Family-Oriented Requirements Engineering

Dissertationsschrift
zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

vorgelegt der Fakultät für Informatik und Automatisierung
der Technischen Universität Ilmenau

von Dipl.-Informatiker Detlef Streitferdt
geboren am 5. Juni 1972 in Heltau

vorgelegt am 25. August 2003

Gutachter:

1. Prof. Dr.-Ing. habil Ilka Philippow
2. Prof. Dr.-Ing. habil. Dietrich Reschke
3. Dr. Günter Böckle

wissenschaftliche Aussprache am 23. April 2004
Verf.-Nr.: IA 120

Vorwort

Diese Arbeit entstand am Fachgebiet Prozessinformatik unter der Leitung von Frau Prof. Dr.-Ing. habil. Ilka Philippow im Rahmen der Forschungskooperation zum Thema „Systemfamilien“ mit der Siemens Forschungsabteilung CT SE 3 in München, vertreten durch Dr. Günter Böckle. Darüber hinaus war die Arbeit auch Teil des Forschungsprojektes Alexandria (Ziel des Projektes ist die durchgängige methodische Unterstützung der Systemfamilienentwicklung) am Fachgebiet, welches von Dr. Matthias Riebisch geleitet wird.

Danken möchte ich Ilka Philippow, Matthias Riebisch und Günter Böckle für die Betreuung im Verlauf dieser Arbeit und die vielen interessanten Diskussionen. Meine Anerkennung gilt den Studenten, die sich in ein umfangreiches Gebiet eingearbeitet haben und durch zahlreiche Studien- und Diplomarbeiten an dem digitalen Videoprojekt mitgearbeitet haben. Ein spezieller Dank geht an meine Kollegen, die auch für meine nichtwissenschaftlichen Fragen ein offenes Ohr hatten. Hochachtung bringe ich all denen entgegen, die mich trotz meiner wissenschaftlichen Tätigkeiten nicht vergessen haben. Bei der wichtigsten Person in meinem Leben möchte ich mich dafür bedanken, dass sie da ist und Ihr, Maria, diese Arbeit widmen.

Kurzfassung

Heutige Softwareprodukte sollen in kurzer Zeit bei gleichzeitig hoher Qualität entwickelt werden, wobei die Software-Technik dieser Forderung durch die Vorfertigung einzelner Komponenten gerecht wird. Die geplante und umfassende Wiederverwendung von Komponenten innerhalb einer Anwendungsdomäne wird durch das Konzept der Systemfamilienentwicklung unterstützt. Eine Systemfamilie basiert auf einer Referenzarchitektur, die aus Anteilen besteht, die allen Familienmitgliedern gemein sind und Anteilen, die optional sind. In allen Phasen der Systemfamilienentwicklung müssen gemeinsame und optionale Anteile berücksichtigt und korrekt verarbeitet werden, wobei Fehler in der *Requirements-Engineering*-Phase, dem Beginn der Entwicklung, den größten wirtschaftlichen Schaden nach sich ziehen.

In dieser Arbeit zeigt die Analyse der *Requirements-Engineering*-Phase der Systemfamilienentwicklung, dass die Systemfamilienmodelle der meisten, existierenden Ansätze auf der Merkmalmodellierung basieren, die jedoch inkonsistent und nicht automatisiert verarbeitbar ist. Der hier beschriebene Lösungsansatz – *Family-Oriented Requirements Engineering* (*FORE*) – erweitert Merkmalmodelle und integriert sie in ein neues Datenmodell, das alle im Verlauf der *Requirements-Engineering*-Phase erarbeiteten Informationen enthält. Zur Modellierung aller Abhängigkeiten innerhalb von Merkmalmodellen und von Merkmalen zu weiteren Modellelementen bietet FORE die neue *Feature Constraint Language* (FCL) mit 30 vordefinierten Beziehungen an, wodurch Systemfamilienmodelle überprüfbar sind. Durch die kundenspezifische Auswahl von Merkmalen wird ein Familienmitglied basierend auf der Systemfamilie abgeleitet. Die erweiterten Merkmalmodelle von FORE ermöglichen die automatisierte Überprüfung einer Merkmalauswahl, sodass nur gültige Familienmitglieder abgeleitet werden können. Sowohl die Entwicklung eines Systemfamilienmodells als auch die Ableitung von Familienmitgliedern werden durch den FORE-Entwicklungsprozess unterstützt, der existierende Ansätze und die erweiterte Merkmalmodellierung in einem *Requirements-Engineering*-Prozess für Systemfamilien integriert. Anhand eines Prototyps wird die Realisierbarkeit des Lösungsansatzes gezeigt, indem das FORE-Datenmodell in eine XML-Struktur umgesetzt und die Benutzung des Prototyps dem FORE-Entwicklungsprozess folgt.

FORE wurde im Rahmen eines universitären Projektes, wie auch durch diverse studentische Arbeiten mit Industriepartnern überprüft. Die Ergebnisse dieser Arbeiten haben die Praxistauglichkeit von FORE gezeigt, den Prozess und das Datenmodell verbessert und dessen Grenzen aufgezeigt.

Abstract

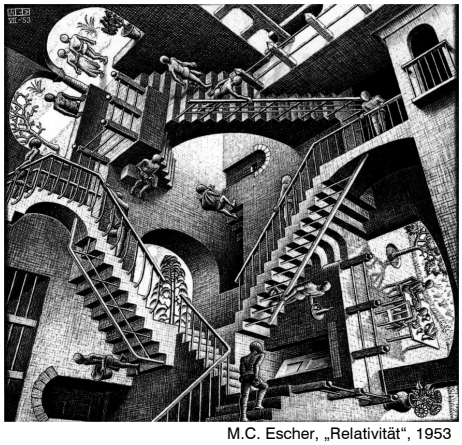
Modern software products shall be developed within a short time and at the same time they should be of a high quality. Software engineering is able to fulfill these requirements by prefabricating components. Within a domain, planned and comprehensive reuse of components is supported by the concept of system family development. A system family is based on a reference architecture made of assets, which are common to all family members and assets, which are variable. Commonalities and variabilities have to be considered in all phases of the system family development and they have to be elaborated correctly. Mistakes made in the requirements engineering phase, as the beginning of a development, will cause the most damage to the overall development.

The analysis of the requirements engineering phase of system family development in this paper shows, that most of the current scientific solutions in this field are based on feature modeling, although it is inconsistent and cannot be automatically processed. The proposed solution of this paper – *Family-Oriented Requirements Engineering* (*FORE*) – extends feature modeling and integrates it into a new data model, capable of holding all the information acquired within the requirements engineering phase. Dependencies within feature models and between features and further model elements can be modeled with the new FORE *Feature Constraint Language* (FCL). FCL offers 30 predefined dependencies for verifiable system family models. Extended feature models of FORE allow the automated verification of a subset of features. Thus, only valid family members can be derived of the system family. The development of a system family model as well as the derivation of family members are supported by the FORE development process. The FORE-process integrates current processes and the extended feature modeling of FORE into a requirements engineering process for system families. The proposed solution was prototypically realized by implementing the FORE-Data Model as XML-Schema. The usage of the prototype is aimed at the FORE-Development Process.

FORE was tested within a University project as well as in several student works with industry partners. The results of this paper have shown the applicability of FORE, improved its process and data model and revealed its limits.

Inhaltsverzeichnis

Kapitel 1



M.C. Escher, „Relativität“, 1953

Einleitung und Motivation des Themas

Dieses Kapitel beschreibt das Umfeld, aus dem die Problemstellung der vorliegenden Arbeit erwachsen ist. In [Funk 1999] werden über 60% der deutschen Softwareunternehmen als mittelständisch bis groß ausgewiesen. Die Unternehmensgröße sowie die Ausrichtung der Projekte haben in den letzten Jahren dazu geführt, dass Softwareunternehmen ihre Produktpalette zusammenfassen und innerhalb der Grenzen der jeweiligen Anwendungsdomäne ausrichten. Softwareentwicklung innerhalb einer bestimmten Problemdomäne kann und wird derzeit oft durch Nutzung einer an die Systemfamilienentwicklung angepassten Methode betrieben. Dadurch können Unternehmen über die gesamte Lebensdauer der Produkte ihre Ressourcen optimal nutzen und verteilen. Unzulänglichkeiten dieser speziellen und modernen Art der Softwareentwicklung werden vorgestellt und anhand von Beispielen erläutert. Aus diesen Betrachtungen allgemeiner Natur wird eine Problemstellung formuliert, die als Basis für die Arbeiten des nächsten Kapitels dient.

1.1 Umfeld

Von moderner Softwareentwicklung wird gefordert, Produkte in kurzer Zeit mit hoher Qualität zu erzeugen. Dieser marktwirtschaftlichen Notwendigkeit wird zunehmend mit dem Konzept der Vorfertigung begegnet. Wie auch in anderen Ingenieurdisziplinen nutzt die Informatik einmal erbrachte Entwicklungsleistungen, um ein gegebenes Problem schneller und besser zu lösen. Durch die Vorfertigung ist die Wiederverwendung von Entwicklungsleistungen in der Zukunft möglich.

In Zusammenhang mit Wiederverwendung wird der Begriff der Komponente in vielen, unterschiedlichen Arbeits- und Wissensgebieten, also Domänen, genutzt und ist daher jeweils leicht unter-

schiedlich definiert. Der Duden definiert eine Komponente ganz allgemein als „Bestandteil eines Ganzen“. In der Softwareentwicklung ist diese Definition jedoch zu vage und wird für die weiteren Ausführungen in dieser Arbeit präzisiert. Es handelt sich dabei um die Definition von [Jaco 1997], die sich auf die *Unified Modeling Language* aus [UML 1999] bezieht, als Sprache des Softwareentwurfs. Durch Diagramme der UML wird ein Abbild der Realität modelliert, welches sich dann in Software umsetzen lässt.

Definition: Komponente nach [Jaco 1997]

> *A component is a high-quality type, class, or other UML workproduct, designed, documented, and packaged to be re-usable. A component is cohesive and has a stable interface. Components include interfaces, subsystems, use-cases, types, actors, or classes and attribute types. Components also include other workproducts, such as templates, or test case specifications.*
>
> Eine Komponente ist qualitativ hochwertig und ist eine Klasse, ein Datentyp oder ein anderes in der UML notiertes Ergebnis, welches entwickelt, dokumentiert und zusammengesetzt wurde, um wiederverwendbar zu sein. Eine Komponente weist einen starken inneren Zusammenhalt auf und hat stabile Schnittstellen. Komponenten enthalten Schnittstellen, Teilsysteme, *Use-cases*, Datentypen, Aktoren oder Klassen und Attribute. Komponenten enthalten auch andere Entwicklungsergebnisse, wie beispielsweise *Templates* der Programmierung oder spezifizierte Testfälle.

Definition: Domäne, domänenspezifisch

> Als Domäne werden bestimmte Arbeits- und Wissensgebiete bezeichnet. Wird Software für eine Domäne entwickelt, sind immer wiederkehrende und für diese Domäne typische Probleme zu lösen. Diese Probleme werden als *domänenspezifisch* bezeichnet.

In der Softwareentwicklung werden komplexe Probleme in überschaubare und handhabbare Teilprobleme zerlegt, deren Lösung nach obiger Definition, als Komponente der Wiederverwendung zugeführt werden kann. Zunächst gibt es Komponenten, die eine allgemein nutzbare Funktionalität zur Verfügung stellen. Hierunter fallen in der Softwareentwicklung beispielsweise einzelne Algorithmen zur Bearbeitung von Daten, ganze Datenbanken, Betriebssystemfunktionen oder andere Werkzeuge beziehungsweise Verfahren der allgemeinen Datenverarbeitung. Diese Komponenten werden von allen Applikationen in der einen oder anderen Form genutzt. In dieser traditionellen Wiederverwendung werden Teile des entstandenen Codes, Klassen oder ganze Anwendungen jeweils an anderer Stelle nochmals genutzt, um Zeit und Kosten zu sparen. Allerdings ist es bei der Erstellung der Komponenten unmöglich, alle Fälle der Wiederverwendung vorherzusehen. Sind

Komponenten für sehr spezielle Anforderungen vorgesehen, sind sie lediglich innerhalb einer bestimmten Anwendungsdomäne einsatzfähig. Sie können nur von einer begrenzten Anzahl Applikationen dieser Domäne genutzt werden. Wiederverwendung wird dann nur eingeschränkt erreicht. Als Beispiel werden Komponenten zur Darstellung von topografischen Informationen eines Kartografiesystems nur von Applikationen genutzt, die sich innerhalb der Domäne der Kartografiesysteme befinden, die somit domänenspezifische Komponenten darstellen.

Schließlich ist für jede Applikation eine Architektur notwendig, welche die Integration der unterschiedlichen Komponenten zu einem Ganzen ermöglichen muss. An dieser Stelle sind der Wiederverwendung Grenzen gesetzt, da die Schnittstellen der Komponenten natürlich zu den Schnittstellen der Architektur passen müssen. Schnell kann der eventuelle Aufwand für die Anpassung einer Komponente die Zeitersparnis durch die Wiederverwendung dahinschmelzen lassen.

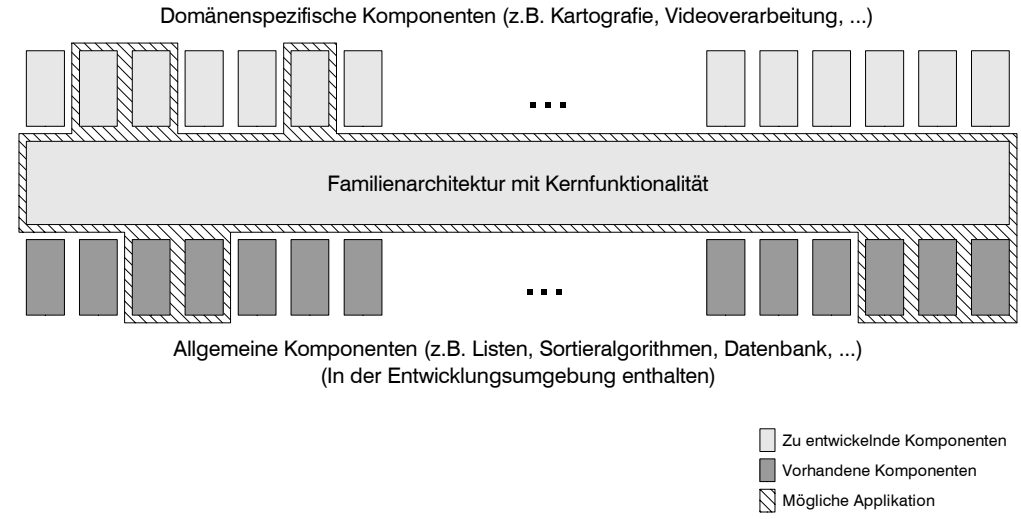


Abbildung 1.1: Idee der Systemfamilien

Die Idee der Systemfamilien wurde bereits in den 70er Jahren von Parnas in [Parn 1976] beschrieben und ist in Abbildung 1.1 dargestellt. Im Verlauf des Softwareentwicklungsprozesses wird zunächst eine zentrale Architektur entwickelt, deren Ziel die Integration existierender und noch zu entwickelnder Komponenten ist, die auch als Variabilitäten bezeichnet werden. Diese Architektur findet sich in jeder aus der Familie abgeleiteten Applikation wieder und weist den höchsten Grad an Wiederverwendung auf. Grundlegende Konzepte der Informatik, wie Listen, Sortieralgorithmen oder Muster werden in Komponenten realisiert, die in den meisten Entwicklungsumgebungen enthalten sind und damit direkt genutzt werden können. Die speziell auf die Problemdomäne zugeschnittenen Komponenten müssen dagegen noch entwickelt, als existierende Teile alter Projekte an die Familienarchitektur angepasst oder in Form von *Commercial-off-the-Shelf*-Produkten (COTS) in das System integriert werden. Dabei handelt es sich um Komponenten, die anstelle einer Neuentwick-

lung eingekauft und in das Zielsystem eingebaut werden, um Zeit und Kosten zu sparen. Eine Applikation wird immer aus der Kernarchitektur und einzelnen Komponenten zusammengesetzt. Dieser Vorgang wird im weiteren Verlauf der Arbeit als Konfiguration eines Systems bezeichnet, wobei für das entstehende System auch die Begriffe Applikation und Familienmitglied synonym verwendet werden. Die im Verlauf der Konfiguration gewählten Komponenten müssen noch für das betreffende Familienmitglied eingestellt, also parametriert werden. In den diversen Komponenten werden die Merkmale eines Systems realisiert, welche von einem bestimmten Kunden oder einer Kundengruppe gewünscht werden. Durch die Merkmale unterscheiden sich dabei die Familienmitglieder.

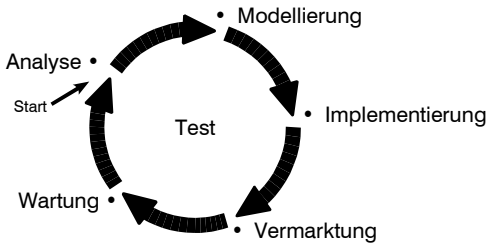


Abbildung 1.2: Phasen der Softwareentwicklung

Im Verlauf eines modernen Softwareentwicklungsprozesses treten in den frühen Phasen häufig Probleme auf. Im Folgenden wird zunächst der allgemeine Softwareentwicklungsprozess erläutert. Die Entwicklung von Software nach modernen Methoden ist durch einzelne Phasen gekennzeichnet, die iterativ abgearbeitet werden. Ergebnisse einer Phase stellen dabei die Datenbasis für die jeweils folgende Phase dar. Die in [Balz 1996], [Shar 1998], [Brue 2000] und [Hoff 1999] beschriebenen Aktivitäten der Softwareentwicklung werden hier in sechs Phasen eingeteilt, die für die Betrachtungen in dieser Arbeit eine ausreichende Abstraktion darstellen, wie in Abbildung 1.2 dargestellt. Die Entwicklung beginnt mit der Analyse eines Problems, das mithilfe eines rechnergestützten Systems zu lösen ist. Der englische Begriff des *Requirements Engineering* wird im weiteren Verlauf dieses Kapitels synonym zum Begriff Analysephase verwendet. In der Analysephase werden Marktanalysen erstellt, um die Machbarkeit und Rentabilität des Projektes zu bewerten und Anforderungen aufgenommen, die vom zukünftigen System erfüllt werden müssen. Darauf folgend wird die in der Realität vorgefundene Problematik durch Modelle beschrieben und damit abstrahiert. Die Modellierung führt zu einer semiformalen Beschreibung des Problembereichs. Schließlich wird das System basierend auf den Modellen implementiert. Die Implementierung stellt eine formale und damit für den Rechner verständliche Abbildung der Lösung für ein Problem dar. Während der Wartungsphase werden auftretende Fehler behoben und neue Anforderungen an das System gesammelt, um gegebenenfalls einen weiteren Entwicklungszyklus für deren Einarbeitung zu starten. Viele der aktuellen Softwareprojekte laufen iterativ ab, sodass die einzelnen Entwicklungsphasen in Zeitabständen von wenigen Wochen abgearbeitet werden. Pro Zyklus werden dabei nur Teile der gesamten Anforde-

rungen realisiert. Nach mehreren, überschaubaren Zyklen steht dann das angestrebte Produkt zur Verfügung, wobei die Vermarktung erst beginnen kann, wenn das Produkt einen genügend großen Funktionsumfang umfasst. Die Zeitspanne bis zum Verkauf des Produktes wird als *Time-To-Market* (TTM) bezeichnet und hat einen direkten Einfluss auf die Entwicklungskosten des Produktes. Neben den Entwicklungskosten kann eine lange Entwicklungszeit dazu führen, dass ein Konkurrent ein ähnliches Produkt früher auf den Markt bringt und damit eigene Bemühungen zunichte macht. Die sechste Phase, in Abbildung 1.2 in der Mitte dargestellt, findet in jeder der vorgenannten Phasen Berücksichtigung. Bereits zu Beginn der Entwicklung muss festgelegt werden, wie Anforderungen überprüft werden können. Diese Prüfvorschriften oder auch Testfälle werden während der Modellierung weiter verfeinert und finden sich in der Implementierung in Form eines zusätzlichen Prüfcodes wieder. In der Wartungsphase können die spezifizierten und implementierten Tests zur Feststellung von unerwünschten Nebeneffekten herangezogen werden, die bei Veränderungen am System auftreten können.

In jeder der genannten Entwicklungsphasen wird durch Wiederverwendung versucht, einen wirtschaftlichen Vorteil zu erreichen. In der von [Nada 1998] durchgeführten Untersuchung hat sich gezeigt, dass 57% der befragten Unternehmen den größten Nutzen aus der Wiederverwendung von Analyseergebnissen ziehen. Der Analysephase kommt aufgrund der oben angesprochenen Abhängigkeit der Ergebnisse der einzelnen Entwicklungsphasen eine strategisch wichtige Bedeutung zu. Daher hat die Wiederverwendung von Ergebnissen aus der Analysephase den größten Einfluss auf die Qualität und Entwicklungsgeschwindigkeit des Produktes. Wie in Abbildung 1.2 dargestellt und im obigen Abschnitt beschrieben, startet die iterative Entwicklung mit der Analysephase. Fehlern in dieser Phase kommt eine besondere Bedeutung zu, da alle anderen Phasen auf den Ergebnissen der Analyse aufbauen. Unerkannte Fehler in der Analysephase vervielfachen ihre negative Wirkung mit dem Fortschreiten des Projektes, weil sie umfangreiche Nacharbeiten erfordern, bis hin zu einer kompletten Neuentwicklung.

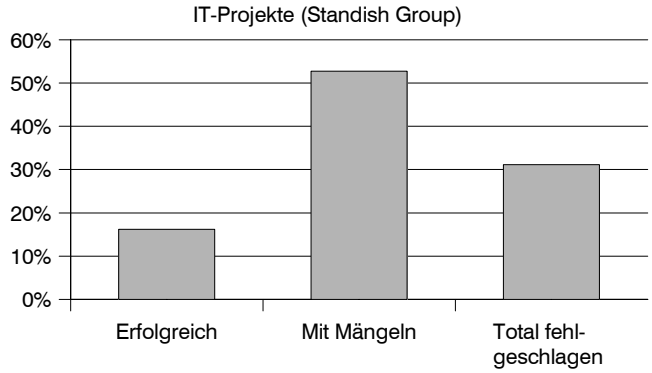


Abbildung 1.3: Ergebnis der Untersuchung von Softwareprojekten

Im Rahmen der CHAOS-Studie [CHAO 1995], dargestellt in Abbildung 1.3, hat sich herausgestellt, dass ein großer Teil der US-amerikanischen Softwareentwicklungsprojekte fehlschlug. Bei der Untersuchung der Gründe für die Misserfolge zeigte sich, dass bei über 20% der befragten Firmen fehlende oder sich ändernde Anforderungen Grund für die Probleme waren. Anforderungen werden in der Analysephase erhoben und verarbeitet. Veränderungen der Anforderungen ziehen die erneute Bearbeitung der Analysephase nach sich. Daher lassen sich aus den Ergebnissen der CHAOS-Studie Mängel in der Analysephase der Softwareentwicklung ableiten.

Eine europäische Studie, deren Ergebnisse in [ESPI 1996] veröffentlicht wurden, bestätigt die Probleme moderner Softwareentwicklung in der Analysephase. Die überwiegende Mehrheit der 3805 im Rahmen dieser Studie befragten Unternehmen haben verstärkt Probleme im Bereich der Anforderungsanalyse angegeben. Essenzielle Probleme bei der Anforderungsspezifikation werden von 55% der Befragten konstatiert und die Verwaltung von Kundenanforderungen stellt für über 50% der Unternehmen ein ernst zu nehmendes Problem dar. Die Umsetzung der Ergebnisse der Analysephase in einen Entwurf verursacht bei mehr als der Hälfte der Unternehmen immer noch kleinere Probleme. Erst während der Implementierungsphase nehmen die ernsten Probleme ab.

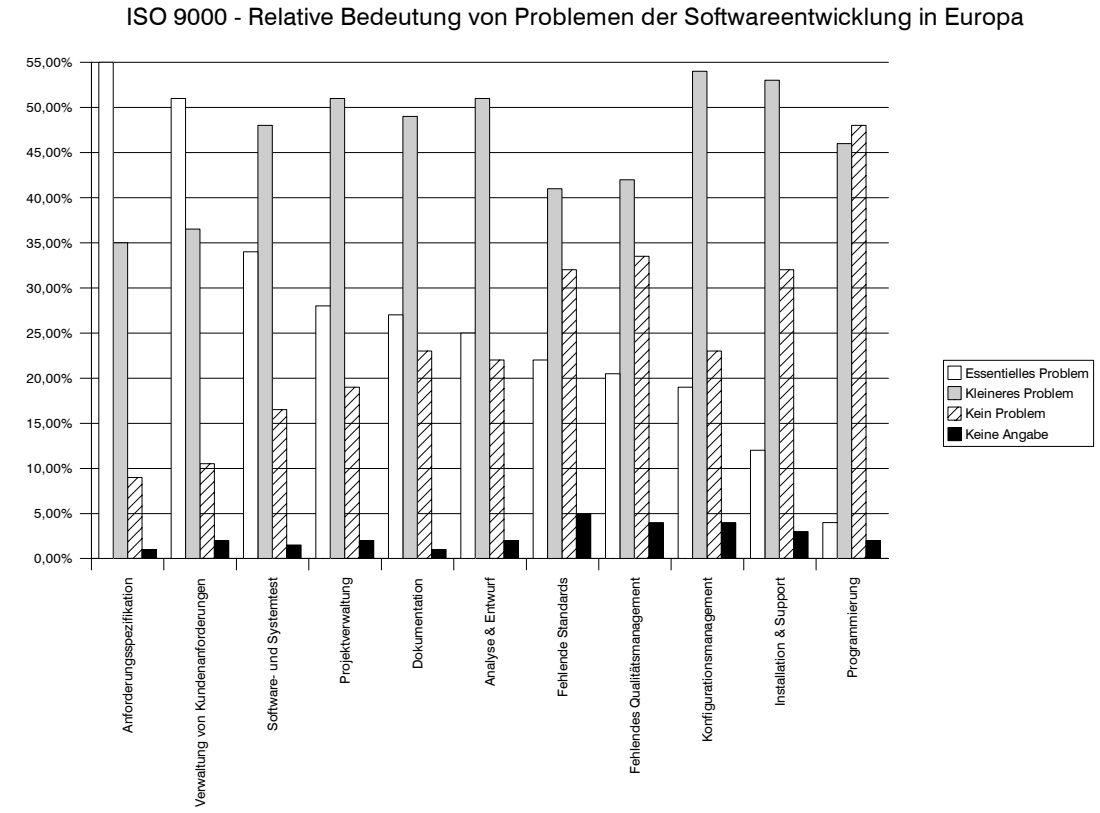


Abbildung 1.4: ESPITI-Untersuchung

Abbildung 1.4 zeigt die zusammengefassten Ergebnisse der ESPITI-Studie. Für die Entwicklung von Systemfamilien haben die gezeigten Studien eine große Bedeutung, da die Anforderungsanalyse in der Systemfamilienentwicklung für viele zukünftige Applikationen genutzt werden soll und damit Fehler noch weitreichender sind, als dies bei Einzelsystementwicklungen der Fall ist. Die Ergebnisse der Anforderungsanalyse werden wiederverwendet und müssen daher korrekt sein. Die obigen Studien zeigen auch, dass die Komplexität aktueller Softwareprojekte einen hohen Zeit- und Ressourcenaufwand erfordert, um eine gute und korrekte Anforderungsanalyse durchzuführen. Der hohe Aufwand an Zeit und Ressourcen erfordert ein methodisches Vorgehen, um die Entwicklung vorhersagbar zu gestalten und Optimierungspotenziale wahrnehmen zu können. Nach obigen Studien haben viele Unternehmen in der Analysephase Probleme.

Die Ergebnisse der Analysephase der Systemfamilienentwicklung zielen auf die Variabilitäten und Gemeinsamkeiten der Architektur der Familie ab. Eine korrekte Anforderungsanalyse im Sinne der Systemfamilienentwicklung bedeutet, ein richtiges und dauerhaftes Modell der Variabilitäten und Gemeinsamkeiten der Familie zu erarbeiten und damit die Basis für eine gute Architektur der Systemfamilie zu legen.

1.2 Problemstellung

Im Umfeld der Softwareentwicklung ist die Wiederverwendung eine Möglichkeit zur Steigerung der Effizienz und Qualität. Mithilfe definierter Vorgehensweisen wird versucht die Wiederverwendung im Entwicklungsprozess zu verbessern. Durch Systemfamilien kann Wiederverwendung auf einem abstrakten Niveau betrieben werden. Produktübergreifend können Entwicklungsleistungen immer wieder genutzt werden, wodurch sowohl die Entwicklungsgeschwindigkeit als auch die Produktqualität positiv beeinflusst werden. Aus Erfahrungen im Rahmen von Industrieprojekten ist die verkürzte Zeit bis zur Auslieferung einer Applikation (*Time-To-Market*) das Hauptargument für die Nutzung des Systemfamilienansatzes.

Dennoch schlägt ein Großteil der Softwareprojekte fehl, wie die vorgestellten Studien gezeigt haben, was hauptsächlich an der mangelhaft berücksichtigten Analysephase liegt. Es wird nicht genug Zeit und Aufwand in die Analysephase investiert, das methodische Vorgehen ist nur vage umrissen und die Ergebnisse der Analysephase stehen kaum mit den Ergebnissen der folgenden Phasen in Verbindung. Die Ergebnisse der Analysephase beeinflussen die nachfolgenden Phasen der Softwareentwicklung jedoch ganz entscheidend und damit auch den Erfolg von Wiederverwendung. Missverständnisse, Fehler und Versäumnisse in dieser frühen Phase führen zu schweren Mängeln im entstehenden Produkt oder verhindern gänzlich dessen Fertigstellung, wie die ESPITI-Studie gezeigt hat.

Im Rahmen dieser Arbeit werden die Verfahren und Methoden der Anforderungsanalyse für systemfamilienbasierte Softwareentwicklung untersucht. Die Mängel der existierenden Ansätze werden identifiziert, um die in Kapitel 1.1 genannten Probleme beheben zu können. Folgende Fragestellungen liegen den Untersuchungen bestehender Methoden und Technologien im nächsten Kapitel zugrunde.

- Wie unterstützen herkömmliche *Requirements-Engineering*-Methoden die Systemfamilienentwicklung?
- Welche Methoden zur Systemfamilienentwicklung gibt es und wie unterstützen sie die Analysephase?
  Wie wird diese Entwicklungsphase von den existierenden Methoden der Systemfamilienentwicklung unterstützt und welche Erweiterungen oder Änderungen bieten diese Methoden gegenüber der Softwareentwicklung für Einzelsysteme?
- Welche Ergebnisse erzeugt die Analysephase der Systemfamilienentwicklung?
  Wie werden diese organisiert und in welcher Beziehung stehen sie zu den weiteren Ergebnissen der Softwareentwicklung?
- Wie können die Ergebnisse der Analysephase wiederverwendet werden?
  Was leisten aktuelle Methoden in Bezug auf die Wiederverwendung der in der Analysephase erzeugten Entwicklungsergebnisse? Wie wird von den einmal gemachten Erfahrungen in zukünftigen Projekten profitiert?

Die Ergebnisse der Untersuchungen obiger Fragen fließen in eine neue Methode für die *Requirements-Engineering*-Phase der Systemfamilienentwicklung ein. Die Methode ist Gegenstand dieser Arbeit und besteht aus einem Datenmodell, einer Prozessbeschreibung und einem Kernanteil, der erweiterten Merkmalmodellierung.

1.3 Organisation der Arbeit

In Kapitel 2 wird das digitale Video Projekt als durchgängig genutztes Beispiel vorgestellt. Neben einer kurzen Einführung in das Projekt mit dessen Umfeld wird eine Liste mit allgemeinen Anforderungen an das Projekt dargestellt, die im weiteren Verlauf als Basis für die vorgestellten *Requirements-Engineering*-Methoden und die später erläuterte, neue Methode dient.

In Kapitel 3 wird der derzeitige Stand der Technik umrissen und jeweils am konkreten Beispiel erläutert. Die Methoden der Anforderungsanalyse im Allgemeinen und speziell im Systemfamilienumfeld werden beschrieben. Dabei stellt die Merkmalmodellierung im Bereich der Systemfamilien

eine Kernmethode der Modellierung dar. Es werden jeweils die Schwächen der vorgestellten Ansätze dargelegt, um daraus am Ende des Kapitels die präzisierte Problemstellung abzuleiten.

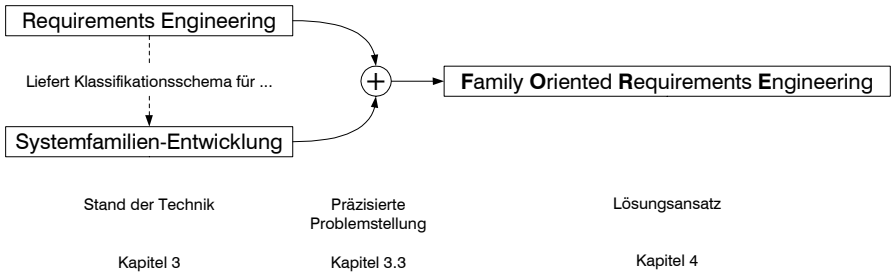


Abbildung 1.5: Übersicht der Kapitel 3 und 4

Kapitel 4 enthält den eigenen Lösungsansatz, *Family-Oriented Requirements Engineering* (FORE), zu der formulierten, präzisierten Problemstellung. In Abbildung 1.5 ist dargestellt, wie aus den Erkenntnissen der Analyse des Standes der Technik eine präzisierte Problemstellung und daraus der Lösungsansatz entstehen. Den Kernanteil der Lösung stellt die erweiterte Merkmalmodellierung dar, die eine automatisierte Verarbeitung der *Requirements-Engineering*-Daten erlaubt. Sie ist in einen dazugehörigen Entwicklungsprozess eingebettet, der alle notwendigen Schritte in der *Requirements-Engineering*-Phase der Systemfamilienentwicklung beschreibt. Alle Ergebnisse der Arbeiten dieser Phase werden schließlich in einem eigenen Datenmodell abgelegt, das am Ende des vierten Kapitels erläutert wird.

Die in Kapitel 5 dargestellte, prototypische Implementierung beschreibt die Umsetzung des Lösungsansatzes. Das eigene Datenmodell mit den erweiterten Merkmalmodellen wird mithilfe der *Extensible Markup Language* (XML) beschrieben. Durch einen eigenen Compiler wird die automatisierte Verarbeitung des FORE-Datenmodells anhand der vorgestellten FORE-Methode prototypisch realisiert.

In Kapitel 6 wird der eigene Ansatz kritisch beleuchtet. Die Ergebnisse werden bewertet und es wird eine Überprüfung der Zielvorgaben durchgeführt. Im anschließenden Kapitel 7 wird die Arbeit zusammengefasst und durch einen Ausblick auf zukünftige, aus dieser Arbeit entstandene Forschungsthemen, abgeschlossen.

Kapitel 2

M.C. Escher, „Belvedere“, 1958

Fallbeispiel: Digitales Videoprojekt

Am Beispiel eines digitalen Videosystems wurde der Lösungsansatz dieser Arbeit überprüft, weiterentwickelt und bewertet. In Kapitel 4 wird der Lösungsansatz vorgestellt und erläutert. Dieses Kapitel stellt das System vor und beschreibt dessen wichtigste Eigenschaften. Für beispielhafte Erläuterungen im weiteren Verlauf dieser Arbeit wird das **D**igitale **V**ideo**P**rojekt (DVP) zugrunde gelegt.

2.1 Einleitung

Als Beispiel einer Systemfamilie wurde im Rahmen der Forschungen zu dieser Arbeit das digitale Videoprojekt aufgesetzt und aus existierenden Soft- und Hardwarekomponenten aufgebaut. Das Projekt dient der praktischen Umsetzung, der Überprüfung und der Bewertung der in dieser Arbeit vorgestellten Methode. Wie ein herkömmlicher Videorecorder kann auch dieses System Fernsehsendungen aufzeichnen und wiedergeben. Im Unterschied zum herkömmlichen, analogen Fernsehen und herkömmlichen Videorekordern werden die Sendungen hier als digitaler MPEG (*Motion Picture Encoding Group*) Videodatenstrom bereitgestellt und aufgezeichnet. Diese digitalen Fernsehdaten werden über eine Satellitenschüssel mit Hilfe von DVB-Karten (*Digital Video Broadcasting*) empfangen und können nunmehr als Datei abgelegt werden, z.B. auf einer Festplatte. Mit der heutigen Rechentechnik können digitale Videodaten flexibel und einfach weiter verarbeitet werden.

Die Analyse existierender Systeme ergab eine jeweils pro Hersteller begrenzte Produktvielfalt. Jeder Hersteller bietet eine Produktpalette an, die seiner Meinung nach die meisten Kunden zufrieden stellen wird. Damit kann ein Kunde nur aus den aktuell am Markt befindlichen Systemen wählen, wobei er seine Wünsche den Funktionen und Eigenschaften der jeweiligen Geräte unterordnen

muss. Es ist nicht möglich, die Eigenschaften der Geräte eines Herstellers zu kombinieren oder gar eine herstellerübergreifende Kombination zu erwerben. Das Ziel der Systemfamilienentwicklung ist, einem Kunden die flexible Auswahl der Eigenschaften bzw. Merkmale seines gewünschten Gerätes zu ermöglichen. Am Beispiel des digitalen Videoprojektes wurden existierende Ansätze der Systemfamilienentwicklung überprüft und bewertet. Das digitale Videoprojekt wurde aus vorhandenen Komponenten aufgebaut und zur Systemfamilie weiterentwickelt. Im weiteren Verlauf der Arbeit beziehen sich alle Beispiele auf dieses System.

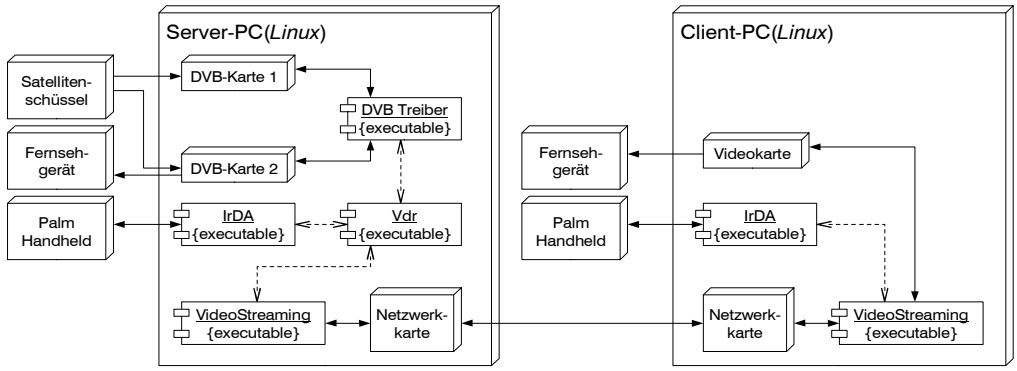


Abbildung 2.1: Grundstruktur des DV-Projektes

In Abbildung 2.1 ist der prinzipielle Aufbau des Prototyps mit Soft- und Hardwarekomponenten dargestellt. Das prototypische System besteht aus zwei identischen Rechnern, die jeweils mit zwei DVB-Karten ausgestattet sind. In der Darstellung aus Abbildung 2.1 werden die beiden Rechner anhand ihrer Funktion unterschieden. Zum einen gibt es den Server-PC, der die komplette Verarbeitung der Videosignale von der Satellitenschüssel übernimmt, zum anderen steht auch ein *Client-PC* zur Verfügung, der nach dem *Video-On-Demand*-Prinzip Videoströme vom Server anfordern und darstellen kann. Auf der linken Seite ist die Satellitenschüssel dargestellt, an die beide DVB-Karten des Servers angeschlossen sind. Über den DVB-Treiber von [Metz 2002] werden die Karten angesteuert. Als Basissystem wurde das *Video-Disk-Recorder*-System (VDR) genutzt, welches von [Schm 2002] stammt. Auf der Seite des Servers werden die Videodaten über den TV-Ausgang einer DVB-Karte ausgegeben. Durch eine Besonderheit der benutzten DVB-Karten ist die Einblendung von Zusatzinformationen möglich, um eine Rückkopplung bei der Bedienung des Gerätes zu erhalten. Die Bedienung des Systems erfolgt mit Hilfe eines *Handheld*-Computers. Durch Integration der Ergebnisse des IrDA-Projektes von [Lirc 2002] ist diese Steuerung möglich. Um die Variantenvielfalt der entstehenden Systemfamilie zu erhöhen, wurde die Verteilung der Videodaten vom Server-PC aus realisiert, indem das *Video-Streaming*-Projekt von [Lata 2002] durch die Diplomarbeit von [Meff 2003] in das digitale Videoprojekt integriert wurde.

Im Vordergrund des Videoprojektes stand die Idee eine Umgebung bereitzustellen, die es am konkreten Beispiel ermöglicht, Entwicklungsmethoden, Strategien und Implementierungsideen aus dem Systemfamilienumfeld zu testen. Da auf allen Ebenen der Systementwicklung gearbeitet werden musste und somit auch der jeweilige Quellcode erforderlich war, wurde die offene Linux-Plattform gewählt. Darüber hinaus liegen auch die verwendeten Komponenten im Quellcode vor.

2.2 Analyse des Umfeldes

Im ersten Schritt wurde eine Analyse des Umfeldes, also der Domäne, durchgeführt, um die speziellen, technologischen Hintergründe und Grundlagen der digitalen Ver- und Bearbeitung von Videomaterial zu untersuchen. Darüber hinaus wurden Merkmale existierender und potentieller, zukünftiger Produkte mittels Marktanalysen und Umfragen erarbeitet, um als Grundlage für die zu entwickelnde Systemfamilie zu dienen.

Zunächst wurde aus allgemein verfügbaren Quellen wie Internet, einschlägigen Zeitschriften und Informationen aus dem Fachhandel, eine Liste von Anforderungen erarbeitet. Verfeinert wurden diese Anforderungen durch die Analyse der unter Linux bereits bestehenden und in Abbildung 2.1 dargestellten Systeme. Eine Liste der Anforderungen für das digitale Videoprojekt ist in Anhang A enthalten. An dieser Stelle sind nur die wichtigsten Anforderungen aufgelistet, die jeweiligen Verfeinerungen sind dem Anhang zu entnehmen. Darüber hinaus sind die Anforderungen, die die Wartbarkeit oder Skalierbarkeit betreffen, nicht enthalten, da sie im weiteren Verlauf der Entwicklung erarbeitet werden müssen, an dieser Stelle jedoch keine Relevanz haben.

Tabelle 2.1 enthält einen Auszug der Anforderungen an das Videosystem. Die Nummerierung der Anforderungen repräsentiert gleichzeitig deren hierarchische Anordnung, wobei an dieser Stelle nur Anforderungen der höchsten Ebene enthalten sind, die Anforderung 1.1 oder 2.2 sind nicht enthalten. In der zweiten Spalte der Tabelle sind die in der Anforderungserhebungsphase erkannten Beziehungen hinterlegt. Dabei handelt es sich um Beziehungen, denen technologische Randbedingungen zugrunde liegen. Beispielsweise ist die *Picture-In-Picture*-Funktion nur möglich, wenn zwei Empfänger, also DVB-Karten, im System vorhanden sind.

Nr.	Anforderung	Beziehung zu ...
1.	Das System soll fernsteuerbar sein.	
2.	Das System muss Audio- und Video-Datenmaterial abspielen können.	
3.	Das System muss eine elektronische Fernsehzeitung enthalten.	
4.	Einstellungen sollen individuell anpassbar sein.	
5.	Das System soll die „Picture-In-Picture“ Funktion unterstützen.	Benötigt zwei Empfänger!

Nr.	Anforderung	Beziehung zu ...
6.	Mit dem System müssen Sendungen angehalten und zeitversetzt betrachtet werden können. („*time-shifting*“)	Die Festplatte muss entsprechend leistungsfähig sein.
7.	Das System soll Werbeblöcke ausblenden können.	
8.	Das System soll Videomaterial nachbearbeiten können.	
9.	Das System soll als Video-On-Demand Server agieren.	
10.	Das System soll mehrere Tonformate unterstützen.	Andere Formate müssen entsprechend angepasst, „heruntertransformiert“ werden.
11.	Das System soll Pay-TV unterstützen.	
12.	Das System soll Videoeingänge haben.	
13.	Das System soll eine automatische Senderprogrammierung haben.	
14.	Das System soll Fehler anzeigen.	siehe auch 15.
15.	Das System soll unterschiedliche Sprachen unterstützen.	
16.	Das System soll Videomaterial auch auslagern können.	
17.	Das System soll Zusatzgeräte ansteuern können.	
18.	Das System muss leise sein.	
19.	Nutzer sollen sich gegenüber dem System authentifizieren.	

Tabelle 2.1: Anforderungen aus Prospekten und dem Internet

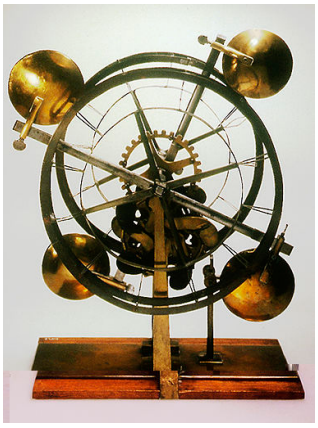
Prospekte beschreiben ein Produkt anhand von Merkmalen, an denen sich Kunden orientieren können. Basierend auf diesen Merkmalen sollen Kunden ein Produkt verstehen und auswählen. Während der Anforderungserhebung wurden diverse Merkmale aus den zur Verfügung stehenden Prospekten zusammengetragen.

Nr.	Merkmal	Erläuterung
1.	DVD	Das System kann DVDs abspielen.
2.	Video-CD	Das System kann Video-CDs abspielen.
3.	Musik-CD	Das System kann Musik-CDs abspielen.
4.	Werbung	Das System kann Werbeblöcke erkennen und herausschneiden.
5.	Externe Daten	Das System kann Videomaterial von Camcordern aufnehmen.
6.	Infrarot	Das System ist durch jede beliebige Infrarot-Fernbedienung steuerbar.
7.	Handheld	Das System ist durch einen Handheld steuerbar.
8.	*Video On-Demand*	Das System kann Videodaten in einem Haushalt verteilen.
9.	Diashow	Das System kann digitale Dias anzeigen.
10.	Kanalprofil	Das System hält Einstellungen für jeden Sender separat vor.
11.	Nutzerprofil	Das System kann mehrere Nutzer verwalten.
12.	*Picture-In-Picture*	Das System kann gleichzeitig zwei Kanäle anzeigen.
13.	Zugangskontrolle	Das System kann Nutzer authentifizieren.
14.	*Electronic Program Guide*	Das System bietet eine elektronische Programmzeitschrift an.
15.	Internet	Das System ist über das Internet steuerbar.
16.	Telefon	Das System ist über das Telefon steuerbar.

Tabelle 2.2: Merkmale des digitalen Videosystems

Darüber hinaus wurden weitere Merkmale in Gruppendiskussionen erarbeitet. Die wichtigsten Merkmale sind in Tabelle 2.2 zusammengefasst, während die komplette Liste der Merkmale in Anhang-A enthalten ist. Das zukünftige, digitale Videosystem kann mit den oben angegebenen Merkmalen viele unterschiedliche Audio- und Videodatenströme be- und verarbeiten.

Kapitel 3



Modell des Perpetuum Mobile von *Leonardo da Vinci*

Stand der Technik

In diesem Kapitel werden die für die vorliegende Arbeit relevanten Technologien und Methoden vorgestellt, erläutert und in einen Gesamtzusammenhang gestellt. Zunächst werden Techniken des *Requirements Engineering* vorgestellt, um einen Rahmen für die Analyse und Bewertung der Methoden der Systemfamilienentwicklung zu schaffen.

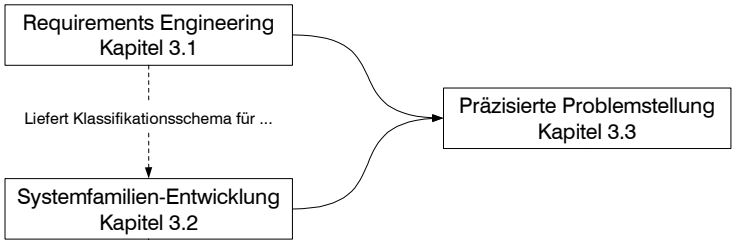


Abbildung 3.1: Übersicht der Arbeit

In Abbildung 3.1 sind die nächsten Kapitel im Überblick zusammengefasst. Im folgenden Unterkapitel werden Anforderungen an die *Requirements-Engineering*-Phase abgeleitet, die als Klassifikationsschema im nächsten Kapitel eine Bewertung der Methoden und Techniken der Systemfamilienentwicklung erlauben. Probleme und Mängel der Ansätze werden jeweils erörtert und am Ende des Kapitels in die präzisierte Problemstellung eingearbeitet. Diese Problemstellung ist die Grundlage für den im nächsten Kapitel vorgestellten Lösungsansatz, *Family-Oriented Requirements Engineering*.

3.1 Requirements Engineering

Bevor ein System entwickelt und implementiert werden kann, müssen sich alle Projektbeteiligten (*Stakeholder*) darüber im Klaren sein, was genau zu entwickeln ist. Nach [Louc 1995] sollen in der *Requirements-Engineering*-Phase die Bedürfnisse der Benutzer verstanden und als präzise, vollständige Ausdrücke formuliert werden. Daraus entsteht eine Anforderungsspezifikation, welche die erhobenen Anforderungen in einem Modell zusammenfasst. Schließlich sollen in dem Spezifikationsdokument Prüfvorschriften abgelegt werden, durch welche die Entsprechung des entwickelten Systems mit den erhobenen Anforderungen überprüft werden kann. Das Erheben der Anforderungen ist damit eine hochgradig intellektuelle Tätigkeit. Es gilt eine Barriere zu überwinden, zwischen dem Wissen der *Stakeholder*, die das System entwickeln sollen und den Personen, die Anforderungen an das System stellen. Die Entwicklung eines Systems erfordert das Verständnis eines gegebenen Problems und dieses Verständnis kann nur aufgebaut werden, wenn die Problemdomäne verstanden worden ist. Je nach Vorkenntnissen kann das zu einer langen Einarbeitungszeit für die Entwickler führen. Wie schon in der Einleitung erwähnt, ist das mangelnde Verständnis der Probleme von Kunden einer der Hauptgründe für das Scheitern von Softwareprojekten.

Die Entwicklung eines Softwaresystems teilt sich in mehrere iterativ zu bearbeitende Phasen auf. Am Beginn steht die Analysephase, die in der Literatur auch als Systemanalyse [Balz 1996], Anforderungsanalyse oder als *Requirements-Engineering*-Phase bezeichnet wird. Der aus dem Englischen übernommene Terminus *Requirements Engineering* hat sich als umfassender Begriff etabliert und wird daher auch in dieser Arbeit genutzt. Zentrales Element dieser Entwicklungsphase ist die Anforderung (englisch *requirement*) selbst, die zunächst in einer informellen Form, beispielsweise als Text, aufgenommen wird. Die meisten Autoren beziehen sich auf die im IEEE 610.12 Standard enthaltene Definition.

Definition – Anforderung nach IEEE 610.12:

> „*A requirement is a (1) condition or capability needed by a user to solve a problem or achieve an objective; (2) condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents; (3) documented representation of a condition or capability as in (1) or (2).*“
>
> Eine Anforderung ist eine (1) von einer Person benötigte Bedingung oder Fähigkeit, um ein Problem zu lösen oder ein Ziel zu erreichen; (2) eine Bedingung oder Fähigkeit, die ein System oder eine der Komponenten erfüllen bzw. aufweisen muss, um einem Vertrag, einem Standard, einer Spezifikation oder anderen formal festgesetzten Dokumen-

ten zu genügen; (3) eine dokumentierte Form einer Bedingung oder Fähigkeit, wie in (1) bzw. (2) beschrieben.

Anforderungen beschreiben ein System aus der Sicht einer Person, die stellvertretend für beliebige, am Projekt beteiligte Personen, die *Stakeholder*, steht. Ein gegebenes Problem der *Stakeholder* wird durch deren Anforderungen beschrieben und soll mit Hilfe eines Soft- und Hardwaresystems gelöst oder vereinfacht werden, wobei dieses System aus mehreren Komponenten bestehen kann und in ein gegebenes Umfeld integriert werden muss.

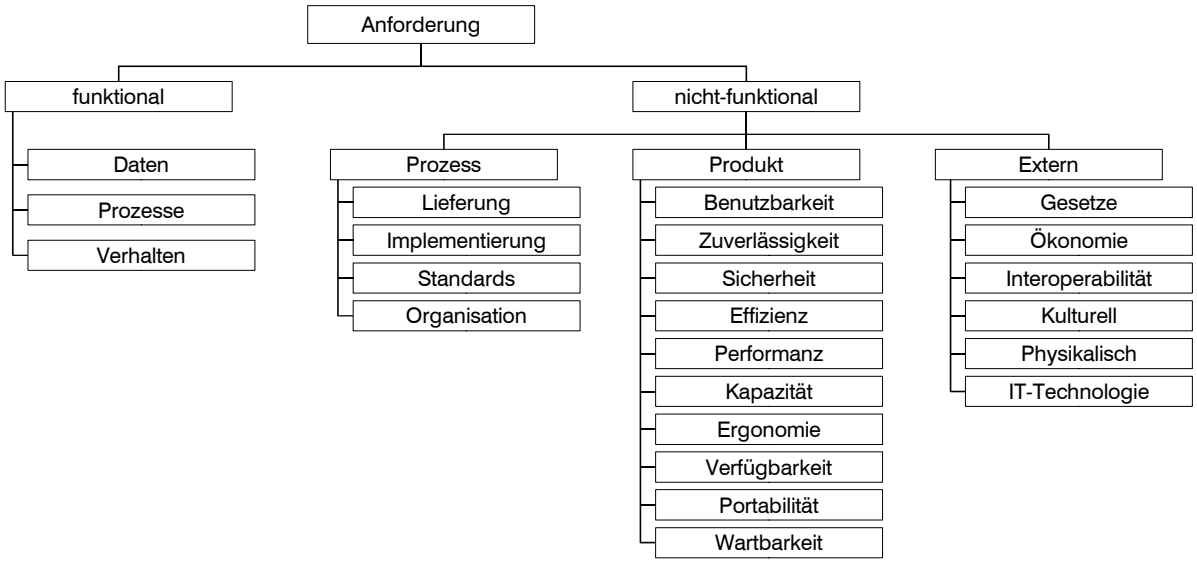


Abbildung 3.2: Anforderungstypen

Wie in [Schi 2002] und [Hofm 2000] beschrieben und in Abbildung 3.2 zusammengefasst, wird zwischen zwei Kategorien von Anforderungen unterschieden. Zum einen können funktionale Anforderungen an das zu entwickelnde System gestellt werden. Diese Anforderungen beschreiben die Funktionen, welche das System aus Sicht eines Benutzers anbieten soll, womit das Verhalten des Systems beschrieben wird. Darüber hinaus werden in dieser Kategorie die Daten bzw. Datenstrukturen beschrieben, die das zu erstellende System verarbeiten soll. Zusammen mit den Abhängigkeiten und Zuständen werden die Daten als Entitäten bezeichnet. Schließlich werden auch die Prozesse, welche die gegebenen Daten jeweils in kleinen Schritten verarbeiten, zu der Kategorie der funktionalen Anforderungen gezählt. In der zweiten Kategorie befinden sich die nicht-funktionalen Anforderungen. Hier werden die das System umgebenden Faktoren mit ihren jeweiligen Wirkungen auf das System beschrieben. Firmeninterne Prozessvorgaben, Standards, gesetzliche Rahmenbedingungen und qualitative Anforderungen, wie Benutzbarkeit, Sicherheit oder Effizienz müssen berücksichtigt werden. Mit den meisten Spezifikationssprachen können nicht-funktionale Anforderungen gar nicht oder nur unzureichend modelliert werden, in [Lams 2000] werden die Umgebungsfaktoren gar nicht als

Anforderungen im Rahmen des *Requirements Engineering* angesehen. Diese Faktoren werden vielmehr als Eigenschaften der Domäne betrachtet, die aber letztlich auch von dem zu entwickelnden System berücksichtigt werden müssen, wodurch diese Unterscheidung eine Typisierung von Anforderungen darstellt. Vorgehensweisen und Modelle zur Be- und Verarbeitung der vorgestellten Anforderungstypen werden in den folgenden Unterkapiteln thematisiert.

Ein wichtiger Punkt im Rahmen der *Requirements-Engineering*-Phase sind die so genannten *Stakeholder,* die in drei Hauptkategorien eingeteilt werden. In Abbildung 3.3 sind die drei Kategorien, als Zusammenfassung der in [KeWi 1997], [KoSo 1998] und [Kula 2000] erläuterten Kategorien dargestellt. Zusätzlich sind auch die jeweiligen Rollen der *Stakeholder* dargestellt, die im Rahmen des *Requirements Engineering* von Bedeutung sind. Die Zuordnung der einzelnen Rollen zu Personen und beteiligten Unternehmen muss für jedes Projekt individuell vorgenommen werden. Dabei ist von einer Verteilung auf drei Unternehmen, mit Auftraggeber, Auftragnehmer und den Benutzern, bis hin zu der Vereinigung aller *Stakeholder*-Rollen in einer einzigen Person eines sehr kleinen Softwarehauses alles möglich.

Die Gruppe der Auftraggeber enthält all diejenigen Personen, die ein festgestelltes Problem mit Hilfe eines Softwaresystems beheben wollen und die Lösung dieses Problems einem Auftragnehmer übertragen. Diese Gruppe hat ein monetäres Interesse an dem Verkauf oder Weiterverkauf des Produktes. Es handelt sich um Personen aus der Managementebene des Auftraggebers, die in Abbildung 3.3 unter dem Begriff des Kunden zusammengefasst sind.

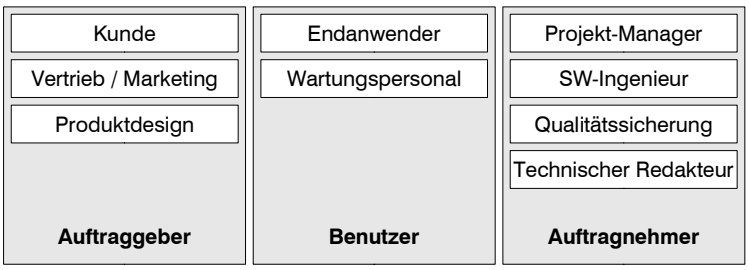


Abbildung 3.3: Die verschiedenen Rollen der *Stakeholder*

Die allgemein gehaltenen Problembeschreibungen des Auftraggebers werden von Personen aus der Benutzergruppe verfeinert. Sie beschreiben die Fähigkeiten, die ein System aufweisen muss, um sie bei der Lösung ihrer Probleme zu unterstützen. In [Maca 1996] wird noch zwischen Gelegenheitsnutzern und häufigen Nutzern unterschieden, wobei alle Benutzer an der Lösung eines Problems durch das zu entwickelnde System interessiert sind oder dazu angehalten werden, Arbeitsabläufe unter Zuhilfenahme des Systems zu verbessern. Auch die Wartung des Systems ist der Benutzergruppe zugeordnet, da sie letztlich auch Funktionen des System nutzen, um den Endanwendern die Arbeit mit dem Produkt zu ermöglichen.

Schließlich besteht die Gruppe der Auftragnehmer aus Personen, die fähig sind das Problem mit Mitteln der Softwaretechnik zu lösen und damit direkt an der eigentlichen Entwicklung des Produktes beteiligt sind. Die Gruppe enthält Projekt-Manager in leitender Funktion, die Gruppe der Software-Ingenieure, die mit allen im Rahmen der konkreten Entwicklung anfallenden Aufgaben betraut sind, Personen aus den Qualitätssicherungsabteilungen und technische Redakteure, die erst seit wenigen Jahren einen neuen Berufszweig bilden und mit allen Tätigkeiten im Umfeld der Dokumentation des Produktes und des Projektes betraut sind. Die mit der Funktion eines Auftragnehmers verbundene berufliche Erfahrung führt zu weiteren Anforderungen an das zu entwickelnde System.

Requirements Engineering muss den ersten Schritt der Formalisierung eines in natürlicher Sprache beschriebenen Problems gehen. Am Anfang einer Produktentwicklung stehen frei formulierte und rein informelle Anforderungen, die beschreiben, was ein mögliches Softwaresystem leisten soll. Am Ende steht ein Softwareprodukt, dem eine völlig formale Beschreibung in einer Programmiersprache zugrunde liegt. Zunächst startet eine Produktentwicklung mit der Planungsphase, deren Ziel die Bewertung der Durchführbarkeit einer Entwicklung ist. Wie in [Balz 1996] beschrieben, werden dazu Trendstudien, Marktanalysen, Kundenbefragungen und eine Vorentwicklung durchgeführt, um letztlich eine Durchführbarkeitsstudie (*feasibility study*) zu erstellen. In dieser Phase werden erste Anforderungen erhoben, um sie zusammen mit der Durchführbarkeitsstudie in einem Lastenheft zu dokumentieren. Das Lastenheft stellt ein informelles Dokument dar, das nach dem VDI/VDE 3694 Standard beschreibt, was für ein Problem zu lösen ist und in welchem Anwendungsgebiet das Problem zu lösen ist. Anhand dieses Dokumentes muss entschieden werden, ob das Produkt tatsächlich entwickelt werden soll bzw. ob sich die Entwicklung lohnt. Das Lastenheft soll alle Anforderungen des Auftraggebers enthalten und beiden Seiten als Vertragsgrundlage dienen, um einen rechtsgültigen Vertrag über die Entwicklung eines Softwaresystems zwischen Auftraggeber und Auftragnehmer zu schließen. Bei einer positiven Entscheidung wird das Lastenheft verfeinert und zum Pflichtenheft ausgebaut. Das Pflichtenheft enthält das Lastenheft und fügt die technischen Details zur Umsetzung der Anforderungen hinzu. Nach dem VDI/VDE 3694 Standard enthält das Pflichtenheft Informationen darüber, wie Anforderungen umzusetzen sind und womit sie umgesetzt werden können.

In Abbildung 3.4 sind die beiden Ergebnisse der *Requirements-Engineering*-Phase dargestellt, die nach [Balz 1996] zwei Meilensteine dieser Phase kennzeichnen. Die einzelnen Tätigkeiten dieser Phase führen jeweils zu Teildokumenten, die zunächst in einem Lastenheft, später in einem Pflichtenheft zusammengefasst werden und das zukünftige Produkt spezifizieren. Aus eigener Erfahrung ist diese strikte Trennung der beiden Dokumente in der Praxis nur selten zu finden. Stattdessen entsteht mit beiderseitiger Beteiligung, der Auftragnehmer und der Auftraggeber, ein Spezifikationsdokument, welches die alleinige Grundlage für einen Vertrag zur Entwicklung des Systems bildet.

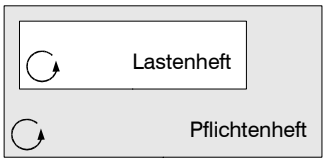


Abbildung 3.4: Ergebnisse der *Requirements-Engineering*-Phase

Wie in [Wieg 1999] und [Rupp 2002] beschrieben, besteht die *Requirements-Engineering*-Phase aus der Entwicklung und der Verwaltung von Anforderungen. Damit werden zum einen die initialen Vorgänge adressiert, um die Anforderungen zu erheben, zu modellieren und zu prüfen. Zum anderen geht es darum, Vorgänge zu berücksichtigen, die Änderung von Anforderungen, die Versionierung, die Beziehung der Anforderungen zu den weiteren Entwicklungsdaten und den Status jeder Anforderung beschreiben.

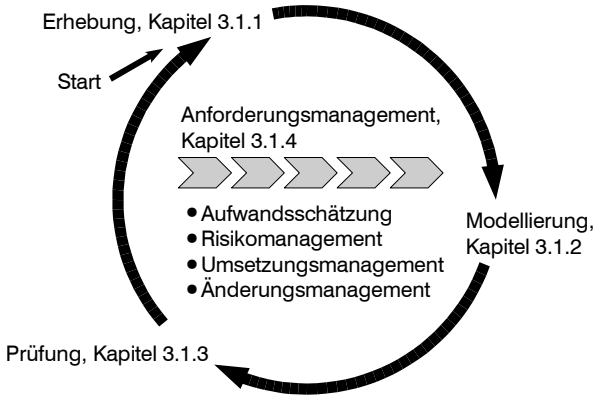


Abbildung 3.5: Prozesse des *Requirements Engineering*

In Abbildung 3.5 ist der Überblick über die Teilprozesse innerhalb der *Requirements-Engineering*-Phase dargestellt, der eine Zusammenfassung der in [Schi 2002], [KoSo 1998] und [Rupp 2002] dargestellten Ergebnisse ist. Der gesamte Prozess beginnt mit der Erhebung der Anforderungen, fährt mit der Abbildung der Anforderungen auf ein Modell fort, kommt schließlich zur Prüfung der Anforderungen durch Validierung und Verifikation und wird durch das parallel ablaufenden Anforderungsmanagement unterstützt. Im Rahmen jedes Teilprozesses kommen diverse Methoden und Techniken zur Anwendung. In dieser Arbeit wurden häufig verwendete und die aus den eigenen Erfahrungen des digitalen Videosystems bewährten Methoden zur Erläuterung ausgewählt. Die in Kapitel 4 beschriebene, neue Entwicklungsmethode baut auf diesen herkömmlichen *Requirements-Engineering*-Methoden auf. Die hier vorgestellten Methoden und Ansätze werden zur besseren Einordnung nach folgenden Fragestellungen bewertet.

- Kann eine Methode oder ein Ansatz mit Variabilität umgehen bzw. ist es möglich, die in einer Systemfamilie wichtigen, variablen Anteile zu berücksichtigen?
- Ist es möglich, die Ergebnisse einer Methode oder eines Ansatzes in das Systemfamilienmodell zu integrieren oder stellen die Ergebnisse sogar einen Teil eines Systemfamilienmodells dar?
- Kann eine Methode oder ein Ansatz bei der Konfiguration eines Familienmitgliedes unterstützend wirken?
- Unterstützen eine Methode oder ein Ansatz die Parametrierung von Komponenten?
- Sind die Ergebnisse der Methode überprüfbar?

3.1.1 Erheben von Anforderungen (*Elicitation*)

Im Rahmen einer Systementwicklung muss zunächst das Problem identifiziert und verstanden werden. Diese zwei Vorgänge sind untrennbar miteinander verbunden, da sich ein Problem nur korrekt identifizieren lässt, wenn ein grundlegendes Verständnis für das Problem selbst und die umgebenden Bereiche aufgebaut worden ist. In [KoSo 1998] und [Hofm 2000] werden vier für die Anforderungserhebung relevante Bereiche dargestellt. Das Verstehen der Applikationsdomäne, des zu lösenden Problems, des Geschäftsumfeldes und der Bedürfnisse der *Stakeholder*.

Innerhalb des ersten Bereichs muss Wissen über die Anwendungsdomäne der Entwicklung erlernt werden, sofern dieses Wissen nicht schon vorhanden ist. Beispielsweise erfordert die Entwicklung eines digitalen Videosystems rudimentäre Kenntnisse der digitalen Videoübertragung, der Videodatenverarbeitung und der in diesem Bereich existierenden, nutzbaren Softwarekomponenten. Derartiges Wissen muss durch das Studium einschlägiger Literatur, die Befragung von Domänenexperten oder den Besuch entsprechender Schulungen erworben werden.

Im zweiten Bereich müssen die Entwickler neben dem allgemeinen Domänenwissen des Systems das zu lösende Problem im Kontext des betreffenden Auftraggebers verstehen. Wer wird das digitale Videosystem nutzen und wie nutzt diese Person im Moment ein eventuell bestehendes Altsystem? Oft haben die entsprechenden Personen exponierte Positionen inne, sodass ihr Terminplan nur kurze Gespräche erlaubt. Des Weiteren stehen firmenpolitische und auch persönliche Entscheidungen hinter den Antworten, die ein Entwickler im Rahmen der Anforderungsanalyse bekommt. Damit ist neben den rein technischen Fähigkeiten auch Einfühlungsvermögen und Verständnis für die Situation des Befragten notwendig. Im Falle des digitalen Videosystems hätte die Allgemeinheit der potentiellen Nutzer befragt werden müssen. Es konnten jedoch nur stichprobenartige Befragungen vorgenommen werden, um den Aufwand in den gegebenen Grenzen des prototypischen Projektes zu halten.

Der dritte Bereich muss Wissen aus dem Geschäftsumfeld zusammentragen. Es geht um die Frage, in welches Umfeld das zu entwickelnde System integriert werden muss, welche existierenden Systeme es bereits gibt, welche Schnittstellen berücksichtigt werden müssen und welches Ziel das System in diesem Umfeld erreichen muss. Für das digitale Videosystem muss das darunter liegende Betriebssystem und die Schnittstelle der Videokarte im Rechner berücksichtigt werden. Ein Ziel ist die Nachbildung der Funktionalität eines herkömmlichen Videorecorders.

Schließlich müssen die Entwickler die Bedürfnisse und Eigenheiten der *Stakeholder* verstehen. Alle an dem Projekt beteiligten Personen haben eigene Vorstellungen bezüglich des zu entwickelnden Produktes und sie haben alle einen individuellen Blickwinkel, der im Gespräch unbedingt beachtet werden muss. Darüber hinaus müssen diejenigen Personen berücksichtigt werden, die ein negatives Interesse an der Fertigstellung des Projektes haben, die dem Projekt eigentlich schaden wollen.

Die folgenden neun Abschnitte stellen derzeit verwendete Methoden und Techniken zur Erhebung von Anforderungen vor. Dokumentanalyse [Hofm 2000], Protokollanalysen [Eric 1984], Brainstorming [Clar1989], Interviews [KoSo 1998], Workshops [Hofm 2000], ethnographische Beobachtungen [Lang 2003], Kategorien [Stew 1997] und Fragenkataloge [Milt 2002] werden jeweils kurz erläutert. Aufgrund der Recherchen zu dieser Arbeit und eigener Erfahrungen in studentischen Projekten stellen die gewählten Ansätze einen guten Querschnitt dar und sind daher für die Ziele dieser Arbeit ausreichend.

Dokumentanalyse

Im Projektumfeld existierende Dokumente werden analysiert, um erste Anforderungen abzuleiten. Von Nutzen sind Dokumente wie Marktanalysen, strategische Pläne, Bücher, relevante Patente, gesetzliche Bestimmungen, Standards / Normen, Handbücher bereits existierender Systeme und auch Literatur zur Softwareentwicklung selbst. Diese Dokumente enthalten Informationen über die Problemdomäne. Entwickler müssen sich mit Hilfe dieser Informationen in die Domäne einlesen, um bei Kundengesprächen die Sprache des Kunden und dessen Probleme zu verstehen. Des Weiteren kann durch die strategischen Pläne, die aus den Marktanalysen abgeleitet werden, Wissen um den zu erzielenden Nutzen des Systems aufgebaut werden. Standards und gesetzliche Rahmenbedingungen liefern erste einschränkende Anforderungen an das System, die von Kundenanforderungen nicht verletzt werden dürfen. Eventuelle Zusatzkosten an Dritte können durch eine Patentrecherche frühzeitig festgestellt und in die Gesamtkalkulation aufgenommen werden. Wie in [Hofm 2000] zusammengefasst, gibt es bereits Werkzeuge zur automatisierten Analyse von Textdokumenten, die bis zu Werkzeugen zur automatisierten Zusammenfassung von Texten reichen.

Die Dokumentanalyse kann sich im Systemfamilienumfeld in einen domänenspezifischen und mehrere variable Anteile aufteilen. Die domänenspezifischen Anteile stellen Dokumente dar, die innerhalb der Domäne allgemein gültig sind. Dazu zählen Patente, Gesetze, Standards und Normen.

Die daraus abgeleiteten Anforderungen werden später zum Kern der Systemfamilie gehören. Handbücher bereits existierender Systeme beschreiben jeweils eine Variante der Familie, wobei die entstehenden Anforderungen der Variante zugeordnet werden. Damit können diese Ergebnisse direkt im Familienmodell der Systemfamilie genutzt werden. Die spätere Modellierung von Varianten wird vereinfacht, da durch die Dokumentanalyse bereits ein Teil der Varianten und die dazugehörigen Anforderungen erarbeitet werden können. Die Analyse der Dokumentation existierender bzw. einzubindender Komponenten stellt hierbei eine wichtige Vorarbeit für die Konfiguration und Parametrierung dar. Zum einen werden eine Reihe von Komponenten in das Familienmodell aufgenommen, die später für einzelne Familienmitglieder gewählt werden können. Zum anderen werden die Parameter für die jeweiligen Komponenten erhoben und ebenfalls im Familienmodell hinterlegt. Die Dokumentanalyse kann diese Aufgaben nur erfüllen, wenn ein Familienmodell vorhanden und entsprechend definiert ist. Eine automatisierte Überprüfung der Ergebnisse aus der Dokumentanalyse ist jedoch nicht möglich. Nur ein erneutes Lesen der Dokumente kann Fehler aufdecken.

Protokollanalysen

Geht es um Systeme zur Automatisierung von Geschäftsabläufen innerhalb einer Firma, können die einzelnen Arbeitsvorgänge analysiert werden, indem die beteiligten Personen gebeten werden, die von ihnen durchgeführten Arbeitsvorgänge während der Bearbeitung zu beschreiben. Auf Basis der bei den Befragungen erstellten Protokolle werden Anforderungen erhoben, um die Arbeitsprozesse durch ein Computersystem abbilden zu können. Allerdings ist es selbst für erfahrene Personen schwierig, während der Arbeit ihre eigenen Arbeitsvorgänge zu beschreiben.

Für eine systemfamiliengerechte Aufarbeitung der protokollierten Arbeitsschritte muss die Protokoll-Analyse durch einen Vergleichsschritt erweitert werden. Hierbei werden ähnliche oder identische Schritte identifiziert. Die Arbeitsschritte müssen innerhalb eines Baumes derart angeordnet werden, dass die Wege von der Wurzel zu den Blättern jeweils ganze Arbeitsabläufe innerhalb eines Unternehmens ergeben. Pro Unternehmen wird ein Baum erstellt, wobei auch hier Ähnlichkeiten ausgenutzt werden, um bereits beschriebene Arbeitsschritte wiederzuverwenden. Diese Bäume aus Prozessschritten erweitern damit das Familienmodell. Es entstehen sowohl wiederverwendbare Einzelschritte als auch zusammengesetzte Arbeitsabläufe, wobei die in jeder Variante vorhandenen Arbeitsschritte und Abläufe dem Kern der Familie zugeordnet werden. Die restlichen Arbeitsschritte und Abläufe stellen die optionalen Anteile dar. Für Systeme, die zur Unterstützung betrieblicher Abläufe dienen, stellt die Protokoll-Analyse die einzelnen für ein System konfigurierbaren Teilschritte zur Verfügung. Werden zu den Prozessschritten auch die jeweils erforderlichen Ein- und Ausgabedaten als Parameter erhoben, kann auch die Parametrierung der Prozessschritte realisiert werden. Die automatisierte Überprüfung der Ergebnisse einer Protokoll-Analyse ist nicht möglich und selbst die

manuelle Prüfung ist sehr aufwändig, da sie die nochmalige Konsultation der beteiligten *Stakeholder* erfordert.

Brainstorming

Einzelne Probleme werden in einer Sitzung von einem Moderator kurz erläutert. Die Teilnehmer äußern nun ihre Gedanken und Ideen zu dem vorgestellten Thema. Diese werden solange notiert, bis es keine weiteren neuen Ideen gibt. Im Zuge einer *Brainstorming*-Sitzung ist keine Kritik an den Ideen anderer Teilnehmer erlaubt, es ist wichtig viele Ideen zu sammeln, bereits notierte Ideen dürfen und sollen weiterentwickelt werden und es ist erwünscht, dass auch auf den ersten Blick völlig abstruse oder weit hergeholte Ideen geäußert werden.

Im Umfeld einer Systemfamilienentwicklung müssen zusätzliche Sitzungen einberufen werden, um über zukünftige Variationsmöglichkeiten zu diskutieren und entsprechende Ideen zu sammeln. Dabei geht es sowohl um komplett neue Varianten einer Familie, die unter Umständen die Grenzen verschieben, innerhalb derer sich die Systemfamilie bis dahin bewegt hatte, als auch um neue Abwandlungen von Systemen, die schon Teil der Familie sind. Dadurch kann das Familienmodell erweitert werden, wobei jedoch die Einarbeitung der Daten in das Modell im Rahmen einer *Brainstorming*-Sitzung nur durch grobe Ideen und Ratschläge unterstützt werden kann. Die Konfiguration eines Familienmitgliedes und dessen Parametrierung sind gar nicht betroffen, da hierzu tiefergehende Betrachtungen eventueller Komponenten notwendig sind. Die Ergebnisse von *Brainstorming*-Sitzungen sind per Definition vage und können daher nicht automatisiert überprüft werden.

Interviews / Fragenkataloge

Anforderungen der Personen, die am Beginn der Entwicklung für das Projekt als relevant festgestellt wurden, werden im Rahmen von strukturierten und unstrukturierten Befragungen erhoben. In jedem Fall geht es um ein Frage-Antwort-Spiel, in dessen Verlauf der Entwickler versucht, Anforderungen zu erheben und diese zu verstehen.

Die unstrukturierte Befragung kommt dabei eher einer Unterhaltung nahe, dauert lange und führt zu einer Flut von Informationen, die im Nachgang gefiltert werden müssen. Trotz dieser Nachteile kann ein Entwickler, der sich sehr gut in der Domäne auskennt, auch im Rahmen einer unstrukturierten Befragung sinnvolle Anforderungen erheben.

Bei der strukturierten Variante wird mehr Zeit in die Vorbereitung der Befragung investiert. Die Themen werden vorab definiert, sodass jede Person über exakt die gleichen Themen befragt wird. Die Vergleichbarkeit der Antworten ist daher besser, obwohl die vorbereiteten Fragestellungen entsprechende Antworten mitunter schon erwarten lassen.

Die dritte Möglichkeit ist damit der Fragenkatalog mit teilweise vorgegebenen Antworten. Hierbei können sehr viele Personen in kurzer Zeit befragt werden, wobei der Vorbereitungsaufwand nochmals erhöht ist und sich Fragebogen hauptsächlich für einfache Fragen und Antworten mit bezifferbaren Größen eignen. Fragenkataloge, als spezielle, strukturierte Form von *Interviews*, helfen bei der Erhebung von Anforderungen wichtige Fragen nicht zu vergessen. Eine spezialisierte Form von *Interviews* sind Fokusgruppen (*Focus Groups*). In Sitzungen, die mehrere Stunden andauern können, werden ganz spezifische, einzelne Problemstellungen erörtert. Die Teilnehmer einer Fokusgruppe sind Spezialisten für und Interessierte an genau dem Thema, welches erörtert werden soll. Wie jeder Moderator, muss auch der Moderator der Fokusgruppe in der Lage sein, die Diskussion zu leiten und zu führen, um effizient zu einem guten Ergebnis zu gelangen.

Interviews werden im Systemfamilienumfeld durch Fragen zu Varianten und Variationsmöglichkeiten ergänzt. Zunächst muss jeweils festgehalten werden, welcher zukünftigen Variante eine Befragung eventuell zugeordnet ist. Darüber hinaus sollten auch Fragen zu potentiellen Variationsmöglichkeiten einer diskutierten Anforderung gestellt werden, die gleichzeitig priorisiert werden müssen, um festzuhalten, wie wertvoll das Vorhandensein einer optionalen Anforderung für einen Kunden sein würde. Schließlich werden auch Variationsmöglichkeiten diskutiert, die nicht dem Anforderungskatalog eines Kunden entstammen, jedoch dazu dienen, wiederum den Wert einer Variation abzuschätzen. Je mehr Kunden zusätzlich in einem System realisierte Anforderungen kaufen würden, desto höher ist deren Priorität im Entwicklungsprozess. Damit lassen sich neben den Anforderungen, die den Kern der Familie ausmachen, sehr frühzeitig diejenigen optionalen Anteile der Systemfamilie herausarbeiten, die zu Beginn der Entwicklung realisiert werden müssen. Zusammen mit der notwendigen Zuordnung der Anforderungen zu den avisierten Varianten der Familie, ist eine zielgerichtete Planung der Entwicklungsaktivitäten möglich. Das Familienmodell kann mit eventuell neuen Varianten angereichert werden. Es sind damit neue Konfigurationen möglich, wobei die Entwickler die Komponenten, die für eine bestimmte Konfiguration notwendig sind, an dieser Stelle noch nicht kennen, ebenso wenig wie die eventuell notwendigen Parameter. Automatisierte Überprüfungen der Ergebnisse von Interviews sind nicht möglich, lediglich die Auswertung kann automatisiert werden.

Workshops

Workshops sind Treffen der *Stakeholder*, die entweder strukturiert, mit einem vorher klar definierten Ziel abgehalten werden oder sich an die Kreativität und Vorstellungskraft der Beteiligten richten. Der erste Fall wird als *Joint Application Design* (JAD) bezeichnet und wurde von IBM in den späten Siebzigern beschrieben. Typischerweise dauern die Treffen ein bis zehn Tage, wobei den Teilnehmern eine Reihe von Formularen zur Verfügung gestellt wird, um die jeweiligen Ergebnisse festhalten zu können. Nach [Wood 1995], [Geie 2000] und [Jenn 2002] sind JAD-Treffen klar auf ein The-

ma fokussiert. Die Treffen beinhalten neben der konkreten Vorbereitung der Treffen selbst, auch die Koordination der oft weit verteilten Beteiligten. Dabei muss der Koordinator dafür sorgen, dass nur Personen an dem Treffen teilnehmen, die auch ein positives Interesse an dem zu entwickelnden System haben. Die Teilnehmer dieser zielorientierten und disziplinierten Treffen werden in sechs Gruppen eingeteilt, die jeweils eine bestimmte Rolle innehaben. Der Moderator führt die zwei bis drei Entwickler und drei bis fünf Endnutzer durch die Diskussion. Im Falle einer Pattsituation muss ein Schlichter eine Entscheidung fällen, der meist aus den Reihen der Manager benannt wird. Experten der Domäne nehmen bei Bedarf beratend an den Treffen teil, während zwei bis drei Beobachter nur passiv teilnehmen.

Demgegenüber stehen die *Future Workshops*, die ein gegebenes Problem kreativ erforschen sollen [Hofm 2000]. Alle Beteiligten werden durch das Problem beeinflusst und versuchen nun in drei Phasen des *Workshops* zu einer möglichen Lösung zu gelangen. Während der Kritikphase sollen möglichst viele Teilprobleme identifiziert und für die spätere Diskussion geordnet werden. Die Teilprobleme werden gruppiert, um in der Folge von kleineren Gruppen analysiert zu werden. Die anschließende Fantasiephase gibt den Teilnehmern die Möglichkeit, Ideen und Visionen zur Problemlösung zu sammeln, ohne dabei auf Nachteile oder Beschränkungen der Ideen Rücksicht nehmen zu müssen. In der abschließenden Implementierungsphase werden die notwendigen Ressourcen für die Realisierung der Ideen abgeschätzt. Betrachtungen zur Machbarkeit schließen die letzte Phase eines *Future Workshops* ab.

Zur schnellen und effizienten Einarbeitung in die Entwicklung von Systemfamilien sind zusätzliche *Workshops* zum Thema Systemfamilien sinnvoll. Die Wissensstände der einzelnen Mitglieder einer Entwicklergruppe werden angeglichen und die Entwickler selbst haben die Möglichkeit, durch eigene Vorträge Wissen zu vertiefen und zu festigen. *Workshops* eignen sich gut zur Prüfung der Inhalte des Systemfamilienmodells. Die beteiligten Entwickler können durch vorbereitete Diskussionen Unklarheiten, Missverständnisse oder logische Fehler im Modell aufdecken. Zur Diskussion einzelner Konfigurationen und entsprechender Parametrierungen der Komponenten bieten sich eigene *Workshops* oder kleinere Gruppen, Fokusgruppen, innerhalb eines *Workshops* an. Hier können die Entwickler die Details jeweils einer Konfiguration klären.

Ethnographische Beobachtung

Untersuchungen der Personen, die das System später nutzen sollen, führen zu deren Eigenheiten und kulturellen Besonderheiten. Zum einen können ganz spezielle, personenbezogene Vorgehensweisen bei Arbeitsschritten zu neuen oder veränderten Anforderungen führen. Zum anderen haben unterschiedliche Kulturkreise einen Einfluss auf das Domänenwissen, welches die Voraussetzung für das Verständnis von Anforderungen ist. Beispielsweise ist das Bildungsniveau der Nutzer des digitalen Videosystems wichtig für das Bedienkonzept. Es müssen also unterschiedliche Konzepte in

das System integriert werden für Personen mit keinem bis hin zu Personen mit viel Wissen aus der Domäne. Daraus ergeben sich direkt Variationsmöglichkeiten für die Systemfamilie, die sich in den optionalen Anteilen der späteren Produkte niederschlagen. Die unterschiedlichen Bedienkonzepte werden als Optionen in die Familie integriert und können später von Kunden ausgewählt werden. Darüber hinaus müssen die bereits etablierten, länderspezifischen Einstellmöglichkeiten für Sprache und Alphabet ebenfalls als Variationsmöglichkeiten in der Systemfamilie hinterlegt werden. Die Überprüfung geschieht durch Lektorieren der Texte in den unterschiedliche Sprachversionen und Akzeptanztests mit Vertretern der unterschiedlichen ethnographischen Zielgruppen.

Kategorisieren

Ein Schritt in Richtung eines teil-formalen oder formalen Modells ist die Einteilung der Anforderungen in Kategorien. Damit werden zusammengehörige Anforderungen hierarchisch in einem Baum angeordnet. Nicht so enge Beziehungen können über Referenzen der beteiligten Anforderungen zueinander realisiert werden.

Zur Unterstützung der Einteilung kann ein so genanntes *Repertory Grid* erstellt werden, beschrieben im Artikel von [Stew 1997]. Es handelt sich dabei um eine Tabelle, die bewertete Aussagen über gewählte Themen aus dem Systemumfeld enthält. Als Beispiel wird die Bedienung des digitalen Videosystems betrachtet. Wie in Tabelle 3.1 zu sehen, werden vier Möglichkeiten der Bedienung des Gerätes bewertet, die in der letzten Zeile notiert sind. Diese Möglichkeiten sind nach drei Kriterien zu bewerten, für die Werte zwischen 1 und 4 zu vergeben sind. Der Wertebereich ist willkürlich gewählt, wobei sich Bereiche über 7 als zu genau herausgestellt haben. Nun wird die Tabelle für jedes der möglichen Geräte mit den entsprechenden Zahlenwerten gefüllt. Die Bedienung des digitalen Videosystems mittels eines *Handheld*-Gerätes ist in der Anschaffung die teuerste Variante, einfach zu bedienen, jedoch komplexer als eine herkömmliche Fernbedienung. Der *Handheld* wird hauptsächlich zur direkten Bedienung des Systems genutzt, kann aber, je nach Ausstattung des *Handhelds*, auch aus dem Büro zur Programmierung der Aufnahme einer Sendung genutzt werden. Mit Hilfe dieser Tabellen können Begriffe und Komponenten des Systems nach eigenen Maßstäben bewertet werden. Die zugehörigen Anforderungen sind mit der Bewertung verknüpft. Die Analyse des Systems durch veränderte oder neue Anforderungen wird durch den Vergleich mit den bereits tabellarisch erfassten Anforderungen erleichtert.

Befragungen können auch mit einer definierten, kleineren Menge von Fragen durchgeführt werden. Die Fragen zielen auf die Verfeinerung, Abstraktion und Ähnlichkeit von erhobenen oder noch zu erhebenden Anforderungen ab. Damit wird automatisch eine Hierarchie aufgebaut, in die sich die Anforderungen einordnen. Die Befragung beginnt, indem der Moderator einen interessanten und zu diskutierenden Punkt herausgreift, der in der Tabelle als Konstrukt bezeichnet wird. Die Fragen aus Tabelle 3.2 müssen für alle Diskussionspunkte beantwortet werden, sodass sich am Ende der Befra-

gung die Einteilung der Anforderungen in Kategorien und damit die hierarchische Einordnung ergibt.

von 1 ...					... bis 4
direkt	1	2	3	4	weit entfernt
einfach	1	2	4	2	komplex
teuer	4	1	2	3	billig
	IR-Fernbedienung	Palm-Handheld	Mobiltelefon	Internet	

Tabelle 3.1: *Repertory Grid* der Bedienung

Derzeit ist das *Repertory Grid* für die Organisation von Anforderungen für Einzelsysteme gut geeignet. Für die Systemfamilienentwicklung muss es in einer abgewandelten Form genutzt werden, die eine Kategorisierung der Anforderungen in Kern und variable Anteile erlaubt, sowie die Aufteilung der variablen Anteile auf die entsprechenden Varianten zulässt. Das Familienmodell wird durch weitere Varianten angereichert, die allerdings zunächst ohne den Bezug zu späteren Komponenten und deren Parametrierung stehen. Überprüft werden kann ein *Repertory Grid* nur durch erneute Befragung der *Stakeholder*.

Frage	Ist ein ...	Hat das Ziel ...	Ist Teil von ...
Verallgemeinerung	Von welchem Typ ist <Konstrukt> ?	Was ist das Ziel von <Konstrukt>?	Von was ist <Konstrukt> ein Teil?
Spezialisierung (Hierarchie)	Gibt es speziellere Typen von <Konstrukt>?	Gibt es speziellere Ziele von <Konstrukt>?	Gibt es speziellere Teile von <Konstrukt>?
Spezialisierung (Attribute)	Gibt es Attribute von <Konstrukt>?	Gibt es Attribute von <Konstrukt>?	Gibt es Attribute von <Konstrukt>?
Erweiterung	Gibt es andere Beispiele für <Konstrukt>?	Gibt es andere Beispiele für <Konstrukt>?	Gibt es andere Beispiele für <Konstrukt>?

Tabelle 3.2: *Laddering* Fragen nach [Hofm 2000]

Szenarien und Ziele

Die weite Verbreitung der objektorientierten Softwareentwicklung hat die Benutzungsfälle, auch als *Use-cases* bezeichnet, zu einem wichtigen Mittel der Anforderungserhebung werden lassen [Jark 1999]. Kern eines Benutzungsfalles ist die Beschreibung des Szenarios. Hierbei werden aus Nutzersicht die notwendigen Schritte zur Benutzung des System in chronologischer Reihenfolge festgehalten. Die jeweiligen Abfolgen der Handlungen sind als Szenario im *Use-case* enthalten. *Use-cases* sind das erste Mittel der objektorientierten Anforderungserhebung, wobei auch die nicht technisch versierten Benutzer nach [Kula 2000] diese Diagramme verstehen und mit ihnen arbeiten können. Der

Benutzer eines System interessiert sich nur für die Interaktion mit dem System und nicht für dessen internen Aufbau. *Use-cases* beschreiben diese Interaktionen auf einem hohen und für den Benutzer tauglichen Abstraktionsniveau.

In Abbildung 3.6 ist eine Auswahl der *Use-cases* des digitalen Videosystems dargestellt. Das Piktogramm einer Person auf der linken Seite des Diagramms steht stellvertretend für einen Akteur. Meistens handelt es sich dabei um eine Person, es können jedoch auch Systeme als Akteure agieren, wenn beispielsweise eine Suchanfrage des Nutzers von seinem Rechner umgesetzt und an ein Datenbanksystem geschickt wird. In diesem Fall ist der Rechner aus Sicht der Datenbank ein Akteur. Des Weiteren sieht man im Diagramm, dass die beiden *Use-cases* „Aufnehmen“ und „Pause“ eine Beziehung zu einem weiteren *Use-case* „Akt. Kanal aufnehmen“ haben, wodurch *Use-cases* untergliedert werden können.

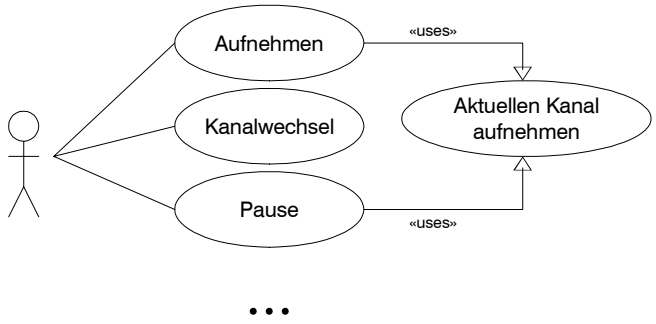


Abbildung 3.6: Use-case des digitalen Videosystems

Neben der grafischen Notation muss jeder *Use-case* durch einen angegliederten Text näher beschrieben werden. In Abbildung 3.7 ist das Schema aus [Kula 2000] für einen solchen Text dargestellt. Die Erklärungen der Felder auf der linken Seite befinden sich jeweils direkt daneben. Damit werden durch *Use-cases* Anforderungen als Menge von Interaktionen eines Nutzers mit dem System aufgenommen. Der Nutzer selbst interessiert sich dabei hauptsächlich für die Benutzungsfälle, während der Systementwickler sein Hauptaugenmerk auf die angegliederten Szenarien legt.

Ein Problem ist der oft fehlende oder nur unzulänglich enthaltene Bezug der *Use-cases* untereinander. Im Rahmen des *Cooperative-Requirements-Engineering-with-Scenarios*-Projektes (CREWS) wurde die herkömmliche Sichtweise auf *Use-cases* durch Ziele und eine verbesserte Entwicklungsmethode erweitert [Raly 1999]. Ziele stellen dabei aus Sicht der *Stakeholder* „einen zukünftigen Systemzustand oder ein Verhalten, dass es zu verhindern, unterstützen, erreichen, einzustellen, etc. gilt“, dar [CREW 1999]. Der erweiterte Entwicklungsprozess ist in Abbildung 3.8 zusammen mit den im Projekt entstandenen Strategien dargestellt.

Use-case Name	Wie heißt der Use-case?
Akteure	Wer / was ist an dem Use-case beteiligt?
Kurze Beschreibung	Was soll dieser Use-case leisten?
Notwendige Aktionen (Szenario)	Welche Schritte müssen im Rahmen dieses Use-case ausgeführt werden? 1. 2. 3. ...
Alternativen	Welche Alternativen, aber seltenen Szenarien oder Abwandlungen des obigen Szenarios sind möglich?
Ausnahmen	Welche Fehler können auftreten und was ist dann zu tun?
Erweiterungspunkte	An welcher Stelle in diesem Use-case werden andere Use-cases über die «extends» Beziehung referenziert? Jeweils mit kurzer Erläuterung.
Vorbedingung	Was muss gelten, damit dieser Use-case ausgeführt werden kann?
Nachbedingung	Was wird dem System zugesichert, wenn dieser Use-case ausgeführt wurde?
Beziehungen zu *Business Rules*	Welche firmeninternen Regelungen und Vorgänge beeinflussen diesen Use-Case oder werden von ihm beeinflusst?
Autor	Wer hat diesen Use-case geschrieben?
Datum	Wann wurde der Use-case erstellt?

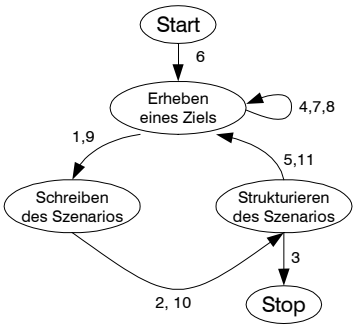
Abbildung 3.7: Use-case Schema

Zunächst werden die Ziele der *Stakeholder* aus den Anforderungen abgeleitet. Weitere Ziele können erhoben und existierende verfeinert oder erweitert werden, was durch die von CREWS vorgeschlagenen Strategien unterstützt wird, beispielsweise durch ein entsprechendes Formular. Aus den Zielen werden die Szenarien erarbeitet, indem der herkömmliche Text nach dem bereits dargestellten *Use-case* Schema geschrieben wird oder wiederum in ein von CREWS vorgeschlagenes Formular eingetragen werden. Schließlich müssen die Szenarien strukturiert werden, was alle Szenarien in eine definierte und automatisiert verarbeitbare Form überführt. Mit Hilfe von linguistischen und die Einzelschritte eines Szenarios betreffenden Regeln wird die Vollständigkeit der erhobenen Szenarien überprüft.

Die CREWS Methode stellt eine Erweiterung des *Object-Oriented Software Engineering* (OOSE) Ansatzes dar [Raly 1999]. Sie wurde im Rahmen einer Studie bewertet [Tawb 1999], wobei die teilnehmenden Entwickler durch die strukturierte Vorgehensweise Fehler vermeiden und gegenüber einer Vergleichsgruppe mehr Sonderfälle für das Ablaufverhalten des Systems berücksichtigen konnten.

Wie bereits im Abschnitt über die Protokollanalysen erläutert, müssen auch die Szenarien und die referenzierten *Use-cases* den Variationsmöglichkeiten einer Systemfamilie gerecht werden. Hier gibt es bereits einen in [John 2002] beschrieben Ansatz, der Variationen in *Use-cases* durch Stereotypen der UML modelliert, um Variabilität für Systemfamilien zu realisieren. Der im weiteren Verlauf dieser Arbeit beschriebene Lösungsansatz nutzt diese *Use-cases* als ein Beschreibungsmittel für Anforderungen und erweitert damit das Familienmodell. Die Konfiguration und Parametrierung von Kom-

ponenten muss durch eine Referenz von einem *Use-case* zu den entsprechenden Komponenten realisiert werden. Die Überprüfung der Szenarien kann nur manuell geschehen.



CREWS Strategien

1 - Schreibe ein Szenario als Prosatext.
2 - Strukturiere ein Szenario unterstützt durch ein Werkzeug.
3 - Nutze die Vervollständigungsregeln für ein Szenario.
4 - Erhebe ein alternatives Ziel durch strukturgetriebene Strategie.
5 - Erhebe ein Ziel mit Hilfe einer alternativen Suchstrategie.
6 - Erhebe ein Ziel mit einer Identifikationsstrategie.
7 - Erhebe ein Ziel mit einer formularbasierten Strategie.
8 - Erhebe ein Ziel mit Hilfe der linguistischen Strategie.
9 - Schreibe ein Szenario mit Hilfe der formularbasierten Strategie.
10 - Strukturiere ein Szenario manuell.
11 - Erhebe ein Ziel mit Hilfe der Kompositions-Suchstrategie.

Abbildung 3.8: CREWS Methode

Zusammenfassend für die Erhebung von Anforderungen in der *Requirements-Engineering*-Phase der Softwareentwicklung sind die Methoden und deren Einordnung in die Systemfamilienentwicklung genannt. Die Einordnung beschreibt, wie die Ergebnisse der jeweiligen Methode im Systemfamilienumfeld genutzt werden können. Andernfalls können die Ergebnisse nicht direkt genutzt werden und sind in der Tabelle Anforderungen enthalten, die eine Abwandlung der jeweiligen Methode beschreiben.

Methode	Einordnung und Anpassungen für die Systemfamilienentwicklung
Dokumentanalyse	• Dokumente wie Patente, Gesetze, Standards und Normen sind in der Domäne allgemein gültig und werden später zum Kern der Systemfamilie gehören. • Varianten werden durch Handbücher bereits existierender Systeme beschrieben.
Protokollanalyse	• Die protokollierten Arbeitsschritte werden in jeweils einem Baum pro Systemvariante organisiert. • In jedem Baum vorkommende Arbeitsschritte gehören zum Kern. • Einzelschritte wie auch immer wieder vorkommende, aus Arbeitsschritten zusammengesetzte Arbeitsabläufe sind wiederverwendbar und stellen optionale Anteile der Systemfamilie dar.
Brainstorming	• Es sind zusätzliche Sitzungen zu Variationsmöglichkeiten für neue oder abgewandelte Varianten notwendig.
Interviews / Fragenkataloge	• Eine Befragung muss jeweils einer Variante zugeordnet werden. • Kunden müssen auch nach Variationsmöglichkeiten befragt werden. • Der Wert bzw. die Priorität einer Variationsmöglichkeit muss ebenfalls durch Interviews festgestellt werden und beeinflusst den Entwicklungsprozess.
Workshops	• Zusätzliche Workshops zum Thema Systemfamilienentwicklung sollten abgehalten werden, um eine schnelle und effiziente Einarbeitung der Entwickler zu gewährleisten.
Ethnographie	• Die länderspezifischen Varianten zur Bedienung der Systeme, die jeweilige Sprache und unterschiedliche Alphabete stellen Variationsmöglichkeiten im Sinne der Systemfamilienentwicklung dar.
Kategorisieren	• Ein *Repertory Grid* kann für die Kategorisierung der Anforderungen durch einen Kern und variable Anteile in einer angepassten Form genutzt werden.

Methode	Einordnung und Anpassungen für die Systemfamilienentwicklung
Szenarien	• Anpassungen an die Entwicklung von Systemfamilien sind in [John 2002] beschrieben. • Szenarien sind Teil des Familienmodells, der Systemfamilienentwicklung.

Tabelle 3.3: Erhebungsmethoden und deren Einordnung in die Systemfamilienentwicklung

3.1.2 Modellierung von Anforderungen (*Modeling*)

Nachdem Anforderungen in Form von Texten, Skizzen und sprachlichen Aufzeichnungen vorliegen, werden diese Daten in Modellen abgelegt. Die Modelle formalisieren einen Teil der Anforderungen. Neben den funktionalen, den datenorientierten und den dynamischen Modellen, werden in diesem Kapitel objektorientierte und zielorientierte Ansätze vorgestellt, welche die derzeit wichtigsten Vertreter für die Modellierung von Anforderungen darstellen und für diese Arbeit relevant sind.

Funktionale Modelle

Zu den funktionalen Modellen gehören De Marcos Datenflussdiagramme, beschrieben in [Balz 1996]. Die Wege der Daten sowie deren Transformation in Zusammenhang mit den notwendigen Funktionen werden modelliert. Die Diagramme enthalten Schnittstellen zu der umgebenden Umwelt, Speicher zur Aufnahme und Wiedergabe beliebiger Informationen, Funktionen zur Ver- und Bearbeitung der Daten und die Daten selbst. In Abbildung 3.9 ist ein Datenflussdiagramm für das digitale Videosystem zu sehen. Der Fernsehschirm, die Satellitenschüssel und der Benutzer, über den Umweg der Fernbedienung, stellen die Schnittstellen des Systems zur Umwelt dar. Der Benutzer kann das System über die TV-Steuerung für einen Fernsehabend nutzen oder aber über die Rekordersteuerung die erweiterten digitalen Videofunktionen in Anspruch nehmen. Mit der Videoverarbeitung kommt der Benutzer nicht direkt in Kontakt, sondern nutzt diese Funktionalität über den Umweg der Steuerungen im System.

Die strukturierte Analyse bietet im Bereich der funktionalen Modellierung eine Erweiterung der Datenflussdiagramme. Ein gegebenes Problem wird hierarchisch in immer kleinere Teile zerlegt. Diese Zerlegung führt zum einen zu handhabbaren Teilproblemen, zum anderen aber werden diese Probleme auf der untersten Ebene schon sehr formal beschrieben. Diverse CASE-Werkzeuge, wie beispielsweise *Statemate* [Ilog 2003], setzen die strukturierte Analyse so weit um, dass eine komplette Codegenerierung möglich ist. Auf der obersten Ebene der strukturierten Analyse steht das Kontextdiagramm, welches das System als einen einzigen Prozess mit den benötigten Eingabe- und Ausgabedaten beschreibt. Auf dieser Ebene gibt es noch keine Datenflüsse und auch die Speicher der Datenflussdiagramme sind noch nicht enthalten. Erst in der nächsten Ebene der Zergliederung des Systems werden die Datenflussdiagramme zur Verfeinerung des ersten und im folgenden der neu entstandenen Prozesse benutzt. Im Rahmen der strukturierten Analyse wird ein so genanntes *Data Dic-*

tionary aufgebaut, welches die Zusammenhänge der einzelnen Datenflüsse enthält. Die Datenflüsse werden durch die hierarchische Dekomposition ebenfalls zerlegt. Die Zusammengehörigkeit und Zusammensetzung eines bestimmten Datenflusses aus mehreren anderen Datenflüssen wird im *Data Dictionary* beschrieben. Schließlich werden die Prozesse auf der untersten Ebene durch Mini-Spezifikationen beschrieben, die das Vorgehen zur Umsetzung der Eingabedaten in die Ausgabedaten enthalten. Die Mini-Spezifikationen werden in Form von Pseudocode, Entscheidungstabellen oder Entscheidungsbäumen beschrieben.

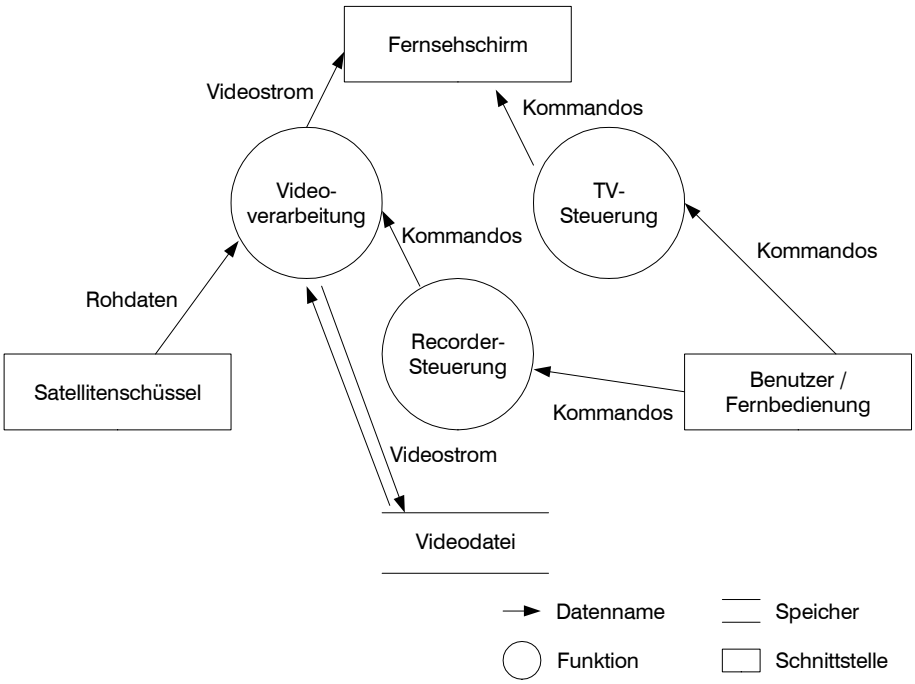


Abbildung 3.9: Datenflussdiagramm eines Teils des digitalen Videosystems

De Marco schlägt eine Methode zur Entwicklung von Datenflussdiagrammen vor, die von [Balz 1996] verfeinert und durch die folgenden 10 Punkte beschrieben wird.

1. Festlegung der Schnittstellen zur Umwelt des Systems.
2. Eingabe- und Ausgabedatenflüsse identifizieren.
3. Zu den Datenflüssen gehörende Prozesse identifizieren.
4. Notwendige Speicher identifizieren, eventuell ein *Entity-Relationship*-Modell (siehe unten) erstellen und Speicher aus diesem Modell ableiten.
5. Verfeinern der Datenflüsse mit Zuordnung zu Prozessen und Speichern.
6. Datenflüsse und Speicher im *Data Dictionary* definieren.

7. Iterative Verbesserung des Modells, bis das komplette System zufrieden stellend modelliert ist.
8. Überarbeitung der hierarchischen Ebenen des Diagramms.
9. Mini-Spezifikationen für nicht weiter zu verfeinernde Prozesse entwickeln.
10. Weitere Iterationen, falls dem Modell nicht allseitig zugestimmt wird, ansonsten beenden der Methode.

Die leicht erlernbare und von diversen CASE-Werkzeugen unterstützte strukturierte Analyse ist ein weit verbreitetes Vorgehen zur Entwicklung von Systemen. Allerdings können weder Schnittstellen noch die Speicher der Datenflussdiagramme verfeinert werden, wodurch die Weiterentwicklung des Systems behindert wird.

Für die Systemfamilienentwicklung ist die Modellierung funktionaler Anforderungen durch Datenflussdiagramme eine ausreichende Darstellung abgeschlossener Systemteile. Die Modellierung von Variationsmöglichkeiten ist derzeit allerdings nicht möglich. Es gibt Datenströme, die immer vorhanden sind, also zum Kern der Familie gehören und solche, die nur in einem Teil der Varianten, die aus der Systemfamilie abgeleitet werden, vorhanden sind. Diese Datenströme sind optional, was jedoch in Datenflussdiagrammen nicht modelliert werden kann. Bei der Konfiguration von Familienmitgliedern können die variablen Funktionen genutzt werden, wobei die entsprechenden Komponenten und deren Parametrierung noch berücksichtigt werden müssen. Eine automatisierte Überprüfung von Datenflussdiagrammen ist mit entsprechenden CASE-Werkzeugen möglich, muss aber für die Datenflussdiagramme mit Variabilität angepasst werden.

Datenmodelle

Wie schon bei den Datenflussdiagrammen angesprochen, ist neben der prozessorientierten Sichtweise auf ein System auch eine detaillierte Analyse der zu ver- und bearbeitenden Daten notwendig. Kommen einfache Systeme wie ASCII-Texteditoren mit eher simplen Datenstrukturen aus, so muss beispielsweise das digitale Videosystem sehr unterschiedliche und vielschichtige Daten verarbeiten. Zur Modellierung der Daten und deren Zusammenhänge haben sich die von Chen 1976 entwickelten *Entity-Relationship*-Modelle (ER-Modelle) etabliert. Entitäten stehen dabei für beliebige, der realen Welt entstammende Objekte, die durch Attribute näher beschrieben werden. Über Relationen können Entitäten zueinander in Beziehung stehen.

In Abbildung 3.10 sind die Daten für die elektronische Programmzeitung als *Entity-Relationship*-Modell dargestellt. In den grau unterlegten Feldern befinden sich die Entitäten, die weißen Rauten modellieren die Beziehung der Entitäten. Durch die Kardinalitäten werden die Zahlenverhältnisse der Beziehungen angezeigt. Eine Video-CD enthält mindestens eine oder aber auch mehrere Sendungen, eine beliebige Sendung kann wiederum auf mehreren Video-CDs enthalten sein. Im rech-

ten Teil der Abbildung sind die Kardinalitäten an einem Beispiel erläutert. Betrachtet wird die Beziehung zwischen DVDs und Filmen. Mit der 1:1-Beziehung wird ausgedrückt, dass eine DVD einen Film enthält und jeder Film auf einer DVD abgelegt wird. Werden Filme in einer schlechteren Qualität aufgenommen, kann das durch die 1:M-Beziehung modelliert werden. Mehrere Filme werden auf genau einer DVD abgelegt. Falls weniger Filme aufgenommen werden als auf eine DVD passen, benötigen wir die 1:MC-Beziehung. Gegenüber der 1:M-Beziehung ist der Fall berücksichtigt, dass eine DVD übrig bleiben kann, also keinen Film enthält. Die N:M Beziehung wird genutzt, wenn mehrere DVDs zur Verfügung stehen und mehrere Filme auf diesen DVDs abgelegt werden.

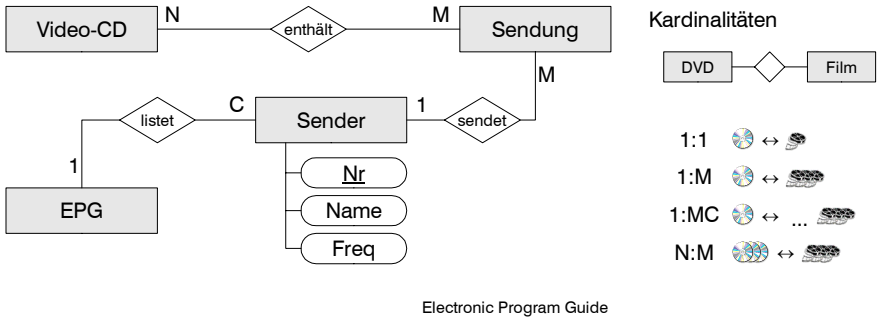


Abbildung 3.10: Entity-Relationship Modell des digitalen Videosystems

ER-Modelle werden von diversen CASE-Werkzeugen unterstützt und können von diesen direkt in die entsprechende Tabellenstruktur einer relationalen Datenbank umgesetzt werden. Für den Bereich der Datenmodellierung zusammen mit der Nutzung einer Datenbank stellen ER-Modelle das Mittel der Wahl dar und können auch in der Systemfamilienentwicklung für die Modellierung und Verarbeitung von Daten in einer Datenbank ohne Veränderung übernommen werden. Die Berücksichtigung von variablen Anteilen in einem ER-Modell ist bislang nicht möglich. Im Rahmen des digitalen Videosystems wurde das ER-Modell in jeder Systemvariante komplett übernommen. Lediglich die durch Variabilitäten betroffenen Tabellenspalten wurden beim Wegfall eines variablen Anteils nicht genutzt, also einfach leer gelassen. Mit diesen Einschränkungen ist auch die Konfiguration und Parametrierung einer Datenbankkomponente möglich.

Objektorientierte Modellierung

Nachdem auszuführende Funktionen und die zu verarbeitenden Daten modelliert sind, kann noch das interne Verhalten des Systems spezifiziert werden. Im Bereich der Verhaltensmodellierung werden die Zustände, in denen sich ein System befinden kann, und die Ereignisse modelliert, auf die das System mit einer Zustandsänderung reagiert. Zustandsdiagramme nach Harel [Hare 1987], Petri Netze [Reis 1982] oder endliche Automaten werden genutzt, um das Verhalten zu modellieren. Die

Zustandsdiagramme sind weit verbreitet und werden auch im Rahmen der objektorientierten Modellierung genutzt. Auf dem objektorientierten Ansatz baut ein großer Teil dieser Arbeit auf.

Objektorientierung wurde in den letzten Jahren als die Methode der Softwareentwicklung schlechthin betrachtet, die alle Qualitätsanforderungen positiv beeinflussen sollte. Die Objektorientierung stellt auch für die Modellierungsphase der Systemfamilienentwicklung eine wichtige Basis dar. Im Rahmen der Entwicklung des digitalen Videosystems wurden objektorientierte Konzepte intensiv genutzt und werden daher im nächsten Absatz in einem kurzen Überblick zusammengefasst.

Nach [Meye 1997] muss Softwareentwicklung folgende, grundlegende Qualitätsanforderungen erfüllen, um erfolgreich zu sein. Software muss ...

- ... korrekt und verifizierbar gegenüber der Spezifikation sein.
- ... robust gegenüber abnormen Zuständen reagieren.
- ... erweiterbar sein, falls sich die Spezifikation ändert.
- ... effizient sein, also nur die notwendigen (bzw. so wenig wie möglich) Ressourcen nutzen.
- ... portabel sein, um auf beliebigen Plattformen eingesetzt werden zu können.
- ... für alle Kunden leicht zu benutzen sein.

Des Weiteren besteht Software aus Elementen, die ...

- ... der Wiederverwendung zugeführt werden sollen.
- ... untereinander kompatibel sein sollen.

Ferner soll die Entwicklung die projektierten Zeitvorgaben einhalten und nur die gewünschten Funktionen beinhalten. Die Objektorientierung begegnet diesen Qualitätsanforderungen durch den modularen Aufbau eines Softwaresystems mit dem zentralen Konzept der Klasse. Sehr oft gehört eine Untermenge von Funktionen eines Systems zu einer Untermenge von Daten des Systems. Beide sind logisch und semantisch verbunden und wurden früher in Softwaremodulen gekapselt. Die Klasse aus der Objektorientierung fasst Daten und Funktionen in einer so genannten Datenkapsel, auch abstrakter Datentyp genannt, zusammen, wobei Funktionen auch als Methode bezeichnet werden. Die Klassen stellen den Bauplan der späteren Objekte dar und werden durch die Instanziierung zu den konkreten Objekten. Sie werden definiert durch einen Klassennamen, durch Attribute, welche den Zustand, also die Daten und Eigenschaften der Klasse enthalten und Methoden, über die der Zustand eines Objektes verändert werden kann. Dieses Basiskonzept wird im Folgenden weiter verfeinert.

Generische Klassen werden genutzt, um Softwareelemente zu entwickeln, die mit unterschiedlichen Datentypen operieren können, wie beispielsweise eine Liste, die sowohl Zahlen als auch Textstücke verwalten kann. Diese generischen Klassen können durch die jeweils benötigten Datentypen para-

metriert werden. Die Abstraktion und Spezialisierung von generischen Klassen führt zur schrittweisen Erweiterung von Programmierkonzepten. Aus einer Klasse zur Verwaltung einer Menge von Objekten wird eine Klasse zur Verwaltung einer Liste, also einer geordneten Menge von Objekten und schließlich kann daraus auch eine Klasse zur Verwaltung verketteter Listen entstehen. Diese spezielle Art der Modularisierung führt nach [Meye 1997] zu robusterem und wiederverwendbarem Code.

Die Wiederverwendung von Softwareelementen wird weiterhin durch das Konzept der Vererbung unterstützt. Dabei kann eine Klasse vorher als vererbbar definierte Eigenschaften einer gewählten Elternklasse erben. Eine neu zu erstellende Klasse kann auch von mehreren Elternklassen erben und damit so viele Eigenschaften wie notwendig durch Vererbung auf die neue Klasse übertragen. Neben der reinen Vererbung ist Polymorphismus ein weiteres mächtiges Konzept der Objektorientierung, um die Vorzüge der Vererbung weiter auszubauen. Hierbei können die Schnittstellen, über die auf Objekte zugegriffen wird, für unterschiedliche Objekte in gleicher Art und Weise genutzt werden. Bei der Ausführung von Methoden kommt es damit auf den Kontext an, der letztendlich darüber entscheidet, welches Objekt und damit welche Implementierung der Methode zum Einsatz kommt.

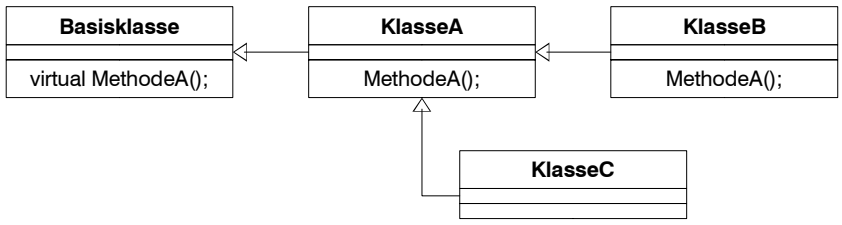


Abbildung 3.11: Polymorphismus

In Abbildung 3.11 wird das Prinzip des Polymorphismus anhand von drei Klassen grafisch erläutert. Die „Basisklasse“ stellt eine Methode zur Verfügung, die jedoch keine Implementierung aufweist. Die „KlasseA“ erbt von der „Basisklasse“ und implementiert die „MethodeA“. „KlasseB“ erbt wiederum von „KlasseA“ und muss deren Implementierung der „MethodeA“ durch eine eigene Implementierung abändern. „KlasseC“ erbt auch von „KlasseA“, verändert deren Implementierung der „MethodeA“ aber nicht. Obwohl Objekte der drei Klassen nach außen völlig identisch erscheinen, stellen sie in jedem Fall die „MethodeA“ zur Verfügung. Es wird immer anhand des Kontextes entschieden, welche Implementierung letztlich ausgeführt wird. Objekte der „KlasseA“ und „KlasseC“ führen immer die Implementierung der „KlasseA“ aus, während Objekte der „KlasseB“ die dort definierte Implementierung ausführen.

Der Begriff *Design by Contract* steht für die Entwicklung eines Systems durch Angabe von Korrektheitsbedingungen für alle auszuführenden Systemteile. In der Objektorientierung werden die Methoden durch Korrektheitsbedingungen ergänzt. Für jede Methode wird eine Vorbedingung angege-

ben, die erfüllt sein muss, bevor die Methode ausgeführt werden kann. Da nicht jede Programmiersprache die explizite Eingabe von Vor-, Nachbedingungen und Invarianten unterstützt, bietet beispielsweise C++ so genannte *Assertions* an. Hierbei wird lediglich ein Bool'scher Ausdruck ausgewertet. Ergibt die Auswertung den Wert *false*, bricht das Programm mit einem Hinweis auf die den Abbruch verursachende Quellcodezeile ab.

Diverse Notationen führten in der Vergangenheit zu einigen Verwirrungen, wenn es darum ging von fremden Entwicklern erstellte Software und deren Architektur zu verstehen. Bemühungen um einen Standard brachten schließlich die *Object Management Group* (OMG) hervor, die ursprünglich von elf Firmen gegründet wurde. Die *Unified Modeling Language* (UML) [UML 1999] ist das Ergebnis dieser Standardisierungsbemühungen und der Arbeit der OMG. Die UML stellt neun Diagramme zur Verfügung, die mehrere Sichtweisen auf ein objektorientiert zu entwickelndes System ermöglichen [Alhi 1998].

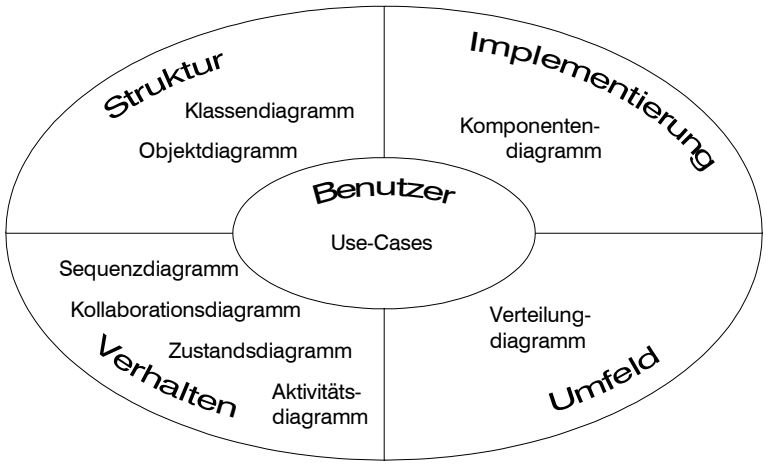


Abbildung 3.12: „4+1"-Sicht auf die UML-Diagramme

Die so genannte „4+1"-Sicht aus Abbildung 3.12 fasst die Diagramme der UML zusammen und ordnet sie den unterschiedlichen Sichtweisen zu. Zentral angeordnet ist die Benutzersicht, die das Verhalten des Systems in Form von *Use-cases*, also Benutzungsfällen, festhält und bereits im Kapitel über die Anforderungserhebung betrachtet wurde. Die UML bietet zur Beschreibung der Struktur des Systems das Klassendiagramm an. Nach dem Erheben und Entwickeln der *Use-cases*, werden Klassen mit Attributen und Methoden abgeleitet. Dabei orientieren sich die Entwickler an den Substantiven für potentielle Klassen und Attributen und den Verben der *Use-case* Beschreibungen für potentielle Methoden der Klassen. Dennoch ist die Kreativität des Entwicklers gefragt, um die für das System tatsächlich relevanten Klassen mit deren Beziehungen zu erarbeiten. Objektdiagramme werden genutzt, um den Zustand des gesamten Systems zu einem bestimmten Zeitpunkt darzustellen. Nach dem Start des Systems werden die Objekte, basierend auf den Klassenbeschreibungen, dy-

namisch erzeugt und auch wieder gelöscht. Über die Zeit ändert sich die Anzahl der Objekte und es verändern sich auch die jeweils gültigen Beziehungen zwischen den existenten Objekten. Das Objektdiagramm zeigt alle existenten Objekte und Beziehungen zu einem gegebenen Zeitpunkt auf. Die Verhaltensdiagramme modellieren die Interaktionen beteiligter Objekte für die Ausführung einer bestimmten Methode. Hier werden, in zeitlich richtiger Reihenfolge, die bei den beteiligten Objekten auftretenden Methodenaufrufe modelliert.

Das Umfeld und die Implementierung weisen jeweils nur ein Diagramm zur Modellierung auf. Das Verteilungsdiagramm enthält alle an der Gesamtlösung beteiligten Systeme und setzt sie zueinander in Beziehung. Alle Rechner, Drucker, weitere zu integrierende Systeme sowie zusätzliche Softwarekomponenten werden im Verteilungsdiagramm verzeichnet, um einen Überblick der Abhängigkeiten aller Systemteile zu erhalten. Demgegenüber bezieht sich das Komponentendiagramm auf das zu entwickelnde Softwaresystem selbst. Abhängigkeiten der Softwaremodule untereinander werden festgehalten, um die Erzeugungsprozesse der Software besser steuern zu können.

Neben der reinen Modellierung von Systemen bietet die UML auch die Möglichkeit, Randbedingungen und beschränkende Anforderungen mit Hilfe der *Object Constraint Language* (OCL) zu beschreiben. Mit dieser Sprache können Bedingungen formuliert werden, die zwischen beliebigen Modellelementen einer UML-Entwicklung gelten müssen. Die Verletzung einer in OCL formulierten Bedingung bedeutet, dass das zugrunde liegende Modell einen Fehler enthält. Mit OCL ist es möglich, jede Eigenschaft eines Modellelementes abzufragen. Im Falle der Klassendiagramme wären das beispielsweise die Attribute und Methoden einer Klasse. Jede OCL-Bedingung enthält die beteiligten Modellelemente und definiert Vor-, Nachbedingungen und Invarianten für die beteiligten Methoden.

Die Unterstützung von Varianten im Sinne der Systemfamilienentwicklung wurde über die Erweiterung der UML von [Goma 2000] realisiert. Mit Hilfe von Stereotypen werden variable Modellteile markiert, wobei sich der Entwickler bei der Ableitung eines Familienmitgliedes an diesen Stellen für eine Variante entscheiden muss. Dieser Ansatz hat zwar Schwächen bei der grafischen Darstellung, kann aber im Übrigen genutzt werden. Ein Entwickler muss die Konfiguration anhand des Entwurfs und der darin enthaltenen Variabilitäten durchführen. Die Parametrierung der Komponenten, die noch referenziert werden müssen, ist ebenfalls nur mit dem Entwurf möglich.

Goal-Oriented Requirements Engineering

Eine höhere Abstraktionsebene bieten Ziele, englisch *Goals*, die durch ein System erreicht werden sollen. Ziele können dabei, wie auch Anforderungen, funktionaler und nicht funktionaler Natur sein. Wie auch bei dem bereits dargestellten CREWS Ansatz, werden Ziele durch Eigenschaften beschrieben, die von einem zukünftigen System sichergestellt werden müssen. Anforderungen realisieren Ziele und stehen so auch mit den Zielen in Verbindung. Ein Ziel wird von einer oder mehreren

Anforderungen realisiert. Methoden wie *Knowledge Acquisition in Automated Specification of Software* (KAOS) von [Lams 2001] oder *Goal-Based Requirements Analysis* (GBRAM) von [Anto 1996] nutzen Ziele zur Modellierung im Rahmen der Anforderungsanalyse. Diese Ziele werden formal beschrieben, um durch ein Computersystem automatisch überprüft und weiterverarbeitet zu werden.

Die Ziele werden hierarchisch in einem Baum organisiert. Diese Hierarchie spiegelt die Verfeinerung und Abstraktion wieder, die es zwischen den Zielen zu modellieren gilt. Die Hierarchie ist in Abbildung 3.13 dargestellt. Bei dem Beispiel handelt es sich um die Übertragung von Videosignalen über ein Netzwerk im Rahmen des digitalen Videosystems. An oberster Position im Diagramm befindet sich das allgemeinste Ziel, die Realisierung der Videoübertragung. Dieses Ziel wird durch die tieferliegenden Ebenen weiter verfeinert. In dem Beispiel sind die beiden Ziele „Realisiere die ruckfreie Übertragung“ und „Unterstütze die sichere Übertragung“ über eine UND-Verknüpfung mit dem übergeordneten Ziel verbunden. Alle in einer UND-Verknüpfungsmenge enthaltenen Ziele müssen zwingend erreicht werden, um auch das übergeordnete Ziel zu erreichen. Neben den UND-Verknüpfungen sind auch ODER-Verknüpfungen möglich, wobei in diesem Fall eine beliebige Auswahl an Zielen zur Erreichung des übergeordneten Zieles ausreicht. Damit werden die Ziele des Modells in obligatorische und optionale Ziele eingeteilt. Obligatorische Ziele müssen in jedem Fall erfüllt werden. Optionale Ziele können, je nach Kundenwunsch, in die Realisierungsphase mit aufgenommen werden. Nach Aussagen der Autoren haben sich Ziele als Mittel der Kommunikation zwischen Entwicklern und Kunden bewährt.

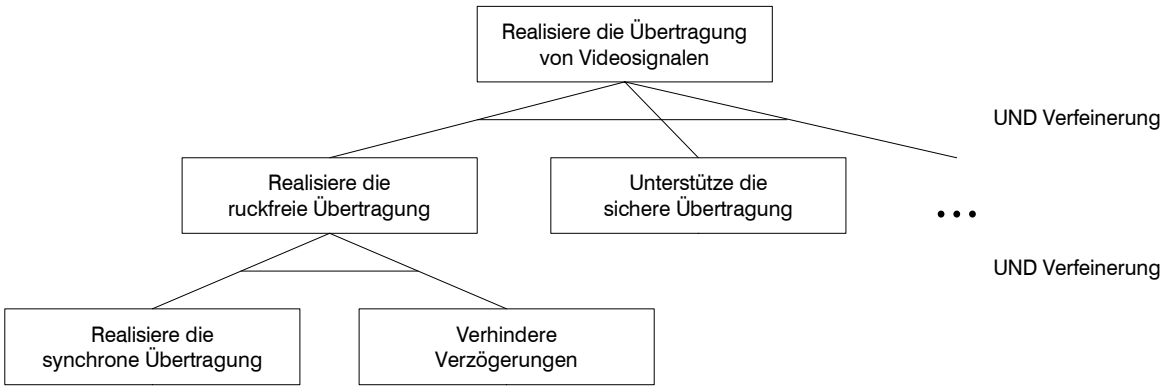


Abbildung 3.13: *Goal-Model* einer digitalen Videoübertragung

Die Modellierung von Zielen beinhaltet auch Attribute und Beziehungen zwischen Zielen, wofür die KAOS-Methode ein Metamodell definiert hat. Darüber hinaus sollen die Modelle verifizierbar sein, was zu einer formalen Spezifikation aller in einem Modell enthaltenen Elemente führt. Als Beispiel hierfür ist in Abbildung 3.14 die formale Spezifikation eines Sicherheitszieles zu sehen. Das informale Ziel, die Beachtung der FSK Beschränkung, wird formal beschrieben und kann damit au-

tomatisch durch Software überprüft werden. Der Nutzen der Modellierung mit Zielen ist hauptsächlich in der Verwendung der formalen Spezifikation zu sehen.

Goal Maintain [Altersbeschränkung]
InstanceOf SicherheitsZiel
Informal Definition Nur berechtigte Personen sollen Filme mit FSK Altersbeschränkung sehen.
FormalDef (f: AktuellerFilm, p: Personen)
• p.Alter>=f.FSK

Abbildung 3.14: Formale Spezifikation eines Zieles

Formale Spezifikationen haben den Vorteil der automatisierten Verarbeitung und den Nachteil des höheren Einarbeitungsaufwandes für die Entwickler. In einer Studie von [Lars 1996] wurde der Nutzen formaler Spezifikationen für Industrieprojekte untersucht. Aufgrund von Redundanzanforderungen musste dasselbe System von zwei Entwicklergruppen unabhängig voneinander entwickelt werden. Für die Studie entwickelte die Gruppe A das System herkömmlich, die andere Gruppe B mit Hilfe formaler Spezifikationsmethoden.

Bei der Anforderungsanalyse wurden einige Ausnahmesituationen von der Gruppe A nicht erkannt, was zu Fehlern im späteren Code führte. Der Grund hierfür liegt in den unklar formulierten und den nicht-funktionalen Anforderungen an ein System. Inkonsistenzen innerhalb der Anforderungen können mit herkömmlichen Mitteln nur schwer oder gar nicht erkannt werden. Durch die formale Spezifikation eines Systems müssen alle Randbedingungen berücksichtigt und spezifiziert werden. Allerdings kostet dieser Vorgang auch Zeit und Geld, was die Untersuchung zweifelsfrei belegt. Larsen kommt zu dem Schluss, dass formale Spezifikationen 25% mehr an Entwicklungszeit erfordern. Darüber hinaus seien formale Spezifikationen nur sinnvoll, wenn Anforderungen nicht mehr eindeutig aufgeschrieben werden können, Datenstrukturen sehr komplex sind und deren Korrektheit / Überprüfbarkeit eine unabdingbare Forderung für die Funktion des Systems ist, die korrekte Funktion direkt für Leib und Leben von Personen verantwortlich ist oder Funktionen im System vorhanden sind, die sehr umfangreich sind und viele Ausnahmefälle enthalten. Letztlich müssen die Entwickler mit formalen Methoden umgehen können oder diese Fähigkeiten erlernen, das Projekt muss genügend Ressourcen für den zusätzlichen Aufwand bereitstellen und in der Untersuchung hat sich herausgestellt, dass die Unterstützung formaler Methoden durch CASE-Werkzeuge eine wichtige Voraussetzung für die Akzeptanz und die Nutzung der Methoden ist.

Die Umsetzung und Modellierung von Anforderungen durch Ziele ist eine gute Methode, kleinere, abgegrenzte Systemteile zu beschreiben. Variabilitäten können jedoch durch die „oder“ Verknüpfung nur unzureichend modelliert werden. Müssen beispielsweise mindestens zwei Ziele aus einer Menge gewählt werden, reichen die angebotenen Möglichkeiten der Modellierung nicht aus und die Methode stößt an ihre Grenzen. Eine Verbindung zu den Komponenten der Familie ist nicht vor-

handen, wie auch die entsprechenden Information zur Parametrierung fehlen. Diese müssten in das Modell integriert werden. Die automatisierte Überprüfung des Modells ist allerdings durch die formal spezifizierten Modellelemente möglich, muss aber für die um Variabilität erweiterten *Goal-Oriented*-Modelle angepasst werden.

In Tabelle 3.3 ist ein Überblick der in diesem Kapitel vorgestellten Ansätze enthalten, wobei für jeden Ansatz eine kurze Erläuterung und eventuelle Anpassungen für die Systemfamilienentwicklung angegeben sind.

Ansatz	Einordnung und Anpassungen für die Systemfamilienentwicklung
Funktionale Modelle	• Funktionale Anforderungen abgeschlossener Systemteile können durch Datenflussdiagramme modelliert werden. • Variationsmöglichkeiten werden derzeit nicht unterstützt. Datenflussdiagramme müssten durch immer vorhandene und zum Kern der Familie gehörende Datenströme sowie optionale Datenströme erweitert werden, um uneingeschränkt für die Systemfamilienentwicklung tauglich zu sein.
Datenmodelle	• Durch *Entity-Relationship*-Modelle werden relationale Datenbanken beschrieben und können auch in der Systemfamilienentwicklung ohne Veränderung genutzt werden. • Die Berücksichtigung von variablen Anteilen in einem ER-Modell ist bislang nicht möglich, kann jedoch durch Weglassen von Tabellenspalten realisiert werden. Eine tiefergehende methodische Untersuchung von Variabilität in ER-Modellen wird zukünftigen Forschungsvorhaben überlassen.
Objektorientierte Modelle	• Die UML bietet 9 Diagramme für die objektorientierte Modellierung eines System an. • Derzeit unterstützt die UML keine Variabilität in den Diagrammen. Es gibt jedoch Erweiterungen, beschrieben in [Goma 2000], die dieses Problem beseitigen.
Goal-Oriented Modelle	• Anforderungen werden durch Ziele abstrahiert, die in Form eines Baumes angeordnet werden. Variabilitäten können durch die „oder“ Verknüpfung von Elementen in einem Zweig des Baumes modelliert werden. • Es ist allerdings nicht möglich, beliebige Vielfachheiten bei der Auswahl von Zielen aus einer Menge anzugeben. Für eine durchgängige Unterstützung der Systemfamilienentwicklung müssten die Modelle angepasst werden.

Tabelle 3.4: Ansätze zur Modellierung von Anforderungen und Einordnung in die Systemfamilienentwicklung

3.1.3 Prüfung von Anforderungen (*Validation and Verification*)

Alle Daten des Spezifikationsdokumentes müssen auf ihre Richtigkeit hin überprüft werden. Im Bereich der Prüfung gibt es die Validierung und die Verifikation. Die Prüfung der Folgerichtigkeit der Anforderungen und die Überprüfung des Modells auf Übereinstimmung mit den Anforderungen wird als Validierung bezeichnet. Es soll festgestellt werden, ob die Anforderungen schlüssig sind und das Modell den Anforderungen des Kunden entspricht, bevor das Modell implementiert wird. Validierende Methoden sind informeller Natur und können nur schwer oder gar nicht automatisiert werden.

Die Verifikation prüft, ob die Implementierung tatsächlich der Realisierung des Modells entspricht. Die Umsetzung aller Modellelemente in Programmcode wird in diesem Schritt überprüft. Diese Prüfungen sind automatisierbar, falls der Entwurf der Software genügend genau spezifiziert ist, die Entwickler also Gebrauch von formalen Spezifikationstechniken gemacht haben. Die bereits weiter oben genannten Nachteile der längeren Entwicklungszeit und des Verbrauchs von mehr Ressourcen stehen hier einer automatisierten Überprüfung des Modells gegenüber. Die automatischen Prüfungen von Modellen können nur mit Werkzeugen durchgeführt werden, welche die jeweilige formale Spezifikationssprache verarbeiten können. Die Verifikation erfordert damit entsprechende Werkzeuge und kann jedoch derzeit zur Überprüfung großer, komplexer Systeme nicht herangezogen werden, da heutige Verfahren noch nicht den nötigen Reifegrad besitzen [Brue 2000]. Im Rahmen dieser Arbeit wurde die Verifikation von Systemen nicht betrachtet. Demgegenüber wurden manuelle, validierende Verfahren betrachtet und in dem Lösungsansatz entsprechend berücksichtigt.

Reviews

Reviews werden im Softwareengineering verwendet, um erbrachte Entwicklungsleistungen zu überprüfen, evtl. zu überarbeiten und zu bewerten [Brue 2000]. Dabei wird der unpersönlichen Darstellung der Ergebnisse große Bedeutung beigemessen. Es sollen nicht einzelne Personen/Entwickler durch die Kritik angegriffen werden, sondern die Ergebnisse bzw. Zwischenergebnisse der Entwicklung sollen diskutiert und kritisch beleuchtet werden. Dabei sollen Probleme, Widersprüche oder Fehler aufgedeckt werden. Die Präsentation der *Review*-Ergebnisse erfordert ein gewisses Maß an Fingerspitzengefühl, um zu verhindern, dass sich die betreffenden Personen angegriffen fühlen und sich sofort in eine defensive Position begeben, die eine sachliche Diskussion völlig unmöglich macht.

Für ein *Review* werden zunächst die zu prüfenden Dokumente ausgewählt. Die Dokumente werden zusammen mit einer Liste der zu prüfenden Eigenschaften und den Prüfkriterien an die Personen versandt, die das *Review* durchführen. Die Prüfkriterien werden für den jeweiligen Fall festgelegt und enthalten beispielsweise die Einhaltung der Dokumentationsrichtlinien für Quellcode, die vollständige und verständliche Dokumentation der Testfälle, die Lesbarkeit der Dokumente oder die Verständlichkeit von Quellcode.

Im Bereich der Systemfamilien steht die Variabilität der erhobenen und modellierten Anforderungen im Vordergrund. Dazu müssen alle Stellen, die in den Dokumenten und Modellen als variabel gekennzeichnet sind, auf Sinnhaftigkeit, Vollständigkeit und Konsistenz hin überprüft werden. Diese Prüfungen hängen von der gewählten Technologie zur Umsetzung der Systemfamilie ab. Eine detaillierte Betrachtung von Konsistenzkriterien für *Requirements-Engineering*-Modelle von Systemfamilien befindet sich in Kapitel 4. Für eine Systemfamilie müssen *Reviews* der einzelnen Konfigurationen, also der Familienmitglieder, durchgeführt werden.

Testfälle

Nach [Schu 2000] werden für erhobene Anforderungen oder *Use-cases* Testfälle definiert, die es erlauben, das spätere System auf Einhaltung der Spezifikation zu überprüfen. Testfälle definieren jeweils Eingabegrößen und Startbedingungen und geben das erwartete Ergebnis vor. Nach Ausführung eines Testfalles werden Ist- und Soll-Ergebnisse verglichen, um eine eventuelle Verletzung des Testfalles und damit einen Fehler im System festzustellen. Derartige Testfälle stellen eine *Black Box* Sichtweise dar, bei der die Interna des Systems nicht betrachtet werden. Intensive Unterstützung von Werkzeugen gibt es in der Implementierungsphase, durch den eigens entwickelten Testcode, die so genannten *Unit*-Tests [Beck 2000]. Dafür werden Testfälle so weit heruntergebrochen, bis das Niveau einzelner Methoden oder Attribute erreicht ist. Datentypen, Übergabeparameter, Vor- und Nachbedingungen sowie Invarianten können durch Testcode geprüft werden. Damit werden in der Entwicklungsphase die in der *Requirement-Engineering*-Phase notierten Testfälle realisiert. In der vorliegenden Arbeit spielen derartige Tests keine Rolle und werden daher nicht weiter betrachtet.

Ein ereignisorientiertes System, kann durch formalisierte Ausdrücke nach [ChGa 1994] teilweise automatisch verifiziert werden. Allerdings gilt hier, wie auch bei anderen formalen Ansätzen, dass der Aufwand für die komplett formale Spezifikation berücksichtigt werden muss. Nicht jedes Projekt erfordert das gleiche Maß an Korrektheit für das entstehende System und nicht jeder Kunde kann oder will die Kosten dafür tragen.

Für die Systemfamilienentwicklung müssen Testfälle die Variabilität einzelner Komponenten berücksichtigen und einen Gesamttest der jeweils abgeleiteten Familienmitglieder erlauben. Ein Testfall muss sich damit je nach gewählten, optionalen Komponenten des Systems unterschiedlich verhalten und unterschiedliche Einzeltests abarbeiten. Durch die unterschiedlichen Konfigurationen und Parametrierungen werden Systemtests ebenfalls beeinflusst. Die nur rudimentär vorhandene Unterstützung systemfamilienbasierter Tests legt nahe, das gesamte Umfeld des Testens in zukünftigen Forschungsarbeiten genauer zu untersuchen.

Ansatz	Einordnung und Anpassungen für die Systemfamilienentwicklung
Reviews	• Im Rahmen von *Reviews* werden Ergebnisse einer Entwicklung durchgesehen und kritisch überprüft. • Für Systemfamilien müssen speziell die Stellen in Dokumenten und Modellen geprüft werden, die als variabel gekennzeichnet sind.
Testfälle	• Für Anforderungen sollten Testfälle definiert werden, die eine Prüfung der jeweiligen Anforderung erlauben. • Für die Systemfamilienentwicklung müssen Testfälle die Variabilität einzelner Komponenten berücksichtigen und einen Gesamttest der jeweils abgeleiteten Familienmitglieder erlauben.

Tabelle 3.5: Ansätze zur Überprüfung von Anforderungen und Einordnung in die Systemfamilienentwicklung

3.1.4 Anforderungsmanagement

Im Anforderungsmanagement sind die Prozesse enthalten, welche parallel zu der Erhebung, Modellierung und Prüfung von Anforderungen ablaufen. Nachdem die ersten Anforderungen erhoben sind und die Entwickler einen groben Überblick über das zu entwickelnde System haben, wird zu Beginn der Entwicklung eine Aufwandsschätzung durchgeführt. Diese Schätzung enthält auch die potentiellen Kosten des Systems, denen der Kunde selbstverständlich zustimmen muss, bevor das Projekt gestartet werden kann. Viele Risikofaktoren beeinflussen die Qualität dieser Schätzung und müssen daher so gut wie möglich im Vorfeld analysiert werden.

Weiterhin müssen während der gesamten Entwicklungszeit eines Projektes Software-Management Aufgaben, wie Planung, Organisation, Personalverwaltung, Leitungsaufgaben und Kontrolltätigkeiten ausgeübt werden. Die Betrachtung der Software-Management Tätigkeiten ist jedoch nicht Teil dieser Arbeit.

Die Software-Qualitätssicherung stellt eine weitere Aktivität im Rahmen der Softwareentwicklung dar. Nach DIN ISO 9126 ist Software-Qualität die Gesamtheit der Merkmale und Merkmalswerte eines Software-Produktes, die sich auf dessen Eignung beziehen, festgelegte oder vorausgesetzte Erfordernisse zu erfüllen. In [Balz 1996] werden dabei sechs Qualitätsmerkmale des DIN ISO 9126 Standards für das Produkt unterschieden.

- **Funktionalität** setzt das Vorhandensein von Funktionen mit festgelegten Eigenschaften voraus und ist gut, wenn die Funktionen die definierten Anforderungen erfüllen.
- **Zuverlässigkeit** ist gut, wenn die Software ihr Leistungsniveau unter festgelegten Bedingungen über einen festgelegten Zeitraum bewahren kann.
- **Benutzbarkeit** ist gut, wenn der Aufwand der Benutzung für eine festgelegte oder vorausgesetzte Benutzergruppe vertretbar ist, wird individuell bewertet.
- **Effizienz** ist gut, wenn die Software ein hohes Leistungsniveau bei gleichzeitig niedrigem Ressourcenverbrauch hat.
- **Änderbarkeit** ist gut, wenn der Aufwand für Änderungen gering ist.
- **Übertragbarkeit** ist gut, wenn die Software schnell an neue Umgebungen angepasst, installiert oder ausgetauscht werden kann und die gültigen Normen und Standards zur Übertragbarkeit erfüllt.

Im Rahmen der Software-Qualitätssicherung wird auf Einhaltung der obigen Qualitätsmerkmale im Verlauf einer Softwareentwicklung geachtet. Dabei kann der ISO9000-Ansatz [Funk 1999] zur Sicherung der Produktqualität genutzt werden.

Das *Total Quality Management* (TQM) [Humm 2002], das *Capability Maturity Model* (CMM) [SEI 2003] oder der *Software Improvement and Capability Determination* (SPICE) [Balz 1996] Ansatz werden genutzt, um die Qualität des Entwicklungsprozesses zu überprüfen und zu verbessern. Die Qualität wird durch so genannte *Audits* überprüft, bei denen zertifizierte Qualitätsingenieure eine unabhängige Befragung und Bewertung eines Unternehmens durchführen.

Im Bereich der Qualitätssicherung ist das *Quality Function Deployment* (QFD) [QFD 2003] ein etabliertes Verfahren, um aus Kundenwünschen über eine Matrix systematisch Produkteigenschaften abzuleiten, die in 9 Schritten erstellt wird, wie in [Balz 1996] und [Kami 2002] dargestellt.

1. Die Kundenanforderungen werden erhoben.
2. Die Anforderungen werden priorisiert.
3. Der Wettbewerbsvergleich ordnet das System in den Markt ein.
4. Die technischen Merkmale werden ermittelt.
5. Richtwerte für die Erfüllung der Merkmale müssen geliefert werden.
6. Abhängigkeiten zwischen technischen Merkmalen ermitteln.
7. Beziehungen in die Matrix eintragen.
8. Bedeutung jedes technischen Merkmals ablesen.
9. Vergleich mit den Wettbewerbern.

QFD setzt die Kundenwünsche in Produktmerkmale um, sodass das Endprodukt den Kundenwünschen entspricht. Die Kundenanforderungen werden zunächst erhoben, wobei mehr als 20 Anforderungen die entstehende Matrix zu groß werden lassen. Die Anforderungen werden sinnvoll nach Oberbegriffen gegliedert. Die Priorisierung zeigt, welche der Anforderungen wirklich wichtig sind und welche eher eine untergeordnete Rolle spielen. Für jede Anforderung wird ein Wert zwischen 1=unwichtig und 10=wichtig vergeben. Nun sollen die Kunden das neu zu entwickelnde System mit Konkurrenzprodukten vergleichen, wobei für jede Anforderung angegeben werden muss, ob sie von der Konkurrenz besser oder schlechter erfüllt wird. Die Entwickler müssen nun für die angegebenen Anforderungen technische Produktmerkmale finden, durch die die Kundenanforderungen erfüllt werden können. Jedem Produktmerkmal wird eine Optimierungsrichtung zugeordnet, die angibt, wie das Merkmal verändert werden muss, um es zu verbessern. Nun werden die Beziehungen der Anforderungen und der Produktmerkmale erarbeitet. Hat ein Produktmerkmal Einfluss auf eine Anforderung, wird dies in einer Beziehungsmatrix festgehalten. Nach Beendigung der Arbeiten an der Beziehungsmatrix ist abzulesen, ob alle Kundenanforderungen durch Produktmerkmale erfüllt werden. Dabei müssen alle stark gewichteten Kundenanforderungen in einer starken Beziehung zu Produktmerkmalen stehen, da sonst der Entwurf der Merkmale nicht den Kundenanforderungen entspricht. Nun werden die Beziehungen der Merkmale untereinander in eine weitere Matrix einge-

tragen. Bei einer positiven Beziehung beeinflusst die Verbesserung eines Merkmals ein anderes ebenfalls positiv, bei negativen Beziehungen negativ. Für jedes Merkmal wird auch die erwartete technische Schwierigkeit ermittelt, um das Merkmal umzusetzen (von 1=leicht bis 10=schwierig) und eine Maßgröße erarbeitet, die einen Richtwert für die Erfüllung des Merkmals liefert. Im letzten Schritt werden die Gewichtungen der Kundenanforderungen mit den Bewertungen aus der Beziehungsmatrix multipliziert und spaltenweise aufaddiert. Daraus ergibt sich die Bedeutung jedes Merkmals und eine Rangfolge für die Merkmale kann erstellt werden.

In der Systemfamilienentwicklung kann das QFD-Verfahren eingesetzt werden, um die für einen Kunden relevanten Merkmale herauszufiltern. Dem Kunden kann ein Vorschlag für eine an ihn angepasste Applikation aus der Systemfamilie unterbreitet werden. Allerdings sind in QFD nicht alle Beziehungen zwischen Merkmalen berücksichtigt. Beispielsweise haben das *Picture-In-Picture*-Merkmal und das Merkmal, Sendungen zeitversetzt sehen zu können, eine Beziehung zu der Anzahl der im System erforderlichen DVB-Karten und eine Beziehung zu der erforderlichen Leistungsfähigkeit des Systems. Diese Beziehung kann nicht in der QFD Matrix hinterlegt werden. Sie muss in einem anderen Modell verwaltet werden.

QFD eignet sich auch im Systemfamilienumfeld zur Bewertung und Priorisierung von Merkmalen und ergänzt daher die bereits erläuterten Ansätze. Wie oben gezeigt, können nicht alle Beziehungen in einer QFD Matrix berücksichtigt werden, weswegen ein anderes Konzept zur Modellierung der Beziehungen von Merkmalen notwendig ist.

Aufwandsschätzung

Für jedes Softwareprojekt ist es wichtig, den Aufwand für die Erstellung der Lösung so früh wie möglich abzuschätzen. Zum einen basieren die Kosten auf diesen Schätzungen, zum anderen werden Prognosen für den etwaigen Zeitpunkt der Fertigstellung/Auslieferung des Produktes möglich. Letztlich entscheiden die geplanten Kosten und die avisierte Entwicklungszeit über den Start des Projektes oder dessen frühzeitiges Ende. In der Diplomarbeit von [Klug 2003], die begleitend zu dieser Arbeit entstanden ist, werden verschiedene Verfahren der Aufwandsschätzung, wie Vergleichsmethoden, Kennzahlmethoden, *Function Points* sowie das *Constructive-Cost-Model* (COCOMO) und COCOMOII untersucht. Dabei hat sich das COCOMOII Verfahren durch die breite Akzeptanz sowie die dadurch häufige Nutzung und eigene Erfahrungen als zeitgemäß nutzbares Verfahren der Aufwandsschätzung erwiesen. Für die Systemfamilienentwicklung kann das COCOMOII Verfahren direkt übernommen werden, wobei lediglich die mehrfache Wiederverwendung variabler Anteile berücksichtigt werden muss. Durch Berechnungen unter Nutzung des COCOMOII Verfahrens in [Weis 1999] und [Klug 2003] hat sich gezeigt, dass sich die anfänglichen Investitionen in die Entwicklung der Systemfamilie erst ab einer verkauften Anzahl von vier Systemen amortisieren.

Dies stellt gleichzeitig eine Abbruchbedingung dar. Ist nicht zu erwarten, dass mehr als vier Systeme verkauft werden können, lohnt die Systemfamilienentwicklung nicht.

Die Anpassungen des COCOMOII Verfahrens für die Systemfamilienentwicklung von [Klug 2003] ermöglichen die Berücksichtigung der Kosten von Variabilitäten und können diese auch überprüfen und berechnen. Einzig die Kombination dieser Kostenrechnung mit der Konfiguration eines Familienmitgliedes bleibt offen und späteren Forschungsvorhaben vorbehalten.

Risikoabschätzung

Die Abschätzung des Aufwandes ist sehr stark von den Risikofaktoren eines Projektes abhängig. Je mehr solcher Faktoren in einem Projekt vorhanden sind und je weniger über die Faktoren bekannt ist, desto größer kann der Fehler sein, der in einer Schätzung liegt. Die folgende Liste stellt eine Zusammenstellung der in [Schi 2002] und [Schu 2000] enthaltenen Risikofaktoren dar.

- Die *Komplexität* eines Systems stellt ein hohes Risiko dar, wenn einzelne zu integrierende Komponenten des System sehr komplex sind, völlig neue Komponenten sind oder sehr viele verschiedene Komponenten in einem System zu integrieren sind.
  Eine Systemfamilie will die Zieldomäne so gut wie möglich abdecken, wodurch viele Einzelkomponenten zu integrieren sind. Darüber hinaus müssen alle Abhängigkeiten der einzelnen Komponenten berücksichtigt werden, um die Ableitung von Familienmitgliedern zu ermöglichen. Um das Risiko der hohen Komplexität einer Systemfamilie abzufangen, muss die Familie adäquat modelliert werden, wobei das Modell überprüfbar sein muss.
- Der *Reifegrad* der zu nutzenden *Technologie* bedeutet ein Risiko, wenn neue, eventuell ungetestete Algorithmen, Computersprachen oder Rechnerumgebungen in einer Entwicklung genutzt werden sollen. Erhöht wird das Risiko nochmals durch eine Entwicklergruppe mit wenig Erfahrung.
  Im Bereich der Systemfamilien sollten noch nicht etablierte Technologien gar nicht genutzt werden, da sich das Risiko durch die hohen anfänglichen Investitionen in die Familie zu stark erhöhen würde.
- Die *Qualität der Anforderungen* wird durch sich häufig verändernde Anforderungen, mangelnde Nutzerbeteiligung, unrealistische Kundenerwartungen, Konflikte zwischen Anforderungen oder unvollständige, zu viele, diffuse und zu umfangreiche Anforderungen negativ beeinflusst. Je mehr dieser Einzelfaktoren auftreten desto größer ist das Risiko, ein System zu entwickeln, das nicht den Wünschen des Kunden entspricht. Darüber hinaus verzögert sich die Entwicklung und wird zu teuer.
  Dieser Risikofaktor gilt gleichermaßen für alle Softwareprojekte und wird bei der Sys-

temfamilienentwicklung durch die größere Anzahl der beteiligten *Stakeholder* und die größere Anzahl von Anforderungen weiter verschärft. Größere Sorgfalt und mehr investierter Aufwand in der *Requirements-Engineering*-Phase ist demnach angebracht.

- Die *Testbarkeit* stellt ein erhöhtes Risiko dar, je mehr Systemkomponenten sich nur sehr schwer oder gar nicht testen lassen oder einen hohen und kaum schätzbaren Testaufwand erfordern.

  Die Entwicklung von Systemfamilien ist stark komponentenbasiert. Nach eigenen Erfahrungen geben insbesondere eingekaufte Komponenten von Drittanbietern oder bereits existierende Komponenten selten Einblick in ihre internen Strukturen und stellen meist keine Testfälle zur Verfügung. Damit entsteht ein schwer kalkulierbares Risiko, welches durch geeignete und von Drittanbietern zu fordernde Testbeschreibungen leicht reduziert werden kann. Durch die unternehmensinterne Entwicklung und Verwaltung von Komponenten (auch als *In-House*-Komponenten bezeichnet) können die Risiken einer komponentenbasierten Entwicklung reduziert werden. Allerdings müssen die Kosten für die firmeninterne Komponentenabteilung mit berücksichtigt werden.

Für jedes zu entwickelnde System muss individuell entschieden werden, wie groß das Risiko durch die einzelnen Faktoren ist. Für Systemfamilien werden die Risikofaktoren entsprechend verstärkt, wie in den einzelnen Punkten erläutert. Die Überprüfung der Risikofaktoren ist vorab nicht möglich. Erst nach Beendigung eines Projektes zeigt sich, wie gut die Risikoabschätzung war.

Umsetzungsmanagement

Nach [Schi 2002] setzt sich das Umsetzungsmanagement aus der Statusverfolgung und der Umsetzung der Nachvollziehbarkeit von Anforderungen, der *Traceability*, zusammen. Der Status eines Projektes ist jederzeit abrufbar, wenn ein definierter, verfolgbarer Prozess zusammen mit den Ergebnissen jedes Prozessschrittes vorhanden ist. Die Prozessschritte und Arbeitsergebnisse werden mit Hilfe von Workflow-Systemen, wie beispielsweise [Soph 2003], verwaltet und automatisiert verfolgt. *Traceability* bezeichnet die Beziehung von Artefakten, also Entwicklungsergebnissen, untereinander. In [Schi 2002] ist *Traceability* als Übersetzung aus dem IEEE1223 Standard wie folgt definiert.

Definition: Nachvollziehbarkeit (*Traceability*)

Der Grad, der angibt, wie stark die Beziehung zwischen zwei oder mehreren Artefakten eines Entwicklungsprozesses ist. Dies betrifft speziell Vorgänger-Nachfolger-Beziehungen oder hierarchische Beziehungen, z.B. die Beziehung zwischen Anforderungen und *Design*-Elementen.

Nach [KoSo 1998] werden Anforderungen mit ihrem Ursprung verbunden, um ihre Herkunft zu dokumentieren. Diese Verbindung enthält den Grund der Existenz einer Anforderung. In entgegengesetzter Richtung werden die Quellen mit den Anforderungen verbunden. Ausgehend von den Anforderungen werden Beziehungen zu den Elementen des Entwurfs genutzt, um festzuhalten, wo genau sich eine Anforderung im Entwurf und auch in der späteren Implementierung wiederfindet. Von den Entwurfselementen aus gibt es entsprechende Beziehungen zurück zu den Anforderungen, die aufzeigen, aus welchen Anforderungen ein Entwurfselement entstanden ist.

Für Systemfamilien sind die Beziehungen zwischen Anforderungen von großer Bedeutung, da Anforderungen für jeweils mehrere Applikationen Gültigkeit haben können und gleichzeitig möglichst frei zusammengesetzt werden sollen, um ein Familienmitglied zu erzeugen. Die oben beschriebenen Beziehungen können mit Hilfe eines Werkzeugs wie [DOOR 2003] genutzt und durch eine Typisierung erweitert werden. Über Matrizen können die Beziehungen überprüft werden, wobei die für Systemfamilien geforderte Variabilität nicht berücksichtigt wird. Die Varianten einer Familie mit den zugehörigen Anforderungen sind ebenfalls noch nicht berücksichtigt. Ein Werkzeug müsste die Gültigkeit einer Variante anzeigen, die ebenfalls auf den Beziehungen der Anforderungen untereinander beruht. Die Konfiguration eines Familienmitgliedes ist derzeit nicht möglich. Eine entsprechende Erweiterung der Datenmodelle und der Werkzeuge zur Unterstützung der Konfiguration wäre daher erforderlich.

Änderungsmanagement

Veränderungen der Problemstellung, veränderte Rahmenbedingungen, neue oder geänderte Kundenwünsche oder auch Veränderungen des technischen Umfeldes resultieren nach [Youn 2001] aus einer Änderung der Anforderungen. Hierbei ist es wichtig, die Änderungen nicht wahllos in ein System einfließen zu lassen, sondern gezielt vorzugehen und die Vor- und Nachteile von Veränderungen oder Neuerungen zu analysieren und erst dann, ganz bewusst, in das System einzuarbeiten.

Änderungen haben im Systemfamilienumfeld dieselben Folgen für ein laufendes System wie bei der Einzelsystementwicklung, allerdings können sich Änderungen auf alle Familienmitglieder auswirken und damit an Bedeutung gewinnen, wodurch einer Änderungen am Kern der Familie die größte Bedeutung zukommt. Eine Änderung der Anforderungen von Systemfamilien hat Auswirkungen auf die Varianten des Systems und damit auch auf die Konfiguration und Parametrierung der Komponenten. Derartige Veränderungen müssen durch den Entwicklungsprozess berücksichtigt werden und müssen jeweils gestützt durch Aufwandsschätzungen wohlüberlegt sein.

Ansatz	Einordnung und Anpassungen für die Systemfamilienentwicklung
Aufwandsschätzung	• Ein etabliertes Verfahren der Aufwandsschätzung ist das *Constructive-Cost-Model* II (COCOMOII), das den wahrscheinlichen Zeitaufwand und die wahrscheinlichen Kosten für ein Projekt errechnet. • In der Arbeit von [Klug 2003] wurde das COCOMOII Verfahren auf Systemfamilien mit Erfolg angewandt. Es sind jedoch Anpassungen für die Konfiguration eines Familienmitgliedes notwendig.
Risikoabschätzung	• Die vier wichtigsten Risikogruppen sind Komplexität, Reifegrad der Technologie, Qualität der Anforderungen und Testbarkeit. • Für Systemfamilien werden die einzelnen Risikofaktoren entsprechend verstärkt.
Umsetzungsmanagement	• Das Umsetzungsmanagement besteht aus der Statusverfolgung, die durch so genannte Workflow Systeme automatisiert bearbeitet werden kann, und der Umsetzung der Nachvollziehbarkeit von Anforderungen (*Traceability*). • Für die Nachvollziehbarkeit sind die Beziehungen zwischen Anforderungen von großer Bedeutung und können in einer einfachen Form von CASE-Werkzeugen verarbeitet werden. • Für Systemfamilien sind Beziehungen der Anforderungen zu dem Systemfamilienmodell und damit zu den Varianten erforderlich. Darüber hinaus müssen die Beziehungen bei der Konfiguration eines Familienmitgliedes überprüfbar sein, um die Gültigkeit der Konfiguration feststellen zu können.
Änderungsmanagement	• Veränderungen von Anforderungen wirken sich im Systemfamilienumfeld auf alle Familienmitglieder aus, beeinflussen die Varianten des Systems und damit auch die Konfiguration und Parametrierung von Komponenten. • Veränderungen müssen durch den Entwicklungsprozess berücksichtigt werden.

Tabelle 3.6: Ansätze zur Verwaltung von Anforderungen und Einordnung in die Systemfamilienentwicklung

3.1.5 Spezifikationsdokumente

Die Ergebnisse aller Arbeiten der *Requirements-Engineering*-Phase werden in einem Datenmodell hinterlegt, aus dem das Spezifikationsdokument abgeleitet wird. Die in dem Datenmodell zur Verfügung stehenden Informationen werden in ein Dokument überführt, wobei zum einen die Struktur des Dokumentes und zum anderen die Zuordnung der Informationen aus dem Datenmodell zu den Dokumentteilen von Bedeutung ist. Das entstehende Dokument ist eine Verhandlungbasis und stellt die rechtsverbindliche Vertragsgrundlage der Projektpartner dar.

Mögliche Strukturen von Spezifikationsdokumenten werden von diversen Standards vorgegeben, wie beispielsweise IEEE 830 beschrieben in [KoSo 1998], DIN EN 62079, DIN EN ISO 9001 oder IEEE1063 beschrieben in [Klug 2002]. Basierend auf diesen Standards wurden auf den praktischen Einsatz optimierte Strukturen von [Rupp 2002] und [Robe 1999] entwickelt, wobei der letztgenannte frei verfügbar ist und stellvertretend für die anderen Standards im Folgenden beschrieben wird.

Basierend auf Standards und eigenen Erfahrungen haben Suzanne und James Robertson in [Robe 1999] ein Schema für Spezifikationsdokumente und Anforderungen beschrieben. Die Erhebung von

Anforderungen wird durch ein Formular unterstützt, welches den Entwickler an die wichtigsten Punkte beim Erheben einer Anforderung erinnert.

In Abbildung 3.15 ist das Schema einer *Volere*-Anforderungskarte enthalten. Jede Anforderung ist durch eine Nummer eindeutig identifiziert und durch den Anforderungstyp einem der Kapitel des Spezifikationsdokumentes zugeordnet, was weiter unten erläutert wird. Handelt es sich um funktionale Anforderungen, wird der zugehörige *Use-case* mit notiert. Durch einen prägnanten Satz wird die Anforderung beschrieben, Hintergrund und Ursprung der Anforderung werden ebenfalls kurz notiert. Für den späteren Test werden schon beim Aufnehmen der Anforderung Metriken festgehalten, die einen sinnvollen Test überhaupt möglich machen. Der Grad der Zufriedenheit und Unzufriedenheit für eine Anforderung wird als Wert zwischen 1 und 5 angegeben. Ein Wert von 1 für die Zufriedenheit bedeutet dabei, dass ein Kunde an der betreffenden Anforderung gar nicht interessiert ist. Ein Wert von 5 für die Unzufriedenheit bedeutet, dass ein Kunde das Fehlen der Realisierung der betreffenden Anforderung höchst negativ bewerten würde. Die Bewertungsskala basiert auf den Erfahrungen der Autoren [Robe 1999], kann jedoch nach eigenen Vorstellungen abgeändert werden. Eine Wertung nach der Kano-Methode aus [Kano 1993] kann genutzt werden, um die fünf Werte zur Priorisierung der Anforderungen gleichzeitig mit dem subjektiven Empfinden des Kunden zu verknüpfen. Durch Fragen zur gewünschten Funktionalität, gekoppelt mit Fragen zu nicht-funktionalen Eigenschaften, entsteht eine Matrize, deren Analyse bessere Rückschlüsse auf die tatsächlich zu realisierenden Anforderungen zulässt.

Anforderung Nr.: *Eindeutige Nummer*	**Anforderungstyp:** *Einordnung im Volere-Spezifikationsdokument (Kapitel)*	**Ereignis/Use Case Nr.**: *Zuordnung*
Beschreibung:	*Ein Satz, der die Anforderung beschreibt.*	
Hintergrund:	*Warum ist diese Anforderung wichtig? Warum soll sie aufgenommen werden?*	
Ursprung:	*Woher stammt die Anforderung? Wer hat die Anforderung gestellt?*	
Test:	*Wie kann die Anforderung quantifiziert werden, um sie später zu testen?*	
Kundenzufriedenheit: *Wie stark ist der Kunde an dieser Anforderung interessiert?*	**Folgen von Unzufriedenheit:** *Was wäre, wenn die Anforderung nicht realisiert wird?*	
Abhängigkeiten: *Andere beeinflusste Anforderungen.*	**Konflikte:** *Widersprüchliche Anforderungen.*	
Weitere Materialien:	*Hinweise auf weitere Informationen zu dieser Anforderung.*	
Historie:	*Jeder, der etwas an der Anforderung verändert, muss sich hier eintragen.*	

Abbildung 3.15: *Volere*-Anforderungskarte

Abhängigkeiten zu anderen Anforderungen sind wichtig, um bei Änderungen an einer Anforderung eventuelle Einflüsse auf weitere Anforderungen berücksichtigen zu können. Dies ist insbesondere für Entwickler wichtig, die nicht an der initialen Entwicklung eines Systems beteiligt waren, also

nur für die Anpassung und Veränderung eines für sie neuen System verantwortlich sind. Hinweise auf widersprüchliche Anforderungen führen zu einer Revision der beteiligten Anforderungen oder zu unterschiedlichen Systemvarianten, wobei sich dann in Konflikt stehende Anforderungen ausschließen. Für ein schnelles Verständnis und kurze Einarbeitungszeiten sind Hinweise auf weitere Materialien und Informationen nützlich. Schließlich enthält die *Volere*-Anforderungskarte alle Personen in einer chronologischen Reihenfolge, die an der Erstellung und Veränderung der Anforderung beteiligt waren.

Der bereits angesprochene Anforderungstyp ordnet die Informationen der Anforderungskarte in die Struktur eines Spezifikationsdokumentes ein, das detailliert in [Robe 1999] beschrieben ist und dessen Grobstruktur in Abbildung 3.16 enthalten ist. Das Dokument ist dabei klassisch in Kapitel eingeteilt, die eine sinnvolle und einfach lesbare Struktur ergeben. Im Kapitel der *Project Constraints* werden die Hintergründe beschrieben, die zur Initiierung des Projektes geführt haben, sowie die Ziele, welche den Sinn und den Nutzen des zu erstellenden Produktes enthalten. Des Weiteren werden hier die beteiligten Personen, Kunden, Klienten der Kunden und andere Beteiligte aufgeführt. Die Nutzer des Produktes sind enthalten, um eventuelle spezielle Anforderungen zu erkennen, die aufgrund des Umfeldes entstehen, in dem sich dieser Nutzer befindet. Jedem Nutzer wird eine Priorität zugeordnet sowie der zu erwartende Beitrag für das Projekt, um ihn besser einordnen zu können. Es sind einschränkende Anforderungen enthalten, die durch das technologische Umfeld entstehen. Das Umfeld, in dem das Produkt installiert werden wird sowie weitere Systeme, die zur Benutzung des Systems wichtig sind, sind ebenfalls enthalten. Informationen über den zeitlichen und finanziellen Rahmen des Projektes sind in diesem Bereich für die Projektverwaltung von Bedeutung. Ein Glossar und Anforderungen, die aus dem Umfeld, wie beispielsweise gesetzlichen Bestimmungen entnommen sind, schließen diesen Teil des Anforderungsdokumentes ab.

Die *Functional Requirements* beschreiben Daten, logische Abläufe und Algorithmen. Der Kontext wird beschrieben, in dem sich die Arbeiten des Projekt bewegen und eine Partitionierung der Arbeiten findet statt, um eine Verteilung und Parallelisierung zu ermöglichen. Daneben wird auch das Produkt selbst einer Einordnung in das Umfeld unterworfen, was über *Use-cases* realisiert wird.

Nicht-funktionale Anforderungen enthalten Einträge zur Benutzerschnittstelle, dem Stil des Produktes, der Benutzbarkeit und Erlernbarkeit. Betrachtungen zur Geschwindigkeit sowie der Sicherheit des Produktes, der Genauigkeit von Angaben, Zuverlässigkeit, Verfügbarkeit und Kapazitätsangaben sind enthalten. Umwelteinflüsse, erwartete Technologieentwicklungen, benutzte Fremdapplikationen, Wartbarkeit, Portabilität, Ausfallsicherheit, kulturelle-/politische Rahmenbedingungen und gesetzliche Bestimmungen runden diesen Punkt ab.

Letztlich gibt es noch die *Project Issues*, welche offene Punkte, gekaufte Software, neue Probleme, die nötigen Arbeitsschritte mit zusätzlichen Bedingungen zur Auslieferung des Produktes, Risiken, Kosten, Schulungen der späteren Nutzer, eventuell zukünftig zu berücksichtigende Anforderungen und

Lösungsideen enthalten. Mit diesen Informationen sollte der Entwickler ein sinnvolles und vollständiges Anforderungsdokument erstellen.

In Abbildung 3.16 ist das Inhaltsverzeichnis eines *Volere*-Spezifikationsdokumentes abgedruckt. Die herkömmliche Erzeugung eines Dokumentes, welches dieser Struktur entspricht, wird mit Hilfe einer Textverarbeitung realisiert. Es hat sich jedoch gezeigt, dass die manuelle Bearbeitung von Spezifikationsdokumenten sehr schnell zu Inkonsistenzen im Dokument führt, die es bereits nach wenigen Änderungen unbrauchbar werden lässt. Diverse Produkte, von umfangreichen CASE-Werkzeugen bis hin zu kleinen Hilfswerkzeugen haben sich im Bereich des *Requirements Engineering* etabliert und werden im nächsten Abschnitt betrachtet.

Für die Systemfamilienentwicklung muss die Struktur eines Spezifikationsdokumentes angepasst werden. In der Einleitung fehlen Erklärungen zu Systemfamilien im Allgemeinen. Die Variabilitäten der Anforderungen müssen dokumentiert werden. Varianten müssen beschrieben werden und die Komponenten, aus denen die Varianten zusammengesetzt werden, müssen ebenfalls im Dokument vorhanden sein. Neben diesen generellen Forderungen muss es immer möglich sein, die Dokumentstruktur an die speziellen Erfordernisse einer Firma anzupassen.

Inhaltsverzeichnis

1. Einschränkungen (*Product Constraints*)
   - 1.1. Sinn und Zweck des Systems
   - 1.2. Alle *Stakeholder*
   - 1.3. Nutzer des Produktes
   - 1.4. Einschränkende Anforderungen
   - 1.5. Kodiervorgaben und Definitionen
   - 1.6. Relevante Fakten
   - 1.7. Annahmen
2. Funktionale Anforderungen (*Functional Requirements*)
   - 2.1. Umfang des Produktes
   - 2.2. Funktionale und Datenanforderungen
3. Nicht-funktionale Anforderungen (*Non-functional Requirements*)
   - 3.1. *Look and Feel* der Anwendung
   - 3.2. Anforderungen an die Benutzbarkeit
   - 3.3. Anforderungen an die Performanz
   - 3.4. Anforderungen an den Betrieb des Systems
   - 3.5. Anforderungen an die Wartbarkeit und die Portabilität
   - 3.6. Anforderungen an die Sicherheit
   - 3.7. Kulturelle und politische Anforderungen
   - 3.8. Gesetzliche Anforderungen
4. Zusätzliche Punkte (*Project Issues*)
   - 4.1. Offene Punkte
   - 4.2. Verwandte oder genutzte COTS
   - 4.3. Neue Probleme
   - 4.4. Arbeitsabläufe
   - 4.5. Migration alter Datenbestände
   - 4.6. Risiken
   - 4.7. Kosten
   - 4.8. Benutzerdokumentation
   - 4.9. Alles, was übrig war ...

Abbildung 3.16: Struktur eines *Volere*-Spezifikationsdokumentes

3.1.6 Requirements Engineering Werkzeuge

Die Verarbeitung von Anforderungen ist ohne Werkzeugunterstützung kaum zu leisten. Selbst für einfache Systeme, beispielsweise Bordinstrumente im Kraftfahrzeugbau, werden sehr umfangreiche

Spezifikationen in einem Umfang von mehr als tausend Seiten benötigt. Die Beziehungen der einzelnen Anforderungen untereinander gestalten die Benutzung der gedruckten Form einer Spezifikation sehr schwierig. Im Laufe der Entwicklung werden Anforderungen verändert oder angepasst, was den häufigen Ausdruck der Spezifikation erfordert. Darüber hinaus sind viele der heutigen Entwicklungsabteilungen über die ganze Welt verteilt, was die manuelle Bearbeitung einer Spezifikation mit Hilfe einer Textverarbeitung nicht zuletzt zu einem logistischen Problem werden lässt. Diese Probleme werden durch eine Reihe von auf dem Markt befindlichen Werkzeugen adressiert, die teilweise in ganze Werkzeuglandschaften für die Softwareentwicklung eingebettet sind. Anhang B enthält einen Überblick derzeit verfügbarer Werkzeuge, der aus [INCO 2002] stammt. Nach [Schi 2002] dominieren die Produkte DOORS der Firma *Telelogic* und *RequisitePro* der Firma *Rational* den Weltmarkt mit einem Anteil von fast 70%. Beispielhaft werden daher diese beiden Werkzeuge nach Auswertungen in [Schi 2002] kurz erläutert.

Anforderungen in DOORS werden als attributierte Objekte verwaltet und in beliebig strukturierbare Dokumente eingeordnet. Zwischen den Anforderungen können typisierte Beziehungen modelliert und in *Traceability*-Matrizen verwaltet werden. Weiterhin erlaubt DOORS die Versionierung der Daten und bietet Schnittstellen zu anderen Werkzeugen an. Unter Berücksichtigung der definierten Beziehungen zwischen Anforderungen können entsprechend „verlinkte“ HTML-Dokumente ausgegeben werden, wie auch ein Datenexport für gängige Textverarbeitungen möglich ist. *RequisitePro* arbeitet mit *Microsoft Word*-Dokumenten, wobei Anforderungen in einer Datenbank verwaltet und mit dem jeweiligen Textstück im *Word*-Dokument verknüpft werden. Des Weiteren lassen sich die Anforderungen auch mit beliebigen anderen Elementen verknüpfen. Diese Verknüpfungen werden in *Traceability*-Matrizen verwaltet und können so vom Entwickler analysiert und überprüft werden. Durch Nutzung weiterer Werkzeuge wie *ClearCase* und *ClearQuest* ist ein Versions- und Änderungsmanagement möglich. Auch hier ist ein Export in Form von HTML-Seiten möglich.

Die derzeit am Markt befindlichen Werkzeuge bieten keine explizite Systemfamilienunterstützung an, sodass sie für die Nutzung im Bereich der Systemfamilienentwicklung angepasst werden müssten. Das größte Manko ist die fehlende Systemfamiliensicht auf die Daten. Hier müssten größere Anpassungen an den Werkzeugen vorgenommen werden. Des Weiteren werden die Beziehungen zwischen den Anforderungen als einfache Verbindungen modelliert und können nur durch rudimentär automatisierte Überprüfungen verarbeitet werden, was ebenfalls entsprechende Anpassungen erfordert. In dieser Arbeit wurde daher zunächst auf die Erweiterung eines bestehenden Werkzeugs verzichtet, zugunsten eines eigenen Prototypen. Der Prototyp ist jedoch so aufgebaut, dass die Integration des Lösungsansatzes dieser Arbeit in existierende *Requirements Engineering* Werkzeuge im Rahmen zukünftiger Forschungsvorhaben möglich ist und nach Möglichkeit auch angestrebt werden soll.

3.1.7 Zwischenergebnis

Requirements Engineering teilt sich in die Phasen Erhebung, Modellierung sowie Überprüfung und einer dazu parallel ablaufenden Aktivität, der Verwaltung von Anforderungen, auf. In jeder Phase gibt es etablierte Ansätze, die in Tabelle 3.5 im Überblick zusammengefasst sind. Diese Ansätze sind jedoch nur bedingt für die Entwicklung von Systemfamilien tauglich, wobei sie speziell die Modellierung und Verwaltung der Variabilität einer Familie unvollständig unterstützen. Dennoch stellen sie wichtige Ansätze des Requirements Engineering dar, die auch in einer Systemfamilienentwicklung nicht fehlen dürfen. Auf diesen Ansätzen bzw. Abwandlungen davon baut der in Kapitel 4 dargestellte eigene Lösungsansatz dieser Arbeit auf. Die Spalten der Tabelle stellen Klassifikationskriterien dar, nach denen auch die im folgenden Kapitel 3.2 vorgestellten Systemfamilienentwicklungsansätze bewertet werden.

In der ersten Spalte ist vermerkt, ob der betreffende Ansatz Variabilität unterstützt. Neben den einzelnen variablen Elementen muss auch die Familie als solche modelliert werden, um daraus Varianten ableiten zu können. Die Möglichkeiten der Ansätze, das Modell der Systemfamilie zu unterstützen, werden in der zweiten Spalte notiert. Die dritte Spalte enthält die Möglichkeiten der Ansätze die Konfiguration von Varianten zu unterstützen. Die Konfiguration stellt dabei jeweils eine bestimmte Auswahl von Komponenten dar, die zur Realisierung einer Variante notwendig sind. Neben der Auswahl der Komponenten für eine Variante müssen diese Komponenten oftmals auch parametriert werde, der entsprechenden Variante also angepasst werden. Die Unterstützung der Ansätze für diese Einstellungen wird in der vierten Spalte vermerkt. Schließlich wird in der fünften Spalte die automatisierte Überprüfbarkeit der Ansätze bewertet.

Erfüllt ein Ansatz ein Kriterium, wird in die entsprechende Zelle der Tabelle ein „●“ eingetragen. Diese Wertung wurde auch vergeben, wenn der Ansatz zwar nicht für den im Kriterium angegebenen Einsatzzweck entwickelt wurde, trotzdem aber dafür genutzt werden kann. Ist ein Ansatz nicht ohne Veränderungen bzw. Anpassungen für den angegebenen Einsatzzweck nutzbar, wird in die entsprechende Zelle der Tabelle ein „⊙“ eingetragen. Ist ein Ansatz gar nicht für einen Einsatzzweck tauglich und kann er auch nicht mit vertretbarem Aufwand angepasst werden, wird dies durch ein „○“ in der entsprechenden Zelle dargestellt.

	Methode / Technik	Werden folgende Konzepte unterstützt?				
		Variabilität	Familienmodell	Konfiguration	Parametrierung	Überprüfung
Erhebung von Anforderungen	Dokumentanalyse	●	⊙	●	●	○
	Protokoll-Analysen	⊙	⊙	●	●	○
	Brainstorming	⊙	⊙	⊙	○	○
	Interviews / Fragenkataloge	⊙	⊙	⊙	○	○
	Workshops	●	●	⊙	⊙	●
	Ethnographie	●	⊙	●	●	●
	Kategorisieren	⊙	⊙	○	○	○
	Szenarien und Ziele	●	●	⊙	⊙	⊙
Modellierung von Anforderungen	Funktionale Modelle	⊙	⊙	⊙	⊙	⊙
	Datenmodelle	⊙	⊙	⊙	⊙	⊙
	Objektorientierte Modelle	⊙	⊙	⊙	⊙	⊙
	Goal-Oriented Modelle	⊙	⊙	○	○	●
Prüfen von Anforderungen	Reviews	●	●	●	●	●
	Testfälle	⊙	⊙	○	○	⊙
Anforderungsmanagement	Aufwandsschätzung (COCOMOII)	●	●	⊙	⊙	⊙
	Risikoabschätzung	●	●	⊙	⊙	○
	Umsetzungsmanagement	⊙	⊙	○	○	○
	Änderungsmanagement	⊙	⊙	○	○	○
Spezifikationsdokumente	Dokumentstruktur (*Volere*)	○	○	○	○	○

● Wird unterstützt.
⊙ Wird mit Einschränkungen unterstützt.
○ Wird nicht unterstützt.

Tabelle 3.7: Klassifikationsschema mit den Techniken der *Requirements-Engineering*-Phase.

Die erste Phase des *Requirements Engineering*, die Erhebung von Anforderungen, ist eine kreative Tätigkeit, die von Entwicklern ein umfassendes Verständnis der Domäne und hohe soziale Kompetenz erfordert. Die untersuchten Methoden und Techniken sind informeller Natur und liefern Ergebnisse, die sehr stark von der Erfahrung und den Fähigkeiten des Entwicklers abgängig sind. Die Erhebungsphase bietet, wie in obiger Bewertung gezeigt, Methoden und Techniken an, die im Rahmen der Entwicklung von Systemfamilien direkt oder mit geringen Anpassungen genutzt werden können. Die nicht automatisierte Prüfung von Anforderungen ist ebenfalls direkt in der Systemfamilienentwicklung anwendbar, da das Prüfvorgehen unverändert bleibt. Lediglich die bei einem *Review* zu prüfenden Kriterien sind an die Anforderungen der Systemfamilienentwicklung anzupassen.

Ein Mangel ist bei der Modellierung der Anforderungen festzustellen, da alle untersuchten Methoden und Techniken die Kriterien der Systemfamilienentwicklung nur eingeschränkt unterstützen. Die Modellierung kleiner, abgeschlossener Systemteile ist möglich. Die Modellierung der Systemfamilie, bestehend aus mehreren Komponenten mit den dazugehörigen Beziehungen der Komponenten untereinander, ist jedoch nicht möglich. Ansätze, die versuchen diesen Mangel zu beheben, werden im folgenden Kapitel analysiert. Die Methoden und Techniken der Anforderungsmanagementphase weisen bei der Nachvollziehbarkeit Lücken auf. Beziehungen zwischen den Komponenten der Systemfamilie, die die Konfiguration und Parametrierung eines Familienmitgliedes erlauben, können nicht modelliert werden. Eine Überprüfung eines Systemfamilienmodells ist durch die fehlenden Beziehungen nicht möglich. Schließlich unterstützt keine der untersuchten Strukturen für Spezifikationsdokumente die Systemfamilienentwicklung. Neue oder veränderte Strukturen für Spezifikationsdokumente sind hier notwendig.

Zusammenfassend sind die eindeutige und überprüfbare Modellierung einer Systemfamilie und die Struktur eines Spezifikationsdokumentes für die Systemfamilienentwicklung die Kritikpunkte an den untersuchten Methoden und Techniken. Die Modellierung einer Systemfamilie und die methodische Unterstützung der *Requirements-Engineering*-Phase werden im nächsten Kapitel näher betrachtet.

3.2 Systemfamilienentwicklung

Neben den im letzten Kapitel beschriebenen Methoden und Techniken des herkömmlichen *Requirements Engineering*, geht es in diesem Kapitel um spezialisierte Methoden der Systemfamilienentwicklung und deren Beitrag zum *Requirements Engineering* im Systemfamilienumfeld.

Die Softwaretechnik versucht kurze Entwicklungszeiten und eine hohe Qualität durch die Wiederverwendung bereits erbrachter Entwicklungsleistungen zu erreichen. Einmal erarbeitete Problemlösungen werden abgelegt, um beim nächsten Auftreten desselben Problems effizient zu einer Lösung beizutragen. Über die Zeit hinweg können Fehler und Unzulänglichkeiten wiederverwendbarer Komponenten durch Verbesserung und Weiterentwicklung behoben werden. Programm- bzw. Funktionsbibliotheken und Klassenbibliotheken sind schon länger genutzte Konzepte der Wiederverwendung, die sehr stark mit der jeweiligen Entwicklungsumgebung und Sprache verwoben sind. Derzeit werden allgemeinere Konzepte wie *Frameworks*, komponentenbasierte Softwareentwicklung oder die Systemfamilienentwicklung genutzt, um eine weitreichende Wiederverwendung zu realisieren. Neben diesen Konzepten der Wiederverwendung ist der Entwicklungsprozess und der Systemtyp einer Systemfamilie von Bedeutung.

Zum besseren Verständnis von Systemfamilien sind in Abbildung 3.17 unterschiedliche Systemtypen dargestellt. Für jeden der Systemtypen sind dabei unterschiedliche Entwicklungsstrategien erforderlich, wobei immer versucht wird, soviel Entwicklungsleistungen wie möglich wiederzuverwenden. Auf der Abszissenachse ist die Anzahl der zu erwartenden Produkte aufgetragen, die einen Bereich von individuell entwickelten Produkten bis hin zu Massenprodukten überspannt. Auf der Ordinatenachse werden Produkte bezüglich ihrer Zusammensetzung eingeteilt. Hardwaresysteme sind im unteren Bereich des Diagramms angesiedelt, reine Softwaresysteme stehen dagegen ganz oben.

Die Steuersoftware der Apollo Rakete war seinerzeit eine absolute Neu- und Individualentwicklung, wie auch der Roboterarm der Internationalen Raumstation individuell gefertigt wird, wobei allerdings im Gegensatz zur Steuersoftware auch eine Reihe von Hardwarebausteinen entwickelt werden musste. Schließlich sind die Tiefziehformen der Autoindustrie reine Einzelanfertigungen ohne jeden Softwareanteil. Alle in dieser Spalte beschriebenen Systeme erfordern einen hohen Aufwand bei der Entwicklung, der sich letztlich in einem sehr hohen Preis niederschlägt. Sie alle müssen jeweils für einen Spezialfall entwickelt werden, der kaum ein zweites Mal auftreten wird. Die Wiederverwendung ist höchstens im Bereich der genutzten Basistechnologien möglich, was sich auch wiederum negativ auf den Preis auswirkt.

Auf der anderen Seite stehen Massenprodukte, deren Entwicklungskosten auf eine große Zahl von Kunden umgelegt werden können und damit den Endpreis eines Einzelproduktes nur geringfügig beeinflussen. Allerdings muss sich der Kunde mit den Gegebenheiten solcher Systeme abfinden. Nur wenige Anpassungen sind möglich, sodass der Kunde seinen Arbeitsstil und sein Umfeld an das System anpassen muss. Im Bereich der Software stellt das Textverarbeitungssystem der Firma Microsoft ein Massenprodukt dar. Viele Personen nutzen *Word*, wobei sich aber alle an das von *Word* vorgegebene Bedienkonzept halten und mit dem Funktionsumfang abfinden müssen. Eine individuelle Anpassung von *Word* an den jeweiligen Benutzer ist nicht möglich. Eine Mischfrom aus Hard- und Software stellen Drucker dar. Auch sie können nur gemäß den Vorgaben des Herstellers genutzt werden. Ist beispielsweise die Bedienung eines Druckers kompliziert oder fehlt eine gewünschte Funktion, hat der Nutzer einzig beim Kauf die Möglichkeit aus der Produktpalette zu wählen. Die Anpassung eines Druckers an die Bedürfnisse des Benutzers ist aber nicht möglich. Schließlich stellen Mikroprozessoren, als reine Hardwarekomponenten, die letzte Gruppe in der Spalte der Massenprodukte dar. In einem Massenmarkt werden immer wieder ähnliche Produkte gefertigt, sodass große Anteile der Systeme durch Wiederverwendung erstellt werden können.

Als Kompromiss bieten die angepassten Individualsysteme dem Kunden eine umfassendere Berücksichtigung seiner Anforderungen bei gleichzeitig niedrigen Kosten, die unter denen eines Individualsystems liegen. Mit dem Konzept der Systemfamilien wird genau der Bereich zwischen Individualsystemen und Massenprodukten abgedeckt. Die Entwickler versuchen die Vorteile der Massenproduktion zu nutzen und den Kunden gleichzeitig so weit wie möglich zufrieden zu stellen.

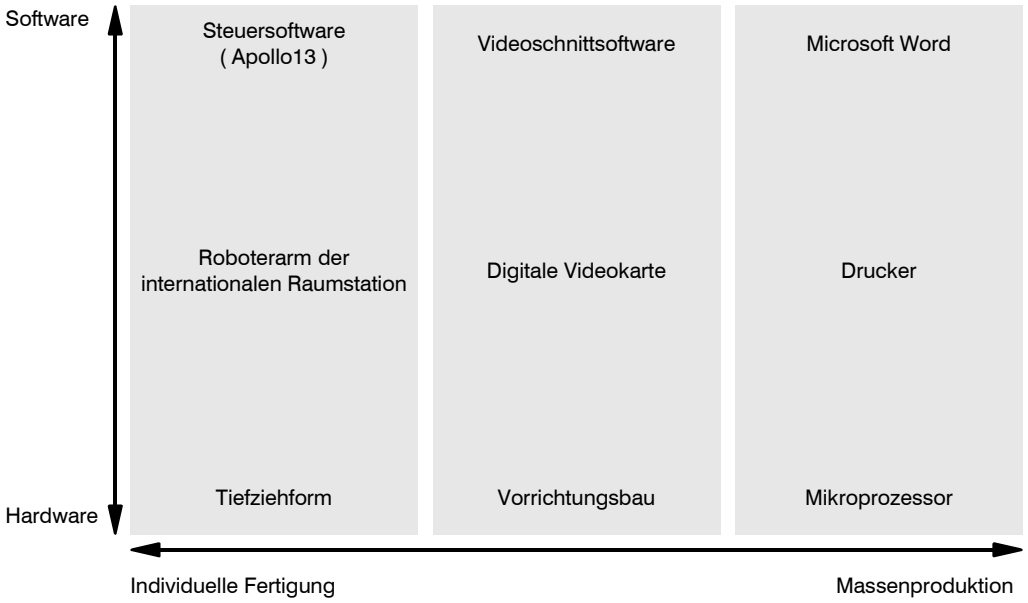


Abbildung 3.17: Systemtypen in der Systementwicklung

Die individualisierte Massenfertigung, in der mittleren Spalte aus Abbildung 3.17, stellt dabei eine sehr umfassende Art der Wiederverwendung dar und nutzt das Konzept der Entwicklung von Systemen als Familie. Bestehende Ansätze zur Entwicklung von Systemfamilien sind in die Bereiche Entwicklung der Systemfamilie selbst, Verwaltung der wiederverwendbaren Komponenten und Nutzung der Systemfamilie aufgeteilt.

Im so genannten *Six-Pack*-Modell [Duen 2001], dargestellt in Abbildung 3.18, sind die einzelnen Entwicklungsphasen enthalten. Im oberen Teil der Abbildung sind die drei Phasen des *Domain Engineering* [SEIB 2000] zu sehen, welche die Entwicklung einer Referenzarchitektur mit zugehörigen Komponenten zum Ziel haben. Im Mittelteil der Abbildung sind die Ergebnisse der *Domain-Engineering*-Phase enthalten, die gleichzeitig den Ausgangspunkt für die Entwicklung einer Applikation auf Basis der entstandenen Referenzarchitektur bilden. Somit hat ein Kunde die Möglichkeit, auf die Erfahrung, das Wissen und die bereits existierenden Komponenten einer bestimmten Domäne zurückzugreifen, indem er die Systemfamilie eines Anbieters aus der betreffenden Domäne nutzt. Die Entwicklung und Wartung einer Referenzarchitektur lässt sich mit den herkömmlichen Methoden der Softwareentwicklung nicht mehr effizient durchführen. Neue Kundenwünsche führen zu wiederholten Anpassungen der Systemarchitektur, was zu der in [BucA 1999] beschriebenen Architekturdrift führt. Die Architektur wird unübersichtlich, schwer verständlich und ist bereits nach kurzer Zeit kaum noch wartbar.

Das umfassende Konzept der Systemfamilien begegnet den oben beschriebenen Problemen. Seit dem Beginn der Systemfamilienentwicklung gibt es diverse Anstrengungen der Softwaretechnik, diese Idee weiter zu entwickeln und sie in der Praxis einzusetzen. Die für diese Arbeit relevante *Re-*

quirements-Engineering-Phase wird jedoch nicht von allen Methoden unterstützt. Aus einer Untersuchung im Rahmen des *Engineering Software Architectures, Processes and Platforms for System-Families* Projektes (ESAPS) [Duen 2001], der Arbeit von Czarnecki [Krzy 1998] und eigenen Recherchen entstand der in Abbildung 3.19 angeführte Überblick der Systemfamilienentwicklung. Es sind lediglich die Methoden aufgeführt, die im Rahmen dieser Arbeit untersucht wurden und eine Unterstützung der *Requirements-Engineering*-Phase bieten. Die allgemeine Zusammenstellungen der Methoden ist den in der Abbildung angegebenen Referenzen zu entnehmen.

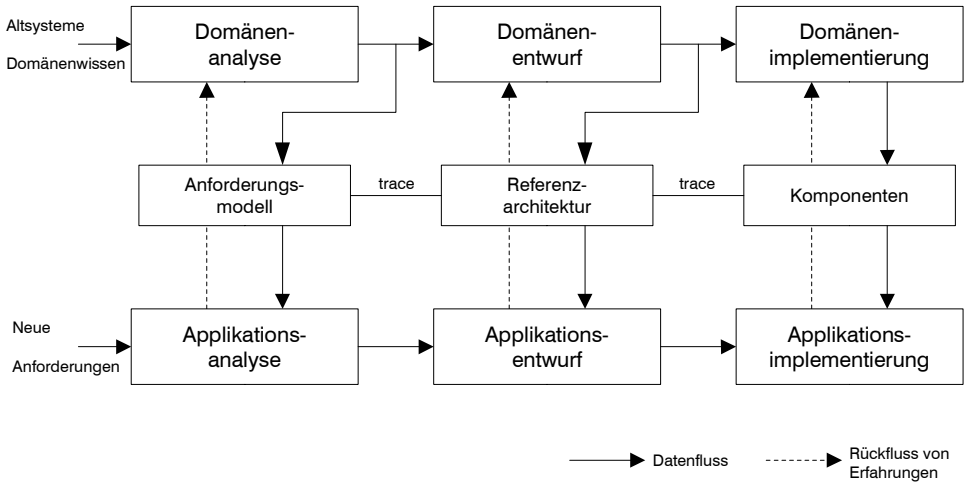


Abbildung 3.18: *Six-Pack* Modell der Systemfamilienentwicklung

Die Untersuchungen dieser Arbeit beziehen sich auf die grau unterlegten Teile aus Abbildung 3.19. Die einzelnen Methoden werden im Hinblick auf die im letzten Kapitel erstellte Struktur der *Requirements-Engineering*-Phase untersucht. Dabei geht es um die Zuordnung beziehungsweise Erweiterung der Methoden und Ansätze aus der Anforderungserhebungs-, Modellierungs- und Überprüfungsphase sowie den begleitenden Anforderungsmanagement Prozessen.

In den folgenden Unterkapiteln sollen dabei folgende Fragen geklärt werden:

- Auf welchen *Requirements-Engineering*-Methoden des letzten Kapitels bauen die auf Systemfamilien basierenden Methoden auf?
- Durch welche Konzepte, Ideen und Datenmodelle wird die Variabilität der Familie unterstützt?
- Welche neuen Prozessschritte werden genutzt?

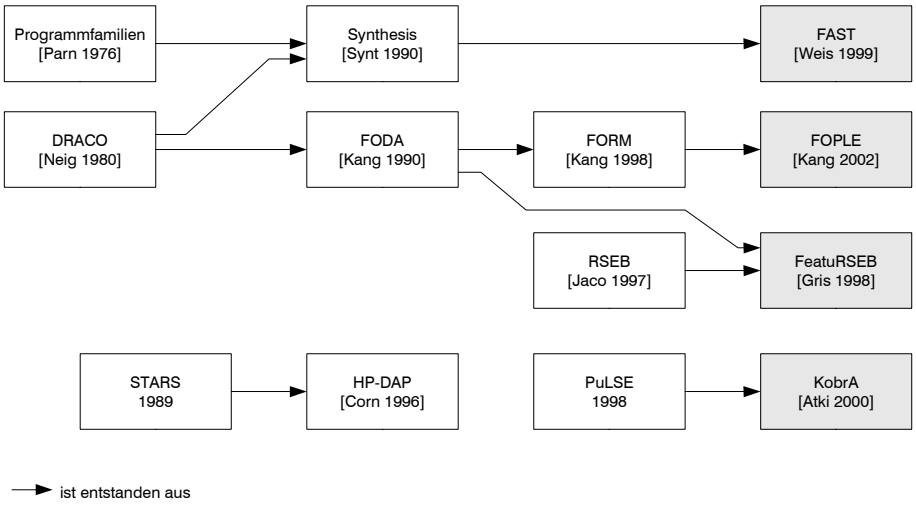


Abbildung 3.19: Historische Entwicklung des Systemfamilienansatzes

3.2.1 FAST

Der von Lucent Technologies entwickelte FAST Prozess (*Family-Oriented Abstraction, Specification, and Translation*) beschreibt ein Vorgehen zur Entwicklung einer Systemfamilie und der daraus ableitbaren Applikationen. Wie in [Weis 1999] dargestellt, besteht FAST aus einem *Activity Tree*, der alle Prozessschritte für die Entwicklung von und mit einer Systemfamilie enthält und einem *Artifact Tree*, der alle dabei entstehenden Informationen beschreibt. Das Ziel des Prozesses ist die Entwicklung einer eigenen, domänenspezifischen Sprache, mit der die jeweiligen Applikationen der Systemfamilie beschrieben und erstellt werden können. Jeder Prozessschritt ist durch ein PASTA-Modell (*Process and Artifact State Transition Abstraction*) definiert, welches beschreibt, wer, was, wann und wie ausführen soll, was dabei entstehen soll, welche Ressourcen dafür benötigt werden und schließlich, welche Bedingungen erfüllt sein müssen, damit ein Prozessschritt beendet werden kann. In Abbildung 3.20 ist ein Überblick der einzelnen FAST-Aktivitäten dargestellt.

In der *Domain-Engineering*-Phase werden Gemeinsamkeiten und Unterschiede diverser existierender Familienmitglieder untersucht, um für die Zukunft, aufbauend auf einer Systemfamilie, neue Applikationen herstellen zu können. Dieser Prozess wurde bei *Lucent Technologies* in über 25 Domänen erfolgreich angewandt [Weis 1999].

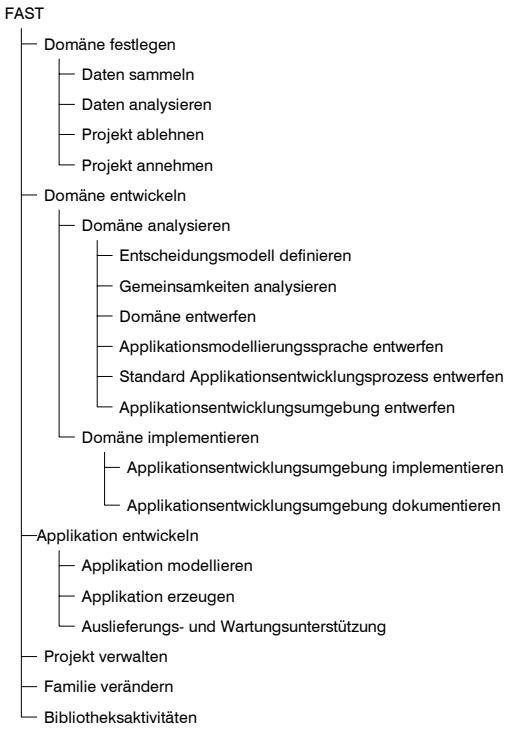


Abbildung 3.20: FAST-Prozess

Im Zuge des *Domain Engineering* erzeugen mehrere Experten ein Textdokument, welches die Gemeinsamkeiten und Unterschiede der Systemfamilie beschreibt. Dieser Vorgang wird als C*ommonality Analysis* bezeichnet. Das Dokument enthält Szenarien, die bestimmte Sachverhalte und Ideen verdeutlichen sollen. *Usability Scenarios* beschreiben Vorgänge aus Sicht des Benutzers. *Variability Scenarios* beschreiben schließlich die Unterschiede der einzelnen Familienmitglieder. Das entstehende *Commonality Analysis* Dokument ist in Abbildung 3.21 dargestellt. Es enthält alle Informationen in Form von Text und Bildern.

1. **Einleitung** – kurze Beschreibung der Domäne und der Beziehung zu anderen Domänen.
2. **Definitionen** – enthält das Glossar mit technischen Begriffen der Domäne.
3. **Gemeinsamkeiten** – eine Liste von Voraussetzungen, die für alle Familienmitglieder gelten.
4. **Variabilitäten** – eine Liste von Voraussetzungen, die für die Unterschiede der Familienmitglieder gelten.
5. **Variationsparameter** – eine Parameterliste der Variabilitäten, jeweils mit Wertebereich und Bindungszeitpunkt.
6. **Hintergründe** – eine Liste wichtiger Entscheidungen und Alternativen.
7. **Szenarien** – enthält beispielhafte Beschreibungen von Gemeinsamkeiten und Unterschieden

Abbildung 3.21: Struktur des *Commonality-Analysis*-Dokumentes

Das *Commonality-Analysis*-Dokument enthält ein so genanntes *Decision Model*, in dem alle Entscheidungen hinterlegt werden, die bei der Erzeugung einer Applikation getroffen werden müssen. FAST geht davon aus, dass alle Variabilitäten der Systemfamilie über Parameter beschrieben werden können, sodass sich die Entscheidungen bei der Entwicklung einer neuen Applikation auf diese Pa-

rameter beziehen. Parameter haben immer einen vorgegebenen Wertebereich und einen Standardwert, falls der Nutzer für einen Parameter nichts angibt.

Die Entwicklung des Systemfamilie ist in FAST gleichbedeutend mit der Entwicklung einer neuen, domänenspezifischen Sprache. Variabilitäten der einzelnen Familienmitglieder können durch die Konstrukte der neuen Sprache ausgedrückt werden, die von FAST als *Jargon* bezeichnet wird. Für jeden *Jargon* muss ein eigener Compiler entwickelt und in eine ebenfalls zu erstellende Werkzeuglandschaft bzw. Entwicklungsumgebung integriert werden. Mit einem neu entwickelten *Jargon* lassen sich dann Systeme aus einer Domäne beschreiben und in lauffähigen Code umsetzen. Ein gewünschtes Familienmitglied muss mit Hilfe des *Jargons* und der erfragten Parameter entwickelt und implementiert werden.

FAST lässt völlig offen, welche konkreten *Requirements-Engineering*-Methoden genutzt werden sollen. Speziell bei der Erstellung einer Applikation basierend auf der Systemfamilie ist die Unterstützung der FAST Methode durch das *Decision Model* nicht ausreichend, wenn nicht nur reine Parameter Variabilitäten darstellen, sondern auch das Verhalten des Systems variabel gestaltet werden soll. Das Entscheidungsmodell selbst ist ein guter Ansatz zur Beschreibung einer Systemfamilie aus Nutzersicht, kann jedoch in der derzeitigen Textform nur unzureichend automatisiert verarbeitet werden. Die Anforderungserhebung für eine Applikation basierend auf der Familie erfordert Detailkenntnisse der domänenspezifischen Sprache und aller Variationsparameter. Die von FAST vorgeschlagenen Konzepte zur Berücksichtigung der Variabilität können damit die Brücke zwischen einem Nutzer bzw. potentiellen Käufer eines Systems und den Entwicklern der Systemfamilie nicht schlagen. Der von FAST entwickelte Entwicklungsprozess ist hingegen sehr übersichtlich dargestellt. Neu und für die Systemfamilienentwicklung sehr wichtig ist die *Commonality / Variability* Analyse, in der die variablen Anteile der Familie herausgearbeitet werden. FAST schlägt hier aber keine konkreten Methoden vor. Ein weiterer Prozessschritt ist die Entwicklung einer domänenspezifischen Sprache, des oben erwähnten *Jargons*. Die Entwicklung einer eigenen Sprache stellt nicht für alle Domänen eine gute Lösung dar. Die Komplexität eines Videosystems würde an die Entwicklung einer domänenspezifischen Sprache sehr hohe Anforderungen stellen. In einem Videosystem sind mehrere Plattformen vorhanden, wie PCs, Handhelds und die speziellen Hardwareumgebungen der Videokarten. Das führt im Falle einer neuen Sprache gleich zu mehreren Compilern für die jeweiligen Zielsysteme. FAST schlägt zur Lösung dieser Probleme lediglich die Einbeziehung von sehr erfahrenen Entwicklern vor.

Ein Kunde kann die Möglichkeiten der ihm angebotenen Systemfamilie damit schlecht oder gar nicht überblicken. Er steht in direktem Kontakt zu einem Entwickler des Systems, der im Einzelfall entscheiden muss, ob sich das gewünschte System mit der domänenspezifischen Sprache erstellen lässt oder nicht. Schließlich können Entwickler durch Programmiersprachen dazu verleitet werden alles zu ermöglichen, was gefordert wird, auch wenn es wirtschaftlich nicht tragbar ist [Broo 1995].

In [Weis 1999] wird nicht deutlich, wie die einzelnen Phasen des *Requirements Engineering* von FAST unterstützt werden. FAST überlässt die Wahl der Mittel dem Entwickler. Demgegenüber stellt das *Decision Model* eine gute Möglichkeit dar, die Parametrierung des Systems über definierte Wege durch einen Fragenkatalog zu realisieren. FAST unterstützt die Modellierung des Systems mit den gängigen Diagrammen, passt diese jedoch nicht für die Modellierung variabler Bestandteile an. Gleiches gilt auch für die Prüfung der Anforderungen. In FAST werden Anforderungen aufgenommen und direkt durch eine domänenspezifische Sprache kodiert. Dieser Programmcode wird gestützt durch Werkzeuge in eine Applikation umgesetzt. Der syntaktische Test der Applikation ist durch die ebenfalls erstellte Entwicklungsumgebung möglich, die Überprüfung der Anforderungen ist jedoch schwierig, da ein Systemfamilienmodell fehlt.

Die Erstellung eines Spezifikationsdokumentes ist in FAST vorgesehen und wird durch eine Reihe von Vorgaben für dessen Struktur unterstützt. Eine Anpassung der Dokumente an firmeneigene Standards ist nicht explizit vorgesehen und muss daher in den FAST-Entwicklungsprozess integriert werden.

3.2.2 Feature Modellierung

Eine Möglichkeit zur Beschreibung und Modellierung der Variabilitäten einer Systemfamilie hat Kang 1990 in [Kang 1990] vorgestellt. Weiterentwickelt wurde die *Feature-Oriented Domain Analysis* (FODA) ebenfalls von Kang in [Kang 1998] und [Kang 2002]. Erweiterungen der Notation wurden von [Czar 2000] eingeführt. Ausgangspunkt für die Beschreibung einer Familie sind dabei Merkmale, die ein zukünftiger Benutzer zur Spezifikation des von ihm gewünschten Systems verwenden soll. Ein Merkmal beschreibt dabei einen „hervorstechenden oder unterscheidbaren, von einem Nutzer sichtbaren Aspekt, eine Qualität oder eine Charakteristik eines Softwaresystems oder Systems“. Durch eigene Erfahrungen und in Untersuchungen von [Kang 1998] und [Krzy 1998] hat sich herausgestellt, dass die Beschreibung eines Systems über dessen Merkmale für einen Benutzer oder Käufer des Systems einfach und schnell zu verstehen ist. Daher nutzen Kunden die Merkmale als Basis für die Auswahl des von ihnen gewünschten Systems.

Merkmale sind hierarchisch angeordnet, wobei die Wurzel des Baumes den Konzeptknoten enthält, der stellvertretend für die gesamte Systemfamilie steht. Jedes Merkmal des Baumes kann als optional oder obligatorisch markiert werden, wodurch eine Einteilung der Systemfamilie in einen Kern und eine Anzahl von variablen Anteilen vorgenommen wird.

In Abbildung 3.22 ist ein Teil der Merkmale des digitalen Videosystems als Merkmalmodell nach Kang dargestellt. Im linken Teil des Bildes sind die vom System unterstützten Datenformate zu sehen. Zwingend notwendig ist nur das MPEG-2-Datenformat, welches die Basis digitalen Fernse-

hens darstellt. Alle anderen Formate sind durch den Kunden wählbar. Neben dem obligatorischen Abspielen von Videosignalen könnte sich der Kunde auch entschließen, digitale Dias auf dem Videosystem zu betrachten. Dafür ist allerdings die Unterstützung des Photo-CD-Formates notwendig, was sich zum einen auf das zu verbauende CD-Laufwerk, andererseits aber auch auf die dafür notwendige Bildbetrachtungssoftware auswirkt. Dies ist durch die *requires*-Beziehung in Abbildung 3.22 modelliert.

Zum Schutz von Kindern vor Erwachsenenprogrammen kann der Kunde die Zugangskontrolle des Systems wählen. Damit würde die Fernbedienung personalisierbar sein, womit sich jeder Nutzer erst authentifizieren müsste, bevor er das System bedienen könnte. Für jeden Nutzer des Gerätes würden dann Rechte zur Ausführung bestimmter Funktionen des Systems vergeben. Ein Gehäuse muss vorhanden sein, allerdings kann der Kunde entscheiden, ob neben der Fernbedienung auch noch eine Bedienung über Taster direkt am Gerät möglich sein soll. Diese Wahlmöglichkeit darf natürlich nicht mehr vorhanden sein, wenn das Merkmal „Zugangskontrolle“ gewählt ist. Durch eine direkte Bedienung am Gerät könnte die Zugangskontrolle umgangen werden. Der Ausschluss der direkten Bedienung des Gerätes wird durch die *excludes*-Beziehung in Abbildung 3.22 modelliert.

Mit Hilfe der Merkmalmodellierung können Systemfamilien sehr eingängig und kundennah modelliert werden. Leider sind die Begriffe „Merkmal“ und „Merkmalmodell“ in [Kang 1990] nur vage definiert. Die Baumstruktur der Merkmalmodellierung steht fest, allerdings ist nicht klar, wie der Baum aufgebaut werden soll und welche Bedeutung den Ebenen des Baumes zukommt. Darüber hinaus sind komplexe Beziehungsgeflechte in einem FODA-Modell nicht realisierbar. Die in FAST genutzten Parameter können in FODA nicht berücksichtigt werden. Beispielsweise kann die Anforderung einer bestimmten Netzwerkbandbreite eines Videoservers, abhängig von der Anzahl der Klienten, in einem Merkmalmodell nach FODA nicht modelliert werden.

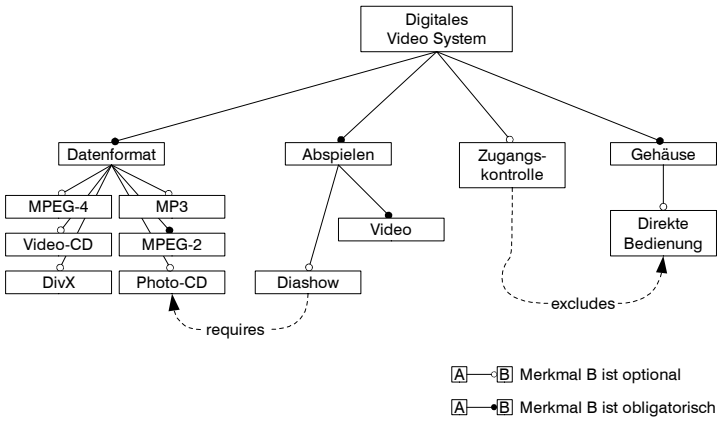


Abbildung 3.22: Merkmale des digitalen Videosystems

In den neueren Schriften [Kang 1998] und [Kang 2002] wird ein Vorgehen zur Entwicklung eines Merkmalmodells vorgeschlagen. Das Vorgehen orientiert sich an dem bereits vorgestellten *Six-Pack*-Modell. Für Merkmale des Systems werden vier Kategorien vorgestellt, die dem Entwickler als Schema dienen, um Merkmale zu erarbeiten. In der *Capability-Features*-Kategorie befinden sich alle von einem Nutzer wahrnehmbaren und vermarktbaren Merkmale eines System, sowie alle internen Merkmale, die eine Voraussetzung der erstgenannten Merkmale sind. In der *System-Environment-Features*-Kategorie werden alle Merkmale der Umgebung des zu entwickelnden Systems festgehalten. Zusätzlich benötigte Geräte sowie Schnittstellen und Protokolle sind hier als Merkmal enthalten. Domänenspezifische Technologien, die im Rahmen der Systemfamilienentwicklung genutzt werden sollen, legt *Feature-Oriented Product Line Engineering* (FOPLE) in der *Domain-Technology-Features*-Kategorie ab. Auf einer etwas allgemeineren Ebene befinden sich Merkmale der *Implementation-Technique-Features*-Kategorie. Implementierungsentscheidungen oder Entwurfsentscheidungen werden durch diese Merkmale beschrieben. Die vier Kategorien finden sich nun in einem in vier Ebenen aufgeteilten Merkmaldiagramm wieder. Es wird in [Kang 2002] nicht klar, welchen Vorteil ein derart aufgeteiltes Modell haben soll und Methoden der *Requirements-Engineering*-Phase werden gar nicht angesprochen. Hierfür werden lediglich eine Reihe von Richtlinien vorgestellt. Der Entwickler soll ein Glossar der Domäne aufbauen, Unterschiede der verschiedenen Produkte finden und diese dokumentieren. Die Empfehlungen sind zusammen mit der unklaren Definition der Merkmale nicht ausreichend für eine zielgerichtete Systemfamilienentwicklung. Das Konzept der Merkmalmodelle zur Beschreibung einer Systemfamilie ist sehr vorteilhaft und günstig für ein schnelles Verständnis der Familie, jedoch müssen Merkmalmodelle für eine durchgängige Softwareentwicklung präziser definiert und in einen familienbasierten Entwicklungsprozess eingebettet werden. Die grafische Notation der Merkmalmodelle ist mit Mehrdeutigkeiten behaftet. Zum einen leidet dadurch die Nutzbarkeit der Modelle, zum anderen ist es unmöglich, mehrdeutige Modelle durch Werkzeuge in einem Rechner zu be- und verarbeiten.

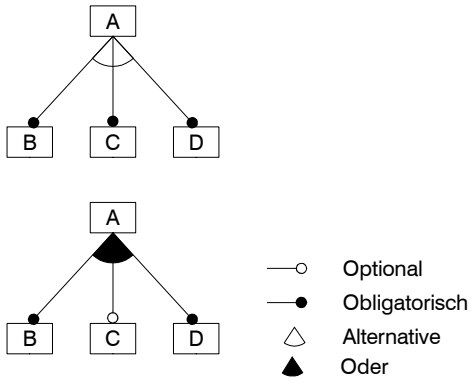


Abbildung 3.23: Mehrdeutige Merkmaldiagramme

In Abbildung 3.23 sind die Mehrdeutigkeiten von Merkmaldiagrammen nach [Czar 2000] dargestellt. Es geht um vier Merkmale, bezeichnet mit A bis D. Das obere Diagramm zeigt die alternativen Merkmale B, C und D. Unklar ist jedoch, wie die als obligatorisch gekennzeichneten Merkmale alternativ behandelt werden, da eigentlich nur ein Merkmal ausgewählt werden soll. So auch im zweiten Fall, unten im Bild. Wie stehen die als obligatorisch oder optional markierten Merkmale B bis D zu der ODER-Verknüpfung? Sind B und D immer in der Auswahl, wäre die logische ODER-Verknüpfung der Merkmale nicht mehr gegeben. Es ist unklar, wie diese Notation zu lesen ist.

Das Problem der Beziehungen zwischen Merkmalen wird als *Feature Interaction* bezeichnet. Es gibt diverse Arbeiten zu diesem Thema und unterschiedliche Ansätze zur Lösung der Probleme. In Tabelle 3.6 ist ein Überblick der Interaktionsmöglichkeiten mit einer jeweils kurzen Beschreibung zusammengestellt.

Interaktion	Beschreibung	Referenz
Use interaction	Wird genutzt, um Funktionalitäten unterschiedlicher Merkmale zu verbinden. Beispielsweise wird bei der Durchsicht der verfügbaren Filme auf der Platte des Videosystems die Abspielfunktion aufgerufen, falls der Benutzer einen der Filme sehen will.	[Lore 2001]
Share interaction	Sie besteht zwischen unterschiedlichen Merkmalen, die auf dieselbe Ressource zugreifen möchten. Der Nutzer hat eingestellt, dass das Videosystem den Start einer Aufnahme kurz am Bildschirm anzeigen soll. Betrachtet der Benutzer gerade die EPG Informationen eines Senders und fällt der Start einer Aufnahme genau in diese Zeitspanne, müssen sich Merkmale den Bildschirm – das so genannte *Overlay* – teilen.	[Lore 2001]
Modify behavior interaction	Sie tritt auf, wenn ein Merkmal ein anderes ausschaltet oder dessen Verhalten verändert. Je nach dem welche Hardware der Benutzer zum Abspielen von CDs gewählt hat, können Videodaten auf CD ausgelagert werden oder nicht. Hat sich der Nutzer gegen einen CD-Brenner entschieden, wird das Merkmal „Daten auslagern“ deaktiviert.	[Lore 2001]
Incompatibility	Diese Interaktion tritt auf, wenn die Anforderungen mehrerer Merkmale nicht zueinander passen. Beispielsweise führen widersprüchliche Anforderungen in Merkmalen zu deren gegenseitigem Ausschluss.	[Blai 2001]
Behavior	Tritt auf, wenn ein Merkmal als Folge des Vorhandenseins eines anderen Merkmals, ein unterschiedliches Verhalten aufweist.	[Blai 2001]
Aggregation	Ein Merkmal besteht aus mehreren anderen Merkmalen.	[Jia 2001]
Generalize	Beziehung zwischen einem abstrakten und einem verfeinerten Merkmal.	[Jia 2001]
Dependency	Das Verhalten zweier Merkmale beeinflusst sich gegenseitig.	[Jia 2001]
Association	Eine allgemeine Beziehung, welche die semantische Ähnlichkeit zweier Merkmale beschreibt.	[Jia 2001]
Mutex	Nur in einem bestimmten Kontext tritt ein Konflikt zwischen zwei Merkmalen auf.	[Clau 2001]
Requires	Ursprüngliche Beziehung der FODA-Methode	[Kang 1990]
Excludes	Ursprüngliche Beziehung der FODA-Methode	[Kang 1990]

Tabelle 3.8: Feature Interaction Übersicht

Nur ein Teil der Autoren liefert konkrete Vorschläge zur Modellierung und Weiterverarbeitung der beschriebenen Beziehungen zwischen Merkmalen. [Lore 2001] modelliert die Beziehungen mit Hilfe von gefärbten Petri-Netzen, die um *Message Sequence Charts* (MSC) erweitert wurden. Die MSCs werden genutzt, um die Abfolge einer geordneten Menge von Ereignissen zu modellieren, die wie-

derum das Petrinetz steuern. Es geht bei diesem Ansatz um die Interaktion von Merkmalen zur Laufzeit des Systems. Die Unterstützung der Merkmalauswahl zur Erstellung eines Familienmitgliedes ist jedoch nicht in diesem Ansatz enthalten. Diese Auswahl ist ein zentral wichtiger Anteil der *Requirements-Engineering*-Phase im Systemfamilienumfeld. Die gesamte Systemfamilienentwicklung zielt darauf ab, ein System zu entwickeln, mit dem es möglich sein muss, Varianten so schnell und einfach wie möglich abzuleiten. Die Auswahl einer Variante ist integraler Bestandteil der eigenen Lösung, beschrieben in Kapitel 4.

Mit Hilfe künstlicher Intelligenz beschreibt [Hein 2001] eine Möglichkeit zur Konfiguration von Familienmitgliedern basierend auf Merkmalen und deren Beziehungen untereinander. Das Merkmalmodell wird in ein *Customer-Feature*-Modell und ein *System-Engineering-Feature*-Modell aufgeteilt. Durch einen interaktiven Prozess soll der Nutzer durch die Merkmalauswahl geführt werden. Die einzigen Beziehungen, die [Hein 2001] zulässt, sind die bereits in [Kang 1990] definierten *Requires*- und *Excludes*-Beziehungen. Wurden alle Produktvarianten des Merkmalmodells vorhergesehen, handelt es sich um ein statisches Modell, das als *Routine Configuration* bezeichnet wird. Inkonsistenzen werden durch *Backtracking* aufgelöst. Produktvarianten, die außerhalb der ursprünglichen Familie liegen, werden als *Innovative Configuration* bezeichnet und müssen als neue Merkmalauswahl in die Familie aufgenommen werden. Dabei können neue Merkmale durch Modellierung neuer Beziehungen in die Familie eingepasst werden oder entstehende Inkonsistenzen müssen durch Hinzunahme weiterer Merkmale aufgelöst werden. Die konkrete Umsetzung der Beziehungen im Modell bleibt der Autor schuldig und auch die Beschränkung auf die bereits von Kang eingeführten Beziehungen reicht, wie schon gezeigt, nicht aus.

In einem größeren, europäischen Projekt, dem ESAPS-Projekt, welches das gesamte Umfeld der Systemfamilienentwicklung analysiert hat, wird die Merkmalmodellierung komplementär zur Domänenanalyse gesehen. Während der Domänenanalyse werden die Konzepte der Domäne modelliert. Diese Konzepte können erst anschließend durch Merkmale näher beschrieben werden [Savo 2001]. Beziehungen zwischen Merkmalen werden gesondert modelliert, was in Abbildung 3.24 dargestellt ist. Damit wird die Variabilität der Systemfamilie durch Merkmale ausgedrückt. Das Konsistenzproblem, welches entsteht, wenn die Beziehungen zwischen Merkmalen nicht berücksichtigt werden, wird zwar in [Savo 2001] beschrieben, die Umsetzung der reinen Modellelemente, die eine Beziehung beschreiben, in definierte und weiter verarbeitbare Ausdrücke, bleibt dieser Artikel schuldig.

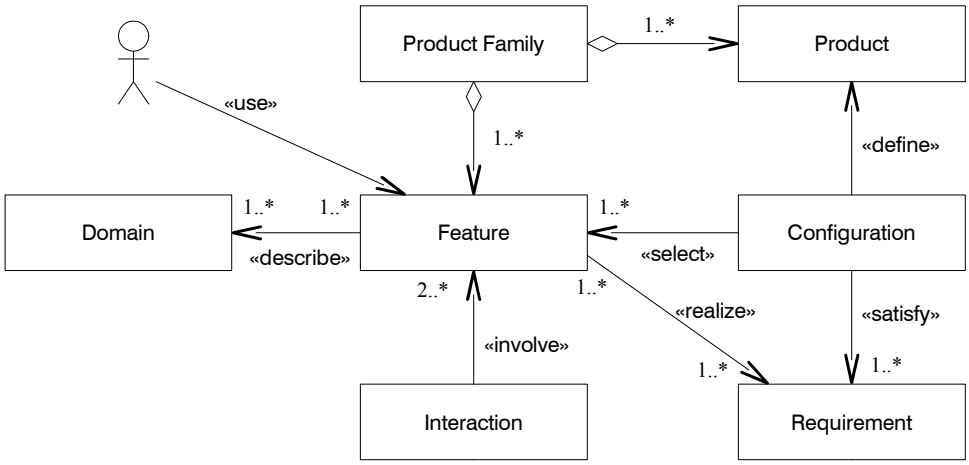


Abbildung 3.24: Merkmalmodellierung in ESAPS

3.2.3 FeatuRSEB

Eine sehr weitreichende Entwicklungsmethode stellt FeatuRSEB (*Featured* RSEB) als Zusammenfassung von FODA und RSEB (*Reuse-driven Software Engineering Business*) dar. RSEB ist ein *Use-case* zentrierter Ansatz zur Entwicklung von Systemfamilien. *Use-cases* werden als Dreh- und Angelpunkt bei der Entwicklung einer Systemfamilie verwendet. Durch Variationspunkte ist die Erstellung von Familienmitgliedern aus wiederverwendbaren Komponenten möglich, wobei Variationspunkte in *Use-cases* und in Klassendiagrammen zum Einsatz kommen. Dabei werden die verschiedenen Varianten der Teile der Systemfamilie an die Variationspunkte geheftet, sodass ein Modell mit vielen Varianten entsteht. Der Entwickler kann sich nun an den jeweiligen Variationspunkten für eine bestimmte Variante entscheiden und so ein Mitglied der Familie erzeugen.

Mehrere Prozesse beschreiben die Vorgehensweisen bei der Entwicklung der Systemfamilie. Eine Schichtenarchitektur der Systemfamilie, die Komponentenentwicklung und die Entwicklung der einzelnen Applikationen werden in RSEB beschrieben. Was fehlt, ist eine globale Sichtweise auf die Systemfamilie, die von FODA geliefert wird. FODA kann die Gemeinsamkeiten und Unterschiede der Systemfamilie darstellen und bietet daher dem Entwickler und Kunden einen guten Überblick über die Systemfamilie. Das Merkmalmodell spielt daher eine zentrale Rolle in FeatuRSEB und stellt einen Katalog aller Gemeinsamkeiten und Unterschiede dar.

Erste Ansätze zur Konfiguration einzelner Applikationen aus der Systemfamilie heraus sind in FeatuRSEB durch die Verwendung der Merkmalmodellierung gegeben. Allerdings werden keinerlei Erweiterungen für die Merkmalmodelle vorgestellt, sodass die gleichen Kritikpunkte gelten, die schon in Kapitel 3.2.2 angesprochen wurden. Aus dem Merkmalmodell können Varianten abgeleitet wer-

den, deren Gültigkeit, insbesondere in Bezug auf die Funktion der zu erzeugenden Applikation, nicht garantiert werden kann.

3.2.4 KobrA

Die „Komponentenbasierte Anwendungsentwicklung“ KobrA-Methode [Atki 2002] basiert auf dem *Product Line Software Engineering* (PULSE™) des *Fraunhofer Institute for Experimental Software Engineering* (IESE). Die Entwicklung der Systemfamilie basiert auf drei Prozessteilen. Innerhalb der Kontext-Realisierung wird ein Unternehmensmodell mit Konzept- und Prozessdiagrammen zur Beschreibung der Geschäftsprozesse, ein Strukturmodell zur Beschreibung von strukturellen Interaktion mittels Klassendiagrammen, ein Aktivitätsmodell bestehend aus *Use-cases* und Aktivitätsdiagrammen, ein Interaktionsmodell bestehend aus Sequenz- und Kollaborationsdiagrammen und ein Entscheidungsmodell, englisch *Decision Model*, erstellt. Nach der Kontext-Realisierung findet das *Framework Engineering* für die Systemfamilie oder das *Application Engineering* für ein Familienmitglied statt. Das *Framework* enthält die Referenzarchitektur der Systemfamilie, die das Ableiten einer Applikation aus den gewünschten Komponenten erlaubt. Auch das *Framework* selbst stellt eine Komponente dar, wodurch KobrA als rekursiver Prozess bezeichnet wird, d.h. alles wird als Komponente behandelt und kann wiederum aus Komponenten zusammengesetzt sein. Im dritten Teil der KobrA-Methode findet die Implementierung der Komponenten statt. Zum einen können bereits existierende Komponenten genutzt werden oder, falls existierende Komponente den gegebenen Anforderungen nicht entsprechen, müssen entsprechende Komponenten neu implementiert werden.

Das zentrale Konzept von KobrA ist die Trennung von Produkten und Prozessen. Produkte stellen dabei die Ergebnisse der Entwicklung dar und werden unabhängig von den Prozessen und zeitlich davor definiert. Alle Produkte zielen auf die Beschreibung von individuellen Komponenten ab, in denen die Variabilität der Familie enthalten ist.

Die *Requirements-Engineering*-Phase in KobrA basiert auf textuellen Beschreibungen und *Use-case* Diagrammen der UML, die Variationen über Stereotypen modellieren. Die Variationen der einzelnen Komponenten werden in einer Tabelle vermerkt, indem für jede Variante der Familie die benötigten Merkmale eingetragen werden, was in KobrA als *Product Map* bezeichnet wird. Für die spätere Ableitung von Applikationen wird in KobrA noch ein *Decision Model* erarbeitet, in dem alle Variationspunkte enthalten sind, wobei hier eine Ähnlichkeit zu den Merkmalmodellen von Kang besteht, ohne dass die Merkmalmodelle von KobrA genutzt werden. Vor der Ableitung einer Variante muss für jeden Variationspunkt entschieden werden, ob er im System vorhanden ist. Darüber hinaus sind auch Parameter als Variationspunkte möglich, wobei diese für eine konkrete Applikation mit einem Wert belegt werden müssen. Im *Decision Model* sind eine Reihe von Fragen enthalten, die der Kunde zur Ableitung einer Applikation beantworten muss. Verlangt ein Kunde etwas, was nicht als Kom-

ponente in der Systemfamilie enthalten ist, muss diesem Wunsch durch Integration einer neuen Komponente in die Familie nachgekommen werden. Dabei werden Ähnlichkeiten zu bereits existierenden Komponenten der Familie ausgenutzt. Die Interaktionen der einzelnen erfragten Variabilitäten des *Decision Models*, als *Effect* bezeichnet, sind ebenfalls in textueller Form im Modell hinterlegt und müssen vom Entwickler manuell aufgelöst werden.

Wie in Tabelle 3.7 dargestellt, ist für jede Entscheidungsmöglichkeit eines Variationspunktes die Auswirkung auf die Elemente der Familie und damit auf die Elemente des zu erstellenden Familienmitgliedes hinterlegt. Weitere Interaktionen zwischen der Variabilitäten sind jedoch nicht modellierbar und auch nicht automatisiert zu verarbeiten. Auch die angegebenen Auswirkungen (*Effects*) sind nicht automatisiert auswertbar.

ID	Variation	Resolution	Effect
1	Video-Aufzeichnungslänge	param(length)	3: adjust size
2	*Video-On-Demand-Server*	yes	-
		no (default)	remove Component videoStream
3	Plattengröße	param(size)	-

Tabelle 3.9: Teil eines KobrA *Decision Models* des digitalen Videosystems

KobrA stellt eine umfassende Methode zur Entwicklung von Systemfamilien dar und ist der neueste, in dieser Arbeit untersuchte Ansatz. Das von KobrA genutzte *Decision Model* zur Erfassung und Verarbeitung von Variabilitäten in der *Requirements-Engineering*-Phase stellt eine gute Basis für die Ableitung eines Familienmitgliedes dar, kann jedoch mit der Übersichtlichkeit und Verständlichkeit der Merkmalmodelle nicht konkurrieren. Das *Decision Model* kann nicht automatisiert verarbeitet werden, insbesondere müssen die enthaltenen Auswirkungen (*Effects*) manuell durch den Entwickler überprüft werden.

3.2.5 Wertung

Die Wertung der im letzten Kapitel vorgestellten Ansätze hat sich durch die systemfamilienspezifischen Ansätze dieses Kapitels leicht verschoben. Hinzugekommen sind nunmehr ein etabliertes und leicht verständliches Mittel der Modellierung von Systemfamilien. Über Merkmale und die von Kang entwickelten Merkmalmodelle lassen sich die Variabilitäten von Systemfamilien sehr gut modellieren und durch die Merkmaldiagramme visualisieren. Die Kritik dieses Kapitels bezieht sich auf die Vollständigkeit der Merkmalmodelle. Es ist nicht klar, wie die in Kapitel 3.2.2 beschriebenen Interaktionen der Merkmale berücksichtigt werden sollen, und es ist auch nicht klar, wie derartige Interaktionen automatisiert verarbeitet werden können. Darüber hinaus ist auch die Integration der

Merkmalmodellierung in einen *Requirements-Engineering*-Prozess für Systemfamilien nur unvollständig vorhanden.

In Tabelle 3.8 sind die in Kapitel 3.1 und in diesem Kapitel erarbeiteten Ergebnisse zusammengefasst. Die Tabelle enthält die Einordnung der untersuchten Methoden der Systemfamilienentwicklung in die Phasen des *Requirement Engineering* mit einer Bewertung der Methoden.

RE Phase / Ergebnis	**Methode / Technik**	**Wie werden die Konzepte / Methoden unterstützt?**								
		FAST	**FODA**	**FeatuRSEB**	**KobrA**	**Variabilität**	**Familienmodell**	**Konfiguration**	**Parametrierung**	**Überprüfung**
Erhebung von Anforderungen	Dokumentanalyse	⊙	●	●	⊙	●	⊙	●	●	○
	Protokoll-Analysen	⊙	⊙	⊙	⊙	⊙	⊙	●	●	○
	Brainstorming	⊙	⊙	⊙	⊙	⊙	⊙	⊙	○	○
	Interviews / Fragenkataloge	⊙	⊙	⊙	⊙	⊙	⊙	⊙	○	○
	Workshops	⊙	⊙	⊙	⊙	●	●	⊙	⊙	●
	Ethnographie	○	○	○	○	●	⊙	●	●	●
	Kategorisieren	○	○	○	○	⊙	⊙	○	○	○
	Szenarien und Ziele	○	⊙	⊙	⊙	●	●	⊙	⊙	⊙
Modellierung von Anforderungen	Funktionale Modelle	●	●	●	●	⊙	⊙	⊙	⊙	⊙
	Datenmodelle	⊙	●	●	●	⊙	⊙	⊙	⊙	⊙
	Objektorientierte Modelle	⊙	⊙	●	●	⊙	⊙	⊙	⊙	⊙
	Goal-Oriented Modelle	○	○	○	○	⊙	⊙	○	○	●
Prüfen von Anforderungen	Reviews	●	⊙	●	●	●	●	●	●	●
	Testfälle	●	○	●	●	⊙	⊙	○	○	⊙
Anforderungsmanagement	Aufwandsschätzung	⊙	⊙	⊙	●	●	●	⊙	⊙	⊙
	Risikoabschätzung	●	⊙	●	●	●	●	⊙	⊙	○
	Umsetzungsmanagement	●	⊙	●	●	⊙	⊙	○	○	○
	Änderungsmanagement	●	⊙	●	●	⊙	⊙	○	○	○
Spezifikationsdokumente	Dokumentstruktur (*Volere*)	●	⊙	●	●	○	○	○	○	○
Systemfamilienentwicklung	FAST					●	●	⊙	⊙	⊙
	Merkmalmodelle					●	●	⊙	⊙	○
	FeatuRSEB					●	●	⊙	⊙	⊙
	KobrA					●	●	⊙	⊙	⊙
		● Wird unterstützt. ⊙ Wird mit Einschränkungen unterstützt. ○ Wird nicht unterstützt.								

Tabelle 3.10: Komplette Bewertung der untersuchten Verfahren des aktuellen Standes der Technik.

3.3 Präzisierte Problemstellung

Die Untersuchungen dieses Kapitels haben die Möglichkeiten und Mängel existierender Ansätze für die *Requirements-Engineering*-Phase der Systemfamilienentwicklung aufgezeigt. Dabei lassen sich zwei unterschiedliche Problembereiche abgrenzen.

Auf der einen Seite steht das Vorgehen, nach dem eine Systemfamilie und deren Applikationen entwickelt werden. Hierbei hat sich eine Einteilung in für die Domäne relevante Aufgaben und für die jeweiligen Applikationen relevante Aufgaben herausgebildet. Diese Teile sind wieder in Analyse, Entwurf und Implementierung gegliedert. Thema dieser Arbeit ist die Analysephase, welche in die klassischen vier Aktivitäten Anforderungserhebung, Anforderungsmodellierung, Anforderungsüberprüfung und Anforderungsmanagement aufgeteilt wird.

Auf der anderen Seite steht das Anforderungsmodell, wie im *Six-Pack*-Modell aus Abbildung 3.18 gezeigt. Es steht zwischen der Domänenanalyse und der Applikationsanalyse und muss alle Informationen aufnehmen, die während der *Requirements-Engineering*-Phase entstehen. Dieses Anforderungsmodell muss in der Lage sein, sowohl formale als auch informale Daten zu verwalten und diese mit den Daten weiterer Entwicklungsschritte in Beziehung zu setzen.

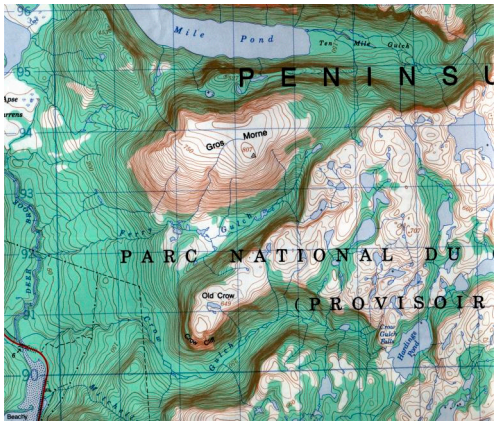
Ein erfolgreiches *Requirement Engineering* für Systemfamilien erfordert einen Prozess, der auf die herkömmlichen Methoden und Techniken des *Requirement Engineering* zurückgreift, um ein systemfamilienbasiertes Datenmodell mit den notwendigen Informationen zu füllen. Das Datenmodell muss eine Beschreibung der Systemfamilie enthalten, wobei sich derzeit das Merkmalmodell als das Mittel der Wahl herausgestellt hat. Basierend auf diesem Modell müssen dann Mitglieder der Familie möglichst fehlerfrei abgeleitet werden können. Diese Anforderungen führen zu den folgenden Teilproblemen, die es zu lösen gilt:

- Merkmale sind, wie gezeigt, von einem Kunden einfach und schnell zu verstehen. Damit bildet die Merkmalmodellierung eine Brücke zwischen den Kunden und Entwicklern einer Systemfamilie. Merkmalmodelle eignen sich daher zur Beschreibung einer Systemfamilie. Die für die Systemfamilienentwicklung wichtige Ableitung eines Familienmitgliedes auf Basis eines Merkmalmodells kann jedoch nur gelingen, wenn Beziehungen zwischen den Modellelementen, wie Merkmalen oder Anforderungen, im Modell selbst hinterlegt und bei der Auswahl berücksichtigt werden können. Daher muss die Merkmalmodellierung nach Kang erweitert werden, um die Ableitung einer mit dem Modell konsistenten Systemvariante zu ermöglichen und die Beziehungen, die sich aus den Wechselwirkungen der Merkmale ergeben, in dem Modell hinterlegen zu können. Derart erweiterte Merkmalmodelle lassen sich darüber hinaus auch automatisiert innerhalb von CASE-Werkzeugen verarbeiten.

- Die Bearbeitung eines derart erweiterten Merkmalmodells muss durch einen entsprechend angepassten *Requirements-Engineering*-Entwicklungsprozess berücksichtigt werden. Daher müssen die in diesem Kapitel beschriebenen, existierenden Ansätze zu einem angepassten *Requirements-Engineering*-Entwicklungsprozess zusammengesetzt werden.
- Das für den Entwicklungsprozess notwendige *Requirements-Engineering*-Datenmodell muss erweiterte Merkmalmodelle im Verbund mit den weiteren Daten der *Requirements-Engineering*-Phase, wie *Stakeholder*, Dokumentstruktur usw. berücksichtigen. Daher muss ein angepasstes *Requirements-Engineering*-Datenmodell auf Basis existierender Modellbestandteile erstellt werden.

Die genannten Probleme werden im nächsten Kapitel aufgegriffen und durch ein neues Datenmodell mit einem erweiterten Merkmalmodell und einer an die Bedürfnisse der Systemfamilienentwicklung angepassten Entwicklungsmethode für die *Requirements-Engineering*-Phase gelöst.

Kapitel 4



„ Gros Morne“ Nationalpark, Kanada

Eigener Lösungsansatz – FORE

Family Oriented Requirements Engineering (FORE) baut auf den im letzten Kapitel vorgestellten Ansätzen auf, integriert sie zu einer durchgängigen Methode und erweitert die für die Entwicklung von Systemfamilien essentielle Merkmalmodellierung. Es handelt sich um eine zweigeteilte Lösung für die *Requirements-Engineering*-Phase der Systemfamilienentwicklung, die sowohl das methodische Vorgehen als auch ein Modell der Entwicklungsdaten enthält.

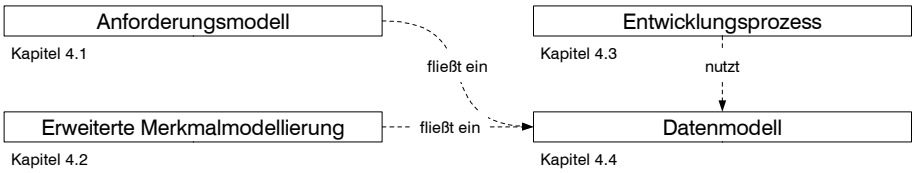


Abbildung 4.1: Übersicht dieses Kapitels und der vier Anteile des Lösungsansatzes

Die folgenden Kapitel teilen sich auf, wie in Abbildung 4.1 gezeigt. Zunächst wird das von dieser Arbeit an Systemfamilien angepasste Anforderungsmodell mit internen Beziehungen vorgestellt. Damit verknüpft, wird die erweiterte Merkmalmodellierung als Basis von FORE vorgestellt. In dieser Arbeit stellt die erweiterte Merkmalmodellierung mit den nun möglichen Konsistenzprüfungen eine Neuerung im Bereich der Systemfamilienentwicklung dar. Darauf folgend werden die einzelnen Entwicklungsschritte des FORE-Prozesses erläutert, die für die Nutzung von FORE notwendig sind. Der FORE-Entwicklungsprozess wurde im Rahmen dieser Arbeit auf Basis bestehender Prozesse integriert. Das Datenmodell, welches die im Rahmen der *Requirements-Engineering*-Phase erarbeiteten Informationen adäquat aufnehmen kann und in Ergänzung zum Entwicklungsprozess erarbeitet wurde, schließt dieses Kapitel ab.

4.1 Anforderungsmodell

FORE baut auf den in [KoSo 1998] und [Robe 1999] vorgestellten und in Kapitel 3 beschrieben Modellen für Anforderungen, der Einteilung beteiligter Personen nach Rollen und den Vorschlägen für Strukturen von Spezifikationsdokumenten auf. FORE integriert diese Ansätze in ein Anforderungsmodell, definiert Beziehungen zwischen den einzelnen Modellen und erweitert die Modelle um Anteile, die für die Systemfamilienentwicklung notwendig sind. In diesem Kapitel ist das gesamte Modell mit den Beziehungen und den Erweiterungen beschrieben.

Wie in Abbildung 4.2 dargestellt, besteht das Anforderungsmodell aus strukturiert angeordneten Anforderungen, einem Modell, der an dem Projekt beteiligten Personen und der Dokumentstruktur des zu erstellenden Spezifikationsdokumentes.

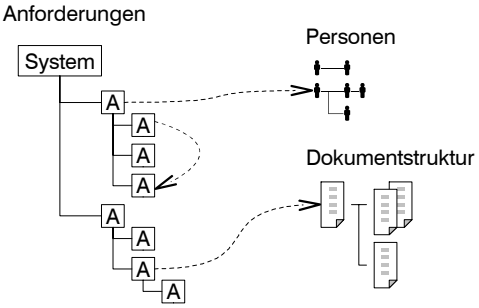


Abbildung 4.2: FORE-Anforderungsmodell

Die Anforderungen im linken Teil der Grafik werden hierarchisch angeordnet, wobei Anforderungen höherer Hierarchiestufen immer allgemeiner werden. Auf dem Weg zu den Blättern des Baumes werden die Anforderungen durch die Hierarchiebeziehung immer weiter konkretisiert, wodurch Verfeinerungen oder Abstraktionen modelliert werden. Die Hierarchiebeziehungen zwischen Anforderungen reichen jedoch für die Systemfamilienentwicklung nicht aus. Im Folgenden werden weitere notwendige Beziehungen vorgestellt und erläutert.

Anforderungen können über Parameter voneinander abhängen. Diese Parameter kennzeichnen Größen, die durch Anforderungen beschrieben werden und quantifizierbar sind. Beispielsweise hängt die maximale Aufzeichnungsdauer eines Videosystems von der Festplattengröße, dem gewählten Komprimierungsalgorithmus und den aufzuzeichnenden Sendungen ab. Da die Art der Sendungen im Voraus nicht bekannt ist, müssen Mittelwerte genutzt werden. Diese Beziehungen werden zunächst natürlichsprachlich beschrieben, können aber auch schon durch eine mathematische Gleichung präzisiert werden. Abhängigkeiten mehrerer Parameter untereinander können angegeben werden, wobei alle beteiligten Anforderungen referenziert werden müssen. Im obigen Beispiel wären dies die Festplattengröße, der Komprimierungsalgorithmus und die Art der Sendungen.

Anforderungen können auch vom Vorhandensein bestimmter Hard- oder Softwarebausteine abhängen, womit eine Anforderung nur zusammen mit einer oder mehreren Anforderungen in einer Applikation realisiert werden kann. Will ein Nutzer des Videosystems, dass laufende Sendungen kurzfristig unterbrochen werden können, so sind zwei DVB-Karten erforderlich. Sobald die Pause-Taste betätigt wird, puffert das Videosystem die aktuelle Sendung auf der Festplatte. Soll die Sendung nach der Unterbrechung fortgesetzt werden, sieht der Nutzer den gepufferten Datenstrom über die eine DVB-Karte, während die andere Karte die eventuell noch laufende Sendung weiter aufnehmen muss. Somit entsteht ein Ringpuffer dessen Größe der Pausenzeit entspricht. Damit erfordert die Unterbrechung einer Sendung das Vorhandensein zweier DVB-Karten. Im Gegensatz dazu können sich zwei Anforderungen auch ausschließen, was als Widerspruch in dem Modell hinterlegt wird. In der Folge und in Zusammenarbeit mit dem Auftraggeber müssen dann alle Widersprüche aufgelöst werden. Die korrekte Auflösung eines Widerspruchs ist auch dann gegeben, wenn der Widerspruch bestehen bleibt, im Modell jedoch gefordert wird, dass jeweils nur eine der beiden Anforderungen in einer Applikation realisiert werden kann. Im Rahmen der Systemfamilienentwicklung werden die beschriebenen Abhängigkeiten zwischen Anforderungen als Variationspunkte behandelt. Es entstehen Varianten, welche die jeweiligen Abhängigkeiten berücksichtigen.

Für jede Anforderung muss eine Änderungshistorie existieren, um jederzeit einen alten Stand der Anforderungen zu Wiederherstellungs- oder Analysezwecken restaurieren zu können. Darüber hinaus ist diese Versionierung auch für das so genannte *Baselining* notwendig. Hierbei wird ein bestimmter, zu einer Zeit von allen Seiten akzeptierter und konsistenter Stand des Anforderungsmodells festgehalten. Basierend auf diesem Stand kann eine Entwicklung weiterlaufen und auf einem definierten Fundament ein System erstellen. Änderungen an den Anforderungen finden dann erst in einer späteren Systemversion Berücksichtigung.

Nicht-funktionale Anforderungen nehmen eine besondere Rolle ein, da sie sich nicht immer direkt einem Software- oder Hardwaremodul zuordnen lassen. Vielmehr beeinflussen sie andere, funktionale Anforderungen, indem sie einschränkend bzw. erweiternd auf sie wirken oder die bereits erwähnten Parameterzusammenhänge beeinflussen, die Parameter sozusagen korrigieren. Das Videosystem wird über Kommandos gesteuert, die es über ein Netzwerk erhält. Dadurch können Interoperabilitätsanforderungen berücksichtigt werden, das System lässt sich leicht in andere Umgebungen integrieren. In einem völlig anderen Anforderungszweig soll das Videosystem die elektronisch über Satellit empfangenen Programminformationen durch die Abfrage von Internet Filmdatenbanken ergänzen, womit auf das Videosystem von jedem Rechner des Internets zugegriffen werden kann. Die Sicherheitsanforderungen für Netzwerke erfordern eine erhöhte Sicherheit gegenüber externen Zugriffen auf das Videosystem. Zum einen führen die nicht-funktionalen Interoperabilitätsanforderungen zu der Steuerung des Videosystems über ein Netzwerk, zum anderen werden die An-

forderungen an eine netzbasierte Steuerung durch die ebenfalls nicht-funktionalen Sicherheitsanforderungen beeinflusst.

Sollen bereits existierende Applikationen einer bestimmten Domäne durch eine Systemfamilie ersetzt werden, kann durch *Refactoring* [Fowl 1999] soviel Entwicklungsleistung wie möglich aus den existierenden Applikationen übernommen werden. Die Analyse der existierenden Applikationen führt zu diversen Anforderungen, wobei für jede Anforderung angegeben wird, welcher Applikation die Anforderung entstammt – die Anforderung hat damit eine Beziehung zu einer späteren Variante der Systemfamilie. Das Ziel der zu entwickelnden Systemfamilienarchitektur ist, alle vorher existenten Applikationen mit gleichem Funktionsumfang als Varianten der Familie wieder ableiten zu können und darüber hinaus viele weitere Applikationen, mit nunmehr geringem Aufwand, ableiten zu können. Die Ableitung von Varianten basiert auf der im nächsten Kapitel erläuterten und im Rahmen von FORE erweiterten Merkmalmodellierung. Durch die Zuordnung der Anforderungen und Merkmale zu den Varianten, lassen sich Zusammenhänge einzelner Elemente im gesamten Modell leicht verfolgen, was als *Traceability* bezeichnet wird.

Anforderung Nr.: *Eindeutige Nummer*	**Dokumentzuordnung:** *Einordnung in das FORE-Spezifikationsdokument (Kapitel)*	**Ereignis/Use Case Nr.**: *Zuordnung*
Komponente: Welcher Komponente ist diese Anforderung zugeordnet?	**Variante:** Welcher Variante ist diese Anforderung zugeordnet?	
Beschreibung:	*Ein Satz, der die Anforderung beschreibt.*	
Bedeutung:	*Warum ist diese Anforderung wichtig? Warum soll sie aufgenommen werden?*	
Ursprung:	*Woher stammt die Anforderung? Wer hat die Anforderung gestellt?*	
Überprüfung:	*Wie kann die Anforderung quantifiziert werden, um sie später zu prüfen?*	
Kundenzufriedenheit: *Wie stark ist der Kunde an dieser Anforderung interessiert?*	**Folgen von Unzufriedenheit:** *Was wäre, wenn die Anforderung nicht realisiert wird?*	
Abhängigkeiten: *Andere beeinflusste Anforderungen.*	**Konflikte:** *Widersprüchliche Anforderungen.*	
Weitere Materialien:	*Hinweise auf weitere Informationen zu dieser Anforderung.*	
Historie:	*Jeder, der etwas an der Anforderung verändert, muss sich hier eintragen.*	

Abbildung 4.3: FORE-Anforderungskarte

Für eine effiziente und einfache Erhebung von Anforderungen im Systemfamilienumfeld stellt FORE eine Anforderungskarte bereit, welche die *Volere* [JaSu 2000] Anforderungskarte aus Kapitel 3.1.5 erweitert. In dieser Karte werden alle Informationen eingetragen, die für eine neu aufzunehmende Anforderung notwendig sind. Die handschriftliche Nutzung ist möglich, allerdings erfordert eine automatisierte Verarbeitung der Daten das aufwändige Einpflegen der Kartendaten in eine Da-

tenbank, wodurch die Nutzung eines entsprechenden Werkzeugs dringend empfohlen wird. Im Kapitel „Prototyp“ wird ein solches Werkzeug vorgestellt.

In Abbildung 4.3 ist eine FORE-Anforderungskarte enthalten. Die grau unterlegten Anteile stellen die von FORE veränderten oder hinzugefügten Anteile dar. Die Felder „Variante“ und „Komponente“ wurden neu hinzugefügt. In den Feldern „Abhängigkeiten“ und „Konflikte“ werden die in diesem Kapitel vorgestellten Beziehungen genutzt, um eine konsistentes Modell und eine automatisierte Verarbeitung der Einträge zu ermöglichen. Diese Änderungen sind erforderlich, um die Variabilitäten und Gemeinsamkeiten der Systemfamilie zu berücksichtigen.

Die Anforderungsnummer wird automatisch vergeben, um Duplikate auszuschließen. Die Dokumentzuordnung bezieht eine Anforderung auf eine bestimmte Stelle im FORE-Spezifikationsdokument, um eine automatische Erzeugung dieses Dokumentes zu ermöglichen. Das FORE-Spezifikationsdokument wird in Kapitel 4.1.2 näher erläutert. Durch die Dokumentzuordnung wird gleichzeitig die Hierarchiebeziehung des Anforderungsmodells realisiert. Funktionale Anforderungen werden einem UML *Use-case* zugeordnet, was in der objektorientierten Softwareentwicklung ein akzeptiertes und etabliertes Mittel der Kommunikation zwischen Auftraggeber und Auftragnehmer ist.

Durch einen kurzen, prägnanten Satz wird eine Anforderung beschrieben. Eine Erläuterung, quasi eine Rechtfertigung, wird als Hintergrundinformation eingetragen, um die Anforderungen auch nach längerer Zeit richtig verstehen zu können. Im Feld „Ursprung“ wird vermerkt, ob eine Anforderung als direkte Folge anderer Anforderungen entstanden ist oder ob eine Anforderung neu ist und zunächst für sich steht. In jedem Fall gibt es zu jeder Anforderung einen Autor, was in FORE als Beziehung der Anforderung in das Personenmodell realisiert und im folgenden Kapitel 4.1.1 beschrieben wird. Jede Anforderung verweist auf eine mögliche Überprüfung. Im einfachsten Fall sind Anforderungen quantifizierbar, wodurch sich eine Prüfung konkret auf diese Größen beziehen kann. Ansonsten werden hier in Prosa beschriebene Prüfszenarien genutzt, die sich später als Testcode niederschlagen. Die Kundenzufriedenheit für den Fall, dass die Anforderung umgesetzt wird, wird durch Ziffern zwischen 1 und 5 bewertet, wie in Kapitel 3.1.5 beschrieben. Die eventuellen Folgen einer Unzufriedenheit unterstreichen diese Bewertung. Auf weitere Materialien wird in den Literaturstellen hingewiesen. Diese Literaturstellen befinden sich in einem Literaturverzeichnis, das ebenfalls Teil des in Kapitel 4.4 beschriebenen FORE-Datenmodells ist. Schließlich werden alle Änderung an Anforderungen mit Angabe von Zeit und Autor in der Historie verwaltet.

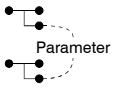
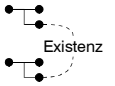
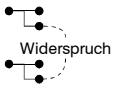
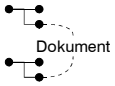
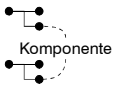
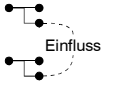
Komponenten, die aus den bereits existierenden Applikationen stammen oder von Drittanbietern eingekauft werden, müssen in der Beschreibung einer Anforderungen festgehalten werden, falls die Anforderung in der betreffenden Komponente realisiert ist. In FORE kann eine Anforderung einer Komponente zugeordnet werden, indem sie die Beschreibung der Komponente referenziert. Für jede Komponente sind dabei folgende Anteile in die Dokumentation aufzunehmen, die eigenen Erfahrungen bei der Komponentenentwicklung entstammen:

- **Art der Komponente** – Die genutzte Technologie und Architektur der Komponente werden festgehalten und auch der Typ der Komponente, ob mit/ohne GUI, als *Control*, als Teil eines *Frameworks* oder als Bibliothek, wird notiert. Speziell technische Besonderheiten sind von großer Bedeutung und müssen dokumentiert werden.
- **Sinn und Zweck der Komponente** – Die Hintergründe und Vorteile der Komponente werden kurz beschrieben.
- **Schnittstellenbeschreibung** – Alle Schnittstellen müssen detailliert beschrieben werden, wobei jeweils kleinere Beispiele die effiziente Nutzung erheblich erleichtern.
- **Nutzungsbeispiel** – Ein schnelles Einarbeiten in die Nutzung der Komponente wird durch ein Nutzungsbeispiel erreicht. Insbesondere die Integration der Komponente können anhand eines Beispiels am besten erläutert werden.
- **Parametrierung** – Für die Nutzung von Komponenten ist es wichtig zu wissen, wie Einstellungen an den Komponenten vorgenommen werden können. Diese Einstellmöglichkeiten stellen gleichzeitig die Variabilität dar, die im Rahmen einer Systemfamilienentwicklung, wie bereits gezeigt, von zentraler Bedeutung ist.
- **Integration** – Die Nutzung von Komponenten unterschiedlicher Technologien erfordert mitunter die Untersuchung von *Wrappern* zur Integration in das eigene System. Hierbei geht es um den Aufwand für die Entwicklung des *Wrappers*, wobei diese Kostenabschätzung auch zur Neuentwicklung der Komponenten führen kann.
- **Testfälle / Testergebnisse** – Sollen Komponenten von Drittanbietern integriert werden, stellen deren Testergebnisse gute Daten für eine erste Bewertung der Komponenten dar. Die Testfälle selbst sind aber von größerer Bedeutung, da sie im Rahmen von Integrationstests für das gesamte System genutzt werden.
- **Leistungsanalyse** – Neben eventuell vorhandenen Testergebnissen liefern eigene Leistungsanalysen objektive Ergebnisse für die Bewertung einer Komponente. Diese Leistungsanalysen werden für jeden Anwendungsfall erneut durchgeführt und können jeweils angepasst oder erweitert werden, so dass nach mehrfacher Nutzung und Leistungsanalyse einer Komponente eine gute Bewertung zur Verfügung steht.
- **Source Code** – Bei Eigenentwicklungen steht der Quellcode zur Verfügung, bei eingekauften Komponenten kommt es auf die Vertragsbedingungen und die avisierte Nutzung der Komponenten an. Auch die geforderte Sicherheit und die daraus resultierenden Haftungsbedingungen entscheiden darüber, ob sich die Kosten für den Quellcode einer Komponente tragen. Eine Steuersoftware für Linienflugzeuge stellt hohe Anforderung an die Betriebssicherheit des Systems, was die Kosten für den Quellcode der Komponenten und die daraus entstehende Transparenz des Systems rechtfertigt.

- **Support** – Bei Integration von Fremdkomponenten ist ein funktionierender Support für die Komponenten wichtig, da darauf das eigene System aufgebaut und Probleme mit Komponenten unkompliziert gelöst werden müssen, um keine Folgeprobleme zu erzeugen.

Jede untersuchte und in die Familie zu integrierende Applikation, sowie alle zukünftig avisierten Varianten der Systemfamilie werden den Anforderungen zugeordnet, indem für jede Anforderung verzeichnet wird, welcher Variante sie entstammt. Es wird jeweils nur der Name der Variante genutzt. Zwischen den Varianten gibt es in diesem Entwicklungsstadium noch keine Beziehungen.

Abhängigkeiten und Konflikte werden in FORE durch Beziehungen der Anforderungen untereinander beschrieben. Wie in diesem Kapitel erläutert, unterstützt FORE eine Reihe von Beziehungen, die in Tabelle 4.1 zusammengefasst sind.

Name	Art	Beschreibung
Hierarchie	 Anforderung – Unteranforderung	Verfeinerung / Abstraktion von Anforderungen.
Parameter	 Anforderung – Anforderung	Quantifizierbare Größen beeinflussen einander.
Existenz	 Anforderung – Anforderung	Eine Anforderung erfordert die Existenz einer anderen Anforderung.
Widerspruch	 Anforderung – Anforderung	Eine Anforderung schließt eine andere Anforderung aus. Es entsteht ein Widerspruch, der aufgelöst werden muss.
Historie	 Anforderungsbaum – Anforderungsbaum	Durch Änderung oder Verbesserung entstehen mehrere Versionen einer Anforderung oder eines ganzen Anforderungsunterbaumes.
Autor	 Anforderung – Person	Bei der Erstellung oder Veränderung einer Anforderung muss der Autor über eine Referenz in dem Personenmodell mit angegeben werden.
Dokument	 Anforderung – Dokumentelement	Anforderungen können einer bestimmten Position in der Struktur des Spezifikationsdokumentes über eine Referenz zu einem Element des Dokumentmodells zugeordnet werden.
Komponente	 Anforderung – Dokumentelement	Anforderungen können bereits in Komponenten realisiert sein. Die Anforderung referenziert dann die Beschreibung der Komponente innerhalb der Dokumentation.
Einfluss	 Anforderung – Anforderung	Nicht-funktionale Anforderungen können funktionale Anforderungen erweitern, beschränken oder beeinflussen.

Name	Art	Beschreibung
Variante	 Anforderung – Variante / Applikation	Entstammt eine Anforderung der Analyse einer bereits existierenden Applikation aus der Domäne der zu erstellenden Systemfamilie, wird diese Zuordnung für jede Anforderung notiert.
Modell	 Anforderung – UML-Modell(e)	Eine Anforderung kann durch eines der UML-Modelle näher beschrieben oder erläutert werden. Von einer Anforderung können daher Beziehungen zu ganzen Modellen oder Modellteilen der Entwurfsmodelle bestehen.

Tabelle 4.1:Beziehungen im FORE Anforderungsmodell

4.1.1 Personenmodell

Die Größe eines Projektes und die räumliche Verteilung der *Stakeholder* erschweren den Kontakt einzelner Personen. Anforderungen und Merkmale werden immer mit einem Hinweis auf den Autor, als Ansprechpartner bei Unklarheiten und Nachfragen, erhoben. Für Projektleiter sind Firmenzugehörigkeit und hierarchische Verhältnisse von Nutzen, wenn es um die Planung und Bewertung von personellem Ressourcenbedarf geht. Bisherige Systemfamilienansätze enthalten jedoch lediglich die Rollen der an einem Projekt beteiligten Personen. In FORE werden die Rollen und die Personen in einem Modell zusammengefasst. Die in FORE nutzbaren Beziehungen von Anforderungen und Merkmalen zu Personen weisen auf das Personenmodell und bieten damit dem Nutzer auch Informationen über das Umfeld der Person an.

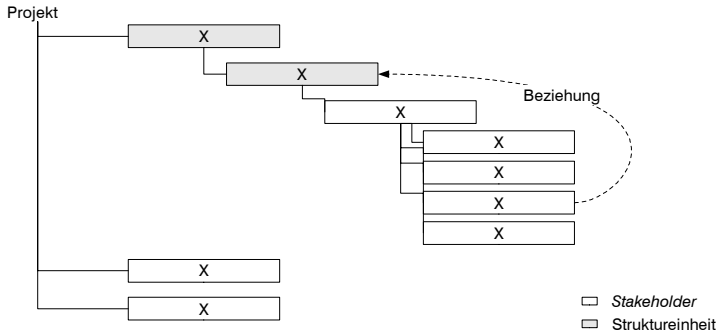


Abbildung 4.4: Personenmodell

FORE ergänzt das Anforderungsmodell durch ein Personenmodell, dargestellt in Abbildung 4.4, in dem alle *Stakeholder* enthalten sind, zusammen mit den jeweiligen Beziehungen der Personen untereinander und den jeweiligen Rollen der Personen im Projekt. Die Hierarchie des Personenmodells entspricht der Hierarchie in einem Unternehmen, während Beziehungen über Unternehmensgrenzen hinweg die projektspezifischen Konstellationen der Stakeholder widerspiegeln. Das Modell besteht aus zwei Elementtypen. Grau unterlegt sind die strukturellen Einheiten, denen die jeweiligen Personen zugeordnet werden. Hinter jedem Personeneintrag stehen die personenspezifischen

Daten, die in einer „Visitenkarte“ abgelegt werden. Beziehungen zwischen beliebigen Elementen werden als gestrichelter Pfeil mit einem die Beziehung beschreibenden Text dargestellt.

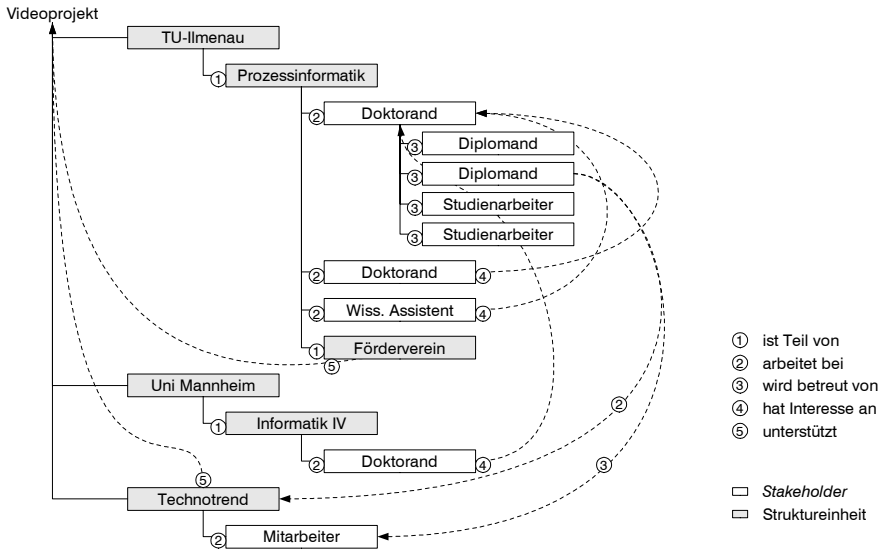


Abbildung 4.5: Personenmodell des Videoprojektes (aus Datenschutzgründen ohne Namen)

In Abbildung 4.5 ist als Beispiel das Personenmodell des Videoprojektes dargestellt. Aus Datenschutzgründen sind in der obigen Abbildung nur die Rollen der Stakeholder angegeben. Wie oben beschrieben, enthalten die *Stakeholder*-Elemente im Modell alle personenbezogenen Daten. Die Beziehungen sind in der Abbildung aus Gründen der Übersichtlichkeit nummeriert und gesondert aufgelistet.

4.1.2 Dokumentmodell

Das Dokumentmodell enthält die Struktur des Spezifikationsdokumentes, welches ein wichtiges Ergebnis der *Requirements-Engineering-Phase* darstellt. Die Dokumentstruktur spiegelt die vom Unternehmen gewählte und auf den in Kapitel 3.1.5 beschrieben Standards basierende Struktur des Spezifikationsdokumentes wider. Um die Dokumentstruktur an spezielle Verhältnisse in einem Unternehmen anpassen zu können, bietet FORE auch eine generische Dokumentstruktur an, die in die gewünschte Struktur überführt werden muss.

Sind keine Anforderungen an eine unternehmensspezifische Struktur vorhanden, kann die von FORE bereits vordefinierte Struktur genutzt werden. Für die Systemfamilienentwicklung wird in FORE eine von dem „Volere“-Schema abgeleitete Struktur genutzt, die auch die im nächsten Kapitel beschriebenen Merkmale aufnehmen kann. In Abbildung 4.31 ist das Dokumentmodell dargestellt, welches den Anforderungen des *DocBook*-Standards [Wals 2003] entspricht und damit durch

diesen beschrieben werden kann, der dann eine flexible Umsetzung des Dokumentes in viele Formate erlaubt.

Die Erweiterungen der Dokumentstruktur sind fett eingetragen. Es handelt sich dabei um neue Teile einer Spezifikation, die den Besonderheiten der Systemfamilienentwicklung Rechnung tragen, wozu die Dokumentation der Variabilitäten einer Familie in Form von Merkmalen und die detaillierte Betrachtung genutzter Systemkomponenten zählen.

Im FORE-Spezifikationsdokument werden *Stakeholder* und Nutzer zusammen mit ihren Beziehungen untereinander durch das FORE-Personenmodell modelliert. In einem neuen Unterkapitel „Merkmalmodell“ wird zunächst das Merkmalmodell generell beschrieben, um den Leser in diese Art der Beschreibung einer Systemfamilie einzuführen. Danach wird jedes Merkmal einzeln beschrieben. In dem zweiten Unterpunkt werden die Varianten des Systems dokumentiert, die eine kundenspezifische Auswahl von Merkmalen darstellen. Jede Variante wird hier aufgelistet, um bei späteren Analysen jederzeit eine kurze Beschreibung der Variante, die an der Variante beteiligten Merkmale und die damit verbundenen Anforderungen überblicken zu können.

Inhaltsverzeichnis

Abbildung 4.6: Struktur des FORE-Spezifikationsdokumentes

Schließlich ist im Systemfamilienumfeld die Dokumentation der genutzten Fremdkomponenten von größerer Bedeutung, da auf der Referenzarchitektur der Systemfamilie viele Applikationen basieren. Daher wurde der Dokumentation der Komponenten in FORE ein eigenes Kapitel gewidmet, um die Komponenten selbst, deren Schnittstellen und deren Prüfmöglichkeiten festzuhalten, um bei eventuellen Fehlern im späteren System die betreffende Komponente schnell identifizieren zu kön-

nen. Eine detailliertere Beschreibung der Inhalte einer Komponentenbeschreibung ist bereits in Kapitel 4.1 enthalten.

Das Spezifikationsdokument stellt einen wichtigen Meilenstein in der Entwicklung einer Systemfamilie dar, da sie als Vertragsgrundlage zwischen Auftraggeber und Auftragnehmer dient. Ihre Inhalte sind rechtsverbindlich und können von beiden Seiten bei Nichteinhaltung eingeklagt werden.

4.2 Erweiterte Merkmalmodellierung

Den Kernanteil des hier dargestellten Lösungsansatzes stellt die Erweiterung der Merkmalmodellierung nach Kang [Kang 1990] dar. Merkmalmodelle beschreiben die Variabilitäten einer Systemfamilie, haben jedoch Schwächen bei der Definition und Integration in den Softwareentwicklungsprozess für Systemfamilien, wie bereits in Kapitel 3 gezeigt. Beziehungen von Elementen des Anforderungsmodells und des Entwurfsmodells zu Elementen des Merkmalmodells sind bislang nur vage definiert und formal nur unzulänglich beschrieben. Für eine komplette Integration der Merkmalmodellierung in den Softwareentwicklungsprozess müssen diese Beziehungen eindeutig und formal definiert sein, um eine automatisierte Be- und Verarbeitung der Daten zu ermöglichen. Insbesondere die Prüfung der Konsistenz von Modellen kann durch entsprechend definierte Abhängigkeiten im Modell ebenfalls automatisiert geprüft werden. Damit stellt die erweiterte Merkmalmodellierung von FORE eine Neuerung im Bereich der *Requirements-Engineering*-Phase der Systemfamilienentwicklung dar.

Die Merkmalmodellierung steht als weitere Abstraktionsschicht zwischen dem Anforderungsmodell und dem Entwurf des Softwaresystems, wie in Abbildung 4.7 dargestellt. Die mit Zahlen gekennzeichneten Beziehungen werden in folgender Auflistung kurz erläutert:

- Mit ① bezeichnet sind die Beziehungen innerhalb des FORE-Anforderungsmodells, welche bereits im Kapitel 4.1 beschrieben wurden.
- Mit ② bezeichnet sind die Beziehungen des FORE-Anforderungsmodells zu dem Entwurf, wobei FORE davon ausgeht, dass der Entwurf durch Diagramme der UML erstellt wird. Diese Beziehung ist ebenfalls in Kapitel 4.1 beschrieben.
- Auf die Punkte ③, ④ und ⑤ wird in diesem Kapitel Bezug genommen, wobei es sich um die von FORE automatisiert verarbeitbaren Beziehungen handelt, die eine tiefergehende Betrachtung erfordern.

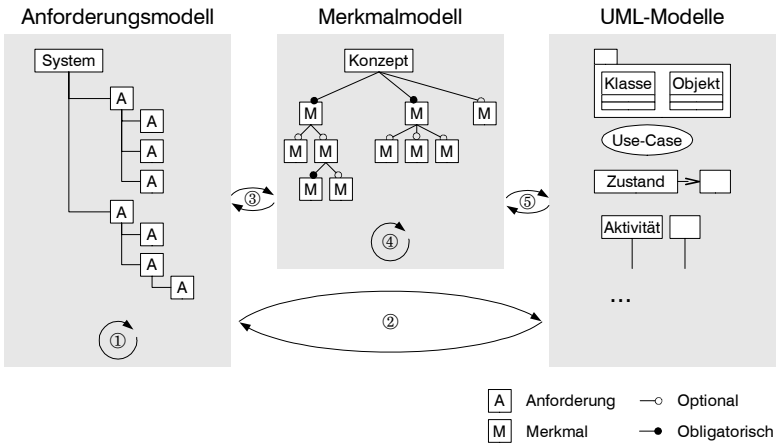


Abbildung 4.7: Merkmale und ihr Beziehungsgeflecht

Das Merkmalmodell, im Mittelteil von Abbildung 4.7, ist einerseits ein Ziel der systemfamilienorientierten Anforderungsmodellierung und stellt andererseits den Ausgangspunkt für die spätere Entwicklung der Systemfamilienarchitektur dar. Dabei spiegelt die Anordnung der Elemente in obigem Bild nicht die Reihenfolge der Bearbeitung wider. Eine detaillierte Darstellung des Entwicklungsprozesses ist in Kapitel 4.3 enthalten.

Die Merkmalmodellierung nach Kang [Kang 1990] beschreibt und modelliert eine Systemfamilie anhand ihrer Merkmale. Die Einteilung in einen Kern und variable Anteile geschieht dabei über die Markierung der Merkmale als optional oder obligatorisch. Darüber hinaus sind rudimentäre Querbezüge zwischen den Merkmalen möglich, die den Ausschluss oder den Einschluss weiterer Merkmale angeben. FODA ist ein Ausgangspunkt für die Modellierung einer Systemfamilie, hat aber Schwächen bei der Eindeutigkeit der Notation und bei der Definition der Begriffe Merkmal und Merkmalmodell und der Art der Anwendung dieser Begriffe. Für die weiteren Ausführungen in dieser Arbeit wird zunächst der Begriff des Merkmals im Sinne von FORE definiert und erläutert. Darauf folgend wird eine durch FORE definierte Merkmalkarte zur Erhebung von Merkmalen sowie eine im Rahmen dieser Arbeit neu entstandene und in [Rieb 2002] beschriebene Notation für Merkmalmodelle vorgestellt.

FORE Definition: Merkmal (*Feature*)

> Anforderungen an das System werden von Merkmalen überblicksartig zusammengefasst, wobei durch Merkmale die Variabilität des Software Systems modelliert wird. Ein Merkmal repräsentiert einen Aspekt, der für einen Kunden wertvoll ist und durch ein einzelnes Wort ausgedrückt wird. In FORE werden Merkmale in die folgenden drei Kategorien eingeteilt:

1. *Funktionale Merkmale* drücken ein Verhalten des Systems oder eine mögliche Interaktion der Benutzer mit dem System aus.

2. *Schnittstellen-Merkmale* drücken die Integration von Standards oder Subsystemen in das Produkt aus.

3. *Parameter-Merkmale* drücken Zahlenwerte, in Zahlenwerten darstellbare Umgebungseigenschaften oder nicht-funktionale Eigenschaften aus. Sie haben jeweils einen voreingestellten Wert.

4. *Strukturierungsmerkmale* dienen lediglich der Strukturierung des zu entwickelnden Modells. Sie können nicht ausgewählt werden und werden im Verlauf der Modellierung genutzt, um eine Reihe von Merkmalen in einem Teilbaum unterhalb des *Strukturierungsmerkmals* durch einen leicht verständlichen Oberbegriff logisch zusammenzufassen.

Mit Hilfe der Merkmale werden alle Eigenschaften eines Systems beschrieben, die aus Sicht eines Kunden interessant sind und die Attraktivität des Produktes steigern. Merkmale sind damit gleichsam die Verkaufsargumente, die aus Werbeprospekten bekannt sind. Auf der anderen Seite werden durch Merkmale auch technologische Eigenschaften des Systems zum Ausdruck gebracht, die für die meisten Kunden nicht von Belang sind. Diese Merkmale werden von Entwicklern benötigt, die auf Basis der Merkmale eine Architektur erarbeiten. Schließlich dienen Merkmale der Unterscheidung von Applikationen einer Systemfamilie.

Neben der obigen Definition der Merkmale bietet FORE eine Merkmalkarte zur Erhebung von Merkmalen an, wie in Abbildung 4.8 dargestellt. Ähnlich den Anforderungskarten können hier alle notwendigen Informationen festgehalten werden. Die handschriftliche Nutzung der Karte ist möglich, viel besser ist die direkte Eingabe am Rechner, wie es im Kapitel „Prototyp“ beschrieben wird.

Die Karte enthält den Namen des Merkmals, der sehr sorgfältig gewählt werden muss, um Missverständnisse zu vermeiden. In Zusammenarbeit mit dem Kunden und unter Berücksichtigung des Glossars wird dieser Begriff erarbeitet. Eine kurze Beschreibung rundet die präzise Definition des Merkmalnamens ab.

Merkmal Nr.: *Eindeutige Nummer*	**Dokumentzuordnung:** *Einordnung in die Struktur des FORE-Spezifikationsdokumentes*		**Merkmaltyp**: *Funktion, Schnittstelle oder Parameter*
			Parametertyp ODER **Schnittstellenbeschreibung**
Name:	*Ein einzelnes Wort, welches das Merkmal beschreibt.*		
Entscheidung:	*Ein Verweis in das Decision Model.*		
Beschreibung:	*Kurze Beschreibung des Merkmals.*		
Anforderungen:	*Welchen Anforderungen ist dieses Merkmal zugeordnet?*		
Motivation:	*Warum sollte ein Kunde dieses Merkmal wählen?*		
Varianten:	*Gibt es Varianten, denen dieses Merkmal entstammt oder denen es zugeordnet werden kann?*		
Quelle:	*Woher stammt das Merkmal? Wer hat das Merkmal erstellt?*		
Prüfung:	*Wie kann das Merkmal später geprüft werden? (→ Integration der Prüfung von Anforderungen)*		
Abhängigkeiten:	*Andere beeinflusste Merkmale.*	**Konflikte:** *Widersprüchliche Merkmale.*	
Weitere Materialien:	*Hinweise auf weitere Informationen zu diesem Merkmal.*		
Historie:	*Jeder, der etwas an dem Merkmal verändert, wird in die Historie eingetragen.*		

Abbildung 4.8: FORE-Merkmalkarte

Die Dokumentzuordnung enthält die Position in dem Spezifikationsdokument, an der das Merkmal eingeordnet wird. FORE verwendet für die Dokumentstruktur ein flexibles Datenmodell, welches beliebige Strukturen zulässt, um auch an individuelle und firmenspezifische Wünsche angepasst zu werden. In Kapitel 4.4, „Datenmodell“, wird diese flexible Struktur näher beschrieben.

Für den Typ eines Merkmals gibt es nach der oben bereits genannten Definition drei Möglichkeiten. Es kann sich um ein funktionales Merkmal handeln. In diesem Fall muss die Beschreibung auch die gewünschte Funktion des Merkmals enthalten. Dabei wird natürlich auf die bereits bei den Anforderungen beschriebenen Funktionen referenziert. Diese Referenzen müssen explizit unter dem Punkt „Anforderungen“ enthalten sein, können im beschreibenden Text jedoch wiederholt werden. Als weitere Möglichkeit kann durch ein Merkmal eine Schnittstelle beschrieben werden. Dabei ist es unerheblich, ob diese Schnittstelle aufgrund einer genutzten Komponente unterstützt werden muss oder weil das Merkmal einen Standard widerspiegelt, den es zu unterstützen gilt. Schließlich kann es sich auch um ein Parametermerkmal handeln, welches für ein Produkt einen konkreten Wert enthält. Dieser Wert ist typisiert und kann Maximal- beziehungsweise Minimalwerte aufweisen, die bei Bedarf mit angegeben werden. FORE unterscheidet zwischen Zahlenwerten und Texten. In der Zeile „Anforderungen“ werden Beziehungen zwischen den Anforderungen und Merkmalen notiert. Diese Beziehungen werden im nächsten Unterkapitel thematisiert.

Eine Variante stellt eine von einem Kunden gewünschte, gültige Auswahl von Merkmalen dar. In der Merkmalkarte werden zunächst nur die Namen der Varianten notiert, denen ein Merkmal zugeordnet sein kann, wobei jedes Merkmal einer oder mehreren Varianten zugeordnet werden kann. Existieren bereits Varianten, die in eine Systemfamilie überführt werden sollen, wird an dieser Stelle natürlich die Variante notiert, durch deren Analyse ein Merkmal entstanden ist.

Mögliche Prüfungen werden für ein Merkmal beschrieben, müssen jedoch im Hinblick auf dessen optionale oder obligatorische Einbettung in die Systemfamilie modifiziert werden. Die Überprüfng sollte auch in beliebiger Kombination mit anderen Systemkomponenten sinnvolle und aussagekräftige Ergebnisse liefern. Der Bezug von Anforderungen des Anforderungsmodells zu *Use-cases* der UML liefert mögliche Problem- und Ausnahmefälle der Ausführung. Über die Zuordnung der Anforderungen zu den Merkmalen sind diese Informationen für einfache Überprüfungen verfügbar. Ein systemfamilienorientiertes Testvorgehen ist jedoch nicht Teil von FORE und wird daher zukünftigen Forschungsvorhaben überlassen.

Alle Merkmale werden in einem Merkmalmodell angeordnet und grafisch aufbereitet, welches nach [Kang 1990] einen sehr simplen Aufbau hat, das jedoch, wie in Kapitel 3.2.2 gezeigt, nicht genau genug definiert ist. Daher wird zunächst der Begriff des Merkmalmodells definiert.

FORE Definition: Merkmalmodell (*Feature Model*)

Ein Merkmalmodell gibt die Merkmale in einer hierarchischen Struktur wieder. Zusätzlich zu den genannten Kategorien der Merkmale gibt es in einem Merkmalmodell auch noch abstrakte Merkmale, die jeweils Konzepte darstellen (z.B. die abstrakte Kraftübertragung, die ein Konzept darstellt, mit den konkreten Möglichkeiten Automatikgetriebe oder Schaltgetriebe, die wiederum wählbare Merkmale sind). Die Wurzel des Baumes ist immer ein Konzept. Die Merkmale des Baumes werden über Beziehungen verbunden, für die folgende Regeln gelten:

1. *Merkmal-Untermerkmal-Beziehungen* werden genutzt, um die Hierarchie abzubilden, wobei die Stellung eines Merkmals in der Hierarchie den Einfluss dieses Merkmals auf die Systemarchitektur widerspiegelt. Der Auswahlprozess wird ebenfalls durch diese Hierarchie beeinflusst, indem ein Kunde die für die Architektur wichtigsten Merkmale zuerst auswählen muss. Die Hierarchiebeziehungen weisen auf obligatorische (englisch: *mandatory*) oder optionale (englisch: *optional*) Merkmale hin. Alle Wege beginnend bei der Wurzel des Baumes, die nur über obligatorische Merkmale gehen, enthalten die Merkmale des Kerns der Systemfamilie, während die optionalen Merkmale den variablen Anteilen zugeordnet werden.

2. Durch die Hierarchie werden semantisch entweder die Notwendigkeit (englisch: *Requires*) eines Merkmals oder die Verfeinerung (englisch: *refinement*) eines Merkmals dargestellt.

3. Die *Verfeinerung* führt zu detaillierteren Untermerkmalen und drückt semantisch die „ist-ein"- oder „ist-Teil-von"-Beziehung aus.

4. Einschränkungen zwischen Merkmalen werden durch die *Gruppierungsbeziehung mit Angabe der Vielfachheit* für Gruppen von Merkmalen mit dem gleichen Elternmerkmal oder durch *Einschluss-* bzw. *Ausschlussbeziehungen* (englisch: *requires* bzw. *excludes*) zwischen beliebigen Merkmalen dargestellt.

5. Beziehungen zwischen Parametermerkmalen können durch *mathematische Ausdrücke* genau definiert werden, welche im Modell hinterlegt werden.

6. *Vorschläge* für die Auswahl weiterer sinnvoller Merkmale werden durch die „Siehe auch" (englisch: *hint*) Beziehung dargestellt und beeinflussen damit den Auswahlprozess des Kunden.

Die Definition des Merkmalmodells erlaubt die einfache Nutzung von Merkmalen zur Abstraktion von Anforderungen im Systemfamilienumfeld und der Modellierung der Varianten einer Familie. Die Gruppierung, die Beziehungen zwischen Parametermerkmalen und die Vorschläge zwischen Merkmalen sind neu und in keiner anderen Definition enthalten. Der Vorteil für den Benutzer ist die Möglichkeit, eine Systemfamilie eindeutig und überprüfbar zu modellieren, was mit den bisherigen Ansätzen nicht möglich war, da die existierenden Definitionen von Merkmalmodellen mehrdeutig sind und existierende Abhängigkeiten zwischen Modellelementen nur textuell angegeben werden und damit nicht überprüfbar sind.

Das Merkmalmodell gibt dem Kunden die Möglichkeit, die Familie anhand der Merkmale zu verstehen und wiederum aus den Merkmalen diejenigen zu wählen, die seine Bedürfnisse am besten zufrieden stellen können. Der Auswahlprozess wird durch die Anordnung der Merkmale im Baum beeinflusst. Je näher ein Merkmal an der Wurzel des Baumes steht, desto früher wird ein Kunde nach diesem Merkmal gefragt. Es ergibt sich somit eine Reihenfolge der Fragen, die sich an den Ebenen des Baumes orientiert. Je früher eine Frage gestellt wird, desto größer ist der Einfluss des entsprechenden Merkmals auf die Architektur des Systems. Damit spiegeln sich die Abhängigkeiten der einzelnen Architekturkomponenten in der Hierarchie des Merkmalmodells wider. Die Fragereihenfolge wird durch den Kunden und dessen Vorkenntnisse beeinflusst. Fragen können schneller abgearbeitet werden, falls der Kunde über Hintergrundwissen verfügt, welches ihm das Verständnis der Merkmale einer Systemfamilie erleichtert.

Wie in Abbildung 4.9 an zwei Beispielen dargestellt, wurde die alte Notation zugunsten einer vereinfachten und eindeutigen neuen Notation in FORE ersetzt, der im Beitrag [Rieb 2002] beschrieben ist. Im obigen Beispiel geht es immer um vier Merkmale, bezeichnet mit A bis D. Das linke obere Diagramm zeigt die alternativen Merkmale B, C und D. Unklar ist jedoch, wie die als obligatorisch gekennzeichneten Merkmale alternativ behandelt werden, da eigentlich nur ein Merkmal ausgewählt werden soll. So auch im zweiten Fall, links unten im Bild. Wie stehen die als obligatorisch oder optional markierten Merkmale B bis D zu der ODER-Verknüpfung? Sind B und D immer in der Auswahl, wäre die logische ODER-Verknüpfung der Merkmale nicht mehr gegeben. Es ist unklar, wie diese Notation zu lesen ist, obwohl sie [Czar 2000] entspricht. Auf der rechten Seite von Abbildung 4.9 stehen die entsprechenden Diagramme in der neuen Notation. Hierbei wurde eine UML-ähnliche Notation für Vielfachheiten eingeführt. Zunächst sind in einer Gruppe von Merkmalen immer alle Merkmale als optional deklariert. In dem Kreisbogen, der die Gruppe umspannt, steht die mögliche Vielfachheit. Dabei kann ein einziger Zahlenwert genutzt werden, wie im Diagramm rechts oben. Aus dieser Gruppe der Merkmale kann genau ein Merkmal gewählt werden. Im zweiten Beispiel, rechts unten, wird ein Startwert und ein Endwert vergeben. Hier kann kein Merkmal gewählt werden oder beliebig viele Merkmale, was durch die Null und den Stern gekennzeichnet wird.

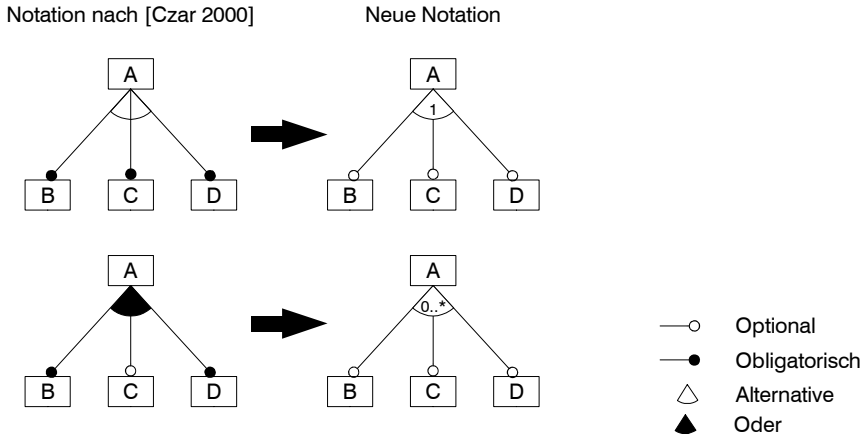


Abbildung 4.9: Neue Merkmalnotation

Die Konsistenz der im Merkmalmodell enthaltenen Informationen ist von großer Bedeutung, da dieses Modell die Basis für die Auswahl einer Variante ist und die Architektur des Systems beeinflusst. Zur Wahrung der Konsistenz müssen die Beziehungen innerhalb des Modells und von dem Modell ausgehend zu anderen Modellelementen klar strukturiert und in ihrer Semantik eindeutig definiert werden. Basierend auf diesen Beziehungen müssen Prüfvorschriften definierbar sein, die Konsistenzbedingungen überprüfen und Verletzungen lokalisieren können. Für effiziente Konsis-

tenzprüfungen müssen die Prüfvorschriften in entsprechende Werkzeuge integriert werden, um eine automatisierte Überprüfung zu realisieren.

Nur aus einem konsistenten Modell kann eine korrekte Variante einer Applikation abgeleitet werden. Die Auswahl einer Menge von Merkmalen stellt dabei den ersten Schritt zu einer neuen Variante und damit einer Applikation dar. Die Auswahl muss jedoch den Anforderungen genügen, die im Rahmen der Anforderungsanalyse aufgestellt wurden. Neben diesen vom Nutzer gewünschten Beschränkungen müssen auch Randbedingungen berücksichtigt werden, die sich aus den technologischen Grenzen ergeben, z.B. CPU-Geschwindigkeit, Betriebssystem oder genutzter Compiler. Diese Randbedingungen werden durch Beziehungen der einzelnen Modellelemente untereinander beschrieben und im folgenden Unterkapitel näher erläutert.

In den folgenden Unterkapiteln werden zunächst die Beziehungen erläutert, die für ein Merkmalmodell zu berücksichtigen sind. Darauf aufbauend werden Konsistenzbedingungen formuliert, die ein Merkmalmodell erfüllen muss. Schließlich wird eine Notation in Form einer neuen Sprache vorgestellt, die sowohl die Formulierung von Beziehungen und Konsistenzbedingungen, als auch deren Überprüfung erlaubt.

4.2.1 Beziehungen von Merkmalmodellen

Zunächst werden die Beziehungen vom Anforderungsmodell zu den Merkmalen des Merkmalmodells betrachtet, um bei Änderungen an beliebigen Stellen des Modells alle beteiligten Modellelemente identifizieren zu können. Inkonsistenzen können vermieden werden, da die Entwickler die betreffenden Elemente des Modells nunmehr erkennen und gegebenenfalls überarbeiten können. Aus der bereits dargestellten Definition der Merkmale wird deutlich, dass es sich bei Merkmalen um eine Abstraktion von Anforderungen handelt. Mehrere, teilweise sehr spezifische oder technische Anforderungen werden durch Merkmale abstrahiert, die aus Kundensicht verständlich sind. Für den Entwickler stecken im Merkmalmodell durch die Beziehungen zu weiteren Modellelementen die Informationen, die notwendig sind, um die Interna der Systemfamilie zu finden und zu verstehen. Beziehungen von Anforderungen zu Merkmalen sind nicht gerichtet, da sie sowohl bei Veränderung von Anforderungen wie auch Merkmalen genutzt werden. Wird eine Anforderung verändert, muss das eventuell betroffene Merkmal vom Entwickler ebenfalls überprüft und angepasst werden. Die Änderung eines Merkmals erfordert die Überprüfung der gegebenenfalls durch das Merkmal abstrahierten Anforderungen. Durch die Beziehungen von Anforderungen zu Merkmalen kann ein Entwickler automatisiert von einem Entwicklungssystem auf anstehende Überprüfungen hingewiesen werden, so dass Nebenwirkungen reduziert werden können. Der Entwickler wird durch die Entwicklung „geführt“, wodurch die Konsistenz des Modells verbessert wird.

Anforderungen werden durch Merkmale abstrahiert. FORE erlaubt es, mehrere Anforderungen mit mehreren Merkmalen zu verbinden. Im linken Teil von Abbildung 4.10 ist diese Beziehung dargestellt. Beispielhaft ist ebenfalls in Abbildung 4.10 ein Teil der Beziehungen zwischen Anforderungen und Merkmalen dargestellt. Auf der linken Seite sind die Anforderungen zu sehen, auf der rechten die Merkmale. Die detaillierte Beschreibung der Anforderungen ist Anhang-C zu entnehmen. Anforderung 1.1 und 1.2 inklusive ihrer Verfeinerungen beziehen sich alle auf die Möglichkeit, das digitale Videosystem mit Hilfe einer Infrarot-Fernbedienung zu steuern. Für eine mögliche Realisierung des Merkmals „Infrarot" ist also die Berücksichtigung dieser Anforderungen unter 1.1 und 1.2 notwendig. Die Verteilung einer Anforderung auf mehrere Merkmale ist möglich, eigene Erfahrungen haben jedoch gezeigt, dass derartige Bezüge vermieden werden sollten, um die Anforderungen bei der späteren Umsetzung der Merkmale nicht über mehrere Komponenten zu verteilen und die Verständlichkeit des Modells zu verbessern.

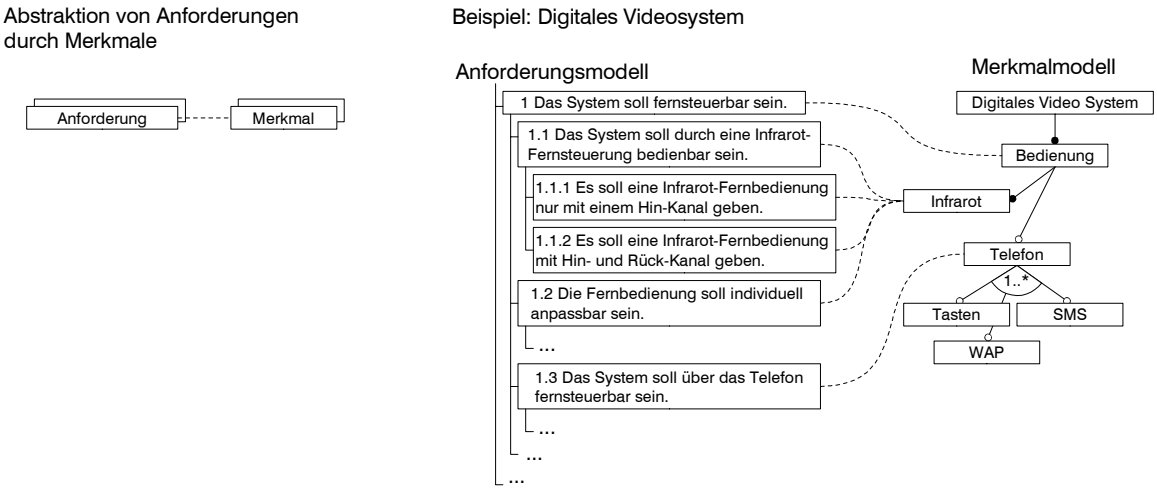


Abbildung 4.10: Beziehung zwischen Anforderungen und Merkmalen

Innerhalb des Anforderungsmodells können nicht-funktionale Anforderungen einen Einfluss auf funktionale Anforderungen haben, indem sie erweiternd oder beschränkend wirken. Werden nicht-funktionale Anforderungen durch Merkmale abstrahiert, so wird die „Einfluss"-Beziehung aus dem Anforderungsmodell in ein Merkmal übernommen. Im linken Teil von Abbildung 4.11 ist die Beziehung dargestellt. Ausgehend von einer nicht-funktionalen Anforderung wird über ein Merkmal eine mögliche Konkretisierung dieser Anforderung durch eine funktionalen Anforderung modelliert. Der Vorteil für den Kunden besteht in der Vereinfachung eines Sachverhaltes durch ein einziges Merkmal.

Im rechten Teil von Abbildung 4.11 hat die nicht-funktionale Anforderung der Sicherheit des Videosystems gegen ungewollte Nutzung durch Kinder einen Einfluss auf die Authentifizierung des Nut-

zers gegenüber dem System. Das Videosystem realisiert die beschriebene Art der Sicherheit über das Merkmal „Zugangskontrolle“, das ein Kunde wählen kann. Nur bei Vorhandensein dieses Merkmals muss auch eine Komponente zur Authentifizierung im System vorhanden sein und bei der Bedienung des Systems berücksichtigt werden.

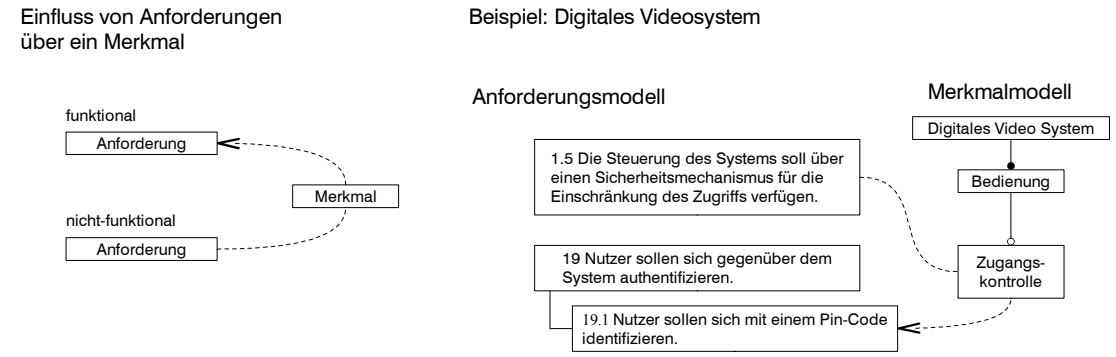


Abbildung 4.11: Einflussbeziehung

Tabelle 4.2 enthält eine Übersicht der Beziehungen zwischen Anforderungsmodell und Merkmalmodell. Die Abstraktion von Anforderungen durch ein Merkmal stellt die häufigste Beziehung dar. Eine Anforderung kann auch mit mehreren Merkmalen in Beziehung stehen, was sich allerdings durch Erfahrungen des digitalen Videoprojektes als ungünstig herausgestellt hat. Die Verbindung einer Anforderung mit mehreren Merkmalen führt zu einer Verteilung der Anforderung auf mehrere Komponenten der Systemfamilie. Bei Änderungen müssen jeweils alle durch die Beziehungen beteiligten System- oder Modellteile mit verändert werden, was sehr aufwändig ist.

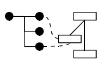
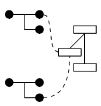
Name	Art	Beschreibung
Abstraktion	 Anforderungen – Merkmal	Das Merkmal stellt einen allgemeinen und den für einen Kunden verständlichen Oberbegriff einer Menge von Anforderungen dar.
Einfluss	 Anforderung – Merkmal – Anforderung	Die Beziehung von nicht-funktionalen Anforderungen zu funktionalen Anforderungen kann über ein Merkmal modelliert werden, welches das Kaufverhalten eines Kunden positiv beeinflusst.

Tabelle 4.2: Beziehungen zwischen Anforderungen und Merkmalen

In FORE gilt die Entwurfsempfehlung, einzelne Anforderungen gar nicht oder nur in Ausnahmefällen mit mehreren Merkmalen in Beziehung zu setzen. Diese Ausnahmefälle müssen dann ausführlich dokumentiert werden, um bei Änderungen am System nicht vergessen zu werden.

Die zweite Beziehung aus Tabelle 4.2 spiegelt den Einfluss einer nicht-funktionalen Anforderung über den Umweg der Merkmale auf funktionale Anforderungen wider. Als Beispiel kann die nicht-

funktionale Forderung nach einer guten Performanz des Systems nur berücksichtigt werden, wenn sie durch funktionale Anforderungen konkretisiert wird. Die entsprechenden Hard- und Softwarekomponenten des digitalen Videosystems müssen so gewählt werden, dass beispielsweise konkrete Zeitvorgaben eingehalten werden. Die nicht-funktionale Anforderung wird nun mit einem Merkmal verbunden, welches von einem Kunden verstanden wird. Das Merkmal enthält alle eventuellen Parameter, welche die Umsetzung durch die Komponenten des Systems erlauben. Die Parameter sind wiederum aus den funktionalen Anforderungen entstanden, welche die Konkretisierung der nichtfunktionalen Anforderung darstellen. Damit hat eine nicht-funktionale Anforderung Einfluss auf funktionale Anforderungen, was als Beziehung über ein Merkmal mit entsprechenden Parametern modelliert wird.

Zwischen den Elementen des Merkmalmodells sind ebenfalls Beziehungen notwendig, um die reibungslose Entwicklung der Systemfamilie und die Konfiguration von Varianten, also Familienmitgliedern, gewährleisten zu können. Im Verlauf der Konfiguration einer Applikation als Variante der Systemfamilie gilt es, aus der Fülle der Merkmale diejenigen auszuwählen, die den funktionalen Interessen und finanziellen Möglichkeiten des Kunden entsprechen. Aus technologischen, wirtschaftlichen oder logischen Gründen sind nicht alle rechnerisch möglichen Varianten eines Merkmalbaumes auch realisierbar. Um das Merkmalmodell einsetzen zu können, müssen die einschränkenden Faktoren in dem Modell hinterlegt werden und über Beziehungen zwischen Merkmalen in FORE modelliert werden. Die Variantenvielfalt wird reduziert, der Entscheidungsprozess des Kunden beeinflusst. Grafisch werden alle Beziehungen in FORE durch eine gestrichelte und beschriftete Line mit oder ohne Pfeil dargestellt. Im Folgenden werden die für ein Merkmalmodell notwendigen Beziehungen anhand des Videosystems dargestellt und erläutert.

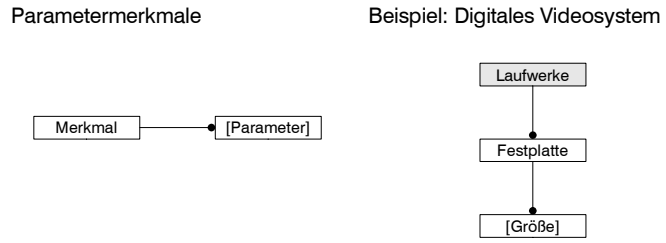


Abbildung 4.12: Parametermerkmal

Eine Möglichkeit der Beeinflussung von Hard- oder Softwarekomponenten ist deren Parametrierung. Parametermerkmale erlauben die zusätzliche Angabe von entsprechenden Parameterwerten. Der Name des Parameters wird in eckige Klammern gesetzt, wodurch ein Merkmal zum Parametermerkmal wird. Ein Parameter ist immer einem Merkmal zugeordnet, wie in Abbildung 4.12 links dargestellt. Darüber hinaus wird im Modell ein voreingesteller Wert für jeden Parameter hinterlegt. Beispielsweise ist die Festplatte, in Abbildung 4.12 rechts dargestellt, ein obligatorisches Merkmal

des Videosystems, welches jedoch einen Parameter für die Größe aufweist. Der Kunde muss sich bei der Konfiguration seines Systems für eine Plattengröße entscheiden und dadurch die Festplatte des Systems parametrieren.

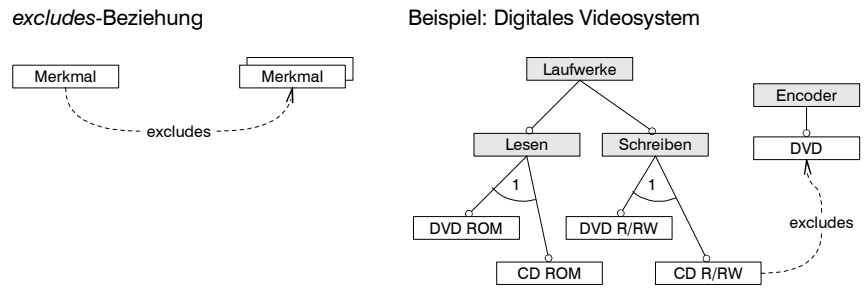


Abbildung 4.13: Merkmalbeziehung – *Excludes*

Die Modellierung des Ausschlusses eines oder mehrerer Merkmale bei Auswahl eines anderen Merkmals durch den Kunden wird mit Hilfe der *Excludes*-Beziehung modelliert. Im linken Teil von Abbildung 4.13 ist die Beziehung zwischen zwei Merkmalen dargestellt. Im rechten Teil von Abbildung 4.13 ist beispielhaft eine *Excludes*-Beziehung zwischen dem Merkmal „CD-R/RW“ und „DVD“ dargestellt. Aufgrund der für das digitale Videosystem vorrätigen Geräte muss sich ein Kunde für DVD- oder CD-Brenner entscheiden. Beide können nicht gleichzeitig eingebaut werden. Entscheidet sich ein Kunde für den CD-Brenner, so wird das Merkmal „DVD“ als *Encoder*-Komponente ausgeschlossen, da selbst kurze Aufzeichnungen nicht auf eine CD passen. Die grau unterlegten Elemente in Abbildung 4.13 sind Strukturmerkmale, die das Modell übersichtlicher gestalten.

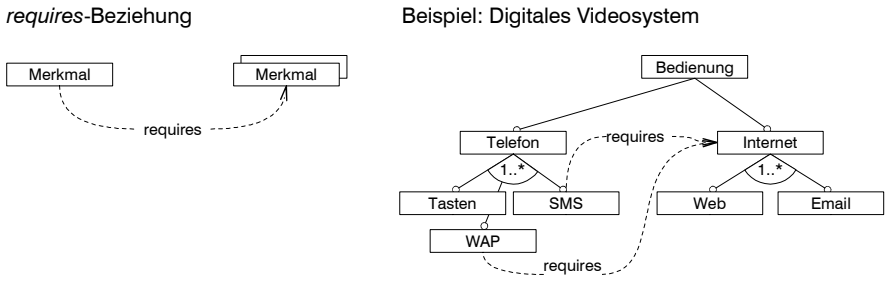


Abbildung 4.14: Merkmalbeziehung – *Requires*

Über die *Requires*-Beziehung werden ein oder mehrere Merkmale modelliert, die bei Auswahl eines Merkmals notwendig sind. Im linken Teil von Abbildung 4.14 sind die beteiligten Merkmale und die *Requires*-Beziehung dargestellt. Das Beispiel im rechten Teil von Abbildung 4.14 enthält zwei *Requires*-Beziehungen des digitalen Videosystems. Über SMS-Nachrichten oder E-Mails lässt sich das System steuern. Aus Kundensicht handelt es sich um zwei völlig voneinander getrennte Merkmale.

Aus Entwicklersicht und unter Ausnutzung des entsprechenden Domänenwissens wird jedoch schnell deutlich, dass zwischen diesen Merkmalen ein technologischer Zusammenhang besteht. Unabhängig vom gewählten Merkmal wird ein definiertes Format für Steuernachrichten vom System verarbeitet, da auch SMS-Nachrichten in Form einer E-Mail über das Internet verschickt werden können. Sobald die Steuerung des Videosystems über SMS-Nachrichten eines Mobiltelefons gewünscht wird, muss das Merkmal „Internet“ aus einem anderen Teilbaum des Merkmalmodells ebenfalls ausgewählt werden, um nun die SMS-Steuernachricht als E-Mail an das Videosystem zu übertragen.

Neben der normalen *Requires*-Beziehung bietet FORE auch eine bedingte *RequiresC*-Beziehung an. Die Darstellung ist analog zur *Requires*-Beziehung, jedoch wird noch eine Bedingung angegeben, die erfüllt sein muss, damit die *RequiresC*-Beziehung gültig ist. Im linken Teil von Abbildung 4.15 ist die *RequiresC*-Beziehung dargestellt. Das Beispiel im rechten Teil von Abbildung 4.15 zeigt, dass das Videosystem auf Wunsch durch einen *Handheld* bzw. *Personal Digital Assistant* (PDA) gesteuert werden kann, was durch das Merkmal „Handheld“ modelliert wurde. Dieses Merkmal erfordert das Vorhandensein einer IrDA Schnittstelle in dem Basisgerät, dem „Server“.

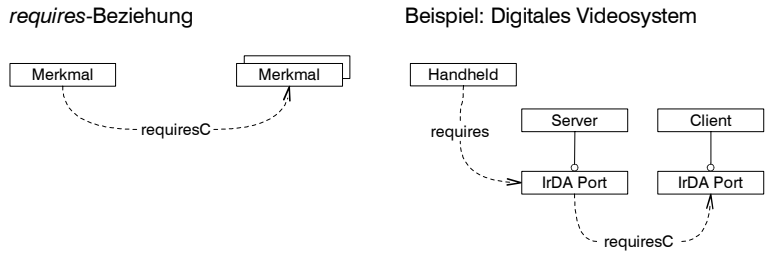


Abbildung 4.15: Die bedingte *RequiresC* Beziehung

Hat der Kunde sich für einen „On-demand“ Server entschieden, müssen in der Folge auch bei den im Haus verteilten Klienten des Servers IrDA-Schnittstellen vorhanden sein. Allerdings darf eine *Requires*-Beziehung nur bei Vorhandensein des „On-demand“ Merkmals gültig sein. Zu diesem Zweck stellt FORE die bedingte *RequiresC*-Beziehung bereit, bei der noch eine zusätzliche Bedingung angegeben werden muss. Nur wenn diese Bedingung zutrifft, ist auch die Beziehung gültig. In dem obigen Beispiel gilt die *RequiresC*-Beziehung nur dann, wenn das „On-demand“ Merkmal gewählt ist.

Im Rahmen eines guten Verkaufgespräches werden Kunden auf sinnvolle Ergänzungen zu einem Produkt hingewiesen. FORE bietet dazu Hinweise an, die im Merkmalmodell hinterlegt werden. Die Hinweise gehen, wie im linken Teil der Abbildung 4.16 dargestellt, von einem Merkmal aus und verweisen auf ein oder mehrere Merkmale. Der Hinweis kann positiv sein, wenn die weiteren Merkmale auch gewählt werden sollten, oder negativ, wenn die Auswahl der weiteren Merkmale

nicht sinnvoll ist. Die Informationen des Merkmaldiagramms bieten einem Kunden damit die Möglichkeit, effizient ein Verständnis für das beschriebene System zu erlangen. Er kann sich ein Bild des Systems aufbauen und ist damit in der Lage, die für ihn relevanten Merkmale auszuwählen. Im Laufe des Auswahlprozesses durchläuft der Kunde das Merkmalmodell von oben nach unten und muss dabei die jeweils angebotenen Merkmale auswählen oder weglassen. Um für den Systementwickler und den Kunden einen optimalen Auswahlprozess zu gestalten, sollte der Kunde auf jeweils sinnvolle Ergänzungen seiner Auswahl hingewiesen werden. Hinweise werden von Entwicklern in Zusammenarbeit mit Verkaufsabteilungen erstellt und im Modell verankert. Bei einem späteren Verkaufsgespräch fließen die Hinweise in die Verkaufsstrategie ein. Je mehr Merkmale der Kunde auswählt desto mehr muss er für das Endprodukt bezahlen. Aus Sicht des Kunden bieten Hinweise die Möglichkeit einer zweckmäßigen Ergänzung des von ihm konfigurierten Systems.

Im rechten Teil der Abbildung 4.16 hat sich ein Kunde zunächst gegen die Möglichkeit des Editierens von Datenströmen entschieden, wodurch er nicht gefragt wird, ob er Datenströme nachvertonen möchte. Entscheidet er sich allerdings für das Merkmal „Diashow“, um mit dem Videosystem Dias von Foto-CDs oder aus digitalen Fotoapparaten sichten zu können, ist ein nochmaliger Hinweis auf die Möglichkeit des Nachvertonens angebracht, da dies für die Betrachtung von Bildern eine sinnvolle Ergänzung der Fähigkeiten des Videosystems darstellt.

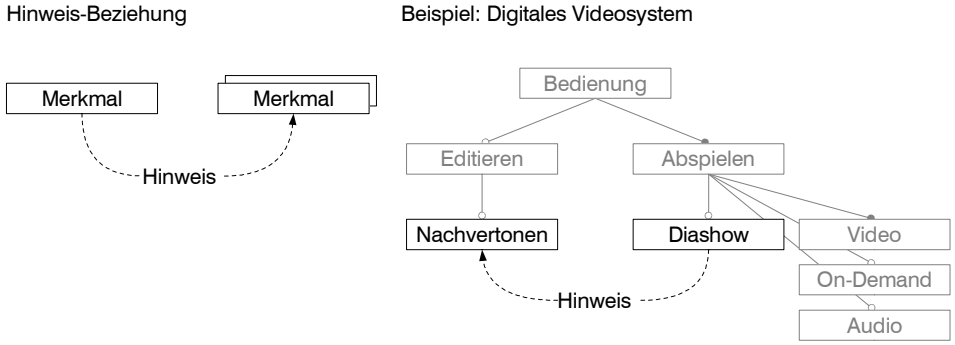


Abbildung 4.16: Empfehlungen als Hinweis-Beziehung

Die beiden Merkmale „Nachvertonen“ und „Diashow“ sind durch einen Hinweis miteinander verbunden. Vom Start des Pfeiles wird auf das an der Pfeilspitze befindliche Merkmal hingewiesen. Wird das Merkmal „Diashow“ gewählt, steht der Hinweis auf die mögliche Nachvertonung einer Diashow zur Verfügung. Somit werden sinnvolle, aber nicht zwingend notwendige Merkmale als Ergänzung zu einer bestehenden Auswahl angeboten. Diese Beziehung beeinflusst und unterstützt damit direkt den Auswahlprozess eines Kunden.

Eine große Zahl von Einstellmöglichkeiten werden durch Parametermerkmale realisiert. Neben den bereits beschriebenen Beziehungen zwischen Merkmalen müssen noch weitere Einschränkungen

bei der Vergabe von Parameterwerten berücksichtigt werden. FORE stellt mathematische Beziehungen zur Verfügung, um Bedingungen zwischen Parametern modellieren zu können. Eine mathematische Beziehung verknüpft mehrere Merkmale miteinander. Mathematische Beziehungen haben eingehende Kanten von Parametermerkmalen deren Werte als Ausgangsgrößen dienen und ausgehende Kanten von Parametermerkmalen deren Werte errechnet oder beschränkt werden, je nachdem, ob Gleichungen oder Ungleichungen angegeben werden. Die zwei Merkmalmengen und die mathematische Beziehung sind im linken Teil von Abbildung 4.17 dargestellt.

Im rechten Teil von Abbildung 4.17 kann das digitale Videosystem in einem Haushalt als *Video-On-Demand*-Server genutzt werden. Zunächst entscheidet der Kunde durch ein Merkmal, ob er diese Funktionalität überhaupt wünscht. In einem nächsten Schritt muss er die Anzahl der Klienten angeben, die der Server bedienen soll. Zwischen dieser Anzahl und der dafür notwendigen Netzwerkbandbreite gibt es einen eindeutig definierbaren Zusammenhang.

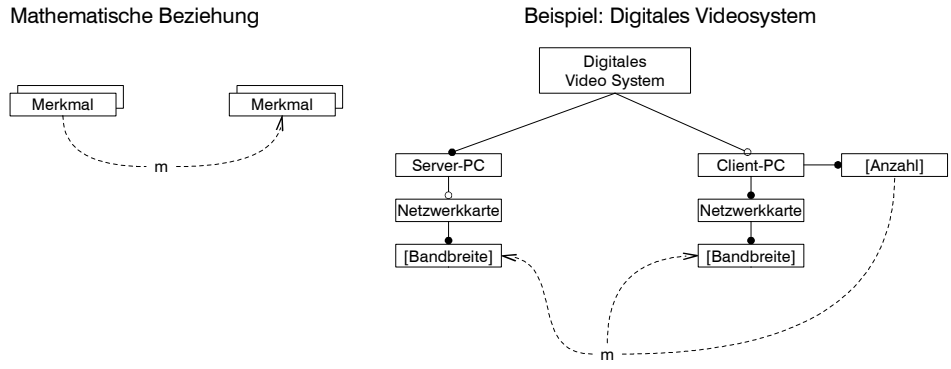


Abbildung 4.17: Quantifizierbare Merkmalbeziehung

Die Bandbreite eines Klienten ist definiert durch einen einzelnen Videostrom, der jeweils angezeigt werden soll. Der Server und das benutzte Netzwerk müssen jedoch eine Bandbreite aufweisen, die sich aus dem Produkt der Klientenbandbreite und der Anzahl der Klienten ergibt. Die Beziehung geht von dem Parametermerkmal für die Anzahl der Klienten aus und beeinflusst den Parameter für die Bandbreite bei dem Server unter Nutzung der bekannten Bandbreite des Klienten. Das kleine „m“ als Kennzeichnung dieser Beziehung steht in FORE für den mathematischen Ausdruck, der im Modell hinterlegt wird.

Die schon in der Definition erwähnten Schnittstellenmerkmale können Auswahlkriterien für Kunden darstellen. In einem komponentenbasierten System müssen Schnittstellen jedoch an den entsprechenden Stellen übereinstimmen. Dazu bietet FORE eine Beziehung an, die eine Überprüfung von Schnittstellen vorschlägt. Im linken Teil von Abbildung 4.18 ist der prinzipielle Aufbau der *interface-match*-Beziehung dargestellt. Es gibt bei dieser Beziehung nur ausgehende Kanten zu betrof-

fenen Merkmalen. Für die Merkmalmenge muss der Entwickler dann überprüfen, ob die jeweiligen Schnittstellen zueinander passen.

Im rechten Teil von Abbildung 4.18 ist ein Beispiel mit unterschiedlichen Netzwerktechnologien und Protokollen dargestellt. Die Netzwerkschnittstellen der Klienten müssen zu der des Servers passen. Insbesondere bei einem späteren Nachkauf des Klienten müssen derartige Beziehungen berücksichtigt werden, die in FORE durch die *interface-match*-Beziehungen modelliert werden.

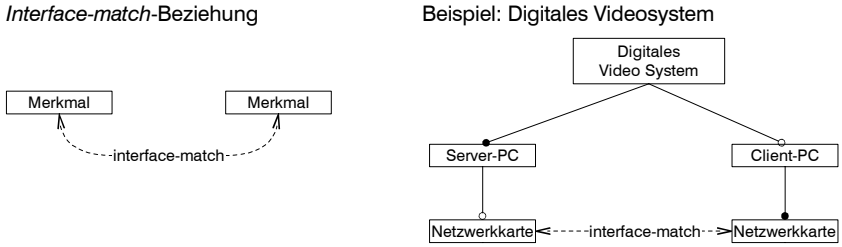


Abbildung 4.18: *Interface-match*-Beziehung

Die Auswahl von Merkmalen führt zu der vom Kunden gewünschten Variante des Systems, was im FORE-Modell hinterlegt wird. Für jedes Merkmal wird vermerkt, welchen Varianten es angehört. Handelt es sich um ein Parametermerkmal muss auch die Belegung und der Name der Variante festgehalten werden. FORE sieht hierfür ein gestricheltes Element vor, das den Namen der Variante und die Belegung des eventuellen Parameters enthält. Im linken Teil von Abbildung 4.19 ist der prinzipielle Aufbau der Varianten-Beziehung dargestellt. Als Beispiel ist im rechten Teil von Abbildung 4.19 die Variante „Basis“ des Servers dargestellt. Für das Merkmal „Server“ ist nur der Name der Variante, für die Parametermerkmale auch die für die Variante gültige Belegung vermerkt.

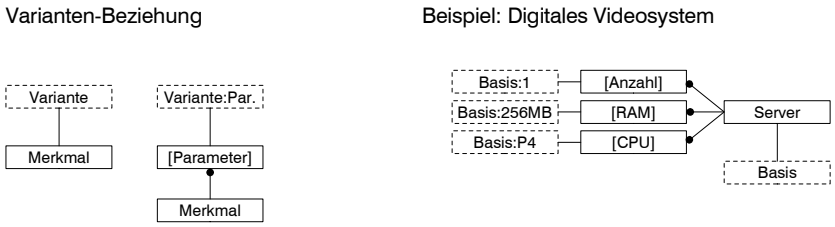


Abbildung 4.19: Varianten-Beziehung

Das Merkmalmodell hat auch Beziehungen, die aus dem Modell hinausgehen, zu anderen Modellelementen. Dazu zählen Beziehungen zu dem Personenmodell, für die jeweils an Veränderungen verantwortlichen Personen und eine Beziehung zum Versionsmanagement, um die Entwicklungshistorie festzuhalten. Des Weiteren wird jedes Merkmal dem Dokumentmodell zugeordnet, um die Position im Spezifikationsdokument festzulegen. Werden bereits existierende Komponenten genutzt, können diese direkt über Merkmale referenziert werden. Die *Commercials Off The Shelf*

(COTS) werden durch die Beziehung eines Merkmals zu einer Komponente des Komponentenmodell referenziert. FORE stellt für Komponenten zwei Beziehungen zur Verfügung. Die *realizedIn*-Beziehung referenziert direkt die Komponente, durch die ein Merkmal realisiert wird. Demgegenüber ist die *realizedC*-Beziehung an eine Bedingung geknüpft, in die beliebige Merkmale eingebunden sein können. Im linken Teil von Abbildung 4.20 sind die Beziehungen aus dem Merkmalmodell zu weiteren Modellen dargestellt.

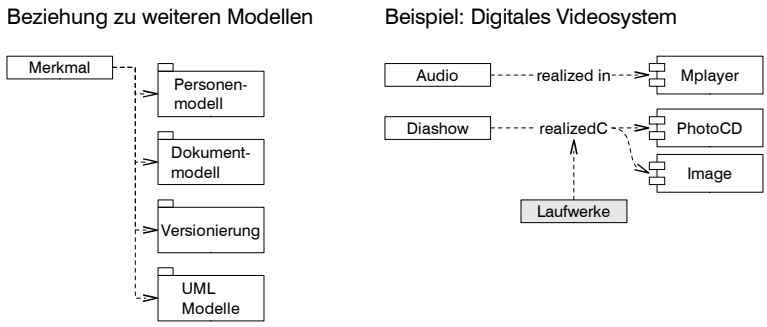


Abbildung 4.20: Komponentenbeziehungen

Als Beispiel aus dem Videoprojekt ist im rechten Teil von Abbildung 4.20 das in der Komponente „Mplayer“ realisierte Merkmal „Audio“ dargestellt. Die Komponente ist in der Lage, Audiodaten abzuspielen. Für einen Nutzer ergibt sich nur ein neuer Menüeintrag in dem Bildschirmdialog zur Steuerung des Systems. Nach Integration des Merkmals kann der Nutzer über diesen Menüeintrag beliebige Audiodateien wählen, um sie abzuspielen. Analog verhält es sich bei dem Merkmal „Diashow“. Hier können allerdings zwei Komponenten genutzt werden, was durch die *realizedC*-Beziehung modelliert wurde. Die Bedingung hängt von dem gewählten Laufwerk ab. Ist das Laufwerk in der Lage das Photo-CD-Format zu lesen, kann die „Photo-CD“-Komponente in das System integriert werden, sonst ist nur die „Image“-Komponente integrierbar.

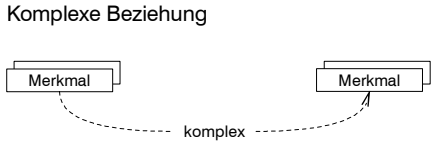
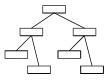
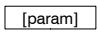
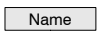
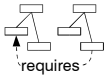
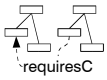
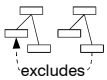
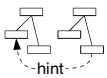
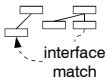
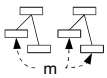
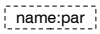


Abbildung 4.21: Komplexe Beziehung

Schließlich stellt FORE noch eine allgemeine Beziehung bereit, um komplexere Sachverhalte modellieren zu können, die nicht durch die bereits erläuterten Beziehungen modellierbar sind. Die als „komplex“ bezeichnete Beziehung ist in Abbildung 4.21 dargestellt und nutzt die Flexibilität und

Erweiterbarkeit der *Feature Constraint Language* aus, die in Kapitel 4.2.3 näher erläutert wird. Über einen Ausdruck in der FCL werden alle beteiligten Merkmale miteinander verknüpft.

In Tabelle 4.3 sind alle in FORE definierten Beziehungen mit einer kurzen Erläuterung aufgelistet.

Name	Art	Beschreibung
Hierarchie	 Merkmal – Untermerkmal	Die Hierarchie drückt einerseits die Teil-Ganzes-Beziehung aus. Andererseits spiegelt die Hierarchie die Reihenfolge der vom Kunden zu fällenden Entscheidungen wider.
Parameter	 Merkmal	Dieses Merkmal stellt einen vom Kunden zu wählenden Parameter dar.
Struktur	 Merkmal	Dieses Merkmal dient lediglich der Strukturierung des Modells.
Requires	 Merkmal(e) – Merkmal(e)	Die Auswahl eines Merkmals erfordert das Vorhandensein eines anderen Merkmals.
RequiresC	 Merkmal(e) – Merkmal(e)	Die Auswahl eines Merkmals erfordert das Vorhandensein eines anderen Merkmals, wobei diese Einschränkung an eine Bedingung geknüpft ist.
Excludes	 Merkmal(e) – Merkmal(e)	Die Auswahl eines Merkmals verbietet das Vorhandensein eines anderen Merkmals.
Hinweis	 Merkmal(e) – Merkmal(e)	Durch Hinweise auf weitere sinnvolle Merkmale, als Ergänzung zu einem gerade gewählten Merkmal, sind Hinweise als Verkaufsargumente zu sehen.
Schnittstelle	 Merkmal(e) – Merkmal(e)	Werden mehrere Komponenten in einem System integriert, müssen die jeweiligen Schnittstellen zusammenpassen.
Mathematische Ausdrücke	 Merkmal(e) – Merkmal(e)	Quantifizierbare Beschränkungen mit einstellbaren Parametermerkmalen werden mit Hilfe mathematischer Ausdrücke beschrieben.
Variante	 Merkmal – Variante	Einem Merkmal kann eine Variante mit Namen und eventuellem Parameterwert zugeordnet werden. Damit wird die Auswahl der Merkmale für eine Variante im Modell hinterlegt.
Entwurf	 Merkmal – UML Modell(element)	Je nach genutzter Systemfamilientechnologie werden allgemeine Beziehungen zwischen Merkmalen und Entwurfselementen modelliert.
Historie	 Merkmalbaum – Merkmalbaum	Durch Änderung oder Verbesserung entstehen mehrere Versionen eines Merkmals oder eines ganzen Merkmalbaumes.
Autor	 Merkmal – Person	Bei der Erstellung oder Veränderung eines Merkmals wird der Autor über eine Referenz in das Personenmodell mit angegeben.

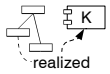
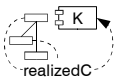
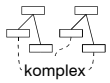
Name	Art	Beschreibung
Dokument	 Merkmal – Dokumentelement	Merkmale können an einer bestimmten Position in der Spezifikation hinterlegt werden. Über eine Referenz in das Dokumentmodell wird diese Position angegeben.
realized	 Merkmal – Komponente(n)	Ein Merkmal kann direkt durch eine eventuell bereits existierende Komponente realisiert werden. Die Beziehung referenziert Komponenten, die im Komponentenmodell der UML enthalten sind.
realizedC	 Merkmal(e) – Komponente(n)	Hängt die Auswahl einer Komponente von einer Untermenge von Merkmalen ab, werden diese Abhängigkeiten in dieser Beziehung hinterlegt. Je nach Auswahl werden andere Komponenten in das System integriert, die dann auch unterschiedlich parametriert werden.
Komplex	 Merkmal(e) – Merkmal(e)	Komplexe Zusammenhänge, die nicht sinnvoll durch die bereits genannten Beziehungen beschreibbar sind, werden durch ein frei definierbares Beziehungsgeflechtes modelliert.

Tabelle 4.3: Beziehungen und Elemente im FORE-Merkmalmodell

4.2.2 Konsistenz von Merkmalmodellen

Mit Hilfe der erweiterten Merkmalmodellierung werden zunächst alle für die Entwicklung und Produktkonfiguration relevanten Daten erfasst. Im ersten Schritt einer Entwicklung helfen die bereits vorgestellten Anforderungs- und Merkmalkarten, um zunächst vollständige Datensätze für Anforderungen und Merkmale zu erheben, bevor diese in das Datenmodell eingepflegt werden. Alle vom Entwickler genutzten Beziehungen vom, zum und innerhalb des Merkmalmodells müssen überprüft werden, um die Konsistenz des Modells bei der Erstellung und bei späteren Änderungen wahren zu können. Die Konsistenz des Modells sollte dabei von entsprechenden Werkzeugen automatisiert überprüft werden, um eine dem heutigen Stand der Technik angemessene Entwicklung zu ermöglichen. Im Folgenden werden die verschiedenen Arten der Konsistenzprüfungen unter Nutzung der Modellinformationen beschrieben.

Ausgehend von den Anforderungen muss überprüft werden, ob sich alle nicht-funktionalen Anforderungen in den funktionalen Anforderungen oder den Merkmalen des Systems widerspiegeln. Für alle nicht-funktionalen Anforderungen, ohne jegliche Beziehung zu anderen Modellelementen, muss deren Gültigkeit für das System überprüft werden. Nicht-funktionale Anforderungen beeinflussen das System und sollten immer Modellelementen zugeordnet werden, um eine Validierung zu ermöglichen. Fehlt eine solche Zuordnung, kann die Erfüllung der betreffenden nicht-funktionalen Anforderung nicht nachgewiesen werden. In diesem Fall muss der Entwickler seine Entscheidung, eine entsprechende nicht-funktionale Anforderung ohne Bezüge zu anderen Modellelementen aufzunehmen, begründen und diese Begründung im Modell hinterlegen.

Zwischen dem Anforderungsmodell und dem Merkmalmodell muss überprüft werden, ob jedem Merkmal Anforderungen zugeordnet sind, die bei der Implementierung in den entsprechenden

Komponenten umzusetzen sind. In der anderen Richtung müssen auch alle Anforderungen jeweils Merkmalen zugeordnet sein, um die vollständige Systemfamilie realisieren zu können. Dabei gilt, dass durch den hierarchischen Aufbau des Anforderungsmodells bei der Zuordnung einer Anforderung A zu einem Merkmal M, automatisch auch die Anforderungen des Teilbaumes unterhalb von A dem Merkmal M zugeordnet werden. Für das Merkmalmodell gilt dies nicht, so dass hier alle Merkmale den Anforderungen zuzuordnen sind, um ein konsistentes Modell zu erhalten.

Innerhalb des Merkmalmodells muss der Wurzelknoten ein Strukturmerkmal sein. Für jedes Merkmal muss ein Bezug zum Entwurf der Systemfamilie und/oder zu einer Komponenten der Implementierung bestehen. Es muss also klar sein, wie ein Merkmal die Systemfamilienarchitektur oder die Parametrierung einer Komponente beeinflusst. Insbesondere für COTS-Komponenten ist die Integration und eventuelle Parametrierung für die Ableitung von Familienmitgliedern von Bedeutung.

Parametermerkmale werden typisiert und mit einem gültigen Wertebereich versehen, der sofort nach der Eingabe eines Parameterwertes geprüft werden muss. Mathematische Beziehungen zwischen zwei oder mehreren Merkmalen nutzen Parametermerkmale und stellen damit Gleichungssysteme auf. Für jede Variante müssen die Gleichungssysteme der beteiligten Merkmale überprüft werden. Innerhalb einer Systemfamilie dürfen jedoch keine unlösbaren Gleichungen angegeben werden, da dies zur Folge hätte, dass keine Varianten abgeleitet werden könnten.

Wie in Abbildung 4.22 dargestellt, bilden die mathematischen Beziehungen ① und ② der drei beteiligten Parametermerkmale ein Gleichungssystem aus, wobei die paarweise Auswahl von Merkmal A und B oder Merkmal A und C keine Probleme darstellt. Werden jedoch alle drei Merkmale ausgewählt, ergibt sich aus den beiden Gleichungen direkt die Beziehung $c=2b$ zwischen den Merkmalen B und C. Diese Beziehung war vorher nicht vorhanden und gilt nur, wenn für eine Variante alle drei Merkmale ausgewählt wurden. Es muss geprüft werden, ob diese Beziehung mit den Wertebereichen der Merkmale B und C vereinbar ist. Ist dies nicht der Fall, kann eine Variante mit allen drei Merkmalen nicht erzeugt werden. Sind mehr als drei Merkmale in dem Ring enthalten, so entsteht ein Gleichungssystem für alle Merkmale, wobei alle Beziehungen abgeleitet werden können. Bei Angabe eines Wertes können alle weiteren Werte innerhalb des Ringes abgeleitet werden, sofern das Gleichungssystem eine von Null verschiedene Lösung hat. Die Null-Lösung ist zwar mathematisch korrekt, stellt jedoch meist keine praktisch brauchbare Lösung dar. Für jede neu in das Modell eingefügte mathematische Beziehung muss geprüft werden, ob sie nicht von bereits bestehenden mathematischen Beziehungen invalidiert wird oder als redundant verworfen werden kann.

Ergeben sich unlösbare Gleichungssysteme mathematischer Beziehungen, kann immer durch weglassen von ein oder mehreren Gleichungen eine Lösung angegeben werden. In der Folge bedeutet

dies, dass jeweils nur eine Untermenge der Merkmale, die an dem Gleichungssystem beteiligt sind, in einer Konfiguration vorhanden sein darf.

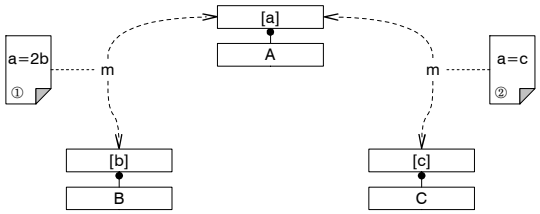


Abbildung 4.22: Ring-Beziehungen im Merkmalmodell

Die *Requires* und *Excludes*-Beziehungen können auch Inkonsistenzen hervorrufen, wenn zwei Merkmale gegenseitig in Beziehung stehen. Wie in Abbildung 4.23 zu sehen, erfordert Merkmal B das Vorhandensein von Merkmal A, was durch die *Requires*-Beziehung modelliert ist. Gleichzeitig aber darf bei Auswahl von Merkmal A aufgrund der *Excludes*-Beziehung das verbundene Merkmal B nicht gewählt werden. Dies ist ein Widerspruch in dem Modell, der nur zum Tragen kommt, wenn sowohl Merkmal A als auch Merkmal B gewählt werden, was allerdings für die modellierten Beziehungen eine zu eliminierende Verletzung des Modells darstellt. Eine der beiden Beziehungen muss aufgehoben oder die Merkmale müssen abgeändert werden, um eine der beiden Beziehungen entfernen zu können.

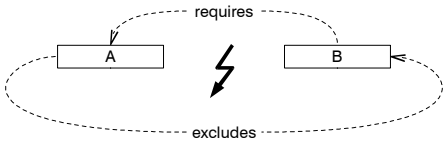


Abbildung 4.23: Requires/Excludes Konsistenz

Die neue Notation für Merkmalmodelle erlaubt die Modellierung von Merkmalmengen unter Angabe einer Vielfachheit für die minimal und maximal wählbaren Merkmale aus der Menge. Natürlich gilt für die Vielfachheit (Beispiel: 1..* oder 3..7, allgemein min..max), dass der Minimalwert immer kleiner oder gleich dem Maximalwert sein muss. Ansonsten wäre auch wieder das entstehende Modell inkonsistent.

Die zu prüfenden Konsistenzregeln innerhalb eines FORE-Modells sind in Tabelle 4.4 zusammengefasst. Neben der Beschreibung der jeweils zu überprüfenden Regel, ist auch ein C ähnlicher Pseudocode enthalten, der eine spätere Umsetzung in einem Werkzeug erleichtert. Es wird davon ausgegangen, dass die Anforderungen in einem hierarchischen Modell, wie in Kapitel 4.1 beschrieben, vorliegen. Des Weiteren werden so genannte *Tree-walker* vorausgesetzt, welche die Traversierung

des Baumes erlauben, wobei es pro Modell einen *Tree-walker* gibt. Alle fett gedruckten Codeanteile stellen einfache Funktionen dar, die hier nicht näher erläutert werden und in einer konkreten Implementierung realisiert werden müssen.

Nummer	Beteiligte Elemente	Beschreibung der Regel

1.	Anforderungen	Alle nicht-funktionalen Anforderungen sollten funktionale Anforderungen oder Merkmale referenzieren.
		
		`void KCheck_1()`
		`{`
		`  Node current = Req_treeWalker->getCurrentNode();`
		`  Bool isOk=false;`
		
		`  if(`**`isNonFuncRequirement`**`(current))`
		`  {`
		`  List* references = `**`getRef`**`(current);`
		`  for(int pos=0; pos<references->getLength();`
		`  pos++)`
		`  {`
		`  if(`**`isFunctionalRequirement`**`(`
		`  references->item(pos)))`
		`  isOk=true;`
		`  }`
		`  }`
		
		`  if(!isOk)`
		`  {`
		`  // Hier muss der Entwickler für die aktuelle`
		`  // Anforderung (current) überprüfen, ob`
		`  // die fehlenden Referenzen gewollt sind.`
		`  }`
		
		`  for(Node* child = Req_treeWalker->firstChild();`
		`  child != NULL;`
		`  child = Req_treeWalker->nextSibling())`
		`  {`
		`  KCheck_1();`
		`  }`
		`  Req_treeWalker->setCurrentNode(current);`
		`}`

Nummer	Beteiligte Elemente	Beschreibung der Regel

2.	Anforderungsmodell, Merkmalmodell	Jedes Merkmal muss Anforderungen im Anforderungsmodell referenzieren. Die Referenz auf eine Anforderung inkludiert automatisch den gesamten Teilbaum dieser Anforderung.
		
		`void KCheck_2()`
		`{`
		`  Node current = Feat_treeWalker->getCurrentNode();`
		`  Bool isOk=false;`
		
		`  List* references = getRef(current);`
		`  for(int pos=0; pos<references->getLength();`
		`  pos++)`
		`  {`
		`  if(isRequirement(references->item(pos)))`
		`  isOk=true;`
		`  }`
		
		`  if(!isOk)`
		`  {`
		`  // Ein Merkmal ohne Referenzen in das`
		`  // Merkmalmodell wurde gefunden!`
		`  // Der Entwickler muss dieses Mekmal`
		`  // überprüfen und die entsprechenden`
		`  // Referenzen setzen.`
		`  }`
		
		`  for(Node* child = Feat_treeWalker->firstChild();`
		`  child != NULL;`
		`  child = Feat_treeWalker->nextSibling())`
		`  {`
		`  KCheck_2();`
		`  }`
		`  Feat_treeWalker->setCurrentNode(current);`
		`}`

3.	Merkmale	Typisierung (es sind Zahlen und Texte möglich) und Prüfung auf Einhaltung des Wertebereichs eines Parametermerkmals. Hier ist die Überprüfung des gesamten Modells enthalten, die Prüfung direkt während der Modellierung wurde nicht berücksichtigt, da sie sich trivialerweise aus der Gesamtüberprüfung ergibt.
		
		`void KCheck_3()`
		`{`
		`  Node current = Feat_treeWalker->getCurrentNode();`
		`  Bool isOk=false;`
		
		`  if(isParamFeature(current))`
		`  {`
		`  switch(getParamType(current))`
		`  {`
		`  case DOUBLE_TYPE : isOK = checkNumBounds(current); beak;`
		`  case STRING_TYPE : isOK = checkStrBounds(current); beak;`
		`  default : break;`
		`  }`
		
		`  if(!isOk)`
		`  {`
		`  // Ein Parametermerkmal verletzt die Grenzen`
		`  // seines Datentyps! Eine Anpassung der Werte`
		`  // ist notwendig.`
		`  }`
		`  }`
		
		`  for(Node* child = Feat_treeWalker->firstChild();`
		`  child != NULL;`
		`  child = Feat_treeWalker->nextSibling())`
		`  {`
		`  KCheck_3();`
		`  }`
		`  Feat_treeWalker->setCurrentNode(current);`
		`}`

4.	Merkmale	Innerhalb einer Gleichung müssen Parametertypen und Wertebereich geprüft werden. Diese Prüfung erfolgt analog zu der Prüfung 3.

<table>
<tr><th>Nummer</th><th>Beteiligte Elemente</th><th>Beschreibung der Regel</th></tr>
<tr><td>5.</td><td>Merkmale, mathematische Beziehung</td><td>Mehrere zusammenhängende, mathematische Beziehungen ergeben ein Gleichungssystem, welches lösbar sein muss, wenn alle beteiligten Merkmale ausgewählt sind. Eine Implementierung dieser Überprüfung ist nur unter Zuhilfenahme von Mathematiksystemen, wie beispielsweise Mathematica [Math 2003], sinnvoll möglich. Die detaillierte Betrachtung dieser Konsistenzprüfung wird zukünftigen Forschungsvorhaben überlassen.</td></tr>
<tr><td>6.</td><td>Merkmale</td><td>Zwischen zwei Merkmalen sind Requires und Excludes Beziehungen nicht gleichzeitig erlaubt. Eine der beiden Beziehungen muss eliminiert werden.

<pre>
void KCheck_6()
{
   Node current = Feat_treeWalker->getCurrentNode();

   List* references = getRef(current);
   for(int pos=0; pos<references->getLength();
       pos++)
   {
       if(!(existsRequires(current,references->item(pos)) &&
            existsExcludes(current,references->item(pos))))
       {
           // Eine der beiden Beziehungen (reqires
           // oder excludes) muss eliminiert werden.
       }
   }

   for(Node* child = Feat_treeWalker->firstChild();
       child != NULL;
       child = Feat_treeWalker->nextSibling())
   {
       KCheck_6();
   }
   Feat_treeWalker->setCurrentNode(current);
}
</pre></td></tr>
<tr><td>7.</td><td>Merkmal</td><td>Der Wurzelknoten muss immer ein Strukturmerkmal sein.

<pre>
void KCheck_7()
{
   Node root = getRootFeatureNode();

   if(!isStructureNode(root))
   {
       // Der Wurzelknoten muss als Strukturmerkmal
       // modelliert werden!
   }
}
</pre></td></tr>
</table>

<table>
<tr><th>Nummer</th><th>Beteiligte Elemente</th><th>Beschreibung der Regel</th></tr>
<tr><td>8.</td><td>Merkmale</td><td>Werden alle COTS- referenziert? Bleiben COTS übrig, muss geprüft werden, ob diese für die Systemfamilie überhaupt notwendig sind.

<pre>
List* components; // Liste Referenzen zu allen COTS enthalten
void KCheck_8()
{
   Node current = Feat_treeWalker->getCurrentNode();

   List* references = getRef(current);
   for(int pos=0; pos<references->getLength();
       pos++)
   {
       if((isUMLComponent(references->item(pos)))
       {
           components->remove(references->item(pos));
       }
   }

   for(Node* child = Feat_treeWalker->firstChild();
       child != NULL;
       child = Feat_treeWalker->nextSibling())
   {
       KCheck_8();
   }

   if(components->getLength()>0)
   {
       // Es bleibt eine Liste mit allen COTS übrig!
       // Jede dieser Komponenten muss nun vom Entwickler
       // überprüft werden.
   }

   Feat_treeWalker->setCurrentNode(current);
}
</pre></td></tr>
<tr><td>9.</td><td>Merkmale, Entwurfselemente, Komponenten</td><td>Für jedes Merkmal muss klar sein, wo sich die Variabilität im Entwurf und/oder in der Implementierung wiederfindet. Fehlende Bezüge müssen eingefügt oder explizit durch den Entwickler begründet werden.

<pre>
void KCheck_9()
{
   Node current = Feat_treeWalker->getCurrentNode();
   Bool noRefsToUML=true;

   List* references = getRef(current);
   for(int pos=0; pos<references->getLength();
       pos++)
   {
       if((isUMLElement(references->item(pos)))
        noRefsToUML=false;
   }

   if(noRefsToUML)
   {
       // Das aktuelle Merkmal (current) hat keinen Bezug
       // zu dem Entwurf bzw. der Implementierung.
       // Der Entwickler muss dieses Merkmal überprüfen.
   }

   for(Node* child = Feat_treeWalker->firstChild();
       child != NULL;
       child = Feat_treeWalker->nextSibling())
   {
       KCheck_9();
   }
   Feat_treeWalker->setCurrentNode(current);
}
</pre></td></tr>
</table>

Tabelle 4.4: Konsistenzregeln eines FORE-Modells

Die Konsistenzregeln sollten immer für das jeweilige Gesamtmodell einer Entwicklung überprüft werden. Für die *Requirements-Engineering*-Phase der Systemfamilien bieten existierende Modelle keine Möglichkeit der Überprüfung an. Ein entsprechendes, überprüfbares Modell für diesen Lösungsansatz wird in Kapitel 4.4 definiert und erläutert.

Im Bereich des Softwareentwurfs hat sich die standardisierte UML etabliert. Die Umsetzung aller in dieser Arbeit vorgestellten Beziehungen erfordert die Verknüpfung der UML-Modelle und der *Requirements-Engineering*-Modelle, die mit Hilfe frei verfügbarer Werkzeuge, wie [XMIT 2003] für UML Modelle, realisiert werden kann. Auf die technischen Details der Implementierung wird in dieser Arbeit verzichtet.

4.2.3 Feature Constraint Language

Die im Verlauf dieses Kapitels beschriebenen Beziehungen zwischen Merkmalen können nur automatisiert verarbeitet werden, wenn sie in einer dafür geeigneten Form vorliegen. Die bereits angesprochene *Object Constraint Language* (OCL), die in der Entwurfsphase bei UML-Modellen Verwendung findet, stellt an dieser Stelle die Basis für die im Folgenden beschriebene und im Rahmen dieser Arbeit entwickelte *Feature Constraint Language* (FCL) dar.

OCL ist per Definition frei von Nebeneffekten, es werden also keine Veränderungen an dem Modell vorgenommen, das es zu überprüfen gilt. OCL-Ausdrücke können damit entweder zutreffen oder verletzt werden. Im ersten Fall müssen keine weiteren Aktionen folgen. Im zweiten Fall bedeutet die Verletzung einer in OCL definierten Einschränkung für das betreffende Modell, dass Veränderungen vorgenommen werden müssen, um die Verletzung zu bereinigen. In einer Entwicklungsumgebung mit OCL-Unterstützung werden damit, wie bei C oder C++ Compilern, lediglich die verletzten *Constraints* zusammen mit den beteiligten Modellelementen angegeben.

Ein wichtiges Ziel der Merkmalmodellierung ist die Bereitstellung der Möglichkeit einer Auswahl von Varianten, wobei diese Auswahl durch die im Merkmalmodell vorhandenen Beziehungen beeinflusst wird. Die *Requires*- und *Excludes*-Beziehungen weisen bei einer Verletzung direkt auf die Merkmale, die selektiert oder de-selektiert werden müssen, um die Konsistenz im Modell wieder herzustellen. Eine zeitgemäße Entwicklungsumgebung muss dem Entwickler die erforderliche Modelländerung zur automatischen Ausführung anbieten. Andere Beziehungen, wie der Hinweis auf weitere Merkmale und die Überprüfung der Übereinstimmung von Schnittstellen mehrerer Merkmale, erfordern einen aktiven Eingriff in den Entwicklungs- und Auswahlprozess, indem Entwicklern oder Kunden wichtige Informationen dargestellt werden.

In FORE ist die *Feature Constraint Language* mit einer Grundform definiert, die aus 4 Teilen besteht und in Klammern eingeschlossen wird. Die beteiligten Merkmale müssen angegeben werden, lokale

Variablen für die Parametermerkmale können angegeben werden, um die Lesbarkeit des Ausdrucks zu verbessern und die eigentliche Beziehung muss formuliert werden. Die an einer Beziehung beteiligten Merkmale werden in der Kontext Definition durch Komma getrennt angegeben. Variablen für die zu nutzenden Parametermerkmale werden in dem Definitionsteil deklariert. Sie sind letztlich nur Platzhalter für die entsprechenden Parameter. Auch hier werden zur besseren Lesbarkeit der Merkmalname und der Parametername angegeben. Dazu kommt noch der Datentyp des Parametermerkmals. Der grundlegende Aufbau eines FCL-Ausdrucks ist in Abbildung 4.24 enthalten.

```
(
      context     <Merkmal>, <Merkmal>, ... ;
      def         <Variablenname> : <Merkmal>.<Parameter>[<Typ>],
                  <Variablenname> : <Merkmal>.<Parameter>[<Typ>],
                  ... ;
      inv         <Boolescher Ausdruck> ;
      action      <Auszuführende Aktion> ;
)
```

Abbildung 4.24: Grundaufbau der neu entwickelten *Feature Constraint Language (FCL)*

Die Invariante selbst beschreibt die Bedingung, die erfüllt sein muss, wenn alle beteiligten Merkmale, enthalten im Kontext des Konstruktes, in einer Variante ausgewählt werden. Diese Bedingung, auch als *Constraint* bezeichnet, wird durch eine Syntax beschrieben, die einer Programmiersprache ähnelt.

Schließlich wird festgehalten, welche Aktionen ausgeführt werden sollen, falls die angegebene Invariante verletzt wird. Dies stellt eine Abweichung von der in der OCL definierten Nebeneffektfreiheit dar. Allerdings kann eine automatisierte Bearbeitung des Modells nur realisiert werden, wenn auch Konstrukte zur Veränderung des Modells existieren. Die Aktionen können das Modell verändern und müssen daher mit großer Sorgfalt eingesetzt werden. Allerdings bieten die automatisierten Veränderungen am Modell einen hohen Komfort bei der Nutzung desselben. Neben den Fragen zu der kundenspzifischen Auswahl von Merkmalen kommen nun weitere Fragen hinzu, die aus den im Modell formulierten Bedingungen resultieren. Das Modell selbst und der Auswahlprozess gewinnen dadurch an Qualität.

In Tabelle 4.5 sind die Ausdrücke für die bereits definierten Beziehungen des Merkmalmodells enthalten. Für die Ausführung der Ausdrücke wird immer davon ausgegangen, dass ein Merkmalmodell mit entsprechenden Varianten existiert und es muss bekannt sein, welche Variante zu prüfen ist.

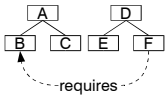
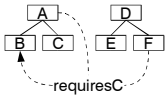
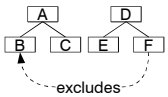
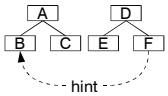
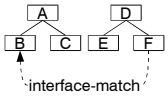
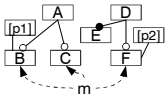
Name	Teilmodell	Beispielausdruck in der *Feature Constraint Language*
Requires		(**context** B,F; **inv** selected(F) and selected(B); **action** if(question("Select B?")) select(B);)
RequiresC		(**context** A,B,F; **inv** selected(F) and selected(B) and selected(A); **action** if(question("Select A and B?")) (select(A); select(B);))
Excludes		(**context** B,F; **inv** selected(F) and (not selected(B)); **action** if(question("De-select B?")) deselect(B);)
Hinweis		(**context** B,F; **inv** selected(F) and selected(B); **action** if(question("B might be usefull, select B?")) select(B);)
Schnittstelle		(**context** B,F; **inv** not(selected(F) and selected(B)); **action** if(note("Check the interface between F and B!")))
Mathematische Ausdrücke		(**context** B,C,F; **def** pB : B.p1[int], pF : F.p2[int]; **inv** if(selected(C) and selected(B) and selected(F)) then (pB <= 5*pF); // Dies ist nur ein Beispiel. **action** ;)

Tabelle 4.5: In der *Feature Constraint Language* formulierte Ausdrücke der Beziehungen des FORE-Merkmalmodells

Wie in Tabelle 4.5 dargestellt, werden von der FCL diverse Ausdrücke formuliert, die eine Umsetzung der Beziehungen erlauben. Im Definitionsteil wird auf die Parametermerkmale des Modells über eine eigens definierte Notation zugegriffen, die sich aus `<Merkmal>.<Parameter>[<Typ>]` zusammensetzt. Ein Parametermerkmal ist immer einem Merkmal zugeordnet. Dieses Merkmal wird durch die Menge der Parametermerkmale parametriert, also „eingestellt“. Darüber hinaus ist jeder Parameter typisiert, um die Überprüfung der ebenfalls anzugebenden Grenzwerte des Parameters zu realisieren. FCL stellt eine Reihe von Befehlen zur Verfügung, um auf das Merkmalmodell zuzugreifen. Im Folgenden werden diese Befehle erläutert.

- `selected(<Merkmal>)` wird genutzt, um festzustellen, ob das gegebene Merkmal in der aktuellen Auswahl enthalten ist oder nicht. Im positiven Fall liefert der Befehl den booleschen Wert *wahr* zurück.
- `select(<Merkmal>)` verändert den Auswahlstatus eines Merkmals. Nach Ausführung des Befehls ist das gegebene Merkmal Teil der aktuellen Auswahl.

- `deselect(<Merkmal>)` verändert den Auswahlstatus eines Merkmals. Nach Ausführung des Befehls ist das gegebene Merkmal nicht mehr Teil der aktuellen Auswahl.
- `question(<Frage>)` präsentiert dem Nutzer eine Frage, die mit ja oder nein beantwortet werden kann. Antwortet der Nutzer mit ja, gibt der Befehl den booleschen Wert *wahr* zurück.
- `note(<Text>)` zeigt dem Nutzer einen Hinweistext an. Dieser Text ist sinnvoll, wenn komplexere Vorgänge zur Behebung von Inkonsistenzen notwendig sind, welche die Mitarbeit des Entwicklers erfordern.

Die mathematischen Ausdrücke, die in FCL formuliert werden, bieten eine größtmögliche Flexibilität. Die Anzahl der beteiligten Merkmale sowie der mathematische Ausdruck selbst sind frei von dem Entwickler wählbar. Der in Tabelle 4.5 angegebene mathematische Ausdruck stellt daher nur ein Beispiel dar.

4.3 Entwicklungsprozess

Die Entwicklung einer Systemfamilie ist in die unterschiedlichen Phasen des bereits im dritten Kapitel beschriebenen *Six-Pack*-Modells unterteilt. Durch den FORE-Entwicklungsprozess wird der in Abbildung 4.25 grau unterlegte Teil der Systemfamilienentwicklung abgedeckt. So wie der gesamte Prozess teilt sich auch der FORE-Entwicklungsprozess in die Domänenentwicklung und die Applikationsentwicklung auf. Das dazwischen liegende Anforderungsmodell muss alle erarbeiteten Daten aufnehmen.

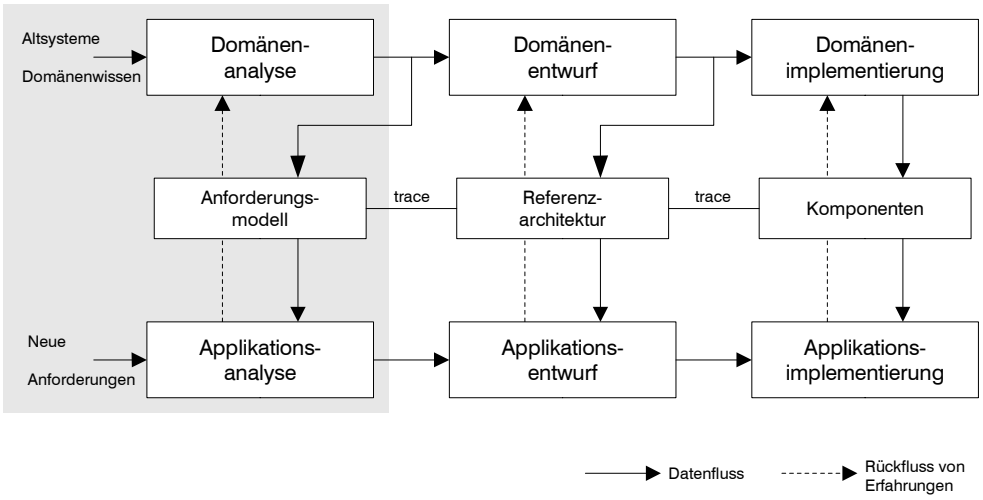


Abbildung 4.25: FORE-Einordnung in die Systemfamilienentwicklung

In der Domänenanalyse geht es um *Requirements Engineering* für die Systemfamilie. Alle Aktivitäten in dieser Entwicklungsphase zielen darauf ab, die Anforderungen und Merkmale systemfamiliengerecht zu modellieren, um so einen soliden Grundstock für die anschließende Entwurfsphase bereit zu stellen. Das Ergebnis dieser Aktivitäten ist das Anforderungs- und Merkmalmodell, welches alle Arbeitsergebnisse enthält und die Konzepte der Systemfamilienentwicklung widerspiegelt.

In der Applikationsentwicklung geht es um die Analyse der Kundenanforderungen an ein bestimmtes Produkt. Im Rahmen der *Requirements-Engineering*-Phase für Systemfamilien müssen die Kundenanforderungen mit den Möglichkeiten der Systemfamilie abgeglichen werden. Je mehr Kundenanforderungen durch die in der Systemfamilie bereits enthaltenen Komponenten abgedeckt werden können, desto schneller und kostengünstiger kann das gewünschte Produkt erstellt werden. Dieser Abgleich stellt die Grundlage für ein Angebot dar, welches dem Kunden unterbreitet wird. Anforderungen, die nicht bereits in der Systemfamilie realisiert sind, können nur durch zusätzlichen Aufwand und zusätzliche Kosten in die Familie eingebracht werden.

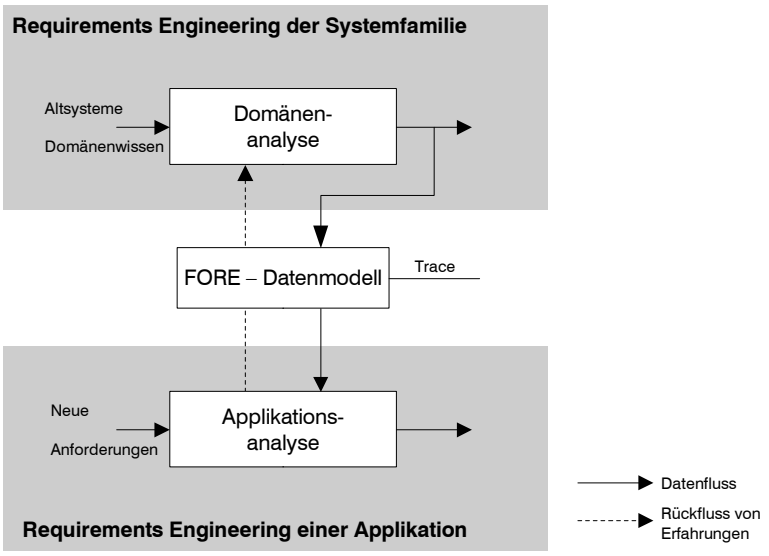


Abbildung 4.26: Überblick, FORE-Prozess

In Abbildung 4.26 ist die grobe Aufteilung der Prozessschritte der FORE-Methode dargestellt. Das FORE-Datenmodell mit der erweiterten Merkmalmodellierung ist eine Neuerung im Bereich der Systemfamilienentwicklung. Die Prozessschritte innerhalb der Domänen- und Applikationsanalyse stellen abgewandelte Varianten existierender Ansätze dar, um das FORE-Datenmodell zu unterstützen. Der FORE-Entwicklungsprozess besteht damit zum einen aus der *Requirements-Engineering*-Phase der Systemfamilie, deren Ziel die Aufbereitung von Informationen für das FORE-Datenmodell ist. Zum anderen werden in der *Requirements-Engineering*-Phase einer Applikation die Anforde-

rungen eines Kunden an eine neue Applikation mit den Gegebenheiten der Systemfamilie abgeglichen, die wiederum in dem FORE-Datenmodell hinterlegt sind.

Die Tätigkeiten der einzelnen Prozessschritte sind sehr vielschichtig und werden in den nächsten Abschnitten genauer erläutert. Die entstehenden Daten, auch als Artefakte bezeichnet, werden kurz beschrieben, um in Kapitel 4.4 in dem FORE-Datenmodell zusammengefasst zu werden.

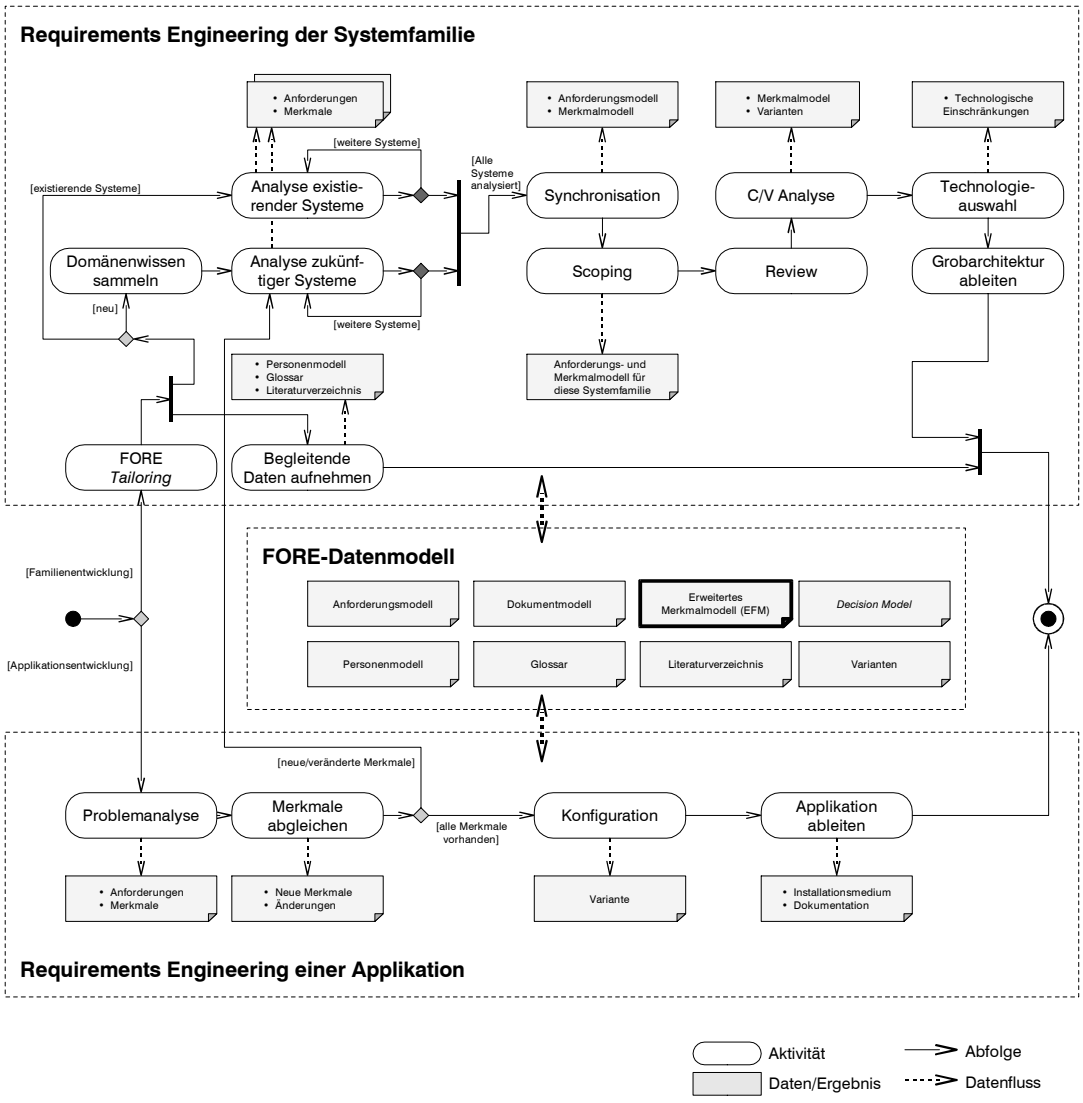


Abbildung 4.27: Iterativer FORE-Entwicklungsprozess

Am Beginn der Entwicklung einer Systemfamilie stehen strategische Entscheidungen, die letztlich auf wirtschaftlichen Gegebenheiten aufbauen. Als wichtigste Größe ist dabei die Zahl von mindestens vier abzuleitenden Varianten zu nennen, die sich aus Berechnungen mit dem *Constructive Cost Model* (COCOMO) ergeben und in [Klug 2003] detailliert dargelegt sind. Mit den bereits in Kapitel 3 erläuterten Einschränkungen der Systemfamilienentwicklung ergeben sich für FORE folgende

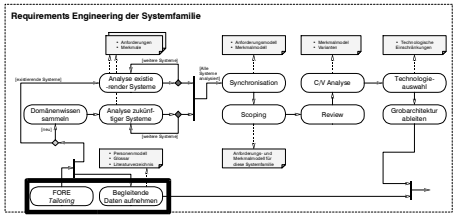
Randbedingungen, die während der gesamten Entwicklung gelten und Abbruchkriterien für die Systemfamilienentwicklung darstellen. Wird ein Kriterium verletzt, muss die Entwicklung der Systemfamilie dringend überdacht werden.

1. Systemfamilien sind nur in Bereichen sinnvoll, die eine individualisierte Massenproduktion zum Ziel haben, wobei Massenproduktion im Sinne von Punkt 2 zu verstehen ist.
2. Die anfänglich investierten Mehrkosten für die Entwicklung einer Systemfamilie amortisieren sich erst nach der vierten verkauften Applikation.
3. Für eine marktgerechte Systemfamilienentwicklung werden drei Struktureinheiten empfohlen, die jeweils für die Familienentwicklung, Komponentenverwaltung und Applikationsentwicklung verantwortlich sind. Stehen in einer Firma weniger als drei Entwickler zur Verfügung, müssen einzelne Entwickler mehrere Rollen gleichzeitig übernehmen.

Die folgenden Unterkapitel beschreiben jeweils die Teilschritte des FORE-Entwicklungsprozesses. Es handelt sich um einen iterativ zu bearbeitenden Prozess. In Abbildung 4.27 ist der gesamte Prozess dargestellt, wobei sich die einzelnen Iterationen an den jeweils zu entwickelnden Applikationen orientieren. Der dargestellte Weg zwischen den Prozessschritten stellt damit eine Richtung dar. Die Teilschritte können bei Fehlern wiederholt abgearbeitet werden und zwischen den Teilschritten kann hin und her gewechselt werden.

Zunächst werden die im oberen Teil der Abbildung enthaltenen Schritte in Kapitel 4.3.1 erläutert, wobei es sich um das *Requirements Engineering* der Systemfamilie handelt. Daran schließen die Schritte des *Requirements Engineering* einer Applikation in Kapitel 4.3.2 an. Zur Orientierung für den Leser, wird Abbildung 4.27 in jedem Unterkapitel verkleinert dargestellt, wobei der jeweils erläuterte Prozessschritt hervorgehoben ist. Das FORE-Datenmodell wird in Kapitel 4.4 beschrieben.

4.3.1 Requirements Engineering der Systemfamilie



Die FORE-Methode ist in einzelne Prozessschritte eingeteilt, die jeweils Eingabedaten in bestimmte Ausgabedaten umwandeln. Im ersten *Tailoring*-Schritt muss ein eventuell vorhandener, firmeneigener Standard für Spezifikationsdokumente, durch ein XML-Schema beschrieben werden. Dabei ist die grundsätzliche Struktur der Kapitel von Bedeutung. Sie wird mit Hilfe der im *DocBook*-Standard festgelegten Elemente beschrieben und kann in der Folge von den im weiteren Verlauf der Entwicklung entstehenden Informationselementen referenziert werden. FORE bietet eine auf *Volere* aufbauende Kapitelstruktur für die Systemfamilienentwicklung an, die bereits in das Datenmodell integriert ist und genutzt werden kann, falls kein eigener Standard vorhanden ist.

Die erste Entscheidung im Rahmen einer Systemfamilienentwicklung bezieht sich auf die Frage, ob es sich um eine völlig neue Systemfamilie in einer bestimmten Anwendungsdomäne handelt oder ob eine Umwandlung existierender Systeme hin zu einer Systemfamilie angestrebt wird.

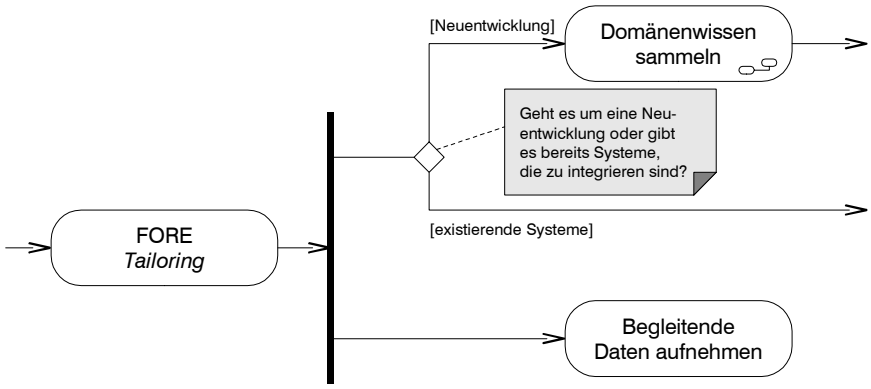
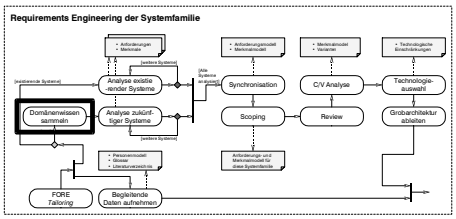


Abbildung 4.28: Initialisierung von FORE

In Abbildung 4.28 ist der Beginn der *Requirements-Engineering*-Phase der Systemfamilienentwicklung dargestellt. Parallel zu dem *Tailoring* können zu jedem Zeitpunkt der Entwicklung begleitende Daten aufgenommen werden. Dies sind Glossareinträge, Literaturverzeichniseinträge oder Personen, die in das Personenmodell aufgenommen werden.

Domänenwissen sammeln



Soll eine Systemfamilie entwickelt werden, ohne dass die Entwicklermannschaft auf die Erfahrungen bereits abgeschlossener Projekte zurückgreifen kann oder soll ein bereits bestehendes Grundwissen über die Domäne erweitert werden, steht als erste Aktivität die Akquisition von Domänenwissen an. Wie in Abbildung 4.29 dargestellt, ist dies ein Lern- und Datenerhebungsschritt, in dessen Verlauf sich alle Entwickler mit den Besonderheiten der Domäne vertraut machen müssen. Zur Erlangung von Wissen sind das Studium der einschlägigen Literatur, entsprechende Kurse oder *Workshops* und der Besuch angepasster Lehrveranstaltungen sinnvoll. Darüber hinaus kann die Lernphase verkürzt werden, indem die Domäne in mehrere Teilgebiete untergliedert wird, die jeweils von einem Entwickler studiert werden. Im Rahmen von periodischen Treffen können die einzelnen Wissensgebiete in Form von kurzen *Workshops* vorgestellt und anderen Entwicklern effizient vermittelt werden.

Besonders hervorzuheben sind in diesem Entwicklungsschritt die speziellen Eigenschaften der Systemfamilienentwicklung. Hier ist es notwendig, dass alle an der Entwicklung beteiligten Personen die Hintergründe der Systemfamilienentwicklung kennen. Alle müssen mit den Zielen der Entwicklung von Systemfamilien vertraut sein, da dieses Wissen für den Erfolg des Projektes von entscheidender Bedeutung ist. Die für FORE wichtige Merkmalmodellierung muss anhand eines kleineren Beispielprojektes erlernt werden. Die Abstraktion von Anforderungen durch Merkmale, die Nutzung der von FORE definierten Beziehungen von Merkmalmodellen und die Konfiguration eines Systems auf Basis der Merkmale sind die Ziele des Beispielprojektes.

Soll eine Systemfamilie zur Unterstützung betrieblicher Abläufe entstehen, so ist die Analyse der bei diesen Prozessen genutzten Dokumente sehr hilfreich. In FORE müssen schon während der Dokumentanalyse Ähnlichkeiten berücksichtigt werden. Der Vergleich von Formularen desselben Betriebes zeigt Ähnlichkeiten auf, die potentiell zu wiederverwendbaren Komponenten innerhalb einer Variante führen. Werden Dokumente mehrerer Betriebe aus der Domäne der Systemfamilie verglichen, können so erarbeitete Komponenten in der gesamten Systemfamilie und damit in vielen abgeleiteten Applikationen wiederverwendet werden. Dabei müssen ähnliche Begriffe mit Hilfe des Glossars identifiziert werden, um Formularfelder mit gleichem Inhalt zu lokalisieren und in dem Familienmodell zu berücksichtigen.

Im Rahmen der Befragung (*Interview*) von Projektbeteiligten, die am Einsatz des Produktes interessiert sind, wird von FORE für Systemfamilien ein neuer Aspekt in den Fragenkatalog aufgenommen. Neben der Erhebung von Anforderungen an ein Produkt müssen Fragen zu Variationsmöglichkeiten ergänzt werden. Es kommt darauf an, mögliche Abwandlungen von Produkten zu erkennen und den *Interview*-Partner nach dem Wert dieser Varianten für ihn zu befragen und seine Bereitschaft zu prüfen, derartige Varianten käuflich zu erwerben. Schließlich werden die finanziellen Möglichkeiten eines Kunden seinen eventuellen Wunsch nach Funktionsvielfalt in dem späteren Produkt bremsen. Variationsmöglichkeiten werden als wichtige Basis jeder Systemfamilie festgehalten, um später auch neuen Kunden offeriert zu werden.

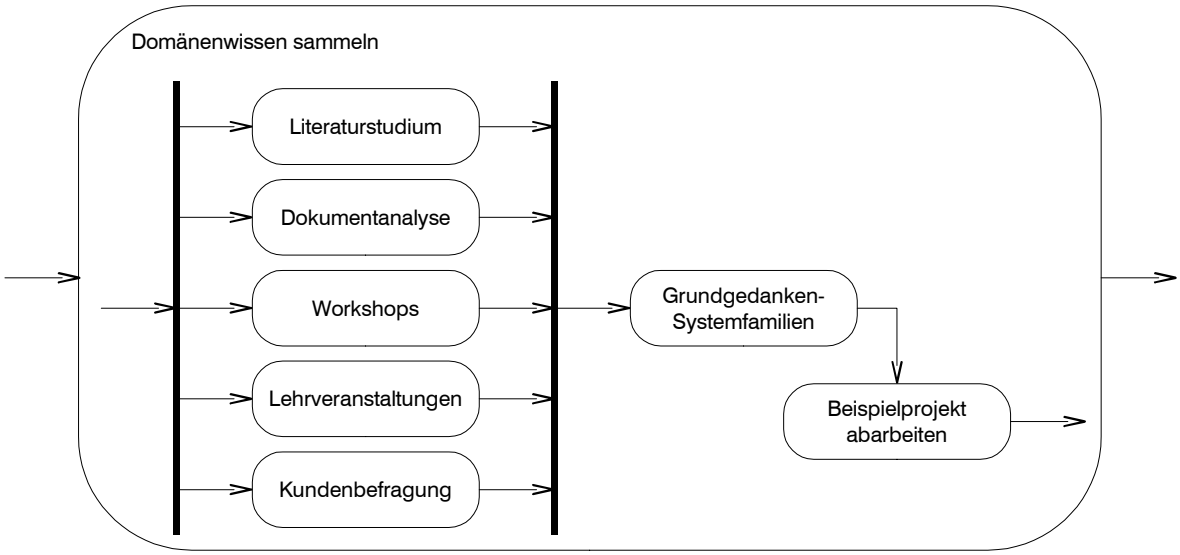


Abbildung 4.29: Domänenwissen sammeln

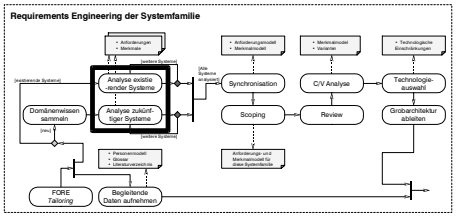
Die Erfahrungen des Videoprojektes haben gezeigt, dass sich Entwickler im Rahmen von kleineren Beispielprojekten das Wissen über Systemfamilienentwicklung am besten aneignen können. Hierbei sind bekannte Domänen vorzuziehen, um die Motivation auf die Systemfamilienentwicklung und nicht auf die Akquisition von Domänenwissen zu fokussieren. Darüber hinaus können die Entwickler durch dieses Projekt ihr Wissen im Bereich der Komponentenentwicklung, wie COM, *Templates,* JavaBeans oder auch CORBA festigen, insbesondere bei den Techniken, die später eingesetzt werden sollen.

Alle Begriffe, die unklar oder missverständlich sind, werden in ein Glossar integriert, welches Begriffe bzw. Fachbegriffe zusammen mit einer Erklärung enthält. Dieses Glossar wird im Verlauf der Entwicklung vervollständigt und kann allen Stakeholdern bei Verständnisfragen weiterhelfen. Bereits auf dem Markt befindliche, elektronische Fachwörterbücher und Lexika bieten einen sehr guten Kompromiss zwischen Aufwand für die Erstellung eines Glossars und dem daraus gezogenen Nut-

zen. Neben der reinen Erläuterung von Begriffen werden auch jeweilige Synonyme festgehalten. In den Formulierungen aller Dokumente werden die Hauptbegriffe genutzt, wobei der jeweilige Eintrag des Glossars referenziert wird. Jeder Leser kann damit ein Dokument schnell lesen und verstehen, da er Unklarheiten anhand der eventuell von ihm präferierten Synonyme und der Erklärungen im Glossar sofort beseitigen kann.

Das Ergebnis dieser Entwicklungsphase ist die Angleichung der unterschiedlichen Wissenstände innerhalb der Entwicklermannschaft und die Einarbeitung in die Domäne der Systemfamilie.

Analyse existierender und zukünftiger Systeme



Mit der Domäne vertraute Entwickler können in diesem zweiten Schritt der *Requirements-Engineering-Phase* mit der Erhebung der Anforderungen beginnen. Sind bereits existierende Produkte aus der Domäne der Systemfamilie vorhanden, müssen diese zunächst analysiert werden. Bei Produkten ohne Dokumentation müssen mit Hilfe von *Reverse-Engineering*-Methoden und Befragungen der ursprünglichen Entwickler die Spezifikationsdokumente so gut wie möglich wieder erarbeitet werden. Basierend auf diesen Dokumenten werden Anforderungen und Merkmale der Produkte mit Hilfe der Anforderungskarten und Merkmalkarten aufgenommen.

Wird eine Systemfamilie ohne bereits existierende Produkte entwickelt oder gibt es noch keine konkreten Kunden, müssen alle potentiell möglichen Produkte in Erwägung gezogen werden. Zusammengefasst in Abbildung 4.30, werden durch Marktanalysen die Bedürfnisse der avisierten Käuferschicht und die innerhalb dieses Marktsegmentes herrschenden Bedingungen analysiert. Umfragen führen zu einem aktuellen Meinungsbild der Kunden und zukünftigen Käufer. Im Unternehmen selbst müssen alle Projektbeteiligten versuchen, sich in einen zukünftigen Kunden hineinzuversetzen, um dessen wahrscheinlichen Bedürfnisse im Rahmen von *Brainstorming* Sitzungen festzuhalten. Im Gegensatz zur Einzelsystementwicklung wird in FORE jedes zukünftige Produkt in die Liste der Varianten der Familie aufgenommen und über seine Anforderungen sowie Merkmale beschrieben. Dazu dienen die in Kapitel 4.1 dargestellten Beziehungen. Die Anforderungen werden mit Merkmalen verknüpft und für jedes Merkmal wird die Variante festgehalten, der es später zugeordnet werden soll.

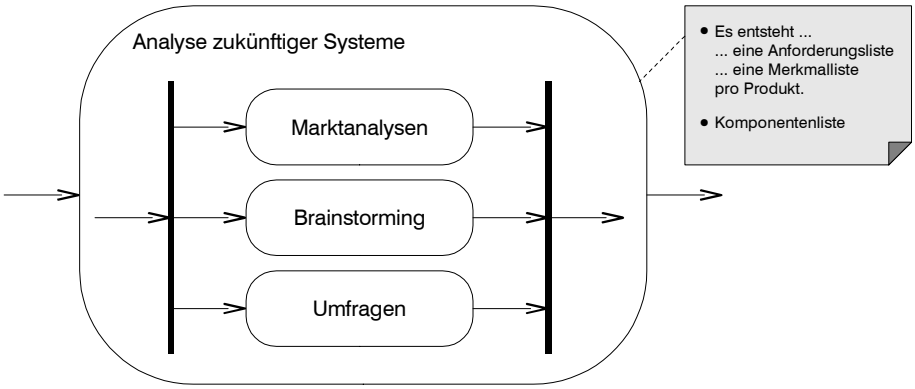


Abbildung 4.30: Analyse zukünftiger Systeme

Den Entwicklern bekannte Komponenten werden an dieser Stelle bereits in eine Komponentenliste aufgenommen, um für die weiteren Entwicklungsschritte zur Verfügung zu stehen.

In Abbildung 4.6 sind die Teilprozesse der Analyse existierender Produkte dargestellt. Die jeweiligen Schritte können nur durchgeführt werden, sofern die dazu notwendige Information in Form einer Dokumentation vorhanden ist. Fehlt jegliche Dokumentation und ist auch der Quellcode nicht vorhanden, so können nur aufwändige Arbeiten mit *Reverse-Engineering*-Methoden die interne Struktur einer Applikation wiedergewinnen. Dabei muss eine Aufwandsschätzung zeigen, ob sich derartige Arbeiten lohnen. Ist eine Dokumentation vorhanden, wird zunächst die Architektur der jeweiligen Produkte analysiert. Im Rahmen des Videoprojektes wurde in diesem Schritt eine *Plug-in*-Architektur festgestellt, die sich hervorragend als Referenzarchitektur der Systemfamilie eignet. Damit konnte auf einem der analysierten Produkte aufgebaut werden, was Ressourcen eingespart hat. Steht eine Dokumentation nicht zur Verfügung, kann sie durch *Interviews* mit den für das jeweilige Produkt verantwortlichen Entwicklern erstellt werden. Entziehen sich die ehemaligen Entwickler dem Zugriff, muss der Quellcode der Produkte analysiert werden, um die ursprüngliche Architektur durch eine neu zu erstellende Dokumentation zu beschreiben.

Wichtig für eine Systemfamilie sind die jeweils genutzten Soft- und Hardwarekomponenten. Entweder stammen sie aus den existierenden Applikationen oder werden von Drittanbietern eingekauft. In jedem Fall müssen alle Komponenten umfassend dokumentiert sein und in einen Katalog integriert werden, um den Entwicklern später zur Verfügung zu stehen. Die Analyse der bereits vorhandenen und in existierenden Applikationen genutzten Komponenten birgt ein hohes Potential für die Wiederverwendung. In FORE wird eine Komponentenanalyse existierender Applikationen betrieben, indem für jede Komponente eine Reihe von Daten erhoben wird, die in dem Komponentenmodell abgelegt werden und der späteren Nutzung in der Systemfamilie zur Verfügung stehen. Für jede Komponente sind die bereits in Kapitel 4.1 beschriebenen Anteile in das Datenmodell aufzu-

nehmen. Diese Informationen finden sich auch in der Dokumentation wieder, welche als FORE-Dokumentstruktur bereits in Kapitel 4.1.2 beschrieben wurde.

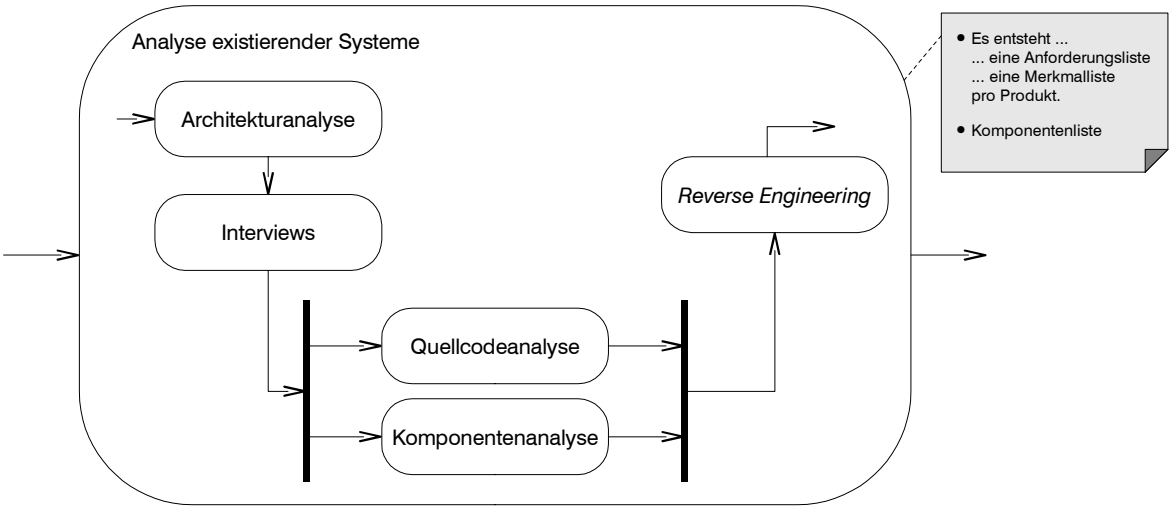
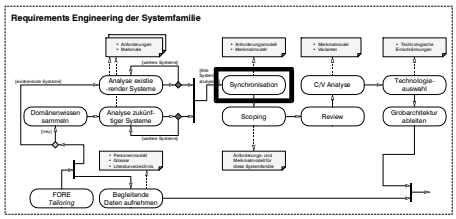


Abbildung 4.31: Analyse existierender Systeme

Alle an der Entwicklung beteiligten Personen werden in dem Personenmodell zusammengefasst, bzw. sukzessive hinzugefügt. Jede der erhobenen Anforderungen muss mit einem Autor in dem Personenmodell durch die entsprechende „Autor“-Beziehung verbunden werden. Darüber hinaus wird eine Anforderung über die „Dokument“-Beziehung in die am Anfang des Prozesses definierte Dokumentstruktur eingebunden

Die grau unterlegten Teile in Abbildung 4.30 und Abbildung 4.6 stellen die Ergebnisse der Prozessschritte dar. Am Ende dieser Schritte stehen Anforderungen und Merkmale in strukturierten Listen zur Verfügung. Dabei entscheidet die Größe der Entwicklermannschaft über die Arbeitsweise. Stehen ausreichend Personen für diesen Schritt zur Verfügung, wird pro Produkt eine Gruppe gebildet, so dass ebenfalls pro Produkt zwei Listen entstehen. Eigene Erfahrungen zeigen, dass eine Analysegruppe aus 3 bis 5 Personen bestehen sollte, um den Koordinations- und Kommunikationsaufwand in vertretbaren Grenzen halten zu können. Stehen nur wenige Entwickler zur Verfügung, müssen die Untersuchungen auf Kosten der Entwicklungszeit nacheinander abgearbeitet werden.

Synchronisation



Während der Analysephase wurden viele Informationen erarbeitet, deren Konsolidierung an dieser Stelle ganz wichtig ist, da sowohl Anforderungen als auch Merkmale bezogen auf jeweils ein Produkt erhoben wurden. In diesem Schritt werden die Informationen zu einem Gesamtmodell zusammengefasst, um Redundanzen zu beseitigen. Es entsteht eine erste Version des FORE-Datenmodells, in dem Anforderungen und Merkmale miteinander in Beziehung stehen.

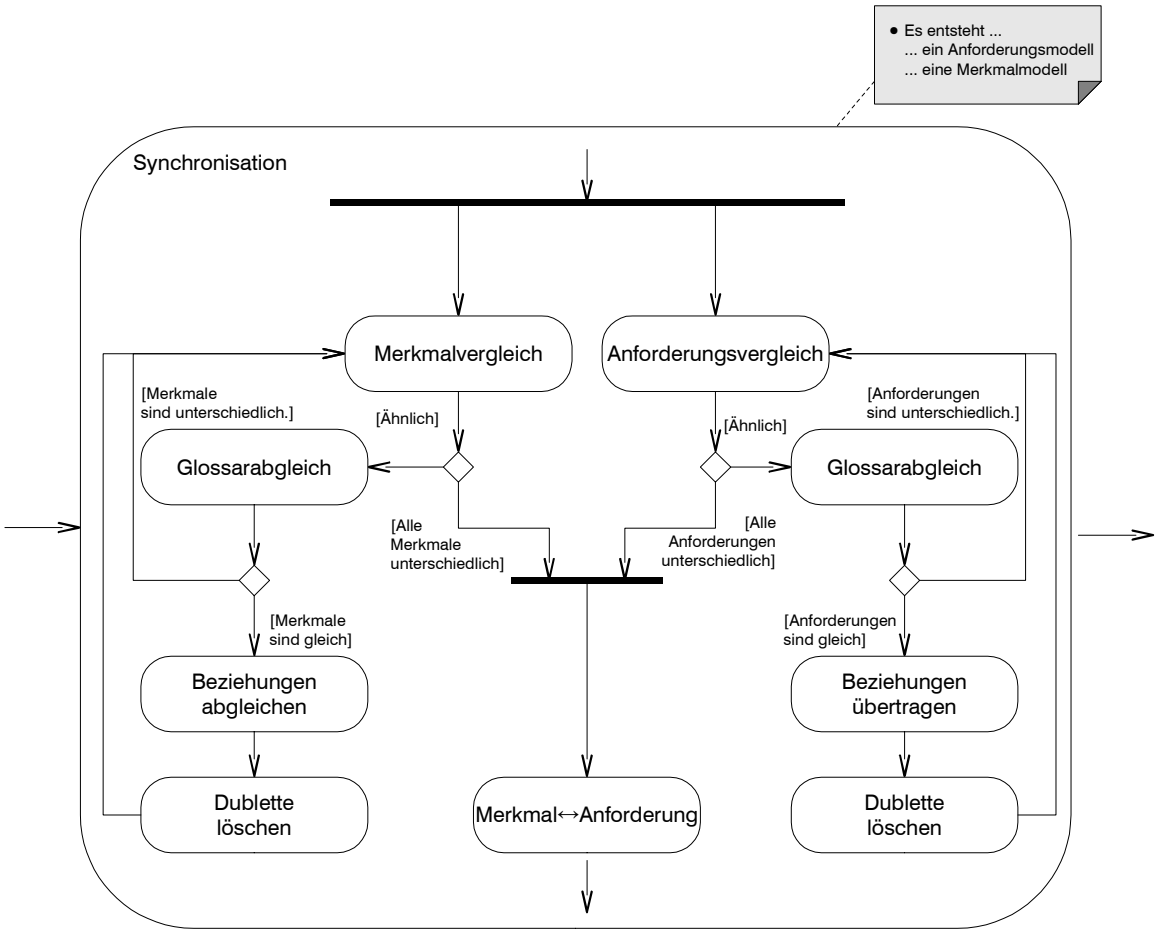


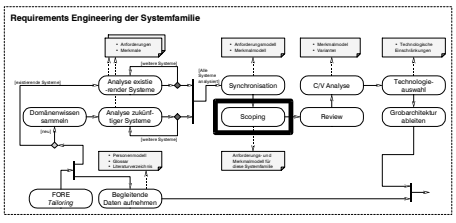
Abbildung 4.32: Synchronisation

Wie in Abbildung 4.32 zu sehen, werden nach Klärung der Begrifflichkeiten in den Dokumenten die erhobenen Anforderungen und Merkmale jeweils auf Dubletten untersucht. Durch die Verwendung des Glossars und der Synonymliste können Ähnlichkeiten automatisiert gefunden werden. Ein endgültiger semantischer Vergleich wird jedoch dem Entwickler überlassen. Er hat dann darüber zu befinden, ob zwei Anforderungen bzw. Merkmale gleich sind oder nicht. Bevor ein Element gelöscht werden kann, müssen eventuell schon vorhandene FORE-Beziehungen in die Dublette übertragen werden. Mit Hilfe eines Werkzeugs lässt sich dieser Schritt automatisieren, ansonsten müssen alle Beziehungen manuell von einem zum anderen Element übertragen werden. Nicht automatisiert werden können die Hierarchiebeziehungen. Hier muss der Entwickler aus den vorhandenen Anforderungs- und Merkmalhierarchien jeweils eine sinnvolle Hierarchie erarbeiten, in welche die Elemente der einzelnen Modelle eingeordnet werden. Schließlich kann das Duplikat gelöscht werden, womit eine redundante Information im Modell beseitigt wurde.

Bei der Erhebung von Anforderungen und Merkmalen in direktem Kundenkontakt wird zunächst „die Sprache des Kunden“ genutzt, um die Informationen festzuhalten. Diese Sprache entstammt der Domäne des Kunden und enthält daher viele Begriffe, die zu Inkonsistenzen führen können, da sie leicht missverstanden werden. Das bereits am Beginn der Entwicklung begonnene Glossar wird nun genutzt und weiterentwickelt. Unklare, mehrdeutige oder missverständliche Begriffe müssen identifiziert und im Dialog mit dem jeweiligen Kunden bzw. Autor geklärt und in das Glossar aufgenommen werden, dazu wird die Beziehung jeder Anforderung zum betreffenden Autor genutzt. Unterschiedliche Begriffe mit gleicher Bedeutung werden dabei ebenfalls vermerkt, so dass das Glossar mit Synonyma angereichert wird.

Im letzten Schritt werden die Referenzen von einem Merkmal zu Anforderungen geprüft. Strukturierungsmerkmale haben keine Beziehungen zu Anforderungen, während alle anderen Merkmale diese Beziehungen haben müssen. Selbst Merkmale, die aus Komponenten von Drittanbietern stammen, müssen durch Anforderungen beschrieben und entsprechend referenziert werden.

Scoping

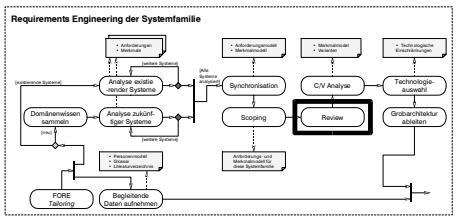


Im Rahmen der Analyse wurden aus Varianten viele Anforderungen und Merkmale erarbeitet und schließlich in einem Modell zusammengefasst. Für jede Variante mit zugehörigen Anforderungen

und Merkmalen muss geprüft werden, ob sich die Variante noch innerhalb der Grenzen der Systemfamilie befindet. Diese Entscheidung hängt dabei von der Ausrichtung des Softwareunternehmens und den zur Verfügung stehenden Ressourcen ab. FORE nutzt zur Beschreibung der Systemfamilie das Merkmalmodell. In diesem als *Scoping* bezeichneten Schritt, werden diejenigen Merkmale aus diesem Modell gewählt, die im weiteren Verlauf der Entwicklung berücksichtigt werden. Im Rahmen des digitalen Videoprojektes mussten aus der Vielzahl der Merkmale eine Untermenge gewählt werden, da durch die studentischen Arbeiten an diesem Projekt nur begrenzte Entwicklungsressourcen zur Verfügung standen. Alle ursprünglichen Merkmale des FORE-Datenmodells bleiben jedoch bestehen, sofern sie sich innerhalb der für die Systemfamilie definierten Domänengrenzen befinden. Nur Merkmale, die thematisch überhaupt nicht zur Systemfamilie passen und auch nur unter großen Anstrengungen in die Systemfamilienarchitektur integrierbar sind, werden ersatzlos aus dem FORE-Datenmodell gestrichen.

Sollen Änderungswünsche von Kunden berücksichtigt werden, steht die Architektur der Systemfamilie bereits fest, so dass nunmehr dem *Scoping* eine weitere Bedeutung zukommt. Veränderungen an Anforderungen und Merkmalen können leicht eine Veränderung der Architektur nach sich ziehen. Je nach Bedeutung eines Kunden für den Systemfamilienhersteller und Bereitschaft des Kunden eventuell hohe Entwicklungskosten zu tragen, können neue Variante zugunsten einer stabilen Referenzarchitektur der Systemfamilie abgelehnt werden oder trotz des eventuell hohen Aufwandes integriert werden. Sehr wichtig ist jedoch, den Einfluss einer Veränderung der Referenzarchitektur einer Systemfamilie zu berücksichtigen. Auch nach einer Veränderung müssen alle bis dahin abgeleiteten Varianten weiterhin gültig und ableitbar bleiben. Im schlimmsten Fall muss die Architektur völlig neu strukturiert werden, was natürlich die Kosten in die Regionen einer Neuentwicklung treibt.

Reviews



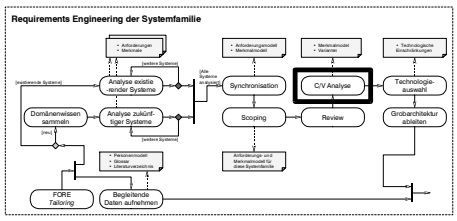
FORE nutzt *Reviews*, die auf den in Kapitel 3 beschriebenen basieren, um die Analyseergebnisse zu prüfen. Dabei müssen für Systemfamilien spezifische Fragestellungen geklärt werden, wobei insbesondere das Merkmalmodell einer Überprüfung unterzogen wird. Die Konsistenz der Familie basiert, wie schon gezeigt, auf den modellierten Beziehungen im Merkmalmodell. In einem *Review*

wird geprüft, ob die Informationen im FORE-Datenmodell korrekt und komplett aufgenommen sind. Neben den Informationen aus dem Datenmodell müssen diverse Textpassagen für das Spezifikationsdokument geschrieben werden, um es leicht verständlich und lesbar zu halten. Hierbei ist speziell das erste und sechste Kapitel des FORE-Spezifikationsdokumentes aus Abbildung 4.31 auf Seite 134 von Bedeutung, da der Großteil der Inhalte nicht im Datenmodell enthalten ist. Einleitende Worte, der Sinn und Zweck des Systems, Kodiervorgaben, Risiken oder die Benutzerdokumentation sind in Prosa zu verfassende Texte, die für das Verständnis des Systems entscheidend sind, ohne jedoch im Datenmodell enthalten zu sein. Diese Textpassagen müssen überprüft und korrigiert werden. Weitere, eventuell fehlende und für das Verständnis wichtige Beschreibungen müssen während des *Review*-Prozesses lokalisiert und von der Entwicklermannschaft eingefordert werden.

Für das Datenmodell muss überprüft werden, ob alle Informationen der Anforderungs- und Merkmalkarten vorhanden sind. Die bereits vorgestellten Karten zur Erhebung von Anforderungen und Merkmalen sind im Datenmodell direkt abgebildet, so dass während eines *Reviews* fehlende Einträge gefunden werden können. Ebenso werden missverständliche Einträge markiert, um eventuell Anpassungen einzuleiten.

Den *Reviewern* bekannte, aber im Personenmodell nicht vorhandene *Stakeholder* werden zur Diskussion gestellt und eventuell in das Personenmodell aufgenommen. Im Gegensatz dazu werden Personen, die von keinem Modellelement referenziert werden, zunächst vermerkt und anschließend zur Diskussion gestellt. Es ist zu entscheiden, ob es sich um für das Projekt wichtige, stille *Stakeholder* handelt oder ob diese Personen für das Projekt nicht mehr bedeutsam sind und damit aus dem Personenmodell gestrichen werden können.

Commonality / *Variability*-Analyse

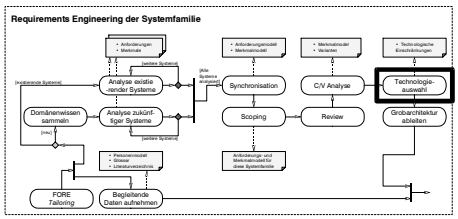


Mit dem Wissen der Domäne und den Informationen des FORE-Datenmodells wird in diesem Schritt entschieden, welche der Merkmale zum Kern der Referenzarchitektur gehören und welche Merkmale variabel bleiben.

Zunächst werden sich die grundlegenden Konzepte der Domäne mit hoher Wahrscheinlichkeit im Kern einer Familie wieder finden, wie in [Copl 1999] beschrieben. In FORE wurde jedoch bereits bei der Analyse der existierenden Applikationen deren Architektur analysiert und auf die eventuelle

Nutzung als Referenzarchitektur der Systemfamilie hin geprüft. In enger Verbindung mit dem nächsten Schritt „Technologieauswahl“ wird nun eine grundlegende Idee für die Referenzarchitektur der Familie erarbeitet. Im Rahmen des digitalen Videoprojektes sind dabei eine *Plug-in*-Architektur für das Basissystem, *Frameworks* bei den Videoverarbeitungswerkzeugen, Architekturen zur Nutzung von COM-Komponenten (*Component Object Model*) und Architekturen zur Nutzung von *JavaBeans* aufgetreten. Basierend auf einer Idee für die Architektur der Familie und den zur Verfügung stehenden Komponenten, die ebenfalls bereits während der Analyseschritte aufgenommen wurden, können die Merkmale in optionale und obligatorische eingeteilt werden. Bei dieser Einteilung ist die Beziehung der Merkmale zu den bereits vorhandenen Komponenten hilfreich.

Technologieauswahl



Die Technologie, die später zur Realisierung der Systemfamilie genutzt wird, hat Einfluss auf das Merkmalmodell der Familie. Die technologisch bedingten Einschränkungen spiegeln sich direkt in den Beziehungen der Merkmale wider. Im Folgenden sind unterschiedliche Technologien zur Realisierung von Variabilitäten aufgeführt.

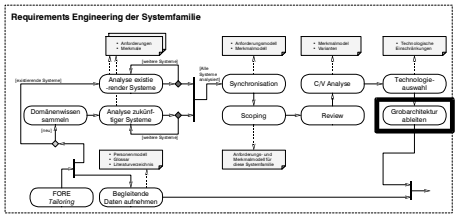
1. Die *Plug-in*-Technologie
   Der Kern der Familie stellt eine so genannte *Plug-in*-API (*Application Programming Interface*) zur Verfügung. Die Variabilitäten der Familie werden nun als Komponenten realisiert, die über die *Plug-in*-API mit dem Kern in Verbindung stehen. Im FORE-Datenmodell werden die entsprechenden Merkmale mit den Komponenten verbunden und es wird eine Schnittstellen-Beziehung genutzt, um die Übereinstimmung der Komponentenschnittstelle mit der *Plug-in*-API überprüfen zu können.
2. Komponentenmodelle wie COM oder JavaBeans
   Hierbei werden Nutzung, Integration und Erstellung von Komponenten durch das entsprechende Modell vorgegeben. Die Architektur der Familie ist flexibel definierbar, jedoch sind Umgebungsparameter durch die Komponentenmodelle vorgegeben. COM erfordert eine *Microsoft Windows* Plattform, während JavaBeans plattformunabhängig sind, allerdings Leistungsgrenzen haben. Dies wird in Form von Parame-

tern im Merkmalmodell hinterlegt, die mathematisch zueinander in Beziehungen stehen.

3. Generieren von Applikationen mit dem *Hyperspace*-Ansatz [Tarr 1999]
   Die Architektur des Systems wird in so genannte *Hyperspaces* eingeteilt, auf die sich wiederum die Merkmale der Familie aufteilen. Alle optionalen Merkmale werden durch eigene *Hyperspaces* realisiert. Während der Generierung werden alle für eine Variante relevanten *Hyperspaces* zusammen „komponiert". Zwischen den *Hyperspaces* müssen vorher Kompositionsregeln definiert worden sein, die im Generierungsschritt genutzt werden.
   Grundsätzlich hat dieser Ansatz kaum Einfluss auf das Merkmalmodell, da prinzipiell alles durch *Hyperspaces* realisierbar ist. Allerdings führen Änderungen am Merkmalmodell und damit auch am konkreten System sehr schnell zu Nebeneffekten, da die Architektur der *Hyperspaces* mitsamt ihren Kompositionsregeln bekannt sein muss. Ergebnisse praktischer Projekte zeigten, dass selbst erfahrene Entwickler Probleme haben, den Überblick bei größeren Systemen zu behalten.
4. Skriptsprachen
   Ein System mit einer Basisfunktionalität kann durch Skripte erweitert werden. Dabei stellt das System eine eigene Skriptsprache zur Verfügung, in der alle variablen Anteile implementiert werden müssen. Alle optionalen Merkmale wären damit den Grenzen der Skriptsprache unterworfen.
5. Konfigurationsdateien
   Verschiedene Einstellung für Komponenten lassen sich in Konfigurationsdateien hinterlegen. Diese Dateien werden beim Programmstart eingelesen, können aber je nach Implementierung auch während des Programmlaufs bei Bedarf eingelesen werden. Fast alle Komponenten des digitalen Videoprojektes nutzen diese Technologie, um Einstellungen beim Systemstart zu laden und sowohl das Verhalten als auch das Aussehen des Systems zu beeinflussen. In einer Komponente können dabei mehrere Merkmale implementiert sein, so dass sich die Abhängigkeiten der Merkmale direkt aus den Gegebenheiten der Komponente ableiten. Des Weiteren gibt es Überschneidungen bei der Funktionalität der existierenden Komponenten, wodurch Merkmale in mehreren Komponenten realisiert sind. Zwischen solchen Merkmalen und Komponenten bestehen Beziehungen, welche die richtige Komponente, bei einer gegebenen Merkmalauswahl, in das entstehende System integrieren.
6. Bedingte Compilierung
   Eine Merkmalauswahl kann zur Compilezeit durch Präprozessoranweisungen berücksichtigt werden. Durch diese bedingte Compilierung können beliebige Codestü-

cke weggelassen oder in das System hineincompiliert werden. Das Problem bei dieser Vorgehensweise ist, dass sich die optionalen Merkmale auf den gesamten Code aufteilen und damit sehr schwer wartbar werden.
Auf das Merkmalmodell hat diese Art der Realisierung von Variabilität keinen Einfluss, da programmiertechnisch alles machbar ist. Allerdings ist der Aufwand der Realisierung unter Umständen zu hoch.

Grobe Architektur ableiten



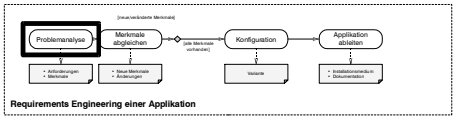
Mit dem Wissen um die verwendeten Komponenten und die zu nutzende Technologie wird an dieser Stelle eine grobe Architektur entwickelt, die schon eine frühe Fassung der Referenzarchitektur der Systemfamilie darstellen kann. In dieser Architektur sind die existierenden Komponenten und deren Beziehungen untereinander modelliert. Die Referenzarchitektur der Familie wird mit Hilfe der Strukturdiagramme der UML notiert, wobei die betroffenen Kernmerkmale mit den Architekturelementen verbunden werden müssen. Die variablen Anteile können durch das Komponentendiagramm der UML dargestellt werden. Dabei sind alle vorhandenen Komponenten, die aus existierenden Applikationen stammen, ersichtlich. Weitere Modellelemente der Architektur müssen damit neu entwickelt werden, wodurch sich der Aufwand für die Entwicklung der Systemfamilie abschätzen lässt.

Mit diesem Prozessschritt endet die *Requirements-Engineering-Phase* für die Systemfamilie. Natürlich darf der schon erwähnte iterative Charakter des Prozesses nicht vergessen werden, so dass der gesamte Prozess, wie auch einzelne Schritte immer wieder ausgeführt werden können, falls Fehler auftreten sollten. Schließlich werden alle Informationen in das FORE-Datenmodell eingepflegt und stehen dem zweiten Teil des Prozesses, dem *Requirements Engineering* einer Applikation zur Verfügung. Ein Kunde kann die aufbereiteten Informationen der Systemfamilie nutzen, um ein für sich angepasstes System, eine personalisierte Applikation, abzuleiten.

4.3.2 Requirements Engineering einer Applikation

Dieser Entwicklungsschritt ist gekennzeichnet durch die Möglichkeit, die ein Kunde hat ein System aus vorhandenen Komponenten zusammenzusetzen. Wichtig ist hierbei, dass der Kunde das System basierend auf Merkmalen auswählt. Damit wird der Auswahlprozess des Kunden nicht mit technologischen oder informationstechnischen Details überfrachtet.

Problemanalyse



Requirements Engineering einer Applikation

Die Problemanalyse dient dazu festzustellen, ob es prinzipiell möglich ist, den Kunden mit der zur Verfügung stehenden Systemfamilie zufrieden zu stellen. Zunächst werden erste Anforderungen und Wünsche des Kunden aufgenommen. Dabei muss der Kunde auf die Systemfamilie mit ihren Merkmalen hingewiesen werden, um zu versuchen, so viele Merkmale wie möglich aus der Familie zu nutzen. Hier werden auch eventuell notwendige Erklärungen zu den Merkmalen durch einen Vertreter des Verkaufs gegeben. Dabei muss der Vertreter das Merkmalmodell genau kennen.

Dieser Schritt ist eng an den nächsten Schritt gekoppelt, da sich die Problemanalyse unter Zuhilfenahme des Merkmalmodells sehr kurz gestalten kann. Es geht dann nur noch um die Anforderungen, die nicht durch die Systemfamilie abgedeckt sind.

Am Beispiel des digitalen Videosystems wird hier der Weg zu einer Variante des Systems aufgezeigt. Der Kunde hat sich entschieden, ein sehr einfaches und damit auch kostengünstiges System zu erwerben. Er formuliert knapp seine Anforderungen, die in der folgenden Liste enthalten sind.

- Er will digitale Fernsehsendungen sehen und aufnehmen können.
- Er will DVDs abspielen und auf DVDs aufnehmen können.
- Er will Sendungen zeitversetzt sehen können.
- Er will zwei Kanäle gleichzeitig sehen können.

Der Verkäufer erkennt schnell, dass diese Anforderungen durch die im Merkmalmodell der Systemfamilie vorhandenen Merkmale abgedeckt werden und berät den Kunden weiter. Diese Beratung geschieht mit dem Wissen um das komplette Merkmalmodell, wobei dem Kunden weitere Merkmale erklärt und angeboten werden können, um das System sinnvoll zu ergänzen.

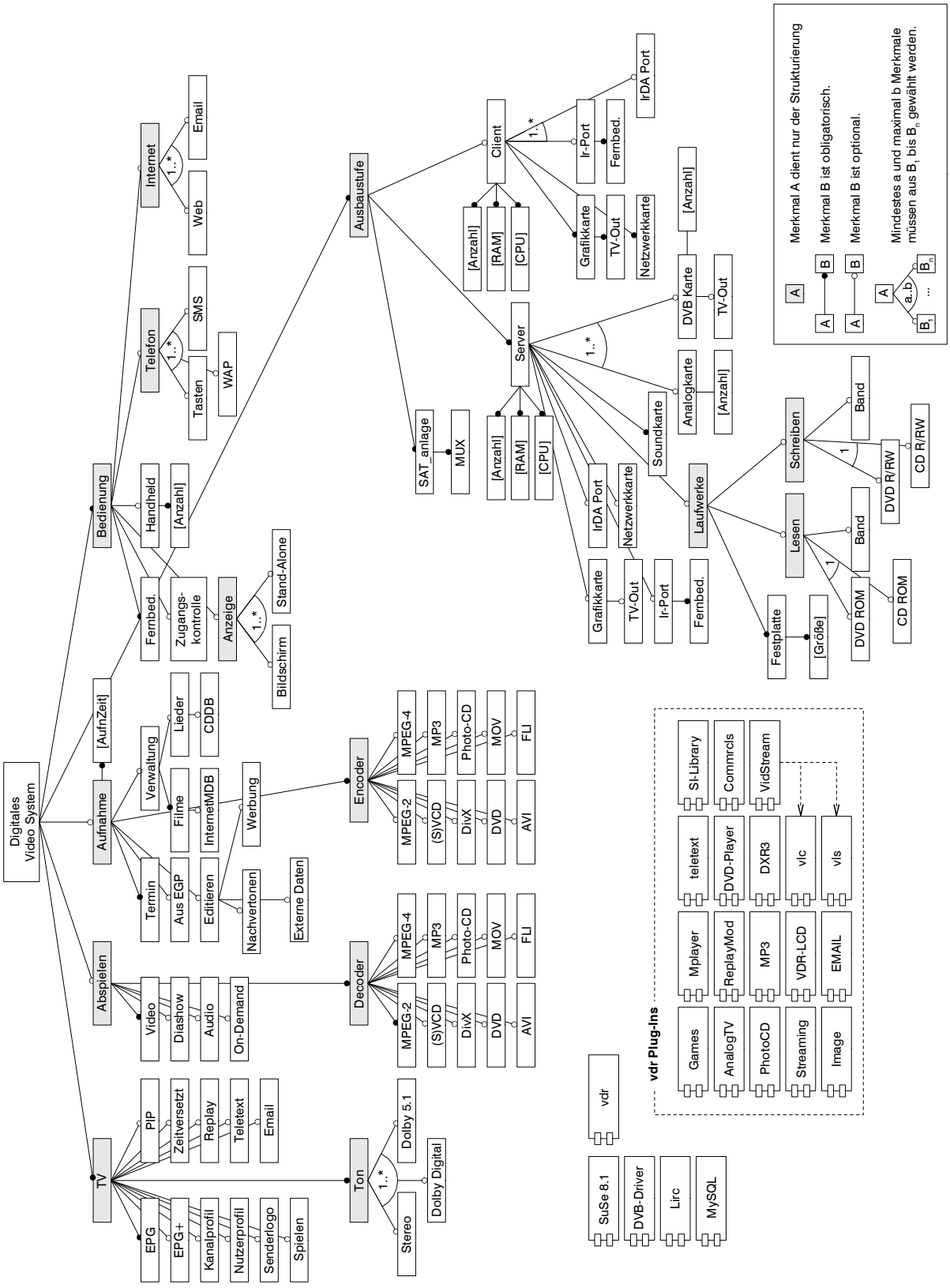


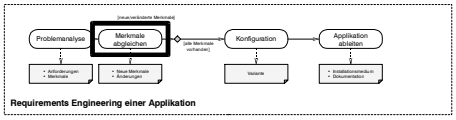
Abbildung 4.33: Merkmalmodell des digitalen Videoprojektes

Das gesamte Merkmalmodell ist in Abbildung 4.33 enthalten. Dabei sind zugunsten der Übersichtlichkeit die Beziehungen zwischen den Merkmalen weggelassen, jedoch in Anhang A enthalten. Nach kurzer Diskussion entschließt sich der Kunde, noch zwei weitere Anforderungen in das von ihm gewünschte System aufzunehmen.

- Er will die Lautstärkenunterschiede der verschiedenen Sender nicht mehr akzeptieren.
- Er will aufzunehmende Sendungen direkt aus einer elektronischen Programmzeitung auswählen.

In kurzer Zeit konnten die Wünsche des Kunden aufgenommen und erweitert werden. Dabei wurden die störenden Lautstärkeunterschiede durch die *recommend*-Beziehung zwischen dem Merkmal „TV“ und „Kanalprofil“ vom Verkäufer erwähnt und vom Kunden ausgewählt.

Merkmale abgleichen



Hier werden die erhobenen Kundenanforderungen und Merkmale mit denen der Systemfamilie abgeglichen. Es ist sinnvoll bei Merkmalen, die nicht in der vom Kunden gewünschten Form in der Familie vorhanden sind, einen Kompromiss anzustreben. Die Systemfamilie bringt für beide Seiten den größten Gewinn, wenn alle Merkmale durch die Familie abgedeckt werden können. Daher sollten auch ähnliche Merkmale im Verlauf des Auswahlprozesses berücksichtigt werden. Der Kunde hätte dann einen finanziellen Vorteil, da ein ähnliches Merkmal, dass 80% seiner Anforderungen erfüllt, erheblich billiger ist, als eine Neuentwicklung, welche die Anforderungen zu 100% erfüllt.

Schließlich werden die Merkmale bestimmt, die in der Familie vorhanden sind und die Merkmale, die neu in die Familie aufgenommen bzw. verändert werden müssten. Fehlen Merkmale, um einen Kunden zufrieden zu stellen, muss zunächst zurück in das *Requirements Engineering* der Systemfamilie verzweigt werden. In der „Analyse zukünftiger Systeme“ wird das neu zu integrierende System evaluiert. Es muss festgestellt werden, wie stark die neuen Anforderungen und damit die neuen Merkmale die Architektur der Familie beeinflussen, ob sich diese Neuerungen überhaupt mit der Referenzarchitektur vereinbaren lassen. Eine positive Entscheidung hat letztlich die Neuentwicklung von Komponenten zur Folge, die von den neuen Merkmalen des FORE-Datenmodells referenziert werden. Kann ein Kunde seine Anforderungen unter keinen Umständen zurückschrauben und kommt die Evaluierung der Neuerungen zu einem negativen Ergebnis, ist der Prozess hier zu Ende.

In allen anderen Fällen kommt der Merkmalabgleich, auch über den Umweg einer Erweiterung der Systemfamilie, schließlich zu einer vollständigen Abdeckung der Anforderungen des Kunden.

Der Abgleich der Merkmale wird für das Beispiel des digitalen Videosystems durch den Vergleich der Kundenanforderungen mit den Merkmalen und der anschließenden Auswahl der gewünschten Merkmale realisiert. In dem vorliegenden Fall werden die Anforderungen durch die Familie vollständig abgedeckt, so dass alle Merkmale bereits vorhanden sind.

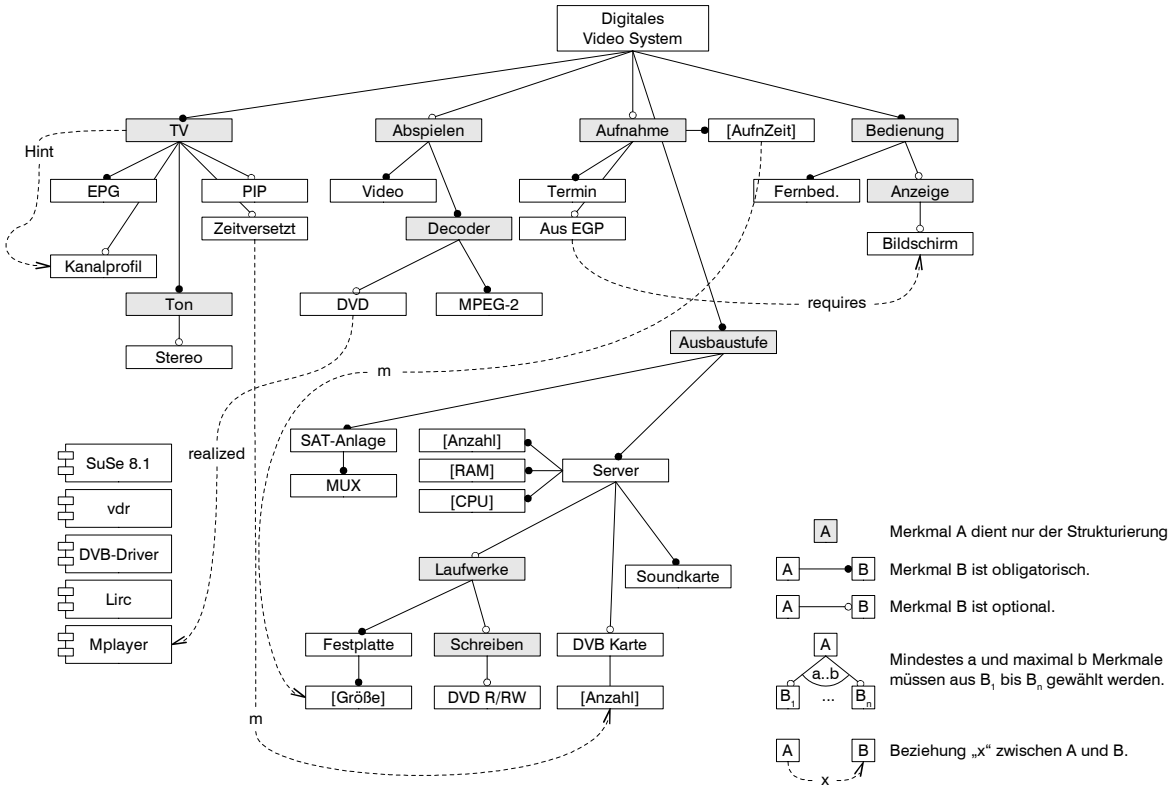
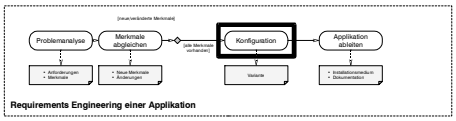


Abbildung 4.34: Die gewählten und abgeglichenen Merkmale des digitalen Videosystems.

In Abbildung 4.34 sind alle mit den Kundenanforderungen abgeglichenen und damit ausgewählten Merkmale als eigener Merkmalbaum dargestellt. Zusätzlich enthält dieser Merkmalbaum die relevanten Beziehungen zwischen den Merkmalen. Die Anzahl der Merkmale übersteigt die Anzahl der Anforderungen, da die Beziehungen zwischen den Merkmalen zur Wahrung der Konsistenz des Modells und für die spätere Ableitung eines korrekten Modells weitere Merkmale in der Variante bedingen. Im Folgenden werden die Beziehungen, die zum obigen Modell geführt haben, kurz erläutert. Wie bereits angesprochen, hat die *Hint*-Beziehung zwischen dem Merkmal „TV“ und „Kanalprofil“ dazu geführt, dass sich der Kunde entschieden hat, auch dieses Merkmal zu wählen. Die ursprüngliche Anforderung, Sendungen zeitversetzt sehen zu können, bedingt das Merkmal „Zeitversetzt“. Die mathematische Beziehung zwischen dem Merkmal „Zeitversetzt“ und dem Parameter-

merkmal für die Anzahl der im System einzubauenden DVB-Karten erfordert, dass mindestens zwei DVB-Karten verbaut werden, um gleichzeitig eine Sendung aufnehmen und eine aufgenommene sehen zu können. Eine weitere mathematische Beziehung stellt eine Verbindung zwischen der gewünschten Aufnahmezeit und dem Parametermerkmal der Festplattengröße her. Hier wird von einem MPEG/2 Datenstrom mit 16 MB/Minute ausgegangen, der letztlich mit der gewünschten Aufnahmezeit in Minuten multipliziert wird. Die Anforderung, direkt in einer elektronischen Programmzeitung eine Sendung zur Aufnahme wählen zu können, führt zu der *Requires*-Beziehung zwischen dem Merkmal „Aus EPG“ und „Bildschirm“, da die Programmierung nun über ein Bildschirmmenü realisiert werden muss. Die Beziehung zwischen dem Merkmal „DVD“ und der Komponente „Mplayer“ zeigt an, dass diese Komponente zusätzlich zu den Kernkomponenten des Systems eingebunden werden muss.

Konfiguration



Die Auswahl der Merkmale des Kunden wird in diesem Schritt als eigene Variante im FORE-Datenmodell hinterlegt. Dazu wird die „Variante“ Beziehung genutzt, wobei die Merkmale der Variante eine Beziehung zu derselben aufweisen. Dadurch lässt sich jederzeit feststellen, welchen Varianten ein Merkmal zugeordnet ist und aus welchen Merkmalen eine Variante besteht.

Die im Modell vorhandenen Parametermerkmale müssen nun mit Werten belegt werden, um danach die Gültigkeit der Variante über die im Modell vorhandenen Beziehungen überprüfen zu können. Es handelt sich dabei um die Einstellungen der von den Merkmalen referenzierten Komponenten, die sich im Fall des digitalen Videoprojektes in den entsprechenden Konfigurationsdateien befinden.

Durch die Parametrierung und Überprüfung der Variante ist diese komplettiert und wird im FORE-Datenmodell abgelegt. Damit kann diese Variante jederzeit wiederhergestellt werden. Allerdings dürfen Veränderungen an der Systemfamilie die Gültigkeit bereits existierender Varianten nicht beeinflussen. Das bedeutet, dass bei jeder Veränderung der Systemfamilie alle Varianten überprüft werden müssen, um die Konsistenz der Familie und der Varianten über lange Zeit hinweg aufrechterhalten zu können. Das parametrierte Merkmalmodell ist in Abbildung 4.35 dargestellt.

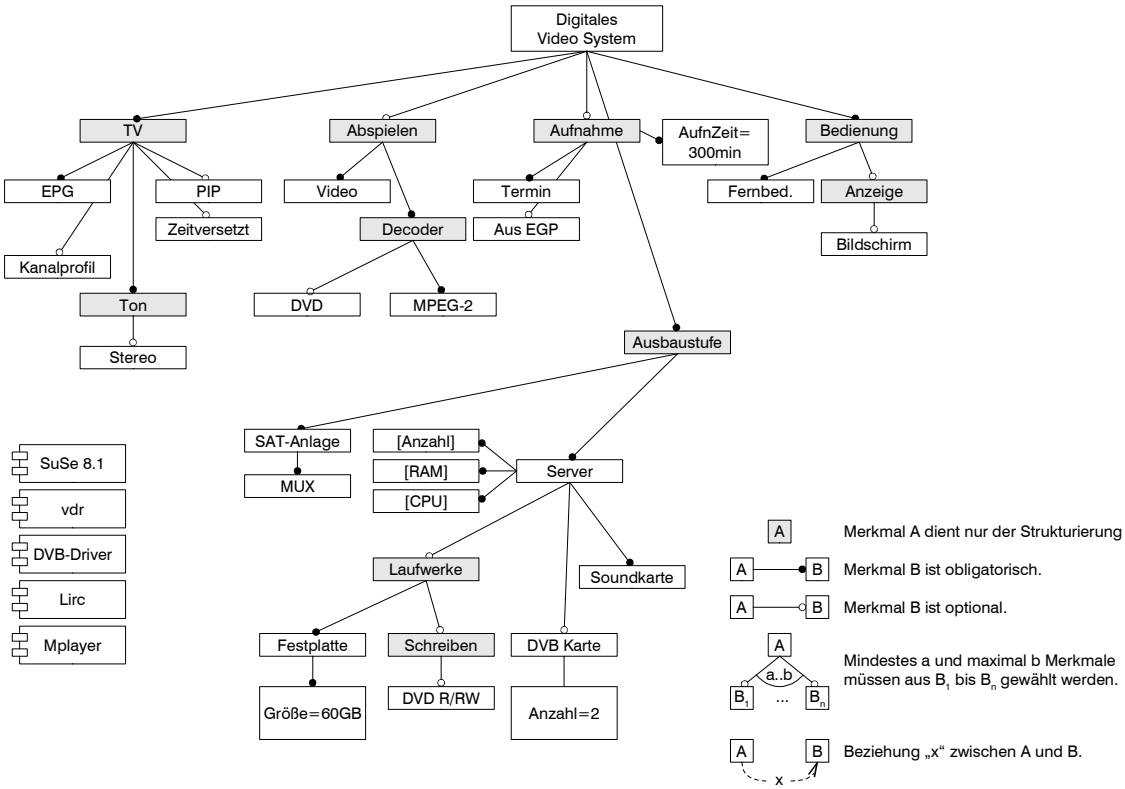
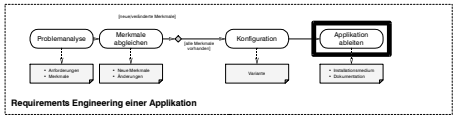


Abbildung 4.35: Variante des digitalen Videosystems als parametriertes Merkmalmodell

Ableiten der Applikation



Nach der Konfiguration steht eine gültige Variante der Systemfamilie zur Verfügung. Nun müssen die gewählten Merkmale und die eingestellten Parameter der Merkmale im realen System umgesetzt werden. Ist die Familie komplett implementiert, kann je nach genutzter Technologie, ein System direkt abgeleitet werden.

Im digitalen Videoprojekt werden durch Merkmale die Komponenten gewählt, die als *Plug-in* in das System integriert werden. Über die Parameter der Merkmale werden die Konfigurationsdateien der *Plug-ins* und des Hauptsystems eingestellt. Die Ableitung eines Systems bedeutet also in diesem Fall, dass die richtigen *Plug-ins* bestimmt werden, deren Konfigurationsdateien an die richtige Stelle der Platte kopiert werden und das Hauptsystem entsprechend aufgerufen wird. In einer zukünftigen Version des in Kapitel 5 beschriebenen Prototypen werden auch ein Installationsmedium und die

entsprechende Dokumentation automatisiert generiert werden können. Aus Abbildung 4.35 wird das in Abbildung 4.36 dargestellte System, bestehend aus Hard- und Softwarekomponenten, abgeleitet.

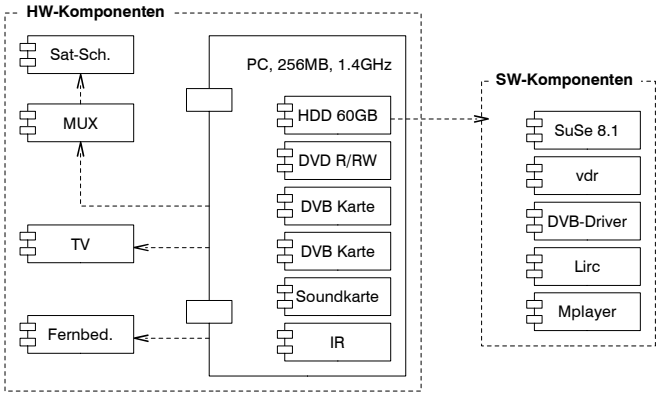


Abbildung 4.36: Komponenten des gewählten digitalen Videosystems

Die Ableitung hat zur Folge, dass die richtigen Systemkomponenten gewählt werden. Neben der reinen Auswahl der Komponenten müssen die Softwarekomponenten noch konfiguriert werden, was ebenfalls durch das Merkmaldiagramm sichergestellt wird. Dies wird am Beispiel des Merkmals DVD und der damit zu konfigurierenden Komponenten „Mplayer“ erläutert. Die Komponente „Mplayer“ stellt ein *Plug-in* für das System „vdr“ dar. Dieses *Plug-in* nutzt beim Systemstart eine Konfigurationsdatei, die das spätere Verhalten des *Plug-ins* beeinflusst. Die „*realized*“-Beziehung im Merkmalmodell von dem Merkmal „DVD“ zu der „Mplayer“-Komponente, fügt in die Konfigurationsdatei der „Mplayer“-Komponente die Zeilen ein, die das korrekte Abspielen von DVDs ermöglicht.

4.4 Datenmodell

Die beschriebenen Beziehungen der erweiterten Merkmalmodellierung aus Kapitel 4.2.1 und die zusätzlich benötigten Modelle für Anforderungen, Personen und die Dokumentstruktur werden in diesem Kapitel detailliert beschrieben und durch Klassendiagramme der UML dargestellt.

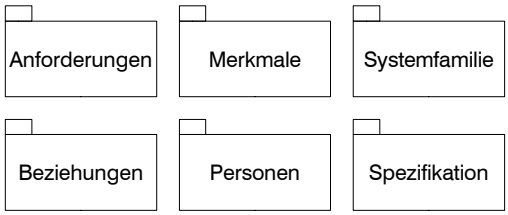


Abbildung 4.37: Globale Sicht auf das Datenmodell

In Abbildung 4.37 sind die Pakete des Datenmodells dargestellt, wobei zugunsten der Übersichtlichkeit auf die Beziehungen der Pakete untereinander verzichtet wurde. Am Ende dieses Unterkapitels ist eine komplette Ansicht des Modells mit allen Beziehungen enthalten, wobei jedes Element des Datenmodells in einer Datenbank vorgehalten werden muss und damit automatisch versioniert werden kann. Für jede neue Version wird der jeweilige Autor in dem Personenmodell referenziert, welches alle *Stakeholder* und beteiligten Firmen enthält.

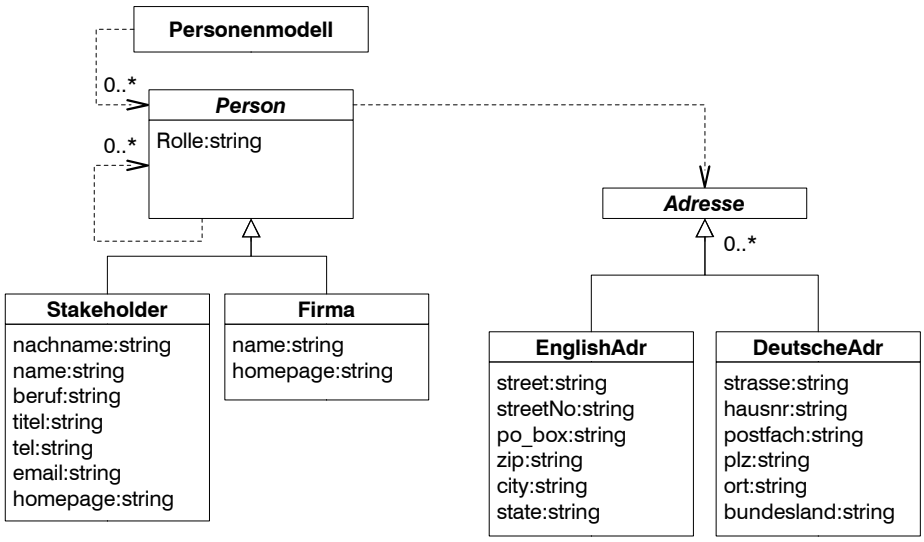


Abbildung 4.38: Personenmodell

Darüber hinaus sind auch die Rollen der *Stakeholder* in dem Modell festzuhalten, wie es in Abbildung 4.38 zu sehen ist. Das Personenmodell kann entweder direkt in das FORE-Datenmodell integriert werden oder muss in dem jeweils genutzten Versionierungswerkzeug nachgebildet werden, da eine Verwaltung der Nutzer des Systems meistens vorhanden ist. Das oben vorgestellte Personenmodell kann durch die abstrakten Personen und Adressen schnell an neue Bedürfnisse angepasst bzw. erweitert werden.

Bei dem Literaturverzeichnis und dem Glossar handelt es sich um Klassen, die als Listen verwaltet werden, wie in Abbildung 4.39 dargestellt. Synonyme können im Glossar als Beziehung zwischen

einzelnen Glossareinträgen modelliert werden. Beide Verzeichnisse müssen jedoch im *Tailoring*-Schritt des FORE-Prozesses an die jeweilige Entwicklungsumgebung angepasst werden. Als Voreinstellung sind die Einstellungen für *StarOffice 6.0* als Werkzeug zur finalen Bearbeitung der Spezifikation angegeben.

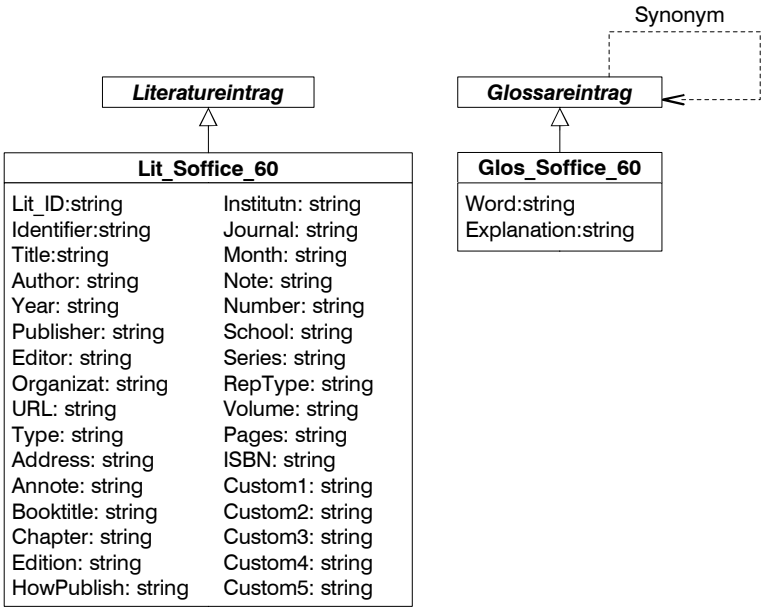


Abbildung 4.39: Literaturverzeichnis und Glossar

4.4.1 Anforderungsmodell

Die Anforderungen werden, wie in Abbildung 4.40 dargestellt, als hierarchisch gegliederte Baumstruktur abgelegt. Das Anforderungsmodell selbst stellt dabei die Wurzel des Baumes dar und enthält dann in beliebig vielen Ebenen mit jeweils beliebig vielen Elementen die eigentlichen Anforderungen. Jede Anforderung wird durch eine eindeutige Identifikation, eine Beschreibung, den Hintergrund, den Ursprung und eine mögliche Überprüfung der Anforderung beschrieben.

Darüber hinaus enthält jede Anforderung beliebig viele Assoziationen zu Varianten, denen sie angehören kann. Anforderungen, die keiner Variante zugeordnet werden, müssen automatisch im Kern der Systemfamilie berücksichtigt werden. Werden Anforderungen durch Entscheidungen parametriert, die bei der Ableitung eines Familienmitgliedes zu treffen sind, werden diese Entscheidungen in das *Decision Model* der Systemfamilie, beschrieben in Kapitel 3.2.1, übernommen und von der Anforderung aus referenziert. Die Assoziation zu dem Spezifikationsdokument zeigt an, wo genau die Anforderung mit den Attributen enthalten ist. Dabei zeigt diese Assoziation auf eine beliebige Position in einem durch Kapitel, Unterkapitel, Absätze usw. gegliederten Dokument. Eine genauere Dar-

stellung der Dokumentstruktur folgt in Kapitel 4.4.5. Handelt es sich um eine funktionale Anforderung, die durch ein Szenario beschrieben werden kann, wird eine weitere Assoziation genutzt, um den entsprechenden *Use-case* zu referenzieren. Sind darüber hinaus schon Komponenten bekannt, die eine Anforderung realisieren oder realisieren können, wird dies durch eine Assoziation zu der entsprechenden Komponente dargestellt. Sowohl der *Use-case* als auch die Komponente sind Diagramme der UML. Weitere erklärende Materialen zu der Anforderung werden durch eine Assoziation modelliert, die beispielsweise einen Literaturverzeichniseintrag oder eine Webseite referenzieren kann. Schließlich werden alle weiteren Abhängigkeiten oder Konflikte durch eine Beziehung realisiert, die durch eine eigene Beziehungsklasse modelliert ist. Diese spezielle Beziehungsklasse wird in Kapitel 4.4.4 näher erläutert.

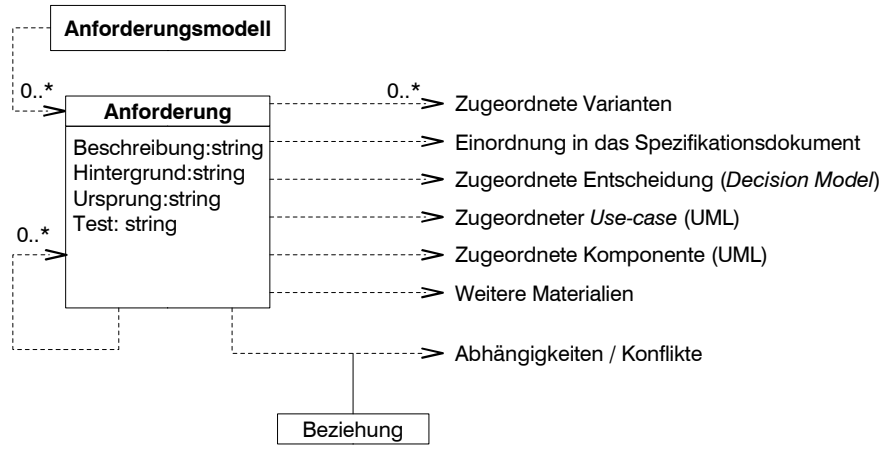


Abbildung 4.40: Anforderungsmodell

4.4.2 Merkmalmodell

Das Merkmalmodell ist wie auch das Anforderungsmodell hierarchisch aufgebaut und enthält an der Wurzel die Klasse Merkmalmodell, die gleichzeitig das Konzept, also die Systemfamilie selbst, repräsentiert. Unterhalb des Konzeptes sind alle Merkmale angeordnet, wobei in dem Modell in Abbildung 4.41 zwischen einem Merkmal, einem Parametermerkmal und einer Menge von Merkmalen unterschieden wird.

Diese Unterscheidung ist notwendig, um die in Kapitel 4.2 dargestellte, definierte Untermenge mit UML-Vielfachheiten modellieren zu können. Die Merkmalmenge stellt durch Vererbung ebenfalls ein gültiges Element des Merkmalmodells dar, wird jedoch nur zur Strukturierung genutzt, wobei die Vielfachheit mit Minimal- und Maximalwert angegeben werden muss. Weiterhin muss eine gültige Merkmalmenge mindestens so viele Merkmale enthalten, wie von dem Maximalwert gefordert

werden. Einem Merkmal können auch Parametermerkmale zugeordnet werden, die für die Parametrierung der den Merkmalen zugeordneten Komponenten bestimmt sind. Jedes Merkmal wird durch eine eindeutige Kennung, einen nach der in Kapitel 4.2 enthaltenen Merkmaldefinition zu vergebenden Merkmaltyp, einen Namen, eine Beschreibung, den Hintergrund, eine mögliche Überprüfung, die die Zusammenfassung der Prüfungen aller zugeordneten Anforderungen darstellt und einem Vermerk, ob es sich um ein optionales oder obligatorisches Merkmal handelt, in Form von Attributen beschrieben. Merkmale werden ebenfalls Varianten zugeordnet, wobei ein Merkmal mehreren Varianten zugeordnet sein kann. Dadurch können für jede Variante die entsprechenden Merkmale aus dem Modell gelesen werden. Merkmale ohne Zuordnung zu einer Variante gehören automatisch dem Kern an.

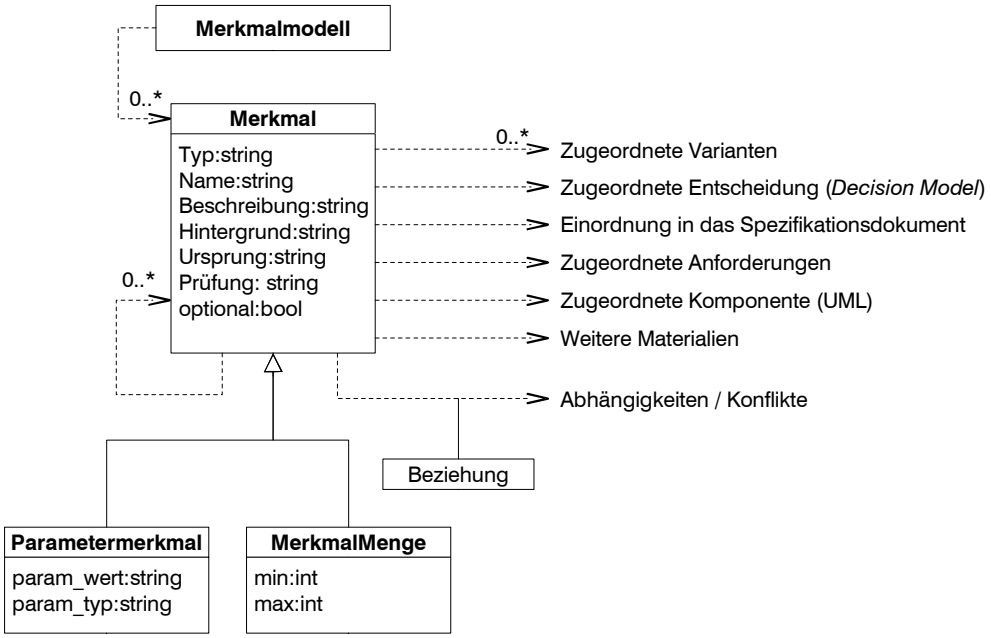


Abbildung 4.41: Merkmalmodell

Werden Merkmale durch Entscheidungen parametriert, die bei der Ableitung eines Familienmitgliedes zu treffen sind, werden diese Entscheidung in das *Decision Model* der Systemfamilie, beschrieben in Kapitel 3.2.1, übernommen und von dem Merkmal aus referenziert. Die Referenz zeigt auf die Stelle im Spezifikationsdokument, an der das Merkmal mit allen Attributen enthalten ist. Die Zuordnung zu den Anforderungen zeigt, welche Anforderungen durch ein Merkmal abstrahiert worden sind. Die Zuordnung zu einer Komponente referenziert ein Entwurfselement einer Komponente, die noch zu entwickeln ist oder eine bereits existierende Komponente darstellt. Wie auch bei den Anforderungen kann auch ein Merkmal eine Assoziation zu weiteren Materialen enthalten, die für das Verständnis nützlich sind. Schließlich werden auch die Beziehungen innerhalb von Merkmal-

modellen sowie Beziehungen nach bzw. von außen durch eine eigene Beziehungsklasse modelliert, die in Kapitel 4.4.4 näher erläutert wird.

4.4.3 Systemfamilienmodell

Das Systemfamilienmodell spiegelt die grundsätzliche Idee der Systemfamilienentwicklung wider und organisiert die Elemente des Datenmodells entsprechend. In Abbildung 4.42 ist die Systemfamilie als Klasse modelliert, die einen Kern und beliebig viele Variabilitäten enthält. Dem Kern sind eine Reihe von Merkmalen zugeordnet.

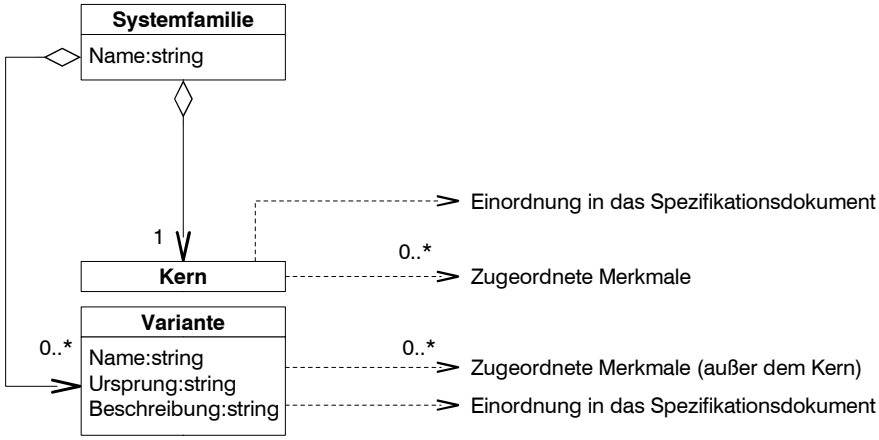


Abbildung 4.42: Systemfamilienmodell

Jeder Variante sind ebenfalls eine Reihe von Merkmalen zugeordnet, wobei die Menge der dem Kern zugeordneten Merkmale und die Menge der den Varianten zugeordneten Merkmale disjunkt sind. Sowohl der Kern als auch jede Variante werden über die Assoziation in das Spezifikationsdokument integriert.

4.4.4 Beziehungen zwischen Modellelementen

Wie bereits in Kapitel 4.2 erläutert, stellen die Beziehungen zwischen Elementen des Datenmodells eine Neuerung und zentral wichtige Erweiterung der Merkmalmodellierung dar. Im Datenmodell werden die in Kapitel 4.2 vorgestellten Beziehungen durch eine Klasse modelliert, die alle an einer Beziehung beteiligten Elemente über Assoziationen referenziert und die in Abbildung 4.43 dargestellt ist.

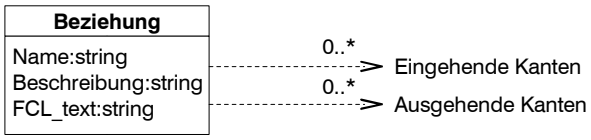


Abbildung 4.43: Beziehungen

Dabei werden für eine Beziehung eingehende und ausgehende Kanten unterschieden, um zwischen Elementen, die eine Beziehung erfordern und Elementen, die durch eine Beziehung betroffen sind, zu differenzieren. Jede Beziehung erhält einen Namen und kann durch einen beschreibenden Text näher erläutert werden.

Die formale Beschreibung einer Beziehung wird in einem dritten Attribut hinterlegt. Dieses Attribut enthält einen Text, der einen Ausdruck der in Kapitel 4.2.3 definierten *Feature Constraint Language* darstellt. Diese Ausdrücke werden im Verlauf der Entwicklung von der Entwicklungsumgebung ausgewertet, wodurch Fehler im Modell oder in einer abgeleiteten Applikation festgestellt werden können.

4.4.5 Dokumentmodell

Das Dokumentmodell definiert die Struktur des Spezifikationsdokumentes, welches die Grundlage der späteren Entwicklung und auch den rechtsverbindlichen Vertrag zwischen Auftraggeber und Auftragnehmer darstellt. Im FORE-Datenmodell handelt es sich dabei um eine einfache Struktur, die über Assoziationen in das *DocBook*-Format [Wals 2003] den vollen Umfang und die Fähigkeiten eines Textsatzsystems erhält.

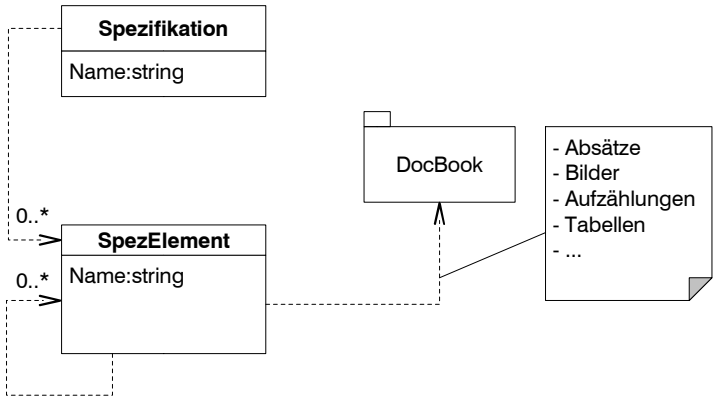


Abbildung 4.44: Dokumentmodell

Mit dem FORE-Dokumentmodell kann die vorgegebene Dokumentstruktur aus Kapitel 4.1.2 modelliert werden. Durch das in Abbildung 4.44 dargestellte Modell ist es möglich, auch firmenspezifische Strukturen zu modellieren und mit dem FORE-Ansatz zu nutzen. Dazu muss diese Struktur im Rahmen des in Kapitel 4.3.1 beschriebenen *Tailoring*-Schrittes modelliert werden.

4.4.6 Zusammenfassung

Alle in Kapitel 4.4 vorgestellten einzelnen Modelle sind in Abbildung 4.45 zusammengefasst. Die zentrale Klasse „FORE_Datenmodell“ enthält die acht Teile, aus denen das Datenmodell besteht, das Anforderungsmodell, das Personenmodell, das Merkmalmodell, das Dokumentmodell, das Systemfamilienmodell, das Glossar und das Literaturverzeichnis. Zur Vereinfachung des Modells wurde die abstrakte Klasse „Modellelement“ eingefügt, von der die Modellelemente der Teilmodelle erben. Dadurch können Beziehungen zwischen beliebigen Modellelemente modelliert werden und alle Modellelemente lassen sich in das Spezifikationsdokument einordnen, was durch die Assoziation des Modellelementes zu einem Element der Spezifikation realisiert ist.

Die Assoziation zu den externen Modellen wurde im Fall der Assoziation zum *DocBook*-Format aus dem bereits in Kapitel 4.4.5 dargestellten Modell übernommen. Für die Assoziationen zu den *Usecases* und den Komponenten des Systems wurde das Paket „UML-Modelle“ eingefügt.

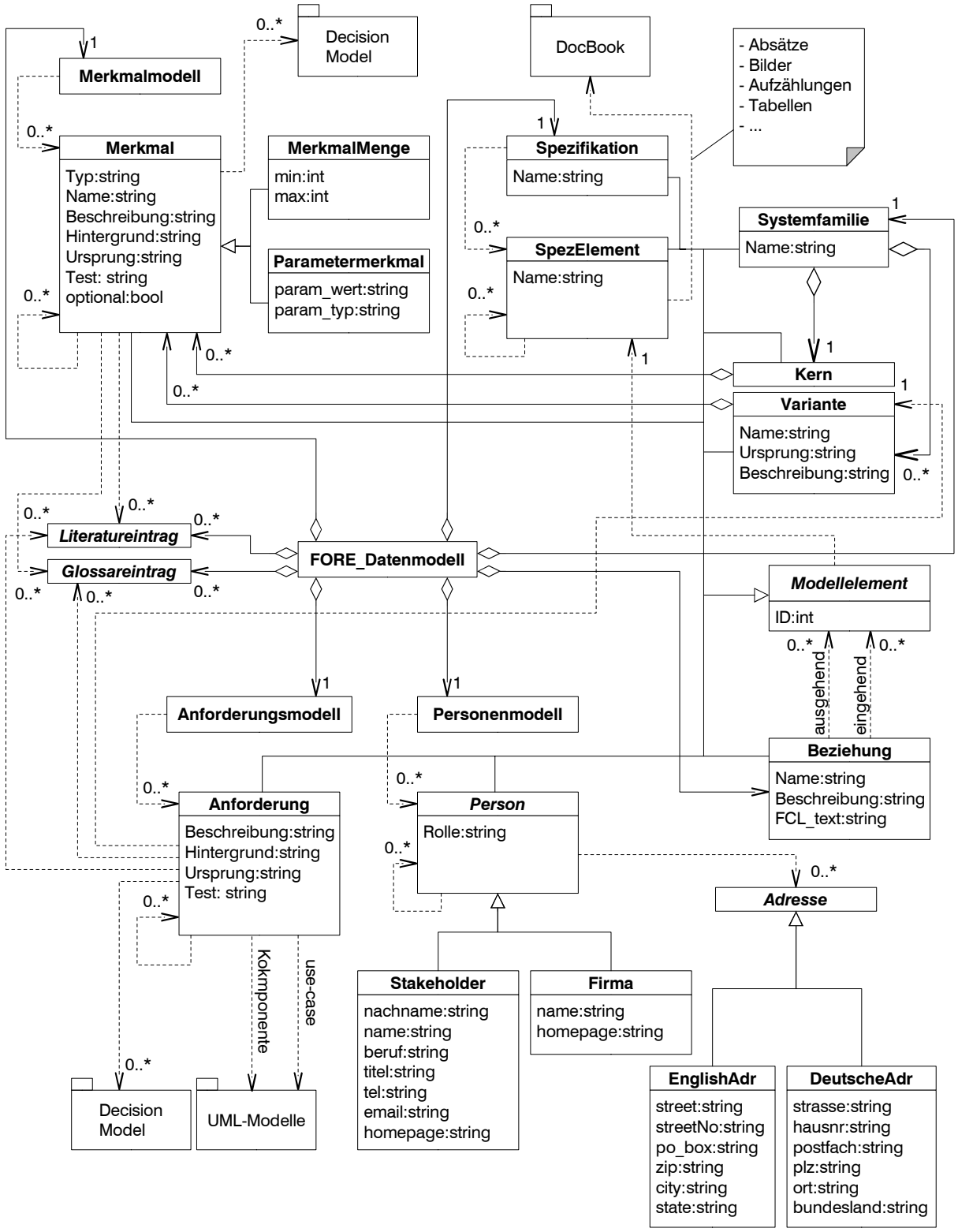


Abbildung 4.45: Gesamtes FORE-Datenmodell

Kapitel 5

Blancpain, Grand Complication 1735

Prototypische Umsetzung

In diesem Kapitel wird der Nachweis der Anwendbarkeit für den in Kapitel 4 beschriebenen Lösungsansatz durch eine prototypische Implementierung erbracht. Die Hintergründe und die Architektur der prototypischen Implementierung werden vorgestellt und erläutert. Zunächst wird das Datenmodell in eine XML-Struktur umgesetzt, auf die über eine definierte FORE-API zugegriffen wird. Dadurch werden die Daten zusammen mit den Zugriffsmethoden gekapselt. Die in Kapitel 4 vorgestellte *Feature Constraint Language* wird von einem eigens entwickelten Parser verarbeitet, der über die Zugriffsmethoden FORE-API genutzt wird. Beide Teile stellen zusammengenommen eine Komponente zur Be- und Verarbeitung von Daten der *Requirements-Engineering*-Phase für die Systemfamilienentwicklung dar. Sie sind in eine Windows-Applikation eingebettet, deren Benutzung an die Vorgaben des in Kapitel 4 dargestellten FORE-Entwicklungsprozesses angelehnt ist. Das Kapitel schließt mit einer Gesamtdarstellung der Architektur des Prototypen ab.

5.1 Umsetzung des Datenmodells

Die objektorientierte Struktur des Datenmodells wird in eine *eXtensible Markup Language* (XML) Struktur überführt, die durch ein XML-Schema definiert ist. Bei der Definition von Schemata wird XML selbst genutzt, um die Struktur zukünftiger XML-Dokumente zu beschreiben, die den Regeln der Schemata genügen müssen. Schemata erlauben die Modellierung einer objektorientierten Struktur mit Vererbung und abstrakten Datentypen mit XML. Zum weiteren Verständnis folgt ein kurzer Überblick über die genutzten XML-Konstrukte.

Mit Hilfe von XML können beliebige Daten strukturiert werden. Dazu wird eine Datenmenge, bei der es sich häufig um Textdaten handelt, durch semantische Informationen angereichert. Ein einfaches Beispiel stellen Adressen dar. Wird eine Adresse als reiner Text angegeben, ist es mitunter schwierig oder unmöglich zwischen Vor- und Nachnamen einer Person zu unterscheiden. XML erlaubt die semantische Information, dass es sich bei einem Textstück um den Vornamen handelt, über so genannte *Tags* anzugeben. Der Name `Hans Maier` würde mit XML-*Tags* als `<Vorname>Hans</Vorname><Nachname>Maier</Nachname>` dargestellt werden. Ein *Tag* ist dabei ein beschreibender Text, der in spitze Klammern eingeschlossen wird und sowohl den Start als auch das Ende einer Information anzeigt, indem dem Namen des *Tags* ein „/“ vorangestellt wird. Alle Informationsteile einer Datenmenge werden in beschreibende Tags eingeschlossen, wobei auch das rekursive Einschließen bereits durch Tags beschriebener Informationen möglich ist, so dass sich eine Baumstruktur wie in Abbildung 5.1 ergibt.

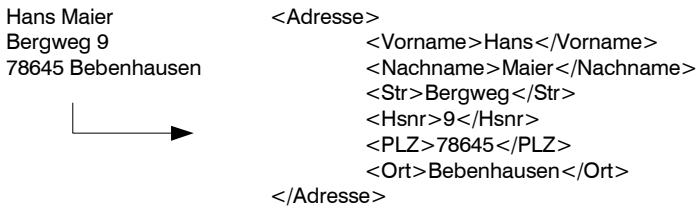


Abbildung 5.1: Strukturierung einer Adresse durch XML

Welche der *Tags* in einer Struktur vorhanden sein müssen, in welcher Reihenfolge sie auftreten müssen und weitere Einschränkungen werden in einem XML-Schema festgehalten. XML-Dokumente können nach Schemata aufgebaut sein, wodurch ihre Struktur gleichzeitig überprüfbar wird.

Die Regeln der Struktur von XML-Dokumenten sind an die Backus-Naur-Form (BNF) für die Definition der Syntax von Computersprachen angelehnt. In Abbildung 5.2 ist ein Auszug aus dem Schema des FORE-Datenmodells zu sehen, der die Struktur eines Merkmals darstellt.

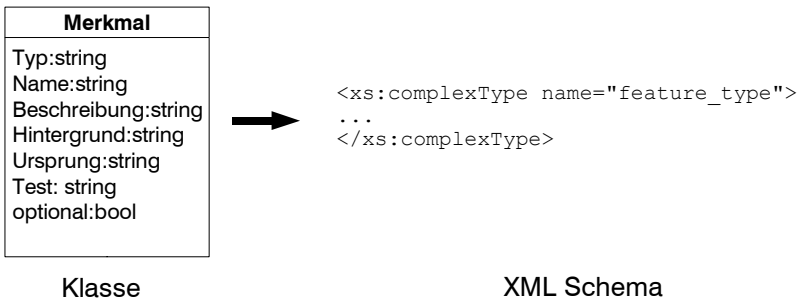


Abbildung 5.2: Umsetzung einer Klasse in ein XML Schema

Die Klasse „Merkmal“ wird in Abbildung 5.2 als neuer XML-Datentyp modelliert. Das obige Schema erzeugt einen Datentyp namens `feature_type`. Durch die Angabe von `xs:complexType` wird angezeigt, dass es sich um einen zusammengesetzten Datentyp handelt. Weiterhin müssen noch die Attribute der Klasse modelliert werden. Der Attributnamen, der Datentyp des Attributes und die Angabe, ob das Attribut erforderlich oder optional ist, werden in das Schema durch das Schlüsselwort `xs:attribute` eingefügt. Attribute der Klasse, die größere Informationsmengen beinhalten, werden in XML als eigene Elemente behandelt. Die Reihenfolge dieser Elemente und der jeweilige Datentyp werden im Schema angegeben. Eine Besonderheit der Klasse „Merkmal“ stellt der Typ des Merkmals dar, da er nach der Merkmaldefinition aus Kapitel 4 drei vorgegebene Werte annehmen kann. Dies wird im Schema durch einen Aufzählungstyp berücksichtigt, was im Schema durch das Schlüsselwort `xs:enumeration` angegeben wird.

```
<xs:complexType name="feature_type">
	<xs:sequence>
		<xs:element name="description" type="xs:string"/>
		<xs:element name="rationale" type="xs:string"/>
		<xs:element name="origin" type="xs:string"/>
		<xs:element name="test" type="xs:string"/>
	</xs:sequence>
	<xs:attribute name="type" use="required">
	<xs:simpleType>
		<xs:restriction base="xs:string">
			<xs:enumeration value="functional"/>
			<xs:enumeration value="interface"/>
			<xs:enumeration value="concept"/>
		</xs:restriction>
	</xs:simpleType>
	</xs:attribute>
	<xs:attribute name="name" type="xs:string" use="required"/>
	<xs:attribute name="optional" type="xs:boolean" use="required"/>
</xs:complexType>
```

Abbildung 5.3: XML Schema Definition für die Merkmalklasse

In Abbildung 5.3 ist die Schema-Definition für die Klasse „Merkmal“ angegeben. Analog werden alle weiteren Klassen des FORE-Datenmodells durch XML-Schemata beschrieben. Vererbungen werden für einen neuen Schema-Datentyp über die Erweiterungsbeziehung `<xs:extension base="element_type">` angegeben. Dabei steht `element_type` für die Elternklasse, von der der aktuelle Typ erbt. Wie im Datenmodell erben auch im XML-Schema alle Elemente von einem abstrakten Modellelement. Die Beziehungen mit angeschlossenen *Feature-Constraint-Language*-Ausdrücken nutzen als eingehende und ausgehende Elemente dieses abstrakte Modellelement und können daher für alle Elemente des XML-Datenmodells genutzt werden.

Der Zugriff auf alle Elemente des XML-Modells erfolgt über den XML4C-Parser von IBM [IBMa 2003] und das *Document Object Model* (DOM) [W3C 2003], welches von diesem Parser unterstützt wird. Eine eigene FORE-API stellt die notwendigen Zugriffsmethoden auf die Daten zur Verfügung, um einem Entwickler die Benutzung von FORE zu ermöglichen. Die FORE-API stellt

gleichzeitig die Schnittstelle der FORE-Komponente dar, die in bereits existierende CASE-Werkzeuge eingebunden werden kann. Damit ist es möglich, FORE als Erweiterung in eine bestehende Werkzeuglandschaft einer Firma zu integrieren. In Abbildung 5.4 ist die Architektur des Datenmodells mit dem XML-Parser und der FORE-API dargestellt. Es sind nur die Methoden enthalten, die für die Betrachtungen in diesem Abschnitt relevant sind, wobei die Bearbeitung der Anforderungen, Merkmale und Varianten der Systemfamilie im Vordergrund stehen. Die Methoden für die Bearbeitung von Einschränkungen nutzen die in Kapitel 4 vorgestellte *Feature Constraint Language* und werden im nächsten Kapitel kurz angesprochen.

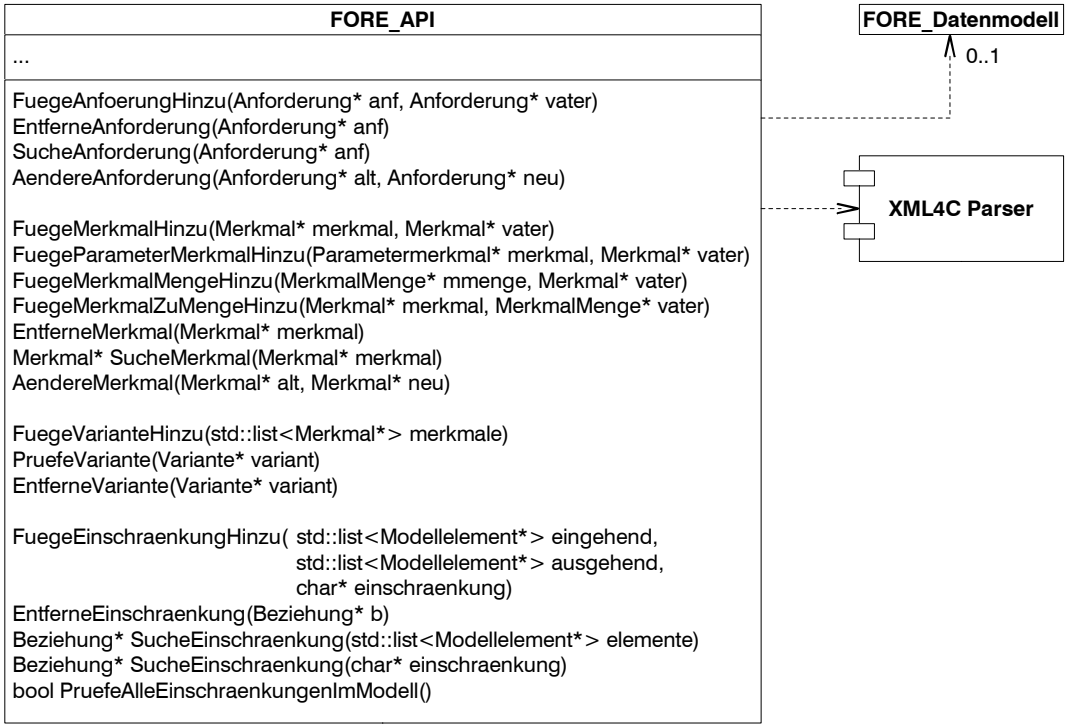


Abbildung 5.4: FORE-API mit XML Parser

5.2 Parser für die *Feature Constraint Language*

Neben der Speicherung der Daten aus der *Requirements-Engineering*-Phase der Systemfamilienentwicklung, müssen die Daten überprüft werden, wie in Kapitel 4 erläutert. Diese Überprüfung basiert auf den durch den Entwickler modellierten, einschränkenden Beziehungen zwischen Modellelementen. Ausgedrückt werden die Einschränkungen in der *Feature Constraint Language* (FCL), die als reiner ASCII-Text als Attribut der Klasse „Beziehung“ im Modell enthalten sind.

In der prototypischen Implementierung wurden die Windows-Varianten von *LEX* und *YACC* der Firma *BumbleBee* [Bumb 2003] genutzt, um einen eigenen Compiler für FCL zu entwickeln. Die Vorteile, FCL durch einen eigenen Compiler umzusetzen, liegen in der Flexibilität dieser Variante begründet. Veränderungen und Anpassungen können ohne großen Aufwand eingearbeitet werden. In Abbildung 5.5 ist die Architektur aus Abbildung 5.4 um den eigenen Compiler erweitert. Die beiden Eingabedateien für die automatische Compilergenerierung sind ebenfalls dargestellt und müssen bei Veränderungen entsprechend angepasst und neu compiliert werden.

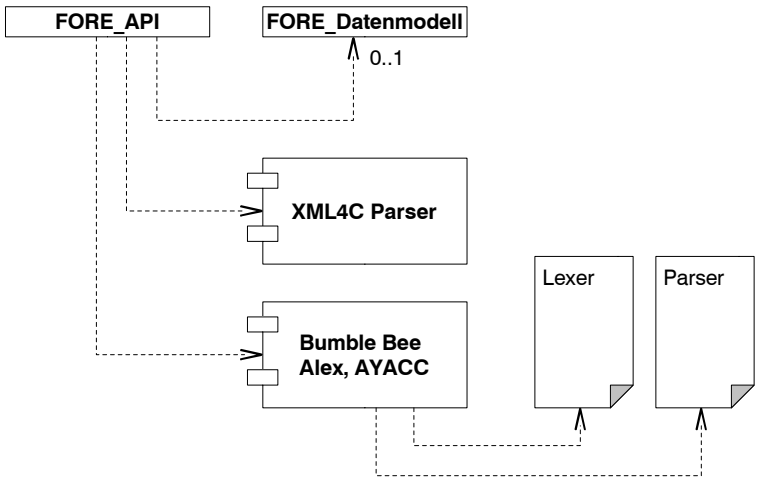


Abbildung 5.5: FORE-API mit XML-Parser und FCL-Compiler

5.3 Architektur des Prototypen

Zur Visualisierung und Prüfung der oben beschriebenen Einzelkomponenten sind diese in eine Windows-Applikation integriert. Eine XML-Datei mit den *Requirements-Engineering*-Daten des digitalen Videoprojektes wird beim Programmstart eingelesen. Daraus erzeugt der Prototyp eine Baumdarstellung der Anforderungen und Merkmale des Systems. Einfache Editierfunktionen stehen im Kontextmenü zur Verfügung und können zur Bearbeitung der Daten genutzt werden. Darüber hinaus lassen sich Beziehungen in der FCL definieren und überprüfen.

In Abbildung 5.6 ist die Architektur des gesamten Prototypen dargestellt. Es handelt sich um eine auf Dialogen basierende Applikation. Durch das Applikationsobjekt wird das Programm gestartet. Der Hauptdialog wird erzeugt und instantiiert seinerseits die FORE-API-Komponente.

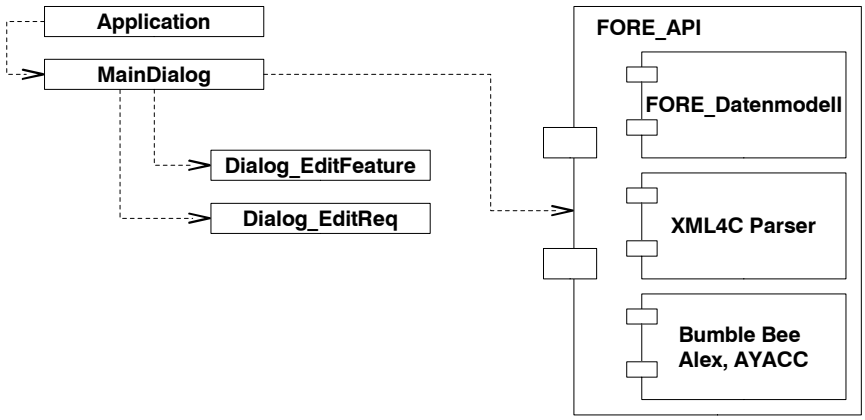


Abbildung 5.6: Architektur des Prototypen

In Abbildung 5.7 ist das Startfenster des Prototypen dargestellt. Auf der linken Seite sind die Daten des digitalen Videoprojektes als Baum enthalten und können editiert werden. Rechts daneben können die FCL-Ausdrücke für die Beziehungen zwischen den Elementen angegeben werden. Durch Auswahl einer Menge von Merkmalen mit Hilfe der *Check-Boxen* kann eine Variante des Systems konfiguriert werden, wobei der FCL-Parser des Prototypen die modellierten Beziehungen zwischen den Elementen prüft, so dass nur gültige und korrekte Varianten der Systemfamilie erstellt werden können.

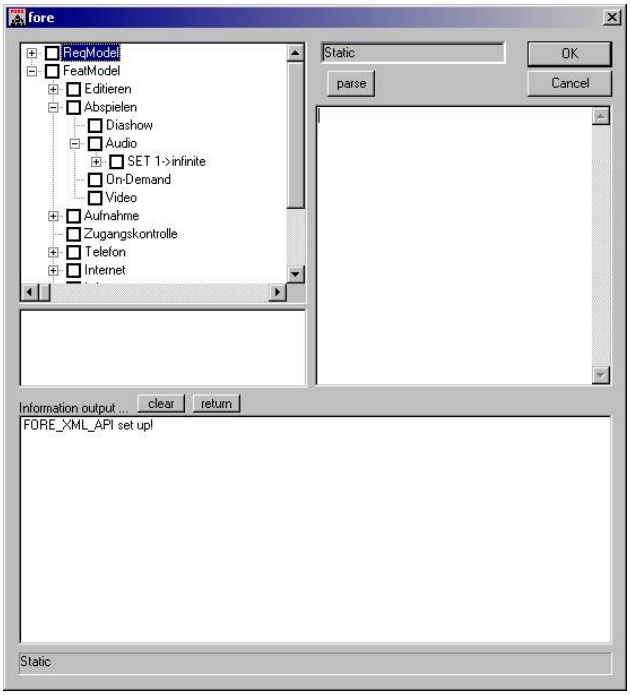
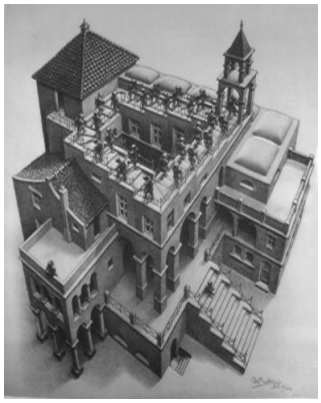


Abbildung 5.7: Startfenster des Prototypen

Kapitel 6



Escher - „Haus“

Validierung und Bewertung

In diesem Kapitel wird der vorgestellte Lösungsansatz den gesteckten Zielen gegenübergestellt und bewertet. Zunächst wird geprüft, ob die gestellten Anforderungen durch den vorgestellten Lösungsansatz erfüllt werden, die Anforderungen werden validiert. Danach wird der Lösungsansatz ausgehend von diversen studentischen Projekten und eigenen Erfahrungen bewertet. Dabei werden die Erfahrungen mit dem FORE-Entwicklungsprozess im Rahmen des digitalen Videoprojektes zur Bewertung herangezogen. Abschließend werden derzeitige Grenzen der Lösung aufgezeigt.

6.1 Überprüfung des eigenen Ansatzes

Anhand der Problemstellung aus Kapitel 1 und der präzisierten Problemstellung aus Kapitel 3 wird hier der eigene Lösungsansatz aus Kapitel 4 überprüft. Basierend auf statistischen Erhebungen im ersten Kapitel wurde die *Requirements-Engineering*-Phase der Systemfamilienentwicklung als Problemfeld dargestellt und in Kapitel 3 tiefergehend untersucht. Diese Untersuchungen haben in der präzisierten Problemstellung am Ende des dritten Kapitels auf Unzulänglichkeiten existierender Ansätze geführt. Folgende drei Forderungen wurden als Ergebnis der Analyse des Standes der Technik aufgestellt.

1) Die Merkmalmodellierung nach Kang muss erweitert werden, um die Ableitung einer mit dem Modell konsistenten Systemvariante zu ermöglichen und die Beziehungen, die sich aus den Wechselwirkungen der Merkmale ergeben, in dem Modell hinterlegen zu können.

2) Die Bearbeitung eines derart erweiterten Merkmalmodells muss durch einen entsprechend angepassten *Requirements-Engineering*-Entwicklungsprozess berücksichtigt werden. Daher müssen die in Kapitel 3 beschriebenen, existierende Ansätze zu einem angepassten *Requirements-Engineering*-Entwicklungsprozess zusammengesetzt werden.
3) Ein angepasstes *Requirements-Engineering*-Datenmodell muss auf der Basis existierender Modellbestandteile erstellt werden.

Die Erweiterung der Merkmalmodellierung nach Kang ist in Kapitel 4 ausführlich beschrieben und stellt eine Neuerung im Bereich der Systemfamilienentwicklung dar. Es wurden insgesamt 30 Beziehungen zwischen Merkmalen, Anforderungen und UML-Modellelementen der Entwurfsphase dargestellt, die durch die neue *Feature Constraint Language* formal beschrieben werden. Durch diese formale Beschreibung und durch entsprechende Werkzeuge, die in Kapitel 5 genannt wurden, ist es möglich, das Merkmalmodell zu überprüfen und Konsistenzprüfungen, wie sie in Kapitel 4 beschrieben sind, durchzuführen. Daher sind die unter Punkt 1) aufgeführten Anforderungen erfüllt.

Wie in Kapitel 3.2.5 dargestellt, werden die variablen Anteile, das Familienmodell, die Konfiguration und Parametrierung sowie die Überprüfung des Systemfamilienmodells von den existierenden Ansätzen nur unzureichend unterstützt. Es gibt also keinen Ansatz, der alle Anforderungen gleichzeitig erfüllt. In Kapitel 4.3 ist ein Systemfamilienentwicklungsprozess beschrieben, der aus dem *Six-Pack*-Modell als grobem Rahmen, dem FAST-Entwicklungsprozess als einer Spezialisierung der Systemfamilienentwicklung und einer Reihe herkömmlicher *Requirements-Engineering*-Ansätze für die konkrete Bearbeitung der Prozessschritte zusammengesetzt ist. Die erweiterte Merkmalmodellierung wird bei der Einarbeitung in die Systemfamilienproblematik, der Abstraktion von Anforderungen durch Merkmale, der Modellierung der Systemfamilie selbst und bei der Ableitung einer Applikation, basierend auf der Familie berücksichtigt. Bei der Ableitung eines Familienmitgliedes wird von der Überprüfbarkeit des Merkmalmodells Gebrauch gemacht, sodass nur noch gültige Familienmitglieder möglich sind. Die Integration der existierenden Methoden und Ansätze in den FORE-Prozess ist in Anhang C, dem FORE-Prozesshandbuch, ausführlich enthalten. Damit sind die Anforderung unter Punkt 2) erfüllt.

Schließlich wurde in Kapitel 4.4 ein Datenmodell vorgestellt, welches alle im Verlauf des FORE-Prozesses entstehenden Daten aufnehmen und entsprechend den Forderungen der erweiterten Merkmalmodellierung verknüpfen kann. Dieses Datenmodell integriert existierende Ansätze im Personenmodell und bietet auch eine Versionierung an, falls entsprechende Werkzeuge nicht zur Verfügung stehen. Das Teilmodell der Anforderungen ist an existierende Ansätze angelehnt, bietet jedoch neue und weitergehende Verknüpfungen zu den weiteren Modellelementen. Das Teilmodell für Merkmale, das Systemfamilienmodell und die flexibel zu verwendenden Beziehungen des FORE-

Datenmodells sind neu. Darüber hinaus bietet die FORE-API einen einfachen Zugriff auf das Datenmodell und erlaubt die Verwendung des gesamten Lösungsansatzes in existierenden CASE-Werkzeugen. Dadurch sind die Anforderungen des Punktes 3) erfüllt.

6.2 Historie der Lösung

In mehreren studentischen Projekten wurden die im Lösungsansatz dieser Arbeit vorgestellten Konzepte der Systemfamilienentwicklung überprüft. Ein Großteil der studentischen Arbeiten fand im Rahmen des in dieser Arbeit entstandenen digitalen Videoprojektes statt, welches in Kapitel 2 bereits vorgestellt wurde. Weitere Arbeiten wurden in anderen Domänen angefertigt.

Bei dem digitalen Videoprojekt ging es um die Entwicklung einer Systemfamilie basierend auf dem existierenden *Open-Source*-Projekt *Video Disk Recorder* (VDR). Dabei ist die Referenzarchitektur der Systemfamilie in Form einer *Plug-in*-Architektur durch VDR bereits gegeben. Über diverse *Plug-in*s besteht die Möglichkeit, zusätzliche Merkmale in das System zu integrieren. In einem Teil der studentischen Projekte wurden Merkmale durch neue *Plug-ins* realisiert. Für diese Arbeit lag der Fokus auf der *Requirements-Engineering*-Phase des digitalen Videoprojektes, da bereits viele *Plug-ins* sowie die Referenzarchitektur des Systems verfügbar sind und den Anforderungen an eine Systemfamilie vollständig genügen.

Es hat sich gezeigt, dass zu Beginn der jeweiligen Projekte lediglich ein Treffen notwendig war, um das Prinzip der Systemfamilienentwicklung zu erläutern. Dazu fand ein kurzer *Workshop* statt, wie in dem Prozessschritt „Domänenwissen sammeln“ des in Kapitel 4.3 vorgestellten FORE-Prozesses erläutert. Während des Prozessschrittes „Analyse existierender Systeme“ konnte die Einarbeitungszeit durch eine eigens erstellte Dokumentation von drei auf zwei Wochen verkürzt werden. Die in Kapitel 3 angesprochene leichte Verständlichkeit der Merkmalmodelle hat sich in den studentischen Projekten bestätigt. Bereits nach einmaliger, etwa zweistündiger Diskussion, konnten die Studenten die Merkmalmodelle mit den Erweiterungen zur Modellierung nutzen. Erweiterungen für das digitale Videoprojekt, wie auch Neuerungen in anderen Domänen wurden mit Hilfe der erweiterten Merkmaldiagramme beschrieben. Ohne die in dieser Arbeit beschriebenen Erweiterungen wären die Merkmalmodelle unvollständig gewesen. Nur durch zusätzliche, textuelle Erläuterungen wären diese Modelle nutzbar gewesen, eine automatisierte Überprüfung der Modelle wäre damit gar nicht möglich gewesen. Die Beziehungen der Merkmale innerhalb einer Erweiterung, also eines *Plug-ins* für das digitale Videoprojekt, wurden von allen Studenten genutzt. Wechselwirkungen der Erweiterung mit anderen Systemteilen und *Plug-ins* konnten jedoch nur von Studenten mit einschlägigen Vorkenntissen über das digitale Videoprojekt erkannt werden. Dieses Wissen hatten sich die Studenten schon vor Beginn ihrer Arbeiten angeeignet. Damit muss einschränkend bemerkt werden, dass

die Integration neuer Komponenten einen erfahrenen Entwickler voraussetzt, um nicht die Architektur der Systemfamilie und die Konsistenz des Merkmalmodells zu gefährden. Obwohl die automatisierte Überprüfung möglich ist, müssen notwendige und sinnvolle Beziehungen von einem Entwickler modelliert werden, der den Überblick über das System hat.

In Tabelle 6.1 sind die relevanten, studentischen Projekte mit dem jeweiligen Zeitrahmen (**D**iplomarbeit, **S**tudienarbeit oder **H**auptseminar) angegeben. Dazu kommt die Einordnung der Projekte in den FORE-Entwicklungsprozess, wobei es hier um eine Abschätzung des Anteils der im jeweiligen Projekt durchlaufenen Prozessschritte geht. Für das Ergebnis, welches zum Teil als erweitertes Merkmalmodell vorliegt, wird die Anzahl der Merkmale angegeben, die von dem jeweiligen Studenten erarbeitet wurden. Des Weiteren wird angegeben, ob die neuen Beziehungen genutzt wurden, um ein korrektes Modell zu erhalten. Ein „-“ zeigt an, dass die Projekte entweder in einer Entwicklungsphase vor der Merkmalmodellierung stattfanden oder lediglich einen kleinen Einfluss auf die vorliegende Arbeit hatten, was bei den letzten beiden Projekten der Fall ist.

Thema	Zeitrahmen	Anteil vom FORE-Prozess	FORE		Erweiterung der Familie? (J/N)	Ergebnis der Arbeit
			Merkmale	Beziehungen? (J/N)		
Arbeiten im Umfeld des digitalen Videoprojektes						
Automatische Werbeblockerkennung [Rege 2003].	D	100%	2	J	J	Komponente zur Nachbearbeitung von Videoströmen.
Videodatenverteilung [Meff 2003].	D	80%	4	J	J	*Plug-in* zur Videodatenverteilung.
Fernbedienung als Systemfamilie [Diet 2003].	D	100%	24	J	J	Eine sehr variabel gestaltbare Fernbedienung auf Basis eines Palm-Handhelds.
Systemfamilienbasierte Dokumentation [Prei 2003].	S	10%	-	-	N	Eine aus dem Quellcode automatisch generierbare Dokumentation.
XML Doku [Berg 2003].	H	5%	-	-	N	Umsetzung der Dokumentation mit XML und XSLT.
Verarbeitung von Service Informationen [Scha 2003].	S	80%	3	J	J	Komponente zur Verarbeitung der kompletten Service Informationen aus einem digitalen Videostrom.
Variable Treiberarchitektur für DVB Karten [Schr 2003].	S	80%	-	-	J	Neue Treiber-Architektur der Systemfamilie der Technotrend DVB-Karten.
Automatisierung der Dokumentation in der SW-Entwicklung, [Klug 2002].	S	5%	-	-	-	Prozess zur Automatisierung der Dokumentation von Softwareprojekten durch XML und *DocBook*.

Thema	Zeitrahmen	Anteil vom FORE-Prozess	FORE		Erweiterung der Familie? (J/N)	Ergebnis der Arbeit
			Merkmale	Beziehungen? (J/N)		
Arbeiten aus anderen Anwendungsdomänen						
Automatische Generierung von Anwendungsoberflächen [Uter 2003].	D	100%	79	J	J	Die Domäne der Rechnungsverarbeitung wurde mit Hilfe der Merkmalmodellierung bearbeitet,
Automatische Generierung geographischer Informationssysteme [Ulri 2002].	D	70%	-	J	J	Ein Generator für die Ableitung von so genannten Fachschalen, also Applikationen, für die Nutzung bestimmter geographischer Daten, z.B. Straßennetz oder Stromnetz.
Begleitende Arbeiten im Umfeld der Systemfamilienentwicklung						
Reverse Engineering von Entwurfsmustern [Naum 2001].	D	5%	-	-	-	Suchalgorithmen für alle der 23 Muster aus [Gamm 1995]. Damit können Variabilitäten gefunden werden.
Reverse Engineering von Entwurfsmustern mit Together [Herb 2003].	S	5%	-	-	-	Prototypische Umsetzung von 3 der Algorithmen aus [Naum 2001] in dem CASE-Werkzeug Together.

Tabelle 6.1: Studentische Projekte im Rahmen des digitalen Videoprojektes

Das Projekt zur automatischen Generierung von Anwendungsoberflächen wurde für ein System im Bereich der Rechnungsverarbeitung eines großen deutschen Kommunikationskonzerns benötigt. Die Rechnungsverarbeitung basiert auf einer Systemfamilie, um verschiedene Unternehmensbereiche mit leicht unterschiedlichen Anforderungen effizient bedienen zu können. Für die Familienmitglieder wurde ein Generator entwickelt, mit dem das Verhalten und Aussehen der Anwendungsoberflächen an die Kundenwünsche angepasst werden kann. Durch ein Merkmaldiagramm sind die Auswahl eines Familienmitgliedes und die notwendigen Parametrierungen für den Generator modelliert worden. Aus der Vielfalt der möglichen grafischen Elemente der Anwendungsoberfläche resultiert die Zahl von 79 Merkmalen. Die Beziehungen von FORE wurden genutzt, um Fehler im Verlauf der Auswahl von Merkmalen zu vermeiden.

Das Projekt zur automatischen Generierung geographischer Informationssysteme nutzte Merkmalmodelle bei der Konzeption der Lösung. Es ging um eine Systemfamilie, die lediglich ein variables Datenmodell aufwies. Die FORE-Prozessschritte bis zur „C/V-Analyse“ wurden erfolgreich durchlaufen, wobei die Technologie wie auch die Referenzarchitektur der Systemfamilie feststanden. Ein Familienmitglied wurde in diesem Fall durch die direkte Anpassung des Datenmodells mit Hilfe eines grafischen Editors abgeleitet.

In den letzten beiden Projekten ging es um das *Reverse Engineering* existierender Applikation, das im Schritt „Analyse existierender Systeme“ des FORE-Prozesses notwendig wird, falls keine Dokumentation vorhanden ist. Aus den Erfahrungen der Projekte zeigt sich, dass die Zeit für die Analyse einer

Applikation durch die Kenntnis von eventuell verwendeten Mustern verkürzt werden kann. Darüber hinaus stellen ein Großteil der in [Gamm 1995] vorgestellten Muster Lösungen zur Einbringung von Variabilität in ein Softwaresystem dar. Mit den Algorithmen von [Naum 2001] zur automatisierten Suche nach den 23 Mustern aus [Gamm 1995] und der prototypischen Implementierung von drei der Algorithmen in [Herb 2003] wird die Unterstützung des FORE-Prozesses in der „Analyse existierender Systeme"-Phase verbessert. Tiefergehende Untersuchungen der Algorithmen und der weitere Ausbau der prototypischen Implementierungen sind jedoch Thema zukünftiger Forschungsaktivitäten.

6.3 Bewertung der Lösung

Der FORE-Entwicklungsprozess stellt eine Erweiterung der existierenden Prozesse des Kapitels 3 dar. Im Bereich der Konfiguration, der Parametrierung und der Überprüfung der Merkmalmodelle konnten die Schwächen der untersuchten Ansätze durch FORE behoben werden. Basierend auf dem existierenden *Video Disk Recorder* sind in Tabelle 6.2 die neu entstandenen *Plug-ins* der studentischen Entwicklungsprojekte enthalten. In den ersten Spalten sind die Größen und Entwicklungszeiten der Projekte angegeben, womit sich diese im Bereich mittlerer Größe und Komplexität einordnen lassen.

Projekt	LOC (Lines Of Code)	Klassen	Merkmale	Bearbeitungszeit (Personenmonate)	Anteil der bisher nicht modellierbaren Beziehungen
Basissystem: *Video Disk Recorder*	39945	194	81	unbekannt	62%
Plug-in: Automatische Werbeblockerkennung	3451	29	2	6 PM	57%
Plug-in: Videodatenverteilung	ca. 150	-	4	6 PM	56%
Plug-in: Fernbedienung als Systemfamilie	4097	48	24	6 PM	65%
Plug-in: Verarbeitung von Service Informationen	3493	19	3	3 PM	60%

Tabelle 6.2: Kenngrößen des digitalen Videoprojektes

In der letzten Spalte ist der Anteil der nicht modellierbaren Beziehungen angegeben, wären die Projekte ohne FORE entwickelt worden. Es hat sich gezeigt, dass jeweils über die Hälfte aller Beziehungen nur durch den Einsatz von FORE adäquat berücksichtigt werden konnten. Mit den bisherigen Ansätzen sind Überprüfungen der Beziehungen im Merkmalmodell nur manuell möglich. Die Modellierung aller einschränkenden Anforderungen als formale Beziehungen und die computerunterstützte Auswahl eines Familienmitgliedes bieten einen Vorteil gegenüber den bisherigen Ansät-

zen, indem FORE einen Prozess und ein Datenmodell zur Entwicklung automatisiert überprüfbarer Modelle anbietet.

In Tabelle 6.3 wird FORE mit den in Kapitel 3 untersuchten Ansätzen in zwei Kategorien verglichen, der Auswahl eines Familienmitgliedes, basierend auf der modellierten Systemfamilie und den Möglichkeiten der Überprüfung des Systemfamilienmodells. Als Bezugsgröße wurden jeweils die von FORE angebotenen Leistungsmerkmale herangezogen.

Kategorie		FAST	KobrA	Auf FODA basierende Ansätze	FORE	Kommentar
Auswahl	Merkmale	○	○	⊙	●	Familienmitglieder bzw. Varianten werden in FORE über das erweiterte Merkmalmodell konfiguriert, d.h. die gewünschten Merkmale werden ausgewählt. Darüber hinaus können Parameter für die automatisierte Konfiguration eines Familienmitgliedes angegeben werden.
	Parameter	⊙	⊙	⊙	●	
Überprüfung	Einzelne Merkmale	○	○	○	●	Über die Beziehungen der erweiterten Merkmalmodelle in FORE können alle Abhängigkeiten aus dem Anforderungsmodell und der Implementierung berücksichtigt werden.
	Varianten	⊙	⊙	○	●	Durch die *Feature Constraint Language* sind abgeleitete Familienmitglieder in FORE überprüfbar.

● Wird unterstützt.
⊙ Wird mit Einschränkungen unterstützt.
○ Wird nicht unterstützt.

Tabelle 6.3: FORE im Vergleich zu den untersuchten Ansätzen.

Das Ziel der Entwicklung einer Systemfamilie ist die effiziente Ableitung von Familienmitgliedern, die jeweils den Anforderungen eines Kunden entsprechen müssen. Diese Anforderungen werden mit den Möglichkeiten der Systemfamilie abgeglichen, woraus eine Auswahl der vom Kunden gewünschten Anteile der Systemfamilie entsteht. FAST bietet die manuelle Umsetzung von Anforderungen in die während der Domänenentwicklung erzeugte, domänenspezifische Sprache an, wobei Merkmale in FAST gar nicht vorgesehen sind. Parameter beziehen sich auf die im Familienmitglied einprogrammierten Einstellmöglichkeiten. Die Unterstützung der Auswahl eines Familienmitgliedes im FAST-Prozess ist nur in Form der manuellen Umsetzung von Kundenanforderungen vorgesehen.

KobrA nutzt keine Merkmale und stellt an deren Stelle das *Decision Model*, welches den Kunden zusammen mit dem Entwickler über eine Reihe von Fragen durch die Variabilitäten der Systemfamilie leiten soll. Mit den Fragen werden auch die Parameter zur Einstellung eines Familienmitgliedes erhoben. Ein über die Beantwortung der Fragen ausgewähltes Familienmitglied wird in einem *Decision Resolution Model* abgelegt, das jedoch nicht automatisiert überprüft werden kann.

Die auf FODA basierenden Ansätze bieten Merkmalmodelle an, sind jedoch nicht in der Lage, alle Anforderungen der Systemfamilie durch Merkmale und ihre Beziehungen zu modellieren. Die Auswahl der Merkmale zur Ableitung eines Familienmitgliedes ist möglich, kann aber nicht überprüft werden, sodass die Gültigkeit einer Merkmalauswahl kaum nachvollzogen werden kann. In den studentischen Projekten aus Tabelle 6.2 wird deutlich, dass Merkmalmodelle ohne die Erweiterungen von FORE nicht alle Anforderungen in das Systemfamilienmodell einbetten können, was zu inkonsistenten Modellen führt, die in der Folge von Entwicklern nicht mehr genutzt werden können.

FORE stellt einen neuen Baustein der Systemfamilienentwicklung dar, durch den die untersuchten Ansätze erweitert werden können. FORE verbessert die Ansätze durch die Möglichkeit der automatisierten Überprüfung von Merkmalmodellen. Dadurch werden Fehler im Systemfamilienmodell und bei der Ableitung eines Familienmitgliedes vermieden.

6.4 Grenzen von FORE

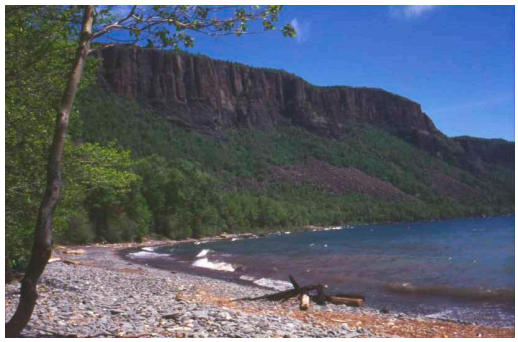
Der in dieser Arbeit vorgestellte Lösungsansatz für die *Requirements-Engineering*-Phase der Systemfamilienentwicklung hat sich in mehreren studentischen Projekten, die in mehreren Anwendungsdomänen durchgeführt wurden, als zielgerichtetes Vorgehen herausgestellt. FORE besteht aus einem Prozess und einem Datenmodell, welche um die erweiterte Merkmalmodellierung angeordnet sind. Die Grenzen von FORE ergeben sich aus den Besonderheiten der Systemfamilienentwicklung selbst und aus den Erfahrungen der bisher durchgeführten Projekte.

- Wie bereits in Kapitel 3 gezeigt, müssen mehr als vier Familienmitglieder abgeleitet werden, bevor sich der anfängliche Aufwand der Entwicklung amortisiert hat. Dabei ist die Größe einer Firma von Bedeutung, da der finanzieller Puffer kleinerer Firmen oftmals für die Zeit der Entwicklung einer Systemfamilie nicht ausreicht.
- Die strukturellen Veränderungen zur Etablierung einer Systemfamilie innerhalb einer Firma können eine Hürde darstellen, die sowohl finanzieller als auch personeller Natur ist. Die Firma muss bereit sein die anfänglichen Kosten der Systemfamilienentwicklung zu tragen und alle Entwickler müssen dem Entwicklungsprozess für Systemfamilien folgen. Trotz dieser Einschränkungen zeigen kommerzielle Systemfamilien von Microsoft, HP oder IBM, dass die Systemfamilienentwicklung gangbar, tragbar und lohnenswert ist.
- FORE wurde für ein Bibliothekssystem, ein Rechnungsverarbeitungssystem, ein geographisches Informationsystem und für das hier ausführlicher dargestellte digitale Videoprojekt genutzt und getestet. Dabei waren bis zu vier studentische Entwickler tä-

tig. Trotz der unterschiedlichen Domänen, für die FORE entwickelt wurde, wird in dieser Arbeit nicht der Anspruch erhoben, FORE in allen denkbaren Domänen anwenden zu können. Jedoch ist FORE so flexibel gehalten, dass es auch an zukünftige Bedürfnisse angepasst werden kann, zum einen durch die frei definierbaren Beziehungen der erweiterten Merkmalmodellierung, zum anderen durch die unterschiedlichen Ansätze und Methoden in den einzelnen Schritten des FORE-Prozesses.

- FORE unterstützt die Überprüfung von Anforderungen und Merkmalen durch Testfälle in der Implementierung derzeit nur rudimentär. FORE leistet keinen kompletten Test der Systemfamilie oder eines Familienmitgliedes. Nur die Auswahl der Merkmale für ein Familienmitglied kann überprüft werden.
- Die spezifischen Anforderungen einer verteilten Entwicklung konnten in FORE bislang nicht berücksichtigt werden. Die Ergebnisse eines konkreten Tests hängen von den genutzten Werkzeugen der verteilten Entwicklung ab. Die Anpassung von FORE an große, verteilte Entwicklergruppen wird zukünftigen Arbeiten überlassen.

Kapitel 7



„ Sleeping Giant“, Ontario, Kanada

Zusammenfassung und Ausblick

Diese Arbeit ist im Umfeld der Systemfamilienentwicklung eingebettet, deren grundsätzliche Idee die Entwicklung einer gesamten Umgebung für die Ableitung von Familienmitgliedern ist. Die Ableitung, auch als Konfiguration bezeichnet, basiert auf einer Referenzarchitektur, die aus Anteilen besteht, die allen Familienmitgliedern gemein sind und Anteilen, die optional sind. Die Unterscheidung zwischen gemeinsamen und optionalen Anteilen zieht sich durch alle Phasen der Entwicklung, was im groben Überblick eine Analyse-, Modellierungs- und Implementierungsphase für die Systemfamilie selbst und die gleichen Phasen für die Ableitung einer Applikation auf Basis der Systemfamilie bedeutet. Wie die gesamte Systemfamilienentwicklung ist auch die *Requirements-Engineering*-Phase in die Bearbeitung der Systemfamilie und die Konfiguration einer Applikation aufgeteilt. Die Kernanteile der Referenzarchitektur und die variablen Komponenten müssen zunächst durch ein Domänenmodell der Systemfamilie beschrieben werden, welches für die weiteren Entwicklungsschritte der Systemfamilie und für die Konfiguration einer Applikation genutzt wird. Allerdings ist dieses Modell unvollständig definiert und kann nicht automatisiert verarbeitet werden. Die existierenden Ansätze FODA, FORM, FOPLE sowie FeatuRSEB nutzen die Merkmalmodellierung, können sie aber nicht zufrieden stellend in einen Entwicklungsprozess integrieren. Diese Probleme der frühen Entwicklungsphasen haben schon bei der Einzelsystementwicklung drastische Folgen und multiplizieren sich in der Systemfamilieentwicklung mit der Anzahl der abgeleiteten Familienmitglieder. Die umfassende und präzise Berücksichtigung der *Requirements-Engineering*-Phase ist wichtig für die nachfolgenden Entwicklungsphasen.

Die Analysen und der Lösungsansatz dieser Arbeit sind auf die *Requirements-Engineering*-Phase der Systemfamilienentwicklung fokussiert. Wie in dieser Arbeit gezeigt, kann das Modell einer Systemfamilie mithilfe der Merkmalmodellierung erstellt werden. Der in dieser Arbeit vorgestellte Lösungs-

ansatz besteht aus der erweiterten Merkmalmodellierung, dem FORE-Entwicklungsprozess und dem FORE-Datenmodell, welches die Ergebnisse des Entwicklungsprozesses aufnehmen kann. Der FORE-Entwicklungsprozess unterstützt einerseits die Entwicklung eines Systemfamilienmodells, welches auf Anforderungen sowie Merkmalen basiert und ermöglicht andererseits die Konfiguration eines Systemfamilienmitgliedes, basierend auf dem erweiterten Merkmalmodell. Ausgehend von den existierenden Merkmalmodellen erweitert FORE diese um Beziehungen, die zwischen Anforderungen, Merkmalen und weiteren Entwurfselementen modelliert werden können. In FORE sind 30 Beziehungen vordefiniert. Es können jedoch beliebige weitere Beziehungen hinzugefügt werden, da Beziehungen in einer neuen Sprache, der *Feature Constraint Language* (FCL), definiert werden. Die FCL ermöglicht die automatisierte Überprüfung der modellierten Beziehungen. Zum einen können dadurch Konsistenzprüfungen des Modells der Systemfamilie durchgeführt werden, zum anderen werden die Beziehungen bei der Konfiguration eines neuen Familienmitgliedes genutzt, um dessen Gültigkeit zu überprüfen.

In der prototypischen Implementierung wurde das FORE-Datenmodell als XML-Struktur umgesetzt, deren Beziehungen als FCL-Ausdrücke formuliert sind, welche durch einen eigenen FCL-Compiler verarbeitet werden. Es entsteht eine Komponente, auf die über eine API zugegriffen werden kann. Der FORE-Prozess wurde in diversen studentischen Projekten mit unterschiedlichen Anwendungsdomänen erfolgreich genutzt und konnte dadurch verbessert werden.

7.1 Zukünftige Arbeiten

Der in dieser Arbeit vorgestellte Ansatz, FORE, wurde in bestimmten Bereichen eingesetzt und hat Grenzen, die in Kapitel 6 aufgezeigt wurden. Zur Erweiterung der Grenzen und Verbesserung von FORE sind weitere Arbeiten notwendig.

- Die Prüfung von Anforderungen wird in FORE textuell beschrieben. Aus den Prüfungen der Anforderungen, die einem Merkmal zugeordnet sind, wird eine Gesamtprüfung formuliert. Für abgeleitete Familienmitglieder müssen alle Prüfungen der Merkmale durchgeführt werden. Ein systemfamilienorientiertes Testvorgehen, welches auch automatisierte Testmethoden berücksichtigt, würde hier eventuell einen Vorteil bedeuten und müsste daher durch ein eigenes Forschungsvorhaben analysiert und umgesetzt werden.
- Die Parametermerkmale werden in FORE für einstellbare Werte von Hardware- und Softwarekomponenten genutzt. Ebenfalls im FORE-Modell enthalten sind Beziehungen der Merkmale zu den Komponenten des UML-Komponentendiagramms.

Hier ist zu prüfen, inwieweit das Komponentendiagramm der UML erweitert werden kann bzw. muss, um die Parameter der jeweiligen Komponenten besser beschreiben zu können. Damit könnte ein Teil der Parametermerkmale durch Referenzen aus dem Merkmalmodell in das UML-Komponentenmodell ersetzt werden.

- Für die Modellierung von Daten nutzt FORE das *Entity-Relationship*-Modell. Es ist hier zu prüfen, ob auch ER-Modelle mit variablen und optionalen Anteilen modelliert werden können und ob sich diese Anteile sinnvoll durch Merkmale beschreiben lassen.
- Für die Analyse existierender Applikationen mit dem Ziel der Integration dieser Applikationen in eine neue Systemfamilie nutzt FORE derzeit die bekannten *Reverse-Engineering*-Methoden. Für das Verständnis existierender Applikationen haben aktuelle Forschungsergebnisse gezeigt, dass Entwurfsmuster in den Applikationen das Verstehen beschleunigen. Die automatisierte, werkzeuggestützte Suche nach Mustern befindet sich bereits im prototypischen Stadium. Hier können Untersuchungen zu existierenden oder neuen, relevanten Mustern für die Systemfamilienentwicklung anschließen, um die Analyse existierender Applikationen in FORE zu verbessern.
- Aus der Analyse existierender Applikationen können in FORE auch erste Hinweise auf die potenzielle, zukünftige Referenzarchitektur der Systemfamilie entstehen. Hier können bereits schon auf Systemfamilien ausgerichtete Applikationen durch *Refactoring* größtenteils in die Systemfamilie übernommen werden. Untersuchungen zu systemfamilienbasierten *Refactoring*-Methoden sind an dieser Stelle sinnvoll.
- Die prototypische Umsetzung ist derzeit in der Lage, die Konfigurationsdateien eines bereits installierten Grundsystems korrekt einzustellen, um ein lauffähiges System zu erhalten. Hier sollte in einem weiteren Forschungsprojekt die Nutzung von FORE für die komplette Installation des digitalen Videosystems angestrebt werden. Dazu ergänzend sollte ein Installationsmedium erstellt werden, dass eine weiterentwickelte Version der prototypischen Implementierung enthält, die die Installation, die Konfiguration des gewünschten Familienmitgliedes und dessen Parametrierung erlaubt.

Anhang – A: DVP Anforderungen

Dieser Anhang enthält die Liste der Anforderungen, die zu Beginn des Projektes aus Prospekten und Internetseiten zusammengestellt worden sind. Die Anforderungen sind hierarchisch gegliedert, was sich in der Nummerierung niederschlägt. In der letzten Spalte sind Abhängigkeiten einer Anforderung zu anderen Anforderungen vermerkt.

Nr.	Anforderung	Erläuterung	Beziehung zu ...
1	Das System soll fernsteuerbar sein.	Hier ist die Grundidee einer Fernbedienung gemeint.	
1.1	Das System soll durch eine Infrarot-Fernsteuerung bedienbar sein.	Es geht nur um die Art der Technologie.	
1.1.1	Es soll eine Infrarot-Fernbedienung nur mit einem Hin-Kanal geben.	Wie herkömmliche Fernbedienungen	
1.1.2	Es soll eine Infrarot-Fernbedienung mit Hin- und Rück-Kanal geben.	Die Fernsteuerung sendet an das System und das System kann auch an die Fernsteuerung senden.	
1.2	Die Fernbedienung soll individuell anpassbar sein.	Jeder Nutzer soll die Fernbedienung an seine Bedürfnisse anpassen können.	Zu 19: Authentifizierung
1.2.1	Vorhandene Bedienelemente der Fernsteuerung sollen vom Nutzer mit einer Funktion belegt werden können.	Einzelne Knöpfe können frei belegt werden.	
1.2.2	Der Nutzer soll eigene Bedienelemente definieren können.		
1.2.3	Komplexe Bedienungsszenarien, bei denen mehrere Funktionen nacheinander aufgerufen werden, sollen vom Nutzer realisierbar / automatisierbar sein.		
1.2.4	Existierende Geräte sollen mit der Fernbedienung ansteuerbar sein.		
1.2.4.1	Es soll ein Lernmodus für existierende Fernbedienungen geben.		
1.2.4.2	Es soll möglich sein existierende Fernbedienungen durch Angabe von Hersteller und Typ in die DVP-Fernbedienung zu integrieren.		
1.3	Das System soll über das Telefon fernsteuerbar sein.		
1.3.1	Das System soll mit Hilfe von SMS Nachrichten steuerbar sein.		
1.4	Das System soll über das Internet steuerbar sein.		
1.4.1	Das System soll über eine Web-Schnittstelle steuerbar sein.		
1.4.2	Das System soll mit Hilfe von E-Mails steuerbar sein.		
1.4.3	Das System soll über ein WAP-Handy steuerbar sein.		
1.5	Die Steuerung des Systems soll über einen Sicherheitsmechanismus für die Einschränkung des Zugriffs verfügen.		
1.5.1	Es muss möglich sein, das System für den unbeabsichtigten Zugriff durch Kinder zu sperren.		Zu 19: Authentifizierung
1.5.2	Es muss möglich sein, das System für den unbefugten Zugriff durch Kinder zu sperren.		Zu 19: Authentifizierung Zu 1.2: Personalisierte Fernbedienung. Es ist klar, wer die Person mit der Fernbedienung ist.

Nr.	Anforderung	Erläuterung	Beziehung zu ...
1.5.3	Eventuelle Altersbeschränkungen von Datenmaterial soll berücksichtigt werden.	FSK	Zu 1.2: Das Alter der Person ist bekannt.
2	Das System muss Audio- und Video-Datenmaterial abspielen können.		
2.1	Das System muss Musik-CDs abspielen können.		
2.2	Das System muss MP3-CDs abspielen können.		
3	Das System muss eine elektronische Fernsehzeitung enthalten.	Der EPG *(Electronic Program Guide) muss unterstützt werden.*	
3.1	Die Sendungen müssen nach Sendern sortiert werden können.		
3.2	Die Sendungen müssen nach Uhrzeit sortiert werden können.		
3.3	Die Sendungen müssen nach Genre sortiert werden können.		
3.4	Der EPG soll in einer Datenbank abgelegt werden.	DB muss auf dem System verfügbar sein.	
3.5	Filme im EPG sollen durch Informationen aus dem Internet angereichert werden.	Stichwort: Internet Movie Database	Braucht Netz, Modem, oder ISDN!
4	Einstellungen sollen individuell anpassbar sein.	Es geht um Bild- und Toneinstellungen.	
4.1	Bild- und Toneinstellungen sollen pro Sender abgelegt werden.	Beim Umschalten soll es nicht mehr zu Bild- oder Tonunterschieden kommen.	
4.2	Jeder Nutzer des Systems soll seine eigenen Einstellungen hinterlegen können.	Nutzer müssen sich anmelden und haben damit ihre eigenen Einstellungen.	
5	Das System soll die „Picture-In-Picture“ Funktion unterstützen.	Zwei Sender gleichzeitig sehen und dabei einen der beiden Sender als verkleinertes Bild auf dem Hauptbildschirm platzieren.	Benötigt zwei Empfänger!
5.1	Es soll möglich sein, Nachrichten automatisch in den Vordergrund zu holen.	Wie beim Radio sollen Nachrichten automatisch auf dem Hauptbildschirm gezeigt werden.	
5.2	Beim automatischen Umschalten soll die aktuelle Sendung angehalten werden, um nachher weitersehen zu können.	Aktuelle Sendung wird auf Platte zwischengespeichert.	zu 6. *„time-shifting“*
6	Mit dem System müssen Sendungen angehalten und zeitversetzt betrachtet werden können. (*„time-shifting“*).	Der Zeitversatz, der beim Anhalten entsteht, wird in einer temporären Datei vorgehalten. Eigentlich sieht man damit die Sendung komplett von der Platte. (LIFO Speicher)	Festplatte muss das mitmachen ...
7	Das System soll Werbeblöcke ausblenden können.	Idee: Schriftzug erkennen, höherer Audiopegel, evtl. Werbecodierung aus den Zwischenzeilen übernehmen.	
7.1	Werbung ausblenden, während die Sendung läuft.	Anforderungen an die CPU sind höher.	
7.2	Werbung aus einer aufgezeichneten Sendung herausschneiden.	Platte muss entsprechend groß sein.	siehe 8.1
8	Das System soll Videomaterial nachbearbeiten können.		

Nr.	Anforderung	Erläuterung	Beziehung zu ...
8.1	Schneiden auch über Schnittmarken.		siehe auch 8.3
8.2	Videoeffekte.		
8.3	Indexpunkte setzen.	Punkte (Schnittmarken) in einem Videostrom setzen, die nachher angesprungen werden können.	
9	Das System soll als Video-On- Demand Server agieren.		
9.1	Das System soll eine Netzwerkkarte enthalten.		
10	Das System soll mehrere Tonformate unterstützen.	Grundsätzlich gibt das System Stereoton aus. Andere Formate müssen entsprechend angepasst, „heruntertransformiert“ werden.	
10.1	Das System soll Dolby Digital unterstützen.		Entsprechende Soundkarte notwendig.
10.2	Das System soll Dolby Surround unterstützen.		Entsprechende Soundkarte notwendig.
11	Das System soll Pay-TV unterstützen.	Die entsprechende Zusatzhardware muss anschließbar und von der SW ansteuerbar sein.	
12	Das System soll Videoeingänge haben.		
12.1	Firewire.		
12.2	SCART mindestens 2 SCART Buchsen.		
13	Das System soll eine automatische Senderprogrammierung haben.		
13.1	Die Signalpegel der Sender sollen angezeigt werden.		siehe 13.2
13.2	Der Empfangspegel der SAT-Schüssel soll angezeigt werden.	Wird genutzt, um die Anlage einfach justieren zu können.	
14	Das System soll Fehler anzeigen.	Sinnvolle Fehlermeldungen für den Benutzer.	siehe auch 15.
15	Das System soll unterschiedliche Sprachen unterstützen.		
16	Das System soll Videomaterial auch auslagern können.		
16.1	CD-ROM / CD-RW.		
16.2	DVD-R.		
16.3	MOD.		
16.4	Videoband.		
17	Das System soll Zusatzgeräte ansteuern können.		
17.1	Barcode Leser für eine externe Videobandverwaltung.	Jedes Video hat einen Barcode zur Identifikation.	
17.2	Automatisches Videoregal mit Videodatenträgern. Datenbank muss erweitert werden.		
17.2.1	Datenträger sollen durch einen Roboterarm gereicht werden.		
17.2.2	Mit LED Anzeige: Es wird angezeigt, wo der gesuchte Datenträger ist.		
17.3	Drucker zur automatischen Beschriftung von Datenträgern.		
17.3.1	Farbig		
17.3.2	S/W		

Nr.	Anforderung	Erläuterung	Beziehung zu ...
18	Das System muss leise sein.		
18.1	Die Festplatte muss leise sein.		
18.2	Der Lüfter muss leise sein falls es einen Lüfter geben muss.		
19	Nutzer sollen sich gegenüber dem System authentifizieren.		
19.1	Nutzer sollen sich mit einem Pin-Code identifizieren.		
19.2	Nutzer sollen sich mit ihrem Fingerabdruck identifizieren.		
19.3	Nutzer sollen sich mit einer Chipkarte identifizieren.		

Tabelle A-1: Liste der Anforderungen für das Video Projekt

Neben den Anforderungen wurden aus den zur Verfügung stehenden Informationen die Merkmale bereits existierender und zukünftig zu entwickelnder Systeme erarbeitet. In Abbildung A-1 sind diese Merkmale in Form eines erweiterten Merkmaldiagramms dargestellt. In der darauf folgenden Abbildung A-2 sind die Beziehungen der Merkmale enthalten, um das Merkmaldiagramm lesbar zu halten. Sowohl die Merkmale als auch die Beziehungen sind vollständig im Merkmalmodell enthalten, wobei in den beiden Abbildung jeweils eine sinnvolle Sicht auf die Daten des Modells dargestellt ist.

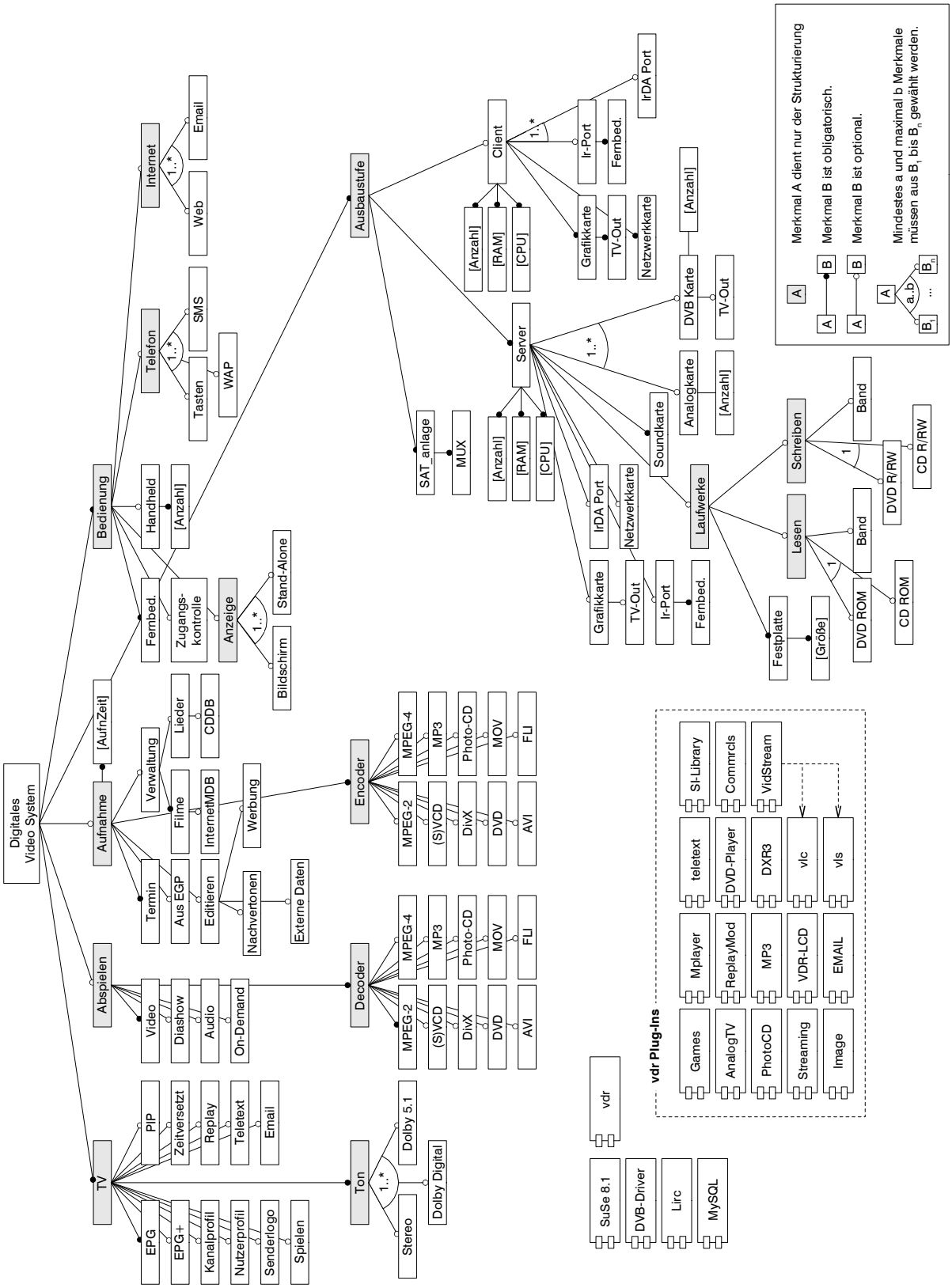


Abbildung A-1: Merkmalmodell des digitalen Videoprojektes

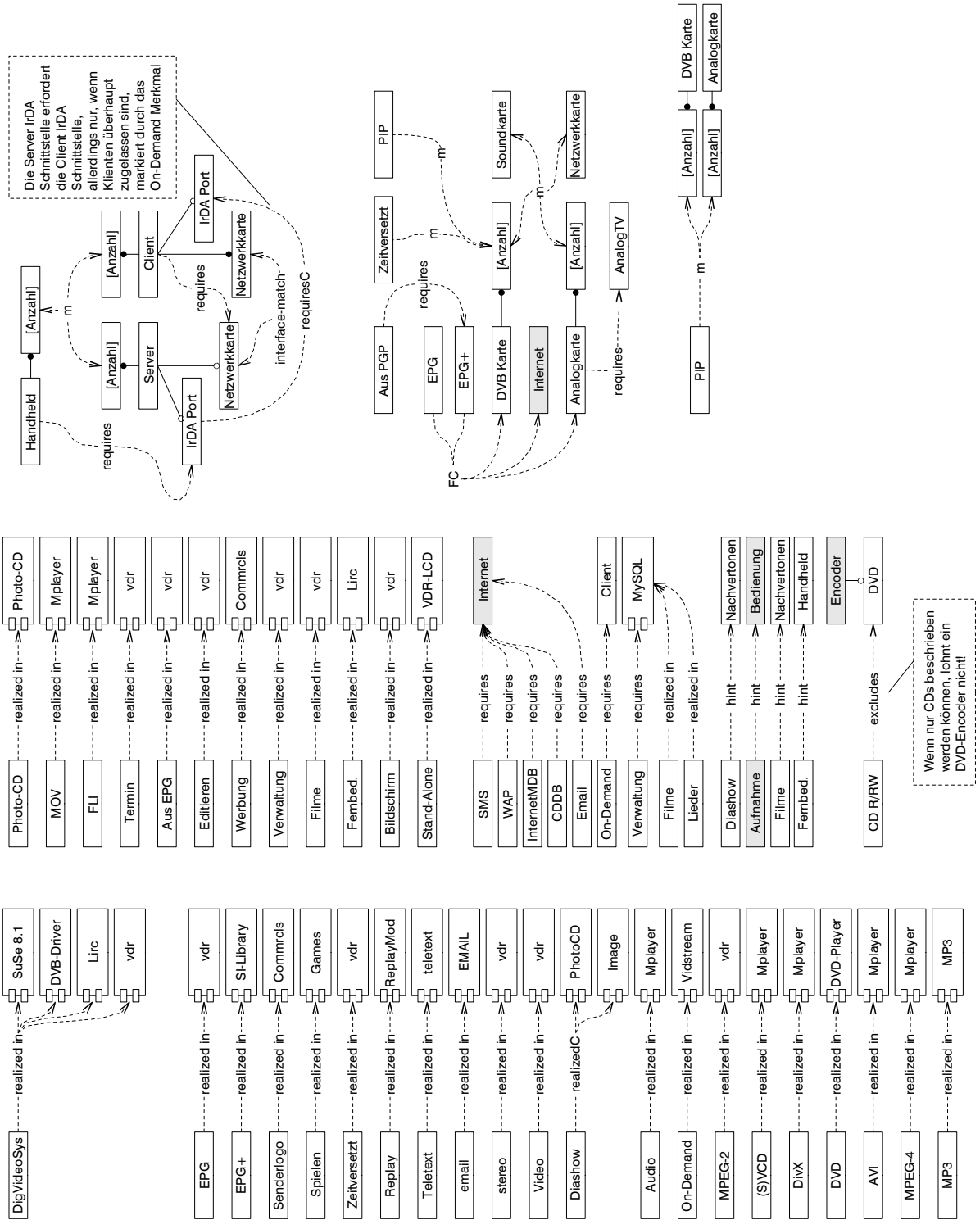


Abbildung A-2: Beziehungen zwischen Merkmalen des digitalen Videoprojektes

Anhang – B: Werkzeuge

INCOSE Werkzeugvergleich (www.incose.org/tools/tooltax.html)

y-volle, n-keine, p-partielle Unterstützung

	Analyst Studio (RequisitePro) v2002	Caliber RM 3.0	C.A.R.E. 3.0	Catalyze 1.0	CORE 4.0	Cradle / SEE 4.0	DOORS 6.0	QSS requireit 1.0	Envision 5.4.2	RMTrak 5.0.4	Team Trace 2.1	Tracer 4.1	RTM 4.x	SLATE 6.1	Systems Engineer 2	Tofs 98	Vital Link	Xtie-RT
Response Date	March 2002	Aug. 2000	June 2002	2002April	2002March	March 2002	March 2002	May 1999	April 2000	April 2002	June 2002	Nov. 2000	June 1999	June. 2002	Jan. 2000	July 1998	Jan. 1997	July 1998
1. Capturing Requirements/Identification																		
1.1. Input document enrichment / analysis	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y	P	N		Y	N
1.1.1. Input document change / comparison analysis	Y	Y	Y	Y	Y	Y	Y	N	Y	Y	P	Y	Y	Y	N	N	P	N
1.2. Automatic parsing of requirements	Y	Y	Y	P	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N	Y	Y
1.3. Interactive/semi-automatic requirement identification	Y	Y	Y	Y	Y	Y	Y	P	Y	Y	Y	Y	Y	P	N	P	Y	Y
1.4. Manual requirement identification	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	P	Y	Y	Y
1.5. Batch mode operation	Y	Y	Y	Y	Y	Y	Y	N	Y	!	Y	P	Y	P	N	P	N	Y
1.5.1. Batch-mode document/source-link update	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y	P		N	P	N
1.6. Requirement classification	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
2. Capturing system element structure															Y			
2.1. Graphically capture systems structure	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	N	Y
2.2. Textual capture of systems structure	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
3. Requirements flowdown																		
3.1. Requirements derivation (req. to req, req. to analysis/text)	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	P	Y	Y
3.2. Allocation of performance requirements to system elements (weight, risk, cost, etc.)	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	P	Y	P	Y	Y	Y	P	Y
3.3. Bi-directional requirement linking to system elements	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y	Y
3.4. Capture of allocation rationale, accountability, test/validation, criticality, issues, etc. -- if so, how and what mechanism does it use.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
4. Traceability analysis																		
4.1. Identify inconsistencies (orphans,...if so, what kind of...)	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
4.2. Visibility into existing links from source to implementation--i.e. follow the links.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
4.3. Verification of requirement (was it done, how was done)	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
4.4. Requirement performance verification from system elements (roll up of actuals)	P	P	P	P	Y	Y	Y	N	Y	P	Y	P	Y	Y	P	P	N	N
5. Configuration Management																		
5.1. History of requirement changes, who, what, when, where, why, how.	Y	Y	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	Y	Y	P	P	P	Y
5.2. Baseline/Version control	P	Y	Y	Y	Y	Y	Y	Y	Y	P	Y	P	Y	Y	N	P	Y	Y
5.3. Access control (modification, viewing, etc.)	P	Y	Y	P	Y	Y	Y	Y	Y	P	Y	P	Y	Y	Y	P	Y	Y
6. Documents and other output media																		
6.1. Standard specification output (if so, what kind)	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	P	Y	Y	Y
6.2. Quality and consistency checking (spell, data dictionary,)	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	P	Y	Y	Y	P	Y	Y	P

	Analyst Studio (RequisitePro) v2002	Caliber RM 3.0	C.A.R.E. 3.0	Catalyze 1.0	CORE 4.0	Cradle / SEE 4.0	DOORS 6.0	QSS requireit 1.0	Envision 5.4.2	RMTrak 5.0.4	Team Trace 2.1	Tracer 4.1	RTM 4.x	SLATE 6.1	Systems Engineer 2	Tofs 98	Vital Link	Xtie-RT
6.3. Presentation output	Y	P	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	P	N	P	Y	Y
6.4. Custom output features & markings (definable tables, security marking)	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	P	P	Y	Y
6.5. WYSIWYG previewing of finished output	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
6.6. Status reporting	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	P	Y	P	Y
6.6.1. Technical Performance Measurement status accounting	Y	Y	Y	P	Y	Y	Y	Y	Y	P	Y	P	Y	Y	P	Y	P	Y
6.6.2. Requirement progress/status reporting	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
6.6.3. Other ad hoc queries and searches	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	P	Y	Y
6.7. Support and display special character sets.		Y	Y						Y		Y	Y						
7. Groupware											Y	Y	!	!				
7.1. Support of concurrent review, markup, and comment	Y	Y	Y	Y	Y	Y	Y	P	Y	P	Y	P	Y	Y	N	Y	Y	Y
7.2. Multi-level assignment/access control	Y	Y	Y	P	Y	Y	Y	N	Y	P	Y	P	Y	Y	P	N	Y	Y
8. Interfaces to other tools																		
8.1. Inter-tool communications	Y	Y	Y						Y				Y	!				Y
8.1.1. Interfaces to other tools?	Y	Y	Y	!	!	!	!	N	Y	Y	Y	!	!	!	N	!	!	!
8.1.2. External Applications Program Interface available	Y	Y	Y	Y	Y	!	Y	N	Y	Y	Y	Y	Y	Y	N	Y	Y	N
8.1.3. Support Open database system (standard query access)	Y	Y	Y	P	P	Y	P	N	Y	Y	Y	Y	Y	Y	Y	!	P	N
8.1.4. Import of existing data from various standard file formats?	Y	Y	Y	!	!	Y	Y	P	Y	Y	Y	P	Y	Y	P	N	P	Y
8.2. Intra-tool communications																!		
8.2.1. Exchange of information between same-tool different installations	Y	Y	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	Y	Y	N	!	Y	Y
8.2.2. Consistency/comparison checking between same-tool datasets	P	N	Y	Y	Y	Y	Y	N	Y	Y	Y	N	P	P	N	N	N	P
9. System Environment																		
9.1. Single user/multiple concurrent users	Y	Y	Y	!	!	!	!	!	!	!	!	!	Y	!	!	!	!	!
9.2. Multiple Platforms/Operating Systems?	Y	Y	Y	!	!	!	!	!	!	!	!	!	!	!	!	!	!	!
9.3. Commercial vs. unique database	Y	Y	Y	!	!	!	!	N	!	!	!	!	!	!	!	!	!	!
9.4. Resource requirements	!														!			
9.4.1. Memory requirements	!	!	!	!	!	!	!	!	!	!	!	!	!	!	!	!	!	!
9.4.2. CPU requirements	!	!	!	!	!	!	!	!	!	!	!	!	!	!	!	!	!	!
9.4.3. Disk space requirements	!	!	!	!	!	!	!	!	!	!	!	!	!	!	!	!	!	!
10. User Interfaces																		
10.1 Doing one thing while you are looking at another	Y	Y	Y	Y	Y	Y	Y	N	P	Y	N	Y	Y	Y	Y	P	Y	Y
10.2 Simultaneous update of open views	Y	Y	Y	Y	Y	Y	Y	N	Y	P	N	P	P	Y	Y	P	Y	Y
10.3 Interactive graphical input/control of data	Y	Y	Y	Y	Y	Y	Y	P	Y	Y	N	Y	Y	Y	N	N	P	N
10.4 Which window standard do you follow?	!	!	!	!	!	!	!	!	!	!	!	!	!	!	!	!	!	!
10.5 Executable via scripts (recordable) or macros	P	Y	Y	P	Y	!	Y	Y	Y	P	Y	P	Y	Y	P	N	P	Y
10.6 Web browser Interface?	Y	Y	Y	Y		Y	Y	N	N	P	Y	P	Y	Y				
10.7 Edit Undo Function Support		P	Y						Y		P	P						
11. Standards--which one do you comply with?	!	!	!	!	!	!	!	!	!	!	!	!	!	!	!	!	!	!
12. Support and maintenance																		
12.1. Warranty	Y	Y	Y	!	!	!	!	!	!	!	!	!	!	!	!	!	!	!
12.2. Network license policy	Y	Y	Y	Y	Y	Y	!	!	Y	!	!	!	Y	!	!	!	!	!

	Analyst Studio (RequisitePro) v2002	Caliber RM 3.0	C.A.R.E. 3.0	Catalyze 1.0	CORE 4.0	Cradle / SEE 4.0	DOORS 6.0	QSS requireit 1.0	Envision 5.4.2	RMTrak 5.0.4	Team Trace 2.1	Tracer 4.1	RTM 4.x	SLATE 6.1	Systems Engineer 2	Tofs 98	Vital Link	Xtie-RT
12.3. Maintenance and upgrade policy	Y	Y	Y	!	!	!	!	!	Y	!	!	!	!	!	!	!	!	!
12.4 On-line help	Y	Y	Y	Y	Y	Y	Y	!	Y	!	!	!	Y	!	!	!	Y	Y
12.5 Internet access/World Wide Web home page location	Y	Y	Y	Y	Y	!	!	!	Y	!	!	!	!	!	!	!	!	!
12.6 Phone support	!	Y	Y	Y	Y	!	Y	!	Y	!	!	!	!	!	!	!	Y	Y
12.7Support User's Group		Y	Y						N		Y	N						
13. Training																		
13.1 Tool specific training classes.	Y	Y	Y						Y		!	!						
13.2 Training available at customer's location.	Y	Y	Y						Y		!	!						
13.3 Recommended training time	Y	Y	Y	!	!	!	!	!	Y	!	!	!	!	!	!	!	!	!
13.4 Software installation with only basic training.	!	Y	Y						Y		!	!						
14. What other requirements management features do you as a tool supplier think are important (modeling, etc.)?	!	!	!	!	!	!	!	!	!	!	!	!	!	!	!	!	!	!

Anhang – C: Prozesshandbuch

Dieser Anhang enthält ein Prozesshandbuch zum FORE-Entwicklungsprozess. Jeder einzelne Prozessschritt ist enthalten und kurz beschrieben, zusammen mit Empfehlungen konkreter Ansätze zur Bearbeitung der beschriebenen Aufgaben. Auf den folgenden Seiten wird zunächst das *Requirements Engineering* für die Systemfamilie und dann das *Requirements Engineering* für eine Applikation erläutert. Die Anordnung der einzelnen Prozessschritte ist im Rahmen dieser Arbeit entstanden und integriert bestehende Ansätze zu einem Gesamtprozess, der im Überblick in Abbildung C-1 dargestellt ist.

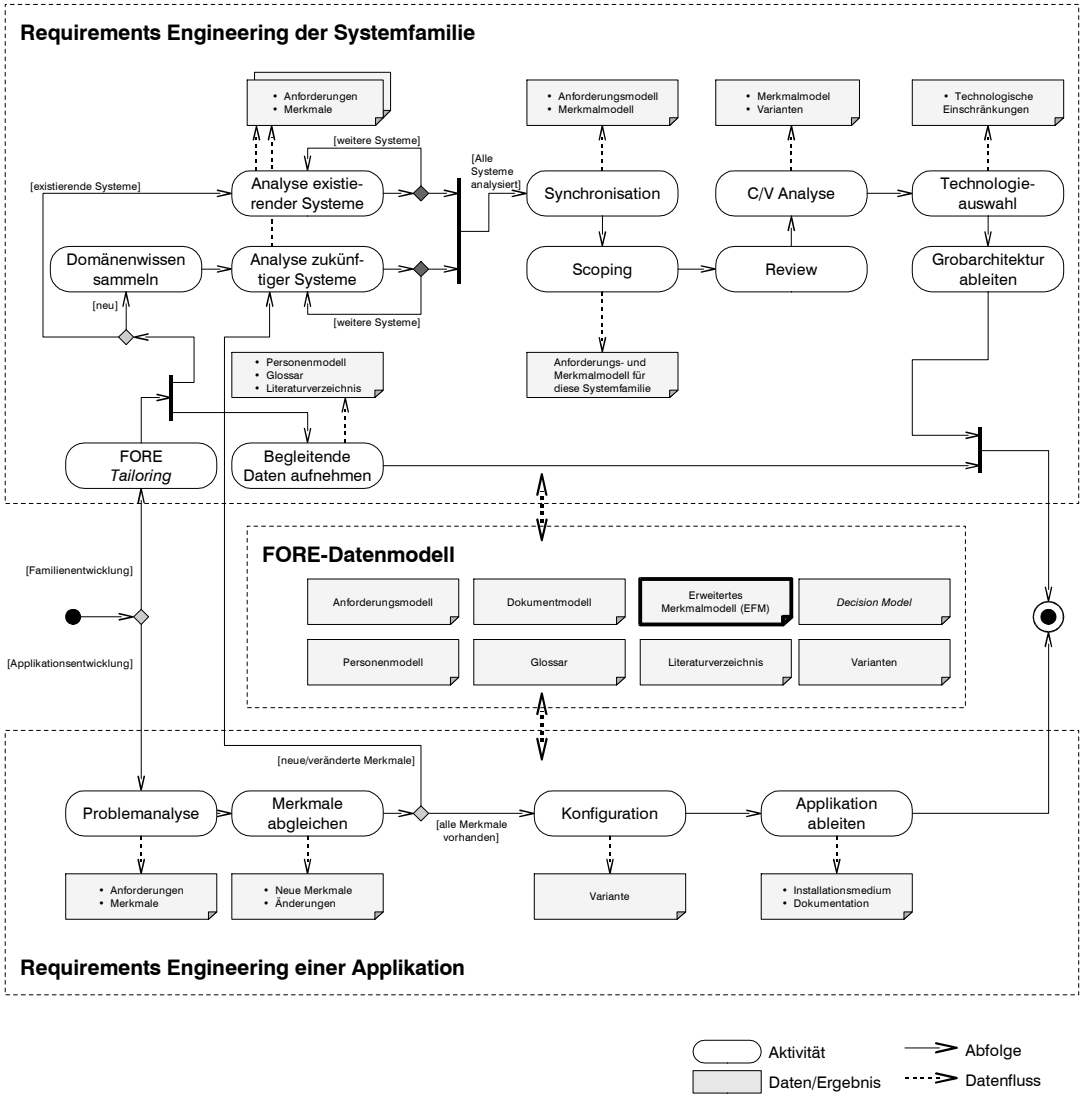
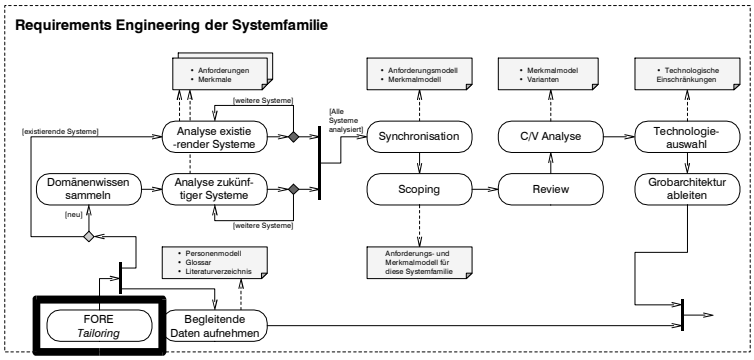
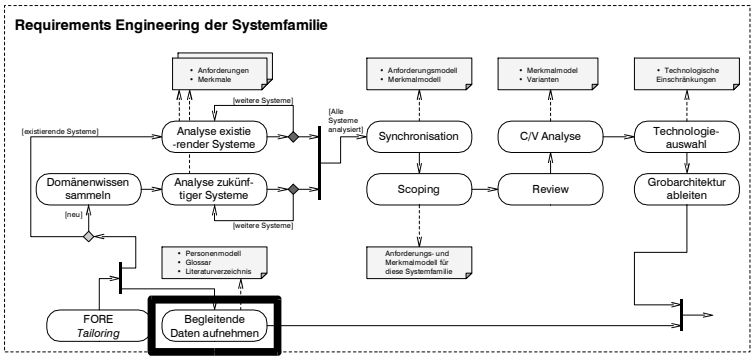


Abbildung C-1: FORE-Entwicklungsprozess

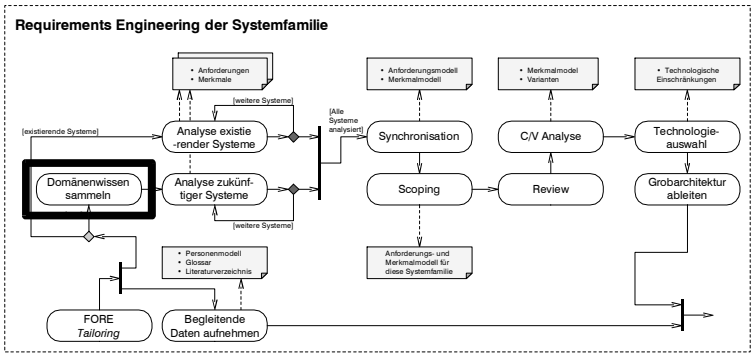
Alle auf den folgenden Seiten mit „**NEU:**“ gekennzeichneten Zeilen stellen die in dieser Arbeit neu entstandenen und in den Prozess eingebrachten Anteile dar.



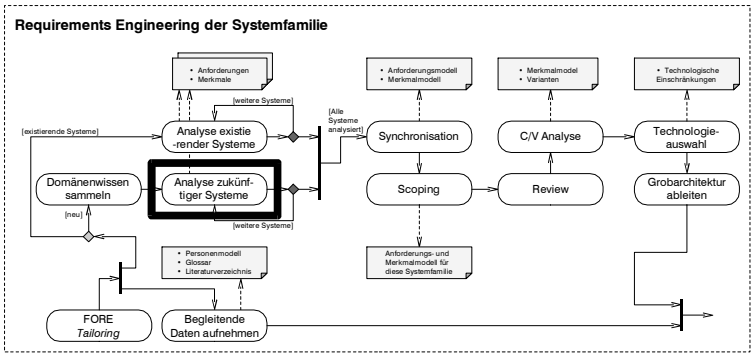
Prozessschritt	**FORE *Tailoring***
Beschreibung	Kann die FORE-Dokumentstruktur nicht genutzt werden, muss ein eventuell vorhandener, firmeneigener Standard für Spezifikationsdokumente, durch ein XML-Schema beschrieben werden. Dabei ist die grundsätzliche Struktur der Kapitel von Bedeutung. Sie wird mit Hilfe der im *DocBook* Standard festgelegten Elemente beschrieben und kann in der Folge von den im weiteren Verlauf der Entwicklung entstehenden Informationselementen referenziert werden. Des Weiteren müssen die Strukturen für Literaturverzeichniseinträge und Glossareinträge definiert werden, falls die für *StarOffice 6.0* vordefinierten Strukturen nicht verwendet werden.
Vorbedingungen	• Die Dokumentstruktur des Spezifikaionsdokumentes muss bekannt sein, falls nicht die FORE-Dokumentstruktur genutzt wird. • Kenntnisse über XML-Schemata müssen vorhanden sein. • Es muss klar sein, mit welchem Werkzeug die Spezifikation verarbeitet werden wird, bzw. in welches Ausgabeformat die Informationen aus dem Datenmodell überführt werden. Dementsprechend muss die Struktur des Literaturverzeichnisses des Glossars aufgebaut werden.
Genutzte Methoden und Techniken	• Entweder die FORE-Dokumentstruktur, Struktur für Literaturverzeichniseinträge und Glossareinträge nutzen, • oder eine eigene Dokumentstruktur für Spezifikationsdokumente, Literaturverzeichniseinträge und Glossareinträge durch ein XML-Schema beschreiben. (➔ Das FORE-Schema als zu erweiternde Basis nutzen)
Ergebnis	• Ein XML-Schema der Dokumentstruktur des Spezifikationsdokumentes, der Struktur für Literaturverzeichniseinträge und Glossareinträge.



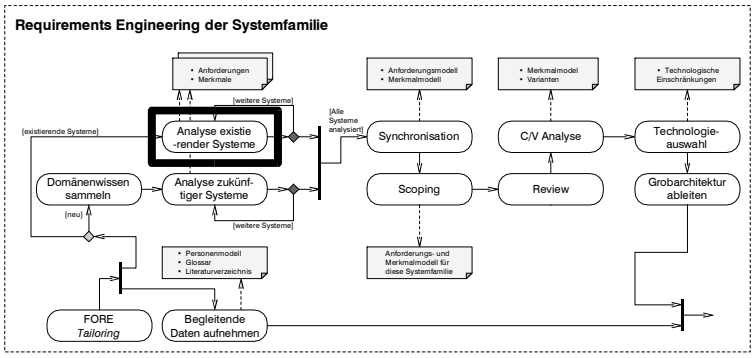
Prozessschritt	**Begleitende Daten aufnehmen**
Beschreibung	Parallel zu allen anderen Aktivitäten können zu jedem Zeitpunkt der Entwicklung begleitende Daten aufgenommen werden. Dies sind Glossareinträge, Literaturverzeichniseinträge oder Personen, die in das Personenmodell aufgenommen werden.
Vorbedingungen	• Die Strukturen für Glossareinträge und Literaturverzeichniseinträge müssen definiert sein.
Genutzte Methoden und Techniken	• Eintragen der Daten in die vorgegebenen Strukturen.
Ergebnis	• Das Glossar, das Literaturverzeichnis und das Personenmodell.



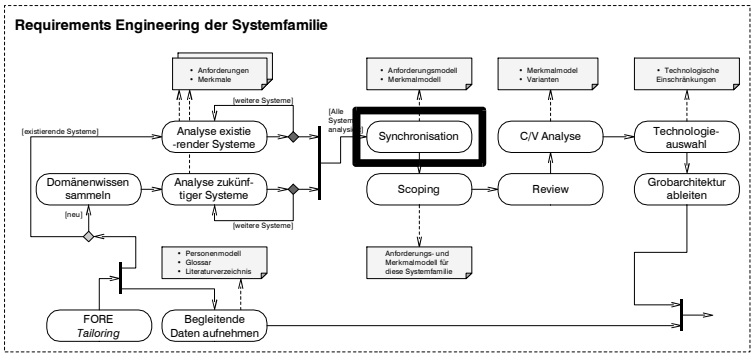
Prozessschritt	**Domänenwissen sammeln**
Beschreibung	Gibt es bei einer Entwicklermannschaft keine Erfahrungen bereits abgeschlossener Projekte, muss das eventuell bestehende Grundwissen über die Domäne erweitert werden.
Vorbedingungen	• keine
Genutzte Methoden und Techniken	• **NEU:** Einführende Kurse zur Systemfamilienentwicklung mit einem begleitenden Beispielprojekt. • *Workshops* zur Angleichung des Wissens der einzelnen Teilnehmer. Siehe Kapitel 3.1.1. • Lehrveranstaltungen, Kurse, Literaturstudium zu Themen der Anwendungsdomäne, siehe Kapitel 3.1.1. • Dokumentanalyse, siehe Kapitel 3.1.1. **NEU**: Bei einer Systemfamilie zur Unterstützung betrieblicher Abläufe müssen die benutzten Formulare untersucht und auf Ähnlichkeiten hin analysiert werden. Die Ergebnisse werden im Spezifikationsdokument in Kapitel 6.3 eingetragen. • Interviews mit den Projektbeteiligten. Siehe Kapitel 3.1.1. **NEU:** Es müssen explizit Fragen zu möglichen Varianten/Abwandlungen gestellt werden. • Missverständliche und/oder neue Begriffe der Domäne in das Glossar eintragen. • Anforderungen, die wichtige Entscheidungen erfordern im *Decision Model* hinterlegen und referenzieren. Siehe Kapitel 3.2.1. • Die Inhalte der Kapitel 1, 1.1, 1.3, 1.4, 1.6, 1.7, 3.7 und 3.8 des Spezifikationsdokumentes verfassen. • Erkannte Risiken notieren.
Ergebnis	• Eine kompetente Entwicklermannschaft. • Das erweiterte Glossar. • Hinweise auf Varianten der zukünftigen Systemfamilie. • Erste Inhalte des Spezifikationsdokumentes.



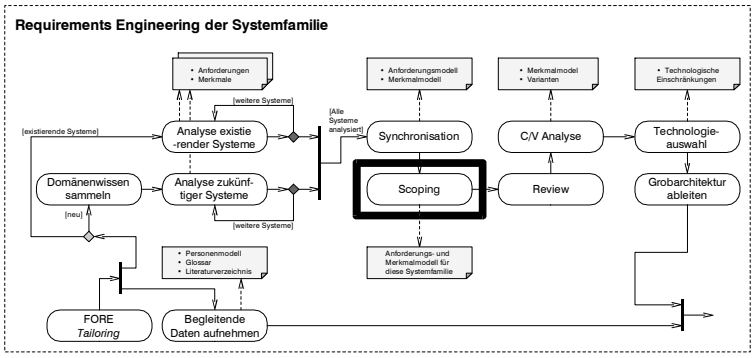
Prozessschritt	**Analyse zukünftiger Systeme**
Beschreibung	Die Entwicklung einer Systemfamilie ohne bereits existierende Applikationen erfordert die Ananlyse potentiell möglicher Produkte.
Vorbedingungen	• Eine Systemfamilie soll entwickelt werden, ohne dass bereits Applikationen in der Domäne verfügbar sind. • Die Entwicklermannschaft hat genügend Wissen über die Domäne der Systemfamilie.
Genutzte Methoden und Techniken	• Marktanalysen und Umfragen zur Analyse der Anforderungen zukünftiger Kunden. Siehe Kapitel 3.1. • *Brainstorming*-Sitzungen der Entwickler. Siehe Kapitel 3.1.1. **NEU:** Jede neue Produktvariante wird durch die Variante des FORE-Datenmodells modelliert. Bereits bekannte Anforderungen und Merkmale ebenfalls aufnehmen und einschränkende Anforderungen durch die in Kapitel 4.2 erläuterten Beziehungen modellieren. **NEU:** Komponenten, die den Entwicklern bekannt sind, müssen in eine Liste aufgenommen werden und bei Bedarf schon von Merkmalen referenziert werden. • Diskussion der zu verwendenden Technologien für die Systemfamilie. Siehe Kapitel 4.3.1. • Daten und Datenflüsse durch *Entity-Relationship*-Modelle und Datenflussdiagramme modellieren. Siehe Kapitel 3.1.2. • Verhalten durch entsprechende Diagramme der UML modellieren. Siehe Kapitel 3.1.2. • Funktionale Anforderungen wo möglich durch *use-cases* beschreiben. Siehe Kapitel 3.1.1. • Ziele mit Hilfe von *Goal-Models* darstellen. Siehe Kapitel 3.1.2. • **NEU:** Anforderungen, Merkmale und bekannte Beziehungen im FORE Datenmodell festhalten. Siehe Kapitel 4.2. • **NEU:** Kapitel 3.1 bis 3.6 und Kapitel 5 des FORE-Spezifikationsdokumentes erarbeiten.
Ergebnis	• **NEU:** Erste Eintragungen in das Datenmodell, basierend auf dem FORE-Datenmodell, mit Varianten, Merkmalen und Anforderungen. • Verweise auf Komponenten der Systemfamilie. • Hinweis auf die zu verwendende Technologie.



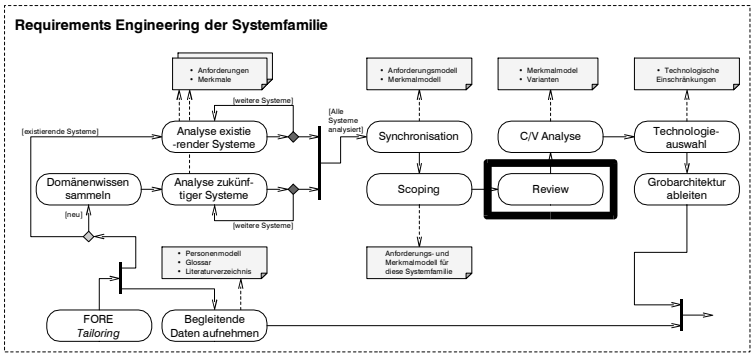
Prozessschritt	**Analyse existierender Systeme**
Beschreibung	Die Entwicklung einer Systemfamilie mit bereits existierenden Applikationen erfordert deren Analyse.
Vorbedingungen	• Eine Systemfamilie soll basierend auf existierenden Applikationen entwickelt werden. • Die Entwicklermannschaft hat genügend Wissen über die Domäne der Systemfamilie.
Genutzte Methoden und Techniken	• Architekturanalyse der existierenden Applikationen. **NEU:** Für Systemfamilien taugliche Architekturen herausfiltern, als Hinweis für die später zu erstellende Referenzarchitektur. • Interviews mit den ehemaligen Entwicklern. Siehe Kapitel 3.1.1. • **NEU:** Quellcodeanalyse zum Auffinden von Variabilitäten und extrahierbaren Komponenten. • *Reverse Engineering* existierender Applikationen. Siehe [Naum 2001] und [Herb 2003]. • Analyse der verwendeten Technologien der existierenden Applikationen. • Daten und Datenflüsse durch *Entity-Relationship*-Modelle und Datenflussdiagramme modellieren. Siehe Kapitel 3.1.2. • Verhalten durch entsprechende Diagramme der UML modellieren. Siehe Kapitel 3.1.2. • Funktionale Anforderungen wo möglich durch *use-cases* beschreiben. Siehe Kapitel 3.1.1. • Ziele mit Hilfe von *Goal-Models* darstellen. Siehe Kapitel 3.1.2. • **NEU:** Anforderungen, Merkmale und bekannte Beziehungen im FORE Datenmodell festhalten. Siehe Kapitel 4.2. • Kapitel 3.1 bis 3.6 und die Einleitung des Kapitels 5 des Spezifikationsdokumentes erarbeiten.
Ergebnis	• **NEU:** Erste Eintragungen in das Datenmodell, basierend auf dem FORE-Datenmodell, mit Varianten, Merkmalen und Anforderungen. • Verweise auf Komponenten der Systemfamilie. • Hinweis auf systemfamilientaugliche Architekturen der existierenden Applikationen. • Hinweis auf die zu verwendende Technologie.



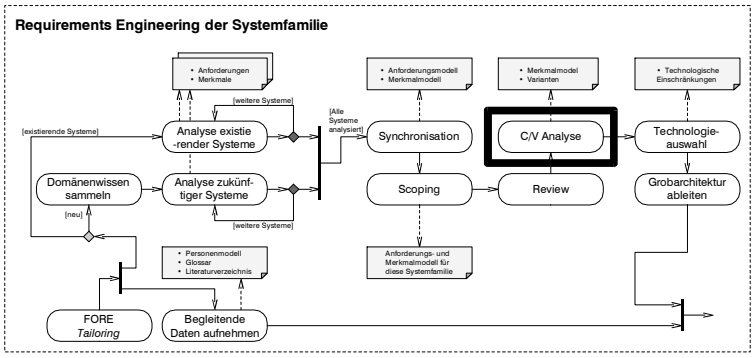
Prozessschritt	**Synchronisation**
Beschreibung	Die Daten der Analyse der Applikationen müssen konsolidiert werden. Die jeweils produktbezogen erhobenen Daten werden in einem Datenmodell zusammengefasst und auf einen konsistenten Stand gebracht.
Vorbedingungen	• Alle avisierten Varianten der Systemfamilie sind analysiert.
Genutzte Methoden und Techniken	• Ähnlichkeiten bei Anforderungen finden und Dubletten eliminieren. **NEU:** Die Daten im FORE-Datenmodell aktualisieren. Beim Löschen von Dubletten die Beziehungen zwischen Modellelementen übernehmen. • Ähnlichkeiten bei Merkmalen finden und Dubletten eliminieren. **NEU:** Die Daten im FORE-Datenmodell aktualisieren. Beim Löschen von Dubletten die Beziehungen zwischen Modellelementen übernehmen. • Glossareinträge abgleichen. • Risikoanalyse mit der Kenntnis um die Varianten der Familie erweitern. • Kapitel 1.5, 3.1,5, 6.4 und 6.5 des FORE-Spezifikationsdokumentes vervollständigen.
Ergebnis	• **NEU:** Die erste Version des Datenmodells, basierend auf dem FORE-Datenmodell, mit entsprechenden Beziehungen zwischen Varianten, Merkmalen und Anforderungen.



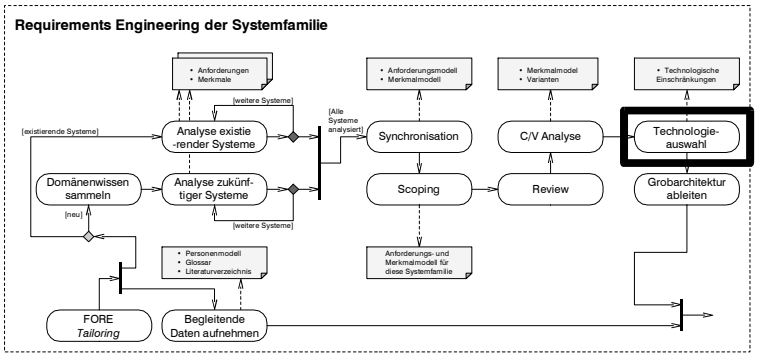
Prozessschritt	***Scoping***
Beschreibung	Es muss geprüft werden, welche Anforderungen, Merkmale und Varianten sich innerhalb der Grenzen der Systemfamilie befinden. Diese Entscheidung hängt dabei von der Ausrichtung des Softwareunternehmens und den zur Verfügung stehenden Ressourcen ab. Das Originalmodell mit allen Merkmalen, Anforderungen und Varianten wird dabei für eventuelle zukünftige Erweiterungen abgespeichert.
Vorbedingungen	• Es muss ein komplettes FORE-Datenmodell existieren.
Genutzte Methoden und Techniken	• Diskussionen • Aufwandsschätzungen. Siehe Kapitel 3.1.4. • Das Spezifikationsdokument um die Kapitel 2.1 und 6.6 ergänzen.
Ergebnis	• Ein reduziertes Datenmodell, welches die relevanten und zu realisierenden Merkmale, Anforderungen und Varianten enthält.



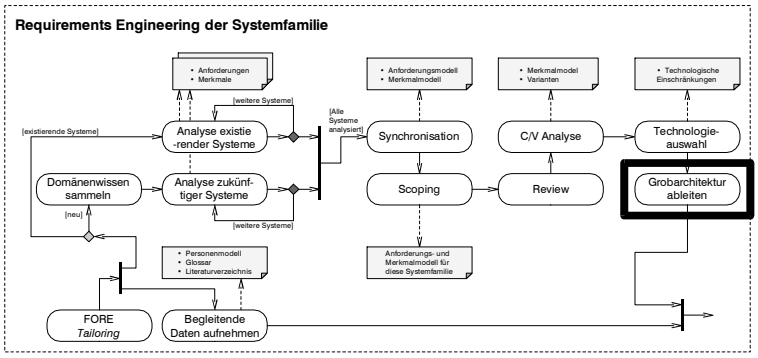
Prozessschritt	***Review***
Beschreibung	Hier wird für die bisherigen Ergebnisse überprüft, ob die Informationen im FORE-Datenmodell korrekt und komplett aufgenommen sind. Darüber hinaus werden die Textpassagen des Spezifikationsdokumentes überprüft.
Vorbedingungen	• Das FORE-Datenmodell ist vollständig und die Texte für die Spezifikation sind erstellt.
Genutzte Methoden und Techniken	• **NEU:** Daten der Anforderungskarten und Merkmalkarten mit dem Datenmodell abgleichen. • Unklarheiten, Missverständnisse und Fehler markieren, diskutieren und beseitigen. • Eventuell fehlende Personen in das Personenmodell aufnehmen • **NEU:** Die Textpassagen des FORE-Spezifikationsdokumentes überprüfen.
Ergebnis	Das überprüfte Datenmodell und eine geprüfte Spezifikation.



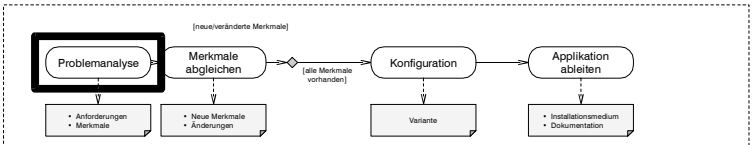
Prozessschritt	**C/V-Analyse**
Beschreibung	Mit dem Wissen der Domäne und den Informationen des FORE-Datenmodells wird in diesem Schritt entschieden, welche der Merkmale zum Kern der Referenzarchitektur gehören und welche Merkmale variabel bleiben.
Vorbedingungen	• Ein vollständiges und überprüftes FORE-Datenmodell ist vorhanden.
Genutzte Methoden und Techniken	• Grundlegende Konzepte gehören in den Kern. • Anforderungen und Merkmale, die bei mehr als 70% der Varianten auftreten, sollten dem Kern der Familie zugeordnet werden. • Anforderungen und Merkmale, die bei bei unter 30% der Varianten auftreten, sollten optional modelliert werden. • **NEU:** Hier werden die Hinweise auf die Referenzarchitektur aufgenommen und weiter konkretisiert. Diese Hinweise am Anfang des 5. Kapitels des FORE-Spezifikationsdokumentes einfügen.
Ergebnis	• Ein vollständiges Modell der Familie, basierend auf dem FORE-Datenmodell mit Anforderungen, Merkmalen, Varianten und den Beziehungen der Elemente untereinander.



Prozessschritt	**Technologieauswahl**
Beschreibung	Die Technologie, die später zur Realisierung der Systemfamilie genutzt wird, hat Einfluss auf das Merkmalmodell der Familie. Die technologisch bedingten Einschränkungen spiegeln sich direkt in den Beziehungen der Merkmale wider. Die Technologie muss gewählt und durch Beziehungen im Modell der Familie berücksichtigt werden.
Vorbedingungen	• Analyse der existierenden und zukünftigen Applikationen abgeschlossen. • Die Entwickler kennen sich mit den relevanten Technologien aus.
Genutzte Methoden und Techniken	• Die *Plug-in*-Architektur ➞ Die Variabilitäten der Familie werden als Komponenten realisiert, die über die *Plug-in*-API mit dem Kern in Verbindung stehen. ➞ **NEU:** Die entsprechenden Merkmale werden mit den Komponente verbunden. • Komponentenmodelle wie COM oder JavaBeans ➞ Nutzung, Integration und Erstellung der Komponenten sind durch deren Modell vorgegeben. ➞ Die Familienarchitektur muss Umgebungsparameter des Komponentenmodells berücksichtigen. ➞ **NEU:** Komponenten werden parametriert, was im Modell durch Parametermerkmale und deren mathematische Beziehungen modelliert wird. • Konfigurationsdateien ➞ **NEU:** Merkmale können über Einstellungen in Konfigurationsdateien realisiert werden. Merkmale verschiedener Komponenten können sich überschneiden, was durch Beziehungen modelliert werden muss. ➞ **NEU:** Ein Merkmal kann durch mehrere Komponenten realisiert werden. Über Beziehungen wird modelliert, welche Komponenten bei welcher Merkmalauswahl in das System integriert werden soll. • **NEU:** Avisierte Technologie am Anfang des Kapitels 4 des Spezifikationsdokumentes ergänzen.
Ergebnis	• Ein Datenmodell, das die zu nutzende Technologie berücksichtigt und als Ausgangspunkt für die Generierung von Familienmitgliedern genutzt werden kann.

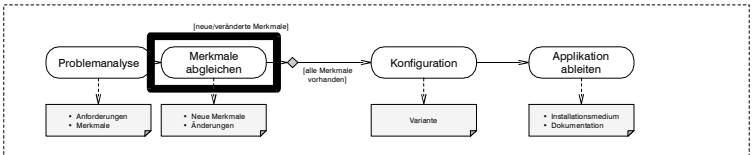


Prozessschritt	**Grobarchitektur ableiten**
Beschreibung	Mit dem Wissen um die verwendeten Komponenten und die zu nutzende Technologie wird an dieser Stelle eine grobe Architektur entwickelt, die Referenzarchitektur der Systemfamilie. In dieser Architektur sind die existierenden Komponenten und deren Beziehungen untereinander modelliert. Die Referenzarchitektur der Familie wird mit Hilfe der Strukturdiagramme der UML notiert, wobei die betroffenen Kernmerkmale mit den Architekturelementen verbunden werden müssen. Die variablen Anteile können durch das Komponentendiagramm der UML dargestellt werden. Dabei sind alle vorhandenen Komponenten, die aus existierenden Applikationen stammen, ersichtlich. Weitere Modellelemente der Architektur müssen damit neu entwickelt werden, wodurch sich der Aufwand für die Entwicklung der Systemfamilie abschätzen lässt.
Vorbedingungen	• Die Technologie zur Implementierung der Systemfamilie muss bekannt sein.
Genutzte Methoden und Techniken	• Komponentendiagramm der UML zur Modellierung der genutzten Komponenten mit Beziehungen untereinander. • **NEU:** Fehlende Verbindungen zwischen Merkmalen/Anforderungen und den Komponenten einfügen. • **NEU:** Die Grobarchitektur am Anfang des 5. Kapitels des FORE-Spezifikationsdokumentes ergänzen.
Ergebnis	• Vervollständigte Spezifikation.



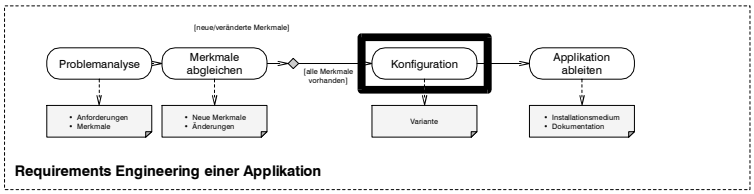
Requirements Engineering einer Applikation

Prozessschritt	**Problemanalyse**
Beschreibung	Es soll festgestellt werden, ob es prinzipiell möglich ist, den Kunden mit der zur Verfügung stehenden Systemfamilie zufrieden zu stellen. Der Kunde muss auf die Systemfamilie mit ihren Merkmalen hingewiesen werden, um zu versuchen, so viele Merkmale wie möglich aus der Familie zu nutzen.
Vorbedingungen	• Das Merkmalmodell muss dem „Verkäufer“ gut bekannt sein. • Der Kunde steht als Diskussionspartner zur Verfügung.
Genutzte Methoden und Techniken	• Erhebung sehr globaler Anforderungen mit einer Wertung bzw. Wichtung der Anforderungen. • Anbieten der Merkmale und Variationen der Systemfamilie. Hier werden neue Varianten „ausprobiert“. **NEU:** Fehler werden sofort durch die modellierten Beziehungen deutlich. Der Kunde lernt die Systemfamilie und deren Möglichkeiten schnell und effizient kennen.
Ergebnis	• Der Kunde kann (oder kann nicht) durch die Merkmale der Systemfamilie bedient werden.



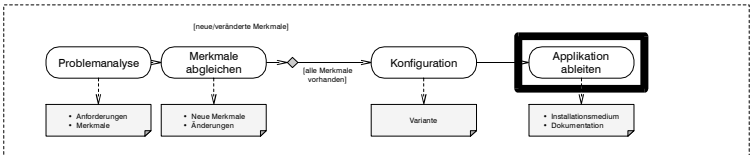
Requirements Engineering einer Applikation

Prozessschritt	**Merkmale abgleichen**
Beschreibung	Die erhobenen Kundenanforderungen und Merkmale werde mit denen der Systemfamilie abgeglichen. Es ist sinnvoll bei Merkmalen, die nicht in der vom Kunden gewünschten Form in der Familie vorhanden sind, einen Kompromiss anzustreben. Die Systemfamilie bringt für beide Seiten den größten Gewinn, wenn alle Merkmale durch die Familie abgedeckt werden können.
Vorbedingungen	• Ein neues Familienmitglied soll abgeleitet werden. • Der Kunde steht als Diskussionspartner zur Verfügung.
Genutzte Methoden und Techniken	• **NEU:** Für neue Merkmale muss eine Aufwandsschätzung durchgeführt werden. Grobe Werte liefert dabei die Summation der Aufwände der bestehenden Merkmale. Siehe Kapitel 3.1.4. • Alle der vom Kunden gewünschten Merkmale werden identifiziert.
Ergebnis	• Ein Überblick der bestehenden und gewünschten Merkmale. • Anforderungen an neu zu implementierende Systemanteile. • Eine Aufwandsschätzung als finanzielle Entscheidungsgrundlage und zeitlicher Rahmen für den Käufer.



Requirements Engineering einer Applikation

Prozessschritt	**Konfiguration**
Beschreibung	Die Auswahl der Merkmale des Kunden wird in diesem Schritt als eigene Variante im FORE-Datenmodell hinterlegt. Die im Modell vorhandenen Parametermerkmale müssen mit Werten belegt werden.
Vorbedingungen	• Alle Merkmale des neuen Familienmitgliedes sind vorhanden. • Der Kunde steht als Diskussionspartner zur Verfügung.
Genutzte Methoden und Techniken	• **NEU:** Alle gewählten Merkmale müssen parametriert werden. Die Parametermerkmale und Beziehungen des Merkmalmodells werden dazu herangezogen.
Ergebnis	• Die komplett parametrierte und aufgrund der Beziehungen im Modell gültige, neue Variante.



Requirements Engineering einer Applikation

Prozessschritt	**Applikation ableiten**
Beschreibung	Die gültige Variante der Systemfamilie wird nun als reales System umgesetzt. Ist die Familie komplett implementiert, kann je nach genutzter Technologie, ein System direkt abgeleitet werden.
Vorbedingungen	• Geprüfte und parametrierte Variante steht zur Verfügung. • Zur kompletten Ableitung müssen alle Teile der Systemfamilie implementiert sein.
Genutzte Methoden und Techniken	• Die Methoden orientieren sich an der eingesetzten Technologie.
Ergebnis	• Eine lauffähige, den Kundenwünschen angepasste Variante der Systemfamilie.

FORE-Dokumentstruktur (Inhaltsverzeichnis)

1. Einschränkungen (Product Constraints)
   1.1. Sinn und Zweck des Systems
   1.2. Alle Stakeholder mit Beziehungen
   1.3. Nutzer des Produktes
   1.4. Einschränkende Anforderungen
   1.5. Kodiervorgaben und Definitionen
   1.6. Relevante Fakten
   1.7. Annahmen
2. Funktionale Anforderungen
   2.1. Umfang des Produktes
   2.2. Funktionale und Datenanforderungen
3. Nicht-funktionale Anforderungen
   3.1. Look and Feel der Anwendung
   3.2. Anforderungen an die Benutzbarkeit
   3.3. Anforderungen an die Performanz
   3.4. Anforderungen an den Betrieb des Systems
   3.5. Anforderungen an die Wartbarkeit und die Portabilität
   3.6. Anforderungen an die Sicherheit
   3.7. Kulturelle und politische Anforderungen
   3.8. Gesetzliche Anforderungen
4. Merkmalmodell
   4.1. Merkmale
   4.2. Varianten
5. Genutzte Komponenten mit Schnittstellen
6. Zusätzliche Punkte (Project Issues)
   6.1. Offene Punkte
   6.2. Neue Probleme
   6.3. Arbeitsabläufe
   6.4. Migration alter Datenbestände
   6.5. Risiken
   6.6. Kosten
   6.7. Benutzerdokumentation
   6.8. Alles, was übrig war ...
7. Glossar
8. Literaturverzeichnis

FORE-Anforderungskarte zur Erhebung von Anforderungen

Anforderung Nr.: *Eindeutige Nummer*	**Dokumentzuordnung:** *Einordnung in das FORE-Spezifikationsdokument (Kapitel)*	**Ereignis/Use Case Nr.**: *Zuordnung*
Komponente: Welcher Komponente ist diese Anforderung zugeordnet?	**Variante:** Welcher Variante ist diese Anforderung zugeordnet?	
Beschreibung:	*Ein Satz, der die Anforderung beschreibt.*	
Bedeutung:	*Warum ist diese Anforderung wichtig? Warum soll sie aufgenommen werden?*	
Ursprung:	*Woher stammt die Anforderung? Wer hat die Anforderung gestellt?*	
Überprüfung:	*Wie kann die Anforderung quantifiziert werden, um sie später zu prüfen?*	
Kundenzufriedenheit: *Wie stark ist der Kunde an dieser Anforderung interessiert?*	**Folgen von Unzufriedenheit:** *Was wäre, wenn die Anforderung nicht realisiert wird?*	
Abhängigkeiten: *Andere beeinflusste Anforderungen.*	**Konflikte:** *Widersprüchliche Anforderungen.*	
Weitere Materialien:	*Hinweise auf weitere Informationen zu dieser Anforderung.*	
Historie:	*Jeder, der etwas an der Anforderung verändert, muss sich hier eintragen.*	

Anf. Nr.:	**Dokumentzuordnung:**	**Ereignis/Use Case Nr.**:
Komponente:	**Variante:**	
Beschreibung:		
Bedeutung:		
Ursprung:		
Überprüfung:		
Kundenzufriedenheit:	**Folgen von Unzufriedenheit:**	
Abhängigkeiten:	**Konflikte:**	
Weitere Materialien:		
Historie:		

FORE-Merkmalkarte zur Erhebung von Merkmalen

Merkmal Nr.: *Eindeutige Nummer*	**Dokumentzuordnung:** *Einordnung in die Struktur des FORE-Spezifikationsdokumentes*	**Merkmaltyp**: *Funktion, Schnittstelle oder Parameter*
		Parametertyp ODER **Schnittstellenbeschreibung**
Name:	*Ein einzelnes Wort, welches das Merkmal beschreibt.*	
Entscheidung:	*Ein Verweis in das Decision Model.*	
Beschreibung:	*Kurze Beschreibung des Merkmals.*	
Anforderungen:	*Welchen Anforderungen ist dieses Merkmal zugeordnet?*	
Motivation:	*Warum sollte ein Kunde dieses Merkmal wählen?*	
Varianten:	*Gibt es Varianten, denen dieses Merkmal entstammt oder denen es zugeordnet werden kann?*	
Quelle:	*Woher stammt das Merkmal? Wer hat das Merkmal erstellt?*	
Prüfung:	*Wie kann das Merkmal später getestet werden? (→ Integration der Anforderungstests)*	
Abhängigkeiten:	*Andere beeinflusste Merkmale.*	**Konflikte:** *Widersprüchliche Merkmale.*
Weitere Materialien:	*Hinweise auf weitere Informationen zu diesem Merkmal.*	
Historie:	*Jeder, der etwas an dem Merkmal verändert, wird in die Historie eingetragen.*	

Merkmal Nr.:	**Dokumentzuordnung:**	**Merkmaltyp**:
		Parameter / **Schnittstelle**
Name:		
Entscheidung:		
Beschreibung:		
Anforderungen:		
Motivation:		
Varianten:		
Quelle:		
Prüfung:		
Abhängigkeiten:		**Konflikte:**
Weitere Materialien:		
Historie:		

Glossar

Abstrakter Datentyp	Ein abstrakter Datentyp ist ein Datentyp, der nicht als elementarer Datentyp (wie z. B. Integer-Variable) auf einem Rechner verfügbar, sondern aus elementaren Datentypen zusammengesetzt ist. Er wird mit den zugehörigen Operationen bereitgestellt und kann vom Benutzer wie ein elementarer Datentyp behandelt werden. Viele Programmiersprachen bieten die Konstruktion von abstrakten Datentypen an, z. B. die Sprache C durch den Konstruktionsoperator »struct«. [Broc 2003]
Akteur	Akteure sind die an einem → *Use-case* beteiligten Personen oder Systeme.
Anforderung	Eine Anforderung ist eine von einem System geforderte Eigenschaft oder Fähigkeit. Siehe auch Definition Kapitel 3.1.
Anforderungsanalyse	Die Anforderungsanalyse wird häufig als Übersetzung für → *Requirements Engineering* verwendet. In dieser Arbeit wurde der englische Begriff genutzt, da er auch die Prüfung und Verwaltung der Anforderungen enthält und damit umfassender ist.
Anforderungsspezifikation	Die Anforderungsspezifikation enthält alle Anforderungen in einer aufbereiteten Form. Siehe auch Definition Kapitel 3.1.5.
Applikation	In dieser Arbeit ist eine Applikation ein Softwaresystem, welches basierend auf einer Systemfamilie abgeleitet wurde. Synonym verwendet zu → Familienmitglied
Baselining	Zu einem bestimmten Zeitpunkt wird der Stand aller Entwicklungsdaten durch eine Datensicherung „eingefroren“, was als *Baselining* bezeichnet wird.
Data Dictionary	Das *Data Dictionary* ist ein Verzeichnis, das Informationen über die Struktur von Daten, ihre Eigenschaften, sowie Verwendung enthält, was meistens in einer modifizierten → BNF beschrieben wird.
Datenkapsel	Klassen aus der Objektorientierung werden auch als Datenkapsel bezeichnet, die Attribute und Methoden enthalten.

Design by Contract	Der Begriff *Design by Contract* steht für die Entwicklung eines Systems durch Angabe von Korrektheitsbedingungen für alle auszuführenden Systemteile. Eingabe von Vor-, Nachbedingungen und Invarianten.
Digitales Videoprojekt	Das digitale Videoprojekt ist ein universitäres Projekt, um die Realisierbarkeit von FORE zu zeigen. Informationen zu dem Projekt sind in Kapitel 2 enthalten.
DocBook	*DocBook* ist eine DTD, die zur Beschreibung von Buch- und Artikelstrukturen genutzt wird.
Domain Analysis	Englisch für → Domänenanalyse.
Domain Engineering	*Domain Engineering* umschließt die Vorgänge der Eingrenzung einer Domäne (→ Scoping), der → Domänenanalyse, der Entwicklung einer → Referenzarchitektur und der Entwicklung von → Komponenten für eine → Systemfamilie.
Domäne	Eine Domäne ist ein bestimmtes und abgegrenztes Arbeits- oder Wissensgebiet. Innerhalb einer Domäne treten immer wieder ähnliche Fragestellungen und dazugehörige Lösungen auf.
Domänenanalyse	Die Domänenanalyse ist Teil des → *Domain Engineering*. Umschließt die Vorgänge zur Erhebung und Organisation von Informationen und → Anforderungen für eine → Systemfamilie, für die Gemeinsamkeiten und → Variabilitäten analysiert werden. Die Wiederverwendung aller Entwicklungsleistungen steht im Vordergrund.
Elicitation	Englisch (in dieser Arbeit) für, Erhebung von Anforderungen.
Entität	Eigenständige Einheit, die von anderen genau abgrenzbar ist.
Entity-Relationship-Modell	Ein Datenmodell, das die Beziehungen zwischen abstrakten Elementen (Entitäten, engl. *entity*) einer größeren Menge behandelt wird als *Entity-Relationship*-Modell bezeichnet. Die einzelnen Entitäten stehen untereinander in Beziehung (Relation).
Familienmitglied	→ Applikation

Feasibility	Englisch für Machbarkeit. Es soll die Machbarkeit eines Vorhabens festgestellt werden.
Feasibility Study	Durchführbarkeitsstudie, mit dem Ziel die Machbarkeit eines Projektes zu bewerten.
Feature	englisch für → Merkmal
Feature Model	englisch für → Merkmalmodell
Handheld	[dt. »in der Hand zu halten«] Sammelbezeichnung für besonders kleine Computer, die kompakt und leicht genug sind, um sie in einer Hand halten und so auch benutzen zu können. [Broc 2003]
Kardinalität	Angabe eines Wertebereichs, z.B. Auswahl an Weinen, von mindestens 2 verschiedenen Sorten bis zu 5 unterschiedlichen Weinsorten. Kardinalität = 2..5
Kernarchitektur	Der Anteil einer Systemfamilie, der in jedem Familienmitglied vorhanden ist.
Komponente	Eine Komponente ist eine über definierte Schnittstellen abgeschlossene Einheit der Soft- und Hardwareentwicklung. Siehe dazu die Definition in Kapitel 1.1.
Lastenheft	Wird zunächst von dem Auftraggeber erstellt, um dem Auftragnehmer die erhobenen Anforderungen aus Sicht des Auftraggebers zu präsentieren und über den Vortgang des Projektes zu entscheiden. Das Lastenheft ist Teil des → Pflichtenheftes.
Merkmal	Eine Systemfamilie wird über Merkmale beschrieben, die für den Kunden von Bedeutung sind und eine Differenzierung einzelner Familienmitglieder erlauben. Siehe auch Definition in Kapitel 4.2.
Merkmalmodell	In einem Merkmalmodell werden alle → Merkmale und die Beziehungen zwischen den Merkmalen hinterlegt. Siehe auch Definition in Kapitel 4.2.
Merkmalmodellierung	Der Vorgang der Abstraktion von → Anforderungen durch → Merkmale wird als Merkmalmodellierung bezeichnet.

Methode	Die abstrakte Beschreibung eines → Prozesses wird als Methode bezeichnet.
Parameter	Einstellbare Werte von → Komponenten und → Parametermerkmalen.
Parametermerkmal	Ein typisierter und einstellbarer Wert für ein Merkmal. Einem Merkmal können mehrere Parametermerkmale zugeordnet sein. Die einstellbaren Parameter korrespondieren mit den Einstellmöglichkeiten zugeordneter → Komponenten.
Parametrieren	Bei der Ableitung eines Familienmitgliedes müssen alle → Parametermerkmale mit konkreten Werten versehen werden. Dieser Vorgang wird in dieser Arbeit als Parametrierung bezeichnet.
Pflichtenheft	Das → Lastenheft wird vom Auftragnehmer zum Pflichtenheft weiterentwickelt und enthält nun auch technische Details zur Umsetzung der Anforderungen. Das Pflichtenheft bildet die Grundlage von vertraglichen Verabredungen (»Verpflichtungen«) zwischen Auftragnehmer und Auftraggeber.
Polymorphismus	[zu griech. polymorphos »vielgestaltig«], bei der objektorientierten Programmierung die »Vielgestaltigkeit« von Basismethoden und vererbten Methoden. [Broc 2003]
Product Line	Englisch für → Systemfamilie.
Produkt	Das fertiggestellte Softwaresystem mit Benutzerhandbuch und allen weiteren mitzuliefernden Komponenten wird als Produkt bezeichnet.
Produktlinie	Begriff für eine → Systemfamilie aus der → Domäne der Betriebswirtschaft. Wird synonym für Systemfamilie verwendet.
Prozess	Die Beschreibung der Abfolge mehrerer Vorgänge wird als Prozess bezeichnet. Dabei können Vorgänge auch konkrete Arbeitsanweisungen enthalten.
Referenzarchitektur	Die Architektur einer Systemfamilie wird als Referenzarchitektur bezeichnet und besteht aus der → Kernarchitektur und diversen optionalen → Komponenten.

Requirement	Englisch für → Anforderung.
Requirements Engineering	Umfassender Begriff für alle Vorgänge zur Erhebung, Modellierung, Prüfung und Verwaltung von → Anforderungen.
Requirements Specification	Englisch für → Anforderungsspezifikation.
Scoping	Sinnvolle Abgrenzung des Umfangs eines Projektes.
Six-Pack-Modell	Globales Modell der Systemfamilienentwicklung.
Spezifikation	Detaillierte Beschreibung eines bzw. mehrerer Sachverhalte. In dieser Arbeit auch synonym für → Anforderungsspezifikation verwendet.
Spezifikationsdokument	Synonym für → Anforderungsspezifikation verwendet.
Spezifikationssprachen	Formale Sprachen zur Spezifikation von Anforderungen.
Stakeholder	Sammelbegriff für beliebige an einem Projekt beteiligte Personen.
Systemfamilie	Bezeichnung eines Softwaresystems, welches durch seine → Referenzarchitektur die effiziente Ableitung von → Familienmitglieder ermöglicht.
Szenario	Beschreibt eine chronologische Abfolge von Handlungen, die einem → *Use-case* zugrunde liegen.
Tailoring	Die Anpassung einer Methode bzw. eines Prozesses wird als Tailoring (zuschneiden) bezeichnet.
Time-to-Market	Die Zeit, die benötigt wird, um ein Produkt von der Idee bis zur Marktreife zu bringen.
Traceability	Die Verfolgbarkeit von Elementen der Softwareentwicklung über die Entwicklungsphasen hinweg wird als *Traceability* bezeichnet.
Use-case	Englisch für Benutzungsfall. Aus Sicht des Benutzers eines Softwaresystems werden die möglichen Interaktionen mit dem System in Form von *Use-cases* beschrieben.

Validierung	Die Prüfung eines Modells auf die Übereinstimmung mit den Anforderungen wird in der Softwareentwicklung als Validierung bezeichnet.
Variabilität	Stellt einen optionalen Anteil einer Systemfamilie dar, der in einer abgeleiteten → Applikation vorhanden sein kann.
Verifikation	Die Prüfung einer Implementierung auf Übereinstimmung mit dem Modell wird in der Softwareentwicklung als Verifikation bezeichnet.
Video-On-Demand	Prinzip nach dem ein Benutzer von seinem *Client* aus Videoströme zu einer beliebigen Zeit von einem Server anfordern kann.
Vielfachheit	Synonym für → Kardinalität.

Abkürzungen

BNF	Backus-Naur-Form.
CASE	*Computer Aided Software Engineering.*
CMM	*Capability Maturity Model.*
COCOMO	*Constructive Cost Model*
COM	*Component Object Model.*
CORBA	*Common Object Request Broker Architecture.*
COTS	*Commercial Off The Shelf.*
DOM	*Document Object Model.*
DTD	*Document Type Definition*.
DVB	*Digital Video Broadcasting*.
DVP	Digitales Videoprojekt bzw. *Digital Video Project.* Siehe proinf.de/DVP.
EFM	*Extended Feature Model*.
ESAPS	*Engineering Software Architectures, Processes and Platforms for System-Families.*
FAST	*Family-Oriented Abstraction, Specification, and Translation.*
FeatuRSEB	*Featured* → RSEB.
FODA	*Feature-Oriented Domain Analysis.*
FOPLE	*Feature-Oriented Product Line Engineering.*
FORE	Familien Orientiertes Requirements Engineering bzw. *Family-Oriented Requirements Engineering.*
FORM	*Feature-Oriented Reuse Method.*
IrDA	*Infrared Data Association*.
JAD	*Joint Application Design.*
LEX	*Lexical Analyzer Generator.*
MPEG	*Motion Picture Encoding Group.*
OCL	*Object Constraint Language.*

PASTA	*Process and Artifact State Transition Abstraction.*
PDA	*Personal Digital Assistant*. Synonym zu → Handheld.
QFD	*Quality Function Deployment.*
RSEB	*Reuse-driven Software Engineering Business.*
TQM	*Total Quality Management.*
UML	*Unified Modeling Language*.
VDR	*Video Disk Recorder.* Siehe [Schm 2002].
XML	*Extensible Markup Language.*
YACC	*Yet Another Compiler Compiler*.

Literaturverzeichnis

[Alhi 1998] Sinan Si Alhir, *UML in a Nutshell*. O'Reilly, 1998.

[Anto 1996] Annie I. Anton, *Goal-Based Requirements Analysis*. In Proceedings of the International Conference on Requirements Engineering (ICRE), 1996.

[Atki 2002] Colin Atkinson, Joachim Bayer, Christian Bunse, Erik Kamsties, Oliver Laitenberger, Roland Laqua, Dirk Muthig, Barbara Paech, Jürgen Wüst, Jörg Zettel, *Component-Based Product Line Engineering with UML*. Addison-Wesley, 2002.

[Balz 1996] Helmut Balzert, *Lehrbuch der Software-Technik*. Spektrum Akademischer Verlag, 1996.

[Beck 2000] Kent Beck, *Extreme Programming Explained*. Addison-Wesley, 2000.

[Berg 2003] Katharina Berg, *Verarbeitung von Requirements Engineering XML-Daten mit XSLT*. Hauptseminar an der TU-Ilmenau, 2003.

[Blai 2001] Lynne Blair, Gordon Blair, Jianxiong Pang, Christos Efstratiou, '*Feature*' *Interactions Outside a Telecom Domain*. In Proceedings of Workshop on Feature Interactions in Composed Systems, European Conference on Object-Oriented Programming (ECOOP), 2001.

[Broc 2003] Joachim Weiß, *Der Brockhaus - Computer und Informationstechnologie*. Brockhaus, 2003.

[Broo 1995] Frederick P. Brooks, Jr., *The Mythical Man Month*. Addison-Wesley, 1995.

[Brue 2000] Bernd Bruegge, Allen H. Dutoit, *Object-Oriented Software Engineering*. Prentice Hall, 2000.

[BucA 1999] Karsten Buch, *Auswirkung von Anforderungsänderungen auf den Systementwurf*. Diplomarbeit an der Universität Kaiserslautern, 1999.

[Bumb 2003] Bumble-Bee Software , *Windows LEX & YACC*. www.bumblebeesoftware.com, 2003.

[CHAO 1995] The Standish Group, *CHAOS report*. standishgroup.com, 1995.

[ChGa 1994] Marsha Chechik, John Gannon, *Automatic Verification of Requirements Implementation*. In Proceedings of the 1994 Internation Symposium on Software Testing and Analysis (ISSTA), 1994.

[Clar1989] Charles Clark, *Brainstorming: How to Create Successful Ideas*. Wilshire Book Company, 1989.

[Clau 2001] Matthias Clauß, *A proposal for uniform abstract modeling of feature interactions in UML*. In Proceedings of the European Conference for Object-Oriented Programming (ECOOP), 2001.

[Copl 1999] James O. Coplien, *Multi-Paradigm Design for C++*. Addison-Wesley, 1999.

[CREW 1999] CREWS Projekt RWTH Aachen, *CREWS Method Base*. panoramix.univ-paris1.fr/CRINFO/CMB/, 1999.

[Czar 2000] Krzysztof Czarnecki, Ulrich Eisenecker, *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.

[Diet 2003] Ralph Dietzel, *Konzeption und Entwicklung einer Systemfamilie für eine Universal-Fernbedienung auf Basis eines Palm-Handhelds*. Diplomarbeit an der TU-Ilmenau, 2003.

[DOOR 2003] Telelogic Deutschland GmbH, *DOORS*. www.telelogic.com, 2003.

[Duen 2001] Editors: Juan C. Dueñas, William El Kaim, Cristina Gacek, *Style, structures and views for*

handling commonalities and variabilities. ESAPS Deliverable (WG 2.2.3), Eureka 2023 Programme, ITEA project 99005, www.esi.es, 2001.

[Eric 1984] Ericsson K. A., Simon H.A., *Protocol Analysis: Verbal Reports as Data*. MIT Press, 1984.

[ESPI 1996] Milagros Ibanez, Helmut Rempp, *European User Survey Analysis*. European Software Institute, www.esi.es, 1996.

[Fowl 1999] Martin Fowler, *Refactoring*. Addison-Wesley, 1999.

[Funk 1999] Thomas Funke, Reinhard Noll, Stefan Niessen, Bruno Weikl, *Softwareentwicklung in mittelständischen Unternehmen mit ISO 9000*. Springer Verlag, 1999.

[Gamm 1995] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[Geie 2000] Jim Geier, *Improving Requirements with Joint Application Design (JAD)*. www.wireless-nets.com/whitepaper_jad.htm, 2000.

[Goma 2000] Hassan Gomaa, *Object Oriented Analysis and Modeling for Families of Systems with UML*. W. B. Frakes (Editor), Proceedings of the Sixth International Conference on Software Reuse, 2000.

[Hare 1987] David Harel, *Statecharts: A Visual Formalism for Complex System*. In Science of Computer Programming, Elsevier Science Publishers, 1987.

[Hein 2001] Andreas Hein, John MacGregor, Steffen Thiel, *Configuring Software Product Line Features*. Springer Verlag, 2001.

[Herb 2003] Jens Herbig, *Reverse Engineering von Entwurfsmustern mit Together*. Studienarbeit an der TU-Ilmenau, 2003.

[Hoff 1999] Jeffrey A. Hoffer, Joey F. George, Joseph S. Valacich, *Modern Systems Analysis & Design*. Addison-Wesley, 1999.

[Hofm 2000] Hofmann Hubert F., *Requirements Engineering*. Deutscher Universitäts-Verlag, 2000.

[Humm 2002] Thomas Hummel, Christian Malorny, *Total Quality Management. Tipps für die Einführung*. Hanser Fachbuch, 2002.

[IBMa 2003] IBM alphaWorks, *IBM XML-Parser*. www.alphaworks.ibm.com, 2003.

[Ilog 2003] I-logix, *I-logix - Statemate*. www.ilogix.com, 2003.

[INCO 2002] International Council On Systems Engineering, *Vergleich von Requirements Engineering Werkzeugen*. www.incose.org/tools/tooltax.html, 2002.

[Jaco 1997] Ivar Jacobson, Martin Griss, Patrik Jonsson, *Software Reuse: Architecture, Process and Organization for Business Success*. Addison-Wesley, 1997.

[Jark 1999] Matthias Jarke, *CREWS : Towards Systematic Usage of Scenarios, Use Cases and Scenes*. Springer Verlag, 1999.

[JaSu 2000] James Robertson, Suzanne Robertson, *Volere Requirements Specification Template*. . Atlantic Systems Guild, 2000.

[Jenn 2002] Bill Jennerich, *Joint Application Design - Business Requirements Analysis for Successful Re-engineering*. www.bee.net/bluebird/jaddoc.htm, 2002.

[Jia 2001] Yu Jia, Yuquing Gu, *Representing and Reasonig on Feature Architecture: A Description Logic Approach*. Springer Verlag, 2001.

[John 2002] Isabel John, Dirk Muthig, *Product Line Modeling with Generic Use Cases*. In Lecture Notes of Computer Science Vol. 2379 of The Second Software Product Line Conference

(SPLC2), 2002.

[Kami 2002] Kamiske Gerd, *ABC des Qualitätsmanagements*. Carl Hanser Verlag, 2002.

[Kang 1990] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, A. Spencer Peterson, *Feature-Oriented Design Analysis (FODA) Feasibility Study*. Report: CMU/SEI-90-TR-21 ESD-90-TR-222, www.sei.cmu.edu, 1990.

[Kang 1998] Kyo C. Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim,Euiseob Shin, Moonhang Huh, *FORM: A Feature-Oriented Reuse Method*. Pohang University of Science and Technology, www.postech.ac.kr/e/, 1998.

[Kang 2002] Kyo C. Kang, Kwanwoo Lee, Jaejoon Lee, *Feature-Oriented Product Line Software Engineering: Principles and Guidelines*. Pohang University of Science and Technology, www.postech.ac.kr/e/, 2002.

[Kano 1993] David Walden (Editor), *A special issue on Kano's Methods for Understanding Customer-defined Quality*. Center for Quality of Management Journal, Volume 2, Number 4, 1993.

[KeWi 1997] Allen Kent, James G. Williams, *Requirements Engineering*. Encyclopedia of Computer Science, Volume 36, 1997.

[Klug 2002] Rene Kluge, Automatisierung der Dokumentation im Bereich der Softwareentwicklung. Studienarbeit an der TU-Ilmenau, 2002.

[Klug 2003] Rene Kluge, *Integration der Aufwandsschätzung in die Requirements Engineering Phase der Systemfamilienentwicklung*. Diplomarbeit an der TU-Ilmenau, 2003

[KoSo 1998] Gerald Kotonya, Ian Sommerville, *Requirements Engineering - Processes and Techniques*. Wiley, 1998.

[Krzy 1998] Krzysztof Czarnecki, *Generative Programming*. Dissertation an der TU- Ilmenau, 1998.

[Kula 2000] Daryl Kulak, Eamonn Guiney, *Use Cases - Requirements in Context*. Addison-Wesley, 2000.

[Lams 2000] Axel van Lamsweerde, *Requirements Engineering in the Year 00: A Research Perspective*. In Proceedings of the 22nd International Conference on Software Engineering (ICSE), 2000.

[Lams 2001] Axel van Lamsweerde, *Goal-Oriented Requirements Engineering: A Guided Tour*. www.info.ucl.ac.be/research/projects/AVL/Re, 2001.

[Lang 2003] Hartmut Lang, Michael Schnegg, *Methoden der Ethnographie*. www.methoden-der-ethnographie.de, 2003.

[Lars 1996] Peter Gorm Larsen, John Fitzgerald, Tom Brookes, *Lessons Learned from Applying Formal Specification in Industry*. IEEE Software, 1996.

[Lata 2002] Simon Latapie et al., *Video Streaming Project*. www.videolan.org/, 2002.

[Lirc 2002] Christoph Bartelmus, *Linux Infrared Project*. www.lirc.org/, 2002.

[Lore 2001] Louise Lorentsen, Antti-Pekka Tuovinen, Jianli Xu, *Modelling Feature Interactions in Mobile Phones*. Proceedings of the European Conference for Object-Oriented Programming (ECOOP), 2001.

[Louc 1995] Loucopoulos Pericles, *System Requirements Engineering*. McGraw Hill, 1995.

[Maca 1996] L.. A. Macaulay, *Requirements Engineering*. Springer Verlag, 1996.

[Math 2003] Wolfram Research, *Mathematica*. www.wolfram.com, 2003.

[Meff 2003] Florian Meffert, *Konzeption einer Videodatenverteilung im Rahmen des Digitalen Video*

Projektes (DVP). Diplomarbeit an der TU-Ilmenau, 2003.

[Metz 2002] Ralph Metzler, Marcus Metzler, *linuxTV.org-Projekt*. linuxtv.org/, 2002.

[Meye 1997] Bertrand Meyer, *Object-Oriented Software Construction 2nd Edition*. Prentice Hall, 1997.

[Milt 2002] Nick Milton, *Laddering Techniques*. www.epistemics.co.uk/Notes/178-0-0.htm, 2002.

[Nada 1998] Nader Nada, David C. Rine, *A Validated Software Reuse Reference Model Supporting Component-Based Management*. International Workshop on Component-Based Software Engineering, 1998.

[Naum 2001] Sebastian Naumann, *Reverse Engineering von Entwurfsmustern*. Diplomarbeit an der TU-Ilmenau, 2001.

[Parn 1976] David Lorge Parnas, *On the Design and Development of Product Families*. IEEE Transactions on Software Engineering, SE-2(1):1--9, 1976.

[Prei 2003] Anne Preiß, Systemfamilienbasierte Dokumentation des Digital Video Project. Diplomarbeit an der TU-Ilmenau, 2003.

[QFD 2003] Georg Herzwurm, *QFD Institut Deutschland e.V.*. www.qfd-id.de, 2003.

[Raly 1999] Jolita Ralyté, Colette Rolland, Véronique Plihon, *Method Enhancement with Scenario Based Techniques*. In Proceedings of CAISE 99, 11th Conference on Advanced Information Systems Engineering Heidelberg, Germany, June 14-18, 1999.

[Rege 2003] Andreas Regel, Entwicklung und Implementierung von Methoden zur Erkennung von Werbeblöcken im digitalen Fernsehen. Diplomarbeit an der TU-Ilmenau, 2003.

[Reis 1982] W. Reisig, *Petri-Netze - Eine Einführung*. Springer Verlag, 1982.

[Rieb 2002] Matthias Riebisch, Kai Böllert, Detlef Streitferdt, Ilka Philippow, *Extending Feature Diagrams with UML Multiplicities*. In Proceedings of the 6th Conference on Integrated Design & Process Technology, Pasadena California, USA, 2002.

[Robe 1999] Suzanne Robertson, James Robertson, *Mastering the Requirements Process*. Addison-Wesley, 1999.

[Rupp 2002] Chris Rupp, *Requirements-Engineering und -Management*. Hanser Verlag, 2002.

[Savo 2001] Juha Savolainen, Klaus Schmid, Siegfried Trosch, A.-P. Tuovinen (editor), Tuomo Vehkomäki, *Feature Analysis*. Workproduct Eureka 2023 Programme, ITEA project 99005, www.esi.es, 2001.

[Scha 2003] Sven Schaepe, Be- und Verarbeitung von Service-Informationen im DVB-Stream. Studienarbeit an der TU-Ilmenau, 2003.

[Schi 2002] Bruno Schienmann, *Kontinuierliches Anforderungsmanagement*. Addison-Wesley, 2002.

[Schm 2002] Klaus Schmidinger, *vdr Projekt Homepage*. www.cadsoft.de/people/kls/vdr, 2002.

[Schr 2003] Andreas Schrankel, *Umsetzung einer bestehenden DVB Software Architektur in eine Systemfamilie*. Studienarbeit an der TU-Ilmenau, 2003.

[Schu 2000] G. Gordon Schulmeyer, Garth R. Mackenzie, *Verification & Validation of Modern Software-Intensive Systems*. Prentice Hall, 2000.

[SEI 2003] Software Engineering Institute, *Capability Maturity Model for Software (SW-CMM)*. www.sei.cmu.edu/cmm/, 2003.

[SEIB 2000] SEI, *Domain Engineering and Domain Analysis*. www.sei.cmu.edu/domain-engineering/domain_engineering.html, 2000.

[Shar 1998] Shari Lawrence Pfleger, *Software Engineering - Theory And Practice*. Prentice Hall, 1998.

[Soph 2003] Sophist GmbH, *C.A.R.E.*. www.sophist.de, 2003.

[Stew 1997] Valerie Stewart, *Business Applications of Repertory Grid*. www.EnquireWithin.co.nz, 1997.

[Tarr 1999] Peri Tarr, Harold Ossher, William Harrison, Stanley M. Sutton Jr., *N Degrees of Separation: Multi-Dimensional Separation of Concerns*. In Proceedings of the 21st Interantional Conference on Software Engineering (ICSE), IEEE Press, 1999.

[Tawb 1999] Mustapha Tawbi, Camille Ben Achour, Fernando Vélez, *Guiding the Process of Requirements Elicitationthrough Scenario Analysis : Results of an Empirical Study*. panoramix.univ-paris1.fr/CRINFO/CMB/, 1999.

[Ulri 2002] Dirk Ulrich, *Automatische Generierung geographischer Informationssysteme.* Diplomarbeit an der TU-Ilmenau, 2002

[UML 1999] Object Management Group, *Unified Modeling Language Specification*. OMG, www.omg.org, 1999.

[Uter 2003] Stefan Utermark, *Automatische Generierung von Anwendungsoberflächen in der Rechnungsverarbeitung unter Nutzung des Systemfamilienkonzeptes*. Diplomarbeit an der TU-Ilmenau, 2003.

[W3C 2003] World Wide Web Consortium, *W3C*. www.w3c.org, 2003.

[Wals 2003] Norman Walsh, Leonard Muellner, *DocBook: The Definitiv Guide*. www.docbook.org/, 2003.

[Weis 1999] David M. Weiss, Chi Tau Robert Lai, *Software Product-Line Engineering: A Family-Based Software Development Process*. Addison-Wesley, 1999.

[Wieg 1999] Karl E. Wiegers, *Software Requirements*. Microsoft Press, 1999.

[Wood 1995] Wood, J. and D. Silver, *Joint Application Development, 2nd ed.*. Wiley, 1995.

[XMIT 2003] IBM alphaWorks, *IBM XMI-Toolkit*. www.alphaworks.ibm.com/tech/xmitoolkit, 2003.

[Youn 2001] Ralph R. Young, *Effective Requirements Practices*. Addison-Wesley, 2001.

Thesen

1. Die anfängliche Investition in die Systemfamilienentwicklung amortisiert sich erst nach mindestens vier verkauften Familienmitgliedern. Dieser Wert ergibt sich aus Berechnungen mit dem *Constructive Cost Model II* .
2. Die Kosten der Entwicklung von Applikationen, basierend auf Systemfamilien, sind geringer als die Kosten der Entwicklung von Individualsystemen. Steht eine Systemfamilie zur Verfügung, können Applikationen die Komponenten der Familie wiederverwenden, was Entwicklungskosten einspart.
3. Die *Requirements-Engineering*-Phase der Systemfamilienentwicklung teilt sich in die Domänenentwicklung, Applikationsentwicklung und die Verwaltung der Anforderungen auf. Während der Domänenentwicklung werden die Gemeinsamkeiten und Unterschiede der Systemfamilie analysiert und in einem Anforderungsmodell hinterlegt. Während der Applikationsentwicklung werden die Anforderungen des Kunden mit dem Anforderungsmodell der Systemfamilie abgeglichen.
4. Mit der in dieser Arbeit dargestellten Methode *Family-Oriented Requirements Engineering* (FORE) können Anforderungen und Merkmale einer Systemfamilie in der *Requirements-Engineering*-Phase modelliert werden. Mithilfe von Merkmalen werden die Unterschiede der Familienmitglieder modelliert. Für ein Familienmitglied muss ein Kunde die gewünschten Merkmale auswählen.
5. FORE erlaubt die Überprüfung der Konsistenz eines Merkmalmodells in der Domänenentwicklung und die automatisierte Überprüfung einer Auswahl von Merkmalen in der Applikationsentwicklung. Die kundenspezifische Merkmalauswahl führt damit immer zu einer aus der Systemfamilie ableitbaren Applikation.
6. FORE ist durch die *Feature Constraint Language* an zukünftige Anforderungen anpassbar. Beliebige Beziehungen zwischen Merkmalen und weiteren Modellelementen sind modellierbar.
7. FORE stellt einen Prozess für die *Requirements-Engineering*-Phase der Systemfamilienentwicklung bereit, der die objektorientierte Entwicklung unterstützt.

Ilmenau, den 25. August 2003

Erklärung

Ich versichere, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Weitere Personen waren an der inhaltlich materiellen Erstellung der vorliegenden Arbeit nicht beteiligt. Insbesondere habe ich hierfür nicht die entgeltliche Hilfe von Vermittlungs- bzw. Beratungsdiensten (Promotionsberater oder anderer Personen) in Anspruch genommen. Niemand hat von mir unmittelbar oder mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen.

Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form einer Prüfungsbehörde vorgelegt.

Ich bin darauf hingewiesen worden, dass die Unrichtigkeit der vorstehenden Erklärung als Täuschungsversuch angesehen wird und den erfolglosen Abbruch des Promotionsverfahrens zur Folge hat.

Ilmenau, den 25.August 2003 Unterschrift