung, Dokumentation und Verwaltung von *Anforderungen* ist das zentrale Aufgabengebiet des Requirements-Engineering. Das IEEE (Institute of Electrical and Electronics Engineers) definiert eine Anforderung wie folgt [vgl. Rup04, S. 138]:

1. Beschaffenheit oder Fähigkeit, die von einem Benutzer zur Lösung eines Problems oder zur Erreichung eines Ziels benötigt wird.
2. Beschaffenheit oder Fähigkeit, die ein System oder System-Teile erfüllen oder besitzen müssen, um einen Vertrag, eine Norm, eine Spezifikation oder andere, formell vorgegebene Dokumente zu erfüllen.
3. Eine dokumentierte Darstellung einer Bedingung oder Fähigkeit gemäß 1 oder 2.

Auch Sommerville beschreibt Anforderungen als Aussagen über Dienste, Einschränkungen und Ziele eines Systems, die durch Beobachtung existierender Systeme und in

Zusammenarbeit mit den potentiellen Anwendern des Systems ermittelt werden und im weiteren Verlauf als Systemspezifikation dienen. [vgl. Som01, S. 45ff.] Er bezieht dabei, stärker als die Definition des IEEE, die Anwender und die Umgebung des zu erstellenden Systems mit in seine Definition ein.

> „Requirements are defined during the early stage of a system development as a specification of what should be implemented. They are descriptions of how the system should behave, application domain information, constraints on the system's operation, or specifications of a system property or attribute. Sometimes they are constraints on the development process of the system." [vgl. KS03, S. 6]

Rupp ersetzt in ihrer Definition den Begriff System durch *Produkt* und versucht ihn dadurch zu erweitern. Damit sollen neben Anforderungen an Software und Hardware auch solche an Handbücher, Protokolle, Planungsdokumente usw. erfasst werden [vgl. Rup04, S. 138f.].

In der vorliegenden Arbeit soll die folgende Definition gelten:

Definition: *Eine* Anforderung *beschreibt eine Eigenschaft oder eine Fähigkeit, die ein System besitzen oder erfüllen muss, um den potentiellen Anwender bei der Bewältigung seiner Arbeit zu unterstützen, und um gesetzliche, vertragliche, organisatorische oder andere formelle Rahmenbedingungen einzuhalten.*

Einigkeit herrscht in der Literatur über die Notwendigkeit, Anforderungen nach bestimmten Merkmalen zu gruppieren. Am weitesten verbreitet ist die grobe Unterteilung in funktionale und nicht-funktionale Anforderungen, die sich immer wieder findet [vgl. Som01; Rup04; RR99]. Laut Rupp ist „den maßgeblichen Literaturquellen gemein, dass sie sich über verschiedene *Arten* und verschiedene *Ebenen* von Anforderungen äußern [vgl. Rup04, S. 139]." Zunehmend wichtiger werde auch die Gruppierung nach der *Priorität*. Zwar existieren eine ganze Reihe weiterer Merkmale, oft sind diese aber nur innerhalb bestimmter Rahmenbedingungen sinnvoll und sollten dementsprechend den Bedürfnissen angepasst werden.

Im Folgenden sollen die funktionalen und nicht-funktionalen Anforderungen als wichtigste Gruppen genauer voneinander abgegrenzt werden.

Funktionale Anforderungen an ein System beschreiben Funktionen oder Tätigkeiten, die das System selbstständig ausführen können soll. In einigen Fällen können sie auch Informationen darüber enthalten, wie sich das System in einer bestimmten Situation *nicht* verhalten soll. Funktionale Anforderungen beschreiben darüber hinaus Interaktionsmöglichkeiten von System und Anwender, also Input und Output des Systems.

Dagegen beschreiben *nicht-funktionale Anforderungen* an ein System keine Funktionen an sich, sondern Eigenschaften des Systems wie Zuverlässigkeit, Qualität oder Zugriffsgeschwindigkeiten. Mit Hilfe von nicht-funktionalen Anforderungen können auch Einschränkungen für das System formuliert werden, wie z.B. bestimmte Standards, die es zu erfüllen gilt.

Eine sehr ausführliche Darstellung verschiedener Typen von nicht-funktionalen Anforderungen findet sich bei SOMMERVILLE [vgl. Som01, S. 102]. Prinzipiell unterscheidet er drei wesentliche Typen:

- *Product requirements* - Anforderungen, die das System- bzw. Produktverhalten spezifizieren, z.B. Anforderungen an die Performanz oder die Zuverlässigkeit eines Systems.
- *Organizational requirements* - Anforderungen, die z.B. von bestimmten Geschäftsprozessen abgeleitet werden. Darunter können Prozessstandards oder Anforderungen an die Implementation (beispielsweise eine konkrete Programmiersprache) fallen.
- *External requirements* - Anforderungen, die von Faktoren beeinflusst werden, die außerhalb des Systems und dessen Entwicklungsprozess liegen. Das kann z.B. die Interoperabilität eines Systems mit anderen Systemen einer Organisation betreffen, gesetzliche Anforderungen oder ethische Anforderungen.

2.2 Requirements-Engineering

Um Anforderungen an ein System zu erfassen und zu verwalten, bedient man sich heute des Requirements-Engineering. Der Begriff lässt sich wie folgt definieren:

Definition: *Das* Requirements-Engineering *umfasst sämtliche Aktivitäten, um Anforderungen an ein System zu erheben, zu dokumentieren und zu validieren. Requirements-Engineering ist ein ingenieurstechnisch orientierter Prozess zur Entwicklung einer Anforderungsspezifikation.*

Der Requirements-Engineering-Prozess kann grafisch wie folgt dargestellt werden:

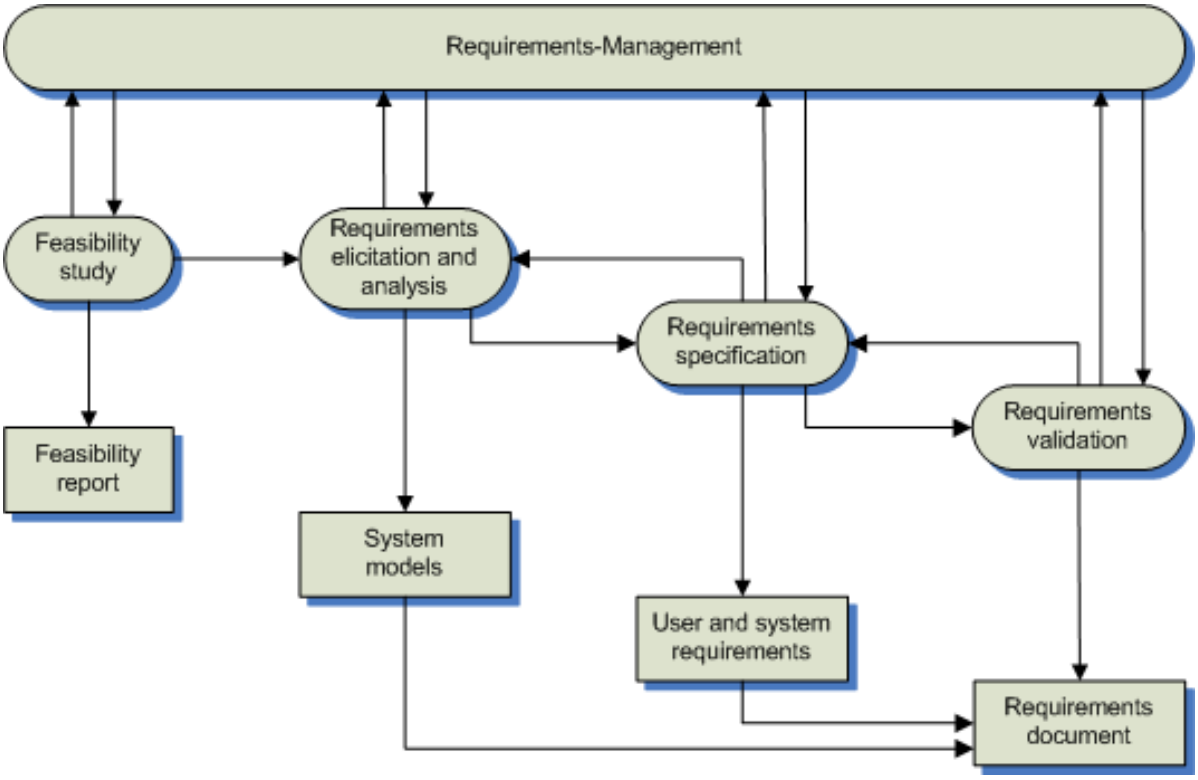


Abbildung 2.1: Der Requirements-Engineering-Prozess, [nach Som01, S. 122]

Demzufolge lässt sich der Requirements-Engineering-Prozess grob in vier Phasen aufteilen. Die einzelnen Phasen beeinflussen sich wechselseitig und sind nicht strikt getrennt voneinander zu betrachten.

In der ersten Phase wird eine *Machbarkeitsstudie* (englisch: *Feasibility Study*) entworfen. Sie soll vor allem sicherstellen, dass das zu erstellende System mit aktuellen (technischen) Möglichkeiten innerhalb eines bestimmten Zeit- und Kostenrahmens implementiert werden kann. Außerdem wird überprüft, ob das System in die bestehende Systemlandschaft der Organisation integriert werden kann.

In der zweiten Phase des Requirements-Engineering-Prozesses werden die Anforderungen an das System erhoben und analysiert (englisch: *Requirements elicitation and analysis*). In dieser Phase arbeiten Systemanalysten und -entwickler mit Kunden und Endnutzern zusammen, um die Systemumgebung kennen zu lernen und Anforderungen an das System zu sammeln. Diese können z.B. die Systemumgebung, gewünschte Funktionalitäten oder Hardwarevoraussetzungen betreffen. Häufig gestaltet sich diese Phase sehr schwierig. Die betroffenen Stakeholder wissen oft noch nicht genau, was sie von dem System im Einzelnen erwarten. Bei einer Vielzahl von Stakeholdern kommt darüber hinaus eine

große Menge unterschiedlicher Anforderungen an ein System zusammen, die sich zum Teil auch widersprechen können. Auch die Sprache, in der die Anforderungen formuliert werden, ist von vielen Faktoren (z.B. Bildung, Nationalität, Arbeitsfeld) abhängig und kann von Person zu Person variieren. Systemanalysten müssen diese Schwierigkeiten erkennen und mögliche Konflikte beseitigen. Darüber hinaus können auch politische, finanzielle und organisatorische Umstände die Erhebung und Analyse von Anforderungen beeinflussen.

In der dritten Phase werden die gesammelten Anforderungen verfeinert (englisch: *Requirements specification*). Aus einer allgemeinen Systembeschreibung werden so nach und nach immer genauere Anforderungen an Teilvorgänge des zu erstellenden Systems entwickelt. Je detaillierter die Anforderungen formuliert werden, desto formaler wird die Sprache. Während die erste, grobe Systembeschreibung z.B. noch in natürlicher Sprache formuliert sein kann, können später Use Cases oder Programmiersprachen zum Einsatz kommen. Dadurch wird es möglich, die Anforderungen auf jeder Detailebene zielgruppenspezifisch zu formulieren.

Die vierte Phase führt schließlich zum *Anforderungsdokument* (englisch: *Requirements document*). Bevor eine Anforderung in diesem Dokument aufgenommen wird, muss sie eingehend überprüft werden (englisch: *Requirements validation*). So soll gewährleistet werden, dass die Anforderungen auch tatsächlich das System nach den Vorstellungen des Kunden beschreiben. Anforderungen sollen auf Vollständigkeit, Relevanz, Zusammengehörigkeit, Konsistenz, Nachverfolgbarkeit und einige weitere Qualitäten hin überprüft werden. Diese Phase ist sehr wichtig, da eine Fehlerbehebung nach dieser Phase im fortschreitenden Entwicklungszyklus immer aufwändiger und teurer wird.

2.2.1 Requirements-Management

In nahezu allen Projekten kommt es früher oder später dazu, dass sich die Anforderungen an das System ändern. Mit Hilfe des *Requirements-Management* sollen diese Änderungen verstanden und kontrolliert werden. Das Requirements-Management wird in Verbindung mit den einzelnen Phasen des Requirements-Engineering-Prozesses durchgeführt. Erst dadurch wird es möglich, Anforderungsänderungen systematisch handhabbar zu gestalten. Im folgenden Abschnitt wird daher näher auf dieses Thema eingegangen.

Definition: *Das* Requirements-Management *bezeichnet den Prozess der systematischen Verwaltung von Anforderungen, deren Abhängigkeiten untereinander und Änderungen an Anforderungen. Es umfasst Maßnahmen zur Steuerung, Kontrolle und Verwaltung von Anforderungen, z.B. Risikomanagement oder Veränderungsmanagement.*

Ein solcher Prozess ist notwendig, da sich Anforderungen aus verschiedenen Gründen im Laufe eines Projekts verändern:

1. Unvollständigkeit von Anforderungen
2. Widersprüchlichkeit von Anforderungen
3. Konkurrenz von Anforderungen verschiedener Stakeholder (z.B. Anwender vs. Techniker)
4. Unbeständigkeit von Anforderungen (z.B. durch steigendes Verständnis der Systemdomäne oder wechselnde Prioritäten der Stakeholder)
5. Wechselnde Rahmenbedingungen (z.B. geänderte Hardware- oder Softwareanforderungen, neue gesetzliche oder organisatorische Bestimmungen)

Anforderungsänderungen sind unvermeidlich. Es ist aber nicht ungewöhnlich, dass einige Anforderungen weniger anfällig für solche Änderungen sind als andere. SOMMERVILLE und KOTONYA unterscheiden daher *stabile* und *unbeständige* Anforderungen [vgl. KS03, S. 115f.]. Während die stabilen Anforderungen eher die grundlegenden Eigenschaften des Systems und seiner Umgebung beschreiben, betreffen die unbeständigen Anforderungen konkrete Instanzen eines Systems in einer bestimmten Umgebung und für einen bestimmten Kunden.

In jedem Fall aber müssen Anforderungsänderungen verwaltet werden, um eine hohe Qualität der Anforderungen zu gewährleisten. Auch die Machbarkeit der Änderungen muss überprüft und deren Realisierbarkeit innerhalb des gesetzten Zeit- und Kostenrahmens überprüft werden. Das Requirements-Management stellt den verantwortlichen Analytikern Techniken und Richtlinien zur Verfügung, mit denen sie diesen Aufgaben nachkommen können.

Die beiden großen Aufgabengebiete sind nach SOMMERVILLE und KOTONYA vor allem *Change Control Management* und *Change Impact Assessment* [vgl. KS03]. Während sich das Change Control Management vor allem mit der Erhebung, Bestätigung und Abwägung von Änderungsvorschlägen befasst, untersucht das Change Impact Assessment vor allem deren mögliche Auswirkungen auf das System als Ganzes. Diese beiden Aufgaben können unter dem Begriff Veränderungsmanagement (englisch: *Change Management*) zusammengefasst werden.

Im Zusammenhang mit dem Veränderungsmanagement ist die Traceability von entscheidender Bedeutung für das Requirements-Management. Mit Hilfe von Links können zusammengehörige Anforderungen verbunden auf Änderungen hin zu überwacht werden. Das Konzept der Traceability wird im folgenden Abschnitt noch einmal ausführlich vorgestellt.

2.2.2 Traceability

Für das Requirements-Management und vor allem das Veränderungsmanagement ist es unverzichtbar, die Auswirkungen von Anforderungsänderungen auf das System zu erfassen. Um Anforderungen und Anforderungsänderungen effektiv verwalten zu können, ist es daher notwending, Informationen über deren Beziehungen untereinander zu pflegen und nachverfolgbar zu gestalten. Eine Anforderung ist genau dann nachverfolgbar (englisch: *traceable*), wenn

- der Ursprung der Anforderung bekannt ist,
- der Grund für die Existenz der Anforderung feststeht,
- verwandte Anforderungen ermittelt werden können und
- die Beziehungen der Anforderungen zu weiteren Komponenten des Systems bekannt sind.

Diese Informationen werden auch als *Traceability-Informationen* bezeichnet [vgl. SS99, S. 217]. In einem Projekt gibt es verschiedene Arten dieser Informationen, da es Beziehungen nicht nur zwischen Anforderungen und Anforderungen, sondern auch zwischen Anforderungen und der Design-Spezifikation sowie zwischen Anforderungen und deren Ursprung gibt. Sommerville unterscheidet daher drei Arten von Traceability-Informationen [vgl. Som01, S. 142]:

1. *Source traceability information* links the requirements to the stakeholders who proposed the requirements and to the rationale for these requirements.
2. *Requirements traceability information* links dependent requirements within the requirements document.
3. *Design traceability information* links the requirements to the design modules where these requirements are implemented.

Besonders interessant ist heute die Frage nach der Nachverfolgbarkeit der Anforderungen bis in den Code hinein. Es gibt vor allem zwei Konzepte, welche diese Themas ausmachen:

- In Verbindung mit dem Konzept der Model Driven Architecture (MDA)[1] werden sich für die gesamte Softwareentwicklung in der Zukunft neue Perspektiven ergeben. Anforderungen können mittels geeigneter Standards in Modelle überführt werden, aus welchen im Idealfall automatisch Code erzeugt werden kann.
- Durch die Wiederverwendung von Anforderungen (vgl. Abschnitt 2.2.3, S. 13) ergeben sich eine Reihe interessanter Möglichkeiten, die Softwareentwicklung effektiver und effizienter zu gestalten. Eventuell können nicht nur Anforderungen, sondern ganze Code-Fragmente, die sich aus ihnen ergeben, in neuen Systemen wieder verwendet werden.

Die konsequente Pflege der Traceability-Informationen ist daher von äußerster Bedeutung für das Requirements-Management und den gesamten Entwicklungsprozess, da nur so Anforderungen und Anforderungsänderungen effektiv verwaltet werden können. Vor allem in hochkomplexen, räumlich verteilten Projekten mit einer Vielzahl von Anforderungen ist dies in der Praxis ohne die entsprechende Softwareunterstützung kaum möglich. Daher sollte ein Requirements-Engineering-Tool mit den entsprechenden Konzepten aufwarten, die ein wirkungsvolles Management der Traceability-Informationen ermöglichen.

2.2.3 Reusability

Viele der Systeme, die heute entwickelt werden, haben zwar spezielle Charakterisitiken, welche sie definieren. Sie sind aber durchaus nicht einzigartig. Teile eines solchen Systems sind eventuell schon in anderen Systemen implementiert und erfüllen dort seit langer Zeit ihre Arbeit. Diese zu wieder zu verwenden bedeutet ein effektiveres und effizienteres Requirements-Engineering. Man spricht in diesem Fall auch von der *Reusability* von Anforderungen.

[1] The Model Driven Architecture (MDA) is a framework for software development developed by the Object Management Group. Key to MDA is the importance of models in the software development process. Within MDA the software development process is driven by the activity of modeling your software system [vgl. Kle03, S. 5f.].

Während also bestimmte Eigenschaften eines Systems nicht auf ein anderes System übertragen werden können, gibt es Situationen, in denen dies eher der Fall ist. SOMMERVILLE und KOTONYA geben dafür einige Beispiele [vgl. KS03, S. 72]:

- Anforderungen, welche die Domäne eines Systems beschreiben.
  Beispiel: Die Spezifikation eines Bremssystems für einen Zug wird auch Informationen darüber enthalten, wie dessen Masse und Geschwindigkeit, das Gefälle der Gleise usw. die Bremszeit des Zuges beeinflussen. Diese Informationen lassen sich für ähnliche System wahrscheinlich wieder verwenden.
- Anforderungen, die sich mit der Präsentation von Informationen beschäftigen.
  Beispiel: Ein konsistentes Look-and-Feel von User-Interfaces wird Übergangsprobleme der Anwender von einem System auf das nächste verringern.
- Anforderungen, welche die Firmenpolitik wiedergeben.
  Beispiel: Wenn die Unternehmenspolitik den Schutz persönlicher Daten vorschreibt, können Anforderungen für die Verschlüsselung solcher Daten Teil mehrerer System-Spezifikationen sein.

SOMMERVILLE und SAWYER beschreiben außerdem zwei verschiedene Möglichkeiten, Anforderungen wieder zu verwenden, die direkte und die indirekte Wiederverwendung [vgl. SS99, S. 106f.]. Bei der *direkten Wiederverwendung* von Anforderungen werden bestehende Anforderungen an ein bestimmtes System mit minimalen Änderungen als Anforderungen für ein nächstes System übernommen. Diese Form der Wiederverwendung ist allerdings meist nur dann möglich, wenn eine große Ähnlichkeit zwischen den Systemen besteht. Bei der *indirekten Wiederverwendung* dagegen werden die bestehenden Anforderungen als Anregung für die Stakeholder benutzt. Diese modifizieren die Anforderungen, um ihre speziellen Wünsche an das zu erstellende System zu beschreiben.

Während noch vor einiger Zeit vor allem die Wiederverwendung von „harten“ Artefakten (wie z.B. Code) interessierte, werden heute vermehrt auch die Anforderungen diesbezüglich untersucht. Denn wenn eine Anforderungen bereits umgesetzt wurde und sich in einem neuen System wieder verwenden lässt, dann gilt das im Idealfall auch für Artefakte, die aus ihr abgeleitet werden können, wie Design-Spezifikationen, Code oder Objekte. So kann manch auch aus den nachgelagerten Produkten der Anforderungen Vorteile ziehen.

Requirements-Engineering-Tools sollten ihren Beitrag dazu leisten, Anforderungen möglichst wiederverwertbar zu gestalten und verwalten. Häufig ist dies Teil der Versionsverwaltung solcher Tools. Ein Konzept, wiederverwendbare Anforderungen zu erhalten, ist das Festhalten von Entwicklungszuständen bestimmter Teilsysteme. Solche Versionen werden allgemein als *Baseline* bezeichnet.

Der Begriff Baseline stammt aus der Softwareentwicklung und bezeichnet nach Versteegen „das Einfrieren eines Entwicklungszustandes zu einem bestimmten definierten Zeitpunkt“ [vgl. Ver04, S. 120]. Genauer definiert es Rupp: „Als Baseline (...) bezeichnet man eine genau spezifizierte Menge an Informationen (z.B. Anforderungen, Abnahmekriterien) zu einem definierten Zeitpunkt, welche im Rahmen des Konfigurationsmanagements verwaltet werden“ [vgl. Rup04, S. 379]. Dieser Zeitpunkt kann zum Beispiel durch das Erreichen eines Meilensteins markiert werden. Eine Baseline ermöglicht es damit, ein Projekt jederzeit wieder auf einen vordefinierten Stand aufzusetzen. Eine Baseline muss nicht notwendigerweise alle Informationen eines Projekts enthalten, sie kann auch nur bestimmte Bereiche eines Projekts betreffen (falls z.B. ein Teilprojekt bereits in sich abgeschlossen ist).

2.3 Referenzmodell

Die Referenzmodellierung setzt sich mit der Konstruktion von Modellen auseinander. Daher erscheint es hilfreich, zunächst die Bedeutung des Modellbegriffs zu klären.

Definition: *Ein* Modell *ist die Abbildung eines Originals. Es abstrahiert das Original und kann damit nur Teile von diesem darstellen.*

Für ein Referenzmodell dagegen soll folgende Definition gelten:

Definition: *Ein* Referenzmodell *stellt ein idealtypisches Modell dar, welches „Menschen zur Unterstützung der Konstruktion von Anwendungsmodellen entwickeln oder nutzen, wobei die Beziehung zwischen Referenz- und Anwendungsmodell dadurch gekennzeichnet ist, dass Gegenstand oder Inhalt des Referenzmodells bei der Konstruktion des Gegenstands oder Inhalts des Anwendungsmodells wieder verwendet werden [vgl. Bro03, S. 34].“*

Daraus ergeben sich die folgenden Merkmale eines Referenzmodells:

- Ein Referenzmodell emöglicht die Planung und Konstruktion spezieller Anwendungsmodelle.
- Ein Referenzmodell ermöglicht Vergleiche mit anderen Modellen, welche die gleichen Sachverhalte beschreiben.

Darüber hinaus wird für ein Referenzmodell gerne ein allgemein gültiger Empfehlungscharakter angenommen. Dieser lässt sich aber objektiv nur schwer nachweisen. Sowohl die Allgemeingültigkeit als auch der Charakter einer Empfehlung können nur unter bestimmten, dem Modell eigenen Voraussetzungen gültig sein. Damit erweist sich schon die

Forderung nach *Allgemein*gültigkeit als sprachlich unpassend. Laut Vom Brocke ist außerdem nicht davon auszugehen, dass eine Gültigkeit von Modellen objektiv überhaupt existiert. Problematisch erweist sich seiner Meinung nach auch die mangelnde Überprüfbarkeit des Gehalts einer Empfehlung. Insbesondere seien „keine kritischen Werte verfügbar, ab denen die Eigenschaft der Empfehlungswürdigkeit zu- oder abzusprechen ist. Ob sie tatsächlich vorliegt, ist somit für ein Referenzmodell nicht objektiv anzugeben, sondern entscheidet sich erst in dessen Anwendung [vgl. Bro03, S. 32]."

3 Auswahl der Tools

Ziel des folgenden Kapitels ist es, aus dem gesamten Angebot kommerzieller und nichtkommerzieller Requirements-Engineering-Tools diejenigen auszuwählen, welche der weiteren Arbeit als Grundlage im Sinne der Aufgabenstellung dienen sollen. Zunächst soll daher ein möglichst vollständiger und objektiver Überblick über den Gesamtmarkt gegenwärtiger Requirements-Engineering-Tools aus Praxis und Forschung gegeben werden. Der eingehenden Recherche folgt ein Auswahlprozess, an dessen Ende drei Tools übrig bleiben sollen[1]. Von diesen Tools sollen zwei kommerzieller Art sein, während das dritte nicht kommerziell verfügbar sein soll, um einen möglichst breiten Überblick zu bieten. Die Ergebnisse der Recherche werden in einer Microsoft (MS) Excel-Tabelle dokumentiert.

3.1 Kommerzielle Tools

Zunächst wird eine Liste der kommerziell erhältlichen Requirements-Engineering-Tools aufgestellt, wie sie in der Praxis in verschiedenem Umfang zum Einsatz kommen. Angelehnt an den von Versteegen beschriebenen Prozess für die „Auswahl eines Werkzeugs für das Anforderungsmanagement“ wird in einem ersten Schritt eine *Longlist* gebildet [vgl. Ver04, S. 83]. Zu jedem Tool werden folgende Informationen zusammengetragen:

- Allgemeine Produktinformationen (z.B. der Name des Produktes und der Hersteller)
- Verfügbare Lizenzmodelle (Evaluierungslizenzen vorhanden?)
- Technische Informationen (z.B. Systemvoraussetzungen des Tools)
- Anwendungsbereich (System-Engineering, Requirements-Engineering, Requirements-Management)

Sämtliche Informationen werden aus im Internet verfügbaren Quellen bezogen. In diesem Zusammenhang sind vor allem die von Suzanne Robertson und James Robertson

[1] Um dem Zeitrahmen der Arbeit gerecht zu werden, wurde sich auf eine Anzahl von drei Tools geeinigt.

durchgeführte Erhebung von Requirements-Engineering-Tools [vgl. RR06], Ian Alexanders Übersicht verschiedener Vertreiber [vgl. Ale06] und der *INCOSE Requirements Management Tools Survey* [vgl. INC06] zu nennen. Die dort vorgefundenen Angaben werden durch Herstellerinformationen ergänzt und den Bedürfnissen der vorliegenden Arbeit entsprechend aufbereitet. Dadurch ergibt sich zunächst ein zum Teil heterogenes Informationsbild, da sowohl Quantität als auch Qualität der Angaben zu dem jeweiligen Produkt von Hersteller zu Hersteller stark variieren.

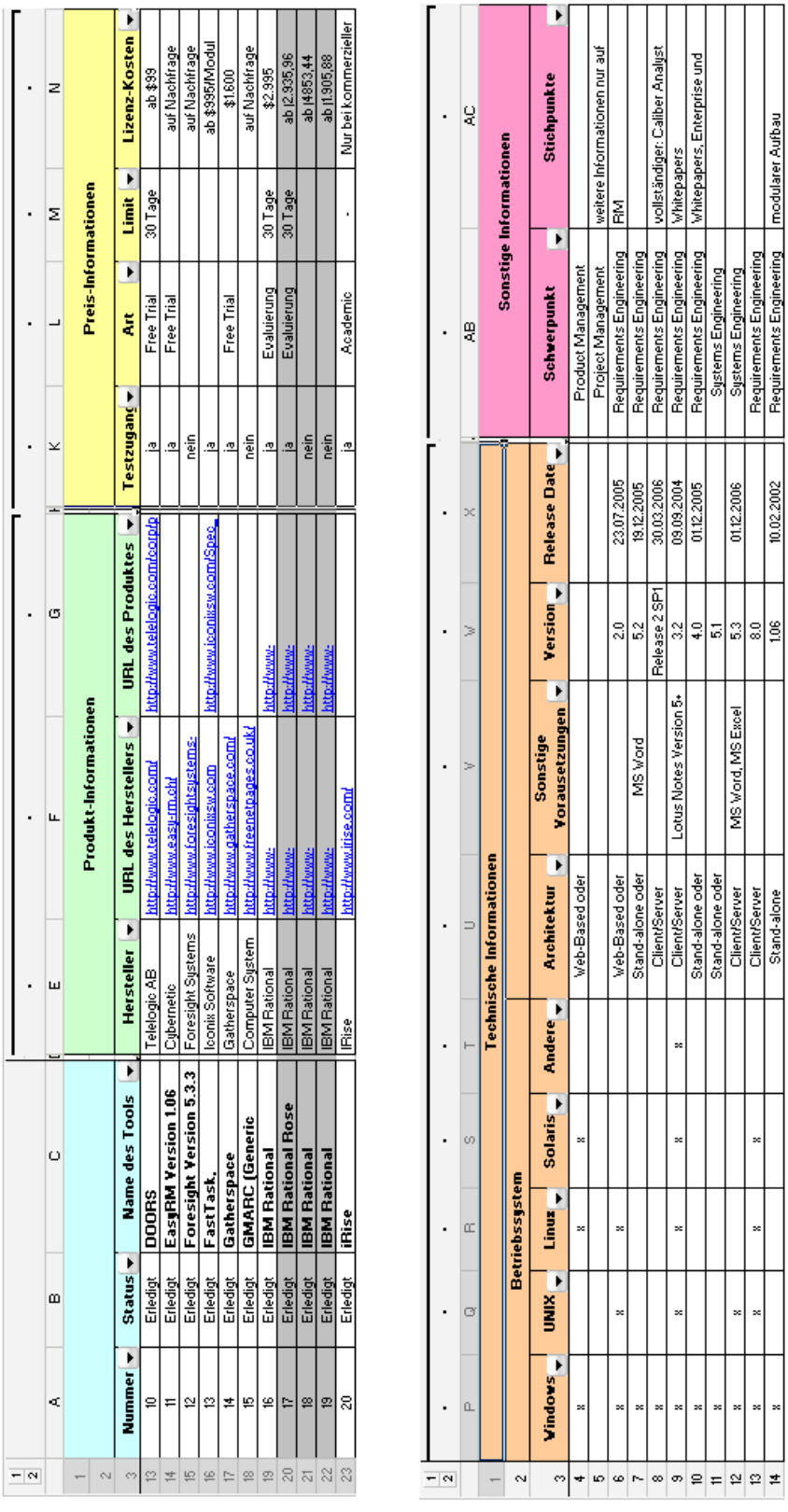
Es ergibt sich eine Liste von insgesamt 46 Tools. Zur Verdeutlichung geben die nachstehenden Abbildungen einen - rein exemplarischen - Auszug der Longlist wieder. Die komplette Liste mit sämtlichen Informationen ist im Anhang der Arbeit zu finden.

Im nächsten Schritt wird diese Zusammenstellung in eine so genannte *Shortlist* überführt [vgl. Ver04, S. 85]. Mittels geeigneter Auswahlkriterien wird die Anzahl der in Frage kommenden Tools so auf eine überschaubare Menge reduziert, laut Versteegen sollen dadurch mindestens 50% der Tools herausfallen. Die Auswahlkriterien werden entsprechend der Zielsetzungen der vorliegenden Arbeit sowie den gegebenen Rahmenbedingungen (z.B. den zur Verfügung stehenden Ressourcen) gewählt.

Die Selektion findet zunächst über die beiden Punkte Schwerpunkt und Testzugang statt. Den Absichten der Arbeit gemäß werden diejenigen Tools in die engere Auswahl gezogen, deren Anwendungsschwerpunkt das Requirements-Engineering ist. Der Einfachheit halber werden zu diesem Zeitpunkt unter dem Begriff Requirements-Engineering verschiedene Teilaspekte, etwa das Requirements-Management, aggregiert. Später werden die Funktionalitäten der verbleibenden Tools näher betrachtet und die Auswahl entsprechend verfeinert.

Produkte, deren Anwendungsschwerpunkt nicht im Requirements-Engineering liegt, fallen an dieser Stelle aus der Auswahl heraus. Im Laufe der Erhebung konnten fünf verschiedene Kategorien von Tools unterschieden werden, nach denen sich diese gruppieren ließen:

1. „Echte“ Requirements-Engineering-Tools - Erfassen, Editieren, Verfolgen von Anforderungen
2. Abhängiger Bestandteil einer kompletten Entwicklungsumgebung
3. Unabhängiger Bestandteil einer kompletten Entwicklungsumgebung
4. Vermittler zwischen einem Editor zum Erfassen und Ändern und einer Datenbank zum Speichern und Abfragen
5. Ergänzung zu Requirements-Engineering-Tools (z.B. Prototyping)



Nummer	Status	Name des Tools	Produkt-Informationen: Hersteller	URL des Herstellers	URL des Produktes	Preis-Informationen: Testzugang	Art	Limit	Lizenz-Kosten
10	Erledigt	DOORS	Telelogic AB	http://www.telelogic.com/	http://www.telelogic.com/corp/p	ja	Free Trial	30 Tage	ab $99
11	Erledigt	EasyRM Version 1.06	Cybernetic	http://www.easy-rm.ch/		ja	Free Trial		auf Nachfrage
12	Erledigt	Foresight Version 5.3.3	Foresight Systems	http://www.foresightsystems.-		nein			auf Nachfrage
13	Erledigt	FastTask.	Iconix Software	http://www.iconixsw.com	http://www.iconixsw.com/Spec_	ja			ab $995/Modul
14	Erledigt	Gatherspace	Gatherspace	http://www.gatherspace.com/		ja	Free Trial		$1.600
15	Erledigt	GMARC (Generic	Computer Systm	http://www.freenetpages.co.uk/		nein			auf Nachfrage
16	Erledigt	IBM Rational	IBM Rational	http://www.-	http://www.-	ja	Evaluierung	30 Tage	$2.395
17	Erledigt	IBM Rational Rose	IBM Rational	http://www.-	http://www.-	ja	Evaluierung	30 Tage	ab i2.935,96
18	Erledigt	IBM Rational	IBM Rational	http://www.-	http://www.-	nein			ab i4853,44
19	Erledigt	IBM Rational	IBM Rational	http://www.-	http://www.-	nein			ab i1.905,88
20	Erledigt	iRise	iRise	http://www.irise.com/		ja	Academic	-	Nur bei kommerzieller

Technische Informationen: Betriebssystem: Windows	UNIX	Linux	Solaris	Andere	Architektur	Sonstige Voraussetzungen	Version	Release Date	Sonstige Informationen: Schwerpunkt	Stichpunkte
x		x	x		Web-Based oder				Product Management	
									Project Management	weitere Informationen nur auf
x	x	x			Web-Based oder		2.0	23.07.2005	Requirements Engineering	RM
x					Stand-alone oder	MS Word	5.2	19.12.2005	Requirements Engineering	
x					Client/Server		Release 2 SP1	30.03.2006	Requirements Engineering	vollständiger: Caliber Analyst
x	x	x	x	x	Client/Server	Lotus Notes Version 5+	3.2	09.09.2004	Requirements Engineering	Whitepapers
x					Stand-alone oder		4.0	01.12.2005	Requirements Engineering	Whitepapers, Enterprise und
x					Stand-alone oder		5.1		Systems Engineering	
x	x				Client/Server	MS Word, MS Excel	5.3	01.12.2006	Systems Engineering	
x	x	x	x		Client/Server		8.0		Requirements Engineering	
x					Stand-alone		1.06	10.02.2002	Requirements Engineering	modularer Aufbau

Abbildung 3.1: Auszug der Tool-Auswahl

Für die weitere Untersuchung kamen nur Tools der ersten und dritten Kategorie in Frage. Die folgende Tabelle nennt einige Beispiele für Tools, die diese Bedingung nicht erfüllen konnten:

Tabelle 3.1: Tools ohne Requirements-Engineering Schwerpunkt

Produkt	Hersteller	Schwerpunkt
Accept 360	Accept Software Corporation	Produktmanagement
METIS Version 3.4	Computas AS	Modellierung
TcSE (Teamcenter Systems Engineering) Version 7.0	UGS Corp.	Systems Engineering

Als zweites Kriterium wird für die verbleibenden Tools das Vorhandensein einer kostenlosen Evaluierungslizenz gewählt. Diese variiert meist zwischen einer zeitlich begrenzten Lizenz zur Evaluierung der Software oder einer unbeschränkten Lizenz für die akademische Nutzung des Tools. Zunächst wird aber lediglich die Existenz einer Evaluierungslizenz an sich überprüft. All diejenigen Tools, die keine derartige Testmöglichkeit zur Verfügung stellen bzw. deren Hersteller dazu keine Angaben machen, werden an dieser Stelle aus der weiteren Untersuchung ausgeschlossen. Die folgende Tabelle nennt einige Beispiele für solche Tools:

Tabelle 3.2: Tools ohne Evaluierungslizenz

Produkt	Hersteller	Schwerpunkt
GMARC	Computer System Architects Ltd.	Requirements-Engineering
Requirements Assistant	Sunny Hills Consultancy BV	Requirements-Engineering
VeroTrace	Verocel	Requirements-Engineering

Nach Anwendung dieser Auswahlkriterien auf die Longlist verbleiben die nachfolgenden 18 Tools, die in der bereits erwähnten Shortlist zusammengefasst werden. Sie erfüllen die beiden genannten Kriterien und können somit für eine weitere Untersuchung in Betracht gezogen werden.

Tabelle 3.3: Shortlist der Tools

Produkt	Hersteller	Schwerpunkt
Agility	Agile Edge	Requirements-Engineering
Analyst Pro	Goda Software, Inc.	Requirements-Engineering
CaliberRM	Borland Software Corporation	Requirements-Engineering
CARE	SOPHIST GROUP	Requirements-Engineering
Catalyze	SteelTrace	Requirements-Engineering
DOORS	Telelogic AB	Requirements-Engineering
EasyRM	Cybernetic Intelligence GmbH	Requirements-Engineering
Gatherspace	Gatherspace	Requirements-Engineering
IBM Rational Requisite-Pro	IBM Rational Software	Requirements-Engineering
iRise	iRise	Requirements-Engineering
IRqA	TCP Sistemas e Ingenieria	Requirements-Engineering
MockupScreens	MockupScreens	-
Objectiver	Respect-IT	Requirements-Engineering
QFD/Capture	International TechneGroup Incorporated	Requirements-Engineering
RMTrak	RBC Product Development	Requirements-Engineering
StP (Software through Pictures)	Aonix	Requirements-Engineering
Truereq PLM Software	Truereq Inc.	Requirements-Engineering
XTie-RT	Teledyne Brown Engineering, Inc.	Requirements-Engineering

Da die Arbeit einen möglichst repräsentativen Charakter haben und das zu entwickelnde Referenzmodell einen Vergleichsmaßstab für gegenwärtige wie kommende Tools setzen soll, bedarf es einer geeigneten Eigenschaft, anhand derer die Verbreitung und Nutzung

der zu untersuchenden Tools in Wirtschaft und Forschung gemessen werden kann. Hier bietet sich der Marktanteil des jeweiligen Tools als Kriterium an. Dieser vermittelt ein Bild von der Position eines Produktes in seinem Marktsegment und kann somit als Indikator für die Verbreitung dieses Produktes gewertet werden. Die Ermittlung des Marktanteils bedarf einer soliden Informationsgrundlage.

Diese zu schaffen erweist sich allerdings als schwierig. Es lassen sich kaum vergleichbare (und vor allem frei verfügbare) Informationen für das gesamte Marktsegment finden. Mit Sicherheit lässt sich aber sagen, dass das Tool *DOORS* der Firma *Telelogic* mit 41% Marktanteil (gemessen an den Einnahmen aus Lizenzverkäufen) Marktführer auf dem Gebiet des Requirements-Engineering ist. Dies geht hervor aus dem *Gartner DataRequest 2004* [vgl. Tel06a]. Die stärksten Wettbewerber sind demnach die Produkte der *IBM Rational Suite* (39%) sowie Borlands Caliber RM (4%). Die verbleibenden 16% verteilen sich auf andere Unternehmen. Weitere Analysten wie *Ovum* oder *IDC* bestätigen die vorherrschende Position von Telelogic DOORS auf dem Markt [vgl. Müh06, S. 1]. Laut *Yphise*, einer führenden unabhängigen Analystenfirma, ist das Tool zudem die technisch durchdachteste Anforderungsmanagementsoftware des Jahres 2005 [vgl. Yph05]. Das Unternehmen untersuchte insgesamt fünf etablierte Tools (Borland Caliber RM, Telelogic DOORS, Compuware Reconcile, IBM RequisitePro, Serena RTM) gemäß des *ISO 9001* zertifizierten Beurteilungsprozesses.

Wendet man diese Informationen auf die Shortlist an, verbleiben als Endergebnis der Erhebung nun folgende Tools:

Tabelle 3.4: Endergebnis kommerzielle Tools

Produkt	Hersteller	Schwerpunkt
CaliberRM	Borland Software Corporation	Requirements-Engineering
DOORS	Telelogic AB	Requirements-Engineering
IBM Rational RequisitePro	IBM Rational Software	Requirements-Engineering

Da am Ende des Auswahlprozesses genau drei Tools übrig bleiben sollen, von denen allerdings nur zwei Tools kommerzieller Natur sein dürfen, ergeben sich am Ende dieser Phase der Marktführer *DOORS* von *Telelogic AB* und der direkte Konkurrent *RequisitePro* von *IBM* als erstes Ergebnis der Untersuchung.

3.2 Nicht-kommerzielle Tools

Um auch aktuelle Entwicklungen im Bereich der Requirements-Engineering-Tools zu berücksichtigen, wird außerdem versucht, auch neue Ansätze aus der Forschung zu untersuchen. Als Informationsquelle dienen vor allem verschiedene Veröffentlichungen des *IEEE (Institute of Electrical and Electronics Engineers)* sowie Artikel des Requirements-Engineering Magazins des *Springer Verlags*. Die vorgefundenen Tools sind jedoch meist auf bestimmte Themen oder Methoden des Requirements-Engineering spezialisiert (z.B. Security-Requirements-Engineering oder mobile Requirements-Engineering-Tools mit einem Fokus auf die Koordination der Anforderungsverwaltung in verteilten Teams), so dass sich diese für eine weitere Untersuchung im Rahmen dieser Arbeit nicht eignen. Der Vollständigkeit halber sollen an dieser Stelle aber auch sie beispielhaft vorgestellt werden.

READS. Hierbei handelt es sich um ein selbständiges Requirements-Engineering-Tool, welches aber ursprünglich Teil einer kompletten System-Engineering-Umgebung werden sollte. Entwickelt wurde READS 1990 von der *Paramax Systems Corporation*, die heute zu *Lockheed Martin* (früher *Unisys*) gehört und traditionell im militärischen Bereich angesiedelt ist. READS soll schwerpunktmäßig Schlüsselaspekte des Requirements-Engineering wie die Erfassung, Analyse, Strukturierung, Wiederverfolgbarkeit und Reporting unterstützen. Es bietet Schnittstellen zu anderen Tools, insbesondere zu *Software through Pictures (StP)*. Durch dieses Interface soll vor allem die Wiederverfolgbarkeit von Anforderungen gewährleistet werden, indem die verschiedenen Artefakte der beiden Tools (READS: Requirements Model; StP: Capability Model) miteinander verknüpft werden [vgl. Smi93].

SMaRT (Scenario Management and Requirements Tool). Bei diesem Tool handelt es sich um ein auf die Szenario-Methode des Requirements-Engineering spezialisiertes Tool. Es bietet eine web-basierte Benutzerschnittstelle, über die der Anwender Szenarien und dazugehörige Anforderungen eingeben, managen, sichten, analysieren und bearbeiten kann [vgl. SA03].

RETH (Requirements-Engineering Through Hypertext). Dieses nicht käuflich zu erwerbende Tool wurde von Prof. Dr. Hermann Kaindl für die *Siemens AG Österreich* entwickelt. Es basiert auf der ebenfalls RETH benannten Methode sowie der Szenario-Methode. Schwerpunkte sind die Erfassung und Verwaltung von Zielen, Szenarien und Anforderungen, welche alle innerhalb des Tools dokumentiert und durch Hyperlinks miteinander verbunden werden. Das gesamte Modell kann anschließend z.B. als HTML-Dokument exportiert werden [vgl. Kai04].

Mit der zunehmenden Leistungsfähigkeit (Rechenkraft, Speicherkapazität, Datenübertragung) mobiler Endgeräte finden auch entsprechende Anwendungen im Requirements-Engineering zunehmend Beachtung in der Forschung. Dabei konzentrieren sich die Anstrengungen vor allem auf die (gemeinschaftliche) Erfassung von Anforderungen und die

Unterstützung der Kommunikation in verteilten Teams. Durch die Möglichkeit, diese Tätigkeiten direkt vor Ort durchführen zu können, verspricht man sich bessere und vollständigere Anforderungen, da die Analysten durch die direkte Beobachtungssituation ein besseres Verständnis für die Arbeitsumgebung der Anwender entwickeln können. Diese mobilen Tools stellen keinen Ersatz, sondern vielmehr eine Ergänzung zu konventionellen Requirements-Engineering-Tools dar.

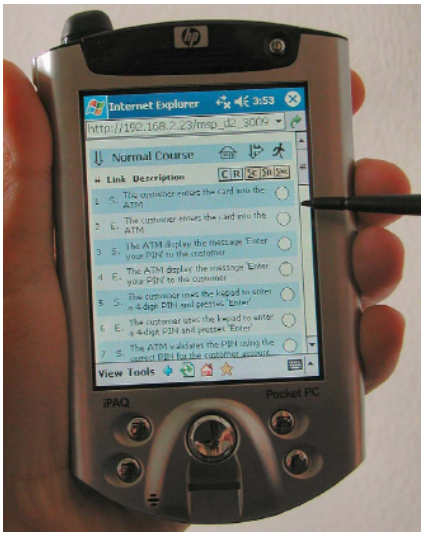


Abbildung 3.2: Mobiles Requirements-Engineering mit Hilfe des Mobile Scenario Presenter [vgl. Sey04, S. 428]

ARENA (Anytime, Anywhere REquirements Negotiation Aids). Dieses web-basierte Tool basiert auf der EasyWinWin-Methode[2] [vgl. Sey04, S. 427]. Das Konzept ist gedacht für eine Reihe unabhängiger, erfolgskritischer Stakeholder. Diesen soll es mit Hilfe des Tools ermöglicht werden, Schritt für Schritt Anforderungen zu sammeln, sie auszuarbeiten und zu priorisieren. Dabei werden folgende Aktivitäten durchgeführt [vgl. Sey04, S. 427]:

- Review and expand negotiation topics
- Brainstorm stakeholder interests
- Converge on Win Conditions
- Capture a glossary of Terms
- Prioritize Win Conditions
- Reveal Issues and Constraints
- Identify Issues, Options and Agreements

ART-SCENE (Analyzing Requirements Trade-offs: Scenario Evaluations). Auch dieses Tool ist web-basiert. Das zugrunde liegende Konzept ist in diesem Fall die Szenario-Methode. Das Tool unterstützt die Bildung von Szenarien aus Use-Case-Spezifikationen. Während die Stakeholder die verschiedenen Szenarien durchlaufen, können sie so Ereignisse und Anforderungen hinzufügen, bearbeiten oder entfernen [vgl. Sey04, S. 427].

MSP (Mobile Scenario Presenter). Dieses Tool soll Analysten dabei unterstützen, Anforderungen mit Hilfe strukturierter Szenarien direkt in der Arbeitsumgebung der Stakeholder zu erheben [vgl. Sey04, S. 428]. Auch in diesem Fall ist die Idee, dass so eine vollständige Sammlung von Anforderungen gefördert wird. Eine Besonderheit dieses Tools ist, dass Anforderungen um Zeichnungen und Audio-Aufnahmen ergänzt werden können. Das Tool ermöglicht darüber hinaus nicht nur die Erfassung neuer Anforderung, sondern auch die Validierung bereits bestehender Anforderungen durch Diskussion mit den Stakeholdern vor Ort.

Als Beispiel für ein nicht-kommerzielles Tool fällt die Wahl schließlich auf RETH. Auf Grund des von RETH verfolgten Ansatzes und einer Reihe von erfolgreichen Einsätzen in der Praxis [vgl. Kai97, S. 320] eignet es sich besonders für eine weitere Untersuchung im Sinne der Aufgabenstellung.

[2] „EasyWinWin is a requirements definition methodology that builds on the win-win negotiation approach and leverages collaborative technology to improve the involvement and interaction of key stakeholders. With EasyWinWin, stakeholders move through a step-by-step win-win negotiation where they collect, elaborate, and prioritize their requirements, and surface and resolve issues to come up with mutually satisfactory agreements [vgl. Cen02].“

Damit ergeben sich als Endergebnis des gesamten Auswahlprozesses die folgenden Tools, welche die Basis für die weitere Arbeit bilden werden:

Tabelle 3.5: Endergebnis des Auswahlprozesses

Produkt	**Hersteller**
DOORS	Telelogic AB
RequisitePro	IBM Rational Software
RETH	Siemens AG Österreich

4 Ablauf der Untersuchung

In diesem Kapitel soll die Vorgehensweise bei der Untersuchung der Tools vorgestellt werden. Im weiteren Verlauf der Arbeit werden die zuvor ausgewählten Tools detailliert präsentiert. Zu jedem Tool werden zunächst einige Hintergrundinformationen (z.B. Herstellerangaben) vorgestellt. Anschließend werden die wichtigsten Konzepte der Arbeitsumgebung des Tools erläutert, um einen gemeinsamen Wortschatz für die weiteren Abschnitte aufzubauen.

Eine Untersuchung der Informationsstruktur des Tools soll Aufschluß darüber geben, wie das Repository des Tools aufgebaut ist, und welche Optionen zur Strukturierung der Informationen dem Anwender zur Verfügung gestellt werden. Ferner interessiert, mit welchen Eigenschaften und Attributen die Datenobjekte versehen werden.

Schließlich soll untersucht werden, ob und in welchem Umfang die ausgewählten Tools die folgenden Funktionalitäten unterstützen:

- Traceability
- Versionsmanagement
- Veränderungsmanagement
- Möglichkeiten zur Analyse der Anforderungen
- Administrative Aspekte wie die Benutzerverwaltung
- Customizing

4.1 Das Bibliothekssystem als Beispiel

Um ein einheitliches Vorgehen bei der Analyse der Tools zu gewährleisten, wird ein Beispiel konstruiert, welches der weiteren Untersuchung als Grundlage dient. Dieses Beispiel basiert auf dem Buchbeispiel, wie es bei [Rup04] zu finden ist. Es handelt sich dabei um die Konstruktion eines Bibliothekssystems. Die dort auf den Seiten 37f. beschriebenen Regeln des Verwaltungsapparates einer fiktiven Stadtbücherei werden in Anforderungen

übersetzt und um einige weitere ergänzt, um eine stimmige Sammlung von Anforderungen zu erhalten. Zwar ergibt sich aus diesen Anforderungen keine vollständige Betrachtung eines Bibliothekssystems, der betrachtete Ausschnitt ist jedoch umfassend genug gewählt, um die eingangs erwähnten Funktionalitäten und Konzepte zu untersuchen, und deren Umsetzung in den entsprechenden Tools zu demonstrieren.

Mit Hilfe dieses Beispiels wird wie folgt verfahren:

Zunächst werden Teile der Anforderungen aus verschiedenen Anwendungen importiert, um die entsprechende Funktionalität des untersuchten Tools zu testen.
Beispiel: Import der Dateien „system_requirements.doc“ und „test_module.txt“

Die verbleibenden Anforderungen werden manuell eingegeben. Dabei soll vor allem versucht werden, mit den gegebenen Mitteln der Tools eine geeignete Struktur aufzubauen.
Beispiel: Manuelle Eingabe der Anforderung UR0001 (s.u.)

Anschließend sollen bestimmte Anforderungen des Beispiels miteinander verknüpft werden. Einige dieser Anforderungen liegen innerhalb eines einzigen Anforderungsdokumentes, andere wiederum sind über mehrere Anforderungsdokumente verteilt.
Beispiel: Verknüpfung der Anforderung UR0019 mit der Anforderung SR0001 (s.u.)

Nachdem die Anforderungen miteinander verknüpft wurden, sollen einige Änderungen an verschiedenen Anforderungen vorgenommen werden, um die Auswirkungen auf die History zu untersuchen.
Beispiel: Änderung der Anforderung SR0010 (Das Bibliothekssystem soll Daten von jedem System einer anderen Zweigstelle innerhalb von 35 statt 30 Sekunden empfangen und verarbeiten können, s.u.)

Daraufhin werden die Analysemöglichkeiten des Tools untersucht und auf das Beispiel angewendet. Schließlich wird das Projekt in seinem aktuellen Zustand festgehalten, was dem Vorgehen beim Erreichen eines Meilensteins in einem Projekt entspricht.

In den folgenden Abschnitten wird das Anwendungsbeispiel vorgestellt. Dabei entsprechen die Abschnitte *User Requirements*, *System Requirements* und *Abnahmekriterien* jeweils einem Anforderungsdokument.

Bezüge auf das Beispiel sind in den folgenden Kapiteln durch eine Einfassung gekennzeichnet.

Die in den einzelnen Dokumenten enthaltenen Anforderungen werden hier zur besseren Lesbarkeit und Referenzierung durchgehend nummeriert. Der Index ergibt sich aus einer Abkürzung des Dokumentennamens (zwei Stellen) und einer fortlaufenden Zahl. Beim Import in die verschiedenen Tools werden diese Nummerierungen entfernt, um zu prüfen,

inwieweit eine entsprechende Indizierung der Anforderungen automatisch erfolgt. Daher können sich die Indizes der folgenden Anforderungen von denen in Abbildungen aus den Tools unterscheiden.

4.2 User Requirements

In diesem Abschnitt werden Anforderungen beschrieben, wie sie von der Anwenderseite, also z.B. Verwaltungsmitarbeitern oder Bibliothekaren, an ein Bibliothekssystem gestellt werden könnten. Die Anforderungen basieren auf dem auf den Seiten 37f. in [Rup04] beschriebenen Verwaltungsapparat einer Bibliothek. Der gewählte Ausschnitt betrachtet verschiedene Anforderungen an die Kundenverwaltung, die Bestandsverwaltung sowie an die Ausleihe und die Rückgabe von Leihobjekten. Der im Beispiel durchgehend genutzte Begriff *Leihobjekte* steht stellvertretend für Bücher der Bibliothek, könnte aber auch Videos, Zeitschriften und weitere Medien umfassen.

4.2.1 Kundenverwaltung

Dieser Abschnitt enthält einige Anforderungen, wie sie Benutzer des Bibliothekssystems (z.B. Angestellte oder Kunden) an die Benutzerverwaltung des Systems stellen könnten. Es werden Anforderungen an die Anmeldung an das System, and den Bibliotheksausweis und an die Verwaltung der Leihobjekte entworfen.

Anmeldung zur Nutzung der Bibliothek

`UR0001` Der Bibliothekskunde muss bei seiner Anmeldung zur Nutzung der Bibliothek ein Anmeldeformular ausfüllen. Das Anmeldeformular muss folgende Daten des Bibliothekskunden erfassen:

- Name
- Anschrift
- Telefonnummer
- E-Mail-Adresse
- Personalausweisnummer

`UR0002` Die Anmeldedaten des Bibliothekskunden werden durch explizite Bestätigung des Bibliothekskunden in der zentralen Datenbank des Bibliothekssystems gespeichert.

UR0003 Das Bibliothekssystem muss nach Bestätigung der Anmeldedaten durch den Bibliothekskunden automatisch eine eindeutige Kundenummer für den Bibliothekskunden erstellen.

Bibliotheksausweis

UR0004 Das Bibliothekssystem muss auf explizite Anweisung des Bibliothekars einen Bibliotheksausweis für den Bibliothekskunden erstellen.

UR0005 Der Bibliotheksausweis muss den Namen und die Kundennummer des Besitzers sowie das Ausstellungsdatum des Bibliotheksausweises anzeigen.

UR0006 Jeder Bibliothekskunde muss einen gültigen Bibliotheksausweis besitzen.

UR0007 Der Bibliotheksausweis ist ein Jahr lang gültig, geltend ab dem Ausstellungsdatum.

4.2.2 Bestandsverwaltung

UR0008 Der Bibliothekar muss zu jedem neuen Leihobjekt folgende Daten erfassen:

- Titel
- Autoren
- Auflage
- Erscheinungsjahr
- Verlag

UR0009 Das Bibliothekssystem muss zu jedem neuen Leihobjekt die manuell eingegebenen Daten des Bibliothekars sowie zusätzlich die folgenden Informationen in der zentralen Datenbank des Bibliothekssystems erfassen:

- Eine eindeutig zuordenbare Objektnummer
- Einen Ausleihestatus („Im Haus“/„Außer Haus“)
- Einen Rückgabetermin

UR0010 Der Datensatz eines jeden Leihobjektes muss vollständig sein.

4.2.3 Ausleihe

In diesem Abschnitt werden Anforderungen beschrieben, wie sie Benutzer des Bibliothekssystems (z.B. Angestellte oder Kunden) an den Leihvorgang des Systems stellen könnten. Darunter fallen u.a. die Berechtigung zur Ausleihe von Leihobjekten, deren Leihfrist sowie die Möglichkeit der Fernleihe von Leihobjekten.

Berechtigung

`UR0011` Leihobjekte der Bibliothek dürfen nur durch angemeldete Benutzer mit gültigem Bibliotheksausweis der Bibliothek entliehen werden.

Leihfrist

`UR0012` Die Leihfrist eines Leihobjektes beträgt vier Wochen, geltend ab dem Ausleihedatum des Leihobjektes.

`UR0013` Der Bibliothekskunde kann die Leihfrist eines Leihobjektes einmalig um vier Wochen verlängern.

`UR0014` Das Bibliothekssystem muss den aktuellen Rückgabetermin für das entliehene Leihobjekt in der zentralen Datenbank des Systems eintragen.

`UR0015` Das Bibliothekssystem muss einmal pro Woche die Rückgabetermine aller als „Außer Haus“ gekennzeichneten Leihobjekte überprüfen.

`UR0016` Das Bibliothekssystem muss eine Mahnung per E-Mail an den Bibliothekskunden senden, wenn die Leihfrist durch diesen überschritten wurde.

`UR0017` Das Bibliothekssystem muss dem Bibliothekskunden für jedes säumige Leihobjekt eine Gebühr von 2,50 € pro angefangene Woche verrechnen.

Fernleihe

`UR0018` Der Bibliothekskunde soll Leihobjekte von anderen Zweigstellen entleihen können.

4.2.4 Rückgabe

Dieser Abschnitt führt Anforderungen auf, wie sie Benutzer des Bibliothekssystems (z.B. Angestellte oder Kunden) an die Rückgabe von Leihobjekten mit Hilfe des Systems

stellen könnten. Dazu zählen z.B. die Behebung von eventuell aufgetretenen Schäden an Leihobjekten.

Schadensbehebung

`UR0019` Der Bibliothekar muss zurückgegebene Leihobjekte auf neue Mängel hin untersuchen.

`UR0020` Ein beschädigtes Leihobjekt muss wieder aufbereitet werden, die Kosten für die Instandsetzung trägt der schuldige Bibliothekskunde.

`UR0021` Ein Leihobjekt muss innerhalb seiner Leihfrist an die Bibliothek zurückgegeben werden.

`UR0022` Das Bibliothekssystem muss jedes zurückgegebene Leihobjekt als „Im Haus“ in der zentralen Datenbank kennzeichnen.

`UR0023` Das Bibliothekssystem muss den Rückgabetermin für das entliehene Leihobjekt aus der zentralen Datenbank des Systems löschen.

`UR0024` Der Bibliothekskunde muss bei der Rückgabe fällige Gebühren zahlen.

4.3 System Requirements

In diesem Abschnitt werden Anforderungen an das Bibliotheksystem formuliert, die sich zum Teil aus den zuvor beschriebenen Anwenderforderungen ergeben. Es handelt sich hierbei bereits um eher technisch orientierte Anforderungen.

Auch die Anforderungen dieses Abschnitts nehmen teilweise Bezug auf das Beispiel aus [Rup04].

4.3.1 Funktionale Anforderungen

In diesem Abschnitt werden einige funktionale Anforderungen an das Bibliothekssystem beschrieben. Unter funktionalen Anforderungen werden in diesem Falle Aktionen verstanden, die das System selbständig ausführen soll.

`SR0001` Das Bibliothekssystem soll Daten von jedem System einer anderen Zweigstelle empfangen und verarbeiten können.

SR0002 Das Bibliothekssystem muss sicherstellen, dass jedes Leihobjekt nur genau einmal im Bestand der Bibliothek vorkommt.

SR0003 Das Bibliothekssystem muss das Rückgabedatum des Leihobjektes in der zentralen Datenbank des Bibliothekssystems speichern.

SR0004 Das Bibliothekssystem muss die Leihfrist des Objektes um vier Wochen verlängern und das Rückgabedatum entsprechend aktualisieren.

4.3.2 Technische Anforderungen

In diesem Abschnitt werden einige technische Anforderungen an das Bibliothekssystem aufgeführt. Diese beschreiben beispielsweise Hardwareanforderungen oder Architekturanforderungen an das System.

SR0005 Das Bibliothekssystem muss unter Windows (64 Bit) laufen.

SR0006 Das Bibliothekssystem soll unter folgenden Betriebssystemen ebenfalls laufen:

- Windows (32 Bit)
- Linux

SR0007 Die Benutzeroberfläche des Bibliothekssystems muss im Internet Explorer fehlerfrei dargestellt werden.

SR0008 Die Benutzoberfläche des Bibliothekssystems soll in folgenden Browsern ebenfalls fehlerfrei dargestellt werden:

- Netscape
- Mozilla
- Mozilla Firefox
- Opera

4.3.3 Qualitätsanforderungen

In diesem Abschnitt werden Qualitätsanforderungen an das Bibliothekssystem beschrieben. Diese betreffen die Qualität des Systems und beziehen sich auf Merkmale wie Zuverlässigkeit, Effizienz oder Benutzbarkeit.

SR0009 Das Bibliothekssystem soll dem Bibliothekskunden 99% der regulären Öffnungszeit zur Verfügung stehen.

`SR0010` Das Bibliothekssystem soll Daten von jedem System einer anderen Zweigstelle innerhalb von 30 Sekunden empfangen und verarbeiten können.

`SR0011` Die Sicherheit der Datenübertragung bei der Kommunikation mit einer anderen Zweigstelle soll durch SSL gewährleistet werden.

4.4 Abnahmekriterien

In diesem Teil werden Abnahmekriterien für die zuvor beschriebenen Anforderungen formuliert. Diese dienen der Überprüfung des Bibliothekssystems vor Inbetriebnahme. Damit soll sichergestellt werden, dass das System die gestellten Anforderungen erfüllt.

> „Ein Abnahmekriterium ist eine Anweisung für den Test bezüglich einer Anforderung (oder eines Anforderungsteils), die die Prüfung und Bewertung des erstellten Produktes oder durchgeführten Prozesses gegenüber dieser Anforderung (oder des Teils) beschreibt“ [vgl. Rup04, S. 308].

Auch die Abnahmekriterien sind teilweise ähnlich in [Rup04] wiederzufinden. Sie werden wie folgt formuliert:

Ausgangssituation: Die Ausgangsituation beschreibt, welchen Zustand der Prüfling besitzen muss, damit das Abnahmekriterium durchgeführt werden kann.
Ereignis: Das Ereignis des Tests (englisch: *event* oder *trigger*) veranlasst den Prüfling, sich der Anforderung entsprechend zu verhalten.
Erwartetes Ergebnis: Das erwartete Ergebnis beschreibt das Soll-Verhalten des Prüflings bei einem der Anforderung entsprechenden korrekten Verhalten.

4.4.1 Benutzerverwaltung

`AK0001` Fehlende Daten bei der Anmeldung eines neuen Bibliothekskunden
Ausgangsituation: Der Bibliothekskunde möchte sich zur Nutzung der Bibliothek anmelden.
Ereignis: Der Bibliothekskunde füllt das Anmeldeformular aus, lässt aber das Feld „Telefonnummer“ frei.
Erwartetes Ereignis: Das Bibliothekssystem hat bei der Bestätigung der Anmeldedaten durch den Bibliothekskunden eine Warnung mit einem Hinweis auf die fehlende Telefonnummer auf dem Bildschirm auszugeben. Der Anmeldevorgang beginnt erneut.

4.4.2 Bestandsverwaltung

AK0002 Fehlende Daten bei der Aufnahme eines neuen Leihobjektes in den Bestand der Bibliothek
Ausgangsituation: Ein neues Leihobjekt wird in den Bestand aufgenommen.
Ereignis: Der Bibliothekar gibt die Daten für das Leihobjekt ein, lässt aber das Feld „Titel“ frei.
Erwartetes Ereignis: Das Bibliothekssystem hat bei der Bestätigung der Daten durch den Bibliothekar eine Warnung mit einem Hinweis auf den fehlenden Titel auszugeben. Der Bibliothekar wird zu Eingabe der fehlenden Daten aufgefordert.

4.4.3 Ausleihe

AK0003 Performance des Bibliothekssystems bei der Kommunikation mit einer anderen Zweigstelle
Ausgangsituation: Der Bibliothekskunde ist im Bibliothekssystem angemeldet.
Ereignis: Der Bibliothekskunde fragt den Bestand einer anderen Zweigstelle der Bibliothek ab.
Erwartetes Ereignis: Das Bibliothekssystem hat die Anfrage des Bibliothekskunden im Durchschnitt innerhalb von 30 Sekunden zu beantworten.

AK0004 Verhalten des Bibliothekssystem bei der Überprüfung eines Bibliotheksausweises auf Gültigkeit
Ausgangssituation: Der Bibliothekskunde möchte ein Leihobjekt entleihen.
Ereignis: Das Bibliothekssystem überprüft den Bibliotheksausweis des Bibliothekskunden auf Gültigkeit.
Erwartetes Ereignis:

- Ist der Bibliotheksausweis gültig, wird der Ausleihevorgang fortgesetzt.
- Ist der Bibliotheksausweis ungültig, wird der Ausleihevorgang abgebrochen, und der Bibliothekskunde wird entsprechend benachrichtigt.

4.4.4 Rückgabe

AK0005 Verhalten des Bibliothekssystems bei der Rückgabe eines Leihobjektes
Ausgangssituation: Der Bibliothekskunde ist im Bibliothekssystem angemeldet.
Ereignis: Der Bibliothekskunde gibt ein entliehenes Leihobjekt zurück.
Erwartetes Ereignis: Das Bibliothekssystem ändert den Ausleihestatus von „Außer Haus“ zu „Im Haus“.

4.5 Assoziationen zwischen den Anforderungen

Um die Traceability-Funktionen der Tools untersuchen zu können, sollen einige der zuvor definierten Anforderungen an das Bibliothekssystem beispielhaft für weitere Assoziationen miteinander verknüpft werden. Im Einzelnen handelt es sich dabei um folgende Anforderungen:

- UR0001 ↔ UR0002
- UR0007 ↔ UR0012
- UR0002, UR0001 ↔ AK0001
- UR0009 ↔ AK0002
- UR0018 ↔ SR0001, SR0010, SR0011
- SR0010 ↔ AK0003

Diese Anforderungen wurden beispielhaft für weitere Assoziationen ausgewählt. Die Kombination der Anforderungen ermöglicht es, Assoziationen

- von einer Anforderung zur nächsten,
- von mehreren Anforderungen zu einer weiteren,
- von einer Anforderungen zu mehreren nachfolgenden und
- über mehrere Ebenen hinweg zu bilden.

5 Telelogic DOORS

Das schwedische Unternehmen *Telelogic AB* bietet als Marktführer mit einem Marktanteil von ca. 41% im Bereich der Produkt-Lebenszyklus-Management-Software mit *DOORS (Dynamic Object Oriented Requirements System)* das derzeit führende Tools im Requirements-Engineering an. Es wird derzeit von mehr als 100.000 Anwendern weltweit in über 1.000 Unternehmen eingesetzt. Ursprünglich von der Firma *QSS Ltd.* aus Oxford entwickelt und vertrieben, gehört es nach der Übernahme des Unternehmens durch die Telelogic AB im Jahr 2000 zu deren Produktportfolio. Telelogic AB wurde im Jahr 1983 gegründet und hat sich zum weltweit führenden Hersteller für die Entwicklung von Lösungen für das Produkt-Lebenszyklus-Management entwickelt.

Im Jahr 2005 erhielt DOORS die Yphise-Auszeichnung für die beste Anforderungssoftware. Yphise, ein unabhängiges Beratungsunternehmen aus Frankreich, bewertet streng nach ISO 9001-Methodologie und bietet so eine objektive Gegenüberstellung verschiedener Softwaresysteme. Laut Yphise bietet DOORS eine weitestgehend vollständige und homogene Abdeckung dieser Bewertungskategorien [vgl. Ver04, S. 17].

Die vorliegende Version des Tools trägt die Bezeichnung Telelogic DOORS 8.1. Das Tool lag in einer zeitlich begrenzten Vollversion vor. Der Zeitraum zur Nutzung dieser akademischen Lizenz wurde auf Nachfrage bei Telelogic von 15 Tagen auf 30 Tage erhöht.

Die folgenden Ergebnisse basieren auf einer eingehenden Untersuchung des Tools nach dem in Kapitel 4 beschriebenen Verfahren und Erkenntnissen aus der integrierten Hilfe des Tools [vgl. Tel06b].

5.1 Arbeitsumgebung - Der DOORS Explorer

Der *DOORS Explorer* (vgl. Abbildung 5.1, S. 39) bildet die Arbeitsoberfläche des Tools. In der Gestaltung wie in der Handhabung erinnert er stark an den MS Explorer. Am oberen Rand des Fensters befindet sich das Hauptmenü, darunter einige kontextabhängige Symbolleisten. Der mittlere Teil des Fensters wird durch den Dateimanager dominiert. Auf der linken Seite des Fensters befindet sich der Verzeichnisbaum. Hier aufgeführte Objekt werden durch Auswahl mit der Maus auf der rechten Seite des Fensters aus-

führlich dargestellt. „Die Menüoptionen passen sich an das gerade selektierte Objekt an, und die Auswahl von Objekten erfolgt über einen Doppelklick. Auch die bekannten Drag-and-Drop-Mechanismen funktionieren wie im MS Explorer, ebenso wie die Auswahlmöglichkeiten“ mit Hilfe der Steuerungstaste [vgl. Ver04, S. 99]. Am unteren Rand des Explorers befindet sich eine Statusleiste, in welcher der Name des aktuellen Benutzers sowie dessen Typ angezeigt werden. Zu unterscheiden sind vor allem die zwei folgenden Sichten:

Project View. In der Projektansicht werden im Verzeichnisbaum Projekte angezeigt, auf die der Benutzer Zugriff hat.

Database View. In der Datenbankansicht wird das Root-Verzeichnis der Datenbank angezeigt. Darunter werden sämtliche Teile der Datenbank aufgelistet, für welche der Benutzer die entsprechenden Zugriffsrechte besitzt.

Abbildung 5.1 zeigt den DOORS Explorer in der Database View mit Blick auf das Bibliothekssystem:

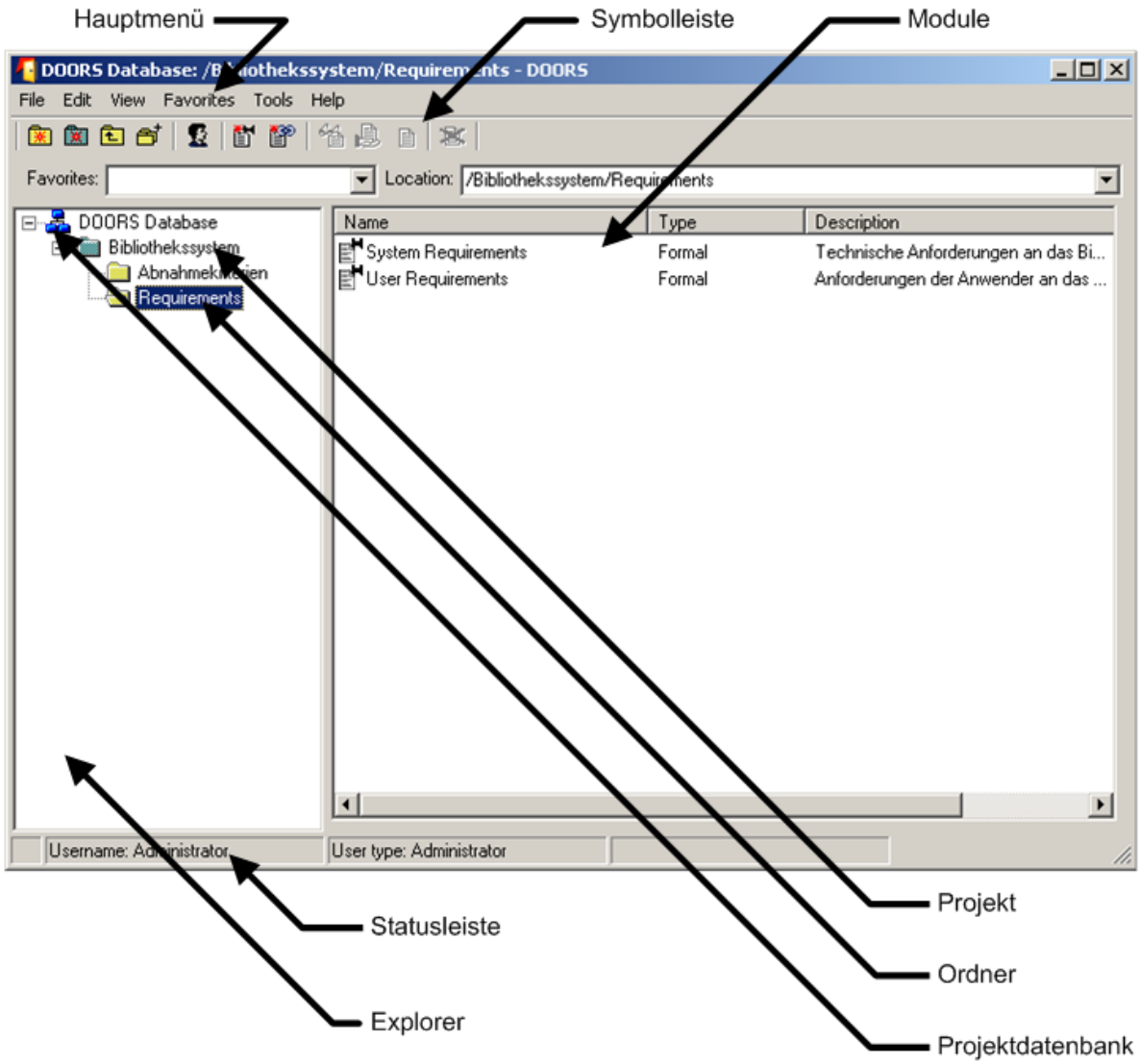


Abbildung 5.1: DOORS Explorer

5.2 Informationsstruktur

Im diesem Abschnitt sollen die wichtigsten Begriffe und Konzepte erläutert werden, die bei DOORS im Zusammenhang mit der Informationsstruktur stehen. Im Folgenden sollen sie erstmals aufgeführt werden, in Klammern steht jeweils die englische Bezeichnung des entsprechenden Begriffs:

- Ordner (*Folders*)
- Projekte (*Projects*)
- Module (*Modules*)
- Objekte (*Objects*)
- Attribute (*Attributes*)
- Links (*Links*)

5.2.1 Ordner

Ordner werden in DOORS genutzt, um Module (vgl. Abschnitt 5.2.3, S. 41) und Projekte (vgl. Abschnitt 5.2.2, S. 40) in der zentralen DOORS-Datenbank zu organisieren. Auf die gleiche Weise werden Dateien auf dem Computer mit Hilfe von Ordnern strukturiert. Ordner können weitere Ordner ebenso wie Projekte und Module enthalten. Ordner können beliebig umbenannt und verschoben werden.

In dem zuvor eingeführten Beispiel[a] werden die Module „User Requirements“ und „System Requirements“ in einem Ordner „Requirements“ zusammengefasst, während das Modul „Abnahmekriterien“ in einem gleichnamigen Ordner gespeichert wird (Ein Ordner wird im DOORS Explorer als gelbes Ordnersymbol dargestellt, vgl. Abbildung 5.1, S. 39).

[a] vgl. Abschnitt 4.1, S. 27

5.2.2 Projekte

Ein *Projekt* ist eine spezielle Art von Ordner und enthält sämtliche Informationen zu einem bestimmten Projekt. Projekte können sowohl weitere Projekte als auch Ordner und Module enthalten. Ein Projekt, welches in einem Projekt enthalten ist, bezeichnet man auch als *Teilprojekt*. Projekte sind anpassungsfähig, sie können also jederzeit umbenannt oder verschoben werden.

Ein Projekt unterscheidet sich in folgenden Punkten von einem Ordner:

- Nur Projekte haben innerhalb der Datenbank einen einzigartigen Namen. Daten innerhalb eines Projekts lassen sich so eindeutig lokalisieren.
- Nur Projekte können in externe DOORS Projektdatenbanken exportiert werden.
- Nur Projekte können archiviert werden.
- Ein Change Proposal System (vgl. 5.3.4, S. 48) kann nur innerhalb eines Projekts angelegt werden.

Im Beispiel[a] wird das Bibliothekssystem als Projekt definiert, welches alle weiteren Elemente enthält (im DOORS Explorer wird ein Projekt durch ein blaues Ordnersymbol dargestellt, vgl. Abbildung 5.1, S. 39).

[a] vgl. Abschnitt 4.1, S. 27

5.2.3 Module

Sämtliche Informationen in der DOORS-Datenbank werden in *Modulen* gespeichert, welche innerhalb von Ordnern oder Projekten verwaltet werden. Ein Modul setzt sich aus Objekten (vgl. Abschnitt 5.2.4, S. 43) zusammen. Mit Hilfe von Modulen kann ein Projekt weiter strukturiert werden.

In DOORS werden folgende Typen von Modulen unterschieden:

Formale Module. Formale Module sind die Standard-Module in DOORS. In diesem Modul können Anforderungen erfasst, hierarchisch geordnet und verwaltet werden.

Beschreibende Module. Beschreibende Module können benutzt werden, um Daten aus einem Textdokument in ein formales Modul zu importieren. Der Import in ein deskriptives Modul stellt dabei einen Zwischenschritt dar: Importiert man ein Textdokument in ein beschreibendes Modul, können die Daten nicht verändert werden, sondern müssen zunächst in ein formales Modul übertragen werden. Es handelt sich bei dem beschreibenden Modul um ein Relikt aus Zeiten, in denen dieses Element die einzige Möglichkeit darstellte, Daten aus anderen Tools zu importieren. Heute wird das beschreibende Modul vor allem dann eingesetzt, wenn man sicherstellen möchte, dass die originalen Daten nicht verändert werden können.

Link-Module. Link-Module enthalten Informationen über Beziehungen zwischen Anforderungen in DOORS (vgl. Kapitel 5.2.6, S. 44). Links zu Daten außerhalb des Tools werden nicht in Link-Modulen gespeichert. Innerhalb eines Link-Moduls werden die Informationen in so genannten Linksets unterteilt. Diese enthalten die Informationen über Links von einem bestimmten Modul zu einem anderen.

Abbildung 5.2 soll das Konzept der Link-Module verdeutlichen. Das als gestricheltes Rechteck dargestellte Link-Modul enthält die Linksets AB und XY. Linkset AB erfasst in diesem Fall sämtliche Informationen über die Beziehungen zwischen den Modul „User Requirements“ und „System Requirements“. Das Linkset XY könnte nun beispielsweise die Beziehungen zwischen dem Modul „System Requirements“ und einem weiteren Modul „Abnahmekriterien“ (in der Abbildung nicht dargestellt) enthalten.

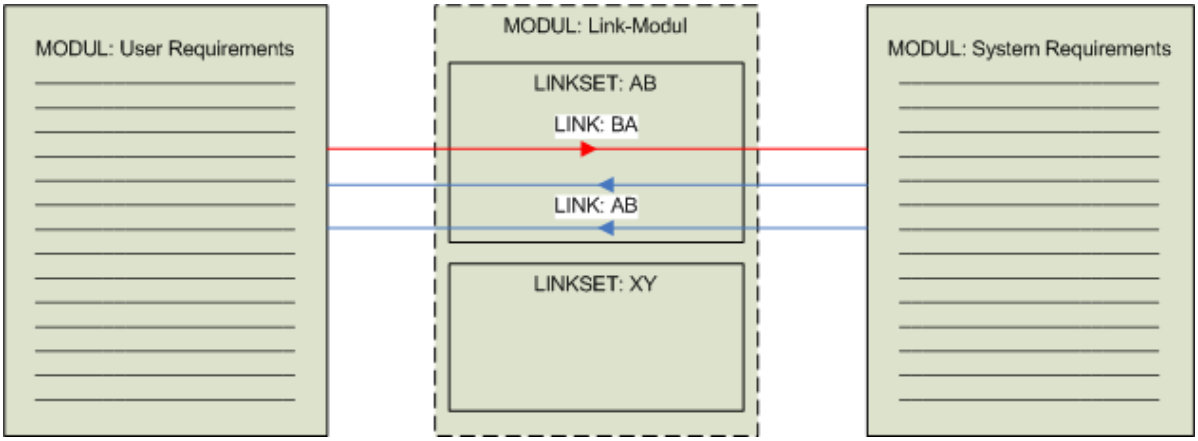


Abbildung 5.2: Link-Module und Linksets in DOORS [nach Tel06b]

Die folgende Abbildung (vgl. 5.3, S. 43) beschreibt den Aufbau des Projekts *Bibliothekssystem*[a]. In diesem Projekt gibt es jeweils ein formales Modul für die Anforderungen der Anwender (*User Requirements*), ein Modul für technische Anforderungen an das Bibliothekssystem (*System Requirements*) und ein Modul für die Testvorgaben (*Abnahmekriterien*). Zwischen den Anforderungen in den verschiedenen Modulen bestehen darüber hinaus Beziehungen (in der Abbildung als Pfeile dargestellt).

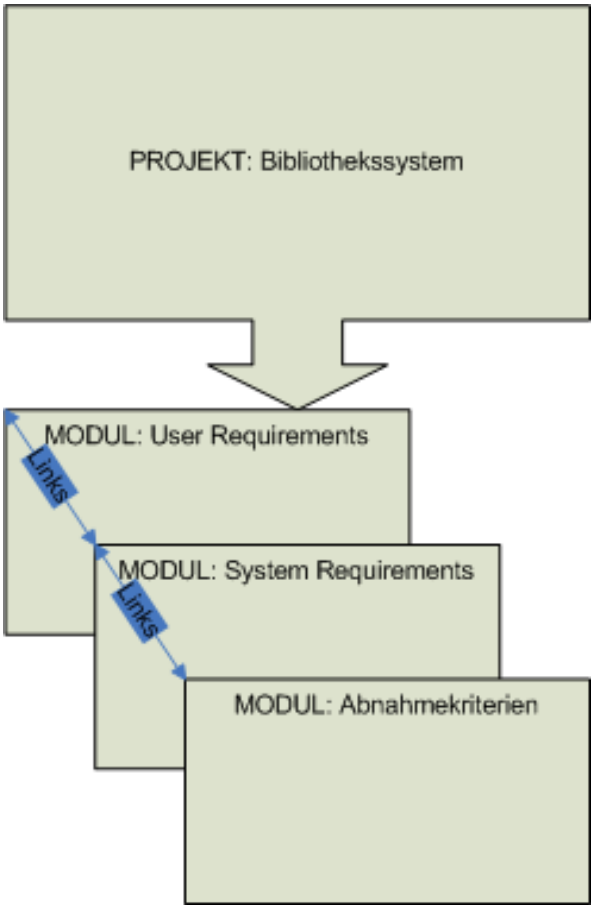


Abbildung 5.3: DOORS Projektstruktur [nach Tel06b]

[a] vgl. Abschnitt 4.1, S. 27

5.2.4 Objekte

Informationen werden innerhalb der Module in *Objekten* und deren Attributen gespeichert. DOORS gibt eine Reihe von Attributen (z.B. Created By oder Modified On) vor, bietet aber auch die Möglichkeit, eigene Attribute (z.B. Priority oder Status) für Objekte

und Module zu erzeugen. Objekte können innerhalb eines Moduls beliebig tief gegliedert werden, so dass man eine Objektstruktur aufbauen kann.

Ein Objekt kann z.B. einer konkreten Anforderung im Modul *Anwenderforderungen* entsprechen. In dem Bibliotheksbeispiel[a] kann im Modul *User Requirements* der Titel des (Teil-)Dokuments *Bestandsverwaltung* ein Objekt vom Typ Heading sein, also eine Überschrift. Die dazugehörigen Anforderungen (z.B. UR0011) werden dann ebenfalls als Objekte, diesmal vom Typ Text, eine Ebene unter der Überschrift eingefügt. So lassen sich die Anforderungen innerhalb des Moduls hierarchisch anordnen.

[a] vgl. Abschnitt 4.1, S. 27

5.2.5 Attribute

Mit Hilfe der *Attribute* werden die Eigenschaften von Modulen, Objekten oder Links beschrieben. Es lassen sich zwei Typen von Attributen unterscheiden:

Systemdefinierte Attribute. Systemdefinierte Attribute werden automatisch von DOORS erstellt und können bis auf wenige Ausnahmen nicht vom Benutzer verändert oder gelöscht werden. Die verschiedenen Elemente besitzen unterschiedliche Attribute. Beispiele sind Name, Description, Type, URL, Created By oder Created On.

Benutzerdefinierte Attribute. Benutzerdefinierte Attribute dagegen werden vom Benutzer definiert. Er legt auch die Zugriffsrechte auf diese Attribute fest.

5.2.6 Links

DOORS ermöglicht es dem Benutzer, Anforderungen mit Hilfe von *Links* miteinander zu verknüpfen. Ein Link bezeichnet eine Verbindung zwischen zwei Objekten. Links werden zwischen Objekten erstellt, die in einer Beziehung zueinander stehen. Alle Links in DOORS sind *gerichtet*, d.h. es existiert immer eine Quelle und ein Ziel. Auf diese Weise wird innerhalb von DOORS die Traceability der Anforderungen über den Projekt-Lebenszyklus hin gewährleistet.

Verknüpfungen sind *bidirektional* traversierbar. Schlägt z.B. der bereits erwähnte Testfall fehl, „kann anhand der Links rückwärts zur falschen Anforderungsdefinition oder Designvorgabe navigiert werden“ [vgl. Ver04, S. 128]. So wird durch die Links auch das Change-Management unterstützt: Die Auswirkungen einer Änderung an einer bestimmten Anforderung auf den Rest des Systems kann ad hoc dargestellt und überprüft werden (vgl. Abschnitt 5.3.5, S. 49).

Die Verknüpfungen der Anforderungen nach dem im Beispiel[a] vorgegebenen Schema kann DOORS ohne weiteres einrichten. Die Anforderung UR0010 aus dem Modul *User Requirements* wird z.B. wie erwartet mit Anforderung SR0010 aus dem Modul *System Requirements* verlinkt, da sie diese Anforderung ergänzt. Anforderung SR0010 wird wiederum mit der Testvorgabe AK0003 verknüpft, welche in dem Modul *Abnahmekriterien* gespeichert ist.

[a] vgl. Abschnitt 4.1, S. 27

5.3 Funktionen

5.3.1 Ein- und Ausgabemöglichkeiten

Anforderungen können direkt in DOORS eingeben werden oder aus einer Reihe von Tools importiert werden. Unterstützt werden *MS Office*, *Adobe FrameMaker*, *Plain Text (ASCII)*, *Rich Text Format (RTF)*, *Comma Seperated Values (CSV)* und *Tab Seperated Values (TSV)*. Die Anforderungen werden innerhalb von Objekten gespeichert und können mit einer Reihe von Attributen versehen werden. Mit Hilfe der Funktion *Restore* ist es auch möglich, archivierte Module wiederherzustellen. Sobald die Anforderungen in DOORS angelegt sind, kann man sie über den gesamten Projekt-Lebenszyklus hinweg verfolgen und verwalten, wobei man von einer Reihe von Funktionen (z.B. Views, Links, Traceability Analyse) unterstützt wird[1].

Die Exportmöglichkeiten von DOORS sind ebenso vielfältig wie die Importmöglichkeiten. Die Daten können in verschiedene Tools exportiert werden (s.o.), außerdem können Archive sowie individuelle Reports erstellt werden.

5.3.2 Traceability

Traceability bezeichnet in DOORS die Möglichkeit, Beziehungen zwischen Anforderungen zu erstellen und zu verwalten sowie Änderungen an diesen zu überwachen. Dazu

[1] Die Möglichkeit, Anforderungen aus einem MS Word-Dokument zu importieren, konnte im Rahmen dieser Untersuchung nicht überprüft werden. Dokumente aus MS Word werden in DOORS importiert, in dem sie aus MS Word in das entsprechende Modul des DOORS Projekts exportiert werden. Die entsprechende Erweiterung des MS Word-Menüs wurde bei der Installation von DOORS aber aus ungeklärten Gründen nicht mitinstalliert. Nachdem das fragliche Dokument jedoch als RTF gespeichert wurde, konnten die Daten ohne Probleme importiert werden. Der Import aus einem Plain Text Dokument funktioniert ebenfalls ohne größere Schwierigkeiten, erfordert aber viel Nacharbeit, da eine Dokumentenstruktur nur bedingt übernommen wird.

werden Anforderungen durch Links miteinander verknüpft (vgl. 5.2.6, S. 44). Ändert sich ein Endpunkt der Verknüpfung, gilt der entsprechende Link fortan als „verdächtig“ (englisch: *suspect*).

Mit Hilfe der *Suspect Links* lassen sich Veränderungen an miteinander verlinkten Objekten verfolgen. Ein Link wird zu einem Suspect Link, wenn sich bestimmte Attribute eines von der Beziehung betroffenen Objektes verändern. Diese Attribute müssen bei ihrer Änderung ein *Logging* veranlassen. So kann beispielsweise eine Änderung einer Anwenderforderung die entsprechenden Abnahmekriterien zum Überprüfen dieser ungültig machen. Diese Änderung würde nun automatisch dazu führen, dass das Objekt am anderen Ende des Links, in diesem Fall die Testvorgabe, als verdächtig markiert werden würde.

Neben dem Management von Anforderungsbeziehungen in aktuellen Projekten bietet DOORS die Möglichkeit, auch zu älteren Versionen eines Projekts, den so genannten *Baselines* (vgl. Abschnitt 5.3.3, S. 47), zu verlinken. Dies ist ein besonderer Vorteil bei der inkrementellen Entwicklung. In einem Projekt wie dem Bibliotheksbeispiel kann man z.B. davon ausgehen, dass eine Gruppe von Analysten die Anwenderforderungen (User Requirements) erhebt, auf denen Ingenieure ihre Systemanforderungen (System Requirements) basieren. Testingenieure wiederum entwickeln die entsprechenden Abnahmekriterien für diese Anforderungen. Ist dieser Prozess abgeschlossen, geht das Projekt in die nächste inkrementelle Phase, und der Vorgang beginnt von neuem. Meistens aber laufen diese Prozesse nicht parallel. So kommt es oft vor, dass die Analysten ihre Erhebung vor den Ingenieuren abschließen. DOORS löst dieses Problem, indem es denjenigen Gruppen, die eine Phase bereits abgeschlossen haben, erlaubt, eine eingefrorene „read-only“ Version dieser Phase für die anderen Gruppen zu hinterlassen. Diese können darauf zugreifen und auch weiterhin zu dieser Version verlinken. Wenn diese Gruppen ihre Phasen dann ebenfalls beenden, können sie die entsprechenden Dokumente zu der eingefrorenen Version hinzufügen.

In Abbildung 5.4 wurden die „User Requirements“ bereits einer Baseline hinzugefügt. Während die „System Requirements“ und „Tests“ weiterhin entwickelt werden, kann von diesen ohne weiteres in die Baseline hinein verlinkt werden.

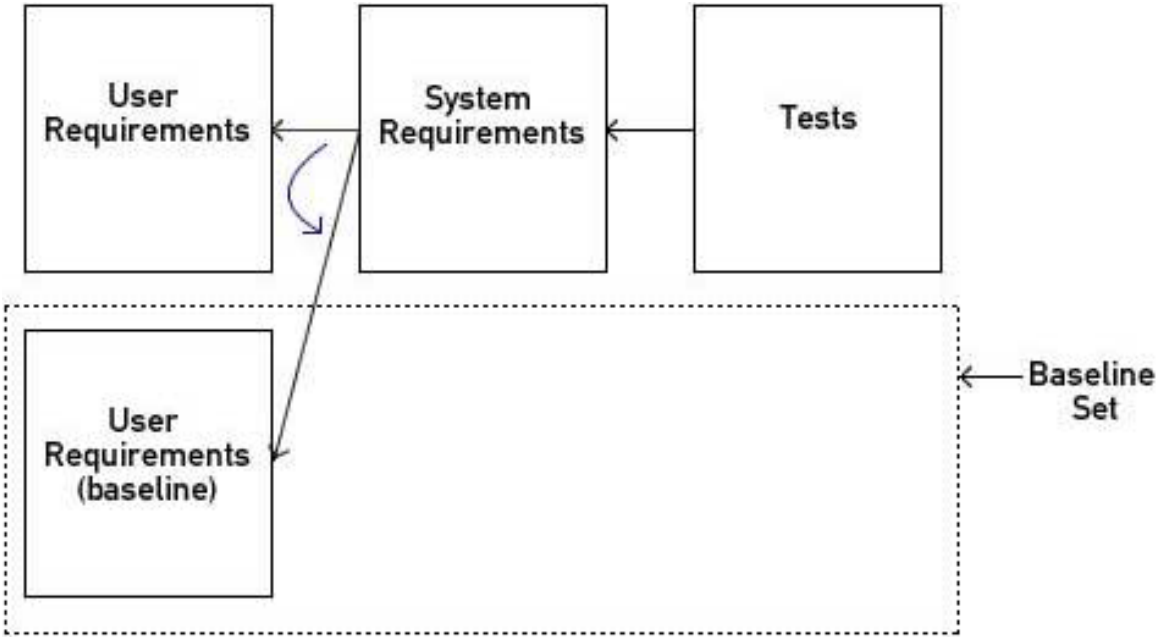


Abbildung 5.4: Baseline Traceability in DOORS [vgl. Tel06b]

Darüber verfolgt DOORS das Konzept der *Intelligent Traceability*. Dies ermöglicht es dem Anwender, beim Bilden einer Baseline (vgl. Abschnitt 5.3.3, S. 47) nicht nur einzelne Dokumente zu speichern, sondern auch die Beziehungen zwischen diesen Dokumenten zu erhalten.[2]

5.3.3 Versionsmanagement

Das Management verschiedener *Versionen* der erfassten Anforderungen über den Projektverlauf geschieht in DOORS mit Hilfe so genannter *Baselines*. In DOORS bezeichnet

[2] Traceability in den Code hinein ist prinzipiell möglich, indem man beispielsweise ein Modul *Code* erzeugt, in dem einzelne Codefragmente als Objekte abgespeichert werden. Diese können dann mit den entsprechenden Anforderungen aus den anderen Modulen verknüpft werden. Eine weitere Möglichkeit wäre, den Code in einem Textdokument zu erzeugen und in ein beschreibendes Modul zu importieren. In diesem Fall könnten z.B. einzelne Klassen des Codes bei Bedarf in ein formales Modul importiert und dort weiter bearbeitet werden, während der originale Code nicht verändert werden kann. So können Anforderungen über den gesamten Entwicklungsprozess hin verfolgt und Änderungen in den Anforderungen und deren mögliche Auswirkungen auf den Code hin untersucht werden.

eine Baseline eine nicht weiter bearbeitbare Kopie eines Moduls. Um in DOORS eine Baseline zu bilden, muss zunächst eine so genannte *Baseline Set Definition* erzeugt werden. Dabei handelt es sich um eine Vorlage für *Baseline Sets*. Baseline Sets wiederum bezeichnen eine Gruppe von Baselines, die für das Projektmanagement als eine Einheit betrachtet werden sollen. Baseline Set Definitionen müssen für alle Module erstellt werden, bei denen Intelligent Traceability angewandt werden soll.

Für jede Phase eines Projekts können inkrementelle Baseline Sets gebildet werden. Baselines Sets werden zusammen mit der Baseline Set Definition gespeichert, so dass sie zu jedem Zeitpunkt verfügbar und vergleichbar sind. Zum Vergleichen zweier unterschiedlicher Baselines bietet DOORS die *Compare*-Funktion. Diese listet alle gefundenen Differenzen zwischen zwei Baselines auf, die dann im Detail betrachtet werden können.

Einmal erstellt, können Baselines auch mit einer *elektronischen Signatur* versehen werden. Um Baselines zu signieren, müssen folgende Voraussetzungen gegeben sein:

- Ein Satz von möglichen Signaturlabels muss im Vorfeld definiert sein, z.B. „genehmigt“, „unter Vorbehalt genehmigt“ und „nicht genehmigt“.
- Die zur Unterschrift berechtigten Personen müssen ausdrücklich benannt sein.

Bezüglich der *Historie* (in Abgrenzung zur Versionierung) von Modulen und Objekten kann in DOORS wie folgt unterschieden werden:

Module History. Hält alle Aktionen auf dem Modul (Erstellung, Festlegen von Attributen u.ä.) zusammen mit dem verantwortlichen Benutzer, dem genauen Zeitpunkt und der Art der Aktion fest.

Object History. Hält alle Aktionen mit Benutzer, Zeitpunkt und Art der Aktion auf allen Objekten eines Moduls oder auf einem bestimmten Objekt fest.

Session History. Hält alle Sitzungen mit Nummer, Zeitpunkt und Benutzer fest.

Die *Historie* hält also alle Veränderungen eines Moduls, seiner Objekte und der entsprechenden Attribute fest, und zwar ab dem Zeitpunkt, zu dem die letzte Baseline erstellt wurde. Falls keine Baseline vorhanden ist, werden alle Daten ab der Erstellung des Moduls bzw. des Objektes angezeigt. Ausgenommen ist die *Session History*, hier werden die Daten ab der Erstellung eines Moduls gespeichert.

5.3.4 Veränderungsmanagement

Das *Veränderungsmanagement* wird in DOORS durch das *Change Proposal System (CPS)* geregelt. Das CPS ermöglicht es dem Anwender, Änderungen für Anforderungen vorzuschlagen. Dabei wird zwischen *Proposals* und *Suggestions* unterschieden: Ein

Proposal bezeichnet einen detaillierten Kommentar, der sich auf ein bestimmtes Objekt eines bestimmten Modules bezieht. Eine Suggestion dagegen ist einen genereller Kommentar, der sich z.B. auf prinzipielle Fragen bezüglich eines Projekts bezieht.

Verschiedene Change Proposals, die sich auf eine bestimmte Anforderung beziehen, können kombiniert werden. Es können aber auch Change Proposals gruppiert werden, die sich auf unterschiedliche Anforderungen beziehen. Darüber hinaus besteht auch die Möglichkeit, *Reviews* durchzuführen und dabei die Auswirkungen der Änderungen auf das gesamte System zu analysieren. Änderungen, die akzeptiert wurden, können verabschiedet und implementiert werden. Wird ein Projekt für ein Review konfiguriert, erstellt DOORS in diesem Projekt automatisch einen *CPS-Ordner*. Dieser Ordner enthält verschiedene Module, die vom CPS genutzt werden, unter anderem:

- Ein *Suggestions Module*, das Kommentare und Vorschläge für ein Projekt zusammenfasst
- Ein *Proposals Module*, das konkrete Änderungsvorschläge enthält
- Ein *Groups Module*, welches Informationen über die entsprechenden Gruppen enthält (nur falls Gruppen von Change Proposals erstellt wurden)

5.3.5 Analysemöglichkeiten

Impact- und Traceability-Analyse

Impact- und Traceability-Analysen geben Aufschluss über ein- und ausgehende Links des aktuellen (markierten) Moduls. Bei der Analyse des Moduls werden nur DOORS-Verknüpfungen untersucht, externe Links zu anderen Tools sind in der Betrachtung nicht eingeschlossen. Die *Impact Analyse* untersucht Links, welche aus dem Modul in ein anderes verweisen, und ermöglicht damit Rückschlüsse über die Auswirkungen von Änderungen im aktuellen Modul auf weitere Objekte. Die *Traceability Analyse* hingegen untersucht eingehende Links aus anderen Modulen. Änderungen an den entsprechenden Ziel-Objekten haben in diesem Fall Auswirkungen auf das aktuelle Modul.

Abbildung 5.5 veranschaulicht den Unterschied zwischen Impact- und Traceability-Analyse. Auf der linken Seite sind die eingehenden Links zu sehen, die auf das Modul in der Mitte verweisen und von der Traceability-Analyse erfasst werden. Aus diesem Modul ausgehende Links, im Bild rechts, werden dagegen von der Impact-Analyse erfasst.

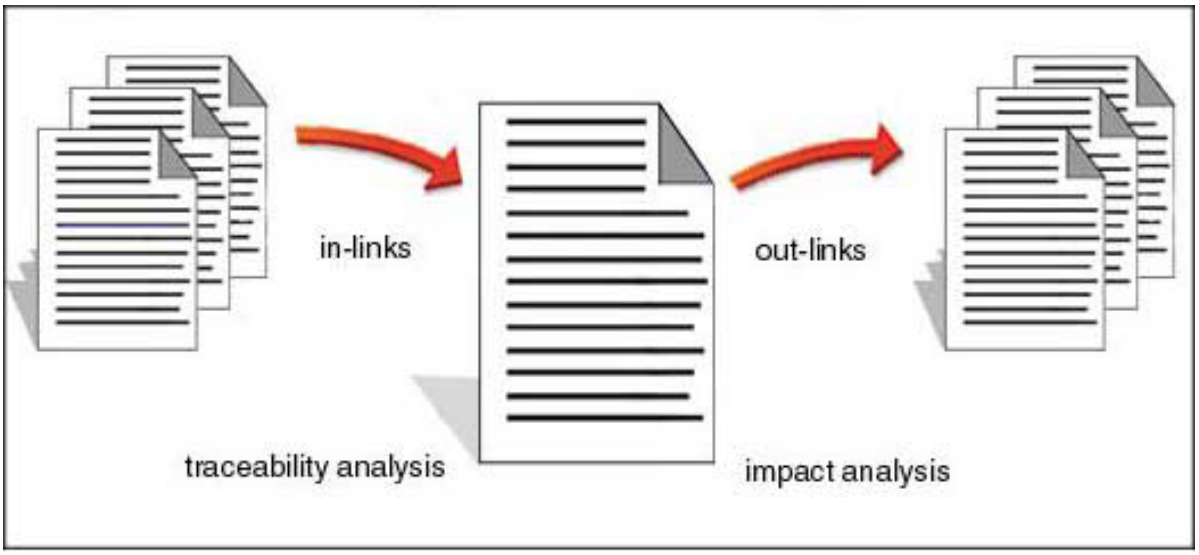


Abbildung 5.5: Konzept der Impact- und Traceability-Analyse [vgl. Tel06b]

Bei beiden Analysemöglichkeiten lässt sich die Tiefe der Untersuchung wählen, d.h. man kann entscheiden, wie weit die Untersuchung entlang der Kette verlinkter Objekte geht. Wählt man z.B. bei der Analyse von Modul A eine Tiefe von 2, werden Informationen über die Objekte aus Modul B angezeigt, die mit Objekten aus dem aktuellen Modul verknüpft sind. Zusätzlich erhält man Informationen über Objekte aus Modul C, die mit den entsprechenden Objekten aus Modul B verknüpft sind. Bei einer Tiefe von 0 werden nur Ziel-Objekte im aktuellen Modul angezeigt. Eine unendlich gewählte Tiefe („n-tief“) ist nicht möglich.

Traceability Explorer

Der *Traceability Explorer* ermöglicht dem Anwender die einfache Navigation zwischen Objekten, die mit Anforderungen aus dem aktuellen Modul verknüpft sind. Sie werden, in einer Baumstruktur geordnet, angezeigt. Mit einem Klick der rechten Maustaste kann man sich das ausgewählte Objekt anzeigen lassen.

Im Beispiel[a] wurden einige Anforderungen aus den Modulen *User Requirements*, *System Requirements* und *Abnahmekriterien* miteinander verknüpft[b]. Abbildung 5.6 zeigt die Auswirkungen dieser Verknüpfungen auf den Traceability Explorer. Die Anforderung UR0018 aus dem Modul *User Requirements* ist mit den Anforderungen SR0001, SR0010 und SR0011 verknüpft. Anforderung SR0010 wiederum steht in Beziehung zu dem Abnahmekriterium AK0003.

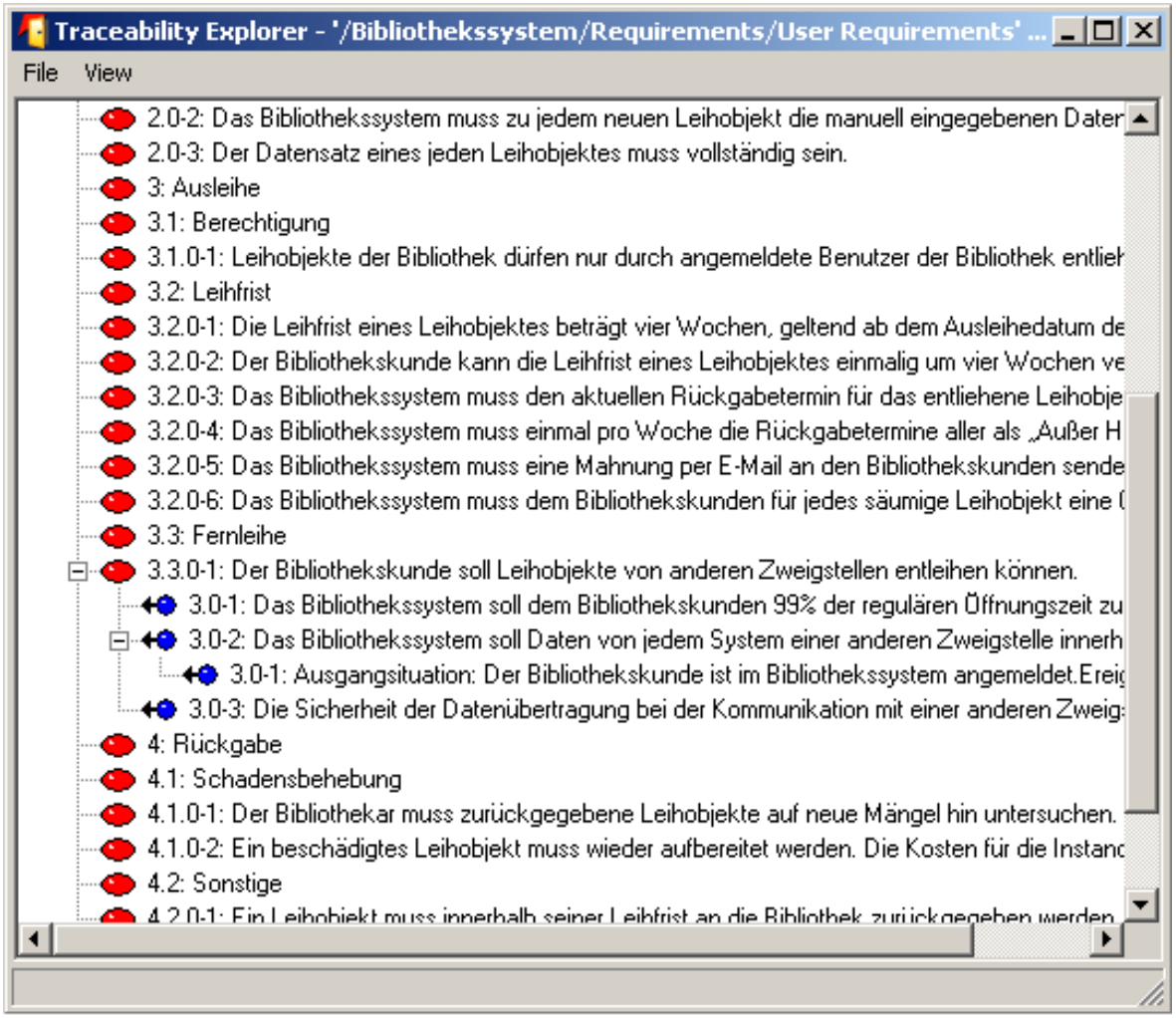


Abbildung 5.6: Traceability Explorer

[a] vgl. Abschnitt 4.1, S. 27
[b] vgl. Abschnitt 4.5, S. 36

Traceability Wizard

Mit dem *Traceability Wizard* lassen sich alle Objekte und deren Links im aktuellen Modul anzeigen. Dazu werden im DOORS Explorer entsprechende Spalten zur aktuellen Ansicht hinzugefügt. Mit Hilfe des Traceability Wizards lassen sich sowohl eingehende als auch ausgehende Links untersuchen, wobei die Analyse in diesem Fall *interne* und *externe* Links umfassen kann. Weiterhin besteht die Möglichkeit, genau zu spezifizieren, welche Module in die Betrachtung mit eingeschlossen werden sollen. Wie bei der Impact- und Traceability-Analyse muss auch im Traceability Wizard die Tiefe der Untersuchung festgelegt werden.

In folgendem Beispiel wurden die eingehenden Links für das Modul „User Requirements“ mit einer Tiefe von 2 untersucht. Das Ergebnis einer solchen Analyse könnte für das Bibliothekssystem[a] beispielsweise wie folgt aussehen:

ID	Anforderungen der Anwender an das Bibliothekssystem	In-links at depth 1	In-links at depth 2
UR26	**3.3 Fernleihe**		
UR27	Der Bibliothekskunde soll Leihobjekte von anderen Zweigstellen entleihen können.	System Requirements SR10 Das Bibliothekssystem soll dem Bibliothekskunden 99% der regulären Öffnungszeit zur Verfügung stehen.	/Bibliothekssystem/Abnahmekriterien/Abnahmekriterien AK6 **Ausgangsituation:** Der Bibliothekskunde ist im Bibliothekssystem angemeldet. **Ereignis:** Der Bibliothekskunde fragt den Bestand einer anderen Zweigstelle der Bibliothek ab. **Erwartetes Ereignis:** Das Bibliothekssystem hat die Anfrage des Bibliothekskunden innerhalb von 30 Sekunden zu beantworten.
		System Requirements SR11 Das Bibliothekssystem soll Daten von jedem System einer anderen Zweigstelle innerhalb von 30 Sekunden empfangen und verarbeiten können.	
		System Requirements SR12 Die Sicherheit der Datenübertragung bei der Kommunikation mit einer anderen Zweigstelle soll durch SSL gewährleistet werden.	

Abbildung 5.7: Ergebnis einer Analyse mit dem Traceability Wizard

Aus Abbildung 5.7 geht hervor, dass die Anforderungen AK0006 aus dem Modul „Abnahmekriterien“ auf die Anforderungen SR0010, SR0011 und SR0012 aus dem Modul „System Requirements“ verweisen. Diese wiederum basieren auf der Anforderung UR0027 aus dem Modul „User Requirements“.

[a] vgl. Abschnitt 4.1, S. 27

5.4 Sonstige Aspekte

5.4.1 Administrative Aspekte

Als *Multi-User-Tool* bietet DOORS verschiedene Möglichkeiten zur Benutzeradministration, allen voran Funktionen zum Einrichten neuer Benutzer, dem Einrichten von Benutzergruppen und das Setzen von unterschiedlichen Zugriffsrechten.

Wird ein neuer Benutzer angelegt, bietet DOORS eine Auswahl verschiedener Benutzertypen an:

Standard-Benutzer. Benutzer diesen Typs können mit den Daten in DOORS arbeiten, können aber keine sonstigen Aufgaben übernehmen wie etwas das Archivieren von Daten oder das Anlegen neuer Benutzer.

Projekt Manager. Benutzer diesen Typs haben über die Rechte eines Standard-Benutzers hinaus auch die Möglichkeit, Daten zu archivieren und zu partitionieren, neue Benutzergruppen anzulegen (aber keine neuen Benutzer), Benutzer einer Gruppen hinzuzufügen oder sie aus einer Gruppe zu entfernen.

Datenbankadministratoren. Diese Art von Benutzer kann sämtliche Aufgaben in DOORS übernehmen. Im Gegensatz zu einem Projekt Manager können sie also auch neue Benutzer anlegen und die Datenbank verwalten.

Individuelle Benutzertypen. Individuell eingerichtete Benutzertypen können die Rechte verschiedener Benutzertypen kombinieren. Zum Beispiel könnte man einen individuellen Benutzertyp mit den Rechten zum Partitionieren von Daten ausstatten. Er wäre damit mächtiger als ein Standard-Benutzer, aber immer noch weniger mächtig als ein Projektmanager. Gleichartige Benutzer können in DOORS in Gruppen zusammengefasst werden. Damit ist es möglich, einer gewissen Gruppe von Projektmitgliedern (z.B. Software-Entwickler) die Zugriffsrechte auf ein bestimmtes Modul eines Projekts zu erteilen. Dadurch ist es nicht mehr nötig, die Zugriffsrechte auf Benutzerebene zu vergeben, was eine effizientere Benutzerverwaltung ermöglicht.

In DOORS wird zwischen den folgenden Zugriffsrechten unterschieden:

Read (R) zum Lesen von Daten,

Create (C) zum Erstellen von Daten,

Modify (M) zum Bearbeiten von Daten,

Delete (D) zum Löschen von Daten und

Admin (A) zum Administrieren von Daten.

5.4.2 Customizing

DOORS bietet eine Möglichkeit an, die Funktionen des Tools zu kontrollieren und zu erweitern. Dazu wird die *DOORS eXtension Language (DXL)* genutzt, eine Skriptsprache. Diese ähnelt syntaktisch den bekannten Programmiersprachen *C und C++*. Mit Hilfe von DXL soll man u.a.:

- Routinen oder regelmäßig zu bearbeitende Arbeitsschritte automatisieren können (z.B. die Kalkulation von Attributwerten),
- auf Ereignisse reagieren können (indem durch sie bestimmte Tools ausgelöst werden) und
- eigene Optionen zum DOORS Menü hinzufügen können.

Um entsprechende Programme entwickeln zu können, bietet DOORS das *DXL Interaction Window*, ein kleiner Editor. Für große Programme wird jedoch die Nutzung einer professionelleren Entwicklungsumgebung angeraten. Der so geschriebene Code kann später in das DXL Interaction Window geladen und ausgeführt werden.

5.5 Klassendiagramm

Im Folgenden wird das Klassendiagramm zu Telelogic DOORS im Einzelnen erläutert. Das Diagramm wurde mit Hilfe der Informationen erstellt, die im Rahmen der Untersuchung des Tools gewonnen werden konnten.

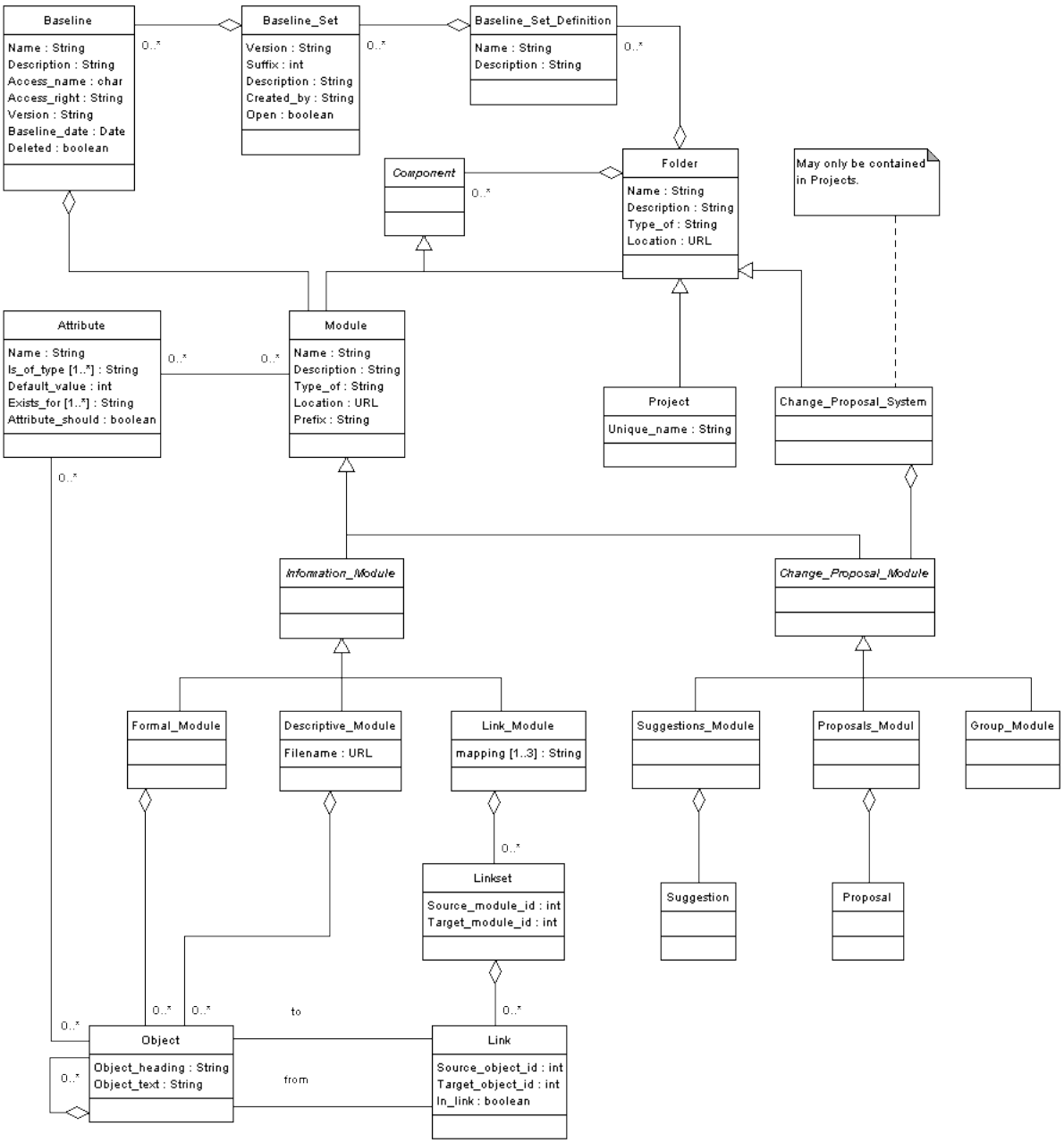


Abbildung 5.8: DOORS-Klassendiagramm

Das vorliegende Klassendiagramm (vgl. Abbildung 5.8, S. 55) zentriert sich um ein *Kompositum*. Diese spezielle Struktur ermöglicht den Aufbau einer Baumtopologie[3]. Im Klassendiagramm von DOORS setzt es sich im Wesentlichen aus den vier Klassen Component (deutsch: Komponente), Module (deutsch: Modul), Folder (deutsch: Ordner) und Project (deutsch: Projekt) zusammen und bildet damit die grundlegende Struktur eines DOORS-Projekts ab (vgl. Abbildung 5.9, S. 56). In dieser speziellen Struktur ist Component abstrakt definiert, d.h. von dieser Klasse können keine Objekte erzeugt werden. Von dieser Klasse dürfen Ordner oder Module gebildet werden, wobei nur ein Ordner weitere Komponenten enthalten darf.

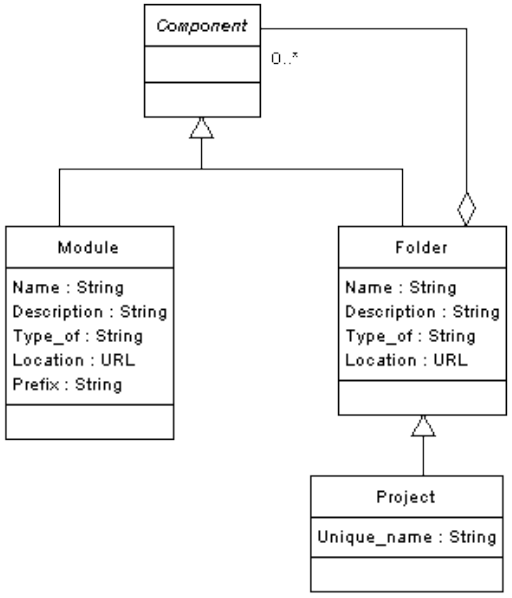


Abbildung 5.9: Composite-Struktur im DOORS-Klassendiagramm

Ordner stellen in DOORS die wesentliche Komponente zur Organisation der gesammelten Informationen dar. Ein *Projekt* ist im Diagramm als Spezialisierung eines Ordners modelliert. Damit wird der Tatsache Rechnung getragen, dass Ordner sowohl Module als auch weitere Ordner oder Projekte enthalten können. Gleichzeitig können Projekte weitere Ordner und Projekte sowie Module enthalten. Ein Modul dagegen kann keine weiteren Objekte dieser Typen enthalten.

Ein Projekt dient ebenfalls dazu, Informationen innerhalb eines DOORS-Projekts zu strukturieren. So können z.B. Projekte in Teilprojekte untergliedert werden. In Bezug auf die Datenstruktur unterscheidet sich Project vor allem durch ihren eindeutigen Na-

[3] Diese ist dadurch gekennzeichnet, „dass ausgehend von einer Wurzel eine Menge von Verzweigungen zu weiteren Knoten existiert, die bis auf die letzte Stufe (»Blätter«) wiederum die gleiche grundsätzliche Struktur mit weiteren Verzweigungen aufbauen“ [vgl. DAT06].

men (vgl. Abschnitt 5.2.2, S. 40) von Folder. Keine weitere Komponente innerhalb eines bestimmten Projekts kann dessen Namen tragen.

Die Klasse Module subsumiert verschiedene Arten von Modulen (vgl. Abbildung 5.10, S. 57), die sich grob in zwei Gruppen aufteilen lassen: *Informations-Module* und *Change-Proposal-Module*.

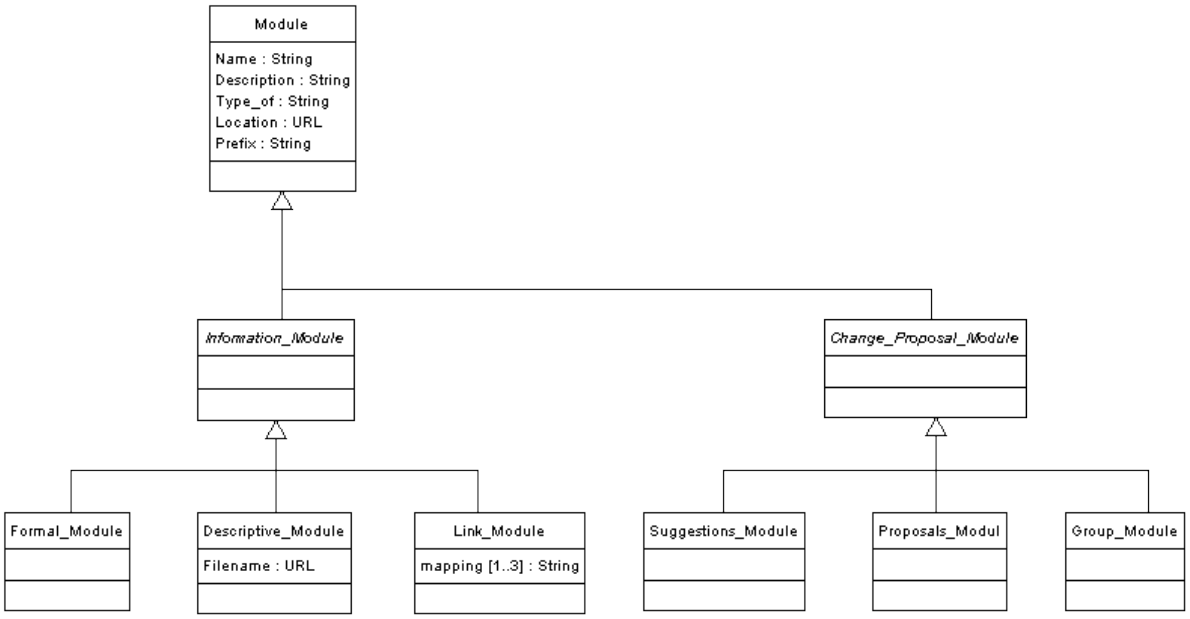


Abbildung 5.10: Module in DOORS

Informations-Module. Module dieser Gruppe werden genutzt, um die Informationen eines DOORS-Projekts zu strukturieren, allen voran die Anforderungen und deren Beziehungen untereinander. Die Gruppe gilt als *complete* und *disjoint*, d.h. sie ist vollständig und disjunkt. In die Kategorie der Informations-Module fallen Formal Module, Descriptive Module und Link Module.

Change-Proposal-Module. Module dieser Gruppe werden genutzt, um Informationen im Rahmen des Veränderungsmanagements zu verwalten. Sie stehen in einer „Teile-Ganzes-Beziehung“ zu Change Proposal System. Das bedeutet, dass die aggregierten Instanzen dieser Klasse als Teil eines Ganzen betrachtet werden, welches durch die Klasse am anderen Ende der Beziehung beschrieben wird. In diesem Falle sind also Instanzen der Klassen Suggestions Module, Proposals Module und Group Module Teile von Change Proposal System. Diese Gruppe gilt als unvollständig und disjunkt. Unvollständig ist die Gruppe, weil zwar im Rahmen der Untersuchung keine weiteren Subklassen ausgemacht werden konnten, diese aber prinzipiell existieren können. Ein Change Proposal System darf nur innerhalb eines Projekts angelegt werden, nicht aber innerhalb eines Ordners.

Allen Klassen dieser beiden Gruppen ist gemein, dass sie sämtliche Eigenschaften von Module vollständig übernehmen. Sämtliche Informationen eines DOORS-Projekts werden in diesen Modulen gespeichert. Anforderungen und deren Beziehungen untereinander werden innerhalb der Informations-Module verwaltet (vgl. Abbildung 5.11, S. 58).

Anforderungen werden als Instanzen der Klasse Object erfasst, die in den Klassen Formal Module oder Descriptive Module zusammengeführt werden. Ein bestimmtes Objekt kann dabei immer nur *entweder* in einem formalen Modul *oder* in einem beschreibenden Modul enthalten sein. Innerhalb eines bestimmten Ordners oder Projekts lassen sich Anforderungen mit Hilfe der Module also weiter strukturieren (vgl. Abschnitt 5.2.3, S. 41).

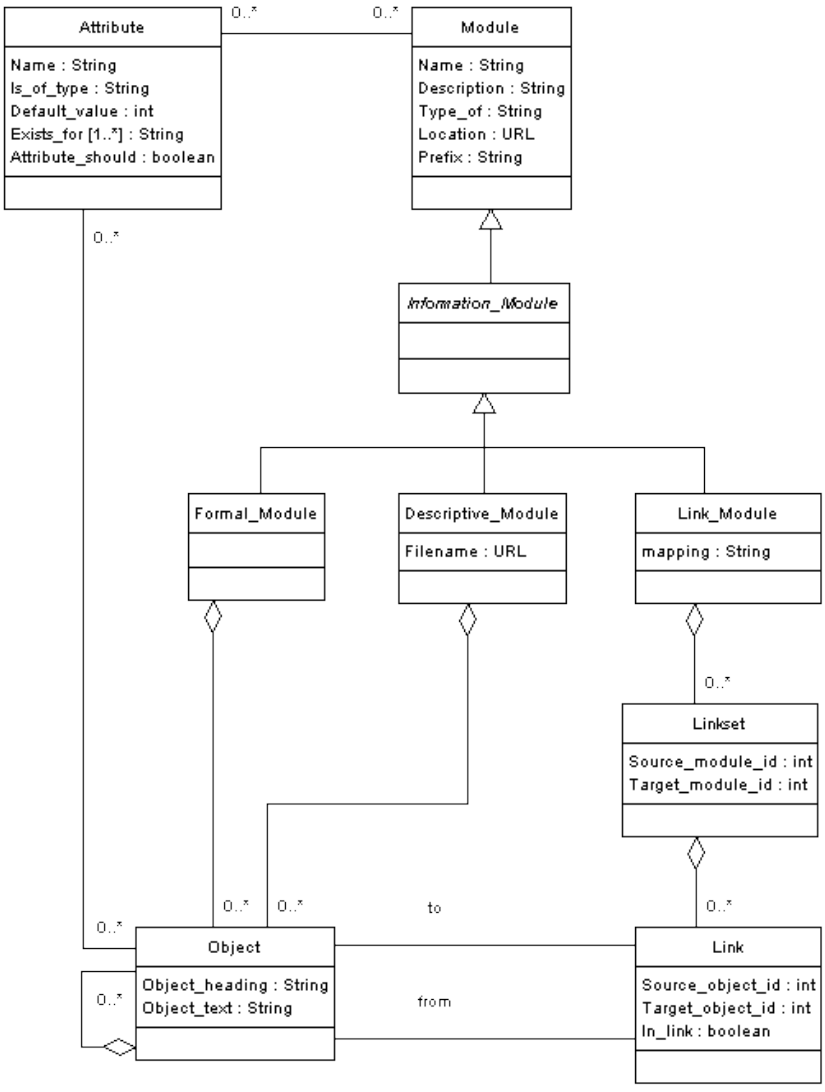


Abbildung 5.11: Informationsstruktur in DOORS

Die Beziehungen der Anforderungen untereinander werden mit Hilfe von Links modelliert. Dabei werden Instanzen der Klasse Links in einem *Linkset* zusammengeführt. Verschiedene Linksets wiederum sind Teile eines Link-Moduls. Das Attribut mapping der Klasse Link-Module ist ausschlaggebend dafür, wie viele Anforderungen eines bestimmten Moduls miteinander verknüpft werden können. So kann festgelegt werden, dass ein Link-Modul entweder optionale (0..1), zwingende (1..1), oder beliebige Beziehungen (0/1..*) unterstützt.

Die Klasse Attribute ermöglicht es, Module und Objekte mittels individueller Attribute zu beschreiben (vgl. Abbildung 5.11, S. 58). Das Attribut is_of_type der Klasse Attribute beschreibt, welche Werte das spätere Attribut annehmen darf.

Zur Auswahl stehen die folgenden Datentypen:

- Text
- String
- Integer
- Real
- Date
- Enumeration
- Username

Mit Hilfe der Klasse Baseline können verschiedene Versionen eines Moduls erfasst (vgl. Abbildung 5.12, S. 60) werden. Eine Baseline bezeichnet in DOORS eine zu einem bestimmten Zeitpunkt erfasste, nicht weiter bearbeitbare Kopie eines Moduls. Dabei erfasst eine Baseline immer genau ein Modul. Diese Baselines wiederum sind Teil eines Baseline-Sets. Mehrere Baselines können in einem solchen Set zusammengefasst werden. So kann Schritt für Schritt ein gesamtes Projekt erfasst werden. Welche Module innerhalb eines Baseline-Sets erfasst werden dürfen, wird durch eine *Baseline-Set-Definition* festgelegt, die einem bestimmten Ordner oder Projekt zugeordnet ist.

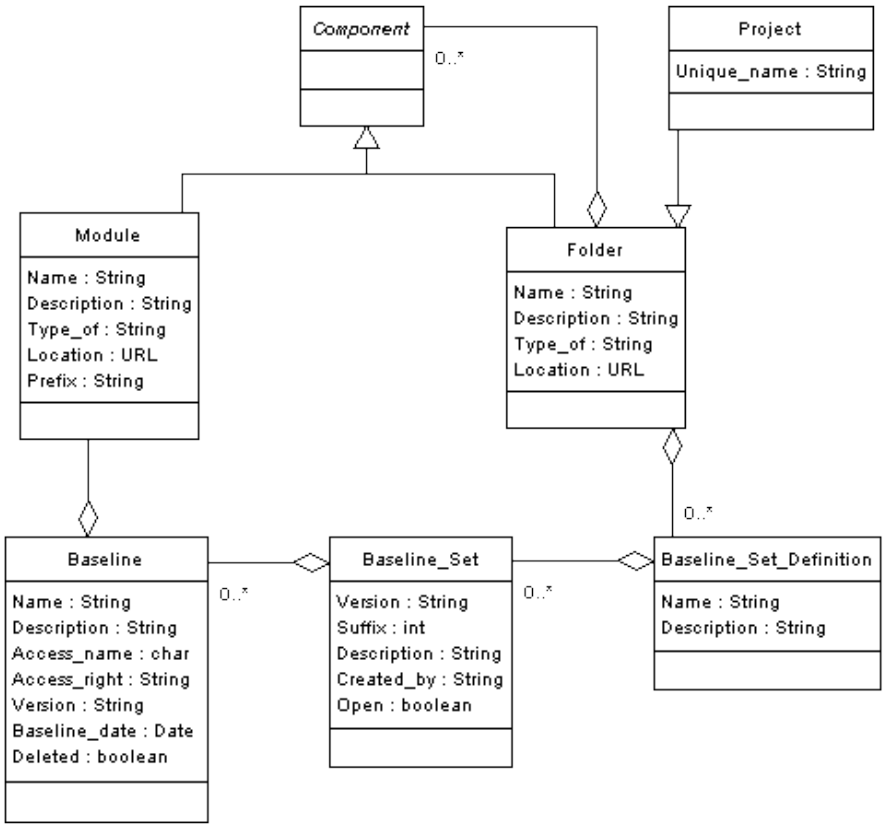


Abbildung 5.12: Baselining in DOORS

6 IBM Rational RequisitePro

IBM Rational RequisitePro ist Teil der *IBM Rational Suite*, eine integrierte Lösung für das Produkt-Lebenszyklus-Management. Mit einem Marktanteil von 39% stellt *IBM* den derzeit stärksten Konkurrent zu den Produkten von Telelogic dar. Als Teil der Rational Suite soll RequisitePro die Erstellung, Analyse und Verwaltung von Anforderungen sowie die Untersuchung von Anwendungsbeispielen unterstützen.

Ursprünglich wurde RequisitePro von der Firma *Requisite Inc.* entwickelt. Das Unternehmen aus Colorado wurde dann im Jahr 1996 von der *Rational Software Corporation* aus Massachusetts aufgekauft. IBM erwarb schließlich im Jahr 2003 die Rational Software Corporation und integrierte sie in die *IBM Software Group*. Hier wurde RequisitePro nun zum Teil der zuvor erwähnten Lebenszykluslösung, die von IBM heute unter dem Markennamen *Rational* vertrieben wird.

Das Tool lag als zeitlich unbeschränkte Vollversion in der Version Rational RequisitePro 2003 (2003.06.15.734.000) vor. Die entsprechende Lizenz wurde im Rahmen des akademischen Programms von IBM zur Verfügung gestellt. Als Projektdatenbank wurde MS Access 2002 (10.4302.4219) SP-2 verwendet.

Die folgenden Ergebnisse basieren ebenfalls auf einer eingehenden Untersuchung des Tools nach dem in Kapitel 4 beschriebenen Verfahren und Erkenntnissen aus der integrierten Hilfe des Tools [vgl. IBM06].

6.1 Arbeitsumgebung - Die RequisitePro Shell

Die Arbeitsoberfläche von RequisitePro, die so genannte *RequisitePro Shell*, erinnert in ihrer Aufmachung und Bedienung ebenfalls sehr an den MS Explorer. Am oberen Rand des Fensters befindet sich das Hauptmenü mit seiner Symbolleiste, die einen schnellen Zugriff auf die häufig benutzten Funktionen bietet. Der mittlere Teil des Fensters wird durch den Explorer dominiert. In diesem Bereich werden die *Artefakte* des Projekts in einer Verzeichnisbaumstruktur angezeigt. RequisitePro kennt vier verschiedene Arten von Artefakten, die in den entsprechenden Abschnitten (vgl. Abschnitt 6.2 und Abschnitt 6.3.5, S. 64 und 77) noch detailliert vorgestellt werden: *Dokumente*, *Anfor-*

derungen, *Sichten* und *Pakete*. Diese werden innerhalb des Explorers nach folgenden Kriterien hierarchisch angeordnet: Pakete (alphabetisch dem Namen nach), Dokumente (alphabetisch dem Namen nach), Sichten (nach Typ und dann alphabetisch innerhalb der Typen) und Anforderungen (nach Typ und dann nach Tag).

Der Explorer gewährleistet darüber hinaus den Zugriff auf die Artefakte: Durch Auswahl mit einem Klick der linken Maustaste lassen sich in einem kleinen Fenster unterhalb des Explorers detaillierte Informationen zu den jeweiligen Artefakten anzeigen. Durch einen Doppelklick mit der linken Maustaste lassen sich Artefakte wie Dokumente oder Sichten öffnen, Anforderungen können direkt bearbeitet werden. Über einen Klick mit der rechten Maustaste gelangt man in das Kontextmenü des entsprechenden Artefaktes. Ebenfalls bekannt sind die Drag-and-Drop-Mechanismen sowie die Auswahlmöglichkeiten mit Hilfe der Steuerungstaste. Am unteren Rand der RequisitePro Shell befindet sich eine Statusleiste, in welcher der Status des aktuell gewählten Artefakts und die Anzahl der in diesem Artefakt enthaltenen Anforderungen angezeigt werden.

Abbildung 6.1 gibt einen Überblick über die RequisitePro Shell:

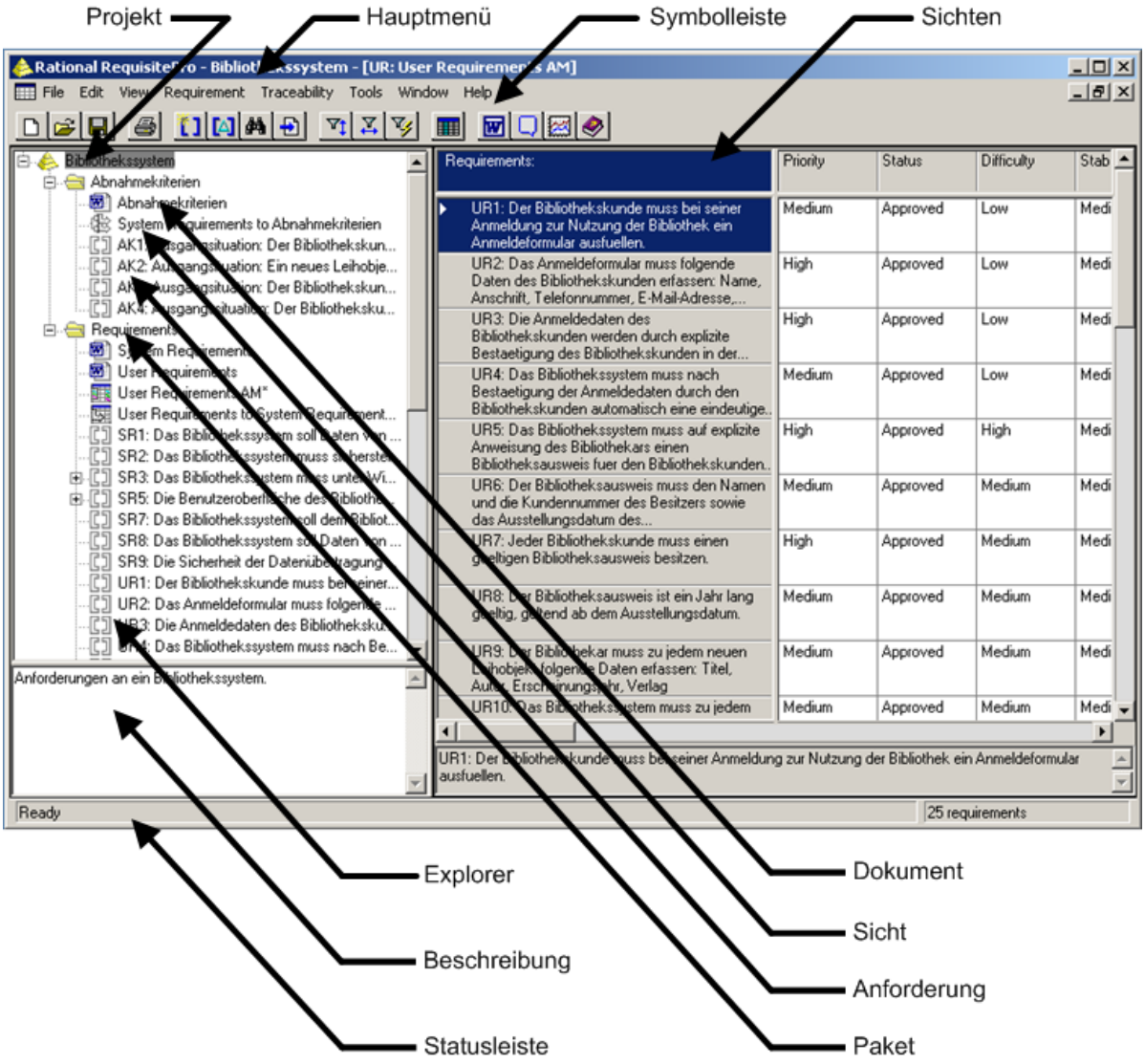


Abbildung 6.1: RequisitePro Shell

6.2 Informationsstruktur

In diesem Abschnitt wird die Informationsstruktur von RequisitePro näher untersucht. Die wichtigsten Begriffe und Konzepte werden an dieser Stelle kurz genannt, in Klammern steht jeweils die englische Bezeichnung des entsprechenden Begriffs:

- Projekte (*Projects*)
- Pakete (*Packages*)
- Dokumente (*Documents*)
- Anforderungen (*Requirements*)
- Attribute (*Attributes*)

6.2.1 Projekte

Ein Rational RequisitePro *Projekt* nutzt Anforderungsdokumente und eine Projektdatenbank, um Anforderungen zu erfassen und zu verwalten.

Ein neues Projekt kann „leer“ erstellt werden, es kann auf einem gesicherten Projekt aufbauen (vgl. Abschnitt 6.3.3, S. 74) oder es kann mit Hilfe von so genannten *Projektvorlagen* (englisch: *Project Templates*) erzeugt werden. Baut ein Projekt auf einer solchen Vorlage auf, übernimmt es sämtliche Eigenschaften dieser Vorlage. Standardmäßig wird RequisitePro mit drei verschiedenen Vorlagen geliefert, die folgende Eigenschaften besitzen:

Traditional Template. Diese Vorlage ist besonders für Projekte geeignet, die ein klassisches Anforderungsmanagement verfolgen.

Use-Case Template. Diese Vorlage ist besonders für Projekte vorgesehen, die den *Rational Unified Process (RUP)*[1] implementieren. Dadurch ist sie besonders angebracht in Projekten, in denen zusätzlich zu IBM Rational RequisitePro auch *IBM Rational Rose* und *IBM Rational ClearCase* genutzt werden. Diese unterstützen die modellgetriebene Entwicklung mittels *UML 2.0* (Rational Rose) sowie das *Konfigurationsmanagement* (Rational ClearCase).

Composite Template. Diese Vorlage kombiniert die Eigenschaften des „Traditional Templates“ mit denen des „Use-Case Templates“. Sie unterstützt durch ihre Do-

[1] „The Rational Unified Process is a software engineering process. It provides a disciplined approach to assigning tasks and responsibilities within a development organization. Its goal is to ensure the production of high-quality software that meets the needs of its end users within a predictable schedule and budget [vgl. Kru04, S. 17]“

kumenten- und Anforderungstypen sowohl klassische Dokumenten-basierte Techniken als auch Use-Case-basierte Verfahren.

Die genannten Vorlagen enthalten Pakete, Sichten und Dokumente, die laut IBM mit den Gestaltungsrichtlinien des Rational Unified Process übereinstimmen. Unter anderem enthalten sie Beschreibungen üblicher Artefakte, Begriffe und Rollen. Dadurch sollen sie in vielen Fällen den grundlegenden Anforderungen eines typischen Projekts gerecht werden. Die folgende Abbildung zeigt das Dialog-Fenster zur Auswahl einer Projektvorlage:

Abbildung 6.2: Auswahl einer Projektvorlage

Das Projekt *Bibliothekssystem*[a] wurde mit Hilfe des *Traditional Templates* angelegt. Dadurch kann auf bestimmte Dokumenten- und Anforderungstypen bereits zugegriffen werden. Dieses Template schien aufgrund des allgemeinen Charakters des Beispiels am ehesten auf das Projekt zuzutreffen. Verfügbare Dokumententypen beinhalten z.B. das *Stakeholder Request Document*, als Anforderungstyp steht unter anderem der *Stakeholder Request* zur Verfügung (zu Dokumententypen vgl. 6.2.3 auf S. 67, zu Anforderungstypen 6.2.4 auf S. 70).

[a] vgl. Abschnitt 4.1, S. 27

6.2.2 Pakete

Innerhalb eines Projekts werden Informationen in *Paketen* organisiert. Ein Paket stellt eine Einheit von verwandten *Artefakten* dar und ist damit eine Art Container für Dokumente, Anforderungen und Sichten. Ein Projekt besitzt immer ein *Root-Paket*, welches die Wurzel des Verzeichnisbaums im Explorer bildet. Alle darin enthaltenen Elemente werden darunter dargestellt.

Pakete können nach Belieben erstellt, umbenannt, verschoben und gelöscht werden. Artefakte (Dokumente, Anforderungen und Sichten), die in einer Beziehung zueinander stehen, können thematisch in Paketen gruppiert werden, die dann bestimmten Teilprojekten oder Teams eines Projekts zugeordnet werden.

Artefakte können z.B. mittels Drag-and-Drop zwischen verschiedenen Paketen verschoben werden. Ein bestimmtes Artefakt darf allerdings immer nur in genau einem Paket enthalten sein. Mit Hilfe von Paketen ist es möglich, eine hierarchisch gegliederte Informationsstruktur aufzubauen. Pakete können verschachtelt werden, d.h. ein Paket kann weitere Pakete enthalten. Abfragen über Anforderungen in einem bestimmten Paket geben als Ergebnis aber immer nur Anforderungen zurück, die in genau diesem Paket enthalten sind. Anforderungen aus untergeordneten Paketen werden nicht berücksichtigt.

Die Anforderungen an das Bibliothekssystem aus dem Beispiel[a] werden in einem Root-Paket namens *Bibliothekssystem* organisiert, welches weitere Pakete enthält, in diesem Fall *Abnahmekriterien* und *Requirements* (vgl. Abbildung 6.1, S. 63).

[a] vgl. Abschnitt 4.1, S. 27

6.2.3 Dokumente

In RequisitePro bezeichnet ein *Dokument* eine Sammlung von Anforderungen. Es beschreibt die Aufgaben und Ziele eines Projekts und kommuniziert den Projektverlauf. Jedes Dokument ist von einem bestimmten Typ (vgl. Abschnitt 6.2.3, S. 67). Die Anforderungen können in RequisitePro direkt in Dokumenten verwaltet werden. Beim Erstellen eines Dokumentes verknüpft RequisitePro es automatisch mit der *Projektdatenbank*, was die Aktualisierung von Information zwischen Dokumenten und Sichten beschleunigen soll.

Dokumente können entweder in RequisitePro Word (eine Erweiterung zu MS Word) erstellt werden oder ein MS Word-Dokument wird direkt in ein RequisitePro-Projekt importiert. Die Benutzeroberfläche von RequisitePro Word entspricht der von MS Word,

das Menü jedoch wurde angepasst, um RequisitePro mehr Kontrolle über ein Dokument zu erlauben. So werden zum Beispiel die Befehle *Speichern unter* und *Beenden* deaktiviert, um Konflikte mit RequisitePro zu vermeiden. Weiterhin gibt es eine RequisitePro Tool-Leiste, die zusätzliche Funktionalitäten zum Bearbeiten eines Dokumentes zur Verfügung stellt.

Nicht alle Dokumente müssen notwendigerweise Anforderungen enthalten. Prinzipiell kann jedes in Word verfasste Dokument mit einem Projekt verknüpft werden und so über den Explorer zur Verfügung gestellt werden, wenn das Projekt geöffnet wird.

Die Anforderungen an das Bibliothekssystem[a] werden in drei MS Word-Dokumenten zusammengefasst (*User Requirements*, *System Requirements*, *Abnahmekriterien*) und anschließend in die zuvor erstellten, entsprechend benannten Pakete importiert. Es besteht die Möglichkeit, entweder nur das Dokument als solches oder das Dokument sowie die enthaltenen Anforderungen gemeinsam zu importieren. Letzteres funktionierte aber in der Praxis nicht, weshalb zunächst die Dokumente als solche importiert wurden. Für das Erfassen der Anforderungen aus den Dokumenten heraus stehen weitere Funktionen zur Verfügung, die in den folgenden Abschnitten genauer erklärt werden.

[a] vgl. Abschnitt 4.1, S. 27

Dokumententypen

Ein *Dokumententyp* identifiziert eine bestimmte Art von Dokument, wie etwa eine *Use Case- oder Software Requirements-Spezifikation*. Der jeweilige Dokumententyp bestimmt die Formatvorlagen für Dokumente seines Typs, z.B. die Schriftart oder die verfügbaren Überschriften- und Absatzformate. Dadurch soll die Konsistenz unter Dokumenten des gleichen Typs gewährleistet werden.

Alle Dokumente des gleichen Typs haben die gleiche Dateierweiterung (z.B. filename.PRD). Die verfügbaren Dokumententypen werden bereits bei der Erstellung eines Projekts festgelegt und im weiteren Verlauf beim Anlegen von Dokumenten mit diesen assoziiert. Die neuen Dokumente erben dann die stilistischen und funktionellen Eigenschaften des Dokumententyps. RequisitePro bietet verschiedene Typen an, die an die speziellen Bedürfnisse eines Projekts angepasst werden können. Sobald Gestaltung und Informationsgehalt eines bestimmten Dokumententyps festgelegt wurden, kann dieser als Standard für ein Projekt verwendet werden.

Die in Abschnitt 6.2.1 beschriebenen Projektvorlagen enthalten die folgenden Dokumententypen:

- Vision Document
- Glossary
- Software Requirements Specification
- Modern Software Requirements Specification
- Supplementary Requirements Specification
- Use Case Specification
- Requirements Management Plan
- Stakeholder Request Document

Abbildung 6.3 zeigt die Auswahl der im Projekt „Bibliothekssystem“ vorhandenen Dokumententypen.

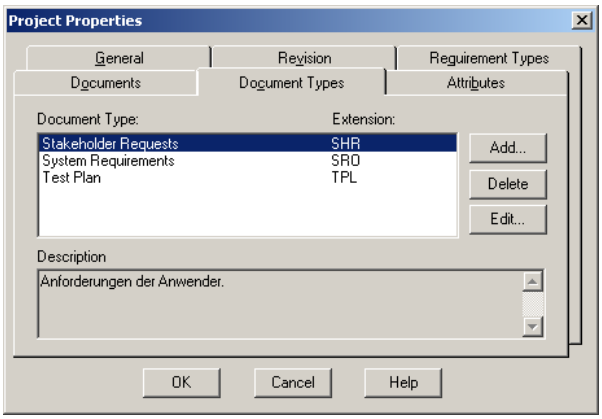


Abbildung 6.3: Auswahl eines Dokumententyps

Im Beispiel[a] werden die Dokumente mit den nachfolgenden Typen versehen. Diese wurden zum Teil neu angelegt, da sie in der benutzten Vorlage nicht vorgesehen waren.:

- Abnahmekriterien → Test Plan (Dateiname: Abnahmekriterien.TPL)
- System Requirements → System Requirements (Dateiname: System Requirements.SRO)
- User Requirements → Stakeholder Requests (Dateiname: User Requirements.SHR)

[a] vgl. Abschnitt 4.1, S. 27

6.2.4 Anforderungen

In RequisitePro wird eine *Anforderung* durch einen *Namen* und einen *Text* beschrieben. Diese Beschreibung kann durch *Attribute* ergänzt werden. Attribute versehen eine Anforderung mit benutzerdefinierten Eigenschaften (z.B. Kosten, Priorität und Status). Anforderungen werden entweder im Explorer, in einer Sicht oder direkt in einem Anforderungsdokument erstellt. Prinzipiell werden aber alle Informationen zu einer Anforderung in der Projektdatenbank gespeichert. Nachdem eine Anforderung erstellt wurde, hat der Anwender folgende Optionen:

- Er kann die Anforderung in eine Sicht oder ein Dokument verschieben oder kopieren.
- Er kann die Anforderung um Attribute ergänzen.
- Er kann die Anforderung mit anderen Anforderungen im Sinne der Traceability verknüpfen.

Eine Anforderung kann durch Spezialisierung mittels hierarchischer Beziehungen (englisch: *hierarchical relationships*) in konkreter formulierte Anforderungen unterteilt werden. Diese Kinder ergänzen ihre Eltern um weitere Anforderungsdetails.

Sämtliche Informationen zu einer Anforderung können in einer Sicht bzw. einem Dokument angepasst werden. Dort können die Anforderungen im Kontext mit weiteren Anforderungen gelesen und gegebenenfalls um weitere Informationen ergänzt werden, die sie unterstützen bzw. bestätigen.

Neben reinem Text können Anforderungen auch die folgenden Elemente enthalten:

- OLE Objekte
- Abbildungen
- Tabellen
- Anmerkungen
- Dokumente
- Listen

Die Anforderungen an das Bibliothekssystem[a] werden aus den zuvor importierten Anforderungsdokumenten (vgl. Abschnitt 6.2.3, S. 66) schrittweise eingefügt. Dazu werden die Dokumente in *RequisitePro Word* geöffnet. Die Anforderungen werden einzeln markiert und über den Button *New Requirement* dem entsprechenden Paket hinzugefügt. So werden beispielsweise die Anforderungen aus dem Dokument *User Requirements* dem Paket *Requirements* hinzugefügt. Die Nummerierung der Anforderungen übernimmt RequisitePro, wobei das Präfix des Indexes durch den entsprechenden *Anforderungstyp* bestimmt wird.

[a] vgl. Abschnitt 4.1, S. 27

Anforderungstypen

Ein *Anforderungstyp* beschreibt eine Vorlage für Anforderungen. Durch Anforderungstypen können Anforderungen klassifiziert werden, um eine effiziente Verwaltung dieser zu gewährleisten. Für einen Anforderungstyp werden verschieden Eigenschaften festgelegt, die dann für alle Anforderungen dieses Typs gelten. Dazu zählen Attribute, Darstellungsweise und Präfixe für den Index einer Anforderung. Für jedes Projekt können so die entsprechenden Anforderungstypen deklariert werden. Die vordefinierten Project Templates, die mit RequisitePro geliefert werden (vgl. 6.2.1, S. 64), enthalten die folgenden Anforderungstypen:

- Feature
- Glossary Item
- Requirements Management Plan Requirement
- Software Requirement
- Stakeholder Request
- Supplementary Requirement
- Use-Case

Abbildung 6.4 zeigt die Auswahl der im Projekt „Bibliothekssystem“ vorhandenen Anforderungstypen.

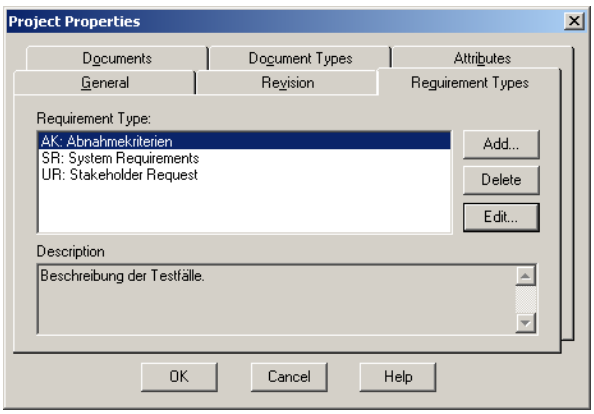


Abbildung 6.4: Auswahl eines Anforderungstypen

Folgende Anforderungstypen wurden für die Anforderungen an das Bibliothekssystem[a] ausgewählt:

- Abnahmekriterien → AK: Abnahmekriterien
- System Requirements → SR: System Requirements
- User Requirements → UR: Stakeholder Request

[a] vgl. Abschnitt 4.1, S. 27

6.2.5 Attribute

In RequisitePro werden Anforderungen durch ihren Anforderungstyp und ihre *Attribute* klassifiziert. Ein Attribut ergänzt eine Anforderung um Informationen, die helfen können, verschiedene Aktivitäten des Entwicklungsprozesses zu planen, zu kommunizieren und zu überwachen. Sie können folgende Informationen enthalten:

- den relativen Nutzen einer Anforderung
- die Implementierungskosten einer Anforderung
- die Priorität einer Anforderung
- das mit einer Anforderung verbundene Risiko und die Schwierigkeit bei ihrer Umsetzung
- die Beziehung einer Anforderung zu einer weiteren Anforderung

RequisitePro bietet eine Reihe von vordefinierten Attributen an, z.B. *Priorität* (hoch, mittel, niedrig), *Status* (vorgeschlagen, zugelassen, aufgenommen, gültig), *Kosten* und *Schwierigkeit* (hoch, mittel, niedrig). Darüber hinaus können Attribute den Bedürfnissen eines Projekts entsprechend erstellt werden.

Attribute können genutzt werden, um zu bestimmen, welche Anforderungen im nächsten *Release* implementiert werden sollten. So könnte man sich z.B. dafür entscheiden, im ersten Release nur diejenigen Anforderungen zu implementieren, die ein geringes Risiko bei mittlerer Schwierigkeit aufweisen. Wurden die entsprechenden Attribute korrekt zugewiesen, lassen sich die zutreffenden Anforderungen leicht ermitteln.

6.3 Funktionen

6.3.1 Ein- und Ausgabemöglichkeiten

Anforderungen werden in RequisitePro in Anforderungsdokumenten erstellt. Diese können direkt in RequisitePro erzeugt oder aus MS Word-Dokumenten importiert werden.

Beim *Import* sind verschiedene Einstellungen in MS Word vorzunehmen. So müssen die Rechtschreib- und Grammatikprüfung ebenso wie der automatische Seitenumbruch deaktiviert sein, da der Import der Dokumente sonst nicht funktioniert. Anforderungen können auch aus *CSV-Dateien* importiert werden. Diese können aus Daten jeder Datenbank, die selbst den Export von Anforderungen und Attributen in das CSV-Format unterstützt (z.B. *SQL Server*, *Oracle*, *MS Excel*, oder *MS Access*) erzeugt werden. Beim Import nach RequisitePro muss die CSV-Datei einen *Header* in einem bestimmten Format (<Tag>, „Requirement Text“, „Name“, <Attribute 1>, <Attribute 2>, <Attribute 3>, ...) enthalten, um korrekt interpretiert werden zu können.

Anforderungen können in RequisitePro auch direkt im Projekt erzeugt werden. Sie werden, ebenso wie Anforderungen aus den Anforderungsdokumenten, letztendlich in der Projektdatenbank gespeichert. Anforderungen eines Projekts müssen also nicht notwendigerweise mit einem bestimmten Anforderungsdokument verknüpft sein. Anforderungen können im RequisitePro Explorer oder in den verschiedenen Sichten (z.B. mittels der *Eigenschaften-Dialogbox*) erstellt werden.

Exportiert werden Anforderungen über die *Sichten* (vgl. Abschnitt 6.3.5, S. 77). Dabei sind folgende Einschränkungen zu beachten:

- Anforderungen aus einer Attribut Matrix oder einem Traceability Baum (vgl. 6.3.5, S. 77) können entweder in eine CSV-Datei oder in ein MS Word-Dokument exportiert werden.
- Anforderungen aus einer Traceability Matrix (vgl. 6.3.5, S. 77) können nur als CSV-Dateien exportiert werden.

Es werden jeweils nur genau diejenigen Informationen exportiert, die auch in der Sicht dargestellt werden (falls z.B. nur Name und Text der Anforderungen angezeigt werden, werden auch nur diese Werte exportiert).

6.3.2 Traceability

In RequisitePro bezeichnet *Traceability* einen methodischen Ansatz, Änderungen zu verwalten, indem man Anforderungen miteinander verknüpft, die in einer Abhängigkeitsbeziehung zueinander stehen. Sollte sich ein Endpunkt der Verbindung verändern, wird der entsprechende Link als verdächtig (englisch: *suspect*) markiert. Diese verdächtigen Links können in RequisitePro mit Hilfe der *Traceability Matrix* und des *Traceability Baums* (vgl. 6.3.5, S. 77) überwacht und kontrolliert werden.

Ein Link wird genau dann verdächtig, wenn Name, Text, Typ oder Attribute einer Anforderung modifiziert werden. Alle direkten Beziehungen zu dieser Anforderung hin oder von ihr weg werden von nun an als verdächtig betrachtet. Es besteht weiterhin die Möglichkeit, Attribute zu erstellen, welche, sobald sie modifiziert werden, ein solches Ereignis auslösen. Besteht die Beziehungen zwischen Anforderungen, die Verweise auf weitere Dateien enthalten, überwacht RequisitePro auch Änderungen an den entsprechenden Dateien.

Bei RequisitePro besteht die Möglichkeit, die automatische Überwachung der Traceability-Informationen (*Auto Suspect Command*) vorübergehend zu deaktivieren. Dies kann hilfreich sein, wenn z.B. lediglich die Rechtschreibung der Anforderungen überprüft werden soll, inhaltlich aber keine Änderungen vorgenommen werden. Es besteht aber in jedem Fall die Möglichkeit, einen Link manuell als verdächtig zu kennzeichnen.

Ein Konzept zur Verbesserung der Reusability von Anforderungen wird in RequisitePro mit der *Cross-Project Traceability* umgesetzt. Die Cross-Project Traceability ermöglicht es, direkte Verknüpfungen zwischen Anforderungen aus verschiedenen RequisitePro-Projekten zu erstellen und zu verwalten. So können Teams Anforderungen, die wiederholt in verschiedenen Projekten Anwendung finden, erneut verwenden, denn oft sind

bestimmte Anforderungen typisch für eine gewisse Art von Projekt. Diese Anforderungen können nun in eigenen Projekten zusammengefasst werden und so zu systemabhängigen Anforderungen in den entsprechenden Projekten verknüpft werden.

Sämtliche Links zwischen Anforderungen werden in einer Tabelle in der Projektdatenbank gespeichert, unabhängig von ihrer Richtung oder davon, ob die Links zwischen Anforderungen aus verschiedenen Paketen bestehen.

Die im Beispiel[a] vorgesehenen Links zwischen den Anforderungen an das Bibliothekssystem konnten vollständig übernommen worden. Ein Ergebnis dieses Vorgangs wird in Abbildung 6.5 dargestellt. Hier ist zu sehen, dass Anforderung UR0018 (in der Abbildung UR19) auf die Anforderungen SR0001, SR0010 und SR0011 verweist (in der Abbildung SR1, SR8, SR9).

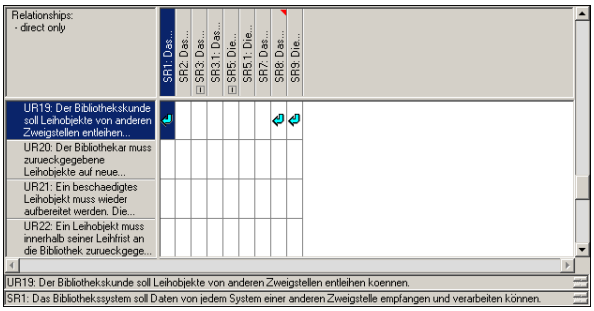


Abbildung 6.5: Verknüpfung von Anforderungen in RequisitePro

[a] vgl. Abschnitt 4.1, S. 27

6.3.3 Versionsmanagement

In RequisitePro können verschiedene Versionen eines Projekts in Archiven oder Baselines erfasst werden.

Archive können direkt aus RequisitePro heraus erstellt werden, um Baselines bilden zu können muss zusätzlich das *Unified Change Management (UCM)* implementiert werden[2]. Allerdings kann nur bei Archiven die Cross-Project Traceability (vgl. 6.3.2, S. 73) erhalten werden. Ist das zu archivierende Projekt mit weiteren Projekten verknüpft, sollten diese ebenfalls archiviert werden, um die Konsistenz der Links zu gewährleisten. In jedem Projekt, das in diesem Zusammenhang archiviert wurde, werden nun die alten

[2] UCM ist ein von Rational vordefinierter Prozess, der die Softwareentwicklung in Bezug auf das Konfigurationsmanagement unterstützt, in dem er die Verwaltung von Artefakten organisiert und automatisiert. Das UCM wird durch Rational ClearCase unterstützt.

Verknüpfungen zu den externen Projekten gelöst. Anschließend wird die Verknüpfung zu dem aktuellen Archiv des entsprechenden Projekts wiederhergestellt.

Sowohl mit Hilfe eines Archivs als auch einer Baseline kann ein neues Projekt aufgesetzt werden. Dies ist z.B. dann hilfreich, wenn ein neues Release eines Projekts auf einer stabilen Version des vorangegangenen Releases aufbauen soll. Erstellt man auf Basis der Baseline ein neues Projekt, werden jedoch die Links zu weiteren Projekten (Cross-Project Traceability), Diskussionen sowie Assoziationen mit Rational ClearCase, Rational Rose und Rational XDE nicht von dem originalen Projekt übernommen.

Wird ein Projekt archiviert, legt RequisitePro zunächst ein Verzeichnis an, in dem die entsprechenden Daten abgelegt werden. In dieses Verzeichnis werden die Datenbank, die Dokumente sowie alle weiteren zugehörigen Dateien des Projekts kopiert. Wird eine Enterprise Datenbank (z.B. SQL Server oder Oracle) als Projektdatenbank genutzt, wird die Datenbank nicht automatisch kopiert. Der entsprechend berechtigte Datenbankadministrator muss in diesem Fall manuell ein Backup der Datenbank erstellen. Bildet man dagegen eine Baseline in RequisitePro, wird eine Kopie des Projekts inklusive der Datenbank erstellt.

Es besteht die Möglichkeit, Tools aus dem Bereich des Konfigurationsmanagement zu nutzen, um den Prozess der Archivierung zu unterstützen, wie z.B. Rational ClearCase, PVCS Version Manager oder MS Visual SourceSafe.

Schließlich werden Änderungen an Projekten, Anforderungen und Dokumenten auch in einer Historie festgehalten. Jede Änderung wird mit einer Revisionsnummer versehen, dazu werden ein Label, das Datum, die Uhrzeit, der Autor und eine Beschreibung der Änderung festgehalten. Die Historie hält alle Änderungen ab dem Zeitpunkt der Erstellung eines Elementes in RequisitePro fest.

6.3.4 Veränderungsmanagement

Mit Hilfe des *Discussion-Features* kann in RequisitePro ein einfaches *Veränderungsmanagement* implementiert werden. Diese Funktion ermöglicht es, eine Diskussuion zu einem bestimmten Thema anzuregen und Fragen und Kommentare an Diskussionsteilnehmer zu adressieren. Eine solche Diskussion kann sich auf ein oder mehrere Anforderungen im Speziellen oder auf das gesamte Projekt als solches beziehen.

Eine Diskussion ist prinzipiell von allen Projektmitarbeitern einsehbar. Der Autor einer Diskussion (oder der Administrator des entsprechenden Projekts) kann die Gruppe der Diskussionsteilnehmer jedoch auf ein oder mehrere Anwendergruppen beschränken. In diesem Falle können alle Projektmitarbeiter die Diskussionsbeiträge weiterhin lesen, nur die zugelassenen Diskussionsteilnehmer jedoch können neue Antworten verfassen.

Optional kann eine Diskussion so konfiguriert werden, dass die Teilnehmer bei neuen Beiträgen durch eine E-Mail benachrichtigt werden.

Hat die Diskussion zu einem Konsens unter den Teilnehmern geführt, kann ihr Status durch den Autor der Diskussion als geschlossen (englisch: *closed*) markiert werden. Weitere Attribute einer Diskussion beziehen sich auf ihren Autor, Datum und Zeit der Erstellung und ihre Priorität.

Abbildung 6.6 zeigt das Diskussions-Dialogfenster. Von hier aus können Diskussionen erstellt, bearbeitet und beantwortet werden, die entsprechenden Rechte vorausgesetzt. Im unteren Teil des Fensters werden das Thema der Diskussion, eventuell beteiligte Anforderungen und zugelassene Teilnehmer angezeigt. Anforderungen, die von der Diskussion betroffen sind, werden in der RequisitePro Shell durch ein kleines rotes Dreieck im Verzeichnisbaum markiert. Über dieses Dreieck kann auch direkt auf die Diskussion zugegriffen werden.

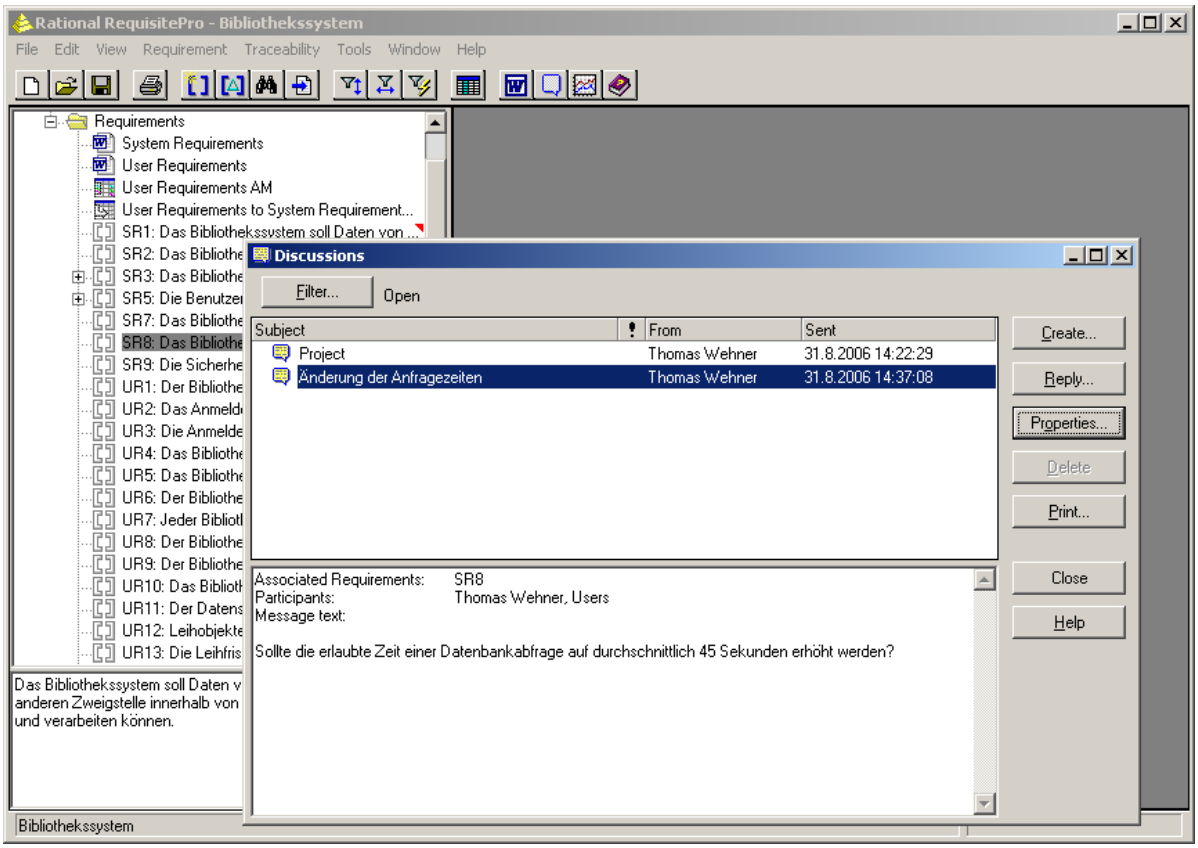


Abbildung 6.6: Diskussionen in RequisitePro

Darüber hinaus besteht in RequisitePro die Möglichkeit, das bereits erwähnte UCM als Teil von Rational ClearCase bzw. Rational ClearQuest zu implementieren. Rational

ClearCase kann auf Basis des Unified Change Managements die wesentlichen Elemente eines aktivitätsbasierten Konfigurationsmanagements einrichten. Rational ClearQuest dagegen integriert Rational ClearCase und bildet zusammen mit diesem Produkt eine Komplettlösung für das Management von Softwarekonfigurationen. Rational ClearQuest stellt somit eine vollständige Fehler- und Änderungsverfolgung über den gesamten Entwicklungszyklus zur Verfügung.

6.3.5 Analysemöglichkeiten

Sichten (Views) erlauben in RequisitePro einen Blick auf die Projektdatenbank. Sie präsentieren damit Informationen über ein Projekt, ein Dokument oder eine Anforderung. Die Darstellung kann grafisch in Form einer Tabelle (Matrix) erfolgen oder in einem Verzeichnisbaum. In den Sichten können Anforderungen, ihre Attribute und ihre Beziehungen untereinander dargestellt und verwaltet werden. Darüber hinaus beinhalten Sichten Abfragefunktionen, mit Hilfe derer sich die Anforderungen und ihre Attribute mittels bestimmter Kriterien filtern und sortieren lassen.

Es gibt drei verschiedene Sichten:

Die Attribut Matrix stellt alle Anforderungen eines bestimmten Typs dar. Die Anforderungen werden in Reihen aufgelistet, ihre Attribute in den dazugehörigen Spalten.

Die Traceability Matrix stellt die Verknüpfungen zwischen Anforderungen zweier verschiedener Typen dar.

Der Traceability Baum stellt die Verknüpfungskette zu oder von einem bestimmten Typ von Anforderung dar.

Die nachfolgenden Abbildungen zeigen der Reihe nach die Attribut Matrix, die Traceability Matrix und den Traceability Baum.

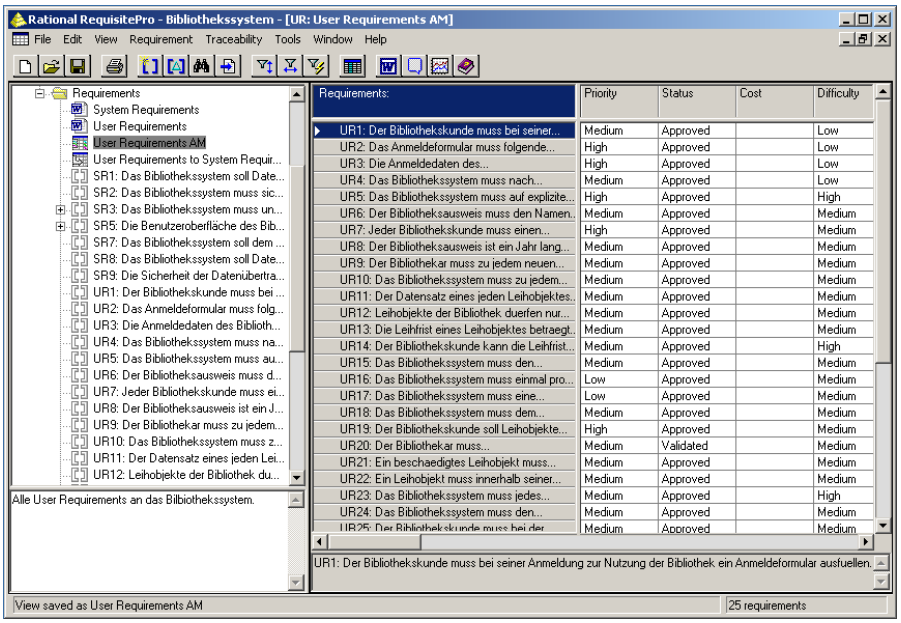


Abbildung 6.7: Attribut Matrix

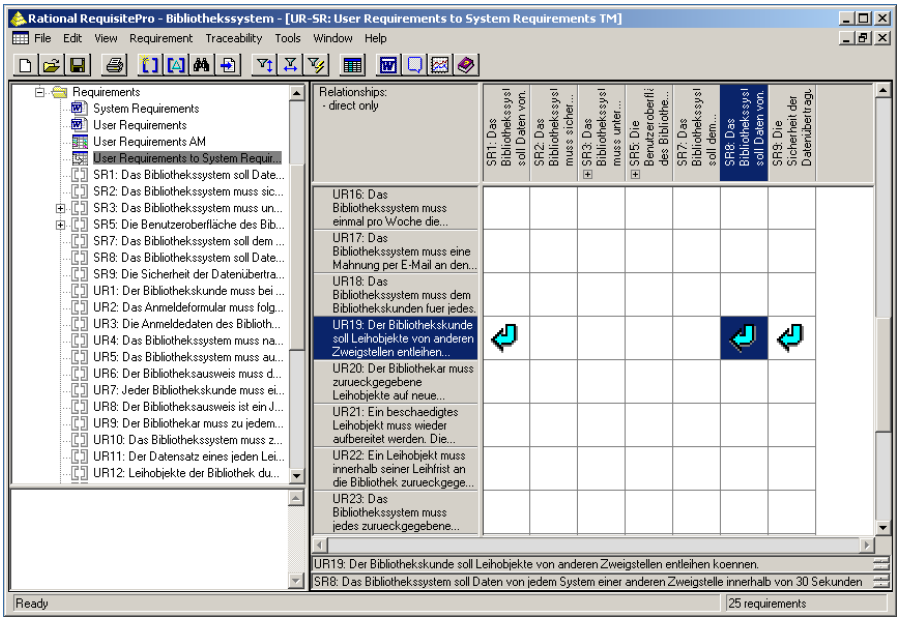


Abbildung 6.8: Traceability Matrix

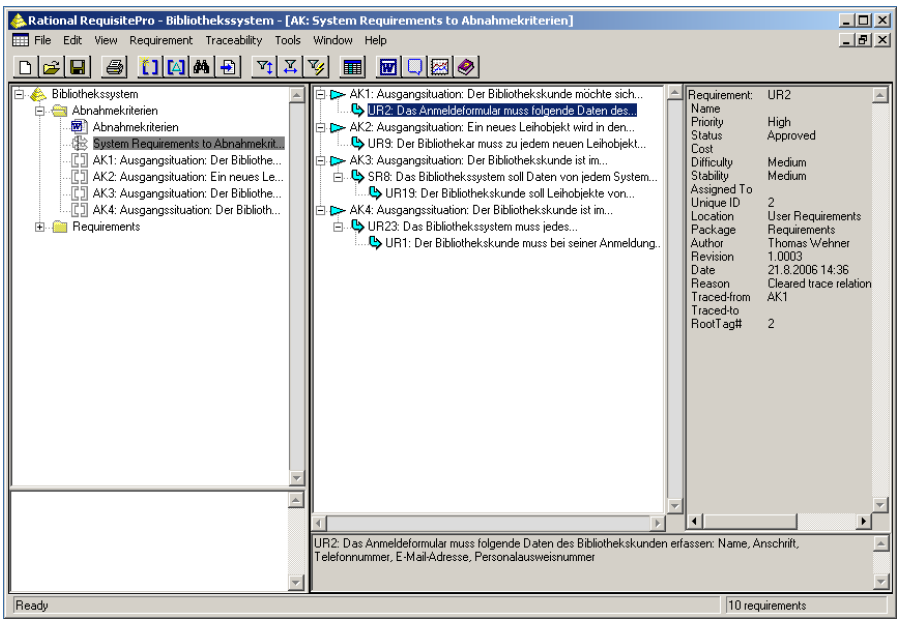


Abbildung 6.9: Traceability Baum

6.4 Sonstige Aspekte

6.4.1 Administrative Aspekte

IBM Rational RequisitePro gestattet es, den Level der Sicherheit and die Bedürfnisse eines Projekts anzupassen. Die so genannte *Project Security*, die Benutzerverwaltung, kann aktiviert oder deaktiviert (Standard) werden. Über die Benutzerverwaltung soll der unautorisierte Zugriff auf Projekte, Dokumente oder Anforderungen verhindert werden. Ist sie aktiviert, werden alle Änderungen an einem Projekt mit dem Namen des entsprechenden Anwenders sowie weiteren Daten (Datum etc.) vermerkt.

Besonders bei Projekten mit vielen Anwendern ist die Benutzerverwaltung von entscheidender Bedeutung. In RequisitePro können Benutzerrechte an einzelne Anwender oder an Gruppen von Anwendern vergeben werden. Nur RequisitePro Administratoren oder Mitglieder einer Gruppe mit *Project Security Permissions* haben das Recht, Benutzerrechte zu vergeben.

Es gibt drei Standard Gruppen:

Administratoren (englisch: *Administrators group*) besitzen volle Rechte für ein Projekt. Sie können die Projektstruktur verändern, Daten sowie Sichten erstellen, ändern und löschen. Außerdem dürfen sie Benutzern bzw. Benutzergruppen Rechte erteilen und entziehen. Benutzer können der Administratorengruppe hinzugefügt oder aus ihr entfernt werden, die Gruppe selbst kann nicht gelöscht werden. Auch die Rechte dieser Gruppe können nicht beeinflusst werden.

Benutzer (englisch: *Users group*) dürfen standardmäßig Dokumente und Anforderungen lesen, Sichten erstellen und an Diskussionen teilnehmen.

Gelöschte Benutzer (englisch: *Deleted Users group*) wurden aus anderen Gruppen entfernt. Sie haben keinerlei Rechte und können sich nicht in RequisitePro anmelden. Diese Gruppe wird vom Administrator des entsprechenden Projekts verwaltet und dient lediglich Dokumentationszwecken.

Die Rechte, die den einzelnen Benutzern bzw. Gruppen von Benutzern vergeben werden können, gliedern sich in drei Gruppen:

Projektbezogene Rechte. Diese Gruppe von Rechten bezieht sich auf die Rechte, Objekte (wie etwa Dokumente oder Anforderungen) innerhalb eines Projekts zu erstellen oder die Rechte von Benutzergruppen zu ändern.

Dokumententyp- und Anforderungstypbezogene Rechte. Diese Gruppe von Rechten bezeichnet klassische Zugriffsrechte wie Lesen oder Erstellen in Bezug auf das Dokumenten- und Anforderungsebene. Die Berechtigungen schließen auch die entsprechenden Attribute und Typen mit ein.

Traceabilitybezogene Rechte. Diese Gruppe von Rechten umfasst neben den klassischen Zugriffsrechten wie Lesen oder Erstellen auch Rechte, die es Benutzern gestatten, Links als verdächtig zu markieren (*Mark Suspect*) bzw. diese Markierung zu entfernen (*Clear Suspect*).

6.4.2 Customizing

IBM Rational RequisitePro ermöglicht es dem Anwender, das Menü des Tools in einem bestimmten Rahmen anzupassen. In einem beliebigen Texteditor muss dazu zunächst eine so genannte *RequisitePro Menu File* erstellt werden, die den auszuführenden Code für den Befehl enthält. Ein entsprechendes Template samt Regeln zur Syntax ist beispielsweise in der Hilfe zu RequisitePro zu finden. Die Datei wird mit der Erweiterung .txt oder .mnu gespeichert und über die Funktion *Tools* → *Add-ins* in das Menü eingebunden. Eigene Befehle können in den folgenden Menüs eingefügt werden: *File, Edit, View, Requirement, Traceability, Tools* und *Help*. Diese Befehle können dann genutzt werden,

um beispielsweise externe Tools wie z.B. einen E-Mail-Client oder einen Texteditor zu starten, oder eine bestimmte Datei mit dem entsprechenden Tool zu öffnen.

6.5 Klassendiagramm

In diesem Abschnitt soll das Klassendiagramm von RequisitePro (vgl. Abbildung 6.10, S. 82) vorgestellt werden. Das Diagramm basiert im Wesentlichen auf den Erkenntnissen der vorangegangenen Untersuchungen.[3]

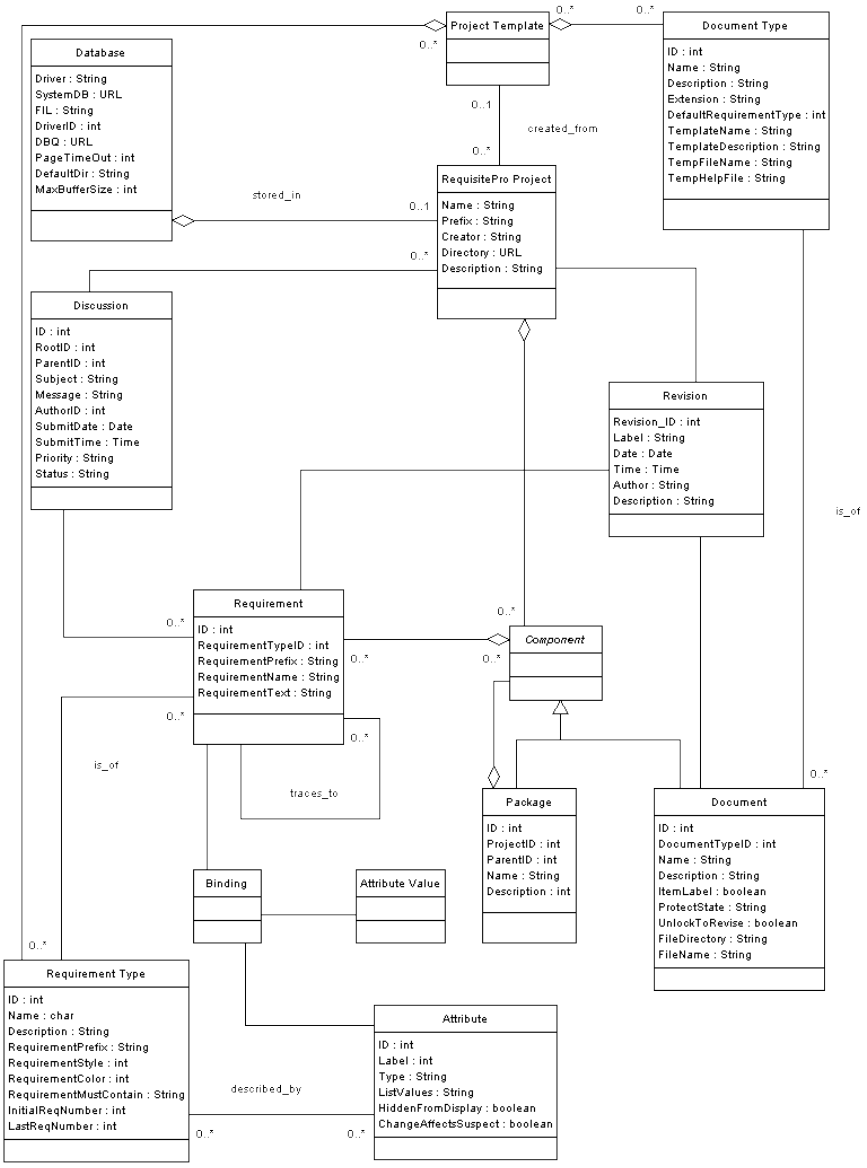


Abbildung 6.10: RequisitePro-Klassendiagramm

[3]Zwar waren die Attribute der Klassen aufgrund der MS Access-Datenbank im Gegensatz zu den anderen Tools größtenteils bekannt, es wurden bei der Modellierung dieses Diagramms jedoch nur die themenbezogen relevanten Attribute übernommen.

Ähnlich dem Modell von DOORS (vgl. Abbildung 5.8, S. 55) wird auch in diesem Klassendiagramm die Projektstruktur durch ein Kompositum beschrieben (vgl. Abbildung 6.11, S. 83). In diesem Falle schließt es die drei Klassen Component (deutsch: Komponente), Package (deutsch: Paket) und Document (deutsch: Dokument) mit ein. Mit Hilfe von Paketen werden in einem RequisitePro-Projekt Informationen strukturiert. Sie dürfen weitere Pakete sowie Dokumente enthalten.

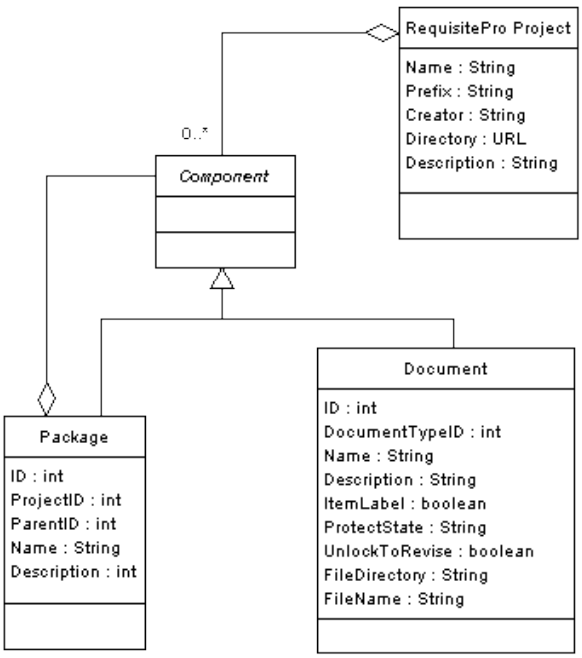


Abbildung 6.11: Composite-Struktur in RequisitePro

Ein Projekt (RequisitePro Project) kann bei RequisitePro in verschiedenen Typen von Datenbanken gespeichert werden (vgl. Abbildung 6.12, S. 83). Diese Möglichkeit unterscheidet es von Tools wie DOORS, die über eine eigene, proprietäre Datenbank verfügen.

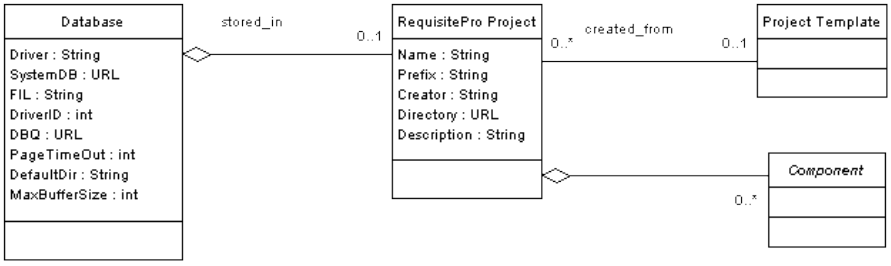


Abbildung 6.12: Projektdatenbank in RequisitePro

Ein Projekt kann mit Hilfe einer Vorlage (Project Template) erstellt werden. Eine Vorlage besitzt verschiedene Dokumententypen (Document Type) und Anforderungstypen (Requirement Type). Mit Hilfe dieser Typen bietet eine Vorlage Gestaltungsrichtlinien für ein bestimmte Objekte eines Projekts an. So können Informationen einheitlich organisiert werden.

Ein Dokumententyp beschreibt Eigenschaften von Dokumenten (Document), z.B. deren Dateierweiterung oder den standardmäßig benutzten Anforderungstyp. Ein Dokument ist also von einem bestimmten Typ. Ähnliches gilt für Anforderungstypen (Requirement Type). Alle Anforderungen (Requirement) eines bestimmten Anforderungstyps teilen sich dessen Eigenschaften, z.B. einen Präfix bzw. bestimmte Erscheinungsmerkmale wie Formateigenschaften oder Farbe (vgl. Abbildung 6.13, S. 84).

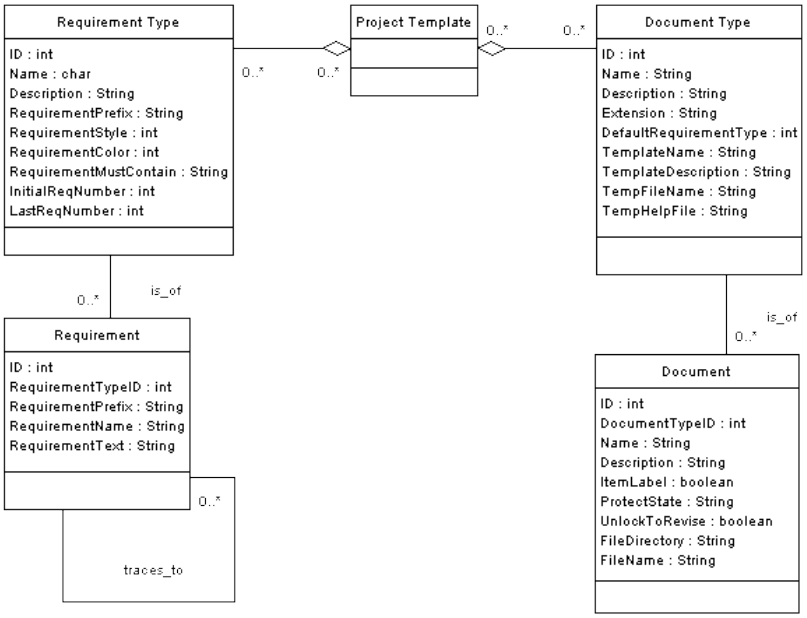


Abbildung 6.13: Dokumenten- und Anforderungstypen

Anforderungen (Requirements) können entweder in Paketen (Package) oder Dokumenten (Document) gespeichert werden. Prinzipiell aber werden sich zusammen mit allen anderen Informationen in der Projektdatenbank eines Projekts gespeichert.

Auch in RequisitePro können eigene Attribute erstellt werden. Diese werden immer für einen bestimmten Anforderungstyp erstellt. Da der eigentliche Wert eines Attributs (Attribute Value) aber nur für eine bestimmte Anforderung gilt, wird sich hier einer Klasse Binding beholfen, welche die drei Klassen Requirement, Attribute und Attribute Value in Beziehung setzt.

Abbildung 6.14 stellt diese Struktur dar:

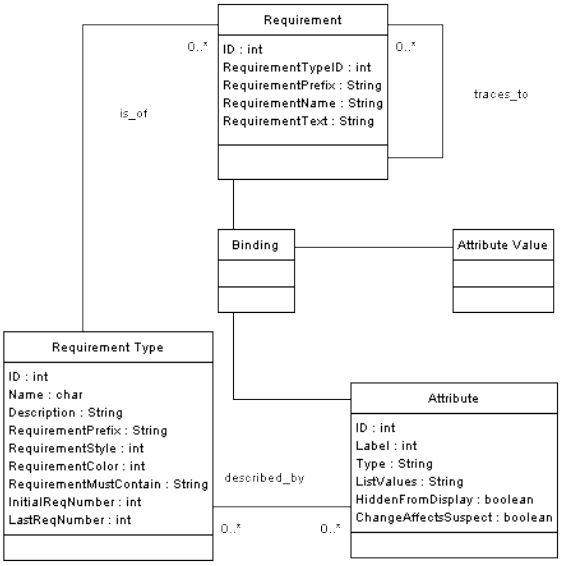


Abbildung 6.14: Attribute in RequisitePro

Änderungen an Projekten (RequisitePro Project), Dokumenten (Document) und Anforderungen (Requirement) werden in einer Historie (Revision) gespeichert. Die Historie speichert sämtliche relevanten Daten, z.B. den Autor sowie Datum und Zeitpunkt der Änderung. Damit können Änderungen an den genannten Objekten jederzeit nachverfolgt werden.

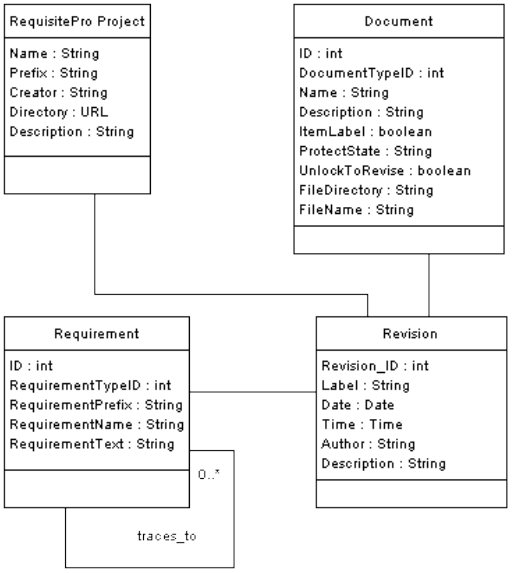


Abbildung 6.15: Änderungsverfolgung in RequisitePro-Projekten

7 RETH

RETH (Requirements Engineering Through Hypertext) bezeichnet zunächst eine Methode für das Requirements Engineering, welche objektorientierte Techniken sowie Hypertext für die Erstellung und Verwaltung von Anforderungen sowie deren Beziehungen untereinander einsetzt. RETH ist darüber hinaus auch ein Tool, welches diese Methode unterstützt.

Anforderungen, die gewöhnlich in natürlicher Sprache (nicht-formal) vorliegen, werden durch *Hypertext* semi-formal[1] dargestellt. Aber auch formale Strukturen können durch diese Methode erfasst werden. Der Hypertext wird in RETH dabei nach *objektorientierten Grundsätzen* organisiert. Anforderungen sowie Entitäten der entsprechenden Systemdomäne werden als Objekte dargestellt. Dadurch wird es möglich, objektorientierte Prinzipien auf die Anforderungen anzuwenden. Durch diese einheitliche Darstellung ist es möglich, Anforderungen mit genau jenen Domänenobjekten zu verbinden, über die sie Aussagen treffen. Darüber hinaus wird das Verständnis der Anforderungen gefördert, da die Definitionen der einzelnen Domänenobjekte jederzeit verfügbar sind.

Entwickelt wurden Methode und Tool von Prof. Dr. Hermann Kaindl von der Technischen Universität Wien für dessen damaligen Arbeitsgeber, die *Siemens AG Österreich*. Das Tool ist nicht kommerziell erhältlich.

Die Version des Tools, welche im Rahmen dieser Evaluation zur Verfügung steht, ist RETH 2.5. Das Tool lag als zeitlich unbeschränkte Vollversion vor, die auf Nachfrage bei Prof. Dr. Hermann Kaindl durch die Siemens AG Österreich zur Verfügung gestellt wurde.

Die folgenden Ergebnisse basieren wie in den vorangegangenen Kapiteln auf einer intensiven Untersuchung des Tools nach dem in Kapitel 4 beschriebenen Verfahren und Erkenntnissen aus der integrierten Hilfe des Tools [vgl. Sie06].

[1] Semi-formal bedeutet im Zusammenhang mit formalen Methoden *nicht vollständig semantisch formalisiert.*

7.1 Arbeitsumgebung - Der RETH Explorer

Auch die Arbeitsoberfläche von RETH, der *RETH Explorer* (vgl. Abbildung 7.1, S. 88), erinnert in seiner Aufteilung zunächst an den MS Explorer. Der obere Fensterrand wird von je einer Menü- und Symbolleiste dominiert, über welche sämtliche Funktionen des Tools abgerufen werden können.

Im linken Teil des Fensters wird die Struktur des geladenen Projekts in einem Baum abgebildet, welcher die Hierarchie der so genannten Hypertext-Knoten darstellt. In RETH wird dieser als *Taxonomie-Baum* bezeichnet. Jeder Knoten dieses Baums repräsentiert nun entweder eine Klasse (erkennbar am Symbol „K“) oder eine Instanz („I“). Wählt man mit einem Klick der linken Maustaste einen Knoten aus, wird dessen Inhalt im rechten Teil des Fensters angezeigt. Über einen Klick mit der rechten Maustaste gelangt man ins Kontextmenü des jeweiligen Objektes.

Am unteren Fensterrand schließlich lässt sich der *Versions-Navigator* ein- und ausblenden, über den bei aktiviertem Versionsmanagement leichter auf Versionen eines Projekts zugegriffen werden kann.

Abbildung 7.1 stellt den RETH Explorer im Gesamtüberblick dar:

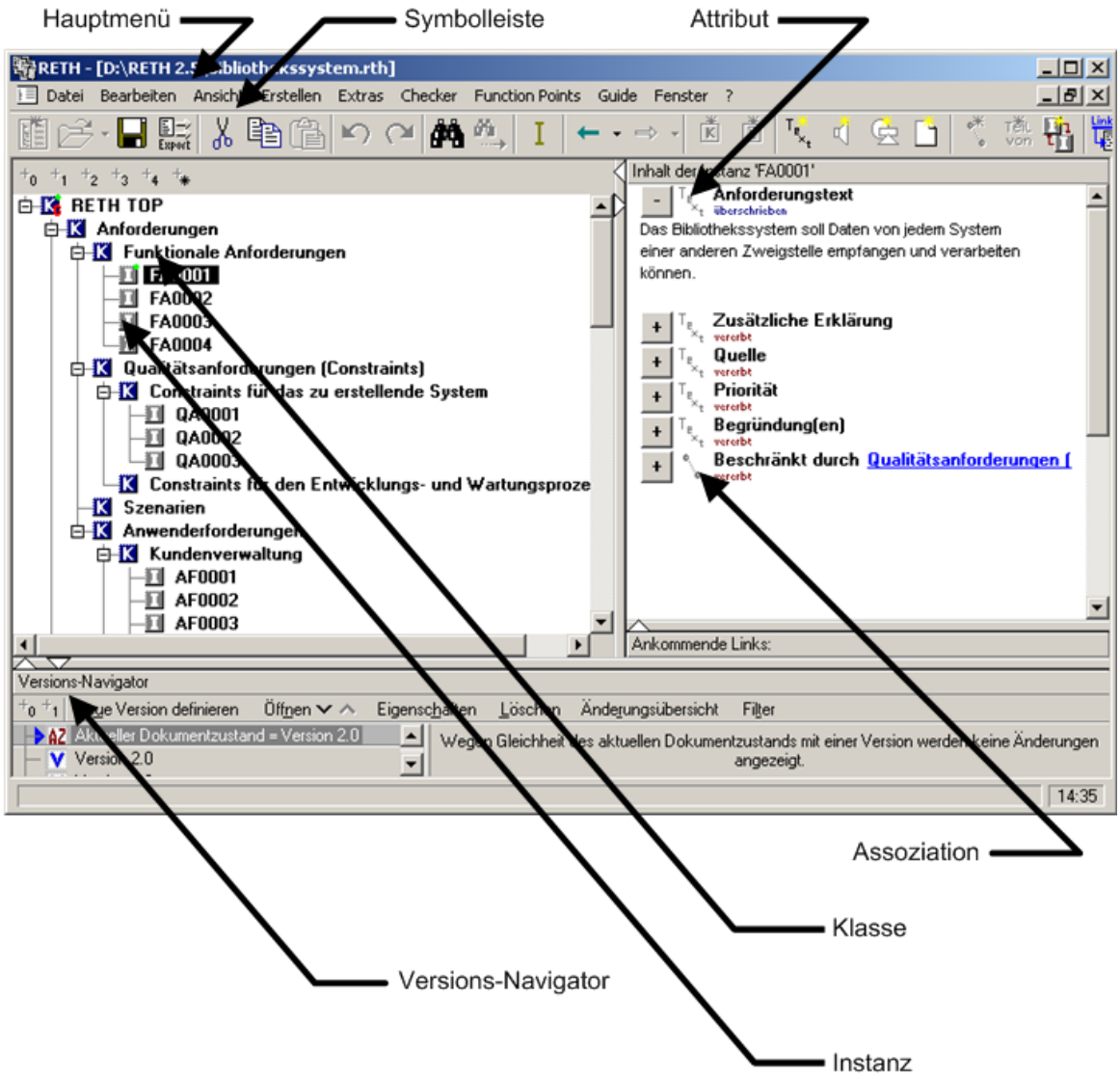


Abbildung 7.1: RETH Explorer

7.2 Informationsstruktur

In den folgenden Abschnitten wird die Informationsstruktur von RETH beschrieben. Wie sich zeigen wird, orientieren sich die verschiedenen Elemente per Definition zum Teil sehr nahe an der objektorientierten Programmierung. Die wichtigsten Begriffe und Konzepte werden an dieser Stelle kurz aufgelistet, bevor sie im Detail beschrieben werden:

- Hypertext-Komponenten
- Klassen
- Instanzen
- Attribute
- Assoziationen
- Aggregationen

7.2.1 Hypertext-Komponenten

Ein großer Teil der Informationsstruktur basiert auf *Hypertext-Komponenten*. In RETH wird zwischen zwei Arten solcher Komponenten unterschieden: (Hypertext-)Knoten und (Hypertext-)Links.

Ein *(Hypertext-)Knoten* oder Knoten wird in RETH entweder durch eine Klasse oder eine entsprechende Instanz repräsentiert. Er fasst direkt zusammengehörige Informationen bezüglich des zu beschreibenden Systems innerhalb von *Partitionen* zusammen. Diese stellen die interne Struktur eines Hypertext-Knoten dar. Alle Partitionen eines Knoten decken gemeinsam den ganzen Knoten ab, wobei sich die Partitionen nicht überschneiden, um redundante Informationen zu vermeiden. Jeder Knoten in RETH besitzt mindestens eine Partition (standardmäßig die so genannte Partition *Main*). RETH unterscheidet anhand der enthaltenen Informationen zwischen drei verschiedenen Typen von Partitionen: Attribute, Assoziationen oder Aggregationen. Bei den Attributen können in Abhängigkeit der enthaltenen Informationen Text-, Audio-, Bild- und Datei-Attribute unterschieden werden.

Ein *(Hypertext-)Link* oder Hyperlink bringt (Hypertext-)Knoten in Beziehung zueinander. Hyperlinks sind in RETH grundsätzlich *gerichtet* und haben einen Text als Quelle, den so genannten *Link-Anker*. Ein Hyperlink verweist immer zu einer bestimmten Partition des Zielknotens und ist *bidirektional* traversierbar. In besonderer Weise unterstützt RETH Glossar-Links, welche in das Glossar verweisen oder Glossar-Einträge miteinander verknüpfen. Diese Links können durch RETH manuell, (semi-)automatisch oder vollautomatisch erstellt werden.

7.2.2 Klassen

Eine *Klasse* fasst in RETH die typischen oder wesentlichen Eigenschaften, Beziehungen und Verhaltensweisen zusammen, die ihren Instanzen gemeinsam sind. Mit ihrer Hilfe lässt sich innerhalb eines Projekts eine hierarchische Struktur von Informationseinheiten bilden. Eine Klasse kann beliebig viele Instanzen enthalten, darf aber auch weitere Unterklassen besitzen, an welche sie ihre Eigenschaften vererbt. Standardmäßig sind in einem neuen Projekt bereits drei Klassen definiert:

Anforderungen. In dieser Klasse werden sämtliche Anforderungen eines Projekts verwaltet. Die Klasse *Anforderung* wird durch folgende Attribute charakterisiert: Anforderungstext, Zusätzliche Erklärung, Quelle, Priorität, Begründung(en). Eine Anforderung wird in RETH also beschrieben durch einen Anforderungstext, welcher die eigentliche Anforderung formuliert, zusätzliche Erklärungen sowie eine Quellenangabe, die beispielsweise auf ein bestimmtes Anforderungsdokument verweist. Darüber hinaus kann einer Anforderung eine Priorität zugewiesen werden. Dieses Attribut ist allerdings, wie alle anderen auch, ein reines Text-Attribut, so dass hier ein beliebiger Text eingegeben werden kann, was eine sinnvolle Nutzung erschwert. Gleiches gilt für die Begründung zur Aufnahme einer Anforderung in ein Projekt. Die Klasse *Anforderung* besitzt außerdem drei Unterklassen: *Funktionale Anforderungen, Qualitätsanforderungen („Constraints“)* und *Szenarien.*

Funktionale Anforderungen. In dieser Klasse werden Ursache und Wirkung von Anforderungen beschrieben. Eine funktionale Anforderung im Sinne von RETH setzt sich zusammen aus einem notwendigen oder gewünschten Resultat und einem bestimmten Verhalten, um dieses Resultat zu erhalten [vgl. Kai97, S. 328]. Diese Klasse erweitert die Klasse *Anforderung* durch eine Assoziation zu der Klasse *Qualitätsanforderungen („Beschränkt durch“).*

Qualitätsanforderungen. In dieser Klasse werden die nicht-funktionalen Anforderungen an ein System dokumentiert. Diese Klasse erweitert die Klasse *Anforderung* durch Assoziationen zu den Klassen *Funktionale Anforderungen („Beschränkt (Funktion)“)* und *Szenarien („Beschränkt (Ablauf)“).*

Szenarien. In dieser Klasse werden Interaktionen zwischen dem zu entwickelnden System und seiner Umgebung beschrieben. Diese Klasse erweitert die Klasse *Anforderung* durch eine Assoziation zu der Klasse *Ziele („Erreicht“).*

Domänenmodell. Mit Hilfe des *Domänenmodells* soll die Umgebung des zu entwickelnden Systems beschrieben werden, um so ein besseres Verständnis für die Anforderungen an dieses System zu entwickeln. Es enthält genau jene Entitäten der Systemumgebung, die im Zusammenhang mit diesem System von Bedeutung sind. Aufgrund der objektorientierten Betrachtungsweise von RETH werden diese Entitäten als Klassen modelliert. Beziehungen zwischen den Entitäten werden durch Assoziationen und Aggregationen ausgedrückt. Zusätzlich zu dieser statischen Darstellung der Domäne können dynamische Interaktionen zwischen dem zu entwi-

ckelnden System und seiner Umgebung in den bereits erwähnten Szenarien modelliert werden. Jedes Domänenobjekt besitzt in RETH einen Glossareintrag als (zusätzliches) Attribut. Hier können erklärende Informationen in natürlicher Sprache hinterlegt werden, die von verknüpften Anforderungen aus jederzeit verfügbar sind und so ebenfalls das Verständnis fördern.

Ziele. Diese Klasse beschreibt die *Ziele*, die mit der Entwicklung eines Systems verfolgt werden, und setzt sie mit Anforderungen in Beziehung. Diese Klasse wird durch einen Zielbeschreibungstext charakterisiert. Dieser sollte eine Zielbedingung formulieren, die eine Gruppe von gewünschten Zuständen bestimmt, wobei jeder dieser Zustände das entsprechende Ziel erfüllt. Zwar kann prinzipiell jedes Ziel modelliert werden, das im Systemzusammenhang wichtig erscheint. Aufgrund der gegebenen Beziehung zu der Unterklasse *Szenarien* sind aber besonders jene Ziele zu berücksichtigen, die durch das Ausführen bestimmter Szenarien erreichbar sind.

Die Anforderungen an das Bibliothekssystem[a] wurden in verschiedenen Klassen wie folgt zusammengefasst:

Für die Anforderungen der Anwender aus dem Anforderungsdokument *User Requirements* wurde eine Unterklasse *Anwenderforderungen* der Klasse *Anforderungen* gebildet. Hier wurden die Anforderungen in entsprechenden Unterklassen weiter unterteilt in Anforderungen an die Kundenverwaltung, die Bestandsverwaltung, die Ausleihe und die Rückgabe. Für all diese Anforderungen wurde festgelegt, dass sie gegebenenfalls durch die funktionalen Anforderungen genauer beschrieben werden.

Die technischen Anforderungen an das Bibliothekssystem (*System Requirements*) wurden größtenteils in der Klasse *Funktionale Anforderungen* eingegeben. Hier werden sie, falls anwendbar, in Beziehung zu den Anforderungen aus dem Bereich *Qualitätsanforderungen* und *Anwenderforderungen* gesetzt.

Die Abnahmekriterien in einer Unterklasse *Abnahmekriterien* der Klasse *Szenarien* eingefügt, da sie Beschreibungen von Interaktionen des Systems mit seiner Umgebung darstellen. Da diese per Definition nur durch die Qualitätsanforderungen beschränkt wird, musste eine weitere Assoziation zur Klasse *Funktionale Anforderungen* erstellt werden, um sie mit den funktionalen Anforderungen verknüpfen zu können.

[a] vgl. Abschnitt 4.1, S. 27

7.2.3 Instanzen

Eine *Instanz* ist eine spezielle Ausprägung einer Klasse. In RETH werden mittels Instanzen konkrete Anforderungen erfasst. Eine Instanz besitzt sämtliche Eigenschaften der Klasse, von der sie abgeleitet wird. Darüber hinaus kann sie um zusätzliche Attribute erweitert werden, welche dann spezifisch für diese eine Instanz gelten.

Sind in der übergeordneten Klasse Assoziationen oder Aggregationen (vgl. Abschnitt 7.2.5, S. 93) definiert worden, kann zwischen den entsprechenden Instanzen eine Verbindung hergestellt werden.

Mittels eines *Hyperlinks* können außerdem Querverweise zwischen mehreren Instanzen oder auch Instanzen und Klassen eingerichtet werden. Dazu müssen der entsprechende Link-Anker (der Ausgangspunkt des Links), der Zielknoten (der Zielpunkt des Links) und das gesuchte Attribut (vgl. Abschnitt 7.2.4, S. 92) des Zielknotens spezifiziert werden.

Die Anforderungen an das Bibliothekssystem[a] wurden als Instanzen der zuvor definierten Klassen erstellt (vgl. Abschnitt 7.2.2, S. 90). Eine Nummerierung der Anforderungen erfolgt in RETH nicht automatisch, so dass diese manuell zu erstellen ist.

[a] vgl. Abschnitt 4.1, 27

7.2.4 Attribute

Ein *Attribut* ist eine durch einen bestimmten Namen bezeichnete Eigenschaft eines Objekts (z.B. einer Klasse oder einer Instanz). Die Struktur eines solchen Objekts wird durch die Menge seiner Attribute bestimmt und in einer Klasse entsprechend definiert. Diese Klasse kann nun ihre Eigenschaften in Form von Attributen an ihre Unterklassen und Instanzen vererben.

Die verschiedenen Standardklassen (vgl. Abschnitt 7.2.2, S. 90) sind mit einer Reihe von Systemattributen ausgestattet. Darüber hinaus kann der Anwender eigene Attribute definieren. Diese können verschiedene Arten von Informationen erfassen:

- Textdaten
- Audiodaten
- Bilddaten
- Dateien

Die Attribute eines Knotens werden in einer Partition gespeichert (vgl. Abschnitt 7.2, S. 89)

Für die Abnahmekriterien des Bibliothekssystems[a] mussten zusätzliche Attribute vergeben werden. Die Attribute *Ausgangssituation*, *Ereignis* und *Erwartetes Ereignis* wurden als einfache Text-Attribute für die Unterklasse *Abnahmekriterien* der Klasse *Szenarien* definiert und werden somit an all ihre Instanzen vererbt.

Die übrigen Anforderungen wurden in dem Attribut *Anforderungstext* der entsprechenden Instanz gespeichert, das für alle Anforderungen der Klasse *Anforderungen* definiert ist. Gegebenfalls (vgl. Anforderung 1 aus Beispiel) wurden Unterpunkte einer Anforderung in dem Attribut *Zusätzliche Erklärungen* gespeichert.

[a] vgl. Abschnitt 4.1, S. 27

7.2.5 Assoziationen

Eine *Assoziation* stellt in RETH eine Relation zwischen (zwei) Objektklassen dar, die Verbindungen zwischen deren Instanzen ermöglicht. Eine solche Assoziation ist dabei immer *gerichtet*, sie hat einen Ausgangs- und einen Zielpunkt.

Zusammen mit einer Assoziation für eine bestimmte Klasse wird immer auch eine Partner-Assoziation für die beteiligte Klasse erstellt. Dadurch kann die Assoziation in beide Richtungen laufen. Darüber hinaus ist es möglich, für eine Assoziation die minimale und maximale Anzahl an beteiligten eigenen Instanzen und Partner-Instanzen festzulegen. Hierdurch lässt sich die Multiplizität einer Assoziation steuern.

Die im Beispiel[a] vorgegebenen Beziehungen wurden in RETH als Assoziationen umgesetzt. Für die Abnahmekriterien musste eine eigene Assoziation zu den funktionalen Anforderungen erstellt werden. Die übergeordnete Klasse „Szenarien“ steht standardmäßig lediglich mit den Zielen und den Qualitätsanforderungen (vgl. Abschnitt 7.2.2, S. 90) in Beziehung.

[a] vgl. Abschnitt 4.5, S. 36

Aggregationen

Eine *Aggregation* ist eine spezielle Form von Assoziation, die sich durch eine „Ganzes-Teil-Beziehung“zwischen einem Aggregat (Ganzes) und einer Teilkomponente (Teil) aus-

zeichnet. Aggregationen werden, ebenso wie die Assoziationen, zwischen zwei Klassen hergestellt. Auch in diesem Fall muss selbstverständlich eine Partner-Klasse ausgewählt werden, wobei der Anwender bestimmen kann, ob diese in der Beziehung die Rolle des Aggregats oder des Teiles übernimmt. Zudem lässt sich auch in diesem Fall die minimale und maximale Anzahlen an Instanzen festlegen, die auf beiden Seiten an der Beziehung teilnehmen.

Für das Bibliothekssystem[a] wurden keine Aggregationen eingesetzt.

[a] vgl. Abschnitt 4.1, S. 27

7.3 Funktionen

7.3.1 Ein- und Ausgabemöglichkeiten

RETH besitzt keine Möglichkeiten, Anforderungen aus anderen Dokumenten zu importieren, so dass diese manuell bzw. per Copy-and-Paste eingefügt werden müssen. Dazu werden bei Bedarf die entsprechenden Klassen und Unterklassen angelegt, um das Projekt zu strukturieren, und anschließend die entsprechenden Instanzen erzeugt. Innerhalb der Instanzen kann dann der Anforderungstext sowie weitere Informationen hinterlegt werden. Auf diese Weise wurden auch die Anforderungen an das Bibliothekssystem in das Projekt eingefügt (vgl. Abschnitt 7.2.3, S. 92).

RETH bietet dafür eine Reihe von Export-Möglichkeiten. Es bietet den Export in objektorientierte Tools, lineare Dokumente sowie Web-Darstellungen an.

- Export für objektorientierte Tools - Teilbäume eines Projekts können in objektorientierte Tools exportiert werden. Derzeit bietet RETH Schnittstellen für *IBM Rational Rose, Mark V Systems ObjectMaker* und *Aonix StP* an.
- Export für lineare Dokumente - Ein RETH-Dokument kann entweder in das Rich Text Format, das MS Word-Format oder das LaTeX-Format exportiert werden. Über die Option „Tiefe“ kann dabei die Anzahl der Ebenen im Inhaltsverzeichnis des zu erzeugenden Dokuments bestimmt werden. Für das RTF- und DOC-Format können auch im Dokument enthaltene Hyperlinks sowie eingebettete Bild- oder Audio-Dateien optional erhalten werden.
- Export in eine Web-Darstellung - Ein RETH-Dokument kann automatisch in eine Web-Darstellung exportiert werden. Es besteht die Möglichkeit, zwischen einer reinen *HTML-Darstellung* oder einer Darstellung für die Verwendung mit *ASP (Active Server Pages)* zu wählen. Enthält das Dokument Hypermedia-Dateien,

können diese automatisch in dem entsprechenden Verzeichnis verfügbar gemacht werden.

7.3.2 Traceability

Traceability wird in RETH entweder über Hypertext-Links oder über Instanzenverbindungen (Assoziationen) eingerichtet. Die so genannten *Traceability-Links* müssen in einer bestimmten Partition (z.B. einer Partition namens *Quelle*, vgl. Abschnitt 7.2, S. 89) der Instanzen des Teilbaums der Produkte (z.B. Softwareanforderungen) vorkommen, und müssen auf eine Instanz im Teilbaum der Vorgängerprodukte (z.B. Anwenderanforderungen) verweisen. Traceability-Links besitzen in RETH stets eine Richtung, wobei sie entweder vorwärts- oder rückwärts-gerichtet sein können:

Rückwärts-Traceability. Bei der Rückwärts-Traceability existiert wenigstens ein Link zu jeder Instanz der Vorgängerprodukte.

Vorwärts-Traceability. Bei der Vorwärts-Traceability existiert wenigstens ein Link zu jeder Instanz der Produkte.

7.3.3 Versionsmanagement

In RETH bezeichnet eine Version einen bestimmten Zustand eines Projekts (in RETH auch als *Dokumentenzustand* bezeichnet). Eine solche Version enthält zusätzliche Informationen über das Projekt, z.B. über den Stand der Anforderungen, der in dieser Version festgehalten ist. Die Versionsverwaltung kann in RETH aktiviert bzw. deaktiviert werden. Ist die Versionsverwaltung für ein Projekt aktiviert, können Versionen im *Versions-Navigator* (dem speziell diesem Zweck gewidmeten Teilfenster unten im Dokumentfenster, vgl. Abbildung 7.1, S. 88) leichter verwaltet werden als die übrigen Dokumentzustände. Der Versions-Navigator ist im unteren Teil des RETH-Explorers unterbracht (vgl. Abschnitt 7.1, S. 88) und kann ein- und ausgeblendet werden.

In RETH werden die folgenden Versionen eines Projekts unterschieden. Bei der Erstellung einer Version muss sich der Anwender auf eine der Varianten festlegen.

Auto Version. Eine Auto Version wird von RETH in regelmäßigen Abständen automatisch angelegt, um die Navigation in den aufgezeichneten Änderungen und Dokumentzuständen zu unterstützen.

Major Version. Eine Major Version wird als wesentlicher Dokumentzustand im Lauf der Entwicklung eines Projekts erachtet.

Minor Version. Eine Minor Version wird als wichtiger Dokumentzustand im Verlauf der Entwicklung eines Projekts erachtet. Eine Minor Version rangiert bezüglich ihrer Bedeutung etwas unterhalb einer Major Version.

7.3.4 Veränderungsmanagement

Ein Veränderungsmanagement im eigentlichen Sinne bietet RETH nicht an. Es gibt keine Möglichkeiten, Änderungsvorschläge einzugeben, zu bearbeiten oder mit bestimmten Teilen eines Projekts in Beziehung zu setzen. In RETH lassen sich Änderungen lediglich nachvollziehen, und zwar mit Hilfe des *Dokumentenzustands*. In RETH wird der Zustand eines Projekts durch einen Dokumentenzustand beschrieben. Mittels der Dokumentenzustände und der Versionsverwaltung werden Änderungen in RETH überwacht.

Ein Dokumentzustand bezeichnet also den Zustand eines bestimmten Projekts zu einem bestimmten Zeitpunkt. Ist die Versionsverwaltung für das Projekt aktiviert, werden neu entstehende Dokumentzustände indirekt durch Aufzeichnung der Änderungen gespeichert (Logging). Bei Bedarf kann ein Dokumentzustand jedoch auch manuell als eine Version definiert werden (vgl. Abschnitt 7.3.3, S. 95).

Durch eine Änderung wird ein Dokumentzustand in einen anderen transformiert. Dies kann z.B. durch das Erstellen oder Editieren eines Knotens geschehen.

Folgende Dokumentzustände können unterschieden werden:

Aktueller Dokumentenzustand. Der aktuelle Dokumentzustand ist der zuletzt durch eine Änderung entstandene und damit neueste Zustand eines Dokuments.

Betrachteter Dokumentenzustand. Der betrachtete Dokumentzustand ist der aktuell im RETH Explorer sichtbare. Ist er gleich dem aktuellen Dokumentzustand, wird der Hintergrund des Fensters weiß dargestellt. In diesem Fall kann das Dokument bearbeitet werden. Ist der betrachtete Dokumentenzustand dagegen nicht aktuell, wird der Hintergrund blau dargestellt, und das Dokument kann nicht direkt geändert werden.

Erster Dokumentenzustand. Der erste Dokumentzustand repräsentiert den ältesten Zustand des Dokuments, der über die Aufzeichnungen von Änderungen erreicht werden kann.

7.3.5 Analysemöglichkeiten

Wurden für die Anforderungen eines Projekts Links angelegt, können die Beziehungen zwischen zwei Teilbäumen mit Hilfe der *Traceability-Überprüfung* untersucht werden. In dem entsprechenden Dialog-Fenster wählt man zunächst die Produkte (also den Startpunkt der Überprüfung) und Vorgängerprodukte (den Zielpunkt der Überprüfung) aus. Anschließend spezifiziert man die Beziehung, die zwischen den beiden Teilbäumen untersucht werden soll. Als Ergebnis werden

- die Anzahl der Produktinstanzen,
- die Anzahl der Produktinstanzen ohne Referenz zu einer Vorgängerproduktinstanz,
- der Prozentsatz der Produktinstanzen ohne Referenz zu einer Vorgängerproduktinstanz,
- die Anzahl der Vorgängerproduktinstanzen,
- die Anzahl der Vorgängerproduktinstanzen ohne Referenz von einer Produktinstanz und
- der Prozentsatz der Vorgängerproduktinstanzen ohne Referenz von einer Produktinstanz angezeigt.

Das Ergebnis beschränkt sich also auf eine Auflistung und Quantifzierung der *nicht* miteinander in Beziehung stehenden Anforderungen. Die tatsächlich verlinkten Anforderungen werden nicht explizit genannt (implizit durch das Ausschlußverfahren). Ob ein bestimmter Link z.B. aufgrund einer Anforderungsänderung überprüft werden sollte, geht aus den gewonnenen Informationen ebenfalls nicht hervor. Daher ist der Nutzen dieser Überprüfung zumindestens fraglich.

Die geforderten Assoziationen zwischen den Anforderungen aus dem Bibliotheksbeispiel[a] wurden in RETH mit Hilfe der Instanzverbindungen modelliert. Eine anschließende Überprüfung der Links zwischen den Anwenderforderungen (im Beispiel die User Requirements) und den Qualitätsanforderungen (im Beispiel Teil der System Requirements) ergab folgendes Ergebnis:

- Anzahl der Produktinstanzen: 3
- Anzahl der Produktinstanzen ohne Referenz zu einer Vorgängerproduktinstanz: 1
- Prozentsatz der Produktinstanzen ohne Referenz zu einer Vorgängerproduktinstanz: 0,3333
- Anzahl der Vorgängerproduktinstanzen: 25
- Anzahl der Vorgängerproduktinstanzen ohne Referenz von einer Produktinstanz: 24
- Prozentsatz der Vorgängerproduktinstanzen ohne Referenz von einer Produktinstanz: 0,9600

[a] vgl. Abschnitt 4.5, S. 36

7.4 Sonstige Aspekte

7.4.1 Administrative Aspekte

Eine Benutzerverwaltung existiert in RETH nicht. Das Tool ist also nicht für eine Multi-User-Umgebung ausgelegt. Lediglich beim Erstellen von Versionen (vgl. Abschnitt 7.3.3, S. 95) besteht die Möglichkeit, einen Autor anzugeben.

7.4.2 Customizing

Die Möglichkeit, RETH z.B. durch weitere Funktionen oder neue Menüeinträge zu erweitern, besteht nicht. Nur das Glossar der Hilfe-Funktion lässt sich über das Domänenmodell ergänzen, da jedes Domänenobjekt auch über einen Glossareintrag als Attribut enthält.

7.5 Klassendiagramm

In diesem Abschnitt soll das Klassendiagramm von RETH (vgl. Abbildung 7.2, S. 99) vorgestellt werden. Das Diagramm basiert auf den Erkenntnissen der vorangegangenen Untersuchung des Tools und der dabei gewonnenen Erkenntnisse.

RETH unterscheidet sich von seiner Struktur her stark von den anderen Modellen. Während diese ein Modell der jeweiligen Anwendung beschreiben (und damit streng genommen selbst Metamodelle sind), wird mit Hilfe der Hypertext-Komponenenten in RETH ein Modell dieser Metamodelle beschrieben - ein Meta-Metamodell. Im folgenden soll zunächst das Meta-Metamodell vorgestellt werden. Daraufhin wird eine mögliche Instanz dieses Meta-Metamodells modelliert, wie sie in einem Standard-RETH-Projekt umgesetzt ist.

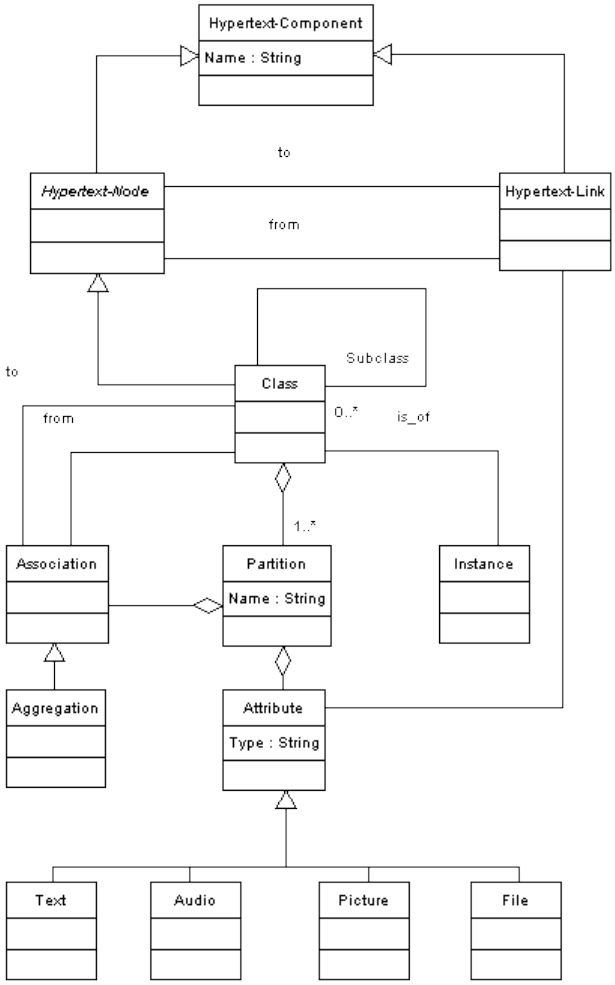


Abbildung 7.2: RETH-Meta-Metamodell

Die wichtigsten Elemente des Meta-Metamodells leiten sich von den Hypertext-Komponenten (Hypertext-Component) ab. Ein Knoten (Hypertext-Node) wird in RETH entweder durch eine Klasse (Class) oder eine entsprechende Instanz (Instance) repräsentiert. Er fasst direkt zusammengehörige Informationen zusammen. Mit Hilfe von Klassen werden diese Informationen auf Metamodell-Ebene strukturiert. Eine solche Klasse besitzt zunächst nur einen Namen. Von einer Klasse dürfen Instanzen gebildet werden.

Ein Hyperlink (Hypertext-Link) beschreibt eine Möglichkeit, Klassen miteinander in Beziehung zu setzen. Sie sind in jedem Fall gerichtet und biderektional traversierbar.

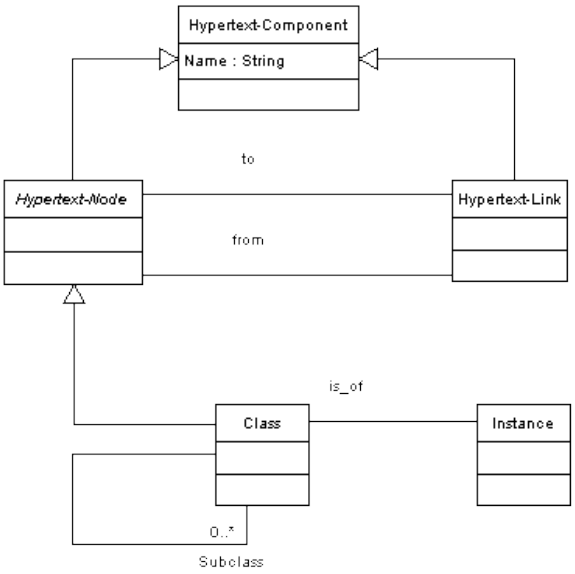


Abbildung 7.3: Hypertext-Komponenten

Eine weitere Möglichkeit, Beziehungen herzustellen, sind Assoziationen (Association). Eine Klasse kann Assoziationen zu einer oder mehreren weiteren Klassen besitzen. Eine Spezialform dieser Assoziation ist die Aggregation. Mit Hilfe einer Aggregation kann in einem RETH-Modell eine „Ganzes-Teil-Beziehung“ zwischen einem Aggregat (Ganzes) und einer Teilkomponente (Teil) gebildet werden. Sämtliche Assoziationen werden in speziellen Partitionen der Klasse gespeichert.

Abbildung 7.4 stellt diese Struktur dar:

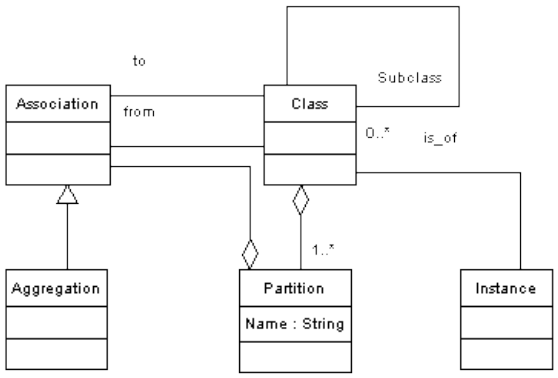


Abbildung 7.4: Assoziationen einer Klasse

Auf Meta-Metamodell-Ebene können einer Klasse verschiedene Arten von Attributen (Attribute) zugeteilt werden, die jeweils bestimmte Formen von Daten enthalten dürfen. Attribute, welche die Eigenschaften einer Klasse beschreiben, werden ebenfalls in Partitionen gespeichert.

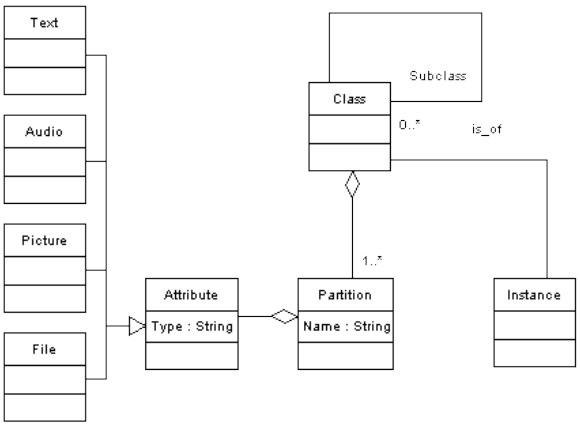


Abbildung 7.5: Attribute einer Klasse

Der folgende Teil widmet sich einer möglichen Instanz dieses Meta-Metamodells. Diese spezielle Instanz ist (zum Teil etwas ausführlicher) in einem Standard-RETH-Projekt als Struktur vorgegeben. Das Modell in der folgenden Abbildung beschränkt sich auf die wesentlichen Aspekte dieser Struktur.

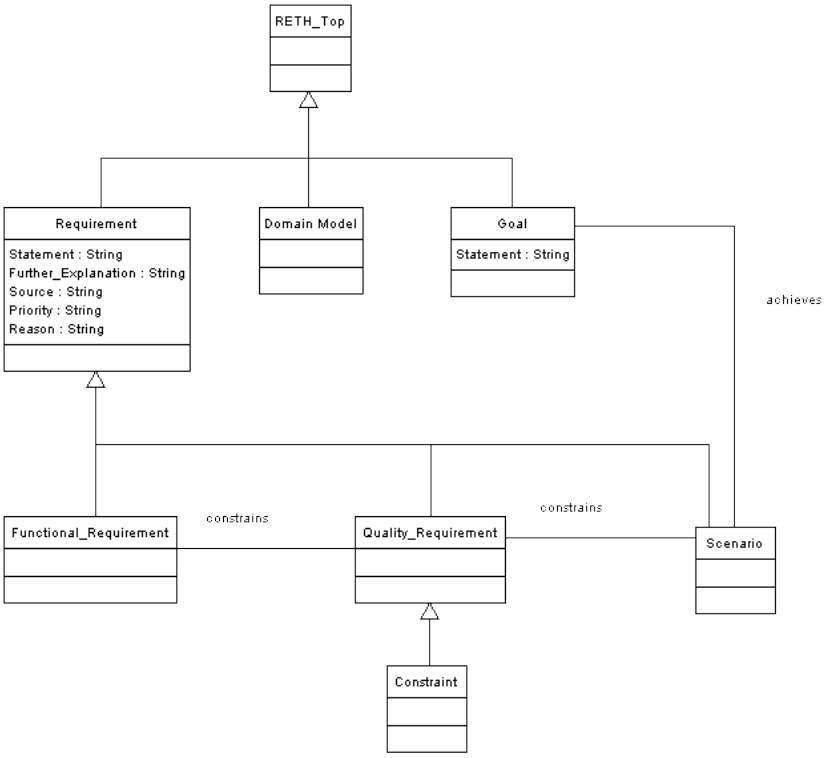


Abbildung 7.6: RETH-Metamodell

Ausgehend von RETH-Top werden mit Hilfe der Klassen Anforderungen (Requirement), Ziele (Goal) und das Domänenmodell (Domain Model) mdelliert. Sowohl die Anforderungen als auch die Ziele verfügen über eine Reihe von Attributen.

Anforderungen werden durch verschiedene Unterklassen weiter spezialisiert, z.B. in Functional_Requirement und Quality_Requirement. Die eigentlichen Anforderungen eines Projekts können als Instanzen der entsprechenden Klassen gebildet werden. Sie besitzen dann sämtliche Eigenschaften dieser Klassen mit konkreten Wertausprägungen.

Die Spezialisierungen der Anforderungen sind durch Links miteinander verbunden, die mit Hilfe von Assoziationen gebildet wurden. In diesem Beispiel beschränken constrains Qualitätsanforderungen (Quality_Requirement) sowohl die funktionalen Anforderungen Functional_Requirement als auch die Szenarien Scenario.

8 Integriertes Referenzmodell

Im folgenden Kapitel wird das Referenzmodell zur Erfassung und Verwaltung von Anforderungen vorgestellt. Das Referenzmodell basiert auf den Erkenntnissen der vorangegangenen Untersuchungen. Es hat den zuvor definierten Anforderungen zu genügen, welche an dieser Stelle noch einmal kurz zusammengefasst werden.

Zum einen soll das Modell die wichtigsten Aspekte der untersuchten Tools integrieren. Diese als *best practice* oder *best of breed* bekannte Methode soll zumindest in einem gewissen Rahmen eine Allgemeingültigkeit des Modells gewährleisten.

Darüber hinaus soll es die wesentlichen Merkmale eines Referenzmodells besitzen, wie sie in Abschnitt 2.3 auf Seite 15f. definiert wurden. Für das Referenzmodell bedeutet dies vor allem, dass es aufgrund seines idealtypischen Charakters zum Vergleich mit anderen Modellen, welche die gleichen Sachverhalte beschreiben, herangezogen werden kann.

Weitere Anforderungen ergeben sich aus den generellen Anforderungen an Requirements-Engineering-Tools (vgl. 1.1, Seite 1):

- Organisation der Anforderungen
- Dokumentation der Anforderungen
- Verwalten der Anforderungen
  - Verwalten von Anforderungsabhängigkeiten
  - Verwalten von Anforderungsänderungen

Im Folgenden wird das Referenzmodell Schritt für Schritt hergeleitet und am Ende in einem Gesamtüberblick vorgestellt. Im Anschluss daran soll es mit den zuvor erarbeiteten Modellen verglichen werden.

8.1 Organisation der Anforderungen

Der Aufbau einer Projektstruktur ist ein wesentlicher Aspekt eines jeden Requirements-Engineering-Tools. Im Referenzmodell wird diese Struktur mittels des bereits bekannten

Kompositums erzeugt. Die Wurzel dieser Struktur ist die Klasse Element. Sie fasst ein Minimum an Informationen zusammen, die allen weiteren Elementen des Referenzmodells gemein sind. So besitzt jedes Element einen Namen, eine Beschreibung, einen Autor und eine Versionsnummer.

Von der Klasse Element leiten sich die weiteren grundlegenden Elemente des Referenzmodells ab:

Container dienen als Behälter für projektbezogene Organisationen und werden durch eine gleichnamige Klasse Container repräsentiert. Ein Container darf weitere Container enthalten und ermöglicht so den Aufbau einer Baumstruktur.

Projektinformationen (z.B. Anforderungen) sind Spezialisierungen der Klasse Linked-Element. Alle Spezialisierungen dieser Klasse können mit Hilfe von Links miteinander verknüpft werden.

Ein Link besitzt im Referenzmodell eine Quelle sowie ein Ziel, ist gerichtet und bidirektional traversierbar. Dadurch soll eine optimale Traceability der Projektinformationen gewährleistet werden. Links werden durch die Klasse Link dargestellt.

Dieser in Abbildung 8.1 dargestellte Aufbau fasst den prinzipiellen Aufbau einer Projektstruktur zusammen, wie sie mit Hilfe des Referenzmodells erzeugt werden kann. Alle weiteren Elemente stellen Spezialisierungen dieser Struktur dar. Damit ist das Referenzmodell jederzeit erweiterbar und ermöglicht ein Höchstmass an Flexibilität.

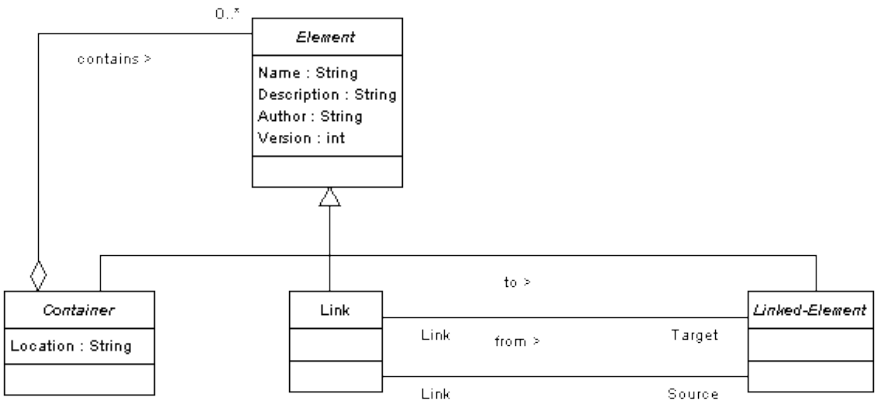


Abbildung 8.1: Projektstruktur des Referenzmodells

Zur Strukturierung eines Projekts stehen verschiedene Typen von Containern zur Verfügung (vgl. Abbildung 8.2, S. 105). Diese Klassifizierung ermöglicht eine *separation of concerns*. Bestimmte Informationen werden in den für sie angedachten Containern gespeichert. Diese können dann über bestimmte Eigenschaften verfügen, welche den Inhalt

genauer beschreiben. Im Folgenden sollen die im Referenzmodell enthaltenen Typen kurz erläutert werden:

- Project: Ein *Projekt* bildet immer die Wurzel einer Projektstruktur. Ein Projekt darf außer weiteren Containern allerdings keine zusätzlichen Projektinformationen (z.B. in Form von Anforderungen) direkt enthalten. Ein Projekt darf weitere Projekte enthalten, in diesem Fall spricht man von *Teilprojekten*. Projekte sind gekennzeichnet durch eine ID, mit Hilfe derer sie jederzeit eindeutig identifizierbar sind.
- Object-Container: Ein solcher Container enthält vor allem *Projektinformationen*, z.B. Anforderungen. Ein Objekt-Container darf aber auch (Teil-)Projekte und Link-Container enthalten.
- Link-Container: Ein solcher Container enthält nur *Traceability-Informationen*. So können beispielsweise die Traceability-Informationen zwischen zwei bestimmten Container (z.B. „Anwenderforderungen“ und „Systemanforderungen“) gruppiert werden. Dies erhöht die Effektivität der Traceability und der damit verbundenen Analysen. Diese Form des Containers darf keine weiteren Container enthalten.
- History: Die *Historie* ist ein spezieller Container, welcher die Änderungen eines bestimmten Elements verfolgt. Jedes Element besitzt also seine eigene Historie. Dieser Container darf nur Änderungen speichern. Diese Form des Containers darf keine weiteren Container enthalten.
- Baseline: Mit Hilfe der *Baseline* können Projektzustände festgehalten werden, z.B. beim Erreichen eines Meilensteins. Baselines können nur Projekte, Teilprojekte oder Container erfassen. Sie sind gekennzeichnet durch Angaben zum Zeitpunkt ihrer Erstellung. Mit ihrer Erstellung veranlassen sie außerdem eine Versionierung der gesicherten Elemente über das Attribut Version.

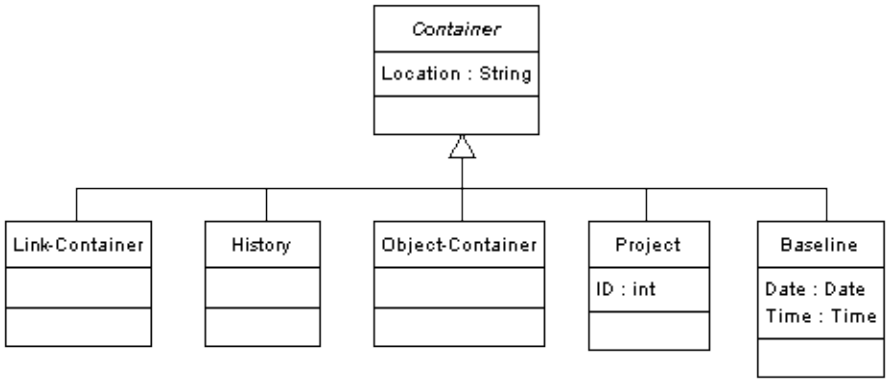


Abbildung 8.2: Containertypen im Referenzmodell

8.2 Dokumentation der Anforderungen

Anforderungen zählen natürlich zu den wichtigsten Informationen, die ein Requirements-Engineering-Tool zu verwalten hat. Die Klasse Requirement erfasst und dokumentiert die wichtigsten Informationen zu jeder einzelnen Anforderung.

Im Folgenden werden die Attribute einer Anforderung kurz vorgestellt:

- ID: Eine Anforderung kann mit Hilfe der *ID* jederzeit eindeutig identifiziert werden.
- Text: Eine Anforderung wird mit Hilfe eines *Anforderungstexts* beschrieben.
- Priority: Eine Anforderung kann *priorisiert* werden, um z.B. die für eine bestimmte Entwicklungsphase die wichtigen von den weniger wichtigen Anforderungen unterscheiden zu können. Mögliche Wertebereiche wären *niedrig, mittel* und *schwer*.
- Liability: Eine Anforderung besitzt eine *rechtliche Verbindlichkeit*. Sprachlich kann diese z.B. durch „muss“ oder „soll“ ausgedrückt werden.
- Status: Eine Anforderung kann einem Zulassungsprozess unterliegen, der durch einen *Status* beschrieben werden kann. Mögliche Wertebereiche wären *offen, zugelassen* und *implementiert*.
- I-Stakeholder: Eine Anforderung kann von einem Stakeholder veranlasst worden sein, der nicht mit dem Autor der Anforderung übereinstimmen muss. Dieser wird im Referenzmodell als *Initial Stakeholder* bezeichnet.
- R-Stakeholder: Eine Anforderung kann mehrere Stakeholder betreffen. Diese können verantwortlich für die Umsetzung der entsprechenden Anforderung sein oder besitzen vielleicht fachliche Informationen. Sie werden im Referenzmodell als *Relevant Stakeholder* bezeichnet.
- Reason: Eine Anforderung wird immer aus einem bestimmten *Grund* in die Anforderungsdokumentation übernommen. Diese Begründung erlaubt zusammen mit den Kenntnissen über den Verfasser der Anforderung die Nachverfolgung einer Anforderung hin zu ihrem Ursprung.

Anforderungen können im Referenzmodell weiter klassifiziert werden. Wie in Abbildung 8.3 dargestellt, sind funktionale und nicht-funktionale Anforderungen im Modell bereits vorhanden.

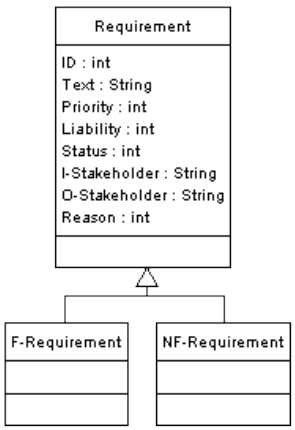


Abbildung 8.3: Anforderungen im Referenzmodell

Weitere Möglichkeiten der Spezialisierung bieten z.B. die nicht-funktionalen Anforderungen:

- Qualitätsanforderungen
- Rechtlich-vertragliche Anforderungen
- ...

Anforderungstypen können beliebig ergänzt und in das Referenzmodell integriert werden. Aus Gründen der Übersichtlichkeit wird aber im Modell auf weitere Typen verzichtet.

Zusammen mit den Anforderungen können weitere Projektinformation erfasst und dokumentiert werden:

- Model: *Modelle*, die sich aus Anforderungen ergeben können. Eine genaue Spezifikation, wie diese Modelle erfasst werden, ist nicht Teil dieser Arbeit.
- Code: *Code*, der sich aus Anforderungen oder Modellen ergeben kann. Eine genaue Spezifikation, wie dieser Code erfasst wird, ist nicht Teil dieser Arbeit.
- Testcase: *Testfälle*, um Anforderungen zu überprüfen.
- Request: *Änderungsanträge* und *Vorschläge*, die mit Anforderungen und anderen Elementen des Referenzmodells verknüpft werden können. Mehr dazu in Abschnitt 8.3.2 auf Seite 110.

Sämtliche Projektinformationen werden in speziellen Containern gespeichert. Im Referenzmodell werden diese durch die Klasse Object-Container repräsentiert. Die Assoziation der Projektinformationen zu diesen Containern (contains_objects) stellt eine Spezialisierung der Assoziation contains_elements dar.

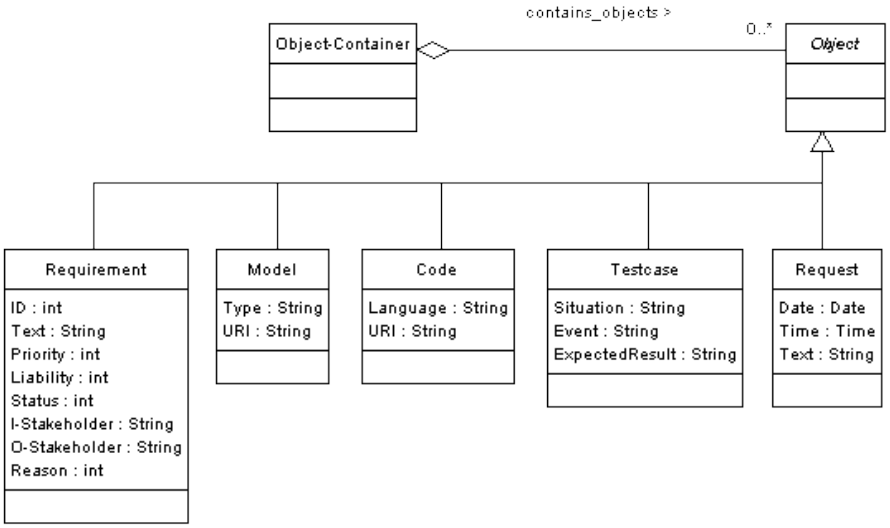


Abbildung 8.4: Projektinformationen im Referenzmodell

8.3 Verwalten von Anforderungen

Das Verwalten von Anforderungen und den damit verbundenen Informationen zählt sicher zu den wichtigsten Aufgaben eines Requirements-Engineering-Tools. Es können zwei wichtige Tätigkeiten unterschieden werden: Die Verwaltung der Anforderungsabhängigkeiten und die Verwaltung der Anforderungsänderungen. Die Umsetzung dieser Konzepte im Referenzmodell soll in den folgenden Abschnitten vorgestellt werden.

8.3.1 Verwalten von Anforderungsabhängigkeiten

Das Konzept der Traceability zwischen Anforderungen wird im Referenzmodell erweitert. Im Referenzmodell können grundsätzlich alle Projektinformationen miteinander in Beziehung gesetzt werden. Anforderungen können nicht nur untereinander verlinkt werden, sondern es besteht die Möglichkeit, Abhängigkeiten auch zwischen Anforderungen, Testfällen, Modellen oder Code zu modellieren. So soll das Referenzmodell eine möglichst effektive Nachverfolgung aller Projektinformationen sicherstellen.

Im Referenzmodell wird die Traceability der Projektinformationen durch Links gewährleistet. Solche Links setzen Elemente (Linked-Elements) miteinander in Beziehung. Wie bereits erwähnt sind Links stets gerichtet und bidirektional traversierbar.

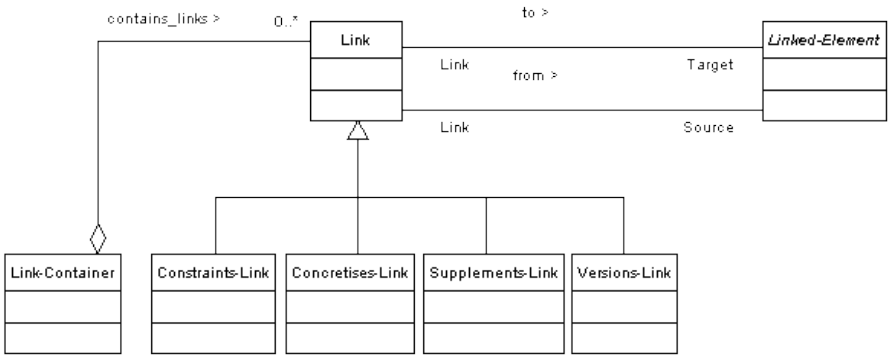


Abbildung 8.5: Traceability im Referenzmodell

Im Referenzmodell werden verschiedene Arten von Links unterschieden:

- Constrains-Link: Eine Anforderung kann eine weitere Anforderung einschränken.
- Concretises-Link: Eine Anforderung kann eine weitere Anforderung konkretisieren.
- Supplements-Link: Eine Anforderung kann eine weitere Anforderung ergänzen.
- Versions-Link: Versionierung von Anforderungen mit Baselines.

Mit Hilfe dieser Typen ist es möglich, diejenigen Elemente genauer zu bestimmen, welche miteinander verlinkt werden dürfen. So kann beispielsweise festgelegt werden, dass nur Anforderungen mit Testfällen verknüpft werden dürfen, indem ein Link vom Typ Tests-Link erstellt und dem Modell ein entsprechender Constraint hinzugefügt wird. Weitere Typen können im Referenzmodell also nach Bedarf ergänzt werden. Dadurch ist das Modell auch in Bezug auf die Traceability flexibel erweiterbar.

Links werden in Link-Container gespeichert. Dies ermöglicht eine Gruppierung von Links nach verschiedenen Merkmalen, was die Effektivität der Traceability erhöht. Die Assoziation der Links zum Link-Container (contains_links) stellt eine Spezialisierung der Assoziation contains_elements dar.

8.3.2 Verwalten von Anforderungsänderungen

Für die Verwaltung der Anforderungen ist auch die Kontrolle der Anforderungsänderungen von hoher Bedeutung. Anforderungsänderungen sollten einen *Zulassungsprozess* durchlaufen. So sollte eine Änderung zunächst nur vorgeschlagen und gegebenenfalls durch betroffene Stakeholder diskutiert werden, bevor sie tatsächlich durchgeführt wird.

Abbildung 8.6 gibt zunächst einen Überblick über die beteiligten Elemente:

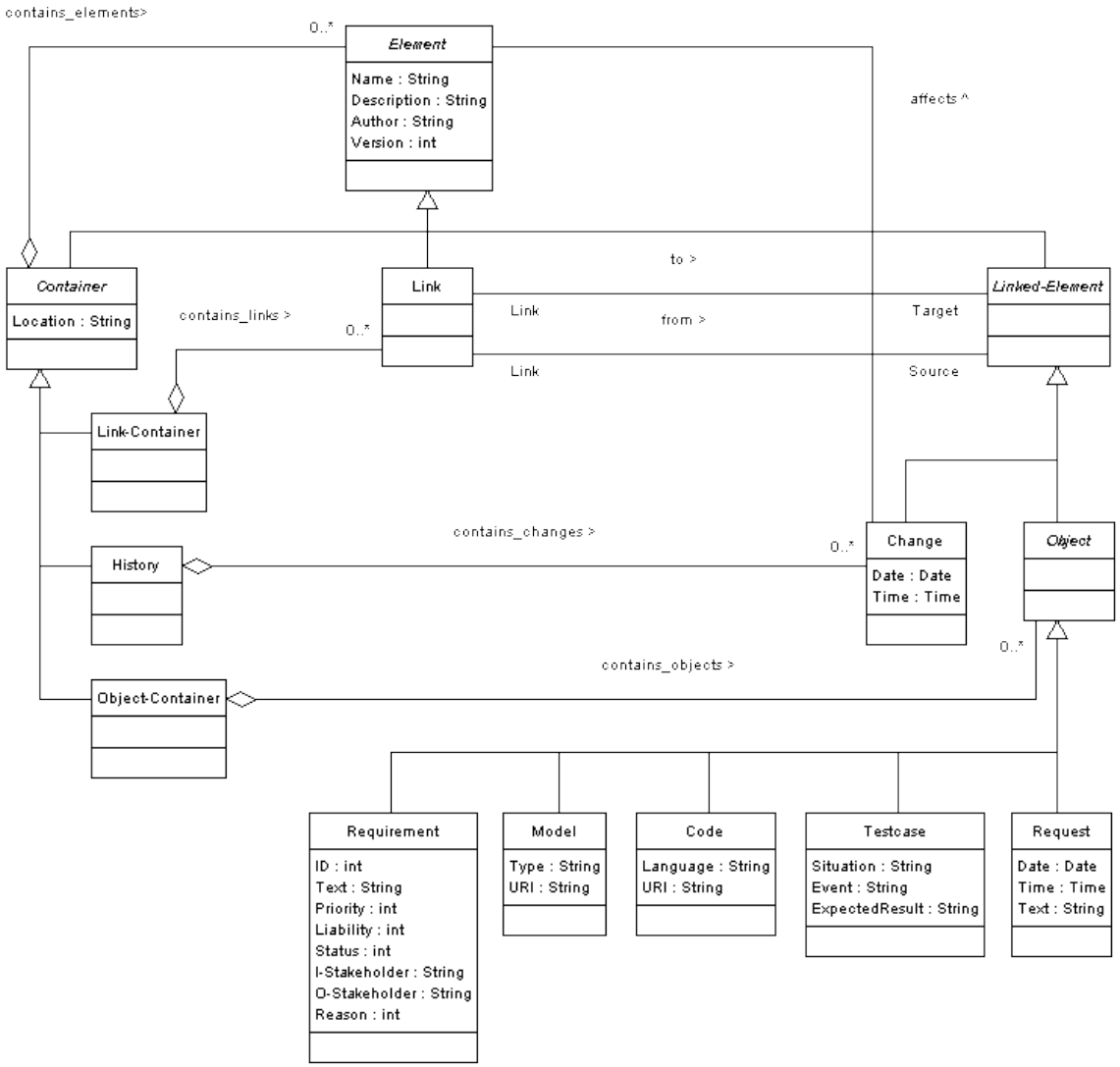


Abbildung 8.6: Veränderungsmanagement im Referenzmodell

Im Referenzmodell wird dieser Prozess durch *Änderungsanträge* unterstützt, dargestellt durch die Klasse Change-Proposal. Diese besitzt, wie eine Anforderung, die Eigenschaften Priorität, Verbindlichkeit und Status. Mit Hilfe der Priorität wird die Bedeutung eines Änderungsantrags festgelegt, die Verbindlichkeit betrifft die rechtliche Verpflichtung zur Umsetzung eines Antrags. Der Status bezeichnet die Phase des Antrags im Prozess der Zulassung.

Der endgültigen Zulassung einer Änderung kann eine Diskussion vorausgehen, wobei Beiträge in Form von Vorschlägen (Suggestions) verfasst werden. Wird eine Änderung schließlich durchgeführt, wird dies in einer Historie dokumentiert, wobei der Autor und der Grund der Änderung zusammen mit dem Datum festgehalten werden.

Änderungen und Änderungsanträge beziehen sich nicht nur auf Anforderungen, sondern auf alle Elemente des Referenzmodells. Während Änderungen in einer elementspezifischen Historie (History) gespeichert werden (contains_changes), zählen die Änderungsanträge und Vorschläge zu den Projektinformationen und werden in entsprechenden Containern zusammengefasst.

8.4 Klassendiagramm

Die folgende Abbildung zeigt das Referenzmodell im Gesamtüberblick. Es integriert die zuvor erläuterten Konzepte.

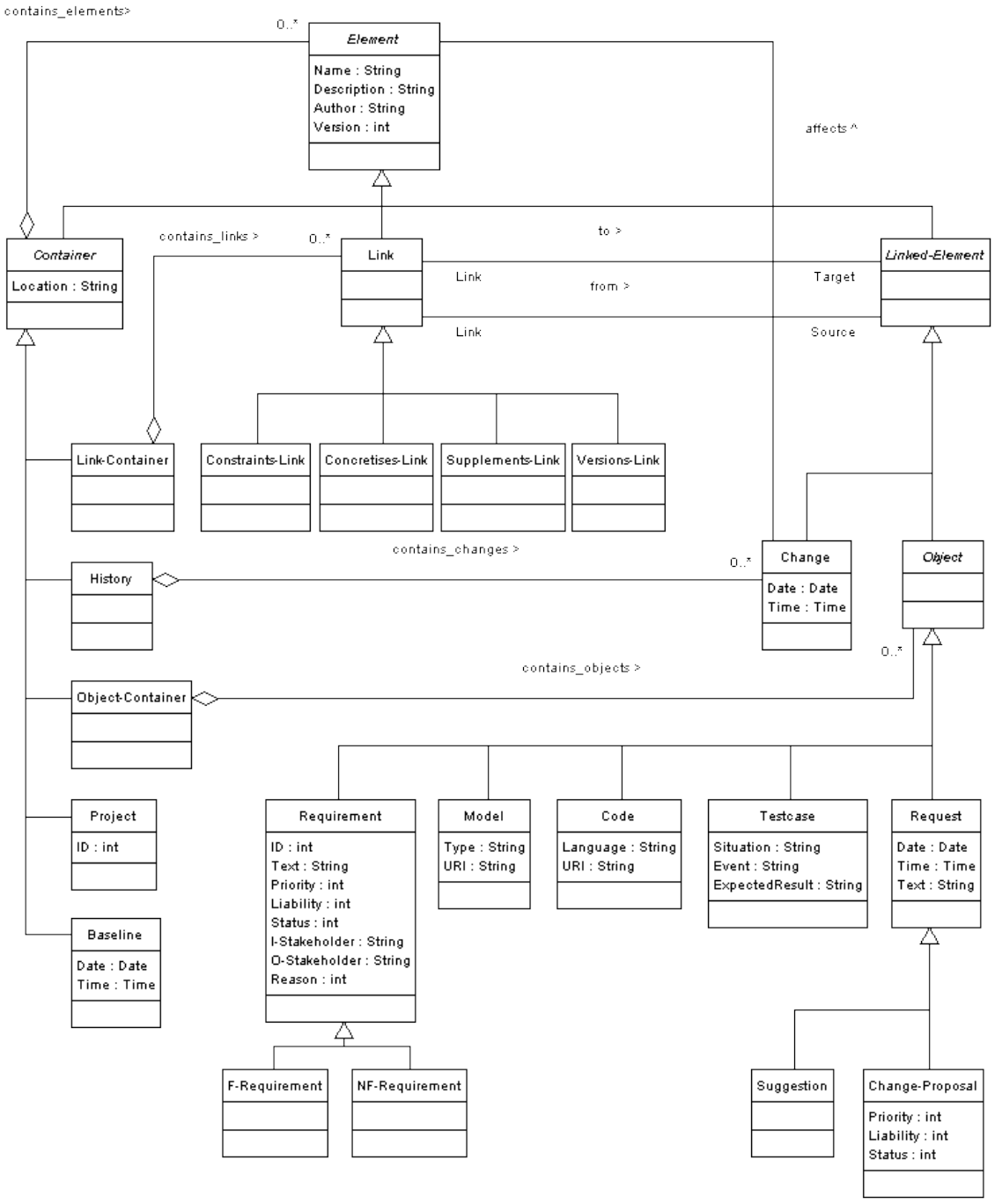


Abbildung 8.7: Referenzmodell im Gesamtüberblick

8.5 Vergleich des Referenzmodells mit den Modellen der untersuchten Tools

In diesem Abschnitt soll das Referenzmodell als Vergleich für die Modelle der beiden kommerziellen Tools herangezogen werden. Dabei soll untersucht werden, inwiefern die Anwendungsmodelle der Tools durch das Referenzmodell abgedeckt werden und welche Änderungen am Referenzmodell vorgenommen werden müssten, um auf eine gemeinsame Basis mit den Modellen der Tools zu kommen.

Auch bei dem Vergleich der verschiedenen Modelle wird besonderer Augenmerk auf die drei Hauptaufaben eines Requirements-Engineering-Tools gelegt: Die Organisation, Dokumentation und Verwaltung von Anforderungen.

8.5.1 Vergleich des Referenzmodells mit DOORS

Vergleich der Organisation von Anforderungen: Die Projektstruktur eines DOORS-Projekts ist mit dem Referenzmodell ohne weiteres abbildbar. Das Konzept der Ordner entspricht weitestgehend dem der Container, so dass hier nur minimale Änderungen (z.B. an den Attributen) notwendig sind (vgl. Abschnitt 5.2.1, S. 40). Projekte sind in DOORS ebenfalls spezielle Ordner, die über einen eindeutigen Bezeichner identifiziert werden können. Auch das Konzept der Teilprojekte ist in DOORS vorhanden (vgl. Abschnitt 5.2.2, S. 40).

Die Module in DOORS müssen als Spezialisierung der Container im Referenzmodell gebildet werden, was aber ohne weiteres möglich ist. Ohnehin übernimmt der Link-Container des Referenzmodells ähnliche Aufgaben wie das Link-Modul in DOORS. Der Objekt-Container wiederum entspricht weitestgehend dem formalen Modul in DOORS, welches dort die Projektinformationen (in DOORS hauptsächlich in Form von Anforderungen) speichert. Das beschreibende Modul lässt sich ohne weiteres als Spezialisierung eines Containers bilden, wenn man dem Container gestattet, Dateien (z.B. Textdokumente) zu speichern (vgl. Abschnitt 5.2.3, S. 41).

Über entsprechende Constraints werden Bedingungen des DOORS-Modells eingehalten. Anforderungen beispielsweise dürfen nur in formalen und beschreibenden Modulen enthalten sein. Auch damit ist das Modell sehr nahe am Referenzmodell, das Projektinformationen nur in den Objektcontainern speichert.

Vergleich der Dokumentation von Anforderungen: Anforderungen und übrige Projektinformationen werden im Referenzmodell detaillierter erfasst als in DOORS. So ist es in DOORS nicht möglich, mittels der Objekte bestimmte Typen von Anforderungen (z.B.

funktionale/nicht-funktionale) zu bilden und diese so zu klassifizieren. Auch gibt es in DOORS keine speziellen Informationstypen für Testfälle.

Die Eigenschaften eines Objekts können aber bei DOORS erweitert werden, so dass bestimmte Attribute (z.B. Intital Stakeholder oder Relevant Stakeholder) ergänzt werden können, um den Vorgaben des Referenzmodells zu entsprechen (vgl. Abschnitt 5.2.4, S. 43).

Vergleich der Verwaltung von Anforderungsabhängigkeiten: Traceability-Links werden in beiden Modellen sehr ähnlich behandelt. Sie sind gerichtet und bidirektional traversierbar. In beiden Modellen werden sie in speziellen Elementen gespeichert, in DOORS sind das die Link-Module, im Referenzmodell sind es die Link-Container.

In DOORS werden Links jedoch nicht direkt in den Link-Modulen gespeichert. Links zwischen zwei bestimmten Modulen werden in einem Linkset gespeichert (vgl. Abschnitt 5.2.3, S. 41). Diese Funktion übernehmen im Referenzmodell bereits die Link-Container, so dass diese eher den Linksets entsprechen.

Eine Klassifizierung verschiedener Linktypen, wie sie im Referenzmodell stattfindet, ist in DOORS nicht vorgesehen. Es wird lediglich zwischen eingehenden und ausgehenden Links unterschieden.

Vergleich der Verwaltung von Anforderungsänderungen: Um das Veränderungsmanagement entsprechend der Möglichkeiten in DOORS umzusetzen, muss das Referenzmodell um zusätzliche Elemente ergänzt werden. Die beiden wichtigsten Ergänzungen sind spezielle Container, welche die Aufgaben der Change-Proposal-Module übernehmen (vgl. Abschnitt 5.5, S. 55). Diese enthalten dann sämtliche Änderungsanträge und Vorschläge. Über einen Constraint muss ausgeschlossen werden, dass diese Informationen weiterhin im Objekt-Container gespeichert werden dürfen.

Ebenfalls muss durch einen Constraint sichergestellt werden, dass die das Veränderungsmanagement betreffenden Ordner nur in Projekten vorkommen dürfen, nicht in anderen Containern.

8.5.2 Vergleich des Referenzmodells mit RequisitePro

Vergleich der Organisation von Anforderungen: Auch in RequisitePro bildet ein Projekt die Wurzel der weiteren Projektstruktur (vgl. Abschnitt 6.2.1, S. 64). Dieses Projekt kann allerdings keine weiteren Teilprojekte enthalten. Dies muss bei einer Konstruktion des Anwendungsmodells mit Hilfe des Referenzmodells über einen entsprechenden Constraint sichergestellt werden muss.

Pakete und Dokumente, die in einem RequisitePro-Projekt die Projektinformationen speichern, können im Referenzmodell z.B. durch Spezialisierungen der Objekt-Container modelliert werden. Dabei ist zu beachten, dass ein Dokument weder Pakete noch Dokumente enthalten darf, sondern lediglich Anforderungen (vgl. Abschnitt 6.2.3, S. 66). Bildet man weitere Spezialisierungen dieser neuen Dokument-Container ist auch eine Klassifizierung der Dokumente vorstellbar, wie sie in RequisitePro durch die Dokumententypen realisiert wird (vgl. Abschnitt 6.2.3, S. 67). Fehlende Attribute werden bei der Modellierung entsprechend ergänzt.

Die Assoziation zwischen Objekt-Containern und Objekten kann nach wie vor bestehen bleiben, da sowohl Pakete als auch Dokumente in RequisitePro Anforderungen enthalten dürfen. Eine bestimmte Anforderung darf darüber hinaus in einem Paket und einem Dokument gleichzeitg vorkommen.

Vergleich der Dokumentation von Anforderungen: Eine Klassifizierung der Anforderungen, wie sie im Referenzmodell mit Hilfe von Spezialisierungen ermöglicht wird, ist in RequisitePro mit Hilfe der Anforderungstypen umgesetzt (vgl. Abschnitt 6.2.4, S. 70). Einer Anforderung lässt sich ein bestimmter Typ zuordnen, den man mit verschiedenen Eigenschaften ausstatten kann. So könnten z.B. für funktionale und nicht-funktionale Anforderungen entsprechende Anforderungstypen konstruiert werden. Dieses System entspricht zwar nicht genau dem im Referenzmodell eingesetzten Vererbungsprinzip, ist aber im Endergebnis recht ähnlich.

Wie bereits erwähnt, lassen sich in RequisitePro die Attribute der Anforderungen über ihre Anforderungstypen ergänzen, so dass der Umfang der erfassten Informationen vergleichbar gestaltet werden kann.

Vergleich der Verwaltung von Anforderungsabhängigkeiten: Die Abhängigkeiten zwischen Anforderungen lassen sich auch in RequisitePro nicht weiter klassifizieren. Grundsätzlich aber verfügen Links auch hier in beiden Modellen über ähnliche Eigenschaften: Sie sind gerichtet, besitzen Quelle und Ziel und sind bidirektional traversierbar.

Eine Besonderheit in RequisitePro ist die Cross-Project Traceability (vgl. Abschnitt 6.3.2, S. 73). Dieses Konzept ermöglicht es, Anforderungen verschiedener Projekte miteinander zu verknüpfen. Prinzipell ist diese Form der Traceability auch im Referenzmodell realisierbar. Darüber hinaus können im Referenzmodell auch Teilprojekte gebildet werden, deren Anforderungen untereinander verknüpft werden können. Mit Hilfe dieser Teilprojekte ist es möglich, häufig wiederkehrende System-Komponenten zu erarbeiten und in ähnlichen Projekten wiederzuverwenden.

Vergleich der Verwaltung von Anforderungsänderungen: In RequisitePro wird für das Veränderungsmanagement ein ähnliches Verfahren genutzt, wie es auch im Referenzmodell umgesetzt ist. Änderungsvorschläge betreffen in RequisitePro ein bestimmtes

Projekt oder beliebig viele Anforderungen dieses Projekts. Diese Änderungsvorschläge können anschließend diskutiert werden.

Im Referenzmodell kann ein Änderungsvorschlag nur Anforderungen, Testfälle, Modelle und Code betreffen, nicht jedoch Container oder Projekte. Würden Links allerdings alle Elemente eines Referenzmodells betreffen, könnten Änderungsvorschläge auch im Referenzmodell diese Verknüpfung herstellen. Allerdings müssten dann über eine Reihe von Constraints sichergestellt werden, dass verschiedene Elemente des Modells *nicht* miteinander verknüpft werden dürfen (z.B. Links mit Links).

Eine weitere Möglichkeit wäre, Änderungsvorschläge über eine zusätzliche Assoziation mit den Elementen zu verknüpfen. In diesem Falle würden Änderungsvorschläge ähnlich behandelt werden wie die eigentlichen Änderungen.

8.5.3 Vergleich des Referenzmodells mit RETH

Ein Vergleich des Referenzmodells mit RETH entsprechend den zuvor geführten Vergleichen mit DOORS und RequisitePro ist nicht möglich. Das liegt daran, dass RETH eher ein Meta-Metamodell zur Verfügung stellt, um (Meta-)Modelle wie die von DOORS oder RequisitePro zu modellieren. Allerdings lässt sich feststellen, dass das Referenzmodell mit Hilfe von RETH nachgebildet werden kann. Die verschiedenen Elemente und deren Beziehungen untereinander können durch RETH-Klassen und -Assoziationen modelliert werden.

8.5.4 Zusammenfassung

Mit Hilfe des Referenzmodells lassen sich alle Konzepte der untersuchten Tools abbilden. Zum Teil werden diese Konzepte durch das Referenzmodell erweitert. Im Folgenden sollen die Ergebnisse der Vergleiche in einer Übersicht dargestellt werden. Ein „+“ steht dabei für die erfolgreiche Umsetzung eines Konzepts innerhalb des jeweiligen Modells, ein „+-“ für eine eingeschränkte Umsetzung und ein „-“ für keine Umsetzung des entsprechenden Konzepts. Bei der Betrachtung der Aussagen zu RETH ist von einer Instanz des Meta-Metamodells auszugehen (vgl. Abschnitt 7.5, S. 99).

Tabelle 8.1: Zusammenfassung der Vergleiche

Konzept	DOORS	ReqPro	RETH	RefMod
Projekte	+	+	+	+
Teilprojekte	+	-	+	+
Hierarchische Projektstruktur	+	+	+	+
Anforderungen	+	+	+	+
Klassifzierung von Anforderungen	-	+	+	+
Traceability von Anforderungen	+	+	+	+
Weitere Elemente (z.B. Testfälle, Modelle, Code)	+-	+-	+-	+
Klassifzierung von Elementen	-	-	+-	+
Traceability von Elementen	+-	+-	+	+
Links	+	+	+	+
Klassifizierung von Links	-	-	+-	+
Veränderungsmanagement	+	+	+-	+
Versionsmanagment	+	+-	+-	+
Baselines	+	+-	-	+
Aktive Weiternutzung der Baselines	+	-	-	+

9 Fazit

Im Rahmen dieser Arbeit wurde eine Referenzmodell zur Erfassung und Verwaltung von Anforderungen mittels entsprechender Tools entwickelt.

Das Referenzmodell ermöglicht es zunächst, mit Hilfe verschiedener Container eine *hierarchisch gegliederte Projektstruktur* aufzubauen. Innerhalb dieser Struktur können verschiedene Container verschachtelt werden. So ist es z.B. möglich, ein Projekt in verschiedene Teilprojekte zu unterscheiden und Anforderungen innerhalb dieser nach verschiedenen Kriterien zu gruppieren. Eine aktive Zerlegung vor allem größerer Projekte in kleinere Teilprojekte ist sinnvoll, um die durchschnittliche Erfolgsrate der Einzelprojekte zu steigern.

Das Referenzmodell erlaubt natürlich die *Erfassung und Dokumentation von Anforderungen*. Neben den Anforderungen können aber weitere relevante Elemente unterschieden werden (z.B. Testfälle, Modelle, Code). Das Referenzmodell ermöglicht es dann, diese Elemente genauer zu klassifizieren. Dadurch ist es z.B. möglich, funktionale und nicht-funktionale Anforderungen voneinander zu unterscheiden und gegebenenfalls mit unterschiedlichen, typenspezifischen Eigenschaften zu versehen.

Das Referenzmodell gestattet die *Nachverfolgung von Anforderungen über den gesamten Produktlebenszyklus*. Um die Traceability von Anforderungen von Beginn an zu gewährleisten, verfügen Anforderungen im Referenzmodell über bestimmte Eigenschaften, die sie vom Zeitpunkt ihrer Erfassung an nachverfolgbar gestalten. Zu jeder Anforderung werden daher neben anderen Informationen sowohl der verantwortliche Stakeholder als auch der Grund für die Aufnahme dieser Anforderung in die Spezifikation erfasst. So lässt sich diese so bis zu ihrem Ursprung zurückverfolgen.

Mittels gerichteter Links, die jeweils Quelle und Ziel besitzen, können Anforderungen im Referenzmodell verknüpft werden. Diese Links sind bidirektional traversierbar, was eine bessere Analyse der Anforderungsabhängigkeiten ermöglicht. Auch Links können im Referenzmodell genauer klassifiziert werden. So können z.B. Links, welche eine Anforderung erweitern, von solchen unterschieden werden, die sie mit einem Testfall in Beziehung setzen. Auch dies dient natürlich der Verbesserung der Traceability-Analyse. Darüber hinaus können dadurch auch Links mit typenspezifischen Eigenschaften versehen werden, was sie vielseitig einsetzbar macht.

Das Referenzmodell bietet zusätzlich die Möglichkeit, Anforderungen nicht nur untereinander, sondern auch mit den weiteren Elementen des Modells (z.B. Testfälle, Modelle, Code) zu verknüpfen. Abhängigkeiten können z.B. zwischen Anforderungen, Modellen und Code zu erstellt werden, was eine Nachverfolgung der Anforderungen über den gesamten Lebenszyklus eines Systems hinweg ermöglicht.

Mittels des Referenzmodells lassen sich auch *Anforderungsänderungen verwalten*. Dazu kann ein Zulassungsprozess initiiert werden, der mit einem Änderungsantrag beginnt und einer möglichen Anforderungsänderung endet, die in einer Historie festgehalten wird. Während dieses Prozesses können Änderungsanträge durch beteiligte Stakeholder diskutiert und beraten werden. Darüber hinaus können Änderungsanträge, als potentiell verlinkbare Elemente, mit Anforderungen, Testfällen usw. verknüpft werden. So sind Anforderungsänderungen nicht nur nachvollziehbar, sondern auch nachverfolgbar gestaltet.

Um die *Reusability von Anforderungen* zu fördern, wurden im Referenzmodell Baselines integriert. Eine Baseline entspricht dem „Einfrieren“ eines Dokumentenzustands zu einem bestimmten Zeitpunkt. Baselines werden im Referenzmodell als eine Art Container verstanden. Mit Hilfe diese Container können Projekte, Teilprojekte oder auch nur bestimmte Container gesichert werden, um sie bei Bedarf wieder abzurufen. Dabei bleiben alle innerhalb der jeweiligen Container enthaltenen Informationen vollständig erhalten. Die Informationen innerhalb einer Baseline können auch weiterhin durch Elemente außerhalb dieser Baseline referenziert werden.

Dieses Konzept ermöglicht es beispielsweise bestimmte Systemkomponenten, die unabhängig von ihrer Umgebung sind, in eigenen (Teil-)Projekten zu entwickeln. Ist die Entwicklung einer bestimmten Phase abgeschlossen, wird dieses Teilprojekt in einer Baseline gesichert. Wird nun ein weiteres System entwickelt, welches die gleiche Systemkomponente implementiert, kann die Baseline in dieses Projekt importiert werden. Sämtliche enthaltenen Informationen können dann wiederverwendet werden.

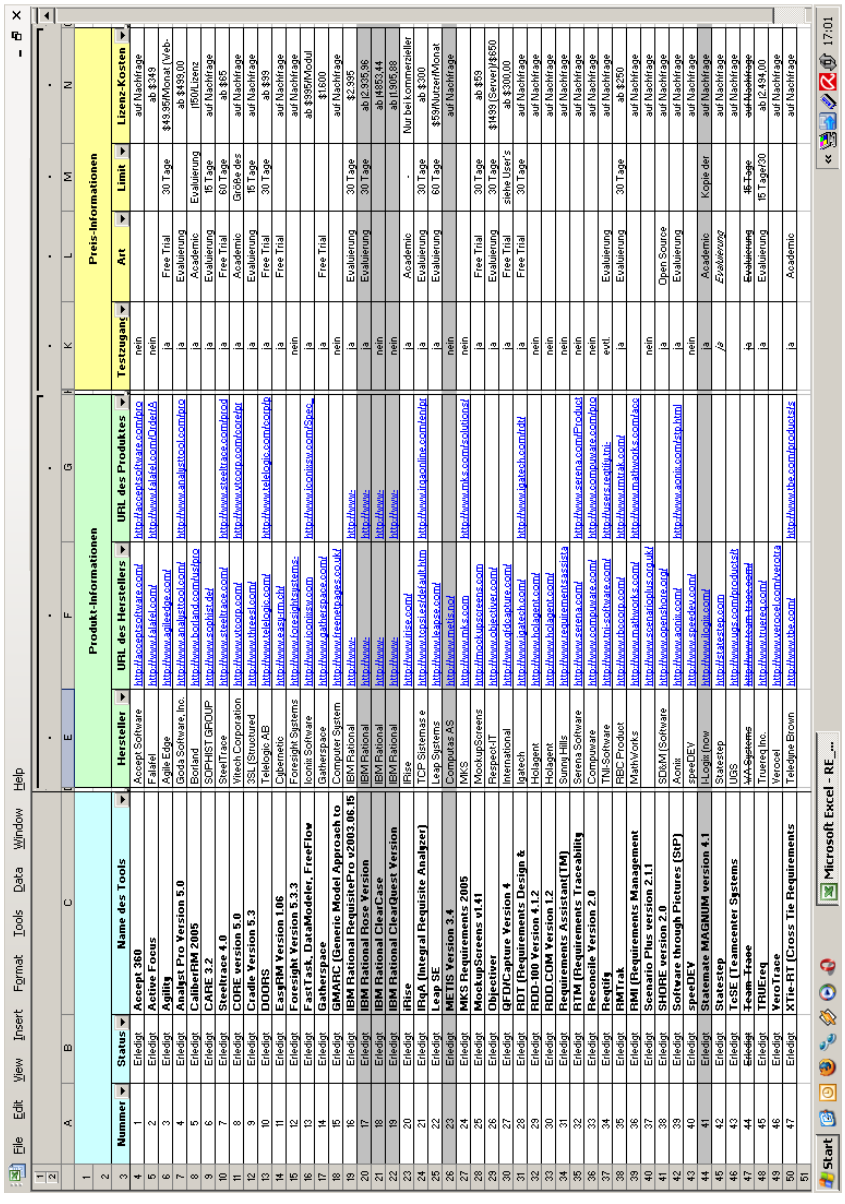
Das Referenzmodell integriert damit die wichtigsten Eigenschaften der im Rahmen dieser Arbeit untersuchten Tools. Wie diese besitzt das Referenzmodell Konzepte, um Anforderungen zu organisieren, zu dokumentieren und zu verwalten. Bestimmte Aspekte, wie etwa die Verwaltung von Anforderungabhängigkeiten und Anforderungsänderungen, werden durch das Referenzmodell aufgenommen und erweitert.

Das Referenzmodell ist darüber hinaus *flexibel erweiterbar* gestaltet. Attribute wie Klassen können ergänzt oder auch entfernt werden. So kann es an kommende Entwicklungen angepasst werden. Gerade im Bereich der Softwareentwicklung wird aktuell intensiv an verschiedenen Ansätzen, wie etwa der Model Driven Architecture, geforscht. Das Modell kann Erkenntnisse aus diesen Entwicklungen aufgreifen und integrieren.

Das Referenzmodell stellt eine Empfehlung dar. Zu Beginn dieser Arbeit wurde zwar festgestellt, dass der Empfehlungscharakter eines Referenzmodells objektiv nur schwer nachzuweisen ist. Im Rahmen dieser Arbeit soll er trotzdem wie folgt begründet sein: Durch den Auswahlprozess wurden zunächst die Qualität und die Bedeutung der Tools, welche als Grundlage für das Referenzmodell dienen, sichergestellt. So wurden im Rahmen dieser Arbeit die beiden marktführenden Tools Telelogic DOORS und IBM Rational RequisitePro sowie RETH, ein vielversprechendes Tool aus der Entwicklung von Siemens, untersucht. Durch die Integration der wichtigsten Konzepte dieser Tools wird dem Referenzmodell der Status „best of breed“ zugesprochen. Ob das Referenzmodell diesem Anspruch gerecht werden kann, wird sicherlich noch nachzuweisen sein.

Das Referenzmodell erfüllt damit *alle Anforderungen* der Aufgabenstellung. Im Idealfall leistet das Referenzmodell dadurch einen Beitrag zur Entwicklung wirkungsvoller Requirements-Engineering-Tools und damit zu einem *effektivem und effizenten Requirements-Engineering*. Die Voraussetzungen dafür sind gegeben.

A Anhang



Nummer	Status	Name des Tools	Hersteller	URL des Herstellers	URL des Produktes	Testzugang	Art	Limit	Lizenz-Kosten
				Produkt-Informationen			Preis-Informationen		
1	Erledigt	Accept 360	Accept Software	http://acceptsoftware.com/	http://acceptsoftware.com/pro	nein			auf Nachfrage
2	Erledigt	Active Focus	Falafel	http://www.falafel.com/	http://www.falafel.com/OrderA	nein			ab $349
3	Erledigt	Agilitis	Agile Edge	http://www.agileedge.com/		ja	Free Trial	30 Tage	$49,95/Monat (Web-
4	Erledigt	Analyst Pro Version 5.0	Goda Software, Inc.	http://www.analysttool.com/pro	http://www.analysttool.com/pro	ja	Evaluierung		ab $499,00
5	Erledigt	CaliberRM 2005	Borland	http://www.borland.com/us/pro		ja	Academic	Evaluierung	1150/Lizenz
6	Erledigt	CARE 3.2	SOPHIST GROUP	http://www.sophist.de/		ja	Evaluierung	15 Tage	auf Nachfrage
7	Erledigt	SteelTrace 4.0	SteelTrace	http://www.steeltrace.com/prod	http://www.steeltrace.com/prod	ja	Free Trial	60 Tage	ab $95
8	Erledigt	CORE version 5.0	Vitech Corporation	http://www.vitech.com/	http://www.vitech.com/corepr	ja	Academic	Größe des	auf Nachfrage
9	Erledigt	Cradle Version 5.3	3SL (Structured	http://www.threesl.com/		ja	Evaluierung	15 Tage	auf Nachfrage
10	Erledigt	DOORS	Telelogic AB	http://www.telelogic.com/	http://www.telelogic.com/corpfo	ja	Free Trial	30 Tage	ab $39
11	Erledigt	EasyRM Version 1.06	Cybernetic	http://www.easyrm.cl/		ja	Free Trial		auf Nachfrage
12	Erledigt	Foresight Version 5.3.3	Foresight Systems	http://www.foresightsystems/		nein			auf Nachfrage
13	Erledigt	FastTask, DataModeler, FreeFlow	Iconix Software	http://www.iconixsw.com/	http://www.iconixsw.com/Spec	ja			ab $395/Modul
14	Erledigt	Gatherspace	Gatherspace	http://www.gatherspace.com/		ja	Free Trial		$1600
15	Erledigt	GMARC (Generic Model Approach to	Computer System	http://www.itsenetpages.co.uk/		nein			auf Nachfrage
16	Erledigt	IBM Rational RequisitePro v2003.06.15	IBM Rational	http://www-	http://www-	ja	Evaluierung	30 Tage	$2.395
17	Erledigt	IBM Rational Rose Version	IBM Rational	http://www-	http://www-	ja	Evaluierung	30 Tage	ab 2.395,96
18	Erledigt	IBM Rational ClearCase	IBM Rational	http://www-	http://www-	nein			ab 1.863,44
19	Erledigt	IBM Rational ClearQuest Version	IBM Rational	http://www-	http://www-	nein			ab 1.905,88
20	Erledigt	iRise	iRise	http://www.irise.com/		ja	Academic	-	Nur bei kommerzieller
21	Erledigt	IRqA (Integral Requisite Analyzer)	TCP Sistemas e	http://www.tcpsa.es/default.htm	http://www.tcpsa.es/demo/index	ja	Evaluierung	30 Tage	ab $300
22	Erledigt	Leap SE	Leap Systems	http://www.leapse.com/		ja	Evaluierung	60 Tage	$59/Nutzer/Monat
23	Erledigt	METIS Version 3.4	Computas AS	http://www.metis.no/		nein			auf Nachfrage
24	Erledigt	MKS Requirements 2005	MKS	http://www.mks.com/	http://www.mks.com/solutions/	nein			
25	Erledigt	MockupScreens v1.41	MockupScreens	http://mockupscreens.com		ja	Free Trial	30 Tage	ab $59
26	Erledigt	Objectiver	Respect-IT	http://www.objectiver.com/		ja	Evaluierung	30 Tage	$1439 (Server)/$850
27	Erledigt	QFDiC-apture Version 4	International	http://www.aldo-acute.com/		ja	Free Trial	siehe User's	ab $300,00
28	Erledigt	RDT (Requirements Design &	Igatech	http://www.igatech.com/rdt/	http://www.igatech.com/rdt/	ja	Free Trial	30 Tage	auf Nachfrage
29	Erledigt	RDD-100 Version 4.1.2	Holagent	http://www.holagent.com/		nein			auf Nachfrage
30	Erledigt	RDD.COM Version 1.2	Holagent	http://www.holagent.com/		nein			auf Nachfrage
31	Erledigt	Requirements Assistant(TM)	Sunny Hills	http://www.requirementsassista		nein			auf Nachfrage
32	Erledigt	RTM (Requirements Traceability	Serena Software	http://www.serena.com/	http://www.serena.com/Product	nein			auf Nachfrage
33	Erledigt	Reconcile Version 2.0	Compuware	http://www.compuware.com/	http://www.compuware.com/pro	nein			auf Nachfrage
34	Erledigt	Reqtify	TNI-Software	http://www.tni-software.com/	http://users.reqtify.tni-	evtl.	Evaluierung		auf Nachfrage
35	Erledigt	RMTrak	RBC Product	http://www.rbcorp.com/	http://www.rmtrak.com/	ja	Evaluierung	30 Tage	ab $250
36	Erledigt	RMI (Requirements Management	MathVerks	http://www.mathverks.com/	http://www.mathverks.com/Asso	nein			auf Nachfrage
37	Erledigt	Scenario Plus version 2.11		http://www.scenarioplus.co.uk/		nein			auf Nachfrage
38	Erledigt	SHORE version 2.0	SDEM (Software	http://www.openshore.org/		ja	Open Source		auf Nachfrage
39	Erledigt	Software through Pictures (StP)	Aonix	http://www.aonix.com/	http://www.aonix.com/stp.html	ja	Evaluierung		auf Nachfrage
40	Erledigt	speeDEV	speeDEV	http://www.speedev.com/		nein			auf Nachfrage
41	Erledigt	Statemate MAGNUM version 4.1	i-Logix Inc.	http://www.ilogix.com/		ja	Academic	Kopie der	auf Nachfrage
42	Erledigt	Statestep	Statestep	http://statestep.com/		/a	Evaluierung		auf Nachfrage
43	Erledigt	TcSE (Teamcenter Systems	USS	http://www.uss.com/products/t					auf Nachfrage
44	Erledigt	Team-Trace	ieh-Systems	http://www.ieh-systems.com/		ja	Evaluierung	45 Tage	auf Nachfrage
45	Erledigt	TRUEreq	Truereq Inc.	http://www.truereq.com/		ja	Evaluierung	45 Tage/30	ab 12.494,00
46	Erledigt	YesTrace	Yeriotel	http://www.yeriotel.com/yestra					auf Nachfrage
47	Erledigt	XTie-RT (Cross Tie Requirements	Teledyne Brown	http://xtie.com/	http://xtie.com/products/rt/	ja	Academic		auf Nachfrage

Abbildung A.1: Liste der Requirements-Engineering-Tools (1)

	Betriebssystem				Technische Informationen			Sonstige Informationen		
	Windows	UNIX	Linux	Solaris	Andere	Architektur	Sonstige Voraussetzungen	Version	Release Date	Schwerpunkt	Stichpunkte
4	x		x	x		Web-Based oder				Product Management	
5										Project Management	weitere Informationen nur auf
6	x	x	x			Web-Based oder		2.0	23.07.2005	Requirements Engineering	RM
7	x					Stand-alone oder	MS Word	5.2	16.12.2005	Requirements Engineering	
8	x					Client/Server		Release 2 SP1	30.09.2006	Requirements Engineering	vollständigen Caliber Analyst
9	x	x	x	x	x	Client/Server	Lotus Notes Version 5-	3.2	01.09.2004	Requirements Engineering	Whitepapers
10	x					Stand-alone oder		4.0	01.12.2005	Requirements Engineering	Whitepapers, Enterprise-und
11	x					Client/Server		5.0		Requirements Engineering	
12	x	x				Client/Server	MS Word, MS Excel	5.3	01.12.2006	Systems Engineering	
13	x	x	x	x		Client/Server		8.0		Requirements Engineering	
14	x	x		x		Stand-alone		1.06	10.02.2002	Requirements Engineering	modularer Aufbau
15	x		x							Systems Engineering	nicht direkt RE, arbeitet mit Third-
16						Client/Server				CASE	nicht direkt RE, modularer Aufbau
17						Online				Requirements Engineering	
18								5.05		Requirements Engineering	mögl. outdated, Ressourcen
19	x					Client/Server	MS Word, Datenbank (MS Access ist			Requirements Engineering	Client: Requisite Web
20	x	x		x		Client/Server				Datenmodellierung	
21	x	x	x	x		Client/Server				Change Management	
22	x	x	x	x		Client/Server				Change Management	
23						Client/Server				Requirements Engineering	modularer Aufbau, Rose Manager
24	x					Client/Server		3.3		Requirements Engineering	mit.NET auch mittels Internet
25	x					Client/Server	MS Access			CASE	Requirements Engineering
26	x					Client/Server		3.4		Modellierung	
27	x									Systems Engineering	modularer Aufbau, MKS
28	x							2.71		Requirements Engineering	Mailprogramm
29	x							1.5		Requirements Engineering	Basiert auf der KADS-Methode
30	x					Stand-alone oder				Requirements Engineering	RE im entfernten Sinne
31	x					Stand-alone oder	MS Word			Systems Engineering	Integration mit AxiomSys
32										Requirements Engineering	vielleicht eher Systems Engineering
33								1.2		Requirements Engineering	vielleicht eher Systems Engineering
34										Requirements Engineering	Analysiert Requirements in
35	x					Client/Server		5.6		Requirements Engineering	interessantes Paper zum ROI
36	x					Client/Server	MS Word	2.0		Requirements Engineering	
37										Requirements Engineering	haupts. Interface, Traceability
38	x					Stand-alone	MS Word	3.0 Beta	01.12.2005	Requirements Engineering	Document-centric
39	x	x					MATLAB 6.3.1, Simulink 3.0.1,	5.2.4		Requirements Engineering	haupts. Interface
40	x	x	x	x			DOORS			Requirements Engineering	nur Add-ons für DOORS
41										Repository	The core system of SHORE is no
42	x	x	x	x						Requirements Engineering	eher Modellierung
43	x					Client/Server	Microsoft SQL Server	3.7		CASE	SpeedEY RM - Requirements
44	x			x				4.1	24.10.2006	Systems Engineering	MBD for Embedded Systems
45	x	x	x	x	x		JRE	1.0		Systems Engineering	alle Informationen nur auf
46										Requirements Engineering	
47	x					Stand-alone oder	MS Word	ca.2	04.04.2003	Requirements Engineering	
48	x				x	Web-Based	MS Access Runtime	1.3	05.11.2004	Requirements Engineering	
49										Requirements Engineering	alle Informationen nur auf
50	x					Stand-alone oder		3.103		Requirements Engineering	Traceability

Abbildung A.2: Liste der Requirements-Engineering-Tools (2)

Literaturverzeichnis

[Ale06] Alexander, Ian. *Requirements Engineering Tools.* 2006-09-20 `http://easyweb.easynet.co.uk/~iany/other/vendors.htm`. 2006

[Bor05] Borland: *Mitigating Risk with Effective Requirements Engineering: How to improve decision-making and opportunity through effective requirements engineering. Part two in a series about understanding and managing risk.* 2006-05-03 `www.borland.com/resources/en/pdf/white_papers/mitigating_risk_with_effective_requirements_engineering.pdf`. 2005. – White Paper

[Bro03] vom Brocke, Jan: *Referenzmodellierung: Gestaltung und Verteilung von Konstruktionsprozessen.* Berlin: Logos Verlag, 2003. – XI, 404 S. : Ill., graph. Darst. S. – ISBN 3–8325–0179–7

[Cen02] Center for Software Engineering, University of Southern California. *EasyWinWin: A Groupware-Supported Methodology For Requirements Negotiation.* 2006-09-27 `http://sunset.usc.edu/research/WINWIN/EasyWinWin/`. 2002

[DAT06] DATACOM Buchverlag GmbH. *ITWissen. Das große Online-Lexikon für Informationstechnologie.* 2006-09-21 `http://www.itwissen.info/definition/lexikon/breitbandnetze/__tree%20topology_baumtopologie.html`. 2006

[IBM06] IBM Rational Software: *RequisitePro integrierte Hilfe.* Rational RequisitePro 2003. 2006. – Software

[INC06] INCOSE. *INCOSE Requirements Management Tools Survey.* 2006-09-20 `http://www.paper-review.com/tools/rms/read.php`. 2006

[Kai97] Kaindl, Herrmann: A Practical Approach to Combining Requirements Definition and Object-Oriented Analysis. In: *Annals of Software Engineering* 3 (1997). – 2006-09-25 `http://www.springerlink.com/content/u841237202862011/`

[Kai04] Kaindl, Hermann: Active Tool Support for Requirements Engineering Through RETH. In: IEEE Computer Society (Hrsg.): *Proceedings of the 12th IEEE International Requirements Engineering Conference* Institute of Electrical and Electronics Engineers, 2004. – ISBN 0–7695–2174–6, S. 362–363

[Kle03] Kleppe, Jos ; Bast W.: *MDA explained : the model driven architecture ; practice and promise.* [Nachdr.]. Boston [u.a.]: Addison-Wesley, 2003 (The

Addison-Wesley object technology series). – XVII, 170 S. : graph. Darst. S. – ISBN 0–321–19442–X

[Kru04] Kruchten, Philippe: *The rational unified process : an introduction.* 3. ed. Boston, Mass.: Addison-Wesley, 2004 (The Addison-Wesley object technology series). – XVIII, 310 S. : Ill., graph. Darst. + 1 Faltblatt S. – ISBN 0–321–19770–4

[KS03] Kotonya, Gerald ; Sommerville, Ian: *Requirements engineering : processes and techniques.* Reprint. Chichester [u.a.]: Wiley, 2003 (Worldwide series in computer science). – XI, 282 S. : graph. Darst. S. – ISBN 0–471–97208–8

[LW03] Leffingwell, Dean ; Widrig, Don: *Managing software requirements : a use case approach.* 2. ed., 1. print. Boston [u.a.]: Addison-Wesley, 2003 (The Addison-Wesley object technology series). – XXXVII, 502 S. : Ill., graph. Darst. S. – ISBN 0–321–12247–X

[Müh06] Mühlbauer, Susanne: Werkzeuge im Anforderungsmanagement. In: *OBJEKTspektrum-Online-Ausgabe* 2 (2006). – 2006-09-21 `http://www.sigs.de/publications/os/2006/RE/muehlbauer_OS_RE_06.pdf`

[RR99] Robertson, Suzanne ; Robertson, James: *Mastering the requirements process.* London [u.a.]: Addison-Wesley, 1999. – XI, 404 S. : Ill., graph. Darst. S. – ISBN 0–201–36046–2

[RR06] Robertson, Suzanne ; Robertson, James. *Requirements Tools.* 2006-09-20 `http://www.volere.co.uk/tools.htm`. 2006

[Rup04] Rupp, Chris: *Requirements-Engineering und -Management : professionelle, iterative Anforderungsanalyse für die Praxis.* 3., neu bearb. Aufl. München [u.a.]: Hanser, 2004. – XII, 512 S. : Ill., graph. Darst. S. – ISBN 3–446–22877–2 Pp. ; 978–3–446–22877–1

[SA03] Stufflebeam, William ; Anton, Annie I.: SMaRT - Scenario Management and Requirements Tool. In: IEEE Computer Society (Hrsg.): *Proceedings of the 11th IEEE International Requirements Engineering Conference* Institute of Electrical and Electronics Engineers, 2003. – ISBN 0–7695–1980–6, S. 351

[Sey04] Seyff, Norbert: Collaborative Tools for Mobile Requirements Acquisition. In: IEEE Computer Society (Hrsg.): *Proceedings of the 19th International Conference on Automated Software Engineering* Institute of Electrical and Electronics Engineers, 2004. – ISBN 0–7695–2131–2, S. 426–429

[Sie06] Siemens AB Österreich: *RETH integrierte Hilfe.* RETH 2.5. 2006. – Software

[Smi93] Smith, Thomas J.: READS: A Requirements Engineering Tool. In: IEEE Computer Society (Hrsg.): *Proceedings of the IEEE International Symposium on Requirements Engineering* Institute of Electrical and Electronics Engineers, 1993. – ISBN 0–8186–3120–1, S. 94–97

[Som01] Sommerville, Ian: *Software engineering.* 6. ed. Harlow, England [u.a.]: Addison-Wesley, 2001 (International computer science series). – XX, 693 S. : graph. Darst. S. – ISBN 0–201–39815–X

[SS99] Sommerville, Ian ; Sawyer, Pete: *Requirements engineering : a good practice guide.* Reprint. Chichester [u.a.]: Wiley, 1999. – XI, 391 S. : graph. Darst. S. – ISBN 0–471–97444–7

[Tel06a] Telelogic AB. *CEO introduction.* 2006-09-20 `http://www.telelogic.com/corp/Company/investor_relations/financials/reference_material/upload/CEO_introduction.ppt`. 2006

[Tel06b] Telelogic AB: *DOORS integrierte Hilfe.* Telelogic DOORS 8.1. 2006. – Software

[The04] The Standish Group International, Inc. *2004 Third Quarter Research Report.* `http://www.standishgroup.com/sample_research/PDFpages/q3-spotlight.pdf/`. 2004

[Ver04] Versteegen, Alexander: *Anforderungsmanagement : formale Prozesse, Praxiserfahrungen, Einführungsstrategien und Toolauswahl.* Berlin ; Heidelberg [u.a.]: Springer, 2004 (Xpert.press). – XV, 284 S. : Ill., graph. Darst. S. – ISBN 3–540–00963–9

[Yph05] Yphise Software Product Assessment. *Requirements-driven application lifecycle management.* 2006-09-25 `http://www.telelogic.com/download/index.cfm?id=3990`. 2005