Requirements Engineering komplexer Standardsoftware

Bernhard Deifel

Institut für Informatik
der Technischen Universität München

Requirements Engineering komplexer Standardsoftware

Bernhard Deifel

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr. Dr. h.c. J. Eickel

Prüfer der Dissertation:

1. Univ.-Prof. Dr. M. Broy
2. Univ.-Prof. Dr. J. Schlichter

Die Dissertation wurde am 25.04.2001 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 13.07.2001 angenommen.

Kurzfassung

Das Requirements Engineering befasst sich mit der systematischen Erfassung von Anforderungen an Software und deren Überführung in eine Spezifikation. Die vorliegende Arbeit stellt eine Methodik für das Requirements Engineering komplexer Standardsoftware vor. Diese Software zeichnet sich durch eine Vielzahl von Funktionalitäten und Variationen aus, wodurch sie im Allgemeinen als ein aus mehreren Softwareprodukten bestehendes System zum Verkauf auf einem globalen Markt angeboten wird. Wesentliche Problemstellungen in diesem Umfeld ergeben sich aus der Wechselwirkung firmeneigener Ziele mit dem Markteinfluss. Firmenziele, wie Kundenorientierung, Zeit- und Kostenoptimierung, müssen vor dem Hintergrund erheblicher Unsicherheiten über Absatzmärkte, ständigen Veränderungen und einer großen Vielfalt von Anforderungen bewältigt werden. Hauptmechanismen zur Problemlösung sind die Entwicklung der Standardsoftware in Variationen und Versionen.

Kern der Arbeit sind ein Modell von Entwicklungsprodukten und ein darauf abgestimmtes Prozessmodell zur systematischen Unterstützung des Requirements Engineerings komplexer Standardsoftware. Das Prozessmodell beschreibt ein Vorgehen für das Herausarbeiten, die Verhandlung und die Spezifikation von Anforderungen, während das Modell Entwicklungsprodukte zu erarbeitende Ergebnisse sowie methodische Vorlagen und Anleitungen dokumentiert.

Durch die Anwendung eines speziellen Modellierungskonzeptes werden insbesondere die Verfolgbarkeit von Anforderungen und von Entscheidungen einzelner Entwicklungsaktivitäten sowie die schnelle Reaktionsfähigkeit auf sich ändernde Anforderungen unterstützt. Entwicklungsschritte werden als Vorarbeiten oder Entscheidungen einzelner Entscheindungssituationen modelliert, während Entwicklungsprodukte genau festgelegte Aufgaben übernehmen. Dadurch wird die Erarbeitung und die Verwendung aller Entwicklungsergebnisse und Entscheidungen nachvollziehbar. Zusätzlich ermöglicht die Modellierung der Inhalte und der Zusammenhänge zwischen Entwicklungsprodukten eine durchgängige Verfolgbarkeit von Anforderungen. Darüber hinaus ist die Formulierung von Konsistenzbedingungen für das Requirements Engineering möglich, womit Aufgaben des Requirements Engineerings, des Requirements Managements und des Konfigurations-Managements ineinander integriert werden.

Eine spezielle Strukturierung des Prozessmodells und des Modells der Entwicklungsprodukte erlaubt einen stufenweisen Übergang von roh gesammelten Anforderungen hin zu eine Spezifikation von zur Entwicklung geplanten Anforderungen. Stufen des Übergangs sind Marktsicht, Systemsicht und Entwicklungssicht. Erstere ist kundenorientiert und setzt ihren Schwerpunkt auf die Identifikation von Variationen. Die zweite Sicht zielt auf eine Kompromissbildung von Marketing- und Entwicklungszielen ab und fokussiert auf die Anforderungsverteilung auf Softwareprodukte. Schließlich konzentriert sich letztere auf die kosten- und zeitoptimale Versionsplanung einer komplexen Standardsoftware. Die Strukturierung ermöglicht ferner kurzfristige Reaktionszeiten durch eine flexible Gestaltung der Entwicklung und die Berücksichtigung langfristiger Entwicklungsziele.

Zur methodischen Unterstützung des Prozessmodells werden darüber hinaus eine Bewertungsmethode für Entscheidungssituationen sowie ein Konzept zur Spezifikation von Entwicklungsprodukten vorgeschlagen.

Dankesworte

Zuerst danke ich Herrn Prof. Dr. M. Broy, der mich inspirierte und mir überhaupt erst ermöglichte, an einem so praxisorientierten Thema zu arbeiten. Ich danke ihm auch für die nützlichen Ratschläge und die Betreuung der Arbeit insgesamt. Herrn Prof. Dr. J. Schlichter gilt der Dank für die Übernahme des Zweitgutachtens und das von ihm aufgebrachte Interesse an meiner Arbeit.

Dann danke ich vor allem meinem Zimmerkollegen Herrn Wolfgang Schwerin. Er war immer der erste Diskussionspartner für meine Ideen und offenen Fragen im Rahmen dieser Arbeit. Er war auch derjenige, dem ich am häufigsten Vorversionen meiner Arbeit aufs Auge drückte, für dessen Durchsicht ich nochmals besonders danken muss.

Ich danke auch den weiteren Reviewern in der Schlussphase meiner Arbeit, Frau Eva Geisberger, Herrn Alexander Pretschner und Herrn Christian Salzmann. Alle gaben mir wertvolle Hinweise für die letztendliche Fassung meiner Dissertationsschrift.

Des weiteren danke ich allen Mitarbeiterinnen und Mitarbeitern des Forschungsverbundes Software Engineering. Insbesondere danke ich meinen Kollegen Herrn Robert Sandner und Herrn Gunnar Billing, die mit mir gemeinsam das Teilprojekts A4 bestritten. Vor allem danke ich auch den Mitarbeitern des Industriepartners Siemens A&D, insbesondere Herrn Clemens Dinges, Herrn Dr. Joachim Holz und Herrn Harald Hammon. Sie brachten mir wertvolle Einblicke in die Problematik des Themas aus Sicht der Praxis. Ohne daraus gewonnene Erkenntnisse und zeitweise auch durchaus kontroverse Diskussionen wäre eine eingehende Behandlung dieses Themas nicht möglich gewesen.

Dank gebührt auch allen Kolleginnen und Kollegen unseres Lehrstuhls für die sehr gute Arbeitsatmosphäre. Es bestand immer eine hohe Bereitschaft, sich bei allen auftretenden Problemen und Problemchen gegenseitig zu unterstützen.

Nicht zuletzt danke ich meiner Freundin Trixi, die mir in der Schlussphase meiner Arbeit seelischen und moralischen Beistand gab. Außerdem danke ich meiner ganzen Familie, die gerade in letzter Zeit viel Verständnis aufbringen musste, dass ich zum Schluss den größten Teil der mir zur Verfügung stehenden Zeit meiner Arbeit widmen musste.

Kapitel 1

Einleitung und Motivation

Die Informationstechnologie ist seit mehreren Jahren einer der stärksten Wachstumsmotoren in Industrieländern. Mit zunehmender Vernetzung wie mobiler Kommunikation und dem weltweiten Internet durchdringt sie immer mehr Bereiche in Wirtschaft und privaten Haushalten. Durch verbesserte Hardwaretechnologien kann zunehmend komplexere Software entwickelt werden, die immer mehr Bedürfnissen ihrer Anwender gerecht wird. Das Wachstumspotential und daraus resultierende Gewinnchancen ziehen zunehmend Unternehmen an, gerade in den vielversprechendsten Anwendungsbereichen eigene Lösungen anzubieten. Daraus resultiert gleichzeitig eine wachsende Konkurrenz. Für viele Unternehmen können daher und wegen eines häufig unprofessionellen Umgangs mit dieser Situation zu hoch gesteckte Gewinnerwartungen nicht erfüllt werden. Dies führt im Extremfall zu Existenzproblemen dieser Unternehmen. Sehr anschaulich äußert sich dies gerade in letzter Zeit an den heftigen Turbulenzen an Technologiebörsen, wie beispielsweise dem deutschen Neuen Markt.

Weiterhin ist mit dem Wachstum ein ständiger Wandel der Softwarelandschaft verbunden, auf den gerade auch Hersteller von Standardsoftware ständig reagieren müssen. Um konkurrenzfähig zu sein, müssen fortlaufend neue Markterfordernisse erkannt werden und daraus Anforderungen an die Standardsoftware abgeleitet werden. Reaktionszeiten sind aufgrund der immateriellen Eigenschaft von Software wesentlich kürzer als bei materiellen Produkten. Gleichzeitig nimmt wegen der zunehmenden Bedürfniserfüllung die Komplexität von Software ständig zu. Um diesen wachsenden Anforderungen an Standardsoftware herstellende Unternehmen gerecht werden zu können, bedarf es einer systematischen Methodik für das Requirements Engineering komplexer Standardsoftware. Die Vorstellung einer derartigen Methodik ist Inhalt der vorliegenden Arbeit.

Bevor die Ausgangssituation, resultierende Problemstellungen und grundlegende Lösungskonzepte vorgestellt werden, erfolgt zunächst die Einführung zentraler Begriffe:

Definition: Komplexe Standardsoftware

> *Unter* Standardsoftware *(engl. commercial off the shelf software, kurz COTS) wird Software verstanden, die nicht im Auftrag einzelner Kunden, sondern für einen gesamten globalen Markt entwickelt wird.* Komplexe Standardsoftware *(engl. complex COTS, kurz CCOTS) ist insbesondere dadurch gekennzeichnet, dass sie aufgrund der Vielzahl von Funktionalitäten und Variationen nicht mehr als ein einzelnes Softwareprodukt verkauft wird. Stattdessen besteht sie aus einer Reihe verschiedener Softwareprodukte, die auf unterschiedlicher Weise eng miteinander zusammenarbeiten und damit ein sogenanntes* Produktsystem *bilden.*

Beispiel:

Beispiele komplexer Standardsoftware sind Microsoft Office, betriebliche Softwarepakete wie SAP R/3 oder BaanERP oder auch das Softwaresystem SIMATIC von SIEMENS.

In der Literatur existieren viele Definitionen des Begriffs *Requirements Engineering*. Eine Auswahl hiervon kann zum Beispiel in [Poh96] nachgelesen werden. In der Informatik hat sich der englische Begriff auch im deutschen Sprachgebrauch etabliert, so dass hierfür im Rahmen dieser Arbeit keine Übersetzung ins Deutsche vorgenommen wird.

Definition: Requirements Engineering

Unter Requirements Engineering *wird ein Teil des Software Engineerings verstanden, der folgende Ziele verfolgt:*

- *Ein systematisches Erarbeiten von Anforderungen an eine Software,*
- *eine Festlegung, welche Anforderungen wo und wann realisiert werden sollen und*
- *eine Dokumentation zu realisierender Anforderungen in einer in sich konsistenten, konfliktfreien und validierten Spezifikation.*

Das Requirements Engineering komplexer Standardsoftware *bezieht sich auf die Entwicklung komplexer Standardsoftware. Hier müssen bei der Erarbeitung und Realisierungsplanung von Anforderungen vor allem Bedürfnisse der Variations-, Produktsystem- und Versionsentwicklung dieser Software berücksichtigt werden.*

Eine Spezifikation repräsentiert im Sinne des Requirements Engineerings Anforderungen, die in einer Software realisiert werden. Wesentlich ist hierfür eine adäquate und verständliche Darstellung für verschiedene an der Entwicklung beteiligte Personen, wie zum Beispiel Softwareentwickler, Manager, Benutzer und Kunden.

Auch über den Begriff Anforderung herrscht in der Literatur wenig Konsens. In dieser Arbeit wird unter Anforderungen folgendes verstanden:

Definition: Anforderung

Eine Anforderung ist eine Forderung bestimmter Eigenschaften oder Funktionen, die eine Software erfüllen soll. Die Anforderung kann von einer beliebigen Person bewusst oder unbewusst gestellt sein, die in irgendeiner Weise Interesse an der Erfüllung der Anforderung in der Software hat.

Das reine Vorhandensein einer Anforderung an eine Software sagt nichts darüber aus, ob sie je realisiert wird. Absolute Vorraussetzung hierfür ist eine explizite Entscheidung im Laufe des Requirements Engineerings.

Die wissenschaftliche Beschäftigung des Requirements Engineerings war bisher ausgerichtet auf die Betrachtung von Auftragsprojekten für Individualsoftware. Ausgangspunkt der vorliegenden Arbeit war eine eingehende Problemanalyse für das Requirements Engineering komplexer Standardsoftware beim Marktführer für sogenannte Engineering-Software in der Automatisierungstechnik. Die hier identifizierten Problemstellungen konnten auch bei führenden Herstellern von ERP-Software (Enterprise Resource Plan-

ning) und bei anderer komplexer Standardsoftware wie Betriebssystemen, Office-Pakten unter anderen festgestellt werden. Der Rest dieses einführenden Kapitels befasst sich ausführlich mit der Darstellung der im Requirements Engineering komplexer Standardsoftware zu behandelnden Problemstellungen und leitet daraus ein Gesamtkonzept der Arbeit ab. In Abschnitt 1.1 wird die Ausgangsituation der Entwicklung komplexer Standardsoftware erläutert. Anschließend werden in Abschnitt 1.2 Auswirkungen auf Aktivitäten des Requirements Engineerings betrachtet und besondere Problemstellungen identifiziert. Danach werden in Abschnitt 1.3 die Ziele der vorliegenden Arbeit vorgestellt, diese wissenschaftlich eingeordnet und letztendlich ein Gesamtkonzept vorgestellt. In Abschnitt 1.4 wird schließlich ein Überblick über den Aufbau der Arbeit gegeben.

1.1 Ausgangssituation

Dieser Abschnitt widmet sich der Erörterung der Grundproblematik der Entwicklung komplexer Standardsoftware. Dazu werden zunächst strategische Ziele eines Unternehmens und der Markteinfluss betrachtet. Beides beeinflusst sowohl die Gestaltung des gesamten Entwicklungsprozesses als auch den Aufbau einer komplexen Standardsoftware erheblich. Wesentliche Ausprägungen sind die Entwicklung der Software in Versionen und Variationen. Der Einfluss spiegelt sich im gesamten Inhalt der Arbeit wieder und wird an entsprechenden Stellen genauer erläutert.

1.1.1 Übergreifende strategische Ziele

In der Betriebswirtschaftslehre hat sich die sehr plakative Bezeichnung *Magisches Dreieck* für die bestimmenden Faktoren unternehmerischen Handelns eingebürgert. Darunter wird das gegenseitige Wechselspiel zwischen den Erfolgsfaktoren Qualität, Zeit und Kosten verstanden. Durch eine adäquate Kompromissbildung zwischen diesen drei Faktoren wird letztendlich ein kontinuierlicher Gewinn angestrebt. Die Erfolgsfaktoren beeinflussen sämtliche Vorgehensweisen und Praktiken in einem Unternehmen, insbesondere die gesamte Wertschöpfungskette eines Produkts von der Entwicklung bis hin zum Verkauf.

Qualität umfasst allgemein die Erfüllung der unmittelbaren Anforderungen an das Produkt einschließlich Lieferung und Service. Bei Individualanfertigungen sind erzielbarer Preis und Qualität des Produktes unmittelbar gekoppelt. Je höher die Qualität eines Produktes ist, desto mehr lässt sich dafür von einzelnen Kunden verlangen. Wird ein Produkt hingegen mehrfach in der gleichen Ausprägung für einen Markt verkauft, so bestimmt sich der Umsatz sowohl aus dem Preis als auch aus der Anzahl verkaufter Produkte. Konsequenterweise kann ein Produkt durchaus gewinnbringend sein, wenn die Qualität nicht maximiert wird. Daher wird in dieser Arbeit von einem modifizierten magischen Dreieck ausgegangen. Anstatt der Qualität wird neben den Zielen Zeit und Kosten die sogenannte Kundenorientierung in den Mittelpunkt der Tätigkeiten gestellt (siehe Abbildung 1). Die drei Begriffe werden in den kommenden Abschnitten erläutert.

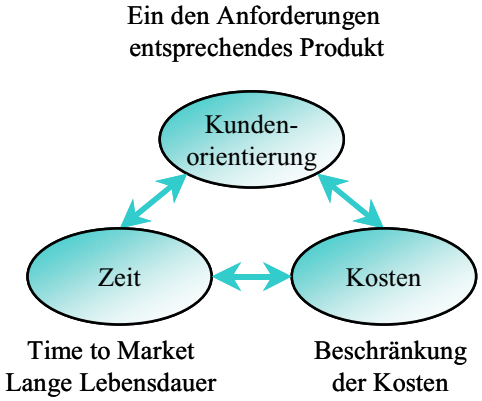


Abbildung 1 – Übergreifende strategische Ziele

Kundenorientierung

Unter Kundenorientierung wird nicht isoliert die Erhöhung der Qualität betrachtet. Im Mittelpunkt steht vielmehr die Maximierung der Umsatzzahlen, deren Haupteinflussfaktoren Qualität und Preis sind. Die Qualität einer Software wird im Übrigen von einzelnen Kunden unterschiedlich eingeschätzt. Dies liegt letztendlich an unterschiedlichen individuellen Anforderungen, die in einer Standardsoftware nicht immer im gleichen Maß erfüllt sein können.

Zeit

Für die Zeit sind sowohl kurzfristige als auch langfristige Aspekte zu berücksichtigen. Kurzfristig gesehen, werden möglichst kurze Durchlaufzeiten (engl. *Time to Market*) der Anforderungen angestrebt. Darunter ist der Zeitraum vom ersten Bekannt werden bis zur Realisierung einer Anforderung zu verstehen. Durch die Verkürzung der Durchlaufzeit kann auf Veränderungen im Markt schneller reagiert werden, was Wettbewerbsvorteile gegenüber Konkurrenten schafft. Insbesondere bei leicht substituierbarer Software ist eine schnelle Reaktion erforderlich. Derartige Software wird mit ähnlicher Funktionalität auch von Mitbewerbern angeboten und kann gegebenenfalls durch diese ersetzt werden. Durch die schnelle Reaktion auf neue Anforderungen wird also damit das Abwandern zu Konkurrenten verhindert.

Bei komplexer Standardsoftware mit hoher Kundenbindung ist hingegen diese Reaktionszeit nicht so entscheidend. Beispielsweise bei der betrieblichen Standardsoftware SAP R/3 sind für die Inbetriebnahme beim Kunden sehr viele aufwändige Anpassungen nötig. Daher ist hier der Umstieg des bestehenden Kundenstamms zu anderen Softwareherstellern weniger wahrscheinlich. Für den Erwerb von Neukunden ist jedoch auch hier eine möglichst aktuelle Software nach wie vor absolute Notwendigkeit. Eine hohe Kundenbindung wird auch dadurch erreicht, dass eine Software eine sehr hohe Marktdurchdringung erreicht hat und dadurch zum De-facto- Standard geworden ist. Beispielsweise ein Betriebssystem wie Microsoft Windows kann sich allein dadurch über einen längeren Zeitraum im Markt behaupten, dass wenig Konkurrenzprodukte mit einer ähnlichen Verbreitung existieren.

Der Zwang zu kurzen Durchlaufzeiten ist auch von der Marktpräsenz einer Anforderung abhängig. Eigene innovative Anforderungen können zu einem vom Unternehmen weitgehend selbst festgelegten Zeitpunkt erfolgen. Ist die Anforderung hingegen bereits auf dem Markt präsent, da ein wichtiger Konkurrent sie bereits in seiner Standardsoftware integriert hat, dann ist eine möglichst schnelle Realisierung erforderlich. Die Dringlichkeit der Umsetzung einer Anforderung wird im Rahmen dieser Arbeit durch die Bewertung von Anforderungen in Abschnitt 6.5 berücksichtigt. Produktions- und Lieferzeiten materieller Produkte können für Software vernachlässigt werden. Die Produktion von Software ist verhältnismäßig einfach, unterscheidet sich kaum zwischen verschiedenen Softwareprodukten und ist dadurch kein verkaufsentscheidender Faktor.

Für kurze Durchlaufzeiten muss der gesamte Entwicklungsprozess wie auch die Software selbst flexibel gestaltet sein. Langfristige Zeitaspekte bezwecken, die Standardsoftware über einen längeren Zeitraum konkurrenzfähig zu halten. Daher muss die Software auf einen Wandel im Markt vorbereitet sein. Dieser Aspekt wird in Abschnitt 1.1.2 erläutert.

Kosten

Der Gewinn ist neben einem Verkaufserfolg durch geringe Gesamtkosten beeinflusst. Für Software sind die eigentlichen Produktionskosten im Verhältnis zu den Entwicklungskosten sehr gering, weshalb hier Kosten vor allem durch einen effizienten Entwicklungsprozess gespart werden können. Daneben können Kosten durch die frühzeitige Auswahl und Bewertung zu realisierender Anforderungen reduziert werden.

Gewichtung übergreifender Ziele

Wie eingangs des Abschnitts erwähnt, sind von den übergreifenden Ziele sämtliche Vorgehensweisen und Praktiken eines Unternehmens abhängig. In der Regel wird eine gewisse Ausgewogenheit der Zielerreichungen angestrebt. Je nach aktueller Marktsituation kann dabei die Gewichtung einzelner Ziele unterschiedlich ausfallen.

Hat ein Hersteller einer Standardsoftware z. B. die entsprechende Marktmacht, so muss er - verglichen zu seinen Konkurrenten - ein geringeres Augenmerk auf die schnelle Reaktion richten. Anforderungen, die seine Konkurrenz bereits realisiert hat, führen dann nicht zur Abwanderung der Kunden. Kunden warten in diesen Fällen lieber ab, bis der Marktführer diese Anforderungen realisiert. Folglich kann bei großer Marktmacht auch der angekündigte Auslieferungszeitpunkt leicht in die Zukunft hinausgeschoben werden.

Anders ist die Situation für einen Hersteller einer Standardsoftware mit geringer Marktmacht. Falls dessen Software Anforderungen nicht erfüllt, die Konkurrenzprodukte bereits realisieren, muss dieser Rückstand zeitweise schnell aufgeholt werden. Dann sind die Anforderungen schnell zu realisieren, unabhängig von zu erwartenden hohen Kosten. Die Kundenorientierung ist dann gegenüber der Kostenreduktion höher priorisiert.

Besonders extrem ist die Möglichkeit finanzkräftiger Softwarehersteller. Anstatt in der eigenen Software Änderungen vorzunehmen, wird der Konkurrent mit der besten Software aufgekauft und entsprechende Anforderungen in das eigene System integriert. Hier kann also im Extremfall in Ruhe abgewartet werden, bis Anforderungen im Konkurrenzprodukt realisiert sind, bevor dann die gesamte Firma „geschluckt“ wird.

1.1.2 Markteinfluss

Der Markteinfluss äußert sich für die Entwicklung komplexer Standardsoftware in drei wesentlichen Merkmalen. Dies sind die Unsicherheit der Anforderungen und des Potentials des Marktes, der kontinuierliche Wandel im sehr innovativen Softwaremarkt und das breite Spektrum variierender Interessen verschiedener Kunden.

Unsicherheit

Bei Standardsoftware müssen gesamtheitliche Anforderungen und ein entsprechendes Umsatzpotential erschlossen werden. Hier ist - im Gegensatz zur Individualsoftware - keine vollständige Befragung aller Kunden möglich. Umgekehrt sind Anforderungen einzelner Kunden nicht immer marktrelevant (vgl. [Dei98a]). Zu eroierende Quellen müssen daher sorgfältig ausgewählt und darüber hinaus adäquat bewertet werden. Aufgrund eines unsicheren Umsatzpotentials ist darüber hinaus auch eine Budgeterung der Kosten schwer durchzuführen.

Wandel

Der Softwaremarkt unterliegt aufgrund sich ändernder Anforderungen und neuer Technologien einem kontinuierlichen *Wandel*. Hierauf muss ein Hersteller von Standardsoftware schnell reagieren. Er kann aber auch selbst den Wandel durch eigene Innovationen voranntreiben, um somit Wettbewerbsvorteile zu erlangen. Schnelle Reaktionszeiten erfordern flexible Entwicklungsprozesse und flexible Software, um möglichst kostenoptimal Neuerungen integrieren zu können. Sowohl für die Reaktionsfähigkeit als auch für die Entwicklung von Innovationen müssen Trends analysiert und langfristige Visionen entwickelt werden. Damit werden künftige Weiterentwicklungen der Standardsoftware sowohl aus marktorientierter als auch aus technischer Sicht geleitet.

Vielfalt

Die große *Vielfalt* individueller Kundeninteressen resultiert in unterschiedlichen, teilweise sogar widersprüchlichen Anforderungen. Durch die erwähnten Kosten - und Zeitrestriktionen kann immer nur ein Teil der gestellten Anforderungen realisiert werden. Daher muss sich ein Softwarehersteller auf gewisse Anforderungen beschränken.

1.1.3 Direkt abgeleitete praktische Situation

In der Praxis der Softwareentwicklung werden fortwährendend Kompromisse der übergreifenden strategischen Ziele unter Berücksichtigung der Markteinflüsse gebildet. Hierfür hat sich für komplexe Standardsoftware die Entwicklung Versionen und Variationen durchgesetzt. In den folgenden Absätzen werden diese Begriffe erläutert und dargestellt, inwiefern hiermit zur Erfüllung der eben angesprochenen Ziele beigetragen wird.

Versionierung

Unter Versionsentwicklung (oder kurz Versionierung) wird in dieser Arbeit die Entwicklung von Software verstanden, die in bestimmten zeitlichen Abständen periodisch weiterentwickelt wird und jeweils in der neuen Version verkauft wird. Meist werden relativ starre Zykluszeiten für die Entwicklung der einzelnen Versionen festgelegt, um eine gewisse Planungsgrundlage zu erhalten. Die Versionierung wird in den Abschnitten 4.4.4 und 7.5.3 noch näher beleuchtet.

Mit Hilfe der Versionierung wird dem ständigen Wandel der Marktanforderungen Rechnung getragen. Daneben bietet die Versionierung die Möglichkeit, unsichere Anforderungen stufenweise über einzelne Versionen hinweg an tatsächlich auf dem Markt vorhandene Bedürfnisse anzugleichen.

Variationen

Unter Variationen werden spezifische Anpassungen an Anforderungen einzelner Kundengruppen verstanden, die bereits während der Herstellung eines Softwaresystems vorgesehen sind. Die Bildung von Variationen stellt einen Kompromiss zwischen Kundenorientierung und Kosten dar. Das Marketing fordert eine möglichst breite Abdeckung der Vielfalt der Kundenwünsche, um dem ständigen Wettbewerb Rechnung zu tragen.

Die Variationsbildung muss unter Berücksichtigung der Einhaltung bestimmter Kostenrahmen erfolgen. Wesentlich hierzu ist die richtige Wahl der sogenannten *Variationstiefe,* also bis zu welchem Grad ein Hersteller komplexer Standardsoftware individuelle Kundenwünsche berücksichtigt. Ein Extrem der Aufwandsreduzierung auf Kosten individueller Bedarfe ist die Herstellung einer „reinen" Standardsoftware. Diese integriert nur Anforderungen, die allen Kunden gemeinsam ist. Das andere Extrem ist die Entwicklung von Individualsoftware, die ausschließlich Anforderungen eines einzelnen Kunden erfüllt. Zur Kompromissbildung zwischen Entwicklungsaufwänden und der Individualität der Kundenwünsche, erlaubt komplexe Standardsoftware häufig eine gewisse Anpassbarkeit. Die Variationsbildung wird in den Abschnitten 4.4.2 und 7.5.1 erläutert.

Die Identifikation gemeinsamer Anforderungen unterschiedlicher Variationen spielt eine wesentliche Rolle, um durch Wiederverwendung von Implementierungsteilen weitere Kosten zu reduzieren. Auch darauf wird in Abschnitt 4.4.2 eingegangen.

1.2 Problemstellung und notwendige Unterstützung

Aus der Ausgangssituation des Abschnittes 1.1 leiten sich diverse Problemstellungen für das Requirements Engineerings komplexer Standardsoftware ab. Diese werden in den folgenden Unterabschnitten 1.2.1 bis 1.2.5 erläutert. Deren Gliederung orientiert sich an dem Prozessmodell von Pohl [Poh96], das Tätigkeiten des Requirements Engineerings in vier Phasen unterteilt. Dies sind das Herausarbeiten (engl. Elicitation), die Verhandlung (engl. Negotiation), die Spezifikation (engl. Specification) und die Validierung/Verifikation (Validation / Verification). Durch die starke Verzahnung von Tätigkeiten des Requirements Engineerings lassen sich diese Phasen nicht immer scharf voneinander trennen. Daher dient diese Unterteilung einer Strukturierung zu betrachtender Aspekte.

Unterschiede des Requirements Engineerings von Individualsoftware und von Standar d-software werden im Rahmen der wissenschaftlichen Einordnung in Abschnitt 1.3.1 erläutert. In Kapitel 2 werden darüber hinaus bestehende Ansätze des Requirements Engineerings vor- und Bezüge zu Ansätzen der vorliegenden Arbeit hergestellt. Da das Requirements Engineering von Standardsoftware nicht isoliert behandelt werden kann, wird auch die Einbettung in sein Umfeld betrachtet (siehe Abschnitt 1.2.5). Dazu gehören ihre Kopplung zur vorgelagerten Marktpositionierung und sowie die Verfolgbarkeit der Anforderungen mittels Requirements Management und Konfigurations -Management.

1.2.1 Herausarbeiten von Anforderungen

Mit dem Herausarbeiten sollen Anforderungen an ein zu erstellendes Softwarepr odukt identifiziert werden. Diese werden in den weiteren Phasen diskutiert, konkret isiert und schließlich detailliert niedergeschrieb en. Vor allem müssen aus vagen Äußerungen der Kunden deren damit verbundenen Anforderungen aufg espürt werden.

Für Anforderungen existieren unterschiedliche Quellen und Kanäle. Unter Quelle ist die eigentliche Herkunft einer Anforderung zu verstehen. Mit de m Kanal wird hingegen der Anlass oder Weg, wie eine Anforderung zu einem Softwarehersteller gelangt, versta nden. Für das Herausarbeiten von Anforderungen für Individualsoftware werden nur relativ wenige Quellen und Kanäle genutzt, da hier die unmittelbaren Ansprechpartner bzw. Kunden bekannt sind. Dagegen muss bei der Entwicklung komplexer Standardsof tware eine ganze Reihe unterschiedlicher Quellen und Kanäle genutzt werden. Auf Que llen und Kanäle bei der Entwicklung komplexer Standardsoftware wird in Absch nitt 7.3 ausführlich eingegangen.

Das Herausarbeiten kann durch unterschiedliche Techniken unterstützt werden. In A bschnitt 2.1 werden existierende Ansätze hierzu vorgestellt. Welche hiervon sich fü r den Einsatz bei komplexer Standardsoftware eignen, wird in Abschnitt 7.3 erläutert.

Das Herausarbeiten ist durch verschiedene Randfaktoren beeinflusst. Dazu gehört das Hintergrundwissen und unterschiedliche Interessen am He rausarbeiten beteiligter Ansprechpartner der Kunden. So gestaltet sich teilweise allein die Art der Kommunikat ion schwierig. Andererseits spielen verschiedene soziale, organisato rische und gesetzliche Einflüsse eine Rolle. Diese werden meist gar nicht ode r nur teilweise in den Tec hniken berücksichtigt. Hinzu kommt noch das Herausfiltern der richtigen Anforderungen aus den Aussagen der Kunden. Dies ist eine der Hauptaufgaben der Verhandlung sphase.

1.2.2 Verhandlung von Anforderungen

Üblicherweise treten bei gesam melten Anforderungen Konflikte auf. Diese sollen in der Verhandlungsphase (engl. Negotiation) aufgedeckt und Lösungsalternat iven gesucht werden. Das Ergebnis der Verhandlung für Individualsoftware ist die Au swahl genau einer bestimmten Alternative. Aufgrun d der möglichen Realisierung von Variationen (vgl. Abschnitt 1.1.3) können bei Standardsoftware mehrere Alternativen gleichzeitig gewählt werden. Die prinzipielle Vorgehensweise der Konsolidierung von Anforderu ngen unterscheidet sich dadurch allerdings nicht von der Individualsoftwar eentwicklung.

Bei Standardsoftware gehört zur Verhandlungsphase zusätzlich die Versionspl anung, also die Verteilung von Anforderungen auf künftige Versionen. Auf Probleme der Vers i-onsplanung wird in den Abschnitten 4.4.4 und 7.5.3 genauer eingegangen.

Wegen beschränkter Zeit-, Personal- und Kapitalressourcen können nicht alle Anford e-rungen gleichzeitig realisiert werden. Daher müssen diese bewe rtet und priorisiert wer-den. Durch die Prioritäten wird die Reihenfolge der Anforderungsrealisi erung festgelegt. Bei Individualsoftware können hierzu im Zweifelsfalle unmittelbar Kunden befragt we r-den. Da Standardsoftware in einem Markt verkauft wird, best eht nur zu einem kleinen Anteil der Kunden ein direkter Kontakt. Daher müssen hier Schätzungen über Bewertu n-gen gemacht werden. Diese stützen sich auf Erfahrungen mit ähnlichen Produ kten in der Vergangenheit stützen oder werden aus Umfrageergebnissen hochg erechnet. Eine aus-führliche Erläuterung der Probleme und eine Vorstellung einer Methodik zur Bewe rtung und Priorisierung wird in Kapitel 6 gegeben.

1.2.3 Spezifikation von Anforderungen

Die Spezifikationsphase (engl. Specification) dient der Spezifikation der während der Herausarbeitungs- bzw. der Verhandlungsphase ermittelten Anforderu ngen.

Diese ist letztendlich Ausgangspunkt für die Erstellung des Systems. Eine Spezifikation dient einerseits als Kommunikationsmittel zur Validierung und zum weiteren Herausarbeiten von Anforderungen zwischen Kunde und Entwickler. Darüber hinaus muss sie Entwicklern alle zur Systementwicklung benötigten Informationen li efern. Schließlich ist sie auch die Basis für eine Verifikation nach der Entwicklung . Spätestens mit der Fertigstellung einer Software muss eine in sich konsistente Spezifikation existi e-ren, die der tatsächlich entwickelten Software entspricht. Je flexibler der Entwick -lungsprozess gestaltet ist, desto später kann eine Spezifikation konsi stent werden. Änderungen der Spezifikation können durch neue Anforderungen oder durch falsch kalkulierte Realisierungskosten und -Zeiten einzelner Anforderungen verursacht sein.

Um ihre Aufgaben zu erfüllen, muss die Spezifikation in für verschiedene Pers onengrup-pen verständlichen Sichten dargestellt sein und auf einer möglichst formalen Basis fu n-dieren. Damit gibt sie dem Entwickler eine eindeutige Vorgabe zur Erstellung der Software.

Die inhaltliche Spezifikation von Anforderungen unterscheidet sich zwis chen Individual-software und komplexer Standardsoftware nicht. Bei Standar dsoftware müssen jedoch zusätzlich Versionen und Variationen adäquat spezifiziert werden können. Spezielle Problemstellungen und adäquate Lösungen hierzu werden in Kapitel 5 erläutert.

1.2.4 Validierung/Verifikation von Anforderungen

Die Begriffe Validierung und Verifikation (engl. Validation/Verification) werden häufig missverständlich und vermischt gebraucht, obwohl sie in ihrer Intention zweierlei Dinge sind. In dieser Arbeit wird unter Validierung die Überprüfung verstanden (vgl. [Rak97]), ob das zu entwickelnde System den Wünschen des Kunden en tspricht. Eine Verifikation dagegen dient dazu, zu kontrollieren, ob ein bereits erstell tes System den spezifizierten Anforderungen entspricht. Andere Definitionen ve rbinden den Begriff der Verifikation

mit dem Beweisen von Eigenschaften des Systems. Dagegen wird dort unter Validierung das nicht beweisbare Nachprüfen von Eigenschaften durch verschiedene Methoden verstanden.

Die Validierung von Standardsoftware kann neben Kundenbefragungen auch mit Vertretern des Marketings des Softwareherstellers erfolgen, die eine Kundensicht vertreten. Eine erste Validierung ist bereits nach der Spezifikation möglich, bevor die Software entwickelt wird. Die letztendliche Validierung lässt jedoch erst sich aus dem Verkaufserfolg des Produktes ablesen.

Die Verifikation erfolgt auf Grundlage der rein inhaltlichen Spezifikation. Da diese sich nicht zwischen Standardsoftware und Individualsoftware unterscheidet, existieren hier für die Verifikation keine eigenen Problemstellungen. Die letztendliche Verifikation muss anhand einer fertiggestellten Version der Software erfolgen.

1.2.5 Einbettung des Requirements Engineerings

Das Requirements Engineering für Standardsoftware ist das Bindeglied zwischen dem betriebswirtschaftlichen Marketing und der technischen Entwicklung. Daher ist es nicht unabhängig von diesen beiden Bereichen zu betrachten.

Marktpositionierung

Der Marktbezug von Standardsoftware ist nicht prinzipiell anders, als bei anderen am Markt verkauften Produkten. Daher lassen sich dort übliche Marketingaktivitäten auf die Software übertragen. Hierzu müssen zunächst Marketingstrategien entwickelt werden. Die Bestimmung von Zielmärkten, deren Größe und deren Verhalten, Gewinnzielen, Preisen, Verkaufsstrategien, Marketingbudgets, langfristige Produktstrategien usw. sind hier zentrale Aufgaben. Daneben sind Verkaufsanalysen sehr wichtig, das heißt die Abschätzung möglicher Umsätze aufgrund aktueller, vergangener und künftiger Umsatzzahlen anderer Produkte.

Die Bedeutung der Marktpositionierung eines Produktes unterstreichen in der Praxis zum Beispiel unterschiedliche Strategien von Microsoft, ihre Marktführerschaft beizubehalten beziehungsweise zu erkämpfen [Bag97]. Danach ist der Erfolg von Microsoft nicht eigenen Innovationen zu verdanken, sondern einer geschickten Politik Marktanteile zu generieren und diese durch eine Preis- und Lizenzpolitik und häufigen Weiterentwicklungen der eigenen Produkte auch zu halten.

Der Einfluss der Marktpositionierung äußert sich in vielen Stellen der Entwicklung komplexer Standardsoftware. Dazu gehören Vorgaben für Entwicklungszeiten und für zur Verfügung stehende Ressourcen. Darüber hinaus haben sie auch einen Einfluss auf grobe Struktur einer Standardsoftware und auf strategische Vorgaben. Die Einflüsse werden an den entsprechenden Stellen in dieser Arbeit beleuchtet.

Verfolgbarkeit

Neben der reinen Beschreibung von Anforderungen an ein zu erstellendes System müssen diese auch verfolgbar sein. Ziel der Verfolgbarkeit ist eine konsistente und lückenlose Sicherstellung der Kundenanforderungen in der letztendlichen Realisierung. Dazu

müssen einerseits Entscheidungen dokumentiert werden, um die Entstehung von Lösungen nachvollziehen zu können. Andererseits sollen bei der Änderung von Anforderungen die Auswirkungen auf die Implementierung der Software (und umgekehrt) einfach zu ermitteln sein. Somit muss eine Verfolgbarkeit von den frühen Phasen zu den späten Phasen als auch rückwärts möglich sein [Wie95].

In [Got95, GoF94] wird zwischen der horizontalen und der vertikalen Verfolgbarkeit unterschieden. Die horizontale Verfolgbarkeit beschäftigt sich mit der Evolution von Dokumenten innerhalb einer Phase, womit klassischerweise sich das *Konfigurations-Management* auseinandersetzt. Dagegen bezieht sich vertikale Verfolgbarkeit auf die Nachvollziehbarkeit der Übergänge durch die aufeinanderfolgenden Phasen schaffen. Die vertikale Verfolgbarkeit teilt sich in die Pre- und die Post-Verfolgbarkeit auf. Erstere soll die Entstehung einer Anforderungsspezifikation nachvollziehbar machen, während letzter die Verfolgung der Umsetzung der Spezifikation in ein fertiges System im Mittelpunkt hat. Für die Behandlung der vertikalen Verfolgbarkeit hat sich in der jüngeren Zeit der Begriff des *Requirements Managements* eingebürgert.

1.3 Ziel, Wissenschaftliche Einordnung und Gesamtkonzept der Arbeit

Ziel der vorliegenden Arbeit ist eine Methodik zur Unterstützung der Planung markt- und entwicklungsgerechter sowie Anforderungsspezifikationen komplexer Standardsoftware. Dabei werden insbesondere Problemstellungen der Variations- und der Versionsentwicklung berücksichtigt. In den folgenden Unterabschnitten wird die Arbeit wissenschaftlich eingeordnet und ihr Gesamtkonzept vorgestellt.

1.3.1 Wissenschaftliche Einordnung der Arbeit

In wissenschaftlichen Arbeiten im Software Engineering wurden bisher die besonderen Problemstellungen der Entwicklung komplexer Standardsoftware weitgehend vernachlässigt. Erst in den vergangenen Jahren wurde nicht zuletzt durch eine Veröffentlichung des Autoren der vorliegenden Arbeit die Sensibilität für dieses Thema geschärft (vgl. [Dei98b, REFSQ'99, REFSQ'00]).

In [CaB95] wurde jedoch ein Prozessmodell für die Entwicklung von Standardsoftware vorgestellt, das Elemente der Entwicklung kommerzieller materieller Produkte mit dem klassischen Software Engineering verbindet. Dieses Prozessmodell stellt ein Skelett für notwendige Entwicklungsaktivitäten dar, muss jedoch in jeder Entwicklungsphase verfeinert werden. Die vorliegende Arbeit konzentriert sich auf die Verfeinerung des Requirements Engineerings.

Aufgrund des unmittelbaren Marktbezuges ist der *Return on Investment* (vgl. Glossar) der Entwicklung komplexer Standardsoftware bis zur Fertigstellung unsicher. In der Marketingforschung (vgl. [Kot97]) existieren umfangreiche Ansätze zur Marktpositionierung (vgl. Abschnitt 1.2.5). Die vorliegende Arbeit konzentriert sich daher auf technische Aspekte des Requirements Engineerings. Diese stehen in enger Kooperation mit den erwähnten Marketingaktivitäten, werden jedoch hier nicht gesondert behandelt.

Hinsichtlich der Phasen des Requirements Engineerings (vgl. Abschnitt 1.2) konzentriert sich die vorliegende Arbeit auf die Phasen Herausarbeiten, Verhandlung und Spezifikation. Wie in Abschnitt 1.2.4 festgestellt wurde, bestehen bei der Validierung und Verifikation kaum Unterschiede zu Individualsoftware. Daher wird diese Phase nicht behandelt. Das Herausarbeiten beschäftigt sich vorrangig mit der Akquisition von Anforderungen. Die vorliegende Arbeit stellt eine Kombination bestehender Methoden im Umfeld komplexer Standardsoftware vor, die ein systematisches Herausarbeiten von Anforderungen ermöglicht (siehe Abschnitt 7.3).

Für die inhaltliche Verhandlung existieren wissenschaftliche Ansätze (vgl. Abschnitt 2.2). Die vorliegende Arbeit konzentriert sich daher auf spezifische Aspekte komplexer Standardsoftware. Dazu gehören die Festlegung von Variationen, deren Verteilung auf unterschiedliche Softwareprodukte, die Versionsplanung und adäquate Bewertungsmethoden. Die vorliegende Arbeit präsentiert ein Prozessmodell (siehe Kapitel 7) und Beschreibungskonzepte zur systematischen Bewältigung dieser spezifischen Aspekte.

Aufgrund der Komplexität der Entwicklung komplexer Standardsoftware werden für das Requirements Engineering besondere Techniken zur Spezifikation benötigt. Insbesondere müssen Variationen und Versionen von Anforderungen gesondert spezifiziert werden. Hinsichtlich dieser Aspekte werden in dieser Arbeit Erweiterungsmöglichkeiten bestehender Spezifikationstechniken diskutiert (vgl. Kapitel 5).

Viele wissenschaftliche Ansätze beschränken sich auf die Verfolgbarkeit von Anforderungen ausgehend von einer Spezifikation bis zur Fertigstellung der Software (vgl. [Wie95]) und auf die Verfolgbarkeit des Herausarbeitens von Anforderungen (vgl. [Got95]). Die Verfolgbarkeit von Verhandlungen hingegen wird stark vernachlässigt und muss insbesondere bei oben erwähnten zusätzlichen Aktivitäten sichergestellt werden. Dazu wird in dieser Arbeit ein Modell von Entwicklungsprodukten (vgl. Kapitel 4) aufgestellt, das die Verfolgbarkeit von Anforderungen gerade durch die unterschiedlichen Stufen der Verhandlung unterstützt. Darüber hinaus wird die Verfolgbarkeit von Entscheidungen sichergestellt. Dazu werden Aktivitäten systematisch als Entscheidungssituationen beschrieben, in denen sowohl die Entscheidungsfindung als auch die Bewertung von Anforderungen hervorgehoben werden. Damit werden sowohl die horizontale Verfolgbarkeit als Aufgabe des Konfigurations-Managements als auch die vertikale Verfolgbarkeit als Aufgabe des Requirements Managements in die Methodik integriert.

Darüber hinaus wird in der gesamten Arbeit ein besonderes Augenmerk auf die schnelle und effiziente Ausführbarkeit der Aktivitäten des Requirements Engineerings gelegt. Damit wird eine schnelle Reaktionsfähigkeit des gesamten Entwicklungsprozesses gewährleistet.

1.3.2 Gesamtkonzept der Arbeit

Um oben aufgeführte Ziele zu erreichen wird in dieser Arbeit eine Methodik für das Requirements Engineering komplexer Standardsoftware vorgestellt. Der Kern hiervon gliedert sich in ein Prozessmodell und in ein Modell der Entwicklungsprodukte für darin zu verarbeitende Informationen vorgestellt. Einen Überblick gibt hierzu Abbildung 2. Dieser Abschnitt erläutert zunächst diesen Kern, bevor das Gesamtkonzept der Arbeit vorgestellt wird.

Kern der Arbeit

Die linke Seite der Abbildung 2 zeigt eine Aufteilung des Prozessmodells in die vier Phasen *Herausarbeiten von Anforderungen*, *Definition der Marktsicht*, *Definition der Systemsicht* und *Definition der Entwicklungssicht*. Jede Phase ist für die Erarbeitung eines bestimmten Teils des Modells der Entwicklungsprodukte zuständig. Dies illustriert die rechte Seite der Abbildung. Die folgenden Absätze erläutern kurz die vier Phasen.

Hierzu dient ein Auszug des durchgehenden Beispiels aus Kapitel 7. Er handelt sich hier um das Produktsystem *Pisa*, das Postkunden beim Versenden größerer Mengen von Poststücken unterstützt. Wesentliche Aufgabe dieses Produktsystems ist eine Optimierung des Briefportos und eine Unterstützung des Versands. Kernaufgaben des Produktsystems sind deshalb das Einlesen von Empfängeradressen, deren Sortierung entsprechend dem optimalen Porto und der Druck von Adressen und von der Post geforderter Formulare.

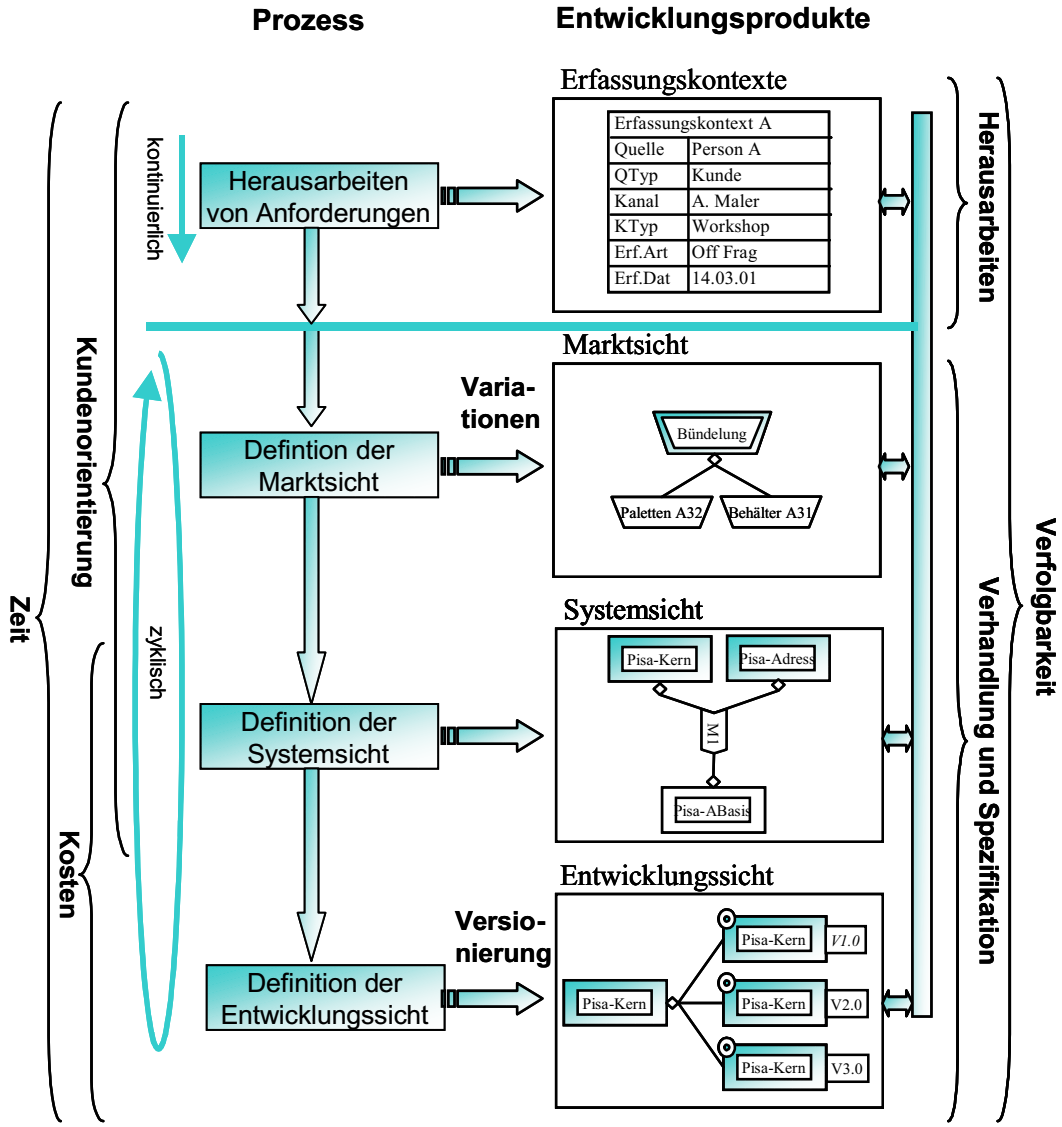


Abbildung 2 – Kern der Arbeit

In der ersten Phase des Prozessmodells (*Herausarbeiten von Anforderungen*) werden Anforderungen aus verschiedensten Quellen und Kanälen erarbeitet und erfasst. Um dem Wandel des Marktes gerecht zu werden, wird die Phase unabhängig von den Zyklen der Versionsentwicklung der Software kontinuierlich durchgeführt. Für *Pisa* sind Quellen z. B. Informationsprospekte oder verschiedenste Kunden. Kanäle für Kunden sind z. B. eigens organisierte Workshops oder Verkaufsgespräche. Das Prozessmodell stellt ein systematisches Vorgehen für das Herausarbeiten von Anforderungen aus unterschiedlichsten Quellen und Kanälen vor. Im Modell der Entwicklungsprodukte können für diese Phase Inhalte der Anforderungen und der Kontext, in dem diese erfasst wurden, festgehalten werden. Durch diesen Erfassungskontext kann die Herkunft jeder Anforderung eindeutig rückverfolgt werden.

In der zweiten bis vierten Phase erfolgt die Verhandlung und Spezifikation der erfassten Anforderungen. Diese Phasen werden in jedem Versionszyklus – also in jeder Weiterentwicklung – eines Produktsystems erneut durchgeführt. Die zyklische Abarbeitung entkoppelt die Weiterentwicklung vom kontinuierlichen Wandel des Marktes und bringt somit eine erhöhte Planungssicherheit für die Entwicklung.

Die Phase *Definition der Marktsicht* dient vorrangig der Strukturierung der Anforderungen, wie sie einem idealen Softwaresystem des anvisierten Gesamtmarkts entsprechen. Schwerpunkt ist die Identifikation von Variationen zur Erfüllung von Anforderungen unterschiedlicher Kundenprofile. In der Marktsicht des Modells der Entwicklungsprodukte werden diese Variationen und vielfältige Abhängigkeiten dokumentiert. Obige Abbildung illustriert z. B. die Variation der Bündelung von Poststücken. Sendungen in größerer Anzahl werden nicht einzeln bei der Post abgegeben, sondern müssen je nach Menge in posteigenen Behältern gebündelt oder auf Paletten gestapelt werden. Zur Unterstützung der Bündelung muss die Software zwei Variationen zur Verfügung stellen.

In der dritten Phase (*Definition der Systemsicht*) werden die Anforderungen auf ein tatsächlich realisierbares Produktsystem übertragen. Hier wird über die Realisierung einzelner Anforderungen und die Verteilung der Anforderungen sowie verfügbarer Ressourcen auf Softwareprodukte entschieden. Die Systemsicht des Modells der Entwicklungsprodukte umfaßt die Strukturierung des Produktsystems in Softwareprodukte und die Dokumentation zugeordneter Anforderungen und Entwicklungsressourcen. Der Ausschnitt der Systemsicht in obiger Abbildung illustriert vereinfacht die Struktur von *Pisa*. Neben einem Softwareprodukt zur Unterstützung der Portooptimierung (*Pisa-Kern*) wird auch eine Adressverwaltung (*Pisa-Adress*) entwickelt. *Pisa-Kern* umfasst Funktionen zum Einlesen bzw. Ausgeben von Adressen sowie die Optimierung bzw. Unterstützung des Versands der Poststücke. *Pisa-Adress* erfüllt neben Einlesen und Ausgeben von Adressen zusätzlich eine komfortable Verwaltung von Adressbeständen. Wesentliche Funktionalitäten der beiden Softwareprodukte überschneiden sich. Dies ist in der Systemstruktur durch die Definition eines sogenannten Basismoduls (*Pisa-ABasis*) berücksichtigt. Dieses realisiert gemeinsame Anforderungen der beiden Softwareprodukte und stellt sie diesen über eine sogenannte Modulschnittstelle (*M1*) zur Verfügung.

Die vierte Phase *Definition der Entwicklungssicht* befasst sich schließlich mit der Planung von Versionen einzelner Softwareprodukte und Basismodule. Dazu werden die Anforderungen auf künftige zu realisierende Versionen verteilt. Die Verteilung erfolgt entsprechend einer in vorhergehenden Phasen ermittelten Wichtigkeit der Anforderun-

gen. Bei der Versionsplanung werden wechselseitige Einflüsse zwischen Anforderungen einzelner Softwareprodukte und Basismodule berücksichtigt. Die Entwicklungssicht des Modells der Entwicklungsprodukte dokumentiert sowohl die Aufteilung der Softwareprodukte und Basismodule als auch die Zuordnung von Anforderungen auf Versionen. Die Abbildung 2 illustriert die Aufteilung von *Pisa-Kern* in drei Versionen *V1.0*, *V2.0* und *V3.0*.

Die Verfolgbarkeit der Anforderungen wird in diesem Kern durch die Integration der Teile des Modells der Entwicklungsprodukte bewerkstelligt. Einerseits lässt sich sowohl die Herkunft als auch die Zuordnung der Anforderungen zu einzelnen Systemteilen nachvollziehen. Andererseits kann auch der Realisierungszustand einzelner Anforderungen in späteren Entwicklungsphasen gut dokumentieret und somit auch verfolgt werden.

Die vorgestellte Struktur des Prozessmodells wurde wesentlich durch die unternehmerischen Ziele beeinflusst. Während die ersten beiden Phasen vorrangig der Kundenorientierung dienen, ist der Schwerpunkt der vierten Phase die Kostenorientierung. Die dritte Phase dient als Schnittstelle zur Verhandlung zwischen diesen beiden Zielen. Die Zeitorientierung spiegelt sich in jeder Phase dadurch wider, dass vorausschauend geplant und auf eine schnell umsetzbare Planung geachtet wird. Details des Prozessmodells sind in Kapitel 7, Details des Modells der Entwicklungsprodukte in Kapitel 4 zu finden.

Der Übersichtlichkeit halber wurde in der Abbildung auf eine Darstellung von Rollen verzichtet. Rollen legen Verantwortlichkeiten für Entwicklungstätigkeiten fest. Auch die Definition der Rollen ist von den strategischen Zielen Zeit, Kundenorientierung, und der Kosteneinhaltung geprägt. Die Zeit schlägt sich in der Trennung zwischen strategischen und operativen Rollen nieder. Darüber hinaus wird zwischen verkaufsorientierten und entwicklungsorientierten Rollen unterschieden. Verkaufsorientierten Rollen zielen auf hohe Umsatzzahlen ab und verkörpern damit die *Kundenorientierung*. Demgegenüber sorgen entwicklungsorientierte Rollen hauptsächlich für eine Kostenoptimierung.

Gesamtkonzept

Der Kern der Arbeit ist der zweite Teil des in Abbildung 3 illustrierten dreiteiligen Gesamtkonzepts der Arbeit. Neben dem Kern besteht dies aus grundlegenden Modellierungskonzepten und methodischen Anleitungen. In den kommenden Absätzen wird ein Überblick über noch nicht erörterte Teile der Arbeit gegeben.

Die *grundlegenden Modellierungskonzepte* definieren ein neuartiges Schema zur Beschreibung von Softwareentwicklungsprozessen (siehe Kapitel 3). Es bildet die Grundlage für die systematische Beschreibung des Requirements Engineerings komplexer Standardsoftware. Das Schema gliedert sich in Beschreibungen des Entwicklungsprozesses und der Entwicklungsprodukte. Mit Hilfe des Schemas werden die Verfolgbarkeit von Anforderungen und Entscheidungen sowie die Systematisierung von Entscheidungsprozessen unterstützt.

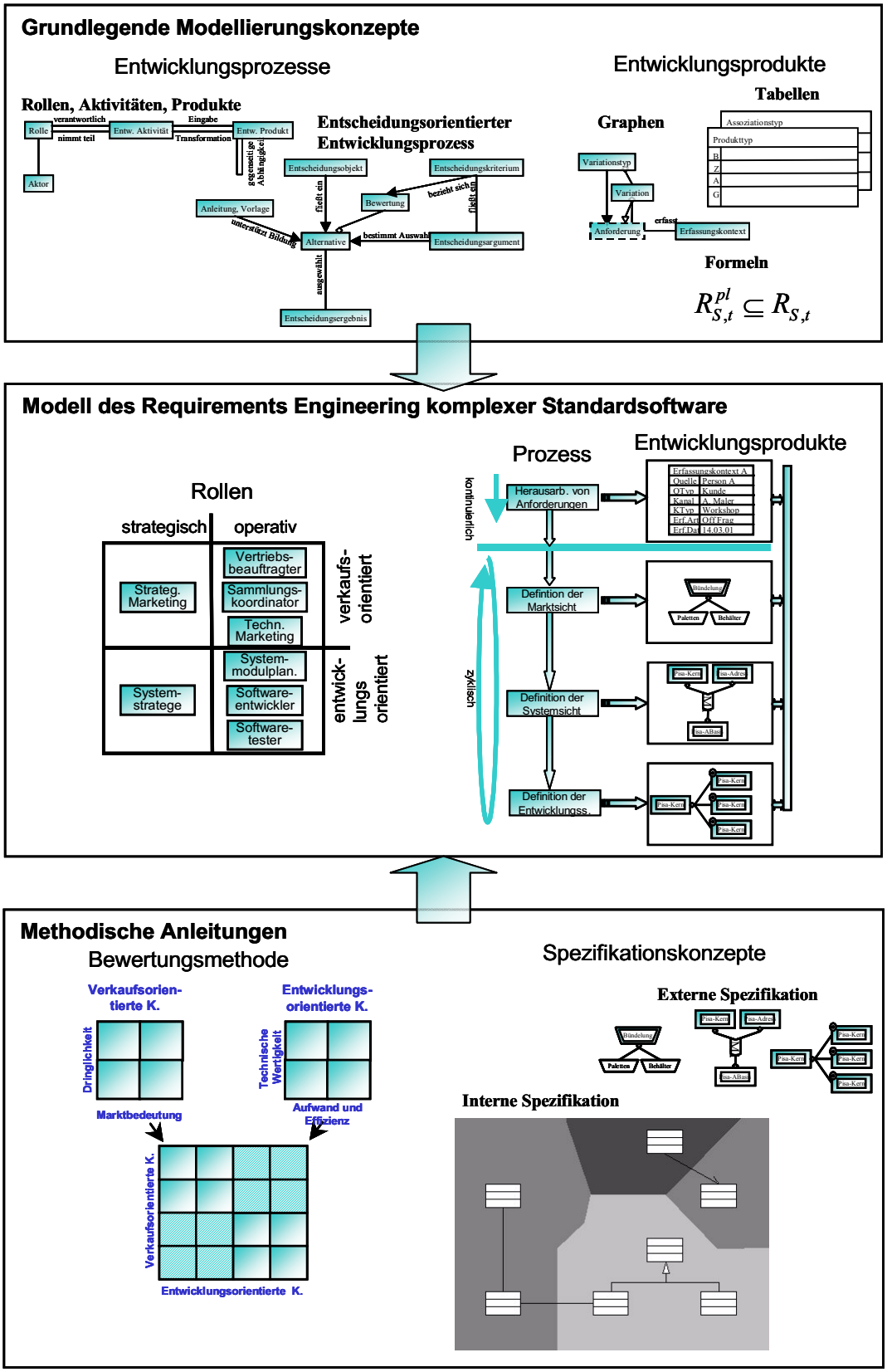


Abbildung 3 – Gesamtkonzept der Arbeit

Ein Entwicklungsprozess besteht prinzipiell aus Rollen, Aktivitäten und Entwicklungsprodukten. Rollen definieren Verantwortlichkeiten an der Entwicklung beteiligter Personen. Durchzuführende Entwicklungstätigkeiten werden durch Aktivitäten strukturiert. Schließlich werden alle Vorgaben, Anleitungen und Ergebnisse von Aktivitäten durch Entwicklungsprodukte dokumentiert.

Zur Systematisierung einzelner Aktivitäten wird ein Modell eines entscheidungsorientierten Entwicklungsprozesses definiert. Jede Aktivität ist danach entweder eine Entscheidung oder eine Vorbereitung für eine andere Entscheidung. Aktivitäten sind so systematisch in zu lösende Fragestellungen, beteiligte Produkte, Anleitungen zur Alternativenbildung, Zielfunktion und entsprechende Bewertungskriterien aufgeteilt. Durch diese Strukturierung wird eine gesamte Entscheidung startend beim Vorgehen zur Alternativenbildung bis hin zur Auswahl eines Entscheidungsergebnisses verfolgbar.

Auch beim Konzept zur Modellierung der Entwicklungsprodukte ist die Verfolgbarkeit zentral. Diese wird durch eine starke Integration unterschiedlicher Entwicklungsprodukte erreicht. Mit Hilfe eines neuen Schemas können wesentliche Charakteristika, Assoziationen und Konsistenzbedingungen innerhalb und zwischen Entwicklungsprodukten festgehalten werden. Zur Beschreibung des Modells der Entwicklungsprodukte werden einander ergänzende graphische, tabellarische und formale Techniken eingesetzt. Graphische Beschreibungen dienen vorrangig der Übersichtlichkeit, tabellarische der systematischen Dokumentation von Inhalten und formale Techniken werden zur exakten Beschreibung von Konsistenzbedingungen eingesetzt. Im Vordergrund der Arbeit steht die Methodik des Requirements Engineerings. Die formalen Techniken unterstützen diese dadurch, dass sie die Entwicklung eines Werkzeugs zur Erstellung, Verarbeitung und Verwendung des Modells der Entwicklungsprodukte erlauben .

Der dritte Teil befasst sich mit der Vorstellung neuer *methodischer Anleitungen*. Es werden eine Bewertungsmethode und Ideen zur Spezifikation des Modells der Entwicklungsprodukte vorgestellt. Abhängig von der Tragweite einer Planungsentscheidung existieren unterschiedlich hohe Anforderungen an die Zuverlässigkeit von Bewertungen. Die Zuverlässigkeit hängt weitgehend vom in die Bewertung investierten Aufwand ab. Zur Optimierung des Aufwands wird eine Bewertungsmethode vorgestellt, die sich abhängig von der Tragweite der Planungsentscheidung skalieren lässt.

Weiterhin werden Techniken zur externen und internen Spezifikation der Entwicklungsprodukte vorgestellt. Die externe Spezifikation dient der übersichtlichen Darstellung einzelner Teile des Modells der Entwicklungsprodukte. Hierzu wird eine UML-Ergänzung mit Hilfe von Stereotypen vorgestellt. Interne Spezifikationen ergänzen detaillierte Spezifikationen von Anforderungen wie beispielsweise Klassendiagrammen oder Zustandsdiagrammen. Sie dienen dort der Darstellung von Versionen, Variationen und Realisierungszuständen einzelner Anforderungen. Dazu wird ein Konzept zur farblichen Untermalung derartiger Graphen vorgestellt.

1.4 Aufbau der Arbeit

Aufbauend auf das im vorherigen Abschnitt vorgestellte Gesamtkonzept gliedert sich die Arbeit, wie folgt:

Zunächst wird in Kapitel 2 ein Überblick über relevante Ansätze für die vorliegende Arbeit im Requirements Engineering gegeben. Hierzu werden Arbeiten zum Herausarbeiten, zur Verhandlung, zur Spezifikation und zur Verfolgbarkeit von Anforderungen vorgestellt.

Kapitel 3 präsentiert grundlegende Konzepte zur Modellierung der Softwareentwicklung. Der erste Abschnitt befasst sich vorrangig mit der Modellierung von Entwicklungsprozessen und stellt das Konzept des entscheidungsorientierten Entwicklungsprozesses vor. Im zweiten Abschnitt wird darauf eingegangen, wie Entwicklungsprodukte und gegenseitige Zusammenhänge modelliert werden können. Darüber hinaus wird ein Bezug zur Modellierung von Entwicklungsprozessen hergestellt. In beiden Abschnitten werden Prinzipien und Möglichkeiten zur Beschreibung eines Modells vorgestellt.

Darauf aufbauend wird in Kapitel 4 ein Modell der Entwicklungsprodukte für das Requirements Engineering komplexer Standardsoftware präsentiert. Der erste Abschnitt gibt einen Überblick über das Kapitel. Anschließend werden die Bestandteile des Modells, gegliedert nach Grundelementen, strategischer Strukturierung und operativer Strukturierung, vorgestellt. Schließlich endet das Kapitel mit einer Erläuterung von Nutzungspotentialen des Modells für das Requirements Engineering, das Requirements Management und das Konfigurations-Management.

Das anschließende Kapitel 5 präsentiert Spezifikationstechniken für das im vorhergehenden Kapitel vorgestellte Modell der Entwicklungsprodukte. Dabei erläutert es zunächst bestehende Ansätze zur Spezifikation und entwickelt daraus anschließend Ergänzungen bzw. neue Techniken.

In Kapitel 6 wird eine Bewertungsmethode präsentiert, die zur Unterstützung von Entscheidungen innerhalb von Entwicklungsprozessen angewendet werden kann. Im ersten drei Abschnitte beschäftigen sich mit bestehenden Bewertungsmethoden. So werden Elemente einer Bewertung und Merkmale bzw. Bausteine bestehender Bewertungsmethoden vorgestellt. Aufbauend auf existierenden Bewertungsmethoden werden im Anschluss daran neue Bewertungskonzepte für das Requirements Engineering komplexer Standardsoftware präsentiert.

Alle in den vorangehenden Kapiteln vorgestellten Teile der Arbeit werden mit dem in Kapitel 7 vorgestellten Prozessmodell für das Requirements Engineering komplexer Standardsoftware integriert. Der erste Abschnitt stellt ein Gesamtkonzept des Prozessmodells vor. Anschließend werden Rollen, das kontinuierliche Herausarbeiten und zyklische Phasen des Prozessmodells präsentiert. Ein durchgängiges Beispiel illustriert das gegenseitige Wechselspiel zwischen Aktivitäten des Prozessmodells, angewendeter Methoden und dem Modell der Entwicklungsprodukte.

Schließlich fasst Kapitel 8 die Arbeit zusammen und gibt einen Ausblick auf mögliche Praxiseinführungen und auf offene wissenschaftliche Fragestellungen.

Kapitel 2

Bestehende Requirements Engineering Ansätze

Dieses Kapitel widmet sich dem Stand der Wissenschaft und Praxis im Requirements Engineering. Das Requirements Engineering ist noch eine unreife Forschungsdisziplin. Dies zeigt sich darin, dass in verschiedenen neuerscheinenden Büchern (vgl. zum Beispiel [KS98, SS97a, Wie99]) ein sehr unterschiedliches Spektrum von Methoden und Ansätzen eingesetzt wird. Gute Artikel zur Einführung sind [Fin94, Poh96]. In [Fin94] wird eine Abgrenzung und eine Einordnung des Requirements Engineerings zu andern Themenfeldern vorgenommen. Insbesondere wird ein Bezug zur Organisation und zu unterschiedlichen Tätigkeiten innerhalb einer Softwarefirma hergestellt. [Poh96] beschäftigt sich mit Kernaktivitäten des Requirements Engineerings. Dazu wird ein Überblick über in der Praxis vorgeschlagene Standards, Verfahrensanweisungen und einsetzbare Beschreibungstechniken gegeben. Vor allem werden aktuelle Forschungsrichtungen vorgestellt. Das vorliegende Kapitel basiert auf Ausschnitten der eigenen Publikation [Dei98c]. Diese wurden um neuere Ansätze erweitert.

Die folgenden Abschnitte stellen hinsichtlich der Ziele der Arbeit (vgl. Abschnitt 1.3) relevante Forschungsergebnisse vor. Sie werden anhand der in Abschnitt 1.2 eingeführten Gliederung präsentiert. Ansätze, die in unterschiedlichen Phasen des Requirements Engineerings genutzt werden können, werden den Abschnitten aufgrund ihrer hauptsächlichen Verwendung zugeordnet. Im ersten Abschnitt 2.1 werden Ansätze zum Herausarbeiten von Anforderungen erläutert. Danach folgt in Abschnitt 2.2 eine Übersicht über Arbeiten zur Verhandlung. In Abschnitt 2.3 werden Spezifikationstechniken vorgestellt. Schließlich werden Ansätze zur Unterstützung der Verfolgbarkeit in Abschnitt 2.4 dargestellt. In den Abschnitten wird jeweils kurz erläutert, wie sich diese Ansätze in den neuen Vorschlägen der vorliegenden Arbeit einbetten. Das vorliegende Kapitel dient der Vorstellung allgemeiner Ansätze des Requirements Engineerings. Für spezielle Fragestellungen wird in den einzelnen Kapiteln auf weitere bestehende Ansätze gesondert eingegangen.

2.1 Herausarbeiten von Anforderungen

Bisherige Ansätze des Requirements Engineerings gehen ausschließlich von einem direkten Kontakt zum Kunden aus. Wie in Abschnitt 1.2.1 erläutert, trifft dies bei Standardsoftware nicht immer zu. Im folgenden Unterabschnitt 2.1.1 werden zunächst allgemeine Techniken zum Herausarbeiten von Anforderungen vorgestellt. Anschließend werden die Szenarioanalyse (Abschnitt 2.1.2) und das Prototyping (Abschnitt 2.1.3) erläutert. Diese

beiden Techniken unterstützen den besonders schwierigen Teil des Herausarbeitens von Abläufen für Software. In Abschnitt 7.3 wird erläutert, für welche Quellen und Kanäle einzelne Techniken zum Herausarbeiten von Anforderungen an Standardsoftware eingesetzt werden können.

2.1.1 Allgemeine Techniken zur Wissensgewinnung

Zu den allgemeinen Techniken gehören zunächst verschiedene Fragetechniken. Dazu zählen Fragebogeninterviews, offene Interviews und die Selbstbefragung [GoL93]. Bei Fragebogeninterviews werden Kunden strikt nur vorgefertigter Fragen interviewed. Damit können verschiedene Interviews gut miteinander verglichen werden. Offene Interviews zielen hingegen auf das Herausarbeiten spezifischer Anforderungen der Befragten ab. Die Selbstbefragung dient der Kosteneinsparung, da statt des Kunden, der Entwickler selbst aufgrund eigener Erfahrungen Anforderungen herausarbeitet. Der Erfolg von Befragungstechniken ist stark von der Erfahrung des Interviewers abhängig.

Neben den Befragungstechniken werden Anforderungen auch aus der künftigen Einsatzumgebung einer Software abgeleitet. Techniken sind hier hauptsächlich die Ethnographie und das unternehmenszielorientierte Herausarbeiten. Bei der Ethnographie arbeitet ein Mitarbeiter des Softwareherstellers über einen längeren Zeitraum in der Organisation des Kunden mit, um dort die Arbeitsweise zu evaluieren und daraus Anforderungen an das Produkt abzuleiten [Som+93]. Beim unternehmenszielorientierten Herausarbeiten werden Anforderungen aus den Unternehmenszielen durch eine sukzessive Verfeinerung hinsichtlich des künftigen Einsatzbereiches der Software abgeleitet. Hierzu zählt auch das Business Process Engineering (vgl. [Dav93]), bei dem Software als Hilfsmittel zur Optimierung von Geschäftsprozessen angesehen wird.

Zur Erleichterung des Herausarbeitens wird die Wiederverwendung früher gesammelter Anforderungen angestrebt. Grundlage sind Anwendungswissensbanken und abstrakte Musteranforderungen (vgl. [MaL97, Rob97, Fow96]). Eine andere Art der Wiederverwendung ist das Herausarbeiten von Anforderungen aus bereits bestehender Software. Hier wird mit Hilfe bekannter Reverse Engineering-Methoden gearbeitet (vgl. [RGS99]).

Die vorgestellten Techniken können zum Herausarbeiten von Anforderungen an komplexe Standardsoftware ebenso gut wie für Individualsoftware verwendet werden. Während Hersteller von Individualsoftware mehrfach ähnliche Projekte durchführen, in denen alte Anforderungen wiederverwendet werden können, ist dieser Fall jedoch bei Standardsoftware eher selten anzutreffen.

2.1.2 Szenarioanalyse

Die Szenarioanalyse ist ein populäres Mittel zum Herausarbeiten von Abläufen eines Softwaresystems. Szenarien definieren dabei exemplarische Nutzungen eines Systems. In der Szenarioanalyse werden gemeinsam mit dem Kunden oder Anwender Szenarien einer zu entwickelnden Software erarbeitet.

Szenarien sind vorrangig zum Herausarbeiten und Verhandeln von Anforderungen geeignet. Mit ihrer Hilfe kann sehr schnell ein grobes Verständnis über Abläufe einer zu entwickelnden Software gewonnen werden, was dann eine sukzessive Ableitung voll-

ständiger Verhaltensspezifikationen unterstützt. Darüber hinaus können sie als Grundlage zur Ableitung von Testfällen für eine spätere Validi erung dienen.

Szenariotechniken haben ein weites Anwendungsspektrum (vgl. [Pa e98]), die vom Herausarbeiten von Arbeitsabläufen (vgl. [Pae98, PTA94] bis hin zur Optimierung der Inte raktion des Nutzers mit der Oberfläche der Software reichen. Hierzu existieren sowohl Systematische Vorgehensweisen als auch spezifische Beschreibungstec hniken. Zur Ableitung von Testfällen für die Validierung werden verschiedene mehr oder weniger a utomatisierbare Vorgehensweisen [Hsi+94, ReR98] vo rgeschlagen.

Szenarien werden nach ihrer Erfassung in den einzelnen Techniken unterschiedlich we iterverwendet. Zur Umsetzung in eine Verhaltensspezifikation existieren unterschie dliche Ansätze. [RKW94] schlägt eine Synthese in eine Art Flussdiagramm vor. Andere Ansä tze leiten teilweise formal und damit automatisierbar [Hsi+94, KoM93, Krü00] und tei lweise methodisch informell [RBP+91] aus Szena rien Zustandsdiagramme ab. Teilweise dienen Szenarien als direkter Teil der Spezifikation. Einen au sführlichen Überblick zu Szenarien gibt [RBC+96].

Szenarien sind intuitiv verständlich und werden heute in der Praxis bereit s häufig eingesetzt. Sie werden neben dem Requirements Engineering auch beim Design von Software verwendet. Wegen ihrer allgemeinen Anwendbarkeit und ihrer häufig informellen Darstellung besteht jedoch die Gefahr, dass Szenarien teilweise von verschiedenen Leuten unterschiedlich interpretiert werden. Daher ist bei ihrem Einsatz stets auf eine klare Definition des konkreten Kontextes zu achten.

2.1.3 Prototyping

Verhaltensanforderungen lassen sich auch gut mit Hilfe des Prototypings herausgearbe itet. Prototyping und Szenariotechniken sind nicht klar voneinander zu trennen, bezi ehungsweise ergänzen sich oft gegenseitig. Mit Szenariotechniken werden meist visionäre Abläufe herausgearbeitet. Demgegenüber stehen beim Prototyping Abläufe an einem gegebenen ablauffähige n Programm im Mittel punkt. Neben der Unterstützung des Requ irements Engineerings kann Prototyping auch in unterschiedlichen Phasen der Softwar eentwicklung eingesetzt werden.

Zweck des Prototypings ist nach [Som96, PoW97] eine möglichst frü hzeitige Evaluation möglicher Lösungen anhand ausführbarer Programme. Die unmittelb are Sichtbarkeit eines Programms verbessert die Kommunikation zwischen Kunden und Softwareherste llern. Dies erhöht die Planungssicherheit, die in einem verringerten Änderungsrisiko durch die frühzeitige Abstimmung mit den Kunden resu ltiert.

Verglichen zu fertigen Softwareprodukten werden beim Prototyping Anforderungen erheblich reduziert, um schneller zu einem ausführbaren Programm zu kommen. Dazu werden sowohl Funktionsumfänge, die Qualität oder die Effizienz verringert. Das Exp erimentieren mit Pro totypen soll auch neue Anfo rderungen aufdecken.

Zum Prototyping existieren isolierte (vgl. [Som96, Dav92]) und mit dem gesamten Sof tware Engineering integrierte Prozessmodelle (vgl. [PPS92]). Die Erstellung von Prototypen wird durch spezielle Programmiersprachen, Bausteinbibliotheken und Code -

generatoren unterstützt (vgl. [Som96, PPS92, HuS97]). Prototypen können wie Szenarien Teil der Spezifikation oder Basis einer systematischen Überführung in e ine Spezifikation sein. Je nach Ansatz wird der Prototypencode in der Implementierung der endgü ltigen Software weiterverwendet oder verworfen.

Das Prototyping erhöht allgemein die Produktqualität und die Produktivität, da nur a n-hand des Prototypen ermittel te Anforderungen realisiert werden (vgl. [PoW97]). Darüber hinaus stehen längeren Entwicklungszeiten kürzere Testphasen gegenüber. Zur Einfüh - rung eines Prototyping-Verfahrens muss jedoch ein hoher Einarbeitungsaufwand in en t-sprechende Werkzeuge veranschla gt werden. Eine Gefahr besteht in der leichten Änderbarkeit des Prototypen. Dadurch wird häufig zu sehr an Details experimentiert, was den Spezifikationsprozess unnötig in die Länge ziehen kann. In Einzelfällen führte dies in der Vergangenheit zu Verzögerungen des Projektablaufs bis hin zur letztendlichen Auslieferung (vgl. Glossar) unausgereifter Prototypen.

2.2 Verhandlung von Anforderungen

Die Verhandlung von Anforderungen wird bisher wenig methodisch unterstützt. Eine Ausnahme stellen Arbeiten zur Konflikta uflösung mit Hilfe von *Viewpoints* (vgl. [NKF94]) dar, die in den anschließenden Absätzen erläutert werden. Darüber hinaus existieren einige wenige Ansätze zur Unterstützung der Bewertung und Priorisierung von Anforderungen. Hierauf wird in Kapitel 6 gesondert eingegangen.

Viewpoints stellen ein Konzept zur Entwicklung von Requirements Engineering Meth o-den zur Verfügung. Anforderungsspezifikationen werden hier aus unterschiedlichen Sichten aufgebaut. Einzelne Sichten fokussieren jeweils auf bestimmte Aspe kte eines zu entwickelnden Softwaresystems, die aufgrund verschiedener Entwicklerrollen oder u n-terschiedlicher Beschreibungstechniken betrachtet werden. Viewpoint -Ansätze unterstüt-zen den Entwurf von Requirements Engineering-Methoden, die eine automat ische Erkennung und Auflösung von Konflikten bereits während der Erstellung von Anford e-rungsspezifikationen ermögl ichen.

Requirements Engineering Methoden bauen sich nach den Viewpoint -Ansätzen aus Sichtvorlagen auf. Sichtvorlagen ent halten Grundelemente einer Sicht und einen sog e-nannte Arbeitsplane. Diese beschreiben unterschiedliche Konstruktionsschritte, mit deren Hilfe sich sukzessive aus den Grundelementen Sichten aufbauen lassen.

Beispiel:

> *Eine Sichtvorlage für einfache Zustandsautomaten besteht aus den Grundelemen-ten Zustand und Transition. Konstruktionsschritte beschreiben ein konsistentes Einfügen von Zuständen und Transitionen. Ein möglicher Konstruktionsschritt wä-re beispielsweise das Einfügen einer Transition zwischen zwei Zuständen.*

Ein Arbeitsplan kann darüber hinaus Konsistenzbedingungen innerhalb einer oder zu anderen Sichten und zugehörige Prüfaktionen enthalten. Für Konstruktionsschritte und Prüfaktionen wird eine exakte Formulierung gefordert, um später in Werkzeugen automa-tisch unterstützt werden zu können.

Bei konsequenter Anwendung des Viewpoint-Ansatzes wird die Verhandlung von Anforderungen wesentlich unterstützt. Voraussetzung hierfür ist jedoch eine weitgehend präzise und formale Formulierung einzelner Anforderungen. Viele Anforderungen können jedoch nur informell festgehalten werden, wodurch hierfür der Viewpoint-Ansatz nicht geeignet ist. Die vorliegende Arbeit greift den Viewpoint-Ansatz insofern auf, dass sie verschiedene Sichten der Entwicklung und eindeutige Bezüge zwischen den Sichten in einem Modell der Entwicklungsprodukte festlegt (siehe hierzu Abschnitte 3.1.1, 3.2 und Kapitel 4).

2.3 Spezifikation von Anforderungen

Zur Spezifikation wird in der gängigen Praxis sehr häufig beliebig gegliederter Prosatext eingesetzt. Die Gliederung kann durch die Anwendung von Standards und Vorgehensmethoden unterstützt werden (vgl. [DoT90]). Darüber hinaus wurde immer wieder versucht, formale textuelle Spezifikationssprachen (vgl. [BFG+93, Dub98]) zu definieren. Diese finden selten die gewünschte Akzeptanz bei Anwendern von Software und können daher lediglich zur Kommunikation zwischen für unterschiedliche Phasen der Softwareentwicklung zuständigen Entwicklern eingesetzt werden.

Neben der textuellen Spezifikation werden zunehmend graphische Beschreibungstechniken eingesetzt. Der Vorteil gegenüber einer rein textuellen Spezifikation liegt in erster Linie in der Übersichtlichkeit und in ihrer intuitiven Darstellung. In den letzten Jahren entstand hierfür die mittlerweile sehr populäre Unified Modeling Language (UML, [RJB99]). Dies ist eine Sammlung und Vereinheitlichung von Beschreibungstechniken unterschiedlicher Softwareentwicklungs-Methoden zur Darstellung verschiedener Aspekte in der Softwareentwicklung. Die UML hat sich mittlerweile zu einem de facto Standard für Beschreibungstechniken im Software Engineering in der industriellen Praxis durchgesetzt. Sie ist als offener Standard mit vielen spezifischen Dialekten zu verstehen, denen teils unterschiedliche Semantiken unterlegt sind.

Eine eindeutige Semantik innerhalb eines Anwendungsbereiches kann die Verständigung zwischen an verschiedenen Phasen beteiligten Entwicklern verbessern. Zur Kommunikation zwischen Kunden und Entwicklern können diese Beschreibungstechniken auch verwendet werden. Hierbei muss jedoch darauf geachtet werden, dass Kunden meist ein individuelles intuitives Verständnis einer Beschreibung haben, das nicht genau mit der en eindeutigen Semantik übereinstimmt. Trotzdem kann die Beschreibung mit Hilfe einer graphischen Beschreibungstechnik die Verständigung zwischen Kunden und Entwicklern verbessern. Der Einsatz der UML im Requirements Engineering wird sich daher in Zukunft noch mehr verstärken.

Es existieren unterschiedlichste Bestrebungen hinsichtlich der Entwicklung von Beschreibungstechniken für verschiedenste Anwendungsbereiche. Ungeachtet davon sind kaum Ansätze zur Beschreibung wesentlicher Aspekte des Requirements Engineerings komplexer Standardsoftware zu finden. Daher wird hierauf in Kapitel 5 gesondert eingegangen.

2.4 Verfolgbarkeit

Bestehende Ansätze zur Verfolgbarkeit lassen sich nach Abschnitt 1.2.5 in Ansätze zur horizontalen und zur vertikalen Verfolgbarkeit unterteilen. Das Requirements Management (vgl. [GoF94, FiE00]) als Teilgebiet des Requirements Engineerings befasst sich vorrangig mit der vertikalen Verfolgbarkeit. Die horizontale Verfolgbarkeit ist hingegen zentraler Inhalt des Konfigurations-Managements. In den folgenden Unterabschnitten werden diese beiden Richtungen erläutert.

2.4.1 Horizontale Verfolgbarkeit durch Konfigurations-Management.

Das Konfigurations-Management für Software (oder auch Software-Konfigurations-Management genannt) behandelt die systematische Identifikation, Organisation und Kontrolle von Änderungen eines Systems und seiner Zerlegungseinheiten (vgl. [Bab86, Rud94]). Seinen Ursprung hat das Software-Konfigurations-Management in der Verwaltung von Zerlegungseinheiten der Implementierung von Softwaresystemen, also deren Quellcodedateien.

Ziel war hierbei einerseits eine fehlerfreie Weiterentwicklung einzelner Zerlegungseinheiten durch eine Verwaltung unterschiedlicher Versionen der Quellcodedateien zu unterstützen. Bei Versionen im Sinne des Konfigurations-Managements wird zwischen Revisionen und Varianten unterschieden. Revisionen dienen der Weiterentwicklung der Software, beispielsweise der Fehlerbehebung. Daher ersetzen neue jeweils ältere Revisionen einer Zerlegungseinheit. Varianten dienen der verteilten Weiterentwicklung derselben Quelldateien, um vorrangig Entwicklungszeiten zu verkürzen. Hierbei muss das Konfigurations-Management Mechanismen zur Wiedervereinung verschiedener Varianten unterstützen.

Andererseits unterstützt das Konfigurations-Management eine fehlerfreie Komposition der Zerlegungseinheiten zu einem integrierten Softwaresystem. Aufgabe hierbei ist, festzulegen, welche Versionen verschiedener Zerlegungseinheiten unter welchen Bedingungen verknüpft werden können. Eine Verknüpfung bestimmter Versionen von Zerlegungseinheiten wird als *Konfiguration* des Softwaresystems bezeichnet.

Mittlerweile beschränkt sich das Software-Konfigurations-Management nicht mehr auf die Implementierung, sondern unterstützt das Änderungsmanagement des gesamten Softwarelebenszyklus. Somit werden sämtliche Dokumente vom Requirements Engineering bis hin zur Inbetriebnahme mit in das Konfigurations-Management einbezogen.

Bestehende Ansätze (vgl. [CoW97]) zur Unterstützung der Weiterentwicklung von Zerlegungseinheiten arbeiten auf dem Prinzip der Delta-Speicherung von Dokumenten. Durch diese Technik werden Unterschiede zwischen verschiedenen Versionen eines Dokuments toolunterstützt identifiziert. So lassen sich nachträglich Änderungen gut nachvollziehen. Klassische Werkzeuge hierfür sind SCCS und RCS (vgl. [Roc75, Tic85]). Statt der Delta-Speicherung können auch Änderungsprotokolle gespeichert werden, die jede einzelne Änderung an einem Dokument nachvollziehbar machen. Unterschiedliche Versionen können hier ausgehend von einer Basisversion generiert werden. Bisherige Konfigurations-Management-Werkzeuge unterstützen die Versionierung textbasierter Dokumente. Derzeit wird an Konzepten zur Delta-Speicherung graphischer Beschreibungstechniken, wie beispielsweise der UML, gearbeitet.

Um die fehlerfreie Komposition einzelner Zerlegungseinheiten zu unterstützen, schlagen Ansätze des Konfigurations-Managements die Formulierung von Kompositionsregeln vor. Diese beschreiben, unter welchen Bedingungen einzelne Zerlegungseinheiten zusammen funktionieren. Beispielsweise kann die Bedingung formulieren, dass eine Zerlegungseinheit A in Version 1 mit Zerlegungseinheit B in Version 1.1, Version 1.2 und Version 1.3 komponiert werden kann. Für Quellcodedateien können sich andere Bedingungen beispielsweise auch auf die Beschreibung von Flags für die Compilierung der Datei beziehen. Für Kompositionsregeln können neben informellen auch formale Beschreibungen zum Einsatz kommen [Wes96]. Vorteil ist dabei die Möglichkeit der werkzeugunterstützten Komposition der Zerlegungseinheiten.

2.4.2 Vertikale Verfolgbarkeit durch Requirements Management

Im Gegensatz zur horizontalen Verfolgbarkeit beschäftigt sich das Requirements Management mit der vertikalen Verfolgbarkeit von Anforderungen. Zur Unterstützung der Pre-Verfolgbarkeit wird in [Got95] ein umfangreicher Lösungsansatz behandelt. Verursacher einer Anforderung ist nicht immer genau eine Person. Daher wurde in dieser Arbeit ein Modell sogenannter Beitragsstrukturen erstellt. Diese dokumentieren den wechselseitigen Einfluss verschiedener Personen und ihre Rollen hinsichtlich der Entstehung einer Anforderung. Beteiligte Personen und Art der Entstehung einer Anforderung lassen sich dadurch leicht analysieren und nachvollziehen. Dieser Ansatz reicht zur Pre-Verfolgbarkeit komplexer Standardsoftware nicht aus. Insbesondere hinsichtlich der Erfassung von Anforderungen aus unterschiedlichen Quellen und Kanälen muss es ergänzt werden. Hierzu wird in Abschnitt 4.4.1 ein gesondertes Entwicklungsprodukt eingeführt.

Voraussetzung einer lückenlosen Post-Verfolgbarkeit von Anforderungen wäre eine eindeutige Modellvorstellung eines schrittweisen Übergangs von einer Anforderungsspezifikation zu einem fertigen System. Auf einem sehr eingeschränkten Bereich existieren hierfür Ansätze mit detaillierten Konstruktions- und Verfeinerungsregeln (vgl. zum Beispiel [SHB96, Rum96]). Vorteil eines derartigen Modells wäre die Möglichkeit einer automatisierbaren Protokollierung durchgeführter Entwicklungsschritte.

Derzeit existiert noch kein derartiges Modell. Daher verwendet man in der Praxis das einfache Prinzip, verschiedenartigste Artefakte der Entwicklung durch attributierte Assoziationen zueinander in Bezug zu setzen. Attribute (vgl. zum Beispiel [RUP90]) einer Assoziation kennzeichnen dabei die Art des Bezugs zweier Artefakte. So kann beispielsweise eine Assoziation den Phasenübergang einer Anforderung zu unterschiedlichen Teilen des Designs nachvollziehbar machen. Diese Methode ist sehr aufwendig, da jede Assoziation einzeln per Hand festgelegt werden muss. Derzeit existieren auf dem Markt verschiedene Requirements Management-Werkzeuge, die eine Verwaltung derartiger Assoziationen unterstützen (vgl. [Her98, JHK+97]). Es wird intensiv daran gearbeitet, andere Softwarewerkzeuge, wie beispielsweise UML-Editoren, mit derartigen Requirements Management-Werkzeugen zu integrieren.

Insgesamt befassen sich das Konfigurations-Management und als auch das Requirements Management mit ähnlichen Problemstellungen. Damit ist eine stärkere Integration vorhandener Ansätze dieser beiden Forschungsgebiete nutzbringend.

Das Konfigurations-Management befasst sich mit relativ groben Einheiten der Softwareentwicklung, wie Quelldateien, Modulen oder Spezifikationen. Die oben erwähnten Kompositionsregeln beziehen sich daher auf diese Einheiten. Für das Design und die Implementierung hängen diese jedoch vorrangig davon ab, was einzelne Zerlegungseinheiten inhaltlich leisten. Anders ausgedrückt, sind die Kompositionsregeln indirekt durch das gegenseitige Zusammenspiel von Anforderungen bestimmt, die einzelne Zerlegungseinheiten erfüllen. Damit kann das Requirements Management das Konfigurations-Management in der Festlegung adäquater Kompositionsregeln unterstützen. Umgekehrt muss im Rahmen des Requirements Managements für eine Post-Verfolgbarkeit von Anforderungen die Zuordnung von Anforderungen zu Zerlegungseinheiten des Konfigurations-Managements nachvollziehbar sein.

Wie die Integration von Konfigurations-Management und Requirements Management im Rahmen dieser Arbeit erreicht wird, erläutert vor allem Abschnitt 4.5. Die vertikale Verfolgbarkeit von Anforderungen wird darüber hinaus durch weitere Elemente im Modell der Entwicklungsprodukte und im Prozessmodell (vgl. Kapitel 7) unterstützt. Entsprechende Konzepte hierzu werden in Kapitel 3 erläutert.

Kapitel 3

Grundlegende Modellierungskonzepte

Eine zunehmende Komplexität der Entwicklungsprozessen erfordert eine klare Strukturierung ihrer Beschreibung. In diesem Kapitel werden unterschiedliche Strukturierungsprinzipien und Techniken zur Beschreibung von Prozessmodellen vorgestellt. Diese sollen einerseits der gegebenen Komplexität gerecht werden. Andererseits sollen die wichtigsten Elemente eines Entwicklungsprozesses schnell und einfach identifizierbar sein. Dies dient einer systematischen Softwareentwicklung.

Im ersten Abschnitt 3.1 wird eine sinnvolle Strukturierung des Ablaufs und beteiligter Elemente eines Entwicklungsprozesses vorgestellt. Der sich anschließende Abschnitt 3.2 widmet sich der Darstellung von Entwicklungsprodukten. Diese dienen vorrangig als Arbeitsmittel zur Erarbeitung und zur Dokumentation von Ergebnissen innerhalb des Entwicklungsprozesses.

3.1 Entwicklungsprozesse

Im kommenden Unterabschnitt 3.1.1 werden die Hauptbestandteile Rollen, Aktivitäten und Entwicklungsprodukte eines Entwicklungsprozesses beschrieben. Zur verbesserten Strukturierung von Entwicklungsaktivitäten wird im sich anschließenden Unterabschnitt 3.1.2 das Konzept des entscheidungsorientierten Entwicklungsprozesses vorgestellt. Schließlich wird in Unterabschnitt 3.1.3 ein Schema zur Beschreibung der Elemente von Entwicklungsprozessen erläutert.

3.1.1 Rollen, Aktivitäten, Entwicklungsprodukte

Die Prozessmodellierung dieser Arbeit basiert auf dem klassischen Prinzip der Einteilung in Rollen, Aktivitäten und Entwicklungsprodukte (vgl. zum Beispiel [DKW99]). Abbildung 4 zeigt Elemente eines Entwicklungsprozesses. Diese Aufteilung zielt auf eine klare Strukturierung eines Entwicklungsprozesses ab. Sie unterstützt beispielsweise eine effiziente Planung benötigter Ressourcen, eine adäquate Aufgabenteilung, die Verfolgbarkeit und Konsistenzprüfungen zwischen unterschiedlichen Entwicklungsprodukten.

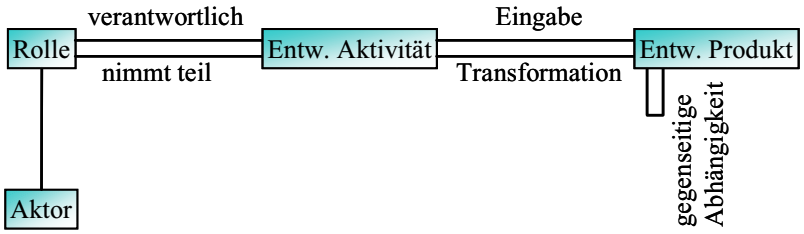


Abbildung 4 – Elemente eines Entwicklungsprozesses

Rolle

Durch Rollen werden Aufgaben und Verantwortlichkeiten der an der Entwicklung beteiligten Personen, sogenannten *Aktoren*, festgelegt. Die Aufgaben und Verantwortlichkeiten sind unmittelbar aus Unternehmenszielen abgeleitet. Jede Person nimmt jeweils vorgegebene Rollen ein. Dadurch konzentriert sich jede Person auf ein bestimmtes Spektrum von Aufgaben und Verantwortlichkeiten. Diese Spezialisierung führt sowohl zu einer erhöhten Arbeitseffizienz als auch zu einer verbesserten Verfolgbarkeit der Unternehmensziele. In Abschnitt 7.2 werden Rollen für das Requirements Engineering komplexer Standardsoftware eingeführt

Entwicklungsprodukt

Zentrale Arbeitsmittel und Ergebnisse der Entwicklung sind Entwicklungsprodukte. Das wichtigste Entwicklungsprodukt ist das Softwaresystem, das letztendlich am Markt verkauft wird. Für eine systematische Entwicklung ist darüber hinaus eine ganze Reihe anderer Entwicklungsprodukte erforderlich. Sie dokumentieren verschiedene Vorgaben und wesentliche Zwischenergebnisse im Laufe der Entwicklung. Derartige Entwicklungsprodukte sind zum Beispiel strategische Vorgaben, Marktstudien, Anforderungsdokumente, Softwareprototypen, Projektpläne usw. Je nach Zweck werden sie unterschiedlich dargestellt. Mögliche Darstellungsformen von Entwicklungsprodukten sind beispielsweise Dokumente, Programme oder Datenbanken.

Zwischen einzelnen Entwicklungsprodukten bestehen unterschiedliche *Abhängigkeiten*. Beispielsweise muss eine Anforderung, die in einer Anforderungsspezifikation zur Realisierung geplant ist, sich in den Designdokumenten in irgendeiner Form wiederfinden. Die Anforderungsspezifikation und Designdokumente sind dadurch voneinander abhängig. Zur Erhaltung der Konsistenz müssen Abhängigkeiten in der Entwicklung laufend berücksichtigt werden. Dies führt beispielsweise dazu, dass ein Design nur dann fertiggestellt werden kann, wenn auch die Anforderungsspezifikation abgeschlossen ist. Das gleiche gilt für etwaige nachträgliche Modifikationen. Wird beispielsweise eine neue Anforderung zur Anforderungsspezifikation hinzugenommen, muss folglich auch das Design aktualisiert werden.

Zur Modellierung von Entwicklungsprodukten wird in Abschnitt 3.2 dieser Arbeit ein eigenständiges Konzept vorgeschlagen. Am Requirements Engineering komplexer Standardsoftware benötigte Entwicklungsprodukte werden in Kapitel 4 eingehend behandelt.

Entwicklungsaktivität

Entwicklungsprodukte entstehen schrittweise als Ergebnis unterschiedlicher *Entwicklungsaktivitäten* oder kurz *Aktivitäten.* Jede Aktivität wird durch Vertreter unterschiedlicher Rollen durchgeführt. Für die Durchführung und das Ergebnis jeder Aktivität übernimmt jeweils eine Rolle die Verantwortung.

Zur Durchführung einer Aktivität sind verschiedene Entwicklungsprodukte als Eingabe erforderlich. Erst wenn ein geforderter Modellzustand (siehe Abschnitt 3.2.3) hinsichtlich dieser Entwicklungsprodukte vorliegt, kann eine Aktivität durchgeführt werden. Eine Aktivität gilt umgekehrt als beendet, sobald das Modell einen bestimmten Endzustand erreicht hat. Das Ergebnis selbst stellt dabei immer einen Kompromiss zwischen den Verantwortlichkeiten und Aufgaben der beteiligten Rollen dar.

Die Durchführung einer Aktivität ist lediglich durch die geforderten Produktzustände mit anderen Aktivitäten gekoppelt. Das heißt insbesondere, dass Aktivitäten nicht in einer starr vorgegebenen Reihenfolge ausgeführt werden müssen. Damit wird im Vergleich zu klassischen Prozessmodellen, wie beispielsweise dem Wasserfallmodell, der Forderung nach hoher Flexibilität Rechnung getragen.

3.1.2 Entscheidungsorientierter Entwicklungsprozess

Zur Modellierung von Entwicklungsprozessen wird das Konzept des *entscheidungsorientierten Entwicklungsprozesses* eingeführt. Dessen Grundidee ist, dass im Laufe der Softwareentwicklung vom Requirements Engineering bis zur Auslieferung (vgl. Glossar) mehrere Entscheidungssituationen durchlaufen werden müssen. Das Grundkonzept dieses Ansatzes ähnelt dem Argumentationsansatz IBIS [KuR70, RiW73], der prinzipiell aus den drei Bausteinen Problemstellung, Lösungsalternativen und Argumentation besteht. Stark vereinfacht definiert dies ein Vorgehen zur Problemlösung, in dem ausgehend von einer Problemstellung zunächst Lösungsalternativen gesucht werden. Diese werden in einem Argumentationsprozess gegeneinander verglichen und schließlich eine Alternative ausgewählt. In der vorliegenden Arbeit sind diese Ideen durch zusätzliche Elemente zur Konkretisierung des Vorgehens ergänzt. Entscheidungssituationen variieren in ihrem Umfang und in ihrer Auswirkung auf die nachfolgende Entwicklung.

Beispiel:

> *Eine sehr umfangreiche Entscheidungssituation mit großer Auswirkung auf die künftige Entwicklung ist die Ableitung einer adäquaten Softwarearchitektur aus gegebenen Anforderungen.*

Entscheidungssituationen bauen sich prinzipiell aus folgenden Elemente auf:

Elemente einer Entscheidungssituation

- *Entscheidungsproblem*: Problemstellung, über die entschieden werden soll. Im obigen Beispiel ist die Problemstellung die Ableitung einer Softwarearchitektur.

- *Entscheidungsobjekt*: Einflussgröße einer Entscheidung, die direkt im Entscheidungsergebnis einfließt. An einer Entscheidungssituation können mehrere Entsche i-

dungsobjekte beteiligt sein. Sie werden immer durch Entwicklungsprodukte repr ä-sentiert. Im obigen Beispiel sind unter anderem Anforder ungen Entscheidungsobjekte.

- *Alternativen*: Für ein gegebenes Entscheidungsproblem werden verschiedenartige Lösungen gesucht. Das Finden von Alternativen ist ein Hauptbestandteil jeder En t-scheidungssituation. Im Beispiel werden Architekturalternativen konstr uiert. Alternativen werden ebenfalls durch Entwicklungsprodukte repräsentiert.

- *Bewertungskriterium*: Zur Auswahl der richtigen Alternative, müssen Alternativen hinsichtlich der Erfüllung verschiedener Kriterien bewertet werden. Im Beispiel sind Anforderungen wie Effizienz oder Anpassbarkeit Bewe rtungskriterien. Methoden zur Bewertung von Alternativen werden in Kapitel 6 ausführlich vorgestellt.

- *Zielfunktion:* Die Zielfunktion legt zu optimierende Größen einer Entscheidungssit u-ation fest. Diese Größen äußern sich in der Maximierung oder Minimierung ve r-schiedener Bewertungskriterien. In einzelnen Alternativen ist die Kriterienbewertung unterschiedlich ausgeprägt. Verschiedene Bewertungskriterien müssen en tsprechend der Zielfunktion in einer Entscheidung adäquat abgewogen werden, um Alternativen miteinander vergleichbar zu machen. Im Beispiel wäre die Zielfunktion die gleichze i-tige Maximierung von Effizienz und Anpassbarkeit.

- *Entscheidungsergebnis*: Das Entscheidungsergebnis repräsentie rt die letztendlich ausgewählte Alternative einer Entscheidungssituation. Das Entscheidungse rgebnis stellt hinsichtlich der Zielfunktion die beste Alternative dar. Daher enthält das En t-scheidungsergebnis die selben Entwicklungsprodukttypen, wie Alternative n (siehe hierzu auch Abschnitt 3.2.5). Das Entscheidungsergebnis im Beispiel ist die tatsäc h-lich abgeleitete Softwarearchite ktur.

- *Anleitungen, Vorlagen*: Meist können für einzelne Entscheidungssituationen sehr viele Alternativen gefunden werden. Deshalb ist häufig aufgrund von Zeit und Ko s-tenrestriktionen das Finden einer absolut optimalen Lösung nicht möglich. Jedoch kann die Erarbeitung von Alternativen durch heuristische *Anleitungen*, *Vorlagen* usw. unterstützt werden. Anleitung en sind im obigen Beispiel Architektu rstile (vgl. [SG96, BMR+96]), die eine Konstruktion von Architekturalternativen unte rstützen. Diese basieren häufig auf Prinzipien, die in der Vergangenheit zu guten Entsche i-dungsalternativen geführt haben und deshalb die Ableitung neuer Alternativen ve r-einfachen.

Abbildung 5 zeigt einen Überblick über wesentliche Zusammenhänge zwischen Eleme n-ten einer Entscheidungssitu ation.

In jeder Entscheidungssituation nehmen Entwicklungsprodukte die Rolle unterschiedl i-cher Entscheidungselemente ein. Dabei ist auch durchaus möglich, dass ein Entwic k-lungsprodukt in unterschiedlichen En tscheidungssituationen unterschiedliche Rollen besetzt.

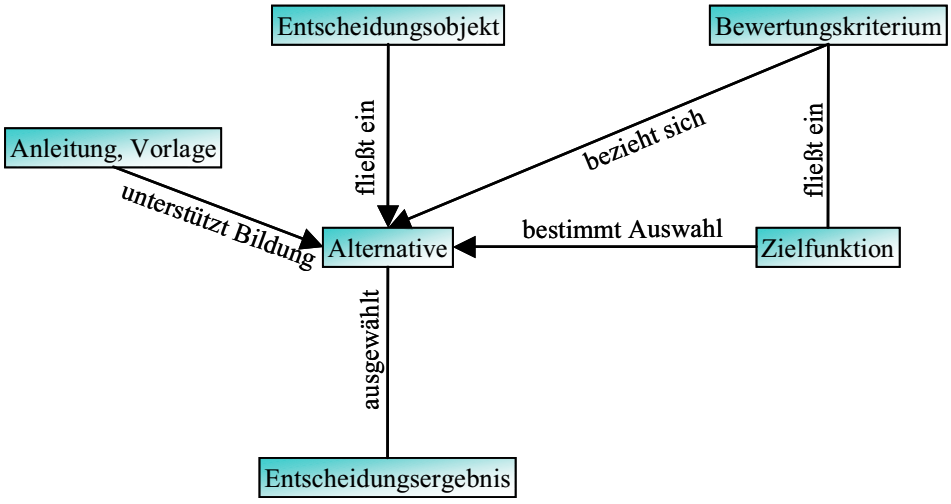


Abbildung 5 – Elemente einer Entscheidungssituation

Durch diese Art der Modellierung wird verglichen zu herkömmlichen Entwicklungsprozessmodellen eine weitaus erhöhte Verfolgbarkeit erreicht:

- Der Bezug einzelner Entwicklungsprodukte zu einer Aktivität ist klar definiert. Damit ist auch verfolgbar, wie Entscheidungsobjekte in Entscheidungsergebnisse eingeflossen sind.

- Durch die Dokumentation verwendeter Anleitungen und Vorlagen ist die Entstehung von Alternativen verfolgbar. So kann beispielsweise bei der Ableitung einer Architektur klar dokumentiert werden, welche Architekturstile zu einer Alternative geführt haben.

- Entwickler werden gezwungen, bei der Auswahl von Entscheidungsergebnissen klare Kriterien festzulegen.

- Die Gründe der Auswahl einer und der Ablehnung anderer Alternativen wird durch die Abwägung der Alternativen hinsichtlich der Zielfunktion nachvollziehbar.

Entwicklungsablauf

Im Ablauf eines entscheidungsorientierten Entwicklungsprozesses werden für jede Entscheidungssituation sukzessive die Inhalte zugeordneter Entwicklungsprodukte erarbeitet. Sobald das Modell der Entwicklungsprodukte einen für die Entscheidungssituation geforderten Zustand erreicht, kann die Entscheidung erfolgen. Für Entscheidungssituationen im Requirements Engineering komplexer Standardsoftware geforderte Zustände werden bei der Beschreibung einzelner Entscheidungssituationen in Kapitel 5 beschrieben.

Die Erarbeitung der Inhalte der Entwicklungsprodukte erfolgt im Rahmen der in Abschnitt 3.1.1 eingeführten Aktivitäten. Es wird hierbei zwischen Vorarbeiten und Entscheidungen unterschieden:

- *Vorarbeiten:* Diese dienen zur Erarbeitung von Entwicklungsprodukten, denen keine explizite Entscheidungssituation vorausgeht.

- *Entscheidungen:* Die Struktur von Entscheidungen wird hingegen durch eine Entscheidungssituation eindeutig vorgegeben.

Diese Art der Modellierung erlaubt eine hohe Flexibilität des Entwicklungsprozesses. Die zeitliche Abfolge der Erstellung einzelner Elemente einer Entscheidungssituation hängt nur von inhaltlichen Abhängigkeiten zwischen den Entscheidungselementen ab. Diese Abhängigkeiten werden ausführlich in Kapitel 4 im Rahmen des Modells der Entwicklungsprodukte erläutert. Weitere Einschränkungen in der möglichen zeitlichen Reihenfolge der Aktivitäten ergeben sich in der Praxis auch noch durch das verfügbare qualifizierte Personal und anderer Ressourcen. Ressourcenbeschränkungen und zugehörige Planungsprobleme werden jedoch im Rahmen dieser Arbeit nicht weiter betrachtet.

Die Entwicklung kann darüber hinaus durch die Möglichkeit temporärer Inkonsistenzen zwischen Entwicklungsprodukten weiter flexibilisiert werden. Klar definierte Beziehungen zwischen Entwicklungsprodukten erlauben deren nachträgliche Erkennung und Korrektur. Dadurch können Aktivitäten stark verzahnt werden. Einen Ansatz für derartige Entwicklungsabläufe schlägt beispielsweise [NKF94] vor.

In Kapitel 5 wird eine Reihe von Aktivitäten für das Requirements Engineering komplexer Standardsoftware vorgeschlagen. Diese geben den Entwicklungsablauf, mit Ausnahme des zeitlichen Ablaufs, relativ starr vor. Insbesondere ist vorgesehen, dass jede Aktivität auch durchgeführt wird. Es kann jedoch in manchen Fällen nötig sein, noch flexibler auf die konkrete Projektsituation reagieren zu können. Hier muss die Möglichkeit bestehen, aus mehreren möglichen Aktivitäten eine der Situation angemessene Aktivität zu wählen.

Die Aktivitätsauswahl kann auch anhand von Entscheidungssituationen modelliert werden. Wesentliche Einflussgröße ist hier der Projektkontext. Je nach Modell erfolgt eine Auswahl aus vorgegebenen Katalogen oder auch es wird die Aktivität erst in der konkreten Situation eigens entwickelt. Das V-Modell [VMod] bietet beispielsweise eine Reihe vorgefertigter Aktivitäten an. Durch das „Tailoring“ wird hiervon vorrangig vor dem Projektstart eine Auswahl vorgenommen. Ein anderer Prozessmodellierungsansatz (vgl. [Rol94]) ist vom Ansatz her noch flexibler. Er sieht explizit die Auswahl oder die Konstruktion von Aktivitäten während des Projektes jeweils im Anschluss an vorher durchgeführte Aktivitäten vor.

3.1.3 Beschreibung von Entwicklungsaktivitäten

Dieser Abschnitt legt Schemata zur Beschreibung der in Abschnitt 3.1.2 vorgestellten Aktivitätstypen Entscheidung und Vorarbeit fest. In Abschnitt 3.1.1 eingeführte Rollen werden durch informellen Text beschrieben (vgl. Abschnitt 7.2). Schemata zur Beschreibung von Entwicklungsprodukten werden erst nach einer detaillierten Erörterung des Modellierungskonzeptes in Abschnitt 3.2 vorgestellt. Dazu wird zunächst ein Schema zur Beschreibung von Entscheidungen erläutert. Anschließend erfolgt die Vorstellung eines Schemas für Vorarbeiten.

Entscheidungen

Zur Beschreibung einer Entscheidung müssen beteiligte Rollen, das Entscheidungsproblem, Entscheidungsobjekte mit Beschreibung des geforderten Zustandes (vgl. Abschnitt 3.2.3), Entscheidungskriterien, Zielfunktion, Entscheidungsergebnis, zugehörige Anleitungen und Vorgaben sowie die konkrete Vorgehensweise beschrieben werden.

Folgendes Schema dient zur systematischen Beschreibung einer Entscheidung. Für die folgenden Zeilenbeschriftungen gelten folgende Abkürzungen: R=Rollen, O=Entscheidungsobjekte, K=Bewertungskriterium, E=Entscheidungsergebnis und A=Anleitungen/Vorlagen (Abkürzungen dienen der Platzersparnis bei der Beschriftung):

Entscheidung Name	
R	Aufzählung beteiligter Rollen. Der Hauptverantwortliche der Aktivität wird mit **fetter Schrift** gekennzeichnet.
O	Aufzählung von Entwicklungsprodukten, die in der Entscheidungssituation Entscheidungsobjekte sind. In Klammern wird jeweils der geforderte Zustand eines Entwicklungsproduktes angegeben. Zu jedem Entwicklungsprodukt erfolgt eine kurze Beschreibung seiner Aufgabe innerhalb der Entscheidungssituation.
K	Gegebenenfalls Aufzählung von Entwicklungsprodukten, die als Entscheidungskriterium herangezogen werden. Darüber hinaus allgemeingültige Kriterien, die unmittelbar aus den eingangs genannten Zielen (vgl. Abschnitt 1.1.1) abgeleitet sind.
E	Beschreibung des durch die Entscheidungssituation beeinflussten Teils des Modells der Entwicklungsprodukte.
A	Aufzählung möglicher methodischer Unterstützungen zur Herleitung von Alternativen.
R=Rollen, O=Entscheidungsobjekte, K=Bewertungskriterium, E=Entscheidungsergebnis und A=Anleitungen/Vorlagen	

Entscheidungsproblem: Informelle Beschreibung des Entscheidungsproblems der gegebenen Entscheidungssituation.

Zielfunktion: Informelle Beschreibung der Zielfunktion der gegebenen Entscheidungssituation.

Restriktionen: Unter Umständen müssen beim Vorgehen gewisse Restriktionen berücksichtigt werden. Diese beziehen sich auf Teile des Modells der Entwicklungsprodukte. Diese werden hier beschrieben.

Vorgehen: Informelle Beschreibung des Vorgehens bei der Durchführung der Aktivität.

Vorarbeiten

Folgendes Schema dient zur systematischen Beschreibung einer Vorarbeit. Für die folgenden Zeilenbeschriftungen gelten folgende Abkürzungen: S=Entscheidungssituation, R=Rollen, E=Vorarbeitsergebnis und A=Anleitungen/Vorlagen (Abkürzungen dienen der Platzersparnis bei der Beschriftung):

Vorarbeit Name	
S	Beschreibung für welche Entscheidungssituation diese Vorarbeit durchgeführt wird.
R	Aufzählung beteiligter Rollen. Der Hauptverantwortliche der Aktivität wird mit **fetter Schrift** gekennzeichnet.
O	Aufzählung von Entwicklungsprodukten, die in die beschriebene Vorarbeit einfließen. Diese Entwicklungsprodukte stellen in der assoziierten Entscheidung Entscheidungsobjekte dar. In Klammern wird der geforderte Zustand des Entwicklungsproduktes angegeben.
E	Beschreibung des durch die Vorarbeit beeinflussten Teils des Modells der Entwicklungsprodukte.
A	Aufzählung methodischer Unterstützungen zur Durchführung der Aktivität.
S=Entscheidungssituation, R=Rollen, O=Vorarbeitsobjekte, E=Vorarbeitsergebnis und A=Anleitungen/Vorlagen	

Motivation: Beschreibung, warum diese Vorarbeit unabhängig von zugehörigen Entscheidungssituationen durchgeführt wird.

Vorgehen: Informelle Beschreibung des Vorgehens bei der Durchführung der Aktivität.

3.2 Entwicklungsprodukte

In diesem Abschnitt wird das Konzept zur Modellierung von Entwicklungsprodukten vorgestellt. Dieses basiert auf [DSV99]. Wie im vorangegangenen Abschnitt festgestellt, sind Entwicklungsprodukte mit ihren wechselseitigen Beziehungen ein wesentlicher Bestandteil von Entwicklungsprozessen. Durch eine klare Definition dieser Beziehungen können Zusammenhänge zwischen verschiedenen Entwicklungsprodukten leichter verfolgt werden. Eine erhöhte Verfolgbarkeit steigert die Änderungsflexibilität der gesamten Entwicklung, da Auswirkungen lokaler Änderungen auf andere Entwicklungsprodukte leicht nachvollziehbar werden. Fokus der vorliegenden Arbeit sind wesentliche Entwicklungsprodukte des Requirements Engineerings komplexer Standardsoftware. Hierzu wird in Kapitel 4 ein *Modell von Entwicklungsprodukten* definiert. Dieses beschreibt Typen von Entwicklungsprodukten und von Beziehungstypen zwischen diesen.

3.2.1 Produkttypen

Zur Strukturierung und Repräsentation der in Entwicklungsprodukten enthaltenen Informationen wird jedes Produkt durch *Attribute* beschrieben. Attribute selbst müssen nicht getypt sein, sondern können auch rein informelle Inhalte enthalten. Jeder Produkttyp besitzt einen eindeutigen Namen.

Produkttypen können aus anderen Produkttypen aggregiert sein. Mit Hilfe der *Aggregation* wird das Prinzip der Gruppierung und der hierarchischen Strukturierung auf Entwicklungsprodukte angewendet. Ein aggregierendes Produkt (*Container*) kann als Sammelbegriff für eine Gruppe aggregierter Produkte (*Teile*) verstanden werden. Hauptaugenmerk der Produktmodellierung ist die Darstellung von Produktinhalten und deren Abhängigkeiten als Basis für eine flexible entscheidungsorientierte Prozessmodellierung. Abhängigkeiten der Lebensdauer, Sichtbarkeitsbeschränkungen oder Exklusivität von Teilen anderer Ansätze bleiben daher außer Betracht.

Um Ähnlichkeiten zwischen unterschiedlichen Produkttypen ausdrücken zu kön nen, wird das Prinzip der hierarchischen *Generalisierung und Spezialisierung* genutzt. Dies wird im Sinne einer „ist-Art-von“ Beziehung verwendet. Das bedeutet, dass jede Spezialisi erung eines Produkttyps sowohl die Attribute seines assoziierten „generische ren“ Produkttyps als auch dessen Rollen innerhalb von Bezi ehungen „erbt“.

Mit Hilfe der Generalisierungsbeziehung zwischen Produkttypen können auch generische Entwicklungsprodukte definiert werden. Diese können als Schnittstelle zwischen unte rschiedlichen Sichten auf die Entwicklung dienen, indem sie jeweils unterschiedlich we iter spezialisiert werden können. So kann sich die Projektplanung beispielsweise auf generische Architekturelemente zu beziehen. Damit muss die Projektplanung nicht auf unnötige Spezifika dieser Elemente eingehen. Aus Managementsicht ist beispielsweise nicht relevant, ob in einem Arbeitspaket eines Projektplans ein GUI -Design oder eine andere Funktionalität realisiert wird. In [DSV99] wurden so die drei Perspektiven des Requirements Engineerings, des Designs und des Projektmanagements zueinander in Bezug gesetzt.

3.2.2 Produktbeziehungen

Zwischen Entwicklungsprodukten bestehen unterschiedlichste Beziehungen, die durch Assoziationen ausgedrückt werden.

Eine *Assoziation* im Produktmodell hat die gleiche Bedeutung wie eine Bezi ehung in der E/R-Modellierung (vgl. [Che76]). Assoziationen können auch durch Attr ibute genauer spezifiziert werden. Der Einfachheit halber sind Assoziationen im Modell dieser Arbeit ausschließlich binär. Um an einer Assoziat ion beteiligte Produkte noch stärker zu chara kterisieren, können ihnen eigene Rollen bezüglich einer Assozi ation zugeordnet werden. Assoziationen können gerichtet oder ungerichtet sein. Bei gerichteten Assoziat ionen wird ein Produkt als *Quelle* bezeichnet, während das andere die *Senke* der Assoziation darstellt.

3.2.3 Zustände des Modells der Entwicklungsprodukte

In Abschnitt 3.1.3 wurde erläutert, dass die Durchführung von Aktivitäten vom Zustand des Modells der Entwicklungsprodukte ab hängig ist. Das Modell hat je nach Fortschritt des Requirements Engineerings unterschiedliche Zustände. Diese äußern sich in der D efinition neuer Entwicklungsprodukte, in deren Zuständen und in dem Assoziationsz ustand.

Der *Produktzustand* eines Entwicklungsproduktes A wird ausgedrückt durch:

- aktuelle Attributbelegungen
- zum Entwicklungsprodukt A aggregierte Entwicklungsprodukte
- Entwicklungsprodukte, zu denen das Entwicklungsprodukt A aggregiert ist.

Attributbelegungen lassen sich abhängig davon, ob es sich um getypte oder ungetypte Attribute handelt, absolut oder qualitativ ausdrücken:

- *Absolute Attributbelegung:* Eine absolute Attributbelegung für ein boolesches Attribut *Antwort* würde beispielsweise mit *Antwort = true* ausgedrückt werden.
- *Qualitative Attributbelegung:* Eine qualitative Attributbelegung beschreibt grob, welche Eigenschaft eine Attributbelegung haben muss, ohne diese absolut festzulegen. Beispielsweise könnte für ein Attribut *Langfristige Vision* die qualitative Attributbelegung „*grob definiert*“ heißen. Die derartige Beschreibung von Belegungen ist methodisch motiviert. Beispielsweise wird zeitweise eine Weiterverwendung grober Zwischenergebnisse vor der endgültigen Belegung aller Attribute angestrebt, um Parallelarbeit zu ermöglichen.

Mit dem *Assoziationszustand* wird hingegen ausgedrückt, für welche Assoziationstypen Assoziationen zwischen entsprechenden Entwicklungsprodukten bereits festgelegt worden sind.

Aufgrund möglicher Änderungen des Zustands des Modells für Entwicklungsprodukte müssen sämtliche Produkte, Aggregationen und Assoziationen einem Konfigurationsmanagement unterstellt werden, um Inhaltsänderungen adäquat verfolgbar zu machen.

3.2.4 Beschreibung von Entwicklungsprodukten

Zur Beschreibung von Instanzen des Produktmodells werden unterschiedliche Hilfsmittel eingesetzt:

- Eine graphische Repräsentation durch UML-Klassendiagramme
- Ein tabellarisches Schema und
- Formale Notationen

Graphische Repräsentation

Graphische Repräsentationen verschaffen einen Überblick über Produkttypen und deren gegenseitigen Zusammenhänge. Die Darstellung ist an die UML-Notation und an die graphische Notation aus [DSV99] angelehnt.

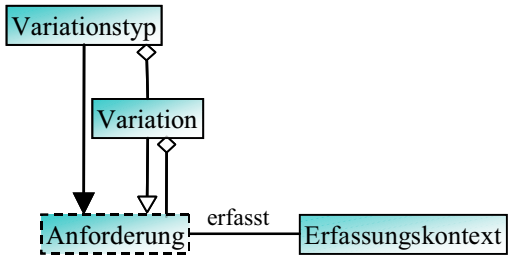


Abbildung 6 – Beispiel einer graphischen Repräsentation des Modells der Entwicklungsprodukte

Abbildung 6 enthält einen kleinen Ausschnitt der graphischen Repräsentation des in Kapitel 4 vorgestellten Produktmodells des Requirements Engineerings komplexer Standardsoftware. Die folgende Tabelle erläutert die einzelnen graphischen Elemente:

Repräsentation	Bedeutung	Beispiel
Kästchen	Produkttyp	Variationstyp
Ausgefüllter Pfeil	Gerichtete Assoziation	Variationstyp ist mit Anforderung assoziiert. Der Variationstyp entspricht der Quelle, die Anforderung der Senke
Einfache Kante	Ungerichtete Assoziation	Anforderung ist mit Erfassungskontext assoziiert. Assoziationen können zur Kennzeichnung des Typs beschriftet werden (hier *erfasst*).
Raute mit Kante	Aggregation	Variationstyp aggregiert Variationen und Variation aggregiert Anforderung
Unausgefülltes Dreieck mit Kante	Generalisierung und Spezialisierung	Variation spezialisiert Anforderung
Gestrichelte Linie	Referenz auf ein Element in einer anderen Abbildung. Das Element ist in einer anderen Abbildung mit weiteren Kanten verbunden	Anforderung

Tabellarisches Schema

Mit Hilfe des tabellarischen Schemas werden Produkttypen und Assoziationen übersichtlich beschrieben. Die folgenden zwei Tabellen stellen zwei Schemata vor und beschreiben deren Inhalte. Die Abkürzungen der einzelnen Zeilen bedeuten: B=Beschreibung, Z=Zweck, A=Attribute, G=Aggregationen (Abkürzungen dienen der Platzersparnis bei der Beschriftung).

Produkttyp Der Name des Produkttypen	
B	Erläuterung der in Instanzen dieses Produkttypen enthaltenen Informationen
Z	Erläuterung der methodischen Rolle und Bedeutung des Produkttypen
A	Erläuterung der „elementaren“ Produkteigenschaften
G	Erläuterung der Aggregationen in denen dieser Produkttyp die aggregierende Rolle spielt
B=Beschreibung, Z=Zweck, A=Attribute, G=Aggregationen	

Schema für Produkttypen inklusive Aggregation und Abhängigkeiten

Assoziationstyp Der Name des Assoziationstypen	
B	Erläuterung der in Assoziation dieses Typen enthaltenen Informationen
Z	Erläuterung der methodischen Rolle und Bedeutung des Assoziationstypen
A	Erläuterung der „elementaren“ Eigenschaften einer Assoziation
B=Beschreibung, Z=Zweck, A=Attribute	

Schema für Assoziationstypen

Formale Notation

Aufgrund der großen Anzahl unterschiedlicher Einflussfaktoren existieren vielfältige Entwicklungsprodukte beim Requirements Engineering komplexer Standardsoftware. Die Entwicklungsprodukte weisen komplexe Zusammenhänge auf. Ein Teil dieser Zusammenhänge kann mit Hilfe einfacher formaler Mittel beschrieben werden und somit durch ein adäquates Softwarewerkzeug automatisch verfolgt werden. Dazu wird eine Reihe auf Prädikatenlogik basierender Definitionen und Axiome verwendet.

Das Modell für Entwicklungsprodukte kann durch seine präzise Darstellung unmittelbar als zentraler Teil des Datenmodells eines Softwarewerkzeugs zur Unterstützung einzelner Entwicklungsaktivitäten verwendet werden. Die tabellarischen Schemata dienen dabei zur Identifikation zu erfassender Attribute von Entwicklungsprodukten und Assoziationen. Die formal beschriebenen Zusammenhänge können zur automatischen Überprüfung oder Wiederherstellung bestimmter Konsistenzbedingungen innerhalb und zwischen den Entwicklungsprodukten eingesetzt werden. Für das in Kapitel 4 eingeführte Modell für Entwicklungsprodukte ergeben sich aufgrund der formalen Beschreibung eine Reihe von Möglichkeiten einer Werkzeugunterstützung. Diese werden gesondert in Abschnitt 4.5 vorgestellt. In diesem Abschnitt wird auch die durch das formale Modell erreichte Integration des Requirements Managements und des Konfigurations-Managements erläutert (vgl. auch Abschnitt 2.4).

Die formale Notation für das Modell der Entwicklungsprodukte besteht aus folgenden Elementen:

- *Definitionen*: Hiermit werden neue Begriffe und formale Notationen festgelegt.
- *Entwicklungsaxiome*: Beschreibung von Konsistenzbedingungen zwischen einzelnen Entwicklungsprodukten, die im Laufe des Requirements Engineerings und der späteren Entwicklungsphasen zu berücksichtigen sind. Diese beeinflussen beziehungsweise limitieren vorrangig die Durchführung einzelner Entwicklungsaktivitäten. Beispielsweise kann mit einem Entwicklungsaxiom gefordert werden, dass alle für eine bestimmte Version zur Realisierung geplanten Anforderungen auch tatsächlich realisiert werden.
- *Laufzeitaxiome*: Beschreibung von Konsistenzbedingungen, die sich unmittelbar auf das realisierte Softwaresystem auswirken. Diese müssen also bei der Entwicklung und Implementierung der Software adäquat berücksichtigt werden.
- *Hilfsdefinitionen*: Festlegung von Begriffen und Notationen, die nur lokal zur Beschreibung besonders komplexer Axiome benötigt werden.

Zusammenhang zwischen den Notationen

Die formalen Notationen sind mit den bei den anderen Darstellungen des Produktmodells konsistent. Assoziationen werden durch Prädikate dargestellt, Entwicklungsprodukte und Aggregationen durch Mengen. Die graphische Notation dient lediglich zur übersichtlichen Darstellung des Modells, lässt sich jedoch aus der tabellarischen Beschreibung erzeugen. Demgegenüber enthalten sowohl die tabellarische als auch die formale Notation eigene wesentliche Informationen zur vollständigen Beschreibung des Modells und sind daher unabdingbare Bestandteile.

3.2.5 Entwicklungsprodukte und entscheidungsorientierter Entwicklungsprozess

Wie bereits bei der Einführung des entscheidungsorientierten Entwicklungsprozesses beschrieben wurde, nehmen die Entwicklungsprodukte in verschiedenen Entscheidungssituationen unterschiedliche Rollen ein. Prinzipiell können einzelne Elemente einer Entscheidungssituation durch unterschiedliche für die Rolle adäquate Produkttypen des Modells der Entwicklungsprodukte besetzt sein. Eine Ausnahme sind hier die Alternative und das Entscheidungsergebnis. Diese enthalten die gleichen inhaltlichen Informationen und bestehen daher innerhalb einer Entscheidungssituation jeweils aus den selben Produkttypen. Bei der Alternativenbildung kann zwischen *flachen Alternativen* und *hierarchischen Alternativen* unterschieden werden:

Flache Alternativen: Im einfachsten Fall ist jede Alternative eine gesonderte Instanz eines gegebenen Produkttypen. Daraus wird genau eine ausgewählt, die dann das Entscheidungsergebnis darstellt. In diesem Fall werden die einzelnen Alternativen als *flache Alternativen* bezeichnet, da nur eine einzige Entscheidung notwendig ist.

Hierarchische Alternativen: Demgegenüber stehen *hierarchische Alternativen.* Diese werden verwendet, wenn ein Entscheidungsproblem hierarchisch in Teilprobleme zerlegt wird und für diese dann jeweils unterschiedliche Alternativen gesucht werden. In diesem Fall ist dann das Entscheidungsergebnis eine Komposition der Auswahl jeweils einer Lösungsalternative für jedes Teilproblem.

Beispiel:

> *Beim Systemdesign werden die Entscheidungsprobleme anhand der Anforderungen in unterschiedliche Teile aufgeteilt, anstatt Alternativen für das gesamte Design auf einmal zu konstruieren. Für jeden Teil der Anforderungen werden Designalternativen gesucht und anhand der gestellten Anforderungen lokal Entscheidungen getroffen. Die letztendliche Architektur komponiert sich dann aus den verschiedenen Teillösungen.*

Die Dekomposition des Entscheidungsproblems muss sich nicht notwendigerweise in der Dekomposition der Struktur des Softwaresystems widerspiegeln. Beispielsweise können Alternativen einer Software durchaus auch komplett unterschiedliche Dekompositionen in Komponenten, Verbindungen und Kommunikationsprotokollen enthalten.

Zur Unterstützung der Dekomposition eines Entscheidungsproblems in Teilprobleme kann das Prinzip der *Zerlegung nach Betrachtungsfokus* (engl. *Separation of Concerns*) (vgl. zum Beispiel [Dij76, TOH+99]) angewendet werden. Ein Betrachtungsfokus ist immer eine bestimmte Perspektive von der aus ein gewisser Betrachtungsgegenstand beleuchtet wird.

Beispiel:

> *Betrachtungsfokusse können zum Beispiel technische Realisierungsaspekte hinsichtlich der Softwareentwicklung sein. Andere Betrachtungsfokusse sind Entwicklungsphasen, Projektmanagement usw.*

Die Zerlegung nach Betrachtungsfokus ist das allgemeine Prinzip, dass Problemb ereiche hinsichtlich unterschiedlicher Betrachtungsfokusse analysiert und beschrieben werd en. Auf dem gleichen Prinzip basiert eine in Abschnitt 4.2.2 vorgeschlagene Strukturi e-rungsmöglichkeit von Anford erungen.

Zur Darstellung hierarchischer Alternativen werden zusätzlich für jede Entscheidungss i-tuation die UND- und die ODER- Aggregation eingeführt. Abbildung 7 zeigt die Erweiterung anhand der oben eingeführten graphischen Darste llung. In einer Instanz einer hierarchischen Alternative schließen sich UND - und ODER-Aggregationen gegenseitig aus, also entweder sind alle aggregierten Altern ativen nur UND-Aggregationen oder nur ODER-Aggregationen.

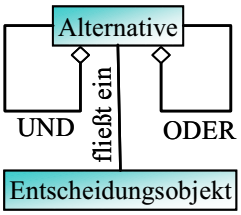


Abbildung 7 – Hierarchische Alternativen

- UND-Aggregation: Die gegebene Alternative wird dekomponiert in Teilprobl eme. Diese sind hier durch mit UND verknüpften Teile repräsentiert. Die mit der Alte rnative verbundenen Entscheidungsobjekte we rden auf die verknüpften Teile verteilt.
- ODER-Aggregation: Jede aggregierte Alternative repräsentiert eine mögliche Lösung der mit der gegebenen Alternative verknüpften Entscheidungsobjekte. Jede aggr e-gierte Alternative „erbt“ alle verknüpften Entscheidungsobjekte.

Bildhaft gesehen spannt eine Instanz einer hierarchischen Alternative einen Baum auf, der an jedem ODER-Knoten eine Entscheidung verlangt. Das letztendliche Entsche i-dungsergebnis entspricht dann einer Ko mposition der Alternat iven an den UND-Knoten und der eindeutigen Entscheidung an den ODER -Knoten. Bei UND-Knoten ist die Wahl aller Söhne zwingend.

Der Einsatz hierarchischer Alternativen eignet sich für sehr komplexe En tscheidungen. Ein Beispiel ist die Ableitung eines kompletten Designs. Pri nzipiell lässt sich diese Technik bei allen Entscheidungssituationen einsetzen. Sinnvoll ist sie alle rdings nur, wenn wirklich eine Zerlegung in Teilpro bleme möglich ist.

In Kapitel 4 werden einzelne Entwicklungsprodukte vorgestellt, ohne noch einmal spez i-fisch auf die hierarchische Alternativenbildung einzugehen. Diese kann zusätzlich auf jedes Produkt als zwei weitere Aggregationstypen aufgeno mmen werden. Wesentlich ist jedoch dann, dass die Bestandteile in einem Produkt sich auch adäquat in Teile zerl egen lassen.

Kapitel 4

Modell der Entwicklungsprodukte

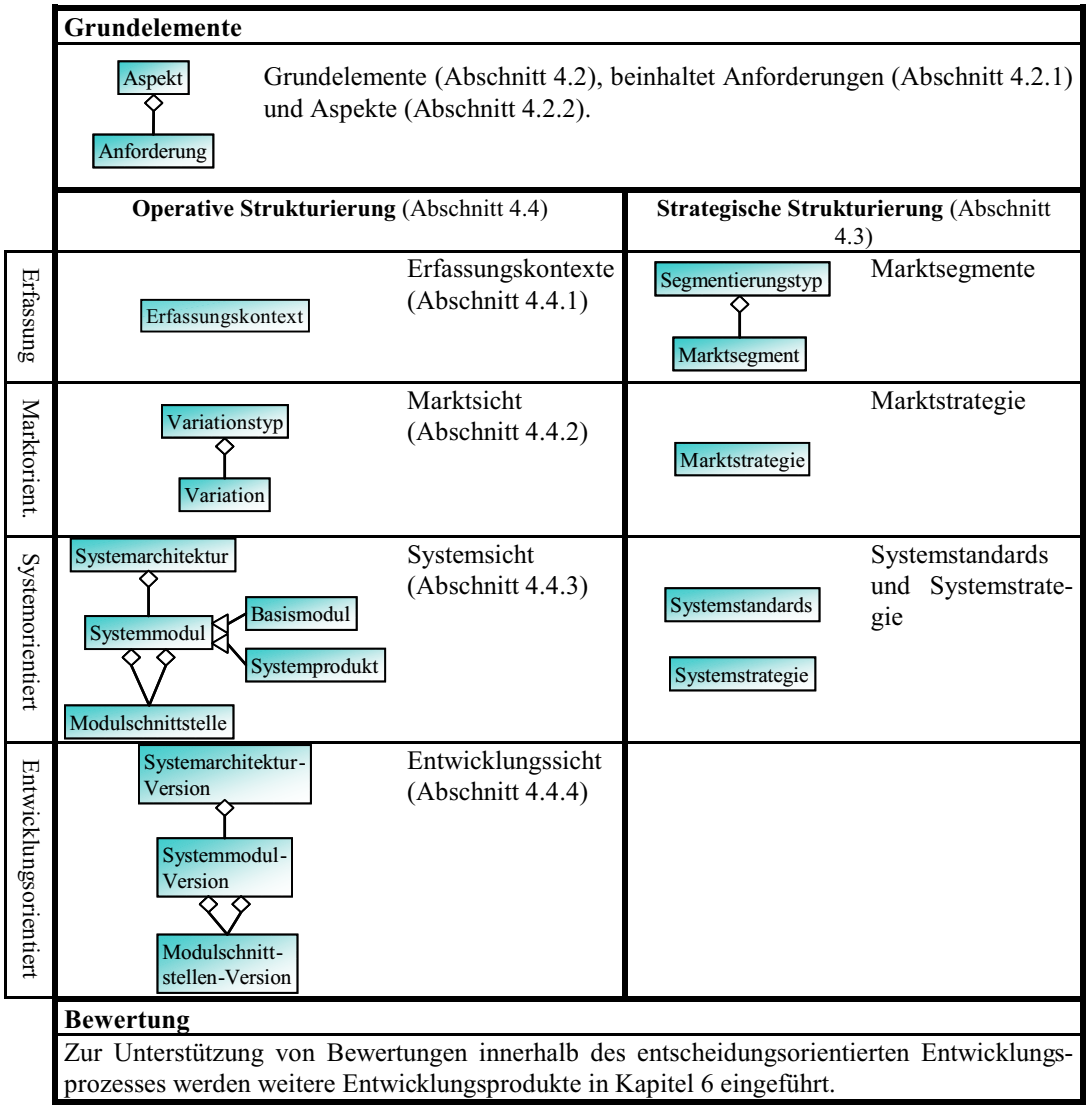
In diesem Kapitel wird aufbauend auf die in Abschnitt 3.2 vorgestellten Modellierungskonzepte ein Modell der Entwicklungsprodukte des Requirements Engineerings komplexer Standardsoftware vorgestellt. Sie dienen zur Dokumentation und zur Unterstützung sämtlicher Entwicklungsaktivitäten innerhalb des Requirements Engineering-Prozesses. Dieser wird in Kapitel 7 ausführlich erläutert. Ein zentrales Ziel dieses Modells ist, durch präzise beschriebene Inhalte und Beziehungen sowohl eine konsistente Entwicklung zu unterstützen, als auch eine gute Verfolgbarkeit zu erreichen.

Die Entwicklungsprodukte und wechselseitige Beziehungen werden mit den in Abschnitt 3.2.4 vorgestellten Hilfsmitteln (graphische Repräsentationen, tabellarische Schemata und formale Notationen) erläutert. Dazu gibt zunächst Abschnitt 4.1 einen Überblick über das Modell. Abschnitt 4.2 führt Grundbegriffe und Grundelemente ein. Im Anschluss daran werden in Abschnitt 4.3 sogenannte strategische Entwicklungsprodukte präsentiert. Dann werden in Abschnitt 4.4 operative Produkte erläutert. Diese repräsentieren Ergebnisse des in Abschnitt 1.3.2 vorgestellten Kerns der Arbeit und werden deshalb sehr ausführlich behandelt. Schließlich wird am Ende dieses Kapitels in Abschnitt 4.5 eine mögliche Werkzeugunterstützung für das Produktmodell diskutiert. Ein Werkzeug ist sowohl für das Requirements Engineering und das Requirements Management als auch das Konfigurations-Management komplexer Standardsoftware behilflich. In diesem Modell wird intensiv eine formale Notation eingesetzt. Diese dient ausschließlich einer präzisen Darstellung der komplexen Zusammenhänge, um die in der Arbeit zentrale Methodik zu unterstützen. Die formale Darstellung erleichtert auch die Realisierung eines Werkzeugs.

Innerhalb der einzelnen Abschnitte wird jeweils auf Aktivitäten aus Kapitel 7 verwiesen, die für die Erarbeitung entsprechender Modellteile zuständig sind. Eine Übersicht definierte Entwicklungsprodukte und Assoziationen wird in Anhang A, eine Übersicht über definierte Mengen und Prädikate in Anhang B gegeben.

4.1 Überblick über das Modell

Bevor die Bestandteile des Modells der Entwicklungsprodukte und deren wechselseitigen Zusammenhänge detailliert beschrieben werden, wird in diesem Abschnitt ein kurzer Überblick gegeben. Den Aufbau des Modells demonstriert die folgende Tabelle.



Durch die dick umrandeten Bereiche sind die vier Hauptteile Grundelemente, operative Strukturierung, strategische Strukturierung und Bewertung des Modells dargestellt. Grundelemente, operative und strategische Strukturierung werden innerhalb dieses Kapitels erläutert. Entwicklungsprodukte der Bewertung werden im Rahmen der Bewertungsmethode in Kapitel 6 präsentiert.

Grundelemente: Grundelemente des Modells der Entwicklungsprodukte sind Anforderungen (vergleiche Kapitel 1) und Aspekte. Aspekte dienen der verschiedenartigen Strukturierung von Anforderungen nach dem Prinzip des Separation of Concerns (vgl. Abschnitt 3.2.5). In Abschnitt 4.2 werden neben Grundelementen auch Grundbegriffe des Modells der Entwicklungsprodukte vorgestellt. Zentrale Begriffe sind Realisierungsstatus und Aktivierung einer Anforderung. Beide dienen einer Verfolgbarkeit beziehungsweise einer konsistenten Realisierung von Anforderungen bis zur Laufzeit einer Software. Darüber hinaus werden Assoziationen zur Dokumentation inhaltlicher Einflüsse von Anforderungen vorgestellt.

Die strategische und die operative Strukturierung sind jeweils nach dem gleichen Prinzip untergliedert. Die Erfassungsorientierung dient der Systematisierung des Herausarbeitens von Anforderungen. Ein Gesamtbild auf Anforderungen eines Marktes wird in der Marktorientierung erarbeitet. Die Systemorientierung zielt auf einen Gesamtblick des zu erstellenden Produktsystems ab. Schließlich dient die Entwicklungssicht der Systemteilspezifischen Versionsplanung.

Strategische Strukturierung: Die strategische Strukturierung dient der langfristigen Ausrichtung der Entwicklung eines Produktsystems. Alle hier enthaltenen Entwicklungsprodukte sind Vorgaben und Anleitungen für Aktivitäten des Prozessmodells in Kapitel 7. Die Marktsegmentierung unterteilt Kunden in Kundengruppen. Für jede Kundengruppe wird in der Marktstrategie festgelegt, welche strategischen Ziele verfolgt werden sollen. Systemstandards und der Systemstrategie legen Vorgaben für die langfristige Entwicklung eines Produktsystems fest.

Operative Strukturierung: Die operative Strukturierung enthält alle Entwicklungsprodukte, die in Abschnitt 1.3.2 als Teil des Kerns der Arbeit vorgestellt wurden. Sie dienen der kurzfristigen Realisierungsplanung der komplexer Standardsoftware. Verglichen mit den strategischen behandeln die operativen Entwicklungsprodukte meist feingranularere Elemente und unterliegen einer sehr viel höheren Änderungsdynamik. Die operative Strukturierung besteht aus:

Erfassungskontexte: Mit Hilfe der Erfassungskontexte wird der Kontext festgehalten, in dem eine Anforderung erfasst worden ist. Damit kann der Ursprung einer Anforderung lückenlos erfasst werden, was deren Pre-Verfolgbarkeit unterstützt (vgl. Abschnitt 1.2.5).

Marktsicht: Die Marktsicht dient der Strukturierung von Anforderungen aus einer gesamtheitlichen Sicht für einen mit der Standardsoftware abgezielten Markt, unabhängig von der Realisierungsstruktur. Hierzu dient die Identifikation von Variationen und die Beschreibung des Einflusses einzelner oder auch mehrerer Variationen auf verschiedene Anforderungen. Es werden unterschiedliche Einflusstypen vorgestellt und deren Wechselwirkung zur Aktivierung und Realisierung von Anforderungen sowie auf die Entwicklung eines Produktsystems erläutert. Variationen werden durch Variationstypen strukturiert.

Systemsicht: Die Systemsicht beschreibt die Aufteilung eines Produktsystems in Systemmodule. Zu Systemmodulen zählen Softwareprodukte, die einzeln verkauft werden, und Basismodule, die der Wiederverwendung von Realisierungsteilen für Softwareprodukte dienen. Systemmodule können gegenseitig voneinander die Erfüllung von Anforderungen fordern. Zur Beschreibung dieser Forderungen dienen Modulschnittstellen. Durch die Aufteilung der Anforderungen auf Systemmodule wird ein Kompromiss zwischen einer marktverkäuflichen und einer möglichst schnell parallel entwickelbaren Software gebildet. Die Systemsicht beschreibt auch Wechselwirkungen des Realisierungszustands von Anforderungen zwischen Systemmodulen und dem Produktsystem. Damit wird eine Verfolgbarkeit der Realisierung der Anforderungen in einzelnen Systemmodulen erreicht.

Entwicklungssicht: Ziel der Entwicklungssicht ist die Unterstützung einer weitgehend parallelen Versionsplanung einzelner Systemmodule. Entwicklungsprodukte der Entwicklungssicht untergliedern die Elemente der Systemsicht in unterschiedliche Versionen. Anforderungen können auf Versionen verteilt werden, womit festgelegt wird, wann sie realisiert werden sollen. Da die Realisierung von Anforderungen nicht immer innerhalb einzelner Versionen abgeschlossen wird, werden Anforderungen unterschiedlicher Dauerhaftigkeit eingeführt. Zur Unterstützung der Verfolgbarkeit werden auch Wechselwirkungen der Versionsplanung hinsichtlich des Realisierungszustands von Anforderungen beschrieben.

In den anschließenden Abschnitten wird das Modell der Entwicklungsprodukte detailliert erläutert.

4.2 Grundbegriffe und Grundelemente des Modells

In diesem Abschnitt werden mehrere Grundbegriffe und -elemente des Modells der Entwicklungsprodukte eingeführt. Zunächst wird das Grundelement Zeitpunkt vorgestellt.

Die Entwicklung komplexer Standardsoftware erfolgt kontinuierlich in einem dynamischen Umfeld. Daher stellen Zustände aller Entwicklungsprodukte jeweils eine Momentaufnahme zu einem gegebenen Zeitpunkt dar.

Definition: Menge aller Zeitpunkte *T*

Die total geordnete diskrete Menge T wird als die Menge aller Zeitpunkte bezeichnet. Sie repräsentiert die Zeit in den formalen Definitionen.

Das zentrale Element des Requirements Engineerings ist die Anforderung. Eine Definition des Begriffs Anforderung wurde bereits in der Einleitung des Kapitels 1 gegeben. Danach ist eine Anforderung ist eine Forderung bestimmter Eigenschaften oder Fähigkeiten, die eine Software erfüllen soll. Der Schwerpunkt dieser Arbeit ist, Anforderungen systematisch zu erarbeiten, zu bewerten, marktrelevante Variationen zu bilden, um sie dann so auf einzelne Entwicklungseinheiten zu verteilen, dass sie effizient entwickelt werden können. Inhaltliche Beschreibungen oder gar semantische Modelle sind daher nur am Rande relevant.

In den folgenden Abschnitten wird zwischen jemals vorhandenen und tatsächlich einem Hersteller einer komplexen Standardsoftware bekannten Anforderungen unterschieden.

Definition: Menge jemals vorhandener Anforderungen

Im gesamten Lebenszyklus einer Standardsoftware in deren Anwendungsbereich jemals vorhandene Anforderungen werden durch die Menge R repräsentiert.

Einem Hersteller von Standardsoftware ist zu jedem Zeitpunkt nur ein Ausschnitt der vorhandenen Anforderungen bekannt. In Abschnitt 7.3 wird vorgestellt, wie Anforderungen an komplexe Standardsoftware herausgearbeitet werden können. In der gesamten Arbeit wird davon ausgegangen, dass nur Anforderungen bekannt sind, die über diesen Weg gesammelt worden sind.

Definition: Menge aller bekannten Anforderungen R_t

Sämtliche zu einem gegebenen Zeitpunkt $t \in T$ einem Hersteller von Standardsoftware aus dem Anwendungsbereich bekannten Anforderungen *werden mit der Menge $R_t \subseteq R$ bezeichnet. Statt bekannter Anforderungen wird auch von* gesammelten Anforderungen *gesprochen.*

Nur bekannte Anforderungen werden als relevant für die Entwicklung komplexer Standardsoftware angesehen. Gesammelte Anforderungen müssen sich nicht ausschließlich auf ein einzelnes Softwaresystem beziehen, sondern können auch an andere Softwaresysteme des selben Anwendungsbereiches beziehen.

Index *t*: Der Index *t* kennzeichnet im Rahmen dieser Arbeit Mengen, deren Inhalte einer zeitlichen Dynamik unterworfen sind. So entstehen im Laufe der Zeit beispielsweise neue Anforderungen, während alte Anforderungen an Bedeutung verlieren können. Die Menge R_t enthält nur Anforderungen, die zu dem gegebenen Zeitpunkt tatsächlich auch zumindest für einzelne Personen mit Interesse an der Software (vgl. Definition aus Kapitel 1) von Bedeutung sind. Für Mengen mit einer zeitlichen Dynamik muss eine horizontale Verfolgbarkeit mit Hilfe eines adäquaten Konfigurations-Managements bewerkstelligt werden (siehe auch Abschnitte 1.2.5 und 4.5).

Definition: Produktsystem (1)

Komplexe Standardsoftware wird auch als Produktsystem bezeichnet. Sie wird formal durch eine Menge S repräsentiert.

(Eine genauere Erläuterung über Eigenschaften eines Produktsystems und Elemente der Menge S erfolgt in Abschnitt 4.4.3.)

Rein informell ist unter einem Produktsystem eine Ansammlung verschiedener Softwareprodukte zu verstehen, die aufgrund bestimmter Rahmenbedingungen gemeinsam entwickelt werden.

Beispiel: Produktsystem

Beispiele hierfür sind Office-Produktfamilien (zum Beispiel Microsoft Office mit Softwareprodukten Word, Excel, Powerpoint und so weiter), betriebliche Standardsoftware (zum Beispiel SAP R/3 mit eigenen Softwareprodukten zum Einkauf, Buchhaltung, Versand und so weiter) oder umfangreiche Softwaresysteme zur Unterstützung der Automatisierungstechnik (zum Beispiel Siemens SIMATIC mit verschiedenen Softwareprodukten zur Unterstützung der Entwicklung und zum Betrieb automatisierter mechanischer Systeme).

Definition: Für ein Produktsystem bekannte Anforderungen $R_{S,t}$

Die Teilmenge $R_{S,t} \subseteq R_t$ repräsentiert die Menge aller speziell für ein Produktsystem S gesammelten Anforderungen zu einem Zeitpunkt $t \in T$.

Im Grunde können sich im Laufe der Zeit alle Anforderungen ändern. In dieser Arbeit wird jedoch davon ausgegangen, dass die Standardsoftware zumindest ihren allgemeinen Charakter beibehält. Bezogen auf die Anforderungen bedeutet das nichts anderes, als dass zu jedem Zeitpunkt noch Anforderungen existieren, die bereits zu Beginn der Entwicklung eines Produktsystems auch schon bestanden.

Manche Anforderungen verlieren jedoch im Laufe der Zeit ihre Bedeutung. Teilweise verlieren sie ihre Gültigkeit ganz, so dass nicht notwendigerweise immer $R_{t_1} \subseteq R_{t_2}$ für $t_1 < t_2$ gelten muss. Diese Erkenntnis ist dann wichtig, wenn bei der Entwicklung neuer Versionen eines Produktsystems die Kompatibilität zu seinen Vorgängerversionen gefordert wird. Kompatibilität muss also nicht unbedingt bedeuten, dass sämtliche in einem Produktsystem realisierten Anforderungen in einer neuen Version erhalten bleiben. Nur für einen Zeitpunkt noch gültige Anforderungen muss die Kompatibilität der Software erhalten bleiben. Diese Problematik wird im Folgenden nicht weiter betrachtet.

4.2.1 Anforderung

Im Modell für Entwicklungsprodukte werden generell Anforderungen an das Produktsystem, also $r \in R_{S,t}$, betrachtet. Sämtliche Entwicklungsprodukte werden durch das in Abschnitt 3.2.3 eingeführte Schema beschrieben. Anforderungen können nach dem folgenden Schema charakterisiert werden:

Produkttyp Anforderung	
B	Repräsentiert eine einzelne Anforderung an das Produktsystem. Hierzu gehören inhaltliche Beschreibungen und Attribute beziehungsweise Aggregationen, die ihren Entwicklungsfortschritt dokumentieren.
Z	Zentrales Element der Entwicklung komplexer Standardsoftware.
A	- **Versionierungstyp** (Normal/ Multi-Version/ Zwischenlösung/ Permanent/ Bedingt): gibt an, wie die Anforderung in der Versionsplanung eingeplant ist. Details hierzu siehe in Abschnitt 4.4.4. - **Filterung** (keine/ja/nein): gibt an, ob die Anforderung grundsätzlich in der Entwicklung weiterbetrachtet werden soll. Details hierzu siehe in Abschnitt 7.5.1.
G	- **Beschreibung:** aggregiert Beschreibungen der Anforderung. Je nach Typ der Anforderung können dies textuelle, graphische oder andere Darstellungsformen (Filme, Tonbandaufnahmen, Prototypen usw.; vgl. Abschnitt 2.3) sein. - **MV-Teil**: aggregiert alle Teilanforderungen, in die eine Multi-Versionsanforderung aufgeteilt worden ist. Eine genaue Erläuterung von Multi-Versionsanforderungen siehe in Abschnitt 4.4.4**.** Aggregate existieren genau dann, wenn der Versionierungstyp *Multi-Version* ist.
B=Beschreibung, Z=Zweck, A=Attribute, G=Aggregationen	

Beispiel: Anforderung

Eine Anforderung an betriebliche Standardsoftware ist beispielsweise „Unterstütze Buchungsabläufe eines Unternehmens". Die Attribute und Aggregationen dieser Anforderung werden sukzessive im Verlaufe der Entwicklung belegt.

Korrelation und Widerspruch zwischen Anforderungen

Von manchen Anforderungen ist nicht das ganze zu realisierende Softwaresystem betroffen. Derartige Anforderungen beziehen sich also lediglich auf spezifische andere Anforderungen. Zur Kennzeichnung dieser Korrelation wird folgende Assoziation eingeführt:

Assoziationstyp korreliert (Anforderung – Anforderung)	
B	Die ungerichtete Assoziation *korreliert* drückt aus, welche Anforderungen sich auf welche anderen Anforderungen beziehen. Dadurch wird die konkrete Ausprägung einer referenzierten Anforderung spezifischer. Hierzu siehe auch nachfolgendes Beispiel.
Z	Die korreliert-Assoziation dient der Unterstützung der Verfolgbarkeit von Anforderungen bei Änderungen. Zur Realisierung einer Anforderung die zu einer anderen Anforderung korreliert wird, müssen alle Bezüge berücksichtigt werden. In diesem Sinne schränkt sich damit das Spektrum ihrer Realisierungsmöglichkeiten einer Anforderung ein. Damit stellt jede korreliert-Assoziation eine Verfeinerung dieser Anforderung dar. Durch eine korreliert-Assoziation entstehen jedoch im Gegensatz zu der gleichnamigen Beziehung von Variationen (siehe Abschnitt 4.4.2) keine neuen Anforderungen.
A	-
B=Beschreibung, Z=Zweck, A=Attribute	

Definition: Korrelation zwischen Anforderungen

Seien $r_1, r_2 \in R$ Anforderungen mit folgender Eigenschaft:

r_1 und r_2 sind logisch nicht voneinander unabhängig.

Diese Eigenschaft stellt eine Assoziation aus $R \times R$ dar und wird mit $correl(r_1, r_2)$ bezeichnet. Die Assoziation wird als Korrelation zwischen zwei Anforderungen *bezeichnet.*

Beispiel: Korrelation zwischen Anforderungen

Bei eingebetteten Systemen korreliert die Anforderung nach hoher Laufzeiteffizienz mit einer ganzen Reihe anderer Anforderungen. Hohe Laufzeiteffizienz ist bei Kernfunktionen einer Steuerung erforderlich. Bei einem Airbag sind z.B. kurze Laufzeiten bei einem Aufprall notwendig, da hier Reaktionen im Millisekundenbereich erfolgen müssen. Statusanzeigen, wie beispielsweise eine Kontrollleuchte zur Anzeige der Aktivierung des Airbags, sind weniger zeitkritisch. Hier reicht eine Reaktionszeit im Sekundenbereich. Die Laufzeiteffizienz korreliert daher nur mit der Aufprallsteuerung.

Eine weitere Relation zwischen Anforderungen ist die widerspricht-Assoziation. Durch die Vielfalt von verschiedenen Kundengruppen sind widersprüchliche Anforderungen an eine komplexe Standardsoftware möglich. Dies repräsentiert die folgende Assoziation:

Assoziationstyp widerspricht (Anforderungsmenge – Anforderungsmenge)	
B	Assoziierte Anforderungsmengen widersprechen einander, können also nicht gleichzeitig in einem Softwaresystem erfüllt sein.
Z	Kennzeichnung von Anforderungsmengen, bei deren gleichzeitigen Realisierung

	Widersprüche in einem Softwaresystem auftreten können. Dies ist insbesondere ein Hinweis darauf, wo Vorkehrungen zur nicht gleichzeitigen Aktivierung von Anford e-rungen getroffen werden müssen (siehe unmittelbar nachfolgenden A bschnitt).
A	-
B=Beschreibung, Z=Zweck, A=Attribute	

Definition: Widerspruch zwischen Anforderungen

Seien $R_1, R_2 \subseteq R$ zwei Mengen von Anforderungen, mit der Eigenschaft, dass R_1 und R_2 sich inhaltlich widersprechen. Das heißt die beiden Mengen können nicht gleichzeitig in einer Software erfüllt sein.

Diese Eigenschaft definiert eine Relation aus $R \times R$, deren Elemente durch das Prädikat $cont(R_1, R_2)$ ausgedrückt werden..

Beispiel: Widerspruch zwischen Anforderungen

Ein einfaches Beispiel für den Widerspruch zwischen Anforderungen ist die Landessprache für textuelle Ausgaben. Einzelne Worte innerhalb eines Textes können nie gleichzeitig in zwei Sprachen dargestellt werden.

Realisierungsstatus einer Anforderung

Der Realisierungsstatus dient der Verfolgbarkeit einer Anforderung, nachdem diese zur Realisierung eingeplant worden ist. Aus der Sicht des Requirements Engineerings re i-chen zwei Realisierungsstati zur sinnvollen Unte rscheidung aus. Demnach kann eine Anforderung *geplant* oder *realisiert* sein. Eine Anforderung kann in unterschie dlichen Teilen eines Produktsystems gleichzeitig verschiedene Realisierungsstati haben (D etails vgl. Abschnitt 4.4.2).

Beispiel: Realisierungsstatus einer Anforderung

Die Anforderung zur Unterstützung der Buchungsabläufe zum Erfassungszeitpunkt weder geplant noch realisiert. Während des Requirements Engineerings wird irgendwann entschieden, dass sie realisiert werden soll. Dann ist sie für das Produktsystem geplant. *Wird sie anschließend in der Software implementiert, dann geht sie in den Realisierungsstatus* realisiert *über.*

Definition: Realisierungsstatus von Anforderungen

Für einen gegebenen Zeitpunkt $t \in T$ und gegebenen bekannten Anforderungen $R_{S,t}$ eines Produktsystems S werden folgende Mengen definiert:

- $R_{S,t}^{pl} \subseteq R_{S,t}$*: alle Anforderungen, die zur Realisierung in S* geplant *sind.*
- $R_{S,t}^{r} \subseteq R_{S,t}$ *: alle Anforderungen, die in S* realisiert *sind.*

Geplante Anforderungen sind gesammelte Anforderungen, für die zum gegebenen Zei t-punkt $t \in T$ die Entscheidung besteht, dass sie im Produktsystem realisiert we rden sollen. Realisierte Anforderungen waren genau einmal geplant und sind zum Zeit punkt $t \in T$ im zur Auslieferung (vgl. Glossar) geplanten oder ausgelieferten Produktsystem S implementiert.

Manche Anforderungen verlieren im Laufe der Zeit ihre Gültigkeit und sind damit nicht mehr zur Realisierung in einem Produktsystem gefordert. Derartige Anforderungen werden durch folgende Menge repräsentiert:

Definition: Verworfene Anforderungen

Die Menge $R_t^c \subseteq R$ *repräsentiert alle ehemals in beliebigen Produktsystemen zur Realisierung geplanten Anforderungen, die zu einem vorgegebenen Zeitpunkt* $t \in T$ *verworfen worden sind.*

Für eine verfolgbare Entwicklung eines Produktsystems ist eine klare Festlegung des Übergangs zwischen Planung und Realisierung von Anforderungen erforderlich. Dies wird in den folgenden Axiomen und Definitionen ausgedrückt.

Entwicklungsaxiom: Übergang von geplanten zu realisierten Anforderungen

Zu einem gegebenen Zeitpunkt $t \in T$ *und für ein gegebenes Produktsystem* S *wird gefordert:*

$$R_{S,t}^r \subseteq \bigcup_{t_0=0}^{t-1} R_{S,t_0}^{pl}$$

Jede zur Zeit $t \in T$ im Produktsystem S realisierte Anforderung ist also vor t geplant worden. Dabei muss nicht jede vor t geplante Anforderungen zum Zeitpunkt t realisiert sein. Umgekehrt können auch verworfene Anforderungen aus $r \in R_t^c$ realisiert sein.

Beispiel: Realisierte und gleichzeitig verworfene Anforderung

Eine Anforderung nach Unterstützung für ein spezifisches Betriebssystem ist realisiert. Das Betriebssystem existiert aber mittlerweile nicht mehr auf dem Markt und ist daher verworfen.

Die Realisierungsstati erlauben, Anforderungen zu verfolgen. So ist beispielsweise ein Vertriebsbeauftragter interessiert, ob, wo und wann von ihm genannte Anforderungen im Produktsystem berücksichtigt werden. Darüber hinaus definiert sich aus den Stati eindeutig, wann eine neue Version eines Produktsystems fertiggestellt ist. Dazu muss jedoch ein klarer Zusammenhang zwischen geplanten und realisierten Anforderungen bestehen.

Üblicherweise werden Anforderungen ein einziges mal geplant und realisiert. Solche Anforderungen werden im folgenden als *verbrauchbar* bezeichnet, da sie nach ihrer Realisierung keine direkten Entwicklungsaufwände mehr verursachen. Für Standardsoftware gibt es weitere Anforderungen, die bei jeder neuen Version erneute Entwicklungsaufwände verursachen. Diese werden als *permanente* Anforderungen bezeichnet.

Definition: Permanente und verbrauchbare Anforderungen

Eine Anforderung $r \in R$ *wird als* permanent *bezeichnet, wenn für sie folgende Eigenschaft gilt:*

$$\exists t_0 \in T : \forall t \geq t_0 : r \in R_{S,t}^{pl} \vee r \in R_t^c$$

Die Menge der geplanten permanenten Anforderungen wird mit $R_{S,t}^{pl_{\text{perm}}} \subseteq R_{S,t}^{pl}$ *bezeichnet. Permanente Anforderungen bleiben immer geplant und kommen nie in einen realisiert-Zustand (vergleiche hierzu aber Abschnitt 4.4.4). Folglich müssen sie bei jeder Weiterentwicklung immer wieder neu berücksichtigt werden. Alle anderen irgendwann einmal geplanten Anforderungen, die nicht diese Eigenschaft aufweisen, werden als* verbrauchbar *bezeichnet.*

Entwicklungsaxiom: Disjunktheit geplanter und verworfener Anforderungen

Für ein gegebenes Produktsystem S wird gefordert: $\forall t \in T : R_{S,t}^{pl} \cap R_t^c = \phi$

Eine permanente Anforderung gilt ab dem ersten Planungszeitpunkt t_0 kontinuierlich zur Realisierung geplant. Einzige Ausnahme ist, dass sie explizit verworfen wird. Verbrauchbare Anforderungen sind nur bis zu ihrer Realisierung geplant:

Entwicklungsaxiom: Verbrauchbare Anforderungen

Für verbrauchbare Anforderungen $r \in R$ *eines Produktsystems S wird gefordert:*
$\forall t \in T : r \in R_{S,t}^r \wedge r \notin R_{S,t}^{pl_{perm}} \Rightarrow r \notin R_{S,t}^{pl}$

In Abschnitt 4.4.4 wird im Zusammenhang mit der Versionsplanung von Produktsystemen detailliert auf permanente Anforderungen eingegangen.

Werden UND-Aggregationen (vgl. Abschnitt 3.2.5) zur Strukturierung von Anforderungen genutzt, so sind befinden sich eine Anforderung und sämtliche mit UND aggregierte Anforderungen immer im selben Realisierungsstatus.

Beispiel: Verbrauchbare und permanente Anforderungen

Die Anforderung „Unterstützung der Buchungsabläufe" ist verbrauchbar. Permanent hingegen sind beispielsweise Anforderungen nach Performanz oder hoher Robustheit.

Konsistentes Produktsystem und Aktivierung

Bei jeder Softwareentwicklung wird die Realisierung konsistenter Softwaresysteme angestrebt. Eine variationsfreie Software wird dann als konsistentes Softwaresystem verstanden, wenn in dem System realisierte Anforderungen sich nicht widersprechen. Bei der Realisierung von Software mit Variationen können jedoch sich widersprechende Anforderungen gleichzeitig realisiert werden. Zur Laufzeit muss dieser Widerspruch behoben werden. Das heißt, dass sich immer nur ein Teil der realisierten Anforderungen auch tatsächlich auf die Funktion der Software auswirken kann. Von einem konsistenten Softwaresystem kann in diesem Fall also dann gesprochen werden, wenn zu jedem Zeitpunkt der Laufzeit dessen Funktion keinen Widerspruch aufweist. Anforderungen, die sich zu einem gegebenen Zeitpunkt tatsächlich auf die Funktion einer Software auswirken, werden als *aktiv* bezeichnet. Die folgenden Absätze dienen der Erläuterung der Zusammenhänge zwischen der Aktivierung von Anforderungen und der Konsistenz eines Softwaresystems.

Definition: Aktivierung einer Anforderung

Gegeben seien ein Zeitpunkt $\tau \in \mathrm{T}$ *und ein Produktsystem S. Eine Anforderung* $r \in R_{S,\tau}$ *heißt genau dann* aktiv*, wenn das Produktsystem S die Anforderung zu dem gegebenen Zeitpunkt erfüllt. Eine Anforderung ist* erfüllt*, wenn sie sich zu diesem Zeitpunkt* $\tau \in \mathrm{T}$ *tatsächlich auf die Funktion des Produktsystems auswirkt.*

Formal wird die Aktivierung mit dem Prädikat $active_S(r,\tau)$ *beschrieben. Nicht aktive Anforderungen werden als* inaktiv *bezeichnet, in Zeichen* $inactive_S(r,\tau)$.

Der Zeitpunkt in der Definition kann sich sowohl auf die Entwic klungszeit, als auch die Laufzeit beziehen. Das heißt, dass eine Anforderung zeitweise auch während der Lau fzeit einer Software nicht erfüllt sein kann.

Zu inaktiven Anforderungen eines Produktsystems gehören sowohl gar nicht realisierte Anforderungen als auch realisierte Anforderungen. Realisierte Anforderungen sind z um gegebenen Zeitpunkt sozusagen „ausgeschaltet“ worden. Eine Anforderung gilt als real isiert, wenn sie sich im Programmcode in irgendeiner Weise widerspiegelt. Eine derartige Anforderung kann entweder während der Softwareinstallation oder während seiner L aufzeit aktiv werden. Eine Anforderung kann nur aktiv sein, wenn Sie auch realisiert ist.

Beispiel: Aktivierung einer Anforderung

Eine Anforderung an ein Softwaresystem lautet: „Die textuelle Ausgabe sämtlicher Systemmeldungen soll einheitlich in einer Sprache erfolgen“. Zwei Variationen hiervon sind Textausgaben in deutsch *und in* englisch*. Werden in einer Software beide Sprachen realisiert, so kann eine davon entweder während der Installation oder während der Laufzeit aktiviert werden. Wird die Sprache* deutsch *aktiviert, erfolgen danach alle Textausgaben in Deutsch. Diese Variation ist erfüllt. Die zweite Variation ist nicht erfüllt, da eine gleichzeitige Ausgabe in* englisch *nicht erfolgt, sie ist also inaktiv..*

Auf der Aktivierung von Anforderungen baut si ch der Begriff der zulässigen Konfigur ation auf.

Definition: Zulässige Anforderungs-Konfiguration eines Produktsystems

Gegeben seien Zeitpunkt $\tau \in \mathrm{T}$ *und ein Produktsystem S. Eine* Anforderungs-Konfiguration des Produktsystems S (in Zeichen $Conf_{S,\tau}$*) repräsentiert sämtliche Aktivierungen aller realisierten Anforderungen* $r \in R^r_{S,\tau}$*. Eine* zulässige Anforderungs-Konfiguration $Conf_{S,\tau}$ *ist eine Anforderungs-Konfiguration für die gilt:*

$$\forall r_1, r_2 \in R^r_{S,\tau} : cont(r_1, r_2) \Rightarrow \neg(active_S(r_1,\tau) \wedge active_S(r_2,\tau))$$

In einer zulässigen Anforderungs-Konfiguration eines Produktsystems sind keine zwei widersprüchliche Anforderungen gleichzeiti g aktiv. Daraus wird der Begriff eines kons istenten Produktsystems abgeleitet:

Definition: Konsistentes Produktsystem

Sei S ein Produktsystem mit Anforderungs-Konfigurationen $Conf_{S,\tau}$ *für Zeitpunkte* $\tau \in \mathrm{T}$. *S wird als* konsistent *bezeichnet gdw. gilt:*
$\forall t \in \mathrm{T}: Conf_{S,t}$ *ist eine zulässige Konfiguration.*

Für die Entwicklung komplexer Standardsoftware wird also folgendes gefordert:

Entwicklungsaxiom: Entwicklung eines Konsistenten Produktsystems

Für die Entwicklung eines Produktsystems wird die Entwicklung eines konsistenten Produktsystems gefordert.

Da ein Produktsystem widersprüchliche Anforderungen realisieren kann, können in einer Spezifikation geplanter Anforderungen Widersprüche auftreten. Diese dürfen jedoch nicht willkürlich sein, sondern müssen die Realisierung eines konsistenten Produktsystems erlauben. In nachfolgenden Absätzen und in Abschnitt 4.4.2 werden hierzu sogenannte *Aktivierungsregeln* aufgestellt, die Abhängigkeiten zwischen Anforderungen hinsichtlich deren Aktivierung beschreiben. Diese unterstützen die Entwicklung eines konsistenten Produktsystems.

Eine Aktivierungsregel kann explizit im Requirements Engineering durch die benötigt-Assoziation aufgestellt werden:

Assoziationstyp benötigt (Anforderung → Anforderung)	
B	Beschreibt die notwendige Aktivierung einer Anforderung aufgrund der Aktivierung einer anderen Anforderung.
Z	Darstellung von Konsistenzbedingungen für die Aktivierung von Anforderungen.
A	-
B=Beschreibung, Z=Zweck, A=Attribute	

Definition: benötigt-Assoziation zwischen Anforderungen

Sei S ein Produktsystem. Eine Anforderung $r_1 \in R$ *benötigt die Aktivierung einer Anforderung* $r_2 \in R$ *gdw. gilt:* $\forall \tau \in T: active_S(r_1, \tau) \Rightarrow active_S(r_2, \tau))$. *Diese gerichtete Assoziation aus* $R \times R$ *wird durch das Prädikat* $need(r_1, r_2)$ *ausgedrückt.*

Die in diesem Abschnitt eingeführten Anforderungen werden im Rahmen von Aktivitäten des Herausarbeitens in Abschnitt 7.4.3 identifiziert. Assoziationen werden in der ersten zyklischen Phase, der Definition der Marktsicht in Abschnitt 7.5.1 festgelegt. Als zentrales Entwicklungsprodukt werden sie in allen anderen Aktivitäten unter unterschiedlichen Gesichtspunkten eingesetzt.

4.2.2 Aspekt

In Abschnitt 3.2.5 wurde das Prinzip der Zerlegung nach Betrachtungsfokus (engl. *Separation of Concerns*) vorgestellt. Danach können allgemein unterschiedliche Problembereiche strukturiert werden. Durch die aspektorientierte Programmierung hat dieses klassische Prinzip wieder an Popularität gewonnen. Mittlerweile existieren Bestrebungen, auch in vorgelagerten Entwicklungsphasen zu integrieren (vgl. hierzu [CTO99,

KLM+97, WBM99]). In diesen neueren Berichten wird der Begriff *Aspekt* anstelle von Betrachtungsfokus verwendet. Ohne diese Materie tiefer zu beleuchten, wird in dieser Arbeit davon ausgegangen, dass sich ein komplexes Standardsoftwaresystem mit Aspekten sinnvoll in Teile zerlegen lässt.

Produkttyp Aspekt	
B	Beschreibung eines spezifischen Aspektes der Anforderungen an ein Produktsystem und Zuordnung der entsprechenden Anforderungen.
Z	Strukturierung der Anforderungen nach dem Prinzip des Separation of Concerns
A	**Beschreibung**: Informelle Beschreibung der Eigenschaften des spezifischen Aspekts.
G	**Anforderung**: Anforderungen, die zu diesem Aspekt gehören. Allgemein können Anforderungen zu unterschiedlichen Aspekten aggregiert sein.
B=Beschreibung, Z=Zweck, A=Attribute, G=Aggregationen	

Beispiel:

Aspekte einer betrieblichen Standardsoftware sind beispielsweise Buchung, Bilanzierung, Grafiken und Statistiken. Hier orientiert sich die Aufteilung also an zu erfüllenden Funktionen.

Definition: Menge aller Aspekte

Sei S ein Produktsystem, das in unterschiedliche Aspekte gegliedert ist. Die Menge $ASP_{S,t}$ *repräsentiert die Menge aller Aspekte von S zu einem gegebenen Zeitpunkt* $t \in T$. *Für einen gegebenen Aspekt* $A \in ASP_{S,t}$ *werden mit* $R_{A,t} \subseteq R_{S,t}$ *die zugehörigen Anforderungen bezeichnet. Die Anforderungen aus* $R_{A,t}$ *genügen den Eigenschaften des Aspektes.*

Anforderungen müssen gegebenenfalls zerlegt werden, um sie Aspekten zuordnen zu können. Aspekte werden zu verschiedensten - möglicherweise auch zueinander orthogonalen - logischen Strukturierungen eines Softwaresystems verwendet. Daher wird aus methodischer Sicht gefordert, dass jede Anforderung an das Produktsystem mindestens einem Aspekt zugeordnet wird. Dies stellt das folgende Axiom dar:

Entwicklungsaxiom: Zuordnung von Anforderungen zu Aspekten:

Sei $ASP_{S,t}$ *die Menge aller seiner Aspekte eines Produktsystems S zu einem gegebenen Zeitpunkt* $t \in T$. *Es wird gefordert:*

$$\bigcup_{A \in ASP_{S,t}} R_{A,t} = R_{S,t}.$$

Die Strukturierung eines Softwaresystems in Aspekte muss der späteren physischen Strukturierung in Komponenten und Beziehungen nicht entsprechen. In der Regel wird sich die physische Struktur jedoch an der Zerlegung nach bestimmten Aspekten orientieren.

Bei zunehmender Anzahl von Anforderungen ist es vom Aufwand her nicht mehr akzeptabel, jede einzelne *korreliert*-Assoziation zwischen Anforderungen zu spezifizieren. Zur Reduktion des Spezifikationsaufwandes wird daher bewusst in Kauf genommen, dass

nicht mehr jede einzelne Korrelation unmittelbar erkennbar ist. Umgekehrt sind trotzdem Hinweise auf eventuelle Einflüsse notwendig. Dazu wird die folgende Assoziation zwischen Aspekten eingeführt:

Assoziationstyp korreliert (Aspekt - Aspekt)	
B	Die gerichtete Assoziation *korreliert* stellt eine von der gleichnamigen Assoziation zwischen Anforderungen eine abgeleitete Beziehung dar. Wenn ein Aspekt mit einem anderen Aspekt korreliert, dann existieren Anforderungen aus den beiden Aspekten, die in *korreliert*-Assoziation zueinander stehen. Siehe auch nachfolgendes Beispiel.
Z	Die *korreliert*-Assoziation für Aspekte dient der übersichtlichen Darstellung von Korrelationen zwischen Gruppen von Anforderungen. Anstatt der Darstellung jeder einzelnen Beziehung zwischen Anforderungen werden die zugehörigen Klassen von Anforderungen in Form der Aspekte dargestellt.
A	-
B=Beschreibung, Z=Zweck, A=Attribute	

Beispiel: Korrelation zwischen Aspekten

Angenommen, Benutzerfreundlichkeit und die Ein-/Ausgabe sind Aspekte eines Produktsystems. Software gilt allgemein als benutzerfreundlich, wenn die Nutzerinteraktion entsprechend gestaltet ist. Daher kann Forderung nach Benutzerfreundlichkeit auf die Ein-/Ausgabe beschränkt werden. Ein System erfüllt also dann diese Forderung, wenn die Ein-/Ausgabe benutzerfreundlich gemäß der Anforderungen im Aspekt der Benutzerfreundlichkeit gestaltet ist.

Definition: Korrelation zwischen Aspekten

Seien $A_1, A_2 \in ASP_{S,t}$ *zwei Aspekte eines Produktsystems S zum Zeitpunkt* $t \in T$. *Zwischen* A_1 *und* A_2 *besteht eine* Korrelation *gdw. gilt:*

$\exists r_1 \in R_{A_1,t} \exists r_2 \in R_{A_2,t} : correl(r_1, r_2)$.

Diese Assoziation aus $ASP_{S,t} \times ASP_{S,t}$ *wird durch das Prädikat* $correl(A_1, A_2)$ *bezeichnet. Zwei Aspekte beeinflussen sich also dann, wenn mindestens eine Anforderung des einen Aspektes mit mindestens einer Anforderung des anderen Aspektes in Korrelation steht.*

In Kapitel 5 wird zwischen Spezifikationstechniken zur Schaffung eines Überblicks über ein zu entwickelndes System und Techniken zur Unterstützung der detaillierten Entwicklung unterschieden. Für die übersichtliche Darstellung reicht in der Regel die Spezifikation der Korrelation zwischen Aspekten. Eine lückenlose Verfolgbarkeit wird hingegen nur erreicht, wenn jede einzelne Korrelation zwischen Anforderungen spezifiziert wird.

Aspekte und zugehörige Assoziationen werden durch Aktivitäten aus Abschnitt 7.5.1 festgelegt. Sie werden zur Definition der Systemarchitektur und der Zuordnung von Anforderungen zu Systemmodulen eingesetzt (vgl. Abschnitte 4.4.3 und 7.5.2). Abbildung 8 gibt einen Überblick über bisher definierte Entwicklungsprodukte.

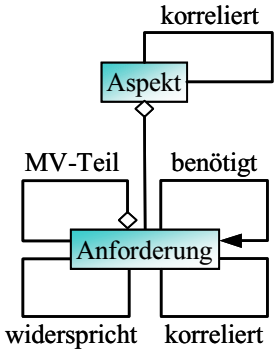


Abbildung 8 – Aspekte und Anforderungen

4.3 Strategische Strukturierung

Nach der Einführung von Grundelementen kann in diesem Ab schnitt die Vorstellung der *strategischen Entwicklungsprodukte* erfolgen. Wegen des Wandels auf dem Softwar emarkt müssen langfristige Strategien künftige Entwicklungsrichtungen sowohl aus Marktsicht als auch aus Entwicklungssicht festlegen. Entwicklungsprod ukte der strategischen Strukturierung dienen der Dokumentation derartiger langfristiger Planu ngen.

Im Rahmen des Prozessmodells in Kapitel 7 werden strategische Entwicklungsprodukte als gegeben vorausgesetzt. Sie we rden in gesonderten Aktivitäten erarbeitet und dienen in den Aktivitäten als Vorgaben und Anle itungen.

Marktstrategie

Die Markstrategie dient der langfristigen Ausrichtung der Standardsoftware auf den Markt. Sie wird von der Rolle strategisches Marketing erstellt (vg l. Abschnitt 7.2.1).

Produkttyp Marktstrategie	
B	Dokumentation identifizierter Trends auf dem Markt und Festl egung strategischer Vorgaben für das Requirements Engineering und die Entwicklung. Die Marktstrat egie wird kontinuierlich an neue Marktgegebenheiten angepasst und ist nicht unmi ttelbares Ergebnis des Requirements Engineering Prozesses.
Z	Die Marktstrategie dient dazu, eine komplexe Standardsoftware langfri stig auf dem Markt wettbewerbsfähig zu halten.
A	- **Markttrends**: Fasst eine Beobachtung des Marktes zusammen und erläutert da raus abgeleitete künftig erwartete Entwicklu ngen auf dem Softwaremarkt. Dies schließt sowohl grobe künftige Richtungen von Anforderungen als auch mögliche Umsatztendenzen abhängig vom Marktwachstu m als auch von der Konkurrenzs ituation ein. Diese Informationen dienen der langfristigen bedarfsgesteuerten Au srichtung der komplexen Standardsoftware. - **Technologietrends**: Neue Technologien können der Ausgangspunkt für innovat ive Anforderungen sein, wenn d iese überhaupt erst dadurch mit vertretbarem Au fwand realisierbar werden. Technologien können erst auch neue Bedürfnisse im Markt erzeugen (vgl. [CoK87]). Daher müssen sich auf dem Markt befin dliche

	Technologien beobachtet und Trends analysiert werden. Neben daraus mögliche rweise erkennbaren Anforderungen werden neue Technologien auch zur Steigerung der Kosteneffizienz in der eigenen Entwicklung genutzt. - **Kernkompetenzen**: Aus den ermittelten Trends werden notwendige Kernkompetenzen der Software abgeleitet. Mit deren Hilfe werden künftige Entwicklungsrichtungen der Software festgelegt. Diese beeinflussen sowohl die Planung der Software selbst als auch eine zielgerichtete Entwicklungsvorbereitung hinsichtlich benötigter Ressourcen und notwendigen Know-hows. - **Beurteilungsrichtlinien von Anforderungen**: Festlegung von Richtlinien, anhand derer Anforderungen innerhalb des Requirements Engineering Prozesses beurteilt werden sollen. Beispielsweise wird vorgegeben, in welcher Weise unterschiedliche Quellen bezüglich ihrer Aussagekräftigkeit und ihres Marktpotentials beurteilt werden sollen. - **Schlüsselkunden:** Als *Schlüsselkunden* werden Kunden bezeichnet, deren Kaufverhalten als strategisch wichtig betrachtet wird. Diese sind entweder selbst sehr große Abnehmer der Software oder sehr innovative Kunden sind. Aus deren Verhalten können meist künftige Marktpotentiale abgeleitet werden. Die Festlegung der Schlüsselkunden kann nicht unabhängig von der Festlegung der Beurteilungskriterien für Anforderungen passieren. Schlüsselkunden sind häufig Kandidaten aktiver Kontakte beim Herausarbeiten von Anforderungen (siehe Abschnitt 7.3) - **Richtlinien zur Risikobereitschaft:** Mit den Richtlinien zur Risikobereitschaft wird festgelegt, wie mit unsicheren und sehr vagen Anforderungen umgegangen wird. Beispielsweise wird der Kostenanteil für Anforderungen mit sehr geringer Vorhersagbarkeit hinsichtlich ihres möglichen Markterfolges festgelegt. Eine höhere Risikobereitschaft kann zu höheren Innovationen führen. Diese leitet sich unmittelbar von den Unternehmenszielen ab. - **Strategische Gewichtung der Entwicklungstätigkeiten:** Der kurzfristige Gewinn ist für die mittelfristige Richtungsvorgaben einer Standardsoftware nicht immer ausschlaggebend. So kann beispielsweise die Erhöhung des Marktanteiles notwendig sein, um langfristig konkurrenzfähig bleiben zu können. Möglicherweise ist die Standardsoftware nur Bestandteil einer umfangreicheren anderen Software (vgl. beispielsweise in der Vergangenheit die Diskussion um den Microsoft Explorer als Teil von Windows) oder eines anderen Produktes ist. Die Standardsoftware ist in solchen Fällen lediglich ein Erfolgsfaktor für das andere Produkt, ohne selbst wettbewerbsfähig zu sein. Anforderungen können in solchen Fällen nicht nach deren Marktwert bewertet werden.
G	-
B=Beschreibung, Z=Zweck, A=Attribute, G=Aggregationen	

Systemstandards

Zur langfristigen Auslegung der Standardsoftware werden aus Entwicklungssicht Anleitungen und Vorlagen benötigt, die eine einheitliche Entwicklung eines Produktsystems erlauben. Das folgende Entwicklungsprodukt dient der Dokumentation derartiger Standards. Es wird vom Systemstrategen festgelegt (vgl. entsprechende Rolle in Abschnitt 7.2.2).

Produkttyp Systemstandards	
B	Legt in der Entwicklung eines Softwaresystems zu verwendende Sta ndards fest. Dazu gehören zum Beispiel Entwurfsmuster (vgl. z.B. [GHJ+96], Kommunikationsmech anismen und Schnittstellena nleitungen, Programmier-Standards. Die Systemstandards werden vor der ersten Entwicklung definiert und sollen mö glichst weitgehend stabil bleiben.
Z	Entwicklungszeit und Kosten werden wesentlich durch die Änderbarkeit eines Sof twaresystems bestimmt. Diese verringert sich bei alter nder Software fortwährend. „Altern" von Software bedeutet nach Parnas (vgl. [Par94]) Software, die laufend erweitert wird und daher zunehmend an klaren Strukturen verliert. Dies liegt häufig an einer mangelnden Dokumentation und an der Tatsache, dass im Laufe der Zeit verschiedene Designprinzipien angewendet werden (vgl. z.B. [Leh98]). Systemstandards sind beispielsweise für Anforderungen an Effi zienz, Zuverlässigkeit oder Wiederverwendbarkeit von Teilen eines Softwaresystems von besonderem Int eresse. Durch klare Vorgaben können auch einheitliche Software Architekturen entw ickelt werden, womit die Geschwindigkeit des Alterns der Standardsoftware verringert wird. Weitere Systemstandards können sich beispielsweise auf zu verwe ndende Kommunikationsmechanismen beziehen, um die Interoperabilität zu anderen Softwaresystemen zu ermöglichen.
A	Informelle Beschreibung unterschiedlicher Systemstandards
G	-
B=Beschreibung, Z=Zweck, A=Attribute, G=Aggregationen	

Marktsegmente

Mit Hilfe der Marktsegmente wird festgelegt, welche groben Kundenprofile langfristig durch ein Produktsystem unterstützt werden sollen. Gleiche Kundenprofile weisen Ku nden eines Produktsystems auf, die ähnliche Anforderungen an ein Produktsystem stellen. Ein Markt kann nach unterschiedlichen Kriterien segmentiert werden, wie beispielsweise Region, Branche, Unternehmensgröße usw. Abhängig davon entstehen u nterschiedliche Marktsegmente. Daher werden zur Beschreibung von Marktsegmenten zwei Produktt ypen eingeführt:

- Segmentierungstyp und
- Marktsegmente

Marktsegmente werden durch das Strategische Marketing festgelegt (vgl. entspr echende Rolle in Abschnitt 7.2.1).

Produkttyp Segmentierungstyp	
B	Festlegung einer Kriterienauswahl, wonach der Markt segmentiert we rden kann.
Z	Dokumentation, anhand welcher Gesichtspunkte der Markt segmentiert wurde.
A	Informelle Beschreibung der Kriterien.
G	**Marktsegmente:** alle Marktsegmente, die nach diesem Kriterium en tstanden sind.
B=Beschreibung, Z=Zweck, A=Attribute, G=Aggregationen	

Produkttyp Marktsegment	
B	Beschreibung bestimmter Kundenprofile hinsichtlich einer bestimmten Ausprägung der Kriterien eines Segmentierungstyps.
Z	Langfristige Auslegung des Produktsystems durch strategische Festl egung, welchen Teilmarkt dieses unterstützen soll.
A	- **Informelle Beschreibung:** Beschreibung der Eigenschaften und allgemeiner A nforderungen des Marktsegments. - **Bewertung der strategischen Wichtigkeit**
G	-
B=Beschreibung, Z=Zweck, A=Attribute, G=Aggregationen	

Sämtliche Marktsegmente müssen mindestens einem Segmentierungstyp zugeordnet sein.

Systemstrategie

Mit Hilfe der Systemstrategie werden strategische Vorgaben für die künftige Weiteren twicklung des Produktsystems festgelegt. Der Systemstratege ist für die Erstellung der Systemstrategie verantwortlich (vgl. entspreche nde Rolle in Abschnitt 7.2.1).

Produkttyp Systemstrategie	
B	Beschreibt die langfristigen Vis ionen und Zwischenstufen eines Produktsystems. Darüber hinaus werden für einzelne Versionen Schwerpunkte festgelegt. Diese kö nnen sich sowohl auf Anforderungen als auch auf realisierungstechnische Änderungen beziehen. Weiterhin werden die künftig geplante Entstehung neuer, das Verschwi nden und die mögliche Verschmelzung bestehender Systemmodule b eschrieben.
Z	Unterstützt die Entwicklung in der Vorbereitung des Produktsystems auf künftige geplante Veränderungen. Darüber hinaus werden für die Entscheidung de r Verteilung von Anforderungen auf Versionen separate strategische Zi elen definiert. Eines dieser Ziele kann beispielsweise die Forderung sein, dass jede Version zumi ndest eine Anforderung enthält, die bei Kunden eine derartige Begeisterung auslösen, dass sie eine neue Version ka ufen.
A	- **Langfristige Vision:** Beschreibung, wohin sich das Produktsystem langfristig entwickeln und wie das allgemein erreicht werden soll. - **Schrittweise Planung:** Beschreibung, in welchen Schritten die langfristige Vision angestrebt werden soll. Diese Schritte sind ein wichtiger Anhaltspunkt für die Ve rsionsplanung. - **Geplante Systemarchitekturänderungen:** Beschreibung, welche neuen Syste mmodule entstehen, welche verschwinden und welche miteinander verschme lzen (zur Beschreibung der Sys temmodule siehe auch Abschnitt 4.4.3).
G	-
B=Beschreibung, Z=Zweck, A=Attribute, G=Aggregationen	

4.4 Operative Strukturierung

In diesem Abschnitt werden Entwicklungsprodukte der operativen Strukturierung vorgestellt. Zunächst wird ein Entwicklungsprodukt zur Unterstützung des Herausarbeitens von Anforderungen (vgl. Abschnitt 1.2.1) erläutert. Im Anschluss daran werden drei unterschiedliche Strukturierungen von Anforderungen betrachtet. Diese leiten stufenweise von einer marktorientierten Betrachtungsweise in eine entwicklungsadäquate Sicht über. Dies sind Marktsicht, Systemsicht und Entwicklungssicht.

4.4.1 Erfassungskontexte

Das Entwicklungsprodukt Erfassungskontext dient der Unterstützung der Pre-Verfolgbarkeit (vgl. Abschnitt 1.2.5) von Anforderungen. Dieses gliedert sich in drei wesentliche Elemente, der *Quelle*, dem *Kanal* und *der Erfassungsmethode*. Eine Anforderung kann mehrfach in unterschiedlichen Erfassungskontexten erfasst worden sein. Abbildung 9 zeigt die Ergänzung des bisherigen Produktmodells.

Der Erfassungskontext dient nicht nur der reinen Verfolgbarkeit von Anforderungen, sondern wird auch zur Bewertung von Anforderungen eingesetzt, wie in Abschnitt 6.7.4 erläutert wird.

Produkttyp Erfassungskontext	
B	Dokumentation der Herkunft einer Anforderung durch Beschreibung des Urhebers, des Anlasses seiner Erfassung und einer eventuell angewendeten Methode.
Z	Dient der Pre-Verfolgbarkeit der Anforderung und der Unterstützung ihrer Bewertung. (Details siehe Abschnitt 6.7)
A	- **Quellenbeschreibung:** Beschreibung des Urhebers (oder Quelle) der Anforderung. - **Quellentyp:** Anforderungen können beispielsweise von Kunden stammen oder aus Normen herausgelesen werden. Zur Unterscheidung verschiedenartiger Quellen wird der Typ der Quelle festgelegt. (Details siehe Abschnitt 7.4.2). - **Kanalbeschreibung:** Beschreibt den eigentlichen Anlass oder Weg der Erfassung einer Anforderung. Hier werden beispielsweise Personen genannt, die eine Anforderung erfasst haben oder die konkrete Situation, in der sie erfasst wurde. - **Kanaltyp:** Wie beim Quellentyp kann zwischen verschiedenartigen Kanälen unterschieden werden. Dies wird durch den Typ des Kanals eindeutig bestimmt (Details siehe Abschnitt 7.4.3) - **Erfassungstyp:** Dokumentiert die Methoden, die zur Erfassung von Anforderungen angewendet wurden. - **Erfassungsdatum:** Datum, an dem die Anforderung erfasst wurde.
G	-
B=Beschreibung, Z=Zweck, A=Attribute, G=Aggregationen	

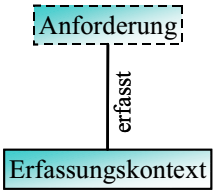


Abbildung 9 – Erfassungskontext einer Anforderung

Beispiel: Erfassungskontext

Im Rahmen eines Verkaufsgesprächs des Vertriebsbeauftragten B. Müller mit einem Mitarbeiter des Kunden H. Schmid wird die Anforderung nach einer Betriebssystemkompatibilität des Produktsystems mit Linux gefordert. Damit ist Quelle der Anforderung H. Schmid, der vom Quellentyp Kunde ist. Kanal ist B. Müller, der zum Kanaltypen passive Kontakte (speziell Verkaufsgespräche) gehört. Da die Erfassung von Anforderungen in Verkaufsgesprächen nicht im Mittelpunkt steht, wurde hierfür keine besondere Erfassungsmethode eingesetzt.

Der Erfassungskontext beschränkt sich auf für diese Arbeit wesentliche Attribute. Ein Modell zur Dokumentation weiterer Attribute von Erfassungskontexten wird zum Beispiel ausführlich in der Arbeit zu Beitragsstrukturen [Got95] behandelt.

4.4.2 Marktsicht

Die Marktsicht dokumentiert Anforderungen an ein Produktsystem aus der Perspektive eines gesamten betrachteten Marktes. Sie umfasst Anforderungen sämtlicher Kunden, auf die das Produktsystem abzielt. Die Sicht ist völlig unabhängig von der Realisierungsstruktur. Es wird daher von einer Aufteilung in Softwareprodukte abstrahiert. Damit wird der Tatsache Rechnung getragen, dass einem Kunden vorrangig an der Erfüllung seiner Anforderungen gelegen ist. Für ihn ist die Aufteilung des Produktsystems in einzelne Softwareprodukte von geringerer Bedeutung. Man gewinnt durch diese Strukturierung vor allem, dass Anforderungen nicht doppelt in unterschiedlichen Softwareprodukten realisiert werden. Umgekehrt werden alle Softwareprodukte einheitlich gestaltet, da erst übergreifende Anforderungen identifiziert werden, die erst anschließend verteilt werden.

Aufgrund der großen Anzahl von Kunden kann nicht jede individuelle Kundensicht einzeln spezifiziert werden. Daher wird der Gesamtmarkt in unterschiedliche Gruppen aufgeteilt. Ein einzelner Kunde gehört dann zu der Gruppe, die seiner eigenen Sicht am meisten ähnelt. Die Gruppierung erfolgt aus unterschiedlichen Perspektiven. Mögliche Perspektiven sind beispielsweise verschiedene Segmentierungstypen für Marktsegmente (vgl. Abschnitt 4.3). Zur Darstellung der Marktsicht werden die Anforderungen in Variationen aufgeteilt. Jede Variation repräsentiert Anforderungen einer bestimmten Gruppe.

Eine wesentliche Gruppenaufteilung wird etwa durch die Marktsegmentierung der strategischen Strukturierung der Entwicklungsprodukte vorgegeben. Variationen sind jedoch nicht auf die Marktsegmentierung beschränkt. Sie können beispielsweise auch zur ganz feinen Unterscheidung einzelner Anforderungen herangezogen werden, wie z. B. die unterschiedliche farbliche Darstellung des Hintergrunds einzelner Bildschirmfenster.

Bei der Marktsicht wird auch von einer Aufteilung auf Versionen abstrahiert, um einen Überblick über sämtliche augenblicklich gestellten Anforderungen zu bekommen. In den folgenden Unterabschnitten werden Entwicklungsprodukte und Assoziationen der Marktsicht vorgestellt. Abbildung 10 gibt hierzu einen Überblick. In [Dei99a] wurden erste Ideen hinsichtlich Variationen und möglichen Assoziationen veröffentlicht. Diese wurden im vorliegenden Kapitel erweitert und in das gesamte Modell der Entwicklungsprodukte integriert.

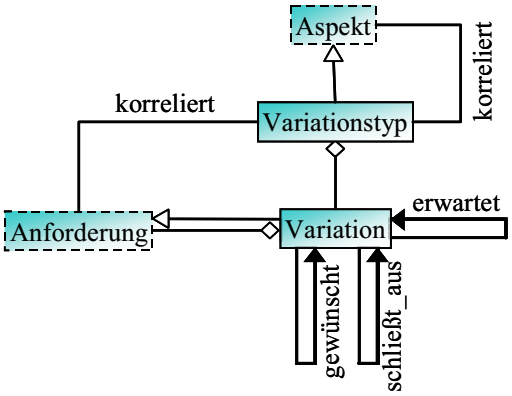


Abbildung 10 – Entwicklungsprodukte der Marktsicht

Die Erarbeitung der Marktsicht geschieht im Rahmen der zyklischen Phasen des Prozessmodells in der Phase Definition der Marktsicht in Abschnitt 7.5.1. Die hier definierten Abhängigkeiten müssen für Realisierungsentscheidungen und die Definition der Entwicklungssicht (vgl. Abschnitte 4.4.4 und 7.5.3) zur Schaffung eines konsistenten Produktsystems berücksichtigt werden.

Variationstypen und Variationen

Für jede Kundengruppe bestehen hinsichtlich einzelner Aspekte variierende Anforderungen an ein Produktsystem. Zur Modellierung derartiger Variationen wird zunächst ein Subtyp von Aspekten, der sogenannte *Variationstyp*, definiert:

Produkttyp Variationstyp (abgeleitet von Aspekt)	
B	Beschreibung des Zusammenhangs unterschiedlicher Variationen eines Softwaresystems.
Z	Explizite Darstellung variierender Teile eines Softwaresystems, um die Verfolgbarkeit bei Änderungen, Neuerscheinen oder Ausscheiden einzelner Variationen zu erhöhen.
A	-
G	**Variation:** alle Variationen, die hinsichtlich dieses Variationstyps variieren.
B=Beschreibung, Z=Zweck, A=Attribute, G=Aggregationen	

Beispiel:

Ein möglicher Variationstyp ist die Landessprache für textuelle Ausgaben eines Softwaresystems.

Variationstypen unterscheiden sich von allgemeinen Aspekten (vgl. Abschnitt 4.2.2) durch eine Unterteilung zugehöriger Anforderungen in Variationen.

Definition: Menge aller Variationstypen

Sei S ein Produktsystem und $ASP_{S,t}$ die Menge aller Aspekte zu einem gegebenen Zeitpunkt $t \in T$. Variationstypen entsprechen einer Teilmenge aller Aspekte und werden mit $VT_{S,t} \subseteq ASP_{S,t}$ bezeichnet.

Variationstypen aggregieren mehrere Variationen, die folgendem Schema entsprechen:

Produkttyp Variation (abgeleitet von Anforderung)	
B	Bezogen auf einen Variationstyp beschreibt eine Variation jeweils eine spezifische Ausprägung von Anforderungen. Eine Variation wird auch als *Instanz eines Variationstypen* bezeichnet.
Z	Benannte Zusammenfassung aller zu einer Variation gehörenden Anforderungen.
A	-
G	**Anforderung:** aggregiert alle Anforderungen, die zu einer Variation gehören. Wie bei der UND-Aggregation von Anforderungen (vergleiche Abschnitt 3.2.5) müssen für eine geplante oder realisierte Variation auch sämtliche aggregierten Anforderungen geplant oder realisiert sein.
B=Beschreibung, Z=Zweck, A=Attribute, G=Aggregationen	

Beispiel: Variationen

Mögliche Variationen hinsichtlich des Variationstyps Landessprache sind Englisch, Deutsch und Französisch.

Definition: Menge aller Variationen und zugehöriger Anforderungen

Sei $W \in VT_{S,t}$ ein Variationstyp eines Produktsystems S zum Zeitpunkt $t \in T$. $I_{W,t} \subseteq R_{W,t}$ bezeichnet die Menge aller Variationen für den Variationstyp W. Die zu einer gegebenen Variation $i \in I_{W,t}$ aggregierten Anforderungen werden mit $R_{i,t} \subseteq R_{W,t}$ bezeichnet.

Eine Reihe von Anforderungen eines Variationstyps W gehören sämtlichen Variationen gleichermaßen an:

Definition: Gemeinsamer Kern eines Variationstypen

Sei S ein Produktsystem, $t \in T$ ein Zeitpunkt und $W \in VT_{S,t}$ ein Variationstyp. Die Menge $K_{W,t}$ sei durch folgende Eigenschaften definiert:

$$K_{W,t} = \bigcap_{i \in I_{W,t}} R_{i,t}$$

Die Menge $K_{W,t}$ wird als gemeinsamer Kern hinsichtlich des Variationstyps *W oder kurz als* Kern von *W bezeichnet.*

Mit dem Kern werden Anforderungen identifiziert, die für unterschiedliche Variationen eines Variationstypen identisch sind. Diese können einmalig für einen Variationstypen entwickelt und dann in jeder Variation wiederverwendet werden. Dadurch wird Doppel-

arbeit vermieden, die beispielsweise dann auftreten würde, wenn unterschied liche Variationen von verschiedenen Entwicklergruppen realisiert werden würden, ohne Gemei nsamkeiten vorher zu identifizieren.

Beispiel: Gemeinsamer Kern

Moderne Betriebssysteme unterstützen Druckerfunktionen für verschiedene Druckertypen. Diese unterscheiden sich beispielsweise hinsichtlich Papiereinzugssteuerung, Zweiseitendruck, Druckersprache usw. Der gemeinsame Kern beim Drucken ist das einheitliche Aussehen einzelner Seiten, unabhängig welcher Druckertyp verwendet wird. Microsoft Windows realisiert dies beispielsweise durch eine interne Druckersprache für das Layout von Ausdrucken, wobei Spezifika durch unterschiedliche Druckertreiber unterstützt werden.

Der Kern identifiziert Anforderungen, die allen Variationen eines Variationstypen gle ichermaßen gemeinsam sind. Damit ist jedoch noch nicht ausgeschlossen, dass ein Teil der Variationen weitere Anforderungen gemeinsam hat. Beispielsweise sind unterschie dliche Drucker eines einzigen Druckerherstellers häufig über obige Gemeinsamkeiten hinaus auch in best immten Ansteuerungsbefehlen identisch. Wiederverwendbare Anfo rderungen sollen in den späteren Entwicklungsphasen leicht auffindbar sei n. Daher wird methodisch festgelegt, unterschiedliche Variationen eines Variationstypen nur im Kern gemeinsame Anforderung en haben. Dies wird durch folgendes Axiom au sgedrückt:

Entwicklungsaxiom: Partition des Kerns und der Variationen

Sei S ein Produktsystem, $t \in T$ *ein Zeitpunkt und* $W \in VT_{S,t}$ *ein Variationstyp. Für den Kern* $K_{W,t}$ *und die restlichen Anforderungen der Variationen* $R_{i,t} \setminus K_{W,t}$ *von W wird gefordert:*

(1) $\forall i, j \in I_{W,t}, i \neq j : \left(R_{i,t} \setminus K_{W,t}\right) \cap \left(R_{j,t} \setminus K_{W,t}\right) = \varnothing$,

(2) $\bigcup_{i \in I_{W,t}} R_{i,t} \cup K_{W,t} = R_{W,t}$.

Da trivialerweise auch

(3) $\forall i \in I_{W,t} : (R_{i,t} \setminus K_{W,t}) \cap K_{W,t} = \varnothing$ *gilt,*

bilden damit der Kern und die restlichen Anforderungen eine Partition.

Für das Beispiel des Herstellers unterschiedlicher Drucker ist damit konsequenterwe ise die Ableitung eines eigenständigen Variationstypen erforderlich.

Realisierung und Aktivierung von Variationen

Um spezifischen Anforderungen einzelner Kundengruppen nachkommen zu können, kann pro Variationstyp immer hö chstens eine einzige Variation aktiv sein. Nach den Definitionen zur Aktivierung von Anforderungen (vgl. Abschnitt 4.2.1) wird folgendes Axiom gefordert:

Laufzeitaxiom: Aktivierungsregel von Variationen

Sei W *ein Variationstyp eines Produktsystems S und* $i \in I_{W,\tau}$ *eine zugehörige Variation. Dann gilt zu einem beliebigen Zeitpunkt* $\tau \in T$ *:*

$active_S(i,\tau) \Rightarrow (\forall r \in R_{i,\tau} : active_S(r,\tau)) \wedge (\forall r \in R_{W,\tau} \setminus R_{i,\tau} : inactive_S(r,\tau))$

Die Aktivierung von i *erfordert, dass alle aggregierten Anforderungen* $R_{i,\tau}$ *ebenso aktiv sind. Alle anderen Anforderungen des Variationstyps sind inaktiv.*

Es ist auch möglich, dass keine Variation eines Variationstypen aktiv ist. In diesem Fall ist zum gegebenen Zeitpunkt keine der zu einem Variationstypen gehörigen Anforderungen erfüllt. Für die Realisierung von Variationen gelten die gleichen Axiome wie für Anforderungen.

Klassen von Variationstypen

In der Literatur über Produktlinien (vgl. z.B. [KCH+90, HLS+98]) wird häufig zwischen *widersprüchlichen Variationen* (oder *alternativen Variationen*) und *optionalen Variationen* unterschieden. In den folgenden Abschnitten wird kurz erläutert, wie sich diese im hier vorgestellten Modell von Variationstypen und Variationen widerspiegeln.

Von einem w*idersprüchlicher Variationstyp* wird gesprochen, wenn sich unterschiedliche Variationen gegenseitig ausschließen. Dies ist der Fall, wenn jeweils zwischen zwei Variationen eines Variationstyps widersprüchliche Mengen von Anforderungen (vgl. Definition in Abschnitt 4.2.1) bestehen. Dadurch können die beiden Variationen nicht gleichzeitig aktiviert werden. Durch die Aktivierungsregel sind Variationstypen in dieser Arbeit ausschließlich widersprüchlich.

Widersprüchliche Variationstypen werden nicht nur zur Modellierung von Anforderungen verschiedener Kundentypen verwendet. Sie dienen auch zur Einhaltung von Kompatibilitätsanforderungen neuer Softwareversionen. Neue Softwareversionen können Anforderungen realisieren, die Anforderungen in älteren Versionen widersprechen. Wegen der Kompatibilitätsforderung werden alte Anforderungen nicht ersatzlos gestrichen. Stattdessen kann ein widersprüchlicher Variationstyp eingeführt werden. Welche der beiden Anforderungen erfüllt sein soll, kann dann der Benutzer entscheiden.

Mit dem *optionalen Variationstyp* können über bereits aktivierte Anforderungen hinaus zusätzliche Anforderungen aktiviert werden. Im Unterschied zum widersprüchlichen Variationstyp ist hier keine Deaktivierung anderer Anforderungen nötig. Trotz des auf den ersten Blick existierenden Unterschieds zum widersprüchlichen Variationstypen ist dies lediglich ein Spezialfall hiervon. Ein optionaler Variationstyp kann als widersprüchlicher Variationstyp mit genau zwei Variationen angesehen werden. Die spezifischen Anforderungen der ersten Variation entsprechen genau den zusätzlichen Anforderungen. Dagegen enthält die zweite Variation keine eigenen spezifischen Anforderungen. Deren Anforderungen entsprechen also genau dem gemeinsamen Kern. Der häufig in graphischen Spezifikationen verwendete optionale Variationstyp ist somit aus rein formaler Sicht kein Sonderfall. Für optionale Variationstypen sind somit keine weiteren Axiome nötig.

Der Schnitt zwischen den zwei unterschiedlichen Variationen eines optionalen Variationstypen hat jedoch hohe praktische Relevanz. So werden häufig ergänzende Variationen mit ihren Spezifika als eigenständige Produkte entwickelt. Beispielsweise wird diese Art der Aufteilung in der bei der Firma SIEMENS entwickelten Engineering Software SIMATIC für die Automatisierungstechnik verwendet. Die Au fteilung besteht aus einem zentralen Produkt, das den Kern der Anforderungen darstellt und verschiedenen Optionsprodukten, die jeweils als ergänzende Variation eines eigenen Variationstypen zu sehen sind. In diesem Fall bilden sogar unterschiedliche Variationstypen einen gemei nsamen Kern.

Korrelation zwischen Variationstypen und Anforderungen

Ähnlich wie Anforderungen korrelieren können, kann auch eine Korrelation zwischen Variationstypen und Anforderungen bestehen. Während die Korrelation zwischen Anfo rderungen sich lediglich unmittelbar auf betroffene Anforderungen selbst auswirkt, en tstehen aus der Korrelation mit Variationstypen neue Anforderungen und haben somit eine erhebliche Auswirkung auf mögliche Realisierungsaufwä nde.

Die Korrelation zwischen Variationstypen und Anforderungen wird durch folgende A ssoziation dargestellt:

Assoziationstyp korreliert (Variationstyp - Anforderung)	
B	Ein Variationstyp besteht aus Variationen, die wie allgemeine A nforderungen andere Anforderungen in gleicher Weise beeinflussen können. Die *beeinflusst*-Assoziation zwischen einem Variationstyp und einer Anforderung drückt aus, dass seine Variationen nicht unabhängig von einer Anforderung sind. Jede einzelne Variation eines Variationstypen konkretisiert die referenzierte Anfo rderung unterschiedlich. Davon bildet jede Konkretisierung eine eigene aus der A nforderung und dem referenzierenden Variationstypen *abgeleitete Variation.* Abgeleitete Variation müssen eigens realisiert werden. Die aus einer beeinflusst - Assoziation abgeleiteten Variationen bilden gemeinsam einen *abgeleiteten Variationstypen*.
Z	Variationstypen können mit mehreren Anforderungen korrelieren. Die korreliert-Assoziation zwischen Variationstyp und Anforderung dient der kompakten Spezifikation abgeleiteter Variationen in Überblicksspezifikationen. A nstatt diese jeweils explizit zu spezifizieren, wird lediglich die Ursache abgeleiteter Variationen dokumentiert. Dies verhindert eine Explosion von zu spezifizierenden Anforderu ngen, die durch die Kombination aller beeinflussenden Variationen mit den refere nzierten Anforderungen entstehen würde. In detaillierten Spezifikat ion zur Unterstützung der Entwicklung sollte aus Gründen der Verfolgbarkeit jede abgeleit ete Variation auch einzeln spezifiziert werden (vgl. auch Kapitel 5).
A	-
B=Beschreibung, Z=Zweck, A=Attribute	

Beispiel: Korrelation eines Variationstypen mit einer Anforderung

Ein sehr einfaches Beispiel ist die korreliert-Assoziation zwischen einem Variationstypen Farbausgabe und einer Anforderung zur Darstellung einer bestimmten Fehlermeldung. Die Assoziation sagt aus, dass die Fehlermeldung in Abhängigkeit

von der Aktivierung der Farbe unterschiedlich dargestellt wird. Jede Kombination aus Farben und der Fehlermeldung ist eine eigenständige Anforderung. Zur Illustration ist in Abbildung 11 ist eine graphische Spezifikation des Einflusses des Variationstype Farbausgabe auf die Anforderung Fehlermeldung dargestellt. Es handelt sich bei dieser Darstellung um eine UML-Erweiterung von Klassendiagrammen durch Stereotypen. Dunkel umrandete Trapeze repräsentieren hier Variationstypen, einfache Trapeze Variationen. Rechtecke mit abgerundeten Ecken sind Anforderungen. In der Abbildung ist dargestellt, dass der Variationstyp Farbausgabe mit drei Variationen rot, blau und gelb mit der Fehlermeldung korreliert.

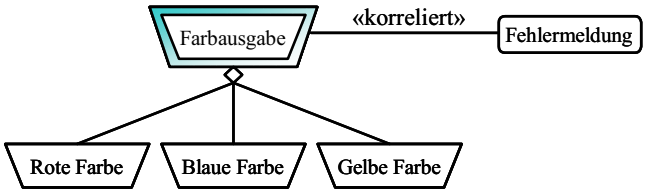


Abbildung 11 – Korrelation eines Variationstypen mit einer Anforderung

Definition: Korrelation zwischen Variationstyp und Anforderung

Sei $W \in VT_{S,t}$ *ein Variationstyp,* $I_{W,t}$ *die Menge aller Variationen von* W *und* $r \in R_{S,t}$ *eine Anforderung für ein Produktsystem S zum Zeitpunkt* $t \in T$. Zwischen Variationstyp und Anforderung *besteht eine Korrelation gdw. gilt:*

$\forall i \in I_{W,t} : correl(i, r)$

Die Korrelation definiert eine Relation aus $VT_{S,t} \times R_{S,t}$ *und wird mit dem Prädikat* $correl(W, r)$ *dargestellt.*

Definition: Aus der Korrelation Abgeleitete Variationen

Sei $W_1 \in VT_{S,t}$ *ein Variationstyp und* $r \in R_{S,t}$ *eine Anforderung für ein Produktsystem S zum Zeitpunkt* $t \in T$*, für die gilt* $correl(W_1, r)$. *Durch diese Korrelation gibt es einen weiteren Variationstypen* $W_2 \in VT_{S,t}$, *dessen Variationen* $i_2 \in I_{W_2}$ *mit* $i_1 \in I_{W_1}$ *und* $r \in R_{S,t}$ *eine Relation aus* $I_{W_1,t} \times R_{S,t} \times I_{W_2,t}$, *(i. Z.* $abgel(i_1, r, i_2)$*), mit folgender Eigenschaft bilden:*

$$correl(W_1, r) \Leftrightarrow$$
$$\forall i_1 \in I_{W_1} \exists i_2 \in I_{W_2} : abgel(i_1, r, i_2) \wedge$$
$$\forall i_2 \in I_{W_2} \exists i_1 \in I_{W_1} : abgel(i_1, r, i_2)$$

W_2 *wird als* aus der Korrelation $correl(W_1, r)$ abgeleiteter Variationstyp, *seine Variationen* $i_2 \in I_{W_2}$ als aus der Korrelation $correl(W_1, r)$ abgeleitete Variationen *bezeichnet.*

Durch die Korrelation eines Variationstypen $W_1 \in VT_{S,t}$ zu einer Anforderung *r* gibt es für jede Variation $i_1 \in W_1$ eine abgeleitete Variation $i_2 \in W_2$. Umgekehrt darf der abgeleitete Variationstyp keine weitere Variation enthalten. Für obiges Beispiel darf der abgeleitete Variationstyp nur die erwähnten roten, blauen und gelben Fehlermeldungen, jedoch

nicht eine grüne Fehlermeldung aufweisen. Abgeleitete Variationen werden im Requirements Engineering nicht explizit spezifiziert.

Beispiel: Aus der Korrelation Abgeleitete Variationen

Im obigen Beispiel entsteht aus der Korrelation des Variationstyps Farbausgabe mit der Fehlermeldung ein abgeleiteter Variationstyp „Farbige Fehlermeldung" mit Variationen „rote Fehlermeldung", „blaue Fehlermeldung" und „gelbe Fehlermeldung".

Mit der Korrelation eines Variationstypen zu einer Anforderung entsteht auch eine Abhängigkeit hinsichtlich der Realisierung und Aktivierung abgeleiteter Variationen. Diese drücken folgende Axiome aus.

Entwicklungsaxiom: Planung abgeleiteter Variationen

Seien $W_1, W_2 \in VT_{S,t}$ *Variationstypen und* $r \in R_{S,t}$ *eine Anforderung für ein Produktsystem S zu einem gegebenen Zeitpunkt* $t \in T$. *Für eine aus der Korrelation* $correl(W_1, r)$ abgeleitete Variation $i_2 \in I_{W_2}$ *mit* $abgel(i_1, r, i_2)$ *wird gefordert:*

$$i_1 \in R_{S,t}^{pl} \wedge r \in R_{S,t}^{pl} \Rightarrow i_2 \in R_{S,t}^{pl} \vee i_2 \in R_{S,t}^{r}$$

Werden eine Variation und eine Anforderung zur Realisierung geplant, so folgt, dass daraus abgeleitete Variationen geplant oder realisiert sind. Wird im obigen Beispiel die rote Farbausgabe und die Fehlermeldung geplant, so muss auch die rote Fehlermeldung geplant werden.

Den Zusammenhang der Realisierung zueinander in Korrelation stehender Variationstypen und Anforderungen und daraus abgeleiteten Variationen drückt anschließende Definition aus. Dazu wird eine Hilfsdefinition benötigt:

Hilfsdefinition: Aus einer Variation abgeleitete Variationen

Sei $W \in VT_{S,t}$ *ein Variationstyp, und* $i \in I_W$ *eine Variation für ein Produktsystem S zu einem gegebenen Zeitpunkt* $t \in T$. *Sei*

$$R_{infl,t} = \left\{ r \in R_{S,t} \middle| correl(W, r) \right\}$$

die Menge aller mit W *korrelierten Anforderungen. Für eine Anforderung* $r_j \in R_{infl,t}$ *sei mit* $W_j \in VT_{S,t}$ *der aus* $correl(W, r_j)$ *abgeleitete Variationstyp bezeichnet. Dann bezeichnet*

$$W_{abgel,i,t} = \left\{ i_{abg} \middle| abgel(i, r_j, i) \wedge r_j \in R_{infl,t} \wedge i_{abg} \in I_{W_j} \right\}$$

die Menge aller aus i *abgeleiteten Variationen.*

Definition: Realisierung in Korrelation stehender Variationen und Anforderungen

Seien $W, W_{abg} \in VT_{S,t}$ *Variationstypen, und* $r \in R_{S,t}$ *eine Anforderung für ein Produktsystem S zu einem gegebenen Zeitpunkt* $t \in T$ *mit* $correl(W, r)$.

- *Eine in Korrelation stehende Anforderung r gilt als* realisiert*, gdw. für jede Variation* $i \in I_W$ *mit* $\exists t_0 \leq t : i \in R^{pl}_{S,t_0}$ *und aus deren Korrelation abgeleitete Variation* $i_{abg} \in I_{W_{abg}}$ *mit* $abgel(i, r, i_{abg})$ *gilt:*

  $i_{abg} \in R^r_{S,t}$

- *Die Variation* $i \in I_W$ *gilt als* realisiert*, gdw. für jede aus ihr abgeleiteten Variationen* $i_{abg} \in W_{abgel,i,t}$ *mit* $\exists t_0 \leq t : i \in R^{pl}_{S,t}$ *gilt:*

  $i_{abg} \in R^r_{S,t}$

Wenn ein Variationstyp in *korreliert*-Assoziation zu einer Anforderung steht, so werden Variationen nicht eigenständig realisiert. Sie stehen in einer engen Abhängigkeit mit der assoziierten Anforderung. Variationen gelten dann als realisiert, wenn sämtliche aus ihr assoziierten abgeleiteten Variationen realisiert.

Beispiel: Realisierung in Korrelation stehender Variationen und Anforderungen

Im obigen Beispiel wurde nur die Korrelation des Variationstyps Farbausgabe auf Fehlermeldungen dargestellt. Hier ist die Farbe rot dann realisiert, sobald die rote Fehlermeldung realisiert ist. Die Fehlermeldung hingegen gilt dann als realisiert, wenn sämtliche aus ihr abgeleiteten Variationen realisiert sind. Wird rote und blaue Farbausgabe geplant, müssen die rote und die blaue Fehlermeldung realisiert werden, um die Fehlermeldung zu realisieren.

Diese Abhängigkeiten sind zur konsistenten Verfolgung der Realisierung abgeleiteter Variationen beziehungsweise in mit Variationen korrelierenden Anforderungen wesentlich. Analog zur Realisierung bestehen auch Zusammenhänge für die Aktivierung mit Variationstypen korrelierenden Anforderungen:

Definition: Aktivierungsregel abgeleiteter Variationen

Sei $W \in VT_{S,t}$ *ein Variationstyp,* $i \in I_W$ *eine Variation und* $r \in R_{S,t}$ *eine Anforderung für ein Produktsystem S zu einem gegebenen Zeitpunkt* $t \in T$ *mit* $correl(W, r)$.

- *Die Variation i ist genau dann* aktiviert*, wenn alle aus ihr abgeleiteten Variationen* $i_{abg} \in W_{abgel,i,t}$ *aktiviert sind.*
- *Die Anforderung r ist genau dann* aktiviert*, wenn eine der abgeleiteten Variationen aktiv ist. In formaler Notation ausgedrückt:*

  $active_S(r,t) \Leftrightarrow \exists i_1 \in I_{W_1} : \exists i_2 \in I_{W_2} : abgel(i_1, r, i_2) \wedge active_S(i_2, t)$.

Korrelation mehrerer Variationstypen mit einer Anforderung

Dieser ohnehin schon komplizierte Zusammenhang wird noch schwieriger, wenn eine gegebene Anforderung gleichzeitig mit mehreren Variationstypen korreliert wird. Dieser Fall wird nun erörtert.

Beispiel:

An ein Finanzbuchhaltungsmodul wird die Anforderung gestellt, Abschreibungen für betriebliche Investitionen zu unterstützen. Hierzu gibt es unterschiedliche Abschreibungsverfahren, die abhängig von Landesgesetzen sind. Da das Finanzmodul international einsetzbar sein soll, muss es gleichzeitig in verschiedenen Landessprachen realisiert werden. Sowohl die Abschreibungsverfahren als auch die Landessprachen beeinflussen die Anforderung nach Abschreibung. Es ist also im allgemeinsten Fall notwendig, für jedes Abschreibungsverfahren Ein-/ Ausgaben in jeder Sprache zu realisieren. In Abbildung 12 ist das Beispiel graphisch dargestellt. Aus dieser mehrfachen Korrelation ergeben sich insgesamt neun abgeleitete Variationen (Degressive Abschreibung in englisch, in deutsch und in französisch, progressive Abschreibung in englisch, in deutsch und in französisch und lineare Abschreibung in englisch, in deutsch und in französisch).

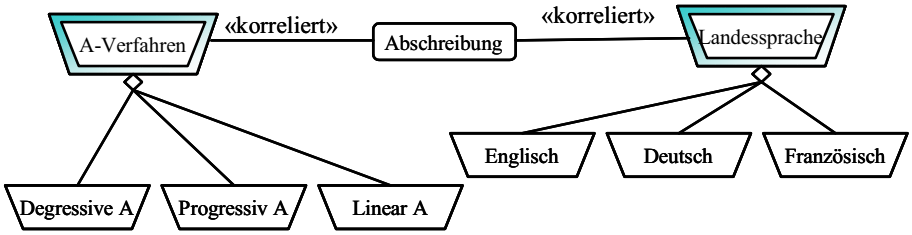


Abbildung 12 – Mehrfache Korrelation von Variationstypen zu Anforderungen

Dieses Beispiel zeigt, dass nicht für jede Variation isoliert ein neuer abgeleiteter Variationstyp entsteht, sondern verschiedene Variationstypen miteinander kombiniert werden.

Bei einer mehrfachen Korrelation ist nicht immer jede beliebige Kombination von Variationen unterschiedlicher Variationstypen möglich. So erfordert in manchen Fällen die Aktivierung einer Variation eines Variationstypen die gleichzeitige Aktivierung bestimmter Variationen eines anderen Variationstypen. Dies drückt die *erwartet*-Assoziation zwischen Variationen aus:

Assoziationstyp erwartet (Variation → Variation)	
B	Diese Assoziation drückt aus, dass die Aktivierung einer Variation $i_1 \in W_1$ eines Variationstypen $W_1 \in VT_{S,t}$ die Aktivierung einer dazu assoziierten Variation $i_2 \in W_2, W_2 \in VT_{S,t}$ erwartet. Dies bedeutet, dass genau eine der Variationen aus W_2, die gleichzeitig so mit i_1 assoziiert sind, aktiviert werden muss.
Z	Dient zur Spezifikation von Mindestanforderungen für ein konsistentes System bei der Aktivierung bestimmter Variationen zu erhalten.
A	-
B=Beschreibung, Z=Zweck, A=Attribute	

Beispiel:

Ein Finanzbuchhaltungsmodul soll für verschiedene Länder jeweils die landesspezifischen Gesetze berücksichtigen. Aufgrund dieser Gesetze sind jeweils nur eine bestimmte Auswahl der möglichen Abschreibungsverfahren zulässig. Für das Modul heißt dies dann, dass die Aktivierung einer bestimmten Ländervariante die Aktivierung eines der zulässigen Abschreibungsverfahrens erwartet. In Deutschland ist beispielsweise keine progressive Abschreibung möglich.

Definition: Erwartung zwischen Variationen und zugehörige Aktivierungsregel

Seien $W_1, W_2 \in VT_{S,t}$ *Variationstypen und* $i_1 \in I_{W_1}, i_2 \in I_{W_2}$ *Variationen eines Produktsystems S zu einem Zeitpunkt* $t \in T$.

Die Erwartung *zwischen den Variationen definiert eine Relation aus* $I_{W_1} \times I_{W_2}$ *(i. Z.* $expect(i_1, i_2)$*) mit folgender Eigenschaft:*

$$\forall \tau \in T : active_S(i_1, \tau) \Rightarrow (\exists k \in \left\{ j \in I_{W_2} \middle| expect(i_1, j) \right\} : active_S(k, \tau))$$

Aus der Aktivierung einer Variation $i_1 \in I_{W_1}$ *folgt also die Aktivierung einer der erwarteten Variationen aus* I_{W_2}.

Die schwache Erwartung *zwischen den Variationen definiert eine Relation aus* $I_{W_1} \times I_{W_2}$ *(i. Z.* $weakexpect(i_1, i_2)$*) mit folgender Eigenschaft:*

$$\forall \tau \in T : active_S(i_1, \tau) \Rightarrow$$
$$(\exists k \in \left\{ j \in I_{W_2} \middle| weakexpect(i_1, j) \right\} : active_S(k, \tau)) \vee$$
$$\forall k \in I_{W_2} : inactive_S(k, \tau)$$

Ist eine Variation $i_1 \in I_{W_1}$ *aktiviert, so kann nur eine der schwach erwarteten Variationen aus* I_{W_2} *aktiv sein. Im Gegensatz zur Erwartung ist es auch möglich, dass keine der schwach erwarteten Anforderungen aktiv ist.*

Umgekehrt schließen sich manche Variationen verschiedener Variationstypen gegenseitig aus. Dies wird durch folgende Assoziation ausgedrückt:

Assoziationstyp schließt_aus (Variation - Variation)	
B	Diese Assoziation drückt aus, dass zwei zueinander assoziierte Variationen nicht gleichzeitig aktiviert sein dürfen.
Z	Dient der Festlegung einer Konsistenzbedingung für die richtige Konfiguration des Systems.
A	-
B=Beschreibung, Z=Zweck, A=Attribute	

Auch diese Assoziation wird formal wie folgt beschrieben:

Definition: Ausschluss zwischen Variationen und zugehörige Aktivierungsregel

Seien $W_1, W_2 \in VT_{S,t}$ *Variationstypen und* $i_1 \in I_{W_1}, i_2 \in I_{W_2}$ *Variationen eines Produktsystems S zu einem Zeitpunkt* $t \in T$ *Der* Ausschluss *zwischen den Variationen definiert eine Relation aus* $I_{W_1} \times I_{W_2}$ *(i. Z.* $exclude(i_1, i_2)$*) mit folgender Eigenschaft:*

$\forall \tau \in T : active_S(i_1, \tau) \Rightarrow inactive_S(i_2, \tau) \wedge active_S(i_2, \tau) \Rightarrow inactive_S(i_1, \tau)$.

Die beiden Variationen dürfen nicht gleichzeitig aktiv sein.

Beispiel:

In Umkehrung der erwartet-Assoziation kann die Beziehung von Ländervariationen zu Abschreibungsverfahren in einem Finanzbuchhaltungsmodul derart dargestellt werden, dass für bestimmte Länder verbotene Verfahren als Ausschluss dargestellt werden.

Durch die Erwartungs- und die Ausschluss-Assoziationen werden mögliche Kombinationen der Aktivierung verschiedener Variationen eingeschränkt. In Anhang E wird gezeigt, dass der Ausschluss zweier Variationen auf die schwache Erwartung zurückführbar ist. Daher beschreibt folgende Definition sämtliche erlaubte Kombinationen verschiedener Variationen.

Hilfsdefinition: Erlaubte Kombinationen verschiedener Variationen

Seien $W_1, ..., W_n \in VT_{S,t}$ *für* $n \geq 2$ *unterschiedliche Variationstypen. Die* erlaubte Kombination verschiedener Variationen *bestimmt, welche Kombinationen dieser Variationen gleichzeitig aktiviert sein dürfen. Die Kombinationen werden als Tupel einer n-dimensionalen Menge* $allowed(W_1, ..., W_n) \subseteq I_{W1} \times ... \times I_{W_n}$ *dargestellt. Es gilt:*

$(i_1, ..., i_n) \in allowed(W_1, ..., W_n)$ *mit* $k \in \{1, ..., n\} : i_k \in I_{W_k}$ *genau dann wenn:*
$\forall l, m \in \{1, ..., n\} \wedge l \neq m :$
$((\exists j \in I_{W_m} : expect(i_l, j)) \Rightarrow expect(i_l, i_m)) \wedge$
$((\exists j \in I_{W_m} : weakexpect(i_l, j)) \Rightarrow weakexpect(i_l, i_m))$

Aus der Korrelation mehrerer Variationstypen zu einer Anforderung entsteht aus jeder erlaubten Kombination verschiedener Variationstypen eine abgeleitete Variation:

Definition: Kombinierter abgeleiteter Variationstyp

Seien $W_1, ..., W_n \in VT_{S,t}$ *für* $n \geq 2$ *Variationstypen mit* $\forall j \in \{1, ..., n\} : correl(W_j, r)$ *für eine Anforderung* $r \in R_{S,t}$. *Ein* kombinierter abgeleiteter Variationstyp $W_a \in VT_{S,t}$ *ist ein Variationstyp mit folgender Eigenschaft:*

$\forall (i_1, ..., i_n) \in allowed(W_1, ..., W_n) : \exists i_a \in I_{W_a} : abgel(i_1, r, i_a) \wedge ... \wedge abgel(i_n, r, i_a)$

Beispiel:

Für das Finanzbuchhaltungsbeispiel sind dies alle erlaubten Abschreibungsverfahren unterschiedlicher Länder.

Der kombinierte Variationstyp enthält für jede erlaubte Kombination von Variationen der beeinflussenden Variationstypen eine abgeleitete Variation. Aus kombinatorischen Gründen kann der kombinierte Variationstyp damit bis zu $\left|I_{W_1} \times ... \times I_{W_n}\right|$ abgeleitete Variationen enthalten. Abhängig von der Art der Variationstypen und der Anforderung können daraus erhebliche Aufwände für die Realisierung dieser großen Anzahl abgeleiteter Variationen entstehen. Um Entwicklungsaufwände zu reduzieren, sollte gegebenenfalls auf die Realisierung mancher Variationen verzichtet werden. Zur Unterstützung der richtigen Auswahl zu realisierender Variationen wird der Assoziationstyp gewünschter Variationen eingeführt.

Assoziationstyp gewünscht hinsichtlich einer Anforderung (Variation – Variation)	
B	Eine gewünscht-Assoziation kann zwischen zwei Variationen $i_1 \in I_{W_l}$, $i_2 \in I_{W_m}$ mit $l,m \in \{1,...,n\}$ und $l \neq m$ hinsichtlich einer mit $W_1,...,W_n \in VT_{S,t}$ korrelierenden Anforderung $r \in R_{S,t}$ bestehen. Die Assoziation definiert eine Teilmenge $I_{gew} \subset I_{W_a}$ von Variationen des aus dieser mehrfachen Korrelation kombinierten abgeleiteten Variationstypen $W_a \in VT_{S,t}$. Für diese Teilmenge gilt: $\forall i_a \in I_{Gew} : abgel(i_1, r, i_a) \wedge abgel(i_2, r, i_a)$ Sie enthält alle in Kombination von i_1 und i_2 abgeleiteten Variationen. Die gewünscht-Assoziation drückt aus, dass die Realisierung von Variationen aus I_{gew} bevorzugt werden soll, wenn die Realisierung aller Variationen aus I_{W_a} zu aufwändig ist. In diesem Fall sollen nur Variationen aus I_{gew} realisiert werden Im Gegensatz zur erwartet-Assoziation folgt aus dieser Assoziation keine Bedingung an die Aktivierung der Variationen.
Z	Hilfestellung zur Reduktion der Anzahl abgeleiteter Variationen. Durch die Assoziation werden bevorzugte abgeleitete Variationen festgelegt.
A	**Gewicht:** gegebenenfalls können Gewichte angegeben werden, die unterschiedliche Wünsche noch einmal untereinander priorisieren.
B=Beschreibung, Z=Zweck, A=Attribute	

Beispiel:

Angenommen, ein Finanzbuchhaltungsmodul wird in mehreren länderspezifischen Variationen realisiert. Wie jedes Modul mit Ein- und Ausgabe korreliert es nicht nur mit unterschiedlichen Landesgesetzen, sondern soll auch in verschiedenen Sprachen realisiert werden. Zur Reduktion der Kombination unterschiedlicher Variationen könnte der Standardsoftwarehersteller festlegen, dass lediglich die Landessprachen des spezifischen Landes gewünscht sind. Beispielsweise für eine Schweizer Ländervariante wäre dies Deutsch, Italienisch und Französisch oder für eine Deutsche Ländervariante lediglich Deutsch.

Eine weitere Möglichkeit zur Beschränkung der Vielfalt von Landessprachen wäre auch zum Beispiel, dass sämtliche Fehlermeldungen für Abstürze lediglich in englischer Sprache erfolgen sollen, da diese sowieso nur von den Softwareentwickler relevante Informationen enthalten.

Werden nur gewünschte Kombinationen von Variationen umgesetzt, so können Realisierungsaufwände reduziert werden. Ein Wunsch schließt nicht die Realisierung weiterer Sprachen aus. Sollte beispielsweise beim Einsatz der Software in internationalen Firmen auch die Sprache Englisch benötigt werden, können auch sämtliche Länderanpassungen auch in der Landessprache Englisch realisiert werden.

Korrelation zwischen Variationstypen und Aspekten

Wie zwischen Aspekten wird auch eine *korreliert*-Assoziationen zwischen Variationstypen und Aspekten definiert.

Assoziationstyp korreliert (Variationstyp - Aspekt)	
B	Diese Assoziation ist analog zur Assoziation zwischen Aspekten eine abgeleitete Beziehung. Steht ein Variationstyp in korreliert-Assoziation zu einem Aspekt, so existiert innerhalb des Aspekts mindestens eine Anforderung mit dem Variationstypen korreliert. Korrelieren verschiedene Variationstypen mit einem Aspekt gleichzeitig, so muss es sich nicht unmittelbar um eine mehrfache Korrelation (vgl. letzter Unterabschnitt) zu einer einzelnen Anforderung handeln.
Z	Ähnlich zur *korreliert*-Assoziation für Aspekte dient diese Assoziation der übersichtlichen Darstellung einer Korrelation eines Variationstypen zu Gruppen von Anforderungen.
A	-
B=Beschreibung, Z=Zweck, A=Attribute	

Die korreliert-Assoziation zwischen einem Variationstypen und einem Aspekt wird mit Hilfe der korreliert-Assoziation zwischen einem Variationstypen und einer Anforderung definiert.

Definition: Korrelation zwischen Variationstypen und Aspekten

Sei S ein Produktsystem zu einem gegebenen Zeitpunkt $t \in T$. *Zwischen einem Variationstyp* $W \in VT_{S,t}$ *und einem Aspekt* $A \in ASP_{S,t}$ *besteht eine Korrelation (i. Z.* $correl(W, A)$*) gdw. gilt:*

$\exists r \in A : correl(W, r)$.

4.4.3 Systemsicht

In Kapitel 1 und Abschnitt 4.2 wurden bereits Beispiele von Produktsystemen vorgestellt. Dazu ist eine systematische Aufteilung der Anforderungen auf einzelne Systemmodule erforderlich. Die Systemsicht wird im Rahmen der Definition der Systemsicht in Abschnitt 7.5.2 erarbeitet. Sie ist Planungsgrundlage für die Definition der Entwicklungssicht.

Charakterisierung eines Produktsystems

Bevor einzelne Entwicklungsprodukte der Systemsicht vorgestellt werden, soll wird ein Produktsystem genauer charakterisiert. Wesentliche Elemente eines Produktsystems sind miteinander gekoppelte Systemmodule:

Definition: Systemmodul *Ein Systemmodul ist die gröbste Zerlegungseinheit eines Produktsystems. Systemmodule sind für die Erfüllung an sie gestellter Anforderungen verantwortlich. Dazu können sie diese selbst realisieren oder die Realisierung mit einer Forderung über Modulschnittstellen an andere Systemmodule delegieren.*

Definition: Modulschnittstelle

Eine Modulschnittstelle dient zur Dokumentation von Abhängigkeiten zwischen Systemmodulen. Diese Abhängigkeiten werden durch Anforderungen repräsentiert, die von einer Menge von Systemmodulen gestellt werden und von einem anderen Systemmodul realisiert werden.

Beispiel: Systemmodul und Modulschnittstelle

Systemmodule von Microsoft Office sind beispielsweise Word, Excel oder Draw. Sowohl an Word und Excel werden Anforderungen zur Unterstützung von Zeichnungen gestellt. Diese werden über eine Modulschnittstelle an Draw delegiert. Für einen Anwender von Word oder Excel ist Draw nicht sichtbar.

Definition: Menge aller Systemmodule und Modulschnittstellen

Systemmodule werden als Elemente einer Menge P und Modulschnittstellen als Elemente einer Menge M repräsentiert.

Systemmodule eines Produktsystems sind miteinander gekoppelt:

Definition: Von Systemmodulen geforderte und erfüllte Modulschnittstellen

Sei $p \in P$ ein Systemmodul, das über eine endliche Teilmenge $M_p^{ford} \subseteq M$ der Modulschnittstellen die Erfüllung von Anforderungen durch andere Systemmodule fordert. Dann wird M_p^{ford} als Menge der von p geforderten Modulschnittstellen bezeichnet.

Sei umgekehrt $q \in P$ ein Systemmodul, das von einer endlichen Teilmenge $M_q^{erf} \subseteq M$ der Modulschnittstellen gestellte Anforderungen realisiert. Dann wird M_q^{erf}, als Menge der von q erfüllten Modulschnittstellen bezeichnet.

Definition: Kopplung zwischen Systemmodulen

Seien $p,q \in P$ *zwei Systemmodule,* M *die Menge aller Modulschnittstellen,* $M_p^{ford}, M_p^{erf}, M_q^{ford}, M_q^{erf} \subseteq M$ *wie in obiger Definition. Zwischen* p *und* q *besteht eine* Kopplung *(i. Z.* $worktogether(p,q)$*) gdw. gilt:*

$$worktogether(p,q) \Leftrightarrow (\exists m \in M : (m \in M_p^{ford} \wedge m \in M_q^{erf}) \vee (m \in M_q^{ford} \wedge m \in M_p^{erf}))$$

Zwei Systemmodule sind also miteinander gekoppelt, wenn eine Modulschnittstelle existiert, die eines der Systemmodule fordert und das andere erfüllt.

Basierend auf diesen Definitionen werden nun Produktsysteme genauer charakterisiert. In Abschnitt 4.2 wurde ein Produktsystem als eine Menge S definiert.

Definition: Produktsystem (2)

Ein Produktsystem S repräsentiert eine Teilmenge $S \subseteq P$ *von Systemmodulen mit folgenden Eigenschaften:*

(1) $\forall p,q \in S, p \neq q$: $worktogether^+(p,q)$. *(Dabei steht* $^+$ *für die transitive Hülle).*

(2) Anforderungen eines mit $worktogether(p,q)$ *gekoppelten Systemmoduls* $q \in S$ *an Systemmodul* $p \in S$ *werden bei der Weiterentwicklung von* p *berücksichtigt.*

Im Grunde müsste diese Menge $S \subset P$ als zeitbehaftet modelliert werden. Da jedoch das Hinzukommen neuer und das Verschwinden alter Systemmodule vergleichsweise selten im Vergleich zur Dynamik der Anforderungen ist, wird der Einfachheit halber entsprechender Zeitindex weggelassen.

Beispiel: Kein Produktsystem

Viele Softwareprodukte erfüllen Standardschnittstellen wie OLE [Bro95] oder CORBA [OMG98] und können darüber mit anderen Softwareprodukten gekoppelt werden. Beispielsweise werden damit von einer Anwendungssoftware häufig Betriebssystemdienste genutzt. Anwendungssoftware und Betriebssystem werden jedoch unabhängig voneinander weiterentwickelt. Insbesondere berücksichtigt das Betriebssystem Anforderungen der Anwendungssoftware nicht. Die beiden Softwareprodukte sind daher ***kein Produktsystem****.*

Die Kopplung spielt in der Entwicklung einzelner Softwareprodukte eines Produktsystems eine entscheidende Rolle. Diese wird im Entwicklungsprozess berücksichtigt. Die Kopplung beeinflusst die parallele Versionsentwicklung einzelner Systemmodule. Entsprechende Konsequenzen werden in Abschnitt 7.5.3 dargestellt.

Entwicklungsprodukte der Systemsicht

Dieser Unterabschnitt erläutert Entwicklungsprodukte der Systemsicht. Abbildung 13 gibt hierzu einen Überblick.

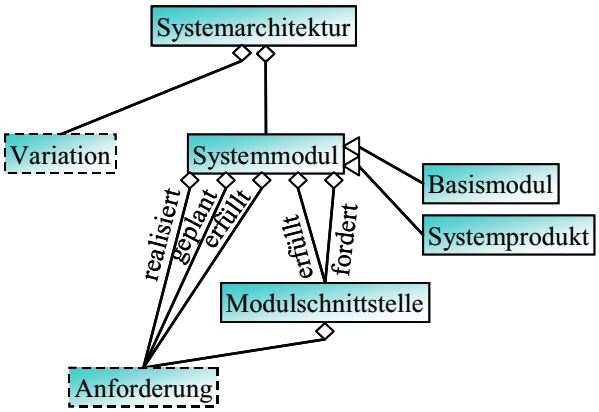


Abbildung 13 – Entwicklungsprodukte der Systemsicht

Folgendes Schema beschreibt ein Systemmodul $p \in S$ eines Produktsystems::

Produkttyp Systemmodul	
B	Gröbste Zerlegungseinheit eines Produktsystems (siehe Definition weiter oben in diesem Abschnitt).
Z	Die Zerlegung des Produktsystems in einzelne Systemmodule dient in erster Linie zur Beherrschung der Komplexität und der Erhöhung der Flexibilität der Entwicklung durch die Möglichkeit der weitgehend unabhängigen Entwicklung einzelner Produkte.
A	- **Leitlinie:** Beschreibung welche allgemeine Aufgabe das Systemmodul erfüllen soll. Diese dient der Unterstützung der Zuordnung von Anforderungen zu Systemmodulen und nötiger Kompetenzen. - **Ressourcen:** Für die Entwicklung zur Verfügung stehende Ressourcen, aufgeschlüsselt nach Kompetenzen.
G	- **Anforderung (erfüllt):** Verweis auf direkt an das Systemmodul gestellte Anforderungen. Diese müssen in dem Systemmodul realisiert werden, sobald eine Realisierungsentscheidung gefallen ist (vgl. Abschnitt 7.5.2 des Prozessmodells). - **Anforderung (geplant):** Verweis auf alle in dem Systemmodul zur Realisierung geplanten Anforderungen (vgl. anschließender Unterabschnitt zur Planung und Realisierung von Anforderungen). - **Anforderung (realisiert):** Verweis auf alle in dem Systemmodul realisierten Anforderungen (vgl. anschließender Unterabschnitt zur Planung und Realisierung von Anforderungen). - **Modulschnittstelle (fordert):** Verweis auf eine Modulschnittstelle, an das direkt an das Systemmodul gestellte Anforderungen delegiert werden. Das Systemmodul kann nur dann alle seiner direkten Anforderungen erfüllen, wenn delegierte Anforderungen durch ein anderes Systemmodul erfüllt werden. - **Modulschnittstelle (erfüllt):** Verweis auf eine Modulschnittstelle, deren Anforderungen durch das gegebene Systemmodul erfüllt werden sollen.

	- **Systemmodul-Version:** Aggregiert alle zu dem Systemmodul gehörenden Systemmodul-Versionen. Systemmodul-Versionen werden im nachfolgenden Abschnitt 4.4.4 behandelt.
B=Beschreibung, Z=Zweck, A=Attribute, G=Aggregationen	

Es gibt zwei Verfeinerungen von Systemmodulen:

- **Softwareprodukte:** Diese werden aus Sicht des Marketings (einzeln) auf dem Markt verkauft. Sämtliche geforderten Anforderungen müssen auf Softwareprodukte verteilt werden. Für das Marketing sind nur Softwareprodukte relevant.

- **Basismodule:** Diese sind nur für die Entwicklung relevant. Sie realisieren Anforderungen, die für mehrere Softwareprodukte gemeinsam existieren und von diesen deshalb delegiert werden. Daher werden sie mit verschiedenen Softwareprodukten mitausgeliefert. Sie werden automatisch bei der Installation mit den Softwareprodukten mit installiert, so dass sie für den Kunden nicht sichtbar sind.

Beispiel: Softwareprodukte und Basismodule

Angenommen, ein Produktsystem PSystem besteht aus zwei Softwareprodukten und einem Basismodul, wie in Abbildung 14 dargestellt ist.

Oberhalb der Linie ist das Produktsystem aus Sicht des Marketings dargestellt. An das Softwareprodukt P1 sind Anforderungen r_1, r_2, r_3 und an das Softwareprodukt P2 Anforderungen r_1, r_4, r_5 gestellt. Realisiert werden die Anforderungen von P1, P2 und dem Basismodul B, wie unterhalb der Linie dargestellt ist. Zur Erfüllung von r_1 muss B realisiert werden, obwohl dieses Basismodul für den Markt nicht sichtbar ist.

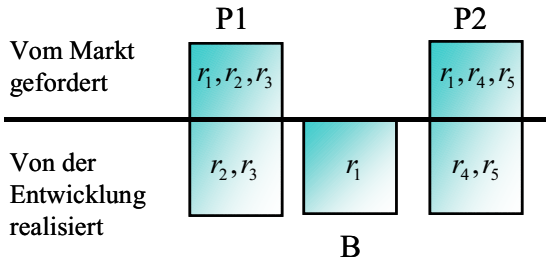


Abbildung 14 – Anforderungen aus Marketingsicht und aus Entwicklungssicht

Folgende Schemata stellen diese beiden Entwicklungsprodukte vor:

Produkttyp Softwareprodukt (abgeleitet von Systemmodul)	
B	Softwareprodukte repräsentieren aus Marketingsicht Teile eines Produktsystems, die unabhängig voneinander verkauft werden sollen. Anforderungen an Produkte stammen sowohl direkt vom Markt als auch von anderen Systemmodulen. Die Anforderungen von anderen Systemmodulen werden indirekt über Modulschnittstellen gestellt.
Z	Die Softwareprodukte sind das wesentliche Ergebnis der gesamten Softwareentwicklung.

A	**Verkaufsinfos:** Alle Informationen, die zur Abwicklung des Verkaufes notwendig sind. (z.B. Preise, Bestellnummer usw.)
G	-
B=Beschreibung, Z=Zweck, A=Attribute, G=Aggregationen	

Produkttyp Basismodul (abgeleitet von Systemmodul)	
B	Repräsentieren Teile eines Systems, welche gemeinsame Aufgaben unterschiedlicher Softwareprodukte erfüllen. Sie enthalten keine direkt vom Markt gestellten Anforderungen und werden auch nicht explizit verkauft. Anforderungen werden immer indirekt von einzelnen Systemmodulen über Modulschnittstellen gestellt.
Z	Mit der Verwendung von Basismodulen können gemeinsame Anforderungen verschiedener Produkte isoliert entwickelt werden, um Wiederverwendungspotentiale optimal auszunutzen.
A	-
G	-
B=Beschreibung, Z=Zweck, A=Attribute, G=Aggregationen	

Die Zerteilung eines Produktsystems in verschiedene Systemmodule bildet einen Kompromiss zwischen verkaufs- und entwicklungsorientierten Rollen (Rollen werden in Abschnitt 7.2 ausführlich behandelt). Verkaufsorientierte Rollen zielen auf die Identifikation von Softwareprodukten ab, die den Markterfolg optimieren. Demgegenüber peilen entwicklungsorientierte Rollen eine parallele Entwicklung und eine Unterstützung der Wiederverwendung an. Um die richtige Systemmodulzerlegung zu finden, müssen die Marktsicht, strategische Ziele und die Systemarchitektur der Vorgängerversion des Produktsystems berücksichtigt werden.

Produkttyp Modulschnittstelle	
B	Mit Hilfe einer Modulschnittstelle werden Abhängigkeiten zwischen verschiedenen Systemmodulen dargestellt (vgl. Definition weiter oben in diesem Abschnitt). Die konkreten Abhängigkeiten werden durch aggregierte Anforderungen repräsentiert. Jede Modulschnittstelle wird mindestens zu zwei Systemmodulen aggregiert. Eines davon aggregiert die Modulschnittstelle in der Rolle *fordert*. Dieses Systemmodul stellt Anforderungen an andere Systemmodule. Ein zweites Systemmodul aggregiert die Modulschnittstelle in der Rolle *erfüllt*. Dieses ist für die Realisierung der zur Modulschnittstelle aggregierten Anforderungen verantwortlich.
Z	In Modulschnittstellen werden die einzigen gegenseitigen Abhängigkeiten zwischen Systemmodulen festgehalten. Daher unterstützen sie die weitestgehend unabhängige Entwicklung einzelner Systemmodule. Mit ihrer Hilfe erfolgt eine klare Trennung zwischen an ein Systemmodul gestellten Anforderungen und von einem Systemmodul selbst realisierten Anforderungen.
A	-
G	**Anforderung:** Verweis auf alle Anforderungen, die von mindestens einem Systemmodul gefordert werden und von einem anderen Systemmodul erfüllt werden müssen.
B=Beschreibung, Z=Zweck, A=Attribute, G=Aggregationen	

Beispiel: Produktsystem mit Systemmodulen und Modulschnittstellen

Das Produktsystem PSystem in Abbildung 14 besteht aus zwei Softwareprodukten P1 und P2 und dem Basismodul B. Da sowohl P1 als auch P2 die Anforderung r_1 an B delegieren, reicht eine einzige Modulschnittstelle aus. Diese wird im Beispiel mit M bezeichnet. Daraus ergeben sich folgende Aggregationen:

- *P1 aggregiert r_2 und r_3 als zu erfüllende Anforderungen und fordert Modulschnittstelle M, erfüllt jedoch keine Modulschnittstelle.*
- *P2 aggregiert r_4 und r_5 als zu erfüllende Anforderungen und fordert Modulschnittstelle M, erfüllt auch keine Modulschnittstelle.*
- *Modulschnittstelle M aggregiert Anforderung r_1.*
- *B aggregiert keine erfüllende Anforderungen und erfüllt aber Modulschnittstelle M, fordert jedoch keine Modulschnittstelle.*

In Kapitel 5 werden Spezifikationstechniken zur Darstellung von Entwicklungsprodukten vorgestellt. Abbildung 19 zeigt dort eine Spezifikation dieses Beispiels.

Obwohl die Systemarchitektur der operativen Strukturierung zugeordnet ist, müssen bei der Architekturdefinition auch strategische Ziele berücksichtigt werden. Die folgenden Entwicklungsprodukte repräsentieren die Systemarchitektur und ihre Teile.

Produkttyp Systemarchitektur	
B	Die Systemarchitektur dokumentiert die entwicklungsseitige Zerlegung eines Produktsystems in Systemmodule und Modulschnittstellen sowie das gesamte Produktsystem betreffende operative Informationen.
Z	Dient der Dokumentation operativer Informationen und darüber hinausgehend einer übersichtlichen Darstellung der Zerlegung des Produktsystems. Die Zerlegung ließe sich jedoch auch indirekt aus den Systemmodulen und zugehörigen Modulschnittstellen erschließen.
A	- **Bezeichner:** Bezeichnung des Produktsystems. - **Planungshorizont:** Anzahl künftiger Versionen (vgl. Abschnitt 4.4.4), die vorausschauend geplant werden sollen.
G	- **Systemmodul:** alle zu einem Produktsystem gehörenden Systemmodule - **Systemarchitektur-Version**: Aggregiert alle zu der Systemarchitektur gehörenden Systemarchitektur-Versionen. Systemarchitektur-Versionen werden im nachfolgenden Abschnitt 4.4.4 behandelt.
B=Beschreibung, Z=Zweck, A=Attribute, G=Aggregationen	

Zur Unterstützung einer Entwicklung eines konsistenten Produktsystems werden im kommenden Abschnitt Regeln zur Verteilung von Anforderungen auf Systemmodule aufgestellt.

Verteilung von Anforderungen auf Systemmodule

Für die Realisierung welcher Anforderungen ein Systemmodul verantwortlich ist, wird eindeutig festgelegt. Dies sind einerseits direkt aggregierte Anforderungen und andere r-

seits Anforderungen an Modulschnittstellen, für deren Erfüllung es zuständig ist . Die Gesamtheit dieser an ein Systemmodul gestellten Anforderungen wird im formalen M odell folgendermaßen repräsentiert:

Definition: An ein Systemmodul gestellte Anforderungen

Sei $p \in P$ *ein Systemmodul. Die Gesamtheit der* an p gestellten Anforderungen *wird durch eine Menge* $R_{p,t} \subseteq R_t$ *mit folgenden Elementen repräsentiert*:

- *sämtliche direkt zum Systemmodul aggregierte Anforderungen und*
- *sämtliche Anforderungen, die zur Realisierung der mit der erfüllt-Aggregation aggregierten Modulschnittstellen nötig sind.*

Bei der Planung der Entwicklung des Produktsystems müssen alle in der Marktsicht en t-haltene Anforderungen auf die Syste mmodule verteilt werden. Viele Anforderungen lassen sich eindeutig auf einzelne Systemmodule verteilen. Andere wirken sich auf me hrere Systemmodule gleichzeitig aus. Für diese gestaltet sich die Verfolgbarkeit schwier i-ger. Ursache hierfür ist die weitgehend unabhängige Entwicklung ei nzelner Systemmodule, weshalb sich niemand unmittelbar für deren Realisierung veran twortlich fühlt.

Abhilfe schafft hierfür die Pflicht zur strikten Verteilung sämtlicher Anforderu ngen auf einzelne Systemmodule. Übergreifende Anforderungen sind dadurch mehrfach in unte r-schiedlichen Systemmodulen enthalten und müssen dort jeweils lokal verfolgt we rden. Unter Umständen wird eine übergreifende Anforderung auf die Systemmodule au fgeteilt. Letztendlich ist eine übergreifende Anforderung dann realisiert, wenn sie für alle Sy s-temmodule, in denen sie enthalten ist, realisiert ist. In den folgenden A bsätzen wird dies in formalen Regeln festgehalten:

Entwicklungsaxiom: Verteilung von Anforderungen auf Systemmodule

Sei $S \subseteq P$ *ein Produktsystem. Aufgrund der obigen Pflicht zur Verteilung von Anforderungen auf Systemmodule wird gefordert, dass alle Anforderungen auf Systemmodule* $p \in S$ *verteilt sind:*

$$R_{S,t} = \bigcup_{p \in S} R_{p,t} .$$

$R_{p,t} \subseteq R_t$ enthält alle tatsächlich an ein Systemmodul p gestellten Anforderungen.

Will ein Kunde Teile des Produktsystems kaufen, so muss ein Verkäufer aus dem Pr o-duktsystem diejenigen Softwareprodukte auswählen, die dessen Anforderungen erfüllen. Ein Softwareprodukt kann aus Kundensicht maximal alle An forderungen erfüllen, die es selbst realisiert oder die über eine Modulschnittstelle gefordert werden und dadurch in anderen Systemmodulen realisiert sind. Bei der Auslieferung müssen hierfür alle Sy s-temmodule mit dem Softwareprodukt mitinstalliert werden, die Anforderungen der en t-sprechenden Modulschnittstelle erfüllen. Handelt es sich bei diesen Systemmod ulen um Basismodule, so werden diese kostenlos mitgeliefert und sind für den Kunden nicht sichtbar. Erfüllen andere Softwareprodukte diese Anfo rderungen, so müssen sie separat gekauft werden. Durch das Modell der Entwicklungsprodukte können zur Erfüllung g e-gebener Anforderungen notwendige Systemmodule automatisch gefunden werd en.

Planung und Realisierung von Anforderungen

Um eine weitgehend unabhängige Entwicklung einzelner Systemmodule gewährleisten zu können, muss auch der Fortschritt der Entwicklung in jedem Systemmodul isoliert verfolgbar sein. Daher wird der Realisierungsstatus jeder Anforderung für einzelne Systemmodule getrennt festgelegt. Dazu werden folgende Mengen definiert:

Definition: Realisierungsstatus eines Systemmoduls

Für ein Systemmodul $p \in S$ *zu einem Zeitpunkt* $t \in T$ *bedeuten folgende Mengen:*

- $R_{p,t}^{pl} \subseteq R_{p,t}$*: alle Anforderungen, die in* p *zum Zeitpunkt* t *zur Realisierung* geplant *sind.*
- $R_{p,t}^{r} \subseteq R_{p,t}$*: alle Anforderungen die in* p *zum Zeitpunkt* t realisiert sind.

Sämtliche über den Realisierungsstatus von Anforderung in Abschnitt 4.2.1 für ein Produktsystem definierten Regeln gelten analog für Systemmodule.

Für ein Systemmodul geplante Anforderungen werden im Systemmodul selbst realisiert. Zwischen gestellten und geplanten Anforderungen besteht folgender Zusammenhang:

Entwicklungsaxiom: Gestellte und geplante Anforderungen eines Systemmoduls

Sei $p \in S$ *ein Systemmodul und* $R_{m,t} \subseteq R_{p,t}$ *die Anforderungen die* p *von einer Modulschnittstelle* $m \in M_p^{ford}$ *fordert dann wird gefordert:*

$$R_{p,t}^{pl} \subseteq R_{p,t} \setminus \bigcup_{m \in M_p^{ford}} R_{m,t}.$$

In den bisherigen Abschnitten wurde der Realisierungsstatus von Anforderungen für Systemmodule und ein Produktsystem isoliert betrachtet. Durch die Darstellung des Zusammenhangs zwischen Planung auf Systemmodulebene und auf Produktsystemebene wird der Realisierungsstatus einer Anforderung systemweit verfolgbar:

Entwicklungsaxiom: Planung einer Anforderung auf Modul- und Systemebene

Für eine Anforderung $r \in R_{S,t}$ *eines Produktsystems* S *zum Zeitpunkt* $t \in T$ *wird gefordert:*

$$(\exists p \in S : r \in R_{p,t}^{pl}) \Leftrightarrow r \in R_{S,t}^{pl}$$

Eine Anforderung gilt für ein Produktsystem geplant, wenn sie für mindestens ein Systemmodul geplant ist. Sie gilt sie als realisiert, wenn sie für alle Systemmodule realisiert ist, für die sie einmal geplant war. Dies drückt folgendes Axiom aus:

Entwicklungsaxiom: Realisierung einer Anforderung auf Modul- und Systemebene

Für eine Anforderung $r \in R_{S,t}$ *eines Produktsystems* S *zum Zeitpunkt* $t \in T$ *wird gefordert:*

$$(\forall p \in S : \exists t_1 < t : r \in R_{p,t_1}^{pl} \Rightarrow r \in R_{p,t}^{r}) \Leftrightarrow r \in R_{S,t}^{r}$$

Verfolgbarkeit zwischen Marktsicht und Systemsicht

Die Marktsicht und Systemsicht unterscheiden sich in erster Linie in ihrer Art, Anforderungen zu strukturieren. Die Marktsicht ist durch die Wechselwirkungen von Variationen gekennzeichnet. Demgegenüber ist die Hauptstrukturierung der Systemsicht die Zerlegung des Produktsystems in gekoppelte Systemmodule.

In der Marktsicht interessiert man sich vorrangig für die Verfolgbarkeit der Realisierungsstati von Anforderungen und Variationen hinsichtlich des gesamten Produktsystems. Dabei ist es irrelevant, auf welche Systemmodule diese verteilt sind und wie sie realisiert werden sollen.

Für die Systemsicht wurde der Zusammenhang zwischen dem Realisierungsstatus von Anforderungen in einzelnen Systemmodulen und dem in einem gesamten Produktsystems beschrieben. Variationen sind entsprechend ihrer Definition in Abschnitt 4.4.2 ebenfalls Anforderungen und werden in der Modellierung der Systemsicht auch nicht gesondert behandelt. Methodisch ist hingegen eine gesonderte Verfolgbarkeit von Variationen wünschenswert. Daher wird in Abschnitt 5.2.1 eine gesonderte Spezifikation für die Abbildung von Variationen auf Systemmodule vorgeschlagen.

4.4.4 Entwicklungssicht (Versionierung)

Die Entwicklungssicht baut auf der strukturellen Zerlegung eines Produktsystems aus Abschnitt 4.4.3 auf. Hauptfokus der Entwicklungssicht ist die Aufteilung von Anforderungen auf unterschiedliche Versionen. Diese trägt der einleitend in Abschnitt 1.1.3 vorgestellten Notwendigkeit zur Versionierung Rechnung. In den folgenden Unterabschnitten wird zunächst das Entwicklungsprodukt Version eingeführt. Anschließend werden versionierte Pendants für Systemmodule, die Systemarchitektur und Modulschnittstellen vorgestellt. Abbildung 15 gibt einen Überblick über alle Produkte der Entwicklungssicht. Erste Vorarbeiten zu diesem Abschnitt wurden in [Dei99b] publiziert.

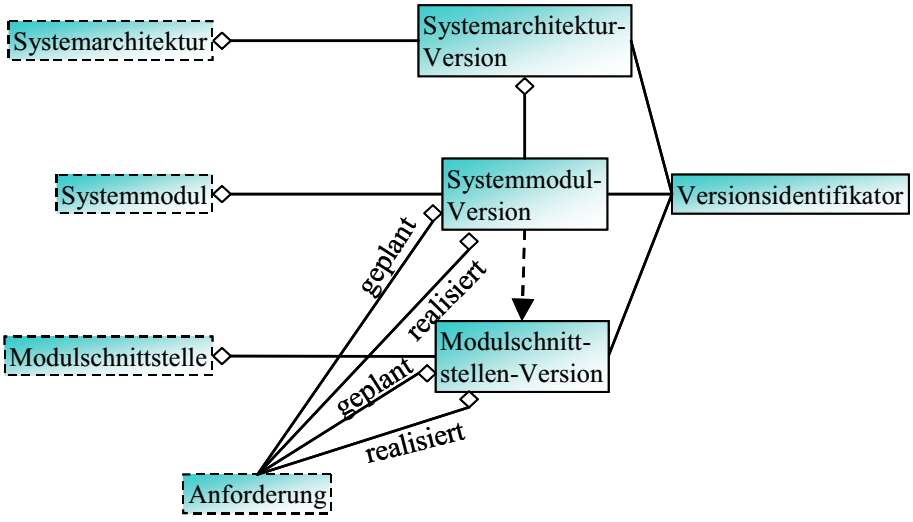


Abbildung 15 – Entwicklungsprodukte der Entwicklungssicht

Die Entwicklungssicht wird im Rahmen der Phase *Definition der Entwicklungssicht* in Abschnitt 7.5.3 erarbeitet. Sie ist Grundlage für die sich daran unmittelbar anschließende tatsächliche Entwicklung Standardsoftware.

Entwicklungsprodukte der Entwicklungssicht

In diesem Unterabschnitt werden die Entwicklungsprodukte der Entwicklungssicht vorgestellt. Zunächst wird zur Unterscheidung von Versionen der Versionsidentifikator eingeführt:

Produkttyp Versionsidentifikator	
B	Identifikator zur Kennzeichnung von Versionen von Softwaresystemen und Produkten.
Z	Dient der Überprüfung, ob zwei Softwareprodukte zusammenarbeiten können, beziehungsweise ob in der Entwicklung deren Zusammenarbeit garantiert werden muss (siehe Abschnitt 7.5.3).
A	**Nummer**: Eine eindeutige Bezeichnung.
G	-
B=Beschreibung, Z=Zweck, A=Attribute, G=Aggregationen	

Beispiel: Versionsnummern

Mögliche Nummern für Versionsidentifikatoren sind Jahreszahl, wie beispielsweise zuletzt in Softwareprodukten von Microsoft oder Kombination von Namen und Nummer wie Intel Pentium III oder Sequenz von durch mit Punkten getrennten Zahlen, wie 1.2.3 sein.

Im formalen Modell repräsentieren Versionsidentifikatoren Elemente einer Menge:

Definition: Menge der Versionsidentifikatoren

Die Menge aller Versionsidentifikatoren wird durch die strikt wohlgeordnete Menge V repräsentiert.

Die Wohlordnung von V identifiziert eine Sequenz aufeinanderfolgender Versionen. Aus methodischen Gründen beschränkt sich der Begriff *Version* auf einen rein zeitlichen Aspekt. Im Gegensatz dazu wird der Begriff *Version* im Kontext des Konfigurations-Managements (vgl. [BD91] und Abschnitt 2.4.1) auch für Softwarevariationen verwendet, die parallel entwickelt werden. Hierzu wurde in Abschnitt 4.4.2 der Begriff Variation eingeführt.

Durch die Versionierung eines Produktsystems wird ein Softwarehersteller zur stufenweisen Verteilung der Realisierung von Anforderungen auf verschiedene Versionen einzelner Systemmodule gezwungen.

Produkttyp Systemmodul-Version	
B	Repräsentation des Inkrements eines Systemmoduls für eine einzige Version.
Z	Ist die Basis für die Spezifikation und Entwicklung einer Version eines Systemmoduls. Aufgrund der hier aggregierten Anforderungen wird die konkrete Entwicklung geplant und verfolgt.

A	**Zykluszeit:** Angestrebter Entwicklungszeitraum der Version mit Anfangszeitpunkt und Endzeitpunkt.
G	- **Anforderung (geplant):** aggregiert alle Anforderungen, die innerhalb der assoziierten Version in der Systemmodul-Version zur Realisierung geplant sind. - **Anforderung (realisiert):** aggregiert alle Anforderungen, die innerhalb der assoziierten Version in der Systemmodul-Version bereits realisiert worden sind.
B=Beschreibung, Z=Zweck, A=Attribute, G=Aggregationen	

Definition: Systemmodul-Version

Sei $P_V \subseteq P \times V$ die Menge aller existierenden Kombinationen von Systemmodulen $p \in P$ mit Versionsidentifikatoren $v \in V$. P_V wird die Menge aller Systemmodul-Versionen genannt. Eine Systemmodul-Version wird mit $p_v = (p,v) \in P_V$ bezeichnet.

In der Regel besteht keine Gleichheit zwischen P_V und $P \times V$, da nicht jedes Systemmodul in allen Versionen erstellt wird, für die Versionsidentifikatoren existieren.

Beispiel:

Zur Fehlerkorrektur müssen abhängig von einzelnen Systemmodulen unterschiedlich viele Serviceversionen erstellt werden.

Da in der Praxis Systemmodule ausschließlich in Versionen entwickelt werden, kann es folglich kein Systemmodul geben, für das keine zugehörige Systemmodul-Version existiert. Daher wird gefordert:

Entwicklungsaxiom: Existenz von Systemmodul-Versionen

$\forall q \in S \exists u \in V : q_u \in P_V$.

Informell bezeichnet $p_v \in P_V$ ein Systemmodul, das in Version v erstellt wird oder worden ist. Systemprodukte werden in jeder dieser Versionen am Markt ausgeliefert.

Entwicklungsaxiom: Zusammenhang zwischen Systemmodul und Systemmodul-Versionen

Für ein Systemmodul $p \in P$ wird gefordert, dass es alle Systemmodul-Versionen $p_v \in P_V$ aggregiert.

In Abschnitt 4.4.3 wurden Produktsysteme über die Kopplung einzelner Systemmodule definiert. Diese hat erhebliche Auswirkungen auf eine versionierte Entwicklung. Systemmodul-Versionen sollen weitgehend unabhängig voneinander entwickelt werden können, wobei gleichzeitig Kopplungen zwischen Systemmodulen konsistent sein sollen. Dazu werden Systemarchitektur-Versionen eingeführt.

Definition: Systemversionsidentifikator

Gegeben sei ein Produktsystem S. Mit $V_S \subseteq V$ wird die Menge aller möglichen Systemversionsidentifikatoren *bezeichnet. Die Menge identifiziert existierende Versionen eines Produktsystems.*

Für jede Systemversion existiert genau eine Systemarchitektur -Version:

Produkttyp Systemarchitektur-Version	
B	Gruppiert Systemmodul-Versionen, die einer Systemversion zugehören.
Z	Unterstützt die Einhaltung von Kompatibilitätsanforderungen zwischen verschiedenen Systemmodul-Versionen.
A	-
G	**Systemmodul-Version:** alle Systemmodul-Versionen, die zu der Systemarchitektur-Version gehören (siehe die kommenden Absätze).
B=Beschreibung, Z=Zweck, A=Attribute, G=Aggregationen	

Definition: Zu einer Systemarchitektur-Version gehörende Systemmodul-Versionen

Gegeben sei ein Produktsystems S und ein Systemversionsidentifikator $v \in V_S$. Die zu einer Systemarchitektur-Version in Systemversion v aggregierten Systemmodul-Versionen werden mit der Menge $S_v \subseteq P_V$ bezeichnet.

Hilfsdefinition: Nachfolger-Operator *succ*

Sei U eine diskrete total geordnete Menge. Für zwei Elemente $a,b \in U$ sei $succ_U(a)$ der Nachfolger-Operator für U, wenn folgende Eigenschaft gilt:

$$b = succ_U(a) \Leftrightarrow a < b \wedge (\forall u \in U \wedge u \neq b : u < a \vee b < u)$$

Jede Systemmodul-Version $p_v \in P_V$ gehört genau einer Systemarchitektur-Version an. In der Praxis müssen aufgrund von Entwicklungsfehlern häufig neue Serviceversionen eines Systemmoduls ausgeliefert werden. Teilweise werden in ihnen zusätzlich zur Fehlerkorrektur dringende neue Anforderungen realisiert. Davon unabhängig gehören die Systemmodule der gleichen Systemarchitektur-Version wie das Vorgängermodul an. Dies hat Auswirkungen auf die Kompatibilität des Systemmoduls. Dieser Tatsache trägt folgendes Axiom Rechnung:

Entwicklungsaxiom: Zuordnung von Systemmodul-Versionen zu Systemarchitektur-Versionen

Sei $p \in S$ und $v_{sys} \in V_S$ ein Systemversionsidentifikator. Es wird gefordert:
$\forall v \in V : p_v \in S_{v_{sys}} \Leftrightarrow v_{sys} \leq v < succ_{V_s}(v_{sys})$

Eine Systemmodul-Version p_v gehört zu der Systemversion $S_{v_{sys}}$ des nächstkleineren Nachbarn v_{sys} von v. Die Einhaltung der Kompatibilität zwischen unterschiedlichen Systemmodul-Versionen wird im Rahmen dieser Arbeit als Aufgabe des Entwicklungsprozesses angesehen. Details hierzu werden in Abschnitt 7.5.3 vorgestellt.

Beispiel:

Angenommen, das Produktsystem PSystem wurde bisher mit drei Verschiedenen Systemversionen, V 4.0, V 5.0, und V 6.0 *ausgeliefert. Das Softwareprodukt P1 und Basismodul B existieren jeweils bereits seit der Version V 4.0. P1 wurde nacheinander in den aufsteigenden Versionen V 4.0, V 4.0.1 V 4.1, V 5.0 und V 6.0 ausgeliefert. B existiert in Versionen V 4.0, V 4.1, V 5.0, V 5.1 und V6.0. Damit*

gehören P1 in V 4.0, V 4.0.1 und V 4.1 beziehungsweise B in V 4.0 und V4.1 zur Systemarchitektur-Version V 4.0; P1 in V 5.0 und B in V 5.0 und V 5.1 zu PSystem V 5.0 und schließlich P1 in V 6.0 beziehungsweise B in V 6.0 zu PSystem V 6.0.

In der Praxis ist die Zugehörigkeit einer Systemmodul-Version zu einer Systemarchitektur-Version häufig aus der speziellen Nummerierung der Versionsidentifikatoren ablesbar. So sind im vorigen Beispiel die erste Ziffer der Versionsnummer der Systemmodul-Versionen und zugehöriger Systemarchitektur-Versionen jeweils gleich.

In Analogie zu Systemmodul-Versionen werden Modulschnittstellen-Versionen definiert:

Produkttyp Modulschnittstellen-Version	
B	Repräsentation des Inkrements einer Modulschnittstelle für eine einzige Version.
Z	Ist Grundlage der Spezifikation und Entwicklung einer Version einer Modulschnittstelle. Aggregierte Anforderungen werden in der Entwicklung geplant und verfolgt.
A	-
G	- **Anforderung (geplant):** aggregiert alle Anforderungen, die innerhalb der assoziierten Version in der Modulschnittstellen-Version zur Realisierung geplant sind. - **Anforderung (realisiert):** aggregiert alle Anforderungen, die innerhalb der assoziierten Version in der Modulschnittstellen-Version bereits realisiert worden sind.
B=Beschreibung, Z=Zweck, A=Attribute, G=Aggregationen	

Definition: Modulschnittstellen-Version

Sei $M_V \subseteq M \times V$ die Menge aller tatsächlich existierenden Kombinationen von Modulschnittstellen $m \in M$ mit Versionsidentifikatoren $v \in V$. M_V wird die Menge aller Modulschnittstellen-Versionen *genannt. Eine Modulschnittstellen-Version wird mit $m_v = (m, v)$ bezeichnet.*

Analog zu Systemmodulen besteht auch zwischen M_V und $M \times V$ in der Regel keine Gleichheit. Eine weitere Analogie zu Systemmodulen ist die Forderung nach der Existenz von Modulschnittstellen-Versionen für jede Modulschnittstelle:

Entwicklungsaxiom: Existenz von Modulschnittstellen-Versionen

$\forall m \in M \exists u \in V : m_u \in M_V$.

Entwicklungsaxiom: Zusammenhang zwischen Modulschnittstellen und Modulschnittstellen-Versionen

Für eine Modulschnittstelle $m \in M$ wird gefordert, dass sie alle Modulschnittstellen-Versionen mit $m_v \in M_V$ aggregiert.

Definition: Von Systemmodulen geforderte und erfüllte Modulschnittstellen

Sei für $p \in P$, $v \in V$ $p_v \in P_V$ eine Systemmodul-Version und M_V die Menge aller Modulschnittstellen-Versionen. Analog zu entsprechenden Mengen der Systemsicht bezeichnet:

(1) *Die endliche Teilmenge* $M_{p_v}^{ford} \subseteq M_V$
die Menge aller von p_v geforderten Modulschnittstellen-Versionen und

(2) *Die endliche Teilmenge* $M_{p_v}^{erf} \subseteq M_V$
die Menge aller von p_v erfüllten Modulschnittstellen-Versionen.

Modulschnittstellen sind eng mit dem Systemmodul gekoppelt, das seine Anforderungen erfüllen soll. Vereinfachend sollen nur Modulschnittstellen mit den selben Versionsidentifikatoren existieren, wie Systemmodule, die sie realisieren. Gleichzeitig sollen auch nur diese zu Systemmodul-Versionen aggregiert sein:

Entwicklungsaxiom: Zusammenhang zwischen Modulschnittstellen und Systemmodulen

Sei $p \in P$ *ein Systemmodul und* $m \in M_p^{erf}$ *eine Modulschnittstelle, die* p *erfüllt.*
Es wird gefordert:
$\forall v \in V : \exists m_v \in M_V \Leftrightarrow \exists p_v \in P_V$ *und*
$\forall v \in V \forall m_v \in M_V \forall p_v \in P_V : m_v \in M_{p_v}^{erf}$ *und*
$\forall v \in V \forall u \in V \wedge u \neq v : m_u \notin M_{p_v}^{erf}$

Durch die enge Kopplung zwischen Modulschnittstellen-Versionen mit Systemmodul-Versionen ist keine gesonderte Versionsplanung für Modulschnittstellen erforderlich. Dies wird im anschließenden Unterabschnitt erläutert.

Versionsplanung und Realisierung von Anforderungen

In diesem Unterabschnitt werden Realisierungszustände von Anforderungen in versionierten Produktsystemen behandelt. Dazu wird zunächst der Realisierungsstatus für versionierte Systemmodule definiert:

Definition: Realisierungsstatus von Anforderungen eines versionierten Produktsystems

Sei $p_v \in P_V$ *eine Systemmodul-Version. Dann bedeuten folgende Mengen:*

- $R_{p_v,t}^{pl} \subseteq R_{p,t}^{pl}$:
  alle in p_v *zum Zeitpunkt* $t \in T$ *zur Realisierung geplanten Anforderungen (entspricht mit geplant aggregierten Anforderungen der Systemmodul-Version).*
- $R_{p_v,t}^{r} \subseteq R_{p,t}^{r}$:
  alle in p_v *zum Zeitpunkt* $t \in T$ *realisierten Anforderungen (entspricht mit realisiert aggregierten Anforderungen der Systemmodul-Version).*

Beispiel: Geplante Anforderungen einer Systemmodul-Version

Die Grafik in Abbildung 16 veranschaulicht als Beispiel die Menge $R^{pl}_{p_v}$.

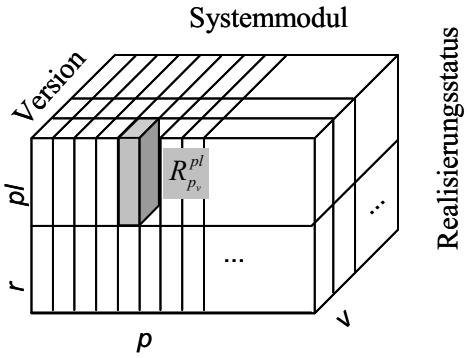


Abbildung 16 – Dimensionen eines versionierten Produktsystems

Die Planung unterliegt einer zeitlichen Dynamik, solange nicht endgültig festgelegt wo r-den ist, welche Anforderungen tatsächlich realisiert werden sollen.

Definition: Endgültiger Realisierungsstatus von Versionen

Sei eine Systemmodul-Version $p_v \in P_V$. *Dann bedeuten:*

- $R^{pl}_{p_v}$ *alle endgültig zur Realisierung in* p_v *geplanten Anforderungen.*
- $R^{r}_{p_v}$ *alle endgültig in* p_v *realisierten Anforderungen.*

Der Einfachheit halber wird angenommen, dass alle realisierten Anforderungen nach ihrer Realisierung immer noch gültig sind, also vom Softwareanwender gefordert werden. Damit muss nicht mehr zwischen jemals geplanten und realisierten Anforderungen unterschieden zu werden. Die Menge wird fortan mit $R^{r}_{p_v}$ bezeichnet.

In der Praxis ist zeitweise aufgrund falsch eingeschätzter Aufwände oder neuer Anforderungen eine Umplanung der Versionszuordnung einzelner Anforderungen erforderlich. Daher müssen nicht alle irgendwann für eine Version geplanten Anforderungen auch tatsächlich in der Systemmodul-Version p_v realisiert werden. Endgültig realisierte Anforderungen einer Version stehen erst zum Fertigstellungstermin fest:

Entwicklungsaxiom: In einer Systemmodul-Version geplante Anforderungen

Sei eine Systemmodul-Version eines Produktsystems S mit $p \in S, v \in V$, *dann wird gefordert:*

$$p_v \in P_V \Rightarrow \exists t_0 \in T : R^{pl}_{p_v,t_0} \neq \phi \wedge \exists t_1 \in T : R^{r}_{p_v,t_1}$$

Das heißt, für jede Systemmodul-Version werden Anforderungen zur Realisierung geplant und auch tatsächlich realisiert.

Definition: Fertigstellungstermin einer Systemmodul-Version

Der Fertigstellungstermin $t_{ready} \in T$ *einer Systemmodul-Version* $p_v \in P_V$ *ist der Zeitpunkt mit folgenden Eigenschaften:*

(1) $\exists t_0 \in T \wedge \forall t_1 \in T \wedge t_0 \leq t_1 < t_{ready} : R^{pl}_{p_v,t_1} \neq \phi \wedge R^{pl}_{p_v,t_{ready}} = \phi$,

(2) $R^{pl}_{p_v,t_{ready}} = \phi$ *und*

(3) $\forall t_2 \in T \wedge t_{ready} \leq t_2 : R^{pl}_{p_v,t_2} = \phi$

Der Fertigstellungstermin beschreibt den Zeitpunkt, an dem keine Anforderungen mehr für eine Systemmodul-Version geplant ist. Vom Fertigstellungstermin an startet in der Praxis die Auslieferung der Systemmodul-Version p_v. Der Termin lässt sich leicht identifizieren, wenn der Realisierungsstatus aller Anforderungen verfolgt wird.

Entwicklungsaxiom: Endgültig realisierte Anforderungen einer Systemmodul-Version

Sei $t_{ready} \in T$ *der Fertigstellungstermin einer Systemmodul-Version* $p_v \in P_V$. *Es wird gefordert:*
$R^r_{p_v} = R^r_{p_v,t_{ready}}$

In der Praxis wird versucht, den Fertigstellungstermin möglichst gut vorauszuplanen. Dazu dient die Festlegung einer Frist, ab der keine Änderungen an der Versionsplanung einer Systemmodul-Version mehr zugelassen werden. Eine Frist ist einerseits nötig, um sogenannte *Moving Targets* zu vermeiden. Darunter wird ein ewiges Hinausschieben der geplanten Auslieferungszeit einer neuen Version eines Systemprodukts verstanden. Andererseits soll die Planung von Tests des Softwaresystems weitgehend parallel zu deren Implementierung möglich sein. Wesentliche Basis hierfür ist eine endgültige Anforderungsspezifikation für das Softwaresystem beziehungsweise einzelner Systemmodul-Versionen. Die Frist wird wie folgt definiert:

Definition: Frist der Versionsplanung

Sei $t_{ready} \in T$ *der Fertigstellungstermin einer Systemmodul-Version* $p_v \in P_V$. *Der Zeitpunkt* $t_0 \in T, t_0 < t_{ready}$ *mit folgenden Eigenschaften:*

$\forall t \in T \wedge t_0 \leq t \leq t_{ready} : R^{pl}_{p_v,t} \cup R^r_{p_v,t} = R^r_{p_v}$

wird als Frist für die Versionsplanung *bezeichnet*

In der Praxis erfolgt bisher eine isolierte Planung der nächsten zu realisierenden Version. Anforderungen, die erst in sich anschließenden Versionen realisiert werden sollen, werden meist gar nicht oder nur unsystematisch spezifiziert. Dadurch wird die Möglichkeit eines vorausschauenden Designs der Software nicht ausreichend genutzt.

Vorraussetzung einer vorausschauenden Planung ist die eindeutige Zuordnung von Anforderungen zu Systemmodul-Versionen. Da nur für eine bestimmte Anzahl künftiger Versionen eine ausreichende Planungssicherheit möglich ist, erfolgt nur für diese eine explizite Planung. Für weiter in die Zukunft reichende Versionen, werden Anforderungen

nur gesammelt und zu einem späteren Zeitpunkt explizit auf Versionen verteilt. Folgendes Axiom beschreibt diesen Zusammenhang:

Entwicklungsaxiom: Verteilung geplanter Anforderungen auf Systemmodul-Versionen

Sei $t_{ready} \in T$ *der Fertigstellungstermin der zuletzt fertiggestellten Version des Systemmoduls p zum Zeitpunkt* $t \in T$, $u = \max\left\{v \in V \middle| t_{ready} \le t\right\}$ *der zugehörige Versionsidentifikator,* $w \in V$ *der größte explizit geplante künftige Versionsidentifikator und* $R^{pl}_{p_\infty,t}$ *die Menge aller Anforderungen, deren Realisierung nach der letzten explizit geplanten Systemmodul-Version geplant ist. Dann wird gefordert:*

$$R^{pl}_{p,t} = \bigcup_{\{v \in V | u < v \le w\}} R^{pl}_{p_v,t} \cup R^{pl}_{p_\infty,t}.$$

Das Axiom enthält drei wesentliche Aussagen:

- Die Planung von Anforderungen erfolgt über mehrere künftige Versionen vorausschauend.
- Es können nur Anforderungen für künftige noch nicht fertiggestellte Systemmodul-Versionen geplant sein.
- Es werden nur die Versionen mit den Versionsidentifikatoren $u+1$ bis w explizit geplant.

Zur Fertigstellung einer Version müssen alle letztendlich dafür geplante Anforderungen realisiert werden. Mit der Realisierung einer Anforderung innerhalb einer Version sind verbrauchbare Anforderungen insgesamt realisiert:

Entwicklungsaxiom: Realisierung verbrauchbarer Anforderungen

Sei $p_v \in P_V$ *eine Systemmodul-Version eines Produktsystems* $S \subseteq P$ *zu einem Zeitpunkt* $t \in T$ *und* $R^{pl_{perm}}_{S,t} \subseteq R_t$ *permanent geplante Anforderungen. Dann wird gefordert:*

$$\forall r \in R : r \in R^r_{p_v} \wedge r \notin R^{pl_{perm}}_{S,t} \wedge r \notin R^c_t \Rightarrow r \in R^r_{p,t}$$

Der Zusammenhang zwischen permanenten Anforderungen und deren Realisierungsstatus wird im anschließenden Abschnitt erläutert.

Dauerhaftigkeit von Anforderungen

In Abschnitt 4.2.1 wurde zwischen verbrauchbaren und permanenten Anforderungen unterschieden. Diese Unterscheidung wurde vorgenommen, um über die Zeitdauer Aussagen treffen zu können, in der eine Anforderung explizit in der Planung berücksichtigt werden muss. Bisher wurde bei verbrauchbaren Anforderungen implizit davon ausgegangen, dass sie innerhalb einer einzigen Version realisiert werden können. Auch dies ist nicht immer der Fall. In der Praxis wurden verschiedenen Sonderfälle über die Dauerhaftigkeit von Anforderungen identifiziert. Diese haben unterschiedliche Auswirkungen auf die Versionsplanung:

1. Anforderungen, die über mehrere Versionen hinweg realisiert werden (*Multi-Versions-Anforderungen*). Dies ist dann der Fall, wenn beispielsweise verfügbare Ressourcen oder Zeit nicht zur Realisierung innerhalb einer Version ausreichen. Häufig werden hier zunächst Zwischenlösungen für eine Anforderung realisiert. Diese verbleiben in der Software mindestens so lange, bis die Realisierung der Original-Anforderung abgeschlossen ist.

2. Anforderungen, die permanent gelten (*Permanente Anforderungen*). Diese wurden in Abschnitt 4.2.1 eingeführt.

3. Anforderungen, die von Zeit zu Zeit die Entstehung anderer Anforderungen bedingen. So kann zum Beispiel eine Anforderung bestehen, dass das Standardsoftwaresystem immer kompatibel zur neuesten Version des basierenden Betriebssystems ist. Ein Versionswechsel des Betriebssystems bedingt dann eine Reihe von neuen Anforderungen an das Standardsoftwaresystem, deren Umsetzung diese Kompatibilität wiederherstellt. Die Anforderung nach der Kompatibilität gehört zur Menge der *bedingten Anforderungen*.

Zur Integration von Anforderungen unterschiedlicher Dauerhaftigkeit in das Modell werden weitere Mengen und Bedingungen zwischen Anforderungen festgelegt. Diese dienen wiederum der kontinuierlichen Konsistenzerhaltung der Versionsplanung. In den folgenden Absätzen wird jeder der drei Fälle betrachtet.

Multi-Versions-Anforderungen: Für Multi-Versions-Anforderungen werden sowohl die Aufteilung der Realisierung der Anforderungen auf mehrere Versionen und als auch die Einführung von Zwischenlösungen betrachtet.

Beispiel:

Ein HTML-Browser enthält eine einfache Editorfunktion. Kurz vor der Auslieferung der neuen Version des Browsers wird ein neuer Standard für HTML verabschiedet. Der Browser soll diesen neuen Standard berücksichtigen. Um die Auslieferung nicht zu verzögern wird beschlossen, in der aktuellen Version nur die Darstellung des neuen Standards einzubauen. Neue Editorfunktionen sollen erst in der nachfolgenden Version integriert werden.

Aufteilung auf Versionen: Die Realisierung einer Multi-Versions-Anforderung verteilt sich über mehrere Versionen. Dazu muss eine Anforderung in Teil anforderungen aufgeteilt werden, wovon jede in einer anderen Version realisiert wird. Die in Abschnitt 4.2.1 erwähnte Aggregation von MV-Teilanforderungen ist auf eine Hierarchieebene beschränkt. Eine Anforderung, die MV-Teil einer anderen Anforderung ist, kann nicht auch noch ihrerseits MV-Teile aggregieren. Des weiteren können nur normale Anforderungen oder Zwischenlösungen aggregiert sein. Dadurch können MV-Teilanforderungen eindeutig zu Versionen zugeordnet werden.

Definition: Multi-Versions-Anforderung

Sei $t \in T$ ein Zeitpunkt und $r_{mv} \in R_{p,t}$ Anforderung eines Systemmoduls $p \in S$, die als MV-Teil Anforderungen (i. Z. $R_{r_{mv},t} \subseteq R_{p,t}$) mit folgenden Eigenschaften aggregiert:

(1) Planung: $r_{mv} \in R^{pl}_{p,t} \Rightarrow \forall r \in R_{r_{mv},t} : r \in R^{pl}_{p,t} \vee r \in R^{r}_{p,t}$

(2) Realisierung: $(\forall r \in R_{r_{mv},t} : r \in R^{r}_{p,t}) \Rightarrow r_{mv} \in R^{r}_{p,t}$

(3) Versionsplanung: $u = \max\left\{v \in V \mid \exists r \in R_{r_{mv},t} : r \in R^{pl}_{p_v,t}\right\} \Rightarrow r_{mv} \in R^{pl}_{p_u,t}$

Dann wird r_{mv} *als* Multi-Versions-Anforderung *bezeichnet.*

MV-Teilanforderungen einer Multi-Versions-Anforderung werden in der Regel in jeweils unterschiedliche Versionen eingeplant. Zur Erläuterung der Definition:

(1) Planung: jede als MV-Teilanforderung wird zur Realisierung geplant, sobald die Multi-Versions-Anforderung geplant wird.

(2) Realisierung: durch die Realisierung aller als MV-Teil aggregierten Anforderungen gilt die Multi-Versions-Anforderung als realisiert.

(3) Versionsplanung: eine Multi-Versions-Anforderung wird in die höchste Version zur Realisierung eingeplant, in der eine ihrer MV-Teilanforderungen geplant ist.

Da Multi-Versions-Anforderungen innerhalb einer einzigen Version explizit eingeplant sind, bleiben sämtliche Axiome gültig.

Beispiel: Multi-Versions-Anforderung

Die Anforderung „Integration des neuen HTML-Standards“ wird in zwei MV-Teilanforderungen aufgeteilt: „Integration des neuen HTML-Standards in der Visualisierung“ und „Integration des neuen HTML-Standards im Editor“.

Zwischenlösungen: Zur schrittweisen Realisierung von Multi-Versions-Anforderungen müssen manche Teilanforderungen durch vorläufige Zwischenlösungen ersetzt werden. Angenommen eine komplexe Funktionalität mit einer anspruchsvollen Benutzerschnittstelle wird geplant. Da der aktuellen Version wenige Ressourcen zur Verfügung stehen, soll momentan eine einfache Benutzerschnittstelle als Zwischenlösung realisiert werden. In einer späteren Version kann die aufwendige Schnittstelle nachgezogen werden.

Zwischenlösungen sind spezielle Anforderungen die mit der letztendlichen Lösung einer Anforderung korrespondieren. Es besteht folgende Abhängigkeit:

Definition: Zwischenlösung einer Anforderung

Seien $r_z \in R$ *und* $r \in R$ *und* $r \in R^{pl}_{S,t_0}, r_z \in R^{pl}_{S,t_0} \cup R^{r}_{S,t_0}$ *zu einem gegebenen Zeitpunkt* $t_0 \in T$ *mit folgender Eigenschaft:*

$\forall t_1 \in T : t_1 > t_0 \wedge r \in R^{r}_{S,t_1} \Rightarrow (\forall t \geq t_1 : r_z \notin R^{pl}_{S,t_1} \cup R^{r}_{S,t_1})$ *und*

$\exists t_1 \in T, t_1 > t_0 : r \in R^{r}_{S,t_1}$

Dann stellt r_z *eine Zwischenlösung von* r *dar.*

Eine Zwischenlösung r_z von r ist also nur so lange geplant oder realisiert, bis r selbst realisiert ist. Dabei muss r auch tatsächlich irgendwann realisiert werden.

Beispiel:

Für die Multiversions-Anforderung des obigen HTML-Browsers könnte für den HTML-Editor eine Zwischenlösung integriert werden, bei der anstatt mit graphischer Unterstützung ein einfacher Texteditor zur Eingabe der neuen Konstrukte geöffnet wird.

In der Praxis müssen Zwischenlösungen häufig im System verbleiben. Ursache hierfür ist die Gewöhnung der Anwender an die Zwischenlösung. In diesem Fall migriert die Zwischenlösung zu einer vollwertigen Anforderung, die unabhängig von r existiert.

Permanente Anforderungen: Als nächstes werden permanente Anforderungen beschrieben. Wie in Abschnitt 4.2.1 bereits festgelegt, bleiben diese kontinuierlich im geplanten Realisierungszustand, sobald sie erstmals geplant sind. Einzige Ausnahme ist, wenn sie explizit verworfen wird. Wird eine Anforderung verworfen, muss im Einzelfall entschieden werden, ab welcher Version diese Anforderung nicht mehr gültig sein soll.

Ansonsten gilt eine permanente Anforderung automatisch für jede künftige Version als geplant. Um in der Entwicklung nachvollziehen zu können, dass eine permanente Anforderung in einer Version berücksichtigt worden ist, wird sie wie alle anderen Anforderungen behandelt. Dazu wird die Anforderung für jede einzelne Version verbraucht.

Eine Anforderung kann nicht gleichzeitig verbrauchbar und permanent sein. Bezieht sich eine Anforderung auf mehrere Systemmodule, dann ist sie für alle Module entweder permanent oder verbrauchbar.

Bedingte Anforderungen: Die letzte Art dauerhafter Anforderungen sind bedingte Anforderungen. Diese werden anhand des folgenden Beispiels erläutert:

Beispiel:

Eine Anforderung an eine Standardsoftware ist die fortlaufende Kompatibilität mit einem vorgegebenen Betriebssystem. Wird eine neue Version des Betriebssystems auf den Markt gebracht, so muss diese überprüft werden, ob diese Auswirkungen auf die Standardsoftware hat. Gegebenenfalls werden dadurch Anforderungen verursacht, die zur Wiederherstellung der Kompatibilität zu diesem Betriebssystem notwendig sind.

Bedingte Anforderungen sind ein Sonderfall permanenter Anforderungen. Diese wirken sich jedoch nicht unmittelbar auf das Produktsystem aus, sondern verursachen das Entstehen anderer Anforderungen. Im Beispiel ist die Kompatibilitätsanforderung eine bedingte Anforderung. Anforderungen zur Wiederherstellung der Kompatibilität sind verursachte neue Anforderungen. Diese müssen, wie andere Anforderungen normal in der Versionsplanung eines Produktsystems berücksichtigt werden.

Bedingte Anforderungen werden nicht in der Versionsplanung, sondern während des Herausarbeitens, berücksichtigt. Dazu werden Bedingungen notiert, in welchen Fällen durch eine bedingte Anforderung neue Anforderungen verursacht werden. Diese Bedingungen müssen beim Herausarbeiten kontinuierlich berücksichtigt werden. Die Bedingung im Beispiel ist die Auslieferung einer neuen Version des Betriebssystems.

4.5 Mögliche Werkzeugunterstützung

In den vorangegangenen Abschnitten wurde intensiv eine formale Notation eingesetzt, um Zusammenhänge zwischen Entwicklungsprodukten zu beschreiben. Diese formuliert unterschiedliche Regeln zur Entwicklung komplexer Standardsoftware. Werden diese Regeln eingehalten, so wird eine in sich konsistente Entwicklung und eine hohe Verfolgbarkeit der Anforderungen unterstützt. Durch die weitgehend formale Definition des Modells kann leicht ein Softwarewerkzeug abgeleitet werden, das die Einhaltung der Regeln überprüft beziehungsweise automatisch gewährleisten kann. Darüber hinaus kann ein entsprechendes Werkzeug weitere Tätigkeiten unterstützen.

Ein derartiges Werkzeug soll sämtliche Entwicklungsprodukte und Assoziationen des Modells der Entwicklungsprodukte adäquat erfassen und verwalten können. Darüber hinaus soll es die Überprüfung der Entwicklungsaxiome unterstützen. Das Werkzeug kann folgende Unterstützung für den Requirements Engineering Prozess, das Requirements Management und für das Konfigurations-Management komplexer Standardsoftware leisten:

Der **Requirements Engineering Prozess** (vgl. Kapitel 7) wird folgenderweise unterstützt:

- **Verfolgung von Abhängigkeiten:** Konsequenzen von Abhängigkeiten auf die Planung der Realisierung können verfolgt werden. Stehen beispielsweise zwei Anforderungen zueinander in benötigt-Assoziation (vgl. Abschnitt 4.2.1), so müssen beide Anforderungen in der selben Version zur Realisierung geplant werden. Dies kann automatisch anhand des Produktmodells überprüft werden.
- **Automatische konsistente Spezifikationen:** Die Änderungsflexibilität äußert sich auch in der Möglichkeit, automatisch verschiedene Spezifikationen aus dem Modell der Entwicklungsprodukte zu generieren (Details hierzu siehe in Kapitel 5, insbesondere das Beispiel zur textuellen Spezifikation in Abschnitt 5.2.2). Die automatische Generierung erlaubt darüber hinaus spezifische Spezifikationen für einzelne Aktivitäten des Entwicklungsprozesses, was zusätzlich den Ablauf der Entwicklung erheblich unterstützt (vgl. auch den Vorschlag in Abschnitt 5.2.1).
- **Überprüfung von Konsistenzregeln:** Konsistenzregeln, wie beispielsweise die Verpflichtung der Zuordnung aller Anforderungen zu Aspekten, einzelnen Systemmodulen oder einzelnen geplanten Versionen lassen sich automatisch überprüfen.
- **Überprüfung von Modellzuständen:** Schließlich können auch Modellzustände (vgl. Abschnitt 3.2.3), die Voraussetzung für den Start einzelner Aktivitäten (vgl. Abschnitt 3.1.1 und Kapitel 7) sind, automatisch überprüft werden.

Das selbe Werkzeug kann für das **Requirements Management** (vgl. Abschnitt 1.2.5) wie folgt behilflich sein:

- **Verfolgung des Realisierungsstatus:** Die Planung und der Realisierungsstatus des gesamten Produktsystems und auch einzelner Anforderungen können lückenlos verfolgt werden. Durch die formalen Definitionen und Axiome ist beispielsweise eindeutig festgelegt, unter welchen Bedingungen eine Anforderung als realisiert gelten. Der Realisierungsstatus jeder Anforderung kann bis hin zu einzelnen Systemmodulen nachvollzogen werden. Dadurch wird für jedes Systemmodul leicht identifizierbar,

welche Anforderungen noch realisiert werden müssen, um zu einem konsistenten Produktsystem (vgl. Definition in Abschnitt 4.2.1) zu kommen. Dies dient auch unmittelbar einer fehlerfreien Weiterentwicklung einzelner Zerlegungseinheiten einer Software im Sinne des Konfigurations-Managements (vgl. Abschnitt 2.4.1).

- **Änderungsflexibilität:** Ein Werkzeug erhöht auch die Änderungsflexibilität des gesamten Requirements Engineerings. Da zwischen einzelnen Entwicklungsprodukten Abhängigkeiten klar definiert sind, lassen sich bei Modifikationen auch mögliche Auswirkungen auf andere Entwicklungsprodukte leicht nachvollziehen. In manchen Fällen können Änderungen sogar automatisiert werden. Falls z. B. die Planung einer Multi-Versions-Anforderung (vgl. Abschnitt 4.4.4) um Versionen verschoben werden muss, so können sämtliche aggregierten Teil-Anforderungen automatisch mitverschoben werden. Gegebenenfalls können bei derartigen Verschiebungen auch auftretende Ressourcenengpässe überprüft werden. Modifikationen des Modells der Entwicklungsprodukte können leicht horizontal verfolgbar gemacht werden, indem einzelne Änderungen dynamischer Entwicklungsprodukte (vergleiche Index t aus Abschnitt 4.2) und Assoziationen automatisch protokolliert werden.

Einen wesentlichen Beitrag leistet das Modell der Entwicklungsprodukte vor allem zur Unterstützung des **Konfigurations-Managements** komplexer Standardsoftware durch die Festlegung von Kompositionsregeln (vgl. Abschnitt 2.4.1). Diese sind sowohl unmittelbar vom zu entwickelnden Softwaresystem als auch von dessen Systemklasse abhängig. Die vorliegende Arbeit definiert Kompositionsregeln für die Systemklasse komplexer Standardsoftware. Speziell für die drei Sichten Marktsicht, Systemsicht und Entwicklungssicht (vgl. Abschnitte 4.4.2, 4.4.3 und 4.4.4) bestehen folgende Zusammenhänge zu Kompositionsregeln:

- Die in der **Marktsicht** definierten Aktivierungsregeln können abhängig von der Entscheidung, wie Variationen in der Standardsoftware realisiert werden sollen, Kompositionsregeln sein. Wird entschieden, dass einzelne Variationen der Software als eigenständige Systemmodule entwickelt werden sollen, so wird die Art der Zerlegung unmittelbar durch die Aktivierungsregel bestimmt. Die mittels der Aktivierungsregeln ausgeschlossene oder geforderte gleichzeitige Aktivierung von Anforderungen bestimmt unmittelbar, welche Anforderungen gemeinsam in einem Systemmodul realisiert werden müssen. Der gegenseitige Ausschluss mancher Systemmodule führt dazu, dass sie keiner Konfiguration gleichzeitig angehören dürfen. Dies ist unmittelbar eine Kompositionsregel für das Konfigurations-Management. Falls unterschiedliche Variationen eines Variationstyps innerhalb eines Systemmoduls zur Realisierung vorgesehen werden, wirken sich die Aktivierungsregeln hingegen nicht mehr auf das Konfigurations-Management aus, sondern stellen unmittelbar Anforderungen an die Funktionalität der Software selbst dar.
- In der **Systemsicht** wird die Aufteilung des Produktsystems in Systemmodule und Modulschnittstellen bestimmt. Aus dieser können unmittelbar Kompositionsregeln für das Konfigurations-Management abgeleitet werden, da dadurch feststeht, welche Systemmodule zusammenarbeiten können. Durch die nachvollziehbare Verteilung aller Anforderungen auf die Systemmodule ist sogar zu jedem Zeitpunkt der Entwicklung eindeutig identifizierbar, welche Systemmodule beim aktuellen Entwicklungsfortschritt bereits komponierbar sind.

- Aus der **Entwicklungssicht** lassen sich in Analogie zur Systemsicht Kompositionsregeln für Systemmodul-Versionen ableiten. Grundvoraussetzung für den gemeinsamen Einsatz zweier Systemmodul-Versionen ist, dass sie auch bereits in der Systemsicht zusammenarbeiten können. Die Komposition unterschiedlicher Systemmodul-Versionen wird zusätzlich durch Vorgaben eingeschränkt, die eine parallele Weiterentwicklung erlauben. Diese werden in Abschnitt 7.5.3 im Rahmen des Prozessmodells für das Requirements Engineering komplexer Standardsoftware vorgestellt.

Spezialisierte Konfigurations-Management-Werkzeuge, wie beispielsweise CoMa (vgl. [Wes96]), können die Einhaltung von Kompositionsregeln unterstützen. Mit derartigen Werkzeugen könnte das Requirements Engineering-Werkzeug gekoppelt werden. Aus dem aktuellen Zustand des Modells der Entwicklungsprodukte könnten so Kompositionsregeln in für das Konfigurations-Management-Werkzeug generiert werden.

Insgesamt können die in dem Modell der Entwicklungsprodukte vorgestellten Konzepte auch über den Anwendungsbereich komplexer Standardsoftware hinaus angewendet werden. Die Marktsicht kann für Software mit Variationen verwendet werden, die Systemsicht für Software mit mehreren isoliert entwickelten Teilen und die Entwicklungssicht für versionierte Software. Das Besondere dieser Arbeit ist jedoch, dass die drei Sichten derart aufeinander abgestimmt sind, dass zwischen ihnen Anforderungen lückenlos verfolgbar sind.

Ein entsprechendes Requirements Engineering-Werkzeug dient nicht nur der Unterstützung, sondern ist sogar notwendige Voraussetzung, die Entwicklung komplexer Standardsoftware adäquat durchführen zu können. Durch die präzise Darstellung des Modells der Entwicklungsprodukte ist eine Voraussetzung für ein Werkzug geschaffen. Eine detaillierte Beschreibung eines Werkzeugkonzeptes würde jedoch den Rahmen der vorliegenden Arbeit sprengen.

Kapitel 5

Spezifikation von Entwicklungsprodukten

In Kapitel 4 wurden Entwicklungsprodukte und Assoziationen zur U nterstützung des Requirements Engineerings komplexer Standardsoftware vorgestellt. Bestehende Spez i-fikationstechniken von Anforderungen reichen hierzu nicht umfassenden Spezifikat ion aus. In den folgenden Abschnitten werden Konzepte zur der Spezifikation komplexer Standardsoftware vorgestellt. Um nicht den Rahmen der Arbeit zu sprengen, werden lediglich grobe Ziele für Spezifikationstechniken und L ösungsideen vorgestellt. Dabei erfolgt eine Konzentration auf die Spezifikation der Entwicklungsprodukte und Assozi a-tionen des Kerns der Arbeit, also der operativen Strukturi erung (vgl. Abschnitt 4.4). Ansätze zur Spezifikation spezieller Aspekte komplexer Standardsof tware werden in diese Betrachtungen miteinbezogen. Mit Spezifikationstechniken für komplexe Standar d-software werden drei wesentliche Ziele verfolgt. Dies sind die Unterstützung des Requ i-rements Engineering Prozesses selbst, die Unte rstützung der Planung weiterer Entwicklungsaktivitäten und schließlich die Unterstützung der Verfolgbarkeit der Anfo r-derungen.

Zur Unterstützung des Requirements Engineering Prozesses einer komplexen Standar d-software muss die darin enthaltene Komplexität beherrscht werden. Dazu werden S pezi-fikationstechniken benötigt, mit deren Hilfe grobe Zusamme nhänge zwischen Entwick-lungsprodukten übersichtlich beschrieben werden können. Um sich einen Übe rblick über das gesamte Produktsystem verschaffen zu können, ist hier eine Abstraktion von Detail s nötig. Die Beschreibung grober Zusammenhänge wird im folgenden unter dem Begriff *externe Spezifikation* zusammengefasst.

Dagegen müssen bei Planung der Entwicklung zunächst Anforderungen inhaltlich b e-schrieben werden können. Darüber hinaus müssen auch Zus ammenhänge, die sich aus der Entwicklung komplexer Standardsoftware ergeben, in Bezug zu diesen inhaltl ichen Beschreibungen gebracht werden können. Es muss beispielsweise spezifiziert we rden können, zu welchen Aspekten, Variationen oder Versionen einzelne Anforderungen ge-hören. Insbesondere die übersichtliche Darstellung von Gemeinsamkeiten und Spezif ika von Variationen unterstützt eine Wiederverwendung von Implementierungsteilen. We i-terhin kann durch die gleichzeitige Spezifikation aktueller und künftig ge planter Anfor-derungen vorausschauend für künftige Versionen entwickelt werden. Die Darstellung dieser Zusammenhänge wird als *interne Spezifikationen* bezeichnet.

Die Verfolgbarkeit von Anforderungen ist für die an der Entwicklung komplexer Standardsoftware beteiligten Rollen von entscheidender Bedeutung. Einerseits dient sie dem Controlling des Fortschritts der Entwicklung selbst. Dazu muss der Realisierungsstatus einzelner Anforderungen adäquat spezifiziert werden. Andererseits muss beispielsweise für das Marketing und den Vertrieb erkenntlich sein, welche Anforderungen wo realisiert werden. Es muss mit geringem Aufwand zu identifizieren sein, welche Anforderungen das gesamte Produktsystem erfüllt und was davon einzelne Softwareprodukte leisten. Hierbei muss berücksichtigt werden, in welcher Version welche Anforderungen zur Realisierung geplant sind, beziehungsweise bereits realisiert sind. Schließlich müssen Auswirkungen neuer Anforderungen auf das bestehende System leicht identifizierbar sein.

Im Abschnitt 5.1 werden bestehende Spezifikationstechniken im Hinblick auf ihre Eignung für die Spezifikation komplexer Standardsoftware eingeordnet. Der sich anschließende Abschnitt 5.2 präsentiert neue Ideen, wie sich die Spezifikation komplexer Standardsoftware verbessern lässt.

5.1 Bestehende Ansätze zur Spezifikation

In Abschnitt 2.3 wurden allgemeine Ansätze zur Spezifikation von Anforderungen vorgestellt. Wie mehrfach erwähnt, müssen zur adäquaten Spezifikation komplexer Standardsoftware zusätzliche Spezifikationstechniken eingeführt werden. Die folgenden Unterabschnitte beschreiben, welche wesentlichen Aspekte des Modells für Entwicklungsprodukte aus Kapitel 4 beschrieben werden müssen und zeigen den aktuellen Stand hierfür anwendbarer Beschreibungstechniken.

5.1.1 Grundelemente

Zu den Grundelementen des Modells für Entwicklungsprodukte (vgl. Abschnitt 4.2) gehören Anforderungen und Aspekte. Zur Spezifikation von Anforderungen existieren unterschiedliche Ansätze, auf die bereits in Abschnitt 2.3 eingegangen worden ist. In Ergänzung hierzu wird lediglich eine Möglichkeit zur Beschreibung der in Abschnitt 4.2.1 eingeführten Assoziationen benötigt.

Zur Beschreibung von Aspekten hingegen existieren bisher nur sehr wenige Ansätze, da sich die Wissenschaft und Forschung wieder erst in den letzten Jahren intensiv mit dem Separation of Concerns (vgl. Abschnitt 3.2.5) beschäftigt. Ein Beispiel hierfür ist der sogenannte Aspect Browser in [GKY99]. Mit Hilfe dieser Technik werden Aspekte im Quellcode von Programmen durch verschiedene Farben voneinander isoliert dargestellt. Zusätzlich gibt es die Möglichkeit der graphischen Darstellung des prozentualen Anteils von Aspekten innerhalb einzelner Dateien. Damit lässt sich beispielsweise leicht identifizieren, auf welche Module eines Programms welche Aspekte sich auswirken.

5.1.2 Marktsicht

Die Marktsicht wird hauptsächlich durch Variationen und zugehörige Assoziationen repräsentiert (siehe Abschnitt 4.4.2). Die Entwicklung von Variationen spielt in der Wissenschaft im Kontext von Produktlinien (vgl. z. B. [BCC97]) und im Umfeld der Wiederverwendung von Software (vgl. z. B. [JGJ97]) eine Rolle. Beide Gebiete konzentrieren sich auf die Entwicklung generischer Architekturen und Komponenten.

Diese werden in unterschiedlichen Produkten wiederverwendet, die an spezifische Kundenbedürfnisse angepasst sind. Der Fokus der beiden Ansätze wird ausschließlich auf die Variation konkreter Funktionen einer Software gelegt.

Die Forschung um Produktlinien kennt Beschreibungstechniken zur Darstellung von derartigen Funktions-Variationen. Externe Sichten, häufig *Feature-Bäume* genannt (vgl. [KCH+90, Lam98, HLS+98, Dav95, Tra95, GFA97]) sind UND-ODER-Bäume, worin jeder Knoten eine Funktion repräsentiert. Kinder eines ODER-Knotens repräsentieren alternative Funktionen während UND-Knoten Aggregationen repräsentieren. Zusätzlich wird für jeden Kind-Konten eine Unterscheidung zwischen Pflicht- und optionalem Gebrauch gemacht (vgl. Abschnitt 4.4.2). In allen Ansätzen ist jedoch die Definition des Begriffs der Variation unklar. Hinsichtlich des Modells der Entwicklungsprodukte aus Abschnitt 4.4.2 fehlen diesen Spezifikationstechniken Assoziationstypen und die klare Unterscheidung zwischen Variationstypen und Variationen.

Interne Sichten hinsichtlich Variationen repräsentieren konkret die Gemeinsamkeiten und Unterschiede unterschiedlicher Variationen. Diese beziehen sich einerseits auf graphische Architekturbeschreibungen mit variierenden Submodulen (vgl. Variation spunkte in UML Package-Diagrammen [JGJ97] oder [HLS+98]). Andererseits werden traditionelle Beschreibungstechniken, wie ER-Diagramme, Zustandsdiagramme, Systemstrukturdiagramme (vgl. [KCH+90, CAJ98]) oder Statecharts und Activititycharts (vgl. [HLN+90]) erweitert. Zur Unterscheidung zwischen Variationen werden unterschiedliche Präsentationsstile für graphische Elemente (z. B. gestrichelte statt durchgezogene Linien) und Beschriftungen (z. B. kursiver statt normale Schreibweise) verwendet. Allen Ansätzen ist vor allem gemeinsam, dass die speziellen Darstellungsstile jeweils sehr stark von der verwendeten Beschreibungstechnik abhängig sind. In Abschnitt 5.2 wird ein neuer Ansatz vorgestellt, der eine einheitliche Erweiterung verschiedener Beschreibungstechniken ermöglicht.

5.1.3 Systemsicht

Wie in Abschnitt 4.4.3 eingeführt, umfasst die Systemsicht die Zerlegung eines Produktsystems in Systemmodule und Modulschnittstellen. Wesentliche Gesichtspunkte sind hierbei, welche Systemmodule existieren, welche Modulschnittstellen wie aggregiert sind und wie sich die Anforderungen auf Systemmodule verteilen.

Zur externen Beschreibung eignen sich prinzipiell übliche Repräsentationen von Graphen mit Knoten und Kanten, die in der Praxis häufig zur abstrakten und informellen Beschreibung von Softwarearchitekturen verwendet werden. Zur internen Spezifikation von Softwarearchitekturen existieren unterschiedliche mehr oder weniger ausgereifte Architekturbeschreibungssprachen (vgl. unter anderem WRIGHT, Darwin, Rapide, [All97, MDEK95, MK96, LAK+95], die erlauben, das Verhalten von Komponenten, deren Schnittstellen und die Kommunikation zwischen Komponenten genauer zu beschreiben. Unabhängig davon, welche Architekturbeschreibungssprache verwendet wird, muss für die interne Spezifikation der Systemsicht der Unterschied zwischen gestellten und erfüllenden Anforderungen (vergleiche Abschnitt 4.4.3) ersichtlich sein.

5.1.4 Entwicklungssicht

Die Entwicklungssicht repräsentiert die Verteilung der Entwicklung des Produktsystems auf verschiedene Versionen (vgl. Abschnitt 4.4.4). Hierzu müssen verschiedene Systemmodul-Versionen und die Verteilung von Anforderungen auf diese repräsentiert werden.

Im Konfigurations-Managements wurden zur externen Spezifikation von Versionen sogenannte Versionsgraphen eingeführt (vgl. z. B. [Wes96]). Mit Hilfe dieser Graphen können Zusammenhänge verschiedener Versionen einzelner Module in der Softwareentwicklung beschrieben werden. Diese Graphen lassen sich prinzipiell zur Beschreibung von Versionen eines Produktsystems einsetzen. Im Konfigurations-Management wird jedoch prinzipiell nicht zwischen Versionen und Variationen im Sinne der vorliegenden Arbeit unterschieden. Daher muss auf diesen konzeptionellen Unterschied zu den entsprechenden Entwicklungsprodukten geachtet werden.

Zur internen Spezifikation von Versionen existieren keine adäquaten Ansätze. In der Praxis erfolgt eine relativ rudimentäre Versionsplanung mit Hilfe einzelner Tabellen. Spezifikationen repräsentieren dabei nur isolierte Versionen einzelner Systemmodule. Beziehungen zu anderen Systemmodulen und anderen Versionen werden - wenn überhaupt - nur per Hand innerhalb einzelner Dokumente eingefügt. Änderungen müssen damit immer manuell erfolgen, was leicht zu Fehlern führen kann.

5.2 Neue Ansätze zur Spezifikation

In diesem Abschnitt werden neue Ansätze zur Spezifikation von Entwicklungsprodukten komplexer Standardsoftware vorgestellt. Zunächst wird im kommenden Unterabschnitt eine Technik zur externen Spezifikation von Entwicklungsprodukten präsentiert. Der sich anschließende Abschnitt befasst sich dann mit Möglichkeiten, die Bezüge der Entwicklungsprodukte komplexer Standardsoftware in internen Spezifikationen darzustellen.

5.2.1 Externe Spezifikation

Zur externen Spezifikation der Entwicklungsprodukte wird eine Erweiterung von UML-Klassendiagrammen vorgeschlagen. Klassendiagramme eignen sich sehr gut zur Darstellung sämtlicher in Kapitel 4 vorgestellten Entwicklungsprodukte, Assoziationen und Aggregationen. Damit verschiedene Entwicklungsprodukttypen und verschiedene Assoziationstypen gut voneinander unterschieden werden können, werden jeweils Stereotypen eingeführt. Entwicklungsprodukttypen stellen Stereotypen von UML-Classifiern dar. Für wesentliche Entwicklungsprodukttypen werden eigene graphische Symbole eingeführt, die in Abbildung 17 dargestellt sind.

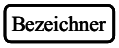
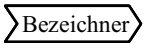
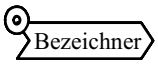
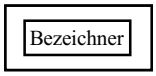
Produkttyp	Symbol	Produkttyp	Symbol
Aspekt		Variationstyp	
Anforderung		Variation	
		Version	
Systemarchitektur		Systemarchitektur-Version	
Systemmodul		Systemmodul-Version	
Modulschnittstelle		Modulschnittstellen-Version	
Softwareprodukt			
Basismodul			

Abbildung 17 – UML-Stereotypen für Entwicklungsprodukte

Assoziationstypen des Modells der Entwicklungsprodukte werden als Stereotypen der UML-Assoziationen dargestellt. Diese werden nicht durch eigene graphische Symbole, sondern mit der üblichen Sterotypen-Notation der UML als Kantenbeschriftung dargestellt. Abbildung 18 gibt hierfür einen Überblick über unterschiedliche Assoziationstypen. Eine UML-Assoziation mit Pfeil stellt hierbei eine gerichtete Assoziation des Produktmodells, eine UML-Assoziation ohne Pfeil eine ungerichtete Assoziation dar.

Assoziationstyp	Symbol	Assoziationstyp	
beeinflusst	«korreliert»	erwartet	«erwartet»
widerspricht	«widerspricht»	schließt aus	«schließt aus»
benötigt	«benötigt»	gewünscht	«gewünscht»

Abbildung 18 – UML-Stereotypen für Assoziationstypen

Dieser Ansatz integriert im Gegensatz zu bestehenden Ansätzen aus Abschnitt 5.1 verschiedene Entwicklungsprodukttypen in eine gemeinsame Darstellungsart. Damit können diese gleichzeitig innerhalb eines Diagramms dargestellt werden. Somit können sämtliche Beziehungen zwischen Entwicklungsprodukttypen übersichtlich dargestellt werden. Umgekehrt unterscheiden sich Entwicklungsprodukttypen durch verschiedene Stereotypen und deren graphischen Repräsentationen voneinander.

Zur übersichtlichen Darstellung verschiedener Gesichtspunkte des Modells der Entwicklungsprodukte können unterschiedliche Diagramme verwendet werden, in denen jeweils nur ein bestimmter Ausschnitt vorhandener Entwicklungsprodukte und entsprechender Assoziationen dargestellt wird. In Übersichtsdarstellungen können einzelne Anforderungen weggelassen werden. In den Phasen (vgl. Kapitel 5) des Requirements Engineerings komplexer Standardsoftware sind jeweils nur bestimmte Entwicklungsprodukttypen relevant. Entsprechend der Gliederung operativer Entwicklungsprodukte aus Abschnitt 4.4 sind daher folgende Darstellungen sinnvoll:

Globalsicht	Darstellungsname	Dargestellte Entwicklungsprodukttypen
Grundelemente	Externe Anforderungssicht	Aspekte und Anforderungen
Marktsicht	Externe Variationssicht	Variationstypen und Variationen
Systemsicht	Externe Systemarchitektursicht	Systemmodule und Modulschnittstellen
Systemsicht	Abbildung von Variationen auf Systemarchitektur	Variationen und Systemmodule
Entwicklungssicht	Produktsystem-Versionssicht	Systemmodule und Systemmodul-Versionen, Produktsystem-Versionen

Beispiel:

Die externe Systemarchitektursicht des Beispiels aus Abbildung 14 in Abschnitt 4.4.3, ergänzt um aggregierte Anforderungen, kann wie in Abbildung 19 dargestellt werden.

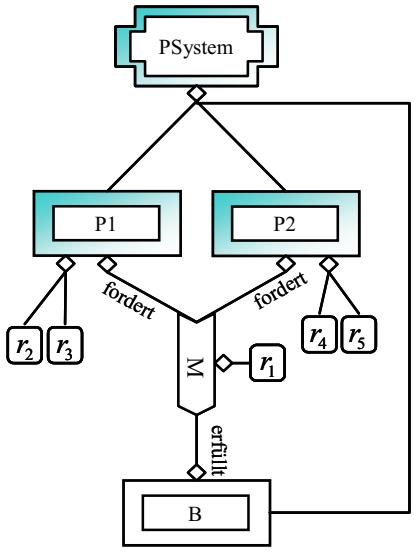


Abbildung 19 – Beispiel einer externen Systemarchitektursicht

5.2.2 Interne Spezifikationen

In diesem Unterabschnitt werden Erweiterungen vorhandener Beschreibungstechniken zur internen Spezifikation von Versionen, Variationen und Realisierungsständen vorgestellt. Wie bereits in Abschnitt 2.3 erläutert, erfolgt in der Praxis die inhaltliche Spezifikation von Anforderungen meist durch verschiedenartige Graphen und informelle Texte. Für graphische Beschreibungen setzt sich die Unified Modelling Language (UML [RJB99]) zunehmend durch.

Daher werden zunächst Erweiterungen graphischer Beschreibungstechniken am Beispiel der UML betrachtet. Im Anschluss daran wird auf informelle Texte eingegangen. Die im Folgenden vorgestellten Erweiterungen werden anhand der Darstellung von Versionen präsentiert. Variationen und Realisierungszustände können in analoger Weise beschrieben werden.

Graphische Spezifikationen

Graphische Beschreibungstechniken bestehen vorrangig aus unterschiedlichen Knoten und Kanten. Die Grundidee der Erweiterung vorhandener Techniken ist, dass Knoten unterschiedlicher Versionen verschieden dargestellt werden. Prinzipiell besteht die Möglichkeit, hierzu unterschiedliche Formen, Farben, Beschriftungen oder auch Rahmentypen einzelner Knoten zu verwenden. Da es sich bei der Darstellung von Versionen jedoch um eine Zusatzinformation handelt, sollte es nicht nötig sein, das grundsätzliche Aussehen bereits vorhandener Spezifikationen zu verändern.

Hierzu eignet sich am besten die Färbung des Hintergrunds der Fläche, auf dem ein Knoten in der Zeichenebene positioniert ist, wie Abbildung 20 illustriert. Verschiedene Versionen können damit optisch gut hervorgehoben werden, wobei gleichzeitig der Graph selbst nicht verändert werden muss.

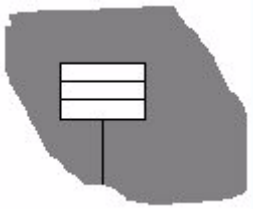


Abbildung 20 – Färbung des Hintergrunds eines Knotens

Für jeden Knoten eines Graphen muss eindeutig identifizierbar sein, mit welcher Farbe sein Hintergrund gefärbt ist. Dazu können aus der algorithmischen Geometrie bekannte Voronoi-Flächen (vgl. [PS85]) genutzt werden. Die endlich vielen Knoten eines Graphen können idealisiert durch die gleiche Anzahl ausgezeichneter Punkte einer Ebene dargestellt werden. Zu jedem dieser Punkte kann eine sogenannte Voronoi-Fläche bestimmt werden. Eine Voronoi-Fläche eines ausgezeichneten Punktes ist die Menge aller Punkte der Ebene, die diesem Punkt näher als allen anderen ausgezeichneten Punkten sind. Die Zeichenebene kann so eindeutig in unterschiedliche Flächen unterteilt werden. Abbildung 21 zeigt beispielsweise die Unterteilung eines UML-Klassendiagramms in Voronoi-Flächen einzelner Klassenknoten.

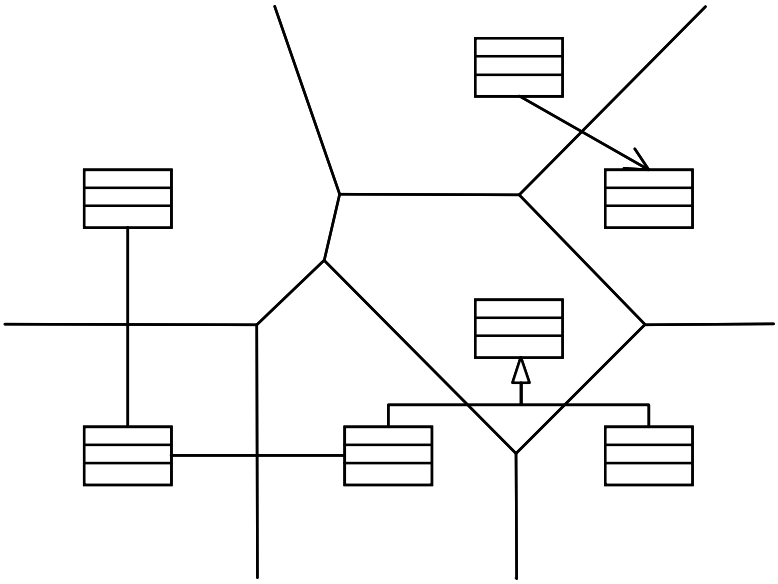


Abbildung 21 – Voronoi-Flächen eines UML -Klassendiagramms

Angenommen, die Realisierung der Klassen eines derartigen Klassendiagramms ist in drei unterschiedlichen Versionen geplant. Die Version, in der ein einzelner Knoten real i-siert werden soll, lässt sich nun eindeutig durch eine entsprechende Färbung der zugeh ö-rigen Voronoi-Fläche darstellen. In Abbildung 22 wird zum Beispiel die Versions-planung des UML-Klassendiagramms aus Abbildung 21 gezeigt.

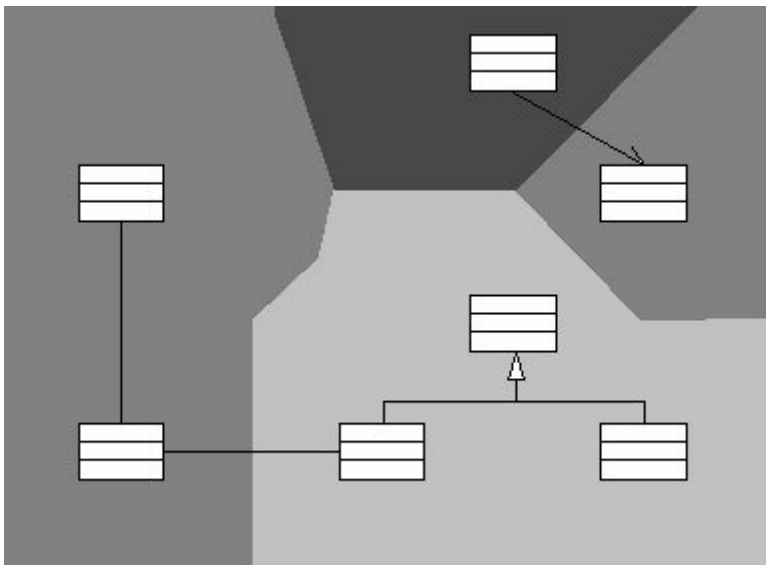


Abbildung 22 – UML-Klassendiagramm mit farbigen Hintergründen

Voronoi-Flächen können mit Hilfe geometrischer Algorithmen effizient berechnet we r-den (vgl. [PS85]). Daher kann die Färbung jederzeit auch nachträglich automatisch in graphischen Repräsentationen generiert werden.

In Abschnitt 4.4.4 wurde von unterschiedlichen Dauerhaftigkeiten der Planung von Anforderungen gesprochen. Diese können zusätzlich in der Färbung des Hintergrunds berücksichtigt werden. So kann für permanente Anforderungen und bedingte Anforderungen eine eigenständige Farbe verwendet werden. Handelt es sich hingegen um eine Multi-Versions-Anforderung, so kann diese durch einen abwechselnd mit den Farben dieser Anforderung gestreiften Hintergrund dargestellt werden.

Diese Färbungstechnik kann ebenso zur Darstellung von Variationstypen verwendet werden. Hierzu werden Hintergründe unterschiedlicher Variationen eines Variationstypen mit verschiedenen Farben dargestellt. Der gemeinsame Kern erhält analog zu permanenten Anforderungen der Versionsdarstellung ebenfalls eine gesonderte Farbe.

Textuelle Spezifikationen

Graphische Beschreibungstechniken ermöglichen immer nur die Beschreibung eines Teils der in Entwicklungsprodukten zu erfassenden Information. Daher bedarf es in jedem Fall einer zusätzlichen Ergänzung durch strukturierten informellen Text. Die am häufigsten angewendeten Strukturierungsprinzipien sind Textuntergliederungen in Abschnitte und Tabellen.

In Abbildung 23 ist ein Beispiel eine mögliche Spezifikation eines Versionsplans eines Softwareprodukts dargestellt. Der Versionsplan ist der aktuellen Status einer bestimmten Systemmodul-Version eines Systemmoduls (in diesem Fall eines Softwareprodukts). Diese kann automatisch aus vorhandenen Entwicklungsprodukten und zugehörigen Assoziationen generiert werden.

Im Folgenden wird erklärt, wie sich diese Spezifikation aus einzelnen Elementen des Modells für Entwicklungsprodukte zusammensetzt:

① Attributbelegungen der spezifizierten Systemmodul-Version und des Softwareprodukts, zu dem die Systemmodul-Version aggregiert ist.

② Attribut der assoziierten Version.

③ Berechneter Realisierungszustand der Systemmodul-Version. Wie aus dem formalen Modell aus Abschnitt 4.4.4 zu entnehmen ist, ist eine Systemmodul-Version genau dann realisiert, wenn sämtliche vorher als geplant aggregierten Anforderungen realisiert sind.

④ Systemarchitektur-Version, zu dem eine Systemmodul-Version gehört. Diese kann mit Hilfe des Entwicklungsaxioms *Zuordnung von Systemmodul-Versionen zu Systemarchitektur-Versionen* aus Abschnitt 4.4.4 automatisch berechnet werden

⑤ Zu der Systemmodul-Version als *geplant* oder *realisiert* aggregierte Anforderungen.

⑥ Wegen des Entwicklungsaxioms *Zuordnung von Anforderungen zu Aspekten* aus Abschnitt 4.2.2 gehört jede Anforderung mindestens einem Aspekt an. Dadurch können Aspekte zur Gliederung der Anforderungen einzelner Systemmodule herangezogen werden. Die Anforderungen sind in dieser speziellen Spezifikation nach Aspekten und Variationstypen gegliedert. Die entsprechenden Aspekte und Variationstypen können indirekt über die dem Softwareprodukt aggregierten Anforderungen ermittelt werden.

Softwareprodukt XYZ (1)	VERSIONSPLAN	**V 3.1** (2)			
System-Version	**V 3.0** (4)	Datum:	15.04.01		
Zykluszeit (1)	von **01.04.01**	bis **30.09.01**	Realisierung: **geplant** (3)		
Aspekt	**Aspekt 1** (6)				
Anford.bez.	Beschreibung	Vers.	VT	RStat.	Prio
Anforderung 1 (5)	Diese Beschreibung sagt, was Anforderung 1 zu leisten hat	V 3.1	N	gepl (7)	1
Anforderung 2	Beschreibung von Anforderung 2	V 3.1	N	real	20
Anforderung 3	Beschreibung von Anforderung 3	V 3.1	N	gepl	23
Anforderung 4	Beschreibung von Anforderung 4	V 4.0	N	gepl (8)	
Aspekt	**Aspekt 2**				
Anford.bez.	Beschreibung	Vers.	VT	RStat.	Prio
Anforderung 5	Beschreibung von Anforderung 5	V 3.1	N	gepl	3
Anforderung 2	(siehe Aspekt 1) (9)	V 3.1	N	real	20
Anforderung 6	Beschreibung was Anforderung 6 zu leisten hat	V 3.1	P	real	24
Anforderung 7	Beschreibung von Anforderung 7	V 4.0	N	gepl	
Anforderung 8	Beschreibung von Anforderung 8	V 4.0	N	gepl	
Variationstyp	**Variationstyp 1** (6)				
Anford.bez.	Beschreibung	Vers.	VT	RStat.	Prio
Variation 1	Beschreibung von Variation 1	V 3.1	N	gepl	4
Var 1 Anf 9	Beschreibung von Anforderung 9	V 3.1	N	real	22
Var 1 Anf 10	Beschreibung von Anforderung 10	V 3.1	M	gepl	25
Variation 2	Beschreibung von Variation 2	V 3.1	N	gepl	5
Var 2 Anf 11	Beschreibung von Anforderung 11	V 3.1	N	real	17
Var 2 Anf 12	Beschreibung von Anforderung 12	V 3.1	N	gepl	19
Gemäß VT 1 (10)	Beschreibung von Variation 2				
GVT 1 Anf 13	Beschreibung von Anforderung 13	V 3.1	N	real	26
GVT 1 Anf 14	Beschreibung von Anforderung 14	V 3.1	N	gepl	14

Abbildung 23 – Versionsplan eines Softwareproduktes

(7) Attribute von Anforderungen beziehungsweise Informationen, die aus dem Modell für Anforderungen, die zu einer bestimmten Systemmodul-Version aggregiert sind, extrahiert werden können.

(8) Darstellung der vorausschauenden Planung von Anforderungen in künftigen Versionen. Anforderungen, deren Realisierung in künftigen Versionen des Softwareprodukts geplant sind, können gesondert gekennzeichnet werden. Hier ist der Hintergrund des Textes in der gleichen Farbe dargestellt, wie die selbe Version auch in graphischen Spezifikationen (siehe unmittelbar vorhergehenden Unterabschnitt).

(9) Da Anforderungen mehreren Aspekten gleichzeitig angehören können, kann in dieser Art der Spezifikation eine Anforderung mehrmals in einer Tabelle auftauchen. Hier wurde eine Referenz auf ihr erstes Auftreten eingefügt. Stattdessen können jedoch auch genauso gut alle notwendigen Informationen wiederholt dargestellt werden.

(10) Hierarchische Darstellung von Anforderungen. Eine Variation als Spezialisierung von Anforderungen wird prinzipiell gleich dargestellt, wie alle anderen Anforderungen

dargestellt. Im Beispiel sind Anforderung 11 und Anforderung 12 Teilanforderungen der Variation 2. Dies wird in der Tabelle durch Wiederholung der Bezeichnung der Variation in den entsprechenden Zeilen der Anforderungen gekennzeichnet.

Wie in diesem Beispiel zu sehen ist, setzt sich eine textuelle Spezifikation eines Entwicklungsproduktes (hier einer Systemmodul-Version) aus unterschiedlichen Bestandteilen zusammen. Diese bestehen immer direkt oder indirekt aus den Attributen, den Assoziationen, den Aggregationen und den formal in Kapitel 4 formulierten Zusammenhängen verschiedener Entwicklungsprodukte. Eine Spezifikation ist also nicht eine isolierte Betrachtung einzelner Entwicklungsprodukte, sondern stellt meist einen mehrere Entwicklungsprodukte gleichzeitig umfassenden Ausschnitt des Modells der Entwicklungsprodukte dar. Umgekehrt müssen nicht alle Attribute und Zusammenhänge eines Entwicklungsproduktes spezifiziert sein. So wurden in der textuellen Spezifikation des obigen Beispiels alle Zusammenhänge zu Modulschnittstellen-Versionen weggelassen.

Dies ist dadurch möglich, dass sämtliche Informationen in einem Modell festgehalten sind und lediglich bei Bedarf extrahiert werden. Es besteht die Möglichkeit, ähnlich wie bei einer Datenbank, mit Hilfe vordefinierter Schemata die zu einem bestimmten Augenblick benötigte Information automatisch zu generieren. Durch die Generierbarkeit wird darüber hinaus die Änderungsflexibilität der gesamten Informationen des Requirements Engineerings unterstützt und diese ist jederzeit verfolgbar.

Für einzelne Aktivitäten des Prozessmodells aus Kapitel 5 können zusätzlich jeweils spezifische Ausschnitte des Modells der Entwicklungsprodukte generiert werden. Diese unterstützen dann einzelne auszuführende Arbeitsschritte wesentlich.

Kapitel 6

Bewertungsmethode

Für einen entscheidungsorientierten Entwicklungsprozess (vgl. Abschnitt 3.1.2) ist die Bewertung von Alternativen einer Entscheidungssituation ein wesentliches Element. In diesem Kapitel werden neue Konzepte für eine Bewertungsmethode von Anforderungen vorgestellt. Abschnitt 6.1 stellt zunächst grundlegende Elemente einer Bewertung als Erweiterung des Modells der Entwicklungsprodukte aus Kapitel 4 vor. Anschließend werden in Abschnitt 6.2 notwendige Merkmale von Bewertungsmethoden beschrieben. Der prinzipielle Aufbau bestehender Bewertungsmethoden wird in Abschnitt 6.3 erläutert. Einzelne Bausteine werden dabei hinsichtlich der Merkmale aus Abschnitt 6.2 qualitativ beurteilt. Abschnitt 6.4 stellt bestehende Bewertungsmethoden vor und zeigt, welche Bausteine aus Abschnitt 6.3 diese verwenden. Daraus resultiert eine Liste von Verbesserungspotentialen, die in einer neuen Bewertungsmethode münden. Diese wird in den Abschnitten 6.5 bis 6.7 vorgestellt. Abschnitt 6.5 präsentiert hierzu zunächst Kriterien zur Bewertung von Anforderungen komplexer Standardsoftware. Schließlich werden in den Abschnitten 6.6 und 6.7 verbesserte Methoden für ausgewählte Bausteine einer Bewertungsmethode präsentiert. Der Einsatz der Bewertungsmethode wird in Kapitel 7 im Rahmen eines durchgehenden Beispiels für den dort vorgestellten Requirements Engineering Prozess illustriert.

6.1 Elemente einer Bewertung

Dieser Abschnitt dient der Festlegung eines einheitlichen Begriffsverständnisses für Bewertungen und der Erweiterung des Modells der Entwicklungsprodukte aus Kapitel 4 um Entwicklungsprodukte für die Bewertung von Anforderungen. Die in diesem Kapitel vorgestellten Konzepte werden anhand eines einfachen Beispiels erläutert:

Beispiel: FinSoft

Der Standardsoftwarehersteller FinSoft entwickelt ein Produktsystem, das unter anderem aus einem Softwareprodukt zur Finanz- und Bankkontenverwaltung besteht. Herr Maier von der Firma X nennt die Anforderung "Bezahlungen im Internet sollen über eine Payphone-Funktionalität abgewickelt werden können" in einem Verkaufsgespräch mit Herrn Schmidt von der örtlichen Niederlassung von FinSoft in München. Die Anforderung ist eine von über 200 Anforderungen, die bewertet werden sollen. Die Bewertung der Anforderungen erfolgt – da der Firma FinSoft die Bewertungsmethode der vorliegenden Arbeit nicht bekannt ist – grob nach einem Kosten-/Nutzenverhältnis durch einen Entwickler. Dieser bewertet sowohl die Kosten als auch den Nutzen aller Anforderungen in einer Skala von 1 bis

5 und teilt anschließend den Nutzen durch die Kosten. Obige Anforderung wird mit Kosten=4 und Nutzen=3, also einem Nutzen-/Kostenverhältnis von ¾ eingeschätzt. Die Anforderungen werden dann in eine Reihenfolge entsprechen diesem Verhältnis gebracht. In der gleichen Reihenfolge soll ihre Realisierung erfolgen. Mit dieser Vorgehensweise auf ein möglichst hohes Nutzen-/Kostenverhältnisses der Entwicklung abgezielt.

Die Tätigkeit der Bewertung – oder kurz *Bewertung* genannt – erfolgt im Rahmen von Aktivitäten (vgl. Abschnitt 3.1.1) und wird von daran beteiligten Rollen durchgeführt. Im obigen Beispiel *FinSoft* ist der Entwickler die durchführende Rolle. Wie in Abschnitt 7.5 zu sehen sein wird, kann sich die Bewertung von Anforderungen auch über mehrere Aktivitäten erstrecken.

Für die Bewertung werden mehrere neue Entwicklungsprodukte benötigt. Abbildung 24 gibt einen Überblick über Erweiterungen des Modells der Entwicklungsprodukte aus Kapitel 4.

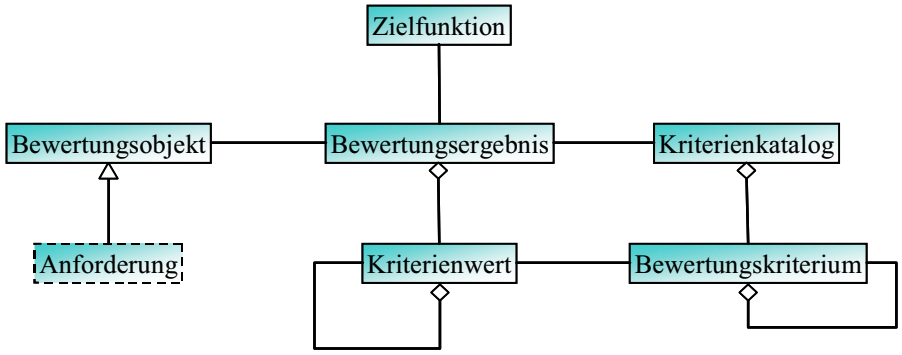


Abbildung 24 – Entwicklungsprodukte zur Bewertung

In den folgenden Absätzen werden die Entwicklungsprodukte einer Bewertung mit Hilfe des in Abschnitt 3.2.3 vorgestellten Schemas erläutert. Zentrales Entwicklungsprodukt ist das Bewertungsobjekt.

Produkttyp Bewertungsobjekt	
B	Darunter wird das zu bewertende Objekt verstanden.
Z	Dieses Entwicklungsprodukt lediglich das Bindeglied zwischen Bewertungen und beliebigen Entwicklungsprodukten, die bewertet werden sollen. Der Entwicklungsproduktty p kann nicht selbst instantiiert werden. Nur entsprechende Verfeinerungen, wie z.B. die Anforderung in Abbildung 24, werden instanziiert.
A	-
G	-
B=Beschreibung, Z=Zweck, A=Attribute, G=Aggregationen	

Beispiel:

Im Beispiel FinSoft *sind Anforderungen Bewertungsobjekte.*

Im Rahmen dieses Kapitels werden ausschließlich Anforderungen bewertet. Die in den vorgestellten Bewertungsmethoden können jedoch prinzipiell zur Bewertung beliebiger Bewertungsobjekte herangezogen werden. Bewertungsobjekte können beispielsweise Alternativen beliebiger Entscheidungssituationen eines entscheidungsorientierten Entwicklungsprozesses (vgl. Abschnitt 3.1.2) sein.

Die Bewertung eines Bewertungsobjektes wird anhand eines *Kriterienkatalogs* durchgeführt, der unterschiedliche *Bewertungskriterien* aggregiert. Verschiedene Anforderungen werden immer mit dem gleichen Kriterienkatalog und damit den gleichen Bewertungskriterien bewertet. Das aus der Bewertung resultierende Ergebnis (*Bewertungsergebnis*) aggregiert Kriterienbewertungen, die zu dessen Ermittlung beigetragen haben. Das Bewertungsergebnis dokumentiert den Wert eines Bewertungsobjektes im Verhältnis zur Zielfunktion (vgl. auch Abschnitt 3.1.2).

Beispiel:

Der Kriterienkatalog des Beispiels FinSoft besteht aus den beiden Bewertungskriterien Nutzen (Wert der Kriterienbewertung=3) und Kosten (Wert der Kriterienbewertung=4). Das Bewertungsergebnis stellt das Nutzen-Kostenverhältnis dar (Wert des Bewertungsergebnisses = ¾). Zielfunktion für diese Bewertung ist die Maximierung des Nutzen-/Kostenverhältnisses der Softwareentwicklung.

Folgenden Tabellen beschreiben restliche Entwicklungsprodukttypen einer Bewertung:

Produkttyp Kriterienkatalog	
B	Ansammlung unterschiedlicher Bewertungskriterien, die zur Bewertung eines Bewertungsobjektes herangezogen werden können.
Z	Dokumentation der Kriteriensammlung und der Vorgehensweise der Ermittlung einer Bewertung von Bewertungsobjekten.
A	**Skalierung:** Hier wird festgelegt, welche unterschiedlichen Werte ein Bewertungsergebnis aller Einzelbewertungen haben kann. Zum Beispiel kann hier festgelegt werden, dass Werte zwischen 1 (sehr gut) bis 10 (sehr schlecht) liegen sollen.
G	**Bewertungskriterium**: Zum Katalog zugehörige Kriterien.
B=Beschreibung, Z=Zweck, A=Attribute, G=Aggregationen	

Produkttyp Bewertungskriterium	
B	Einzelnes Kriterium zur Bewertung eines Bewertungsobjektes.
Z	Beschreibung des Kriteriums und des Vorgehens zur Ermittlung einer Einzelbewertung.
A	- **Beschreibung:** Beschreibt, was durch das Kriterium genau bewertet werden soll. - **Skalierung:** Hier wird festgelegt, welche unterschiedlichen Werte eine Bewertung hinsichtlich dieses Kriteriums annehmen kann. Wie die Skalierung des Kriterienkatalogs, kann auch hier beispielsweise eine Skala von 1 bis 10 möglich sein. - **Vorgehen der Bewertung:** Beschreibung, wie Einzelbewertungen des Kriteriums ermittelt werden. Beispielsweise können paarweise Vergleiche eingesetzt werden.
G	-
B=Beschreibung, Z=Zweck, A=Attribute, G=Aggregationen	

Produkttyp Zielfunktion	
B	Dokumentation der Zielfunktion (vgl. Abschnitt 3.1.2), anhand derer Bewertungsobjekte bewertet werden sollen und eines Vorgehens, wie Bewertungen ermittelt werden sollen. Allgemein soll eine Bewertung um so höher ausfallen, je stärker ein Bewertungsobjekt die Zielfunktion erfüllt.
Z	Durch die Zielfunktion wird bestimmt, nach welchen Maßstäben Bewertungsobjekte bewertet werden sollen. Dadurch werden Alternativen miteinander vergleichbar, wodurch die Entscheidung für eine Alternative verfolgbar wird.
A	- **Ziel:** Beschreibung der Kriterien, die optimiert werden sollen. - **Vorgehen der Bewertung:** Beschreibung, wie ein Bewertungsergebnis anhand einzelner Kriterienbewertungen ermittelt wird. Gegebenenfalls wird hier beschrieben, welche Gewichtung die einzelnen Bewertungskriterien haben.
G	-
B=Beschreibung, Z=Zweck, A=Attribute, G=Aggregationen	

Produkttyp Bewertungsergebnis	
B	Resultierender Wert einer Bewertung anhand verschiedener Bewertungskriterien.
Z	Durch das Bewertungsergebnis sollen verschiedene Bewertungsobjekte hinsichtlich der Erfüllung der Zielfunktion miteinander vergleichbar sein.
A	- **Wert:** Ausprägung der Skalierung des Kriterienkatalogs. - **Kommentar:** Begründung, wie diese Bewertung zustande kommt. Diese ist speziell dann notwendig, wenn das Vorgehen zur Gesamtbewertung eine Verhandlung vorschlägt.
G	**Kriterienwerte:** Alle Kriterienwerte des assoziierten Bewertungsobjektes werden aggregiert. Für jedes Bewertungskriterium des zum Bewertungsergebnis assoziierten Kriterienkatalogs muss genau eine Kriterienbewertung festgelegt sein.
B=Beschreibung, Z=Zweck, A=Attribute, G=Aggregationen	

Produkttyp Kriterienwert	
B	Bewertung eines Bewertungsobjektes hinsichtlich eines einzelnen Kriteriums.
Z	Dokumentation einer Einzelbewertung eines Bewertungsobjekts.
A	**Wert:** Eine Ausprägung der Skalierung des zugehörigen Kriteriums.
G	-
B=Beschreibung, Z=Zweck, A=Attribute, G=Aggregationen	

Die Aufwandsschätzung in der Software-Entwicklung ist sehr schwierig. Deshalb kann zum Zeitpunkt der Bewertung noch keine endgültige Entscheidung getroffen werden kann, welche Anforderung tatsächlich realisiert werden können. Im Beispiel *FinSoft* wurden daher die Anforderungen zum Schluss der Bewertung in eine Reihenfolge gebracht. Die Aufstellung einer derartigen Reihenfolge wird auch *Priorisierung* genannt. Die Position einer einzelnen Anforderung in der Reihe bezeichnet man auch als *Priorität*. Bewertungsobjekte mit hoher Priorität haben Vorrang vor der Auswahl von Bewertungsobjekten mit niedrigerer Priorität. Prioritäten können unmittelbar durch eine Sortierung der Bewertungsergebnisse abgeleitet werden.

6.2 Merkmale einer Bewertungsmethode

Nachdem im vorhergehenden Abschnitt 6.1 grundlegende Elemente einer Bewertung vorgestellt wurden, stehen ab jetzt für den Rest des Kapitels Bewertungsmethoden im Mittelpunkt. Im Beispiel *FinSoft* wurden beispielsweise zwei Kriterien zur Bewertung einer Anforderung herangezogen und ad-hoc daraus ein Bewertungsergebnis berechnet. Die folgenden Abschnitte zielen nun darauf ab, eine Systematik in die Bewertung von Anforderungen zu bringen. Dazu werden zunächst erstrebenswerte Merkmale für einer Bewertungsmethode für komplexe Standardsoftware diskutiert.

Die Bewertung einer Anforderung muss widerspiegeln, welchen Beitrag sie zum Gesamterfolg einer Software leistet. Dieser misst sich letztendlich an der Erfüllung der in Abschnitt 1.1.1 ausführlich erläuterten strategischen Ziele Kundenorientierung, Zeit und Kosten. Durch die Bewertung werden Anforderungen gegeneinander priorisiert, womit die Planung und die flexible Umplanung von Produktsystemen, Systemmodulen beziehungsweise deren Versionen unterstützt wird. Umplanungen sind - trotz teilweise sehr hoher Kosten - bei falschen Aufwandsschätzungen beziehungsweise aufgrund neuer oder sich ändernder Anforderungen erforderlich. Von Umplanungen können sowohl die Versionsplanung der Realisierung als auch Inhalte einzelner Anforderungen betroffen sein. Die Auswahl von Kandidaten derartiger Umplanungen wird durch Prioritäten unterstützt.

Bestehende Bewertungsmethoden sind in der Praxis für manche Zwecke zu aufwendig (vgl. [Dei98a]). Daher wird häufig ad-hoc ohne Methode bewertet. Daraus resultierende Prioritäten sind meist wenig nützlich, da sich die Anforderungen nicht fein genug voneinander unterscheiden lassen. Für eine Bewertungsmethode sind also folgende Merkmale wesentlich:

1. Die Bewertung muss möglichst verfolgbar sein, das heißt es muss weitreichend nachvollziehbar sein, wie Bewertungsergebnisse zustande gekommen sind.
2. Anforderungen müssen in ihren Bewertungsergebnissen gut differenzierbar sein.
3. Der Aufwand einer Bewertungsmethode muss zum Zweck der Bewertung adäquat sein.

Beispiel:

Wie der Entwickler im Beispiel FinSoft zu einzelnen Kriterienbewertungen gekommen ist, ist nicht verfolgbar. Der weitere Weg zum Bewertungsergebnis ist hingegen nachvollziehbar, da dieser nach einem Berechnungsschema erfolgt ist.

Die Verfolgbarkeit kann vor allem durch einen erhöhten Bewertungsaufwand verbessert werden. Daher sollte für eine Methode abhängig von der Tragweite einer aus der Bewertung resultierenden Entscheidung mehr Aufwand erlaubt sein.

6.3 Bausteine von Bewertungsmethoden

Analysiert man bestehende Bewertungsmethoden hinsichtlich ihres Aufbaus, so kann man feststellen, dass diese prinzipiell gleich strukturiert sind. In diesem Abschnitt werden Bausteine von Bewertungsmethoden erläutert. Der anschließende Abschnitt stellt dann bestehende Methoden vor und ordnet diese hinsichtlich verwendeter Bausteine ein.

Sämtliche Kriterienbewertungen (vgl. Abschnitt 6.1) beruhen auf Schätzungen einzelner oder einer Gruppe von Personen. Dadurch sind keine absolut messbaren Aussagen über die Güte der ermittelten Werte möglich. Mit jeder Methode wird daher vom Anwender ein gewisses Vertrauen in die ermittelten Ergebnisse erwartet. Zur Erhöhung dieses Vertrauens sollte bei Bewertungen möglichst weitreichend auf vorhandene Erfahrungen zurückgegriffen werden können.

Der folgenden Einschätzung von Bausteinen einer Bewertungsmethode liegen folgende zwei Prämissen zu Grunde:

1. Ein Bewertungsergebnis wird um so zuverlässiger und verfolgbarer, je mehr Bewertungskriterien zu Rate gezogen werden. Die Prämisse basiert auf der Annahme, dass ein Urteil umso genauer wird, je mehr Gesichtspunkte zur Urteilsfindung in Betracht gezogen werden.

2. Ein Bewertungsergebnis wird um so zuverlässiger, je systematischer es aus gewonnenen Erfahrungen abgeleitet wird.

In den folgenden Abschnitten werden Bausteine von Bewertungsmethoden vorgestellt und hinsichtlich zu erwartendem Aufwand der Bewertung und Verfolgbarkeit und der Differenzierbarkeit resultierender Bewertungsergebnisse eingeschätzt.

6.3.1 Anzahl der Bewertungskriterien

Der erste Baustein einer Bewertungsmethode ist die Anzahl der Bewertungskriterien, anhand derer eine Anforderung bewertet wird. Es wird zwischen der direkten Bewertung und der Bewertung mit verschiedenen Bewertungskriterien unterschieden.

Direkte Bewertung

Bei der direkten Bewertung wird jede Anforderung mit einem einzigen Bewertungskriterium beurteilt. Die Werte einzelner Anforderungen können unmittelbar zur Bestimmung ihrer Priorität genutzt werden.

Aufwand: Für jede Anforderung muss nur ein einziger Wert ermittelt werden. Daher ist der Aufwand der direkten Bewertung verglichen mit der Bewertung mit verschiedenen Kriterien gering.

Verfolgbarkeit: Die Verfolgbarkeit derart ermittelter Bewertungsergebnisse ist relativ gering. Falls z. B. implizit widersprüchliche Kriterien den Wert beeinflussen, ist dies völlig intransparent.

Differenzierbarkeit: Ein einziger Wert ermöglicht keine differenzierte Bewertung.

Bewertung durch verschiedene Bewertungskriterien

Die Bewertung einer Anforderung erfolgt bei der *Bewertung nach verschiedenen Bewertungskriterien* in zwei Stufen. In der ersten Stufe werden für jede Anforderung Kriterienbewertungen vorgenommen. Die zweite Stufe setzt diese nach einer vorgegebenen Systematik in Bezug zueinander und ermittelt somit das Bewertungsergebnis. Im Beispiel *FinSoft* wurde eine Bewertung durch verschiedene Bewertungskriterien vorgenommen.

Aufwand: Mit der Anzahl der Bewertungskriterien steigt der Aufwand sowohl wegen zunehmenden einzelnen Kriterienbewertungen als auch wegen dem steigenden Aufwand für Ermittlung des Bewertungsergebnisses an. Daher ist die Bewertung mit verschiedenen Bewertungskriterien in der Regel aufwändiger als eine direkte Bewertung.

Verfolgbarkeit: Der Prämisse zufolge steigt die Verfolgbarkeit einer Bewertung mit der Anzahl der Kriterien an.

Differenzierbarkeit: Mit der Zunahme der Kriterien lassen sich einzelne Bewertungsergebnisse besser voneinander unterscheiden.

6.3.2 Bewertung einzelner Kriterien

Zur Bewertung einzelner Kriterien sind zwei Verfahren üblich: Der paarweise Vergleich und die Skalenbewertung. Bei der Skalenbewertung kann zwischen direkter Skalenbewertung und indirekter Skalenbewertung unterschieden werden.

Die Bewertung von Anforderungen für Standardsoftware kann darüber hinaus auch mittels Umfragen durchgeführt werden. Diese sind jedoch mit einem hohen personellen und zeitlichem Aufwand verbunden. Sie können daher nur dann eingesetzt werden, wenn es sich um strategische Entscheidungen oder um Massenprodukte handelt. Vorgehensweisen zur Durchführung von Umfragen werden zum Beispiel in [Kot97] beschrieben. Die vorliegende Arbeit beschränkt sich auf Bewertungsmethoden von Anforderungen mit einer relativ geringen Anzahl Beteiligter. Diese sind eigenen Studien zufolge (vgl. [Dei98a]) für die Entwicklung komplexer Standardsoftware von höherer Relevanz. Statistische Methoden sind aufgrund der Notwendigkeit einer größeren Anzahl von Stichproben für derartige Bewertungen wenig geeignet und finden daher hier keine Verwendung. Auch die Anwendung von Umfragemethoden scheidet aus diesem Grunde für eine derartig geringe Anzahl beteiligter Personen aus.

Paarweiser Vergleich

Beim paarweisen Vergleich werden alle Anforderungen hinsichtlich eines Bewertungskriteriums durch an der Bewertung beteiligten Personen miteinander verglichen. Dieses Vorgehen wurde in der Analytic Hierarchy Process-Methode (kurz AHP-Methode vgl. [Saa80]) eingeführt. Das Vergleichsergebnis zwischen zwei Anforderungen wird mit

ganzzahligen Werten zwischen 1 und 9 ausgedrückt. Beim Wert 1 sind zwei Anforderu ngen gleich. Der Wert 9 bedeutet dagegen einen extremen Mehrwert der ersten Anford erung. Ist die erste Anforderung weniger wert, so wird das Vergleichse rgebnis mit dem reziproken Wert von 1 bis 1/9 aufgeschrieben. Die Bewertung eines Kr iteriums einer Anforderung wird durch eine Summenbildung über einzelne Vergleichsergebnisse ermi ttelt. Die Bewertung erfolgt ausschließlich relativ zu den vorhandenen Anforderu ngen. Unter bestimmten Annahmen können bei dieser Methode mathematische Plausibilität stests durchgeführt werden. Diese dienen der Erhöhung des Vertrauens in die Ergebnisse dieser Methode.

Aufwand: Der Aufwand der Bewertungsmethode hängt vor allem von der Anzahl zu vergleichender Anforderungen ab und steigert sich quadratisch. Daher ist bei einer gr ößeren Anzahl von Anforderungen ein relativ hoher Aufwand zur Bewe rtung notwendig.

Verfolgbarkeit: Die Ergebnisse des Verfahrens selbst sind durch die direkten Vergle iche und durch das einheitliche Berechnungsschema zur Ermittlung einer Gesamtbewe rtung transparent. Da jedoch alle Werte sich relativ an den vorhandenen Anforderungen orientieren, fehlt es an einer Vergleichbarkeit mit absoluten Werten.

Differenzierbarkeit: Da für ein Bewertungskriterium jede Anforderung mit jeder and eren Anforderung direkt verglichen wird, sind alle Kriterienbewertungen gut voneinander unterscheidbar.

Direkte Skalenbewertung

Skalenbewertungen basieren auf der Bewertung eines Kriteriums nach einer vorgegeb enen Skala. Die Bewertung erfolgt aufgrund einer reinen Schätzung. Im Be ispiel *FinSoft* wurde sowohl für die Kosten als auch für den Nutzen eine direkte Skalenbewertung a ngewendet. Die Skalen waren dabei jeweils Werte von 1 bis 5.

Aufwand: Da die Skalenbewertung pro Bewertungskriterium direkt erfolgt, ist der Au fwand der Bewertung gegenüber dem paarweisen Vergleich gering und pro A nforderung konstant.

Verfolgbarkeit: Die Bewertung selbst erfolgt aufgrund einer Einschätzung und ist damit nicht unmittelbar transparent. Gegebenenfalls besteht die Notwendigkeit zur Dokument ation der bewertenden Person, um spätere Nachfragen hinsichtlich der Einschätzung zu ermöglichen.

Differenzierbarkeit: Diese hängt für allem von der Wahl der Skala und der Möglichkeit der Einschätzung unterschiedlicher Kriterienbewertungen ab. Für manche Kriterien ist es oft nicht möglich, eine feine Unterteilung in mehrere Skalenwerte zu machen.

Indirekte Skalenbewertung

Bei der indirekten Skalenbewertung erfolgt die Beurteilung einer Anforderung ebenfalls aufgrund einer vorgegebenen Skala. Anstatt Anforderungen direkt zu bewerten, stützt sich diese Bewertung auf Erfahrungen in der Vergangenheit. Diese können in einem vorgegebenen Schema systematisch festgehalten werden, wie später in Abschnitt 6.7 noch ausführlicher dargestellt wird. Das Schema basiert auf unterschiedlichen Klassifik a-

tionen, mit deren Hilfe gesammelte Erfahrungen auf zu bewertende Anforderungen zugeordnet werden können. Bewertungen lassen sich durch Erfahrungen mehr objektivieren, da sie sich nicht mehr rein auf die Intuition der durchführenden Personen stützen. Die Verwendung der indirekten Skalenbewertung bedarf einer Vorbereitung, die in Abschnitt 6.7 detailliert erläutert wird.

Aufwand: Sowohl zur Vorbereitung als auch zur Durchführung der indirekten Skalenbewertung entstehen Aufwände. Die Aufwände der Vorbereitung sind unabhängig von der Anzahl der zu bewertenden Anforderungen. Daher schlagen sie umso weniger zu Buche, je mehr Anforderungen mittels der indirekten Skalenbewertung beurteilt werden. Die Zuordnung von Anforderungen zu Erfahrungen kann automatisiert werden. Damit ist der Aufwand für die Durchführung der indirekten Skalenbewertung verglichen zur direkten Skalenbewertung in etwa gleich hoch.

Verfolgbarkeit: Da indirekte Bewertungen auf Erfahrungen basieren, lassen sich diese besser nachvollziehen als direkte Bewertungen. Das Zutreffen einer Erfahrung kann jedoch nicht garantiert werden.

Differenzierbarkeit: Hier gilt das gleiche, wie für die direkte Skalenbewertung.

6.3.3 Ermittlung eines Bewertungsergebnisses

Zur Bestimmung des Bewertungsergebnisses der Bewertung mit verschiedenen Bewertungskriterien existieren zwei unterschiedliche Vorgehensweisen. Es kann berechnet oder im Rahmen einer Verhandlung zwischen unterschiedlichen Entwicklungsrollen (vgl. Abschnitt 7.2) ermittelt werden.

Berechnung des Bewertungsergebnisses

Um ein Bewertungsergebnis berechnen zu können, müssen einzelne Kriterienbewertungen als Zahlen vorliegen. Eine gebräuchliche Berechnungsart ist die gewichtete Summenbildung. Angenommen n verschiedene Bewertungskriterien sind von $k = 1,...,n$ durchnummeriert. Die Kriterienbewertung eines Kriteriums k hinsichtlich einer Anforderung $r \in R_t$ (Menge aller bekannten Anforderungen zum Zeitpunkt $t \in T$) sei mit $c_{r,k} \in \mathbb{N}^+$ bezeichnet. Jedes Bewertungskriterium wird mit einem Wert $w_k \in \mathbb{R}_0^+$ gewichtet. w_k wird dabei aufgrund von Erfahrungen vor der Berechnung der Gesamtbewertung geschätzt. Das Bewertungsergebnis $value_r$ einer Anforderung berechnet sich als folgende Summe:

$$value_r = \sum_{k=1}^{n} w_k c_{r,k} \ .$$

Für die gewichtete Summenbildung ist die richtige Kalibrierung einzelner Bewertungskriterien entscheidend. Wie stark ein Bewertungskriterium k die Gesamtbewertung beeinflusst, hängt sowohl vom Gewicht w_k als auch von der Intervalllänge möglicher Werte $c_{r,k}$ ab. Mit zunehmendem Gewicht und zunehmender Intervalllänge kann man die Bedeutung eines Kriteriums erhöhen. Kriterienbewertungen mit textueller Skala müssen auf Zahlen abgebildet werden. Kriterienbewertungen paarweiser Vergleiche müssen auf ein

von der Anzahl der Anforderungen unabhängiges Intervall normiert werden. Im Beispiel *FinSoft* wurde die gewichtete Summenbildung mit dem Gewicht 1 für beide Kriterien angewendet.

Aufwand: Die Ermittlung einer Gesamtbewertung kann automatisiert werden und stellt damit einen geringen Aufwand dar.

Verfolgbarkeit: Die schematische Berechnung erlaubt zwar eine gute Verfolgbarkeit der Ermittlung der Gesamtbewertung aus den Einzelbewertungen. Hinsichtlich der Interpretation des ermittelten Wertes ist jedoch einige Skepsis angebracht. So wird durch die Berechnung mit qualitativen Werten umgegangen, als ob sie absolut quantifizierbar und sogar addierbar wären. Durch die Berechnung geht in der Regel Information verloren, da unterschiedliche Kombinationen von Einzelbewertungen zur gleichen Gesamtbewertung führen können. Demgegenüber wird in der Praxis (vgl. [FOR97]) trotz eigentlich rechnerisch nicht kombinierbaren Einzelbewertungen, ein zu großes Vertrauen in derart ermittelte Gesamtbewertungen gesteckt. Das Vertrauen basiert auf der Annahme, dass Berechnungen keine falschen Ergebnisse liefern können. Daher entfallen häufig eigentlich nötige Plausibilitätsprüfungen berechneter Gesamtbewertungen.

Differenzierbarkeit: Diese hängt vor allem von der Wahl der Berechnungsformel ab. Daher können hierzu keine allgemeingültigen Aussagen getroffen werden.

Verhandlung eines Bewertungsergebnisses

Bei der Verhandlung werden Bewertungsergebnisse im Rahmen von Diskussionen gebildet. Daran beteiligen sich verschiedene Rollen mit unterschiedlichen Zielvorstellungen. Zur Ermittlung des Bewertungsergebnisses einer jeden Anforderung muss daher ein Konsens gefunden werden. Kriterienbewertungen dienen hierfür als Diskussionsgrundlage.

Aufwand: Da zur Ermittlung des Bewertungsergebnisses unterschiedliche Rollen beteiligt sein müssen, ist die Verhandlung mit hohem personellem Aufwand verbunden.

Verfolgbarkeit: Durch die Diskussion werden verschiedene Argumente zur Bestimmung eines Bewertungsergebnisses vorgebracht und gegeneinander abgewogen. Werden diese entsprechend dokumentiert, so lassen sich ermittelte Bewertungsergebnisse gut nachvollziehen. Die Verhandlung wird jedoch durch eine zunehmende Anzahl berücksichtigenden Kriterien erschwert. Daher muss hierfür einerseits die Anzahl direkt zu verhandelnder Kriterien möglichst gering gehalten werden. Praxiserfahrungen zufolge kann mit etwa vier bis fünf Kriterien gut verhandelt werden (vgl. [FOR97]). Will man möglichst viele Kriterien zur Bewertung aufgrund der eingangs erwähnten Prämisse heranziehen, so bietet sich als Kompromiss für diese eine hierarchische Strukturierung an. Für die Verhandlung mit mehreren Kriterien bedarf es zusätzlich einer besonderen visuellen Unterstützung. In dieser Arbeit wird hierfür in Abschnitt 6.6 eine besondere Technik vorgestellt, die dieses bewerkstelligt.

An der Verhandlung eines Bewertungsergebnisses werden alle Rollen beteiligt, die an der Bewertung von Anforderungen interessiert sind. Daher ist für diese unmittelbar die Entstehungsgeschichte der Bewertungsergebnisse nachvollziehbar. Gerade dadurch besteht auch insgesamt ein hohes Vertrauen in die Richtigkeit der ermittelten Werte.

Differenzierbarkeit: Je nach Zielvorstellung kann das Bewertungsergebnis unterschiedlich dokumentiert werden. Davon abhängig ist auch dessen Differenzierbarkeit.

6.4 Einordnung bestehender Bewertungsmethoden

In diesem Abschnitt werden bekannte Bewertungsmethoden vorgestellt und hinsichtlich der Bausteine von Bewertungsmethoden eingeordnet. Anhand der Einordnung werden Verbesserungspotentiale für eine neue Bewertungsmethode identifiziert. Die hier vorgestellten Methoden werden in der Literatur auch häufig als Priorisierungsmethoden bezeichnet, wobei ihre Vorgehensweisen sich grundsätzlich auf Bewertungen stützen.

6.4.1 Vorstellung bestehender Bewertungsmethoden

In diesem Abschnitt werden bestehende Bewertungsmethoden kurz vorgestellt. Hauptaugenmerk der Vorstellung ist dabei die Einordnung der einzelnen Methoden in die in Abschnitt 6.3 vorgestellten Bausteine von Bewertungsmethoden. Daraus lassen sich unmittelbar Unterschiede einzelner Bewertungsmethoden untereinander als auch Vor- und Nachteile ableiten. Daher wird hierauf nicht explizit eingegangen.

Analytic Hierarchy Process

Eine in der Literatur sehr häufig zitierte Methode zur Bewertung von Anforderungen ist die Analytic Hierarchy Process-Methode (kurz AHP, vgl. [Saa80]). Deren zentraler Kern ist eine direkte Bewertung mit paarweisen Vergleichen. Problem bei AHP ist die quadratische Zunahme von zu vergleichenden Paaren in Abhängigkeit von der Anzahl von Anforderungen. Diese können durch eine Hierarchisierung der Anforderungen reduziert werden. Vergleiche erfolgen lediglich auf den oberen Hierarchieebenen. Anforderungen unterer Ebenen erben dann die Priorität ihrer Väter.

Quality Function Deployment

Auch Quality Function Deployment (kurz QFD, vgl. [Zul92, Bro91]) wird häufig im Zusammenhang mit Priorisierung genannt. Dies ist im eigentlichen Sinne keine Bewertungsmethode für Anforderungen, sondern ein Prozessmodell zur Entwicklung von Softwareprodukten. Der Hauptfokus von QFD liegt auf der Verfolgbarkeit bereits bewerteter Anforderungen und auf der Optimierung von Entwicklungsaufwänden. Kern von QFD sind verschiedene Matrizen zur übersichtlichen Darstellung von Assoziationen zwischen Entwicklungsprodukten unterschiedlicher Phasen. Entwicklungsprodukte einer Phase werden dabei in den Zeilen einer Matrix aufgelistet, die der anderen Phasen in den Spalten. Eine Assoziation zwischen zwei Entwicklungsprodukten wird durch einen Eintrag an der Zelle, an der sich entsprechende Zeile und Spalte kreuzen, dargestellt. Bei QFD sind die Einträge meist Zahlen, die fallabhängig unterschiedlich interpretiert werden.

Bekannt wurde QFD vor allem durch das House of Quality (vgl. [HaC88]). Dieses unterstützt die Auswahl zu entwickelnder technischer Funktionalitäten eines Pr oduktes durch die Berücksichtigung von Kundenprioritäten, Konkurrenzbewertungen und eigener tec hnischer Fähigkeiten. Ein Berechungsschema schreibt vor, wie verschiedene Bewertung skriterien einzelner Funktionalitäten zu einem Gesamtwert summiert werden mü ssen. Nach [Kar97] ist das House of Quality ohne Anpassung nicht für die Software-Entwicklung geeignet. Insbesondere wird es als sehr aufwändig beu rteilt.

QFD dient in der Praxis häufig als Schlagwort. Meist hat dessen Verwendung keinen direkten Bezug zur QFD-Methode. Häufig wird bereits die Verwendung von Matrizen oder von einem dem House of Quality ähnlichen Berechnungsschema als QFD bezeic hnet. Derartige Matrizen werden in der Pr axis zur übersichtlichen Darstellung der direkten Skalenbewertung mit verschied enen Bewertungskriterien verwendet.

Priorisierung von Karlsson und Ryan

Ein weiterer Ansatz zur Bewertung von Anforderungen wird in [KaR97] präsentiert. Diese Methode nutzt den paarweisen Vergleich, um für jede Anforderung relative Ante ile des Gesamtnutzens und der Gesamtkosten zu ermitteln. Die Gesamtb ewertung erfolgt in einer Verhandlung anhand der graphischen Darstellung der Einzelb ewertungen. Zur optischen Veranschaulichung werden die Werte jeder Anforderung in einem zweidime nsionalen Koordinatensystem eingetr agen.

Bewertung von Wiegers

In [Wie99a, Wie99b] wird ein Ansatz zur Bewertung anhand der Kriterien Kundennu tzen, Strafe, Entwicklungskosten und Risiko vorgestellt. Dabei we rden mit der Strafe eventuelle negative Auswirkungen bewertet, falls bestimmte Anforderungen nicht real isiert werden. Das Risiko bewertet unvorhersehbare Abweichungen von den g eschätzten Kosten und zur Verfügung stehender Ressourcen. Die Gesamtbewertung wird als g ewichtete Summe der vier Werte berechnet. Die berechneten Werte werden in einer Ve rhandlung validiert. Die Bewertung einzelner Kriterien erfolgt durch vord efinierte Rollen.

6.4.2 Verbesserungspotentiale für Bewertungsmethoden

Die folgende Tabelle gibt einen Überblick über Bewertungsmethoden (vgl. Abschnitt 6.4.1) und verwendete Bausteine:

Methode	#Kriterien	Bewertungsmethode	Gesamtbew.
AHP	1	Paarweiser Vergleich mit Skala 1-9	---
House of Quality	4	Direkte Skala unterschiedlicher Werte	Berechnet
Karlsson	2	Paarweiser Vergleich mit Skala 1-9	Verhandelt
Wiegers	4	Direkte Skala 1-9	Berechnet

Durch die Betrachtung der Tabelle können Verbesserungspotentiale für Bew ertungsmethoden von Anforderungen komplexer Standardsoftware identifiziert we rden:

Anzahl Bewertungskriterien

Eingangs des Kapitels wurde die Prämisse aufgestellt, dass sich die Verfolgbarkeit einer Bewertung mit zunehmender Anzahl von Kriterien allgemein erhöht. Die vorhandenen Methoden verwenden lediglich maximal vier Kriterien. Vor allem berücksichtigen die vorgeschlagenen Kriterien nicht alle Ziele der Entwicklung komplexer Standardsoftware ausreichend. Damit bestehen Verbesserungspotentiale in der Erhöhung der Anzahl an Kriterien und in einer spezifischen Anpassung der Kriterien an die Problemstellungen von Standardsoftware. Dieser Arbeit schlägt deshalb in Abschnitt 6.5 einen geeigneten Kriterienkatalog zur Bewertung von Anforderungen komplexer Standardsoftware vor.

Bewertung einzelner Kriterien

Sämtliche Beurteilungen einzelner Kriterien in bestehenden Bewertungsmethoden werden rein intuitiv durch paarweise Vergleiche oder durch direkte Skalenbewertungen vorgenommen. Die Verfolgbarkeit der Urteilsfindung könnte nach der ersten Prämisse weiter verbessert werden, wenn Erfahrungen vergangener Bewertungen von Anforderungen systematisch herangezogen werden würden. Die vorliegende Arbeit schlägt eine Vorgehensweise zur systematischen Aufbereitung von Erfahrungen in Abschnitt 6.7 vor.

Gesamtbewertung

Wie bereits in der Vorstellung des entsprechenden Bausteins erläutert wurde, bestehen hinsichtlich der Aussagekraft berechneter Gesamtbewertungen erhebliche Unsicherheiten. Daher sollte bei wichtigen Entscheidungen eine Gesamtbewertung durch Verhandlungen erfolgen, da durch die Beteiligung wichtiger Rollen, diese für jeden einzelnen eher nachvollziehbar ist. Mit Ausnahme der Methode von Karlsson und Ryon spielt die Verhandlung in bestehenden Methoden keine ausgezeichnete Rolle. Karlsson und Ryon hingegen unterstützen die Diskussion von Bewertungen durch eine Visualisierung anhand eines Koordinatensystems. Dieses beschränkt sich allerdings auf lediglich zwei Kriterien. Um auch über eine größere Anzahl Kriterien sinnvoll verhandeln zu können, bedarf es hierfür einer verbesserten Visualisierungstechnik. Dazu wird in dieser Arbeit in Abschnitt 6.6 eine Portfoliotechnik vorgestellt.

Skalierung einer Bewertungsmethode

Eine verbesserte Verfolgbarkeit einzelner Gesamtbewertungen bedeutet meist mehr Aufwand für die Bewertung selbst. Eine Bewertungsmethode muss daher an Bedürfnisse einer Bewertungssituation angepasst werden können. Dies geschieht durch die Verwendung von jeweils adäquaten Bausteinen. Beispielsweise sollen bei hoher Relevanz einer Entscheidung paarweise Vergleiche eingesetzt, viele Kriterien herangezogen und die Gesamtbewertung verhandelt werden können. Demgegenüber soll bei niedriger Relevanz eine Gesamtbewertung von Anforderungen möglichst einfach ermittelt werden können. Daher sollten hier Skalentechniken mit möglichst geringer Skalenteilung, einer geringen Anzahl an Kriterien und gegebenenfalls eine automatische Gesamtbewertung mit Hilfe von Berechnungen verwendet werden können.

6.5 Bewertungskriterien für Anforderungen

In diesem Abschnitt werden Kriterien zur Bewertung von Anforderungen für die Entwicklung komplexer Standardsoftware vorgestellt. Die Kriterien wurden im Rahmen intensiven Diskussionen gemeinsam mit Herstellern komplexer Standardsoftware im Rahmen von [FOR97, BDJS99] erarbeitet. In Abschnitt 6.6 wird eine Methode zur Gesamtbewertung von Anforderungen vorgestellt, anhand der vier Kriterien visuell verhandelt werden können. Es existieren jedoch weitaus mehr Bewertungskriterien für Anforderungen an Standardsoftware. Daher gliedern sich diese hierarchisch unter vier Hauptkriterien, der Dringlichkeit, der Marktbedeutung, der technischen Wertigkeit beziehungsweise dem Aufwand und der Effizienz. Durch die Dringlichkeit und Markbedeutung bewerten Anforderungen aus einer eher verkaufsorientierten Sicht. Technische Wertigkeit beziehungsweise Aufwand und Effizienz spiegeln hingegen eine Bewertung von Anforderungen aus eine eher entwicklungsorientierten Sicht wider. Es folgt eine kurze Beschreibung der einzelnen Hauptkriterien:

1. **Dringlichkeit:** Zeigt an, wie zeitkritisch der Markteintritt einer neuen Anforderung ist. Je höher die Dringlichkeit, desto früher muss sie realisiert werden.

2. **Marktbedeutung:** Zeigt die Bedeutung einer Anforderung für den Markteintritt der Standardsoftware an. Hier wird Marktsegmentspezifisch auch der unterschiedliche Wert verschiedener Variationen berücksichtigt.

3. **Technische Wertigkeit**: Unter der Technischen Wertigkeit werden Auswirkungen der Realisierung von Anforderungen auf die *zukünftige* Weiterentwicklung (inkl. Vertrieb) der Standardsoftware verstanden.

4. **Aufwand und Effizienz**: Der Aufwand berücksichtigt den relativen Aufwand der Anforderung in der Software im Vergleich zu anderen Anforderungen. Die Effizienz vergleicht die eigenen Entwicklungsaufwände zur Konkurrenz und das Verhältnis zur Effizienz zu anderen Anforderungen hinsichtlich der Verfügbarkeit von Technologien. Die Effizienz relativiert den zu erbringenden Aufwand im Verhältnis zu anderen Zielen. So kann beispielsweise eine hohe Effizienz im Vergleich zur Konkurrenz bedeuten, dass eine Anforderung an sich zwar einen hohen Aufwand zur Realisierung bedeutet, aber die Konkurrenz noch mehr Aufwand hätte. Insofern wird dadurch ein Konkurrenzvorteil ausgedrückt.

Die folgende Tabelle gibt einen Überblick über sämtliche insgesamt erarbeiteten Kriterien zur Bewertung von Anforderungen von Standardsoftware. Aus den Kriterien kann vor der Durchführung einer Bewertung eine Auswahl erfolgen. Es müssen also nicht immer alle Kriterien gleichermaßen berücksichtigt werden. Vor allem sollen bei der Auswahl Strategische Vorgaben aus der Marktstrategie und der Systemstrategie berücksichtigt werden:

1	Dringlichkeit
1.1	Aktuelle Marktdurchdringung
1.2	Verbleibende Dauer bis zum voraussichtlichen Ende des Lebenszyklus
1.3	Popularität – Werbeeffekt
1.4	Technologieabhängigkeit
1.5	Weiterentwicklungsbasis
2	Marktbedeutung
2.1	Positive Differenzierung vom Wettbewerb durch Flexibilität
2.2	Weiterentwicklungsfähigkeit
2.3	Erfüllung der Kundenanforderungen nach KANO (siehe Glossar)
2.4	Kundenanteil
2.5	Substituierbarkeit
2.6	Marktvolumen
3	Technische Wertigkeit
3.1	Unmittelbar Entwicklungsbezogene Kriterien
3.1.1	Flexible Weiterentwicklung vorhandener Software
3.1.2	Änderbarkeit des zu entwickelnden Features für zukünftige Anforderungen
3.1.3	Wiederverwendbarkeit einer realisierten Funktionalität
3.2	Mittelbar Entwicklungsbezogene Kriterien
3.2.1	Abhängigkeit von fremder Entwicklungshard- und Software
3.2.2	Anforderungen an das Entwicklungspotential
3.3	Vertriebs- und Kundenserviceorientierte Kriterien
3.3.1	Diagnoseaufwand im Fehlerfall
3.3.2	Installationsaufwand
3.3.3	Variantenreduktion: Allgemeiner Logistikaufwand im Vertrieb
4	Aufwand und Effizienz
4.1	Aufwandsschätzung
4.2	Vorhandenes Know-how
4.3	Konkurrenzvergleich
4.4	Aufwand der Benutzerunterstützung
4.5	Entwicklungsstand der benötigten Technologien
4.6	Auswirkungen auf die Termintreue

Details zu den einzelnen Kriterien können in [BDJS01] nachgelesen werden.

6.6 Portfoliomethode zur Bewertung von Anforderungen

In diesem Anschnitt wird eine neuartige Methodik zur Bewertung von Anforderungen vorgestellt. Sie basiert auf der Portfolio-Analyse, die bereits zur Bewertung des Einkaufs von Waren und Lieferanten erfolgreich eingesetzt wird (vgl. [Wil99]).

Zentraler Kern der Portfoliomethode ist die Visualisierung von Kriterienbewertungen der Einzelkriterien. Diese unterstützt eine Verhandlung der Bewertungsergebni sse zwischen daran beteiligten Rollen. Wesentliche Entscheidungen in einem Softwareprojekt sollen so verhandelt werden. Wie bereits im vorangegangenen Abschnitt erwähnt, können mit Hilfe dieser Methode vier voneinander unabhängige Bewertungskriterien graphisch da rgestellt werden.

Zur Bewertung von Anforderungen mit Hilfe der Portfoliomethode wurden drei Schritte identifiziert:

1. **Ermittlung von Kriterienbewertungen**: Ermittlung von Informationen und Bewertungen einzelner Kriterien.
2. **Portfolio-Analyse**: Aufstellung verschiedener Portfolios zur visuellen Darstellung der Kriterienbewertungen. Diese unterstützen die nachfolgende Verhandlung des Bewertungsergebnisses.
3. **Verhandlung von Bewertungsergebnissen**: Ermittlung des Bewertungsergebnisses Gesamtbewertung beziehungsweise Modifikation von Inhalten basierend auf den Portfolios.

Die Portfolio-Technik ist auf eine langfristige Planung eines Softwaresystems ausgelegt. So können Bewertungsergebnisse bereits eingeordneter Anforderungen bei jedem neuen Versionszyklus weiterverwendet werden. Gegebenenfalls lassen sie sich auch dynamisch an die Planung neuer Versionen eines Softwaresystems oder Softwareproduktes anpassen. Bewertungsergebnisse können jederzeit angepasst werden, sobald neue Anforderungen identifiziert werden oder sich äußere Rahmenbedingungen ändern. Die drei Schritte der Bewertung können sich auf unterschiedliche Aktivitäten innerhalb des Requirements Engineering Prozesses verteilen. So teilt sich beispielsweise die Bewertung von Anforderungen in Kapitel 7 auf vier unterschiedliche Aktivitäten auf (siehe Abschnitte 7.5.1, 7.5.2 und 7.5.3). In diesem Abschnitt werden nur die Konzepte der Portfoliomethode vorgestellt. Ein ausführliches Beispiel zur Anwendung der Methode ist im Rahmen des Prozessmodells aus Abschnitt 7.5 zu finden.

6.6.1 Ermittlung von Kriterienbewertungen

Zur Vorbereitung der Portfolio-Analyse werden zuerst Anforderungen anhand der vier Hauptkriterien aus Abschnitt 6.5 bewertet. Die Bewertung kann je nach Bedarf und Aufwand beliebig an die konkrete Situation angepasst werden. Subsumiert eine Dimension lediglich ein Kriterium, so können hierzu die Verfahren angewendet werden, welche in Abschnitt 6.3.2 erläutert sind. Im einfachsten Fall erfolgt pro Kriterium eine pauschale Bewertung in Hoch/Niedrig.

Sind hingegen mehrere Bewertungskriterien in einer Dimension enthalten, so können bestehende Priorisierungsmethoden, wie beispielsweise Matrizen aus QFD oder die hier vorgestellte Portfoliomethode rekursiv angewendet werden.

6.6.2 Portfolio-Analyse

Die Portfolio-Analyse ermöglicht eine transparente Verknüpfung zueinander konkurri e-render Kriterien. Für diesen Zweck kombiniert dieser Ansa tz Informationen der vier verschiedenen Kriterien in zwei Stufen. In der ersten Stufe werden jeweils zwei Krite rien in getrennten Teilportfolios gegenübergestellt. Im Beispiel der Bewertung der Entwic k-lungssicht werden dafür ein Teilportfolio mit verkaufso rientierten Kriterien und ein Tei l-portfolio mit entwicklungsorientierte Kriterien aufgestellt. Die getrennte Bewertung spiegelt sich auch im Prozessmodell aus Abschnitt 7.5 wieder. Dort sind hierfür zwei isolierte Aktivitäten definiert. Diese werden in einer zweiten Stufe zu einem Gesamtpor t-folio kombiniert. Damit erreicht man eine systematische Gesamtbewertung, die verschi e-dene Einflussfaktoren klar voneinander trennt. Abbildung 25 zeigt die verwendeten Portfolios schematisch.

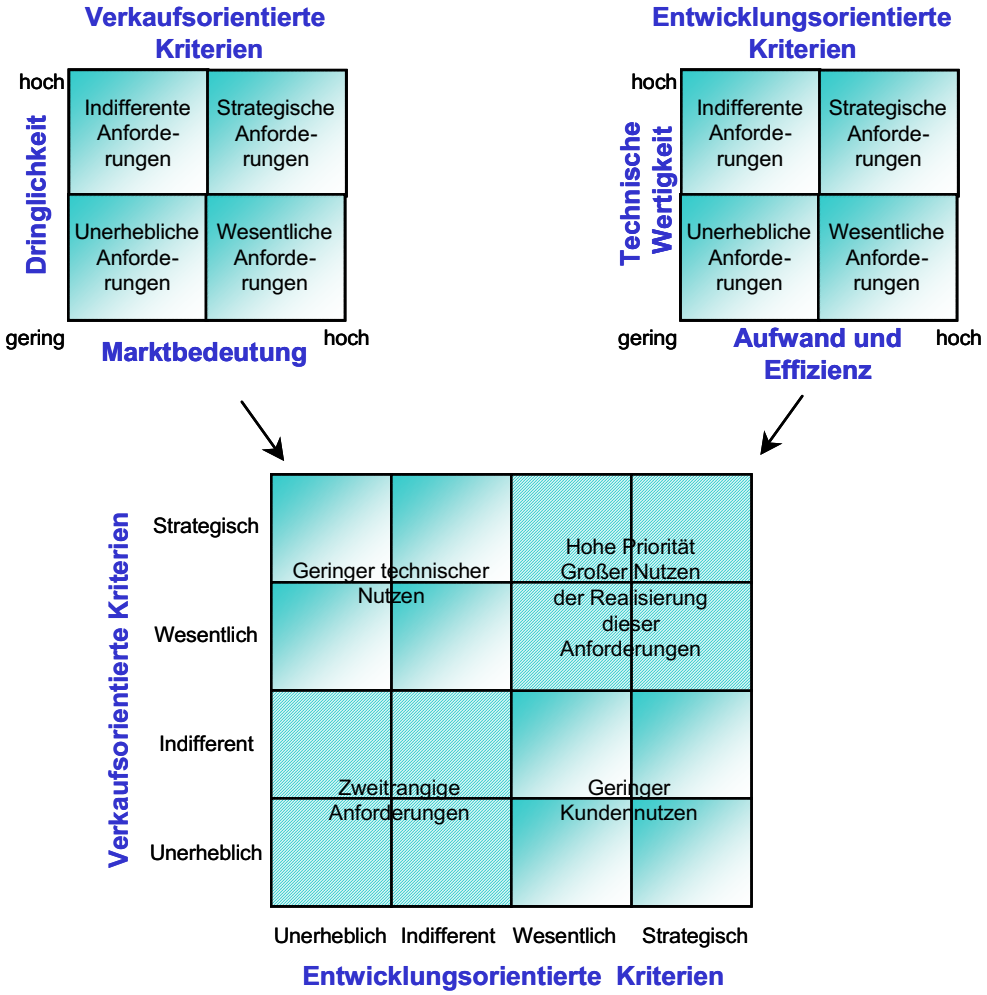


Abbildung 25 – Portfolio zur Verhandlung von Prioritäten

Teilportfolios

In den Teilportfolios werden zwei unterschiedliche Bewertungskriterien gegenüberg estellt. Eine Koordinatenachse des Portfolios entspricht jeweils einem Kriterium und wird entsprechend beschriftet und skaliert. Jede bewertete Anforderung entspricht damit e inem Punkt innerhalb des Portfolios. Zur Unterstützung der Verhandlung ist das Portfolio in vier Teilzellen aufgeteilt, die durch jeweils einen Schwellenwert von benachbarten Zellen getrennt sind. Jeder der Teilzellen kann somit je nach Bedarf unterschiedliche Wichtigkeit zugeordnet werden. Strategische Anforderungen haben eine hohe Bedeutung, da sie in beiden Kriterien hoch bewertet worden sind. Indifferente Anforderungen und wesentliche Anforderungen sind in einem Kriterium als hoch bewertet, im anderen als gering. Unerhebliche Anforderungen innerhalb eines Teilportfolios sind für beide Kriterien als gering bewertet.

Gesamtportfolio

Durch Kombination der beiden Teilportfolios zu einem Gesamtportfolio erhält man eine 16-zellige Matrix. Die Matrix kann in unterschiedliche Teilflächen eingeteilt werden. Für jede Teilfläche können im Voraus Vorgehensweisen festgelegt werden, wie mit entsprechend eingeordneten Anforderungen im Verhandlungsprozess umgegangen wird. In welcher Weise die Zellen der Teilportfolios zu einem Gesamtportfolio angeordnet werden, ist nicht starr vorgegeben, sondern bedarf einer strategischen Entscheidung.

Eine mögliche Einteilung zeigt die obige Abbildung 25. Von hohem Bewertungsergebnis sind Anforderungen, die strategisch oder wesentlich in beiden Teilportfolios sind. Diese versprechen hohen Profit im Falle ihrer Realisierung. Anforderungen die in beiden Portfolios indifferente oder unerhebliche Anforderungen sind, haben hingegen eine sehr geringe Bedeutung. Anforderungen mit geringem technischen Nutzen haben geringen Wert bezüglich einer technischen Verbesserung der Software. Umgekehrt haben Anforderungen mit geringem Kundennutzen wenig Wert bezüglich der Verbesserung des Verkaufserfolges. Diese grobe Klassifizierung des Bewertungsergebnisses von Anforderungen ist Basis der Verhandlung von Bewertungsergebnissen.

6.6.3 Verhandlung von Bewertungsergebnissen

Das Ziel der Verhandlung ist die detaillierte Ermittlung von Bewertungsergebnissen einzelner Anforderungen. Die Portfolio-Darstellung der Anforderungen ermöglicht eine einfache visuelle Nachvollziehbarkeit der vier Kriterienbewertungen verschiedener Anforderungen. Das Bewertungsergebnis wird durch einen qualitativen Vergleich zu anderen Anforderungen entschieden. Diskussionsbedarf besteht vor allem bei Anforderungen, die nahe an Schwellengrenzen zu benachbarten Zellen sind.

Je nach der konkreten Situation kann nicht nur über die Bewertungsergebnisse einer Anforderung verhandelt werden, sondern auch über deren inhaltliche Modifikation. Beispielsweise kann eine Anforderung durch Hinzunahme von Teilanforderungen in ihrem Marktwert gesteigert werden, um somit höhere Priorität zu erlangen. Umgekehrt kann die Realisierung einer sehr dringlichen Anforderung auf unterschiedliche Versionen aufgeteilt werden, um kritische Teile sofort umsetzen zu können (vgl. auch Abschnitt 4.4.4).

Aus den ermittelten Bewertungsergebnissen werden Prioritäten für Anforderungen abgeleitet. Diese dienen dann Grundlage für die Auswahl zu realisierender Anforderungen (vgl. Abschnitt 7.5.2) und für die Planung von Systemmodul-Versionen (vgl. Abschnitt 7.5.3).

6.7 Indirekte Skalierung anhand eines Kontextmodells

In den vom Autoren dieser Arbeit durchgeführten Studien (vgl. [Dei98a]) wurde festgestellt, dass innerhalb eines Unternehmens eine ganze Reihe von Erfahrungen vorhanden ist, die zur Ad-hoc-Beurteilung von Anforderungen verwendet werden können. Um nicht nach dem Zufallsprinzip eingesetzt werden zu müssen, ist hierfür eine systematische Erfassung erforderlich. Dadurch werden Beurteilungen später besser nachvollziehbar.

Abhilfe schafft hierfür ein sogenanntes Kontextmodell. Erfahrungen treffen immer nur auf einen bestimmten Kontext zu. Zur systematischen Nutzung von Erfahrungen werden daher diese im Kontextmodell nach ihrem Wirkungskontext klassifiziert. Damit ist es möglich bei erneutem Auftreten dieses Kontextes, zugehörige Erfahrungen schnell und mit geringem Aufwand zu identifizieren. Das Kontextmodell kann für beliebige Erfahrungen und beliebige Kontexte genutzt werden. Hauptfokus in dieser Arbeit liegt jedoch in der Verwendung von Erfahrungen zur Bewertung anhand der indirekten Skalierung, wie in Abschnitt 6.3.2 vorgestellt worden ist.

Das Kontextmodell baut sich aus zwei Teilen, dem Kontext und dem Erfahrungsmodell, auf. Mit dem Kontext wird eine konkrete Situation beschrieben, für die Erfahrungen herangezogen werden sollen. Das Erfahrungsmodell beschreibt eine Klassifikation unterschiedlichster Kontexte und ordnet diesen Erfahrungen systematisch zu. Um Erfahrungen nutzen zu können, muss der Kontext der Situation in der Klassifikation eingeordnet werden. Erfahrungen, die der selben Klassifikation zugehörig sind, können dann unmittelbar auf die Situation angewendet werden.

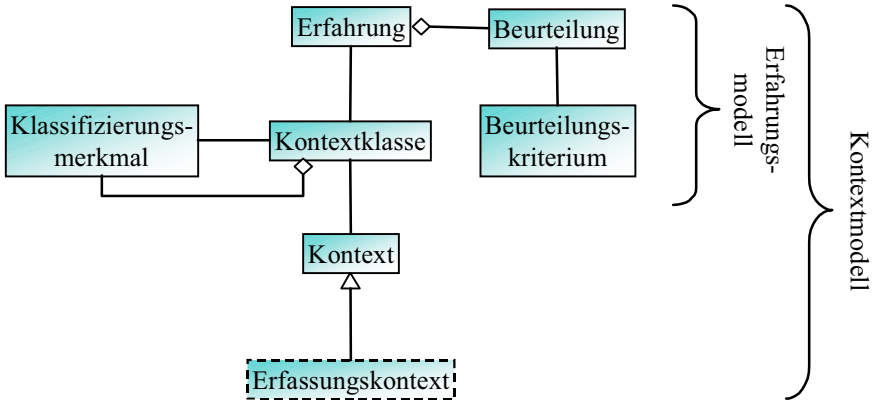


Abbildung 26 – Entwicklungsprodukte des Kontextmodells

Das Kontextmodell erweitert das Modell für Entwicklungsprodukte aus Kapitel 4. Abbildung 26 gibt einen Überblick über zugehörige Entwicklungsprodukte. Das Entwicklungsprodukt Erfassungskontext (vgl. Abschnitt 4.4.1) zur Dokumentation von Situationen, in denen Anforderungen erfasst werden, ist eine Verfeinerung eines Kontextes.

In den folgenden Unterabschnitten werden zunächst das Erfahrungsmodell und darauffolgend das Kontextmodell genauer beschrieben. Im Anschluss daran wird beispielhaft dargestellt, wie aus einem Erfassungskontext einer Anforderung heraus Erfahrungen gewonnen werden können. Schließlich wird im letzten Unterabschnitt ein Beispiel eines Erfahrungsmodells vorgestellt, das aus praktischen Erfahrungen abgeleitet wurde.

6.7.1 Erfahrungsmodell

Das Erfahrungsmodell besteht aus den fünf Entwicklungsprodukttypen Erfahrung, Beurteilungskriterium, Beurteilung, Kontextklasse und Klassifizierungsmerkmal. Zur Beschreibung der einzelnen Entwicklungsprodukte wird das Schema aus Abschnitt 3.2.3 verwendet. Die Entwicklungsprodukte werden anhand eines Beispiels für einen Erfassungskontext einer Anforderung illustriert.

Produkttyp Klassifizierungsmerkmal	
B	Beschreibt ein Merkmal, wonach ein Kontext klassifiziert werden kann.
Z	Erfahrungen treffen meist nur auf bestimmte Kontexte zu. Mit Hilfe der Klassifikation der Kontexte nach unterschiedlichen Klassifizierungsmerkmalen lassen sich die Erfahrungen spezifisch auf ihre Kontexte abbilden.
A	**Abgrenzungskriterium:** Beschreibt das Abgrenzungskriterium, wonach sich einzelne Kontextklassen des Klassifizierungsmerkmales unterscheiden.
G	-
B=Beschreibung, Z=Zweck, A=Attribute, G=Aggregationen	

Beispiel:

Ein mögliches Klassifizierungsmerkmal für den Erfassungskontext ist die Kundengröße. Für die Kundengröße kann als Abgrenzungskriterium die Umsatzzahl und der Innovationsgrad herangezogen werden.

Produkttyp Kontextklasse	
B	Kontextklassen klassifizieren Kontexte bezüglich eines bestimmten Klassifizierungsmerkmals in verschiedene Ausprägungen.
Z	Eindeutige Beschreibung einer Kontextklasse mit dem Zweck von klaren Abbildungen von Kontexten zu Erfahrungen
A	- **Klassenbezeichnung:** Eindeutige Bezeichnung für Kontextklasse - **Kürzel:** Eindeutige Abkürzung für Kontextklasse - **Abgrenzungskriterium:** Abgrenzung zu anderen Klassen eines Klassifizierungsmerkmals.
G	Kontextklassen können hierarchisch durch andere Klassifizierungsmerkmale verfeinert sein.
B=Beschreibung, Z=Zweck, A=Attribute, G=Aggregationen	

Beispiel:

Für das Klassifizierungsmerkmal Kundengröße kann zwischen Großkunden, innovativen Kleinkunden und Kleinkunden unterschieden werden. Folgende Tabelle enthält drei Kontextklassen für dieses Klassifizierungsmerkmal.

Klassifizierungsmerkmal Kundengröße		
Klassenbezeichnung	Kürzel	Abgrenzungskriterium
Großkunde	K-G	> 1 Mrd. DM Umsatz
Innovativer Kleinkunde	K-IK	<= 1 Mrd. DM Umsatz, innovative Ideen
Kleinkunde	K-K	<= 1 Mrd. DM Umsatz, keine innovativen Ideen

Die Wahl von Kontextklassen und zugehörigen Abgrenzungskriterien hängt immer von den gesammelten Erfahrungen ab. Mit Hilfe von Kontexten kann die Zuordnung aktueller Situationen zu jeweils bestimmten Kontextklassen ausgewählter Klassifizierungsmerkmale beschrieben werden. Eine Erfahrung wird einer Kontextklasse zugeordnet.

Produkttyp Erfahrung	
B	Eine *Erfahrung* beschreibt eine bezüglich einer bestimmten Kontextklasse früher gemachte Erfahrung. Diese kann anhand einer informellen Beschreibung oder mit Hilfe unterschiedlicher Beurteilungskriterien skizziert werden. Pro Kontextklasse können unterschiedliche Erfahrungen festgehalten werden. Erfahrungen können sowohl auf frühere Anforderungen als auch auf beliebige andere entwicklungsbezogene Kontexte beziehen.
Z	Systematische Dokumentation einzelner Erfahrungen, die bei Bedarf für Situationen, die zu einem bestimmten Kontext passen, angewendet werden können.
A	- **Kürzel:** Eindeutige Abkürzung für Erfahrung - **Beschreibung:** Gegebenenfalls informelle Beschreibung einer Erfahrung.
G	**Beurteilung**: Bezüglich dieser Erfahrung gefällte Beurteilung nach genau einem Beurteilungskriterium.
B=Beschreibung, Z=Zweck, A=Attribute, G=Aggregationen	

Werden Erfahrungen schematisch anhand von Beurteilungskriterien beschrieben, so können sie direkt zur indirekten Skalierung bei der Bewertung angewendet werden. Bestenfalls finden die Bewertungskriterien aus Abschnitt 6.5 zur Beschreibung von Erfahrungen Anwendung.

Produkttyp Beurteilungskriterium	
B	Beschreibung eines Kriteriums zur Beurteilung einer Erfahrung.
Z	Damit können gewonnene Erfahrungen schematisch aufgeschrieben werden können.
A	- **Bezeichnung**: Eindeutige Bezeichnung für ein Beurteilungskriterium. - **Mögliche Ausprägungen**: Aufzählung der verschiedenen Ausprägungen einer Beurteilung nach dem genannten Beurteilungskriterium.
G	-
B=Beschreibung, Z=Zweck, A=Attribute, G=Aggregationen	

Beispiel:

Sämtliche in Abschnitt 6.5 vorgestellten Bewertungskriterien stellen mögliche Beurteilungskriterien für Erfahrungen dar.

Produkttyp Beurteilung	
B	Konkrete Ausprägung eines einzelnen Beurteilungskriteriums
Z	Strukturierte Aufschreibung von genau einem Beurteilungskriterium
A	**Wert**: Ausprägung eines einzelnen Beurteilungskriteriums. Das Beurteilungskriterium ist zur Beurteilung assoziiert.
G	-
B=Beschreibung, Z=Zweck, A=Attribute, G=Aggregationen	

Beispiel:

Mit Großkunden wurde die Erfahrung gemacht, dass deren Anforderungen meist ziemlich populär auf dem Markt sind und einen hohen Kundennutzen liefern. Dies lässt sich schematisch folgendermaßen darstellen:

Erfahrungen zur Kundengröße: Großkunden (K-G)			
Kürzel	Beurteilungskriterium	Ausprägung	Bemerkung
E 1	Popularität/Werbeeffekt	Hoch	(im Vergleich zu Kleinkunden)
E 1	KANO-Bewertung (siehe Glossar)	Hoch	

Die Erfassung von Anforderungen erfordert zunächst einen gewissen Aufwand. Für eine einzelne Erfahrung ist dieser in etwa vergleichbar mit dem Aufwand der direkten Skalenbewertung mit mehreren Bewertungskriterien für eine Anforderung. Einmal so erfasste Erfahrungen können beliebig oft verwendet werden und erhöhen nach der zweiten Prämisse aus Abschnitt 6.3 die Zuverlässigkeit von Bewertungen. Daher überwiegt der zu erwartende Nutzen dem Aufwand der systematischen Erfassung der Anforderungen.

6.7.2 Kontextmodell

Dieser Abschnitt stellt das Kontextmodell für die indirekte Skalierung vor. Hierzu beschreibt folgende Schema das Entwicklungsprodukt Kontext.

Produkttyp Kontext	
B	Beschreibung einer Situation im Zusammenhang mit der Entwicklung komplexer Standardsoftware anhand unterschiedlicher Attribute. Dieses Entwicklungsprodukt wird nicht unmittelbar verwendet, sondern muss verfeinert werden. Ein Beispiel stellt hierfür der Erfassungskontext dar.
Z	Anhand der Attributbelegungen sollen jeder Situation systematisch zugehörige Erfahrungen zugeordnet werden können.

A	Für unterschiedliche Verfeinerungen des Entwicklungsproduktes *Kontext* sind hier unterschiedliche Attribute zur Informellen Beschreibung wie auch zur Klassifizierung eines Kontextes enthalten.
G	-
B=Beschreibung, Z=Zweck, A=Attribute, G=Aggregationen	

Der Erfassungskontext einer Anforderung für die Entwicklung von Finanzsoftware mit Bankkontenverwaltung könnte folgendermaßen aussehen:

Beispiel:

Herr Schmidt hat für das Beispiel FinSoft *aus Abschnitt 6.1 Informationen zur Erfassung der Anforderung in folgendem vorgefertigten Formular notiert:*

Produkt	Finanzsoftware mit Bankkontenverwaltung.
Anforderung	Bezahlungen im Internet sollen über eine Payphone-Funktionalität abgewickelt werden können
Quellenbeschreibung	Maier Josef, Firma X
Quellentyp	Kunde
Kanalbeschreibung	Schmidt Joachim, Niederlassung München
Kanaltyp	Verkaufsgespräch
Erfassungstyp	Informelles Gespräch
Erfassungsdatum	01.09.2000

Im obigen Formular dokumentierte Informationen stellen nur einen Teil des gesamten Erfassungskontextes dar. Darüber hinaus gehören beispielsweise noch genauere Informationen zur Firma X hinzu und welche Rolle Joachim Schmidt einnimmt usw. In obigem Formular sind nur Informationen notiert, die spezifisch für die Erfassung einzelner Anforderungen sind.

Andere Informationen, wie zum Beispiel Firmendaten, wurden von dem Erfassungsformular getrennt. Damit wird der Aufwand der konkreten Anforderungserfassung reduziert. Fehlende Attribute dieses Erfassungskontextes können mittels in diesem Formular notierter Referenzen gefunden werden. Für die Firma X könnten beispielsweise folgende Informationen zur Beschreibung des Erfassungskontextes von Bedeutung sein:

Beispiel:

Firma X aus Beispiel FinSoft *ist eine deutsche Firma mit Sitz in München, ist spezialisiert auf Fahrzeugbau und operiert vorwiegend im europäischen Raum. Im Jahr 1999 erzielte sie einen Umsatz von 20 Mrd. DM.*

Je umfangreicher in einem Erfassungskontext enthaltene Informationen sind, desto genauer können vorhandene Erfahrungen darauf abgebildet werden.

6.7.3 Abbildung von Erfahrungen auf eine Anforderung

Die im Erfassungskontext enthaltenen Informationen müssen eine Einordnung in einzelne Kontextklassen erlauben, um auf ihn zutreffende Erfahrungen abbilden zu können. Der Erfassungskontext des obigen Beispiels enthält folgende Zusatzinformationen:

Beispiel:

Die Firma von Josef Maier hatte 1998 einen Umsatz von 20 Mrd. DM erzielt und ist damit nach dem Klassifizierungsmerkmal Kundengröße ein Großkunde. Daraus kann folgende Erfahrung abgeleitet werden, die folgendermaßen tabellarisch skizziert ist:

Erfahrungen aus dem Erfassungskontext einer Anforderung				
Softwareprodukt		Finanzsoftware mit Bankkontenverwaltung.		
Anforderung		Bezahlungen im Internet sollen über eine Payphone-Funktionalität abgewickelt werden können		
Kontext-Kürzel	Erfahrungs-Kürzel	Popularität/Werbeeffekt	KANO-Bewertung	...
K-G	E 1	Hoch	Hoch	...
...				

Diese Tabelle lässt sich bei entsprechender Werkzeugunterstützung automatisch generieren, sobald Erfahrungen systematisch erfasst sind. Mit der Ermittlung von Erfahrungen für die Bewertung einzelner Anforderungen ist also kein Aufwand verbunden. Zur Ableitung von Bewertungen von Anforderungen müssen die Bewertungskriterien mit den generierten Erfahrungen abgeglichen werden. Dies bedeutet einen geringfügig, jedoch nicht quantifizierbaren Mehraufwand verglichen zur direkten Skalenbewertung. Dem steht jedoch der Nutzen einer erhöhten Zuverlässigkeit gewonnener Bewertungen gegenüber.

6.7.4 Beispiel eines Erfahrungsmodells

In diesem Abschnitt wird das Erfahrungsmodell anhand eines praktischen Beispiels illustriert. Im Rahmen der Studien [Dei98a] wurde eine Reihe von Erfahrungen identifiziert, die speziell für den Bereich der Entwicklung von Standardsoftware in der Automatisierungstechnik gelten.

Die Genauigkeit der indirekten Skalenbewertung hängt im wesentlichen von der Systematik der Erfassung von Erfahrungen ab. Da im obigen Beispiel dies relativ unsystematisch erfolgte, können daraus lediglich sehr grobe Beurteilungen für Anforderungen gewonnen werden. Die folgende Tabelle zeigt beispielhaft eine pauschale Beurteilung der Bewertungskriterien aus Abschnitt 6.5 anhand von Erfahrungen für unterschiedliche Erfassungskontexte.

Die Zeilen sind Beurteilungskriterien (hier Bewertungskriterien aus Abschnitt 6.5), die durch die Erfahrungen beurteilt werden können. Spalten sind Klassen eines Erfassung s-kontextes (vgl. auch Abschnitt 4.4.1) Die ausgefüllten Felder haben folgende B edeutung:

- x: keine Aussage aufgrund gewonnener Erfa hrungen möglich
- +: Kriterienbewertung ist tendenziell hoch
- o: Kriterienbewertung ist tendenziell mittelhoch
- -: Kriterienbewertung ist tendenziell niedrig

		Quellen								Kanäle				
		Firmens.		Klassen Kund. u. Anw.						Firmens.		Klassen		
		Intern	Extern	Groß	Klein	Innov.	Innovationsman.	Basissystem	Konkurrenzprod.	Intern	Extern	Aktive Kontakte	Passive Kontakte	Zusammenarbeit
1	**Dringlichkeit**													
1.1.	Marktdurchdringung	x	x	o	+	-	-	o	+	x	x	o	-	x
1.2.	Lebenszyklusrestdauer	x	x	o	+	-	-	o	+	x	x	o	-	x
1.3.	Popularität/Werbeeffekt	x	x	+	o	o	o	-	-	x	x	x	x	x
2	**Marktbedeutung**													
2.3.	KANO-Bewertung	+	-	+	o	+	+	o	o	x	x	+	o	+
2.4.	Kundenanteil	+	-	+	o	+	o	+	+	+	o	+	o	x
2.5.	Substituierbarkeit	x	x	x	x	x	x	+	+	x	x	x	x	x
2.6.	Marktvolumen	+	-	+	o	o	o	+	+	+	o	+	o	x
3	**Technische Wertigkeit**													
3.1.1.	Flexib. Weiterentwicklung	x	x	x	x	x	+	x	x	x	x	x	x	x
3.2.1.	Fremden Abhängigkeit	x	x	x	x	x	x	-	x	x	x	x	x	x

Die Kontextklassen der Tabelle entsprechen unterschiedlichen Quellen - und Kanaltypen eines Erfassungskontextes (vgl. Abschnitt 4.4.1), wobei das Klassifizierungsmerkmal Kunden und Anwender weiter verfeinert worden ist. Überl egungen über die Entstehung dieser Beurteilungen können im Anhang F nachgel esen werden.

Weitere Erfahrungen aus Erfassungskontexten können beispielsweise aufgrund des E r-fassungszeitpunktes und der Nennungshäufigkeit von Anforderungen abgeleitet we rden. Liegt beispielsweise der Erfassungszeitpunkt lange zurück, so verschiebt sich der L e-benszykluszustand der Anforderung nach hinten. Von einer hohen Nennungshäufigkeit kann ein hoher Kundenanteil abgele itet werden.

Über den Erfassungskontext hinaus können Erfahrungen auch anderen Klass ifizierungen zugeordnet werden. So wurde vom Autoren dieser Arbeit in [Dei99a] eine Klassifikation von Anforderungen nach drei Aspekten (vgl. Abschnitt 4.2.2) vorgenommen. Dies sind die Anwendungsdomäne, der Nutzungskontext und Basissyst eme.

Die Anwendungsdomäne beschreibt die Einsatzumgebung einer Software, in der sie spezifische Aufgaben erfüllen soll. Der Nutzungskontext konzentriert sich auf die unmittelbaren Interaktionen zwischen der Software und ihrer Umgebung. Schließlich nutzt komplexe Standardsoftware häufig Dienste von Basissystemen (vgl. Glossar). Anforderungen können diesen drei Aspekten zugeordnet und so aufgrund von Vergangenheitserfahrungen pauschal bewertet werden. In der Anwendungsdomäne sind Kernanforderungen an das Softwaresystem gestellt. Daher sind zugehörige Anforderungen vom höchsten Marktwert, während Anforderungen in anderen Aspekten geringer bewertet werden.

Die Einordnung von Anforderungen in diese drei Aspekte kann auch zur Beurteilung von deren Änderungswahrscheinlichkeit herangezogen werden. Während die Änderungswahrscheinlichkeit in der Anwendungsdomäne von der spezifischen Art der Domäne abhängt, ist sie beim Nutzungskontext von der Anwendungsdomäne und technischen Innovationen beeinflusst. Basissysteme sind wie die zu entwickelnde Software selbst Standardsoftware und werden daher meist in regulären Zyklen weiterentwickelt. Daher ändern sich Anforderungen innerhalb dieses Aspektes künftig mit relativ hoher Wahrscheinlichkeit. Mit Hilfe dieser Abschätzung kann eine Architektur derart ausgelegt werden, dass sie insbesondere an jenen Stellen änderungsfreundlich ist, wo Anforderungen mit hoher Änderungswahrscheinlichkeit realisiert werden.

Kapitel 7

Prozessmodell

Dieses Kapitel beschreibt das Prozessmodell für das Requirements Engineering komplexer Standardsoftware. Das Prozessmodell ist nach dem Modell der Entwicklungsprodukte aus Kapitel 4 der zweite Teil des Kerns der Arbeit. Kapitel 5 und Kapitel 6 stellten methodische Techniken vor. Das vorliegende Kapitel präsentiert einerseits das Prozessmodell selbst. Es zeigt andererseits aber auch, wie die Inhalte der vorhergehenden Kapitel integriert werden. Darüber hinaus demonstriert es die Tragfähigkeit der gesamten Arbeit anhand eines ausführlichen Fallbeispiels.

Abschnitt 7.1 stellt zunächst das Gesamtkonzept des Prozessmodells vor. Die sich anschließenden Abschnitte beschreiben den Requirements Engineering Prozess anhand der in Abschnitt 3.1 vorgestellten Modellierungskonzepte für Entwicklungsprozesse. In Abschnitt 7.2 werden zunächst am Requirements Engineering beteiligte Rollen vorgestellt. Abschnitt 7.3 führt das Illustrationsbeispiel für die sich anschließenden Abschnitte ein. In Abschnitt 7.4 werden Charakteristika und das Vorgehens des Herausarbeitens von Anforderungen erläutert. Schließlich wird in Abschnitt 7.5 der schrittweise Übergang von lose gesammelten Anforderungen hin zu einer konkreten Planung verschiedener Versionen eines Produktsystems gezeigt. Die Grundideen des vorliegenden Kapitels wurden in [Dei99c] publiziert. Diese wurden im Rahmen der vorliegenden Arbeit weiter verfeinert, systematisiert und mit den Konzepten der drei vorhergehenden Kapitel integriert.

7.1 Gesamtkonzept des Prozessmodells

Dieser Abschnitt stellt das Gesamtkonzept des Prozessmodells des Requirements Engineerings komplexer Standardsoftware vor. Hierzu wird zunächst in Abschnitt 7.1.1 ein Modell eines Entwicklungsprozesses komplexer Standardsoftware skizziert, in der sich die Aktivitäten des Requirements Engineerings einbetten. Anschließend werden in Abschnitt 7.1.2 Spezifika der Entwicklung komplexer Standardsoftware vorgestellt, ehe in Abschnitt 7.1.3 ein Überblick über das Vorgehen des Requirements Engineerings gegeben wird.

7.1.1 Übergreifender Entwicklungsprozess

Dieser Abschnitt präsentiert ein Modell eines übergreifenden Entwicklungsprozesses, in den sich der Requirements Engineering Prozess dieser Arbeit einbettet. Die Studie [CaB95] analysiert bestehende Prozessmodelle der Software Entwicklung und der Entwicklung materieller Produkte und vergleicht sie hinsichtlich der Problemstellungen der Entwicklung von Standardsoftware. Sie betrachtet Software-Prozessmodelle wie serielle

(zum Beispiel Wasserfall) und inkrementelle (zum Beispiel Wegwerf- und Evolutionäre Prozessmodelle [BeD91,Gra89]), den Cleanroom Ansatz [Mil87], wie auch post-inkrementelle Ansätze (zum Beispiel das Spiralmodell von Boehm [Boe88]).

All diese Modelle beschäftigen sich mit der Entwicklung von Individualsoftware und berücksichtigen daher den Marktaspekt von Standardsoftware nicht in ausreichendem Maße. Wie eigene Untersuchungen im Rahmen von [ABD+99] ergaben, ist auch das V-Modell sehr stark auf die Entwicklung von Individualsoftware ausgerichtet. Hauptsächliche Mankos sind hier die reine Ausrichtung auf isolierte abgeschlossene Projekte und ein vollkommen fehlender Marktbezug. Die Entwicklung komplexer Standardsoftware bedarf hingegen einer umfassenden Unterstützung der Entwicklung von Versionen und Varianten eines Produktsystems, wie dem Kapitel 4 zu entnehmen ist.

Die meisten Prozessmodelle für die Entwicklung materieller Produkte sind ähnlich strukturiert wie Software-Prozessmodelle. Erstere haben jedoch einen besseren Bezug zu Aktivitäten des Marketings, wie beispielsweise der Suche nach Anforderungen auf dem gesamten Markt. Die Modelle für materielle Produkte sind jedoch zu unspezifisch für Problemstellungen der Softwareentwicklung. Aus dieser Analyse leiten [CaB95] ein Prozessmodell zur Entwicklung von Standardsoftware ab. Dieses dient als Grundlage für das Modell des übergreifenden Entwicklungsprozesses in der vorliegenden Arbeit.

Abbildung 27 skizziert den Ablauf des Entwicklungsprozesses. Das Requirements Engineering umfasst sowohl kontinuierliche Arbeiten als auch zyklische Arbeiten. Zu Aktivitäten des Requirements Engineerings wird in Abschnitt 7.1.3 ein Überblick gegeben.

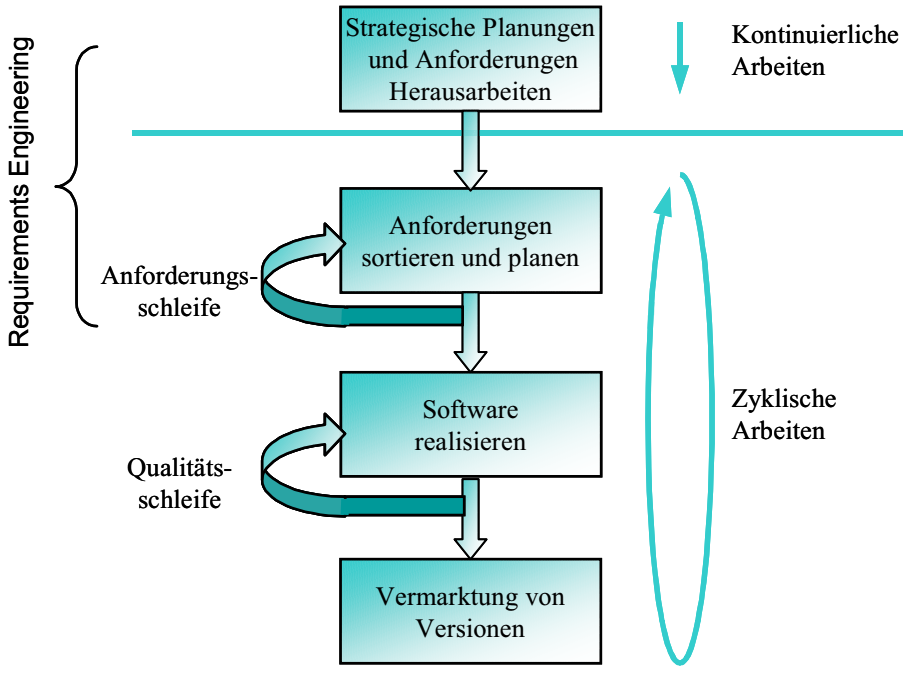


Abbildung 27 – Schematischer Entwicklungsprozess

Kontinuierliche Arbeiten: Diese werden unabhängig von Versionszyklen der Standardsoftware durchgeführt. Dazu zählen sowohl die Erarbeitung strategischer Entwicklungsprodukte aus Abschnitt 4.3 als auch das Herausarbeiten von Anforderungen (vgl. das Entwicklungsprodukt *Erfassungskontext* aus Abschnitt 4.4.1).

Zyklische Arbeiten: Die zyklischen Arbeiten teilen sich in die Anforderungsschleife, die Qualitätsschleife und schließlich die Auslieferung von Versionen auf. Die beiden Schleifen symbolisieren eine inkrementelle Vorgehensweise bei der Weiterverarbeitung von Anforderungen und bei der Realisierung der Software.

Anforderungsschleife: In der Anforderungsschleife sind die Requirements Engineering Phasen (vgl. [Poh96]) Verhandlung, Spezifikation und Validierung angesiedelt.

Qualitätsschleife: Basierend auf der Versionsplanung startet die Entwicklung. In der Qualitätsschleife werden Anforderungen in der Software realisiert und in jedem Schleifendurchlauf die Qualität ihrer Realisierung überprüft und verbessert. Die Qualitätskontrolle wird durch iterative Alpha- und Beta-Tests, in der Praxis häufig auch außerhalb der Entwicklungsabteilung, durchgeführt. Damit ist die Requirements Engineering Phase Verifikation in diese Schleife miteinbezogen. Sobald die Software einen vordefinierten Grad an Fehlerfreiheit erreicht hat, wird die Version ausgeliefert, das heißt produziert, verpackt und verkauft. Die Qualitätsschleife wird in dieser Arbeit nicht genauer beleuchtet.

7.1.2 Spezifika der Entwicklung komplexer Standardsoftware

Dieser Abschnitt beschreibt Spezifika, die bei der Entwicklung einer komplexen Standardsoftware auftreten. In den Abschnitten 3.1.1 und 3.2.3 wurde erläutert, dass in dem Modellierungskonzept keine explizite Ausführungsreihenfolge von Aktivitäten vorgegeben ist. Die Ausführbarkeit von Aktivitäten ist ausschließlich durch vorgegebene Modellzustände bestimmt. Dadurch können Entwicklungsprozesse komplexer Standardsoftware flexibler gestaltet werden. Tragende Eckpfeiler hierfür sind das *Concurrent Engineering* (vgl. [Dei98a, Aoy93]) und die Überlappung von Entwicklungsphasen unterschiedlicher Versionen. Beides hat sich in der Praxis bereits bewährt.

Beim Concurrent Engineering werden einzelne Systemmodule eines Produktsystems weitestgehend unabhängig voneinander entwickelt. Damit ist auch die Entwicklung unterschiedlicher Systemmodule nicht zeitlich starr aneinander gekoppelt. So kann die Entwicklung einer Version verschiedener Systemmodule zu unterschiedlichen Zeitpunkten starten und enden (vgl. Attribut Zykluszeit im Entwicklungsprodukt Systemmodul-Version aus Abschnitt 4.4.4). Unter der Überlappung von Entwicklungsphasen wird verstanden, dass parallel zu den späten Phasen einer Version die frühen Phasen der Entwicklung einer nachfolgenden Version beginnen können. So kann sich beispielsweise eine Version eines Systemmoduls noch in der Realisierung befinden, während für die nachfolgende Version bereits das sortieren und planen der Anforderungen begonnen hat. Dadurch wird für Anforderungen die Gesamtdurchlaufzeit durch die Entwicklung vom Herausarbeiten bis hin zu ihrer Realisierung verkürzt.

7.1.3 Überblick über das Vorgehen des Requirements Engineerings

In diesem Abschnitt wird ein Überblick über das Vorgehen des Requirements Engineering komplexer Standardsoftware gegeben. Wie in Abschnitt 7.1.1 bereits angedeutet, umfasst das Vorgehen des Requirements Engineerings komplexer Standardsoftware sowohl kontinuierliche als auch zyklische Arbeiten. Abbildung 28 gibt einen Überblick über die Phasen und sämtliche zugehörige Aktivitäten. Die folgenden Absätze skizzieren die Aufgaben der einzelnen Phasen. Die einzelnen Rollen und Aktivitäten werden in den Abschnitten 7.2, 7.4 und 7.5 detailliert erörtert.

Kontinuierliches Herausarbeiten von Anforderungen: Anforderungen für Standardsoftware stammen aus einer Vielzahl von Quellen, für deren Erfassung es einer systematischen Vorgehensweise bedarf. Die Vorgehensweise hängt vor allem von der Art der Quelle und von der Art des Kanals, über dessen Weg eine Anforderung von der Quelle her erfasst wird ab. Daraus resultieren 11 Aktivitäten für das Herausarbeiten von Anforderungen, die sich in den beteiligten Rollen, den anwendbaren Anleitungen und im Vorgehen unterscheiden. Sie haben allesamt den gleichen Zweck und die gleichen Ergebnisprodukttypen. Das Kontinuierliche Herausarbeiten wird in Abschnitt 7.4 ausführlich erläutert.

Zyklische Phasen: Die eigentliche Entwicklung einer Standardsoftware beginnt mit den zyklischen Phasen. Diese dienen, wie eingangs erwähnt, einer stufenweisen Planung von zu realisierenden Anforderungen. Die zyklischen Phasen werden pro Version des Produktsystems komplett durchlaufen. Während des Sortierens und Planens von Anforderungen werden das Produktsystem und herausgearbeitete Anforderungen in mehreren Schritten analysiert, konkretisiert, bewertet und strukturiert. Letztendliches Ergebnis ist eine stufenweise Versionsplanung des Produktsystems. Diese besteht aus Systemmodulen, Versionen und Zeiträumen, in denen bestimmte Anforderungen realisiert werden sollen. Systemmodulen sind zusätzlich zur Verfügung stehende Ressourcen zugeordnet und für Anforderungen Prioritäten festgelegt. Die drei logisch aufeinanderfolgenden Phasen verfolgen unterschiedliche Ziele:

Definition der Marktsicht: Die Marktsicht dokumentiert isoliert die Kundenorientierung. So festgehaltene Anforderungen zielen auf einen optimalen Verkaufserfolg des Produktsystems für den Gesamtmarkt ab. Vorrangig werden Entwicklungsprodukte der Marktsicht aus Abschnitt 4.4.2 erarbeitet. Aktivitäten umfassen ein Ausfiltern irrelevanter Anforderungen, ein Strukturieren der Anforderungen nach Aspekten und Variationstypen und deren verkaufsorientierte Bewertung.

Definition der Systemsicht: In der zweiten Phase werden diese Anforderungen auf ein tatsächlich realisierbares Softwaresystem übertragen. Hier wird von der aktuell zur Verfügung stehenden Entwicklungszeit abstrahiert, um möglichst weitgehend zukünftige Visionen bereits in die Planung des Produktsystems zu integrieren. Vorrangig werden in dieser Phase Entwicklungsprodukte der Systemsicht aus Abschnitt 4.4.3 erarbeitet. Die Aktivitäten umfassen die Definition der Systemarchitektur, die Aufteilung der Ressourcen, die Zuordnung, die entwicklungsorientierte Bewertung und Auswahl zu realisierender Anforderungen.

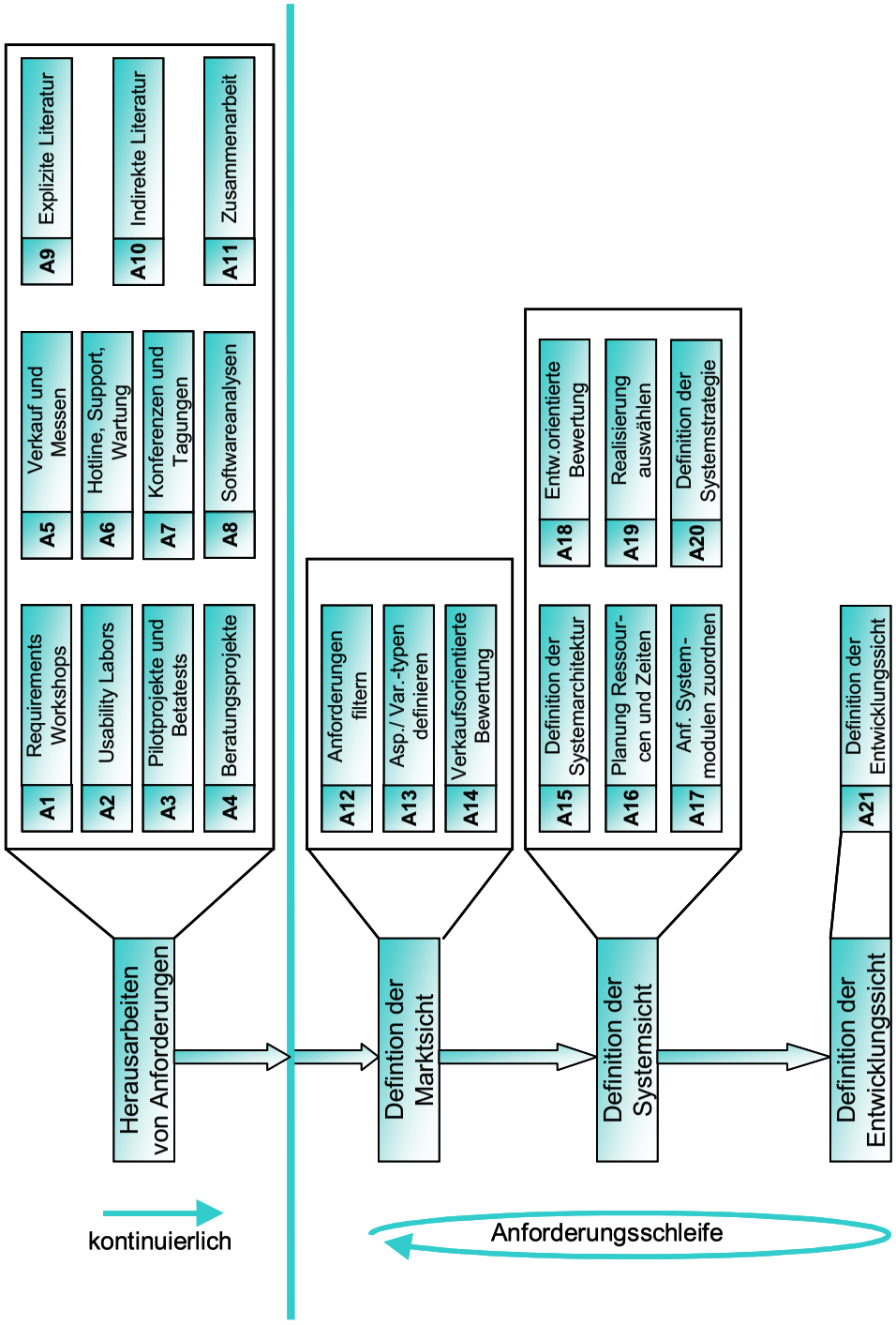


Abbildung 28 – Überblick über Aktivitäten des Prozessmodells

Definition der Entwicklungssicht: Schließlich spielt die zur Verfügung stehende Entwicklungszeit in der Realisierung von Anforderungen eine wesentliche Rolle. Dazu wird in der dritten Phase eine detaillierte Versionsplanung der Realisierung von Anforderungen aufgestellt. Durch die Versionsplanung werden die Anforderungen auf Versionen verteilt, in denen sie realisiert werden sollen. Für die Durchführung der Versionsplanung wird eine Frist gesetzt, um Verzögerungen der Fertigstellungszeitpunkte aufgrund fortwährend neuer Anforderungen zu vermeiden. Ergebnis dieser Phase ist die in Abschnitt 4.4.4 erläuterte Entwicklungssicht, die Basis für die Entwicklung der Software ist. Die Definition der Entwicklungssicht erfolgt in isolierten Aktivitäten für jedes Systemmodul.

Aktivitäten der zyklischen Phasen werden in Abschnitt 7.5 ausführlich erläutert. Das Vorgehen des Requirements Engineerings unterscheidet sich dabei abhängig davon, ob es sich um die erste Version oder um eine Weiterentwicklung handelt:

Initiierung der Entwicklung: Die Entwicklung eines neuen Produktsystems wird in der Regel durch Ideen aus unterschiedlichen Quellen initiiert. Häufig wird in Vorstudien des Innovationsmanagements (vgl. Abschnitt 7.4.1) der mögliche Zielmarkt bestimmt und die Rentabilität untersucht. Zum Herausarbeiten von Anforderungen stehen vor Auslieferung der ersten Version des Produktsystems weniger Quellen zur Verfügung. Abhängig vom Innovationsgrad der Software findet in der Praxis vor der ersten Version ein mehr oder weniger reguläres Herausarbeiten von Anforderungen statt. Je nach Dringlichkeit können auch strategische Vorgaben zu Beginn der Entwicklung eines Produktsystems missachtet werden. Die ersten Aktivitäten laufen eher unstrukturiert ab, um die Kreativität einzelner Beteiligter am Entwicklungsprozess zu fördern. Nach der Anlaufphase muss dann möglichst nahtlos auf den geregelten Entwicklungsprozess übergegangen werden. Dadurch können die eingangs ausführlich erläuterten Ziele des Prozessmodells verfolgt werden.

7.2 Rollen

In diesem Abschnitt werden die am Requirements Engineering komplexer Standardsoftware beteiligten Rollen (vgl. Abschnitt 3.1.1) vorgestellt. Abbildung 29 gibt einen Überblick über alle Rollen. Es wird zwischen verkaufsorientierten und entwicklungsorientierten Rollen unterschieden. Beide Gruppen sind noch feiner untergliedert. So lassen sich sowohl verkaufsorientierte als auch entwicklungsorientierte Rollen in Rollen mit kurzfristigem und langfristigem Fokus unterscheiden. Verkaufsorientierte Rollen können hinsichtlich ihres Horizonts verglichen zum Markt untergliedert werden. Dies sind einzelne Kunden, Marktsegmente (vgl. Abschnitt 4.3) oder der Gesamtmarkt der Software. Eine Unterscheidung entwicklungsorientierter Rollen kann zusätzlich hinsichtlich deren Fokus auf Einheiten der zu entwickelnden Software gemacht werden. Hier werden Produktsystem und Systemmodul unterschieden.

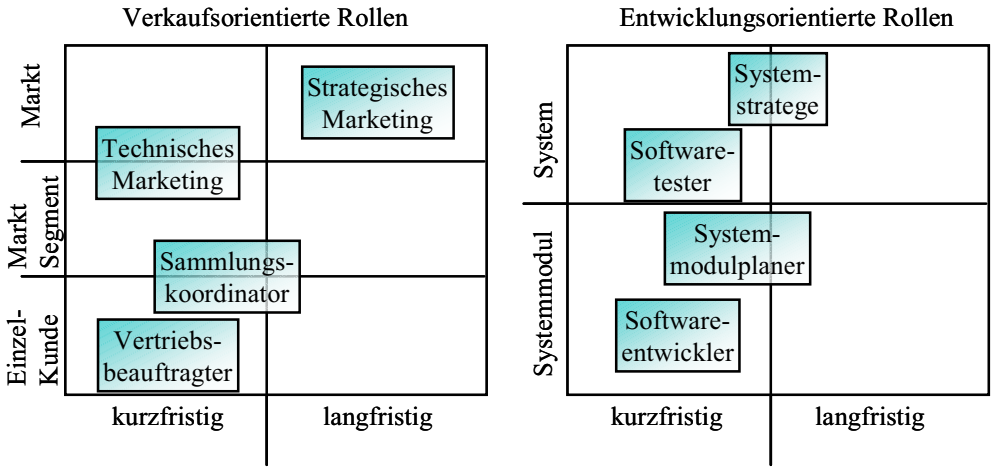


Abbildung 29 – Rollen im Requirements Engineering komplexer Standardsoftware

7.2.1 Verkaufsorientierte Rollen

Verkaufsorientierte Rollen zielen auf die Maximierung des Umsatzes ab. Die kann vor allem dadurch erreicht werden, dass ein Softwareprodukt angeboten wird, das für viele Kunden attraktiv ist. Deshalb forcieren verkaufsorientierte Rollen die Realisierung von möglichst vielen Anforderungen. Darüber hinaus sind verkaufsorientierte Rollen auch für den langfristigen Erfolg eines Softwareproduktes zuständig.

Vertriebsbeauftragter

Vertriebsbeauftragte sind für die Auswahl und den Verkauf von Produkten verantwortlich, die jeweils den Bedürfnissen seiner Kunden am meisten entsprechen. Bei Bedarf unterstützen sie den Kunden bei deren individuellen Anpassungen und Erweiterungen. Vertriebsbeauftragte dienen als vermittelnde Schnittstellen zwischen einzelnen Kunden und der Entwicklung der Software. Ihr Hauptaugenmerk ist dabei immer die Maximierung der Umsätze und die Bildung von engen Beziehungen zum Kunden.

In Abschnitt 7.4 wird erläutert, dass der Vertriebsbeauftragte am Herausarbeiten von Anforderungen beteiligt ist. Einzelne Kunden haben häufig eigene Anforderungen an das Produkt. Vertriebsbeauftragte sind bestrebt, ihre Kunden langfristig an sich zu binden. Daher bemühen sie sich um die Durchsetzung dieser Anforderungen in der Entwicklung und üben deshalb in der Regel einen hohen Druck auf die Entwicklung aus.

Sammlungskoordinator

Der Sammlungskoordinator ist für das kontinuierliche Herausarbeiten roher Anforderungen für eine komplexe Standardsoftware zuständig. Dazu koordiniert er alle Aktivitäten, die für eine systematische Erfassung von Anforderungen notwendig sind.

Am Herausarbeiten von Anforderungen arbeiten verschiedenartigste Personen in unterschiedlichsten Situationen mit (siehe Abschnitt 7.4). Daher organisiert der Sammlungskoordinator für an Aktivitäten des Herausarbeitens beteiligte Rollen unterschiedliche Trainings für hierfür adäquate Methoden.

Technisches Marketing

Das Technische Marketing ist für die letztendliche Definition der Marktsic ht (vgl. Abschnitt 4.4.2) zuständig. Um die große Breite von Kunden in den Griff zu bekommen, werden Kunden mit ähnlichen Anforderungen in Mark tsegmente (vgl. Abschnitt 4.3) gruppiert.

Strategisches Marketing

Das Strategische Marketing ist für die langfristige Behauptung einer Standardsoftware auf dem Markt verantwortlich. Hierzu ist es vor allem für das Entwicklungsprodukt Marktstrategie zuständig. Darüber hinaus definiert es die Mark tsegmentierung (beides siehe Abschnitt 4.3).

7.2.2 Entwicklungsorientierte Rollen

Entwicklungsorientierte Rollen sind letztendlich für die Realisierung von A nforderungen zuständig. Sie stellen sicher, dass innerhalb eines gegebenen Zeitfensters u nter verfügbaren Ressourcen so viele Anforderungen, wie möglich, implementiert werden. Gleichze itig muss die resultierende Software flexibel für künftige Anforderungsänderu ngen sein.

Systemstratege

Der Systemstratege ist das entwicklungsseitige Pendant des strategischen Mark etings. Seine Hauptaufgabe ist, langfristig eine effiziente Weiterentwicklung der Software s icherzustellen. Dazu definiert er Systemstandards, Systemstrategie (vgl. Abschnitt 4.3), und Systemarchitektur (vgl. Abschnitt 4.4.3) der zu entwickelnden Software. Systemstr ategie und Systemarchitektur werden von Version zu Version aktualisiert (si ehe Abschnitt 7.5.2. Aktivität *Definition der Systemarchitektur* und *Definition der Systemstrategie*).

Systemmodulplaner

Systemmodulplaner sind für die Planung, Fortschrittskontrolle und Realisierung einze lner Systemmodule (vgl. Abschnitt 4.4.3) zuständig. Dazu entscheiden sie, welche Anfo rderungen in welcher Version in welchem Umfang realisiert werden sollen. Falls es aus irgendwelchen Gründen zu Änderungen im ursprünglich geplanten Entwicklungsa blauf kommt, nehmen sie gegebenenfalls Umplanungen vor. Sie sind hauptve rantwortlich für die Entscheidung über den Auslieferungszeitpunkt eines Softwareprodukts.

Softwareentwickler

Die letztendliche Entwicklung der Software, das heißt das Feindesign, die Implementi erung und die Integration der Systemmodule übernehmen Softwareentwickler. Die En twicklung basiert auf den Anforderungen, den Versionsplänen, der Systemarchitektur, der Systemstrategie und den Systemstandards (vgl. Abschnitte 4.2.1, 4.4.2 und 4.4.3). Softwareentwickler unterstützen die Versionsplanung (vgl. Abschnitt 7.5.3) durch die Bewertung alternativer Lösungen und notwendiger Aufwandsabschätzu ngen.

Softwaretester

Softwaretester sind für die Verifikation der Software (siehe Abschnitt 1.2.4) hinsichtlich der zur Realisierung geplanten Anforderungen zuständig. Für diesen Zweck definieren sie Testfälle basierend auf der Anforderungsspezifikation für ein Systemmodul. Um insbesondere wichtige Anforderungen adäquat zu testen, werden Bewertungen der Anforderungen verwendet (vgl. Kapitel 6).

7.3 *Beispielanwendung* Pisa

Dieser Abschnitt stellt das durchgängige Fallbeispiel vor, anhand dessen einzelne Aktivitäten der Abschnitte 7.4 und 7.5 demonstriert werden. Hierzu dient das Requirements Engineering eines Produktsystems mit dem Namen *Pisa*. Das Beispiel ist bewusst klein gehalten, um es im Rahmen dieser Arbeit behandeln zu können. Zu Demonstrationszwecken wird jedoch unabhängig davon eine Aufteilung der Anforderungen auf unterschiedliche Systemmodule vorgenommen. In den folgenden Unterabschnitten wird zunächst die zu lösende Problemstellung des Produktsystems beschrieben (Abschnitt 7.3.1), ehe auf die konkrete Ausgangssituation für die Entwicklung des Produktsystems eingegangen wird.

7.3.1 Problemstellung der Beispielanwendung Pisa

Die Deutsche Post AG bietet zwei gesonderte Postversendungsarten, genannt *Infopost* [DPa] und *Infobrief* [DPb] an. Diese sind für Kunden gedacht, die eine größere Anzahl von Poststücken mit dem gleichen Inhalt an verschiedene Empfänger versenden wollen. Beachtet der Absender einige Regeln, so erhält er als Gegenleistung eine beträchtliche Portoersparnis. So muss er beispielsweise die Sendungen in einer vordefinierten Weise sortieren, stempeln und verpacken. Die Höhe der Preisreduktion hängt dabei vom Ziel, der Anzahl und der Art der Sendungen ab. Abhängig davon muss der Absender die Sendungen möglicherweise auch in unterschiedliche Container-Arten verpacken, das heißt entweder in speziell vorgefertigte Behälter oder auf Paletten.

Derartige Sendungsformen können z. B. Briefsendungen für das Direktmarketing einzelner Firmen sein. Laut einer Studie [DP01] wurde allein hierfür 1998 37 Mrd. DM aufgewendet, wobei ein zunehmender Anteil an mittleren Unternehmen sich wesentlich erhöhte. Der Anteil adressierter Werbesendungen, zu denen auch *Infopost* und *Infobrief* zählen, betrug 37%, was 302000 Unternehmen entsprach.

7.3.2 Ausgangssituation der Entwicklung

Zu Beginn der Entwicklung von *Pisa* sind bereits strategische Entwicklungsprodukte festgelegt. In Anhang D sind Ausschnitte der Marktstrategie und der Marktsegmentierung (vgl. Abschnitt 4.3) illustriert. Die Marktstrategie enthält in diesem Fall vor allem die Kernkompetenzen und charakterisiert Schlüsselkunden. Darüber hinaus wurden vorab wesentliche Quellen von Anforderungen identifiziert. Von diesen Informationen ausgehend werden in Abschnitt 7.4 und 7.5 vorgestellte Aktivitäten durchgeführt. Das Beispiel wird sukzessive in diesen Abschnitten anhand der Aktivitäten fortgeführt.

7.4 Kontinuierliches Herausarbeiten von Anforderungen

Jetzt wird auf das Kontinuierliches Herausarbeiten von Anforderungen eingegangen. Um eine Standardsoftware konkurrenzfähig zu halten, müssen kontinuierlich Aktivitäten durchgeführt werden, die neue Anforderungen herausarbeiten. In dieser Arbeit wird zwischen Anforderungen mit kurzfristigem Zeithorizont und Anforderungen mit langfristigen Zeithorizont unterschieden. Mit einer Abgrenzung der beiden Zeithorizonte und einer Beschreibung von Aktivitäten zum Herausarbeiten von Anforderungen des langfristigen Zeithorizonts befasst sich Abschnitt 7.4.1.

Für das „klassische“ Requirements Engineering ist das Herausarbeiten von Anforderungen mit kurzfristigem Zeithorizont wesentlich. Das Herausarbeiten von Anforderungen mit langfristigem Zeithorizont ist hingegen Aufgabe des Innovationsmanagements (siehe auch Abschnitt 7.4.1). In Abschnitt 7.4.2 werden systematisch verschiedene Quellen von Anforderungen vorgestellt. Anschließend werden in Abschnitt 7.4.3 unterschiedliche Erfassungskanäle präsentiert und daraus resultierende Aktivitäten des Herausarbeitens diskutiert. Sind für eine Kanalart bestehende Methoden des Herausarbeitens von Anforderungen (vgl. Abschnitt 2.1) geeignet, so wird hierfür gesondert darauf hingewiesen.

7.4.1 Abgrenzung vom Innovationsmanagement

In diesem Abschnitt erfolgt eine Abgrenzung des Innovationsmanagements vom Requirements Engineering. Für Standardsoftware reicht es wie bei vielen materiellen Produkten nicht aus, nur Anforderungen herauszuarbeiten, die kurzfristig in neue Versionen integriert werden. Daher werden weit über das eigentliche Requirements Engineering hinausgehende Aktivitäten benötigt. In [BDD99] wurde hierzu gemeinsam mit anderen Autoren ein Prozessmodell zum Innovationsmanagement komplexer Standardsoftware entwickelt.

Zwischen Requirements Engineering und Innovationsmanagement kann keine scharfe Abgrenzung gemacht werden. Tendenziell sind jedoch drei Unterschiede festzustellen. Requirements Engineering und Innovationsmanagement unterscheiden sich sowohl in der Intensität des Bezugs der Anforderungen zum bestehenden Produktsystem (*Produktbezug*), der mittleren Zeitdauer von der Ermittlung einer Anforderung bis zu ihrer Integration in das Produktsystem (*Zeithorizont*) und im Umfang vorliegenden Hintergrundwissens über eine Anforderung im Unternehmen (*Neuartigkeit*).

Im Rahmen des Requirements Engineering werden tendenziell produktnahe Anforderungen ermittelt, die in einem relativ kurzen Zeitraum umgesetzt werden. Meist ist auch bereits entsprechendes Hintergrundwissen vorhanden, wie beispielsweise hinsichtlich des Marktnutzens und der Möglichkeiten der Integration in das Produktsystem.

Demgegenüber befasst sich das Innovationsmanagement mit Anforderungen, die sich nicht unmittelbar am bestehenden Produktsystem orientieren müssen und häufig erst nach einem längeren Zeitraum in das Produktsystem integriert werden. Als Ausgangspunkt zu ermittelnder Anforderungen dienen Neuerungen über die in der Regel kein oder nur geringes Hintergrundwissen im Unternehmen vorliegt. Der Unterschied wird in Abbildung 30 illustriert.

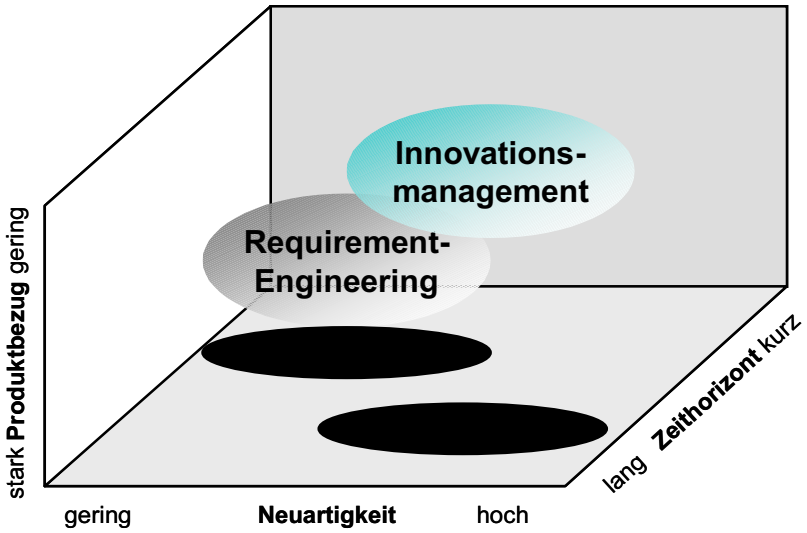


Abbildung 30 – Unterschied zwischen Requirements Engineering und Innovationsmanagement

Bei Innovationen kann zwischen zwei wesentlichen Arten unterschieden werden: den anforderungsgetriebenen und technologiegetriebenen Innovationen . Anforderungsgetriebene Innovationen entstehen aus der reinen Beobachtung des Anwendermarktes. Dabei werden bestehende Bedarfe identifiziert, die durch Neuentwicklungen gedeckt werden sollen. Demgegenüber sind technologiegetriebene Innovationen in erster Linie von der Weiterentwicklung der Technologie getrieben. Mit Unterstützung neuer Technologien können neue Produkte entwickelt werden, die neue Bedarfe am Markt wecken sollen. Ein Beispiel hierfür ist der momentan in Diskussion stehende Mobilfunkstandard UMTS, der durch die höhere Übertragungsraten neue Nutzungsmöglichkeiten mobiler Endgeräte schaffen soll.

Die vorliegende Arbeit konzentriert sich auf das Requirements Engineering. Aus diesem Grunde folgt hier nur eine kurze Zusammenfassung der Tätigkeiten des Innovationsmanagements. Es besteht im wesentlichen aus drei Teilprozessen:

- **Planung:** Übergeordneter Prozess, der im wesentlichen der Organisation des gesamten Innovationsmanagements dient. Sie definiert die generelle Ausrichtung, die betrachteten Themenschwerpunkte und durchzuführende konkrete Untersuchungen.

- **Überwachung:** Diese dient der ständigen Erfassung und groben Auswahl von Themen durch die aktive oder passive Auswertung spezifischer Quellen. Die Themenerfassung kann durch Auswertungen externer Partner unterstützt werden, wie beispielsweise von Instituten für die Marktbeobachtung. Jeder Überwachungsprozess besitzt eine spezifische thematische Ausrichtung, die jedoch bewusst unscharf gehalten ist, um nicht innovationshemmend zu wirken. Für jeden Themenbereich werden zusätzlich Überwachungsschwerpunkte festgelegt. Ein Themenbereich könnten beispielsweise HW/SW-Trends sein, wobei hier die Mensch-Maschine-Schnittstelle einen Schwerpunkt darstellen könnte.

- **Schwerpunktuntersuchung:** Aufgrund intuitiver Einschätzungen werden manche Themen als interessant eingestuft. Für diese werden dann Untersuchungsprozesse mit engerer Abgrenzung der Themenschwerpunkte und Zielsetzungen initiiert. Schwerpunktuntersuchungen können sich auf sehr unterschiedliche Themenbereiche konzentrieren. Beispiele hierfür sind Entscheidungsvorbereitungen für die Umsetzung von Ideen in einem Produktsystem oder Analysen zu erwartender Entwicklungen in spezifischen Kundengruppen.

Ein wesentliches Problem für die Standardsoftwareentwicklung ist die Schnittstelle zwischen dem Innovationsmanagement und der konkreten Systementwicklung. Generell sind Anforderungen des Innovationsmanagements neuartiger. Das heißt, sie verursachen weitaus umfassendere Änderungen im gesamten Produktsystem, als Anforderungen aus dem produktnahen Requirements Engineering. Während Anforderungen aus dem Requirements Engineering in einem laufenden zyklischen Weiterentwicklungsprozess integriert werden, sind Innovationen häufig mit höheren Aufwänden verbunden, da sie gegebenenfalls enorme Umstrukturierungen nach sich ziehen. Problematisch ist zusätzlich, dass die Abschätzung des möglichen Erfolges für Innovationen aufgrund weniger vorhandener zuverlässiger Informationen noch wesentlich schwerer ist. Daher ist die Entscheidung für die Integration von Innovationen in einem Produktsystem stark von der Risikobereitschaft des Unternehmens abhängig.

7.4.2 Quellen von Anforderungen

Dieser Abschnitt stellt unterschiedliche Arten von Quellen von Anforderungen vor. Mit *Quelle* wird in dieser Arbeit der eigentliche Ursprung einer Anforderung bezeichnet. In diesem Abschnitt werden für die Entwicklung komplexer Standardsoftware relevante Quellentypen und deren Charakteristika vorgestellt. Die Klassifikation stellt das Ergebnis eingehender Diskussionen mit Praktikern dar (vgl. [FOR97]). gibt einen Überblick über Quellen von Anforderungen.

Es kann zwischen Kunden und Anwendern, Ideenlieferanten, Software, Innovationsmanagement sowie Normen und Standards unterschieden werden. Darüber hinaus lassen sich Quellen aus Firmensicht in interne und externe Quellen differenzieren. Die Unterscheidung in Quellentypen dient einerseits der Definition spezifischer Aktivitäten des Herausarbeitens (siehe Abschnitt 7.4.3). Andererseits können sie als Klassifizierungsmerkmale zur Beurteilung von Erfahrungen herangezogen werden (vgl. Abschnitt 6.7.4 und Anhang E).

Firmensicht

Aus der Sicht eines Herstellers einer Standardsoftware lassen sich Quellen folgendermaßen charakterisieren:

Interne Quellen: Von internen Quellen wird gesprochen, wenn die gesammelten Informationen von innerhalb des Unternehmens stammen. Hierzu gehören vor allem das eigene Softwareprodukt, eigene Mitarbeiter und verschiedene interne Standards.

Externe Quellen: Bei externen Quellen stammen die Informationen von außerhalb der Firma. Dazu zählen Kunden, Konkurrenzprodukte, Basissysteme, Forschungseinrichtungen, Gesetze, Standards sowie sonstige Informationsquellen, wie zum Beispiel verschiedenste Medien.

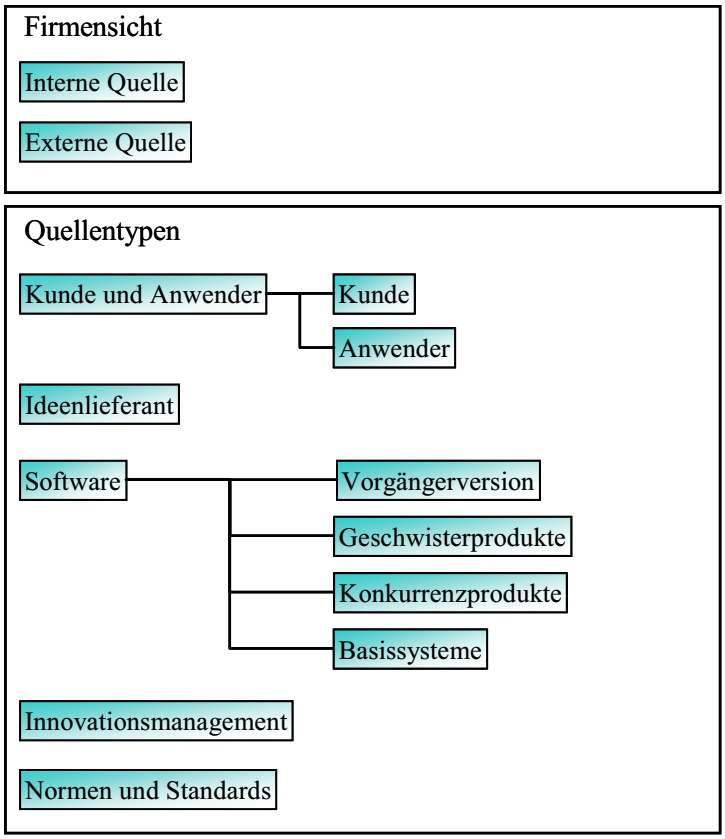


Abbildung 31 – Quellen von Anforderungen

Kunde und Anwender

Kunde und Anwender sind wesentliche Anforderungslieferanten für Standardsoftware. Aufgrund des vielfältigen Marktes sind diese hier jedoch nicht die alleinentscheidende Quelle, wie es meist bei Individualsoftware der Fall ist.

Kunde: Unter Kunden wird der eigentliche Käufer einer Standardsoftware verstanden. Er bestimmt den Anwendungskontext der Software und legt daher vor allem Anforderungen an dessen eigentliche Kernfunktionalität fest. Der Anwendungskontext einer betrieblichen Software ist beispielsweise die Unterstützung von Geschäftsprozessen.

Anwender: Der Anwender bedient die Software unmittelbar, um damit Aufgaben in dem erwähnten Anwendungskontext zu erledigen. Daher beschränkt sich sein Blickfeld lediglich auf einen Ausschnitt der Kernfunktionalitäten. Der Anwender liefert verglichen mit dem Kunden vor allem zusätzliche Anforderungen an die Bedienung der Software.

Kunde und Anwender eines Softwaresystems können sowohl firmenintern als auch firmenextern sein. Zu den firmeninternen Kunden und Anwendern zählen neben den eigentlichen Softwareanwendern beispielsweise auch Handbuchautoren und Schulungsleiter.

Ideenlieferanten

Ideenlieferanten sind nicht unmittelbar Kunden oder Anwender eines Softwaresystems, sondern haben einen anderen Bezug zur Software. Dazu zählen beispielsweise Entwickler und Führungsebenen. Unter Ideen werden unkonventionelle und neuartige Anforderungen verstanden.

Ideenlieferanten können in ihrer Kreativität besonders unterstützt werden. Dazu sieht beispielsweise das Prozessmodell von Microsoft in jeder Entwicklungsphase gesonderte Pufferzeiten vor. Diese können von Entwicklern genutzt werden, um unter anderem eigene Ideen zu entwerfen (vgl. [CuS95]). Neben internen Ideenlieferanten können auch besonders innovative externe Firmen und Institutionen eine gute Quelle von Ideen sein.

Software

Ein weiterer Quellentyp ist sämtliche bestehende Software, die in irgendeinem Bezug zu dem zu entwickelnden Softwaresystem steht. Dazu gehören neben der Vorgängerversion des eigentlichen Softwaresystems auch Geschwisterprodukte, Konkurrenzprodukte und Basissysteme. Im Folgenden wird kurz auf die Charakteristika dieser verschiedenen Softwaresysteme eingegangen.

Vorgängerversion: Beim Kauf einer neuen Version einer Standardsoftware wird vom Kunden implizit die Kompatibilität zu deren Vorgängerversion erwartet. Es müssen also gewisse Anforderungen an die alte Programmversion weiterhin erfüllt sein. Bei Neuerungen muss also auch der Einfluss alter Anforderungen berücksichtigt werden.

Geschwisterprodukte: Komplexe Standardsoftware besteht aus mehreren isoliert verkauften Softwareprodukten. Diese nutzen jeweils Funktionalitäten ihrer Geschwisterprodukte. Dadurch entstehen Anforderungen an das gegenseitige Zusammenspiel der einzelnen Softwareprodukte untereinander.

Konkurrenzprodukte: Unter Konkurrenzprodukten wird Standardsoftware anderer Hersteller verstanden, die im gleichen Anwendungsbereich wie die eigene Standardsoftware eingesetzt werden kann. Aus deren Analyse können sowohl Anforderungen an das eigene Produkt als auch Differenzierungsfaktoren gegenüber den Konkurrenzprodukten identifiziert werden. Darüber hinaus können auch Informationen für die Bewertung bestehender Anforderungen gewonnen werden (vgl. [Pfe93]).

Basissysteme: Standardsoftware nutzt in der Regel Funktionalitäten von unterschiedlichen Basissystemen (siehe Glossar). Dazu zählen zum Beispiel Betriebssysteme und Datenbanksysteme. Durch die Weiterentwicklung von Basissystemen entstehen neue Anforderungen an die eigene Standardsoftware.

Innovationsmanagement

In unregelmäßigen Abständen werden vom eigenen Innovationsmanagement des Standardsoftwareherstellers neue Anforderungen an die Entwicklung herangetragen (vgl. Abschnitt 7.4.1).

Normierungen und Standards

Sowohl für viele Anwendungsbereiche als auch für die technische Umsetzung einer Standardsoftware bestehen unterschiedliche Normierungen und Standards. Diese können sowohl innerbetrieblich definiert als auch extern von Gremien, Regierungen usw. festgelegt sein. Beide sind eine wichtige Quelle von Anforderungen an die Standardsoftware.

7.4.3 Aktivitäten des Herausarbeitens von Anforderungen

In diesem Abschnitt werden Aktivitäten zum Herausarbeiten von Anforderungen für Standardsoftware beschrieben. Neben den erläuterten Quellen existieren zum Herausarbeiten von Anforderungen auch unterschiedliche Informationskanäle oder kurz Kanäle. Diese beschreiben den Weg, worüber eine Anforderung an die Standardsoftware herausgearbeitet wurde. Abhängig von Quelle und Kanal unterscheiden sich die Vorgehensweisen der Aktivitäten und die beteiligten Rollen.

Eine zentrale Rollen spielt hier der Sammlungskoordinator. Dieser ist für eine Systematisierung aller Aktivitäten des Herausarbeitens zuständig. Er motiviert, animiert und koordiniert alle beteiligten Rollen und übernimmt auch die Verwaltung gesammelter Anforderungen. Allgemein lässt sich das Herausarbeiten von Anforderungen mit dem Schema aus Abschnitt 3.1.3 wie folgt beschreiben:

Aktivitäten des Herausarbeitens von Anforderungen

Vorarbeit Herausarbeiten von Anforderungen	
S	Vorarbeit für die zyklische Aktivität *Anforderungen filtern* (vgl. Abschnitt 7.5.1) und die Anwendung des Kontextmodells (vgl. Abschnitt 6.7) zur Bewertung von Anforderungen.
R	Abhängig von Quellen- und Kanaltypen
O	-
E	Neue Anforderungen (vgl. Abschnitt 4.2.1, aggregierte Beschreibung erfasst, Filterung=keine) und zugehörige Erfassungskontexte (vgl. Abschnitt 4.4.1)
A	Abhängig von Kanaltyp
S=Entscheidungssituation, R=Rollen, O=Vorarbeitsobjekte, E=Vorarbeitsergebnis und A=Anleitungen/Vorlagen	

Motivation: Um schnellstmöglich auf Änderungen am Markt reagieren zu können, müssen fortwährend – unabhängig von Versionszyklen – Anforderungen herausgearbeitet werden. Durch das kontinuierliche Herausarbeiten werden neue Anforderungen früher erkannt, als wenn erst nach Beendigung der Entwicklung einer Version Anforderungen für die sich anschließende Version herausgearbeitet werden würden. Treten sehr dringliche Anforderungen auf, so kann unter Umständen unmittelbar die Entwicklung einer neuen Version eines oder mehrerer Systemmodule initiiert werden.

Vorgehen: Je nach Quellen- und Kanaltyp gibt es unterschiedliche Verfeinerungen der Aktivität des Herausarbeitens von Anforderungen. Außer bei passiven Kontakten (siehe gleich im Anschluss) ist der Sammlungskoordinator für die Initiierung verschiedener Aktivitäten zuständig. Er übernimmt die Bedarfsermittlung und Durchführungsplanung der Aktivitäten des Herausarbeitens von Anforderungen.

In den folgenden Absätzen werden verschiedene Kanaltypen des Herausarbeitens und quellen-spezifische Aktivitäten vorgestellt. Abbildung 32 gibt zunächst einen Überblick über Kanäle des Herausarbeitens von Anforderungen. Die Firmensicht unterscheidet zwischen internen und externen Kanälen. Darüber hinaus wird zwischen aktiven Kontakten, passiven Kontakten, Softwareanalysen, Literaturrecherchen und Erfahrungen aus Zusammenarbeit differenziert.

Spezifische Aktivitäten des Herausarbeitens von Anforderungen unterscheiden sich zwar aufgrund unterschiedlicher Quellen und Kanäle, nicht jedoch in der Entscheidungssituation, wofür sie eine Vorarbeit leisten. Ebenso sind deren Ergebnisproduktttypen und Motivation identisch. Daher wird in diesem Abschnitt ein vereinfachtes Schema zur Beschreibung einzelner Aktivitäten verwendet, das nur beteiligte Rollen und das spezifische Vorgehen enthält. Anleitungen und Vorlagen sind lediglich vom übergreifenden Kanaltyp (linke Seiten in Abbildung 32) abhängig. Es folgt nun eine Beschreibung der einzelnen Kanaltypen, der daraus abgeleiteten Aktivitäten sowie zugehöriger Anleitungen und Vorlagen.

Firmensicht

Aus Firmensicht kann man interne und externe Kanäle unterscheiden.

Interne Kanäle: Werden die Sammelaktivitäten von firmeneigenen Mitarbeitern durchgeführt, so wird dies mit internen Kanälen bezeichnet.

Externe Kanäle: Um externe Kanäle handelt es sich, wenn Firmenexterne mit der Akquisition von Anforderungen beschäftigt sind. Die Nutzung externer Kanäle verspricht die Reduktion von eigenen Aufwänden für das Herausarbeiten von Anforderungen. Insbesondere in sehr spezialisierten Anwendungsbereichen verringert sich der Einarbeitungsaufwand erheblich, wenn auf externe Expertise zugegriffen werden kann. Das Sammeln von Anforderungen über externe Kanäle kann sowohl im Auftrag der eigenen Firma oder auftragsunabhängig erfolgen. Auftragsunabhängige Sammelaktivitäten reduzieren den Aufwand eines Standardsoftwareherstellers.

Diese Klassifizierung macht hinsichtlich der Vorgehensweisen keinen Unterschied. Lediglich im Hinblick auf die Bewertung einzelner Anforderungen können sich abhängig von der Firmensicht unterschiedliche Erfahrungen ableiten (vgl. Abschnitt 6.7 und Anhang E).

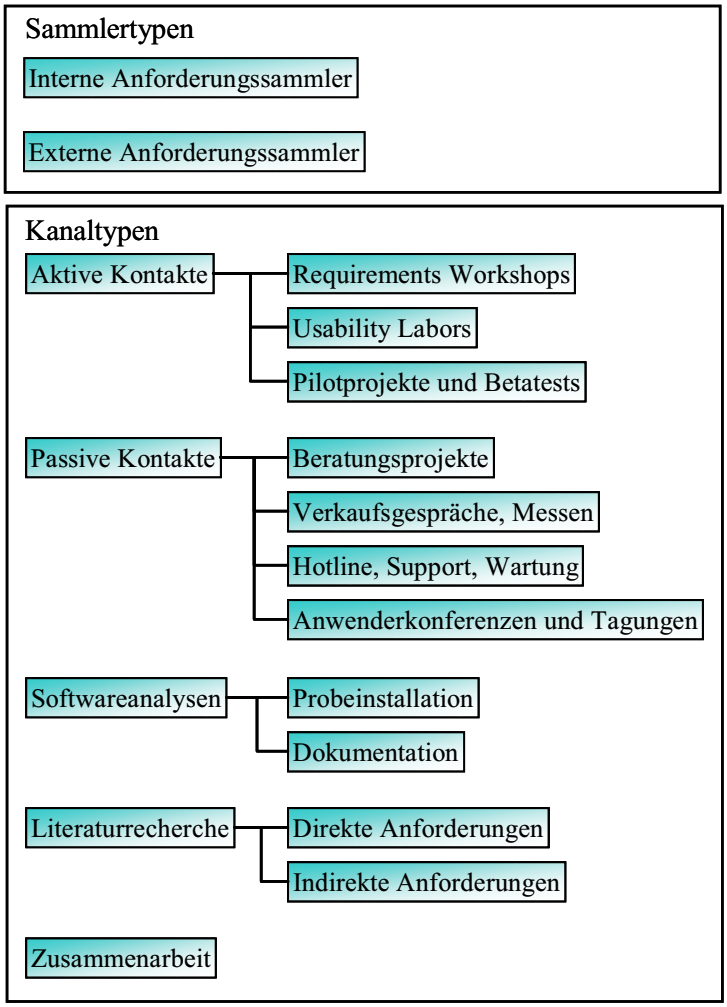


Abbildung 32 – Kanäle des Herausarbeitens

Aktive Kontakte

Gerade durch das Manko eines direkten Auftragsgebers sind intensive Kontakte zu unterschiedlichen Quellen für einen Standardsoftwarehersteller unabdinglich. Unter aktiven Kontakten werden gezielte Veranstaltungen zum Herausarbeiten von Anforderungen mit ausgesuchten Teilnehmern verstanden.

Anleitungen/Vorlagen: Für alle aktiven Kontakte können die in Abschnitt 2.1 vorgestellten Techniken zum Herausarbeiten von Anforderungen mit geringen Anpassungen angewendet werden. Zur Aufwandsreduktion bieten sich fallweise statt eigenständiger Veranstaltungen auch schriftliche Umfragen an.

Aktivitäten: Zu den aktiven Kontakten gehören Requirements Workshops, Usability Labors beziehungsweise Pilotprojekte und Betatests:

A 1 Herausarbeiten von Anforderungen durch Requirements Workshops

Vorarbeit Herausarbeiten von Anforderungen durch Requirements Workshops (Quellentyp: ausgewählte Kunden)	
R	**Sammlungskoordinator** und andere
R=Rollen, Rest gleich zu Herausarbeiten von Anforderungen	

Vorgehen: Requirements Workshops dienen gleichzeitig dem Herausarbeiten, dem Verhandeln und der Validierung von Anforderungen aus dem Anwendungsumfeld (siehe Glossar). Je nach Fortschritt der laufenden Entwicklung werden in den Veranstaltungen gezielte Interviews durchgeführt, Folien präsentiert oder erste Mock-Ups (siehe Glossar) beziehungsweise Prototypen mit möglichen Ablaufszenarien vorgeführt. Requirements Workshops werden zusammen mit ausgewählten Kunden veranstaltet.

A 2 Herausarbeiten von Anforderungen durch Usability Labors

Vorarbeit Herausarbeiten von Anforderungen durch Usability Labors (Quellentyp: ausgewählte Anwender)	
R	**Sammlungskoordinator** und andere
R=Rollen, Rest gleich zu Herausarbeiten von Anforderungen	

Vorgehen: Das Hauptinteresse der sogenannten Usability Labors liegt in der Optimierung der Bedienbarkeit der Software. Diese werden anhand von Prototypen mit Anwendern der Software durchgeführt.

A 3 Herausarbeiten von Anforderungen durch Pilotprojekte und Betatests

Vorarbeit Herausarbeiten von Anforderungen durch Pilotprojekte und Betatests (Quellentyp: ausgewählte Kunden und Anwender)	
R	**Sammlungskoordinator**, Vertriebsbeauftragter, Softwaretester und andere
R=Rollen, Rest gleich zu Herausarbeiten von Anforderungen	

Vorgehen: Kurz vor Fertigstellung eines Standardsoftwaresystems können Pilotprojekte mit ausgewählten Anwendern und Kunden durchgeführt werden. Diese dienen vorrangig einem letzten Feinschliff beziehungsweise dem Testen der Software. Darüber hinaus werden in Pilotprojekten Anforderungen an nachfolgende Versionen ermittelt. Aufgrund des geltenden Zeitfensters können diese meistens nicht mehr in die aktuelle Version integriert werden.

Passive Kontakte

Anforderungen können auch aus Kontakten gewonnen werden, die nicht unmittelbar dem Zweck des Herausarbeitens dienen. Die an diesen sogenannten passiven Kontakten beteiligten Personen müssen daher hierfür besonders sensibilisiert werden. Insbesondere ist hier die Aufgabe des Sammlungskoordinators diese Personen besonders intensiv zu betreuen, um diese Anforderungsquelle gut nutzen zu können. Eine adäquate Werkzeugunterstützung, wie in Abschnitt 4.5 kurz skizziert, trägt darüber hinaus wesentlich zur sinnvollen Nutzung dieses Kanals bei.

Anleitungen/Vorlagen: Da passive Kontakte nicht unmittelbar für das Herausarbeiten von Anforderungen veranlasst sind, erfolgt hier weder ein gezielter Methodeneinsatz noch werden Ansprechpartner gezielt für das Requirements Engineering ausgewählt.

Aktivitäten: Anforderungen aus passiven Kontakten können im Rahmen von Beratungsprojekten, Verkaufsgesprächen und Messen, Hotline, Support und Wartung beziehungsweise Anwenderkonferenzen und Tagungen gewonnen werden:

A 4 Herausarbeiten von Anforderungen in Beratungsprojekten

Vorarbeit Herausarbeiten von Anforderungen in Beratungsprojekten (Quellentyp: Kunden)	
R	**Vertriebsbeauftragter**
R=Rollen, Rest gleich zu Herausarbeiten von Anforderungen	

Vorgehen: Komplexe Standardsoftware wird häufig vom Softwarehersteller selbst im Rahmen von Beratungsprojekten beim Kunden eingeführt. Die Berater können dabei Erfahrungen sammeln, woraus sich neue Anforderungen an die Software ableiten lassen (vgl. auch [CuS95]).

A 5 Herausarbeiten von Anforderungen in Verkaufsgesprächen und Messen

Vorarbeit Herausarbeiten von Anforderungen in Verkaufsgesprächen und Messen (Quellentyp: ausgewählte Kunden)	
R	**Vertriebsbeauftragter**
R=Rollen, Rest gleich zu Herausarbeiten von Anforderungen	

Vorgehen: In Verkaufsgesprächen mit Kunden und Anwendern werden Erwartungen an das Softwaresystem diskutiert. Diese können teilweise als Anforderungen an künftige Versionen der Software formuliert werden. Messen können darüber hinaus auch für Konkurrenzanalysen genutzt werden.

A 6 Herausarbeiten von Anforderungen bei Hotline, Support und Wartung

Vorarbeit Herausarbeiten von Anforderungen bei Hotline, Support und Wartung (Quellentyp: Kunden)	
R	**Hotline-Mitarbeiter/Softwaretester**
R=Rollen, Rest gleich zu Herausarbeiten von Anforderungen	

Vorgehen: Hotline, Support und Wartung dienen der Diskussion von Problemen bei der konkreten Nutzung der Software. Ursachen von Problemen können neben Realisierungsfehlern auch nicht erfüllte Anforderungen sein, die dann sofort erfasst werden müssen.

A 7 Herausarbeiten von Anforderungen in Anwenderkonferenzen und Tagungen

Vorarbeit Herausarbeiten von Anforderungen in Anwenderkonferenzen und Tagungen (Quellentyp: Kunden, Anwender, Ideenlieferanten)	
R	**Entwickler/Vertriebsbeauftragte** oder andere.
R=Rollen, Rest gleich zu Herausarbeiten von Anforderungen	

Vorgehen: Zur Präsentation neuer Entwicklungen der Software führen Standardsoftwarehersteller Anwenderkonferenzen durch. Da auch hier viele Anwender und Kunden anwesend sind, ist auch dies eine Plattform zur Diskussion von Anforderungen. Insbesondere zur Akquisition von Innovationen können Tagungen im Anwendungsumfeld (vgl. Glossar) der Software beziehungsweise Technologietagungen genutzt werden.

Softwareanalysen

Aus bestehender Software können Anforderungen durch gesonderte Analysen abgele itet werden.

Anleitungen/Vorlagen: Zur Softwareanalyse können Reverse Engineering Methoden verwendet werden (vgl. Abschnitt 2.1.1).

Aktivitäten: Für Softwareanalysen gibt es keine weitere Untergliederung von Aktivit ä-ten:

A 8 Herausarbeiten von Anforderungen durch Softwareanalysen

Vorarbeit Herausarbeiten von Anforderungen durch Softwareanalysen (Quellentyp: Software)	
R	**Systemmodulplaner/Technisches Marketing**
R=Rollen, Rest gleich zu Herausarbeiten von Anforderungen	

Vorgehen: Bestehende Software wird anhand von Probeinstallationen oder vorhandener Dokumentation analysiert um Anforderungen abzuleiten.

Literaturrecherche

Anforderungen können auch aus allgemein verfügbarer Fachliteratur ermittelt we rden.

Anleitungen/Vorlagen: Hierzu existieren bisher in der Requirements Engineering-Literatur keine gesonderten Methoden.

Aktivitäten: Für Literaturrecherchen kann zwischen expliziter und indirekter Darste l-lung unterschieden werden:

A 9 Herausarbeiten von Anforderungen aus Literatur mit expliziter Darstel lung

Vorarbeit Herausarbeiten von Anforderungen aus Literatur mit expliziter Darstellun g (Quellentyp: Normen, Standards)	
R	**Sammlungskoordinator**
R=Rollen, Rest gleich zu Herausarbeiten von Anforderungen	

Vorgehen: Schriftstücke mit expliziter Darstellung beschreiben unmittelbar Anforderu n-gen an eine Standardsoftware. Dazu zählen verschiedenste No rmen und Standards, die ohne Änderung in Anforderungsspezifikationen übernommen werden können. Beim Herausarbeiten dieser Anforderungen bedarf es keiner beso nderen Techniken.

A 10 Herausarbeiten von Anforderungen aus Literatur mit indirekter Darste llung

Vorarbeit Herausarbeiten von Anforderungen aus Literatur mit indirekter Darstellung (Quellentyp: Software, Ideenlieferant, Kunde, Anwender)	
R	**Sammlungskoordinator**
R=Rollen, Rest gleich zu Herausarbeiten von Anforderungen	

Vorgehen: Fachliteratur, die nicht auf die explizite Darstellung von Anforderungen a b-zielt, kann als auftragsunabhängiger externer Kanal angesehen werden. Anstatt be i-spielsweise in einer Firma selbst Vergleichstests mit Konkurrenzsoftware durchzuführen,

kann entsprechende Fachliteratur genutzt werden. Aus den Vergleichstests können sowohl Anforderungen als auch eventuelle Gewichtungen abgelesen werden. Darüber hinaus können spezielle Rubriken mit Anwenderfragen in Fachzeitschriften sowohl als Ergänzung zur eigenen Hotline als auch zur Identifikation von Schwachstellen in Konkurrenzprodukten genutzt werden. Schließlich können viele andere Artikel in Fachzeitschriften, Internet usw. genutzt werden, um auch Ideen für innovative Erweiterungen zu ermitteln. Zum Vorgehen existiert hier keine Standardmethode.

Erfahrungen aus Zusammenarbeit

Zuletzt können vielfältige Erfahrungen als auch Anforderungen aus der Zusammenarbeit mit Kunden, Anwendern und Lieferanten von Basissystemen gewonnen werden.

Anleitungen/Vorlagen: Die systematische Sammlung von Erfahrungen wird durch den Einsatz des in Abschnitt 6.7 vorgestellten Kontextmodells unterstützt. Aus den Erfahrungen können Anforderungen extrahiert werden.

Aktivitäten: Für das Herausarbeiten von Anforderungen aus Zusammenarbeit gibt es keine weitere Untergliederung von Aktivitäten:

A 11 Herausarbeiten von Anforderungen aus Zusammenarbeit

Vorarbeit Herausarbeiten von Anforderungen aus der Zusammenarbeit mit Kunden, Anwendern und Lieferanten von Basissystemen (Quellentyp: ausgewählte Kunde, Anwender und Basissysteme)	
R	Vertriebsbeauftragter, Entwickler, **Sammlungskoordinator** und andere.
R=Rollen, Rest gleich zu Herausarbeiten von Anforderungen	

Vorgehen: Ein Mitarbeiter des Herstellers der Standardsoftware arbeitet vorübergehend beim Kunden, Anwender oder Hersteller von Basissystemen (siehe Glossar) mit, um deren Arbeitsweise und Denkweise zu analysieren. Daraus gewonnene Erkenntnisse werden ausgewertet und letztendlich in Anforderungen an die eigene Standardsoftware formuliert.

Beispielanwendung *Pisa*

Anforderungen für den zentralen Kern des Produktsystems Pisa wurden vorrangig durch Herausarbeiten von Literatur mit expliziter Darstellung gewonnen. Bei der Deutschen Post AG existieren verschiedene Informationsschriften, die Voraussetzungen für den Versand von Infopost und Infobrief beschreiben. Vor allem hinsichtlich der Optimierung des Portos ist dies auch der einzige notwendige Informationskanal.

Darüber hinaus wurden Schlüsselkunden im Rahmen eines Requirements Workshops mit einer offenen Befragung befragt. Hierzu wurde ein Vorstandsmitglied A eines Vereins mit ca. 1000 Mitgliedern, ein Angestellter B des Wahlreferats einer Großstadt mit 200000 Wahlberechtigten sowie zwei Marketingbeauftragte C, D einer Lotteriegesellschaft und eines Kosmetikherstellers eingeladen. Da es sich hier um vier verschiedene Kunden handelt, werden hierzu vier unterschiedliche Erfassungskontexte (vgl. Abschnitt 4.4.1) erfasst. In Anhang D – Profile der Schlüsselkunden *sind Profile der beteiligten Schlüsselkunden angegeben.*

Daneben wurden über Vertriebsbeauftragte von anderen Kunden Anforderungen erfasst. In Anhang D – Erfasste Anforderungen *ist ein kleiner Ausschnitt der erfassten ungefilterten Anforderungen (vgl. Abschnitt 4.2.1) zusammen mit den zugehörigen Erfassungskontexten aufgelistet.*

7.5 Zyklische Phasen

In diesem Abschnitt werden die Aktivitäten der zyklischen Phasen des Requirements Engineerings erläutert. Ein Überblick hierzu wurde bereits in Abschnitt 7.1.3 gegeben. Bei der Entwicklung der ersten Systemversion müssen sämtliche Entwicklungsprodukte von Grund auf neu erarbeitet werden. Bei jeder Weiterentwicklung hingegen müssen die meisten Entwicklungsprodukte der vorhergehenden Version modifiziert oder erweitert werden. Insbesondere müssen bereits vorhandene Entwicklungsprodukte in jeder Weiterentwicklung berücksichtigt werden.

In den Abschnitten 7.5.1, 7.5.2 und 7.5.3 folgt eine detaillierte Darstellung der Aktivitäten drei zyklischen Phasen der Definition der Marktsicht, der Definition der Systemsicht und der Definition der Entwicklungssicht. Zur Beschreibung einzelner Aktivitäten werden die Schemata des entscheidungsorientierten Entwicklungsprozesses aus Abschnitt 3.1.3 verwendet.

7.5.1 Definition der Marktsicht

A 12 Anforderungen filtern

Entscheidung Anforderungen filtern	
R	**Technisches Marketing**, Sammlungskoordinator, Vertriebsbeauftragter, teilweise Vertreter der Entwicklung
O	Sämtliche gesammelten Anforderungen (Filterung=„keine“) (siehe Abschnitt 4.2.1)
K	Mögliche Kriterien sind Marktrelevanz, Bezug zu Kernkompetenzen und die generellen Realisierbarkeit hinsichtlich der aktuell verfügbaren Softwaretechnologie.
E	Sämtliche Anforderungen sind mit dem Attribut Filterung= „ja“ oder Filterung=„nein“ belegt. Nur mit „ja“ gefilterte Anforderungen sind für die gesamte weitere Planung relevant.
A	Einfache Checklisten, wonach Anforderungen beispielsweise nach jedem Kriterium mit hoch/mittel/niedrig bewertet werden. Diese orientieren sich an der Marktstrategie (vgl. Abschnitt 4.3)
R=Rollen, O=Entscheidungsobjekte, K=Bewertungskriterium, E=Entscheidungsergebnis und A=Anleitungen/Vorlagen	

Entscheidungsproblem: Der Aufwand des gesamten Entwicklungsprozesses ist durch die Anzahl der zu verarbeitenden Anforderungen wesentlich geprägt. Daher ist es notwendig, möglichst früh in der Entwicklung zunächst grob Anforderungen auszufiltern, die nicht relevant sind. Ausgefilterte Anforderungen sollen zwar aus Verfolgbarkeitsgründen weiter erfasst bleiben, werden jedoch zunächst nicht mehr weiter betrachtet und fließen nicht in die weitere Entwicklung ein.

Zielfunktion: Ziel der Filterung ist, momentan unrealistische Anforderungen zu identifizieren und auszufiltern. Unrealistisch sind einerseits Anforderungen mit einem absolut geringen Marktnutzen. Ein Beispiel hierfür wäre ein Auskunftssystem zur Identifikation der nächsten Pizzeria mit einem UMTS-Handy in einem Flugzeug. Andererseits sollen hier Anforderungen mit einem überdimensionalen oder gar nicht zu bewältigenden Entwicklungsaufwand gestrichen werden. Es muss ein realistisches Kosten-/Nutzenverhältnis für eine Anforderung gegeben sein.

Vorgehensweise: Jede Anforderung wird grob nach den Kriterien anhand einer Checkliste beurteilt. Aus der Beurteilung heraus wird entschieden, ob eine Anforderung gefiltert werden soll oder nicht.

Beispielanwendung *Pisa*

Die grobe Beurteilung der Anforderungen für Pisa ergab, dass alle Anforderungen (vgl. Anhang D – Erfasste Anforderungen*) außer A16, A22 und A23 bei der Filterung mit „ja“ bewertet werden. A16 entspricht nicht den Kernkompetenzen des Produktsystems. Gegebenenfalls könnte in Zukunft eine Schnittstelle zur Abfrage entsprechender Informationen realisiert werden. A22 widerspricht dem Angebot der Post, das explizit das Versenden von Verkaufswaren ausschließt, liefert also keinen Kundennutzen. A23 ist nur in Kombination mit A22 sinnvoll, wird daher auch ausgefiltert.*

A 13 Aspekte und Variationstypen definieren

Entscheidung Aspekte und Variationstypen definieren	
R	**Technisches Marketing**, Vertriebsbeauftragter, Sammlungskoordinator, gegebenenfalls Vertreter der Entwicklung
O	Anforderungen (Filterung=„ja“) (siehe Abschnitt 4.2.1)
K	Kriterienauswahl ist unmittelbar von der konkreten Anwendung abhängig. Sinnvolle Kriterien für eine Zerlegung in Aspekte können sein: - Verbesserung der Kommunikation der Entwicklung zum Marketing oder zu verschiedenen Quellen. Möglicherweise orientiert an einer funktionalen Strukturierung der Anwendung **Beispiel:** *Aufteilung der Anforderungen einer Finanzsoftware in Aspekte Buchung, Bilanzierung, Grafiken und Statistiken* - Unterstützung einer effizienten Entwicklung durch eine Zerlegung nach Kompetenzen, nötigen Entwicklungsressourcen, Strukturellen Zerlegungen (vgl. beispielsweise Architekturpatterns, wie MVC oder Pipes- and Filters [BMR+96]), Wiederverwendung, Plattformzerlegung. Die Strukturierungen der Aspekte beeinflusst die physikalische Strukturierung des Systems (oder Softwarearchitektur), da für das Design eine Vermeidung struktureller Brüche angestrebt wird. Die Softwarearchitektur enthält hauptsächlich die Aufteilung eines Softwaresystems in Module (vgl. [Par72]) und dadurch notwendige Kommunikationsbeziehungen. Isolierung von Variationstypen, um Widersprüche und Gemeinsamkeiten zwischen einzelnen Variationen identifizieren zu können.

E	Marktsicht ist festgelegt (Aspekte und Variationstypen definiert, Anforder ungen zu Aspekten und Variationen aggregiert und zugehörige Assoziationen festgelegt; vgl. Abschnitte 4.2.1 und 4.4.2), Dauerhaftigkeit einer Anforderung ist festg elegt (vgl. Abschnitt 4.4.4).
A	- Marktsegmente (grob, relativ stabil): Unterschiedliche Marktsegmente entha lten unterschiedliche Anforderungen und sind meist Urheber für Variationstypen (siehe Abschnitt 4.3). - Systemstandards (langfri stig stabil) (siehe Abschnitt 4.3)
R=Rollen, O=Entscheidungsobjekte, K=Bewertungskriterium, E=Entscheidungsergebnis und A=Anleitungen/Vorlagen	

Entscheidungsproblem: Gesammelte Anforderungen müssen in Aspekte und Variat ionstypen strukturiert werden. Aspekte dienen der allgemeinen Strukturierung von Anfo rderungen, wie bereits in Abschnitt 4.2.2 erläutert wurde. Durch Aspekte können gleichzeitig verschiedene zueinander or thogonale Strukturierungen def iniert werden. So ist beispielsweise parallel Strukturierungen nach technischen Gesicht spunkten (vgl. obige Architekturpatterns) und nach kundenorientierten Gesichtspunkten (z.B. nach bestim mten anwendungsgetriebenen Funktions gruppen) möglich. Die Strukturierung von Variat ionstypen und Variationen (vgl. Abschnitt 4.4.2) wird zur Darstellung ähnlicher Anforderungen unterschiedlicher Kundengruppen (vgl. Abschnitt 4.3) verwendet.

Neben der reinen Festlegung der Aspekte und Variationstypen müssen die im Modell der Entwicklungsprodukte erläuterten Assoziationen zwischen Anforderungen, Aspekten und Variationstypen bestimmt werden. Gegebenenfalls müssen vorhand ene Anforderungen adäquat abstrahiert werden.

Zielfunktion: Die Wahl der Zielfunktion sollte abhängig von der strategischen Ausric htung im konkreten Einzelfall sein. Allgemein muss die Anzahl der Aspekte übe rschaubar gehalten werden, um nicht das Gegenteil der gewollten Str ukturierung zu erreichen. Bei der Strukturierung in Aspekte sollte ebenfalls immer der dafür notwendige Verwaltung saufwand berücksichtigt werden.

Vorgehensweise: Die Identifikation von Aspekten und Variationstypen wird schwe rpunktmäßig in der ersten Iteration der Versionsentwicklung durchgeführt. In späteren Versionen werden zur Vermeidung von aufwändigen Umstrukturierungen diese weites tgehend stabil gehalten. Es folgen Überlegungen, die bei der Identifikation von Aspekten und Variationstypen wesentlich sind.

Neben der Strukturierung muss bei der Identifikation von Aspekten die Unterstü tzung der Verfolgbarkeit von Änderungen von Anforderungen berücksichtigt werden. Dazu dient der korreliert-Assoziationstyp zwischen Anforderungen, Aspekten und Vari ationstypen. Eine möglichst hohe Verfolgbarkeit wird dann erreicht, wenn einzelne Beziehu ngen direkt zwischen Anforderungen spezifiziert werden. Diese sehr feingranulare Spezifik ation ist in vielen praktischen Fällen zu au fwendig.

Alternativ können Beziehungen zwisc hen Aspekten und Variationstypen angegeben we rden. Im Falle von Anforderungsänderungen wird hiermit der Suchraum von der Änd erung tangierter Anforderungen zumindest auf bestimmte Aspekte eingeschränkt. Je weniger Anforderungen einem Aspekt zugeordnet sind, desto leichter ist die Identifik ation von einer Änderung betroffener Anforderungen. Mit der Zunahme ve rschiedener

Aspekte reduziert sich allerdings auch die Übersichtlichkeit einer Spezifikation. Darüber hinaus muss umso mehr Aufwand in die Konsistenzerhaltung derartiger Spezifikationen gesteckt werden.

Wie fein zwischen verschiedenen Aspekten unterschieden wird, hängt auch von der Notwendigkeit einer schnellen Verfolgbarkeit ab. Es ist beispielsweise sinnvoll, bei Anforderungen mit hoher Änderungswahrscheinlichkeit möglichst fein zu unterscheiden. Damit können Auswirkungen von Änderungen schnell identifiziert werden. Umgekehrt können Verwaltungsaufwände reduziert werden, wenn wenige unterschiedliche Aspekte dort definiert sind, wo wenige künftige Änderungen zu erwarten sind (zur Beurteilung der Änderungswahrscheinlichkeit vgl. z. B. Abschnitt 6.7.4).

Die Definition von Aspekten hat auch einen wesentlichen Einfluss auf Entwicklungsaufwände. Dies ist dann der Fall, wenn sich ein Variationstyp auf einen Aspekt bezieht.

Beispiel:

> *Der Variationstyp Landessprache korreliert mit der Ausgabe von Texten (vgl. auch die Beispiele in Abschnitt 4.4.2). Zur Spezifikation dieser Korrelation kann ein Aspekt Textausgabe definiert werden, der mit der Landessprache in korreliert-Assoziation steht.* Textausgaben *lassen sich jedoch noch feiner untergliedern, wie beispielsweise im Abschnitt 4.4.2 angedeutet worden ist. Es kann also zwischen den Aspekten* normalen Meldungen *und* Systemabsturzmeldungen *unterschieden werden, die beide zu der Landessprache korrelieren werden.*

Für die beiden Aspekte *normale Meldungen* und *Systemabsturzmeldungen* aus dem Beispiel können unterschiedliche *gewünscht-Assoziationen* festgelegt werden. Für Systemabstürze wird beispielsweise lediglich die Sprache Englisch gewünscht. Für andere Systemmeldungen hingegen wird keinerlei *gewünscht-Assoziation* spezifiziert. Dadurch können Entwicklungsaufwände für die sprachenspezifische Entwicklung von Systemabsturzmeldungen eingespart werden.

Beispielanwendung *Pisa*

> *In der Beispielanwendung wurde mehrere Aspekte und Variationstypen identifiziert (siehe Anhang D –* Identifizierte Aspekte, Variationstypen und Variationen*). Wesentliche Aspekte und Variationstypen beziehen sich hierbei auf zu leistende Anwenderfunktionalität des Produktsystems. Weitere Aspekte umfassen die Betriebssystemkompatibilität, Qualität und Bedienung. In Abbildung 38 des Anhangs wird ein Ausschnitt der externen Spezifikation (vgl. Abschnitt 5.2.1) dargestellt. Wie daraus ersichtlich ist, sind insbesondere die Optimierung und die Sortierung mit mehreren Variationstypen korreliert. Hieraus ergibt sich eine hohe Anzahl von abgeleiteten Variationen, die im Falle ihrer Realisierung große Aufwände mit sich bringen können. Alle Anforderungen mit Ausnahme der einfachen Bedienerführung (A2) wurden als verbrauchbar identifiziert (vgl. Abschnitt 4.2.1). A2 wird als permanent festgelegt, da die einfache Bedienerführung auch für neue Funktionen in künftigen Versionen von* Pisa *berücksichtigt werden soll.*

A 14 Verkaufsorientierte Bewertung für Anforderungen

Vorarbeit Verkaufsorientierte Bewertung für Anforderungen	
S	Vorarbeit für Entscheidungen über die Realisierung (A 19) und die Versionsplanung von Anforderungen (A 21)
R	**Technisches Marketing**, Systemstrategen, Sammlungskoordinatoren
O	Anforderungen (Filterung= „ja“)
E	Bewertung des erwarteten Verkaufserfolgs und der Änderungswahrscheinlichkeit (siehe Abschnitt 6.1) von Anforderungen. Kriterien zur Bewertung des Verkaufserfolgs wurden in Abschnitt 6.5 gemeinsam mit Kriterien zur Bewertung der Entwicklungserfordernisse vorgestellt (vgl. anschließenden Unterabschnitt 7.5.2).
A	Hier können Methoden der Bewertung verwendet werden, die Kapitel 6 ausführlich diskutiert wurden. Die Bewertung von Änderungswahrscheinlichkeiten kann mit Hilfe aktueller Marktanalysen oder einer systematischen Sammlung von Erfahrungen unterstützt werden. Ein Beispiel hierzu findet sich in Abschnitt 6.7.4.
S=Entscheidungssituation, R=Rollen, O=Vorarbeitsobjekte, E=Vorarbeitsergebnis und A=Anleitungen/Vorlagen	

Motivation: Zur klaren Trennung von Markt- und Entwicklungserfordernissen müssen für diese beiden Sichten isolierte Bewertungen von Anforderungen erfolgen. In der verkaufsorientierten Bewertung wird analysiert, wie stark die Realisierung einer Anforderung zum Verkaufserfolg einer Standardsoftware beiträgt.

Daneben sind Anforderungen an Standardsoftware häufig im Laufe der Zeit Änderungen unterworfen. Diese lässt sich teilweise aufgrund ihrer Zugehörigkeit zu bestimmten Marktsegmenten vorhersagen. Beispielsweise ist im Telekommunikationsmarkt der gesamte Digitale Sektor einer wesentlich höheren Dynamik unterworfen als der analoge Sektor. Durch eine adäquate vorausschauende Definition einer Systemarchitektur des Produktsystems lassen sich künftige Änderungsaufwände erheblich reduzieren. In der verkaufsorientierten Bewertung erfolgt daher zusätzlich zur Erfolgsabschätzung eine Abschätzung der Änderungswahrscheinlichkeit einzelner Anforderungen.

Vorgehen: In Abschnitt 6.6 wurde eine allgemeine Vorgehensweise zur Bewertung vorgeschlagen. Die verkaufsorientierte Bewertung von Anforderungen erfüllt Teile der dort beschriebenen ersten zwei Schritte Kriterienbewertung (vgl. Abschnitt 6.6.1) und Portfolio-Analyse (vgl. Abschnitt 6.6.2). Kriterienbewertungen werden für die Kriterien *Dringlichkeit* und *Marktbedeutung* (Details hierzu siehe in Abschnitt 6.5) festgelegt. Für die Portfolio-Analyse wird das Teilportfolio der verkaufsorientierten Kriterien aufgestellt. Die selbe Bewertung kann sowohl für die Aktivität *Auswahl zu realisierender Anforderungen und Variationen* (in Abschnitt 7.5.2) als auch für die Versionsplanung von Anforderungen (in Abschnitt 7.5.3) verwendet werden. Gegebenenfalls muss für die Versionsplanung eine noch feinere Bewertung erfolgen, die eine bessere Unterscheidbarkeit unterschiedlicher Anforderungen ermöglicht.

Beispielanwendung *Pisa*

In Anhang D – Einzelbewertungen ausgewählter Anforderungen *ist ein Ausschnitt der Einzelbewertungen von Anforderungen des Produktsystems* Pisa *dargestellt. Die Einzelbewertungen wurden anhand der direkten Skalierung (vgl. Abschnitt 6.3.2) mit Skalenwerten von 1 bis 5 vorgenommen. Abbildung 33 zeigt das Teilportfolio der verkaufsorientierten Bewertung. Viele Anforderungen sind jeweils*

gleich bewertet. Dies ist insbesondere bei der ersten Version des Produktsystems nicht weiter verwunderlich, da mehrere Anforderungen zum Anwendungskern gehören, ohne den eine Auslieferung eines Softwareprodukts nicht sinnvoll wäre.

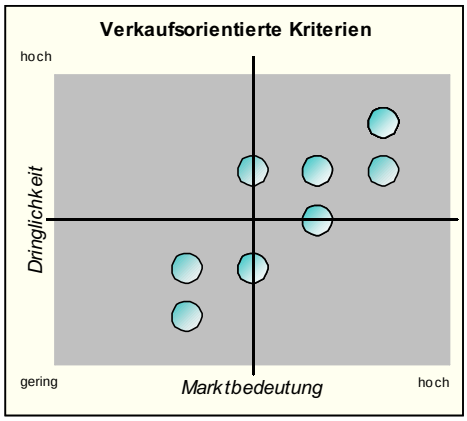


Abbildung 33 – Verkaufsorientierte Bewertung von *Pisa*

7.5.2 Definition der Systemsicht

In der Marktsicht wird von sämtlichen technischen Einflüssen abstrahiert, um eine klare Struktur der vom Markt geforderten Anforderungen und Variationen zu erhalten. Die Definition der Systemsicht dient der Vorbereitung der technischen Umsetzung der vom Markt geforderten Anforderungen.

Sie schafft Rahmenbedingungen für eine effiziente und flexible Entwicklung eines Produktsystems. Dazu werden eine Systemarchitektur mit weitgehend unabhängigen Systemmodulen definiert, Ressourcen und Entwicklungszeiten entsprechend der Markt bedeutung auf diese verteilt und eine Systemstrategie erstellt. Darüber hinaus werden Anforderungen Systemmodulen zugeordnet und letztendlich zu realisierende Anforderungen ausgewählt.

Um Visionen für künftige Versionen entwickeln zu können, wird diese Auswahl unabhängig von tatsächlichen Zeit- und Kostenrestriktionen einer Versionsplanung gemacht. Eine adäquate Verteilung von Anforderungen auf Versionen erfolgt im Rahmen der Definition der Entwicklungssicht.

Sämtliche Aktivitäten der Definition der Systemsicht werden von Systemstrategen hauptverantwortlich durchgeführt.

A 15 Definition der Systemarchitektur

Entscheidung Definition der Systemarchitektur	
R	**Systemstratege,** Systemmodulplaner, Strategisches Marketing
O	Aspekte und Variationstypen (definiert) (siehe Abschnitte 4.2.2, 4.4.2)
K	- **Verkaufsziele:** Für das Marketing stehen Verkaufsziele im Vordergrund. Eine geschickte Aufteilung der Anforderungen in unterschiedliche Produkte kann gegebenenfalls den Verkaufserfolg des Gesamtsystems erhöhen. Beispielsweise können in der Summe niedrigpreisige produktexterne Variationen den Gesamtu m-

	satz eines teureren, jedoch alle Variationen umfassenden, Softwareprodukts übe r-steigen. - **Entwicklungsziele:** Demgegenüber steht bei kom plexer Standardsoftware das Ziel, Systemmodule allgemein unabhängig voneinander und insbesond ere Kerne von Variationstypen isoliert entwickeln zu können. Dies dient den Zielen einer hohen Flexibilität und einer weitest gehenden Wiederverwendung. Darüber hi naus spielen allgemeine Architekturziele, wie unter anderem unabhängige Entwic klung, Übersichtlichkeit, Effizienz und Austauschbarkeit (vgl. zum Beispiel [SG96, JGJ97, BMR+96]) ebenfalls eine entscheide nde Rolle
E	Systemarchitektur (Systemmodule aggregie rt, Leitlinien einzelner Systemmodule definiert) (vgl. Abschnitt 4.4.3).
A	- **Standardarchitekturen und Architekturstile:** geben unterschiedliche Modular i-sierungen vor und zeigen auf, welche Vor - und Nachteile dies e beinhalten. Modularisierungen werden anhand von verschiedenen Aspekten vorgenommen (vgl. [SG96, JGJ97, BMR+96, Jac+92]). - **Marktsegmentierung** (grob, relativ stabil) (siehe Abschnitt 4.3): Aus Marketing-sicht kann es sinnvoll sein, hierfür sogenannte produktexterne Variationen zu bi l-den. Bei produktexternen Variationen wird jede Variation eines Variationst ypen durch ein separates Systemprodukt repräsentiert. Der gemeins ame Kern wird dann durch ein Basismodul reali siert. Ein Kunde erwirbt jeweils nur die Softwarepr o-dukte, die für seine individuellen Bedürfnisse erforderlich sind und konf iguriert diese einmalig durch seine Installation. Im Gegensatz dazu werden bei der pr o-duktinternen Variationsbildung hingegen werde n keine separaten Softwareprodu k-te für einzelne Var iationen eines Variationstypen gebildet. Hier erfolgt die Aktivierung von Variationen (vgl. Abschnitt 4.4.2) innerhalb eines Syste mmoduls und kann gegebenenfalls mehrmals verän dert werden. - **Marktstrategie** (langfristig stabil) (siehe Abschnitt 4.3): Anhand der Trends der Gewichtung künftiger Marktsegmente erkannt we rden können. - **Systemstandards** (langfristig stabil) (siehe Abschnitt 4.3): Diese können Restrik-tionen hinsichtlich der Aufteilung in Systemmod ule beschreiben.
R=Rollen, O=Entscheidungsobjekte, K=Bewertungskriterium, E=Entscheidungsergebnis und A=Anleitungen/Vorlagen	

Entscheidungsproblem: Mit der Definition der Systemarchitektur wird die Aufteilung des Produktsystems in Systemmodule festgelegt. In der Regel bleibt die Systemarchite k-tur langfristig über mehrere Versionen hinweg stabil. Daher wird sie hauptsäc hlich beim ersten Versionszyklus definiert und gegebenenfalls in höheren Versionszyklen durch kleine Änderungen aktualisiert.

Zielfunktion: Zwischen den Zielen der Kundenorientierung (vergleiche Abschnitt 1.1.1) und der entwicklungsoptimalen Aufteilung m uss ein Kompromiss gefunden werden. Für die Unterscheidung zwischen externen und internen Variationen lassen sich allg emeine Aussagen treffen. Für externe Varianten spricht die bessere Nachvollziehbarkeit für Kunden und die geringere Komplexität einzelner Softwareprodukte. Dafür ist jedoch die Vernetzung der Softwareprodukte groß. Bei geschickter Partitioni erung ist diese bei der internen Variantenbildung erheblich reduziert. Umgekehrt erhöht sich dafür aber die produktinterne Komplexität der einze lnen Softwaremodule.

Vorgehensweise: Verschiedene Aspekte und Variationstypen geben bereits verschiedene Möglichkeiten zur Strukturierung eines Produktsystems vor. Hieraus muss nun unter Abwägung der unterschiedlichen Zielvorstellungen eine Systemarchitektur gefunden werden. Anhand der Architekturstile können verschiedene technische Möglichkeiten diskutiert werden. Die Marktsegmentierung und Marktstrategie hingegen geben Hinweise auf mögliche Verkaufsvorteile. Die beste Auswahl hiervon kann iterativ optimiert werden.

Hinweise zur Beherrschung der Komplexität im Zusammenhang mit Produktvariationen existiert in der Betriebswirtschaftlichen Literatur (siehe zum Beispiel [Wil97]), die auf die Softwareentwicklung adaptiert werden kann.

Beispielanwendung *Pisa*

Eine Analyse der ursprünglichen Ziele der Marktstrategie und der vorhandenen Anforderungen hat ergeben, dass das Produktsystem Pisa aus vier Softwareprodukten und einem Basismodul bestehen soll. Nachfolgende Tabelle gibt einen Überblick über Modultypen, Bezeichnungen und Leitlinien.

Übersicht über Systemmodule des Produktsystems *Pisa*		
Modultyp	Bezeichnung	Leitlinie
SWProdukt	*Pisa-Kern*	realisiert minimale Anforderungen, die für alle Segmente relevant sind, um Infopost versenden zu können.
SWProdukt	*Pisa-Adress*	reine Adressverwaltung mit Kopplung zu *Pisa-Kern*.
SWProdukt	*Pisa-Groß*	realisiert Sonderanforderungen für Großkunden.
SWProdukt	*Pisa-Direkt*	realisiert Kopplung zu anderer Direktmarketing-Software.
Basismodul	*Pisa-ABasis*	realisiert gemeinsame Adressfunktionalitäten von *Pisa-Kern* und

Das Basismodul Pisa-ABasis garantiert die Wiederverwendung von gemeinsamen Anforderungen aus Pisa-Kern und Pisa-Adress.

A 16 Planung von Ressourcen und Entwicklungszeiten einzelner Systemmodule

Entscheidung Planung von Ressourcen und Entwicklungszeiten	
R	**Systemstratege**, Systemmodulplaner, Technisches Marketing
O	Systemarchitektur (definiert) (siehe Abschnitt 4.4.3)
K	Entscheidend für die Verteilung der Ressourcen ist der zu erwartende Erfolg einzelner Systemmodule (vgl. hierzu Target Costing [SS97b]). Dieser kann anhand der Leitlinien der Systemmodule (vgl. Abschnitt 4.4.3) abgewogen werden. Kriterien hierzu sind die Marktbedeutung, die Dringlichkeit und die technische Wertigkeit, die einem Systemmodul zugesprochen werden können. Die Marktbedeutung und die Dringlichkeit einzelner Systemmodule kann auch abhängig von der Marktstrategie sein.
E	In dieser Aktivität werden der Planungshorizont der Entwicklung (vgl. Abschnitt 4.4.3), Zykluszeiten der einzelnen Systemmodul-Versionen und die Verteilung der Ressourcen auf die einzelnen Systemmodule festgelegt.
A	- Gegebenenfalls können Instrumente der Zeit- und Ressourcenplanung verwendet werden (vgl. [WK00]). Zur Gewichtung einzelner Systemmodule kann die Portfolio-Methode aus Abschnitt 6.6 verwendet werden.

	- Marktstrategie (langfristig stabil) (siehe Abschnitt 4.3)

R=Rollen, O=Entscheidungsobjekte, K=Bewertungskriterium, E=Entscheidungsergebnis und A=Anleitungen/Vorlagen

Entscheidungsproblem: Für die Entwicklung eines Produktsystems stehen in einem Unternehmen nur begrenzte Ressourcen zur Verfügung. Darüber hinaus sind zur Entwicklung eines jeden Systemmoduls unterschiedliche Kompetenzen erforderlich. Schließlich verlangt auch noch der Marktdruck relativ starre Versionszyklen. Unter diesen Voraussetzungen müssen vorhandene Ressourcen auf die Systemmodule adäquat verteilt werden. Durch die Ressourcenverteilung wird eine Planungsgrundlage für die Verteilung von Anforderungen auf Systemmodule und Versionen geschaffen.

Zielfunktion: Systemmodule mit hoher Marktbedeutung oder hoher technischer Wertigkeit sollen verhältnismäßig mit mehr Ressourcen ausgestattet werden. Die zeitliche Ressourcenverteilung muss sich an der Dringlichkeit einzelner Systemmodule orientieren. Wegen der weitgehend unabhängigen Systemmodule kann eine parallele oder zumindest zeitlich überlappende Entwicklung verschiedener Systemmodule erfolgen.

Restriktionen: Die Hauptrestriktion ist die Ressourcenplanung, die beschreibt, welche Ressourcen mit welchen Kompetenzen künftig insgesamt für das Produktsystem zur Verfügung stehen. Dies wird von der Entwicklungsleitung dem Systemstrategen übermittelt. Sie berücksichtigt auch langfristige Entwicklungen, wie beispielsweise Expansionen aufgrund erwarteter Umsatzsteigerungen.

Vorgehensweise: Anhand der Leitlinien einzelner Systemmodule werden zunächst qualitativ benötigte Kompetenzen identifiziert. Die einzelnen Systemmodule werden dann gegeneinander gewichtet, so dass schließlich vorhandene Ressourcen auf die Systemmodule verteilt werden können. Die Verteilung erfolgt für jede Kompetenzart gesondert im Verhältnis zu hierfür vorhandenen Ressourcen.

Beispielanwendung *Pisa*

Die Auslieferung des Produktsystems soll in einem halben Jahr erfolgen. Erstes Softwareprodukt soll hierbei Pisa-Kern *sein. Daher bekommt es den frühest möglichen Anfangszeitpunkt. Kernkompetenzen hinsichtlich Ressourcen haben sich in der ersten Version noch nicht gebildet, so dass lediglich gleichwertige Ressourcen verteilt werden müssen.* Pisa-Kern *und* Pisa-ABasis *bekommen zusammen zunächst alle notwendigen Ressourcen zugesprochen, da diese Module Grundlage für die strategischen Kernkompetenzen sind. Verfügbare Ressourcen sollen in Aufwandseinheiten gemessen werden. Diese werden später zur groben Abschätzung der Entwicklungsaufwände einzelner Anforderungen verwendet. Für die erste Version von* Pisa-Kern *sollen 30 Aufwandseinheiten zur Verfügung stehen.*

A 17 Zuordnung von Anforderungen zu Systemmodulen

Entscheidung Zuordnung von Anforderungen zu Systemmodulen	
R	**Systemstrategen**, Technisches Marketing, Systemmodulplaner beteiligt.
O	- Aspekte, Variationstypen und Anforderungen (gefiltert, zugehörige Assoziationen definiert) (siehe Abschnitte 4.2.1, 4.2.2, 4.4.2) - Systemarchitektur (definiert, Leitlinien der Systemmodule festgelegt) (siehe Abschnitt 4.4.3)

K	Wesentliche Kriterien für die Verteilung von Anforderungen sind die Leitlinien und zugeteilte Kernkompetenzen der einzelnen Systemmodule. Darüber hinaus ist eine geringe Vernetzung von Systemmodulen aufgrund unterschiedlicher Assoziationen anzustreben.
E	Modulschnittstellen zwischen Systemmodulen sind festgelegt. Es ist definiert, welche Systemmodule Modulschnittstellen fordern und welche diese erfüllen sollen. Anforderungen sind auf Systemmodule und Modulschnittstellen verteilt (vgl. Abschnitt 4.4.3). Durch diese Verteilung ist noch nicht entschieden, dass die Systemmodule tatsächlich diese Anforderungen realisieren werden. Diese Entscheidung wird in der Aktivität *Auswahl zu realisierender Anforderungen und Variationen* gefällt.
A	
R=Rollen, O=Entscheidungsobjekte, K=Bewertungskriterium, E=Entscheidungsergebnis und A=Anleitungen/Vorlagen	

Entscheidungsproblem: In der Marktsicht sind Anforderungen noch nicht Systemmodulen zugeordnet. Mit der Zuordnung von Anforderungen zu Systemmodulen wird festgelegt, welche Systemmodule welche Anforderungen realisieren sollen. Dabei kann sich eine Anforderung auf mehrere Systemmodule auswirken oder alternativ in unterschiedlichen Systemmodulen realisiert werden.

Zielfunktion: Eine optimale Zuordnung von Anforderungen orientiert sich an der Nähe der zugeordneten Anforderungen zu den Leitlinien der Systemmodule, einer möglichst geringen Vernetzung unterschiedlicher Systemmodule und einem hohen Wiederverwendungsgrad von Kernen von Anforderungen.

Vorgehensweise: Die Zuordnung von Anforderungen zu Systemmodulen erfolgt in zwei Stufen. In der ersten Stufe wird die Verteilung der Anforderungen auf einzelne Softwareprodukte vorgenommen. Damit wird die Kundensicht auf das entstehende Produktsystem dokumentiert. Ein Kunde kann seine individuelle Konfiguration durch den Kauf unterschiedlicher Softwareprodukte des Produktsystems zusammenstellen.

In der zweiten Stufe werden die Anforderungen so verteilt, wie sie anschließend auch für die Entwicklung geplant werden sollen. Dazu werden zwischen den Systemmodulen Modulschnittstellen definiert, mit deren Hilfe Systemmodule Anforderungen an andere Systemmodule delegieren können. Anschließend erfolgt eine Zuordnung von Anforderungen zu den Modulschnittstellen. Damit wird eindeutig dokumentiert, welche Systemmodule die Erfüllung delegierter Anforderungen benötigen und welche Systemmodule delegierte Anforderungen erfüllen sollen.

Beispielanwendung *Pisa*

In Anhang D – Zuordnung der Anforderungen zu Systemmodulen *sind beispielhafte Zuordnungen von Anforderungen zu den Softwareprodukten* Pisa-Kern *und* Pisa-Adress *angegeben. In einem ersten Schritt wurden hierzu die Anforderungen so verteilt, als ob eine isolierte Entwicklung der Softwareprodukte erfolgen würde. Hierbei wurden vor allem die Leitlinien beachtet. Daher bekommt* Pisa-Kern *alle Anforderungen zugeschrieben, die zur Erfüllung der Basisfunktionalität zur Versendung von Infopost dienen. Hierzu ist auch eine Reihe von Adressverarbeitungsfunktionen nötig.* Pisa-Adress *hingegen erhält lediglich Adressverarbeitungs-*

funktionen. Hierbei sind aber auch viele spezifische Anforderungen enthalten, die über die eigentlichen Kernanforderungen der Infopostversendung hinausgehen. Umgekehrt werden andere Anforderungen ausgespart, die spezifisch für die Adressverarbeitung für Infopost sind. Erst im zweiten Schritt wurden gemeinsame Anforderungen der beiden Module auf aggregierte Anforderungen und Modulschnittstellen verteilt. Abbildung 34 zeigt die resultierende Systemarchitektur zusammen mit den Modulschnittstellen.

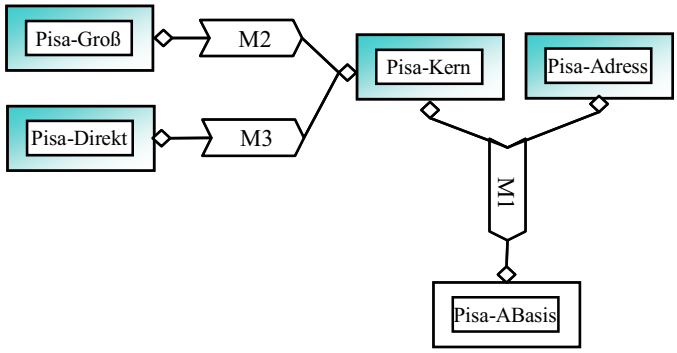


Abbildung 34 – Externe Sicht der Systemarchitektur von Pisa

A 18 Entwicklungsorientierte Bewertung für Anforderungen

Vorarbeit Entwicklungsorientierte Bewertung für Anforderungen	
S	Vorarbeit für Entscheidungen der generellen Realisierung und der Versionsplanung von Anforderungen
R	**Systemstratege**, Systemmodulplaner und Entwickler. Entwickler sind in erster Linie für die Beurteilung von Realisierungsaufwänden nötig.
O	Anforderungen (gefiltert=ja)
E	Ergebnis ist eine entwicklungsorientierte Bewertung (siehe Abschnitt 6.1) der betrachteten Anforderungen. Kriterien hierzu wurden in Abschnitt 6.5 gemeinsam mit Kriterien zur Bewertung des Verkaufserfolgs vorgestellt (vgl. vorhergehenden Abschnitt 7.5.1).
A	Wie zur entwicklungsorientierten Bewertung können auch hier die in Kapitel 6 vorgestellten Methoden verwendet werden. Für Aufwandsschätzungen können Methoden der Kostenschätzung verwendet werden (vgl. z.B. [WK00]).
S=Entscheidungssituation, R=Rollen, O=Vorarbeitsobjekt, E=Vorarbeitsergebnis und A=Anleitungen/Vorlagen	

Motivation: Wie die verkaufsorientierte Bewertung für Anforderungen wird die entwicklungsorientierte Bewertung in einer eigenen Aktivität durchgeführt. Ziele der entwicklungsorientierten Bewertung sind einerseits die Identifikation des technischen Wertes einer Anforderung und andererseits die Abschätzung von Aufwand und Effizienz.

Vorgehen: Als Gegenstück zur verkaufsorientierten Bewertung für Anforderungen aus Abschnitt 7.5.1 werden hinsichtlich der entwicklungsorientierten Bewertung ebenfalls Teile der ersten beiden Schritte der Portfolio-Methode angewendet. Bewertungen werden hier hinsichtlich der Kriterien *Technische Wertigkeit* und *Aufwand und Effizienz* (vgl. Abschnitt 6.5) erstellt. Auch diese Bewertung kann wie schon die verkaufsorientierte Bewertung sowohl für die Auswahl von Anforderungen als auch zur Versionsplanung

verwendet werden. Ein Teil der entwicklungsorientierten Bewertung ist die grobe Schätzung von Entwicklungsaufwänden. Hierbei müssen hauptsächlich verschiedene Assoziationen adäquat berücksichtigt werden. Beispielsweise verursacht die Realisierung einer Anforderung A Aufwände für die Realisierung einer korrelierten Anforderung B. Da diese Aufwände nur auftreten, wenn Anforderung A realisiert wird, werden diese dieser Anforderung zugerechnet. Für permanente Anforderungen erfolgt keine entwicklungsorientierte Bewertung. Für sie wird abhängig von der aktuellen Marktbewertung festgelegt, wie viel Aufwand sie verursachen dürfen.

Beispielanwendung *Pisa*

In Anhang D – Einzelbewertungen ausgewählter Anforderungen *sind analog zu den verkaufsorientierten Bewertungen auch Einzelbewertungen entwicklungsorientierter Kriterien angegeben (ebenfalls direkte Skalenbewertung mit 1 bis 5 bewertet). Viele Anforderungen sind hier mit mittlerer technischer Wertigkeit bewertet. Dies könnte ein Indiz dafür sein, dass beim Herausarbeiten von Anforderungen zu wenig Augenmerk auf Anforderungen mit dem Ziel der technischen Unterstützung der Softwareentwicklung gelegt wurde. Konsequenterweise könnten Aktivitäten zum Herausarbeiten aus Quellen und Kanälen forciert werden, die eine Identifikation technisch wertvoller Anforderungen versprechen. Im Beispiel wird jedoch der Einfachheit halber davon ausgegangen, dass alle Quellen und Kanäle ausreichend genutzt wurden. Abbildung 35 zeigt das zugehörige Teilportfolio. Die Achse Aufwand und Effizienz entspricht in diesem Beispiel einer groben reinen Aufwandsschätzung. Andere in Abschnitt 6.5 vorgeschlagene Kriterien spielen hier also keine Rolle gespielt. Die Werte dieser Achse entsprechen somit den bei der Aktivität* A 16 Planung von Ressourcen und Entwicklungszeiten *erwähnten Aufwandseinheiten.*

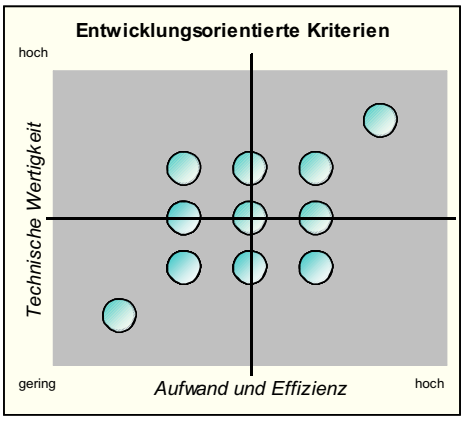


Abbildung 35 – Entwicklungsorientierte Bewertung von *Pisa*

A 19 Auswahl zu realisierender Anforderungen und Variationen

Entscheidung Auswahl zu realisierender Anforderungen und Variationen	
R	**Systemstratege**, Strategisches Marketing, Technische Marketing, Systemmodulplaner und Entwickler
O	- Systemarchitektur (definiert und Ressourcen verteilt) (siehe Abschnitt 4.4.3) - Anforderungen (aggregiert zu Systemmodulen, Assoziation zu verkaufsorientierter und zu entwicklungsorientierter Bewertung, Bewertungen definiert) (siehe A b-

	schnitte 4.2.1, 6.1) - Variationen und Abhängigkeiten (definiert) (siehe Abschnitt 4.4.2)
K	Entscheidend für die Auswahl von Anforderungen sind die in den Vorarbeiten gewonnenen Bewertungen der Anforderungen aus verkaufsorientierter und aus entwicklungsorientierter Sicht.
E	Als Entscheidungsergebnis ist festgehalten, welche Anforderungen tatsächlich realisiert werden sollen. Zusätzlich existiert für jede dieser Anforderungen eine Bewertung (vgl. Abschnitt 6.1)
A	Zur Unterstützung der Verhandlung kann die in Abschnitt 6.6 vorgestellte Portfolio-Methode herangezogen werden.
R=Rollen, O=Entscheidungsobjekte, K=Bewertungskriterium, E=Entscheidungsergebnis und A=Anleitungen/Vorlagen	

Entscheidungsproblem: Durch die grobe Filterung wurden lediglich Anforderungen ausgefiltert, die zum Zeitpunkt der Filterung mit Sicherheit nicht für den Gesamterfolg eines Produktsystems relevant waren. Die Auswahl zu realisierender Anforderungen ist nun ein zweiter Filterungsschritt. Mit den zusätzlich gewonnenen Erkenntnissen aus den vorgelagerten Aktivitäten soll nun endgültig über die Realisierung von Anforderungen entschieden werden. Die Entscheidung soll dabei aus einem systemweiten Blickwinkel erfolgen. Damit wird vermieden, dass übergreifende Anforderungen in verschiedenen Systemmodulen unterschiedlich behandelt werden. Um Visionen für künftige Versionen entwickeln zu können, wird diese Auswahl unabhängig von tatsächlichen Zeit- und Kostenrestriktionen einer Versionsplanung gemacht.

Zielfunktion: Das Entscheidungsargument wird in einer Verhandlung, wie sie in Kapitel 6 vorgestellt wird, ermittelt. Im Mittelpunkt steht hierbei eine Kompromissbildung unterschiedlicher Zielvorstellungen beteiligter Rollen.

Restriktionen: Wesentliche Restriktionen ergeben sich aus verschiedenen Assoziationen zwischen Anforderungen, Aspekten und Variationen (vgl. Abschnitte 4.2.1, 4.2.2 und 4.4.2). Bei Anforderungen, die andere *benötigen,* sind entsprechend assoziierte Anforderungen auszuwählen. Für *erwartete* Variationen muss mindestens eine davon ausgewählt werden. Wird eine Anforderung realisiert die mit einer anderen *korreliert*, so ist diese Korrelation adäquat bei der anderen Anforderung zu berücksichtigen usw.

Vorgehensweise: Bei benötigt-Assoziationen zwischen Anforderungen müssen voneinander abhängige Anforderungen gleichzeitig ausgewählt werden. Bei Variationen ist zu berücksichtigen, dass sich Entwicklungsaufwände zwischen der ersten Variation eines Variationstypen und weiteren Variationen unterscheiden. Während für die erste Variation zusätzlich der gemeinsame Kern entwickelt werden muss, schlagen bei jeder weiteren Variation lediglich die Realisierungsaufwände der Spezifika zu Buche. Darüber hinaus hat die Korrelation zwischen Variationen erhebliche Auswirkungen auf zu realisierende Anforderungen, wie bereits in Abschnitt 4.4.2 erläutert worden ist. Für eine permanente Anforderung wird beim ersten Auftreten anhand ihrer Marktbedeutung über ihre Realisierung entschieden. Danach gilt sie automatisch in jeder Version als geplant.

Zur Unterstützung der Auswahl wird aus der verkaufsorientierten und der entwicklungsorientierten Bewertung ein Portfolio entsprechend dem zweiten Schritt der Bewertungsmethode aus Abschnitt 6.6.2 gebildet. Entsprechend dem dritten Bewertungsschritt wird

anhand des Portfolios die Bewertung einzelner Anforderungen verhandelt (vgl. Abschnitt 6.6.3).

Beispielanwendung *Pisa*

In Abbildung 36 ist das aus den Teilportfolios aus Abbildung 33 und Abbildung 35 ermittelte Gesamtportfolio dargestellt. Die Achsenwerte der beiden Dimensionen wurden durch gewichtete Summenbildung ermittelt. Bei den verkaufsorientierten Kriterien wurden Dringlichkeit und Marktbedeutung als gleichwertig einberechnet. Für entwicklungsorientierte Kriterien wurden hingegen die technische Wertigkeit mit dem Faktor 0,5 und Aufwand/Effizienz mit 1,5 einberechnet, da die technische Wertigkeit eine geringe Differenzierbarkeit aufweißt. Das resultierende Portfolio zeigt, dass jede einzelne Anforderung unterschiedlich eingestuft werden konnte. Bei der ersten Aufstellung des Gesamtportfolios war dies nicht der Fall. Für Anforderungen mit gleichem Wert wurde daher ein direkter Vergleich durchgeführt, was zur Umwertung verschiedener Einzelbewertungen führte. Kritische Kandidaten zur Auswahl sind Anforderungen von geringem verkaufsorientierten und gleichzeitig geringem technischen Wert. Je nach Strategie kann man hierzu im Portfolio verschiedene Bereiche einbeziehen. Im Beispiel sollen hierzu die vier Felder, die gleichzeitig unterhalb der Mitte der verkaufsorientierten und links der Mitte der entwicklungsorientierten Kriterien liegen, zählen. Kritische Anforderungen sind demnach A10, A11, A12 und A31.

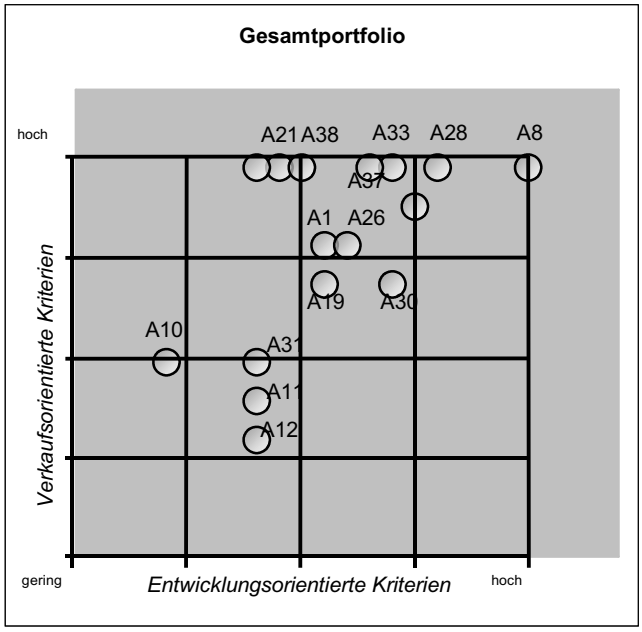


Abbildung 36 – Portfoliobewertung für ausgewählte Anforderungen von *Pisa*

Da A11 und A12 voll in diesem Bereich sind, werden diese für die Standardsoftware nicht berücksichtigt. Für derartige Anforderungen könnte im Einzelfall mit den entsprechenden Kunden über eine spezifische Anpassung in dessen Auftrag verhandelt werden. Der Rest der Anforderungen soll in der weiteren Entwicklung berücksichtigt werden.

A 20 Definition der Systemstrategie

Entscheidung Definition der Systemstrategie	
R	**Systemstratege**, Systemmodulplaner
O	- Systemarchitektur (definiert) (siehe Abschnitt 4.4.3) - Ressourcen (künftige Planung, definiert)
K	Als Kriterien können Tendenzen, die aus der Markt strategie abgeleitet worden sind, verwendet werden. Wichtige Tendenzen sollten in künftigen Entwicklungen im Pr o-duktsystem berücksichtigt we rden
E	Das Entscheidungsergebnis ist eine aktualisierte Systemstrategie, in der zumindest Schwerpunkte der Entwickl ung der einzelnen Versionen innerhalb des Planungsh o-rizonts definiert sind. Langfristige Visionen sind von dieser Aktivität nicht unmitte l-bar betroffen. Diese werden bereits vor der Initiierung des Produktsystems festgelegt und gegebenenfalls gesondert akt ualisiert.
A	- Markstrategie (erkennbare Tendenzen) (siehe Abschnitt 4.3) - Systemstrategie (alte Version) (siehe Abschnitt 4.3)
R=Rollen, O=Entscheidungsobjekte, K=Bewertungskriterium, E=Entscheidun gsergebnis und A=Anleitungen/Vorlagen	

Entscheidungsproblem: Zur Unterstützung der Definition der Entwicklungssicht muss die Systemstrategie vor dieser Aktivität auf den neuesten Stand gebracht werden. Dazu müssen innerhalb des in der Systemarchitektur fes tgehaltenen Planungshorizonts Schwerpunkte für einzelne Versionen festgelegt werden.

Zielfunktion: Bei der Definition der Systemstrategie ist keine Festlegung einer eindeut i-gen Zielfunktion möglich. Da die Systemstrategie selbst sehr vage Formulierungen en t-hält, ist hier auch keine klare Argumentation mö glich.

Neue Produkte entstehen entweder dann, wenn die Bedeutung mancher Marktsegmente zunimmt und dafür neue produktexterne Variationen erstellt werden oder aufgrund von Innovationen. Das Verschmelzen von P rodukten ist dann notwendig, wenn bestehende Marktstrukturen sich ändern und daher stärker verzahnt sind. Produkte verschwinden bei abnehmender Marktbedeutung oder bei einer allgemeinen Strategieumorientierung.

Vorgehensweise: Wie auch beim Innovationsmana gement (vgl. Abschnitt 7.4.1) kann auch bei der Definition der Systemstrategie schwer eine systematische Vorgehensweise festgelegt werden. Zumindest jedoch sind bei der Festl egung einer Systemstrategie das vorhandene System und alte Systemstrategien zu berücksichtigen, um nicht unnötig gr o-ße Änderungsaufwände zu verurs achen.

Beispielanwendung *Pisa*

Für die Systemstrategie für Pisa wird festgelegt, dass die ersten Versionen von der Erfüllung der Kernkompetenzen geprägt sein sollen. In der ersten Systemversion soll daher erst Pisa-Kern *ausgeliefert werden, dann* Pisa-Adress. *Hierbei ist darauf zu achten, dass die wesentlichen Anforderungen erfüllt sind. Die anderen beiden Softwareprodukte sollen erst in der nachfolgenden Systemversion in die Planung miteinbezogen werden.*

7.5.3 Definition der Entwicklungssicht

Der letzte Abschnitt dieses Kapitels stellt lediglich eine einzelne Aktivität vor, die Definition des Entwicklungssicht. Diese Aktivität wird parallel für jedes Systemmodule durchgeführt.

A 21 Definition der Entwicklungssicht

Entscheidung Definition der Entwicklungssicht	
R	**Systemmodulplaner,** Entwickler
O	- Systemarchitektur (zu Systemmodulen aggregierte Anforderungen, Ressourcen und Entwicklungszeit zugeordnet) (siehe Abschnitt 4.4.3) Bewertung zur Realisierung ausgewählter Anforderungen (vgl. Abschnitt 6.1)
K	Bei der Verteilung der Anforderungen auf verschiedene Systemmodul-Versionen muss zwischen strategischen und operativen beziehungsweise zwischen marktorientierten entwicklungsorientierten Kriterien unterschieden werden. Ein Vorschlag einer Kriterienliste zur gegenseitigen Bewertung von Anforderungen wird in Abschnitt 6.5 vorgestellt. In manchen Fällen sind die Marktbedeutung und der Entwicklungsaufwand abhängig von der Version in der sie realisiert werden. Dies muss bei der Bewertung zusätzlich berücksichtigt werden.
E	Die in Abschnitt 4.4.4 erläuterte Entwicklungssicht repräsentiert das letztendliche Entscheidungsergebnis dieser Phase. Es können hierfür unterschiedliche Alternativen gebildet werden.
A	- Portfolio-Methode zur Bewertung aus Abschnitt 6.6, lineare Optimierung und Rucksack-Methode zur Priorisierung (vgl. [Jun98]) - Systemstrategie (Anzahl expliziter Versionen definiert, schrittweise Planung festgelegt) (siehe Abschnitt 4.3)
R=Rollen, O=Entscheidungsobjekte, K=Bewertungskriterium, E=Entscheidungsergebnis und A=Anleitungen/Vorlagen	

Entscheidungsproblem: Aufgabe der Definition der Entwicklungssicht ist die Planung künftiger Versionen von Systemmodulen. Dazu müssen alle zur Realisierung vorgesehenen Anforderungen auf Versionen verteilt werden. Die Planung der Versionen einzelner Systemmodule erfolgt weitgehend parallel. Dabei müssen jedoch in den nachfolgenden Abschnitten erklärte Restriktionen berücksichtigt werden.

Zielfunktion: Zielfunktion ist der langfristige Erfolg des Produktsystems. Je nach aktueller Situation können Bewertungskriterien aus Abschnitt 6.5 unterschiedlich stark gewichtet werden.

Restriktionen: Hauptsächliche Restriktionen sind einem Systemmodul zugeordnete Ressourcen und zu berücksichtigende Zeitrestriktionen. Bei der parallelen Weiterentwicklung muss daneben auch die Kopplung verschiedener Systemmodule (vgl. Abschnitt 4.4.3) berücksichtigt werden. Darüber hinaus sind bei der Versionsplanung die gleichen Restriktionen hinsichtlich Assoziationen zu beachten, wie bereits bei der Aktivität *Auswahl zu realisierender Anforderungen und Variationen* aus Abschnitt 7.5.2 beschrieben wurde.

Die höchste Flexibilität bei der parallelen Entwicklung wird dann erreicht, wenn selbst Anfangs und Fertigstellungszeitpunkte einzelner Versionen verschiedener Systemmodule unabhängig voneinander sind. Wegen der Kopplung können Systemmodule nic ht vollständig voneinander unabhängig entwickelt werden. Der Grad der Abhängigkeit wird in der Praxis durch Bedingungen festgelegt, die jedes Systemmodul in der Entwicklung einzuhalten hat.

Wird ein Systemmodul $p \in S$ weiterentwickelt, so stellt es an ein zu ihm gekoppeltes Systemmodul $q \in S$ neue Anforderungen. Da sich p in seiner Weiterentwicklung auf diese stützt, muss bekannt sein, welche davon in der neuen Version von q erfüllt sein werden. Daher muss vor der endgültigen Versionsplanung von p mit den Entwicklern von q über die Versionen der Erfüllung einzelner Anforderungen der Modulschnittstelle verhandelt werden.

Unabhängig davon kann bei Auslieferung von p nicht davon ausgegangen werden, dass q ebenfalls bereits in der neuen Version am Markt verfügbar ist. Umgekehrt soll die neue Version von p unmittelbar nutzbar sein. Ein Systemmodul sollte daher so entw ickelt werden, dass es mit zumindest den letzten bereits am Markt verfügbaren Versionen gekoppelter Systemmodule zusammenarbeiten kann. Aus Marketingsicht wäre es darüber hinaus auch wünschenswert, dass Systemmodule auch mit noch älteren Systemmodul-Versionen zusammenarbeiten können. Dadurch könnten Kunden mit bereits installierten älteren Systemmodulen auch die neuere Version eines Systemmoduls p nutzen, ohne andere Systemmodule erneuern zu müssen. Die Einhaltung der Kompatibilität eines Sy stemmoduls mit älteren Versionen anderer Systemmodule bedeutet meist bestimmte Au fwände.

Daher muss die Forderung nach der Zusammenarbeit von Systemm odulen auf bestimmte Umfänge beschränkt werden. Grundsätzliche Idee ist hierbei, dass nur bei Sy stemmodul-Versionen *„nahe“* zusammenliegender Systemarchitektur-Versionen (vgl. Definition in Abschnitt 4.4.4) die Zusammenarbeit garantiert werden muss. Die Nähe wird durch eine natürliche Zahl, der sogenannten *Versionsfensterlänge*, ausgedrückt.

Hierzu muss die in Abschnitt 4.4.3 definierte Kopplung zwischen Systemmodulen (au sgedrückt durch das Prädikat $worktogether(p,q)$) auf die Kopplung zwischen Systemm odul-Versionen verfeinert werden. Dazu wird zunächst das Prädikat $versionsworktogether(p_u,q_v)$ eingeführt:

Definition: Kopplung von Systemmodul-Versionen

Seien $p,q \in S$ *Systemmodule und* $u,v \in V$ *Versionsidentifikatoren. Weiter seien mit* $M_{p_u}^{ford}$ *und* $M_{p_u}^{erf}$ *die Modulschnittstellen-Versionen bezeichnet, die eine Systemmodul-Version* p_u fordert bzw. erfüllt (vgl. Abschnitt 4.4.4). *Zwei* Systemmodul-Versionen p_u und q_v arbeiten *genau dann zusammen (oder* sind gekoppelt), *(in Zeichen* $versionsworktogether(p_u,q_v)$*) wenn gilt:*

$$\exists w \in V \exists m_w \in M_{p_u}^{ford} : m_w \in M_{q_v}^{erf} \vee \exists m_w \in M_{q_v}^{ford} : m_w \in M_{p_u}^{erf}$$

Dies bedeutet informell, dass zwei Systemmodul-Versionen p_u und q_v zusammenarbeiten, wenn zwischen ihnen eine gemeinsame Modulschnittstellen-Version existiert. Dies bedeutet, dass diese beiden Systemmodule gemeinsam einsetzbar sind. Wenn jedoch nicht beide Systemmodule zur gleichen Systemarchitektur-Version gehören, so bedeutet dies einen reduzierten Funktionsumfang für mindestens eines der beiden Systemmodule.

Damit kann die versionierte Kopplung zwischen Systemmodulen definiert werden:

Hilfsdefinition: Subsequenz

Sei M eine diskrete Menge. Dann ist $N \subset M$ genau dann eine Subsequenz von M, $subseqlist(N,M)$ wenn gilt:

$$\forall l \in N : l = \min(N) \vee \exists m \in N : l = succ_M(m)$$

Definition: Versionierte Kopplung zwischen Systemmodulen

Seien $p,q \in S$ Systemmodule, V die Menge von Versionsidentifikatoren und $V_S \subset V$ die Menge von Systemversionsidentifikatoren (Definitionen siehe Abschnitt 4.4.4) und n *die Versionsfensterlänge. Dann* arbeiten zwei Systemmodule in einem versionierten Produktsystem S *genau dann* zusammen *(oder* sind gekoppelt*) (in Zeichen $worktogether(p,q)_n$) wenn gilt:*

$$Synch \subset V_S \wedge |Synch| = n \wedge subseqlist(Synch, V_S) \Rightarrow$$
$$\forall u,v \in V, p_u \in \bigcup_{w \in Synch} S_w, q_v \in \bigcup_{w \in Synch} S_w : versionsworktogether(p_u, q_v)$$

Zur Erläuterung dieser Definition:

- die Menge $Synch$ symbolisiert *n* aufeinanderfolgende Systemversionsidentifikatoren
- $\bigcup_{w \in Synch} S_w$ repräsentiert alle Systemmodul-Versionen, die zu Systemarchitektur-Versionen mit Systemversionsidentifikator aus $Synch$ aggregiert sind.
- $worktogether(p,q)_n$ gilt genau dann, wenn für je zwei Systemmodule $p,q \in S$ beliebige Systemmodul-Versionen zusammenarbeiten können, sobald beide sich innerhalb einer Versionsfensterlänge *n* aufeinanderfolgender Systemarchitektur-Versionen befinden.

Das folgende Beispiel illustriert diesen Sachverhalt noch einmal:

Beispiel:

In Abbildung 37 sind drei aufeinanderfolgende Systemarchitektur-Versionen des Produktsystems PSystem aus Abschnitt 4.4.4 und zugehörige Versionen der Systemmodule P1 und B dargestellt. Eine Kante zwischen zwei Knoten repräsentiert die versionsworktogether*-Beziehung zwischen zwei Systemmodul-Versionen. Die Versionsfensterlänge beträgt hier $n = 2$. Zur Erfüllung von $worktogether(P1,B)_2$ muss zum Beispiel Systemmodul P1 in Version V.6.0 mit Systemmodul B in den*

Versionen V.5.0, V.5.1 und V.6.0 zusammenarbeiten. Dagegen ist keine Zusammenarbeit mit Systemmodul B in den Versionen V.4.0, V.4.1 zu garantieren.

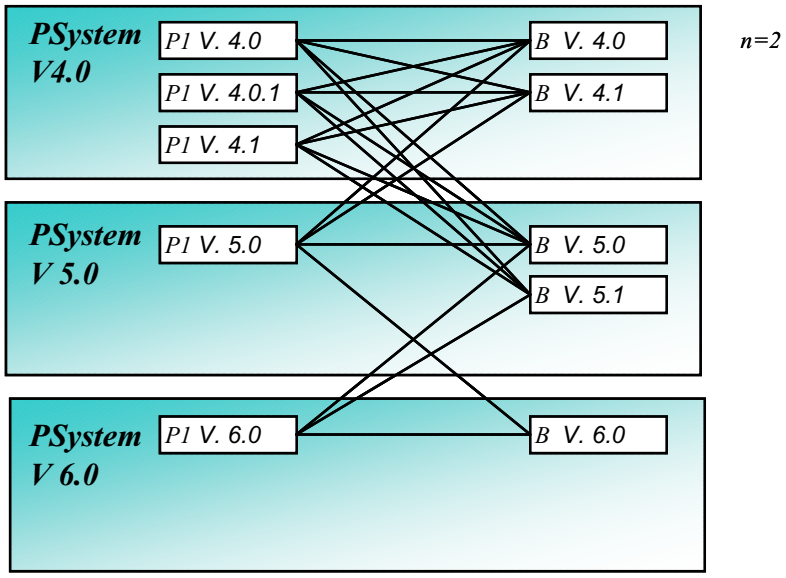


Abbildung 37 – Zusammenarbeit aufeinanderfolgender Systemversionen

Als Konsequenz der Kopplung muss ein Systemmodul auch dann lauffähig sein, wenn manche von ihm über Modulschnittstellen geforderte Anforderungen nicht erfüllt werden. Je nach Installation muss also eine Systemmodul-Version p_v mit festem $v \in V$ mit gekoppelten Systemmodul-Versionen q_u mit unterschiedlichem $u \in V$ zusammenarbeiten, die unterschiedliche Anforderungen erfüllen. Um das zu bewerkstelligen, müssen also für p_v hinsichtlich der Nutzung der Anforderungen von Modulschnittstellen Variationen gebildet werden. Für die Entwicklung in einem versionierten Produktsystem wird in dieser Arbeit gefordert:

Entwicklungsaxiom: Asynchrone Entwicklung eines Produktsystems

Sei S ein Produktsystem und $p, q \in S$ zwei Systemmodule. Für die asynchrone Entwicklung eines Produktsystems wird gefordert:

$worktogether(p,q)_n$ und Versionsfensterlänge $n \geq 2$.

Eine Fensterlänge von $n \geq 2$ bedeutet, dass Systemmodule des letzten Systemversion sidentifikatoren *v* und seines Vorgängers (*v-1*) die *versionsworktogether*-Eigenschaft erfüllen müssen. Damit wird durch dieses Axiom zugesichert, dass Systemprodukt-Versionen (die Spezialisierung von Systemmodul-Versionen in Analogie zu Systemprodukten und Systemmodulen) sofort nach der Fertigstellung verkauft werden können. Unabdingbare Voraussetzung dafür ist die Zusicherung, dass ein Produkt mit jeweils mindestens einer Version aller anderen Produkte zusammenarbeiten kann.

Beispiel:

Angenommen P1 aus Abbildung 37 wird in Version V.6.0 vor Version V.6.0. von B ausgeliefert. In diesem Fall kann P1 bei Einhaltung der obigen Regel in Kombination mit B V.5.0 oder V.5.1 installiert werden. Dann kann teilweise nur reduzierte Funktionalität von P1 abgerufen werden, wenn P1 von V.6.0 über eine Modulschnittstellen-Version neue Anforderungen fordert.

Die Wahl der richtigen Fensterlänge hängt von Marketingzielen und assoziierten Entwicklungsaufwänden ab. Je größer die Versionsfensterlänge, desto höher sind die Marktchancen einzelner Systemprodukte. Diese können dann auch mit umso mehr älteren bereits installierten Systemprodukten zusammen eingesetzt werden. Umgekehrt erhöhen sich gleichzeitig die korellierten Entwicklungsaufwände. Die Versionsfensterlänge wird strategisch festgelegt und bleibt im Normalfall über die gesamten Entwicklungszyklen eines Produktsystems unverändert.

Die Kopplung von Systemmodulen beeinflusst die Versionsplanung einzelner Systemmodule in zweifacher Hinsicht. Einerseits müssen die innerhalb einer Modulschnittstellen-Version zu realisierenden Anforderungen mit der Versionsplanung des damit gekoppelten Systemmoduls verhandelt werden. Andererseits muss jedes Systemmodul die Zusammenarbeit mit verschiedenen Versionen gekoppelter Systemmodule berücksichtigen.

Vorgehensweise: In die Versionsplanung werden alle zur Realisierung ausgewählten Anforderungen einbezogen (vgl. Aktivität *Auswahl zu realisierender Anforderungen* aus Abschnitt 7.5.2). Zuerst erfolgt die Festlegung erlaubter Aufwände von permanenten Anforderungen. Für die dann noch zur Verfügung stehenden Aufwände erfolgt anhand der Bewertungen sukzessive die Versionsplanung für verbrauchbare Anforderungen (vgl. Abschnitt 4.2.1). In der nächsten Systemmodul-Version werden nach absteigendem Wert so viele Anforderungen eingeplant, wie dies aufgrund der verfügbaren Ressourcen und Zeit möglich ist. Zur Planung werden die Aufwandschätzungen für die Realisierung einzelner Anforderungen aus Abschnitt 7.5.2 miteinbezogen.

Haben Anforderungen eine relativ hohe Wichtigkeit, verursachen jedoch gleichzeitig einen hohen Realisierungsaufwand, so können diese gegebenenfalls in Multi-Versions-Anforderungen (vgl. Abschnitt 4.4.4) aufgeteilt werden. Aufgrund wechselseitiger Einflüsse zwischen Systemmodulen muss die Versionsverteilung der Anforderungen auch mit anderen Systemmodulen abgeglichen werden. Zur sukzessiven Planung kann die oben erwähnte Rucksack-Methode herangezogen werden. Gegebenenfalls lässt sich die Zielfunktion auch mathematisch formulieren, so dass auch mit Hilfe der linearen Optimierung eine optimale Anforderungsverteilung gefunden werden kann. Dafür müssten jedoch die erwähnten Restriktion in adäquater Weise berücksichtigt werden.

Die fertige Entwicklungssicht ist die unmittelbare Basis zur werkzeuggestützten automatischen Spezifikation von Anforderungen, die in Abschnitt 5.2 vorgestellt wurde.

Beispielanwendung *Pisa*

Bei der Definition der Entwicklungssicht von Pisa-Kern *wird zuerst die permanente Anforderung* A2 *mit zwei Aufwandseinheiten eingeplant. Damit stehen noch 28 Aufwandseinheiten (vgl. Abschnitt 7.5.2) für verbrauchbare Anforderungen zur*

Verfügung. Über M2 *und* M3 *gekoppelte Systemmodule werden aufgrund der Systemstrategie (vgl. Abschnitt 7.5.2) frühestens ab der nächsten Version entwickelt. Daher entstehen zur ersten Version noch keine Abhängigkeiten über diese Schnittstellen. Dagegen muss über* M1 *verhandelt werden, welche Anforderungen für* Pisa-Kern *in der aktuellen Version von* Pisa-ABasis *eingeplant werden sollen. Einige Anforderungen sind unbedingt notwendig für* Pisa-Kern *und müssen in jedem Fall bereits bei dessen Auslieferung realisiert sein. Daher muss* Pisa-ABasis *mindestens gleichzeitig mit* Pisa-Kern *fertiggestellt sein.*

Die Versionsfensterlänge spielt in der ersten Version noch keine Rolle. Für die zweite Version ist zum momentanen Zeitpunkt noch nicht absehbar, ob hieraus eine weitere Anforderung berücksichtigt werden muss.

*Die Verhandlung aller anderen Anforderungen ergibt folgende Realisierungsentscheidung. Zuerst sollen alle Anforderungen mit maximaler Bewertung bei den verkaufsorientierten Kriterien eingeplant werden (*A8, A28, A33, A37, A38, A21, A29*). Die Einzelverhandlung der restlichen Anforderungen ergibt, dass zusätzlich bei den zur Verfügung stehenden Ressourcen noch* A27 *und* A26 *realisiert werden sollen. Anforderungen* A1, A30, A19, A2, A31 *und* A10 *sollen für die zweite Version eingeplant werden. Aufgrund der Bedeutung der Anforderungen wird darüber hinaus auch noch entschieden, dass die Anforderungen mit maximaler verkaufsorientierter Bewertung auf jeden Fall bereits in der ersten Version realisiert werden sollen. Alle anderen Anforderungen sollen gegebenenfalls auf die nachfolgende Version verschoben werden.*

Kapitel 8

Zusammenfassung und Ausblick

Dieses abschließende Kapitel fasst die wesentlichen Ergebnisse der vorliegenden Arbeit zusammen und gibt einen Ausblick auf künftige Arbeiten. Die Zusammenfassung erfolgt in Abschnitt 8.1. Ein Überblick über die Möglichkeit zur Praxiseinführung wird in Abschnitt 8.2 gegeben. Offene wissenschaftliche Fragestellungen werden schließlich in Abschnitt 8.3 erläutert.

8.1 Zusammenfassung

Das wesentliche Ergebnis dieser Arbeit ist eine Methodik für das Requirements Engineering komplexer Standardsoftware, das die Wechselwirkungen unternehmerischer Ziele mit dem Markteinfluss adäquat berücksichtigt.

Um die Unsicherheit über Anforderungen der gesamten Breite des Marktes einer Standardsoftware weitgehend zu reduzieren, wurde ein Vorgehensmodell zum systematischen Herausarbeiten aus unterschiedlichen Quellen und Kanälen vorgestellt. Dabei wurden für jeden Quellen- und Kanaltyp eigene Aktivitäten vorgeschlagen. Um auch langfristige Visionen untersuchen zu können, wurde ein Bezug zum Innovationsmanagement hergestellt. Dem Ziel der Kundenorientierung und der Vielfalt individueller Anforderungen wurde durch die Marktsicht Rechnung getragen. Diese unterstützt die Identifikation von Variationen und gegenseitiger Abhängigkeiten.

Vorrangiges Ziel der Systemsicht ist die Unterstützung einer weitgehend parallelen Entwicklung einzelner Systemmodule, um kurze Reaktionszeiten auf dem sich wandelnden Markt zu ermöglichen. Dies wird im Modell der Entwicklungsprodukte durch eine präzise definierte Zuordnung von Anforderungen zu Systemmodulen und Modulschnittstellen unterstützt. Dadurch können bei der Weiterentwicklung einzelner Systemmodule Abhängigkeiten zu anderen Systemmodulen leicht identifiziert und adäquat berücksichtigt werden. In der Marktsicht und in der Systemsicht wurden außerdem Möglichkeiten zur Wiederverwendung von Anforderungen mit dem Ziel der Kostenreduktion integriert. Hierzu dient in der Marktsicht die Identifikation gemeinsamer Kerne von Variationstypen. In der Systemsicht können in Basismodulen wiederverwendbare Anforderungen getrennt von anderen Systemmodulen entwickelt werden.

In der Versionsplanung der Anforderungen der Entwicklungssicht wird vorrangig die Einhaltung vorgegebener Zeit- und Kostenrestriktionen verfolgt. Die Verteilung der Anforderungen auf künftige Versionen erfolgt anhand einer an den Unternehmenszielen orientierten Bewertung der Anforderungen. Durch die Entwicklungssicht wird auch ein vorausschauendes Planen künftiger Versionen möglich, was die Reduktion künftiger Änderungsaufwände und –zeiten unterstützt.

Mit den präsentierten unterschiedlichen Sichten werden verschiedene Schwerpunkte im Requirements Engineering gesetzt. Gleichzeitig sind die Sichten jedoch nicht voneinander „isoliert" sondern es finden nahtlose Übergänge zwischen ihnen statt. Dies begründet sich allein schon darin, dass sich die an der Erarbeitung einzelner Sichten beteiligten Rollen überlappen. Umgekehrt wird in den zugehörigen Entwicklungsaktivitäten die Konzentration auf den Schwerpunkt einer Sicht durch wechselnde Hauptverantwortlichkeiten beteiligter Rollen gewährleistet.

In der vorgestellten Methodik ist die wechselseitige Verzahnung einer hohen Flexibilität mit einer guten Verfolgbarkeit ein weiterer Schwerpunkt. Grundvoraussetzung hierfür ist eine starke Integration des Prozessmodells mit dem Modell der Entwicklungsprodukte. Das Konzept eines entscheidungsorientierten Entwicklungsprozesses und eines Modells für Entwicklungsprodukte wurde vorrangig hierfür entwickelt.

Da innerhalb einzelner Entwicklungsaktivitäten die Aufgabe jeden Entwicklungsprodukts klar definiert ist, können Entscheidungen systematisch erarbeitet werden und sind im nachhinein gut verfolgbar. Die präzise Beschreibung des Modells der Entwicklungsprodukte durch Anwendung teilweise formaler Techniken unterstützt darüber hinaus Aufgaben des Konfigurations-Managements und des Requirements Managements. Die Anforderungen sind somit sowohl horizontal als auch vertikal von ihrer Herkunft bis hin zur Spezifikation in der Entwicklungssicht verfolgbar.

Durch die Verfolgbarkeit von Anforderungen wird auch die Flexibilität der Entwicklung erhöht, da dadurch notwendige Änderungen leicht identifizierbar sind. Im Prozessmodell wurde darüber hinaus keine starre Ausführungsreihenfolge einzelner Aktivitäten definiert, um die Flexibilität weiter zu erhöhen. Umgekehrt sind Aktivitäten eng über Modellzustände gekoppelt, damit trotzdem die Verfolgbarkeit gewährleistet werden kann.

Im Mittelpunkt des entscheidungsorientierten Entwicklungsprozesses stehen Verhandlungen zwischen unterschiedlichen Rollen. Hierfür wurde eine methodische Unterstützung durch skalierbare Bewertungsmethoden und Konzepte zur teilweise automatischen Spezifikation unterschiedlicher Sichten vorgestellt.

Schließlich wurde die Tragfähigkeit des gesamten Konzepts mittels des durchgängigen Fallbeispiels *Pisa* illustriert.

8.2 Mögliche Praxiseinführung

In der gesamten Arbeit wurde darauf geachtet, möglichst praxisnahe Konzepte zu entwickeln. Dieser Abschnitt skizziert notwendige Aktivitäten und mögliche Problemstellungen für die Umsetzung der Ergebnisse in die Praxis.

Zentrale Aufgabe für die Praxisumsetzung ist die Entwicklung einer Werkzeugunterstützung, die auf Konzepte aus Abschnitt 4.5 aufbaut. Da verschiedenartigste Benutzer mit einem derartigen Werkzeug umgehen müssen, hängt dessen Praxiserfolg vorrangig von der Berücksichtigung adäquater Benutzerprofile ab. Für einen Vertriebsbeauftragten z.B. ist die Erfassung von Anforderungen eine Nebentätigkeit. Daher sollte das Werkzeug möglichst aus dessen Sicht weitgehend einfach bedienbar sein und umgekehrt auch wenig Zeitaufwand erfordern. Am besten kann die Benutzung des Werkzeugs vor allem dadurch schmackhaft gemacht werden, wenn für den jeweiligen Benutzer ein Zusatznutzen zu bisher vorhandenen Werkzeugen erkenntlich wird. Dieser ist z. B. für einen Vertriebsbeauftragten gegeben, wenn dieser den Realisierungsstatus von Anforderungen seiner Kunden abfragen kann. Daneben bedarf es einer adäquaten Kopplung eines derartigen Werkzeugs beispielsweise mit UML-Werkzeugen. Damit können die Kapitel 5 vorgestellten Ansätze zur Spezifikation realisiert werden.

Umgekehrt können bestehende Ansätze aus der Praxis zur Ergänzung von in dieser Arbeit vorgestellten Konzepten nützlich sein. Beispielsweise befasst sich das Database Marketing [LiH93, LiH05] mit der technischen Integration unterschiedlicher Datenbanken, um insgesamt vorhandene Erfahrungen an verschiedenen Stellen eines Unternehmen gleichzeitig nutzen zu können. Derartige Erfahrungen können auch für die Aufstellung eines Erfahrungsmodells (vergleiche Abschnitt 6.7.1) genutzt werden.

Die Praxiseinführung des gesamten Konzeptes ist je nach bestehender Organisationsform mit enormen Schwierigkeiten verbunden. Häufig sind die vorgestellten Rollen nicht der Organisation eines Unternehmens adäquat, was vor Einführung des Konzeptes zu Umorganisationen führen kann. Besonders wenn eine komplexe Standardsoftware sich bereits seit mehreren Versionen in der Entwicklung befindet, dürfte die Praxiseinführung des gesamten Konzeptes nahezu unmöglich sein. In derartigen Fällen können einzelner Teile des Gesamtkonzeptes das Requirements Engineering verbessern. Die Einführung des Gesamtkonzeptes für eine neu zu entwickelnde Software ist jedoch lohnend.

8.3 Wissenschaftliche Fragestellungen

Die vorrangige wissenschaftliche Herausforderung künftiger Arbeiten ist die Fortsetzung der vorgestellten Konzepte in spätere Phasen der Entwicklung komplexer Standardsoftware. Wesentliche Problemstellung ist hierbei vor allem die klaffende Lücke zwischen dem Requirements Engineering und der Ableitung von Softwareentwürfen. Ursache hierfür sind teilweise sehr profane Problemstellungen wie die mangelnde Kommunikationsfähigkeit zwischen unterschiedlichen Wissenschaftsgemeinden. So zielt das Requirements Engineering auf die methodische Analyse der Anforderungen von Kunden und Anwendern ab. Forscher, die sich mit Softwarearchitekturen befassen, verstehen teilweise die Problemstellungen des Requirements Engineerings nicht und umgekehrt. Sie beschäftigen sich viel lieber mit technischen Fragestellungen, wie die Kommunikationsmechanismen oder Beschreibungssprachen für Architekturen.

Eine erste Brücke über diese Lücke können die in dieser Arbeit verwendete Aspektorientierung (vgl. Abschnitt 4.2.2) und das Konzept des entscheidungsorientierten Entwicklungsprozesses (vgl. Abschnitt 3.1.2) schlagen. Hiermit kann bereits früh in der Entwicklung eine Strukturierung von Anforderungen anhand entwicklungsrelevanter Aspekte erfolgen. Der entscheidungsorientierte Entwicklungsprozess schafft ein Konzept

zur systematischen Strukturierung von Aktivitäten über das Requirements Engineering hinaus. Der Entscheidungsorientierte Entwicklungsprozess kann ohne weiteres zur systematischen Ableitung von Architekturen aus bestehenden Anforderungen genutzt werden (vgl. [DSV99]). Durch eine Fortsetzung des Modells der Entwicklungsprodukte bis zur Implementierung kann die durchgängige Verfolgbarkeit der Anforderungen bewerkstelligt werden.

Neben der allgemeinen Lücke zwischen Requirements Engineering und späteren Phasen existieren auch spezifische Problemstellung der Entwicklung komplexer Standardsoftware. Stichpunkte hierfür sind die Realisierung von Variationen und die Unterstützung des Wandels.

Für die Entwicklung von Variationen sind vorrangig zwei Problemfelder zu untersuchen. Einerseits werden Architekturstile zur Entwicklung gemeinsamer Kerne und der Spezifika der Variationen verschiedener Variationstypen (vergleiche Abschnitt 4.4.2) benötigt. Im Umfeld der Produktlinien (vgl. z.B. [KCH+90, HLS+98]) und der Komponentenorientierung existieren bereits Lösungen ähnlicher Fragestellungen. Es bedarf hierbei noch einer weiteren Ergänzung zur Lösung spezifischer Problemstellungen der Entwicklung komplexer Standardsoftware.

Andererseits müssen systematische Techniken zur Unterstützung einer flexiblen Aktivierung unterschiedlicher Variationen innerhalb eines Softwaresystems gefunden werden. Diese müssen zunächst die Einhaltung der Laufzeitaxiome des Modells der Entwicklungsprodukte (vergleiche Abschnitt 3.2 und Kapitel 4) in der realisierten Software gewährleisten. Darüber hinaus hat die Aktivierung Auswirkungen auf Installation bzw. Konfiguration einer fertiggestellten Software beim Kunden (vgl. z.B. [HHW99]), wie auch auf Realisierungstechniken. Im Idealfall müssen hierfür adäquate Erweiterungen verschiedener Modellierungssprachen entwickelt werden, um eine Trennung der konzeptionellen Darstellung von Variationen und ihrer Realisierung bis hin zur Programmierung zu erreichen.

Zur Unterstützung des Wandels müssen Realisierungstechniken entwickelt werden, die eine Versionierung und Änderungsfreundlichkeit adäquat unterstützen. Das Requirements Engineering muss dazu mögliche künftige Änderungen stärker als bisher charakterisieren, damit entsprechende Realisierungen hierfür vorbereitet werden können. Ideen hierzu wurden in einer Kooperationsarbeit des Autoren publiziert [DS99].

Anhang A Entwicklungsprodukte im Überblick

Produkttypen

Grundelemente

Anforderung	4.2.1
Aspekt	4.2.2

Strategische Strukturierung

Marktstrategie	4.3
Systemstandards	4.3
Segmentierungstyp	4.3
Marktsegment	4.3
Systemstrategie	4.3

Operative Strukturierung

Erfassungskontext	4.4.1
Variationstyp (abgeleitet von Aspekt)	4.4.2
Variation (abgeleitet von Anforderung)	4.4.2
Systemarchitektur	4.4.3
Systemmodul	4.4.3
Softwareprodukt (abgeleitet von Systemmodul)	4.4.3
Basismodul (abgeleitet von Systemmodul)	4.4.3
Modulschnittstelle	4.4.3
Versionsidentifikator	4.4.4
Systemarchitektur-Version	4.4.4
Systemmodul-Version	4.4.4
Modulschnittstellen-Version	4.4.4

Bewertung

Kriterienkatalog	6.1
Bewertungskriterium	6.1
Zielfunktion	6.1
Bewertungsobjekt	6.1
Bewertungsergebnis	6.1
Kriterienwert	6.1

Erfahrungsmodell

Klassifizierungsmerkmal	6.7.1
Kontextklasse	6.7.1
Erfahrung	6.7.1
Beurteilungskriterium	6.7.1
Beurteilung	6.7.1
Kontext	6.7.2

Assoziationstypen

korreliert (Anforderung –Anforderung)	4.2.1
widerspricht (Anforderung – Anforderung)	4.2.1
benötigt (Anforderung → Anforderung)	4.2.1
korreliert (Aspekt –Aspekt)	4.2.2
korreliert (Variationstyp –Anforderung)	4.4.2
erwartet (Variation → Variation)	4.4.2
schließt_aus (Variation - Variation)	4.4.2
gewünscht hinsichtlich einer Anforderung (Variation – Variation)	4.4.2
korelliert (Variationstyp –Aspekt)	4.4.2

Anhang B Formeln im Überblick

Mengen

Zeichen	Bedeutung	Ref.
T	Zeitpunkte	4.2.1
S	Produktsystem	4.2.1, 4.4.3
R	Anforderungen	4.2.1
$R_t \subset R$	Bekannte Anforderungen zum Zeitpunkt $t \in T$	4.2.1
$R_{S,t} \subseteq R_t$	Für Produktsystem $S \subset P$ zum Zeitpunkt $t \in T$ gesammelte Anforderungen	4.2.1
$R_{S,t}^{pl} \subseteq R_{S,t}$	Für Produktsystem $S \subset P$ zum Zeitpunkt $t \in T$ geplante Anforderungen	4.2.1
$R_{S,t}^{r} \subseteq R_{S,t}$	Für Produktsystem $S \subset P$ zum Zeitpunkt $t \in T$ realisierte Anforderungen	4.2.1
$R_t^c \subseteq R$	Zum Zeitpunkt $t \in T$ verworfene Anforderungen	4.2.1
$R_{S,t}^{pl_{\text{perm}}} \subseteq R_{S,t}^{pl}$	Für Produktsystem S zum Zeitpunkt $t \in T$ permanente Anforderungen	4.2.1
$ASP_{S,t}$	Aspekt für Produktsystem S zum Zeitpunkt $t \in T$	4.2.2
$R_{A,t} \subset R_{S,t}$	Zu einem Aspekt $A \in ASP_{S,t}$ gehörige Anforderungen	4.2.2
$VT_{S,t} \subset ASP_{S,t}$	Variationstypen	4.4.2
$I_{W,t} \subset R_{W,t}$	Variationen für den Variationstyp $W \in VT_{S,t}$	4.4.2
$R_{i,t} \subset R_{W,t}$	Anforderungen einer Variation $i \in I_{W,t}$, $W \in VT_{S,t}$	4.4.2
$K_{W,t} = \bigcap_{i \in I_W} R_{i,t}$	Kern für den Variationstyp $W \in VT_{S,t}$	4.4.2
P	Systemmodule	4.4.3
M	Modulschnittstellen	4.4.3
$R_{p,t} \subseteq R_t$	An ein Systemmodul $p \in P$ zum Zeitpunkt $t \in T$ gestellte Anforderungen.	4.4.3
$R_{p,t}^{pl} \subseteq R_{p,t}$	Für Systemmodul $p \in S$ zum Zeitpunkt $t \in T$ geplante Anforderungen	4.4.3
$R_{p,t}^{r} \subseteq R_{p,t}$	Für Systemmodul $p \in S$ zum Zeitpunkt $t \in T$ realisierte Anforderungen	4.4.3
$M_p^{ford} \subset M$	Von Systemmodul $p \in P$ geforderte Modulschnittstellen	4.4.3
$M_p^{erf} \subset M$	Von Systemmodul $p \in P$ erfüllte Modulschnittstellen	4.4.3
V	Versionsidentifikatoren	4.4.4

$V_S \subset V$	Systemversionsidentifikatoren eines Produktsystems S.	4.4.4
$P_V \subset P \times V$	Existierende Kombinationen von Systemmodulen $p \in P$ mit Versionen $v \in V$	4.4.4
$M_V \subset M \times V$	Existierende Kombinationen von Modulschnittstellen $m \in M$ mit Versionen $v \in V$	4.4.4
$S_v \subset P_V$	Zu einer Systemarchitektur-Version in Systemversion $v \in V_S$ gehörende Systemmodul Versionen	4.4.4
$R^{pl}_{p_v,t} \subset R^{pl}_{p,t}$	Für eine Systemmodul-Version $p_v \in P_V$ zum Zeitpunkt $t \in T$ geplante Anforderungen	4.4.4
$R^{r}_{p_v,t} \subset R^{r}_{p,t}$	Für eine Systemmodul-Version $p_v \in P_V$ zum Zeitpunkt $t \in T$ geplante Anforderungen	4.4.4
$R^{r}_{p_v}$	Für eine Systemmodul-Version $p_v \in P_V$ endgültig realisierte Anforderungen	4.4.4

Prädikate

Prädikat	Bedeutung	Ref.
$correl(r_1, r_2)$	Korrelation zwischen Anforderungen $r_1, r_2 \in R$	4.2.1
$cont(R_1, R_2)$	Widerspruch zwischen Anforderungsmengen $R_1, R_2 \subset R$	4.2.1
$active_S(r, \tau)$ ($inactive_S(r, \tau)$)	Für Produktsystem S zum Zeitpunkt $t \in T$ (in)aktive Anforderungen	4.2.1
$Conf_{S,t}$	Für Produktsystem S zum Zeitpunkt $t \in T$ zulässige Konfiguration	4.2.1
$need(r_1, r_2)$	Benötigt-Assoziation zwischen Anforderungen $r_1, r_2 \in R$	4.2.1
$correl(A_1, A_2)$	Korrelation zwischen Aspekten $A_1, A_2 \in ASP_{S,t}$	4.2.2
$correl(V, r)$	Korrelation zwischen Variationstyp $V \in VT_{S,t}$ und Anforderung $r \in R_{S,t}$	4.4.2
$correl(W, A)$	Korrelation zwischen Variationstyp $W \in VT_{S,t}$ und Aspekt $A \in ASP_{S,t}$	4.4.2
$abgel(i_1, r, i_2)$	Ableitung der Variation $i_2 \in I_{W_2}$ aus einer Variation $i_1 \in I_{W_1}$ in Kombination mit Anforderung $r \in R_{S,t}$ der Variationstypen $W_1, W_2 \in VT_{S,t}$	4.4.2
$expect(i_1, i_2)$	Erwartung zwischen Variationen $i_1 \in I_{W_1}, i_2 \in I_{W_2}$, $W_1, W_2 \in VT_{S,t}$	4.4.2
$exclude(i_1, i_2)$	Ausschluss zwischen Variationen $i_1 \in I_{W1}, i_2 \in I_{W_2}$, $W_1, W_2 \in VT_{S,t}$	4.4.2
$worktogether(p, q)$	Kopplung zwischen zwei Systemmodulen $p, q \in P$	4.4.3

Anhang C Glossar

Anwendungsumfeld: Das Anwendungsumfeld oder auch die Anwendungsdomäne beschreibt allgemein die Einsatzumgebung einer Software. Aus dem Anwendungsumfeld entstammen Anforderungen an eine Software. Die Software wird dort eingesetzt, um dort bestimmte Aufgaben zu erfüllen.

Auslieferung/Auslieferungszeit: Unter Auslieferung einer Software versteht man den Vorgang der Produktion, des Verpackens und des Verkaufs der Software. Mit der Auslieferungszeit wird der erste Zeitpunkt verstanden, ab dem eine Software zum Verkauf angeboten wird. Wegen der kurzen Produktionszeit ist dieser Zeitpunkt gleichbedeutend mit dem Fertigstellungszeitpunkt. Dies ist der Zeitpunkt, an dem die Entwicklung einer Software beendet ist.

Basissystem: Unter Basissystemen wird eine Software oder Hardware verstanden, die zum Einsatz der zu entwickelnden Standardsoftware als gegeben vorausgesetzt wird. Eine Standardsoftware nutzt von einem Basissystem bestimmte Dienste, die zur Erfüllung eigener Anforderungen notwendig sind. Beispiele für Basissysteme sind Plattformen wie Betriebssysteme, Datenbanksysteme oder verwendete Rechnerplattformen.

KANO: Gängige Theorie zur Klassifikation der Kundenzufriedenheit. Für Anforderungen wird von drei unterschiedlichen Klassen ausgegangen: Grundmerkmale, Qualitäts- und Leistungsmerkmale und Begeisterungsmerkmale. Grundmerkmale sind nach dieser Theorie Anforderungen, die absolute Notwendigkeit für eine Software sind, ohne besonders erwähnt werden zu brauchen. Sind diese nicht realisiert, so senkt dies die Kundenzufriedenheit sehr stark, wohingegen deren Realisierung kaum bemerkt wird. Qualitäts - und Leistungsmerkmale steigern mit zunehmender Erfüllung kontinuierlich die Kundenzufriedenheit. Begeisterungsmerkmale hingegen steigern die Kundenzufriedenheit überproportional.

Mock-Up: Mock-Ups sind häufig die Vorstufe eines Prototypen. Anstatt eines ablauffähigen Programms werden Skizzen einer Oberfläche verwendet, um über verschiedene Programmausschnitte zu diskutieren.

Return on Investment: Schlagwort aus der Betriebswirtschaftslehre. Unter Return on Investment wird der die Gesamtrentabilität des investierten Kapitals einer Unternehmung verstanden. Einfach ausgedrückt, beschreibt es ob genügend Gewinn für das in die Unternehmung gesteckte Geld erwirtschaftet werden kann.

Usability: Unter Usability einer Software wird allgemein die Brauchbarkeit einer Software für den Praxiseinsatz verstanden. Hierzu zählt vor allem die Benutzerfreundlichkeit, die sich in der Einfachheit, Effizienz und Intuitivität der Bedienung einer Software äußert.

Anhang D Beispielanwendung Pisa

Der Anhang D enthält sämtliche Entwicklungsprodukte der Beispielanwendung Pisa (vgl. Abschnitt 7.3). Genaue Erläuterungen über das Zustandekommen der Entwicklungspr o-dukte werden in Abschnitten 7.4 und 7.5 gegeben.

Strategische Entwicklungsprodukte

Marktstrategie (Ausschnitte)

Kernkompetenzen

Es gibt eine Vielzahl verschiedener Kunden, die den Service Infopost bzw. Infobrief nutzen können. Das Beispiel-Produktsystem soll als Kern einen Kunden der Deutschen Post AG beim Versenden eigener Post unterstützen. Dazu soll die Software zunäch st einen Kunden bei der Entscheidung unterstützen, ob es für ihn Sinn macht, seine Post mit einer dieser Versendungsarten zu verschicken. Falls es sinnvoll ist, soll die Software darüber hinaus alle automatisierbaren Tätigkeiten zur Erfüllung der Vorgaben der Post adäquat unterstützen.

Um diese Kernkompetenzen zu erfüllen, das Produktsystem folgende zentrale Funktion a-litäten aufweisen:

- Einlesen von Adressen
- Sortieren der Adressen und optimieren der Versandkosten
- Druck der sortierten Adressen
- Druck möglicher Formulare

Profile der Schlüsselkunden

In Abschnitt 7.4.3 werden vier Schlüsselkunden im Rahmen eines Requ irements Workshops befragt. Diese haben folgende Profile:

Person A: ist Vorstandsmitglied eines Vereins mit ca. 1000 Mitg liedern und ist zustän-dig für die Öffentlichkeitsarbeit des Vereins. Dazu werden einmal jährlich alle Mitgli e-der mit einer Informationsbroschüre angeschrieben. Die Mitglieder konzentrieren sich vorrangig regional auf einen Landkreis. Adressbestände sind in einem selbstgestrickten Rechnungsprogramm rudimentär erfasst. Mit zunehmendem Mitgliederstand wächst der Wunsch nach einer eigenständigen Adressverwaltung, in der neben Adressen auch and e-re Informationen zu den Mitgliedern erfasst werden können.

Person B: ist Angestellter des Wahlreferats einer Großstadt mit 200000 Wahlberechtigten. Er ist zuständig für Wahlbenachrichtigungen, die mehrmals im Jahr versendet werden müssen. Wahlbenachrichtigungen müssen gesetzliche Standards hinsichtlich Format und Inhalt erfüllen. Adressbestände stammen von der Meldestelle der Stadt, wobei Wahlberechtigte erst ausgefiltert werden müssen.

Person C: ist Marketingbeauftragter einer Lotteriegesellschaft und versendet regelmäßig an ca. 100000 Empfänger verschiedene kleine Werbeprospekte. Mit zunehmender Verbreitung des Internet sollen Werbeaktionen auch mit E-Mail und Web-Seiten gekoppelt werden. Postversand wird mit Direktadressierern durchgeführt.

Person D: ist Marketingbeauftragter eines Kosmetikherstellers, der monatlich variierende Mengen von Probepackungen an Kosmetikartikeln an verschiedene Kunden verschickt. Durch eine eigene Internetpräsenz nimmt in letzter Zeit auch der Versand einzelner bestellter Kosmetikartikel zu, für den sich gegebenenfalls auch *Infopost* eignen könnte.

Marktsegmente

Nach einer ersten Analyse wurden die zwei Segmentierungstypen (vgl. Abschnitt 4.3) Kundengröße und Versendungszweck identifiziert. Die folgenden Tabellen geben einen kurzen Überblick über die Segmentierungstypen und daraus resultierenden Marktsegmenten:

Segmentierungstyp Kundengröße	
	Separiert nach Sendungsmenge, Anwendungshäufigkeit und Vorhandensein einer eigenen Adressverwaltung
Marktsegment	Beschreibung
Großkunde	Versand großer Mengen, häufige Anwendung, eigene umfangreiche Adressverwaltung
Kleinkunde	Versand kleiner Mengen, seltene Anwendung, einfache Adresslisten

Segmentierungstyp Versendungszweck	
	Separiert nach dem Zweck der Versendung
Marktsegment	Beschreibung
Information	reine Information
Direktmarketing	zum Zweck der Direktwerbung

Das Marketing schlägt folgende Gewichtung vor: Großkunden und Kleinkunden sollen in etwa gleichberechtigt berücksichtigt werden, wobei für Großkunden gegebenenfalls ein höherer Preis erzielt werden soll. Der höhere Preis soll durch mehr Funktionalität berechtigt sein. Der Versendungszweck soll in der ersten Version des Produktsystems noch keine große Rolle spielen. Künftige Versionen sollen diesbezüglich eine stärkere Gewichtung haben.

Operative Entwicklungsprodukte

Quellen von Anforderungen

Zu Beginn der Entwicklung von Pisa wurden drei wesentliche Quellen von Anforderu n-gen identifiziert:

- Kunden
- Normen der Post
- Anlagen zur Automatisierung von Versand

Erfassungskontexte

Folgende Tabelle gibt einen Überblick über unterschiedliche Erfassungskontexte für Anforderungen von Pisa:

Id	Quelle	Qtyp	Kanal	KTyp	Erf.Art	Erf.Dat
A	Person A	Kunde	A. Maler	Workshop,	Off Frag	14.03.01
B	Person B	Kunde	A. Maler	Workshop	Off Frag	14.03.01
C	Person C	Kunde	A. Maler	Workshop	Off Frag	14.03.01
D	Person D	Kunde	A. Maler	Workshop	Off Frag	14.03.01
E	Info. Infopost Post	Norm/Sta	A. Maler	Literat, exp	Analyse	10.03.01
F	Info. Infobrief Post	Norm/Sta	A. Maler	Literat, exp	Analyse	10.03.01
G	Person E	Kunde	E. Friedrichs	Verkauf	-	20.01.01
H	Person F	Kunde	R. Schulze	Verkauf	-	20.01.01

Zu den Personen E und F existieren weitere Informationen:

Person E ist Mitglied eines interdisziplinäres Forschungsprojekts, das zum Anlass einer Informationsveranstaltung Adressbestände aus unterschie dlichen Forschungseinheiten zusammenfährt um eine einheitliche Einladung hierfür zu versenden. Bisher ist der Fall einmal aufgetreten. Person F hat sich vor kurzem einen Palm V zugelegt und findet es toll, Adressen automatisch auf seinen Rechner übernehmen zu können.

Erfasste Anforderungen

Folgende Tabelle gibt einen Überblick über erfasste Anforderungen zum Start des ersten Versionszyklus. Sämtliche Anforderungen sind noch nicht gefiltert. Für den Erfassungskontext E ist nur ein Ausschnitt aller Anforderungen aufgeführt.

Id	Anforderungsbeschreibung	E-Id
A1	Optimierung des Postversands für 1000 regionale Adressen	A
A2	Einfache Bedienerführung, um einmal jährliche Benutzung effizient zu unterstützen	A
A3	Möglichkeit zum Einlesen von Adressen aus dem spezifischen Adressformat eines selbstgestrickten Rechnungsprogramms	A
A4	Erfassung anderer Informationen zu einzelnen Adressen	A
A5	Serienbrieffunktion für MS Word 2000	A
A6	Druck einzelner Adressen	A
A7	Druck von Mitgliederlisten	A
A8	Plattform Windows	A
A9	Zusätzliche Erfassung neuer Adressen	A
A10	Ermöglichung der Einhaltung gesetzlicher Layoutstandards für Wahlkarten	B
A11	Optimierung des Postversands für 200000 Adressen innerhalb eines Ortes	B
A12	Pünktliche Zustellung, daher Effizienz für das Programm	B
A13	Möglichkeiten zur Selektion verschiedener Adressen	B
A14	Einfache Bedienerführung, um seltene Benutzung effizient zu unterstützen	B
A15	Auslesen des Adressbestands aus einem Großrechner	B
A16	Rückführung berechneter Kosten in Finanzbuchhaltungssystem	B
A17	Effiziente Adressverwaltung mit Protokollierung der Empfänger	C
A18	Selektion aus Adressbeständen	C
A19	Optimierung des Postversands für ca. 100000 Adressen an verschiedene ausgewählte Adressen innerhalb des Bundesgebietes und im Ausland	C
A20	Kopplung zu anderen Direktmarketingaktivitäten, wie E-Mail und Web-Seiten zur Vermeidung redundanter Werbung	C
A21	Optimierung des Postversands für 100 bis 5000 Adressen für die Werbung	D
A22	Kopplung des Versands von Kosmetikartikeln mit dem innerbetrieblichen Mahnwesen	D
A23	Automatische Kopplung des Systems mit Internet-Bestellsystem zur Ermittlung der jeweiligen Anzahl inhaltsgleicher Versandstücke	D
A24	Zusammenführen mehrerer Adressbestände in einen und Duplikateleminat-ion	F
A25	Übernahme von Adressen aus einem Palm V	G
A26	Sortierung der Adressbestände nach Postleitzahl bei Mengen über 4000	E
A27	Selektion von Adressbeständen einer Leitregion, Sortierung der Adressbe-	E

Id	Anforderungsbeschreibung	E-Id
	stände nach Postleitzahl bei Mengen über 250 und unter 4000, Einliefe rung nur bei einem Postamt innerhalb der Leitregion	
A28	Selektion von Adressbeständen eines Leitbereichs, Sortierung der Adres s-bestände nach Postleitzahl bei Mengen über 50 und unter 250, Einlieferung nur bei einem Postamt innerhalb des Leitbereichs	E
A29	Maschinenlesbare Anschri ften mit genauer Vorgabe	E
A30	Lieferscheindruck	E
A31	Stapelung einzelner Sendungen in Briefbehältern, abhängig von Sendung s-größe und –masse in unterschiedlichen Stückelungen mit besonderen Ken n-zeichnungen	E
A32	Große Mengen ab 100 kg Stapelung auf Paletten mit besonderen Ken n-zeichnungen	E
A33	Berücksichtigung unterschiedlicher Sendungsarten: Unterscheidung in Se n-dungstyp (Brief, Katalog oder Paket), Sendungsfo rmat (Standard, Kompakt oder Groß) und in Verpackungsart (Umschla g oder hüllenlos).	E
A34	Preisberechnung und A dressbeschriftung hängen von Sendungsarten ab.	E
A35	Verwendung von Direktadressierern	C
A36	Schnelle Konfiguration für häufige Nutzung	

Die Optimierung des Postversands soll bei einem ausgewählten Zieladre ssbestand überprüfen, in welche Kategorien der Anforderungen A26 bis A28 die Versendung gehört. Wird eine Mindestmenge unterschritten, so müssen unterschiedliche Alternativen übe rprüft werden. Unter Umständen kann eine Aufteilung der Sendungen auf untersch iedliche Postämter (= Einlieferungsstellen) rentabel sein.

Identifizierte Aspekte, Variationstypen und Variationen

Folgende Tabelle gibt Aufschluss über Identifizierte Aspekte, Variationstypen und Vari a-tionen.

Aspekte und Variationstypen für das Produktsystem *Pisa*			
Typ	Bezeichner		Aggregierte Anforderungen
Asp	Betriebssystem		A8
Asp	Qualität		A12
Asp	Funktion		
Asp		Adr.bearbeitung	A7, A9, A13, A18
Asp		Adr. einlesen	
VT		Quellenanzahl	nur 1, mehrere (A24)
Asp		Optimierung	A1, A11, A19, A21, A33, A32
VT		Kleinstmengen	Std.Opt, Infobrief, Einlieferungsstellen
Asp		Adressausgabe	
Asp		Ausgabelayout	A10, A29, A34
VT		Ausgabeziel	A5, A35, ***A36***
Asp		Formularausgabe	A30, A31, A32
Asp		Direktmarketing	A17, A20
Asp		Sortierung	A31, A32, A6
Asp	Sendungsarten		
VT		Sendungstyp	Brief, Katalog, Paket
VT		Sendungsformat	Standard, Kompakt, Groß
VT		Verpackungsart	Umschlag, Hüllenlos
VT	Adr.format		A4, A3, ***A37***
VT	Adr.schnittstelle		A8, A15, A25, A3
VT	Bündelung		A31, A32
VT	Mengenselektion		A26, A27, A28
VT	Bedienung		
Var		Einf. Bedienung	A2,A14
Var		Automatisierung	A36
VT	Massenvers.		nein, ja (A32, A35)

Asp=Aspekt, VT= Variationstyp, Var = Variation

Durch die Identifikation von Variationstype n sind für die Variationstypen Ausgabeziel und Adr.format neue Anforderungen (A36 und A37) identifiziert worden. Diese entspr e-chen je einem gemeinsamen Kernen der beiden anderen Anforderungen (A36 ist Kern von A5 und A35, A37 ist Kern von A4 und A3).

Hierfür bedarf es auch der Ergänzung eines neuen Erfassungskontexts (Kontext I: Aktivität Aspekte und Variationstypen definieren ...):

Id	Anforderungsbeschreibung	E-Id
A36	Ausgabeziel ist Standarddrucker des entsprechenden Betriebsystems	I
A37	Zum ein- und ausgeben von Adressen wird vorgegebenes Standardformat verwendet	I

Folgende Graphik zeigt einen Ausschnitt der externen Spezifikation von Aspekten Variationstypen und Variationen.

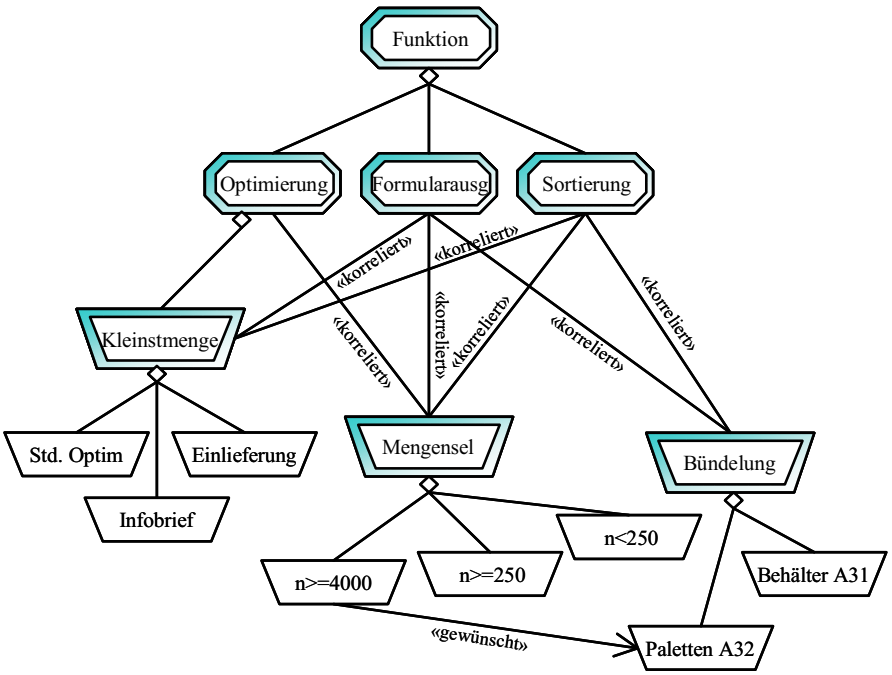


Abbildung 38 – Ausschnitt der externen Spezifikation von Aspekten, Variationstypen und Variationen

Einzelbewertungen ausgewählter Anforderungen

Folgende Tabelle zeigt Bewertungen der ausgewählter Anforderungen (Anforderungen, die das Softwareprodukt *Pisa-Kern* erfüllen soll) des Produktsystems *Pisa*.

Ausschnitt der Bewertung (Anforderungen an *Pisa-Kern*)					
Id	Kurze Beschreibung	Dring-lich-keit	Markt-bedeu-tung	Tech-nik	Auf-wand/E ffiz.
A1	Optimierung,1000 regionale Adressen	4	4	2	3
A8	Windows	5	5	5	5
A10	Wahlkarten	2	3	1	1
A11	Optimierung, 200000 Adressen am Ort	2	2	2	2
A12	Effizienz	1	2	2	2
A19	Optimierung, 100000 Adressen	3	4	2	3
A21	Optimierung, 100 bis 5000 Adressen für die Werbung	5	5	3	2
A26	Sortierung bei Mengen über 4000	4	4	3	3
A27	Sortierung bei Mengen über 250 und unter 4000	4	5	3	4
A28	Sortierung bei Mengen über 50 und unter 250	5	5	4	4
A29	Maschinenlesbare Anschriften	5	5	2	2
A30	Lieferscheindruck	4	3	2	4
A31	Stapelung in Briefbehältern	2	3	2	2
A33	unterschiedliche Sendungsarten	5	5	2	4
A37	Standarddrucker	5	5	4	3
A38	Standardformat für Adressen	5	5	4	2

Bewertung Permanenter Anforderungen (an *Pisa-Kern*)		Dring.	Markt.		Aufw.
A2	Einfache Bedienerführung	3	4		2

große Zahl: sehr gut (max 5)
kleine Zahl: seht schlecht (min 1)

Zuordnung der Anforderungen zu Systemmodulen

Die folgenden Tabellen zeigen exemplarisch die an den Softwareprodukten *Pisa-Kern* und *Pisa-Adress* vorgenommene Verteilung der Anforderungen auf einzelne Systemm o-dule. Die gewählte Darstellung soll die Abb ildung von Aspekten und Variation stypen zu den Systemmodulen leichter nachvollziehbar machen. Die Darstellung ist keine Spezif i-kation, sondern eine „interne“ Darstellung des Modellzustands.

Anforderungen an *Pisa-Kern*				
Typ	Bezeichner		Anforderung (erfüllt)	Über Schnittstelle M1 gefordert
Asp	Betriebssystem		A8	A8
Asp	Qualität		A12	A12
Asp	Funktion			
Asp		Adr. einlesen		
VT			Quellenanzahl		eine
Asp		Optimierung		A1, A11, A19, A21, A33, A32	
VT			Kleinstmengen	...	
Asp		Adressausgabe			
Asp			Ausgabelayout	A10, A29, A34	
VT			Ausgabeziel		***A36***
Asp		Formularausgabe		A30, A31, A32	
Asp		Sortierung		A31, A32, A6	
Asp	Sendungsarten				
VT		Sendungstyp		...	
VT		Sendungsformat		...	
VT		Verpackungsart		...	
VT	Adr.format			***A37***
VT	Adr.schnittstelle			A8
VT	Bündelung		A31	
VT	Mengenselektion		A26, A27, A28	
VT	Bedienung			
Var		Einf. Bedienung	A2,A14	A2,A14

Anforderungen an *Pisa-Adress*					
Typ	Bezeichner			Anforderung (erfüllt)	Über Schnittstelle M1 gefordert
Asp	Betriebssystem			A8	A8
Asp	Qualität			A12	A12
Asp	Funktion				
Asp		Adr.bearbeitung		A7, A9, A13, A18	
Asp		Adr. einlesen			
VT			Quellenanzahl	mehrere (A24)	eine
Asp		Adressausgabe			
Asp			Ausgabelayout	***A38***	
VT			Ausgabeziel	A5, A35	***A36***
VT	Adr.format			A4, A3	***A37***
VT	Adr.schnittstelle			A25, A3	A8
VT	Bedienung				
Var		Einf. Bedienung		A2,A14	A2,A14

Durch die Aufteilung in Systemmodule wurde erneut eine weiter e Anforderung identifiziert. Hierfür bedarf es wiederum der Ergänzung der Erfassungskontexte (Ko ntext K: Aktivität Zuordnung von Anforderungen zu Systemmodulen):

Id	Anforderungsbeschreibung	E-Id
A39	Ausgabe von Adressen erfolgt nach einem Standardlayout	K

Anhang E Folgerungen aus der Erwartung von Variationen

Steht eine Variation in erwartet-Assoziation zu einer anderen Variation, so folgt daraus:

Lemma: Deaktivierung nicht erwarteter Variationen

Seien $W_1, W_2 \in VT_{S,t}$ *Variationstypen,* $i_1 \in I_{W_1}$ *eine Variation und die Menge aller von* i_1 *aus* W_2 *erwarteten Variationen mit* $EXPECT = \{ j \in I_{W_2} \mid expect(i_1, j) \}$ *bezeichnet. Dann gilt:*

$\forall \tau \in T \forall k \in (I_{W_2} \setminus EXPECT) : active(i_1, \tau) \Rightarrow inactive(k, \tau)$

Beweis: *Nach Definition der Erwartung muss bei der Aktivierung von* i_1 *immer eine Variation* $j \in EXPECT$ *aktiv sein. Nach der Aktivierungsregel der Variationen sind alle anderen Variationen aus* I_{W_2} *inaktiv. Folglich ist dann nie eine Variation* $k \in (I_{W_2} \setminus EXPECT)$ *aktiv.*

Für die schwache Erwartung kann folgt analog:

Lemma: Deaktivierung nicht schwach erwarteter Variationen

Seien $W_1, W_2 \in VT_{S,t}$ *Variationstypen,* $i_1 \in I_{W_1}$ *eine Variation und die Menge aller von* i_1 *aus* W_2 *schwach erwarteten Variationen mit* $WEAKEXPECT = \{ j \in I_{W_2} \mid weakexpect(i_1, j) \}$ *bezeichnet. Dann gilt:*

$\forall \tau \in T \forall k \in (I_{W_2} \setminus WEAKEXPECT) : active(i_1, \tau) \Rightarrow inactive(k, \tau)$

Beweis: *Hier gilt eine zur Folgerung aus der Erwartung analoge Argumentation.*

Der Ausschluss von Variationen kann aufgrund dieser Folgerung auf schwach erwartete Variationen zurückgeführt werden:

Lemma: Rückführung des Ausschlusses auf schwach erwartete Variationen

Seien $W_1, W_2 \in VT_{S,t}$ *Variationstypen,* $i_1 \in I_{W_1}, i_2 \in I_{W_2}$ *wobei gilt:* $exclude(i_1, i_2)$. *Seien ferner folgende Mengen definiert:*

$EXCLUDED_1 = \{ j \in I_{W_1} \mid exclude(j, i_2) \}$ *und*

$EXCLUDED_2 = \{ j \in I_{W_2} \mid exclude(i_1, j) \}$, *dann gilt:*

$$(\forall j \in I_{W_2} \setminus EXCLUDED_2 : weakexpect(i_2, j) \wedge \forall j \in I_{W_1} \setminus EXCLUDED_1 : weakexpect(i_1, j)) \Rightarrow exclude(i_1, i_2)$$

Beweis: *Die beiden Mengen* $EXCLUDED_1$, $EXCLUDED_2$ *sind jeweils alle von der anderen Variation ausgeschlossene Variationen. Aufgrund des obigen Lemmas der Deaktivierung schwach erwarteter Variationen folgt der Ausschluss.*

Anhang F Beurteilungen aus Abschnitt 6.7.4

Die in Abschnitt 6.7.4 aus Erfahrungen im Kontext eines Herstellers von Standardsoftware für die Automatisierungstechnik entstanden aus folgenden Überlegungen heraus:

Quellen – Firmensicht: Den internen und externen Quellen wird ein in etwa gleich großes Hintergrundwissen über den Anwendungsbereich und dem Softwaresystem unterstellt. Wesentlicher Unterschied zwischen interner und externer Firmensicht ist daher vor allem eine verschieden starke Intensität der Kontakte. Mit internen Quellen ist eine weitaus intensivere Kommunikation möglich. Dadurch werden die Anforderungen in der Regel präziser und die Bedürfnisse der Kunden werden häufiger getroffen. Damit erfolgt für die Kriterien KANO, Kundenanteil und Marktvolumen bei internen Quellen eine tendenziell hohe und bei externen Quellen tendenziell niedrige Bewertung.

Quellen - Klassen - Kunden und Anwender: Bei Kunden und Anwendern wurde aufgrund vieler unterschiedlicher Kunden eine Verfeinerung vorgenommen. Es wird zwischen großen, kleinen und innovativen kleinen Kunden und Anwendern unterschieden. Eine in der Studie festgestellte Erfahrung war, dass innovative Anforderungen vor allem von großen und besonders innovativen kleinen Kunden und Anwendern stammen. Innovationen müssen aufgrund ihrer Neuartigkeit nicht dringend umgesetzt werden, jedoch werden sie erfahrungsgemäß eine insgesamt hohe Marktbedeutung erlangen. Bei kleinen nicht innovativen Kunden und Anwender kann tendenziell davon ausgegangen werden, dass deren Anforderungen eine hohe Dringlichkeit besitzen.

Die Klassifizierung von Kunden und Anwendern bietet vielfältige weitere Verfeinerungsmöglichkeiten, mit deren Hilfe anhand von Vergangenheitsdaten weitere Erfahrungen abgeleitet werden können. Dazu zählen zum Beispiel vergangene Umsätze, Mengen und Arten von Produkten mit unterschiedlichen Kunden. Aus diesen Daten lassen sich zum Beispiel strategisch wichtige Kunden identifizieren. Durch eine lückelose Verfolgbarkeit von Anforderungen zur Quelle lässt sich auch der Innovationsgrad eines einzelnen Unternehmers beurteilen. Wurden bei einer Quelle bereits in der Vergangenheit innovative Anforderungen genannt, so sind hieraus auch künftig Innovationen zu erwarten.

Quellen - Klassen – Innovationsmanagement: Aufgabe des Innovationsmanagements ist die mittelfristige bis langfristige Erarbeitung neuer Anforderungen. Innovationen sind dem Markt unbekannt und daher zunächst nicht dringend. Vom Innovationsmanagement stammende Anforderungen sind auf einen hohen Kundennutzen ausgelegt, wobei das tatsächliche Marktpotential und der Kundenanteil im Durchschnitt liegt. Bei eigenentwickelten Innovationen wird auch auf die Möglichkeit einer flexiblen Weiterentwicklung Wert gelegt.

Quellen - Klassen – Basissystem: Von Basissystemen abgeleitete Anforderungen haben meist einen relativ geringen Kundennutzen nach KANO (siehe Glossar). Umgekehrt sind sie sowohl dringlich und besitzen ansonsten auch hohe Marktbedeutung, da zur Behauptung der Marktposition des eigenen Produkts auf Neuerungen in Basissystemen reagiert werden muss.

Quellen - Klassen – Konkurrenzprodukte: Bereits in Konkurrenzprodukten realisierte Anforderungen besitzen offenbar hohe Marktbedeutung. Zum Aufholen des Rückstands gegenüber der Konkurrenz besteht eine hohe Dringlichkeit.

Kanäle – Firmensicht: Verglichen zu externen Kanälen können sämtliche Aktivitäten des Herausarbeitens der internen Kanäle spezifisch auf die Strategien und Bedürfnisse der eigenen Firma ausgerichtet werden. Daher werden interne Kanäle verglichen zu externen Kanälen mit höheren Marktbewertung beurteilt.

Durch dir Nutzung von nichtöffentlichem firmeneigenem Know-how können darüber hinaus bei internen Kanälen auch Wettbewerbsvorteile resultieren. Jedoch besteht die Gefahr von Fehlinterpretationen hinsichtlich der Relevanz von Anforderungen und der tatsächlichen Marktsituation, wenn zu sehr interne Maßstäbe zur Bewertung von Anforderungen angelegt werden.

Demgegenüber agieren externe Kanäle aus einem vom Firmenfokus losgelösten Blickwinkel. Deren Informationen können meist aber auch von Konkurrenzunternehmen genutzt werden. Darüber hinaus reduziert sich in der Regel durch die verlängerten Kommunikationswege von der Quelle bis hin zur Entwicklung die Treffsicherheit der Anforderungen.

Kanäle - Aktive Kontakte: Mit aktiven Kontakten können Anforderungen insbesondere von strategischen Zielgruppen sehr gezielt herausgearbeitet werden. Daher besteht eine gute Treffsicherheit für eine hohe Marktbedeutung von Anforderungen.

Kanäle - Passive Kontakte: Weniger gezielt erfolgt die Auswahl der Ansprechpartner bei passiven Kontakten. Durch die große Streuung von Ansprechpartnern können umgekehrt innovative Anforderungen aufgespürt werden, deren Dringlichkeit relativ gering ist.

Kanäle - Erfahrungen aus Zusammenarbeit: Im Gegensatz zu den passiven Kontakten stammen die Anforderungen bei Erfahrungen aus Zusammenarbeit nicht von Aussagen dritter. Stattdessen bringt ein Mitarbeiter selbst aufgrund eigener Erfahrungen Anforderungen an das Softwaresystem ein. Durch das eigene Wissen in der direkten Anwendung der Software ist ein hoher KANO-Wert (siehe Glossar) zu erwarten. Umgekehrt sind Erfahrungen aus der Zusammenarbeit sehr spezifisch und nicht unmittelbar verallgemeinerbar.

Anhang G Literatur

[ABD+99] Ansorge D., Bergner K., Deifel B., Hawlitzky N., Maier C., Paech B., Rausch A., Sihling M., Thurner V., Vogel S., "Managing Componentware Development - Software Reuse and the V-Modell Process", LNCS 1626, Proceedings of CAiSE'99, Seiten 134-148, Heidelberg, 1999.

[All97] Allen R.K., "A Formal Approach to Software Architecture", Carnegie Mellon University Pittsburgh, PhD Thesis, 1997.

[Aoy93] Aoyama M., "Concurrent Development Process Model", IEEE Software, Vol. 10, No. 4, Jul. 1993.

[Bab86] Babich W.A., "Software Configuration Management, Coordination for Team Productivity", Addison Wesley, 1986.

[Bag97] Bager J., "Die Redmond-Strategie - Schlüssel für Microsofts Erfolg", c`t, Nr. 14, S. 88, 1997.

[BCC97] Bass L., Clements P., Cohen S., "Product Line Practice Workshop Report", Carnegie Mellon University, SEI-97-TR003, Jun., 1997

[BD91] Bersoff E.H., Davis A.M., "Impact of Life Cycle Models on Software Configuration Management, Comm. ACM, Vol. 34, No. 8, Aug. 1991

[BDD99] Billing G., Deifel B., Dinges C., "Prozeßmodell des Innovationsmangements komplexer Standardsoftware", Technischer Bericht IMMD II – 9/99, FAU Erlangen-Nürnberg, 1999

[BDJS99] Büyükekici B., Deifel B., Jacobi C., Sandner R. "Prioritization of complex COTS", Proceedings of REFSQ'99, Heidelberg, 1999

[BeD91] E.H. Bersoff, A.M. Davis, "Impact of life cycle models on software configuration management", Comm. ACM, vol. 34, no. 8, pp. 104-118, Aug. 1991.

[BFG+93] Broy M., Facchi C., Grosu R., et. al., "The Requirement and Design Specification Language SPECTRUM An informal Introduction", Technischer Report TUM-I9311/TUM-I9312, Technische Universität München, 1993

[BMR+96] Buschmann F., Meunier R., Rohnert H., Sommerlad P., Stal M., "Pattern – oriented Software Architecture – A System of Patterns", Wiley, 1996

[Boe88] B. Boehm, "A spiral model of software development and enhancement", IEEE Computer, vol. 21, pp. 61-72, May, 1988.

[Bro91] Brown, P.G., "QFD, echoing the voice of the customer", AT&T Technical Journal, S. 18-32, März/April, 1991.

[Bro95] Brockschmidt K., "Inside OLE", 2nd Ed., Microsoft Press , 1995

[CaB95] Carmel, E., Becker S., "A Process Model for Packaged Software Development", IEEE Trans. on Eng. Manag., Vol. 42, No. 1, Februar, 1995

[CAJ98] Cheong Y.C., Anande A. L., Jarzabek S. , "Handling Variant Requirements in Software Architectures for Product Families", ARES II, LNCS 1429, Las Palmas, Spain, 1998

[Che76] Chen P.P., "The entity-relationship model – towards a unified view of data", ACM Transactions on Database Systems, 1(1):9-36, 1976.

[CoK87] Cooper R.G., Kleinschmidt E.J., "New Products: What Separates Winners from Losers", J. Prod. Innov. Manag., No. 4, 1987

[CoW97] Conradi R., Westfechtel B., "Towards a Uniform Version Model for Software Configuration Management", Proc. 7th Intern. Workshop on Software Configuration Management, Boston, LNCS 1235, Boston, 1997

[CTO99] Clarke S., Tarr P., Ossher H., "Designing for Evolution with Subjects", SCE´99 ICSE´99, Los Angeles, 1999

[CuS95] Cusumano M. A., Selby R. W., "Microsoft Secrets", Free Press, New York, 1995

[Dav92] Davis A., "Operational Prototyping: A new Development Approach", IEEE Software, Sept., 1992

[Dav93] Davenport T., "Process Innovation: Reengineering Work through Information Technology", Harvard Business School Press, Boston, Ma, 1993

[Dav95] Davis M.J. "Adaptable Reusable Code", SSR'95, Seattle, WA, USA, 1995

[Dei98a] Deifel B., "SIEMENS A&D AS - Requirements Engineering- und Entwicklung", interner nicht veröffentlichter Bericht, FORSOFT A4, 1998

[Dei98b] Deifel B., "Requirements Engineering for complex COTS", Proceedings of REFSQ'98, Pisa, 1998

[Dei98c] Deifel B., "Theoretische und Praktische Ansätze im Requirements Engineering für Standardsoftware und Anlagenbau", Technische Universität München, Technischer Bericht TUM-I9832, 1998

[Dei99a] Deifel B., "Supporting Reuse and Flexibility in CCOTS Variation Development ", Proceedings of REFSQ'99, Heidelberg, 1999

[Dei99b] Deifel B., "A Model for Version Planning of CCOTS", Proceedings of ICSE-SCE'99, Los Angeles, 1999

[Dei99c] Deifel B., "A Process Model for Requirements Engineering of CCOTS", Proceedings of DEXA-REP'99, IEEE, Florenz, 1999

[Dij76] Dijkstra E., "A Discipline of Programming", Prentice Hall, 1976

[DKW99] Derniame J.-C., Kaba B.A., Wastell D., Software Process: Principles, Methodology, and Technology, LNCS 1500, Springer, 1999

[DoT90] Dorfmann M., Thayer R., "Standards, Guidelines and Examples on System and Software Requirements Engineering", IEEE Computer Society Press - Tutorial, 1990

[DP01] Deutsche Post AG, "Direkt Marketing Monitor Studie 10 – Direkt Marketing Deutschlang 1999", Marktstudie von Infratest Burke InCom, München, 2001

[DPa] Deutsche Post AG, "Infopost und Kataloge national", Produktinformation der Deutschen Post AG

[DPb] Deutsche Post AG, "Infobrief und Kataloge national", Produktinformation der Deutschen Post AG

[DS99] Deifel B., Salzmann C., "Requirements and Conditions for Dynamics in Evolutionary Software Systems", Proc. Int. Workshop Princ. Software Evolution, IWPSE99, Fukuoka, 1999

[DSV99] Deifel B., Schwerin W., Vogel S., "Work Products for Integrated Software Development", Technische Universität München, Technischer Bericht TUM-I9921, 1999

[Dub98] Dubois E., "ALBERT: a formal language and its supporting tools for Requirements Engineering", Proceedings of FASE'98, LNCS 1382, Springer Verlag, Lisboa, 1998.

[FOR97] Forschungsverbund FORSOFT, Teilprojekt A4, Diskussionen mit Industriepartnern und Mitarbeitern unterschiedlicher Teilprojekte, 1997-2000

[Fin94] Finkelstein A., "Requirements Engineering: a review and research agenda", APSEC '94, pp. 10-19, 1994

[FiE00] Finkelstein, A. & Emmerich, W., "The Future of Requirements Management Tools" in Information Systems in Public Administration and Law, Quirchmayr, G., Wagner R. & Wimmer M. (eds) (Oesterreichische Computer Gesellschaft), 2000.

[Fow96] Fowler M., "Analysis Patterns : Reusable Object Models", Addison Wesley, 1996.

[GFA97] Griss M., Favaro J., d'Alessandro M., "Featuring the Reuse Driven Software Engineering Business", Object Magazine, 1997

[GHJ+96] Gamma E., Helm R., Johnson R., Vlissides J., "Entwursmuster - Elemente wiederverwendbarer objektorientierter Software", Addison-Wesley, 1996

[GKY99] Griswold W.G., Kato Y., Yuan J., "Aspect Browser: Tool Support for Managing Dispersed Aspects", First workshop on multi-dimensional separation of concerns in object-oriented systems, Denver, 1999.

[GoF94] Gotel O., Finkelstein A., "An Analysis of the Requirements Traceability Problem", ICRE'94, Colorado Springs, 1994

[GoL93] Goguen J. A., Linde C., "Techniques for Requirements Elicitation", RE '93, pp. 152-164, San Diego, Ca, 1993

[Got95] Gotel O, "Contribution Structures for Requirements Traceability", Ph. D. Thesis, University London, 1995

[Gra89] Grahm D.R., "Incremental development: Review of nonmonolithic lifecycle development models", Inf. and Software Technol., vol. 31, no. 1, Jan. 1989.

[HaC88] Hauser J. R., Clausing D., "The House of Quality", The Harvard Business Review, May-June, No. 3, pp 63-73, 1988

[Her98] Hermodsson K., "Core Functionality of Requirements Engineering Tools", Master Thesis, Lund University, Schweden, 1998.

[HHW99] Heimbinger D., Hall R.S., Wolf A.L., "A Framework for Analyzing Configurations of Deployable Software Systems", Proc. Fifth ICECCS, Las Vegas, 1999.

[HLN+90] Harel D., Lachover H., Naamad A., et. al., "Statemate: A Working Environment for the Development of Complex Reactive Systems", IEEE Trans. Soft. Eng., Vol. 16, No. 4, Apr. 1990

[HLS+98] Hamer P., van der Linden F., Saunders A., te Sligte H. "An Integral Hierarchy and Diversity Model for Describing Product Family Architectures", ARES II, LNCS 1429, Las Palmas, Spain, 1998

[Hsi+94] Hsia P. et. al. "Formal Approach to Scenario Analysis", IEEE Software, S. 33- 49, März 1994

[HuS97] Huber F., Schätz B., "Rapid Prototyping with AutoFocus", in Formale Beschreibungstechniken für verteilte Systeme, GI/ITG Fachgespräch 1997, pp. 343-353, GMD Verlag, 1997

[Jac+92] Jacobson I. et. al. "Object-Oriented Software Engineering - A Use Case Driven Approach", Addison Wesley, 1992

[JGJ97] Jacobson I., Griss M., Jonsson P., "Software Reuse", Addison Wesley Longman, 1997

[JHK+97] Jones D., Hollenbach F., Kar P.C., Bell M., van Gaasbeek J.R., Ellinge R.S., "Interfacing Requirements Management Tools In The Requirements Managament Process – A First Look", Proceedings of the 7th Int. Symp. of the INCOSE – Vol. II, August, 1997.

[Jun98] Jung H.W., "Optimizing Value and Cost in Requirements Analysis", IEEE Software, July/August, 1998

[KaR97] Karlsson J., Ryan K., "A Cost-Value Approach for Prioritizing Requirements", IEEE Software, 14(5), 1997

[Kar97] Karlsson, J., "Managing Software Requirements Using Quality Function Deployment", Sofware Quality Journal, 6 (4), 1997

[KCH+90] Kang K., Cohen S., Hess J., et. al. "Feature-Oriented Domain Analysis Feasibility Study", Tech. Rep. CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburg, 1990

[KLM+97] Kiczales G., Lamping J., Mendhekar A., et. al., "Aspect-Oriented Programming", ECOOP'97, Springer, 1997

[KoM93] Koskimies K., Mäkinen E., "Inferring State Machines from Trace Diagrams", Report A-1993-3, Univ. of Tampere, 1993

[Kot97] Kotler P., "Marketing Management - Analysis Planning, Implementation and Control", Prentice Hall, 1997

[Krü00] Krüger I., "Distributed Systems Design with Message Sequence Charts", Dissertation, Technische Universität München, München, 2000.

[KS98] Kotonya, G. , Sommerville, I., "Requirements engineering - processes and techniques", Wiley, 1998.

[KuR70] Kunz, W. and Rittel, H. W. , "Issues as elements of information systems", Technischer Bericht 0131, Universität Stuttgart, Institut für Grundlagen der Planung, 1970.

[LAK+95] Luckham D.C., Augustin L.M., Kenney J.J., Vera J., Bryan D., Mann W., "Specification and analysis of system architecture using Rapide", IEEE Transactions on Software Engineering, 21(4):336–355, 1995.

[Lam98] Lam W., "A case-study of requirements reuse through product families", Annals of Software Engineering 5, pp. 253-277, 1998

[Leh98] Lehmann M.M., "Software's Future: Managing Evolution", IEEE Software, Jan/Feb, 1998

[LiH93] Link J. Hildebrand V., "Database Marketing und Computer Aided Selling", München, 1993.

[LiH95] Link J./Hildebrand V., "Wettbewerbsvorteile durch kundenorientierte Informationssysteme"; Journal für Betriebswirtschaft; 1/1995, S. 46 – 62, 1995.

[MaL97] Massonet P., van Lamsweerde A., "Analogical Reuse of Requirements Frameworks", Proceeedings RE'97, 1997

[MDEK95] Magee J., Dulay N., Eisenbach S., and Kramer J., "Specifying distributed software architectures", ESEC'95, 1995.

[Mil87] Mills H., Dyer M., Linger R., "Cleanroom software engineering", IEEE Computer, pp. 19-25, Sept. 1987.

[MK96] Magee J. and Kramer J., "Dynamic structure in software architectures", Proceedings of the Fourth ACM SIGSOFT Symposium on the Foundations of Software Engineering, San Fransisco, 1996.

[NKF94] Nuseibeh B., Kramer J., Finkelstein A., "A Framework for Expressing the Relationships Between Multiple Views in Requirements Specification", IEEE Trans. SE, 20 (10), 760-773, Oct., 1994

[OMG98] OMG, "The Common Object Request Broker: Architecture and Specification", rev. 2.2, 1998

[Pae98] Paech B., "The Four Levels of Use Case Description", REFSQ'98, Pisa, 1998.

[Par72] Parnas D.L., "On the Criteria To Be Used in Decomposing Systems into Modules", Comm. ACM, Vol. 15, Num. 12, Dec. 1972.

[Par94] Parnas D.L., "Software Aging", ICSE'94, Sorrento, 1994.

[Par98] Partsch H., "Requirements-Engineering systematisch – Modellbildung für softwaregestützte Systeme", Springer-Verlag, 1998.

[Pfe93] Pfeifer T., "Qualitätsmanagement. Strategien, Methoden, Techniken", Hanser Verlag, München - Wien, 1993

[Poh96] Pohl K., "Requiremens Engineering: An Overview", Aachener Informatik-Berichte, Nr. 96-05, RWTH Aachen, 1996

[PoW97] Pomberger G., Weinreich R., "Qualitative und quantitative Aspekte prototypingorientierter Software-Entwicklung - Ein Erfahrungsbericht", Informatik Spektrum, 20, 33-37, 1997

[PPS92] Pomberger G., Pree W., Stritzinger A., "Methoden und Werkzeuge für das Prototyping und ihre Integration", Informatik Forsch. Entw., 7, 49-61, 1992

[PS85] Preparata F. P., Shamos M. I., "Computational Geometry – An Introduction", Springer, 1985.

[PTA94] Potts C., Takahashi K., Antón A., "Inquiry-Based Requirements Analysis", IEEE Software, S. 21- 32, März 1994

[Rak97] Rakitin S. R., "Software Verification and Validation - A Practioner's Guide", Artech House, 1997

[RBC+96] Rolland C., Ben Achour C., Rlyté J. et. al., "A Proposal for a Scenario Classification Framework", CREWS Report 96-01, http://SunSITE.Informatik.RWTH-Aachen.DE/CREWS/reports.htm#1996, 1996

[RBP+91] Rumbaugh J., Blaha M., Premerlani W., et. al., "Object-Oriented Modeling and Design", Prentice Hall, 1991

[REFSQ'99] "Fifth International Workshop on Requirements Engineering: Foundation for Software Quality", Heidelberg, Juni 1999

[REFSQ'00] "Sixth International Workshop on Requirements Engineering: Foundation for Software Quality", Stockholm, Juni 2000

[ReR98] Regnell B., Runeson P., "Combining Scenario-based Requirements with Static Verification and Dynamic Testing", REFSQ'98, Pisa, 1998.

[RGS99] Rayson P., Garside R., Sawyer P., "Recovering Legacy Requirements", Proceedings of REFSQ'99, Heidelberg, 1999.

[RiW73] Rittel, H. W. J. and Webber, M. M., "Dilemmas in a general theory of planning. Policy Sci-ences, S. 155–169, 1973.

[RJB99] Rumbaugh J., Jacobson I., Booch G., "The Unified Modeling Language Reference Manual". Addison-Wesley, 1999.

[RKW94] Regnell B., Kimbler K, Wesslén A., "Improving the Use Case Driven Approach to Requirements Engineering", ICRE'94, 1994

[Rob97] Robertson S., "Requirement Patterns via Events/Use Cases", http://www.atlsysguild.com/Site/Suzanne/Requirements_Patterns, London, 1997

[Roc75] Rochkind M.J., "The Source Code Control System", IEEE Trans. on Software Engineering, SE-1(4), S. 364-370, 1975.

[Rol94] Rolland C., "A Contextual Approach For The Requirements Engineering Process ", 6th International Conference on Software Engineering and Knowledge Engineering (SEKE'94), Vilniuss, Lithuenia, September 1994.

[Rud94] Rudisch, R., "Software Configuration Management in der objektorientierten Systementwicklung", Diplomarbeit, Institut für Informatik, Johannes Kepler Universität Linz, 1994.

[Rum96] Rumpe B., "Formale Methodik des Entwurfs verteilter objektorientierter Systeme", Dissertation, Technische Universität München, 1996

[RUP90] Ramamoorthy, C. V., Usuda, Y., Prakash A., et. al., "The evolution support environment system", IEEE Transactions on Software Engineering, 16 (11), 1225-1234, Nov., 1990

[Saa80] Saaty T.L., "The Analytic Hierarchy Process", McGraw Hill, Inc., 1980

[SG96] Shaw M., Garlan D., "Software Architecture – Perspectives on an emerging discipline", Prentice Hall, 1996.

[SHB96] Schätz B., Hußmann H., Broy M.,"Graphical Development of Consistent System Specifications", in FME'96: Industrial Benefit and Advances In Formal Methods, S. 248-267, M. Woodcook (ed.), Springer, 1996

[Som+93] Sommerville I. et. al., "Integrating ethnography into the requirements engineering process", RE '93, pp. 165-173, San Diego, Ca, 1993

[Som96] Sommerville I., "Software Engineering", Addison-Wesley, 1996

[SS97a] Sommerville I., Sawyer P., "Requirements Engineering – A Good Practice Guide", Wiley, 1997.

[SS97b] Serfling K., Schulte R., "Target-Costing - Kundenorientierung in Kostenmanagement und Preiskalkulation", in Männel, W. (Hrsg.) Frühzeitiges Kostenmanagement, Wiesbaden 1997.

[TOH+99] Tarr P., Ossher H., Harrison W., Sutton S.M., "N Degrees of Separation: Multi-Dimensional Separation of Concerns", ICSE´99, Los Angeles, 1999

[Tra95] Traz, W. "DSSA: Pedagogical Example", ACM Software Engineering Notes, pp. 49-62, Jul. 1995

[Tic85] Tichy W.F., "RCS – A System for Version Control", Software – Practice and Experience, 15(7), S. 637-654, 1985.

[VMod] IABG, "Das V-Modell", WWW page http://www.v-modell.iabg.de/, 1998.

[WBM99] Walker R.J., Baniassad E.L.A., Murphy G.C., "An Initial Assessment of Aspect-oriented Programming", ICSE´99, Los Angeles, 1999

[Wes96] Westfechtel B., "A Graph-Based System for Managing Configurations of Engineering Design Documents", International Journal of Software Engineering and Knowledge Engineering, 6(4):549--583, 1996.

[Wie95] Wieringa, R., "An Introduction to Requirements Traceability", TR IR-389, Vrije Universiteit, Amsterdam, 1995

[Wie99] Wiegers K.E., "Software Requirements", Microsoft Press, 1999.

[Wie99a] Wiegers K.E., "First Things First: Prioritizing Requirements", Software Develompent, Sep. 1999

[Wie99b] Wiegers K.E., "Software Requirements", Microsoft Press, 1999

[Wil97] Wildemann H., "Leitfaden Variantenmanagement - Leitfaden zur Komplexitätsbeherrschung", Technische Universität München, München, 1997

[Wil99] Wildemann H., "Einkaufspotentialanalyse", TCW-Verlag München 1999

[WK00] Wildemann H., Kersten W., "Kundenorientiertes Management der Software-Entwicklung", Zeitschrift für Betriebswirtschaft, 2000.

[Zul92] Zultner R., "Quality Function Deployment (QFD) for Software". In Total Quality Management for Software, Ed. Shulmeyer G, McManus J., New York, 1992