Institut für Informatik
der Technischen Universität München

Konstruktion modularer Vorgehensmodelle

Methodisches Erstellen und Pflegen von Entwicklungsstandards und Vorgehensmodellen für Prozessingenieure

Marco Kuhrmann

Institut für Informatik
der Technischen Universität München

Konstruktion modularer Vorgehensmodelle

Methodisches Erstellen und Pflegen von Entwicklungsstandards und Vorgehensmodellen für Prozessingenieure

Marco Kuhrmann

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. Florian Matthes
Prüfer der Dissertation:	
1.	Univ.-Prof. Dr. Dr. h.c. Manfred Broy
2.	Univ.-Prof. Dr. Andy Schürr, Technische Universität Darmstadt

Die Dissertation wurde am 13. Dezember 2007 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 20. Mai 2008 angenommen.

Kurzfassung

Die Definition und Einführung eines Vorgehensmodells ist ein anerkanntes Verfahren zur Etablierung von effizienten Projektstrukturen und deren kontinuierlicher Verbesserung. Vorgehensmodelle helfen, die vielfältigen Aufgaben bei der Entwicklung komplexer Software zu strukturieren. Durch die Beschreibung bewährter Praktiken in strukturierter Form und die Beschreibung zu erstellender Ergebnisse unterstützen sie Mitarbeiter bei der Einplanung und Durchführung von Projektaufgaben. Viele Unternehmen haben daher bereits Vorgehensmodelle eingeführt, beziehungsweise sind bestrebt, ihre Prozesse zu definieren und in Form eines Vorgehensmodells niederzuschreiben. Dabei entstehen neben etablierten Standardprozessen wie dem V-Modell XT auch vielfältige individuelle, firmeninterne Vorgehensmodelle.

Aufgrund der Breite der verfügbaren Vorgehensmodelle ist in vielen Bereichen eine Harmonisierung der Prozesse erforderlich. Vorgehensmodelle werden daher in vielen Situationen organisations- oder projektspezifisch angepasst. Diese Variantenbildung verläuft in weiten Bereichen nicht ausreichend strukturiert. Bei der Aktualisierung der Ausgangsprozesse, zum Beispiel durch eine neue Version des Prozesses, entstehen hier zwangsläufig Konsistenzprobleme zwischen dem (neuen) Ausgangsmodell und den eingesetzten Varianten. Verschiedene Lösungsansätze aus Forschung und Industrie versuchen, die entstehenden Probleme mit ingenieurmäßigen Grundsätzen zu lösen. Eine weit gehende Modularisierung von Vorgehensmodellen mit definierten Anpassungs- und Pflegeprozessen wird als viel versprechend angesehen. In Anlehnung an Produktlinienansätze wird deshalb auch im Bereich der Vorgehensmodelle angestrebt, *Entwicklungsstandards* zu etablieren. Sie fassen eine Menge bewährter Konzepte und Inhalte zusammen und stellen die Grundlage für eine *Prozess-Plattform* dar. Bereitgestellte Komponenten der Plattform werden durch verschiedene Versionen und Varianten gemeinsam genutzt.

Kernthema dieser Arbeit ist die *methodische Konstruktion* von Vorgehensmodellen im Kontext von Entwicklungsstandards für die Software-Entwicklung. Wir beschreiben dazu ein Konzept zur *konfigurationsbasierten* und *modularen* Entwicklung von Vorgehensmodellen auf der Grundlage einer standardisierten Plattform. Die Plattform enthält einerseits *Strukturkonzepte* in Form eines Vorgehensmetamodells, andererseits *Lebenszykluskonzepte* in Form eines Prozessmodells, das die Entwicklung und Pflege von Vorgehensmodellen beschreibt. Diese Teile führen wir im *integrierten Modellierungsansatz für Vorgehensmodelle* (IMV) zusammen und bieten somit Prozessingenieuren eine methodische Unterstützung für die Erstellung und Pflege von Vorgehensmodellen.

Physisch eigenständige *Prozesskomponenten* stellen die Grundlage dieser Arbeit dar. Sie definieren die Einheiten zur Bearbeitung von Prozessinhalten und ermöglichen die strukturierte Beschreibung von Teilprozessen. Weiterhin sind sie die Einheiten der Anpassung und der Verteilung. In dieser Arbeit beschreiben wir Prozesskomponenten präzise im Rahmen eines Metamodells. Das Metamodell ist struktureller Kern einer *konfigurationsbasierten* Plattform. Auf den Strukturanteilen aufbauend geben wir Prozesse an, die die Entwicklung und Pflege von Prozesskomponenten und Vorgehensmodellvarianten im Kontext der Prozess-Plattform beschreiben. Sowohl das strukturelle Metamodell als auch die Prozesssammlung sind flexibel erweiterbar.

Die Ergebnisse werden anhand verschiedener Beispiele präsentiert und auf Anwendbarkeit hin überprüft. Durch das V-Modell XT, das als Referenz für die Beispiele dient, steht auch ein Bezugsrahmen zur Einordnung zur Verfügung.

Abstract

The definition and implementation of a development process is an accepted approach to arrange efficient project structures and to optimize them, continuously. Development processes help to structure the manifold tasks in a software development project. They describe best practices and deliverables to support people to schedule and carry out tasks. For this reason many enterprises have already introduced a process or evaluate an introduction. Beside standard process models like the V-Modell XT, many individual and organization-specific processes are developed.

A harmonization is often required because of the number of available processes. Hence, processes are often tailored or customized to fit the needs of an organization or a project. The creation of such variants is often not sufficiently structured. For instance, if the original process is updated due to a new release, consistency-problems may occur. Several approaches from research and industry try to answer such problems using engineering principles. An extensive modularization of a development process and defined variation- and maintenance processes are considered being promising. Referring to product line approaches one tries to establish *development process standards* for development process model, too. Such standards contain well-proven concepts and contents, and build the foundation of a *process-platform*. Process components that are published on such a platform are consequently reused by several releases and variants of a process.

The main topic of this thesis is the *methodical design* of process models and development process standards. We describe a concept for a configuration-based, *modular* development based on a common process-platform. This platform consists of a metamodel and a life cycle model describing development and maintenance of process models. We present the *Integrated Modeling-Approach for Process Models* to support process engineers.

The bases of the thesis are self-contained *process components*. They define units for development, structuring, variation and deployment of process contents. We provide a metamodel that precisely describes process components as the core of a *configuration-based* platform. Furthermore we describe a set of processes for the development and maintenance of process components and process model variants. Results of this thesis are presented and validated by means of samples from the V-Modell XT context.

Danksagung

Mein herzlicher Dank gilt zuerst Prof. Dr. Dr. h.c. Manfred Broy, dass er mir diese Arbeit im sehr freien und aufregenden Umfeld seines Lehrstuhls ermöglicht hat. Die großen Freiräume, die mir in meiner Arbeit zur Verfügung standen, trugen nicht zuletzt dazu bei, nie den Bezug zur Praxis zu verlieren. Ebenso herzlich möchte ich mich bei Prof. Dr. Andy Schürr für seine hilfreichen Kommentare zur UML 2 und die Übernahme des Zweitgutachtens bedanken.

Einen nicht unwesentlichen Anteil am Entstehen dieser Arbeit hat das V-Modell XT-Entwicklungsprojekt *WEIT*, das nicht nur Anregungen lieferte, sondern auch die Möglichkeit zur Erprobung von Ergebnissen. Mein Dank gilt hier dem Projektleiter Prof. Dr. Andreas Rausch sowie dem Projektteam: Thomas, Ulrike, Jan, Marc, Klaus, David, Gernot und all den anderen. Mit euch waren immer sehr fruchtbare Gespräche rund um Vorgehensmodelle möglich, was es mir erlaubte, viele Facetten in meinen Überlegungen zu berücksichtigen. Katharina Spies, Norbert Diernhofer, Jens Calamé und Cindy Werder gilt mein besonderer Dank für die Durchsicht des Manuskriptes und viele hilfreiche Gespräche und Hinweise. Auch Jürgen Münch und seinem Team vom Fraunhofer IESE sei an dieser Stelle für die interessanten Diskussionen gedankt.

Nicht zuletzt möchte ich meiner Familie und meinem Freundeskreis danken, die in den letzten Jahren trotz großer Entfernungen immer hinter mir standen. Danke.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

In heutigen IT-Landschaften findet sich eine Vielzahl verschiedener Hard- und Softwaresysteme. Diese Systeme sind üblicherweise funktional stark integriert aber physisch verteilt, womit komplexe, heterogene Systemstrukturen entstehen. Soll ein neues System auf der Basis einer solchen, komplexen Infrastruktur erstellt beziehungsweise in eine solche integriert werden, sind in vielen Bereichen Spezialkenntnisse erforderlich. Somit entstehen auch Projektteams, die genau so heterogen und verteilt sind, wie die Systeme, die entwickelt und gepflegt werden sollen. Die Technologien für die Entwicklung und Bereitstellung solcher Software sind heute in weiten Bereichen vorhanden und auch etabliert. Trotzdem ist die Entwicklung von Software in einer hoch integrierten, heterogenen Umgebung immer noch aufwändig und risikobehaftet.

Der eigentliche Problembereich wird nicht durch die Technologie gegeben, sondern liegt vielmehr im *Entwicklungsprojekt*. Die Entwicklung von Software ist aufwändig. Anforderungen müssen erfasst und so strukturiert und aufbereitet werden, dass eine Systemarchitektur ermittelbar ist. Diese muss durch eine Software implementiert, getestet, verteilt und üblicherweise in regelmäßigen Abständen fehlerbereinigt werden. Dies alles sind für sich bereits komplexe Aufgaben. In Software-Entwicklungsprojekten sind sie darüber hinaus auch noch sinnvoll zu strukturieren und aufeinander abzustimmen.

Eine Zeit lang war der Besorgnis erregende Trend zu erkennen, dass die Relation zwischen erfolgreichen und nicht erfolgreichen Software-Entwicklungsprojekten in einem ungünstigen Verhältnis stagnierte [Int04, BEJ06]. Da die Technologien eigentlich beherrscht wurden und werden, sind die Ursachen zumeist an anderer Stelle zu suchen. In komplexen Projektsituationen, in denen heterogene, verteilte Teams an einem komplexen Produkt arbeiten, ist sehr viel Sorgfalt bei der *Projektorganisation*, der *Durchführung* und der *Überwachung* notwendig. Definierte Software-Entwicklungsprozesse, so genannte *Vorgehensmodelle*, bieten hier eine Möglichkeit, Entwicklungsprojekte sinnvoll zu strukturieren. Sie ergänzen somit technologisches Know-How um *methodisches*. Wie eine neuere Umfrage [FK07] zeigt, haben viele Unternehmen dies bereits erkannt und Software-Entwicklungsprozesse eingeführt beziehungsweise sind zunehmend bestrebt, solche zu etablieren.

Die Definition und Einführung eines Vorgehensmodells ist als Mittel zur Etablierung von effizienten Projektstrukturen und deren kontinuierlicher Verbesserung anerkannt. Die angestrebte Effizienzsteigerung in der Software-Entwicklung, stellt eine wichtige Komponente zur Sicherung der Wettbewerbsfähigkeit deutscher Unternehmen dar. Aber gerade für kleine und mittelständische Dienstleister, die viele Kunden bedienen, wird es schwieriger, alle möglichen, geforderten Prozesse vorzuhalten und zu *beherrschen*. Ein Weg, der hier zum Beispiel beschritten wird, ist die Definition eines *internen* Vorgehensmodells, das mit den verschiedenen durch die Auftraggeber geforderten Prozesse gekoppelt werden kann. Zunehmend gewinnen hier *Standardvorgehen* (Entwicklungsstandards), wie zum Beispiel Prince 2 [Köh06] oder das V-Modell XT [RH06], als Grundlage für die Interoperabilität von Vorgehensmodellen an Bedeutung.

Ist ein Vorgehensmodell einmal eingeführt, muss es kontinuierlich gepflegt und den aktuellen Anforderungen des Marktes angepasst werden. Da es aber sehr viele Vorgehensmodelle (teilweise auch in verschiedenen Ausprägungen und Versionen) gibt, gestaltet sich die Pflege eines modernen und zu allen Partnern konsistenten Vorgehensmodells schwierig. An dieser Stelle sind wiederum die Entwicklungsstandards interessant, die Ideen einer *Prozessplattform* für Vorgehensmodelle umsetzen. Die Definition solcher Standards ist aber ebenso aufwändig wie die Entwicklung einer komplexen Software. Die Ableitung von organisations- und projektspezifischen Varianten sowie deren effiziente Pflege im Umfeld von Entwicklungsstandards wirft noch viele Fragen auf.

1.1. Vorgehensmodelle in der Softwareentwicklung

Seit vielen Jahren entstehen vielfältige *Vorgehensmodelle*, durch die verschiedene Bestandteile ingenieurmäßigen Vorgehens strukturiert und geeignet kombiniert werden. Vorgehensmodelle finden sich vielfach als Teil verschiedener *Software Engineering* Disziplinen wieder. Software Engineering bezeichnet die Erkenntnis, dass Software analog zu etablierten Ingenieurdisziplinen (zum Beispiel Maschinenbau, Architektur im Bauwesen) erstellt werden sollte. Die Ziele entsprechen sich in weiten Bereichen:

- Erfüllung von Kundenwünschen (Anwenderforderungen, Anforderungen)
- Qualitätsansprüche (zum Beispiel Robustheit, Zuverlässigkeit, ...)
- Effiziente Aufgabenerfüllung (mit dem erstellten Produkt)
- Termintreue bei der Leistungserbringung/Fertigstellung
- Kostengünstige Erstellung
- Wiederverwendung von Bewährtem
- ...

Durch die Anwendung ingenieurmäßiger Methoden soll zum Beispiel kosteneffiziente Entwicklung gewährleistet werden oder es stehen a priori Vorgaben zur Güte der entstehenden Produkte zur Verfügung. Dem Anwender wird hier eine weit reichende Unterstützung angeboten. Chroust [Chr92], Kneuper et al. [KMLO98] beziehungsweise Fritzsche/Keil [FK07] listen eine Auswahl bekannter Vorgehensmodelle auf.

Vorgehensmodelle basieren inhaltlich auf einem Erfahrungsschatz, der in vielen Projekten gesammelt wurde. Sie beschreiben Mengen von Projekten beziehungsweise ein gültiges, angemessenes Vorgehen für ein konkretes Projekt. Anwender werden durch Vorgehensmodelle dabei unterstützt, einmal gemachte Fehler nicht zu wiederholen oder besonders bewährte Praktiken zielführend anzuwenden. Moderne Vorgehensmodelle gehen oft noch weiter und sind integraler Bestandteil verschiedener Werkzeuge [AEH+07, KK07]. Diese bieten allen Beteiligten eines Projekts Unterstützung bei ihren täglichen Aufgaben an. Organisationen wird durch die Anwendung von Vorgehensmodellen jedoch mindestens Unterstützung bei der Regelung von Koordination und Kommunikation in Projekten angeboten. Auf diese Weise werden Schnittstellen zwischen (allen) Beteiligten definiert, die zum Beispiel die Kommunikation oder die Verantwortlichkeiten für Lieferungen und Übergaben festlegen. Diese sind erforderlich, um Aufgaben effizient im Gesamtkontext des Projekts (auch in einem globalen Rahmen [SMP06]) auszuführen.

1.1.1. Erstellung und Anpassung von Vorgehensmodellen

Viele Organisationen verfügen über eigene Vorgehensmodelle, die den eigenen Anforderungen besser genügen, als ein Standardvorgehensmodell. Beispielhaft seien hier das Microsoft Solutions Framework (MSF, [Tur06]) oder der oose Engineering Process (OEP, [OSKZ06]) genannt, die von den betreffenden Firmen aber auch nach verschiedenen Modellen (Beratung, Produkt, integrierendes Werkzeug, ...) vertrieben werden. Standardmodelle wie das deutsche V-Modell XT [RH06] oder das britische Prince 2 [Köh06] sind hier allgemeiner, bieten aber oft Möglichkeiten zur Anpassung für Organisationen an. Diese Fähigkeiten sind beispielsweise beim V-Modell unerlässlich, da es in der standardmäßig vorliegenden Form streckenweise so allgemein ist, dass eine unmittelbare Anwendung nur eingeschränkt möglich ist.

Die vorliegende Arbeit ist wesentlich durch die Erfahrungen während der intensiven Betreuung und Schulung der (Pilot-)Anwender des V-Modells und im weiteren Verlauf des WEIT[1]-Projekts auch durch die Mitwirkung bei der Erstellung des V-Modell XT 1.2Bw[2] entstanden. Die Erfahrungen, die während der Betreuung der Einführungs-, Entwicklungs- und Anpassungsprojekte gemacht wurden [KNB05, Kuh05,

1 Entwicklungsprojekt des V-Modell XT. Analog zur Dokumentation des V-Modell XT bezeichnen wir die aktuelle Version auch kurz als V-Modell.

2 Variante des V-Modell XT mit Erweiterungen für die Anwendung in der Bundeswehr.

KT06, Kuh07, AEH+07], zeigen Handlungsbedarf: Die Erstellung eines organisationsspezifischen Vorgehensmodells ist aufwändig und erfordert detailliertes Wissen über die Domäne auf der einen und die verfügbaren Optionen zur Anpassung auf der anderen Seite. Zu Beginn der breiten Einführung des V-Modells wurden oftmals – in Ermangelung konkreter Erfahrungen – pragmatische Lösungen erstellt. Diese Lösungen entsprechen den Anforderungen zum Zeitpunkt der Erstellung, sind jedoch im aktuell entstehenden V-Modell Konformitätsprogramm neu zu bewerten.

Nicht immer ist eine Organisation bereit (oder in der Lage), einen langwierigen, kostenintensiven Prozessverbesserungs- und Einführungsprozess zu etablieren. Die *unmittelbare* Anwendbarkeit eines Vorgehensmodells („out-of-the-box") ist daher von den Anwendern oftmals explizit erwünscht. Aus diesem Grund bietet beispielsweise das V-Modell XT eine weitere Anpassungsstufe auf der Ebene von Einzelprojekten an. Andere Vorgehensmodelle bieten zumindest ähnliche Optionen (zum Beispiel EPF siehe [Hau06a, Hau06b] oder MSF [Tur06]) beziehungsweise eine entsprechende Werkzeugunterstützung nebst Anleitungen an.

1.1.2. Weiterentwicklung und Evolution

An vielen Stellen entstehen (unabhängig vom V-Modell) durch verschiedene Anpassungsverfahren *Varianten* eines Vorgehensmodells. Entwurf, Entwicklung und Anpassung von Vorgehensmodellen sind jedoch komplexe Prozesse, die langfristige Planungen und somit verlässliche Architekturen benötigen [MRD+04, Boe05, Hum05, NR05, KHTS07]. Aufgrund der inhärenten Notwendigkeit, Varianten auf verschiedenen Organisations- und Abstraktionsebenen bilden zu können und Weiterentwicklungen berücksichtigen zu müssen, sind vorausschauende Basiskonzepte und Verfahren erforderlich, die auch über mehrere *Versionen* hinweg Pflege- und QS-Maßnahmen gestatten. Wir diskutieren diesen Aspekt im Folgenden am Beispiel des V-Modells.

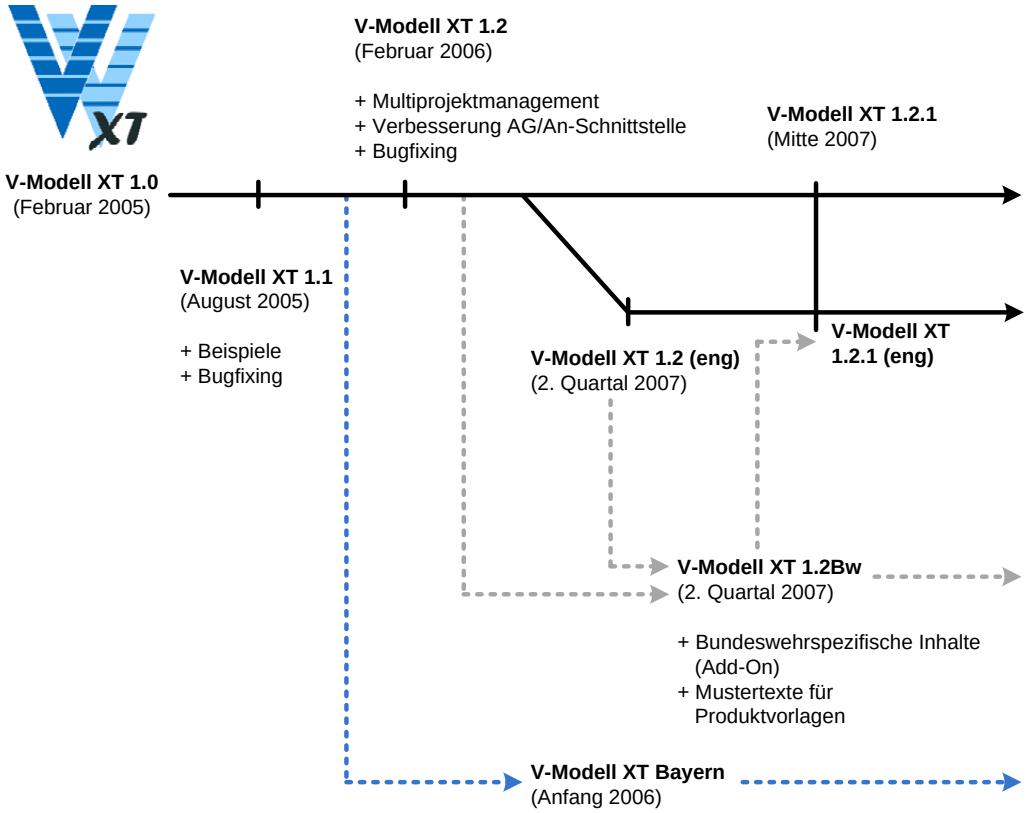


Abbildung 1.1.: Entwicklung, Weiterentwicklung und Derivatbildung des V-Modell XT

Versionen und Varianten des V-Modells. Das Erkennen der Notwendigkeit eines definierten, konstruktiv soliden und langlebigen Entwicklungs- und Pflegeprozesses fand zum Beispiel für das V-Modell statt, als sich abzeichnete, dass verschiedene Organisationen Anpassungen des V-Modells vornahmen[3]. Abbildung 1.1 zeigt einen Ausschnitt des Entwicklungspfades des V-Modells, an dem wir das kurz diskutieren wollen. Bereits im ersten Jahr nach der offiziellen Vorstellung am 4. Februar 2005 wurde mit der Derivat-/*Variantenbildung* begonnen. Abbildung 1.1 zeigt im oberen Teil den Hauptstrang der Entwicklung und darunter liegend die Varianten. Hier zu sehen sind die Varianten V-Modell Bayern und das V-Modell der Bundeswehr. Ende 2006 stellten sich bereits Fragen wie:

- Wie ist die (allgemeine) Weiterentwicklung des V-Modells sicherzustellen?
- Wie wirken sich Weiterentwicklungen (Versionen, Releases) auf die sich im Feld befindenden Varianten hinsichtlich Verbindlichkeit und Rechtssicherheit aus?
- Wie wird Feedback aus einer Variante oder Instanz dem „Hauptstrang" und allen weiteren Varianten zugänglich gemacht?
- Wie wird die Konformität einer Variante zum „Hauptstrang" sichergestellt?
- Ergänzend zu oben stehenden Fragen: Wie wird dann in Bezug auf Versionen verfahren?

Hinzu kam neben den Anpassungen auch eine zeitlich parallele Weiterentwicklung (*Versionsbildung*), nicht nur im Hinblick auf Umfang und Fehlerbeseitigung, sondern auch in Form einer Englischübersetzung. V-Modell Versionen und Varianten müssen also nicht nur bezüglich der Inhalte, sondern auch hinsichtlich der Sprache(n) konsistent gehalten werden. Gnatz [Gna05] hat in seiner Arbeit bereits eine Schwachstellenanalyse des V-Modells vorgenommen, jedoch mit dem Fokus auf strukturelle Eigenschaften, die sich im Bereich der Plangenerierung auswirken. Im praktischen Einsatz des V-Modells und dessen Anpassung ist jedoch nicht nur ein fehlender Pflegeprozess offensichtlich geworden, sondern auch der *erneute* Einfluss der bereits in [Gna05] festgestellten (strukturellen) Schwachstellen. Die im Bereich der Anpassung und Variantenbildung auftretenden – jedoch noch nicht zufrieden stellend gelösten – Probleme bei der Anwendung des V-Modells motivieren zu dieser Arbeit.

1.1.3. Änderungsbedarf bei der Sicht auf Vorgehensmodelle

In den letzten zwei Jahren wandelte sich die Sicht auf das V-Modell XT schrittweise von der eines Rahmen- und Regelwerks hin zu einem *Produkt*. Die Produktsicht findet sich auch in anderen Vorgehensmodellen [KK03, Gau06, Tur06] wieder, die mehr sind als bloße Anwendungsleitfäden. Beispielhaft beziehen wir uns auf das Microsoft Solutions Framework (MSF), das die Produktorientierung aufgrund der Philosophie von Microsoft von Beginn an konsequent umsetzt. Es definiert basierend auf einem allgemeinen Grundmodell und einem Integrationskonzept für eine Menge von Werkzeugen ein Produkt bestehend aus Softwarekomponenten, Anwendungshilfen und Leitfäden für Software Entwicklungsprojekte. MSF adressiert standardmäßig zwei Kundenkreise, die alle auf dieselben Fähigkeiten der Tools und des Prozessmodells zurückgreifen können. Auf der anderen Seite erhalten die Kunden durch die zwei verschiedenen Ausprägungen des Basismodells schon weitgehend angepasste Volumina der Prozessdefinition und -steuerung.

In direkter Folge können wir für ein Vorgehensmodell Ähnlichkeiten zum Begriff der *Produktlinie* finden. Rombach hat hier den Begriff der *Prozesslinie* (Process Line, [Rom05]) verwendet und einen integrierten Ansatz für Prozess- und Produktlinien vorgeschlagen. Die dahinter liegenden Konzepte (stabile Basiskomponenten, Operationen zur Anpassung und Pflege etc.) bilden einen der Schwerpunkte dieser Arbeit.

3 Zum Beispiel: Witt-Weiden [AEH+07] oder die Bundeswehr [Kuh05]

1.1.4. Lebenszyklus eines Vorgehensmodells

Wenn wir ein Vorgehensmodell als Produkt und Varianten und Versionen eines Vorgehensmodells als Bestandteile einer Produktlinie verstehen, müssen wir auch Punkte berücksichtigen, die typisch für Produkte sind, zum Beispiel:

- Wartung,
- Pflege und
- Weiterentwicklung.

Diese Punkte betreffen aber nicht nur einen primären Entwicklungsstrang, sondern auch Versionen und Varianten, die sich im Einsatz befinden. Die Ähnlichkeiten zum so genannten *Software Lifecycle* sind hier offensichtlich. Auch Vorgehensmodelle durchleben verschiedene Entwicklungs- und Reifungsstufen. Sie werden konzipiert und kontinuierlich an sich wechselnde Gegebenheiten und Bedürfnisse angepasst.

1.2. Ziele und Ergebnisse

Im Fokus dieser Arbeit stehen die im Abschnitt 1.1.2 aufgeworfenen Fragen hinsichtlich der Weiterentwicklung und Pflege sowie der effizienten Aktualisierung existierender Vorgehensmodellanpassungen. Sie dienen als Leitlinie, um eine konstruktive, integrierte Methode zu erarbeiten, die Kontinuität, Stabilität und Langfristigkeit eines Vorgehensmodells sicherstellt. Mit dieser Arbeit legen wir einen formal fundierten Rahmen, der als Basis für die methodische Konstruktion modularer Vorgehensmodelle dient. Die konkrete Ergebnisstruktur schlüsseln wir anhand von Abbildung 1.2 auf. Die folgenden Abschnitte gehen detaillierter auf die Ergebnisse ein.

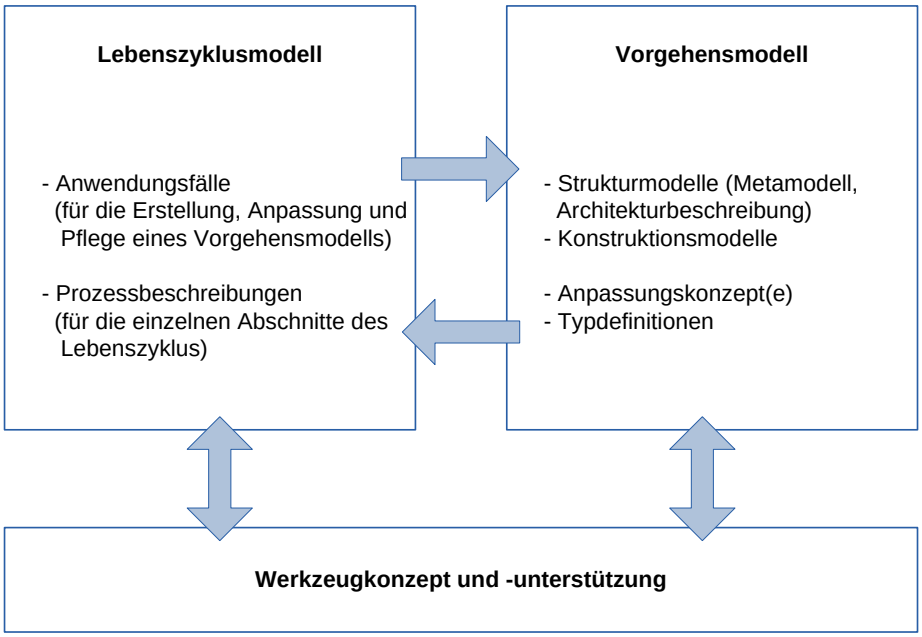


Abbildung 1.2.: Ergebnisstruktur der Arbeit

Die vorliegende Arbeit versteht sich nicht als Ansatz zur Entwicklung eines universellen Vorgehensmodells. Sie liefert Beiträge, die es ermöglichen, bewährte Elemente zu einem Vorgehensmodell flexibel aber dennoch stabil und zukunftssicher zusammenzufügen.

Metamodell eines Vorgehensmodells

Das in dieser Arbeit entwickelte *Vorgehensmetamodell* definiert grundlegende und komplexe *Strukturen* eines Vorgehensmodells. Es basiert auf einer modularen und erweiterbaren Architektur. Es beschreibt beispielhafte Produkt-, Rollen- und Aktivitätsmodelle, sodass es bereits unmittelbar in die Anwendung überführt werden kann. Die entwickelte Architektur enthält ein *konfigurationsbasiertes* Konstruktionsmodell. Dieses ermöglicht die Bildung von *physisch eigenständigen Komponenten*, welche die Modularität in den Dimensionen *Anpassung/Tailoring*, *Entwicklung* und *Verteilung* umsetzen.

Bezug nehmend auf Abschnitt 1.1.2 verallgemeinern wir die aufgestellten Fragestellungen und liefern mit dem Metamodell Antworten hinsichtlich struktureller Eigenschaften und der Erstellung von Versionen und Varianten.

Im Rahmen dieser Arbeit wird neben einer abstrakten Metamodellierung auch auszugsweise die weiter gehende Anwendung der Modellierung auf der Instanzenebene gezeigt. Beispiele weisen die Anwendbarkeit des entwickelten Ansatzes nach.

Lebenszyklusmodell eines Vorgehensmodells

Einen weiteren Beitrag leistet diese Arbeit durch die Definition eines *Lebenszyklusmodells* für Vorgehensmodelle. Mit dem Lebenszyklusmodell verallgemeinern wir ebenfalls die in Abschnitt 1.1.2 aufgestellten Fragen und geben Antworten hinsichtlich Entwicklungs- und Pflegeprozesssen. Das Lebenszyklusmodell basiert auf einem Referenzprozess für Software Produktlinien und enthält *Beschreibungen* einzelner Erstellungs-, Anpassungs- und Pflegeprozesse für Familien von Vorgehensmodellen (*Prozesslinien*). Es beschreibt anhand signifikanter *Anwendungsfälle* Operationen auf Vorgehensmodellen und stellt die Verbindung zum entwickelten Vorgehensmetamodell her.

Das Lebenszyklusmodell ist explizit dazu ausgelegt, die Bildung von Varianten zu unterstützen sowie die Wartung und Pflege zu ermöglichen.

Werkzeugkonzept

Die formale Fundierung auf der Grundlage eines Metamodells und die Verfügbarkeit einer Konstruktionsmethode im Rahmen eines aufbauenden Lebenszyklusmodells legen eine weit reichende Unterstützung durch Werkzeuge nahe. Ein *Werkzeugkonzept* wird ebenfalls im Rahmen dieser Arbeit erstellt.

Adressaten, Positionierung und Einschränkung

Wir konzentrieren uns in dieser Arbeit im Wesentlichen auf Aspekte der Konstruktion eines Vorgehensmodells beziehungsweise einer Menge von Vorgehensmodellvarianten. Wir fokussieren hierbei konstruktiv strukturelle Ansätze und gehen dabei nicht weiter auf konkrete Inhalte einzelner Instanzen ein. Die Ermittlung und das Management von Varianten auf inhaltlicher Ebene überlassen wir zum Beispiel [SM06c, SM06a, SM06b]. Zur inhaltlichen Ausgestaltung zählen wir auch Process Pattern [GMP+03] und konkrete Methoden zum Beispiel zur Softwareentwicklung [Ham08]. Auch diese betrachten wir nicht.

Die Ergebnisse dieser Arbeit dienen primär *Prozessingenieuren*, die ein Vorgehensmodell analysieren/formalisieren, entwickeln oder weiterentwickeln sollen. Optionen zur Übertragung in weitere Phasen des Lebenszyklus, zum Beispiel einer Ausgestaltung im Rahmen eines Projekts, zeigen wir auf; vertiefen sie jedoch nicht. Stellt man beispielsweise wie beim V-Modell ein mehrstufiges Anpassungsverfahren auf, können wir die Inhalte und Ergebnisse der vorliegenden Arbeit sehr präzise zuordnen. Wir betrachten folgende Anpassungs-/*Tailoringstufen*:

Tailoringstufe 1: Diese Stufe bezeichnet einmal den initialen Erstellungsprozess eines Vorgehensmodells (Philosophie, Struktur, Inhalte) beziehungsweise einen projektübergreifenden (organisationsspezifischen) Anpassungsprozess. Aktivitäten, die

der Anpassung und Pflege, somit also der Versions- und Variantenbildung dienen, positionieren wir hier. Diese Stufe ist vollständig formalisierbar.

Tailoringstufe 2: Diese Stufe erfordert ein (organisationsspezifisches) Vorgehensmodell. Dieses wird auf der Ebene eines konkreten Projekts mit konkreten Anforderungen und situationsbezogen angepasst. Die Anpassung erfolgt für jedes Projekt erneut und ist auch nur für dieses gültig[4]. Diese Stufe ist vollständig formalisierbar.

Ausgestaltung: Das Tailoring der Stufe 2 kann in der Regel nur bis zu einem gewissen Grad präzise die Projektanforderungen erfüllen. Deshalb und um im weiteren Verlauf eines Projekts die notwendige Flexibilität zu erhalten, sehen wir die Ausgestaltung eines Projekts auch als Anpassungsstufe. Diese ist jedoch in der Regel nicht beziehungsweise nicht mehr vollständig formalisierbar.

Mit dieser Arbeit ordnen wir uns schwerpunktmäßig in der *Tailoringstufe 1* ein. Damit positioniert sich diese Arbeit auch auf dem Zeitstrahl des Konstruktionsprozesses eines Vorgehensmodells zeitlich vor [Gna05, Ham08] oder [Mün01], die entweder gegebene Vorgehensmodelle voraussetzen oder gezielt Informationen und Strukturen zur Plangenerierung heranziehen. Diese Arbeiten positionieren wir schwerpunktmäßig im Bereich der Tailoringstufe 2 oder sogar noch später. Die Ergebnisse dieser Arbeiten werden jedoch nicht ausgeschlossen und können mit den Ergebnissen der vorliegenden Arbeit kombiniert werden, zum Beispiel im Bereich der Vorgangsmodellierung nach [Gna05] für die automatische Ableitung von Projektplänen.

1.3. Verwandte Arbeiten

Vorgehensmodelle – oder Software-(Entwicklungs-)Prozesse allgemein – gewinnen stetig an Bedeutung. Die bearbeiteten Gebiete zum Beispiel im Bereich der *Software-Prozessmodellierung* sind vielfältig. In diesem Abschnitt gehen wir auf verwandte Arbeiten ein und positionieren diese im Kontext der vorliegenden Arbeit. Verwandtes Material ist im Wesentlichen in zwei Bereichen zu finden. So gibt es einerseits im kommerziellen und industriellen Umfeld Standards und Werkzeuge. Andererseits ist gerade im akademischen, insbesondere im anwendungsorientierten Forschungsbereich ein verstärktes Interesse an Software-Prozessen und deren Modellierung festzustellen.

Werkzeugunterstützte und konkrete Ansätze, Produkte, Standards

Ein bekanntes Vorgehensmetamodell ist das *Software Process Engineering Metamodel* (SPEM, [OMG05]), das als Basis für den Rational Unified Process (RUP, [KK03]) dient. Das so genannte *Eclipse Process Framework* (EPF, [Hau06a, Hau06b, Gau06]), ist ein auf Eclipse aufbauendes Toolset, das einerseits eine eingeschränkte Version des RUP – den Open Unified Process (OpenUP) – anbietet, andererseits jedoch auch mit dem RUP und seiner Werkzeuglandschaft wieder zusammengeführt wird. Speziell OpenUP weist eine Trennung in strukturelle und dynamische Modellanteile auf, die die Modularität und Wiederverwendbarkeit von Einzelelementen steigern sollen. Hervorzuheben an dieser Kombination von Prozess und Produkt ist die vorhandene Möglichkeit, einzelne Module auch physisch eigenständig zu entwickeln. Dabei stehen EPF/OpenUP jedoch hinter den Fähigkeiten des V-Modells zurück, was die internen Strukturen der so genannten Plug-Ins betrifft. Die Kombination aus RUP (OpenUP), EPF und Eclipse als Prozessplattform ist sehr mächtig, jedoch weisen SPEM (und seine Weiterentwicklung in Teilen der Unified Method Architecture (UMA)) ein sehr dichtes Abhängigkeitsgeflecht auf. Was auf der einen Seite eine reichhaltige Unterstützung durch Werkzeuge und Komfort für den Benutzer bietet, stellt sich bei näherer Betrachtung als hinderlich für die Auskopplung, Pflege und (zentrale) Updates von Varianten dar. Wie bereits in Gnatz et al. [GMP+03] festgestellt, eignet sich SPEM eher zur Beschreibung eines konkreten Vorgehensmodells. Varianten- und Versionsbildung wird nicht angemessen unterstützt und steht und fällt mit den Werkzeugen des Prozesses.

4 **Hinweis:** Nur die Stufe 2 wird vom V-Modell XT als Tailoring bezeichnet.

Mit dem *OPEN Process Framework* (OPEN, [GHSY97]) liegt ebenso wie mit SPEM ein Metamodell für Vorgehensmodelle vor. OPEN verfügt über eine Reihe interessanter Konzepte, die wir in dieser Arbeit berücksichtigen (vgl. Kapitel 2.1.4), wird jedoch offensichtlich seit einigen Jahren nicht mehr weiter entwickelt. Eine gewisse Fortführung finden die Konzepte von OPEN im ISO Standard ISO/IEC 24744:2007, der ebenfalls ein genormtes Metamodell für Vorgehensmodelle beschreibt. Er enthält bereits Ansätze zur Modulbildung, jedoch sind weder physische Trennungen noch die weiter gehende Unterstützung eines Lifecycle unmittelbar erkennbar.

Auf derselben Ebene wie SPEM und seine Werkzeuge ist das *V-Modell XT* [RH06] zu positionieren. Auch hier sind die Möglichkeiten für die Werkzeugunterstützung aufgrund seiner formalen Definition mithilfe eines Metamodells [Gna06] stark ausgeprägt. Ähnlich ist auch das mehrstufige, werkzeugunterstützte Anpassungskonzept des V-Modells einzustufen. Jedoch bietet das V-Modell kein physisches Modulkonzept an. Das Modulkonzept des V-Modells ist rein virtuell und findet sich nicht in dessen physischer Repräsentation wieder. Dies führt zu Problemen hinsichtlich Anpassbarkeit und Derivatbildung [KT06]. Da wir das V-Modell trotzdem als eine unserer primären Referenzen heranziehen, detaillieren wir diese Punkte noch weiter (Kapitel 3).

Ähnlich wie die Familie des RUP ist auch das *Microsoft Solutions Framework* (MSF, [Tur06]) ein Produkt, das sich in eine umfangreiche Infrastruktur integriert. Anders als RUP, OpenUP oder V-Modell ist das MSF jedoch nicht in einer generischen Form verfügbar, sondern in der Regel nur im Kontext der Microsoft Entwicklungsumgebungen und hier in Software Entwicklungs-bezogenen Szenarios zu finden. Jedoch bietet das MSF Metamodell interessante Fähigkeiten zur Modellierung weiterer Prozesse und steht somit den SPEM-basierten Prozessen oder dem V-Modell in nichts nach. Hervorzuheben ist die Kompaktheit und weit reichende Werkzeugunterstützung des MSF [Kuh07].

Genauso konkret wie MSF ist der *oose Engineering Process* (OEP, [OSKZ06]). OEP ist hier deshalb zu nennen, weil er als Metamodell-basierter Ansatz Optionen hinsichtlich Modularisierbarkeit, Evolution und Variantenbildung anbietet. OEP ist ein konkreter, aktivitätsgetriebener Prozess, der sehr detailliert ausgearbeitet ist. Oestereich et al. [OSKZ06] geben explizit an, dass OEP ein anderes Anpassungskonzept verfolgt, als beispielsweise das V-Modell oder EPF-basierte Prozesse. Anpassung im Kontext OEP heißt Entfernen nicht benötigter Aktivitäten und Ergänzen eventuell fehlender. OEP ist als Beratungsprodukt einzustufen, dessen Instanziierung somit singulär erfolgt. Im Kontext dieser Arbeit ist dies jedoch nicht relevant. OEP ist nicht werkzeugunterstützt – eine Unterstützung ist auch nicht erkennbar vorgesehen.

Beitrag: Mit dieser Arbeit leisten wir einen Beitrag im Bereich der Modularisierung und Komponentenbildung. Eine Werkzeugunterstützung werden wir zwar konzipieren, jedoch konkurrieren wir nicht mit etablierten Werkzeugen. In dieser Arbeit stellen wir ein (zunächst) werkzeugunabhängiges Konzept vor, mit dem wir physisch eigenständige Komponenten eines Vorgehensmodells modellieren können, andererseits jedoch auch einen komplexen Lebenszyklus unterstützen. Diese Punkte dienen uns als Grundlage, Vorgehensmodelle unter dem Gesichtspunkt der *Plattformbildung*, also im Kontext von Entwicklungsstandards, zu betrachten.

Konstruktive Ansätze, Methoden und Ausgestaltungen

Die Einschränkungen des V-Modell Metamodells stellte Gnatz in [Gna05, GDMR04] bereits fest. Er hat auch entsprechende Anpassungen und Änderungen vorgeschlagen. In [Gna05] fokussiert er diese Änderungen jedoch auf das Ziel, alle notwendigen Informationen einer Vorgehensmodellinstanz für die Ableitung eines Projektplans zu erfassen. Die Planungskomponente (vielmehr die Modellelemente, die zur Planung herangezogen werden) ist im V-Modell XT bereits mehrfach überarbeitet worden, jedoch noch immer nicht zufrieden stellend umgesetzt [Ber06]. Gnatz gibt an, Anpassung, Pflege und Weiterentwicklung mit seiner Arbeit auch zu unterstützen, tut dies jedoch analog zum V-Modell XT nur auf der virtuellen Ebene. Der Nachweis echter Modularität liegt nicht im Fokus seiner Arbeit, dennoch bilden einige seiner Konzepte und Modellierungsansätze ein solide Grundlage, auf der wir auch in Teilen dieser Arbeit aufbauen. Anders

als Gnatz setzen wir in dieser Arbeit auch einen anderen Schwerpunkt bei der Auswahl der Modellierungstechnik. Gnatz priorisiert die Ausdrucksmächtigkeit gegenüber der Einfachheit der Beschreibungsmittel. Wir gehen hier den anderen Weg und favorisieren möglichst einfache Beschreibungsmittel. Die eigentlichen Ziele seiner Arbeit (die Projektplanung und Plangenerierung) betrachten wir in dieser Arbeit indes nicht.
Beitrag: Mit der vorliegenden Arbeit greifen wir den Metamodell-getriebenen Konstruktionsgedanken wieder auf und treiben ihn konsequent weiter. Darüber hinaus definieren wir den Lebenszyklus eines Vorgehensmodells nicht nur ansatzweise, sondern vollständig.

Ebenfalls aufbauend auf dem V-Modell XT befasst sich Hammerschall in ihrer Arbeit [Ham08] mit der Integration von Methoden in Vorgehensmodelle. Der Bedarf entsprechender Integrationskonzepte wird beispielsweise in [KKNT04, Gru05, KT06] deutlich. Sie adressiert einen ausgezeichneten Zeitraum im Lebenszyklus eines Vorgehensmodells, den wir nicht ausdetaillieren. Der grundlegende Ansatz eines Schnittstellenmodells für die Integration in ein gegebenes Vorgehensmodell ist mit den Konzepten dieser Arbeit aber kompatibel. Wir stellen hier jedoch ein umfassenderes Konzept vor, das neben Methoden weitere Prozesselemente adressieren kann. Eine Kopplung beider Ansätze wird nicht ausgeschlossen.
Beitrag: Die Ziele dieser Arbeit liegen im Lebenszyklus zeitlich deutlich vor einer Integration konkreter Methoden und befassen sich mit generelleren Fragestellungen, die eine (transparente) Integration eben dieser erst ermöglichen.

Auf ähnlich (inhaltlich) konkreter Ebene sind die Arbeiten von Münch [MSV97, Mün99, Mün01, Mün04, Mün05] zu positionieren. Auch Münch geht von gegebenen Vorgehensmodellen (oder zumindest Teilen davon) aus und befasst sich mit der Anpassung eines Vorgehensmodells an projektspezifische Anforderungen (Tailoring). Klares Ziel, insbesondere in [Mün01], ist eine optimierte, wiederverwendungsorientierte Erstellung von Projektplänen. Analog zu [Gna05] liegt dies *nicht* im Fokus dieser Arbeit, wenn auch die Modellierungsansätze – insbesondere die Modellierungssprache Multi-View Process Modelling Language (MVP-L) – einige interessante Aufsetzpunkte bieten.

Eine weitere, vergleichsweise neue Prozessbeschreibungssprache bieten Shen/Chen [SC06] mit FLEX an. Diese Sprache ist formal und unterstützt die Strukturmodellierung. Analog zu SPEM und OPEN erscheint FLEX jedoch eher dafür ausgelegt zu sein, ein konkretes Vorgehensmodell beziehungsweise Teile davon zu beschreiben. FLEX erwähnt Submodelle eines Vorgehensmodells. Über die konkrete Implementierung/Umsetzung eines FLEX-basierten Prozesses und die Verfügbarmachung für Endanwender werden jedoch kaum Aussagen getroffen. Auch die Hinweise auf die Evolutionsfähigkeit beziehen sich in der Regel auf konkrete Instanzen.
Beitrag: Wir definieren in dieser Arbeit eine Sprache zur Beschreibung von Vorgehensmodellstrukturen unter besonderer Berücksichtigung der Weiterentwicklung und Anpassung. Den Nachweis der Tragfähigkeit erbringen wir am konkreten Beispiel.

Lebenszyklus und analytische Ansätze

Betrachtet man den Lebenszyklus eines Vorgehensmodells, wird es irgendwann – zum Beispiel im Rahmen einer Weiterentwicklung – erforderlich, Unterschiede und Differenzen zwischen einzelnen Vorgehensmodellinstanzen zu erfassen. Im Kontext dieser Arbeit adressieren wir konstruktiv die Fähigkeiten, bereits auf der Metamodell- und folgend auf der Modellebene Varianten zu erzeugen. Die Identifikation sowie die Ermittlung der Unterschiede auf der Instanzenebene betrachten wir allerdings nicht vollständig. Hier sind Arbeiten im Umfeld von Soto und Münch [SM06c, SM06a, SM06b] zu finden. Zur Δ-Ermittlung konvertieren sie ein gegebenes V-Modell in ein RDF-basiertes XML-Zwischenformat. Sie verwenden zurzeit das V-Modell XT als Fallbeispiel zur Analyse und Δ-Ermittlung. Ähnliche Ansätze zur Differenzbestimmung finden sich zum Beispiel auch bei Kelter [Kel07].
Beitrag: Soto/Münch betrachten die Analyseergebnisse als Beitrag zum Etablieren einer Feedbackschleife für einen Weiterentwicklungsprozess. Wir etablieren mit dieser Arbeit eine Feedbackschleife direkt im Lebenszyklusmodell, da Feedback sich hier unter Umständen auf beliebig viele Instanzen auswirken kann.

Um den Lebenszyklus eines Vorgehensmodells als solches zu betrachten, ist eine entsprechende Grundlage erforderlich. Reifegradmodelle, wie zum Beispiel das Capability Maturity Model Integration (CMMI, [Kne06]) bieten hier Ansatzpunkte. Beispielhafte Beschreibungen und Erfahrungen von Einführungsprozessen sind beispielsweise [GST06, Kom06, AEH+07] zu entnehmen. Nejmeh und Riddle [NR05] adressieren hierbei explizit die Evolution eines Prozesses. Für diese Arbeit ist die Fokussierung nur eines Modells nicht ausreichend. Vielmehr orientieren wir uns am Process Lines Ansatz, vorgeschlagen von Rombach [Rom05]. In ähnlicher Richtung spricht auch Kiebusch von Prozess-Familien [Kie06]. Rombach schlägt vor, Softwareprozesse aus Sicht von Produktlinien [BKPS04, GS04] zu betrachten und einen Prozess analog zu einer solchen zu entwerfen. Er zählt konkrete Punkte hinsichtlich Domain Engineering, generische Prozesselemente sowie zugelassene Variablen auf. Er motiviert weiterhin ein Process Repository, welches wiederum mit Ideen der Prozessmusteransätze [BRSV98, GMP+03] harmoniert. Im Bereich dynamischer Prozessnetze sind zum Beispiel die Arbeiten von Westfechtl und Heller [HJ04, HSW04, HW06, Wes01] noch aufzuführen. Diese Arbeiten beschäftigen sich am Beispiel des AHEAD-Systems (*Adaptable and Human-Centered Environment for the MAnagement of Development Processes*) schwerpunktmäßig mit dynamischen Aufgabennetzen. Hierbei betrachten sie auch graphenbasierte Systeme, die zur Laufzeit auch geändert werden können. Aussagen zu Vorgehensmodellen oder Prozesslinien treffen sie jedoch nicht.
Beitrag: Diese Arbeit leistet einen Beitrag für das Verständnis eines Vorgehensmodells als Produkt. Darauf aufbauend orientieren wir uns am Process Lines Ansatz, um Familien von Vorgehensmodellen modellieren zu können und deren Weiterentwicklung zu unterstützen.

Architektur und Komponenten

Diese Überlegungen leiten bereits zu konstruktiven Fragen über. Hier stehen insbesondere Prozessmuster [BRSV98, GMP+01, GMP+03, Mün04, Mün05] und deren Anspruch, wiederverwendbares Prozesswissen vorzuhalten, im Fokus. Diese Arbeit verfolgt einen Architektur-getriebenen Ansatz, um den Ideen der Prozesslinie zu folgen. Die grundlegenden Konstruktionsmethoden entleihen wir dem Bereich der Software Architektur, hier insbesondere [SG96, HR02]. Beide stellen Komponentenmodelle vor, die aufgrund der strikten Trennung der Einheiten (*Components*) und deren Beziehungen (*Connectors*) ein hohes Maß an Modularität versprechen. Komponenten besitzen auch für Vorgehensmodelle eine hohe Relevanz. Ähnlich zur Software ist jedoch auch hier eine enorme Sprachvielfalt zu verzeichnen. So zum Beispiel spricht das V-Modell von Vorgehensbausteinen, EPF/OpenUP von Method Content, Process Content und Capability Pattern (jeweils auf verschiedenen Abstraktions- und Kompositionsstufen). Aber bereits hier fällt auch der Begriff des Process Element, wie er zum Beispiel auch in [BBM05] zu finden ist. Allen Begriffen gemein ist, dass versucht wird, ein Maß zu definieren, anhand dessen eine geeignete Gruppierung von Inhalten eines Prozesses vorgenommen werden kann. Gruppierungen können dabei zum Beispiel strukturell (V-Modell) oder inhaltlich wie bei [BBM05] erfolgen.

Der Rückgriff auf Komponentenkonzepte der Software Architektur legt nicht nur inhaltliche Betrachtungen nahe, sondern fordert aufgrund der gewählten Basismethoden eine zusätzliche Auseinandersetzung mit der Gesamtarchitektur eines Vorgehensmodells. Hier stehen nicht nur inhaltliche Überlegung an, sondern konkret definierte, physisch eigenständige Komponenten, sowie ein systemtheoretischer Ansatz zur sinnvollen Kombination.
Beitrag: In diesem Punkt hebt sich die vorliegende Arbeit von den betrachteten ab. Sie berücksichtigt nicht nur Teilaspekte der Modularisierung, sondern setzt von Anfang an komponentenorientierte Konzepte um, die einer Maximierung der Modularisierung dienen. In Konsequenz steigt damit die Flexibilität, die Nachnutzbarkeit einzelner Komponenten und somit auch die Wandlungsfähigkeit eines aus solchen Komponenten aufgebauten Vorgehensmodells.

1.4. Aufbau der Arbeit

Die vorliegende Arbeit gliedert sich wie folgt:

Kapitel 2 spiegelt den aktuellen Stand im Bereich der anpassbaren Vorgehensmodelle in wesentlichen Facetten wider. Anhand der ausgewählten Vertreter V-Modell XT, Microsoft Solutions Framework und OpenUP stellen wir wesentliche Konzepte zusammen und fokussieren dabei insbesondere die Anpassungsprozesse der gewählten Vorgehensmodelle. Als zweiten wesentlichen Punkt betrachten wir Produktlebenszyklen und Produktlinienansätze. Kapitel 2 schließt mit der Festlegung der in dieser Arbeit verwendeten Begriffe hinsichtlich der Domäne *Vorgehensmodelle*, der *Modellierung und Metamodellierung* mit UML und MOF sowie der verwendeten Grundbegriffe der *Graphentheorie*.

Kapitel 3 befasst sich allgemein mit den Architekturen von Vorgehensmodellen und deren Metamodellen. Dabei betrachten wir kurz etablierte Standards und Frameworks, die zum Teil den in Kapitel 2.1 ausgewählten Vorgehensmodellen zugrunde liegen. Im Anschluss betrachten wir detailliert Struktur- und dynamische (ablaufbezogene) Elemente eines Vorgehensmodells sowie Abhängigkeiten. Wir stellen mit diesem Kapitel eine Aufstellung aller relevanten Elemente eines Vorgehensmodells zusammen und bereiten diese für die weitere Verwendung vor. Wir schließen dieses Kapitel mit einer Reihe von Anforderungen für physisch verteilbare Prozesskomponenten und Anwendungsfällen, die für weitere Modellierung die Grundlage bilden, ab.

Kapitel 4 In Kapitel 4 erstellen wir das formale Basismodell für unser Vorgehensmetamodell. Wir diskutieren in diesem Kapitel die grundlegenden Elemente der Modellierung – die *Prozesselemente*. Weiterhin führen wir einen *Prozesskomponenten*-Begriff ein. Prozesskomponenten dienen uns als eigenständige Einheiten für Bearbeitung und Verteilung. Weiterhin betrachten wir in diesem Kapitel Optionen zur Komposition allgemeiner Prozesselemente zu Prozesskomponenten sowie darauf aufbauend die Komposition zu Vorgehensmodellen. Wir betrachten in diesem Kapitel außerdem Fragen hinsichtlich der Variabilität und stellen *Variabilitätsoperationen* für Prozesselemente vor. Diese stellen eine Ergänzung zum *konfigurativen* Erstellen von Prozesskomponenten und Vorgehensmodellen dar. Wir diskutieren alle Optionen und deren Auswirkungen auf Ebene des Gesamtmodells. Ein entsprechendes, hierarchisches Modell zur Positionierung von Projekt-, Ogranisations- und Standardmodellen wird in diesem Kapitel ebenfalls eingeführt. Die betrachtungen in diesem Kapitel halten wir abstrakt und nehmen explizit noch keine Typisierung vor. Diese erfolgt erst im Kapitel 5, wo wir dann abstrakte Einheiten und Operationen konkretisieren. Wir stellen mit dieser Zweiteilung in diesem Kapitel die grundlegenden Überlegungen vor, während wir uns mit konkreten Ausprägungen erst im Anschluss befassen. Wir halten damit die Art der konkreten Ausgestaltung solange wie möglich offen.

Kapitel 5 Das Kapitel 5 beinhaltet zwei zentrale Teile: Im Kapitel 5.1 betrachten wir den konkreten (strukturellen) Entwurf des Vorgehensmetamodells. Hierbei interessieren uns verschiedene Modellierungsebenen, sowie die Erstellung von Einzelelementen und deren Komposition, auch über mehrere Stufen hinweg. Neben den inhaltlichen und strukturellen Betrachtungen erläutern wir hier auch ein Verteilungsmodell, das den Anforderungen hinsichtlich einer physischen Komponentenbildung gerecht wird.

Ab Kapitel 5.2 bauen wir auf den Ergebnissen aus Kapitel 5.1 auf und beleuchten den Erstellungs-, Einführungs- und Weiterentwicklungsprozess eines Vorgehensmodells. Wir weiten die Betrachtungen dabei von einem Vorgehensmodell auf eine Menge Vorgehensmodelle aus. Dort fokussieren wir uns dann auf Änderungs- und Pflegeoperationen des Hauptentwicklungsstrangs, sowie deren Auswirkungen auf angepasste Modelle in abgeleiteten Entwicklungssträngen. Wir integrieren diese beiden Teile zum *Integrierten Modellierungsansatz für Vorgehensmodelle* (IMV).

Das Kapitel schließt mit der Beschreibung des Werkzeugkonzepts für den vorgestellten Ansatz.

Kapitel 6 Im Kapitel 6 betrachten wir abschließend die Wirkung der hier entwickelten Konzepte aus der Sicht der Anwender. Wir verlassen dabei die Sicht des Prozessingenieurs und widmen uns auch dem Projektleiter sowie seinen Aufgaben im Bereich Projektinitiierung. Im Wesentlichen zeigen wir in diesem Kapitel die Anwendung des IMV-Ansatzes anhand von Beispielen aus dem V-Modell XT.

Kapitel 7 Dieses Kapitel fasst die Ergebnisse der Arbeit zusammen und enthält eine Bewertung des Erreichten sowie einen Ausblick.

Im Anhang der vorliegenden Arbeit haben wir einige Fallbeispiele zusammengefasst. Im *Anhang A* widmen wir uns schwerpunktmäßig der Anpassung des V-Modell XT und den Erfahrungen, die aus diesen Tätigkeiten gezogen werden konnten. Besonders interessant für diese Arbeit ist dabei der Anhang A.3, da hier ein konkretes Anwendungsfeld liegt, in dem Teilergebnisse dieser Arbeit bei der Überarbeitung des V-Modell XT angewendet werden konnten. Im *Anhang B* zeigen wir beispielhaft die Entwicklung und Weiterentwicklung eines Vorgehensmodells nach dem hier entwickelten Ansatz. Wir beziehen uns dabei auf das Planungsmodell von Gnatz [Gna05] und zeigen dessen Integration mithilfe unseres Ansatzes. Abschließend geben wir in Anhang C einen Einblick in den technische Umsetzung des Metamodells als Teil des Werkzeugkonzepts aus Kapitel 5.

2. Vorgehensmodelle: Stand der Technik

Vorgehensmodelle und Konzepte gibt es viele – so viele, dass es müßig bis unmöglich ist, sie alle vollständig zu erfassen. Aber nicht nur die Menge an Vorgehensmodellen gestaltet diesen Bereich des Software Engineerings komplex, sondern auch der rasante Wandel. In diesem Kapitel geben wir einen kurzen Einblick in den aktuellen Stand der Technik im Bereich der Vorgehensmodelle. Wir stellen zu Beginn ausgewählte Vertreter bekannter Vorgehensmodelle vor. Diese sind so gewählt, dass sie ein breites Spektrum möglicher Einsatzgebiete erfassen. Im Anschluss betrachten wir auf Basis der ausgewählten Vorgehensmodelle die Entwicklung und Anpassung und positionieren im Anschluss Ideen der Produkt- und Produktlinienentwicklung. Das Kapitel schließt mit Begriffsbildungen und Definitionen, die für das Verständnis der Arbeit erforderlich sind.

Am Ende dieses Kapitels hat der Leser einen kompakten Einblick in die Welt der Vorgehensmodelle und Anhaltspunkte zu deren Entwicklung und Anpassung erhalten. Die wesentlichen Begriffe dieser Arbeit sind definiert.

2.1. Ausgewählte Vorgehensmodelle

Im Folgenden stellen wir ausgewählte Vorgehensmodelle detaillierter vor. Diese dienen uns als Bezugspunkte für die Arbeit. Zentral beziehen wir uns auf das *V-Modell XT* und das *Microsoft Solutions Framework*. In Aspekten stützen wir uns auf dem Open Unified Process ab. Weitere Ansätze mit Bezug zu dieser Arbeit stellen wir im Anschluss vor. Wir konzentrieren uns in diesem Kapitel lediglich auf konzeptuelle Fragen.

2.1.1. Das V-Modell XT

Das V-Modell XT[1] [RH06] (kurz V-Modell), das im Rahmen des WEIT-Projekts am Lehrstuhl für Software & Systems Engineering der Technischen Universität München entwickelt wurde, stellt seit November 2004[2] das Standardvorgehen für die Durchführung von IT-Projekten in der öffentlichen Hand dar. Das V-Modell[3] ist ein ergebnisorientiertes Vorgehensmodell. Anstatt Aktivitäten oder Workflows mitsamt Ein- und Ausgaben zu beschreiben, definiert es so genannte Projektfortschrittsstufen, die jeweils durch Entscheidungspunkte abgeschlossen werden. Entscheidungspunkte sind Meilensteine mit Eigenschaften eines Quality Gates, zu denen jeweils Produkte qualitätsgesichert vorgelegt werden müssen. Ein V-Modell-Projekt definiert sich somit durch eine Menge zu erstellender Produkte.

Im Rahmen mehrerer Pilotprojekte [KN05b, KNB05, Kuh05] konnten wir umfangreiche Erfahrungen bei der Einführung und Anpassung des V-Modells sammeln. Zum Zeitpunkt der Erstellung dieser Arbeit ist das V-Modell bereits seit über zwei Jahren im praktischen Einsatz und steht in mehreren Versionen/Derivaten zur Verfügung (vgl. Abbildung 1.1). Viele Erkenntnisse aber auch Probleme insbesondere aus diesen frühen Phasen der Anwendung, sind treibende Kräfte hinter dieser Arbeit. Wir geben an dieser Stelle nur einen kompakten Überblick über das V-Modell. Für detaillierte Ausführungen, insbesondere zum Metamodell [Gna05, Gna06, KMR06], zur Anpassbarkeit

1 Das V-Modell XT Portal: `http://www.v-modell-xt.de`

2 Am 4. November 2004 erfolgte die Empfehlung zur Anwendung des V-Modells durch den Interministeriellen Koordinierungsausschuss (IMKA), siehe V-Modell XT Portal.

3 Obwohl das V-Modell bereits in den 1970'ern vorgestellt wurde, erlangte es erst mit V-Modell 92 eine höhere Bekanntheit. Es war eines der ersten, komplexen und integrierten Vorgehensmodelle. Es wurde durch das Bundesministerium für Verteidigung (BMVg) verpflichtend für seine Auftragnehmer erlassen und später auch vom Bundesministerium des Innern (BMI) übernommen. Seine letzte Aktualisierung fand das V-Modell mit der Version V-Modell 97. Wenn wir in dieser Arbeit vom V-Modell reden, beziehen wir uns immer auf die aktuelle Ausprägung V-Modell XT.

[KHTS07, HKST07, KK07, Kuh07] und zur Erweiterbarkeit [Gru05, KT06, Ham08] verweisen wir auf entsprechende Vorarbeiten.

Konzepte des V-Modell XT im Überblick

Das V-Modell XT basiert auf einem *formalen Metamodell* und einer konkreten Ausprägung dieses Metamodells in Form einer XML-Spezifikation (XML-Schema). Das Metamodell mitsamt aller definierten Konzepte bildet die *Sprache* des V-Modells (siehe auch Kapitel 3.2.1). Es enthält und beschreibt alle Elemente des V-Modells, zum Beispiel Produkte, Abläufe oder Vorgehensbausteine. Die wesentlichen Konzepte führen wir nun kurz ein.

Vorgehensbausteine. Vorgehensbausteine sind Container für die einfachen Bausteine (Produkte, Aktivitäten und Rollen) des V-Modells. Vorgehensbausteine zeigen Ähnlichkeiten zu den Prozessmusteransätzen [GMP+03], sind jedoch grob granularer ausgelegt. Das V-Modell definiert Vorgehensbausteine wie folgt:

Vorgehensbaustein

> Jeder Vorgehensbaustein ist eine eigenständige Einheit und einzeln änder- beziehungsweise erweiterbar. Ein Vorgehensbaustein beinhaltet alle Bestandteile, die zur Bearbeitung einer konkreten Aufgabenstellung [. . .] notwendig sind. [vmx]

Vorgehensbausteine repräsentieren in der Regel einen Teilprozess beziehungsweise eine *Disziplin* eines Projekts und stellen für diese alle relevanten Elemente zur Verfügung. Beispiele für solche Disziplinen sind Projektmanagement, Qualitätssicherung oder SW-Entwicklung (siehe Abbildung 2.1, unten), die als jeweils eigene Vorgehensbausteine vorliegen.

Die Anzahl der Vorgehensbausteine und somit der Umfang des V-Modells variiert zwischen den Versionen. Das V-Modell XT 1.2.1 besteht beispielsweise aus 21 Vorgehensbausteinen; das V-Modell der Bundeswehr aus 22. Ein Gesamtüberblick kann der *V-Modell-Referenz Teil 3: Tailoring* [vmx] entnommen werden. Vorgehensbausteine sind die Komponenten des V-Modells, die als Glas-Box realisiert sind. Im Kontext des Gesamtmodells sind alle Inhalte eines Vorgehensbausteins uneingeschränkt sichtbar und für die Verwendung verfügbar. Obwohl das Konzept Vorgehensbaustein auf den Ideen der Komponentenorientierung basiert, setzt es diese nur unvollständig um. Im Kapitel 3 gehen wir auf diese Problematik vertiefend ein.

Ein konkretes Vorgehen nach V-Modell XT wird im Rahmen des *Tailorings* (siehe *Anpassung des V-Modell XT* in Abschnitt 2.2.2) durch geeignete Kombination mehrerer Vorgehensbausteine abgeleitet. Die „geeigneten" Kombinationen werden durch Beziehungen zwischen Vorgehensbausteinen aber auch durch Projekttypen festgelegt. Das V-Modell legt für Abhängigkeitsdefinitionen zwischen Vorgehensbausteinen folgende Beziehungstypen fest: `basiert_auf` und `kann_basieren_auf`, siehe Kapitel 3.2.1. Diese Beziehungstypen werden verwendet, um beispielsweise Spezialisierungen oder inhaltliche Ergänzungen vorzunehmen.

Beispiel: Die beiden Vorgehensbausteine „Systemerstellung" und „SW-Entwicklung" stehen in einer basiert_auf-Beziehung. Systemerstellung legt dabei den Rahmen für die allgemeine Erstellung eines Systems fest, während SW-Entwicklung eine Ergänzung für Softwaresysteme vornimmt. (vgl. [vmx]).

Beachtet werden muss bei diesen Beziehungstypen aber, dass sie nur auf der Ebene der Vorgehensbausteine wirken. Stehen zwei Vorgehensbausteine v_1 und v_2 in der Relation *basiert_auf* (v_2, v_1), so wird bei Auswahl des Vorgehensbausteins v_2 automatisch auch v_1 mit ausgewählt. Alle in v_1 definierten Inhalte werden somit zu Inhalten des Projekts. Ist die Abhängigkeitsbeziehung jedoch nicht vorhanden, aber Elemente der Vorgehensbausteine referenzieren sich untereinander, wird diese Abhängigkeit nicht weiter beachtet. Mit den Vorgehensbausteinen versucht das V-Modell eine komponentenbasierte Architektur umzusetzen. Der Komponentenbegriff des V-Modells ist jedoch *virtuell* –

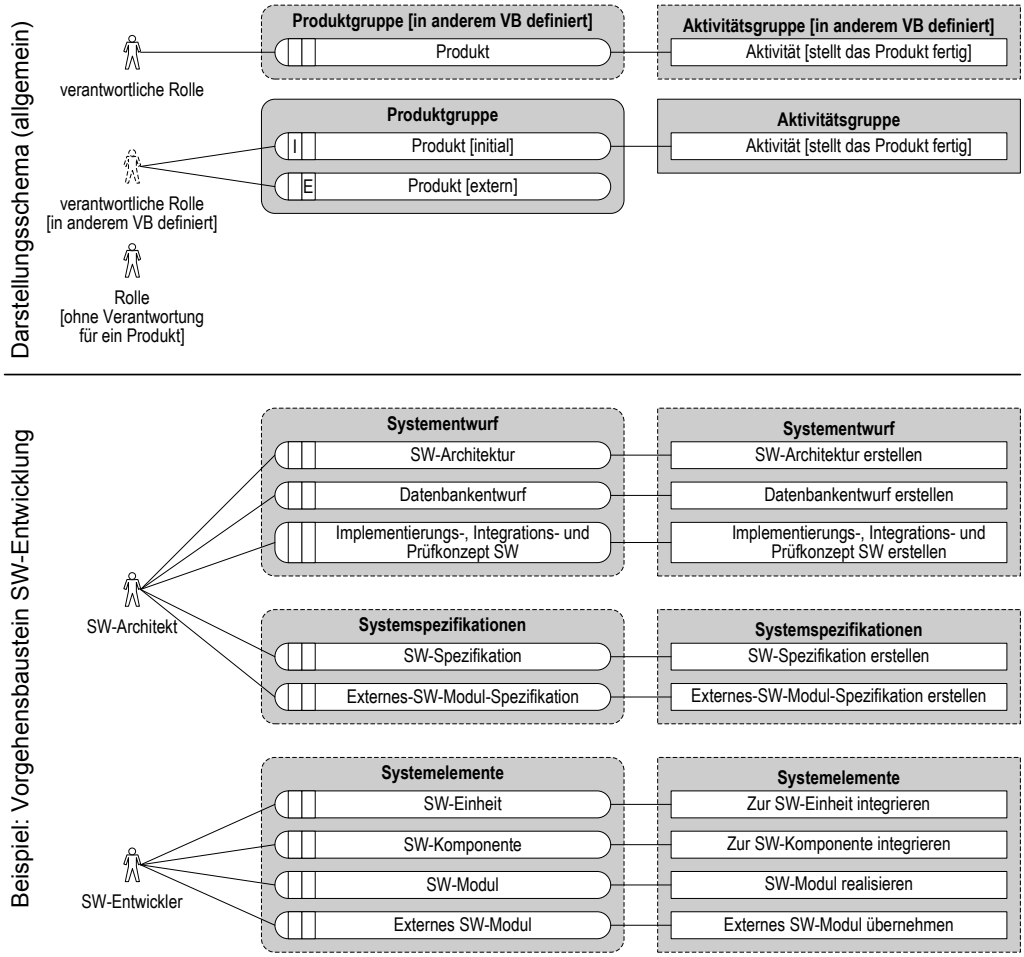


Abbildung 2.1.: Beschreibungsschema und Beispiel eines Vorgehensbausteins

es gibt keine physisch eigenständigen Komponenten. Relationen zwischen Elementen einzelner Vorgehensbausteine oder anderer V-Modell-Teile sind oft direkt und ohne Beachtung von Komponentengrenzen oder Spezialisierungsebenen geknüpft. Die daraus resultierende hohe Verflechtung und Integration *aller* V-Modell Anteile erschwert die Wartung und Pflege des V-Modells – insbesondere angepasster Versionen. Dies führt zu Problemen, die in [Gna05, KT06] bereits aufgeworfen wurden und deren Lösung mit den Arbeiten am so genannten *Task RO* (Anhang A.3) angestrebt wird.

Projekttyp. Projekttypen legen eine Vorauswahl des Projektgegenstands fest und bieten gleichzeitig ein geeignetes Vorgehen zur Abwicklung des Projekts an. Das V-Modell definiert insgesamt vier Projekttypen:

- Systementwicklungsprojekt (Auftraggeber – AG),
- Systementwicklungsprojekt (Auftragnehmer – AN),
- Systementwicklungsprojekt (AG/AN) und
- Einführung und Pflege eines organisationsspezifischen Vorgehensmodells.

Für jeden Projekttyp bietet das V-Modell eine Menge verpflichtender und optionaler Vorgehensbausteine und mindestens eine Projektdurchführungsstrategie an. Neue Vorgehensbausteine oder Projektdurchführungsstrategien sind zu einem Projekttyp ebenfalls hinzufügbar [KHTS07].

Projektdurchführungsstrategien. Projektdurchführungsstrategien stellen Vorlagen für einen prinzipiellen Projektablauf dar. Sie fassen eine Menge von *Entscheidungspunk-*

ten sowie eine Menge möglicher Pfade zwischen einzelnen Entscheidungspunkten zusammen (Abbildung 2.2).

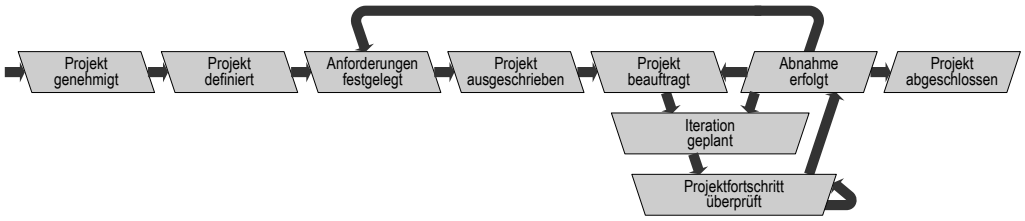


Abbildung 2.2.: Beispiel PDS: Vergabe eines Systementwicklungsprojekts (AG)

Das V-Modell definiert bereits eine Menge von Entscheidungspunkten, die jeweils für verschiedene Strategien wiederverwendet werden können. Im V-Modell ist das Konzept Entscheidungspunkt wie folgt beschrieben:

Entscheidungspunkt

> [...] Ein Entscheidungspunkt weist einen Meilenstein im Projektablauf aus, an dem der aktuelle Stand des Projektes evaluiert wird. Für jeden Entscheidungspunkt ist im V-Modell eine Menge von Produkten definiert, die am Ende der Projektfortschrittsstufe fertig gestellt sein müssen. Auf der Basis dieser Produkte entscheidet das projektübergeordnete Management, ob die Projektfortschrittsstufe mit Erfolg erreicht wurde [...]. [vmx]

Entscheidungspunkte sind den Ablaufkonzepten zugeordnet und von Vorgehensbausteinen getrennt. Die Menge der Produkte, die in einem Entscheidungspunkt vorliegen muss, wird durch die Produkttypdefinition selbst bestimmt. Produkte besitzen ein Attribut `Entscheidungspunkt`, das eine Menge von Referenzen auf Entscheidungspunkte enthalten kann. Dadurch werden Verbindungen hergestellt, die im späteren Projektablauf Zeitpunkt und Umfang der Produkterstellung regeln. Einer Projektdurchführungsstrategie ist gleichzeitig ein Ablaufbaustein zugeordnet. Dieser fasst Ablaufentscheidungspunkte zusammen, die konkrete Nachfolgerbeziehungen zwischen Entscheidungspunkten definieren. Durch die Nachfolgerbeziehungen der Ablaufentscheidungspunkte sowie deren Festlegung als ausgezeichneter Start- oder Endeknoten, werden die möglichen Transitionen zwischen den einzelnen Entscheidungspunkten (die möglichen Pfade) festgelegt.

Projektdurchführungsstrategien und Ablaufbausteine sind zwei verschiedene Sichten auf einen Projektablauf. Während erstere im Wesentlichen die Erstellungsreihenfolge und -häufigkeit von Produkten adressieren, sind letzte die Elemente, die zur automatischen Plangenerierung [Ber06, Fis06, HKST07] verwendet werden können. Die Konzepte Ablaufbaustein und Ablaufentscheidungspunkt sind für den Endanwender des V-Modells nicht sichtbar. Diese Konzepte adressieren ausschließlich Prozessingenieure und Werkzeughersteller.

Werkzeuge des V-Modell XT

Aufgrund seiner Basierung auf einem formalen Metamodell und der Implementierung in XML ist das V-Modell sehr gut durch Werkzeuge zu unterstützen. In einer Reihe von Arbeiten zum Beispiel [Ber06, KT06, MRS06, Fri06] werden die vielfältigen Möglichkeiten gezeigt. Das V-Modell enthält bereits zwei Werkzeuge als Referenzimplementierung[4]. Die Werkzeuge bilden ein Gesamtkonzept und unterstützen *Authoring* und *Tailoring* des V-Modells.

4 Die Werkzeuge werden von der 4Soft GmbH München gefertigt (`http://www.4soft.de`) und stehen als Open Source zur Verfügung.

V-Modell XT Editor. Der V-Modell XT Editor ist ein XML-Editor, der für die Bearbeitung strukturierter (XML-)Texte ausgelegt ist. Der Editor arbeitet XML-Schema-basiert und kann somit die Informationen des V-Modell XML-Schemas direkt auswerten und an den Bearbeiter weitergeben. Der Editor kann beispielsweise die komplexen Strukturen eines Vorgehensbausteins gemäß der Schemadefinition automatisch erstellen. Auch die beschränkte Eingabe von Werten (in Abhängigkeit der Schemadefinition) – beispielsweise Kardinalitäten von Elementen – kann der Editor interpretieren und dem Anwender rückmelden. Der Editor unterstützt die inhaltliche Ausgestaltung von V-Modell-Instanzen. Eine direkte Bearbeitung des zugrunde liegenden XML-Schemas ist hingegen nicht vorgesehen. Für Informationen zu weiteren Möglichkeiten und Anwendungsfällen des V-Modell Editors verweisen wir auf [MRS06, KHTS07, HKST07].

V-Modell XT Projektassistent. Das zweite, wesentlich sichtbarere Referenzwerkzeug ist der V-Modell XT Projektassistent. Dieser dient der werkzeugunterstützten, projektspezifischen Anpassung des V-Modells. Der Projektassistent ist ebenfalls wie der Editor generisch ausgelegt und kann – basierend auf dem V-Modell XT XML-Schema – Instanzen des V-Modells verarbeiten.

Projektassistent und V-Modell-Versionen

Der Projektassistent kann auch für verschiedene V-Modell-Versionen eingesetzt werden, sofern die Schemadefinitionen nicht abweichen. Wichtig ist diese Eigenschaft für Organisationen, die ausgehend von einem Basismodell mehrere Ableitungen erstellen. Diese sind für den Projektassistenten transparent.

Der Projektassistent nimmt gemäß der Vorgaben des Modells sowie der Parametrisierung des Anwenders die projektspezifische Anpassung vor. Hierbei werden in der Regel die Inhalte des V-Modells soweit möglich an die Anforderungen der Anwender angepasst. In [Gna05, KNB05, Kuh06] werden die Konzepte und die Resultate detailliert betrachtet. Neben der Anpassung (dem Tailoring) des V-Modells dient der Projektassistent jedoch auch noch dazu, Eingaben für weiterverarbeitende Werkzeuge zu generieren. Dies sind:

- Ein *initialer Projektplan*, der als CSV-Datei für Microsoft Project oder als XML-Datei im Format für die Open Source Software GanttProject[5] erzeugt werden kann. Dieser enthält einen initialen Meilensteinplan, der aus den Entscheidungspunkten des gewählten Vorgehens erzeugt wird und bereits alle relevanten Aktivitäten für die zu erstellenden Produkte enthält.
- Eine initiale *Produktbibliothek*, die alle ausgewählten Produkttypen jeweils als Vorlage im RTF-Format enthält. Die Struktur der Produktbibliothek orientiert sich dabei an der Struktur der Produktgruppen.
- Eine projektspezifisch angepasste *Dokumentation* des angepassten V-Modells. Der Export ist in HTML, XML und PDF[6] möglich.

Die Werkzeugkette des V-Modells, insbesondere die der Referenzwerkzeuge, ist beliebig erweiterbar. Denkbar sind beispielsweise Modellierungswerkzeuge [RBTK05] oder sonstige Prozesserweiterungen. In [KK07] oder [Kuh07] haben wir die beispielhafte Erweiterung um zusätzliche „Generatorstufen" für die problemorientierte Weiterverarbeitung demonstriert. In [KK07] wird beispielsweise die Generierung eines vollständigen V-Modell XT-Projektportals als Microsoft SharePoint 2007 Lösung gezeigt. V-Modell-Strukturen werden dabei analysiert und interpretiert. Neben der Dokumentation und der Produktbibliothek steht den Anwendern mit dieser Lösung auch eine weiter gehende Prozessunterstützung zur Verfügung.

5 Projektwebseite: `http://ganttproject.biz`

6 Mit dem Release der V-Modell-Version 1.2.1 wurde das Exportsystem des Editors und des Assistenten umgestellt, sodass nun auch ein Export in Open Document Format (ODF) möglich ist.

2.1.2. Das Microsoft Solutions Framework

Das Microsoft Solutions Framework[7] (MSF, [Tur06]) ist ein Produkt der Firma Microsoft und definiert interne Entwicklungsprozesse, die in Form von Beratungsleistungen auch Kunden angeboten werden (vgl. auch das Konzept des OEP [OSKZ06]). MSF ist seit etwa 1993 verfügbar. Es stellt eine Prozessbeschreibung dar, die aus der internen Entwicklung in Zusammenarbeit mit Partnern[8] zu einem Produkt weiterentwickelt wurde. Das letzte Release 3.x [Kee04] definierte bereits umfangreiche Konzepte, wie zum Beispiel ein Team- und ein Prozessmodell. Zu diesen kamen noch so genannte Disziplinen als weitere Kernkomponenten hinzu. Das MSF stellt nicht nur ein Prozessmodell für die Softwareentwicklung dar, sondern verfügt weiterhin über eine Schnittstelle zum Microsoft-Standard für Softwarebetrieb, dem Microsoft Operations Framework (MOF). Die aktuelle Version des MSF ist 4.x.

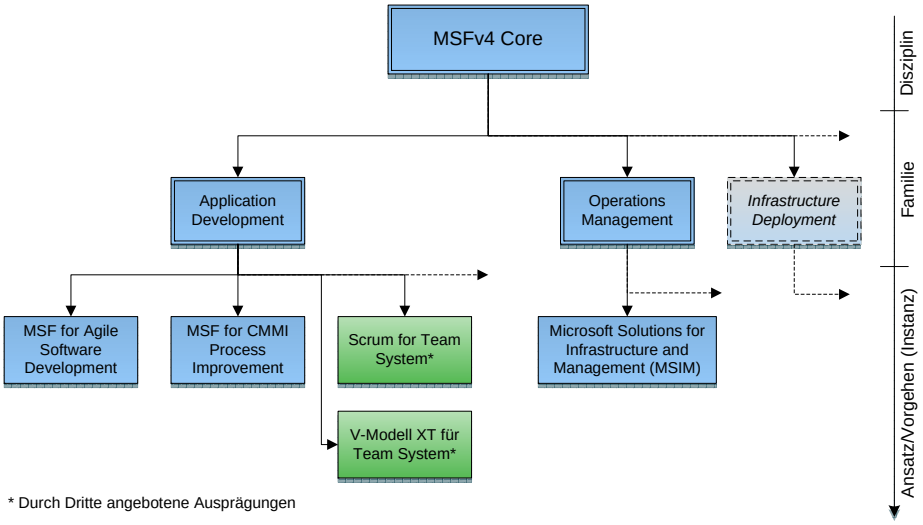


Abbildung 2.3.: Der „Familienstammbaum" des Microsoft Solutions Framework nach [Tur06]

MSF ist ein Prozessframework, das in seiner Grundintention dem V-Modell sehr ähnlich ist. Aus einem abstrakten (präskriptiven) Modell, dem MSF 4.0 Metamodell (MSFv4 Core, Abbildung 2.3), können verschiedene konkretere Modelle abgeleitet werden. Microsoft schlägt eine Familienbildung vor und gruppiert in Entwicklung (Application Development), Betrieb (Operations Management) und Infrastruktur (Infrastructure Deployment). Wir interessieren uns im Kontext dieser Arbeit hauptsächlich für die Entwicklungsmodelle, die wir nun genauer betrachten.

Zurzeit gibt es im Wesentlichen zwei Ausprägungen des MSF direkt von Microsoft:

- MSF 4.0 for Agile Software Development und
- MSF 4.0 for CMMI Process Improvement

Weitere Ausprägungen werden von Drittanbietern (Abbildung 2.3) erstellt und angeboten. Beispiele hierfür sind Scrum [Sch04] oder aber auch das V-Modell XT [Kuh07]. Grundlage des zu erstellenden Prozesses bildet das Metamodell, das ähnlich wie beim V-Modell als Menge von XML-Schemadefinitionen vorliegt ([Kuh07], Anhang B). Dies gestattet es, MSF sehr gut durch Werkzeuge zu unterstützen, beziehungsweise MSF als Grundlage für verschiedene Prozesse zu verwenden.

Im Bereich der Werkzeugunterstützung war MSF bis zur Version 3 nur wenig ausgestaltet. Bis auf eine Projektvorlage in Microsoft Project 2003 (MSF Anwendungsentwicklung) gab es keine wirkliche Unterstützung. Seit der Version 4.0 ist MSF direkt in das

7 Landing Page für MSF: `http://msdn2.microsoft.com/en-us/teamsystem/aa718795.aspx`
8 Process und Partner Templates: `http://msdn2.microsoft.com/en-us/teamsystem/aa718801.aspx`. Mit MSFv4 und dem Team Foundation Server als generischer Basis, wird von Microsoft und seinen Partnern eine umfassende Prozesslandschaft erzeugt.

Visual Studio 2005 integriert. Die Integration erfolgt einmal serverseitig im Backend, einmal clientseitig bei den Mitarbeitern im Projekt. Im Backend wird MSF durch den Team Foundation Server (TFS, [GP06]) realisiert. Dieser stellt Infrastruktur und Datenablage sowie Reportingmechanismen bereit. Clientseitig spiegelt sich MSF in Abhängigkeit von der konkreten Rolle im Projekt wider. Projektmanager sehen beispielsweise Excel, Project oder das Teamportal als SharePoint-Anwendung (vgl. [KK07]) als Frontend, während Entwickler vorrangig mit dem Visual Studio 2005 Team System arbeiten werden. Auch für das Authoring des Prozesses stehen nun Werkzeuge in Form des Process Template Editors als Teil des Visual Studio SDK zur Verfügung.

Konzepte des Microsoft Solutions Framework im Überblick

MSF definiert ebenfalls einige grundlegende Konzepte. Viele davon sind mehr im Bereich der Richtlinien und Empfehlungen anzusiedeln (Microsoft spricht hier von *Governance*) und daher *nicht formal* erfassbar. Dennoch basiert MSF ebenfalls auf einem Metamodell. Dieses Metamodell teilt sich im Wesentlichen in zwei Teile: einen *strukturellen* und einen *dynamischen* Teil. Im strukturellen Teil des Metamodells werden Entitäten definiert, wie zum Beispiel Rollen, Produkte oder Aktivitäten. Dieser Teil ist mit den entsprechenden Metamodellelmenten des V-Modells vergleichbar. Im dynamischen Teil des Metamodells werden Strukturen beschrieben, wie sie für die Instanziierung und Durchführung eines Projekts notwendig sind. Dazu zählen beispielsweise die Zusammenhänge zwischen Checkpunkten (Meilensteinen), Workstreams und Iterationen. Detailliertere Informationen[9] hierzu sind in [KT06, Tur06, Kuh07] zu finden.

Workstreams. Ein elementares Konzept sind *Workstreams*. Ein Workstream dient der Strukturierung und Gruppierung von Aufgabenpaketen. Sie bilden eingenständige Arbeitseinheiten, die im Kontext des Gesamtprojekts über so genannte Checkpunkte miteinander synchronisiert werden. Ein Beispiel für einen Workstream ist der Workstream *Guide Iteration*. Dieser fasst Aktivitäten des Projektmanagers zusammen, die notwendig sind, um eine Iteration eines Projekt zu planen, durchzuführen und zu bewerten.

Work Items. Ein weiteres wichtiges Konzept des MSF sind *Work Items*. Work Items sind als Datenbankeinträge beziehungsweise als Datenelemente definiert. Sie stellen einen Artefakttyp dar, der es relativ einfach gestattet, Tracking und Reporting in einem Projekt zu etablieren (beispielhaft zu sehen in [GP06], Kapitel 4). Jedes Work Item besteht aus einer in XML modellierten Datenstruktur und einem Zustandsautomaten mit einer Menge definierter Übergänge zwischen verschiedenen Stati. Durch die Kopplung von Daten und Workflows tragen Work Items eine eigene Methodik und definieren quasi im „Vorbeigehen" ganze Teildisziplinen eines Projekts wie Risiko- oder Aufgabenmanagement. Beispiele für konkrete Work Items sind:

- Aufgaben (Task),
- Risiken (Risk),
- Anforderungen (Scenario, Quality of Service Requirement) oder
- Änderungsanträge (Change Request) und
- Problemmeldungen (Issue).

Werkzeuge des Microsoft Solutions Framework

Anders, als beispielsweise das V-Modell, ist MSF zum Teil auf verschiedene Werkzeuge (Text, XML und sonstige Editoren) angewiesen, um Inhalte zu erstellen. Dies betrifft insbesondere die Prozessdokumentation, die nicht wie beim V-Modell automatisch generiert wird, sondern separat zu erstellen ist. Liegen jedoch einmal alle erforderlichen

9 Eine Dokumentation des Metamodells direkt von Microsoft ist nicht erhältlich.

Daten vor, kann der Prozess mithilfe des *Process Template Editors* (PTE) zusammengestellt werden. Dieser Editor ist direkt im Visual Studio integriert und wird als Teil des Visual Studio SDK kostenfrei angeboten.

Der PTE gestattet es, Work Items zu definieren, Prozessabläufe festzulegen sowie die (vorgefertigte) Dokumentation sowie ggf. vorhandene Dokumentvorlagen einzubinden. Die Ausgaben des PTE stellen alle notwendigen Daten für den *Team Foundation Server* dar. Dieser Server ist eine Backendkomponente, die Webzugang, Prozesssteuerung und Konfigurationsmanagement übernimmt.

Team Foundation Server

Der Team Foundation Server (TFS) ist eine Anwendung, die auf einer hoch integrierten Serverinfrastruktur aufbaut. Zu dieser gehören Datenbankserver, Quellcodeverwaltung, Webserver und Portalserver. Trotz dieser hoch integrierten Microsoft-Infrastruktur ist der gesamte TFS über Webservices ansprechbar und programmierbar (entsprechende Schnittstellendefinitionen und Beispiele liegen dem SDK bei). TFS ist somit auch von anderen Plattformen aus erreichbar, wie zum Beispiel aus Eclipse[10].

Der Team Foundation Server stellt gleichzeitig zu seinen Serverfunktionen auch passende Client Plug-Ins zur Verfügung (beispielsweise für das Visual Studio oder Microsoft Office). Sofern also ein entsprechendes Plug-In für eine beliebige Anwendung verfügbar ist, kann sie die Dienstleistungen von TFS in Anspruch nehmen. Zurzeit ist dies für Project, Office, Visual Studio 2005 und Eclipse (durch Drittanbieter) der Fall.

2.1.3. Der Open Unified Process

Als Vertreter der SPEM-basierten [OMG05] Vorgehensmodelle betrachten wir den Open Unified Process (OpenUP). OpenUP ist eine leichtgewichtige Version des Rational Unified Process (RUP, [KK03]). Im Rahmen des *Eclipse Process Framework* Projekts (EPF, [Hau06a, Hau06b]) wird OpenUP als ein möglicher Prozess angeboten und weiter entwickelt. OpenUP bietet durch EPF eine weit reichende Werkzeugunterstützung und ein mit dem V-Modell vergleichbares Modulkonzept, weshalb er für uns interessant ist. OpenUP ist sehr eng mit EPF integriert, weshalb die Annahme einer hohen Spezialisierung nahe liegt. Trotzdem ist sowohl der Prozess als auch die Umgebung sehr flexibel und anpassbar ausgelegt und eignet sich auch für die Definition anderer Prozesse. Auf den Projektwebseiten des EPF[11] sind beispielsweise auch Plug-Ins für Scrum [Sch04] oder eXtreme Programming (XP) [Bec03] verfügbar. Auch Friedrich [Fri06] stützte sich in Teilen auf EPF ab.

Konzepte des Open Unified Process im Überblick

OpenUP bezeichnet sich selbst als agile Methode mit Anteilen eines „Unified Process". Er nimmt einige Prinzipien des so genannten *Agile Manifesto*[12] [Bec03] in Anspruch. Die für uns interessanten (technischen) Konzepte des OpenUP begründen sich jedoch aus dem SPEM und dem darauf aufbauenden UMA[13]. Diese beiden Metamodelle und das als Werkzeug aufbauende Eclipse Process Framework legen die Basisstrukturen des Prozesses fest. Das Eclipse Process Framework (EPF) ist ein Teil des von IBM initiierten Eclipse Projekts. Es basiert auf einer Composer-Komponente, die IBM/Rational auch für den „großen" RUP einsetzt. EPF-Composer basiert auf dem Eclipse Modelling Framework (EMF) und der darauf aufbauenden Unified Method Architecture (UMA). Diese beiden Frameworks sind vergleichsweise generisch gehalten, sodass jeder Prozess, der mit den Konzepten dieser beiden Frameworks darstellbar ist, auch mit dem Composer erfasst werden kann.

10 Landing Page von Teamprise: `http://www.teamprise.com`
11 Landing Page des EPF: `http://www.eclipse.org/epf`
12 Siehe auch: `http://www.agilemanifesto.org`
13 UMA – Unified Modelling Architecture ist eine Weiterentwicklung des SPEM 1.1 (IBM) und für die Aufnahme in SPEM 2.0 eingereicht.

Method Content. Mit *Method Content* werden im OpenUP die Strukturelemente bezeichnet, die der inhaltlichen Ausgestaltung des Prozesses dienen. Dies umfasst Artefakttypen, Aktivitäts- und Rollenbeschreibungen sowie sonstigen Inhalt. Wir beschreiben diesen Inhalt in Kapitel 3.2 detaillierter. Das zugrunde liegende Konzept ähnelt dem Vorgehensbaustein des V-Modells (Abschnitt 2.1.1). Method Content kann – einmal definiert – beliebig oft in einem Prozess verwendet werden. Analog zum V-Modell werden hier keine Aussagen gemacht, wann und wie oft Elemente erzeugt/verwendet werden. Dies übernimmt der Prozessanteil.

Process Content. Der dynamische Anteil des OpenUP wird durch den so genannten *Process Content* beschrieben. Process Content beschreibt im Wesentlichen, wann Elemente des Method Content erstellt werden (Artefakte) oder auszuführen sind (Tasks). Process Content bringt diese Elemente in eine Ordnung und fasst sie zu Einheiten zusammen, die als *Capability Patterns* bezeichnet werden. Capability Patterns sind mit dem Konzept Workstream des MSF (Abschnitt 2.1.2) vergleichbar. Sie realisieren eine mögliche Ausprägung des *Process Pattern*-Ansatzes [BRSV98, GMP+03]. Capability Patterns sind auch zu größeren Einheiten komponierbar. Der Bereich des Process Content ist offen, sodass verschiedene Prozesse möglich sind.

Werkzeuge des Open Unified Process

OpenUP ist komplett in Eclipse integriert. Die Integration umfasst dabei ausschließlich das Authoring des Prozesses. EPF selbst, insbesondere der EPF-Composer als Kernkomponenten sind hoch integriert, jedoch gibt es keine mit dem V-Modell XT oder MSF vergleichbar weit reichende Werkzeugkette[14]. Gau [Gau06] macht in seinem Artikel recht deutlich, dass es sich bei EPF lediglich um ein „Frontend" für die UMA handelt. Er unterbreitet auch den Vorschlag, andere Vorgehensmodelle mithilfe von EPF und UMA zu remodellieren. Insbesondere mit Hinblick auf die fehlenden Schnittstellen zur Operationalisierung eines EPF/UMA-basierten Prozesses erscheint diese Forderung fragwürdig.

Die Optionen sind in EPF/UMA jedoch vorhanden: Das Werkzeug basiert auf einem Metamodell und setzt dieses adäquat um. Von den Fähigkeiten zum Editieren eines Prozesses geht EPF sichtbar über das V-Modell XT und MSF hinaus. Geeignete Werkzeuge zum Beispiel vom Typ des V-Modell XT Projektassistenten würden EPF/UMA samt OpenUP zu einer (nach unseren Maßstäben) gleichwertigen Alternative zum V-Modell XT und MSF machen.

2.1.4. Weitere Ansätze

Mit dem V-Modell XT, MSF und OpenUP haben wir nur drei Vertreter der Vorgehensmodelle für die System- und Software-Entwicklung betrachtet. Wir haben diese Vertreter gewählt, weil sie einen guten Querschnitt bieten. Alle sind sie Metamodell-basiert, anpassbar und in unterschiedlicher Form durch Werkzeuge unterstützt. Wir wollen aber trotzdem noch kurz einen Ausblick auf weitere Vorgehensmodelle und Entwicklungsprozesse geben beziehungsweise namentlich bekannte und weit verbreitete Prozesse einordnen.

Entwicklungsmodelle

Alle Vorgehensmodelle, die wir bislang betrachtet haben waren *Entwicklungsmodelle*, d. h. Modelle, die System- beziehungsweise Softwareentwicklung beschreiben und unterstützen. Weitere Modelle, die in diesem Feld Relevanz haben, sind zum Beispiel die immer bekannter werdenden Vertreter der agilen Methoden, wie zum Beispiel das Extreme Programming (XP, [Bec03]) oder aber auch Scrum [Sch04]. Beide Methoden unterscheiden sich nicht in ihrer Grundintention und legen eine Reihe von Werten[15] fest,

14 Diese Aussage bezieht sich ausschließlich auf OpenUP, EPF/UMA. Für kommerzielle Ausprägungen, zum Beispiel den RUP, stehen umfassende Werkzeuge zur Verfügung.

15 siehe hierzu Manifesto for Agile Software Development: `http://www.agilemanifesto.org`

an denen sie sich orientieren. Während XP jedoch eine weitgehend Management-freie Software-Entwicklung fokussiert, ist Scrum an einigen Stellen durch definierte Managementtechniken angereichert. Scrum ist somit auch ein *Managementprozess*.

Einen weiteren Managementprozess definiert Prince 2 (Projects in Controlled Environments, [Köh06]). Prince 2 ist ein allgemeiner Managementprozess, der durch das Office of Government Commerce (OGC) in Großbritanien veröffentlicht und getrieben wird. Prince 2 ist ähnlich generisch wie das V-Modell XT, verfolgt jedoch einen komplett anderen Ansatz. Anstelle einer Ergebnisdefinition liegt Prince 2 ein Modell zugrunde, das Phasen (zum Beispiel *Implementation*) und Prozesse (zum Beispiel *Planning*) beschreibt. Gemeinsam haben beide jedoch eine produktbasierte Planung und Fortschrittskontrolle.

Die oben betrachteten agilen Methoden und Prince basieren nicht auf einem expliziten, formalen Metamodell. Ein Vorgehensmodell, das auf einem formalen Metamodell basiert, das wir hier aber nicht betrachten, ist der oose Engineering Process (OEP, [OSKZ06]). Dieses Vorgehensmodell ist ebenso wie MSF ein firmenspezifisches Vorgehensmodell, das jedoch nicht durch Werkzeuge unterstützt wird und auch nicht sonderlich verbreitet ist. Genauso wie OpenUP und RUP ist OEP UML-basiert und fasst Techniken zur OO-Geschäftsprozessanalyse und -Modellierung zusammen.

Prozessmuster

Prozessmuster (Process Pattern) sind wiederverwendbare Teilprozesse, die zu umfassenderen Teilprozessen und Prozessen zusammengefasst werden können. Prozessmuster sind in einigen Arbeiten im akademischen Umfeld zu finden, zum Beispiel in [BRSV98, GMP+03, Mün01]. Industriell umgesetzt werden Prozessmusteransätze jedoch nur vereinzelt und in stark interpretierter Form. Beispiele für solche musterbasierten Konzepte finden sich im V-Modell XT mit seinen Vorgehensbausteinen oder im OpenUP mit seinen Capability Pattern.

Reifegradmodelle

Reifegradmodelle stellen Fähigkeiten und Fertigkeiten von Unternehmen oder Organisationen allgemein fest und bieten eine Skala zur Bewertung an. Insofern unterscheiden sie sich von Entwicklungsmodellen dahingehend, als dass hier keine Produkte erstellt werden, sondern die *Prozessperformance* festgestellt wird. Das wohl bekannteste Reifegradmodell ist das Capability Maturity Model Integration (CMMI, [Kne06]), das allgemein der Beurteilung und Verbesserung der Reife (im Sinne von Qualität) von allgemeinen Entwicklungsprozessen dient. Somit definiert CMMI keine Anforderungen an einen Entwicklungsgegenstand, sondern an einen (guten) Entwicklungsprozess. CMMI verfügt über eine fünfstufige Skala ($1 \ldots 5$), die der Einstufung eines Prozesses dient (das V-Modell XT beispielsweise erreicht die Stufen 2 oder 3, je nach Anpassungs- und Anwendungsstand).

CMMI unterscheidet sich somit von einem weiteren sehr bekannten Reifegradmodell, der *DIN/ISO 9000*'er Serie[16], insbesondere der DIN 9001:2000[17]. Diese zielt auf eine Organisation als Ganzes ab, während CMMI Entwicklungsprozesse fokussiert. Noch spezieller als CMMI ist *Spice* (Software Process Improvement and Capability Determination, ISO/IEC 15504), das auch einzelne Prozesse beurteilen kann. Köhler [Köh06] bezeichnet „...Spice als spezialisiertes CMM[18] für den Bereich der Softwareentwicklung".

16 ISO 9000'er Serie: Ist auf die im militärischen Bereich noch anzutreffenden AQAP's – Allied Quality Assurance Procedures – zurückzuführen. Für den Bereich Software besonders relevant sind die AQAP 150 und 160 (`http://www.nato.int/docu/stanag/aqap160/aqap160-e.htm`), die für die Anpassung des V-Modell XT für die Bundeswehr berücksichtigt werden mussten, beziehungsweise bereits deren Nachfolger der AQAP 2000'er Serie. Weitere verwandte Normen und Standards sind über die NATO Online Bibliothek verfügbar: `http://www.nato.int/docu/standard.htm`.

17 Die DIN EN 9000'er wird durch das Deutsche Institut für Normung e.V.: `http://www.din.de` vertrieben.

18 CMM – Capability Maturity Model. Vorläufer des CMMI, der auf Watts Humphrey (IBM, etwa 1980) zurückgeführt wird.

2.2. Entwicklung und Anpassung von Vorgehensmodellen

Nachdem wir einige Vertreter der Vorgehensmodelle und ihre Basiskonzepte vorgestellt haben, wollen wir nun unseren Fokus auf die Entwicklung, Anpassung und mögliche Lebenszyklen von Vorgehensmodellen lenken. Wir betrachten noch einmal die drei zuvor vorgestellten Vorgehensmodelle und analysieren ihre Entwicklungs- und Anpassungsprozesse.

2.2.1. Klassifikation unter Anpassungsaspekten

Um Vorgehensmodelle tragfähig klassifizieren zu können, sind Merkmale mit entsprechenden Skalen zu identifizieren und zu benennen. Die Literatur tut sich jedoch schwer, einen vollständigen Katalog anzubieten und liefert Merkmale und Kriterien zur Klassifikation und Bewertung immer nur auszugsweise [Chr92, Gna05, FK07]. Auch im Kontext dieser Arbeit ist es sinnvoll, eine Menge angemessener Merkmale verfügbar zu haben. Aufgrund unserer Ausrichtung, können wir in weiten Teilen auf inhaltliche Merkmale [FK07] verzichten. Wir orientieren uns an Gnatz [Gna05], der folgende Merkmale zur Klassifikation von Vorgehensmodellen auswählt:

- *Anpassbarkeit*,
- Inhaltliches Spektrum,
- Abstraktionsgrad,
- Anwendung und Werkzeugunterstützung sowie
- Projektdurchführungsstrategien.

Wir können im Kontext dieser Arbeit noch weitere Einschränkungen vornehmen (siehe Hervorhebung), da uns vorrangig Fragen der Flexibilität, Modularität und Anpassbarkeit interessieren. Für die Konstruktion eines Vorgehensmodells auf der strukturellen Ebene ist das inhaltliche Spektrum zunächst nicht relevant. Dasselbe gilt zunächst auch für den Bereich Projektdurchführungsstrategie. Auf der strukturellen Ebene existieren Abläufe nur in Form einer Elementtypdefinition (zum Beispiel Entscheidungspunkt, Meilenstein oder Phase). Konkrete Abläufe ordnen wir eher auf einer inhaltlichen Ebene ein.

Anpassbarkeit. Die Anpassbarkeit eines Vorgehensmodells ist auf verschiedenen Ebenen zu betrachten und stellt generell eine der wichtigsten Eigenschaften eines Vorgehensmodells dar. Die Anpassbarkeitseigenschaften dienen aber nicht nur der Individualisierung, sondern auch der Anpassung auf konkrete Projektbedürfnisse und Anwenderanforderungen. Anpassungen eines Vorgehensmodells können auf der

- *inhaltlichen Ebene* oder auf der
- *strukturellen Ebene*

vorgenommen werden. Ersteres findet sich üblicherweise in der so genannten „projektspezifischen Ausgestaltung", also der pragmatischen Anpassung und Interpretation eines Vorgehensmodells in konkreten Projektkontexten. Auch das Bearbeiten beispielsweise von Beschreibungstexten oder das Anpassen von Vorlagen fällt in den Bereich der inhaltlichen Ausgestaltung. Prozessmusteransätze [BRSV98, GMP+03] versuchen dies zu abstrahieren und zu formalisieren. Strukturelle Anpassungen existieren in zwei Ausprägungen:

1. Es werden (existierende) Strukturen auf der Ebene eines Vorgehensmodells angepasst – zum Beispiel das Hinzufügen einer weiteren Aktivität zu einem Workflow[19].
2. Es werden Strukturen auf der Ebene des Vorgehensmetamodells angepasst (zum Beispiel die Definition eines neuen Relationstyps).

19 Zu beachten ist bei diesem Punkt jedoch, dass je nach Anlage des betreffenden Vorgehensmodells die Änderungen von Strukturen auf der Modellebene auch zu impliziten, inhaltlichen Anpassungen führen können.

Münch [Mün01] benennt hierfür „Kompositions- und Generierungsansätze". Auch in [Chr92] finden sich Aussagen zu strukturellen Anpassungen von Vorgehensmodellen. In Tabelle 2.1 fassen wir *Anpassungsgegenstände* zusammen. Wir orientieren uns dabei an [Chr92] (Seite 135 ff.)[20].

Ebene der Anpassung	Gegenstand der Anpassung
Anpassungen des Metamodells	–
Anpassungen des Modells	parametrisierte Änderung/Modelle Interpretationsänderung Werkzeugergänzung Modifizierung der Hilfetexte Modifizierung der Schablonen Änderung von Namen der Resultate und/oder der Aktivitäten Änderung von Attributen Ergänzung/Streichung von Resultaten und Aktivitäten Methodenänderung
Anpassungen einer Instanz	parametrisierte Änderung/Modelle Werkzeugergänzung Modifizierung der Hilfetexte Ergänzung/Streichung von Resultaten und Aktivitäten

Tabelle 2.1.: Anpassungsgegenstände eines Vorgehensmodells nach [Chr92]

Wie der Tabelle 2.1 zu entnehmen ist, zielen seine Anpassungen nur auf inhaltliche Aspekte eines Modells ab. Strukturelle Änderungen, also solche auf der Metamodellebene, betrachtet er nicht[21].

Strukturelle und inhaltliche Anpassungen

Wir definieren an dieser Stelle, dass wir nur bei Anpassungen auf der Metamodellebene von strukturellen Anpassungen reden. Anpassungen auf der Modell- und auf der Instanzenebene ordnen wir den inhaltlichen Anpassungen zu.

Anders als Chroust gehen wir nicht von einer Invarianz der Metamodellstrukturen aus und definieren Anpassungsgegenstände auch auf der Ebene des Metamodells. In Tabelle 2.2 ergänzen wir hierzu unsere Anpassungsgegenstände auch für die Metamodellebene und betrachten ggf. auftretende Änderungsnotwendigkeiten für die anderen Ebenen.

Ebene der Anpassung	Gegenstand der Anpassung
Anpassungen des Metamodells	Erstellen und Anpassen von Elementtypen Erstellen und Anpassen von Assoziationstypen Definieren neuer Kompositionseinheiten
Anpassungen des Modells	s.o.
Anpassungen einer Instanz	s.o.

Tabelle 2.2.: Anpassungsgegenstände eines Vorgehensmodells mit Berücksichtigung des Metamodells

20 Chroust spricht hier nur von „häufigen" Änderungen an Vorgehensmodellen. Wir adaptieren jedoch seine Änderungsliste und ordnen sie in entsprechende Klassen unsererseits ein und ergänzen ggf.

21 Dies mag daran liegen, dass Metamodell-basierte Vorgehensmodelle noch vergleichsweise neue Erscheinungen sind.

2.2.2. Ausgewählte Entwicklungs- und Anpassungsprozesse

Wir betrachten in diesem Abschnitt die Entwicklungs- und Anpassungsprozesse der zu Beginn dieses Kapitels vorgestellten Vorgehensmodelle. Keines der hier betrachteten Vorgehensmodelle definiert einen Prozess zu seiner eigenen Ausgestaltung. Vielmehr ziehen sie sich in der Regel auf die Werkzeuge zurück, und lassen im Rahmen eines Anpassungsprozesses alles zu, was die Werkzeuge ermöglichen. Das V-Modell XT definiert zwar einen Einführungs- und Pflegeprozess für Vorgehensmodelle, der aber so allgemein ist, dass er für viele Prozessmodelle anwendbar ist. Aufgrund dieser Generik beachtet er aber wiederum keine Besonderheiten des V-Modells, womit er das V-Modell auch nicht weiter unterstützt als andere Modelle.

Anpassung des V-Modell XT

Das V-Modell ist ein (per Definition) anpassbares Vorgehensmodell. Es basiert auf einem Metamodell, welches eine Sprache für das V-Modell definiert. Alle verfügbaren Sprachelemente sind dementsprechend im Rahmen einer Anpassung des V-Modells verwendbar. Mit den Referenzwerkzeugen haben Prozessingenieure alle notwendigen Werkzeuge für die Anpassung verfügbar. Das V-Modell setzt einen stufenweisen Anpassungsprozess (vgl. auch Stufenkonzept beim Tailoring, Kapitel 1.2) um. Er unterscheidet zwischen:

- *organisationsspezifischer* und
- *projektspezifischer* Anpassung.

Beide Stufen werden durch entsprechende Werkzeuge unterstützt. In der Abbildung 2.4 haben wir die Abbildung von Schrittfolgen sowie Ereignissen, In- und Outputs vorgenommen. Wir gehen an dieser Stelle auf den Anpassungprozess ein. Er erleichtert – insbesondere im Kapitel 5.2 – das Verständnis zum Lebenszyklus eines Vorgehensmodells.

Anpassungsprozess. Ausgangspunkt für die Anpassung ist ein V-Modell XT (hier als Standard bezeichnet; es kann sich dabei um ein beliebiges, nicht angepasstes V-Modell handeln). Dieses ist Eingabe für den V-Modell XT Editor (Abbildung 2.4, 1a). Der Editor ist das Werkzeug für den Prozessingenieur, der die Anpassung vornimmt. Nachdem die Anpassungsarbeiten abgeschlossen sind, liegt ein organisationsspezifisches V-Modell XT vor. Dieses kann wieder Eingabe für den Editor für einen weiteren Anpassungsschritt sein (Abbildung 2.4, 1b). Alternativ ist es Eingabe für den V-Modell XT Projektassistenten (Abbildung 2.4, 2a), der die projektspezifische Anpassung unterstützt. Neben dem organisationsspezifischen V-Modell erhält der Projektassistent noch Mustertexte und ein Mastertemplate (Abbildung 2.4, 2b-c) als Eingabe.

Mustertexte sind wiederverwendbare „Textpassagen" und dienen der initialen Produktausgestaltung der Vorlagen. Es handelt sich dabei nicht um Elemente des V-Modells, sondern um ein eigenständiges Konzept[22], das durch den Projektassistenten zur Verfügung gestellt wird. Das Mastertemplate ist die Dokumentenvorlage, die für die Produktvorlagen verwendet wird. Mustertexte und Mastertemplate können ebenfalls im Rahmen einer organisationsspezifischen Anpassung zum Beispiel an Corporate Identity angepasst werden. Der Projektassistent ist das Werkzeug des Projektleiters, der das Tailoring durchführt. Es erzeugt die Ausgaben, die den Start des Projekts ermöglichen (Abbildung 2.4, 3). Dies sind das bereits weiter oben erwähnte projektspezifische V-Modell, die Produktbibliothek und der initiale Projektplan.

Wie der Abbildung 2.4 zu entnehmen ist, sind die Ein- und Ausgaben der Anpassungsprodukte durch das V-Modell-Schema definiert. Der Anpassungsprozess selbst ist jedoch *gerichtet*. Eine Rückkopplung ist durch ihn nicht vorgesehen, sondern erfordert organisationsinterne Regelungen. Für das Standard V-Modell XT, das über die Webseite des Bundesministeriums des Innern (BMI) [vmx] beziehbar ist, ist an der gleichen

22 Es ist jedoch geplant, Mustertexte direkt in das V-Modell mit aufzunehmen und somit diese Funktionalität vom Werkzeug zu entkoppeln.

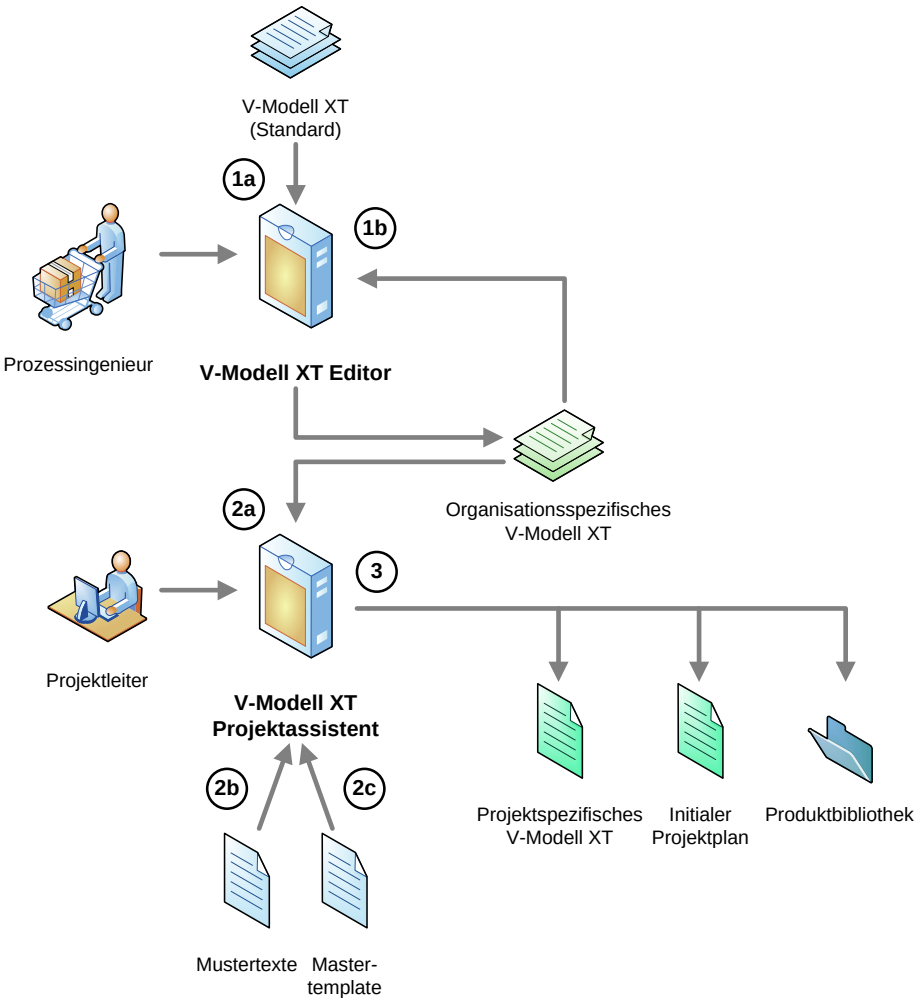


Abbildung 2.4.: Standardwerkzeugkette des V-Modells inklusive der standardmäßigen Anpassungsprodukte

Stelle auch ein Änderungsprozess hinterlegt. Dieser gilt jedoch nur für das Standardmodell beziehungsweise die zentral gepflegten Anpassungen des öffentlichen Bereichs. Eine Verknüpfung zwischen den dort verwalteten Versionen und weiteren organisationsspezifischen Varianten ist jedoch nicht definiert.

Anpassung des Microsoft Solutions Framework

MSF kennt kein Tailoring, so wie es das V-Modell XT versteht, kann aber trotzdem weitgehend angepasst werden. Die erste Anpassungsstufe verläuft ähnlich dem V-Modell: Auf der Basis des Metamodells sind verschiedene Ausprägungen ableitbar (vgl. Abbildung 2.3). Dies kommt dem Gedanken des V-Modell XT Grund- beziehungsweise Referenzmodells sehr nahe. MSF Anwender erhalten also im Wesentlichen schon ein vorgetailortes Vorgehensmodell. Darüber hinaus kann eine konkrete MSF-Instanz noch weiter an die Anforderungen eines Projekts angepasst werden. Den zugrunde liegenden Prozess sowie die beteiligten Ein- und Ausgaben diskutieren wir im Folgenden.

Anpassungsprozess. In Abbildung 2.5 haben wir den Anpassungsprozess des MSF visualisiert. Eine entsprechende Anleitung zur Anpassung des MSF ist auf den Webseiten[23] des Microsoft Developer Network (MSDN) verfügbar. Ausgangspunkte sind

23 Online Guidance für die Anpassung des MSF: `http://msdn2.microsoft.com/en-us/library/ms364097(VS.80).aspx`

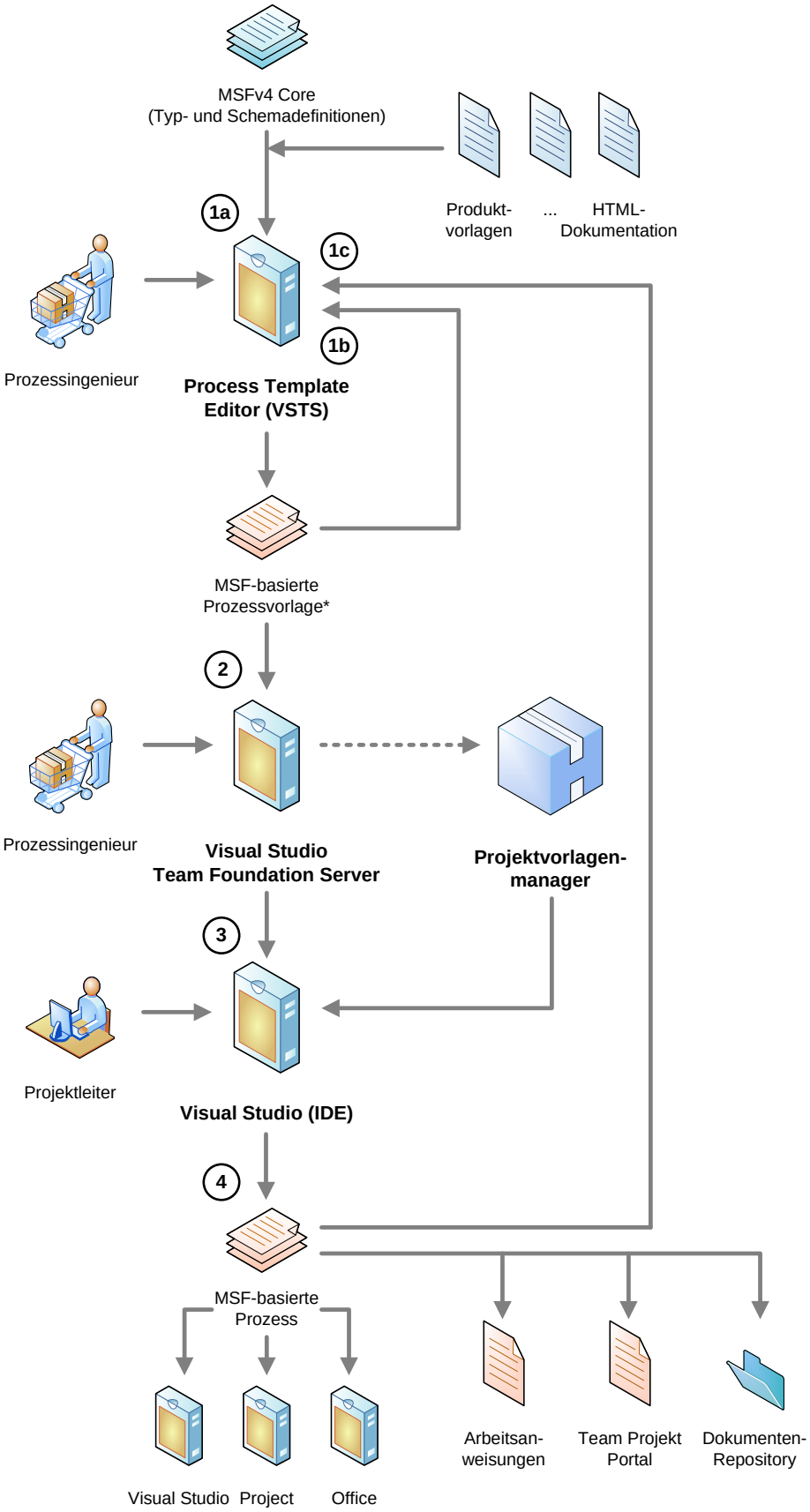


Abbildung 2.5.: Werkzeugkette des MSF mit Anpassungsprodukten für die Anpassung/Ausgestaltung des Prozesses

die einzelnen Bestandteile des Prozesses, das Metamodell des MSFv4 Core und die inhaltlichen Anteile. Diese bilden die Eingaben für den Process Template Editor (PTE, Abbildung 2.5, 1a), das Werkzeug des Prozessingenieurs. Dieser fügt alle Teile des Prozesses zusammen (Assemblierung) und erstellt eine MSF-basierte Prozessvorlage (Process Template). Das Process Template kann als Produkt bereitgestellt oder vertrieben werden. In Abbildung 2.3 sind im linken unteren Teil verschiedene Templates gezeigt. Diese können mit dem PTE auch wieder eingelesen und weiterverarbeitet werden (Abbildung 2.5, 1b). Der abschließende Schritt in dieser Phase ist die Bereitstellung des Prozesses in der Zielumgebung (Deployment). Hierzu ist die Prozessvorlage in den Team Foundation Server (TFS) zu importieren (Abbildung 2.5, 2). Gegebenenfalls, sofern erforderlich ist auch noch eine Anpassung der durch TFS bereitgestellten Vorlage des SharePoint Portals vorzunehmen. Nach erfolgreichem Import wird die Prozessvorlage durch den Prozessvorlagenmanager verwaltet. Neue Teamprojekte können nun auf der Basis des installierten Prozesses angelegt werden.

Diese erste Phase findet entweder zentral und organisationsübergreifend statt (zum Beispiel beim zentral verwalteten MSF) oder zumindest auf der Organisationsebene. Die gerade beschriebenen Anpassungsschritte entsprechen somit der organisationsspezifischen Anpassung des V-Modell XT. Ein weiteres, automatisches und projektspezifisches Tailoring ist durch MSF nicht vorgesehen. Projektleiter wählen zu Beginn eines Projekts eine der angebotenen Vorlagen des Vorlagenmanagers aus (Abbildung 2.5, 3). In (technischen) Details ist die Auswahl noch zu beeinflussen, jedoch wird aus der Vorlage eine entsprechende Projektinfrastruktur automatisch generiert (Abbildung 2.5, 4). Diese umfasst eine Menge generierter Standardarbeitsaufgaben, ein Team Projekt Portal sowie ein Dokumenten-Repository. Obwohl kein automatisches Tailoring zu Beginn eines Projekts vorgesehen ist, können auch im laufenden Projekt Änderungen notwendig werden. Hier ist es möglich, die aktuelle Prozessinstanz zu exportieren, anzupassen und wieder zu importieren (Abbildung 2.5, 1c).

Konzept der Anpassung des MSF. MSF ist in weiten Teilen anpassbar, unterscheidet sich jedoch in der Philosophie vom Anpassungskonzept des V-Modells. MSF ist nicht generisch. Es gibt eine allgemeine Version des MSF (MSFv4 Core, Abbildung 2.3, oben), die jedoch nicht öffentlich zugänglich ist. Zugänglich sind nur die konkreten Instanzen des MSF, die Gegenstand einer Anpassung sind. Beim V-Modell ist das Basismodell selbst Gegenstand der Anpassung. Ein automatisches Tailoring auf projektspezifische Anforderungen wird durch MSF nicht angeboten, da es sich bei den MSF-Instanzen bereits um konkrete, problembezogene Prozesse handelt. Anwender können frei entscheiden und anpassen, jedoch werden keine Feedbackschleifen für den Basisprozess definiert. Eine Aktualisierung eines angepassten und eingeführten MSF-basierten Prozesses ist unter Umständen sogar ausgeschlossen, wenn einmal in der Datenbank des TFS hinterlegte Prozesselemente geändert wurden.

Anpassung des Open Unified Process

Die Fähigkeiten des Werkzeugs für die Prozesserstellung und -anpassung sind intensiv ausgeprägt. So ist OpenUP selbst nur eine mögliche Spielweise im EPF. EPF und insbesondere UMA sind somit auf einer Ebene mit dem V-Modell XT anzusiedeln. Es steht ein allgemeines Metamodell zur Beschreibung verschiedener Prozesse und ein passendes Werkzeug zur Erfassung und Dokumentation zur Verfügung. In Abbildung 2.6 haben wir den vereinfachten Anpassungsprozess auf der Basis EPF/UMA illustriert. Eine detaillierte „Schritt für Schritt"-Dokumentation ist in [Hau06a, Hau06b] zu finden. Voraussetzungen für ein Prozessdesign auf der Basis von EPF sind die Unified Method Architecture (UMA) und diverse Inhalte und Vorlagen, die nicht mithilfe von Eclipse erstellt werden können (üblicherweise Office-Dokumente). Diese Bestandteile müssen zur Verfügung stehen, wenn mithilfe von EPF ein Prozess entwickelt/beschrieben werden soll (Abbildung 2.6, 1a). Das Werkzeug für den Prozessingenieur ist die Eclipse Workbench, für die der EPF-Composer als Plug-In zur Verfügung steht. Auf Basis der gegebenen Inhalte und Vorlagen erstellt der Prozessingenieur so genannte *Method Plug-Ins* und *Konfigurationen*. Method Plug-Ins sind Bausteine eines Prozesses verschiedener

Granularität. Sie können Einzeldisziplinen ebenso kapseln wie vollständige Prozesse. OpenUP ist beispielsweise als Method Plug-In `openup_basic` gefertigt. Eine Konfiguration nach EPF fasst ähnlich wie ein V-Modell XT Projekttyp für einen bestimmten Kontext relevante Anteile der Process/Method Library zusammen. Eine Konfiguration ist somit eine *Sicht* auf der Process/Method Library. In der Process/Method Library werden alle Konfigurationen und alle Method Plug-Ins gesammelt. Diese Bibliothek ist eine Datenablage, in der die Method Plug-Ins als physisch separate Einheiten (in Form einer Verzeichnisstruktur) verwaltet werden. Sämtliche Elemente, die in der Process/Method Library enthalten sind, können auch wieder im EPF-Composer bearbeitet werden (Abbildung 2.6, 1b).

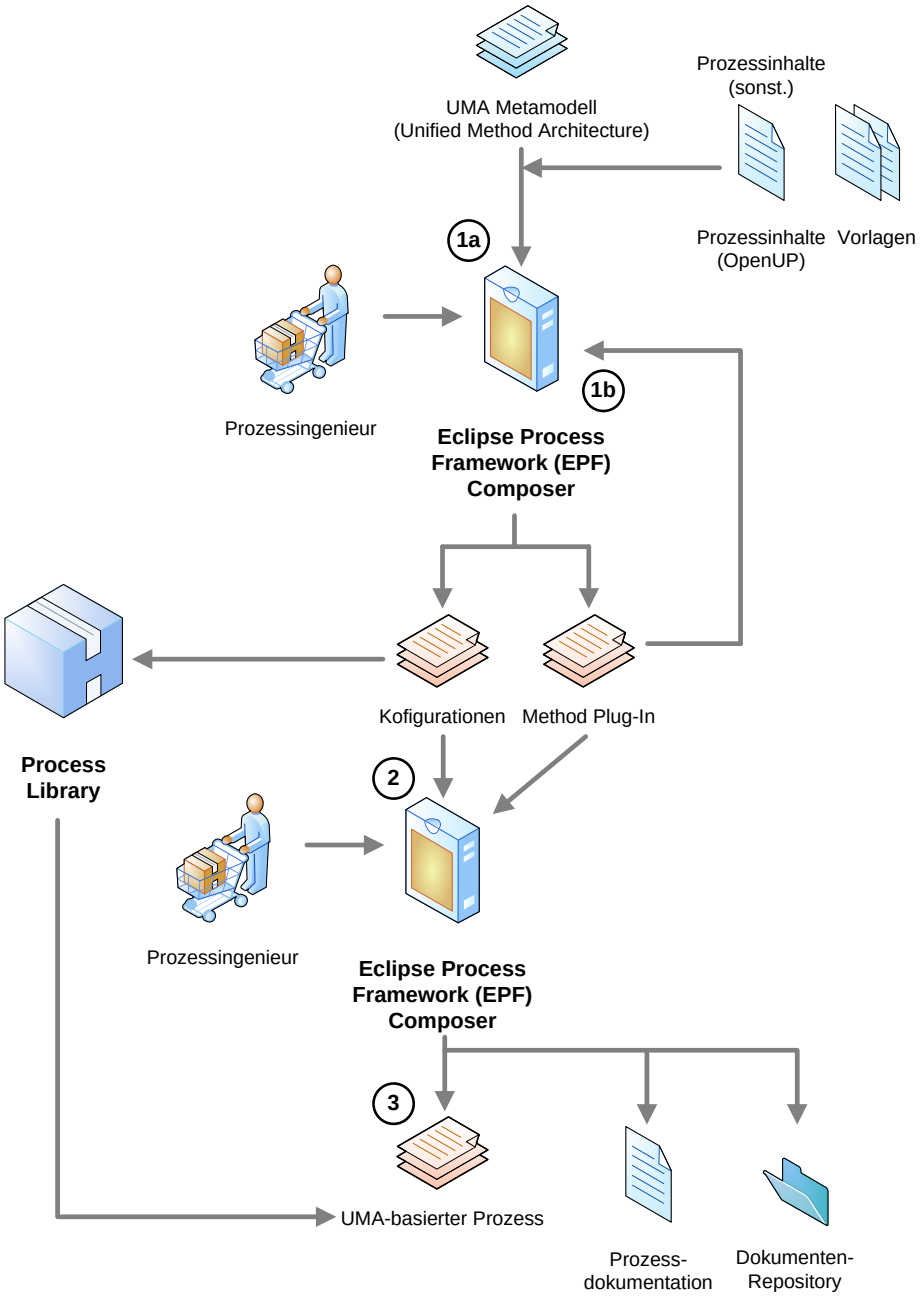


Abbildung 2.6.: EPF-Composer und OpenUP mit Anpassungsprodukten für die Anpassung/Ausgestaltung von Prozessen

Liegen Konfigurationen und Method Plug-Ins vor, können weitere Spezialisierungen vorgenommen werden (Abbildung 2.6, 2). Hierzu können in einer Konfiguration In-

halte vorhandener Method Plug-Ins referenziert werden. Ein Export der Konfiguration (Abbildung 2.6, 3) liefert dann ähnlich wie das V-Modell XT eine Web-basierte Dokumentation sowie eine Menge Produktvorlagen (die hier jedoch nicht dynamisch erstellt und kontextsensitiv angepasst werden). Eine entsprechende Konfiguration kann gespeichert und wieder aufgerufen werden. Dieses Anpassungsverfahren entspricht im Wesentlichen dem des MSF. Ein weiter gehendes automatisches, parametrisierbares Tailoring für einen spezifischen Projektkontext ist durch die Kombination OpenUP/EPF nicht vorgesehen. Auffällig bei diesem Anpasssungsverfahren ist, dass es ausschließlich Prozessingenieure adressiert. Ein Projektleiter findet sich hier nicht wieder, was unter anderem auch auf das Fehlen einer integrierten Werkzeugkette (siehe letzter Abschnitt) zurückzuführen sein mag.

OpenUP liegt für den Anpassungsprozess ein konfigurationsbasiertes Muster zugrunde, das Ähnlichkeiten zu dem in dieser Arbeit verwendeten besitzt. Ein wesentlicher Punkt, in dem wir uns von OpenUP, EPF und UMA unterscheiden ist jedoch die *Art der Konfiguration*. Im Kapitel 3.1 diskutieren wir dies direkt auf dem zugrunde liegenden Metamodell und führen eine entsprechende Positionierung und Einordnung durch.

2.3. Lifecycle- und Lebenszyklusmodelle für Vorgehensmodelle

Um ein für Vorgehensmodelle adäquates Lebenszyklusmodell zu definieren sind viele Ansatzpunkte aus der Produkterstellung [BKPS04, HS03], dem Prozessmanagement [FD02] oder auch der Geschäftsprozessmodellierung [Thu04] zu berücksichtigen. Da Vorgehensmodelle in der Regel immer Teilbereiche der gerade aufgeführten Disziplinen berühren, finden sich somit auch Elemente von Lebenszyklusmodellen in Vorgehensmodellen wieder. Oftmals sind jedoch die Rollen nicht eindeutig verteilt, sodass in weiten Bereichen Wechselwirkungen zu finden sind. So ist es üblich, dass sich Vorgehensmodelle (in Form konkreter Entwicklungsmodelle) auf den Produkterstellungsprozess auswirken, während die Art einer Produkterstellung, insbesondere unter Berücksichtigung externer Effekte, gleichzeitig wieder auf ein Vorgehensmodell wirkt.

2.3.1. Phasen des Lebenszyklus

Phasen des Lebenszyklus von Vorgehensmodellen sind relativ leicht zu identifizieren:

- Soll-/Istanalyse
- Erstellung/Gestaltung
- Pilotierung (punktuelle Einführung und Erprobung)
- Breiteneinführung

Diese Phasen sind jedoch nicht immer problem- oder lösungsangemessen, da sie viel zu grob granular sind. Wir gehen hier pragmatischer zu Werke und orientieren uns zur Identifikation von Phasen deshalb an zwei Größen, die wir zum Teil der Softwareentwicklung entnehmen.

1. Die erste Größe sind die *Operationen* zur Gestaltung beziehungsweise Anpassung eines Vorgehensmodells. Einen Einblick haben wir im letzten Abschnitt 2.2.1 gegeben. Wir stellen hier die Frage: *Was kann zu welchen Zeitpunkten in einem Vorgehensmodell angepasst werden und was sind die Konsequenzen*? Chroust [Chr92] gibt uns hier bereits einen ersten Katalog an die Hand (vgl. auch Tabelle 2.1), den wir aus unserem Erfahrungsschatz (Anhang A) ergänzen.
2. Die zweite Größe, die wir hier in Betracht ziehen müssen liefern uns integrierte Entwicklungs- und Betriebsmodelle, wie wir sie beispielsweise beim MSF [Tur06] finden. Sie definieren bereits etablierte und anerkannte Phasen und darüber hinaus auch Schnittstellen, die eine (kontinuierliche) Weiterentwicklung eines Produkts durch entsprechend definierte Anforderungs-, Änderungs- und Verteilungsprozesse ermöglichen.

Vorgehensmodell = Produkt

Es ist *essenziell*, dass wir ein Vorgehensmodell als Produkt auffassen. Die Charakterisierung als Produkt gestattet es uns, konkrete Parameter/Eigenschaften zu bestimmen und diese für eine Veränderung zu verwenden. Im Gegensatz zu einem „zeitlosen" Konzept stehen uns so konkrete Eigenschaftstypen zur Verfügung, deren Exemplare sich neben einer konkreten Wertbelegung gleichzeitig durch eine zeitliche Konstante definieren.

Der Rückgriff auf die Produktlinie als Bezugspunkt liefert darüber hinaus noch weitere Eigenschaften, wie zum Beispiel zeitlich stabile Basiskomponenten, Soll-Bruchstellen etc. Am Beispiel des V-Modell XT und des MSF stellen wir die vorhandenen Anpassungsprozesse noch einmal vor und positionieren sie in einem übergeordneten Lifecycle.

Einführung und Pflege eines organisationsspezifischen Vorgehensmodells

Das V-Modell XT definiert eigens zur Einführung und Pflege von Vorgehensmodellen[24] einen eigenen Projekttyp. Die mit diesem Projekttyp einhergehende Projektdurchführungsstrategie *Einführung und Pflege eines organisationsspezifischen Vorgehensmodells* ist in Abbildung 2.7 zu sehen. Sie zeigt, dass das V-Modell durch drei Entscheidungspunkte abgeschlossene Projektabschnitte für die Einführung und (kontinuierliche) Pflege eines Vorgehensmodells vorsieht. Das zugrunde liegende (nicht formalisierte) Verfahren haben wir bereits im Abschnitt 2.2.2, insbesondere in Abbildung 2.4 schon skizziert. Wir betrachten dieses Verfahren nun genauer.

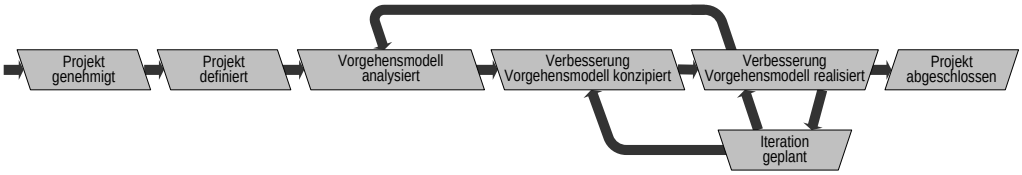


Abbildung 2.7.: Beispiel PDS: Einführung und Pflege eines organisationsspezifischen Vorgehensmodells

Das V-Modell legt[25] hier einen Analyseabschnitt, einen Konzeptionsabschnitt und einen Einführungs-/Realisierungsabschnitt fest, der inkrementell, iterativ durchlaufen werden kann. Die Unterteilung in diese Abschnitte gibt drei ausgezeichnete Phasen im Lebenszyklus eines Vorgehensmodells vor. Diese Unterteilung ist wichtig, zeigt sie doch auf klar definierbare Ein- und Ausgaben, die phasenabhängig nach verschiedenen Vorgängen und ggf. mit verschiedenen Werkzeugen verarbeitet werden.

Produkttypen in V-Modell-Einführungsprojekten. Das V-Modell definiert drei Produkte für Projekte mit dem Ziel einer Prozesseinführung oder -verbesserung (die Rollen lassen wir außen vor). Dies sind:

- *Bewertung eines Vorgehensmodells*, das die Ergebnisse der Ist-Analyse enthält (zum Beispiel Stärken-/Schwächenprofil, Maßnahmenkataloge für die Verbesserung etc.).
- *Verbesserungskonzept für ein Vorgehensmodell*, das aufbauend auf der Bewertung entsprechende Verbesserungsmaßnahmen auswählt und strategische Vorgehensweisen, zum Beispiel zur Pilotierung und Einführung festschreibt.

24 Das V-Modell XT definiert den Projekttyp nicht nur für sich selbst als Einführungs- und Pflegeprozess, sondern für beliebige Vorgehensmodelle. So ist es beispielsweise möglich, einen RUP mithilfe eines V-Modell-konformen Projekts einzuführen.

25 Das V-Modell XT legt diese Abschnitte nur fest, gestaltet sie jedoch nicht aus. Einen definierten Anpassungsprozess, der über die Prozesse des WEIT-Teams hinausgeht und vom Know-how des Projektteams angewendet werden kann, gibt es jedoch nicht. Hier gibt es parallel verschiedene Dienstleister, die eigene Prozesse zur organisationsspezifischen Anpassung des V-Modells anwenden.

- *Organisationsspezifisches Vorgehensmodell*

Das organisationsspezifische Vorgehensmodell ist in diesem Projekttyp das Kernprodukt, das im Rahmen einer Einführung, Anpassung oder Verbesserung kontinuierlich bearbeitet wird. Dabei wird insbesondere bei diesem Produkt klar herausgestellt, dass Vorgehensmodelle komplexe Produkte sind, wie die Tabelle 2.3 zeigt.

Produkttyp	Themen
Organisationsspezifisches Vorgehensmodell	
	Prozessbeschreibungen
	Metrikkatalog
	Erfahrungsdatenbasis
	Schulungskonzept
	Schulungsunterlagen
	Organisationsspezifische Vorgaben und Informationen
	Produktvorlagen

Tabelle 2.3.: Themenstruktur eines organisationsspezifischen Vorgehensmodells nach [vmx]

Die Themen des Produkts *Organisationsspezifisches Vorgehensmodell* zeigen hier sehr genau auf die durchzuführenden Aktivitäten, die bei Anpassungsprozessen anfallen. Im weiteren Verlauf, werden wir einen entsprechenden Abgleich mit den Änderungsgegenständen nach Chroust (Tabelle 2.1) durchführen.

Vorgänge in V-Modell-Einführungsprojekten. Neben den drei Produkten liefert das V-Modell auch Aktiviätsbeschreibungen hoher Abstraktion für die Aktivitäten *Vorgehensmodell bewerten* und *Organisationsspezifisches Vorgehensmodell erstellen, einführen und pflegen*. Wir betrachten diese Aktivitätsbeschreibungen in einem ersten Schritt genauer.

Die Abbildung 2.8 fasst die beiden durch das V-Modell beschriebenen Prozesse zusammen und stellt sie in der V-Modell-Notation dar. Beide Aktivitäten sind durch das Produkt *Bewertung eines Vorgehensmodells* miteinander verbunden. Der Bewertungsvorgang, also die Ist-Analyse in Abbildung 2.8 (oben) kann nach verschiedenen Techniken erfolgen (zum Beispiel mithilfe von CMMI [Kne06]). Der Vorgang ist zum Erreichen des Entscheidungspunkts *Vorgehensmodell analysiert* (Abbildung 2.7) auszuführen. Er kann mehrmals durchlaufen werden, um beispielsweise Zwischenziele zu evaluieren und ggf. neue Ziele zu definieren beziehungsweise vorhandene zu justieren. Der zum Entscheidungspunkt *Verbesserung Vorgehensmodell realisiert* auszuführende Vorgang (Abbildung 2.8 (unten) ermöglicht ebenfalls einen Mehrfachdurchlauf, was der schrittweisen Umsetzung/Realisierung einer Einführung beziehungsweise Verbesserung entspricht. Der Vorgang Abbildung 2.8 (unten) findet sich jedoch in der Beschreibung der Projektdurchführungsstrategie in Abbildung 2.7 nicht unmittelbar wieder. So wird beispielsweise in der Projektdurchführungsstrategie keine explizite Nennung und Unterscheidung zwischen Erstellung, Pilotierung und breiter Einführung eines Vorgehensmodells vorgenommen. Diese werden zum Entscheidungspunkt *Verbesserung Vorgehensmodell realisiert* abstrahiert. An dieser Stelle wirkt sich die Methodenneutralität des V-Modells besonders negativ aus, da das V-Modell zwar einen Einführungs- und Verbesserungsprozess enthält, diesen jedoch nicht hinreichend detailliert[26].

Nichts desto trotz können wir (verallgemeinernd) folgende Phasenabdeckung im V-Modell feststellen: Das V-Modell unterstützt die *Feststellung* vom Bedarf und einer entsprechenden Planung sowie die *Umsetzung* der Anforderungen an ein Vorgehensmodell. Die *Pilotierung* und *Einführung* wird erwähnt, jedoch nicht ausführlich betrachtet.

26 Die Detaillierung des Anpassungsprozesses zieht eine entsprechende Ausgestaltung einerseits und eine Einschränkung andererseits nach sich. In Anbetracht der Sichtbarkeit des V-Modells muss ein solcher Prozess mit entsprechender Würdigung der Tragweite definiert werden. Eine mögliche Ausgestaltung/Interpretation könnte die Ausführung im Abschnitt 2.2.2 darstellen, sofern sie um spezifische Eingabe- und Ergebnistypen erweitert wird.

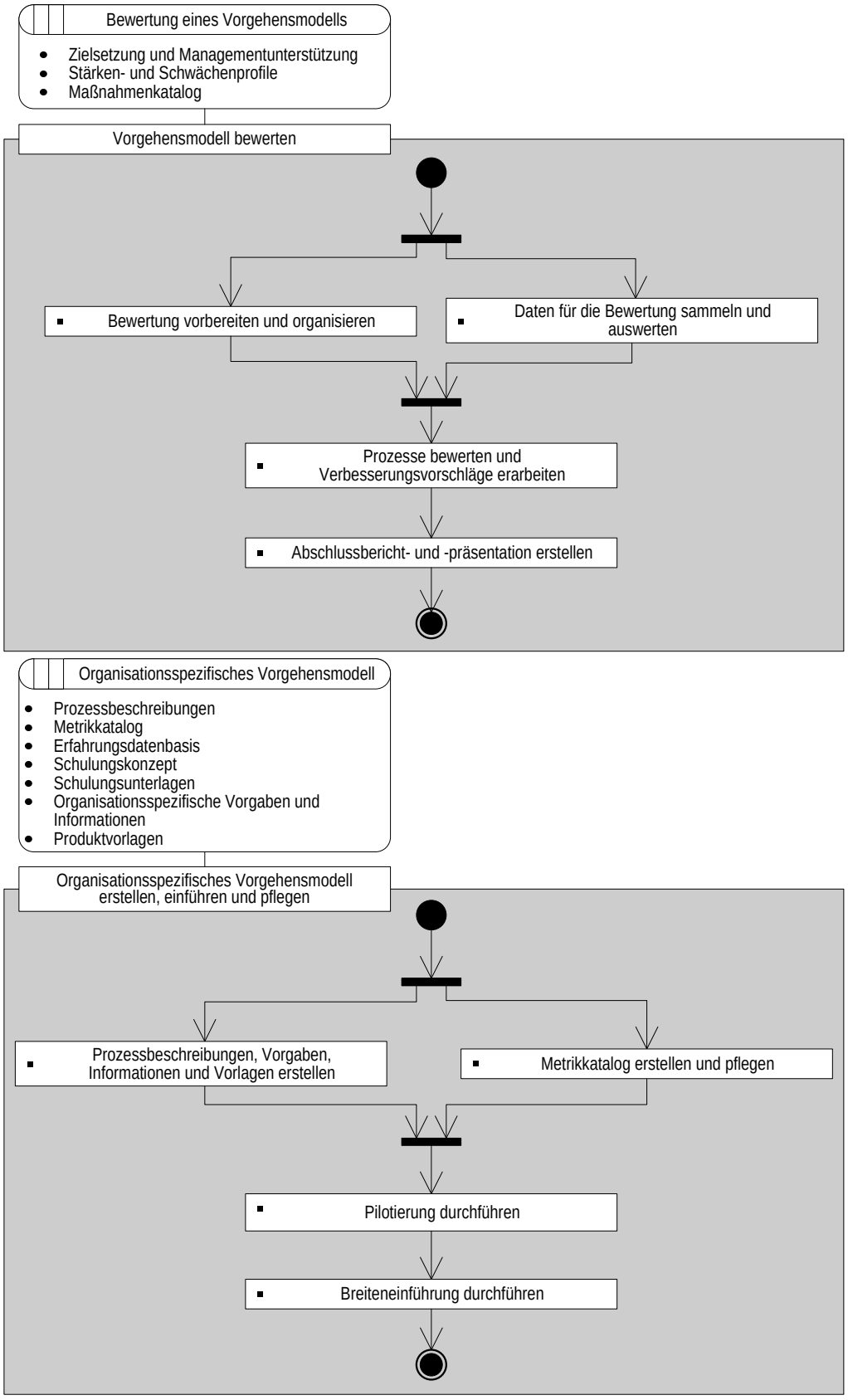


Abbildung 2.8.: Aktivitätsbeschreibungen des V-Modell XT für die Einführung und Pflege eines organisationsspezifischen Vorgehensmodells nach [vmx]

Das V-Modell formalisiert nicht, wie Feedbackschleifen, insbesondere für den Rückfluss von Anpassungen auf der Derivatebene zum Standardmodell, zu realisieren sind. Das V-Modell beschränkt des Weiteren nicht die Art der Anpassung[27].

Microsoft Solution Delivery Process

Wie in der Einführung dieses Abschnitts bereits erläutert, werden Vorgehensmodelle üblicherweise nicht mit einem Einführungs- und Pflegeprozess ausgestattet. Insbesondere Vorgehensmodelle, wie MSF oder EPF-basierte Prozesse, die an konkrete Werkzeuge (und ggf. auch Hersteller) gebunden sind, verzichten darauf und verweisen auf die Fähigkeiten der Tools[28].

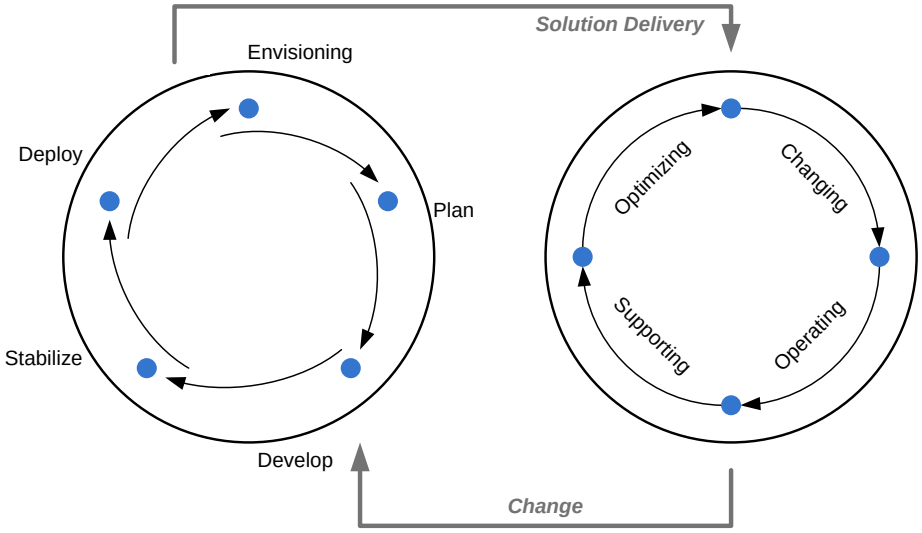


Abbildung 2.9.: Solution Delivery in MSF nach [Tur06]

Häufiger finden wir jedoch komplexe, übergreifende Lifecycle-Modelle für Software beziehungsweise allgemein für Produkte. Diese Modelle integrieren oftmals die Phasen der Entwicklung mit denen des Betriebs und stellen Schnittstellen zur Verfügung. Sie orientieren sich dabei jedoch zumeist nur punktuell an inhaltlichen Fragen von Prozessen. Ein Beispiel für einen solchen integrierten Entwicklungs- und Betriebsprozess ist das Tandem MSF und MOF, das den *Solution Delivery Process* (Abbildung 2.9) bildet. Insbesondere in den Versionen 3.x [HS03, Kee04] des MSF ist eine sehr vollständige Ausarbeitung von Prozessschnittstellen zu finden, die über die reine Systementwicklung hinausgehen und Distributions-, Pflege- und Änderungsprozesse anbinden. Wir greifen für unsere Analyse auf MSF 4 [Tur06] zurück (siehe auch Kapitel 2.1.2), da durch die Konsolidierung der Werkzeuge die Prozesse insgesamt homogener gestaltet sind/werden können. Abbildung 2.9 zeigt den integrierten *Solution Delivery Process* (SDP). Neben dem mehrphasigen Entwicklungsprozess MSF beschreibt er gleichzeitig die Einbindung der MOF Betriebsprozesse und darüber hinaus auch die Koppelstellen zwischen beiden. Eine Koppelstelle wird zwischen Verteilung (Deployment) und Änderung (Changing), die andere zwischen Betrieb (Operating) und Weiterentwicklung (Develop[29]) angegeben.

27 Die Definition der V-Modell XT-Konformität ist zum Zeitpunkt der Erstellung dieser Arbeit noch in der Erarbeitung. Bis zur Fertigstellung und abschließendem Beschluss sind Prozessingenieure frei in der Anpassung und Gestaltung des V-Modells. Lediglich die Einhaltung des V-Modell XT Metamodells, siehe Kapitel 3.2.1, wird *empfohlen*.

28 Der Verweis, dass die Anpassung eines Vorgehensmodells durch ein Tool erfolgt, ist insofern vorteilhaft, dass die Anpassung durch die Möglichkeiten des Tools gesteuert werden kann. Nachteilig daran ist jedoch, dass diese Tools in der Regel kaum methodische Aspekte realisieren, da sie zumeist für Prozessingenieure ungeeignet sind. Das Visual Studio (MSF) oder Eclipse (EPF) sind Arbeitsumgebungen für Entwickler; Editoren für Prozesse sind hier „nur" Aufsätze.

29 Wobei eine Änderungsforderung aus dem Betrieb heraus durchaus auch einen gesamten Entwicklungspro-

Prozessanpassung und SDP. Der gerade gezeigte Solution Delivery Process (SDP) verschiebt unsere Aufmerksamkeit weiter in Richtung Produkterstellung, -pflege und Weiterentwicklung. Wir wollen an dieser Stelle eine kurze Positionierung des Projekttyps *Einführung und Pflege eines organisationsspezifischen Vorgehensmodells* (kurz: Org.) im SDP vornehmen und die Abdeckung betrachten.

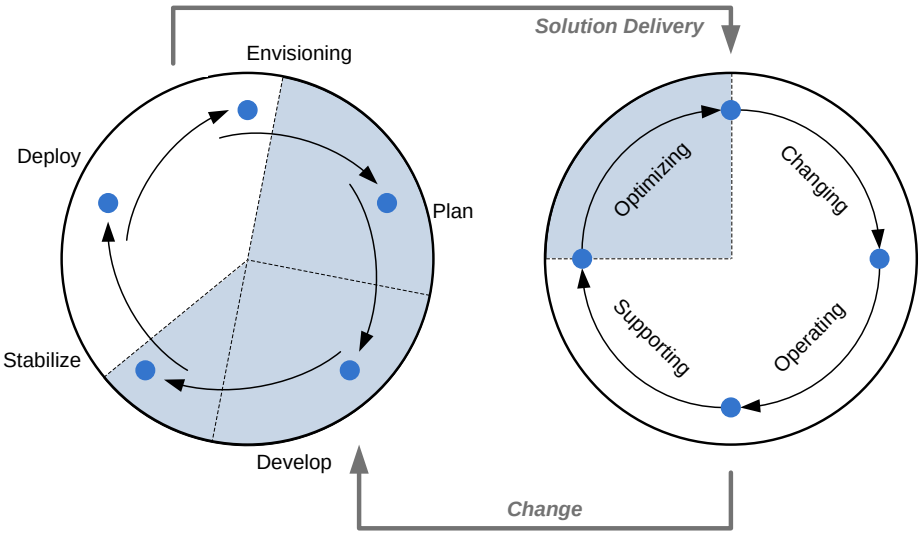


Abbildung 2.10.: Positionierung des Org.-Projekttyps des V-Modell XT im SDP

Abbildung 2.10 illustriert dies und stellt unsere Diskussionsgrundlage dar. Das V-Modell deckt mit seinem Org.-Projekttyp insbesondere Bedarfsfeststellung, Planung, Umsetzung und Optimierung um. Die Tabelle 2.4 stellt V-Modell XT und SDP direkt gegenüber. Es fällt auf, dass wir die Phase *Stabilize* nur teilweise abgedeckt sehen. Die Phase *Deploy* finden wir direkt überhaupt nicht im V-Modell.

SDP-Phase	Bezug zum V-Modell XT
Envisioning	Produkt: *Vorschlag zur Einführung und Pflege eines organisationsspezifischen Vorgehensmodells* Produkt: *Bewertung eines Vorgehensmodells*
Plan	Produkt: *Verbesserungskonzept für ein Vorgehensmodell* Produkt: *Projektplan* (sonst. Produkte und Aktivitäten zur Planung und Kalkulation)
Develop	Produkt: *Organisationsspezifisches Vorgehensmodell* Produkt: *Änderungsstatusliste*
Stabilize	Produkt: *Organisationsspezifisches Vorgehensmodell* Produkt: *Änderungsstatusliste*
Optimizing	Produkt: *Organisationsspezifisches Vorgehensmodell* Produkt: *Änderungsstatusliste*

Tabelle 2.4.: Positionierung zwischen SDP-Phasen und dem V-Modell Org.-Projekttyp

Die Phase *Stabilize* sehen wir nur zum Teil abgedeckt, weil Fragestellungen nach einer Pilotierung oder Breiteneinführung im V-Modell nicht erschöpfend behandelt werden. Gerade Sie stellen jedoch elementare Bestandteile der Feedbackschleife eines lebenden Prozesses dar. Es fällt auch auf, dass wir die Phase *Optimizing* als abgedeckt anerkennen, jedoch ordnen wir ihr genau dieselben Ergebnistypen wie der Phase *Stabilize* zu.

zess durchlaufen kann, sofern die Änderungsvolumina einen einfachen Patch übersteigen.

Dies hat seine Ursache darin, dass die Umsetzung eines organisationsspezifischen Vorgehensmodells iterativ erfolgen kann, also einen Optimierungsprozess einschließt.

Als fehlend identifizieren wir die Phasen *Deploy*, *Changing*, *Operating* und *Supporing*. Alle diese Phasen sind im Lebenszyklus zeitlich nach einer Veröffentlichung eines Vorgehensmodells einzuordnen. Der Phase *Operating* ist beispielsweise die kontinuierliche Überwachung der verwendeten Prozesse zur Laufzeit von Projekten zuzuordnen. Dies ist eine vergleichsweise neue Aufgabenstellung, in der Software- oder Projektcockpits [KMR06, GP06] ein Rolle spielen.

2.3.2. Produktlinien, Entwicklung und Evolution

Der SDP ist ein guter Anhaltspunkt, jedoch nicht der abschließende Maßstab. Wir widmen uns nun direkt dem Bereich der Produktlinien und betrachten hierbei Entwicklungsprozesse vor dem Hintergrund, dass die Entwicklungsgegenstände nicht mehr nur allein stehende Produkte sind, sondern aus komplexen Familien bestehen. Wir führen zunächst einige relevante Begriffe ein.

Produktlinienentwicklung

„. . . Die Produktlinienentwicklung ist ein Ansatz zur Softwareentwicklung mit organisierter Wiederverwendung und organisierter Variabilität auf Basis einer Plattform. *Organisierte Wiederverwendung* heißt, dass Software für die Wiederverwendung in Form einer so genannten *Plattform* entwickelt wird und dass einzelne Softwareprodukte durch Wiederverwendung aus dieser Plattform erstellt werden." [BKPS04]

Böckle et al. führen hier einerseits Plattformbegriffe ein, andererseits führen sie systematische Wiederverwendung von (Software-)Komponenten ein. Rombach [Rom05] definiert darauf aufbauend eine Komponentenbibliothek (*artifact repository*), einen passenden Wiederverwendungsprozess (*reuse process*) und einen adäquaten Managementprozess (*artifact management process*). Weiterhin definiert er zwei globale Prozesse über Produktlinien:

Prozesse über Produktlinien

„. . . Two (2) seperate development processes: One distinguishes between the domain engineering process, by which artifacts for reuse are being created, and the application engineering process, by which project-specific systems are being developed." [Rom05]

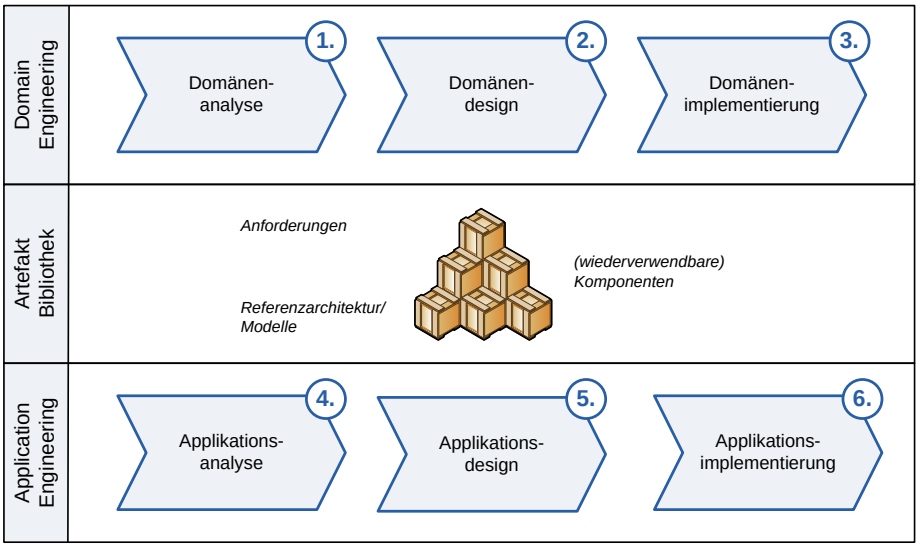
Ausgehend von [BKPS04] können wir eine (Software-)Produktlinie als Produkt *höherer* Integrationsdichte verstehen und somit Produktentwicklungsprozesse, wie zum Beispiel den SDP anwenden. Systementwicklung, -verteilung und -betrieb wären damit abgedeckt. Applikationen, die dann auf Basis einer Plattform erstellt werden, werden als zusätzliche Add-Ons aufgefasst. Rombach differenziert hier mehr: Der Entwicklungsprozess zerfällt hier in zwei Teile, von denen einer eine Plattform definiert, wohingegen der zweite Aussagen darüber trifft, wie auf Basis der Plattform Anwendungen zu erstellen sind. Eine ähnlich differenzierte und detaillierte Unterscheidung zwischen Plattform- und Produktentwickung treffen Greenfield und Short [GS04]:

Plattform- vs. Produktentwickung

„. . . One-off development starts by capturing requirements and finishes with the delivery of a complete product. Software product lines [. . .] use two distict but related processes, one for developing reusable assets and one for applying them." [GS04]

Auf komplexe Softwareplattformen bezogen, ordnen Greenfield und Short dem Produktlinienentwickler Ergebnisse der Größenordnung von Frameworks zu. Produktentwicklern stehen diese Frameworks zur Verfügung. Sie können sie instanziieren, parametrisieren etc. Ein Skizze eines Referenzprozesses für die Produktlinienentwicklung

nach [BKPS04] ist in Abbildung 2.11 zu sehen. Sie zeigt die beiden eigenständigen Entwicklungsprozesse. Den Referenzprozess greifen wir im Anschluss noch einmal auf.



1. in – existierender Code, Domänen Know-How
   out – Domänenterminologie, Referenzanforderungen
2. in – out(1)
   out – Referenzarchitektur
3. in – out(2)
   out – (wiederverwendbare) Komponenten
4. in – produktspezifische Anforderungen

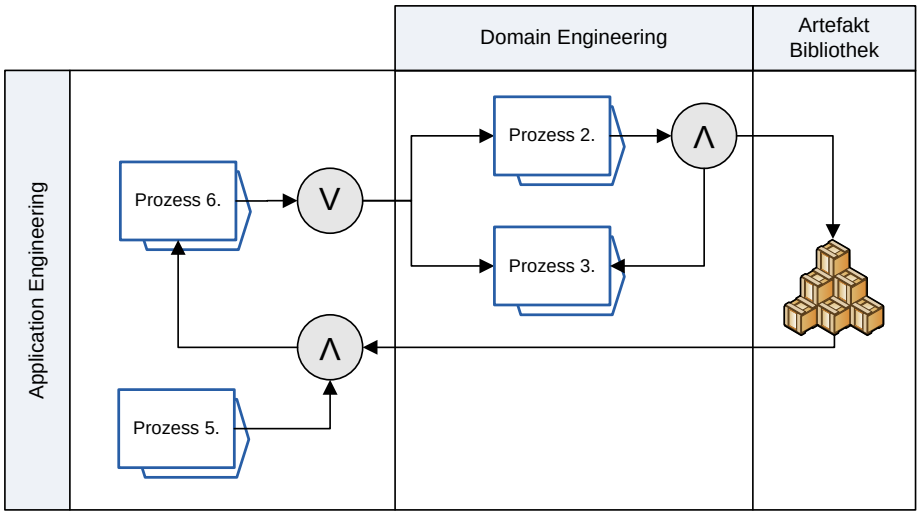


Abbildung 2.11.: Referenzmodell für die Software-Produktlinienentwicklung, nach [BKPS04]

Referenzprozess für Produktlinienentwicklung. Im Referenzprozess für Produktlinienentwicklung sind die beiden getrennten Entwicklungsprozesse Linien und Exemplare nach Rombach und Greenfield, Short zu finden. Domain- und Application Engineeringprozesse arbeiten auf einem gemeinsamen Repository. Das Repository wird hierbei durch das Domain Engineering „gefüllt", während man sich in der Applikationsentwicklung aus dem Repository „bedient". Darüber hinaus wirken Anpassungen oder Erkenntnisse aus der Entwicklung konkreter Applikationen auch wieder in das Domain Engineering zurück. In Abbildung 2.11 ist das anhand von *in-* und *out-*Produkttypen für die Einzelprozesse skizziert. Weiterhin ist die Schnittstelle zwischen Domain- und Application Engineering, die als Feedbackschleife fungiert, im unteren Teil der Abbildung zu sehen. Weitere detaillierte Ausführungen sind zum Beispiel

[BKPS04] zu entnehmen. Die in Abbildung 2.11 gezeigten Prozessschritte und Artefakttypen sind durchaus auch für die Entwicklung von Vorgehensmodellen geeignet. Rombach schreibt dazu:

> „...by means of a domain engineering process, create a generic (set of) process(es) that capture the commonalities and controlled variabilities across a domain." [Rom05]

Im Domain Engineering würden wir also zunächst allgemeine Vereinbarungen, Regelungen und Rahmen finden, diese in passenden Repositories hinterlegen und für die Ableitung konkreter Vorgehensmodelle vorhalten. Im Rahmen einer Erstellung eines konkreten Vorgehensmodells treten Lerneffekte und Optimierungen der Generika auf, die über Feedbackschleifen wieder in das Domain Engineering zurückführen, in die Wissensbasis übernommen werden und wiederum für die weitere Entwicklung neuer konkreter Modelle zur Verfügung stehen.

Probleme beim PLE für Vorgehensmodelle

Eine Problematik bei einer Produktlinienentwicklung, die gerade angerissen wurde, ist die Rückführung aktualisierten Wissens in die Plattform. Durch die Rückführung entwickelt sich die Plattform weiter, sodass Aktualisierungen unmittelbar für Neuentwicklungen zur Verfügung stehen. Insbesondere bei Vorgehensmodellen ist jedoch auch die Frage zu berücksichtigen, wie sich (parallele) Weiterentwicklungen der Plattform auf die Instanzen auswirken. Vorgehensmodelle wie zum Beispiel das V-Modell XT haben in weiten Teilen Regelungscharakter und stellen somit hohe Ansprüche an Stabilität eines und die Konformität zu einem Standard.

Features, Commonalities und Variabilities. In [BKPS04] Seite 13ff. wird Variabilität als das zentrale Konzept einer Produktlinie eingeführt. „Variabilität definiert [...] jene Teile, die durch Selektion an unterschiedliche Kundenbedürfnisse angepasst werden können..." – so schreiben Bühne et al. in ihrem Beitrag in [BKPS04]. Quer durch die Literatur sind in diesem Feld wesentliche Begriffe zu finden: *Features*, *Commonalities* und *(controlled) Variabilities*.

Unter Features werden in der Regel die wesentlichen/bestimmenden Charakteristika einer Plattform verstanden. Zur Modellierung von Features sind verschiedene Techniken verfügbar, wie zum Beispiel baumartige Strukturen [CHE04] (*feature trees*). Eine Menge von Features einer Plattform können wir auch als Leistungsbeschreibung/Spezifikation dieser Plattform interpretieren. Auf den spezifizierten Eigenschaften werden dann *invariante* und *variable* Teile gebildet.

Invariante Anteile einer Plattform werden als Commonalities oder auch *common functions* bezeichnet. Diese Teile zeichnen sich durch einen hohen Grad an Wiederverwendbarkeit aus. Je nach Beschaffenheit der Domäne kann die Wiederverwendbarkeit entweder durch präzise beschreibbare Funktionen oder generische Funktionen erreicht werden. Im Kontext von Software Produktlinien werden hier wahrscheinlich eher konkrete Funktionen, wie zum Beispiel die Verarbeitung von Dateiein-/ausgaben zu finden sein, während wir im Kontext von Vorgehensmodellen eher mit abstrakten/generischen Prozessbeschreibungen rechnen müssen. Die invarianten Anteile einer Plattform definieren also einen (zeitlich) stabilen Kern. Auf der anderen Seite stehen mit den variablen Anteilen (*variabilities*) „Soll-Bruchstellen" zur Verfügung, mit denen eine kontrollierte Variantenbildung ermöglicht werden soll. Böckle et al. führen eine Liste möglicher variabler Anteile auf:

- Variabilität in Features
- Variabilität in Abläufen
- Variabilität in Datenumfang
- Variabilität in Datenformat
- Variabilität in Systemzugang
- Variabilität in Benutzerschnittstellen

- Variabilität in Systemschnittstellen
- Variabilität in Qualität

Um Variabilität in verschiedenen Kontexten sinnvoll einzusetzen fordern sie gleichzeitig eine explizite Repräsentation von Variabilitäten. Abbildung 2.12 zeigt das in diesem Kontext aufgeführte Variabilitätsmodell[30].

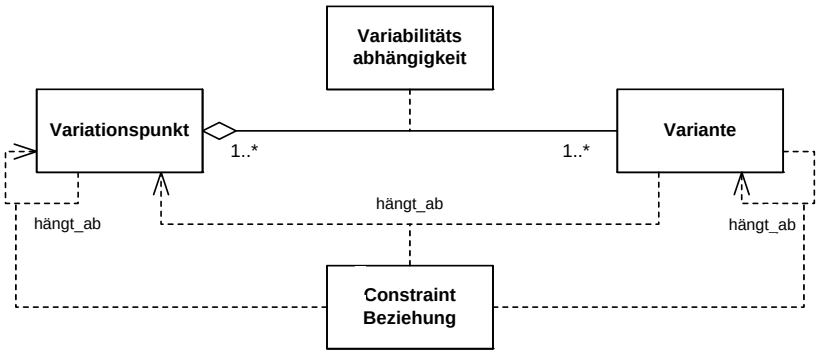


Abbildung 2.12.: PLE-Variabilitätsmodell, nach [BKPS04]

Ein Variationspunkt wird in diesem Modell als „Stelle" definiert, an der die Auswahl von Varianten ermöglicht wird. Eine Variante wird als Ausprägung eines oder mehrerer Variationspunkte definiert. Ein Variationspunkt ist somit eine abstrakte Beschreibung einer bestimmten Fähigkeit, die durch eine Variante erbracht werden kann, wobei die Art der Erbringung nicht relevant ist. Varianten können weiterhin die Erbringung für mehrere Variationspunkte realisieren. Dies ergibt einerseits einen Kompositionsoperator auf der Ebene der Varianten (Bezug zu *n* Variationspunkte) und darüber hinaus auch Variationsmöglichkeiten innerhalb der Variantenmenge. So können Fähigkeitsforderungen verschiedener Variationspunkte kombiniert und auf verschiedene Art von verschiedenen Varianten erbracht werden. Um dies fein granularer steuern zu können, werden im Variabilitätsmodell Assoziationstypen beschrieben. Einmal sind dies Assoziationen, die Abhängigkeiten zwischen Varianten und Variationspunkten ausdrücken. Sie beschreiben, wie eine Variante zu einem Variationspunkt zugeordnet wird. Der zweite Typ Assoziation repräsentiert Constraints zwischen Varianten und Variationspunkten. Im Variabilitätsmodell sind für beide Assoziationstypen bereits konkrete Auspägungen typischer Beziehungen beschrieben. Für die Variabilitätsabhängigkeit sind dies:

- Optionale Beziehungen
- Alternative Beziehungen
- Pflichtbeziehungen
- Ausschlussbeziehungen

Für die Constraints definiert das Variabilitätsmodell:

- Requires Beziehungen und
- Excludes Beziehungen.

Das Variabilitätsmodell ist ein grundlegendes Element einer Produktlinie, da es definiert, in welcher Weise eine (abstrakte/generische) Plattform für die Entwicklung beziehungsweise Auskopplung konkreter Einzelapplikationen verwendet werden kann. Eine Instanziierung des Variabilitätsmodells beschreibt hierbei durch seine Beziehungstypen eine Menge möglicher *Konfigurationen*.

Evolution einer Produktlinie. Wir wollen abschließend noch einmal auf den Themenbereich der Evolution einer Produktlinie zurückkommen. Eine Problematik, die wir vorhin schon angesprochen haben, ist die Evolution der Plattform und die daraus

30 Das Variabilitätsmodell ist in [BKPS04] aus weiteren Quellen abgeleitet. Für eine weiter gehende Recherche, die über die Inhalte dieser Arbeit hinausgeht, verweisen wir daher auf die Quellenangaben aus [BKPS04].

resultierenden Konsequenzen für die bereits entwickelten und natürlich auch für die noch zu entwickelnden Applikationen. Beziehen wir uns auf den Referenzprozess in Abbildung 2.11 sehen wir, dass die Evolution sogar einen erwünschten Regelfall darstellt. Durch die Rückführung erworbenen Wissens, neuer (qualitätsgesicherter) Komponenten etc., tritt automatisch ein Weiterentwicklungs- und Optimierungsprozess der Plattform ein. Dieser Prozess ist erwünscht, da er neben der technischen Aktualisierung gleichzeitig eine Effektivierung der Systemerstellung verspricht[31]. Knauber setzt sich mit diesem Thema in [BKPS04] Seite 177ff. sehr ausführlich auseinander. Er unterscheidet dabei zwischen *proaktiver* und *reaktiver* Evolution und schreibt dazu:

Proaktive vs. reaktive Evolution

„. . . Die proaktive Evolution ist [. . .] üblicherweise direkte Konsequenz der Geschäftsziele der Organisation, welche die Produktlinie entwickelt. Reaktiv entwickelt sich eine Produktlinie aufgrund von reaktiven Änderungen [. . .] weiter. Dies kann ebenfalls Teil der Strategie sein [. . .]" [BKPS04]

Proaktive Evolution ist also eine strategische, vorab geplante Weiterentwicklung. Knauber ordnet unter dieser Evolutionsform die *Plattformerweiterung* und die *Plattformverbesserung* ein. Die reaktive Evolution ergibt sich aus Notwendigkeiten, auf externe Effekte zu reagieren. Knauber ordnet hier *funktionale*, *nichtfunktionale* und *domänenbedingte Ursachen* ein. Diese Punkte sind auch für den Kontext dieser Arbeit sehr wichtig. Wir greifen sie im Kapitel 4.3 auf, konkretisieren sie und zeigen die Anwendung im Kontext von Vorgehensmodellen.

2.3.3. Prozessanpassungen im Produktkontext

Fassen wir Vorgehensmodelle als Produkte auf, so definieren wir im Kontext der Produktlinien die den Vorgehensmodellen zugrunde liegenden Metamodelle sowie einen Teil der enthaltenen Inhalte als Plattform. Um ein konkretes Vorgehensmodell aus seiner Plattform abzuleiten sind entsprechende Prozesse notwendig. Im Abschnitt 2.1 haben wir bereits von ausgewählten Vorgehensmodellen Anpassungsprozesse beschrieben. Wir haben bislang jedoch keine Untersuchungen hinsichtlich einer Plattform für diese Vorgehensmodelle geführt. Dies ist unter anderem Inhalt des folgenden Kapitels 3.

Wir können jedoch bereits auf dieser Ebene eine Positionierung der bislang vorgestellten Vorgehensmodelle im Kontext von Produktlinien vornehmen. Eine entsprechende Abbildung nehmen wir auf Basis des Referenzprozesses aus Abbildung 2.11 vor.

PLE-Referenprozess	V-Modell XT	MSF	EPF/OpenUp	Kriterien
Domänenterminologie	×	×	×	
Referenzarchitektur	×	×	×	Metamodell
Wiederverwendbare Komponenten	–	◯	×	physische Komponenten und Abdeckung des Prozesses
Feedbackschleife	–	–	–	
Applikationsanalyse	×	–	–	Einführungs-/Pflegeprozess
Applikationsdesign	×	–	–	Einführungs-/Pflegeprozess
Applikationsimplementierung	×	×	×	

Tabelle 2.5.: Positionierung der ausgewählten Vorgehensmodelle im PLE-Referenzprozess

31 Für den Kontext V-Modell XT geben wir hier als konkretes Beispiel die Mustertexte an, die aus der Anpassung des V-Modells für die Bundeswehr (Anhang A.1) wieder in den Hauptentwicklungsstrang zurückgeführt wurden und nun allen Anwendern zur Verfügung stehen.

Tabelle 2.5 enthält eine kompakte Abbildung der Vorgehensmodelle auf den PLE-Referenzprozess. Die in der Tabelle aufgeführten applikationsbezogenen Zeilen interpretieren wir auf der Ebene konkreter Vorgehensmodelle. Die domänenbezogenen Eigenschaften tauchen in der Tabelle fast gar nicht auf, da keines der betrachteten Vorgehensmodelle zum Zeitpunkt der Analyse über eine Linienstrategie verfügt. Dass die Voraussetzungen dafür aber gegeben sind, zeigt das Vorhandensein einer Domänenterminologie und einer Referenzarchitektur (in Form eines Metamodells). Wie wir in unserer Zusammenstellung der Basiskonzepte der Vorgehensmodelle in Abschnitt 2.1 bereits feststellten, ist das Vorhandensein von Komponenten unterschiedlich ausgeprägt. EPF/OpenUP definieren beispielsweise ein physisches Komponentenmodell, das jedoch aufgrund seiner hohen Integrationsdichte nur begrenzt wiederverwendbar ist. MSF definiert eigenständige Komponenten, die jedoch nur ausgewählte Teilaspekte des Vorgehensmodells repräsentieren und nicht den Gesamtumfang abdecken. Das V-Modell XT definiert nur einen logischen Komponentenbegriff, sodass die Variantenbildung hier stark erschwert ist.

Bedarf. Aus der Positionierung geht ein Bedarf hervor, den unter anderem auch Rombach [Rom05] schon formuliert hat. Vorgehensmodellen fehlt es offenbar noch an Prozessen aus dem Domain Engineering die notwendig sind, um Vorgehensmodelle auf einer Stufe mit Produktlinien zu positionieren. Bedarf besteht in der Definition einer Referenzarchitektur, die ein Wiederverwendungs- und Variabilitätskonzept umsetzt. Es besteht ein Bedarf an einem Lebenszyklusmodell, das Feedbackschleifen definiert, sodass erworbenes Wissen aus der Entwicklung eines Vorgehensmodells wieder dem Domain Engineering verfügbar gemacht wird. Eine Plattform für Vorgehensmodelle muss definiert werden. Die Plattform muss dabei über das bloße Vorhandensein eines Metamodells hinausgehen und seine Komponenten auch auf inhaltlicher Ebene füllen.

Ziel dieses Abschnitts war es, Phasen im Lebenszyklus eines Vorgehensmodells zu identifizieren und in etablierten Vorgehensmodellen sowie Produktlinien zu positionieren. Der detaillierten Modellierung und Beschreibung des Lebenszyklusmodells, das alle relevanten Phasen enthält, widmen wir uns in Kapitel 5.2. An dieser Stelle benennen wir jedoch zunächst die einzelnen Phasen, die wir hier ermittelt haben:

Feststellung und Planung – In dieser Phase werden Bedarfsermittlung und Planungen durchgeführt, die für eine Entwicklung beziehungsweise Anpassung eines Prozesses erforderlich sind.

Umsetzung – In dieser Phase werden Umsetzungskonzepte beziehungsweise Spezifikationen realisiert.

Stabilisierung – In dieser Phase werden neue beziehungsweise aktualisierte Inhalte/Vorgehensmodelle oder Teile davon erprobt.

Einführung und Bereitstellung – Neu erstellte Vorgehensmodelle oder Teile davon werden in der Breite eingeführt beziehungsweise bereit gestellt.

Diese Phasen sind zunächst einmal produktorientiert und für sich einzeln zu betrachten. Die Ableitung dieser Phasen beruht auf der Analyse der vorhandenen Anpassungsprozesse der betrachteten Vorgehensmodelle sowie dem PLE-Referenzmodell und dem Microsoft SDP. Durch „einfaches Aneinanderreihen" kann mit diesen Phasen *iterativ* ein einzelnes Vorgehensmodell (weiter-)entwickelt werden. Da wir in dieser Arbeit jedoch nicht nur Einzelmodelle sondern Familien von Vorgehensmodellen betrachten, müssen wir diese Phasen entsprechend auf der Ebene des Domain Engineering zusammenfassen und sinnvoll kombinieren. Die Voraussetzungen dafür werden in den nächsten beiden Kapiteln 3 und 4 gelegt, in denen einerseits etablierte Vorgehensmetamodelle untersucht und im Anschluss systematische Grundlagen für den Entwurf hoch modularer Vorgehensmodelle gelegt werden.

2.4. Begriffe und Definitionen

Wir wollen dieses Kapitel mit den Festlegungen und Begriffsdefinitionen abschließen. In diesem Kapitel haben wir bereits einen ersten breiten Einblick in den Bereich Vor-

gehensmodelle gegeben. Wir haben dabei auch schon eine Menge von Begriffen eingeführt, die wir zum Basisvokabular zählen wollen. Im Verlauf dieser Arbeit werden wir weitere einführen. In diesem Abschnitt fassen wir die Begriffe noch einmal zentral zusammen und legen deren Verwendung fest.

2.4.1. Ontologie für Vorgehensmodelle

Vorgehensmodelle (auch Prozessmodelle) stellen äußerst komplexe, in der Regel hoch integrierte Modelle für Entwicklung, Beschreibung und Komposition von Prozessen dar. Es finden sich viele Definitionen, wie zum Beispiel die von Gnatz [Gna05]:

Vorgehensmodell

„. . . Unter einem Vorgehensmodell verstehen wir einen standardisierten organisatorischen Rahmen für den idealen Ablauf eines Entwicklungsprojekts. Vorgehensmodelle werden in Form zu erstellender Produkte, durchzuführender Aktivitäten und zu besetzender Rollen beschrieben." [Gna05]

Die Definition von Gnatz stützt sich auf den *Submodellen* eines Vorgehensmodells ab. Abbildung 2.13 illustriert diese Vorstellung und baut sie noch weiter aus. Die Bestandteile der Definition von Gnatz finden sich innerhalb des „Womit?"-Rahmens. Es handelt sich um die Submodelle:

- Rollenmodell,
- Aktivitätsmodell und
- Artefakt-/Produktmodell.

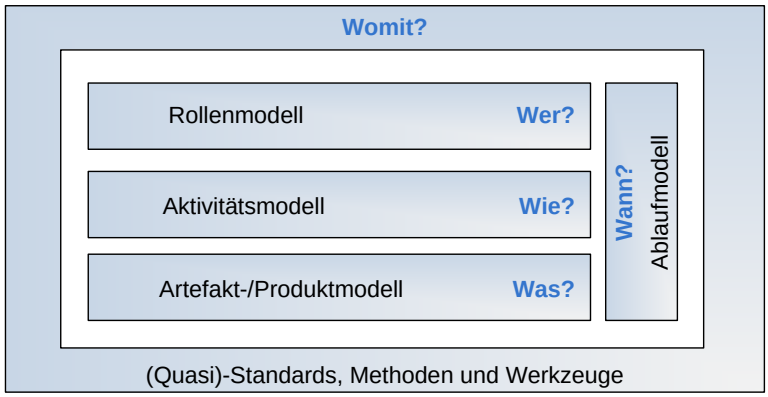


Abbildung 2.13.: Allgemeines Konzept von Vorgehensmodellen als Stack von Submodellen

Submodelle eines Vorgehensmodells. Unter einem Submodell verstehen wir dabei ein Teilmodell, das eine spezifische Charakteristik detailliert beschreibt, jedoch im Sinne komponentenbasierten Denkens eine hohe Transparenz besitzt. Abgeschlossene Submodelle sind eine der Schlüsselkomponenten für flexible Vorgehensmodelle. Zusätzlich zu den oben aufgeführten Submodellen haben wir weitere (zum Teil optionale) Submodelle zu betrachten:

- Ablaufmodell[32] – Es dient zum Beispiel zur Ablaufspezifikation, die im Rahmen einer Plangenerierung Verwendung finden kann.
- Referenz- und Konventionsmodelle
- Werkzeugmodell

32 Das Ablaufmodell wird in der Literatur auch als Prozessmodell bezeichnet. Um hier eine Schärfung der verwendeten Terminologie zu erreichen verwenden wir den Begriff Ablaufmodell.

Ein Vorgehensmodell fasst in der Regel immer Querschnitte dieser Modelle zu Einheiten höherer Integrationsdichte zusammen. Dabei ist der gesteckte Rahmen nicht zwangsweise abgeschlossen, sondern kann variieren. Alle zurzeit verfügbaren Vorgehensmodelle (unerheblich, ob ihnen ein Metamodell zugrunde liegt oder nicht) lassen sich in diesem Rahmen unterbringen.

Der Vorgehensmodellbegriff dieser Arbeit. Im Kontext dieser Arbeit definieren wir Vorgehensmodelle aus einer systemtheoretisch geprägten Sichtweise. Wir legen ein Kompositionsmodell (Kapitel 4) zugrunde, in dem wir *Prozesselemente*, *Prozesskomponenten* und *Vorgehensmodelle* entsprechend einordnen.

Unter *Prozesselementen* wollen wir dabei alle atomaren Einheiten verstehen, die in einem Prozess/einem Vorgehensmodell relevant sind. In Bezug auf Abbildung 2.13 sprechen wir also von Elementen aus den jeweiligen Submodellen (zum Beispiel Rollen, Artefakte, Abläufe etc.).

Unter *Prozesskomponenten* wollen wir *Einheiten* im Sinne einer eigenständigen physischen Komponente verstehen, die arbeitsteilige Entwicklung und Pflege sowie die Verteilung unterstützt. Eine Prozesskomponente ist dabei gleichzeitig ein Container für Prozesselemente. Ferner ist sie ein Baustein für Vorgehensmodelle und somit auch Gegenstand der Konfiguration und Anpassung.

Unter einem *Vorgehensmodell*[33] verstehen wir (strukturell) eine Konfigration von Prozesskomponenten. Inhaltlich ist ein Vorgehensmodell eine abstimmte, konsistente Menge von Prozesselementen, die miteinander in Beziehung stehen.

Der Vorgehensmodellbegriff dieser Arbeit setzt also auf einem hierarchischen Konzept auf, das konfigurativ zunächst Komponenten zusammenfasst und darauf aufbauend inhaltliche Strukturen abbildet. Im Rahmen der vorliegenden Arbeit beschränken wir uns jedoch auf rein strukturelle Aspekte und gestalten nur Prozesskomponenten detailliert aus. Der gewählte Modellierungsansatz gestattet jedoch Erweiterungen, sodass beispielsweise das Verfahren zur Plangenerierung von Gnatz [Gna05] prinzipiell integriert werden kann.

2.4.2. Modellierungs-, Meta- und Modellebenen

Ein großer Teil dieser Arbeit befasst sich mit der Modellierung beziehungsweise Metamodellierung von Vorgehensmodellen. Neben dem formalen Basismodell in Kapitel 4 erarbeiten wir in Kapitel 5 einen integrierten Modellierungsansatz als Umsetzung des Basismodells. Dieser setzt mit einem Metamodell an, das wesentliche Struktureigenschaften beschreibt und führt zu einem Prozessmodell (Lebenszyklusmodell) das diverse Operationen, zum Beispiel die Instanziierung des Metamodells beschreibt. Wir führen nun im Folgenden die wesentlichen Begriffe zum Themenbereich Modellierung und Modellebenen ein.

UML und MOF. Zur Modellierung des Vorgehensmetamodells verwenden wir die Unified Modeling Language (UML) [JRH+03, WO06, OMG07a, OMG07b]. Die UML ist eine allgemein anerkannte Modellierungssprache, die sich insbesondere für die Strukturmodellierung eignet. Sie bietet hierfür die entsprechenden Sprachmittel (Klassendiagramme, Komponentendiagramme) an. Einschränkungen etc. sind im Rahmen der UML üblicherweise durch die so genannte Object Constraint Language (OCL, [OMG06b]) beschrieben. Die UML selbst ist wieder durch die Sprachmittel der UML beschrieben. Zum Einsatz kommt hier eine Modell- und Metamodellhierarchie der OMG (Abbildung 2.14), an deren Spitze die Meta Object Facility (MOF, [OMG06a]) steht.

Um die Zusammenhänge etwas klarer zu gestalten, beschreiben wir zunächst die Begriffe *Modell* und *Metamodell*, positionieren sie im Anschluss in der MOF-Hierarchie und stellen dem dann die Inhalte der vorliegenden Arbeit gegenüber.

33 Für die Verfeinerung der Terminologie im Kontext Vorgehensmodell verweisen wir auf die Kapitel 4 und 5

	MOF-Hierarchie	UML als MOF-Instanz	Gegenüberstellung der Vorgehensmetamodellierung dieser Arbeit
M3	Meta-Metamodell	MOF	
M2	Metamodell	UML-Sprache	Vorgehensmetamodell (Sprachmittel der UML, Basis MOF)
M1	Modell	Klassenmodell	Vorgehensmodell (Entwicklungsstandard, Vorgehensmodellvariante)
M0	Instanzen	Laufzeit-/Objektmodell	Projektmodell (üblicherweise repräsentiert durch einen Projektplan)

Abbildung 2.14.: Meta Object Facility und ihre Anwendung in dieser Arbeit

Ausgangspunkt für die Positionierung ist der Begriff *Modell*. Dieser kann verschiedenartig definiert sein. Broy und Steinbrüggen [BS04] bezeichnen die „Interpretation einer Spezifikation" als Modell. Greenfield und Short [GS04] gehen hier pragmatischer vor und schreiben: „A model is an abstract description of software." Im Allgemeinen wird in der Informatik unter einem Modell eine *Abstraktion eines Gegenstandsbereichs der realen Welt* verstanden. Unwesentliche Details entfallen hierbei in der Regel, um die Komplexität des Modells beherrschbar zu machen.

Die darauf aufbauende Motivation zum Begriff des *Metamodells* ist sehr intuitiv in [WO06] zu finden. Prinzipiell geht es darum, dass Modellierungssprachen Sprachen sind, mit denen man Modelle erstellen kann [GS04]. Um eine Modellierungssprache zu erstellen benötigt man also wiederum eine Sprache. Gnatz [Gna05] argumentiert hier sehr gründlich über die so genannten *Objekt-* und *Metasprachen* – wir übernehmen diese Terminologie. Betrachten wir also eine Modellierungssprache als Objektsprache, so ist die Sprache zur Erstellung von Modellierungssprachen in diesem Kontext eine Metasprache. Dieses Ebenenmodell, in dem ein aktueller Gegenstandsbereich für eine Sprache beschreibbar ist, die selbst wieder durch eine Metasprache beschrieben wird, lässt sich in einer Hierarchie einordnen. Für die UML ist das durch die *Object Management Group* (OMG) geschehen (Abbildung 2.14). Im Fokus stehen hier jedoch Modelle, die mit eben der gleichen Terminologie versehen werden können. Die Objektsprache wird üblicherweise als Modell bezeichnet. Wie wir aber gerade schon erläutert haben, können Modelle selbst wieder als Interpretation betrachtet werden, womit wir den Begriff Metasprache auf ein *Metamodell* übertragen.

Allgemein versteht man unter einem Metamodell das Modell eines Modells. Das übergeordnete Modell ist dabei ein Beschreibungsmodell, das die Sprache für das untergeordnete Modell festlegt. Mit der MOF [OMG06a] steht eine solche Modellhierarchie zur Verfügung, die wir kurz anhand von Abbildung 2.14 beschreiben:

M0: beschreibt ein so genanntes *Laufzeit-* beziehungsweise *Objektmodell*. M0 ist die Ebene der konkreten Instanzen. Im Kontext einer Programmiersprache sind die also die Speicherobjekte.

M1: beschreibt auf der ersten Abstraktionsebene die Modelle der Ebene M0. Im Kontext der UML finden wir hier beispielsweise *Klassendiagramme* als Abstraktion von Speicherobjekten.

M2: beschreibt die Metamodellebene. Hier finden wir zum Beispiel die Definition der UML-Sprache (in Form von UML-Diagrammen). Beispielsweise finden sich hier Abstraktionen von Klassen (als Sprachmittel) oder Assoziationen.

M3: beschreibt die Meta-Metamodellebene. Hier werden Sprachen zur Beschreibung von Metamodellen beschrieben. Auf dieser Ebene finden wir zum Beispiel die MOF, die die wesentlichen Sprachmittel zur Beschreibung des UML-Metamodells

beschreibt. Die MOF ist beispielsweise Ansatzpunkt für Werkzeughersteller, die auf ihr aufbauend Modellierungswerkzeuge erstellen können. Ein Beispiel hierfür ist das MOFLON-Toolset [ABS04, ARS05, AKRS06], das die MOF vollständig abbildet und somit für die Erstellung von Metamodellen gut geeignet ist.

UML und MOF in dieser Arbeit. Für die Modellierung von Vorgehensmodellen und Vorgehensmetamodellen setzen wir in dieser Arbeit ebenfalls auf die UML 2. In Abbildung 2.14 positionieren wir die Inhalte entsprechend in der MOF-Hierarchie.

Wir setzen mit unserer Modellierung in der Ebene *M2* an. Wir verwenden die MOF und UML als Modellierungssprache für das *Vorgehensmetamodell* (vgl. Kapitel 5.1). Wir beschreiben auf dieser Ebene die wesentlichen Entitäten (Rollen, Produkte etc.) und Strukturen (Abhängigkeiten). Auf der Ebene *M1* finden sich dann konkrete Vorgehensmodelle, Vorgehensmodellvarianten und auch die Ausgestaltungen im Kontext von Entwicklungsstandards, die ja im Wesentlichen mehrere Vorgehensmodellvarianten umfassen. Hier finden wir konkrete Instanzen der Metamodellelemente, so zum Beispiel *Projekthandbücher* als Instanzen vom Typ Produkt. Insbesondere im Kapitel 6 finden wir viele Beispiele für diese Ebene. Die Ebene *M0* erreichen wir mit der vorliegenden Arbeit nicht. Auf ihr positionieren sich die so genannten *Projektmodelle*, die beispielsweise in Form von Projektplänen vorliegen. Für detaillierte Informationen hierzu verweisen wir auf Gnatz [Gna05].

Wir bewegen uns mit der vorliegenden Arbeit also nur auf den MOF-Hierarchieebenen *M2* (für die Modellierung der Vorgehensmetamodelle) und *M1* (für die Modellierung der Vorgehensmodelle).

2.4.3. Graphen

Unser formales Basismodell bedient sich der Graphentheorie. Die wesentlichen Konstrukte führen wir nun ein. Die Graphentheorie ist eine der grundlegendsten Beschreibungs- und Modellierungstechniken der Informatik. Hierarchische Datenstrukturen, Wegberechnungen und Optimierungsprobleme sind nur einige Beispiele der Anwendung. Auch viele weitere Modellierungstechniken, wie zum Beispiel die gerade besprochene UML sind auf Graphen zurückzuführen und verwenden zum Beispiel *Graphgrammatiken* für Änderungen [Kön05a, KS06].

Grundlagen und Begriffe. Wir geben nun die Grundlagen (Definitionen und Begriffe) der Graphentheorie an. Weiter beschreiben wir im Vorgriff auf Kapitel 4 die Anwendung im Kontext dieser Arbeit. Wir orientieren uns in den mathematischen Definitionen an [SW06] und [BL95]. Seien V und E endliche Mengen von Knoten (englisch: *vertices*) und Kanten (englisch: *edges*).

$$G = (V, E) \tag{2.1}$$

heißt endlicher Graph. G heißt *ungerichtet*, wenn $E \subseteq \{\{v_1, v_2\} \,|v_1, v_2 \in V, v_1 \neq v_2\}$ gilt. Ein Element $\{v_1, v_2\} \in E$ heißt Kante zwischen v_1 und v_2. G wird als *gerichteter* Graph bezeichnet, wenn $E \subseteq (V \times V) \setminus \{(v, v) \,|v \in V\}$ gilt. Ein Paar $(v_1, v_2) \in E$ heißt gerichtete Kante von v_1 nach v_2. Im Kontext dieser Arbeit genügen uns die einfachen Graphen gemäß oben stehender Formel (2.1) nicht immer, da hier „parallele" Kanten und so genannte Schlingen nicht unterstützt werden. Wir erweitern daher:

$$G = (V, E, e) \tag{2.2}$$

heißt *ungerichteter* endlicher *Multigraph*, wenn für die Kantenmengen alle Forderungen von oben erfüllt sind und darüber hinaus gilt: $e : E \to \{\{v_1, v_2\} \,|v_1, v_2 \in V\}$. Gilt für die Abbildung e jedoch: $e : E \to V \times V$, heißt G *gerichteter* endlicher Multigraph. In dieser Arbeit verwenden wir grundsätzlich *gerichtete* Graphen und Multigraphen.

Wir vervollständigen die Begriffsliste noch weiter: Zwei Knoten v_1 und v_2, die durch eine Kante $(v_1, v_2) \in E$ verbunden sind, heißen *adjazent* oder benachbart. Zu einem

Knoten ist ferner die *Nachbarschaftsmenge* aller benachbarten Knoten durch $N(v) = \{v' \,|(v, v') \in E\}$ bestimmt. Gilt für einen Knoten $N(v) = \emptyset$, heißt dieser Knoten *isoliert*. Nachbarschaftsbeziehungen zwischen Knoten sind für uns insbesondere dann wichtig, wenn mehrere, nicht zusammenhängende Graphen miteinander verknüpft werden sollen. Hier ist es insbesondere für gerichtete Graphen wichtig, Anfang und Ende einer Kante bestimmen zu können. Für eine Kante $(v_1, v_2) \in E$ heißt v_1 *Anfangsknoten* und v_2 Endknoten. Darauf aufbauend ist die Anzahl derjenigen Kanten, für die ein Knoten v Anfangsknoten ist, bestimmt durch den *Außengrad* $d^+(v)$. Die Anzahl der Kanten, für die der Knoten v Endknoten ist, ist bestimmt durch den *Innengrad* $d^-(v)$.

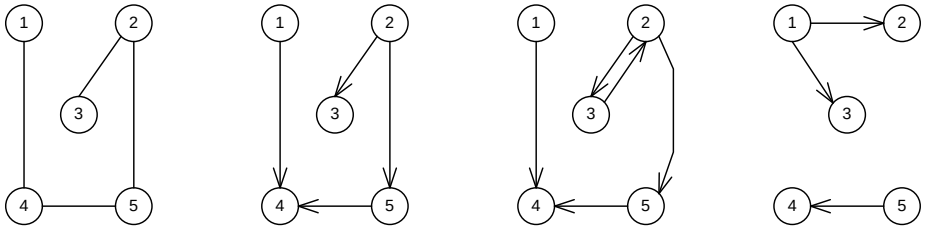


Abbildung 2.15.: Graphen (ungerichtet, gerichtet, multi) und Teilgraphen

In Abbildung 2.15 sind die gerade definierten Graphen noch einmal skizziert. Weiterhin zeigt Abbildung 2.15 (rechts) zwei Teilgraphen. Zwei Graphen $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$ seien gegeben. G_2 heißt Teilgraph von G_1, wenn gilt $E_2 \subseteq E_1$ und $V_2 \subseteq V_1$. Nehmen wir zum Beispiel den Graphen in Abbildung 2.15 (Mitte links): $G_1 = (\{1, 2, 3, 4, 5\}, \{(1, 4), (5, 4), (2, 3), (2, 5)\})$, dann ist der Graph $G_2 = (\{4, 5\}, \{(5, 4)\})$ (Abbildung 2.15, rechts) ein Teilgraph von G_1. Vereinigungs- und Differenzoprationen über Graphen bilden wir immer über den Mengen der Knoten und Kanten. Modellierungsbedingt sind die Graphen dieser Arbeit hinsichtlich ihrer Knoten- und Kantenmengen immer disjunkt, womit eine Vereinigung der (Teil-)Graphen immer nicht zusammenhängende Graphen erzeugt.

Graphen in dieser Arbeit. In dieser Arbeit verwenden wir im Wesentlichen zwei verschiedene Arten von Graphen: *Bäume* und *gerichtete (Multi-)Graphen*. Wir gehen nun basierend auf einem Auszug des Beispiels aus Kapitel 4.2.3 auf die hier verwendeten Techniken ein. Abbildung 2.16 zeigt den betreffenden Ausschnitt.

Im oberen Teil der Abbildung 2.16 ist der *Strukturkompositionsgraph* zu sehen. Dieser Graph ist als *Baum* organisiert und dient der Bereitstellung von Entwicklungs- und Verteilungseinheiten, den Prozesskomponenten. Im unteren Teil der Abbildung 2.16 ist der *Abhängigkeitsgraph* zu sehen, der inhaltlich, strukturelle Abhängigkeiten in Vorgehensmodellen repräsentiert und eine Folge des Strukturkompostionsgraphen ist. Enthält der Strukturkompositionsgraph also die Knoten V_{SKG} und der Abhängigkeitsgraph die Knoten V_{AG}, so gilt: $V_{AG} \subseteq V_{SKG}$. Im Kapitel 4 gehen wir detailliert auf die Modellierung und Anwendung der Graphen für diese Arbeit ein.

Zusammenfassung

In diesem Kapitel haben wir den Stand der Technik dargestellt und dabei neben dem Problembereich auch relevante Techniken vorgestellt, die wir zur Erarbeitung der Lösung benötigen. Dazu haben wir im Abschnitt 2.1 ausgewählte *Vorgehensmodelle* vorgestellt. Diese dienen uns als Bezugspunkte in dieser Arbeit. Eine besondere Rolle fällt dabei dem V-Modell XT zu, das nicht nur Bezugspunkt unserer Arbeit ist, sondern gleichermaßen Beispiele beisteuert. Wesentliche Teile dieser Arbeit sind durch die Entwicklung des V-Modells motiviert. Einige Ergebnisse sind bereits wieder in die V-Modell XT-Entwicklung zurückgeflossen (Anhang A.3).

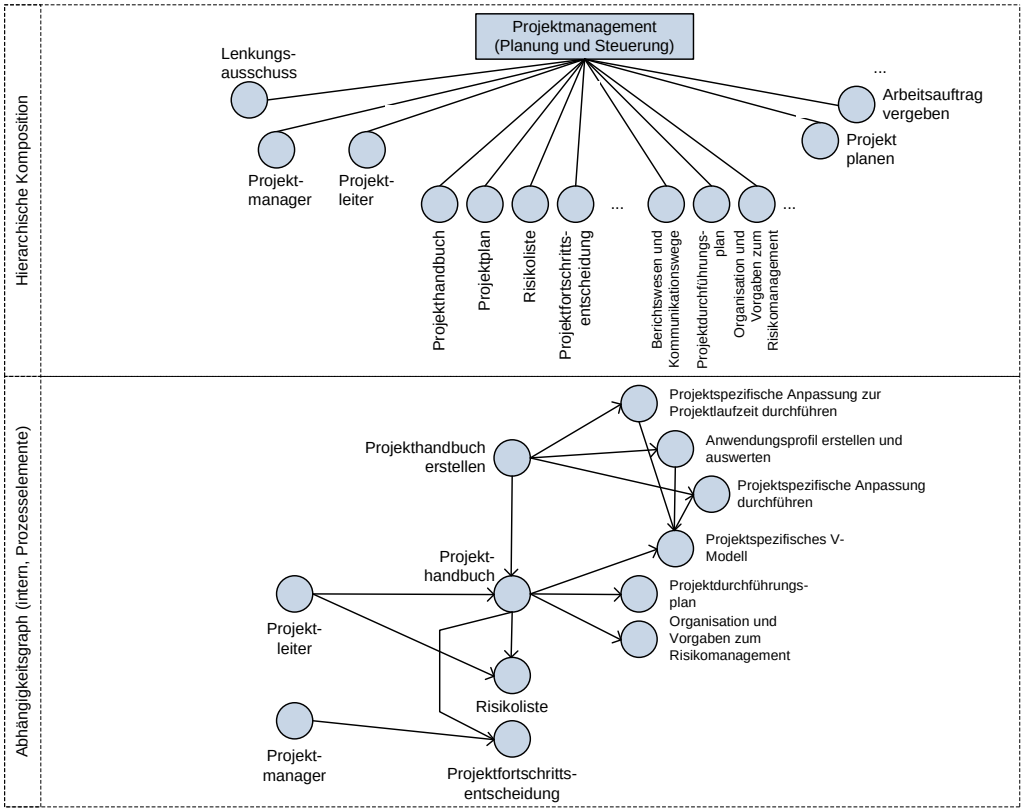


Abbildung 2.16.: Beispiel für die Anwendung von Graphen in dieser Arbeit

Die vorliegende Arbeit befasst sich schwerpunktmäßig mit der Konstruktion von Vorgehensmodellen unter Gesichtspunkten der Entwicklung von *Entwicklungs-/Vorgehensstandards*. Im Abschnitt 2.2 betrachten wir daher zunächst die Erstellungs- und Anpassungsoptionen von Vorgehensmodellen. Diese Optionen sind bereits zum Teil der Literatur zu entnehmen und wurden in diesem Abschnitt, entsprechend der Anforderungen aus zum Beispiel der V-Modell-Entwicklung, angepasst und erweitert. Im Anschluss haben wir die wesentlichen Erstellungs- und Anpassungsprozesse analysiert und beschrieben.

Der Abschnitt 2.3 betrachtet Lebenszyklusmodelle. Die Betrachtung erfolgt dabei anhand eines ausgewählten integrierten Lifecyle-Prozesses – dem Microsoft Solution Delivery Process (SDP). Dieser koppelt Software-Entwicklung und Betrieb. Weiterhin betrachten wir den Referenzprozess für die Software Produktlinienentwicklung (PLE). In der abschließenden Betrachtung von Prozessanpassungen unter Berücksichtigung von SDP und PLE unterstreichen wir noch einmal die bereits motivierte *Produktsicht* auf Vorgehensmodelle und stellen den Bedarf eines integrierten *Lebenszyklusmodells* für Vorgehensmodelle und Vorgehensmodell*linien* fest.

Das Kapitel schließt im Abschnitt 2.4 mit der abschließenden Definition der Begriffswelt dieser Arbeit. Weiterhin gehen wir kurz auf die verwendeten Techniken aus der UML und der Graphentheorie ein.

3. Analyse: Architekturen von Vorgehensmodellen

In diesem Kapitel analysieren wir grundlegende Architekturen und Architekturelemente von Vorgehensmodellen. Es dient uns dazu, etablierte Konzepte bekannter Vorgehensmodelle, beziehungsweise deren Metamodelle, zu ermitteln. Wir betrachten die Konzepte zunächst jeweils eigenständig in den Gruppen Strukturelemente, ablaufbezogene Elemente und Abhängigkeiten. Zu jedem Konzept identifizieren wir Beziehungstypen und leiten Anforderungen für die Bildung von physischen *Prozesskomponenten* her. Die Erfassung relevanter Anwendungsfälle im Kontext der vorliegenden Arbeit greift diese Anforderungen auf und legt den Grundstein für einen modularen Architekturentwurf eines Entwicklungsstandards.

Am Ende dieses Kapitels sind etablierte Frameworks für die Erstellung von Vorgehensmodellen eingeführt und ein Grundverständnis für die Elemente eines Vorgehensmodells und deren Beziehungen gelegt worden. Anforderungen für eine modularisierte Strukturierung von Modellierungselementen sind aufgenommen und formuliert. Anwendungsfälle präzisieren den Modellierungskontext.

3.1. Frameworks für die Modellierung von Vorgehensmodellen

Zunehmend sind auch für die Modellierung von Vorgehensmodellen reichhaltige Frameworks zu finden. Eine Voraussetzung für die Erstellung eines Vorgehensmodells auf der Basis eines Frameworks ist die Basierung auf einem formalen Metamodell. Analog zum bekannten Frameworkbegriff der Software-Architektur [HR02] steht somit über eine Sprach-, beziehungsweise API[1]-Spezifikation hinaus auch partiell vorgefertigter Inhalt zur Verfügung. Vorgehensmodelle stellen dabei in der Regel schon eine sehr reichhaltige, inhaltlich ausgestattete Grundlage zur Verfügung, die zumeist schon für die grobe Anwendung in Projekten genügt. Darüber hinaus stellen sie ggf. auch Mechanismen zur inhaltlichen Anpassung (Tailoring) bereit.

Mechanismen zur Anpassung sind dabei aber keine Fähigkeiten, die das Vorgehensmodell anbietet, sondern eine Fähigkeit, die das dem Vorgehensmodell zugrunde liegende Metamodell definiert. Dieses beschreibt auf einer strukturellen Ebene Abhängigkeiten und Relationen zwischen Typen eines Vorgehensmodells und fügt über passende Abhängigkeitstypen entsprechende Aufsetzpunkte für eine (umfangsmäßige) Anpassung eines Vorgehensmodells hinzu.

Strukturelle und inhaltliche Anpassung

Jede inhaltliche Anpassung eines Vorgehensmodells (Parametrisierung, Reduktion, Erweiterung des Inhalts) wird durch eine entsprechende Struktur auf der Ebene des dem Vorgehensmodell zugrunde liegenden Metamodells ermöglicht.

Strukturelle Anpassungen werden auf der Ebene des Vorgehensmetamodells vorgenommen. Unter strukturellen Anpassungen sind beispielsweise das Hinzufügen neuer oder das Erweitern vorhandener Typen zusammenzufassen. Diese Änderungen/Anpassungen finden auf der Ebene des Metamodells statt, das Typen und Relationen definiert. Strukturelle Anpassungen auf der Ebene des Vorgehensmodells, d. h. Anpassungen, die auf der Instanziierung von Metamodelltypen basieren, zum Beispiel das Neuerstellen

1 API: Application Programming Interface

von Assoziationen oder sonstigen Prozesselementen, bezeichnen wir konsequent als *inhaltliche Anpassung*. Auch die inhaltliche Ausgestaltung von Prozesselementen, zum Beispiel deren Dokumentation ordnen wir hier ein. Gemäß der Vereinbarungen in Kapitel 2.2.1 gilt für uns folgende Einstufung des Anpassungsbegriffs: Strukturelle Anpassungen nehmen wir ausschließlich auf der Ebene des Vorgehensmetamodells vor; alle weiteren Arbeiten subsumieren wir unter dem Begriff inhaltliche Anpassung.

Wie bereits erwähnt, setzen jegliche Anpassungsoptionen auf der inhaltlichen Ebene entsprechende strukturelle Eigenschaften beziehungsweise Fähigkeiten auf der Ebene des Metamodells voraus. Dabei ist die Menge der Begriffe, die wir im Kontext von Vorgehensmodellen betrachten bekannt (vgl. Kapitel 2.4) und weitgehend definiert. Wie wir in Kapitel 2.1 bereits an ausgewählten Vertretern angedeutet haben, liegt die eigentliche Komplexität in den strukturellen Anteilen – also in der Menge der Abhängigkeiten und Relationen zwischen den Typen (vgl. Konzepte der jeweiligen Modelle). Der Grad der Komplexität bezieht sich hierbei hauptsächlich auf die Integrationsdichte der jeweiligen Basiskonzepte. Die Aufstellung der jeweiligen Anpassungsprozesse (Kapitel 2.2.2) zeigte bereits, dass unabhängig von der möglichen Flexibilität des Metamodells, Anpassungen auf Strukturen höherer Integrationsdichte problematischer sind. Um dieser These auf den Grund zu gehen betrachten wir im Folgenden die Metamodelle der in Kapitel 2.1 vorgestellten Vorgehensmodelle detaillierter. Wir betrachten dabei die strukturellen Elemente (Typen) nur kurz und konzentrieren uns verstärkt auf die Abhängigkeitstypen zwischen den Elementen.

3.2. Strukturelemente von Vorgehensmodellen

In diesem Abschnitt befassen wir uns mit den grundlegenden Strukturen eines Vorgehensmodells. Wir betrachten hier Elementtypen, die wir als *Prozesselemente* bezeichnen und modellieren. Die einzelnen Abschnitte, in denen wir die ausgewählten Vorgehensmodelle betrachten, untergliedern wir jeweils angelehnt an die Submodelle eines Vorgehensmodells (Produkt-, Aktivitäts- und Rollenmodell, vgl. Kapitel 2.4).

3.2.1. Das V-Modell XT Metamodell

Das V-Modell XT Metamodell ist ein hoch integriertes Modell, in dem sämtliche Strukturen des V-Modells zentral hinterlegt sind. Zum Zeitpunkt der Erstellung dieser Arbeit befindet sich das Metamodell des V-Modells in einer Überarbeitung. Um trotzdem verlässliche Aussagen und eine tragfähige Analyse vornehmen zu können, beziehen wir uns in unseren Aussagen auf die Version 1.2.1 des Metamodells, das dem V-Modell 1.2Bw zugrunde liegt. Auf Aktualisierungen und Abweichungen in den Aussagen, die sich aufgrund der Änderungen im Metamodell im Rahmen der Bearbeitung des *Task RO*[2] ergeben, gehen wir abschließend in Anhang A.3 ein.

Eine vollständige Dokumentation des Metamodells in Form einer UML-Modellierung liefert bislang nur [Gna06]. Diese Dokumentation ist jedoch als eher rudimentär anzusehen. Für eine Analyse nehmen wir deshalb auf der Basis des gewählten V-Modell XT Metamodells Version 1.2.1 eine erneute Modellierung in der UML 2 vor. Wir stützen uns dabei auch auf Friedrich [Fri06] ab, verfeinern jedoch an einigen Stellen noch weiter, um die Gleichmäßigkeit der Modellierung auch für die anderen betrachteten Metamodelle zu erhalten.

Produktmodell

Das V-Modell XT ist ein *produktzentriertes* Vorgehensmodell. In Abbildung 3.1 ist das Produktmodell des V-Modells dargestellt. Ebenfalls ist dort die Verbindung zum Rollenmodell (Klasse `Rolle`) und zum dynamischen Modell (Klasse `Entscheidungspunkt`)

2 Task RO: Bezeichnet ein Teilprojekt des WEIT-Projekts mit den Aufgaben Redesign des V-Modell XT Metamodells zur Bereitstellung von Referenz- und Organisationsanteilen.

zu sehen. Die Produktzentrierung des V-Modells zeigt sich insbesondere an der stark ausgeprägten Abhängigkeitsstruktur zwischen Produkten. Diese *Produktabhängigkeiten* modellieren nicht nur Ergebnisstrukturen des Modells und dem zufolge eines Projekts, sondern darüber hinaus auch Zusammenhänge in und zwischen verschiedenen Projektabschnitten. Diese Rolle fällt in anderen Vorgehensmodellen vornehmlich den Aktivitäten zu.

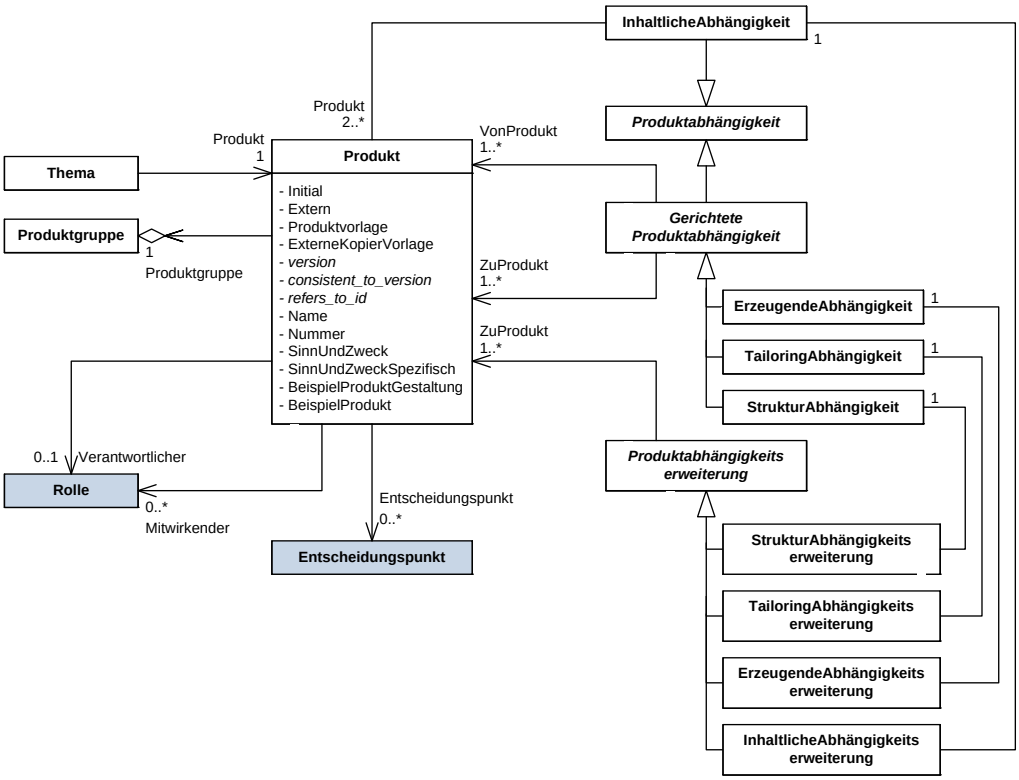


Abbildung 3.1.: Metamodell des V-Modell XT – Sicht: Produktmodell

Über die Richtung der Assoziationen des Modells in Abbildung 3.1 lassen sich Aussagen zur Integration des Konzepts Produkt treffen. Als erstes betrachten wir die referenzierten Typen. Ein Produkt referenziert genau eine *Produktgruppe*, der es zugeordnet wird. Die Produktgruppe ist ebenfalls ein Konzept des Produktmodells, weshalb es sich hier um eine Assoziation innerhalb eines Submodells handelt. Analog ist die Referenzierung zwischen *Themen* und Produkten zu betrachten, wobei ein Thema hier auf das Produkt zeigt, zu dem es gehört. Problematisch ist die Referenzierung von Rollen und Entscheidungspunkten, da es sich hier um Referenzierungen von Elementen anderer Submodelle handelt. Vom Gesichtspunkt der Integration werden hier klare Zuordnungen für Verantwortlichkeiten und Projektfortschrittsstufen getroffen. Die Erfahrung bei der Anpassung des V-Modells hat jedoch gezeigt, dass insbesondere bei Spezialisierungen diese engen Bindungen zwischen den Submodellen nicht unerhebliche Probleme verursachen können[3].

Die Produktabhängigkeiten sind ein feingranular modelliertes Konzept, das ein Abhängigkeitsgeflecht zwischen Produkten modelliert. Unterschieden wird im V-Modell dabei zwischen gerichteten und ungerichteten Abhängigkeiten. Zu letzteren zählt die *inhaltliche* Produktabhängigkeit, die sich in der Regel in Texten oder sonstigen Dokumentationen findet. Die gerichteten Abhängigkeiten sind hier für uns wesentlich interessanter, stellen sie doch das Mittel des V-Modells dar, um Strukturen zu erzeugen. *Struktu-*

3 Konkret betrifft dies das Rollenmodell, das in der Organisationsstruktur der Bundeswehr etabliert ist und im Rahmen der Anpassung des V-Modells entsprechend umgesetzt werden musste. Im Anhang A.1 geben wir einen tieferen Einblick in die Probleme.

relle Produktabhängigkeiten stellen dabei die Interpretation des V-Modells für Produktaggregationen dar, *erzeugende* Produktabhängigkeiten dienen der Definition von Quell- und Zielprodukten für Erstellungsprozesse. *Tailoringabhängigkeiten* dienen abschließend dazu, beim dynamischen Tailoring rückwirkend die Konsistenz und Vollständigkeit sicher zustellen. Obwohl das Abhängigkeitsmodell für Produkte sehr ausgeprägt ist, finden in der Praxis fast ausschließlich die erzeugenden Produktabhängigkeiten Anwendung.

Über die einfachen Produktabhängigkeiten hinausgehend sind im V-Modell auch noch so genannte *Produktabhängigkeitserweiterungen* definiert. Diese dienen dazu, vorhandene (zum Beispiel standardmäßig definierte Assoziationen) dynamisch ergänzen zu können. Beispielhaft wollen wir uns hier auf die `ErzeugendeAbhängigkeitserweiterung` (EAE) beziehen. Nehmen wir an, es existiert eine Relation $\textit{erzeugt}_1\,(p_1, p_2)$ zwischen zwei Produkten, sodass p_2 von p_1 erzeugt wird. Soll diese Relation so erweitert werden, dass zusätzlich auch p_3 erzeugt werden soll, dient die EAE zur Erweiterung der Relation wie folgt: $\textit{eae}\,(\textit{erzeugt}_1, p_3)$. Analog zu den Produktabhängigkeiten wird erfahrungsgemäß nur die erzeugende Produktabhängigkeitserweiterung breit angewendet. In einzelnen Überlegungen im Rahmen der Arbeiten am *Task RO* (Anhang A.3) wurde daher bereits über eine Konsolidierung der Abhängigkeitstypen nachgedacht.

Aktivitätsmodell

Das Aktivitätsmodell (Abbildung 3.2) des V-Modell XT ist im Vergleich zum Produktmodell nur wenig ausgeprägt. Aktivitäten sind hierarchisch analog zu Produkten organisiert (Gruppen, Aktivitäten und Teilaktivitäten). Über diese hierarchische Strukturierung gibt es jedoch keine weiteren Eigenschaften oder Strukturelemente. Aktivitäten sind Produkten zugeordnet (Relation zwischen Aktivität a_1 und Produkt p_1: $\textit{stellt\_fertig}\,(a_1, p_1)$, vgl. Abbildung 3.2). Damit werden Aktivitäten nicht als Planungsgrößen herangezogen.

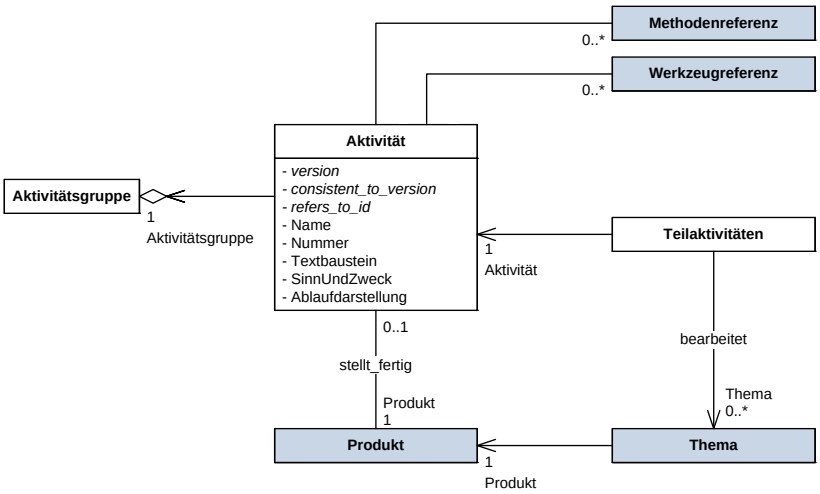


Abbildung 3.2.: Metamodell des V-Modell XT – Sicht: Aktivitätsmodell

Eine Strukturierung von Aktivitäten unterstützt das V-Modell nicht. Es werden in der Regel nur 1:1 Abbildungen vorgenommen. Einzig die Relation zwischen Teilaktivitäten und Themen ist $1..n$, wobei hier durch Mehrfachreferenzierung zwischen verschiedenen Teilaktivitäten und Themen auch $m..n$-Relationen möglich sind. Obwohl Aktivitäten nur als nicht zentrales Element behandelt werden, verweisen sie auf Methoden- und Werkzeugreferenzen. Somit hält eine Aktivität Referenzen auf bis zu drei weiteren Submodellen. Aktivitäten können somit als Koppelstelle zwischen Produkten und weiteren Submodellen dienen. Eine vollständige Entkopplung ist dadurch jedoch nicht erreicht, da das Produktmodell selbst auch weitere Submodelle anbindet.

Rollenmodell

Das Rollenmodell des V-Modell XT ist nur rudimentär ausgebildet. Rollen besitzen neben den üblichen Elementen (Versionsinformationen, Namen oder Beschreibungstexte) keinerlei weitere Assoziationstypen. So ist es im V-Modell beispielsweise nicht möglich, Teamstrukturen zu modellieren. Weiterhin kommt hinzu, dass Rollen nur von außen referenziert werden. Dies erschwert es, auf der inhaltlichen Ebene Aussagen zum Sinn und Zweck einer Rolle zu machen. Weiterhin ist über Rollen keinerlei Spezialisierungs- und Kompositionsmechanismus definiert, sodass Änderungen an Rollen nicht möglich sind (vgl. Anhang A.1). Rollen stellen somit ein konstantes Element im V-Modell XT dar.

3.2.2. Das Microsoft Solutions Framework Metamodell

Das zweite Vorgehensmodell, das wir näher betrachten wollen, ist das Microsoft Solutions Framework (MSF). MSF weist auch ohne tiefer gehende Analysen offensichtliche Modularitätseigenschaften auf. Jedoch ist für MSF keine explizite Metamodellbeschreibung verfügbar. In verschiedenen Arbeiten (zum Beispiel: [KT06, Kuh07]) haben wir daher mittels Reverse Engineering das Metamodell aus den Instanzen (vgl. Kapitel 2.1.2) ermittelt[4]. Diese besprechen wir im Folgenden.

Produktmodell

Das Produktmodell von MSF muss bereits differenziert betrachtet werden. Neben den so genannten *Work Products* gibt es im MSF noch so genannte *Work Items* (Abbildung 3.3, vgl. auch Kapitel 2.1.2). Work Products sind Ergebnisse, beziehungsweise Vorlagen oder Beispiele und somit mit den Produkten des V-Modell XT vergleichbar. Die Dokumentation des MSF ist hier nicht vollständig abgeschlossen, sodass durchaus weitere Produkttypen auftreten können. Eine formale Beschreibung der Produkte und Produktstrukturen in der Detaillierung des V-Modells weist MSF nicht auf. Assoziationen über Produkten werden nicht definiert.

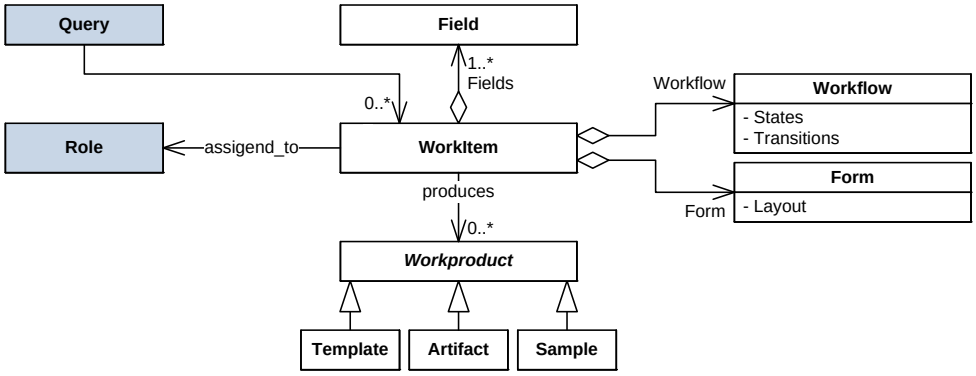


Abbildung 3.3.: Metamodell des MSF – Sicht: Produktmodell

Als zweites wichtiges Konzept im Produktmodell nehmen wir Work Items in die Betrachtung auf. Work Items könnten auch in das Aktivitätsmodell eingeordnet werden, jedoch verfügen sie über Daten (Fields), wie zum Beispiel Statusinformationen oder Historien. Weiterhin sind Work Items Zustandsmaschinen (im Workflow enthalten) zur Seite gestellt, weshalb wir sie von den strukturellen Eigenschaften näher an den Produkten des V-Modells sehen. Work Items repräsentieren alle Prozesselemente, die im

4 Ausgangspunkt für die Metamodellanalyse ist MSF in der Version 4.1. Da sich die Analyse auch auf den Inhalten der Instanzen und den zur Verfügung stehenden XML-Schemabeschreibungen abstützt, kann an dieser Stelle keine abgeschlossene und vollständige Aussage getroffen werden. Alle Aussage beziehen sich also auf den Analysestand.

Rahmen der automatischen Verfolgung und Überwachung berücksichtigt werden. Auf genaue Anwendungsmöglichkeiten gehen wir in Anhang A.2 noch genauer ein.

Work Items bieten hinsichtlich ihrer Modularitätseigenschaften interessante Optionen. Jeder Work Item-Typ wird durch eine XML-Datei beschrieben. Dem Typ liegt ein XML-Schema zugrunde, das allgemeine Work Item Typdefinitionen bestimmt. Die Grundstruktur dieses Metamodells ist in Abbildung 3.3 zu sehen. Die Beschreibung einzelner Work Item-Typen (zum Beispiel Aufgabe oder Bug) ist jeweils in einer separaten Datei hinterlegt. Über das Vorhanden- oder Nichtvorhandensein einer solchen Datei kann somit das inhaltliche Volumen des Prozesses skaliert werden. Microsoft demonstriert dies am Beispiel seiner beiden MSF-Derivate, die neben unterschiedlichen Mengen von Work Item-Typen auch noch über unterschiedliche Ausprägungen verfügen. Da Work Items auch eine eigene Methodik tragen, zum Beispiel methodische Anteile des Risikomanagements, die im Workflow des entsprechenden Work Item-Typs abgelegt sind, kann ein Prozess durch entsprechende Anpassung des Work Items oder dessen Entfall einem einfachen Tailoring unterzogen werden.

Eine Kopplung zu anderen Submodellen besteht im Produktmodell des MSF fast gar nicht. Lediglich eine Referenz zu einer Rolle wird einem Work Item mitgegeben. Diese Referenz wird jedoch bei genauerer Analyse nicht explizit im Metamodell ausgedrückt, sondern durch Variablenreferenzen an die Sicherheitsstruktur eines Windows Servers gebunden. Weitere Verbindungen nach „außen" sind nicht vorgesehen. Da der MSF nicht produktzentriert ist, werden die Strukturierung und Komposition der Einzelelemente an anderer Stelle – im Aktivitätsmodell – vorgenommen.

Aktivitätsmodell

MSF ist ein aktivitätszentrierter Prozess, was sich unmittelbar in der Komplexität des entsprechenden Metamodellanteils zeigt. Das Aktivitätsmodell des MSF (Abbildung 3.4) ist darüber hinaus hoch integriert und sehr eng an das Prozessmodell gekoppelt. Im Rahmen von Abschnitt 3.3 entflechten wir diese Modellanteile noch weiter; hier benötigen wir jedoch die komplette, integrierte Sicht. Zentraler Punkt des Aktivitätsmodells sind Aktivitäten und Sub-Aktivitäten.

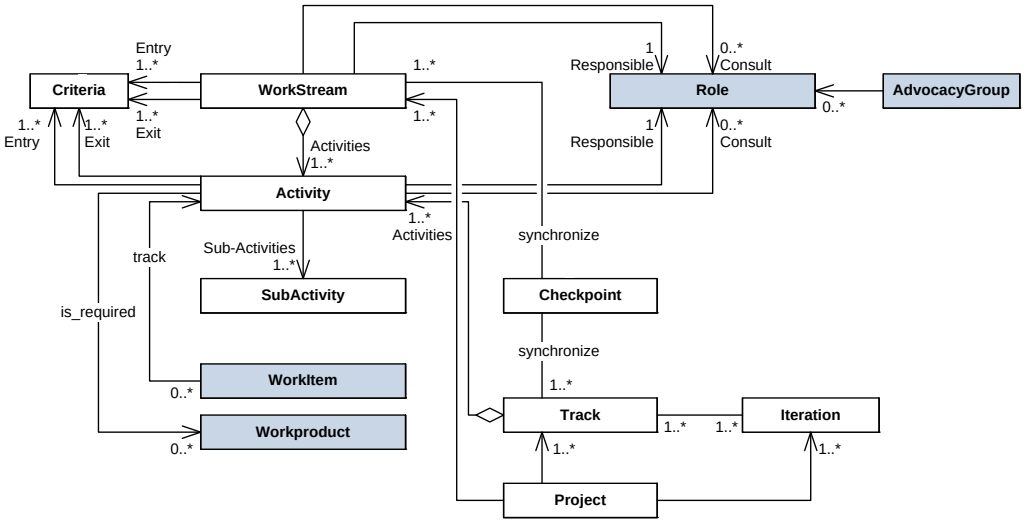


Abbildung 3.4.: Metamodell des MSF – Sicht: Aktivitätsmodell

Aktivitäten sind dabei Anlaufpunkt für Work Items und Work Products. Work Items lösen dabei üblicherweise Aktivitäten aus und verfolgen deren Status. Work Products dienen entweder als Eingabe für eine Aktivität oder als Ergebnis. Jeder Aktivität sind Eintritts- und Austrittskriterien zugeordnet. Für eine Aktivität ist genau eine Rolle verantwortlich; mehrere weitere können beratend hinzugezogen werden. Dies entspricht

im Wesentlichen dem Verantwortungs- und Mitwirkungskonzept des V-Modell XT angewendet auf Aktivitäten. Die Aktivität als Kernelement dieses Submodells referenziert somit Elemente aus zwei weiteren Submodellen. Die Referenzierung erfolgt nicht durch ausgezeichnete Elemente wie zum Beispiel die Produktabhängigkeiten beim V-Modell, sondern durch direkte Referenzen. Analog zum Produktmodell und dort wie Work Items sind Workstreams und Aktivitäten jeweils einzeln in XML-Dateien abgelegt.

Anders als im V-Modell XT definiert MSF Kompositionsoperationen über Aktivitäten. Mehrere Aktivitäten werden entweder zu so genannten *Workstreams* oder zu *Tracks* zusammengefasst. Tracks entsprechen dabei im Wesentlichen einer Projektfortschrittsstufe oder einer Projektphase[5]. Beispiele für Tracks sind Entwicklung oder Deployment. Workstreams stellen ein weiteres Konzept dar, um Aktivitäten zu bündeln. Hier werden Aufgaben kontextbezogen zusammengefasst und unter die Verantwortung einer Rolle gestellt. Daher verfügen Workstreams neben den Rollenzuordnungen auch über Ein- und Austrittskriterien. Tracks und Workstreams werden über so genannte Checkpunkte synchronisiert. Zusammen mit Iterationen stellen sie die Koppelstelle zum *Prozessmodell* her, das wir in Abschnitt 3.3 noch einmal genauer betrachten.

Rollenmodell

Das Rollenmodell des MSF ist ähnlich wie das des V-Modell XT flach und ohne Hierarchie. MSF führt zusätzlich noch einen Terminus *Advocacy Group* ein, der im Wesentlichen mit einem Teamkonstrukt erklärbar ist. Die Vorgängerversion, MSF 3.x hatte hierfür den Begriff des Rollenclusters. Im Metamodell selbst findet diese Konstruktion jedoch zunächst keine weitere Verwendung. Erst wenn es darum geht, MSF-Derivate für sehr kleine Teams zu bilden, gewinnen die Advocacy Groups an Bedeutung, da hier personengebundene Einzelrollen zusammengefasst werden können. Minimale MSF-Teams sollen so mit etwa 4 Personen realisierbar sein [Tur06].

Rollen besitzen von sich aus keine Referenzen in andere Submodelle hinein. Analog zum Produktmodell stehen Rollen somit vergleichsweise lose gekoppelt im MSF Gesamtmodell. Das Aktivitätsmodell übernimmt hier die Kopplung zwischen den einzelnen Submodellen. Zuständigkeiten von Rollen für Produkte werden transitiv über die Aktivitäten festgelegt, wobei eine eindeutige Zuordnung nicht immer gegeben ist.

3.2.3. OpenUP, UMA und Eclipse Process Framework

Das letzte Framework, das wir im Kontext Metamodell-basierter Vorgehensmodelle detailliert betrachten wollen ist die Unified Method Architecture (UMA), die dem Eclipse Process Framework (EPF) zugrunde liegt. Als Ausprägung haben wir in Kapitel 2.1.3 den Open Unified Process (OpenUP) betrachtet und dort auch das Anpassungskonzept analysiert. In diesem Abschnitt betrachten wir die zugrunde liegenden Metamodelle für die Strukturen des Prozesses genauer. Als einziges der bislang betrachteten Vorgehensmetamodelle ist UMA vollständig in UML modelliert. Friedrich [Fri06] hat hier bereits einige Aspekte expliziert, jedoch sind seine Ergebnisse nicht mehr konsistent zur aktuellen Version der UMA. Genauso wie das V-Modell XT ist auch UMA in einem stetigen Weiterentwicklungsprozess. Als Diskussionsgrundlage beziehen wir uns hier auf die Version 1.2[6]. UMA ist ein sehr komplexes Metamodell mit einer sehr hohen Integration der Einzelkomponenten. Wie bereits schon in Kapitel 2.1.3 andiskutiert, wird UMA an einigen Stellen bereits auf der Ebene der UML für Prozesse positioniert. Die damit einhergehende Omnipotenz des zugrunde liegenden Metamodells, jeden möglichen Aspekt der Vorgehens-/Prozessmodell-Modellierung adressieren zu können, hat direkte Auswirkungen auf die Komplexität des Metamodells. Wie in Kapitel 2.2.1 bereits diskutiert, sehen wir dies als sehr kritisch an. Im Folgenden konzentrieren wir uns daher zunächst wieder auf die Architektur der Submodelle für Produkt, Aktivitäten

5 Turner [Tur06] bezeichnet Tracks als das Nachfolgekonzept des MSF Phasenmodells, das bis zur Version 3 des MSF angewendet wurde.
6 Stand: 01. Juni 2007

und Rollen. Bereits dort werden wir die hohe Integration bemerken, weshalb wir sie in weiten Teilen zunächst nicht besprechen und dafür auf die Abschnitte 3.4 und 3.5 verweisen.

Produktmodell

Das Produktmodell (Abbildung 3.5) der UMA spezialisiert sich aus der Klasse `ContentElement` und wird unter dem abstrakten Kind `WorkProduct` zusammengefasst. UMA definiert darauf aufbauend die drei Ergebnistypen `Artifact`, `Outcome` und `Deliverable`.

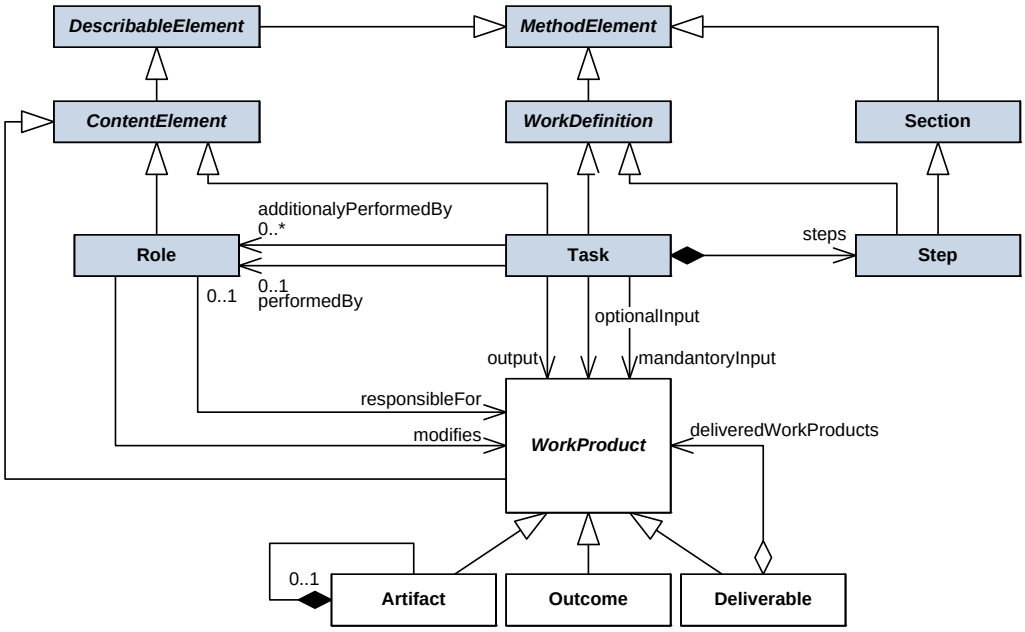


Abbildung 3.5.: EPF/UMA Sicht: Produktmodell

Outcomes werden durch UMA als Produkttypen beschrieben, die nicht formalisierbar beziehungsweise nicht zur Nachnutzung geeignet erscheinen, wie zum Beispiel eine formlose Feedbackmail eines Kunden. Die anderen beiden Produkttypen sind hingegen potenziell komposit, d. h. sie können selbst andere Produkttypen aggregieren. *Artefakte* besitzen hier sogar noch strengere Regeln, indem sie ausschließlich andere Artefakte mittels Kompositionsassoziation zusammenfassen. Ein Artefakt kann beispielsweise ein Dokument oder zum Beispiel Code sein. Ein *Deliverable* (eine Lieferung) kann beliebige Produkttypen zusammenfassen.

Das Produktmodell der UMA ist somit zunächst sehr einfach und ausreichend strukturiert, um alle relevanten Ergebnistypen zu modellieren. Zusätzlich ist durch eine gemeinsame Basisklasse `WorkProduct` ein definierter Aufsetzpunkt für Erweiterungen vorhanden. Betrachten wir die in Abbildung 3.5 modellierten Assoziationstypen, so ist festzustellen, dass Elemente des Produktmodells nur als Ziel auftreten. Das Produktmodell ist somit nicht zentral – UMA ist ein Metamodell zur Modellierung aktivitätsorientierter Vorgehensmodelle. Nichtsdestotrotz ist `WorkProduct` ebenfalls ein direktes Kind von `ContentElement`, genauso wie `Role` und `Task`, die hier die Schnittstelle zu den anderen Submodellen repräsentieren. Bereits hier ist festzustellen, dass `WorkProduct` zum Teil sogar nur transitives Ziel ist, d. h. `Task` referenziert `Rolle` und `Rolle` referenziert `WorkProduct`.

Aktivitätsmodell

Das Aktivitätsmodell der UMA (Abbildung 3.6) zeigt neben der Struktur auch ein grundlegendes, immer wieder zu findendes Designprinzip: *Indirekte Kopplung*. Wir betrachten zunächst die Klasse `ProcessElement`, die ebenfalls von `DescribableElement` abgeleitet ist und auf der selben Spezialisierungsebene wie `ContentElement` (Abbildung 3.5) ist. Im Gegensatz zu `ContentElement`, das „inhaltliche Anteile" eines Prozesses modelliert, repräsentiert `ProcessElement` die Anteile, die wir im weiteren Verlauf von Abschnitt 3.3 im Kontext der dynamischen Anteile noch näher betrachten. Die entsprechende Koppelstelle hierfür ist die Klasse `Activity`, die wiederum durch `Phase` und `Iteration` spezialisiert wird.

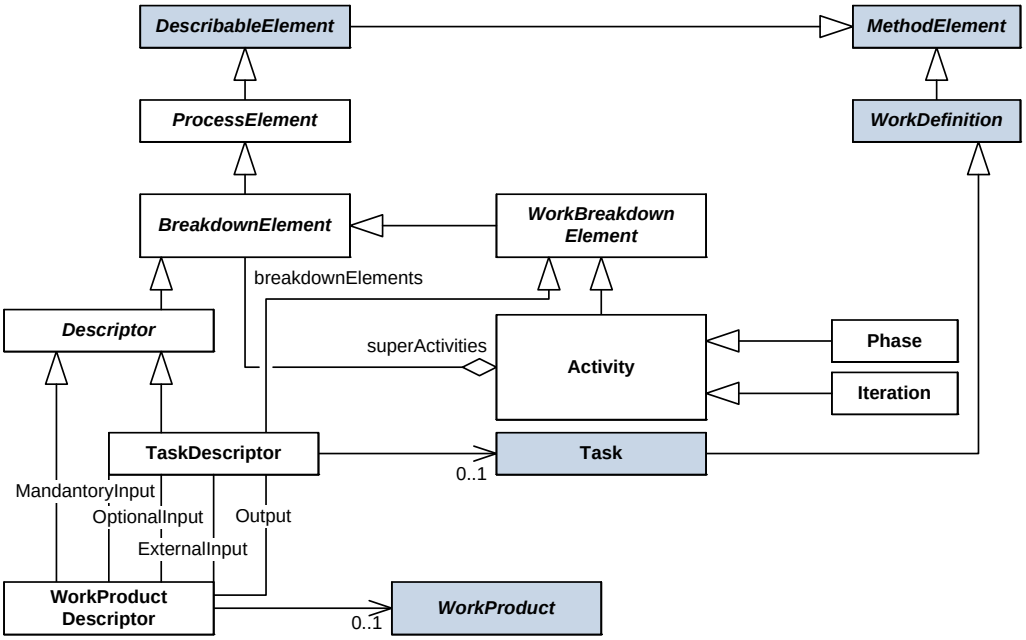


Abbildung 3.6.: EPF/UMA Sicht: Aktivitätsmodell

Eine Aktivität in der UMA ist bereits ein komposites Konstrukt, in dem mehrere Tasks zusammengefasst sind. Hierzu bedient sich UMA der Klasse `Descriptor`, die wie `Activity` ebenfalls ein Kind von `BreakdownElement` ist. Eine Aktivität kann mittels Aggregation mehrere Breakdown-Elemente zusammenfassen. Diese Breakdown-Elemente sind in diesem Teil Spezialisierungen von `Descriptor`, wie zum Beispiel `TaskDescriptor` oder `WorkProductDescriptor`. Assoziationen zwischen Tasks und Work Products werden also über entsprechende Deskriptorklassen aufgelöst. Instanzen dieser Klassen verweisen dann jeweils auf eine ihnen zugeordnete Instanz von `Task` oder `WorkProduct`. `Descriptor` realisiert somit ein *Proxy*-Muster für Elementtypen vom UMA. Die Kopplung von Elementen erfolgt somit über Stellvertreter, was auf der einen Seite eine hohe Abstraktion gestattet, auf der anderen Seite natürlich die strukturelle Komplexität der Instanzen erhöht.

Rollenmodell

Nach dem selben Muster wie das Aktivitätsmodell ist auch das Rollenmodell der UMA (Abbildung 3.7) aufgebaut. Die Basisklasse `Role` leitet sich aus `ContentElement` ab (vgl. auch Abbildung 3.5) und nutzt zur Kopplung an andere Elemente eine Spezialisierung der Klasse `Descriptor`.

Als einziges der bislang betrachteten Metamodelle definiert UMA ein explizites Konzept zur Teambildung und somit ein hierarchisches Rollenkonzept, das Prozessingenieuren als Option zur Verfügung steht. Einerseits können mehrere (einfache) Rollen

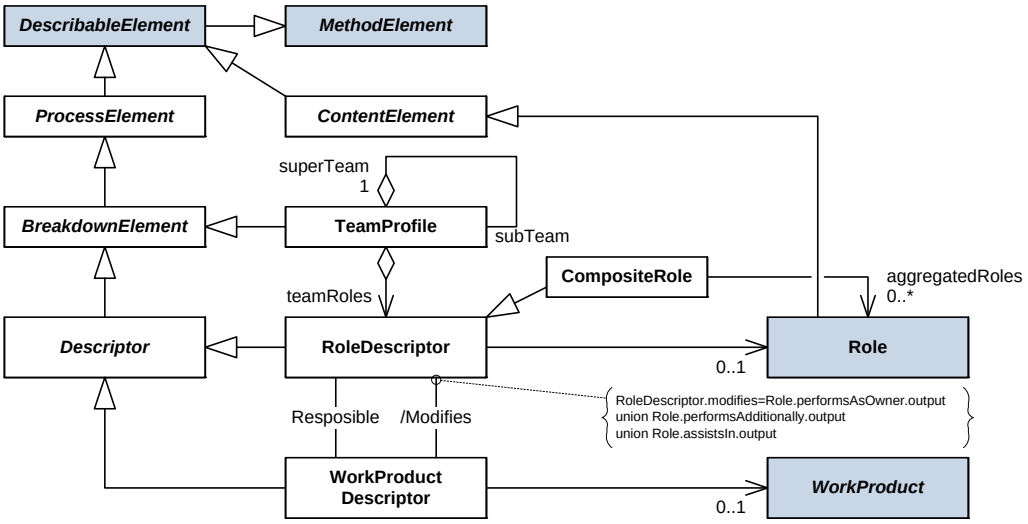


Abbildung 3.7.: EPF/UMA Sicht: Rollenmodell

zu einer `CompositeRole` (als Spezialisierung von `RoleDescriptor`) zusammengefasst werden. Andererseits können Rollen in einem Team aggregiert werden, die selbst wieder strukturierbar sind. Die Aggregation von Rollen (unabhängig davon ob im Team oder zu kompositen Rollen) erfolgt wieder durch Deskriptoren. Analog zum Aktivitätsmodell erfolgt somit die Referenzierung von Elementen aus anderen Submodellen indirekt.

3.2.4. Weitere Ansätze

Weitere Ansätze, die hier zu analysieren wären finden wir in Vorgehensmodellen, beziehungsweise Frameworks mit einem Metamodell als Basis. Wir wollen hier explizit nur auf zwei weitere Vertreter eingehen und diese nicht sehr weit vertiefen. Im Folgenden betrachten wir *OPEN* und *OEP* als Vertreter für ein Framework und ein Vorgehensmodell. Im Anschluss positionieren wir noch *Process Pattern*. Hinzu kommt mit dem *ISO/IEC 24744:2007* ein Standard für die Metamodellierung von Prozessmodellen, den wir ebenfalls kurz betrachten.

OPEN Process Specification/Framework

Das OPEN[7] Process Framework (OPF) ist ein lizenzfreies Framework für den Entwurf von Prozessmodellen. Maßgebliche Informationen zu OPEN, auf die wir uns auch hier weitgehend abstützen, sind im Wesentlichen auf der Webseite von D. Firesmith[8] zu finden, beziehungsweise [GHSY97, FHS01] zu entnehmen. Die Quellen im Web sind jedoch seit Dezember 2005 nicht mehr aktualisiert worden, weshalb OPEN hier nur eine untergeordnete Rolle spielt. Mit dem 2007 erschienen ISO/IEC 24744 Standard findet sich jedoch eine Art Weiterentwicklung von OPEN. Wir gehen darauf im nächsten Abschnitt kurz ein.

Abbildung 3.8 zeigt einen Anteil von OPEN, der im Zentrum die (abstrakten) Basisklassen zeigt. Allgemeine Basisklasse ist `Method Component`, von der alle weiteren Elemente des Frameworks ableiten (siehe polymorphiebasierte Klassenbibliothek [HR02]). Firesmith bezeichnet diese Klasse auch als `Process Component Class`, wobei hier aufgrund der Architektur kein Komponentensystem vorliegt. Kinder von `Method Component` sind `Endeavor`[9], `Producer`, `Language`, `Stage`, `Work Unit` und `Work Product`. Mit

7 OPEN: Object-oriented Processes, Environment and Notation
8 OPEN im Web: `http://www.donald-firesmith.com`, Stand: 15.06.2007
9 Im weitesten Sinne und am treffendsten sicherlich mit „Unternehmung" zu übersetzen. OPEN modelliert

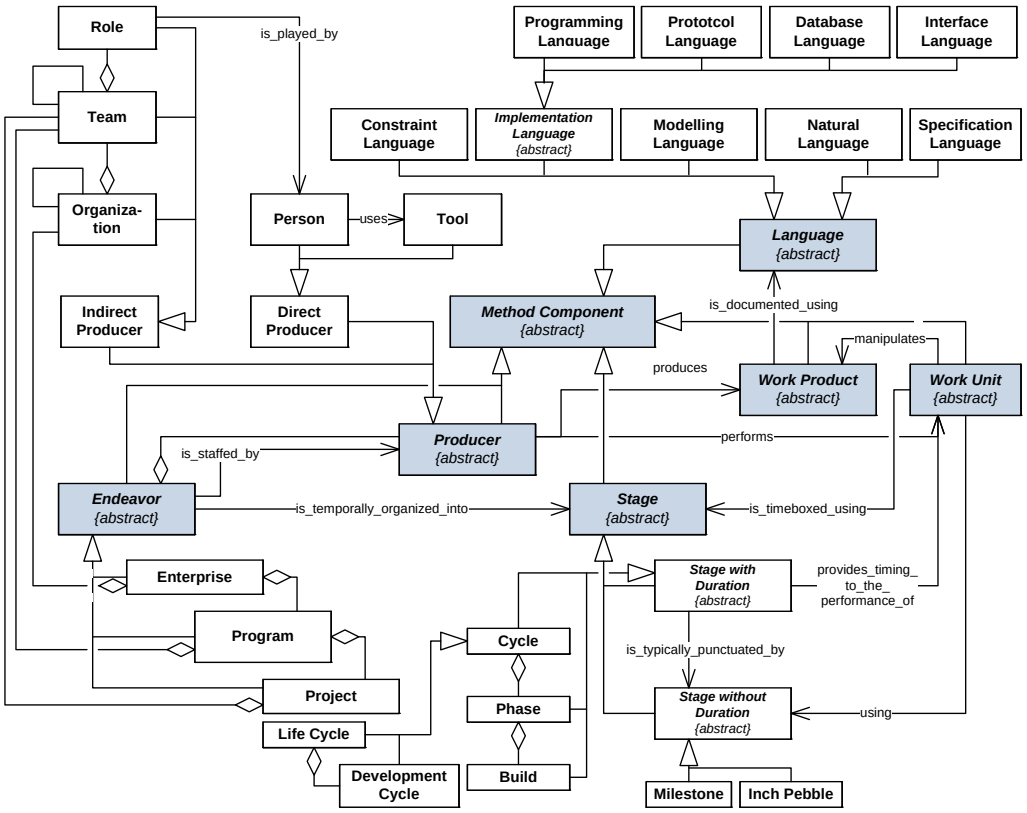


Abbildung 3.8.: Metamodell des OPF

Producern modelliert OPEN sein *Rollenmodell*, das auch Organisationsstrukturen unterstützt. Hierzu besteht eine enge, hierarchische Verknüpfung zu den Kindern von `Endeavor`. Mit Language werden Sprachfamilien dargestellt, die verwendet werden können, um Work Products zu beschreiben[10]. Mit Stages werden Projektstrukturen beziehungsweise Planungsstrukturen definiert. Kinder von `Stage` sind zum Beispiel `Phase`, `Cycle` oder `Milestone`.

Mit `Work Product` und `Work Unit` (Abbildung 3.9) modelliert OPEN sein *Produktmodell* und sein *Aktivitätsmodell*. Unter `Work Product` finden sich eine Reihe konkreter Kinder (die wir hier ausgespart haben), zum Beispiel Architektur- und Spezifikationsprodukte, Modelle oder Code. Unter `Work Unit` wird das Aktivitätsmodell gebildet, das Aufgaben (`Task`) hierarchisch ordnet und in Workflows oder (kompositen) Aktivitäten zusammenfasst.

OPEN verfügt über eine Reihe interessanter Konzepte, die wir auch für die vorliegende Arbeit aufgreifen. Hierzu zählen insbesondere die mehrstufigen, hierarchischen Produkt- und Aktivitätsmodelle, insbesondere deren Entkopplung von einem *Ablaufmodell*. An OPEN fällt jedoch auf, dass es intensiv von Vererbungsmechanismen (der engsten möglichen Kopplung zwischen zwei Modellelementen) Gebrauch macht und kein explizites Modulkonzept anbietet. OPEN ist sehr reichhaltig, aber auch explizit auf objektorientierte Techniken ausgelegt (Software Entwicklung, Geschäftsprozessmodellierung). Nach Stand von Firesmiths Webseite ist OPEN wie gesagt seit 2005 nicht mehr aktualisiert worden. Teile von OPEN, insbesondere durch die Mitwirkung von B. Henderson-Sellers sind jedoch in den neuen ISO Standard für Vorgehensmetamodelle eingeflossen.

mit diesem Element Organisationsstrukturen und kann – bei passender Konkretisierung – Programm- und Projektmanagement adressieren.

10 Je nach Art Work Product kann Language zum Beispiel spezialisiert werden zu einer Modellierungssprache oder einer Programmiersprache.

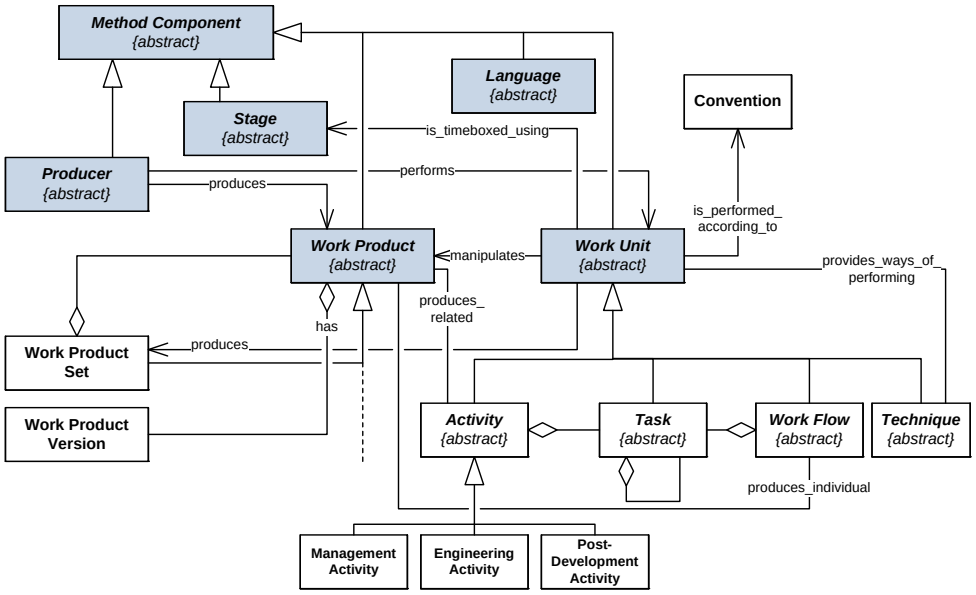


Abbildung 3.9.: Metamodell des OPF (Ausschnitt Workunits und Produkte)

ISO/IEC 24744:2007

Der ISO/IEC 24744:2007 *Software Engineering – Metamodel for Development Methodologies* (SEMDM) definiert ein standardisertes Metamodell für Systementwicklungsprozesse. In [Joi07], Seite 75ff. findet sich eine Konventionsabbildung zwischen OPEN und SEMDM.

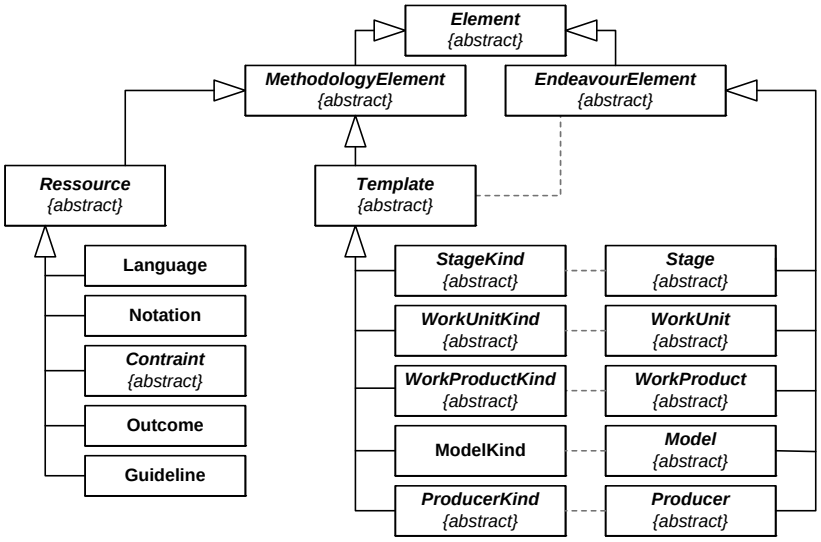


Abbildung 3.10.: Metamodell (Ausschnitt Basiskonzepte) des ISO/IEC 24744:2007

Wir gehen an dieser Stelle nur kurz auf SEMDM ein. Da in [Joi07], Seite 2 Aussagen hinsichtlich der Konformität von Vorgehensmetamodellen zum ISO Standard zu finden sind:

> „... A metamodel is defined in accordance with this [...] Standard if it:
>
> - describes the scope of the concepts in the metamodel in relation to the scope of the elements defined in Clause 7; and

- defines the mapping between the concepts that are addressed in the metamodel, and that are within the scope of this International Standard, and the corresponding elements of this International Standard (i.e. its elements cannot be substituted by others of identical intent but different construction)…"

liegt es nahe, eine Einstufung vornehmen. In Abbildung 3.10 haben wir die wesentlichen Elemente von SEMDM zusammengestellt. Die Ähnlichkeit zu OPEN fällt auf (vgl. Abbildungen 3.8 und 3.9). SEMDM führt jedoch auch ein duales Modellierungskonzept ein, in dem Entitäten in Methodik- und in „Endeavour"-Kontexten parallel definiert sind. [Joi07] beschreibt hierzu ein so genanntes *Powertype Pattern*, in dem `*Kind`-Klassen methodische Entitäten erfassen, die durch Endeavourelemente konkretisiert werden. Beispielhaft sei hier ein Klassendiagramm genannt, das durch ISO/IEC 24744 als spezieller Dokumenttyp (`DocumentKind`) für den Methodenkontext beschrieben wird; im Kontext eines Projekts (Spezialisierung von Endeavour) jedoch als konkretes Dokumentexemplar (`Document`). Im V-Modell XT ist dieser Dualismus in der Modellierung auch zu finden. Alle durch das V-Modell definierten Produkte sind genau genommen *Produkttypen*, die mithilfe von Vorlagen im Rahmen eines Exports in konkrete Produktexemplare überführt werden. Auch zwischen Entscheidungspunkten und Ablaufentscheidungspunkten besteht dieser Dualismus.

Der oose Engineering Process

Als letztes konkretes Vorgehensmodell mit zugrunde liegendem Metamodell wollen wir noch kurz auf den *oose Engineering Process* (OEP) eingehen. Dieses Vorgehensmodell basiert zwar auf einem formalen Metamodell inform eines UML 2 Modells (Abbildung 3.11), nutzt die damit einhergehenden Vorteile (zum Beispiel Werkzeugunterstützung) aber nicht.

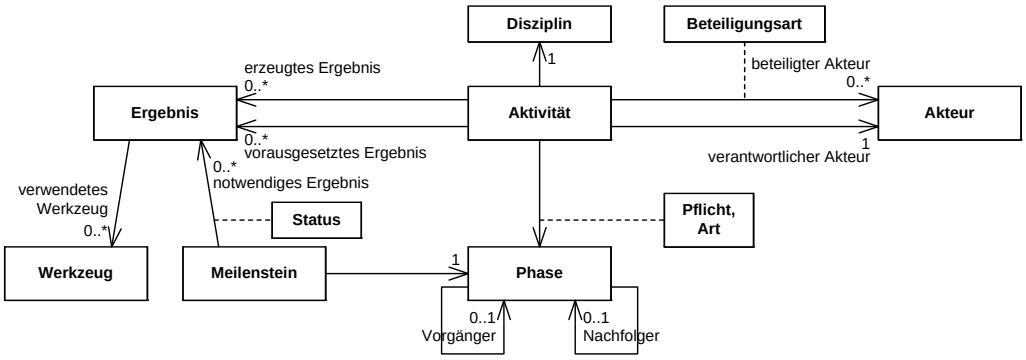


Abbildung 3.11.: Metamodell des OEP nach [OSKZ06]

Das Metamodell des OEP betrachten wir aber deshalb in dieser Arbeit, weil es sich von allen anderen betrachteten Vorgehensmetamodellen und Frameworks durch seine Einfachheit abhebt und die Komplexität des Prozesses in die inhaltliche Ausgestaltung verschiebt. Das OEP Metamodell führt lediglich 7 Entitäts- und 3 (explizite) Assoziationsklassen. Zusätzlich sind zwischen diesen Elementen 11 Assoziationstypen definiert. Die einzelnen Submodelle eines Vorgehensmodells sind jeweils durch eine einzelne Klasse repräsentiert. Inhaltlich werden diese unmittelbar instanziiert und nicht durch weitere Vererbungshierarchien strukturiert.

Process Pattern

Prozessmuster (Process Pattern) sind wiederverwendbare Teilprozesse, die zu umfassenderen Teilprozessen und Prozessen zusammengefasst werden können. Industriell

umgesetzt werden Prozessmusteransätze jedoch nur vereinzelt und in stark interpretierter Form. Beispiele für solche musterbasierten Konzepte finden sich im V-Modell XT mit seinen Vorgehensbausteinen oder im OpenUP mit seinen Capability Pattern. Abbildung 3.12 zeigt ein Metamodell Process Pattern auf der Basis von [GMP+03] (Seite 14). Auch hier finden wir Repräsentanten der bereits mehrfach untersuchten Submodelle.

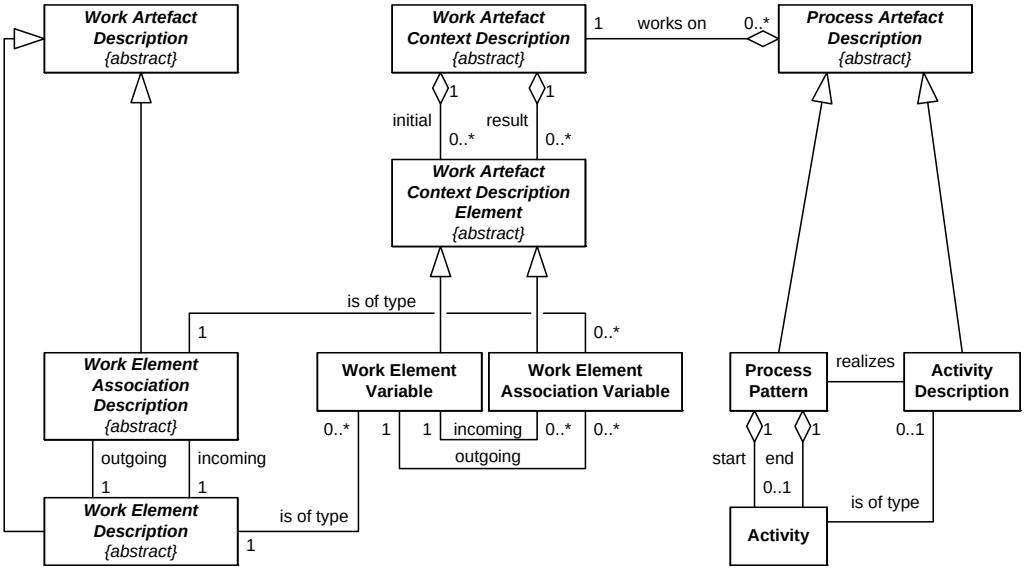


Abbildung 3.12.: Metamodell von Process Pattern (Auszug)

Process Pattern nach Gnatz et ál. verknüpfen Arbeitsergebnisse und korrespondierende Aktivitäten über einen Kontext. Ein- und Ausgabestati sind somit forderbar und prüfbar. Da Prozessmuster in der Regel Teilprozesse modellieren, sind somit Vor- und Nachbedingungen für die Kombination möglich.

3.3. Ablaufbezogene Anteile von Vorgehensmodellen

In diesem Abschnitt betrachten wir die Metamodellanteile der betrachteten Frameworks, die die strukturellen Elemente im Projekt instanziieren und planbar machen. In Kapitel 2.4 haben wir dieses Submodell als *Ablaufmodell* bezeichnet. Ablaufmodelle zu konstruieren ist eine große Herausforderung, da sich Abläufe kontextabhängig auf inhaltliche Volumina eines konkreten Vorgehens einstellen müssen. Gerade im Kontext der automatischen Plangenerierung aus einem konkreten Vorgehensmodell heraus muss sichergestellt werden, dass Abläufe und Inhalte miteinander verträglich sind [Mün01, Gna05].

3.3.1. Projektabläufe und -pläne

Wir beschäftigen uns zuerst mit grobgranularen Ablaufstrukturen und betrachten hierbei zuerst das V-Modell XT, das mit seinem Konzept *Projektdurchführungsstrategie* (PDS) eine allgemeine Ablaufplanvorlage bereitstellt (vgl. Kapitel 2.1.1). Die PDS stellt dabei nur eine Ablaufstruktur dar, die jedoch keine inhaltlichen Aussagen hinsichtlich zu bearbeitender Volumina trifft. Sie definiert lediglich eine Reihe von Projektfortschrittsstufen, die in einer bestimmten Reihenfolge zu erreichen sind. Hierzu werden im V-Modell Entscheidungspunkte verwendet, die die Schnittstelle zwischen Ablauf und Inhalt bilden. In Abbildung 3.13 ist der betreffende Ausschnitt des V-Modell-Metamodells dargestellt, der die Ablaufkonzepte enthält.

Wie beim ISO 24744 Metamodell, ist auch im Ablaufmodell des V-Modells die Trennung zwischen Konzept und Instanz gut zu erkennen. Diese Trennung erfüllt hier auch

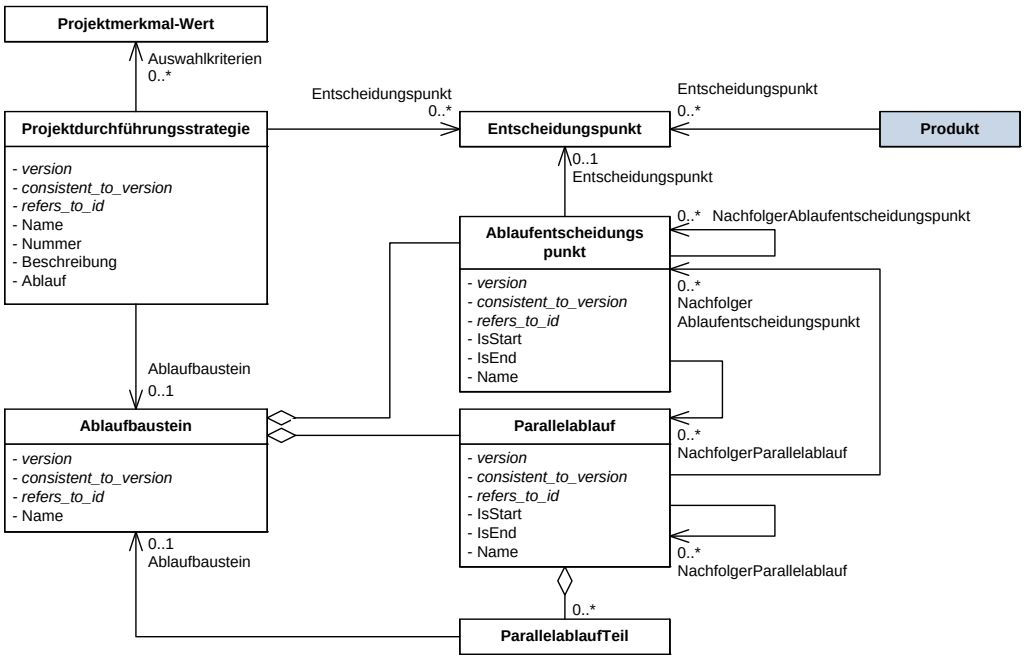


Abbildung 3.13.: Metamodell des V-Modell XT – Sicht: Projektdurchführung und Dynamik

noch einen anderen Zweck als die Abstraktion der Methode von der implementierenden Organisation: Planungselemente des V-Modells können für die Projektplangenerierung automatisiert ausgewertet werden. Dazu ist der PDS ein so genannter *Ablaufbaustein* zugeordnet, der entweder *Ablaufentscheidungspunkte* oder *Parallelabläufe* enthalten kann. Ablaufentscheidungspunkte sind die Instanzen der Entscheidungspunkte für einen konkreten Ablauf. Sie verfügen über eine Nachfolgerrelation, durch die sie eine Ordnung über den Entscheidungspunkten, die sie referenzieren, bilden. Diese Relation repräsentiert sozusagen die Pfeile in den grafischen PDS-Darstellungen (vgl. Abbildung 2.2), also die möglichen Pfade zwischen den einzelnen Projektfortschrittsstufen. Analog dazu können Parallelabläufe für die Realisierung und Planung nebenläufiger Projekte verwendet werden. Die Schwierigkeiten, die damit einhergehen beschreibt zum Beispiel Bergner [Ber06]. Ansonsten ist das Ablaufmodell des V-Modells flach. Über Entscheidungspunkte hinaus gibt es keine weiteren Strukturierungseinheiten. Planungsgrößen sind aufgrund der Philosophie des V-Modells die Produktmengen, die an den Entscheidungspunkten fertiggestellt sein müssen. Die für die Plangenerierung erforderlichen Vorgänge werden transitiv aus den assoziierten Aktivitäten ermittelt. Die Begriffe *Phase* oder *Iteration* kennt das V-Modell hingegen nicht.

MSF. Über die anderen beiden Vorgehensmetamodelle EPF/UMA und MSF können wir vergleichsweise schnell hinweg gehen, da wir deren Optionen für Abläufe im Wesentlichen schon dargestellt haben. MSF nutzt direkt die im Aktivitätsmodell (Abschnitt 3.2.2) hinterlegten Elemente. Zentraler Punkt hierbei sind die Work Items, die entsprechend instanziiert zur Projektplanung herangezogen werden können. Work Items fassen alle hierfür notwendigen Daten, zum Beispiel Beginn, Ende und geschätzter Aufwand einer Arbeitsaufgabe zusammen. Über ein entsprechendes Tooling werden aus selektierbaren Mengen von Work Items Microsoft Project-Pläne[11] erzeugt, die im Rahmen der Projektplanung auch automatisch synchronisiert werden. Anders als beim V-Modell ist das aber keine Fähigkeit des Vorgehensmodells, sondern der zugrunde liegenden Plattform. Dies hat auch zur Folge, dass es keine automatische Assoziation

11 Es stehen für verschiedene Anwendergruppen auch noch weitere Frontends wie zum Beispiel Microsoft Excel zur Verfügung.

von Planungseinheiten zu Ergebnistypen gibt[12]. Diese Assoziationen sind zwar in der Prozessdokumentation beschrieben, werden jedoch nicht automatisiert. Die Konzepte sind jedoch verfügbar (vgl. Abbildung 3.4):

- Checkpoint
- Track
- Iteration

Über Checkpunkte werden Workstreams (Sammelaktivitäten) und Tracks miteinander synchronisiert. Ein Checkpunkt entspricht somit im Wesentlichen einem Meilenstein oder Quality Gate. Tracks sind ein an Phasen angelehntes Konzept, das jedoch analog zu den Projektfortschrittsstufen des V-Modell XT überlappend umgesetzt werden kann. Iterationen sind ebenso wie Tracks Planungsgrößen eines Projekts, wobei Iterationen durch die MSF-Werkzeugkette unterstützt und in Projektplänen als Selektionskriterien berücksichtigt werden können.

Beispiel: Um dies zu verdeutlichen, nehmen wir als Beispiel den Track *Systemerstellung* an, der in mehreren Iterationen bearbeitet wird. Ein weiterer Track *Projektmanagement* findet kontinuierlich über das gesamte Projekt hinweg statt, unabhängig, wie viele Iterationen in bestimmten Projektabschnitten durchlaufen werden.

EPF/UMA. Abbildung 3.14 zeigt den Ausschnitt von EPF/UMA, der das Prozess- beziehungsweise Ablaufmodell beschreibt. Im reich ausgestalteten Aktivitätsmodell (Abschnitt 3.2.3, Abbildung 3.6) sind bereits Aufgaben (Tasks) und Arbeitsergebnisse (Work Products) dargestellt. Im Ablaufmodell wird die Klasse `Activity` zum Container ausgebaut, der durch `Process` spezialisiert wird. Ein Prozess ist somit eine spezielle Aktivität höherer Integrationsdichte. EPF/UMA unterscheidet darauf aufbauend noch weitere spezielle Prozesse, Planungsvorlagen (`ProcessPlanningTemplate`), Process oder Capability Pattern, Auslieferungs- und Erweiterungsprozesse. Auslieferungsprozesse (*Delivery Process*) sind in etwa mit den Projektdurchführungsstrategien des V-Modells vergleichbar. Erweiterungsprozesse (*Process Contribution*) sind methodische Add-Ons zu einem definierten Prozess.

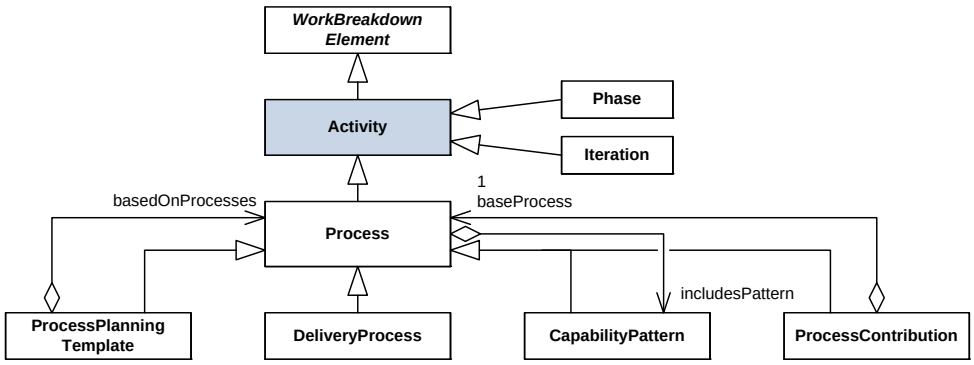


Abbildung 3.14.: EPF/UMA Sicht: Prozess-/Ablaufmodell

EPF/UMA ist prinzipiell planbar. Entsprechende Assoziationen zwischen Vorgängen, Aufgaben und Ergebnistypen sind vorhanden. Aufgrund des Metamodells ist der direkte Bezug jedoch nicht sofort erkennbar, da die Verknüpfung oftmals über Basisklassen in mehreren Vererbungsstufen erfolgt, die auf verschiedene Art miteinander verknüpfbar sind. Dies beinhaltet eine hohe Grundflexibilität, geht jedoch auf Kosten der Einfachheit und Verständlichkeit, sodass ohne entsprechende Werkzeuge die Planung auf Basis EPF/UMA aufwändig ist. Für die betrachtete Kombination EPF/UMA und

12 In der Tat ist es sogar in dieser Umgebung äußerst schwer präskriptiv derartige Abhängigkeitsstrukturen abzubilden. Im Projekt *CollabXT* war dies u.a. eine der Herausforderungen, die Abhängigkeiten aus dem V-Modell XT so aufzubereiten, dass sie im Team Foundation Server adäquat und weitgehend automatisch umgesetzt werden konnten.

OpenUP ist zum Zeitpunkt der Analyse darüber hinaus noch kein Werkzeug verfügbar, das den Planungsfähigkeiten des V-Modells nahe kommt. Professionelle, kommerzielle Ausprägungen wie der RUP verfügen über entsprechende Möglichkeiten.

3.3.2. Workflows

Die grob granulare Planung wird im MSF und in EPF/UMA nicht so stark fokussiert, wie im V-Modell. Dies liegt mitunter auch daran, dass das V-Modell durch seine Produktorientierung hier mehr Vorarbeit leisten muss, um dem Anwender einen Mehrwert zu bieten. Dadurch, dass der Anwender durch bloße Assoziation von Projektfortschrittsstufen und Produktmengen keine Handlungsanweisungen erhält, muss ihm eine Möglichkeit der (automatischen) Planerstellung inklusive einer automatisierten Vorgangsermittlung angeboten werden. In aktivitätsorientierten Vorgehensmodellen ist eine solche Translation nicht erforderlich, da die Planungseinheiten (Aktivitäten, Vorgänge etc.) direkt vorliegen. Aktivitätsorientierte Vorgehensmodelle liefern darüber hinaus auch weitere strukturierende Maßnahmen zur Bildung größerer Einheiten. Die Bezeichnungen sind hier vielfältig und gleichberechtigt unscharf. Allen gemein ist, dass sie mehrere Aktivitäten zusammenfassen und in der Regel Produktflüsse beschreiben.

V-Modell XT. Das V-Modell XT definiert *keine* aktivitätsbasierten Komposita. Es kennt keine Workflows, Arbeitspakete oder ähnliches. Prinzipiell ist es möglich, alle Aktivitäten in einem V-Modell-Projekt zum Zeitpunkt t_0 gleichzeitig zu starten. Lediglich zu den Entscheidungspunkten müssen alle die Aktivitäten, die die vorzulegenden Produkte erstellen, abgeschlossen werden. Zum Zeitpunkt t_{ende} müssen dann alle Aktivitäten abgeschlossen sein. Dadurch, dass Aktivitäten im V-Modell flach sind und keinerlei Strukturierung haben, ist dieses Vorgehen möglich, aber nicht sinnvoll. Als *work around* können im V-Modell jedoch die *erzeugenden Produktabhängigkeiten* dienen (vgl. Abbildung 3.1). Erzeugende Produktabhängigkeiten legen eine Erstellungsordnung fest. Sie definieren „Quellprodukte" (V-Modell Terminologie: initiale Produkte) und „Zielprodukte" (V-Modell Terminologie: abhängige Produkte). Implizit lässt sich somit aus der Reihenfolge der Erstellung der einzelnen Produkte eine analoge Reihenfolge der Ausführung der assoziierten Aktivitäten abbilden. Echte Produktflüsse sind dies indes nicht. Und auch eine Komposition im Sinne eines Arbeitspakets liegt nicht vor, da beispielsweise die Zusammenführung im Entscheidungspunkt erfolgt und dieser nicht als Sammelvorgang, sondern als Meilenstein modelliert wird.

MSF. MSF definiert mit seinen *Workstreams* einen Sammelvorgang. Hierbei muss gleichzeitig noch einmal die Terminologie berücksichtigt werden: Microsoft ordnet einem Work Item einen Workflow zu (vgl. Abbildung 3.3). Dies ist jedoch kein Workflow im eigentlichen Sinne, sondern ein einfacher Zustandsautomat. Produktflüsse oder die Kopplung mehrerer Aktivitäten finden *nicht* auf der Ebene von Work Items statt, sodass wir größere Einheiten verwenden müssen.

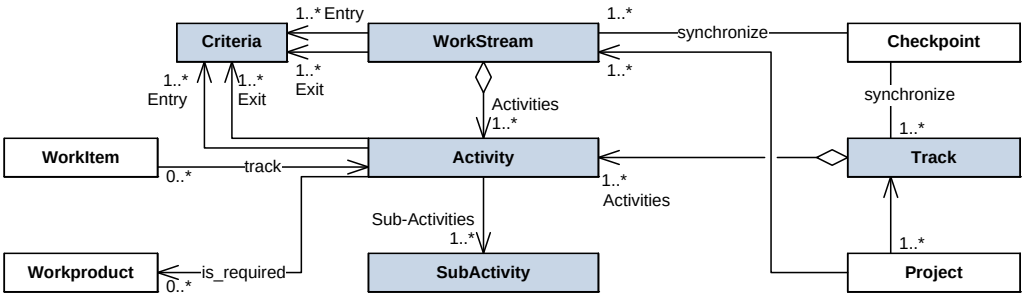


Abbildung 3.15.: MSF Sicht: Prozess-/Ablaufmodell mit Fokus Workflows

Abbildung 3.15 zeigt den hier für uns relevanten Ausschnitt des MSF Metamodells. Zentral sind zwei Klassen: `Workstream` und `Track`. Beide werden über Checkpunkte (Meilensteine) synchronisiert; beide fassen eine Menge von Aktivitäten zusammen. Das Zusammenfassen der Aktivitäten erfolgt zwischen Workstreams und Tracks überschneidend. Workstreams sind logische Aktivitätsgruppierungen (entfernt vergleichbar mit den Aktivitätsgruppen des V-Modells), während Tracks die Aktivitäten zusammenfassen, die für das Erreichen einer Projektfortschrittsstufe erforderlich sind. Wie bereits im letzten Abschnitt skizziert, kann sich ein Workstream über mehrere Tracks erstrecken, während ein Track mehrere Workstreams beinhalten kann. Die Anbindung von Ergebnistypen passiert jedoch eine Ebene tiefer bei den aggregierten Aktivitäten. Die Assoziation `is_required` zwischen `Activity` und `Workproduct` ist im Wesentlichen eine Assoziation die anzeigt, dass ein Produkt von einer Aktivität bearbeitet wird. Wie das genau passiert, wird im Wesentlichen über `SubActivity` geregelt. So kann eine Aktivität mehrere Produkte bearbeiten. Die Statusüberwachung wird einerseits (automatisiert) von Work Items übernommen. Andererseits sind durch die Definition von Entry- und Exit-Kriterien auch Überwachungsmechanismen für eine QS definiert, die zum Beispiel regeln, wann eine Aktivität begonnen werden kann oder wann zum Beispiel ein Workstream beendet werden kann.

EPF/UMA. Auch für EPF/UMA ist eine Gruppierung von Aktivitäten vorgesehen. In der für diese Arbeit verwendeten Version 1.0.2 der UMA ist eine Definition eines Workflowpakets bereits vorgesehen, jedoch noch nicht ausgearbeitet. Aufgrund der Architektur von EPF/UMA ist jedoch davon auszugehen, dass die Konzepte der bereits vorhandenen Modellanteile wiederverwendet werden. In Abbildung 3.16 haben wir das Metamodell noch einmal mit Fokus auf Aktivitäten und Aktivitätsgruppierungen aufgeführt. Hier wird auch wieder die objektorientierte Philosophie deutlich: Eine Aktivitätsinstanz kann beliebige `BreakdownElement`-Instanzen aggregieren. Dies können einmal wieder Aktivitäten sein, die (indirekt) ebenfalls Kinder von `BreakdownElement` sind, beziehungsweise wieder deren Kinder (zum Beispiel Prozesse). Andererseits ist hier zu sehen, dass unter anderem auch Deskriptoren referenziert werden können. Spezielle Deskriptoren sind dann eben zum Beispiel Instanzen von `TaskDescriptor` und `WorkProductDescriptor`. Zwischen diesen können dann wiederum weitere Assoziationen bestehen, zum Beispiel Input- oder Output-Beziehungen (vgl. auch EPF/UMA Produktmodell in Abbildung 3.5).

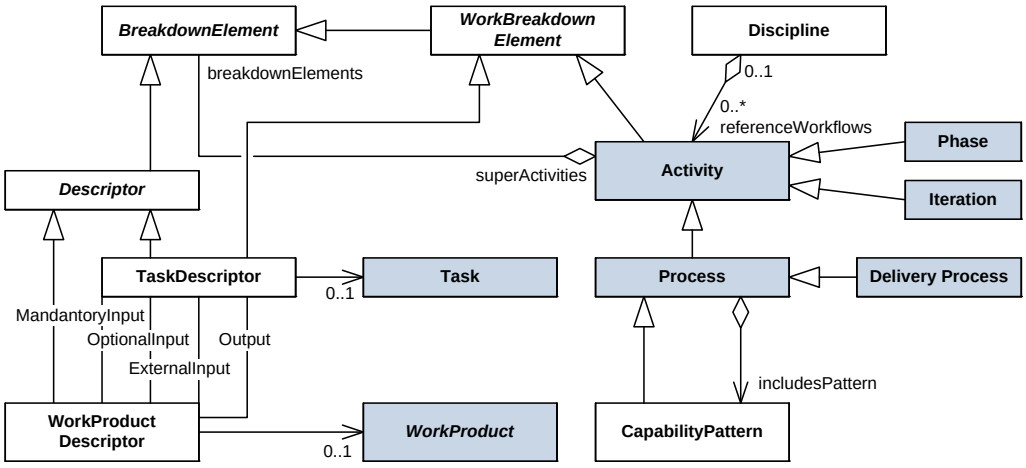


Abbildung 3.16.: EPF/UMA Sicht: Prozess-/Ablaufmodell mit Fokus Workflows

Durch dieses stark strukturierte System lassen sich vielfältige Verarbeitungs- und Kompositionsmuster abbilden. So zum Beispiel wird durch die Deskriptoren ein Produktfluss modellierbar, den das V-Modell gar nicht und MSF nur unscharf darstellen kann. Wie jedoch im letzten Abschnitt schon einmal ausgeführt, geht diese Komplexität und

dieser Funktionsumfang zu Lasten der Einfachheit. Durch die stark polymorphe Auslegung von EPF/UMA ist an dieser Stelle noch gar keine Aussage darüber möglich, *was* als Planungsgröße zu berücksichtigen ist.

3.3.3. Abgrenzung zur Modellierung von Geschäftsprozessen

Da wir uns in diesem Abschnitt mit Ablaufelementen und sonstigen planbaren Größen von Vorgehensmodellen auseinander gesetzt haben, wollen wir uns an dieser Stelle noch kurz zum verwandten Thema *Geschäftsprozessmodellierung* positionieren. Die Geschäftsprozessmodellierung stellt sich nach Thurner [Thu04] wie folgt dar:

Geschäftsprozessmodellierung

„Die Geschäftsprozessmodellierung erfasst und strukturiert diejenigen Informationen über die durch Software zu unterstützenden Tätigkeiten und Prozesse, die für eine zielgerichtete Entwicklung erforderlich sind."

Sie betrachtet dabei einen einzelnen Geschäftsprozess als Muster für einen Arbeitsablauf in einem System. Sie charakterisiert ihn dann weiterhin durch:

- Funktion und Aktivität
- Kausalität und Reihenfolge
- Geschäftsobjekt, Information und Leistung
- Organisation und Ziel

Im Wesentlichen lassen sich Geschäftsprozesse somit auch durch die Ontologie in Kapitel 2.4.1 erklären. Auch strukturell weisen Geschäftsprozesse und Vorgehensmodelle Ähnlichkeiten auf. Im Kontext dieser Arbeit betrachten wir Geschäftsprozesse als ein gleichberechtigtes Konzept neben einem Vorgehensmodell. Strukturell lassen sich beide Konzepte zu *Prozessen* abstrahieren. Unterschiede finden wir in der Regel nur in der Häufigkeit der Instanziierung. So werden Geschäftsprozesse häufiger und sogar kontinuierlich instanziiert und ausgeführt. Vorgehensmodelle hingegen werden in der Regel für genau ein spezifisches Vorhaben instanziiert. Beispielhaft wollen wir die Bestellung in einem Internet-Versandhaus als Vertreter eines Geschäftsprozesses aufführen. Der Bestellvorgang wird mehrfach pro Tag ausgeführt (inklusive der gekoppelten Geschäftsprozesse). Eine Softwareentwicklung wird hingegen nur einmal aufgesetzt und durchgeführt. Strukturell unterscheiden wir uns also gar nicht gravierend von der Geschäftsprozessmodellierung. Einige Konzepte sind in beiden Prozesstypen zu finden.

3.4. Abhängigkeiten

Ebenfalls essenziell für uns sind Abhängigkeitsstrukturen in Vorgehensmodellen. Abhängigkeiten zwischen den Elementen eines Vorgehensmodells sorgen erst dafür, dass sie einen Beschreibungs- oder Anleitungscharakter bekommen. Abhängigkeiten sorgen dafür, dass Werkzeuge zum Beispiel Erstellungspfade errechnen können etc. Dabei sind die Größen, zwischen denen in Vorgehensmodellen Abhängigkeiten zu definieren sind bekannt und deren Anzahl (der Typen) überschaubar. Die in Beziehung zu setzenden Elemente sind im Wesentlichen bereits der allgemeinen Ontologie für Vorgehensmodelle (zum Beispiel Kapitel 2.4.1 oder Kapitel 5.1.2) entnehmbar. Wir vertreten in dieser Arbeit folgende Einstellung zu diesem Thema:

Elemente und Abhängigkeiten

Die Art und Menge der Elementtypen eines Vorgehensmodells ist bekannt. Essenziell für die Erstellung modularer und flexibler Vorgehensmodelle ist die *Art der Elementtypkombination*!

Wie wollen diese Aussage kurz analysieren, bevor wir uns den konkreten Abhängigkeitstypen und deren unterschiedlichen Realisierungen widmen. Ein Vorgehensmodell

(oder allgemeiner sogar jede Prozessbeschreibung) besteht aus den Elementen der in Kapitel 2.4.1 aufgeführten Submodelle. Die Domäne ist also bekannt und erschlossen. Die wesentlichen Unterschiede in den einzelnen Vorgehensmodellen bestehen in Art, Umfang und Ausgestaltung der Kombination der einzelnen Elementtypen, also den Beziehungstypen.

Beispiel: Das V-Modell XT als produktorientiertes Vorgehensmodell adressiert in der Modellierung von Beziehungstypen vorrangig Beziehungen (Assoziationen) zwischen Produkten, wie wir in Abschnitt 3.2.1 gezeigt haben. Weitere Beziehungstypen werden zwar durch das V-Modell ebenfalls definiert, jedoch sind die Produktabhängigkeiten die einzigen, die expliziert werden.

Beispiel: Als weiteres Beispiel wollen wir noch einmal die Aussagen zur Geschäftsprozessmodellierung des letzten Abschnitts aufgreifen. Nach Thurner [Thu04] besteht ein Geschäftsprozessnetz (Def. 3.3.2) aus einer Menge von Geschäftsprozessen, die über eine Menge von Kanälen $C \subseteq Out \times In$ miteinander verbunden sind. Die Kanäle sind typisiert; Vorgänger-/Nachfolgerrelationen sind bestimmbar. Die Objekte, die in diesem System vorhanden sind, sind durch ein entsprechendes Modell erfassbar und als Entitäten wiederverwendbar. Die Kopplung erfolgt durch Verknüpfung.

Das explizite Modellieren von Beziehungstypen in eigenständigen Elementen eines späteren Vorgehensmodells ist eine Technik, die noch eher unüblich ist. EPF/UMA beispielsweise stützt sich auf einer rein objektorientierten Modellierung ab, die mit der UML 2 zwar auch Assoziationen vollständig als Klassen modelliert – jedoch ist die Interpretation dieses Modells so, dass keine eigenständigen Elemente erzeugt werden, sondern Attribute. EPF/UMA bietet dafür jedoch mit seinen Deskriptoren ein anderes interessantes Konzept: Proxy-Objekte. Die Metamodelldokumentation von EPF/UMA schreibt dazu:

Descriptor

„. . . A Descriptor is an abstract generalization for special Breakdown Elements that references one concrete Content Element. A descriptor provides a representation of a Content Element within breakdown structures. In addition to just referencing Content Elements it allows overriding the Content Elements structural relationships by defining its own sets of associations. Descriptors are the key concept for realizing the separation of processes from method content. A Descriptor can be characterized as a reference object for one particular Content Element, which has its own relationships and properties. [. . .]" [epf07]

Mit solchen Proxies können Verknüpfungsstrukturen abstrahiert werden – ein Konzept, das wir als grundlegende Voraussetzung für konfigurative Vorgehensmodellkonstruktion ansehen (siehe dazu Kapitel 4.2). Wie wir in Abbildung 3.1 sehen können, unterscheidet das V-Modell *strukturelle* und *inhaltliche* Produktabhängigkeiten. Erweitern wir dies auf allgemeine *Prozesselementabhängigkeiten*, so finden wir dieses Konzept in ähnlicher, jedoch anders ausgeprägter Form auch in EPF/UMA wieder, wo wir die Deskriptoren beispielsweise zu den strukturellen Prozesselementabhängigkeiten zuordnen können. Im MSF ist eine strukturelle Abhängigkeit auf der Ebene des Metamodells zwar zu finden, implementiert wird sie jedoch nur indirekt durch die Dokumentation des Prozesses einerseits und die Implementierung in der Werkzeuglandschaft andererseits. Alle Beziehungstypen zwischen den Elementen eines Vorgehensmodells können wir in die Klassen:

- strukturelle und
- inhaltliche Abhängigkeiten

einordnen. Wir betrachten diese beiden Abhängigkeitsklassen in den folgenden Abschnitten und geben einige Beispiele hierfür an. Wir stützen uns dieses Mal exklusiv auf das V-Modell XT als Referenz ab. Mit entsprechenden Überlegungen sind die Aussagen auch für andere Vorgehensmodelle anwendbar.

3.4.1. Strukturelle Abhängigkeiten

Bei den strukturellen Abhängigkeiten müssen wir zunächst noch folgende Unterscheidung treffen, die wir anhand des Beispiels in Abbildung 3.17 diskutieren wollen. Das Szenario zeigt eine Entscheidungspunktinstanz von *Projekt definiert* inklusive der Produkte *Projekthandbuch*, *Projektplan* und *Projektstatusbericht*. Zusätzlich ist für jedes der betrachteten Produkte die verantwortliche Rolle *Projektleiter* eingetragen sowie die Aktivitäten, die die Produkte erzeugen.

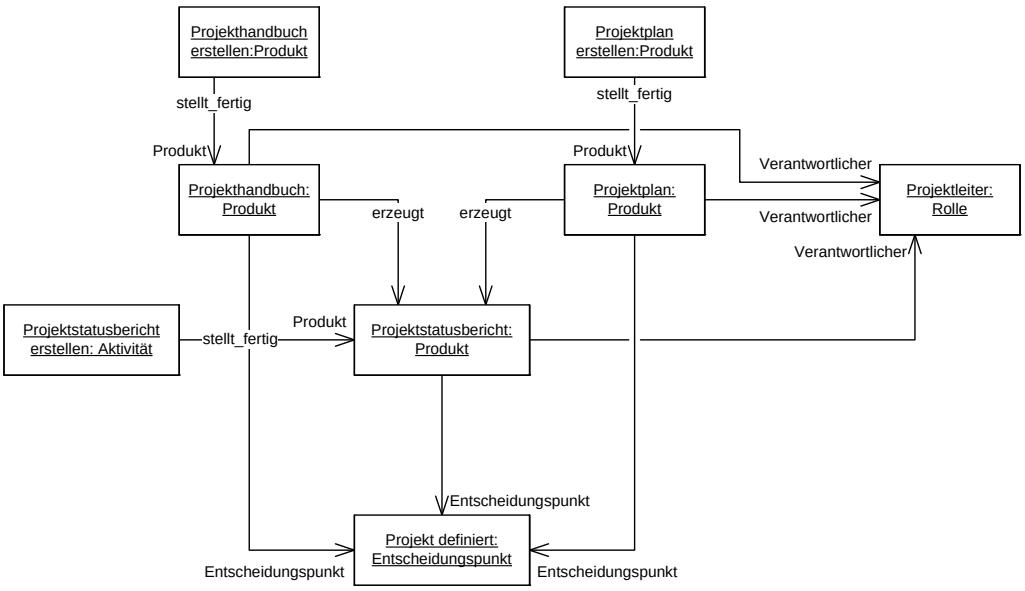


Abbildung 3.17.: V-Modell-Instanz eines Entscheidungspunkts und assoziierter Elemente

Betrachten wir nun die Beziehungen zwischen den gezeigten Elementen, dann sehen wir hier ausschließlich strukturelle Abhängigkeiten. Die gezeigten Beziehungstypen dienen hier dazu, Elemente unterschiedlichen Typs miteinander zu verknüpfen (zum Beispiel Produkt und Aktivität), beziehungsweise ein Abhängigkeitsgeflecht zwischen Elementen eines Typs zu flechten (zwischen Produkten). Abhängigkeitsstrukturen eines Vorgehensmodells auf Basis verschiedener Beziehungstypen, stellen wir als Graphen dar. Abbildung 3.17 zeigt auch, dass es sich dabei um einen *gerichteten* Graphen handelt, in dem die verknüpften Elemente die Knoten und die Verknüpfungen zwischen den Elementen die Kanten sind. In Abbildung 3.18 ist dies in Form eines einfachen Graphen dargestellt. Beide Darstellungen sind stark vereinfacht, da zwischen den einzelnen Elementen in der Regel mehrere Kanten existieren. Vorgehensmodelle bilden ein engmaschiges Netz von Abhängigkeiten. Üblicherweise sind die Graphen, die den Strukturen eines Vorgehensmodells zugrunde liegen *Multigraphen*.

Externalisierung von Beziehungen. Wenn wir die Abbildungen 3.17 und 3.18 näher betrachten, finden wir bereits ein Beispiel, das mehrere Beziehungstypen enthält. Wie vorher ausgeführt, sind dabei die Elemente unspektakulär, insbesondere da wir die elementinternen Strukturen hier nicht weiter berücksichtigen. Vielmehr interessieren uns die Beziehungstypen und dort insbesondere die Frage nach der *Externalisierung*. Beziehungen werden mithilfe der UML 2 auch über Klassen (`Association`) modelliert. Jedoch gibt es verschiedene Interpretations- und Implementierungsformen. Zwei konkrete Implementierungen einer Beziehung zwischen zwei Elementen sind:

Attribut: Ein Attribut des Quellelements enthält eine Referenz auf das Zielelement. Die Beziehung ist somit Teil des Quellelements.

Entität: Zwischen zwei Elementen, die miteinander in Beziehung stehen wird durch ein *drittes* Element eine Verknüpfung hergestellt. Das Verknüpfungselement enthält dabei die Informationen, wer Quelle und wer Ziel der Verknüpfung ist.

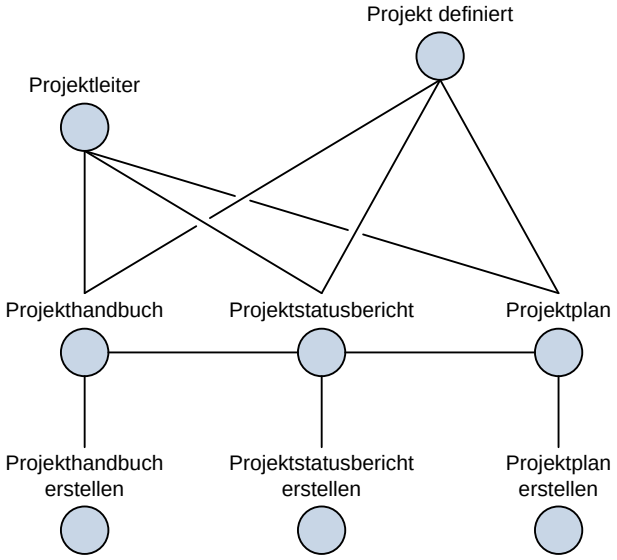


Abbildung 3.18.: V-Modell-Instanz eines Entscheidungspunkts als Graph

Der Vorteil der ersten Methode ist die Einfachheit. Ihr Nachteil ist die mangelnde Flexibilität und die Vermischung von inhaltlichen und strukturellen Fragestellungen. Dies ist die Stärke des zweiten Ansatzes. Er ist sehr flexibel, dynamisch und transparent. Nachteilig ist, dass durch die Verwendung weiterer (Verknüpfungs-)Elemente die Komplexität eines Modells steigt. Die Vorteile dieses zweiten Ansatzes überwiegen jedoch die Nachteile, wie wir am Beispiel von Abbildung 3.17 zeigen können. Lediglich die Kanten auf der Ebene der Produkte, d. h. (*Projekthandbuch*, *Projektstatusbereicht*) und (*Projektplan*, *Projektstatusbericht*) sind durch Produktabhängigkeiten gemäß Produktmodell des V-Modells (vgl. Abbildung 3.1) als eigene Elemente hinterlegt. Alle anderen Beziehungenstypen sind über Attribute realisiert. Die Konsequenzen zeigen sich zum Beispiel bei einer Erweiterung des V-Modells.

Beispiel: Wird beispielsweise ein neuer Entscheidungspunkt eingeführt (zum Beispiel wie in [HKST07]), so kann dieser nicht transparent auf den existierenden Strukturen aufsetzen. Für den neuen Entscheidungspunkt muss zum Beispiel auch eine *Projektfortschrittsentscheidung* herbeigeführt werden, sodass das Produkt *Projektfortschrittsentscheidung* entsprechend angepasst werden muss, obwohl es mit dem Hinzufügen des neuen Entscheidungspunkts nicht unmittelbar in Verbindung steht. Wird jedoch ein zusätzliches Produkt definiert, das auf derselben Grundlage wie beispielsweise der *Projektstatusbericht* erstellt wird, genügt es die erzeugende Produktabhängigkeit mithilfe einer entsprechenden Erweiterung anzupassen. Die beiden Quellprodukte werden davon nicht berührt.

Externalisierte Abhängigkeiten (egal welchen Beziehungstyps) steigern also nicht nur die Flexibilität eines Modells, sondern gleichzeitig auch seine *Stabilität*. Dadurch, dass die Assoziation zwischen zwei (oder mehreren) Elementen separat erfolgt, bietet sich hier einerseits die Möglichkeit der Konfiguration, andererseits bietet sich hier die Möglichkeit zur Definition von *Soll-Bruchstellen*. Die betrachteten Prozesselemente bleiben davon unberührt. Trotzdem ist es nun *nicht* zielführend, alle Beziehungstypen zwischen Prozesselementen zu externalisieren. In den Kapiteln 4 und 5 widmen wir uns der Ermittlung von externalisierbaren Beziehungstypen intensiver bei der Konstruktion unseres Vorgehensmetamodells.

3.4.2. Inhaltliche Abhängigkeiten

Obwohl wir inhaltliche Fragestellungen im Rahmen dieser Arbeit weitgehend ausklammern müssen wir die inhaltlichen Abhängigkeiten zumindest berücksichtigen. Inhaltliche Abhängigkeiten sind Abhängigkeitsstrukturen, die unabhängig von strukturel-

len Eigenschaften eines Modells sind. Üblicherweise trifft dies im Kontext von Vorgehensmodellen auf Dokumentationen und deskriptive Texte zu. Insbesondere mit der Verwendung Hyperlink-basierter Systemen (HTML-Dokumentation) sind oftmals beträchtliche Aufwände investiert worden, um kontextbezogene Informationen zu bündeln. Die Struktur des Prozesses rückt dabei in den Hintergrund. Ein gutes Beispiel, das dieser Charakterisierung entspricht, ist MSF. MSF ist Metamodell-basiert, jedoch ist das Metamodell nicht vollständig umgesetzt, sondern nur soweit es die verwendeten Werkzeuge erfordern. Vollständig definiert und beschrieben ist MSF jedoch auf der Basis seiner Dokumentation. Diese ist jedoch vollständig vom Prozessmodell separiert.

Auf der anderen Seite ist das V-Modell XT selbst beschreibend. Es liegt zu jedem Element eine Beschreibung vor, die im Rahmen eines Exports in eine Dokumentation überführt werden kann[13]. Dokumentation und Struktur des Vorgehensmodells haben hier also eine viel stärkere Bindung. Jedoch können auch in den Beschreibungstexten der einzelnen Elemente Hyperlinks zur Referenzierung anderer (interner und externer) Elemente verwendet werden. Graphentheoretisch unterscheiden sich Prozesslayout und das inhaltliche Abhängigkeitsgeflecht durch die Richtungseigenschaft. Prozesslayouts sind *gerichtete* (Multi-)Graphen, während inhaltliche Abhängigkeitsgeflechte *ungerichtete* (Multi-)Graphen sind.

3.5. Kompositions- und Distributionsmuster

Zum Abschluss dieses Kapitels widmen wir uns den Kompositions- und Distributionseigenschaften und passenden Mustern für die ausgewählten Vorgehensmodelle. Hierunter verstehen wir unter anderem die grundlegenden Strukturen der betrachteten Vorgehensmodelle, die ihren Anpassungsprozessen zugrunde liegen. Darauf aufbauend untersuchen wir noch die Möglichkeiten der Verteilung/Verfügbarmachung von Derivaten der einzelnen Vorgehensmodelle. Insbesondere interessieren uns dabei potenzielle Weiterentwicklungen der Ursprungsversion und die Art des Niederschlags in den angepassten Derivaten.

3.5.1. Erstellungs- und Anpassungsoptionen

Wir betrachten zunächst die Erstellungs- und Anpassungsoptionen. Alle hier betrachteten Möglichkeiten spielen sich im Rahmen der in Kapitel 2.2.2 gezeigten Entwicklungs- und Anpassungsprozesse ab.

Microsoft Solutions Framework. Erstellung und Anpassung des MSF ist wie in Kapitel 2.2.2 bereits erwähnt ein Vorgang, dessen Definition gerade noch im Entstehen ist. Er ist noch nicht formalisiert, jedoch von der Struktur des Vorgehensmodells abhängig. Grundlage dafür liefert das MSF XML-Schema[14], das durch den Process Template Editor interpretiert und umgesetzt wird. Abbildung 3.19 zeigt einen Ausschnitt des Schemas mit dem zentralen Element `Process Template`. Die Erstellung und Bearbeitung/Anpassung des MSF verläuft immer auf Basis dieses Elements[15]. Innerhalb dieses Elements sind die Einzelelemente (bis auf die Reports, die Fähigkeiten des SQL Server 2005 benötigen) editierbar. Jedes der in Abbildung 3.19 aufgeführten Elemente ist anpassbar, wobei Änderungen im Sinne von Anpassungen wiederum nur bedingt sinnvoll sind. Wir betrachten ausgewählte Elemente:

SharePointPortalTemplate: Das SharePoint Portal Template dient als Vorlage für das Erzeugen einer Team Webseite für ein TFS-Projekt. Das Template kann vergleichsweise einfach mit einem entsprechenden Werkzeug (Microsoft SharePoint Designer) angepasst werden, muss dann aber unabhängig vom restlichen Process Template auf dem Server installiert werden.

13 Der Aufbau dieser Dokumentation ist ebenfalls im Rahmen des V-Modell XT Metamodells beschrieben und anpassbar.

14 Hierbei handelt es sich um eine technische Interpretation des Metamodells für die Werkzeuge.

15 Das Process Template ist als einfache XML-Datei realisiert, die ein MSF-artiges Vorgehensmodell als Verzeichnisstruktur ablegt und referenziert.

SharePointPortal: Das Portal selbst kann an zwei Punkten angepasst werden. Einmal während des Designs des Process Templates. Hier können dann Dokumentbibliotheken mitsamt vorgefertigter Dokumente angelegt werden. Zweitens zur Laufzeit eines Projekts im Rahmen der Möglichkeiten von SharePoint.

Workproduct: Produkte können im Rahmen der Anpassung des SharePoint Portals angepasst werden.

WorkItemType: Work Items können ebenfalls zur Designzeit des Process Templates als auch noch eingeschränkt zur Laufzeit eines Projekts angepasst werden.

Der Process Template Manager des Team Foundation Server unterstützt auch die Anpassung eines bereits instanziierten Process Templates, also das Exportieren und Reimportieren des *Living Process*. Dies ist aber nur mit Einschränkungen möglich, die sich unter anderem daraus ergeben, dass zum Beispiel instanziierte Work Item Typen in korrespondierende Datenbankstrukturen übersetzt wurden und nicht mehr ohne weiteres manipuliert werden dürfen.

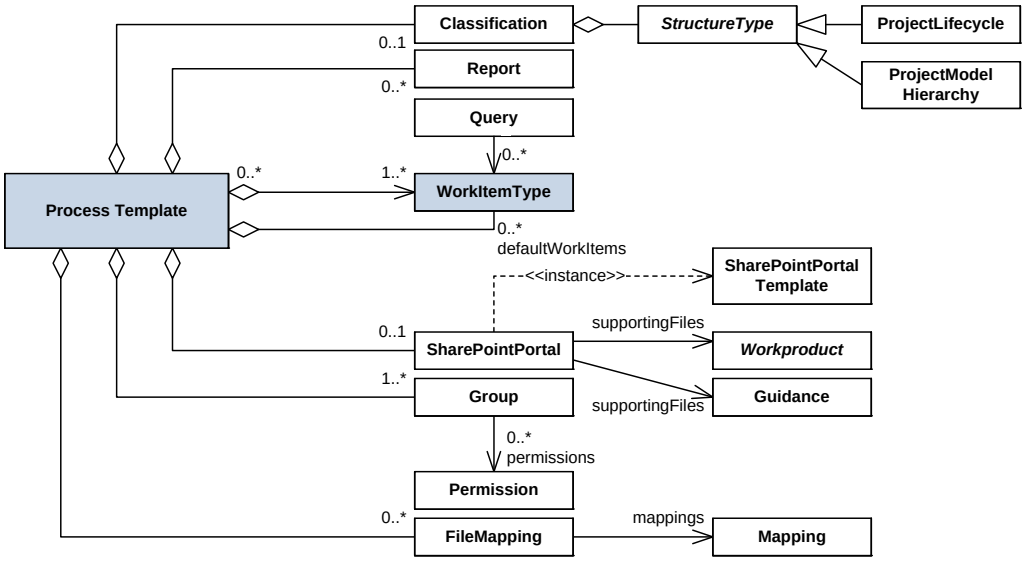


Abbildung 3.19.: MSF Sicht: Anpassung auf der Grundlage des Werkzeugschemas

Die weit reichenste und unkomplizierteste Möglichkeit, ein Process Template an die Bedürfnisse (Organisation oder Projekt) anzupassen, ist mit Work Items zu arbeiten. Beispiele für spezialisierte Work Items sind neben den Standardtypen des MSF Scrumbasierte Items für *Backlogs* oder angepasste Work Items für das V-Modell XT. Work Items eignen sich deshalb gut als Anpassungsgegenstände, da sie *self-contained* sind (vgl. Abbildung 3.3). Ein Work Item definiert seine Datenstruktur, den korrespondierenden Zustandsautomaten und Informationen für die Generierung einer Benutzerschnittstelle[16]. Diese Informationen sind je Work Item Typ in einer eigenen XML-Datei untergebracht, die entsprechend verteilt werden kann.

V-Modell XT. Das V-Modell XT definiert als Bearbeitungseinheit den *Vorgehensbaustein*. Wie Abbildung 3.20 zeigt, ist der Vorgehensbaustein sehr tief in das V-Modell integriert. Die Abbildung 3.21 verdeutlicht das noch weiter, indem sie gleichzeitig die wesentlichen Assoziationen zur Einbindung des Vorgehensbausteins zeigt.

Ein Vorgehensbaustein enthält selbst wiederum Kindelemente wie Produkte oder Aktivitäten (Abbildung 3.20). Wird also im Rahmen einer Erstellung oder Anpassung ein

16 Die integrierten Informationen für die GUI erfüllen dabei zwei wesentliche Zwecke: 1) ermöglichen sie Werkzeugen Work Item-Instanzen zu verarbeiten und dynamisch mit allen relevanten Eingabefeldern zu versehen. 2) ermöglichen sie die Steuerung der GUI im Sinne von Wertebereichsbeschränkungen, zum Beispiel für Auswahllisten.

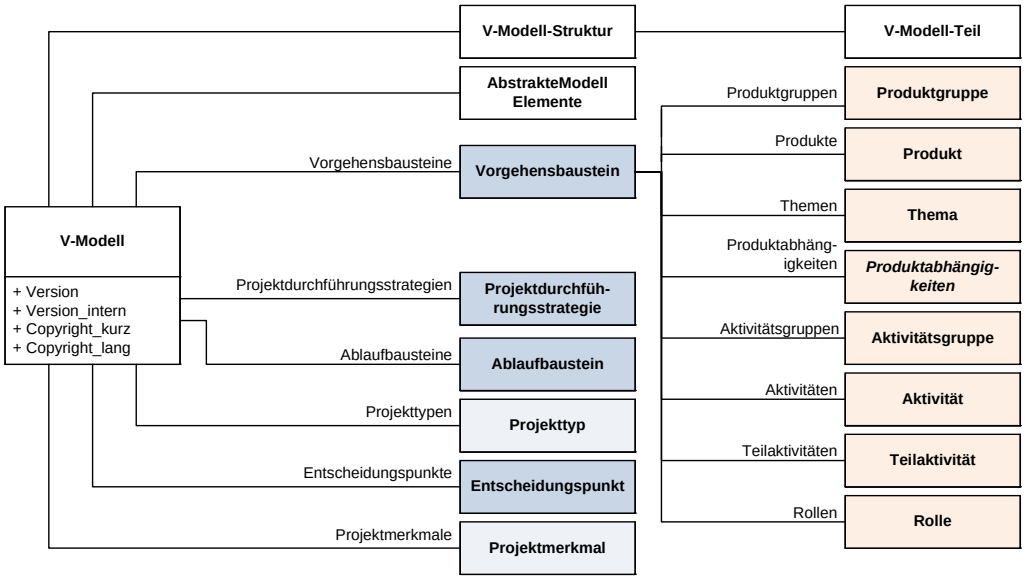


Abbildung 3.20.: V-Modell XT Sicht: Elementhierarchie auf Grundlage des XML-Schemas

neuer Vorgehensbaustein in das V-Modell integriert, müssen diese Elemente entsprechend mit erzeugt werden. Auf derselben Ebene wie der Vorgehensbaustein sind noch Projektdurchführungsstrategien etc. zu finden. Wird ein neuer Vorgehensbaustein eingeführt, so muss dieser den anderen Elementen bekannt gemacht werden. Ein Beispiel findet sich in Abbildung 3.21. Wird ein neuer Vorgehensbaustein eingeführt, muss er einem Projekttyp entweder als verpflichtend oder als optional zugeordnet werden. Enthält ein Vorgehensbaustein Produkte, die an einem spezifischen Entscheidungspunkt vorgelegt werden müssen, so müssen über die reine Integration des Vorgehensbausteins hinaus auch die Assoziationen zwischen Produkt und Entscheidungspunkt korrekt gesetzt werden (vgl. Abbildung 3.13). Vorgehensbausteine sind für die inhaltliche Ausgestaltung zuständig, sodass sie neue oder angepasste Inhalte aufnehmen können. Abläufe sind über Projektdurchführungsstrategien, Entscheidungspunkte und Ablaufentscheidungspunkte zu realisieren. Die Erstellung erfolgt analog zum Vorgehensbaustein; auch die Einbindung in das V-Modell erfolgt analog.

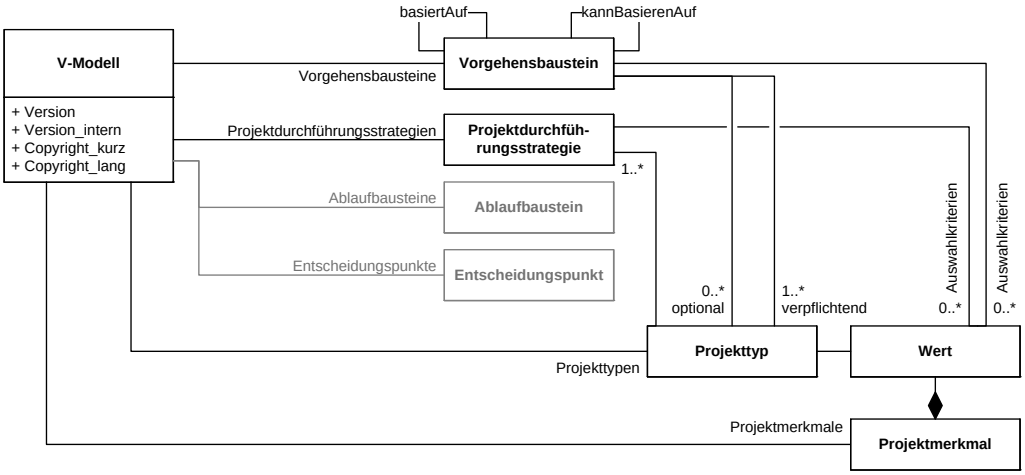


Abbildung 3.21.: V-Modell XT Sicht: Elementhierarchie mit Assoziationen

Wird ein V-Modell XT Derivat erstellt, geschieht das in erster Linie auf der grob granularen Ebene von Vorgehensbausteinen und Projektdurchführungsstrategien sowie de-

ren nachgelagerten Elementen. Ein fein granulareres Anpassungskonzept auf der Ebene des V-Modells ist zurzeit noch nicht verfügbar. Dieses Anpassungsverfahren zieht aber noch weitere Konsequenzen nach sich, die insbesondere bei der Verteilung der Anpassungen und bei der Aktualisierung der Standard-, beziehungsweise Referenzanteile auftreten. Wird eine komplexere Anpassung des V-Modells vorgenommen, wie wir sie in [KHTS07, HKST07] durchgeführt haben, so betrifft die Anpassung in der Regel mehrere V-Modell-Komponenten. Eine Anpassung ist somit genauso hoch integriert wie das Standardmodell. Ein Vorgehensbaustein aus einer Anpassung kann nicht einfach verteilt werden, da er zum Beispiel die ablaufbezogenen Elemente (Entscheidungspunkte, Projektdurchführungsstrategien etc.) gar nicht enthält. Auch ist die Zuordnung zu Projekttypen nicht Bestandteil des Vorgehensbausteins. Wird das Standardmodell im Rahmen einer Aktualisierung geändert, muss im schlimmsten Fall die Anpassung erneut vorgenommen werden, da eine konsistenzerhaltende Integration der angepassten und der Referenzdatei aufgrund des fehlenden Pflege- und Updateprozesses nicht vorgesehen ist.

EPF/UMA. Mit dem V-Modell XT wurde bereits ein Komponentenbegriff eingeführt – der des Vorgehensbausteins. Auch EPF/UMA kennt ein Modulkonzept, das auf verschiedenen Granularitätsstufen die Anpassung und Ausgestaltung eines Vorgehensmodells unterstützt.

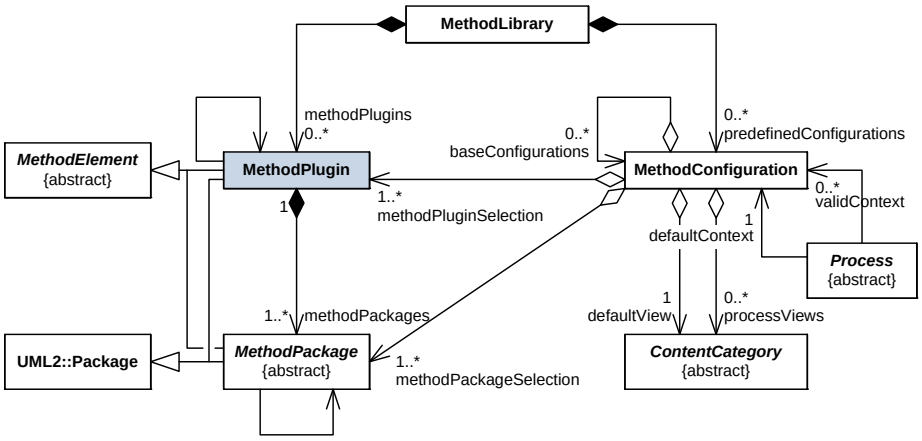


Abbildung 3.22.: EPF/UMA Sicht: Method Plugins als Komponentenmodell

Abbildung 3.22 zeigt das UMA-Konzept des *Method Plugins*, das als Komponentenkonzept hinter EPF steht. Die Metamodelldokumentation beschreibt ein Method Plugin wie folgt:

Method Plugin

„A Method Plugin is a Method Element that represents a physical container for Method Packages. It defines a granularity level for the modularization and organization of method content and processes. A Method Plugin can extend many other Method Plugins and it can be extended by many Method Plugins. It can also be used stand-alone, i.e. with no Extension relationship to other plug-ins. Method Plugin conceptually represents a unit for configuration, modularization, extension, packaging, and deployment of method content and processes. A Process Engineer shall design his Plugins and allocate his content to these Plugins with requirements for extensibility, modularity, reuse, and maintainability in mind." [epf07]

Somit nimmt das Konzept des Method Plugins dieselben Eigenschaften wie der V-Modell-Vorgehensbaustein für sich in Anspruch. Anders als dieser sind Method Plugins jedoch auch dafür ausgelegt, als physisch eigenständige Einheiten zu erscheinen. Von allen bislang betrachteten Konzepten stellen die Method Plugins das mächtigste dar.

Method Plugins können beliebige Inhalte tragen (Grundlage ist die Klasse `MethodElement`) und Anpassungen im Rahmen einer Variabilität durchführen. Die hohe Komplexität des Metamodells und die hohen Anforderungen an dessen Anpassungen sowie die schwere Nachverfolgbarkeit externer Abhängigkeiten stehen dem als nachteilig gegenüber. In [BULL07] ist ein Bericht über die entsprechenden Arbeiten im Rahmen der Erfassung eines organisationsspezifischen Prozesses zu finden.

3.5.2. Verteilung von Derivaten

Da wir in diesem Kapitel auch Erstellungs- und Anpassungsmechanismen ausgewählter Vorgehensmetamodelle betrachtet haben, müssen wir uns abschließend noch einmal den Verteilungs- und Bereitstellungsprozessen widmen. In Kapitel 2.2.2 haben wir die entsprechenden Anpassungsprozesse bereits vorgestellt. Wir greifen sie wieder auf und untersuchen nun die *Verteilungsgegenstände* und gleichen diese mit den hier identifizierten Metamodellelementen ab. Sofern das betreffende Vorgehensmodell ein Modul- oder Komponentenmodell definiert, betrachten wir dieses hier ebenso.

MSF. Für MSF gibt es prinzipiell zwei Verteilungsgegenstände: Einerseits sind das die Work Items (vgl. Abschnitt 3.2.2) und andererseits ein vollständiges Process Template (siehe Abbildung 3.19). Die Work Items eignen sich zur Bereitstellung kompletter Teilprozesse, wie zum Beispiel Aufgaben- oder Risikomanagement. Jedoch sind sie ohne das Process Template als Container nicht instanziierbar (und somit nicht nutzbar). Auch Queries oder Reports sind zwar über Work Items definiert, jedoch sind auch sie Bestandteil eines Process Templates. Process Templates hingegen sind als Verteilungseinheiten umfassend. Sie beinhalten vollständige Vorgehensmodelle (inkl. von Vorlagen, Work Items, Prozessdokumentation etc.). Aufgrund der hohen Integration und der Masse an Inhalten, die potenziell in einem solchen Process Template enthalten sind, ist der Erstellungsprozess dafür sehr anspruchsvoll (vgl. auch Anhang A.2). Aus Sicht der Verteilung gestaltet sich die Weitergabe zunächst einfach. Work Items sind einfach weiter zugeben, indem die XML-Dateien in denen sie definiert sind, verteilt werden. Je Work Item Typ existiert genau eine Datei. Process Templates liegen während der Bearbeitung als Verzeichnisstruktur vor. Für die Verteilung kann das gesamte Verzeichnis weitergegeben werden.

Technisch gestaltet sich das aufwändiger, da die Kombination Process Template, Work Item und Team Foundation Server gewisse Begrenzungen aufweist. Einmal installierte Prozessanteile sind in der Regel bestimmend für alle folgenden. Namen und Namenskombinationen müssen eineindeutig sein, was Aktualisierungsprozesse stark erschwert. Process Templates und Work Items sind sprachgebunden – dies hat zur Folge, dass ggf. eine Vielzahl unterschiedlicher Versions- und Sprachstände verwaltet und (technisch) organisiert werden muss. Aufgrund der hohen Komplexität nicht nur der Prozessengine, sondern auch der gesamten Prozessinfrastruktur ist die Anpassung äußerst aufwändig.

EPF/UMA. Für UMA-basierte Prozesse stellt das im letzten Abschnitt beschriebene Method Plugin den Gegenstand der Verteilung dar. Wie wir in der Strukturanalyse zu EPF/UMA bereits festgestellt haben, gibt es auch ein Metamodellelement `DeliveryProcess`. Dabei handelt es sich um eine problembezogene Zusammenstellung zusammengehöriger, aktivitätsbezogener Elemente. Die Metamodelldokumentation beschreibt einen Deliver Process wie folgt:

Delivery Processes

„A Delivery Processes is a special Process describing a complete and integrated approach for performing a specific project type. It describes a complete project lifecycle end-to-end and shall be used as a reference for running projects with similar characteristics as defined for the process." [epf07]

Eine spezielle Verteilungsform basiert auf dem `ProcessPackage`, das eine Spezialisierung des Metamodellelements `MethodPackage` ist (siehe Abbildung 3.22). Ein Process Package enthält allgemein eine Menge von *Prozesselementen* (Process Elements), die Verallgemeinerungen aller beschreibbaren Elemente eines Prozesses sind. Eine spezielle Form des Process Package, das weiter gehende Anforderungen an Kapselung umsetzen soll, stellen *Process Components* dar. Auch sie fassen, die Eigenschaften von `ProcessPackage` erbend, mehrere Prozesselemente zusammen. Bezug nehmend auf die oben stehende Metamodelldokumentation *kann* (nicht muss) ein Delivery Process auch aus einer Menge abgestimmter Prozesskomponenten bestehen. Konzeptuell ist dieser Ansatz somit sehr mächtig.

Technisch können die Verteilungseinheiten als Verzeichinsstrukturen bearbeitet und verteilt werden. Die Integration erfolgt in eine Werkzeugumgebung, die auf dem Eclipse Process Framework (EPF) aufbaut. Diese übernimmt die Integration in weiten Teilen. Konfiguration und Publikation sind als mächtige Funktionen in das Werkzeug integriert – die Umsetzung eines Prozesses ist aufgrund der hohen Komplexität jedoch nicht intuitiv [BULL07]. EPF unterliegt einer steten Weiterentwicklung, sodass hier kontinuierlich mit Aktualisierungen sowohl auf der technischen als auch auf der inhaltlichen Seite zu rechnen ist.

V-Modell XT. Bereits in den Abbildungen 3.20 und 3.21 werden einseits die Struktur des V-Modells andererseits ein Teil der projekttypbezogenen Assoziationen gezeigt. Für die Verteilung von Anpassungen des V-Modells gibt es zurzeit nur die Option, immer nur ein *vollständiges* V-Modell-Derivat zu verteilen. Eine partielle Verteilung einzelner Vorgehensbausteine, Projekttypen oder von Projektdurchführungsstrategien ist nicht möglich. Da das Anpassungskonzept erst in der Entwicklung ist (vgl. Anhang A.3) und sich auch erst mit dieser Frage auseinandersetzen muss, ist hier erst mittelfristig mit einem umfassenden Konzept zu rechnen.

Technisch erfolgt die Weitergabe des V-Modell XT und eventueller Anpassungen in Form einer Verzeichnisstruktur. Sie fasst im Wesentlichen das V-Modell, Mustertexte und Bilder sowie zusätzliches Material zusammen.

3.6. Anforderungen für modulare Verteilungseinheiten

Als einziges der vertiefend betrachteten Vorgehensmodelle weist das V-Modell XT Eigenschaften eines übergreifenden Entwicklungsstandards auf. Es definiert ein (inhaltliches) Modulkonzept, das trotz seiner hohen Integrationsdichte ein hohes Maß an Flexibilität für problembezogene Anpassungen in Form eines projektspezifischen Tailorings bietet. Dennoch ist gerade im letzten Abschnitt deutlich geworden, dass es einen Bedarf für ein tragfähiges und flexibles Anpassungs- und Verteilungskonzept auf der projektübergreifenden Ebene gibt. Mögliche Techniken werden durch MSF oder EPF/UMA vorgeschlagen und spezifisch umgesetzt. In diesem Abschnitt abstrahieren wir ausgehend vom V-Modell XT hin zu allgemeinen Entwicklungsstandards und formulieren auf der Grundlage des erkannten Bedarfs Anforderungen an ein Anpassungs- und Verteilungskonzept. Die Anforderungen beziehen sich dabei allgemein auf Strukturen, sodass wir an dieser Stelle keine explizite Unterscheidung zwischen Ergebnis- und Planungselementen treffen. Auf Basis der Anforderungen beschreiben wir Anwendungsfälle, in deren Kontext die geforderten Konzepte greifen. Die Anforderungen und Anwendungsfälle bilden im weiteren Verlauf der Arbeit die Grundlagen für die Erstellung der Basis- und Metamodelle.

3.6.1. Strukturelle Anforderungen

Für den Kontext dieser Arbeit sehen wir modulare Architekturen als Lösungsweg für die Notwendigkeiten bei Anpassung und Verteilung im Kontext eines Entwicklungsstandards. Wir orientieren uns daher an üblichen Anforderungen aus dem Bereich der

Software Architektur und überführen sie auf Entwicklungsstandards und Vorgehensmodelle. Diese Anforderungen (sofern umgesetzt) ermöglichen den Aufbau eines hoch flexiblen Systems, das es Prozessingenieuren gestattet, Entwicklungsstandards und Vorgehensmodelle methodisch zu erstellen und zu pflegen. Die darauf ausgerichteten Anwendungsfälle, die wir mit dieser Arbeit unterstützen, beschreiben wir im folgenden Abschnitt 3.6.2. Auf der Grundlagenseite legen wir folgende Anforderungen an Strukturen fest:

Anforderung 3.1 (Metamodell für die Strukturen)**:**
Entwicklungsstandards müssen über ein definiertes Metamodell verfügen, über das sie gültige Elementtypen, Strukturen und Operationen zur Erzeugung von Instanzen definieren.

Die Forderung nach einem Metamodell haben wir bereits indirekt in Kapitel 2 und zu Beginn dieses Kapitels formuliert, indem wir uns schwerpunktmäßig mit Metamodellbasierten Ansätzen beschäftigt haben. Ein Metamodell ist für uns zwingend erforderlich, um in einem definierten Rahmen arbeiten zu können. Wir legen durch das Metamodell nicht nur Strukturen etc. fest, sondern geben damit einem Prozessingenieur die Werkzeuge an die Hand, die er braucht, um neue Vorgehensmodelle, beziehungsweise Anteile davon, zu erstellen. Das Metamodell versetzt uns darüber hinaus auch in die Lage, die *Gültigkeit* neuer oder angepasster Teile zu prüfen und durchzusetzen. Die Möglichkeit über den Konstrukten des Metamodells *Konsistenzbedingungen* zu formulieren ist wesentlich für Prüfung und Qualitätssicherung. Außerdem dient uns ein Metamodell als Grundlage zur Formulierung eines sinnvollen Werkzeugkonzepts [KK07, KDA07], das erforderlich ist, einen Entwicklungsstandard oder zumindest ein dem Standard genügendes Vorgehensmodell mit Akzeptanz einzuführen.

Die Anforderung, ein Metamodell umzusetzen ist noch eher allgemein und wenig spezifisch für den Kontext Modellierung von Vorgehensmodellen. Aus den Analysearbeiten der vorangegangenen Abschnitte können wir diese allgemeine Anforderung jedoch weiter schärfen.

Anforderung 3.2 (Metamodell für die Verteilung)**:**
Das Metamodell eines Entwicklungsstandards/eines Vorgehensmodells muss Anforderungen hinsichtlich der Verteilbarkeit von Instanzen des Standards berücksichtigen.

Wie wir im vorangegangenen Abschnitt 3.5.2 sehen konnten, definieren die beispielhaft betrachteten Vorgehensmodelle alle eine (eigene) Verteilungsmethode. Diese reichen von so mächtigen Ansätzen wie EPF/UMA bis hin zu monolithischen Formen wie beim V-Modell XT. Technische Fragestellungen haben wir dabei auch berücksichtigt und festgestellt: *Das Metamodell muss eine adäquate physische Repräsentation besitzen*. Beziehen wir uns beispielhaft auf das V-Modell XT, so finden wir keine Entsprechungen der logischen Struktur des Modells (XML-Baum) auf der physischen Ebene. Vorgehensbausteine als Komponenten sind nicht separat gefertigt. Projekttypen, also gültige, konsistente Teilmengen des V-Modells, sind nicht separat verteil- und integrierbar. Abläufe in Form von Projektdurchführungsstrategien setzen sich aus mehreren Metamodellkonzepten zusammen, womit sie ebenfalls nicht losgelöst betrachtet oder verteilt werden können. Analoges finden wir auch auf der Ebene MSF, wo die eigentlich *prozesstragenden* Anteile – die Work Items – nicht für sich allein existieren, sondern den umfassenden Rahmen eines Process Template benötigen und dort für erhebliches, systembedingtes Konfliktpotenzial sorgen.

Die Anforderung 3.2 macht daher ein *Modulkonzept* erforderlich, das einerseits das hohe, konsistenzgesicherte, inhaltliche Niveau des V-Modells aufweist, gleichzeitig aber auch prozesstragende Einzelelemente zulässt und das Ganze mit der (physischen) Flexibilität von EPF/UMA kombiniert. Diese Forderung induziert weitere Anforderungen:

Anforderung 3.3 (Soll-Bruchstellen)**:**
Ein Entwicklungsstandard/ein Vorgehensmodell muss *Soll-Bruchstellen* definieren, die eine parzielle Entwicklung, Verteilung und Aktualisierung von Prozessanteilen ermöglichen. Die Soll-Bruchstellen dürfen dabei nicht beschränkt sein, sondern müssen alle prozessbezogenen Teile berücksichtigen.

Diese Anforderung adressiert im Wesentlichen zwei Punkte. Einmal die Entwicklung neuer Inhalte auf der Basis des gegebenen Metamodells. Zweitens die Fähigkeiten zur transparenten Anpassung und Erweiterung des Metamodells. Den zweiten Aspekt behandeln wir im Rahmen von Anforderung 3.6 noch einmal separat. Mit der Definition von Soll-Bruchstellen nehmen wir einen struturellen Schnitt vor und definieren Verteilungseinheiten. Wir orientieren uns hier an den Konzepten des V-Modells, das wesentlich restriktivere inhaltliche Zusammenhänge aufweist als EPF/UMA. Dort wird der Begriff der *Prozesskomponente* eingeführt; aufgrund der Positionierung im UMA-Metamodell kann eine EPF/UMA-Prozesskomponente aber alles enthalten, was im Metamodell definierbar ist. Obwohl EPF/UMA eine konzeptuelle Trennung zwischen Method- und Process Content beschreibt (Kapitel 2.1.3, findet sich diese Trennung auf der Ebene von Method Plugins oder Prozesskomponenten nicht wieder. Das V-Modell hingegen trennt hier logisch sehr sauber in prozessstrukturtragende (Vorgehensbausteine) und planungsorientierte (Ablaufbausteine) Anteile. Ein derartiges Schnittmuster zeigt potenzielle Soll-Bruchstellen auf, durch die Vorgehensbausteine und Ablaufbausteine separierbar, verteilbar und im späteren Verlauf wieder miteinander kombinierbar sind.

Sofern ein adäquates Schnittmuster definiert ist, ist auch eine Umkehroperation erforderlich, die aus den einzelnen Modulen wieder ein konsistentes Ganzes erzeugt.

Anforderung 3.4 (Kompositionsmuster)**:**
Ein Entwicklungsstandard/ein Vorgehensmodell muss Operationen anbieten, die verschiedene Modultypen einheitlich, transparent und überprüfbar zu einem Vorgehensmodell zusammenfasst beziehungsweise in ein gegebenes integriert.

Hinter der Anforderung 3.4 verbirgt sich die Umsetzung und Nutzbarmachung der nach dem Schnittmuster von Anforderung 3.3 separierten Module. Es muss im System des Entwicklungsstandards ein definiertes Schnittstellenkonzept geben (inhärent bereits durch die Forderung nach Modulen gegeben), das die definierte Kopplung verschiedener Modultypen gestattet. Die Kopplung muss in mehreren Szenarios definiert sein. So müssen Operationen zur Erstellung des Entwicklungsstandards ebenso abgedeckt sein, wie seine Pflege und Aktualisierung durch neue Module und ggf. auch gänzlich neue Modultypen. Weiterhin muss unter Berücksichtigung der Anforderungen hinsichtlich verschiedener Produkt- und Produktlinieneigenschaften (vgl. Kapitel 2.3.2) die Anpassung, beziehungsweise die *Konfiguration*, von Instanzen des Entwicklungsstandards abgedeckt werden.

Die Forderung nach der Einheitlichkeit und Transparenz des Kompositionsmusters ergibt sich hierbei aus den Anforderungen, die sich durch Pflege und Weiterentwicklung ergeben. Initial erstellte Inhalte des Entwicklungsstandards, die in abgeleiteten Vorgehensmodellen enthalten sind und dort ggf. erweitert oder angepasst werden, müssen auch wieder in den Entwicklungsstandard zurückfließen können. Dies ermöglicht erst eine definierte, Feedback-orientierte Weiterentwicklung des Entwicklungsstandards. Die in Anforderung 3.3 klar definierten Inhalte sorgen dafür, dass die Schnittstellen im System vergeleichsweise einfach gestaltet werden können. Die Einfachheit der Schnittstellen wiederum gestattet es, ein einheitliches Verfahren zur Kopplung zu entwickeln. Dieses ist erforderlich, da nach der Zusammenführung der beteiligten Module in der Regel nur noch inhaltliche Fragen relevant sind.

Die Anwendung eines Kompositionsverfahrens sowohl auf der physischen als auch auf der logischen Ebene schafft hier Transparenz. Dafür erforderlich ist jedoch auch ein einheitliches Verständnis von der Art der Operationen zur Verknüpfung/Komposition von Inhalten und Strukturen.

Anforderung 3.5 (Kontrollierbares Abhängigkeitssystem)**:**
Ein Entwicklungsstandard/ein Vorgehensmodell muss in seinem Metamodell ein dediziertes Abhängigkeitssystem beschreiben, in dem sämtliche Beziehungstypen instanziierbar und prüfbar hinterlegt sind.

In den Analysen, die wir in diesem Kapitel durchgeführt haben, haben wir festgestellt, dass Abhängigkeiten zwischen den Elementen eines beliebigen Prozesses die eigent-

lichen Herausforderung bei der Modellierung darstellen. Bei MSF beispielsweise finden wir ein System, das relativ lose gekoppelt ist. In Konsequenz ist dieses Vorgehensmodell sehr flexibel, jedoch auch anfällig im Bereich der Konsistenz. EPF/UMA und V-Modell XT verfügen beide über ein ausgeprägtes System von Abhängigkeitstypen, wobei EPF/UMA hier eher generisch ausgerichtet ist, während das V-Modell insbesondere über die strukturellen Abhängigkeiten Ergebnisse und Ergebniserzeugung geradezu spezifiziert. Beispielhaft seien hier die Produkt-/Themen-Zuordnungen zu nennen, die die Anforderungen an ein zu erstellendes Produkt beschreiben. Weiterhin sind beispielsweise über die erzeugenden Produktabhängigkeiten die Erstellungspfade einzelner Produkte ermittelbar.

Ein kontrollierbares Abhängigkeitssystem liegt dann vor, wenn signifikante Abhängigkeitstypen identifiziert und so modelliert sind, dass automatische Verfahren dazu entwickelt werden können, um Aussagen zum Beispiel über die Konformität eines konkreten Vorgehensmodells zu treffen. Weiterhin muss ein kontrollierbares Abhängigkeitssystem effektiv die Konfiguration unterstützen, die im Kontext Ableitung und spezifische Ausgestaltung eines Vorgehensmodells auf der Basis eines Entwicklungsstandards stattfindet. Das Abhängigkeitssystem muss sich dabei in das in Anforderung 3.4 geforderte Kompositionsmuster integrieren. Vielmehr ist das Abhängigkeitssystem im Kontext einer konfigurativen Komposition sogar eines der tragenden Elemente.

Anforderung 3.6 (Erweiterbarkeit und Unterstützung von Variabilität)**:**
Ein Entwicklungsstandard/Vorgehensmodell muss bereits auf der Ebene des Metamodells Optionen zur Apassung, Erweiterung, Konfiguration und Variabilität vorsehen.

Diese Anforderung fasst in Konsequenz bereits einige der zuvor aufgeführten zusammen. Denn zur Umsetzung von Entwicklungsstandards ist die Notwendigkeit zur Anpassung bereits von Anfang an gegeben. Beispielhaft sei hier das V-Modell XT genannt, in dem die erste Operation beim Aufsetzen eines Projekts darin besteht, das gegebene (organisationsspezifische) V-Modell projektspezifisch anzupassen. Diese Notwendigkeit setzt aber nicht erst auf der Ebene konkreter Projekte ein, sonder auch auf der Ebene von Organisationen, die Unschärfen und Generika aus dem Entwicklungsstandard entfernen und dafür Spezifika integrieren müssen. Das V-Modell der Bundeswehr ist ein Beispiel dafür.

Einr organisationsspezifische Anpassung eines Vorgehensmodells, beziehungsweise eine Ableitung aus einem Entwicklungsstandard erfordert:

Ein Metamodell, um die Sprache des Vorgehensmodells zu verstehen und Wissen über die zugelassenen Elementtypen und Beziehungen zu erhalten.

Wissen über Fertigungseinheiten, um zu wissen, wie man ein gegebenes Vorgehensmodell anpasst; wo die Punkte für die Anpassung zu finden und was die „lieferbaren" Fertigungseinheiten sind.

Wissen über Kompositionmuster und Abhängigkeiten, um die Integration der neu erstellten oder angepassten Inhalte zu meistern und wieder ein konsistentes Vorgehensmodell als Ergebnis zu erhalten.

3.6.2. Unterstützte Anwendungsfälle

Die im letzten Abschnitt besprochenen Anforderungen motivieren und bedingen sich aus den Anwendungsfällen, die für einen Prozessingenieur im Kontext der organisationsspezifischen Anpassung oder der Pflege eines Vorgehensmodells/eines Entwicklungsstandards auftreten. In Abbildung 3.23 haben wir die betreffenden Anwendungsfälle vom Standpunkt dieser Arbeit aufgeführt. Wir diskutieren sie im Folgenden. Mit Sicherheit existieren noch weitere Anwendungsfälle, die hier relevant sein können. Im Kapitel 6 finden wir weitere Beispiele. Für diese Arbeit fokussieren wir jedoch explizit den Prozessingenieur und Aufgaben im Zusammenhang mit Entwicklungsstandards, die ihn unmittelbar betreffen. Wie der Abbildung 3.23 zu entnehmen ist, gibt es auch noch weitere Rolle, die wir betrachten können. Wir schließen sie aber zunächst aus.

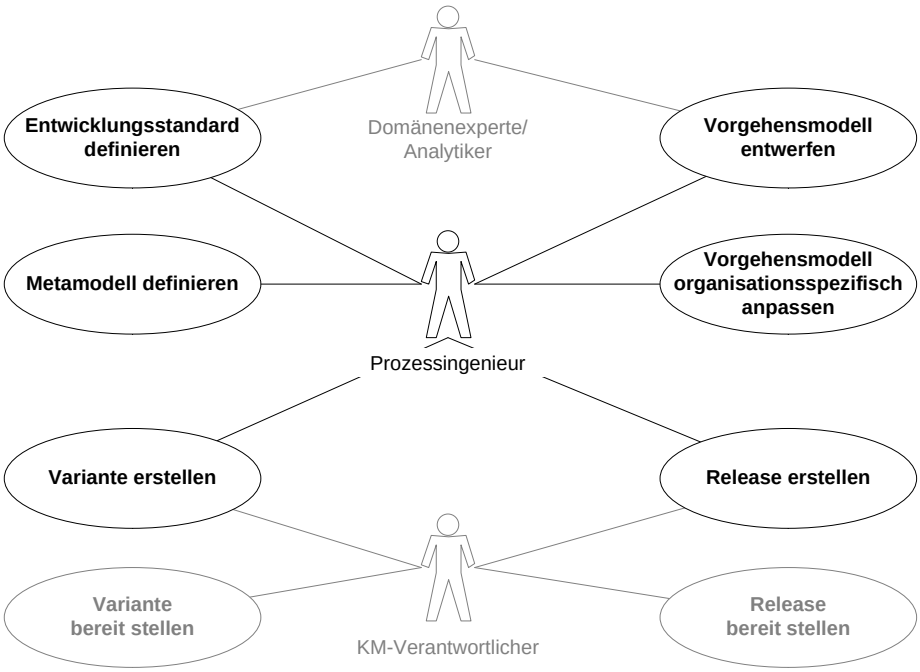


Abbildung 3.23.: Adressierte Anwendungsfälle

Die folgenden Anwendungsfälle stehen für uns im unmittelbaren Zusammenhang mit der methodischen Entwicklung und Pflege von Entwicklungsstandards und Vorgehensmodellen. Bezugspunkte sind hierbei für uns einerseits Fragestellungen aus der Produktlinienentwicklung (Kapitel 2.3.2), andererseits greifen wir auf Erfahrungswerte zurück, die wir im Rahmen verschiedenster Prozessanpassungen (vgl. Anhang A) gemacht haben. Die Anwendungsfälle stehen weiterhin in unmittelbaren Bezug zu den Referenzprozessen, die wir im weiteren Verlauf für die Entwicklung und Pflege von Entwicklungsstandards angeben (Kapitel 5.2).

1. Entwicklungsstandard definieren. Die Definition eines Entwicklungsstandards umfasst einerseits die genaue Analyse der betreffenden Domäne sowie die Erstellung und Bereitstellung eines Referenzmodells. Dieser Anwendungsfall hat Berührungspunkte zu allen anderen Anwendungsfällen[17].

Akteur: Prozessingenieur (ggf. Domänenexperte)

Auslöser: Notendigkeit einer Standardisierung verschiedener Entwicklungsprozesse.

Ergebnis: Ein Entwicklungsstandard mit mindestens einer Ausprägung, dem Referenzmodell.

Vorbedingungen: Es liegen Anforderungen und Kontextwissen aus der Domäne vor.

Nachbedingungen: Der Entwicklungsstandard umfasst dabei alle notwendigen Elemente, um die gerade aufgeführten Anforderungen zu befriedigen, also zum Beispiel ein Metamodell sowie ein Verteilungs- und Anpassungskonzept.

2. Metamodell definieren. Die Definition eines Metamodells, beziehungsweise einer Metamodellerweiterung, dient der Einführung neuer Element- und Beziehungstypen.

Akteur: Prozessingenieur

Auslöser: Notwendigkeit der Neuerstellung beziehungsweise Weiterentwicklung eines Metamodells für einen Entwicklungsstandard.

17 Abhängigkeiten und Beziehungen zwischen den Anwendungsfällen, wie sie durch die Mittel der UML 2 gegeben sind, modellieren wir nicht. Unsere Anwendungsfallstrukturen sind flach. Beziehungen stellen wir erst auf der Ebene des Lebenszyklusmodells in Kapitel 5.2 her, in dem wir die Modellierung präzisieren.

Ergebnis: Ein Metamodell für den Entwicklungsstandard.

Vorbedingungen: Sowohl für die Neuerstellung als auch für die Weiterentwicklung liegen die notwendigen Informationen, zum Beispiel in Form eines *Domänenmodells* vor.

Nachbedingungen: Ein Metamodell, beziehungsweise eine Metamodellerweiterung, gemäß der Anforderungen wurde erstellt. Siehe zum Beispiel Kapitel 5.1

3. Vorgehensmodell entwerfen. Der Entwurf eines Vorgehensmodells findet in der durch das Metamodell definierten Sprache statt. Hier werden neue Inhalte entwickelt und zusammengestellt. Die Entwicklung und Zusammenstellung erfolgt in modularen Einheiten. Ggf. sind auch organisationsspezifische Anpassungen oder die punktuelle Bildung von Varianten möglich.

Akteur: Prozessingenieur (ggf. Domänenexperte)

Auslöser: Eine Menge von Prozessanteilen muss entworfen und ggf. umgesetzt werden.

Ergebnis: Mindestens eine *Prozesskomponente* (modulare Einheit), die die modellierten Prozessanteile enthält und für die anschließende Konfiguration zu einem Vorgehensmodell anbietet.

Vorbedingungen: Die notwendigen Informationen liegen in umsetzbarer Form vor.

Nachbedingungen: Die entworfenen (und ggf. umgesetzten) Prozesskomponenten stehen für die Anwendung im Rahmen eines Releasebaus zur Verfügung.

4. Vorgehensmodell organisationsspezifisch anpassen. Die organisationsspezifische Anpassung eines Vorgehensmodells erfolgt dann, wenn ein gegebenes Vorgehensmodell nicht den Anforderungen einer umsetzenden Organisation genügt. Ggf. sind Teile der Anwendungsfälle *Vorgehensmodell entwerfen* oder *Variante erstellen* relevant.

Akteur: Prozessingenieur

Auslöser: Ein gegebener Entwicklungsstandard, beziehungsweise eine mögliche Ausprägung davon, müssen weiter gehend auf die Anforderungen einer implementierenden Organisation angepasst werden.

Ergebnis: Eine organisationsspezifische Anpassung des Entwicklungsstandards. Als Randfall betrachten wir, dass es nur *eine* Ausprägung gibt. Dann ist dies widerum nur eine *Variante* (siehe dann entsprechender Anwendungsfall).

Vorbedingungen: Ein Entwicklungsstandard ist gegeben und es stehen Anforderungen hinsichtlich einer notwendigen Anpassung bereit.

Nachbedingungen: Das organisationsspezifisch angepasste Vorgehensmodell genügt den Vorgaben des verwendeten Entwicklungsstandards.

5. Release erstellen. Zu einem Entwicklungsstandard werden (regelmäßig) neue Releases veröffentlicht, die eine konsolidierte, fehlerbereinigte Ausgangsbasis für abgeleitete Vorgehensmodelle darstellen. In einem Release werden somit alle relevanten Inhalte zuammengestellt und mit Hinblick auf die Konsistenz harmonisiert.

Akteur: Prozessingenieur (ggf. KM-Verantwortlicher)

Auslöser: Eine neue, integrierte Version eines Entwicklungsstandards beziehungsweise einer Ausprägung muss erstellt werden.

Ergebnis: Eine neue, integrierte und konsistente Version des Entwicklungsstandards.

Vorbedingungen: Entweder eine bereits gültige Konfiguration eines Entwicklungsstandards mit allen Prozessinhalten oder im Falle eines initialen Releasen alle für das Release benötigten Prozessinhalte.

Nachbedingungen: Ein neues, integriertes und konsistentes Release des Entwicklungsstandards.

6. Variante erstellen. Eine Variante eines Vorgehensmodells wird erstellt, in dem ein gegebenes Release eines Entwicklungsstandards oder eines daraus abgeleiteten Vorgehensmodells punktuell angepasst wird. Die erstellte Variante ist in sich konsistent; jedoch nicht zwangsweise zu anderen Varianten. Eine Konformität zum Entwicklungsstandard ist hingegen sichergestellt.

Akteur: Prozessingenieur (ggf. KM-Verantwortlicher)

Auslöser: Zu einem gegebenen Entwicklungsstandard oder einem bereits organisationsspezifisch angepassten Vorgehensmodell muss eine neue Variante erstellt werden.

Ergebnis: Eine neue Vorgehensmodellvariante.

Vorbedingungen: Ein allgemeiner Entwicklungsstandard oder ein bereits initial angepasstes organisationsspezifisches Vorgehensmodell sind gegeben.

Nachbedingungen: Die resultierende Variante ist strukturell konform zum Metamodell des Entwicklungsstandards.

Zusammenfassung

Die Analyse der Metamodelle der hier ausgewählten Vorgehensmodelle hat noch einmal gezeigt, wie „überschaubar" die Domäne *Vorgehensmodell* ist. Gleichzeitig wurde jedoch auch die These unterstützt, dass ob diesem einfachen Aufbau der Schlüssel für Modularität und Flexibilität in der Art der Verknüpfung der Inhalte liegt. Die Analyse in den Abschnitten 3.2 und 3.3 hat dies für statische und ablaufbezogene Metamodellanteile gezeigt.

Im Anschluss haben wir auf der Basis der gegebenen Vorgehensmetamodelle die *Kompositions- und Distributionsmuster* (Abschnitt 3.5) näher untersucht. Wir haben dabei sowohl die konzeptionellen Ideen und Fähigkeiten auf der Basis der analysierten Metamodelle betrachtet, als auch reale technische Möglichkeiten, die wir im Rahmen verschiedener Arbeiten (vgl. Anhang A) untersucht haben. Es wurde an dieser Stelle klar, dass Vorgehensmodelle, auch wenn sie über ein explizites Modulkonzept verfügen, nicht zwangsläufig einfach verteilbar und beliebig konfigurierbar sind. Das V-Modell XT und MSF sind hier zwei Extrembeispiele. Das V-Modell XT definiert ein Modulkonzept, setzt dies aber physisch nicht durch. MSF definiert ein physisches Modulkonzept, jedoch keine tragfähige Integration für einen Gesamtprozess. EPF/UMA liegt hier in der Mitte: Er definiert sowohl ein Modul- als auch ein Integrationskonzept. Seine Komplexität und die (noch) fehlende Plattformorientierung machen ihn aber nur schwer anwendbar [BULL07].

Die Ergebnisse der Analyse der betrachteten Vorgehensmetamodelle wurden für die Feststellung von Anforderungen an modulare Verteilungseinheiten verwendet (vgl. Abschnitt 3.6). Auf der logischen Ebene sind die Konzepte des V-Modells am ausgewogensten. Sie beschreiben ein Modulkonzept, das auf die Modellierung von Teilprozessen ausgelegt ist. Die Konzepte liefern somit einen gut definierten Rahmen und schränken anders als EPF/UMA die Flexibilität sinnvoll ein. Dieses als Grundlage nehmend haben wir Anforderungen auf der Basis der möglichen Aktivitäten der Prozessingenieure festgelegt. Die Anforderungen umfassen:

- Ein Metamodell für die Strukturen und die Verteilung
- Die Definition von Soll-Bruchstellen zur Entwicklung, Verteilung und Anpassung
- Die Definition von Kompositionsmustern für Releasebildung und Anpassung
- Ein kontrollierbares Abhängigkeitssystem
- Die Unterstützung von Anpassungs- und Variabilitätsoperationen im Metamodell

Neben den Anforderungen haben wir den Kontext durch ausgewählte Anwendungsfälle präzisiert. Diese Anwendungsfälle spiegeln sich im späteren Verlauf unmittelbar im *Lebenszyklusmodell* (vgl. Kapitel 5.2) wider.

4. Analyse: Formalisierung für Hierarchien und Varianten

In diesem Kapitel befassen wir uns mit der grundlegenden Fragestellung hinsichtlich der Formalisierung von Vorgehensmodellen im Kontext der Modularität, der Komponierbarkeit sowie der Bildung von Linien und darauf aufbauend der Ableitung von Varianten. Wir betrachten in diesem Kapitel die verschiedenen Hierarchieebenen, ausgehend von einem Entwicklungs-, beziehungsweise Vorgehenssstandard bis hin zu projektspezifischen Modellen. Wir positionieren die einzelnen Ebenen in einem Phasenmodell, dass Lebenszyklen für Vorgehensmodelle beschreiben kann. Auf der Vorstellung der Hierarchieebenen und dem Lebenszyklus aufbauend formalisieren wir allgemein die grundlegenden *Prozesselemente* eines Vorgehensmodells sowie darauf aufbauend einfach strukturierte *Prozesskomponenten*, die uns als Entwicklungs-, Verteilungs- und Kompositionseinheiten für die Entwicklung und Variation von Vorgehensmodellen dienen. Anschließend zeigen wir die schrittweise Komposition eines komplexen Vorgehensmodells aus den elementaren Bausteinen und diskutieren im Folgenden Fragen zur Evolution. Anhand von Beispielen zeigen wir zum Schluss des Kapitels die prinzipiellen Auswirkungen bei der Anpassung eines Vorgehensmodells. Bei den Inhalten und Überlegungen dieses Kapitels handelt es sich um allgemeine Konzepte und Fragestellungen. Daher verzichten wir hier weitgehend auf Typisierungen oder Konkretisierungen. Diese nehmen wir im Kapitel 5 vor, das auf diesem Kapitel aufbaut und die Basiskonzepte in die Anwendung überführt.

Am Ende dieses Kapitels haben wir die grundlegenden Anforderungen an modulare Vorgehensmodelle präzisiert und Umsetzungsmöglichkeiten gezeigt. Die hierfür grundlegenden Begriffe Version und Variante sind definiert und eingeführt. Weiterhin ist das Basiskonzept der Prozesskomponenten eingeführt worden.

4.1. Hierarchien und Integrationsebenen für Vorgehensmodelle

Bereits in Kapitel 2.3.3 wurde auf der Grundlage von Anpassungsprozessen für Vorgehensmodelle und Produktlinienprozesse der Bedarf eines Lebenszyklusmodells herausgestellt. Aufbauend auf den analysierten Prozessen haben wir folgende benötigte Phasen identifiziert:

Feststellung und Planung – In dieser Phase werden Bedarfsermittlung und Planungen durchgeführt, die für eine Entwicklung beziehungsweise Anpassung eines Prozesses erforderlich sind.

Umsetzung – In dieser Phase werden Umsetzungskonzepte beziehungsweise Spezifikationen realisiert.

Stabilisierung – In dieser Phase werden neue beziehungsweise aktualisierte Inhalte/Vorgehensmodelle oder Teile davon erprobt.

Einführung und Bereitstellung – Neu erstellte Vorgehensmodelle oder Teile davon werden in der Breite eingeführt beziehungsweise bereitgestellt.

Abbildung 4.1 illustriert diese Phasen in einem Zyklus. Gehen wir von einzelnen Vorgehensmodellen beziehungsweise ihren Instanzen aus, ist dieser Zyklus *idealtypisch* und findet sich in vielen Anpassungsprojekten (mit marginalen Verfahrensabweichungen) wieder [NR05, AEH+07]. Unser Modell ist grob granularer, skaliert dafür jedoch auch über einzelne Vorgehensmodelle hinaus.

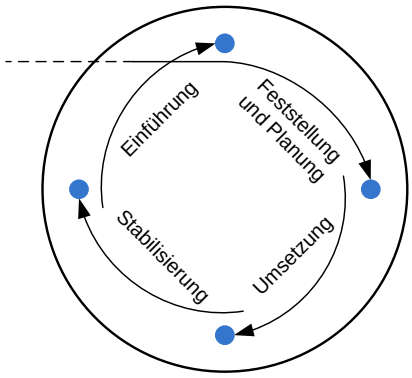


Abbildung 4.1.: Allgemeines, phasenorientiertes Modell für einen Vorgehensmodelllebenszyklus

Überblick und Kontext

Die Notwendigkeit, über einzelne Vorgehensmodelle hinaus zu skalieren wird offensichtlich, wenn wir anstelle eines Einzelmodells einen *Entwicklungsstandard* betrachten. Unter einem Entwicklungsstandard verstehen wir dabei ein umfassendes Rahmenwerk, das neben einem Metavorgehen und grundlegenden Strukturen eine Menge von Operationen zu seiner Anpassung/Ausgestaltung enthält. Von einem Entwicklungsstandard verlangen wir somit, Eigenschaften eines Baukastens bereitzustellen, aus dem Bausteine (Komponenten) für die Ableitung konkreter Prozesse mit definierten Operationen verwendet werden können.

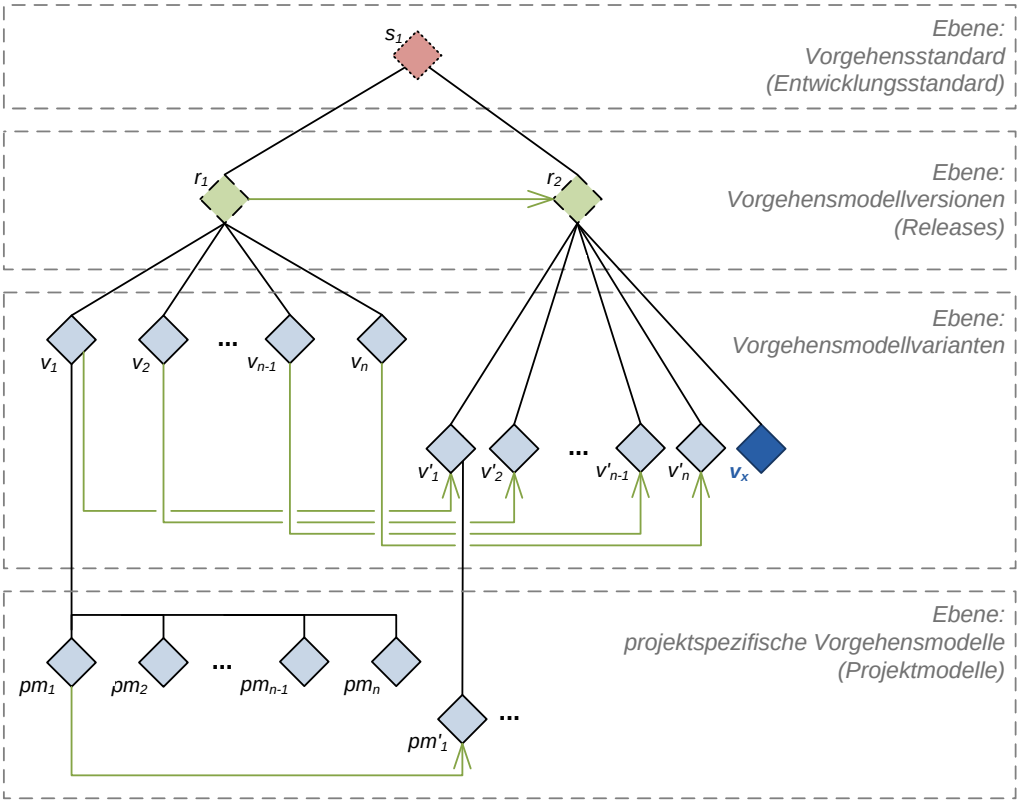


Abbildung 4.2.: Hierarchiebenen eines Vorgehensmodells ausgehend von einem Entwicklungsstandard hin zu projektspezifisch angepassten Projektvorgehen

Anhand von Abbildung 4.2 wollen wir dies näher erläutern. Sie zeigt auf der obers-

ten Hierarchieebene einen Entwicklungsstrandard s_1. Dieser Entwicklungsstandard beschreibt in unserem Modell einmal die wesentlichen strukturellen Basiseigenschaften (Metamodell, siehe Kapitel 5.1, vgl. Kapitel 2.4.1), die für die Erstellung von Vorgehensmodellen notwendig sind. Üblicherweise sind dies die einzelnen Submodelle eines Vorgehensmodells (Rollen, Produkte etc.). Eine erste inhaltliche Ausgestaltung findet sich dann auf der Ebene der Vorgehensmodellversionen.

Definition 4.1 (Vorgehensmodellversion (Version, Release))**:**
Eine Version/Release eines Vorgehensmodells ist eine konsistente, abgestimmte Zusammenstellung aller Komponenten eines Vorgehensmodells ohne die Berücksichtigung ausgewählter Teilaspekte.

Abbildung 4.2 zeigt zwei Versionen r_1 und r_2, die Konkretisierungen beziehungsweise Ausgestaltungen von s_1 sind. Um das etwas plastischer zu gestalten verweisen wir auf Abbildung 4.3, wo wir in den einzelnen Hierarchieebenen eine Positionierung des V-Modell XT vorgenommen haben. Das V-Modell XT fungiert hier als Entwicklungsstandard, Versionen des V-Modells sind die bislang veröffentlichten Releases 1.0 bis 1.2.1. Nach Definition 4.1 betrachten wir eine Vorgehensmodellversion als Zusammenstellung konsistenter Inhalte. In unserem Modell ist hierbei noch eine gewissen Generizität vorgesehen, d. h. die Inhalte der Version decken ein Maximum möglicher Anwendungsfelder ab, ohne Einzelne bevorzugt zu verfeinern. Alle wesentlichen Prozesse sind soweit ausdetailliert, dass eine Projektdurchführung prinzipiell schon ermöglicht wird. Eine Version wird also auf einem allgemeingültigen, generischen *Referenzmodell* definiert, das wesentliche, zwingend erforderliche Prozess beschreibt, in der restlichen inhaltlichen Ausgestaltung jedoch mehr in die Breite als in die Tiefe geht.

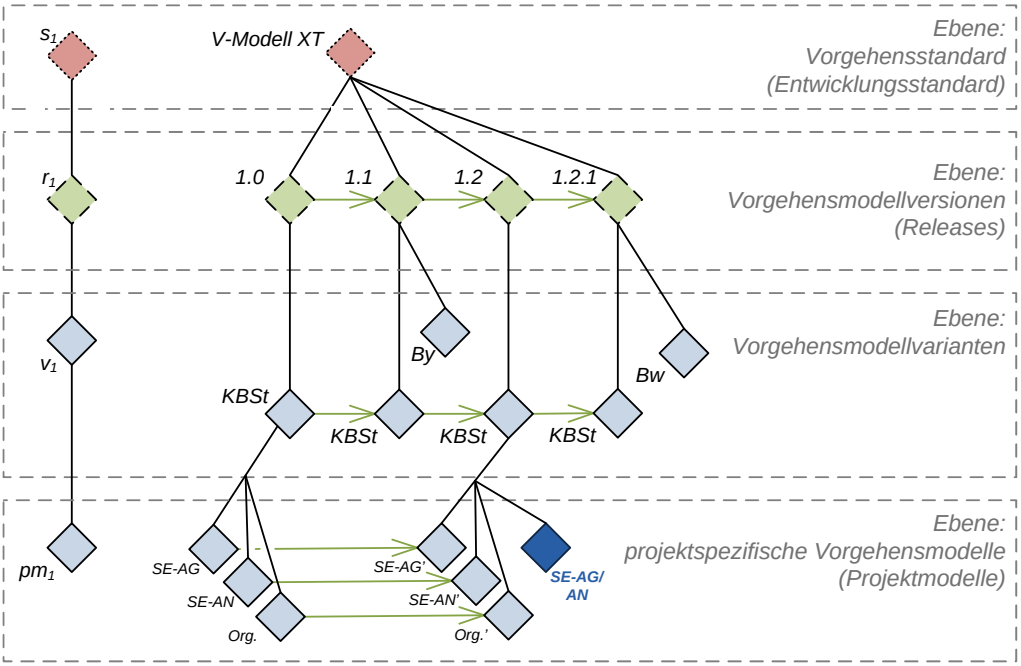


Abbildung 4.3.: Hierarchiebenen eines Vorgehensmodells ausgehend von einem Entwicklungsstandard hin zu projektspezifisch angepassten Projektvorgehen am Beispiel V-Modell XT

Die (punktuelle) Ausgestaltung der Tiefenaspekte einer Vorgehensmodellversion betrachten wir in der *Variantenbildung*.

Definition 4.2 (Vorgehensmodellvariante (Variante))**:**
Eine Variante eines Vorgehensmodells ist eine spezifische und punktuelle Anpassung, die eine Teilmenge der zur Verfügung stehenden Komponenten des Gesamtmodells adressiert. Die Konsistenz zu nicht berücksichtigten Teilen muss nicht sichergestellt sein.

Die Bildung einer Variante eines Vorgehensmodells ist also bereits den *kontextspezifischen Spezialisierungen* zuzuordnen. Relevante Teilaspekte werden ergänzt, angepasst beziehungsweise nicht relevante Teile können hier auch entfernt werden (vgl. Kapitel 2.2). Die Variante hat dabei immer noch eine Restübereinstimmung mit der Version, aus der sie ausgeleitet wurde. Analoges gilt für die Beziehung zum Vorgehensstandard. Weiterhin ist die Variantenbildung als Spezialisierung bereits Teil eines evolutionären Prozesses. Diesen betrachten wir im folgenden Abschnitt 4.1.1 beziehungsweise in Abschnitt 4.3 detaillierter. Bezogen auf die Abbildungen 4.2 und 4.3 finden wir für die Version r_1 eine Menge von n Varianten $\{v_1, v_2, \ldots, v_{n-1}, v_n\}$, die aus der Version abgeleitet werden. Für die nächste Version r_2 finden wir neben den Weiterentwicklungen $\{v'_1, v'_2, \ldots, v'_{n-1}, v'_n\}$ eine zusätzliche Variante v_x, die als Zusatzentwicklung von r_2 zu verstehen ist. Wir können dieses abstrakte Szenario wieder am Beispiel des V-Modell XT zeigen, wo wir als Versionen die Menge der Releases 1.0, 1.1, 1.2 und 1.2.1 finden (ohne die Auskopplungen für das englische V-Modell). Die Version 1.0 des V-Modells bestand im Wesentlichen aus der so genannten *KBSt*-Version, die auch als Standard-V-Modell fungiert. Dass zu einer Version eines Vorgehensmodells nur eine Variante existieren kann, ist möglich und weist, sofern keine weiteren Varianten existieren, auf die Instanziierung eines Referenzmodells hin. Mit der Version 1.1 wurde das KBSt-Modell weiter entwickelt und gepflegt. Gleichzeitig begann die Variantenbildung, sodass neben dem KBSt-Modell gleichzeitig das V-Modell-Bayern (*By*) erstellt wurde. Die umfassenste Variante des V-Modells findet sich ab der Version 1.2.1 mit dem V-Modell Bw[1].

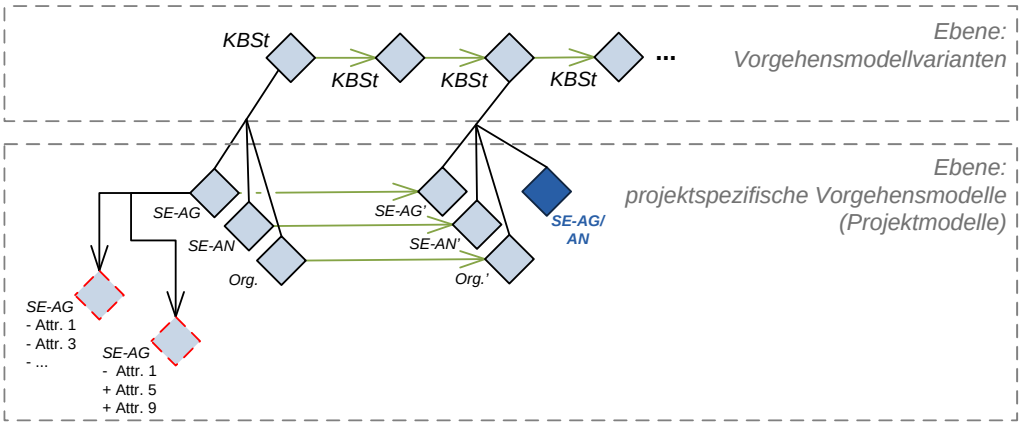


Abbildung 4.4.: Hierarchiebenen eines Vorgehensmodells ausgehend von einem Entwicklungsstandard hin zu projektspezifisch angepassten Projektvorgehen am Beispiel V-Modell XT und seiner weiteren Verfeinerungen auf der Ebene von Projekten

In Abbildung 4.4 haben wir die Abbildungen 4.2 und 4.3 für die Hierarchieebene der projektspezifischen Vorgehensmodelle (Projektmodelle) verfeinert. Auch hier ist es wieder möglich, dass eine Variante durch eine Menge von Projektmodellen weiter spezialisiert wird. Am Beispiel von Abbildung 4.2 ist das für die Variante v_1 die Menge $\{pm_1, pm_2, \ldots, pm_{n-1}, pm_n\}$. Überführt auf das V-Modell XT ergeben sich als erste mögliche projektspezifische Verfeinerungen die Projekttypen, wie das *Systementwicklungsprojekt AG* (SE-AG). Diese sind durch Parametrisierung (Tailoring, vgl. Kapitel 2.1.1) noch weiter anpassbar (Abbildung 4.4). Auch hier gilt wie bei der Beziehung zwischen Varianten und Versionen, dass bestimmte Charakteristika erhalten bleiben.

4.1.1. Ein Beispielszenario: Lebenszyklus eines Vorgehensmodells

Die Ableitungen von Varianten und die Bildung weiterer, darauf aufbauender Spezialisierungen sind Vorgänge, die massiv parallel erfolgen und für die ein Entwicklungsstandard einen definierten Rahmen angeben muss. In Abbildung 4.5 geben wir eine

1 Eigentlich ist das V-Modell Bw bereits ab der Version 1.2. des V-Modells verfügbar, wird jedoch erst synchronisiert mit der Version 1.2.1 und der korrespondierenden englischen Version veröffentlicht.

Instanziierung des Phasenmodells aus Abbildung 4.1 an, in dem wir insgesamt für fünf Vorgehensmodellvarianten (inkl. eines Referenzmodells) in ihrem aktuellen Entwicklungsstand zeigen. Während eine neue Version des Referenzmodells wieder in der Planung ist, befindet sich Variante 1 gerade in der Einführung; bei Variante 2 wurde gerade mit der Umsetzung begonnen; Variante 3 ist bereits eingeführt und wird auch wieder für eine Aktualisierung vorbereitet während Variante 4 gerade in der Pilotierung (Stabilisierung) ist und kurz vor der Einführung steht. Die Varianten 1 bis 4 setzen auf dem Referenzmodell auf, benötigen also jeweils einen definierten Aufsetzpunkt. Gehen wir davon aus, dass das Referenzmodell in der Version 1.0 vorliegt und die Abbildung 4.5 einen Snapshot der Prozesslinie zu einem Zeitpunkt t_0 darstellt, so wird mit Variante 1 eine Variante eingeführt, die auf der Version 1.0 des Referenzmodells basiert. Variante 2 wird ebenfalls auf Basis des Referenzmodells Version 1.0 erstellt, ebenso wie Variante 4 gemäß Version 1.0 des Referenzmodells pilotiert wird. Variante 3 steht an der Schwelle zur Planung, während das Referenzmodell bereits bei der Planung und Spezifikation von Version 1.1 steht. An dieser Stelle ist nun bereits die Frage zu beantworten, ob Variante 3 noch auf der Basis vom Referenzmodell 1.0 spezifiziert wird oder bis zur Fertigstellung der Version 1.1 des Referenzmodells zurückgestellt wird.

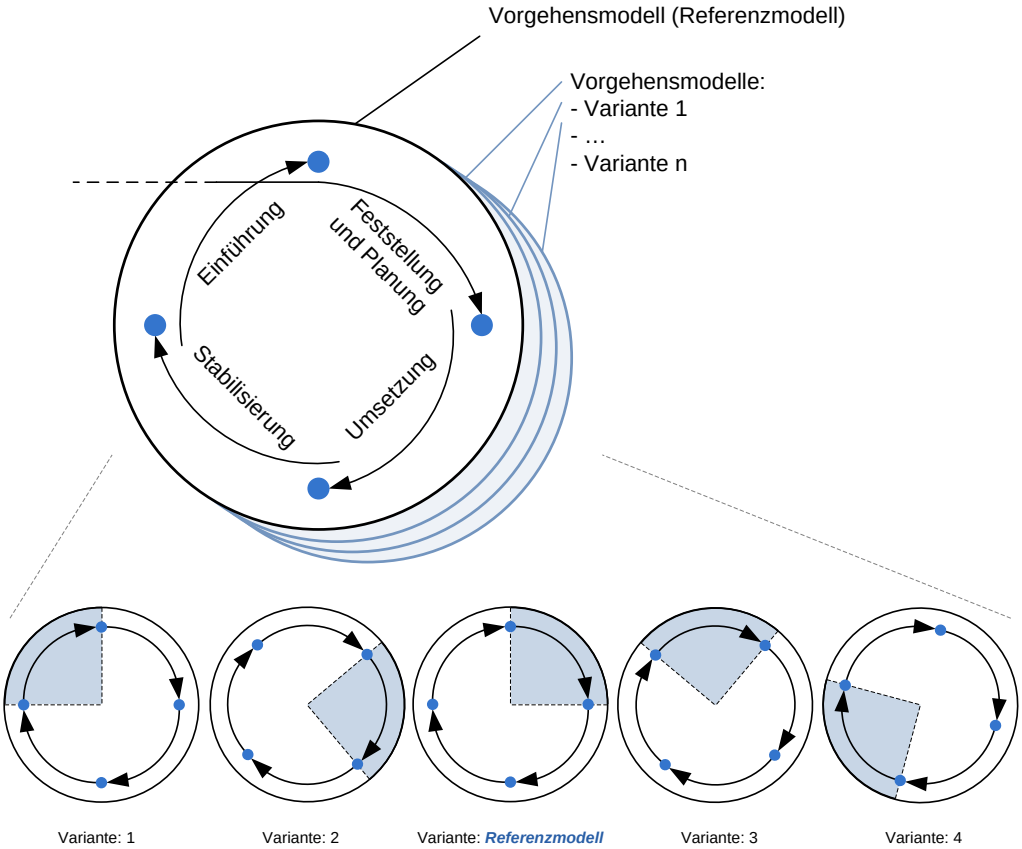


Abbildung 4.5.: Anwendung des allgemeinen Modells für einen Vorgehensmodelllebenszyklus im Kontext einer Prozesslinie

Dieses Beispiel zeigt sehr eindrucksvoll, dass es erforderlich ist, Entwicklungs- und Anpassungsprozesse zu synchronisieren und abzugleichen. Denn es treten noch weitere Fragen auf:

- Wie wirken sich die Änderungen des Referenzmodells auf die abgeleiteten Varianten aus?
- Sind abgeleitete Varianten weiterhin gültig, wenn ein neues Referenzmodell (eine neue Version) vorliegt?
- Wie ist mit nebenläufigen Entwicklungen auf der Variantenebene und der Ebene des Referenzmodells umzugehen, insbesondere, wenn sie widersprüchlich sind?

Die Problematik dieser Situation leitet sich aus der unterschiedlichen Wirkungsweise der Einzelentwicklungsprozesse her. Die Variantenbildung setzt auf einem definierten und stabilen Basismodell – dem Referenzmodell (der Version) – auf. Anpassungen, Ausgestaltungen und alle weiteren Operationen stellen *Variationen* auf den Strukturen des Referenzmodells dar (vgl. Kapitel 2.3.2). Gemäß den Vereinbarungen aus Kapitel 2.2 sind dies *inhaltliche Anpassungen* beziehungsweise *Ausgestaltungen*. Bei der Entwicklung, beziehungsweise Weiterentwicklung, des Referenzmodells ändert sich auch die Basis für alle abgeleiteten Varianten. Die Änderungen, die hier zu erwarten sind, sind einmal inhaltlicher Natur, da das Referenzmodell auch allgemeine Inhalte enthält. Andererseits können bei einer Weiterentwicklung des Referenzmodells auch *strukturelle* Anpassungen auftreten, die in unserem Modell in der Regel durch Änderungen am Vorgehensstandard angestoßen werden. Dies betrifft insbesondere Änderungen am *Metamodell*, das dem Vorgehensstandard zugrunde liegt. Wir betrachten diese Problematik im Folgenden und verfeinern entsprechend die Strukturen sowohl mit Hinblick auf die Varianten als auch mit Hinblick auf die Releases in den einzelnen Hierarchieebenen.

4.1.2. Elemente und Beziehungen zwischen Hierarchieebenen

Bereits in Abbildung 4.2 sind die wesentlichen Beziehungstypen, die wir nun betrachten wollen, zu finden. Dabei handelt es sich einmal um Beziehungen, die der Verfeinerung/Spezialisierung dienen (vertikale Beziehungen). Weiterhin sind Beziehungen zwischen gleichberechtigten Versionen (horizontal) zu identifizieren. In diesem Abschnitt widmen wir uns diesen Beziehungen.

Formalisierbare Beziehungen Eine Unterscheidung können wir bereits unmittelbar treffen: Es gibt Beziehungstypen, die sind (uneingeschränkt) formalisierbar und solche, die günstigstenfalls semi-formal beschreibbar sind. Die Beziehungen, die wir zwischen Entwicklungsstandard, Release, Varianten und projektspezifischen Vorgehen aufbauen können, sind alle formalisierbar. Wie bereits angedeutet, handelt es sich hierbei um Beziehungen, die über *Graphen* aufgebaut werden können und Variationen in den Graphen beschreiben. Um dies weiter gehend diskutieren zu können, verfeinern wir an dieser Stelle unsere Vorstellung eines Vorgehensmodells – zunächst auf einer allgemeinen, abstrakten Ebene. Abbildung 4.6 zeigt die schrittweise Verfeinerung unserer Vorstellung eines modularen Vorgehensmodells, dessen Bestandteile wir noch weiter beschreiben werden. Für die aktuellen Betrachtungen interessieren uns dabei zunächst nur die drei unterschiedlichen *Integrationsebenen* eines Vorgehensmodells:

Ebene 0: Diese Ebene umfasst alle *Prozesselemente*. Da Prozesselemente atomare Bausteine sind, gibt es auf dieser Integrationsstufe keinen weiteren Elementhierarchien. Alle Prozesselemente werden hier gleichberechtigt behandelt.

Ebene 1: Auf dieser Ebene betrachten wir *Prozesskomponenten*, die ihrerseits Mengen von Prozesselementen enthalten und durch Beziehungen strukturieren.

Ebene 2: Auf dieser Ebene betrachten wir *Vorgehensmodellvarianten*, die selbst wieder aus einer Menge von Prozesskomponenten bestehen, diese miteinander in Beziehung setzen und sinnvoll strukturieren.

Prozesselemente. Die kleinste für unseren Ansatz relevante Einheit ist ein Prozesselement (vgl. Abbildung 4.6). Anders als Bhuta et al. [BBM05] betrachten wir Prozesselemente als atomare Einheiten. Sie beschreiben ein Prozesselement wie folgt:

Process Elements

„. . . A Process Element (PE) is a group of project activities, and/or other process elements related by logical dependencies, which when executed (or enacted) provides value to the project."

Weiterhin wird ein Prozesselement mit Vor- und Nachbedingungen, sowie Ein- und Ausgabeschnittstellen definiert. Das Prozesselement nach [BBM05] ist somit eher den

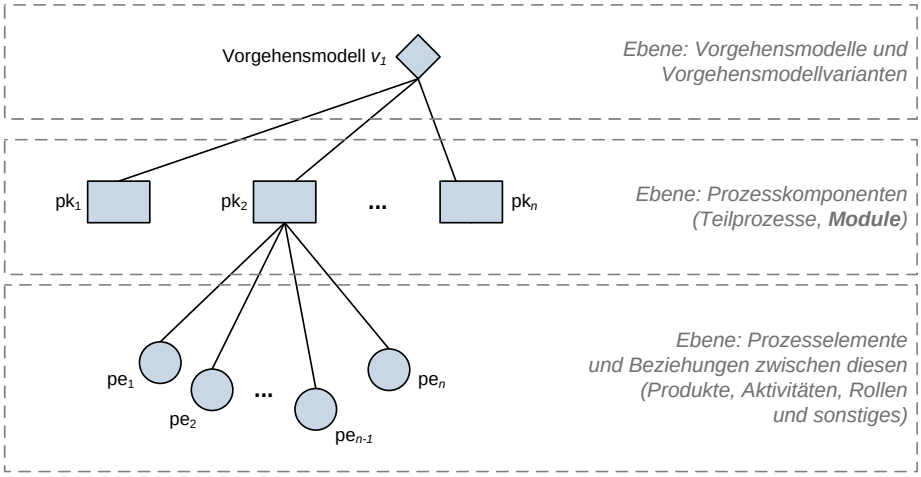


Abbildung 4.6.: Elemente und Integrationsebenen eines Vorgehensmodells

Prozessmustern [GMP+03] zuzuordnen. Wir verstehen unter Prozesselementen jedoch alle Elemente, die zusammengefasst einen Prozess[2] ausmachen.

Definition 4.3 (Prozesselement)**:**
Ein Prozesselement $p \in Pe$ ist ein atomarer Baustein für ein Vorgehensmodell. Prozesselemente sind abstrakt und für die Anwendung in einem Vorgehensmodell passend, also zum Beispiel Produkte, Rollen, Aktivitäten oder Werkzeuge, zu konkretisieren.

Eine entsprechende Verfeinerung mitsamt Typisierung der Prozesselemente nehmen wir in Kapitel 5.1 im Rahmen der Metamodellerstellung vor. Für den Augenblick genügt uns jedoch die Annahme, dass Prozesselemente atomare Einheiten sind. Komplexe, komposite Strukturen über Prozesselementen müssen wir jedoch auch angeben, insbesondere dann, wenn wir mehrere Prozesselemente zu Prozesskomponenten zusammenfassen wollen.

Definition 4.4 (Prozesskomponente (allgemein))**:**
Eine Prozesskomponente $pk \in Pk$ besteht aus einer Menge zusammengehöriger Prozesselemente und deren Beziehungen untereinander.

Die zu betrachtenden Beziehungen zwischen Prozesselementen bei der Komposition zu Prozesskomponenten (Abbildung 4.6, Übergang von Ebene 0 nach Ebene 1) heißen *Prozesselementabhängigkeiten*. Wir unterscheiden bei Prozesselementabhängigkeiten prinzipiell zwischen einer *einfachen* (*pea*) und einer *erweiterten* (*epea*) Prozesselementabhängigkeit. Prozesselementabhängigkeiten sind dabei Relationen über der Menge der Prozesselemente, wobei für die einfache Prozesselementabhängigkeit *pea* gilt:

$$\mathrm{PEA} = \{(x, y) \,|\, x, y \in \mathrm{Pe} \wedge x \neq y\} \tag{4.1}$$

Analog gilt für die erweiterte Prozesselementabhängigkeit *epea*:

$$\mathrm{EPEA} = \{(x, Y) \,|\, x \in \mathrm{Pe} \wedge \emptyset \neq Y \subseteq \mathrm{Pe} \wedge x \notin Y\} \tag{4.2}$$

Für die Relationen, mit denen wir die Prozesselementabhängigkeiten modellieren, wenden wir die Prädikatenschreibweise, also *pea* (x, y) und *epea* $(x, \{y_1, \ldots, y_n\})$, an. Die erweiterte Prozesselementabhängigkeit dient uns dabei als Vereinfachung der Modellierung. Insbesondere im Rahmen der Metamodellerstellung spielen die Spezialisierungen der Prozesselementabhängigkeiten eine wichtige Rolle im Kontext der Strukturmodellierung.

2 Bezogen auf ein Vorgehensmodell ist es auch möglich, von *Vorgehenselementen* zu sprechen. Wir verwenden hier aufgrund der größeren Allgemeinheit und besseren Abbildbarkeit jedoch konsequent den Begriff Prozesselement.

Beispiel: Ein Beispiel für eine einfache, gerichtete Prozesselementabhängigkeit ist eine Verantwortungsbeziehung zwischen Rollen und Produkten, wie sie zum Beispiel das V-Modell XT definiert. Es definiert, dass für jedes nicht externe Produkt genau eine Rolle verantwortlich ist. Produkte und Rollen modellieren wir als Prozesselemente, die Verantwortungsbeziehung als Prozesselementabhängigkeit.

Beispiel: Ein Beispiel für die erweiterte Prozesselementabhängigkeit findet sich ebenfalls wieder im Rollenmodell des V-Modell XT. Neben der Verantwortung können Rollen auch an der Produkterstellung mitwirken. Nehmen wir beispielsweise das Produkt *Projekthandbuch*, so ist für dieses die Rolle *Projektleiter* verantwortlich. Mitwirkend sind hier beispielsweise die Rollen *Projektmanager*, *Projektkaufmann*, *Systemarchitekt*... Werden also die Rollen als Menge von Prozesselementen definiert, findet hier eine Abbildung vom Prozesselement Projekthandbuch in eine Teilmenge der Rollen statt.

Die Abbildung 4.7 illustriert das gerade aufgeführte Beispiel. Sei also die Menge der Prozesselemente bestimmt durch *Projekthandbuch* (PHB), *Projektplan* (PPL), *Projektleiter* (PL), *Projektmanager* (PM), *Projektkaufmann* (PKf) und *Systemarchitekt* (SysA). Beziehungstypen, die wir betrachten, sind: *ist_verantwortlich_fuer* und *wirkt_mit*. Anhand der bislang definierten Regeln gestaltet sich die Modellierung dabei wie folgt:

$$\begin{aligned} \text{Pe} &= \{\text{PHB}, \text{PPL}, \text{PL}, \text{PM}, \text{PKf}, \text{SysA}\} \\ \text{ist\_verantwortlich\_fuer} &\subseteq \text{PEA} \\ \text{wirkt\_mit} &\subseteq \text{EPEA} \end{aligned}$$

$$\begin{aligned} \text{Abhängigkeiten:} \quad & \text{ist\_verantwortlich\_fuer}\,(\text{PL}, \text{PHB}) \\ & \text{wirkt\_mit}\,(\text{PM}, \{\text{PHB}, \text{PPL}\}) \end{aligned}$$

Prozesskomponenten. Die Abhängigkeit *wirkt_mit* $(\text{PM}, \{\text{PHB}, \text{PPL}\})$ können wir auch gleichwertig mit *wirkt_mit'* $(\text{PM}, \text{PHB}) \wedge$ *wirkt_mit'* (PM, PPL) ausdrücken. Mit *wirkt_mit'* nehmen wir die Rückführung der Elemente von *EPEA* auf Relationen aus *PEA* vor. Analog zum gerade gezeigten Ausschnitt modellieren wir mithilfe von Prozesselementen und den über ihnen definierten Relationen *PEA* und *EPEA* weitere Elementtypen und deren Beziehungen untereinander. In Abbildung 4.7 ist dies am Beispiel der Relation *erzeugt* und diverser anderer Prozesselemente zu sehen.

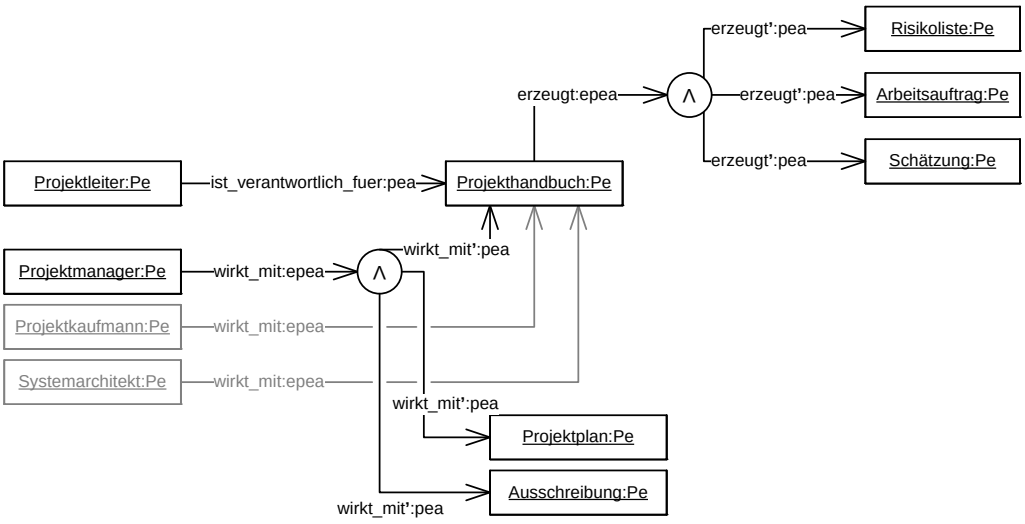


Abbildung 4.7.: Beispiel einer Struktur von Prozesselementen mit verschiedenen Prozesselementabhängigkeiten

Wir konstruieren somit mit dem einfachen Modell des Prozesselements und der verknüpfenden Prozesselementabhängigkeit einen *Abhängigkeitsgraphen* analog zu Abbildung 3.18 (Seite 70). Auf diese Art und Weise entstehen schrittweise die Strukturen eines Vorgehensmodells. Die Spezialisierung der einzelnen Submodelle (vgl. Kapitel 2.4

und Kapitel 3) aus allgemeinen Prozesselementen, die wir in Kapitel 5.1 vornehmen, lässt sich *immer* auf dieses einfache Muster zurückführen. Im oben gezeigten Beispiel sehen wir noch einen weiteren Aspekt: Wir verwenden für die Konstruktion eines Vorgehensmodells *zwei* verschiedene Graphen. Abbildung 4.6 zeigt den ersten Typ, den *Strukturgraphen*[3] S. Der Strukturgraph ist als Baum modelliert (hierarchisch, gerichtet, zyklenfrei). Dieser Graph dient der Komposition von Einzelelementen, wobei Prozesselemente von Prozesskomponenten als Container aufgenommen werden. Prozesskomponenten selbst werden durch Vorgehensmodellvarianten verwendet.

Der zweite Typ Graph, den wir betrachten, ist der *Abhängigkeitsgraph* G. In ihrer allgemeinen Form sind Abhängigkeitsgraphen zunächst nicht zyklenfrei. Sofern notwendig fügen wir die Zyklenfreiheit als Eigenschaft durch die Verfeinerungsschritte bei der Modellierung eines Vorgehensmetamodells (Kapitel 5.1) hinzu. Die Strukturen über Prozesselementen sind für uns grundlegend für die Konstruktion von Vorgehensmodellen. Für den Konstruktionsprozess verwenden wir den Begriff *Konfiguration*. Beide Graphen stehen in der Beziehung: $V_G \subseteq V_S$. Das heißt, dass ein Abhängigkeitsgraph nur Knoten v aus der Menge der Knoten des Strukturgraphen V_S enthalten darf. Die Menge der Anhängigkeiten ist hingegen von diesen Einschränkungen nicht betroffen.

Definition 4.5 (Konfiguration)**:**
Eine Konfiguration von Prozesselementen ist ein gerichteter Graph G_{Pe}, in dem die Prozesselemente die Knoten und die Prozesselementabhängigkeiten die Kanten bilden.

Es existiert ein entsprechender Abhängigkeitsgraph für jede betrachtete Prozesskomponente. Unser Basismodell setzt auf einem *konfigurationsbasierten Ansatz* auf, sodass die Konfiguration von Prozesselementen und weiter aufbauend die Konfiguration von Prozesskomponenten eine zentrale Rolle spielt. Die Konfigurationen sind formalisierbare Beziehungstypen, die berechenbar sind. Strukturen einer Variante eines Vorgehensmodells sind aus ihren Konfigurationsdaten ermittelbar. Diese setzen sich aus den Konfigurationsdaten der zugrunde liegenden Prozesskomponenten zusammen. Sind also pk_1 und pk_2 zwei Prozesskomponenten, die in einer Vorgehensmodellvariante v_1 enthalten sind, so gilt auch für die Abhängigkeitsgraphen G_{pk_1} und G_{pk_2}:

$$\mathrm{G}_{v_1} \supseteq \mathrm{G}_{\mathrm{pk}_1} \cup \mathrm{G}_{\mathrm{pk}_2} \tag{4.3}$$

Die Abhängigkeitsgraphen der Prozesskomponenten sind also Teilgraphen des Abhängigkeitsgraphen der Vorgehensmodellvariante. Dies ist zunächst der naive Ansatz, der für die Variantenbildung ohne Variabilitätsoperationen (siehe Abschnitt 4.3.3) gilt. Sofern Variabilitätsoperationen angewendet werden, müssen wir entsprechende Verfeinerungen dieser Aussage vornehmen. Selbes gilt auch für die korrespondierenden Strukturgraphen S_i und die Knotenmengen der Graphen. Mit den Feinheiten der Komposition befassen wir uns noch im Abschnitt 4.2. Wenn wir uns im Abschnitt 4.1 darauf bezogen haben, dass sich (strukturelle) Ähnlichkeiten von einer Version über aus ihr abgeleiteten Varianten bis hin zu projekspezifischen Modellen widerspiegeln, können wir dies durch Teilgraphen formalisieren.

4.2. Komposition von Prozesskomponenten und Vorgehensmodellvarianten

In diesem Abschnitt greifen wir die Anforderungen an die Modularität der Bausteine von Vorgehensmodellen auf und formulieren weiter gehende Eigenschaften von *Prozesskomponenten*[4].

3 Da wir hier auch strukturelle Verküpfungen auf höheren Ebenen darstellen können, bezeichnen wir diesen Graphen auch als *Strukturkompositionsgraph*.

4 Obwohl es sich um Bausteine eines Vorgehensmodells handelt vermeiden wir den Gebrauch der V-Modell XT-Terminologie: *Vorgehensbaustein*, da dieses Konzept keine physisch eigenständigen Komponenten zur Verteilung vorsieht. Wir orientieren uns statt dessen an den ebenfalls etablierten Begriffen von EPF/UMA und OPF.

4.2.1. Bildung von Prozesskomponenten

Wir betrachten Prozesskomponenten als komplexe, zusammenhängende Strukturen von Prozesselementen. Sie bilden die Einheiten, aus denen ein Vorgehensmodell zusammengestellt wird (vgl. Abbildung 4.6). Prozesskomponenten enthalten nach Definition 4.4 zum Beispiel Produkte oder Rollen, die im Kontext eines Vorgehensmodells geeignet zusammenzufassen sind. Die Strukturierung/die Zusammenstellung erfolgt dann in der Regel unter Gesichtspunkten/Anforderungen der thematischen Gruppierung.

Schnittstelle einer Prozesskomponente. Eine Prozesskomponente fasst verschiedene Prozesselemente zusammen und stellt Beziehungen unter diesen her. Die entstehende Struktur liefert die *Konfiguration einer Prozesskomponente* (vgl. Definition 4.5). Eine Prozesskomponente kann durch die definierte Schnittstelle Koppelstellen für Prozesselemente aus anderen Prozesskomponenten anbieten. Nach Siedersleben [Sie04] führen wir einen Schnittstellenbegriff für $pk\,|_G$ ein. Die Projektion $pk\,|_G$ heißt *angebotene Schnittstelle* einer Prozesskomponente $pk \in Pk$. G bezieht sich dabei auf den Abhängigkeitsgraphen, der für pk eine Menge Prozesselemente Pe und alle Kanten zwischen diesen Elementen durch $PEA \cup EPEA$ zusammenfasst.

Schnittstellen von Prozesskomponenten definieren wir anhand ihrer Konfigurationen. Diese gestatten es, alle enthaltenen Prozesselemente durch verschiedene Operationen zu erfragen. Wir wollen ein einfaches Beispiel betrachten und sehen dabei gleich einen der Gründe, die hinter dem Bestreben eines möglichst einfachen Basismodells stehen. Bereits in dem folgenden, einfachen Beispiel erreichen wir schon eine beachtliche Komplexität. Wir betrachten eine Prozesskomponente pk_1:

$$
\begin{array}{lll}
pe & = & \{a, b, c, d, e\} \\
pea & = & \{(a, b), (d, e)\} \\
epea & = & \{(b, \{c, d\})\}
\end{array}
$$

Schnittstelle:

$$
pk_1\,|_G \quad = \quad (\{a, b, c, d, e\}, \{(a, b), (b, \{c, d\}), (d, e)\})
$$

Anders als Softwarekomponenten (vgl. [HRR98, HR02]) benötigen wir für unsere Modellierung kein so genanntes Prinzipalobjekt, da wir keine „lebenden" Speicherobjekte erzeugen. Wird eine Prozesskomponente instanziiert, sind alle enthaltenen Prozesselemente unmittelbar vorhanden und gleichberechtigt. Die Abbildung 4.8 zeigt eine grafische Umsetzung der gerade skizzierten Prozesskomponente pk_1. Die Darstellung orientiert sich schon an der UML, die wir ab Kapitel 5 konsequent nutzen werden.

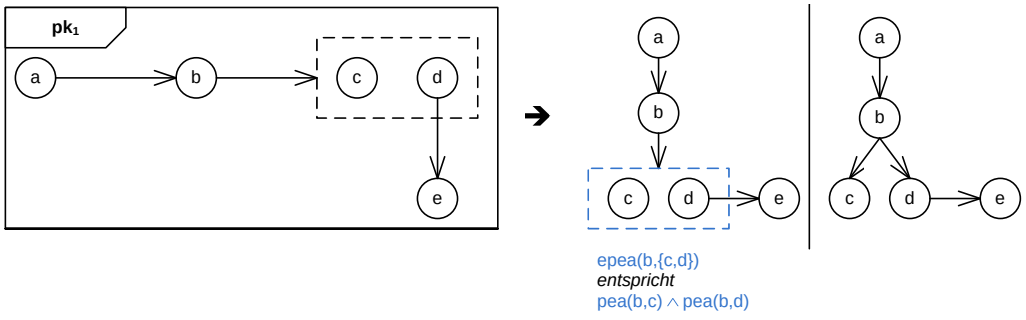


Abbildung 4.8.: Beispiel einer Prozesskomponente und Repräsentation als Graph

Weiterhin zeigen wir mit der Abbildung 4.8 die grafische Interpretation der Relationen PEA und $EPEA$ sowie die Zusammensetzung der Schnittstelle von pk_1. Die Schnittstelle wird hier durch die Menge der Prozesselemente $a, \ldots, e$ und den Prozesselementabhängigkeiten zwischen diesen gebildet.

Weitere Schnittstellentypen. Siedersleben [Sie04] führt neben angebotenen Schnittstellen einer Komponente auch noch *angeforderte* Schnittstellen einer Komponente ein. Mit der UML 2 ist dieses Konzept auch allgemein für die Modellierung verfügbar geworden, weshalb wir uns auch im Kontext dieser Arbeit mit der angeforderten Schnittstelle auseinandersetzen müssen.

Für den Kontext der Prozesskomponenten setzen wir voraus, dass Prozesskomponenten inhaltlich abgeschlossen und vollständig sind. Sie nehmen anders als Softwarekomponenten keine Abstraktion der enthaltenen Strukturen vor, sondern fungieren im Wesentlichen als Container. Setzt eine Prozesskomponente pk_1 nun zwingend eine andere Prozesskomponente pk_2 voraus, heißt das, dass Prozesselemente aus pk_2 benötigt werden, um die vollständige Funktionalität von pk_1 herzustellen. Um den Bedarf auszudrücken, müssen Prozesselementabhängigkeiten formuliert werden, die von einem gegebenen Prozesselement aus pk_1 zunächst auf ein Flag $\perp$ zeigen, dass einen ungültigen Knoten repräsentiert. Dieser Knoten ist nicht Element des Strukturgraphen, sondern eine zu besetzende Position im Abhängigkeitsgraphen.

Auf der abstrakten Ebene des Basismodells ist eine angeforderte Schnittstelle einer Prozesskomponente *pk* somit eine Teilemenge der Prozesselementabhängigkeiten. Für das oben gezeigte Beispiel zeigt Abbildung 4.9 eine Erweiterung um ein angefordertes Prozesselement.

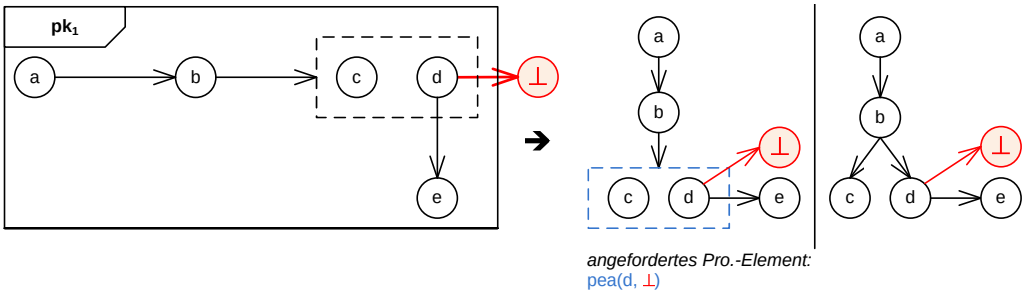


Abbildung 4.9.: Beispiel einer Prozesskomponente mit einem angeforderten Prozesselement

Bei einem angeforderten Prozesselement wie in Abbildung 4.9 gezeigt, modellieren wir für die Prozesskomponente pk_1:

$$
\begin{array}{lll}
 & pe & = \{a, b, c, d, e\} \\
 & pea & = \{(a, b), (d, e), (d, \perp)\} \\
 & epea & = \{(b, \{c, d\})\} \\
\text{Schnittstelle:} & & \\
 & pk_1 \mid_G & = (\{a, b, c, d, e\}, \{(a, b), (b, \{c, d\}), (d, e)\}) \\
 & pk_1 \mid_{G_\perp} & = (\{d\}, \{(d, \perp)\})
\end{array}
$$

Da $\perp$ ein Element des Abhängigkeitsgraphen ist, muss der Knoten passend besetzt werden. Hierzu müssen die beiden nicht zusammenhängenden Abhängigkeitsgraphen G_{pk_1} und G_{pk_2} der beiden Prozesskomponenten pk_1 und pk_2 über die als ungültig definierte Kante $(d, \perp)$ so miteinander verknüpft werden, dass $\perp$ über eine Substitutionsfunktion mit einem gültigen Knoten aus pk_2 belegt wird. Interessant wird dies jedoch erst dann, wenn die Prozesselemente typisiert vorliegen, da dann zum Beispiel weiter gehende Einschränkungen definiert werden können. Im Kapitel 5.1 greifen wir dies wieder auf.

4.2.2. Bildung von Vorgehensmodellen aus Prozesskomponenten

Betrachten wir den üblicherweise sehr hohen Grad an Verflechtung und Integration von Vorgehensmodellen (siehe Kapitel 2 und 3), werden wir ein mehrstufiges Konstruktionsverfahren für Vorgehensmodelle anwenden müssen. Dieses Verfahren muss sich nach den Notwendigkeiten der jeweiligen Organisationen richten. Weiterhin muss das

Verfahren unterscheiden zwischen bestimmten, verbindlich vorgeschriebenen Strukturen und solchen, die in den Anwendungskontexten anpassbar sind. Um einer unmittelbaren Anwendbarkeit von Prozesskomponenten gerecht zu werden, sind die Prozesskomponenten so zu erstellen, dass sie möglichst keine Abhängigkeiten „nach außen" aufweisen. Prozesskomponenten selbst sind also auf der Ebene eines Vorgehensmodells wieder geeignet zu konfigurieren. Die explizite Konfiguration, die das Abhängigkeitsgeflecht liefert, definiert die Aufsetzpunkte für eine Anpassung oder Variation (vgl. Kapitel 2.3.2). Wir verfeinern unter diesem Gesichtspunkt die Definitionen 4.1 und 4.2:

Definition 4.6 (Vorgehensmodell (formal))**:**
Ein Vorgehensmodell $v \in \mathcal{V}$ ist eine Struktur, die sich aus einer Menge von Prozesskomponenten $pk \in Pk$ sowie einer Menge zusätzlicher Relationen zwischen den Prozesselementen der referenzierten Prozesskomponenten zusammensetzt.

Strukturen über Prozesskomponenten

Wir betrachten zunächst die im letzten Abschnitt motivierten Strukturen, die wir über Prozesskomponenten bilden können. Wir unterscheiden bei Prozesskomponentenabhängigkeiten ebenfalls zwischen einer *einfachen* (*pka*) und einer *erweiterten* (*epka*) Prozesskomponentenabhängigkeit. Prozesskomponentenabhängigkeiten sind dabei Relationen über der Menge der Prozesskomponenten, wobei für die einfache Prozesskomponentenabhängigkeit *pka* gilt:

$$\text{PKA} = \{(x, y, Z) \,|x, y \in \text{Pk} \wedge x \neq y \wedge Z \subseteq (\text{PEA} \cup \text{EPEA})\,\} \tag{4.4}$$

Analog gilt für die erweiterte Prozesskomponentenabhängigkeit *epka*:

$$\text{EPKA} = \{(x, Y, Z) \,|x \in \text{Pk} \wedge \emptyset \neq Y \subseteq \text{Pk} \wedge x \notin Y \wedge Z \subseteq (\text{PEA} \cup \text{EPEA})\,\} \tag{4.5}$$

Wir treffen die Annahme, dass beiden Prozesskomponenten bekannt ist, dass sie miteinander in Beziehung stehen. Eine Menge von Prozesselementabhängigkeiten, die zwischen den Prozesselementen zweier oder mehrerer verschiedener Prozesskomponenten existiert, muss die verfügbare Menge von Prozesselementen bekannt gemacht werden. Werden also Abhängigkeitsrelationen über Prozesselementen so gebildet, das Abhängigkeiten zwischen Prozesselementen verschiedener Prozesskomponenten entstehen, muss zwischen den betroffenen Prozesskomponenten ebenfalls eine Abhängigkeitsrelation existieren. Bei der Konfiguration von Prozesskomponenten entstehen im Abhängigkeitsgraphen des Vorgehensmodells/der Vorgehensmodellvariante *keine* neuen Knoten, sondern ausschließlich neue Kanten, um die Abhängigkeitsgraphen der Prozesskomponenten miteinander zu verbinden.

Unterschiedlich ist jedoch an dieser Stelle die Semantik der Abhängigkeitsrelationen, obwohl die Syntax ähnlich ist. Wir lesen beispielsweise *pea* (x, y) derart, das y von x abhängt. Für eine Prozesskomponentenabhängigkeit *pka* (pk_2, pk_1) sagen wir aber, dass pk_2 eine Abhängigkeit zu pk_1 (zum Beispiel pk_2 basiert auf pk_1, siehe Kapitel 5.1.3) hat. Wir geben ein kurzes Beispiel an. Wir betrachten wieder eine Prozesskomponente pk_1:

$$\begin{array}{lll} & \textit{pe} & = \{a, b, c, d, e\} \\ & \textit{pea} & = \{(a, b), (d, e)\} \\ & \textit{epea} & = \{(b, \{c, d\})\} \\ \text{Schnittstelle:} & & \\ & \textit{pk}_1\,|_G & = (\{a, b, c, d\}, \{(a, b), (b, \{c, d\}), (d, e)\}) \end{array}$$

und eine zweite Prozesskomponente pk_2:

$$\begin{array}{lll} & \textit{pe} & = \{e, f, g, h, i, j, k, s\} \\ & \textit{pea} & = \{(e, k), (e, i), (h, g), (j, g), (k, s), (s, g)\} \\ & \textit{epea} & = \{(e, \{f, h, j\})\} \\ \text{Schnittstellen:} & & \\ & \textit{pk}_2\,|_G & = \{\{e, f, g, h, i, j, k, s\}, \\ & & \quad \{(e, k), (e, i), (e, \{f, h, j\}), (h, g) \\ & & \quad (j, g), (k, s), (s, g)\}\} \end{array}$$

In Abbildung 4.10 ist die grafische Modellierung der beiden Prozesskomponenten zu sehen. Die Abhängigkeitsgraphen $pk_1\,|_G$ und $pk_2\,|_G$ sollen nun miteinander verknüpft werden. Als Koppelstelle wählen wir die beiden Knoten e und s. In der Abbildung 4.11 ist das noch einmal auf den „puren" Graphen gezeigt.

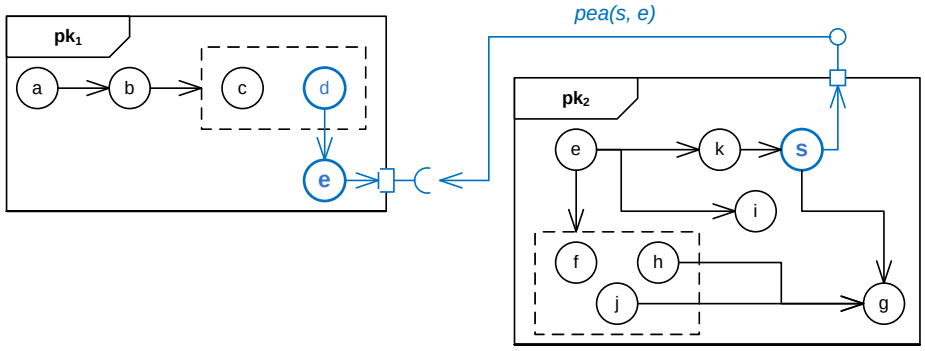


Abbildung 4.10.: Beispiel einer Prozessekomponente mit einer erfüllten offenen Prozesselementabhängigkeit

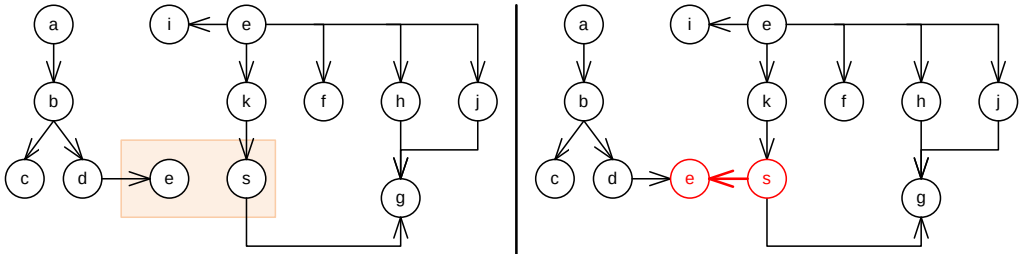


Abbildung 4.11.: Verknüpfung der Graphen zweier Prozesskomponenten durch Belegung der offenen Abhängigkeiten

Die Kopplung erfolgt durch das Hinzufügen einer Kante im Graphen. Es wird also bei der Komposition der beiden Prozesskomponenten wie folgt vorgegangen:

$\mathrm{G}_v = \mathrm{G}_{\mathrm{pk}_1} \cup \mathrm{G}_{\mathrm{pk}_2}$ und Hinzufügen der neuen Kante, also:

$$\mathrm{G}_v = \left(\left(Pe_1 \cup Pe_2 \right), \left(PEA_1 \cup EPEA_1 \cup PEA_2 \cup EPEA_2 \cup pea\,(s,e) \right) \right) \tag{4.6}$$

Zwischen pk_1 und pk_2 existiert somit eine Abhängigkeit $pka\left(pk_2, pk_1\right)$, die zunächst eine Vereinigungsoperation der beiden Graphen zur Folge hat, die die Konfigurationen von pk_1 und pk_2 repräsentieren. Zusätzlich wird noch eine Verbindung zwischen zwei Prozesselementen durch $pea\,(s,e) : e \in Pe_1 \wedge s \in Pe_2$ hergestellt. Diese neu entstandene Kante verknüpft die beiden Teilgraphen $pk_1\,|_G$ und $pk_2\,|_G$ und ist Teil der Konfiguration des Vorgehensmodells beziehungsweise der Vorgehensmodellvariante.

Prozesskomponenten und Bildung von Vorgehensmodellen

In den letzten Abschnitten haben wir bereits die Grundlage für die Bildung von Vorgehensmodellen gelegt. Definition 4.6 sagt aus, dass sich ein Vorgehensmodell aus einer Menge von Prozesskomponenten zusammensetzt. Über die Vereinigung der Abhängigkeitsgraphen bestimmen wir die (inhaltlichen) Strukturen eines Vorgehensmodells. Wir verallgemeinern zunächst ein Vorgehensmodell als vereinigten Graphen über n-Prozesskomponenten.

$$pk_i\,|_G = \left(Pe_i, \left(PEA \cup EPEA \right)_i \right)$$

$$G_{VgM} = \bigcup_{1 \leq i \leq n} pk_i\,|_G \tag{4.7}$$

Die so vereinigten Graphen sind zunächst nicht zusammenhängend. Bei der Zusammenstellung eines Vorgehensmodells sind daher noch entsprechende Verknüpfungen zwischen den Teilgraphen herzustellen. Im letzten Abschnitt haben wir das prinzipiell am Beispiel von nur zwei Prozesskomponenten betrachtet. Im Kontext eines Vorgehensmodells ist die Verknüpfung der Prozesselemente und -komponenten jedoch essenziell, da erst dadurch die benötigte Funktionalität hergestellt werden kann. Im Folgenden betrachten wir dies an einem einfachen Beispiel. Abbildung 4.12 zeigt die vier Prozesskomponenten pk_1, pk_2, pk_3 und pk_4, die wir in einem Vorgehensmodell zusammenführen wollen. Wir haben in den Prozesskomponenten auch ausgezeichnete Prozesselemente α, β und γ, über die wir Verknüpfungen herstellen wollen.

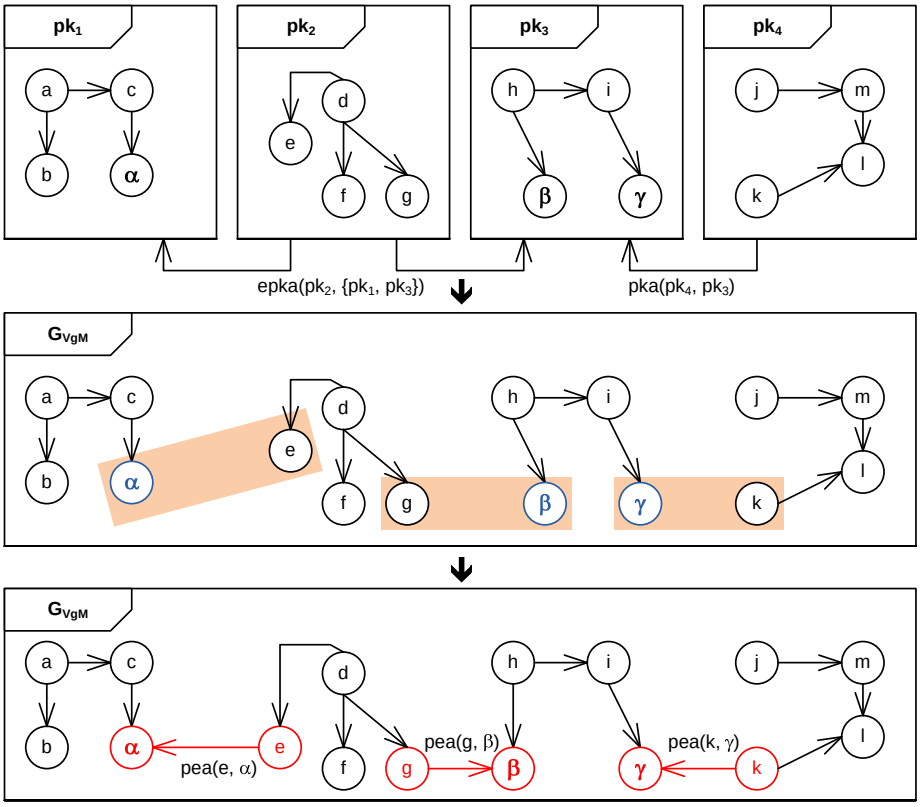


Abbildung 4.12.: Allgemeine Verknüpfung der Graphen von Prozesskomponenten zur Zusammenstellung eines Vorgehensmodells

Der erste Schritt, der zu gehen ist, ist die Aufstellung der Prozesselementabhängigkeiten (*pka* und *epka*). Durch deren Feststellung wird die korrespondierende Vereinigung der Abhängigkeitsgraphen vorgenommen. Abschließend sind neue Kanten an den Koppelstellen[5] zu etablieren. In Abbildung 4.12 sind die einzelnen Schritte, die wir gerade angesprochen haben noch einmal zusehen. Bereits in Abbildung 4.6 haben wir die stufenweise *Konfiguration* besprochen, die einmal auf der Ebene 1 der Prozesskomponenten und noch einmal auf der Ebene 2 des Vorgehensmodells stattfindet. Um dies zu verdeutlichen greifen wir auf das Beispiel aus Abbildung 4.12 zurück und formalisieren es entsprechend unserem Modell. Wir betrachten zunächst die Prozesskomponenten $\{pk_1, pk_2, pk_3, pk_4\}$, die wir in unserem Vorgehensmodell zusammenfassen wollen.

5 Es ist auch möglich, Prozesskomponenten im einem Vorgehensmodell mit einzubinden, die keine Beziehungen/Abhängigkeiten zu anderen Prozesskomponenten haben. Der resultierende Abhängigkeitsgraph ist dann nicht zusammenhängend.

Gemäß Abbildung 4.12 sind die Prozesskomponenten wie folgt zu modellieren:

$$
\begin{array}{rll}
pk_1\text{:} & & \\
 & \mathit{pe} & = \{a, b, c, \alpha\} \\
 & \mathit{pea} \cup \mathit{epea} & = \{(a, b), (b, c), (c, \alpha)\} \\
\text{Schnittstellen:} & & \\
 & \mathit{pk}_1 |_G & = (\{a, b, c\}, \{(a, b), (b, c), (c, \alpha)\}) \\
pk_2\text{:} & & \\
 & \mathit{pe} & = \{d, e, f, d\} \\
 & \mathit{pea} \cup \mathit{epea} & = \{(d, \{e, f, g\})\} \\
\text{Schnittstellen:} & & \\
 & \mathit{pk}_2 |_G & = (\{d, e, f, g\}, \{(d, \{e, f, g\})\}) \\
pk_3\text{:} & & \\
 & \mathit{pe} & = \{h, i, \beta, \gamma\} \\
 & \mathit{pea} \cup \mathit{epea} & = \{(h, i), (h, \beta), (i, \gamma)\} \\
\text{Schnittstellen:} & & \\
 & \mathit{pk}_3 |_G & = (\{h, i\}, \{(h, i), (h, \beta), (i, \gamma)\}) \\
pk_4\text{:} & & \\
 & \mathit{pe} & = \{j, k, l, m\} \\
 & \mathit{pea} \cup \mathit{epea} & = \{(j, m), (k, l), (m, l)\} \\
\text{Schnittstellen:} & & \\
 & \mathit{pk}_4 |_G & = (\{j, k, l, m\}, \{(j, m), (k, l), (m, l)\})
\end{array}
$$

In den Prozesskomponenten pk_1 und pk_3 sind mit den drei Prozesselementen α, β und γ Prozesselemente vorhanden, die wir mit anderen Prozesselementen in Verbindung bringen wollen. Diejenigen Prozesskomponenten, die die Prozesselemente enthalten, mit denen α, β und γ in Beziehung gesetzt werden sollen, verursachen die Prozesskomponentenabhängigkeiten. Wir können unser Vorgehensmodell nun zunächst wie folgt angeben:

$$
\begin{array}{rrll}
\text{Vorgehensmodell:} & & & \\
\text{Prozesskomponenten:} & Pk & = & \{\mathit{pk}_1, \mathit{pk}_2, \mathit{pk}_3, \mathit{pk}_4\} \\
\text{Prozesskomponentenabhängigkeiten:} & & & \mathit{epka}\left(\mathit{pk}_2, \{\mathit{pk}_1, \mathit{pk}_3\}, \right. \\
 & & & \left. \{\mathit{pea}\,(e, \alpha), \mathit{pea}\,(g, \beta)\}\right) \\
 & & & \mathit{pka}\left(\mathit{pk}_4, \mathit{pk}_3, \{\mathit{pea}\,(k, \gamma)\}\right)
\end{array}
$$

In Abbildung 4.13 sind passend zu diesem Beispiel die Stellen der Konfiguration angegeben. Hier ist zu sehen, dass die Konfiguration einer Prozesskomponente intern ist. Sie sorgt dafür, dass eine Prozesskomponente vollständig beschrieben wird. Über die Schnittstellen gibt eine Prozesskomponente ihr Leistungsangebot an.

Vorteile konfigurativ konstruierter Vorgehensmodelle. Die Erfahrung, insbesondere durch Anwendung und Anpassung von V-Modell XT und MSF zeigt, dass konfigurationsbasierte Konstruktionsverfahren Vorteile gegenüber hoch integrierten aufweisen. Die Flexibilität – insbesondere für den Endanwender – ist hierbei besonders zu nennen. Hervorzuheben bei Vorgehenmodellen, die nach unserem Ansatz erstellt wurden, sind dabei folgende Eigenschaften:

- Durch die Prozesskomponentenabhängigkeiten kann die Menge der benötigten Prozesskomponenten mittels Hüllenbildung ermittelt und referenziert werden. Struktur und Umfang eines Vorgehensmodells/einer Vorgehensmodellvariante sind dadurch berechenbar.
- Nachdem alle relevanten Prozesskomponenten $pk_1, \ldots, pk_n$ referenziert worden sind, steht der gesamte Inhalt aller Prozesskomponenten homogen zur Verfügung.
- Da die Prozesskomponenten eigene, interne (gültige) Konfigurationen besitzen, können sie modular in ein Vorgehensmodell eingebunden werden. Durch die definierten Schnittstellen und das eigenständige Kompositionsverfahren auf der Ebene des Vorgehensmodells/der Variante können Prozesskomponenten nahezu beliebig in einem Vorgehensmodell integriert werden. Die Konfiguration der einzelnen Prozesskomponenten im Gesamtmodell ist transparent und darf durch das

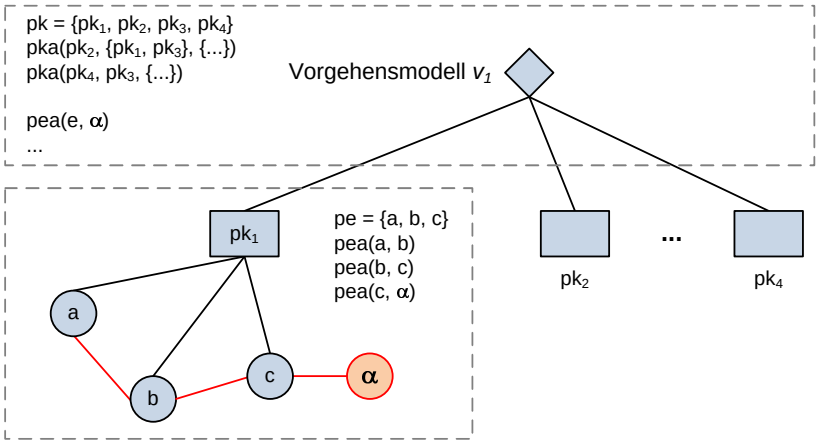


Abbildung 4.13.: Integrationsebenen eines Vorgehensmodells mit Konfiguration enthaltener Elemente

Vorgehensmodell nicht verändert werden. Dadurch wird eine Austauschbarkeit einzelner Prozesskomponenten ermöglicht.

- Aufbauend auf der letzten Eigenschaft: Die einzelnen Prozesskomponenten sind für das Vorgehensmodell transparent. Sofern zwischen zwei Prozesskomponenten Schnittstellenkonformität vorliegt, sind sie dynamisch austauschbar.
- Syntaktisch und strukturell kann der Abhängigkeitsteilgraph auf der Integrationsebene 2 (Vorgehensmodell) auf Gültigkeit geprüft werden. Die Prüfung kann erfolgen sowohl für die Feststellung der ordnungsgemäßen (Initial-)Komposition als auch als Prüfmaßnahme während/nach einer Anpassung oder Aktualisierung.

Gerade mit den letzten beiden Punkten bewegen wir uns zunehmend im Bereich der Evolution von Vorgehensmodellen. Dieser Problematik widmen wir uns im folgenden Kapitel. Zuvor wollen wir die hier vorgestellten Konzepte am Beispiel prüfen.

4.2.3. Anwendung an einem Beispiel

Die in diesem Kapitel vorgestellten, abstrakten Konzepte wollen wir abschließend noch auf ein reales Anwendungsbeispiel anwenden. Abbildung 4.14 greift in einem Ausschnitt auf den Vorgehensbaustein *Projektmanagement* des V-Modell XT zurück, den wir mit unseren Mitteln remodellieren wollen. Bis hierher stehen uns dafür nur Prozesselemente, Prozesselementabhängigkeiten und Prozesskomponenten, jedoch noch keinerlei weiter gehende Typisierung zur Verfügung. Nichtsdestotrotz können wir an dieser frühen Stelle bereits die Eignung für die Modellierung von Strukturen prüfen.

Im ersten Abschnitt von Abbildung 4.14 zeigen wir die hierarchische Komposition im Struktrugraphen S. Einzelne Prozesselemente werden unter einer Prozesskomponente zusammengefasst. Wie bereits besprochen, werden alle Prozesselemente gleichberechtigt behandelt – es gibt keine Hierarchien zwischen ihnen. Im zweiten Abschnitt ist der Abhängigkeitsgraph G zu sehen, der über der flachen Hierarchie der Prozesselemente in der Prozesskomponente eine weiter gehende, inhaltliche Struktur modelliert. Da wir für G fordern, dass es sich um einen gerichteten Graph handelt, haben wir in Abbildung 4.14 die Richtung zwischen den Paaren an den Kanten angegeben. Wir sehen weiterhin auch, dass wir alle Abhängigkeiten bereits auf die einfache Prozesselementabhängigkeit reduziert haben. Anwendungsbeispiele für die vereinfachte Modellierung mithilfe der erweiterten Prozesselementabhängigkeit sind jedoch auch zu finden. Im letzten Abschnitt werden wir etwas konkreter und greifen in Teilen schon etwas Kapitel 5 vor. Der letzte Abschnitt der Abbildung 4.14 zeigt die Prädikatendarstellung der Prozesselemente, wie sie zum Beispiel zum Aufbau einer Wissensbasis mit angeschlossenem Regelsystem verwendet werden kann. Die Prozesselemente liegen hier bereits ebenso wie die Prozesselementabhängigkeiten in typisierter und konkretisierter Form

vor. Bezug nehmend auf das V-Modell XT verwenden wir hier zunächst noch seine Terminologie, solange bis wir im folgenden Kapitel 5.1 eine konkrete Syntax und Semantik angeben.

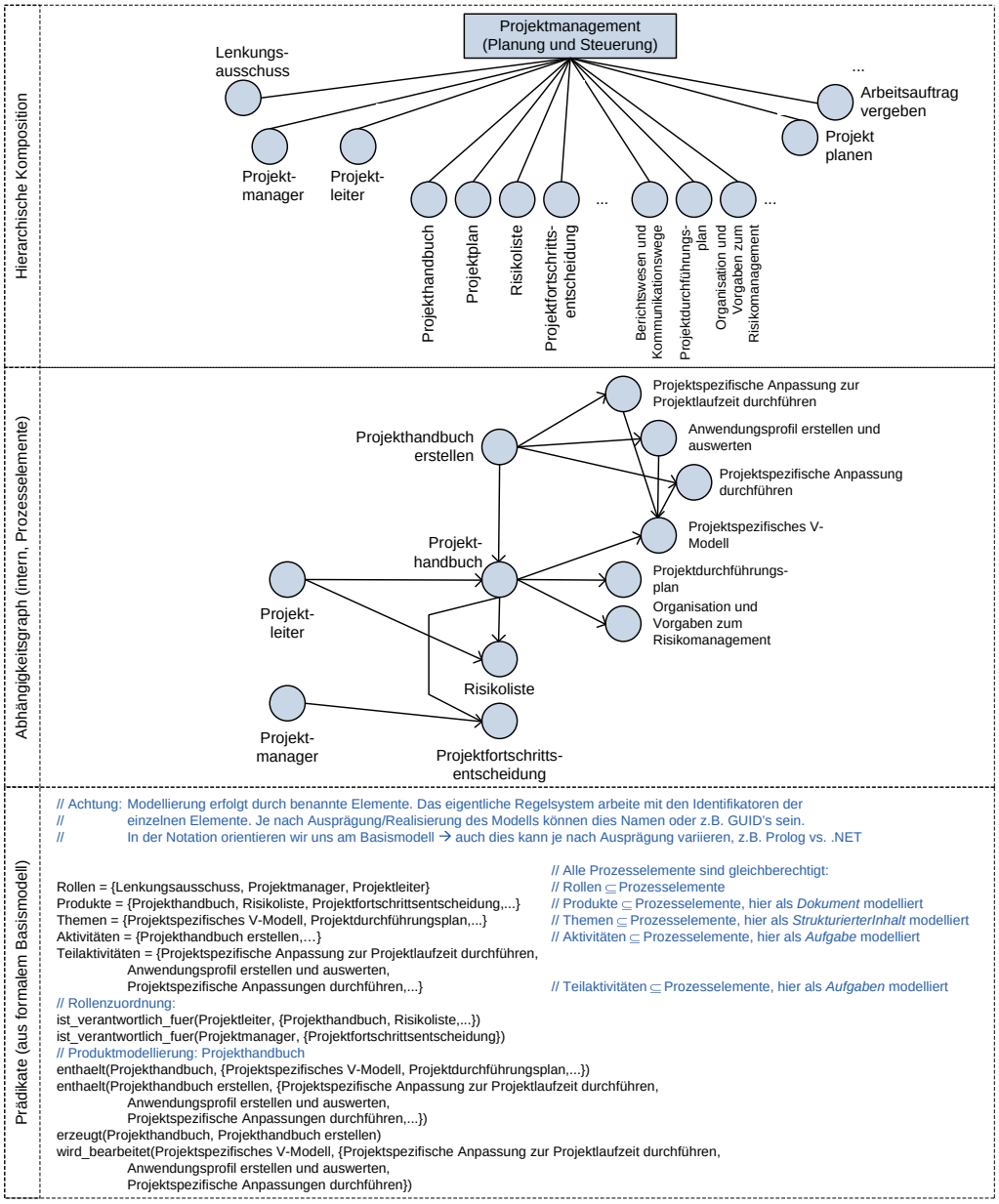


Abbildung 4.14.: Beispielhafte Anwendung im Kontext V-Modell XT

Diese Art der Modellierung ist bereits möglich, jedoch nicht sehr „anwenderfreundlich". Dafür ist eine entsprechende Verfeinerung der hier modellierten Konzepte erforderlich, die wir in Kapitel 5.1 vornehmen. Ebenfalls noch nicht ersichtlich sind die Mehrwerte im Kontext eines Entwicklungsstandards, da wir in diesem Beispiel noch keine Konfiguration auf der Ebene von Prozesskomponeten und darüber hinaus auch Vorgehensmodellvarianten angeben. Dafür verweisen wir auf die verfeinerten Teile und Anwendungsbeispiele im Kapitel 6 und Anhang B.

4.3. Evolution von Prozesselementen und -komponenten

In diesem Abschnitt befassen wir uns mit dem Weiterentwicklungspotenzial von Vorgehensmodellen. Wir betrachten insbesondere die Versions- und Variantenbildung (vgl. Abschnitt 4.1) von Vorgehensmodellen. Im Anschluss geben wir Beispiele an und betrachten unseren Ansatz und Fragestellungen der Variabilität eines Vorgehensmodells.

4.3.1. Abstrakter Konstruktionsprozess

Um die Inhalte dieses Abschnitts besser positionieren zu können, geben wir zunächst noch einen zusammenfassenden, zunächst *abstrakten Konstruktionsprozess* an. Dieser spiegelt die die Anwendung der bisher erarbeiteten Konzepte wider und positioniert gleichzeitig die in diesem Abschnitt behandelten Themen zur Versions- und Variantenbildung von Vorgehensmodellen im Kontext eines Entwicklungsstandards.

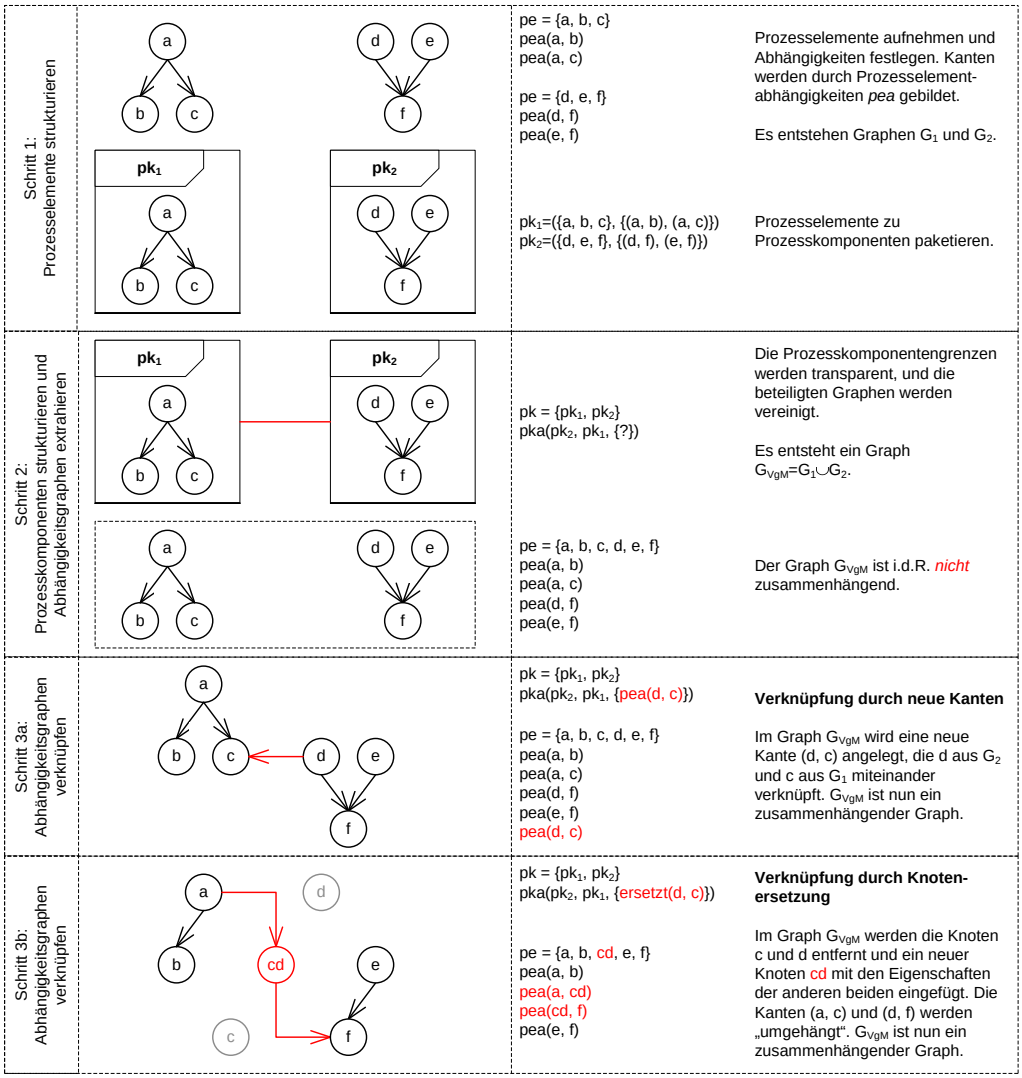


Abbildung 4.15.: Abstrakter Konstruktionsprozess für Vorgehensmodelle

Bereits in den Abschnitten 4.1 und 4.2 sind wir auf die Grundlagen von Prozesselementen und Prozesskomponenten sowie die Beziehungstypen zwischen diesen eingegangen. Die Abbildung 4.15 strukturiert diese in einem kompakten Prozess. Teile dieses

Prozesses haben wir bereits im Abschnitt 4.2.2 verwendet. Im Folgenden beschreiben wir diesen Prozess und gehen dabei gleichzeitig schon auf die Operationen ein, die im Kontext der Versions- und Variantenbildung relevant sind. Der Prozess, den wir angeben ist im Wesentlichen in drei Schritte aufgeteilt.

Schritt 1 dient der Ermittlung, Strukturierung und Paketierung von Prozesselementen. Hier werden also später durch Prozessingenieure Gegenstände der realen Welt so modelliert, das *Prozesskomponenten* entstehen. In Abbildung 4.15 zeigt der Schritt 1 gleichzeitig die graphentheoretische Interpretation. Betrachten wir Gegenstandsmengen der Domäne (Artefakte, Rollen etc.), entstehen zunächst *Prozesselemente pe* – als Knoten im späteren Abhängigkeitsgraphen. Abhängigkeiten, also die Kanten modellieren wir aus Paaren (pe_i, pe_j) (vgl. Abschnitt 4.1.2). Als weiterer Strukturierungsschritt werden (eigenständige) Graphen über *disjunkten* Teilmengen der Prozesselemente und den korrespondierenden Abhängigkeiten gebildet, sodass bezogen auf das Beispiel in Abbildung 4.15 zwei Graphen G_1 und G_2 entstehen. Jeder dieser Graphen repräsentiert eine individuelle Prozesskomponente.

Im *Schritt 2* werden die Prozesskomponenten miteinander verknüpft. Diese Verknpüfung ist erforderlich, um aus einzelnen Prozesskomponenten einen zusammenhängenden, integrierten Prozess zu formen. In diesem Schritt werden über so genannte *Prozesskomponentenabhängigkeiten* diejenigen Prozesskomponenten bestimmt, die für die Zusammenführung infrage kommen. Graphentheoretisch stellt insofern jede ausgewählte Prozesskomponente einen *Teilgraphen* eines integrierten Prozesses dar. Im Schritt 2 wird daher zunächst die Vereinigung der Teilgraphen vorgenommen. Bezogen auf das Beispiel in Abbildung 4.15 bedeutet das: $G_{VgM} = G_1 \cup G_2$. Für *n*-Teilgraphen analog: $G_1 \cup G_2 \cup \ldots \cup G_{n-1} \cup G_n$. Da wir aufgrund der Konstruktion der Prozesskomponenten annehmen können, dass die Mengen der enthaltenen Prozesselemente disjunkt ist (analog die Kantenmengen in den jeweiligen Teilgraphen), gilt auch: $G_1 \cap G_2 \cap \ldots \cap G_{n-1} \cap G_n = \emptyset$. Der resultierende Vereinigungsgraph G_{VgM} ist an dieser Stelle also *nicht zusammenhängend*. Ausnahmetatbestände finden wir dort, wo Prozesskomponenten bereits über ein eigenes, externes Abhängigkeitsgeflecht verfügen. In Kapitel 5.1.3 geben wir explizit entsprechende Abhängigkeitstypen an. In einem solchen Fall muss sichergestellt sein, dass keine Prozesskomponenten doppelt vorkommen und dass bereits existierende Abhängigkeiten korrekt aufgelöst werden. Konkret bedeutet das, dass wenn eine Prozesskomponente pk_1 in einer solchen Konstruktion eine Abhängigkeit zur Prozesskomponente pk_2 besitzt und pk_1 im Rahmen einer Konfiguration von Prozesskomponenten verwendet werden soll, dass auch pk_2 berücksichtigt werden muss. Soll weiterhin eine Prozesskomponente pk_3 verwendet werden und besitzt pk_3 auch eine Abhängigkeit zu pk_2, so darf pk_2 *nicht* noch einmal in die Konfiguration mit aufgenommen werden. Vielmehr muss (zum Beispiel durch Sicherstellung der Identität) geährleistet werden, dass pk_3 mit der bereits integrierten Instanz von pk_2 in Beziehung gesetzt wird. Hierbei ist dann im Rahmen der tatsächlichen Konstruktion die Konsistenz sicher zu stellen.

Die folgenden *Schritte 3a* und *3b* haben wir vom Schritt 2 separiert, obwohl sie sehr dicht mit ihm verflochten sind. Nachdem wir im Schritt 2 den nicht (zwangsweise) zusammenhängenden Vereinigungsgraphen gebildet haben, müssen wir nun die Verknüpfung der jeweiligen, isolierten Graphenteile realisieren. Wir sehen in diesem Schritt im Wesentlichen zwei Wege zur Verknüpfung der einzelnen Graphteile vor:

1. Verknüpfung der Graphen durch neue Kanten (Schritt 3a)
2. Verknüpfung der Graphen durch Knoten- und Kantenersetzung (Schritt 3b)

Für Operation nach Schritt 3a genügen bereits die in Abschnitt 4.1.2 eingeführen Prozesselementabhängigkeiten. Im Schritt 3a werden *keine* neuen Knoten im Graph G_{VgM} erzeugt – nur neue Kanten. Die neuen Kanten sind dabei bereits im Kontext konkreter Prozesskomponentenabhängigkeiten definiert (siehe Abschnitt 4.2.2) oder können noch neu im Rahmen der Vorgehensmodell- oder allgemein Prozesskonfiguration hinzugefügt werden. Da es sich hierbei bereits um konkrete Beziehungen handelt, sofern sie in Prozesskomponentenabhängigkeiten definiert sind, ist deshalb auch die Einbindung bereits existierender Abhängigkeitsgeflechte unter Sicherstellung der Einzigkeit der Elemente und Eindeutigkeit der Beziehungen in der Regel unproblematisch. Für

den Schritt 3a nehmen wir explizit keine Änderung an existierenden Knotenmengen vor. Vorgehensmodelle können, sofern also Standardmechanismen und -inhalte genügen, durch einfaches Auswählen benötigter Prozesskomponenten gebildet werden. Grundlegende Abhängkeitsstrukturen werden dabei ggf. durch die Prozesskomponentenabhängigkeiten bereits zur Verfügung gestellt beziehungsweise durch eine ensprechende Konfiguration des Prozesses festgelegt. Für den Schritt 3b ist das komplizierter, da hier die Menge der Knoten im Abhängigkeitsgraph ggf. angepasst wird (im Beispiel von Abbildung 4.15 ist das bei den Knoten *c* und *d* zu sehen, die aufgrund einer Operation aus dem resultierenden Graphen entfallen und durch einen neuen Knoten *cd* ersetzt); analog die Kantenmenge, die nicht nur erweitert, sondern auch reduziert und umgestaltet werden kann. Schritt 3b kombiniert somit strukturelle Komposition von Graphen mit deren Rekonfiguration. Wir verweisen für weitere Ausführungen dazu auf den Abschnitt 4.3.3, wo wir mit den *Variabilitätsoperationen* eine konkrete Anwendung vorliegen haben. Wir erläutern die diesbezüglichen Fragestellungen dort vor Ort.

4.3.2. Versions- und Variantenbildung

Wir wollen zunächst noch einmal den Problembereich explizieren und beziehen uns dabei wieder auf den Abschnitt 4.1. Am Beispiel von Abbildung 4.16 betrachten wir einen beliebigen Entwicklungsstandard. Dieser ist unser Ausgangspunkt für die weitere Betrachtung. Gemäß den Definitionen der Produktlinienentwicklung (vgl. Kapitel 2.3.2) gehen wir von einem proaktiv entwickelten Standard aus, aus dem verschiedene (konkrete) Varianten abgeleitet werden und der gleichzeitig die Basis für die Weiterentwicklung und Pflege darstellt. Die Abbildung 4.16 greift das bereits mehrfach verwendete Schema dieses Kapitels auf und ergänzt es um einen Zeitstrahl und eine Dimension, die wir als *Variantenraum* bezeichnen. Auf der obersten Hierarchieebene sind die Releases zu finden. Zum Zeitpunkt t_0 finden wir das Release VM_0, das als Grundlage für organisationsspezifische Anpassungen (Tailoring Stufe 1) dient. VM_0 wird durch zwei Varianten $VM_{0,1}$ und $VM_{0,2}$ spezialisiert. Im Kontext V-Modell XT positionieren wir auf dieser Ebene die Ausprägungen V-Modell-Bayern und V-Modell Bw. Die abgeleiteten Varianten können im Rahmen einer projektspezifischen Anpassung (Tailoring Stufe 2) weiter spezialisiert werden. Im Beispiel aus Abbildung 4.16 wird beispielsweise $VM_{0,1}$ mittels projektspezifischer Anpassung durch $VM_{0,1'}$ und $VM_{0,1''}$ spezialisiert.

Dieser gerade skizzierte Anpassungs- und Spezialisierungsprozess findet seinen Ursprung in genau einem Element – dem Vorgehensmodellrelease VM_0. VM_0 wiederum basiert auf den Vorgaben des Entwicklungsstandards *v1*. Wir müssen nun an dieser Stelle mehrere mögliche Pfade für die Weiterentwicklung betrachten. Alle Pfade wirken sich dabei auch auf die durch Tailoring angepassten Varianten aus. Den ersten möglichen Weiterentwicklungspfad finden wir beim Übergang von VM_0 zum Zwischenrelease $VM_{0'}$. Das Zwischenrelease basiert ebenfalls auf dem Entwicklungsstandard *v1* und entsteht beispielsweise durch Bugfixing oder Rückflüsse/Feedback aus Projekten, die mit einer Instanz von VM_0 (oder spezieller) durchgeführt wurden. Konkret finden wir eine Spezialisierung $VM_{0',1}$, von der wir annehmen, dass sie Fehler von $VM_{0,1}$ behebt. Die Art der Fehler (zum Beispiel Schwere, Tragweite etc.) haben hierbei auch Einfluss darauf, ob und wie die Korrekturen beziehungsweise Änderungen allen anderen Instanzen von $VM_{0,1}$ verfügbar gemacht werden. Dies greifen wir im Anschluss noch einmal auf. Gemäß dem PLE-Referenzprozess aus Kapitel 2.3.2, auf dem wir uns hier auch abstützen, müssen wir die Erfahrungen und das (neu gewonnene) Wissen wieder in die Weiterentwicklung und Pflege des Entwicklungsstandards einfließen lassen (Kante von $VM_{0'}$ nach Entwicklungsstandard *v2*). Dies stellt den zweiten möglichen Pfad für die Weiterentwicklung ausgehend vom Entwicklungsstandard *v1* dar. Diese Weiterentwicklung erfolgt in der Regel zeitlich parallel zur Anpassung, Instanziierung und Pflege der Varianten aus *v1*. Zu einem definierten Zeitpunkt müssen entsprechende Änderungen und Korrekturen in die Entwicklung des Entwicklungsstandards mit aufgenommen werden. Basierend auf dem neuen Entwicklungsstandard wird ein neues VM_1 erstellt. Aus diesem lassen sich analog wieder mittels organisations- und projektspezifischer Anpassung verschiedene Varianten erstellen.

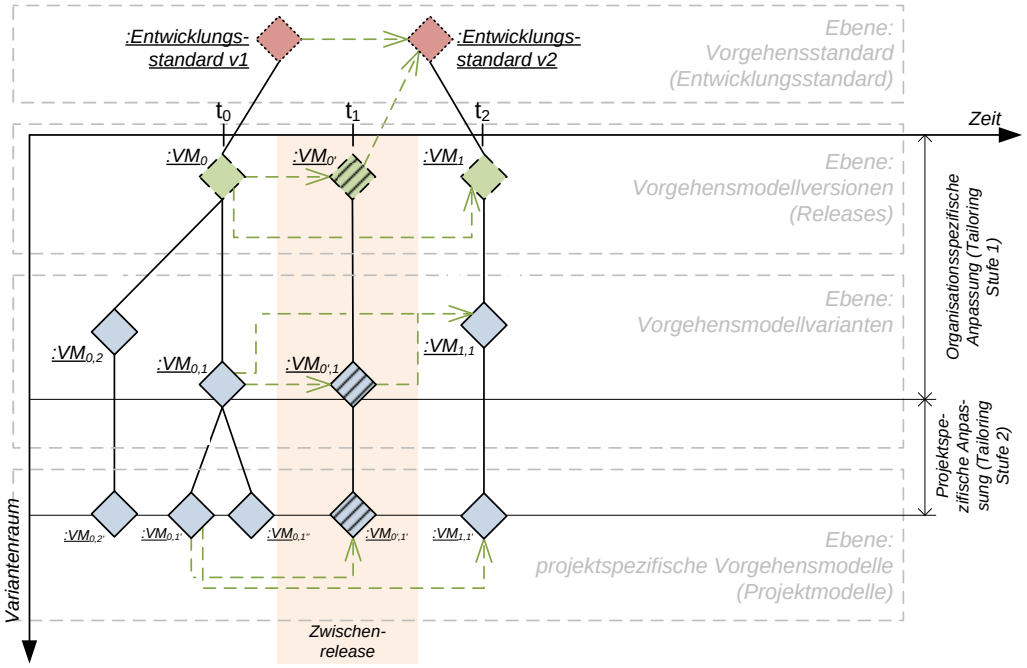


Abbildung 4.16.: Beispiel der Versions- und Variantenbildung von Vorgehensmodellen

Zu beachten ist an dieser Stelle jedoch das Verhalten bestimmter Varianten über verschiedene Releases hinweg. Wir beziehen uns auf die Variante $VM_{0,1'}$, die folgendem Spezialisierungspfad entstammt:

$$VM_{0,1'} \leftarrow VM_{0,1} \leftarrow VM_0 \leftarrow \text{Entwicklungsstandard v1}$$

Wir stellen nun den Zusammenhang zur Variante $VM_{1,1'}$ her, die nach der Weiterentwicklung von *v1* zur Version *v2* des Entwicklungsstandards wie folgt gebildet wird:

$$VM_{1,1'} \leftarrow VM_{1,1} \leftarrow VM_1 \leftarrow \text{Entwicklungsstandard v2}$$

Änderungen/Anpassungen, die auf der obersten Ebene, also der des Entwicklungsstandards, gemacht werden, haben möglicherweise Auswirkungen auf die spezialisierten Varianten. Die Tragweite und konkrete Ausgestaltung dieser Änderungen wird beispielsweise durch [SM06c] mithilfe des Δ-Ansatzes ermittelt. Konstruktiv bedeutet die Änderung oder Anpassung von Komponenten auf der Ebene des Entwicklungsstandards, dass sich Strukturen ändern. Die Änderungen haben ihre Ursache gemäß unserer Festlegungen aus Kapitel 2 entweder in Änderungen am zugrunde liegenden Metamodell oder durch Anpassungen auf der Modellebene. Änderungen am Metamodell schlagen sich dabei in der Regel auch auf der Modellebene nieder.

Beispiel: Um für diesen Kontext ein fassbares Beispiel anzubieten, beziehen wir uns auf das V-Modell XT und dort insbesondere auf den Übergang vom Release 1.1 auf das Release 1.2. Zwischen diesen beiden Releases wurde der den Projektdurchführungsstrategien zugrunde liegende Metamodellteil geändert. Es wurden zum Beispiel definierte Konzepte für Parallelabläufe und Multiprojektmanagement ergänzt. Einerseits waren die Änderungen an den Projektdurchführungsstrategien durch die Anforderung Multiprojektmanagement unterstützen zu können notwendig geworden, andererseits schlugen sich die Änderungen derart durch, dass die Projektdurchführungsstrategien stärker modularisiert werden konnten. Allen Auftragnehmer-Projektdurchführungsstrategien steht nun optional ein Ablaufbaustein für die Unterauftragsvergabe zur Verfügung, der dynamisch eingebunden werden kann. Über das neue Konzept des Parallelablaufs auch beliebig oft.

Wir finden mit dem gerade aufgeführten Beispiel einen typischen Weiterentwicklungs- oder Evolutionszyklus. Aus der Praxis wurde eine Anforderung aufgenommen, die zunächst pragmatisch und nur für den aktuellen Kontext umgesetzt wurde. Im Folgenden

wurde diese Anforderung in die Weiterentwicklung des Entwicklungsstandards mit aufgenommen und dort umgesetzt. Aufgrund einer entsprechenden Weiterentwicklung des Metamodells ergaben sich Änderungen auch auf der Modellebene. Inhaltlich war der Standard dabei vergleichsweise stabil (Produkt- oder Aktivitätsmodelle), während die neue Optionen sich auf das ganze Modell und alle Varianten auswirken.

Referenzmodell und Hauptstrang. Da wir alle Anpassungen aus Abbildung 4.16 bis zum Zeitpunkt t_1 auf die Version VM_0 zurückführen können, bezeichnen wir VM_0 hier als *Referenzmodell*. Ferner betrachten wir nach Abbildung 4.16 die Weiterentwicklung von VM_0 zu VM_1 als gleichberechtigt, womit wir auch VM_1 als Referenzmodell bezeichnen. Von den Referenzmodellen ausgehend bilden wir die Varianten von Vorgehensmodellen. Wir bezeichnen diesen Entwicklungsstrang als *Hauptstrang*. Dieser Hauptstrang gestattet es:

- die „Urversion" eines Vorgehensmodells zu bestimmen.
- einen Referenzentwicklungsprozess zu etablieren.
- ein Referenzmodell als Bezugspunkt für Anpassung und Prüfung zu definieren.

Aufbauend auf den Modellierungen des letzten Abschnitts definieren wir nun Vorgehensmodellversionen und Varianten. Auf der Ebene der Varianten ist es nun möglich, feingranularer zu unterscheiden und verschiedene Operationen zur Ableitung beziehungsweise Ausgestaltung anzuwenden. Eine zentrale Frage stellt sich hier in Bezug auf *Erweiterungen* und *Reduktionen*.

Gehen wir von einem Entwicklungsstandard aus, müssen wir klassifizieren, ob es sich um einen *allumfassenden* oder einen *minimalen* Entwicklungsstandard handelt. Ein allumfassender Entwicklungsstandard ist in der Regel eher generisch gehalten und daher im Rahmen der Anpassung zumeist zu spezialisieren. Die Spezialisierung umfasst dann in der Regel einmal Schritte zur Reduktion nicht benötigter Anteile des Entwicklungsstandards (zum Beispiel Tailoring des V-Modell XT), andererseits sind auch Schritte zur Ergänzung spezifischer Anforderungen und Inhalte erforderlich. Ein minimaler Entwicklungsstandard lässt in der Regel keine Reduktionen mehr zu, sondern fordert kontext- und problemspezifische Ergänzungen. Entsprechend sind auch die Operationen, welche die Ableitung von Varianten realisieren, zu gestalten. Die Gestaltung dieser Operationen hat nicht nur maßgeblichen Einfluss auf die Variantenbildung, sondern gleichzeitig auch Konsequenzen im Bereich der Zertifizierung und Konformität. Sind die Operationen auf der strukturellen Ebene noch vergleichsweise einfach, müssen sie jedoch sicherstellen, dass sich Varianten identifizieren und zuordnen lassen.

Varianten eines Vorgehensmodells

Eine Variante eines Vorgehensmodells setzt eine Menge von Prozesskomponenten voraus, aus denen bestimmte Konstellationen variabel sind. Wir definieren zwei Mengen

$$Pk_1, Pk_2 \subseteq Pk_{gesamt},$$

die zwei Teilmengen aller verfügbaren Prozesskomponenten bilden. Beide bilden gemäß Formel (4.7) zusammen mit den Abhängigkeitsstrukturen im Kontext eines Vorgehensmodells einen korrespondierenden Abhängigkeitsgraphen G_{VgM_1} und G_{VgM_2}. Die beiden resultierenden Vorgehensmodelle v_1 und v_2 heißen Varianten, wenn:

$$\begin{aligned} Pk_1 \cap Pk_2 = Pk_3 \neq \emptyset \wedge \\ G_{VgM_1} \cap G_{VgM_2} = G_{VgM_3} \neq \emptyset \wedge \\ G_{VgM_3} \supseteq \bigcup_{1 \leq i \leq n} pk_i|_G \ : \ pk_i \in Pk_3 \end{aligned}$$

Eine Variante liegt also vor, wenn eine nicht leere Teilmenge der Prozesskomponenten exisitiert und der über den Prozesskomponenten der Teilmenge aufgespannte Anhängigkeitsgraph ein Teilgraph des Anhängigkeitsgraphen der Variante ist. Der aufgespannte Anhängigkeitsgraph muss ggf. noch um zusätzliche Prozesselementabhängigkeiten erweitert werden, um den Abhängigkeitsgraphen der Variante zu erhalten (vgl. Formel (4.6)).

Wir wollen dies in einem kurzen Beispiel diskutieren und einige Möglichkeiten, Varianten zu bilden, besprechen. In Abbildung 4.17 haben wir basierend auf unserem bisherigen Beispiel zwei verschiedene Möglichkeiten zur Variantenbildung skizziert. In der ersten Variante (Abbildung 4.17, a)) finden wir das Szenario, dass zwei Varianten v_1 und v_2 sich durch unterschiedliche Teilmengen Pk_1 und Pk_2 definieren. Der Abhängigkeitsgraph (aufgespannt durch die Öffnung der Prozesskomponenten durch $epka\left(pk_2, \{pk_1, pk_3\}\right)$) entspricht ebenfalls den Forderungen. Beide Varianten unterscheiden sich durch die Nicht-/Einbindung von pk_4 und einer entsprechend angepassten Verküpfungsregel. Für die Variante 1 geben wir folgende Formalisierung an:

$$\begin{aligned} \text{Vorgehensmodell:} \quad & v_1 \\ \text{Prozesskomponenten:} \quad Pk \quad = \quad & \{pk_1, pk_2, pk_3, pk_4\} \\ \text{Prozesskomponentenabhängigkeiten:} \quad & epka\left(pk_2, \{pk_1, pk_3\},\right. \\ & \left.\{pea\left(e, \alpha\right), pea\left(g, \beta\right)\}\right) \\ & pka\left(pk_4, pk_3, \{pea\left(k, \gamma\right)\}\right) \end{aligned}$$

Die Variante 2 ohne die Prozesskomponente pk_4 ist wie folgt definiert:

$$\begin{aligned} \text{Vorgehensmodell:} \quad & v_2 \\ \text{Prozesskomponenten:} \quad Pk \quad = \quad & \{pk_1, pk_2, pk_3\} \\ \text{Prozesskomponentenabhängigkeiten:} \quad & epka\left(pk_2, \{pk_1, pk_3\},\right. \\ & \left.\{pea\left(e, \alpha\right), pea\left(g, \beta\right), pea\left(f, \gamma\right)\}\right) \end{aligned}$$

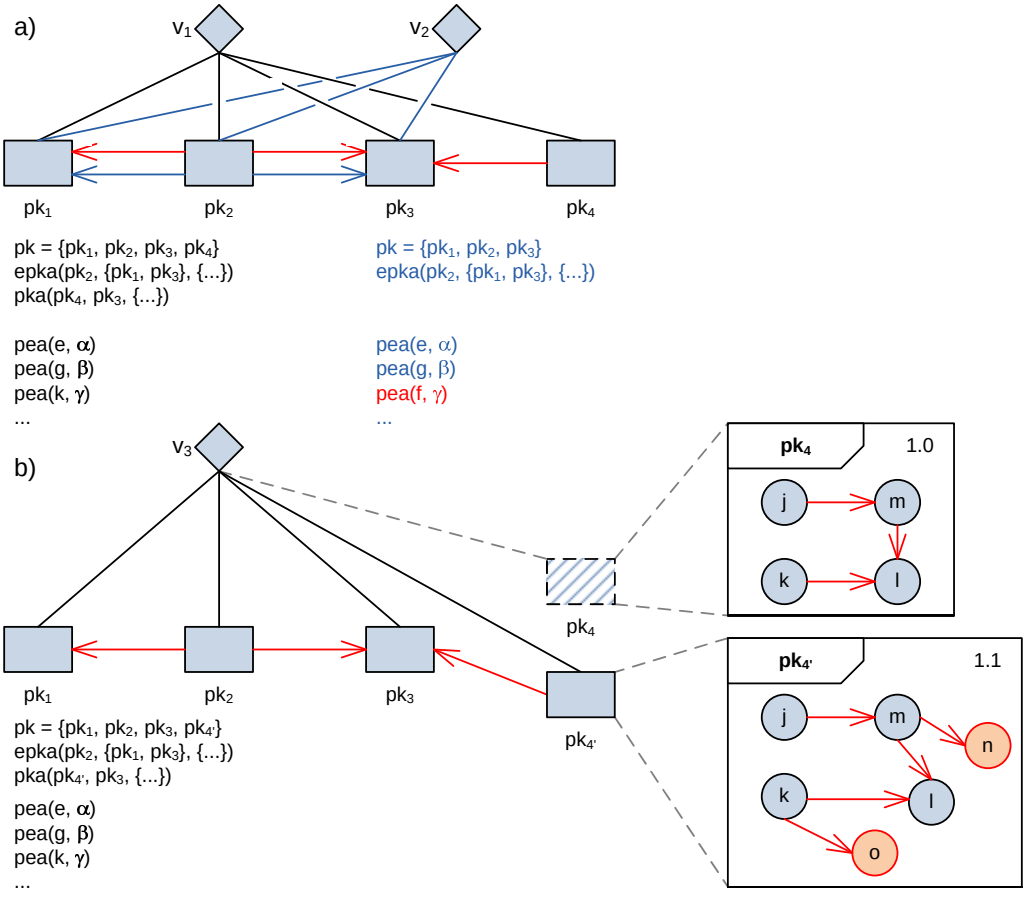


Abbildung 4.17.: Beispielszenarios für die Konfiguration von Varianten

Die zweite Möglichkeit (zu sehen in Abbildung 4.17, b)) zeigt eine Variantenbildung, wie sie in evolutionären Systemen auftreten kann. Die Variante v_3 des Vorgehensmodells bindet anstatt der Prozesskomponente pk_4 eine schnittstellenkompatible Prozesskomponente $pk_{4'}$ ein. Diese Variante ändert also eine „Basiskonfiguration" punktuell um einen Aspekt.

Insbesondere der zweite Teil des Beispiels in Abbildung 4.17, b) hebt noch einmal die Sinnhaftigkeit einer Plattformidee für Vorgehensmodelle hervor. Trotz der Schnittstellenkompatibilität von pk_4 und $pk_{4'}$ ist es erforderlich, beide Prozesskomponenten aufzubewahren und verfügbar[6] zu haben. Nach unserem Beispiel würde sich die Menge der Prozesskomponenten also bereits auf $Pk = \{pk_1, pk_2, pk_3, pk_4, pk_{4'}\}$ belaufen, die für verschiedene Konfigurationen verwendet werden. Bezüglich dieser Menge von Prozesskomponenten sind dann die Varianten zu prüfen, falls eine Konformität festgestellt werden muss.

Versionen eines Vorgehensmodells

Varianten fokussieren Teilaspekte eines Vorgehensmodells. Dies können unterschiedliche Teilmengen referenzierter Prozesskomponenten und unterschiedlich weiterentwickelte Versionsstände einzelner Komponenten sein[7]. Im Unterschied dazu betrachten wir bei einer Vorgehensmodellversion ein abgestimmtes, konsistentes Gesamtpaket.

4.3.3. Variabilitätsoperationen in Abhängigkeitsgraphen

Versionen und insbesondere Varianten eines Vorgehensmodells ist gemein, dass sie sich in Anzahl und/oder Konfiguration der verwendeten Prozesskomponenten beziehungsweise sonstiger durch das Metamodell definierter Teile unterscheiden. Basierend auf der einfachen, graphenbasierten Theorie dieser Arbeit wollen wir nun Variabilitäten und die hierfür erforderlichen Operationen näher untersuchen.

Rekonfiguration von Graphen. Prinzipiell führen wir die Variabilität eines Vorgehensmodells auf seine Bestandteile zurück. Aufgrund des konfigurationsbasierten Ansatzes dieser Arbeit betrifft dies also die Teile, die im Rahmen einer Konfiguration verwendet und referenziert werden. Im vorangegangenen Abschnitt haben wir dazu ein kleines Beispiel gegeben (vgl. Abbildung 4.17). Dieses Beispiel deckt die beiden Aspekte der *konfigurativen* und *evolutionären* Variantenbildung ab. Diese sind in der Regel vollständig durch Konfigurationen und einfache Konfigurationsoperationen abbildbar. Wir müssen im Rahmen der weiter gehenden Anpassungen jedoch auch Varianten betrachten, die nicht rein additiv oder alternativ sind, sondern auch *parzielle Änderungen oder Ergänzungen an gegebenen Inhalten* vornehmen. Ein Beispiel für ein derartiges Verhalten finden wir zum Beispiel im V-Modell Bw (Anhang A.1), wo Verantwortlichkeiten für Produkte geändert und angepasst werden müssen. An dieser Stelle sehen wir auch das Granularitätsniveau dieser Variationen. Variationen, die wir gerade als additiv beziehungsweise alternativ bezeichnet haben, finden auf der Ebene von Prozesskomponenten Anwendung. Änderungen, wie wir sie gerade am Beispiel des V-Modell Bw skizziert haben, werden jedoch direkt auf der Ebene von Prozesselementen vorgenommen und sind somit wesentlich feingranularer.

Ähnlich zu Prozesskomponentenabhängigkeiten (vgl. Abschnitt 4.2.2) werden *Variabilitätsoperationen* nur im Kontext von Prozesskomponenten angewendet. Sämtliche Knoten werden durch die Prozesskomponente bereitgestellt, die die variierten/neuen Elemente enthält. Zum Anwender hin werden also keine neuen Knoten erzeugt sondern nur durch Ersetzung alter Inhalte und Anpassung der Strukturen aktualisiert. Über eine „normale" Prozesskomponentenabhängigkeit hinausgehend, verfügen Variabilitätsoperationen jedoch über eine komplexere Semantik. Variabilitätsoperationen beziehen sich in unserem Modell immer auf ein Paar von Knoten (Prozesselemente): $(pe_1, pe_2) : pe_1 \in pk_1 \wedge pe_2 \in pk_2$, die immer genau aus zwei Prozesskomponenten pk_1 und pk_2 stammen oder auf ein Paar bestehend aus zwei Knoten und einer Menge von Kanten: $(pe_1, pe_2, K) : pe_1 \in pk_1 \wedge pe_2 \in pk_2 \wedge K \subseteq pk_2\,|_G$. Knoten werden durch eine

6 Eine Plattform sollte natürlich darauf achten, dass solche evolutionären Varianten entsprechend behandelt werden und zum Beispiel für die Neuableitung einer Vorgehensmodellkonfiguration nicht mehr angeboten werden. Wir greifen diesen Punkt in Kapitel 5.2.8 noch einmal auf.

7 Wie weiter oben schon gezeigt, können auch Prozesskomponenten mit unterschiedlichen Versionsständen als jeweils unterschiedliche, eigenständige Prozesskomponenten aufgefasst werden.

Variabilitätsoperation miteinander in Beziehung gesetzt, die in der Regel eine Ersetzung des „alten" Knotens im Abhängigkeitsgraphen vornimmt und beschreibt, wie mit den ein- und ausgehenden Abhängigkeiten des ersetzten Knotens zu verfahren ist. Graphentheoretisch (vgl. Abbildung 4.15, Schritt 3b) wird jedoch ein neuer Knoten im Graphen G_{VgM} erzeugt, der gemäß den Regeln der jeweiligen Operation die Eigenschaften der beteiligten Knoten übernimmt. Die beiden alten Knoten bleiben physisch zwar erhalten, sind jedoch aus dem vereinigten und verknüpften Abhängigkeitsgraphen G_{VgM} entfernt. Beziehen sich die Variabilitätsoperationen auch auf eine oder mehrere Kanten, so werden zwischen den beiden referenzierten Knoten nur die Kanten umgehängt, ohne auf die Knoten zu wirken. Dadurch ist es beispielsweise möglich, Beziehungen im Abhängigkeitsgraph des resultierenden Modells zu beeinflussen. Wir geben nun zunächst die unterstützten Variabilitätsoperationen unseres Modells an und zeigen im Anschluss die Anwendung am Beispiel.

Variabilitätsoperationen (Definition und Beschreibung). Wir sehen in unserem Modell die folgenden Variabilitätsoperationen vor (Abbildung 4.18):

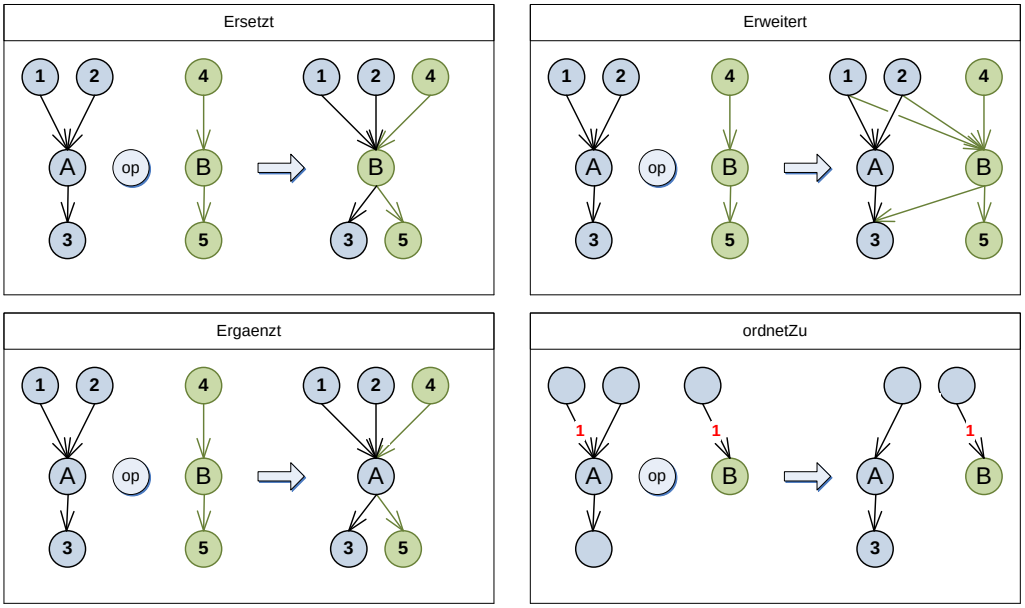


Abbildung 4.18.: Variabilitätsoperationen des Modells (abstrakt)

Diese Operationen definieren wir bereits abstrakt auf der Ebene des Basismodells, da sie für die Restrukturierung der Abhängigkeitsgraphen erforderlich sind. Im Rahmen der konkreten Modellierung des Vorgehensmetamodells sind die Operationen entsprechend zu verfeinern. Diesem Punkt widmen wir uns im Kapitel 5.1.4. An dieser Stelle beschreiben wir die Variabilitätsoperationen allgemein und gehen auf die Konsequenzen der Anwendung ein.

Grundsätzlich betrachten wir bei den Variabilitätsoperationen, wie oben bereits angedeutet, zunächst zwei Typen: Operationen über Knoten und Operationen zur Manipulation von Kanten. Unabhängig davon betrachten wir immer zwei Teilgraphen G_1 und G_2, die wir nach dem Prozess aus Abbildung 4.15 herleiten. In diesen Graphen betrachten wir jeweils ausgewählte Knoten (Prozesselemente, gegeben durch die Operation) pe_1 und pe_2 sowie deren Nachbarn. Eine Nachbarschaftsbeziehung zwischen zwei Prozesselementen pe und x liegt vor, wenn eine Kante (Prozesselementabhängigkeit) pea existiert, für die gilt $pea = (x, pe) \vee pea = (pe, x)$. Wir betrachten weiterhin auch nur Knoten, die in einer solchen Nachbarschaftsbeziehung ersten Grades stehen. Ein Variabilitätsoperation $\circ_{\text{var}}$ gestaltet sich somit im Allgemeinen wie folgt: Für zwei Graphen $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$ mit $E_1 = \{pe_1, x_1, \ldots, x_n\}$ und $E_2 = \{pe_2, y_1, \ldots, y_n\}$ sowie

den entsprechenden Kantenmengen gestaltet sich die Variabilitätsoperation $G_1 \circ_{\text{var}} G_2$ wie folgt – es entsteht ein integrierter Graph $G_{1,2} = (V_{1,2}, E_{1,2})$ mit:

$$E_{1,2} = E_1 \cup E_2 \cup \left\{pe_{1,2}\right\} \setminus \{pe_1, pe_2\}$$
$$V_{1,2} = V_1 \cup V_2 \cup V_{pe_{1,2}} \setminus \left\{V_{pe_1} \cup V_{pe_2}\right\}$$

Die Knotenmenge des integrierten Graphen enthält einen neuen Knoten $pe_{1,2}$, der die Eigenschaften der beiden ursprünglichen Knoten pe_1 und pe_2 vereint. Die Knoten pe_1 und pe_2 sind derweil im resultierenden Abhängigkeitsgraphen nicht mehr enthalten[8]. Analog setzt sich die neue Kantenmenge $V_{1,2}$ zusammen. In ihr sind die Kantenmengen der Ausgangsgraphen vereinigt, wobei diejenigen Kanten $V_{pe_1} \subseteq V_1$ und $V_{pe_2} \subseteq V_2$ entfernt werden, die einen Bezug zu den Prozesselementen pe_1 beziehungsweise pe_2 haben. Stattdessen ist in der neuen Kantenmenge eine neue Teilmenge $V_{pe_{1,2}}$ enthalten, die sich auf das neue Prozesselement $pe_{1,2}$ bezieht. Die genaue Ausgestaltung und Zusammensetzung der Kantenmenge $V_{1,2}$ obliegt hingegen den jeweiligen Variabilitätsoperationen. Für die in Abbildung 4.18 definierten Variabilitätsoperationen gilt jedoch, dass

$$\begin{aligned}\forall\, pea \in \left\{V_{pe_1} \cup V_{pe_2}\right\} . \exists\, pea' \in V_{1,2} : pea = (pe_1, x) &\Rightarrow pea' = \left(pe_{1,2}, x\right) \vee \\ pea = (pe_2, x) &\Rightarrow pea' = \left(pe_{1,2}, x\right) \vee \\ pea = (x, pe_1) &\Rightarrow pea' = \left(x, pe_{1,2}\right) \vee \\ pea = (x, pe_2) &\Rightarrow pea' = \left(x, pe_{1,2}\right)\end{aligned} \tag{4.8}$$

Bei der Manipulation von Kanten im Abhängigkeitsgraph wird die Knotenmenge nicht beeinflusst. Auch die Anzahl der Kanten im Graph verändert sich nicht. Die Manipulation der Kanten bezieht sich auf das Ändern von Zielen beziehungsweise Quellen von *konkreten* Kanten. Graphentheoretisch wird jedoch eine existierende Kante *pea* aus dem Abhängigkeitsgraph entfernt und eine neue Kante *pea'* an ihrer Stelle eingefügt. Für den integrierten Graph $G_{1,2} = (V_{1,2}, E_{1,2})$ gilt somit:

$$E_{1,2} = E_1 \cup E_2$$
$$V_{1,2} = V_1 \cup V_2 \cup \{pea'\} \setminus \{pea\}$$

Für die Kante *pea'* gilt im Weiteren, dass entweder ein neues Ziel oder eine neue Quelle angegeben wird, also:

$$\begin{aligned}pea = (pe_i, x) &\Rightarrow pea' = \left(pe_j, x\right) \vee \\ pea = (x, pe_i) &\Rightarrow pea' = \left(x, pe_j\right)\end{aligned} \tag{4.9}$$

Algorithmische Beschreibung. Die in der Abbildung 4.18 gezeigten Operationen wirken ausschließlich auf die Abhängigkeitsgraphen. Strukturen in Prozesskomponenten, beziehungsweise allgemeine Strukturen von Prozesselementen, werden durch sie nicht berührt. Abbildung 4.18 vermittelt bereits einen Eindruck von der Semantik der Operationen. Weiterhin haben wir die graphentheoretischen Grundlagen dazu besprochen. Im Folgenden wollen wir diese Operationen etwas weiter betrachten. Um den Umgang mit den definierten Variabilitätsoperationen mit Hinblick auf die werkzeugorientierte Metamodellierung im Kapitel 5.1 zu fokussieren, geben wir jede Operation in Pseudocode an, um die Wirkung zu verdeutlichen. Diese Operationen sind dann, sofern konkret implementiert, in dieser Form in die Prozesse des Lebenszyklusmodells (Kapitel 5.2) direkt integrierbar.

8 Dies betrifft jedoch nicht den Strukturkompositionsgraph der jeweiligen Prozesskomponenten. Die Knoten werden „physisch" nicht gelöscht, jedoch bei der Komposition des Prozesses (dem Export) ignoriert.

Operation: `Ersetzt` – Die Operation `Ersetzt` blendet einen gegebenen Knoten aus und übernimmt seine Position im Abhängigkeitsgraph. Wir betrachten dabei zwei nicht zusammenhängende Graphen, die durch die Substitution eines Knotens verknüpft werden. Wir beziehen uns auf Abbildung 4.18 (links, oben). Folgender Algorithmus beschreibt die Variabilitätsoperation *ersetzt*:

```
// Operation: ersetzt - Anwendung: ersetzt(B, A);
// Aktualisiere die Kanten
foreach(kante in Graph.Kanten) {
  // A ist Ziel der Kante
  if(kante.Ziel == A)
    kante.Ziel = B;
  // A ist Quelle der Kante
  else if(kante.Quelle == A)
    kante.Quelle = B;
}
// Substituiere Knoten
A = B;
```

In Konsequenz ist im Anschluss der Knoten A nicht mehr im Abhängigkeitsgraphen enthalten. Wir betrachten an dieser Stelle explizit nicht die Eigenschaften der zu variierenden Knoten. Im Kapitel 5.1.4 gehen wir dann darauf ein und diskutieren die Randbedingungen für konkrete Variabilitätsoperationen.

Operation: `Erweitert` – Die Operation `Erweitert` kopiert und übernimmt die Kanten die zu beziehungsweise von einem gegebenen Knoten führen. In Abbildung 4.18 ist dies rechts oben skizziert. Folgender Algorithmus beschreibt die Variabilitätsoperation *erweitert*:

```
// Operation: erweitert - Anwendung: erweitert(B, A);
// Kopiere die Kanten und verknüpfe die neuen
// Kanten mit dem neuen Knoten
foreach(kante in Graph.Kanten) {
        // A ist Ziel der Kante
  if(kante.Ziel == A) {
    kanteNeu = new Kante(kante);
    kanteNeu.Quelle = kante.Quelle;
    kanteNeu.Ziel = B;
  }
  // A ist Quelle der Kante
  else if(kante.Quelle == A) {
    kanteNeu = new Kante(kante);
    kanteNeu.Quelle = B;
    kanteNeu.Ziel = kante.Ziel;
  }
}
```

In diesem Szenario wird also gewissermaßen eine Spezialisierung eines Knotens vorgenommen, vergleichbar mit einer Vererbungsbeziehung in objektorientierten Systemen. Beide Knoten A und B sind dabei im resultierenden Abhängigkeitsgraphen enthalten. Auch hier sind für konkrete Variabilitätsoperationen entsprechende Rand-, Vor- und Nachbedingungen zu prüfen (vgl. Kapitel 5.1.4).

Operation: `Ergaenzt` – Die Operation `Ergaenzt` bindet einen existierenden Knoten in eine neue Knoten/Kanten-Struktur ein. Der „neue" Knoten ist dabei ein virtueller Knoten, der durch den existierenden Knoten ausgeblendet wird. In Abbildung 4.18 ist dies links unten skizziert. Folgender Algorithmus beschreibt die Variabilitätsoperation *ergaenzt*:

```
// Operation: ergaenzt - Anwendung: ergaenzt(B, A);
// Aktualisiere die Kanten
foreach(kante in Graph.Kanten) {
  // B ist Ziel der Kante
  if(kante.Ziel == B)
    kante.Ziel = A;
  // B ist Quelle der Kante
  else if(kante.Quelle == B)
    kante.Quelle = A;
}
// Entferne Knoten
B = null;
```

Durch diese Operation werden komplexe Teilgraphen miteinander verknüpft, indem die Eigenschaften des neuen an der Koppelstelle denen des aktualisierten Teilgraphen hinzugefügt werden. Der Knoten B ist dabei im Wesentlichen ein Stellvertreter für den zu aktualisierenden Knoten. Er ist im resultierenden Abhängigkeitsgraph nicht mehr enthalten.

Operation: `OrdnetZu` – Die Operation `OrdnetZu` wird auf Kanten im Abhängigkeitsgraph angewendet. Sie führt keine Ersetzung von Knoten durch, sondern dient einzig dazu, definierte Prozesselementabhängigkeiten so zu manipulieren, dass ein neues Ziel oder eine neue Quelle einer Kante angegeben werden kann. Folgender Algorithmus beschreibt die Variabilitätsoperation *ordnetZu*:

```
// Operation: ordnetZu - Anwendung: ordnetZu(B, A, P);
// Ermittle die Kanten, die durch eine
// Menge P gegeben sind...
foreach(kante in P) {
  // A ist Ziel der Kante
  if(kante.Ziel == A)
        kante.Ziel = B; // Kante zeigt jetzt auf B
  else if(kante.Quelle == A)
    kante.Quelle = B; // B ist jetzt Quelle
}
```

Durch diese Operation können Abhängigkeiten im Anhängigkeitsgraph „umgebogen" werden. Ein praktisches Beispiel hierfür finden wir im V-Modell Bw, wo wir mithilfe der Variabilitätsoperationen zwar die Rolle Projektleiter des V-Modells durch den Projektleiter (CPM) spezialisieren könnten, dann aber zwei verantwortliche Rollen pro Produkt hätten. Mit dieser Operation können wir statt dessen einfach eine neue Rolle definieren und dann die Assoziationen im Modell neu ausrichten. Die Option, dass mit dieser Operation auch Quellen manipuliert werden können ist vorhanden, um ggf. Knoten aus dem Abhängigkeitsgraph zu entfernen und ist nicht mit dem einfachen Anlegen einer neuen Kante mit B als Quelle zu verwechseln.

Gerade diese letzte Operation zeigt jedoch, dass es durchaus Fälle gibt, in denen die einfache Anwendung der hier vorgestellten Operationen nicht unmittelbar zum Ziel führt. Deshalb sind die Variabilitätsoperationen auch Gegenstand einer potenziellen Erweiterung. Diese Erweiterung muss sich dann natürlich an den konkreten Anwendungsszenarios orientieren.

In Abbildung 4.19 geben wir ein konkretes, integriertes Beispiel für die Anwendung von Variabilitätsoperationen in einem System von Prozesskomponenten an. Um eine einfachere Einordnung zu ermöglichen, verwenden wir auch hier die Farbkodierung aus Abbildung 4.18 um erweiterte und erweiternde Elemente zu kennzeichnen.

Das Basissystem, das wir betrachten, setzt sich aus den beiden Prozesskomponenten pk_1 und pk_4 zusammen. Die beiden Prozesskomponenten pk_2 und pk_3 werden in das System mithilfe von Prozesskomponentenabhängigkeiten integriert und führen dabei Variabilitätsoperationen aus. Diesen Aspekt verdeutlicht die Abbildung 4.19 im oberen Teil. Wir formalisieren diesen Teil wie folgt:

$$
\begin{array}{rlll}
\text{Vorgehensmodell:} & v_{variiert} & & \\
\text{Prozesskomponenten:} & Pk & = & \{pk_1, pk_2, pk_3, pk_4\} \\
\text{Prozesskomponentenabhängigkeiten:} & & & epka\left(pk_3, \{pk_1, pk_4\},\right. \\
 & & & \left.\{erweitert\,(\beta, c)\,, ergaenzt\,(\gamma, m)\}\right) \\
 & & & pka\left(pk_2, pk_1, \{ersetzt\,(\alpha, b)\}\right)
\end{array}
$$

Die Anwendung der Variabilitätsoperationen hat zur Folge, das nicht nur einzelne Knoten oder Kanten manipuliert werden, sondern dass komplette Teilgraphen reorganisiert werden. Wir betrachten die drei beispielhaften Variabilitätsoperationen aus Abbildung 4.19 und diskutieren deren Wirkung mit Bezug auf den unteren Teil der Abbildung, wo einerseits die beteiligten Teilgraphen vor Anwendung der Operationen zu sehen sind und weiterhin der resultierende Gesamtgraph nach Anwendung der Operationen. Für die Operation *ersetzt* (α, b) betrachten wir die beiden Abhängigkeitsteilgra-

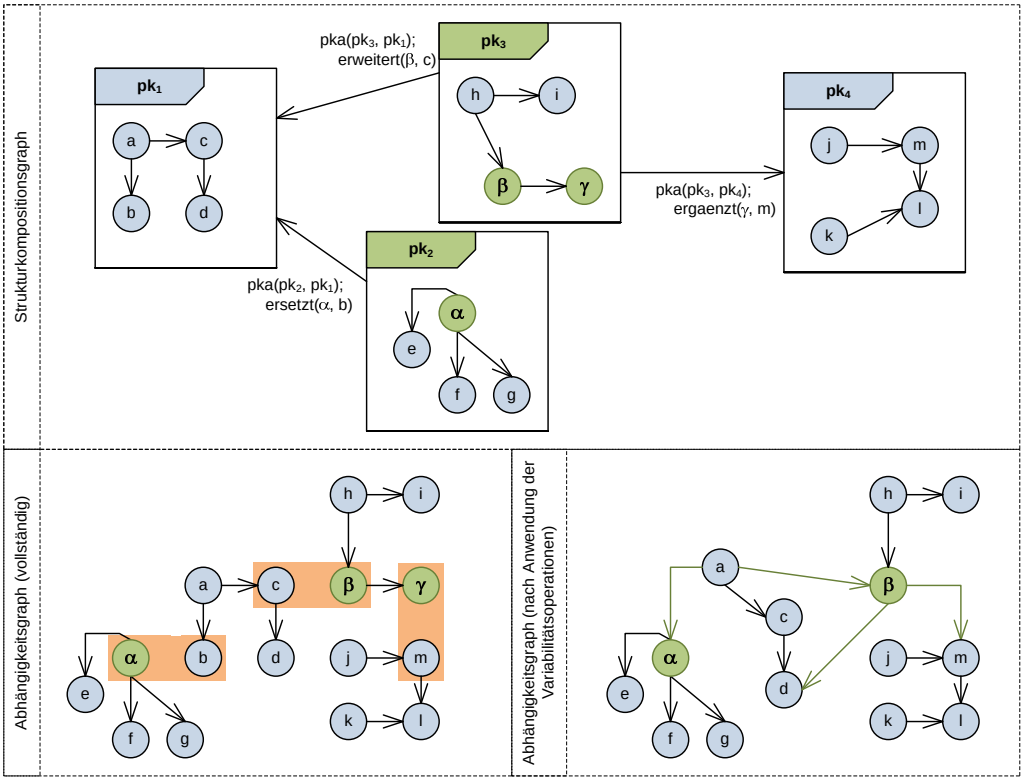


Abbildung 4.19.: Beispielhafte Anwendung der Variabilitätsoperationen

phen[9] G_1 und G_2.

$$\begin{aligned} G_1 = &\quad (\{a,b\},\{(a,b)\}) \\ G_2 = &\quad (\{\alpha,e,f,g\},\{(\alpha,e),(\alpha,f),(\alpha,g)\}) \end{aligned}$$

Der Variationspunkt ist in diesem Fall der Knoten b im Graph G_1. Die Semantik der angewendeten Operation *ersetzt* lautet, dass der neue Knoten (in diesem Fall α) den alten Knoten ersetzt und dabei seine Kanten „übernimmt". Die beiden Abhängigkeitsteilgraphen G_1 und G_2 werden also durch $b = \alpha$ miteinander verknüpft, sodass ein neuer Abhängigkeitsteilgraph $G_{1,2}$ entsteht mit:

$$G_{1,2} = \quad (\{a,\alpha,e,f,g\},\{(a,\alpha),(\alpha,e),(\alpha,f),(\alpha,g)\})$$

Analog gehen wir bei der Operation *erweitert* (β, c) vor und geben die beiden Abhängigkeitsgraphen G_3 und G_4 an. Semantik der Operation für den Variationspunkt c lautet, dass β alle ein- und ausgehenden Kanten von c kopiert. Es entsteht der kombinierte Abhängigkeitsgraph $G_{3,4}$.

$$\begin{aligned} G_3 = &\quad (\{a,c,d\},\{(a,c),(c,d)\}) \\ G_4 = &\quad (\{h,\beta,\gamma\},\{(h,\beta),(\beta,\gamma)\}) \\ G_{3,4} = &\quad (\{a,c,d,h,\beta,\gamma\},\{(a,c),(c,d),(h,\beta),(\beta,\gamma),(a,\beta),(\beta,d)\}) \end{aligned}$$

Für die Operation *ergaenzt* (γ, m) betrachten wir zunächst die beiden Abhängigkeitsgraphen G_5 und G_6. Die Semantik der angewendeten Operation bedeutet für den Knoten m, dass er zusätzliche Kanten von γ bekommt. Nach der Operation ist γ nicht mehr im kombinierten Abhängigkeitsgraphen $G_{5,6}$ enthalten.

$$\begin{aligned} G_5 = &\quad (\{j,m,l\},\{(j,m),(m,l)\}) \\ G_6 = &\quad (\{\beta,\gamma\},\{(\beta,\gamma)\}) \\ G_{5,6} = &\quad (\{j,m,l,\beta\},\{(j,m),(m,l),(\beta,m)\}) \end{aligned}$$

9 Wir betrachten hier für die Bildung der Abhängigkeitsteilgraphen nur die unmittelbaren Nachbarn, da für die Kantenmengen nur die Knoten und Kanten des Graph relevant sind, die einen unmittelbaren Bezug zu den an der Variabilitätsoperation beteiligten Knoten haben.

Die gerade erläuterten Schritte arbeiten jeweils auf Abhängigkeitsteilgraphen. Den gesamten Abhängigkeitsgraphen erhalten wir durch Nacheinanderausführung der Operationen, wie folgt:

Variabilitätsoperationen: *ersetzt* (α, b)
erweitert (β, c)
ergaenzt (γ, m)

Diese Operationen erzeugen die Abhängigkeitsteilgraphen $G_{1,2}$, $G_{3,4}$ und $G_{5,6}$. Insbesondere die Kombination von $G_{3,4}$ und $G_{5,6}$ zeigt, dass hier Knoten und Kanten deutlichen Veränderungen unterliegen können. Den gesamten, resultierenden Abhängigkeitsgraphen für die Ausführung aller Operation zeigt Abbildung 4.19 (rechts, unten). Die Knoten (Prozesselemente) b und γ sind in diesem Graphen nicht mehr enthalten, da sie durch die Variabilitätsoperationen aktualisiert wurden, beziehungsweise nach Weitergabe ihrer Strukturinformationen entfallen sind. Zusätzlich sind jedoch auch neue Kanten, wie zum Beispiel (a, β) oder (β, m) hinzugekommen. *Dieser integrierte Abhängigkeitsgraph repräsentiert ein zur Anwendung exportiertes Vorgehensmodell.*

Konflikte bei der Mehrfach-/Nacheinanderausführung. Im gerade gezeigten Beispiel finden wir einen *idealen* Ablauf bei der Verknüpfung der Abhängigkeitsgraphen bei der Zusammenstellung einer Konfiguration von Prozesskomponenten. Als ideal bezeichnen wir diese Komposition deshalb, weil es keine *konkurrierenden* Variabilitätsoperationen gibt. Konflikte können auftreten, weil die Variabilitätsoperationen nicht assoziativ sind. In Abbildung 4.20 findet sich ein Beispiel, in dem zwei Operationen konkurrierend ein Prozesselement variieren wollen. Wir betrachten ein Szenario bestehend aus den drei Prozesskomponenten pk_1, pk_2 und pk_3. Die Prozesskomponenten pk_1 ist dabei die gegebene Größe, die durch pk_2 und pk_3 angepasst werden soll.

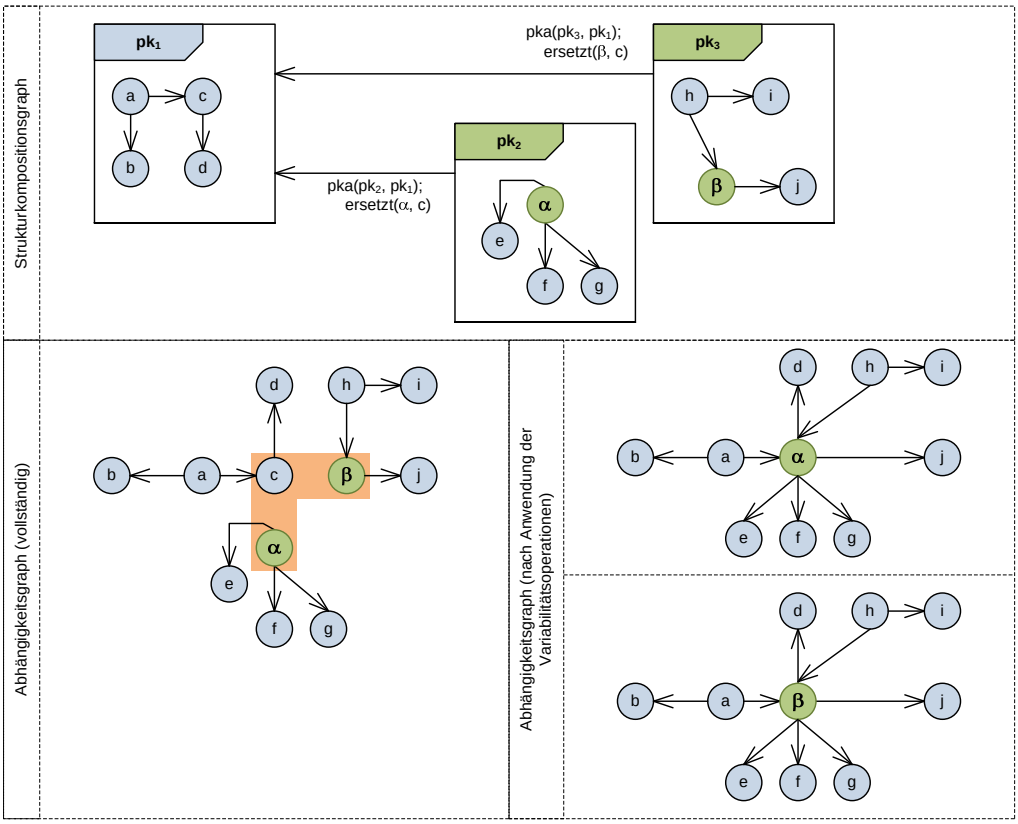


Abbildung 4.20.: Mehrfach- und Nacheinanderanwendung von Variabilitätsoperationen

Abbildung 4.20 zeigt im oberen Teil den Strukturgraph und die beiden Operationen

ersetzt (α, c) sowie *ersetzt* (β, c). Der untere Teil der Abbildung zeigt im linken Bereich die Problemstelle. Beide Operationen wollen den Knoten c ersetzen. Der Konflikt, der hier auftritt, entsteht durch die Mehrfachanwendung einer Operation auf ein Element, durch welche das betroffene *Urelement* aus dem resultierenden Graphen entfernt und durch das neue (variierende) Element ersetzt wird. Im rechten Teil von Abbildung 4.20 ist dies zu sehen: Wird zuerst die Operation *ersetzt* (β, c) und dann *ersetzt* (α, c) angewendet, so sind weder c noch β im finalen Abhängigkeitsgraphen enthalten. Anders herum: Wird zuerst *ersetzt* (α, c) und im Anschluss *ersetzt* (β, c) ausgeführt, sind weder c noch α im Abhängigkeitsgraphen enthalten.

Die Schwierigkeit dabei ist, dass solche Operationen nicht zwangsweise unmittelbar nacheinander stehen, sondern über ein weites Feld von Variabilitätsoperationen verteilt sein können. Ein typisches Szenario, in dem eine solche Situation entstehen kann, ist die Verwendung von 3rd-Party Komponenten. Diese betrachten wir auch im Kontext von Vorgehensmodellen als Black-Box und müssen sie daher so annehmen, wie sie vom erstellenden Prozessingenieur bereitgestellt werden. Tritt in einem Vorgehensmodellkonstruktions- oder Anpassungsprozess nun durch zwei extern gefertigte Prozesskomponenten eine Mehrfachvariation eines gegebenen Elements auf, so ist dies ein *Konflikt*, der gelöst werden muss. Die Ermittlung eines solchen Konflikts kann algorithmisch zum Beispiel wie folgt angegangen werden[10]:

```
// ermittle zuerst alle Prozesskomponenten, die mehrfach
// referenziert werden
// pk  = Prozesskomponente
// pka = ProzesskomponentenAbhaengigkeit
pkCount = 0;
pkMultiRef = new Set<pk>;
foreach(pk in Konfiguration.Prozesskomponenten) {
  foreach(pka in Konfiguration.ProzesskomponentenAbhaengigkeiten) {
          if(pka.Ziel == pk)
            pkCount++;
          if(pkCount > 1)
            pkMultiRef += pk;
  }
}
// prüfe für alle mehrfach referenzierten Prozesskomponenten, ob
// sie durch die Referenzierung variiert werden sollen, ermittle die
// kritischen Elemente...
// pe = Prozesselement
peVar = new Set<pe>;
peMultiVar = Set<varOperation>;
foreach(pk in pkMultiRef) {
  foreach(pka in Konfiguration.ProzesskomponentenAbhaengigkeiten
            where pka.Ziel == pk) {
    foreach(varOperation in pka.ProzesselementAbhaengigkeiten) {
        if(!peVar.contains(varOperation.Ziel))
          peVar += varOperation.Ziel;
        else
          // peMultiVar enthält die kritischen
          // Variabilitaetsoperationen zur
          // Konfliktindentifikation und -beseitigung
          peMultiVar += varOperation;
    }
  }
}
```

Nichtsdestotrotz liegt an dieser Stelle lediglich ein Liste der kritischen Variablilitätsoperationen vor. Wobei wir durch den oben skizzierten, einfachen Algorithmus generell *alle* Mehrfachvariationen erfassen. Dies ist nicht immer notwendig, da zum Beispiel Mehrfachanwendung von *erweitert* oder *ergaenzt* unkritisch ist. Trotzdem sind wir hier an einem Punkt, wo eine Entscheidung getroffen werden muss, die nicht durch ein System erfolgen kann. Hier *muss* der Mensch in Form des Prozessingenieurs entscheiden, wie der Konflikt aufzulösen ist.

10 Die Darstellung erfolgt in an C# angelehnten Pseudocode.

Zusammenfassung

In diesem Kapitel haben wir unser grundlegendes Modell bestehend aus *Prozesselementen*, *Prozesskomponenten* sowie *Prozesselement-* und *Prozesskomponentenabhängigkeiten* formuliert. Dazu haben wir aufbauend auf einem allgemeinen Lebenzyklusmodell Hierarchieebenen für Entwicklungsstandards betrachtet. Wir haben Lebenszyklus und Hierarchieebenen zueinander positioniert und die grundlegende Notwendigkeit von Anpassungen und deren geregelter Durchführung motiviert.

Im Abschnitt 4.1.2 haben wir das für diese Arbeit maßgebende Strukturmodell eingeführt (vgl. Abbildung 4.6). Das Strukturmodell besteht aus drei *Integrationsebenen*. Die kleinsten Einheiten, die wir betrachten, sind die *Prozesselemente*, die wir zu *Prozesskomponenten* integrieren. Gemäß den Analyseergebnissen und Anforderungen aus Kapitel 3 lehnen wir das Konzept der Prozesskomponente am V-Modell-Konzept Vorgehensbaustein an, sind hier jedoch noch abstrakter und verwenden die Prozesskomponente nur als Container für Prozesselemente. Anders als Komponenten der Software Entwicklung abstrahieren Prozesskomponenten nicht. Dennoch bieten sie über die Konfiguration einen geregelten Zugriff auf die enthaltenen Prozesselemente. Prozesskomponenten wiederum integrieren wir zu *Vorgehensmodellvarianten*.

Über allen diesen Konzepten stehen einfache Elemente der Graphentheorie. Wir betrachten im Kontext dieser Arbeit zwei verschiedene Graphen:

- Der *Strukturkompositionsgraph* ist ein Baum, der die Hierarchieebenen umsetzt und hinter dem Modulkonzept steht.
- Der *Abhängigkeitsgraph* ist ein gerichteter Graph, der über den Knoten des Strukturkompositionsgraphs – und dort über den Prozesselementen – gebildet wird. Er beschreibt losgelöst von physischen Strukturen einen integrierten Prozess.

Zur Integration der jeweiligen Elemente zum Konzept der nächst höheren Ebene geben wir ein Abhängigkeitskonzept an. Abhängigkeiten stellen das wesentliche Mittel für die *Konfiguration* dar. Prozesskomponenten werden durch Abhängigkeiten zwischen Prozesselementen gebildet; Vorgehensmodellvarianten wiederum werden durch Abhängigkeiten zwischen Prozesskomponenten gebildet, die selbst wieder Abhängigkeiten zwischen Prozesselementen enthalten. Die Abhängigkeitsstrukturen zwischen Prozesskomponenten geben uns darüber hinaus auch Mittel zur Formulierung verschiedener Schnittstellentypen an die Hand. Beispielhaft haben wir die angeforderte Schnittstelle angerissen. Diese fasst in einer Prozesskomponentenabhängigkeit solche Prozesselementabhängigkeiten zusammen, die nur gültig erfüllt werden können, wenn zwei oder mehrere Prozesskomponenten miteinander gekoppelt werden. Dadurch lassen sich zum Beispiel auch vorkonfigurierte Prozesskomponenten bilden – ein erster Schritt in Richtung eines Referenzmodells. Wir sehen nur diese drei Integrationsebenen vor und betrachten keine rekursiven Strukturen. Die Mächtigkeit dieses einfachen Modells genügt für die Modellierung von Entwicklungsstandards und hat dabei gleichzeitig den Vorteil, dass die flache Hierarchie einfacher aufgebaute Module gestattet.

Ab dem Abschnitt 4.2 befassen wir uns mit Prozesskomponenten und zeigen, wie auf einer abstrakten Ebene höher integrierte Strukturen entstehen. Anhand einfacher Beispiele diskutieren wir die Wirkung des Modells bei der Integration von Prozesskomponenten zu Vorgehensmodellvarianten. Wir untersuchen – wie gerade schon angedeutet – Schnittstellen von Prozesskomponenten und betrachten bereits die Verknüpfung von Teilgraphen im Rahmen dieser Integration.

Der Abschnitt 4.3 diskutiert die Versions- und Variantenbildung sowie die Weiterentwicklungsaspekte. Zu Beginn geben wir noch einmal einen abstrakten Konstruktionsprozess über den einfachen Graphenstrukturen an. Wir diskutieren im Anschluss die Herleitung von Versionen und Varianten und zeigen am Beispiel, die Varianten in unserem Modell ausdrückbar sind. Wir unterscheiden aber auch zwischen *konfigurativen/evolutionären* Varianten, die durch die Konfiguration unmittelbar entstehen, und *anpassungsbedingten* Varianten, die in der Regel im Rahmen einer organisationsspezifischen Anpassung erstellt werden. Diese Varianten werden in unserem Modell durch *Variabilitätsoperationen* erzeugt, die aufbauend auf einer Konfiguration die resultierenden Abhängigkeitsgraphen rekonfigurieren. Als Ergänzung zum Modell geben wir für die-

se Operationen bereits einfache Algorithmen an, die einerseits die Wirkung plastisch darstellen andererseits jedoch mit Hinblick auf die werkzeugunterstützte Metamodellrealisierung einen ersten Ansatzpunkt für die mögliche Umsetzung liefern. In dieser abstrakten Form zeigen sie lediglich die Ansatzpunkt für solche Operationen. Für die vollständige Abdeckung eines wandlungsfähigen Vorgehensmetamodells sind jedoch Operationen erforderlich, die über die hier skizzierten hinaus gehen. Der Rückgriff auf einen vollständigen Ansatz zur Graphentransformation ist für komplexere Operationen notwendig.

Mit diesem Kapitel haben wir unser Basismodell eingeführt. Motiviert aus den Ergebnissen von Kapitel 3 ist das Modell einfach gehalten. Wie bereits im Vorfeld festgestellt, ist die Anzahl der Elementtypen, die wir in der Domäne der Prozesse betrachten, gering. Aus diesem Grund haben wir weitgehend auf rekursive und komplexe, hierarchische Strukturen verzichtet. Stattdessen stellen wir einen *konfigurationsbasierten* Ansatz vor und präzisieren ihn. Die Tragfähigkeit hinsichtlich aller relevanten Punkte (einfache Struktur für Elemente und Verteilung, Abhängigkeitssysteme etc., vgl. Kapitel 3.6) haben wir durch Beispiele gezeigt. Den Schritt in Richtung Umsetzung haben wir punktuell durch Angabe einfacher Algorithmen ebenfalls schon getan. Unser Basismodell ist noch nicht typisiert. Eine Typisierung ist jedoch der nächste erforderliche Schritt, um präzise Aussagen zu konkreten Element- und Beziehungstypen treffen zu können. Auch ist die Typisierung erforderlich, um die verschiedenen Schnittstellen und Konfigurationsoptionen auf der Ebene von Prozesskomponenten zu präzisieren.

Das Basismodell als solches ist abstrakt und dient im Folgenden zur Beschreibung unseres Prozess-Frameworks, das eine Kernkomponente einer Prozess-Plattform darstellt. Dieses stellen wir in Kapitel 5 vor und bauen darauf ein UML-basiertes Metamodell für Vorgehensmodelle auf.

5. Integrierter Modellierungsansatz für Vorgehensmodelle

In diesem Kapitel greifen wir die Ergebnisse der formalen Modellierung aus Kapitel 4 auf und überführen sie auf einen integrierten, UML-basierten Modellierungsansatz für Vorgehensmodelle (IMV). Dieser beschreibt die beiden wesentlichen Ergebnisse dieser Arbeit: das *Metamodell* und das *Lebenszyklusmodell* (vgl. Abbildung 1.2, Kapitel 1.2). Im ersten Teil widmen wir uns den Architektursichten und führen die Basis- und Strukturmodelle ein. Auf der nächsten Modellierungsebene beschreiben wir einen Ansatz zur Komponentenbildung – insbesondere mit Berücksichtigung der Abhängigkeitsstrukturen. Konfigurationen sind unser Vorschlag eines Konstruktionsmodells und stellen das zentrale Element für die Komposition eines Vorgehensmodells sowie die Bildung von Varianten dar. Im zweiten Teil dieses Kapitels greifen wir den Lebenszyklus eines Vorgehensmodells auf. Wir beschreiben einen phasenorientierten Ansatz in Form von UML Verhaltensdiagrammen. Basierend auf den durch diese Arbeit abgedeckten Anwendungsfällen betrachten wir ausgewählte Phasen dieses Lebenszyklusmodells genauer und stellen eine konkrete Modellierung vor. In dieser Modellierung stellen wir gleichzeitig die Verbindung zwischen Metamodell und Lebenszyklusmodell her. Wir schließen dieses Kapitel mit einer Beschreibung des begleitenden Werkzeugkonzepts.

Am Ende dieses Kapitels wurde auf Basis eines formalen Modells ein integrierter Modellierungsansatz für Vorgehensmodelle (IMV) eingeführt. Neben einer Strukturmodellierung ist insbesondere der Lebenszyklus eingeführt. Die dort definierten Prozesse dienen nicht nur der Erstellung eines Vorgehensmodells, sondern auch seiner Pflege und Weiterentwicklung.

5.1. Modellierungssicht: Vorgehensmodell

Wir fokussieren zuerst die Strukturmodellierung. Diese haben wir bislang im Kapitel 4 weitgehend abstrahiert. Wir gehen in diesem Abschnitt auch auf Details ein, die erforderlich sind, um eine Referenzimplementierung zu ermöglichen. Im Anhang C stellen wir ein entsprechendes Konzept in Auszügen vor.

5.1.1. Basis- und Strukturmodelle

Wir nähern uns der Modellierung des Vorgehensmetamodells schrittweise an. Zuerst geben wir die Basisstruktur in Form eines UML 2 Paketdiagramms an (Abbildung 5.1). Anders als Gnatz [Gna05] fokussieren wir nicht planungsorientierte Vorgehensmodelle, sondern stellen allgemeine Strukturen ins Zentrum. Unser Ansatz in Abbildung 5.1 zeigt eine Aufteilung[1] in drei Pakete:

Basis: Das Basispaket definiert dem Gedanken eines Frameworks [HR02] folgend alle notwendigen Basiselement- und Beziehungstypen. Prozesselemente oder Prozesselementabhängigkeiten (Kapitel 4.2) aber auch die Konfigurationen sind hier zu finden.

Prozesskomponenten: Prozesskomponenten enthalten die strukturellen Verfeinerungen des Basispakets, mit denen die inhaltlichen Bestandteile eines Vorgehensmodells modelliert und erstellt werden.

Planung: Das Paket Planung enthält eine Modellbeschreibung für Planungstechniken. Ohne entsprechende Planungseinheiten, also nur mit Elementen der Pakte *Basis*

1 Die Architektur ist modular ausgelegt und daher bereits an dieser Stelle erweiterbar. Auf Erweiterungsoptionen gehen wir in Anhang B noch einmal ein.

und *Prozesskomponenten*, ist ein Vorgehensmodell lediglich ein druckbarer Leitfaden. Zur Operationalisierung sind Planungseinheiten erforderlich, die beispielsweise Gnatz [Gna05] detailliert modelliert hat. Bis auf Weiteres betrachten wir die Planung als Blackbox. Im Anhang B greifen wir dieses Thema jedoch noch einmal auf und zeigen am Beispiel die Integration der Planungsmethode von Gnatz in unser Metamodell.

Vorgehensmodell: Das Paket Vorgehensmodell fasst die Elemente zusammen, die wir für die Ableitung von Vorgehensmodellen benötigen. Im Wesentlichen geht es hier um die Öffnung und gegenseitige Bekanntmachung der Namensräume und Elemente, die zum Erstellen eines Vorgehensmodells benötigt werden.

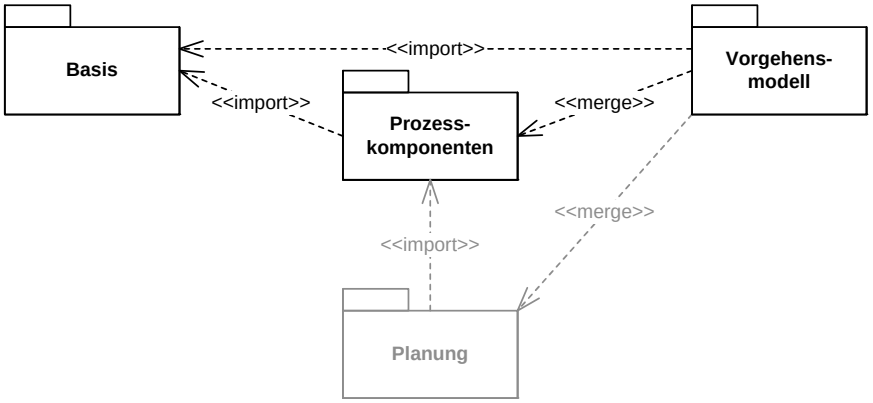


Abbildung 5.1.: Paketstruktur des Vorgehensmetamodells

Im Anschluss stellen wir erst das *Basis*-Paket detaillierter vor und geben dann einen ersten Eindruck des Gesamtsystems. Wir betrachten dabei allgemeine Eigenschaften wie die Variabilität aber auch Dokumentation, Ressourcen und Konfiguration.

Basiselemente und -modell. Vorgehensmodelle sind nach unserer Theorie *komplexe komposite Strukturen*. Als kleinste Elemente definieren wir *Prozesselemente*, die mithilfe einer *Konfiguration* in Prozesskomponenten zusammengefasst werden. *Prozesskomponenten* wiederum fassen wir ebenfalls konfigurativ zu Vorgehensmodellen zusammen. Wir erhalten somit ein hierarchisches Konstruktionsmodell, das wir in Kapitel 4.1 motiviert haben. Neben der reinen Betrachtung von Strukturen (Ergebnisstrukturen) betrachten wir weiterhin Abläufe, die ebenfalls Teil eines Vorgehensmodells sind. Das von uns hier modellierte Konzept ist ein einfaches Basiskonzept, das hier noch hinter dem Umfang etablierter Vorgehensmetamodelle wie zum Beispiel dem V-Modell XT zurücksteht. Wir setzen jedoch auf konsequente Erweiterbarkeit und können mit diesem Basismodell alle benötigen Aspekte modellieren. Im Anhang B geben wir dafür ein Beispiel an.

Nachdem das *Prozesselement* unsere grundlegende Einheit ist, verfeinern wir zunächst diesen Teil des Metamodells (Abbildung 5.2). Gemäß Definition 4.3 muss ein Prozesselement im Rahmen einer Modellierung konkretisiert werden. Wir modellieren das Prozesselement somit als abstrakte Klasse. Zusätzlich modellieren wir auch den grundlegenden Abhängigkeitstyp auf der Basis der *erweiterten Prozesselementabhängigkeit* (die einfache behandeln wir als mögliche Spezialisierung). Im weiteren Verlauf der Arbeit definieren wir entsprechende Spezialisierungen für konkrete Prozesselemente und Prozesselementabhängigkeiten. Die wesentlichen Elemente des Modells in Abbildung 5.2 sind die Klassen `ProzessElement`, `ProzessElementAbhaengigkeit` und `Konfiguration`. Sie bilden die Grundlage aller weiteren Diskussionen. Nicht vordergründig aber dennoch vorhanden ist die Information, dass wir generell versionierbare Elemente betrachten (Klasse `Version`). Auch die Dokumentation der einzelnen Elemente wird nicht vernachlässigt.

Das grundlegende Modell orientiert sich am Konzept aus Kapitel 4.1. Ausgangspunkt

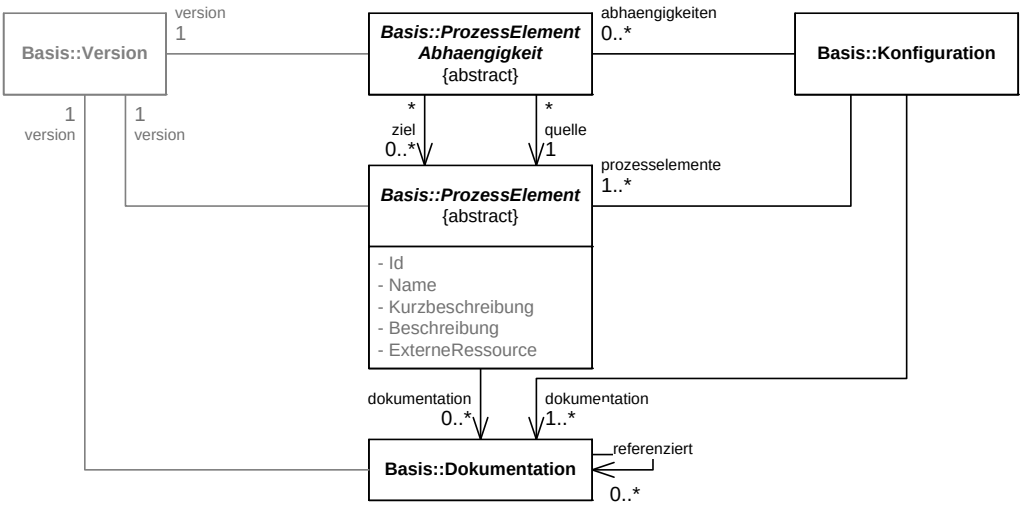


Abbildung 5.2.: Verfeinerung des Pakets Basis des Vorgehensmetamodells

hierfür ist eine Konfiguration, die eine Menge von Prozesselementen und eine Menge von Relationen enthält. Analog zu Kapitel 4.1.2 betrachten wir hier wieder einen Abhängigkeitsgraph G, in dem nun Spezialisierungen der Klasse `ProzessElement` die Knoten bilden und Spezialisierungen der Klasse `ProzessElementAbhaengigkeit` die Beziehungen zwischen den Prozesselementen herstellen (vgl. Abbildung 4.14, unten). Wir modellieren hier die Beziehungstypen als Assoziationsklassen, um sie im Folgenden als Bestandteil einer Konfiguration referenzieren zu können. Ein weiterer Aspekt, der unmittelbar durch das Paket *Basis* abgedeckt wird, ist die *Variationsfähigkeit* von Vorgehensmodellen. Um den Konzepten der Produktlinienentwicklung (vgl. Kapitel 2.3.2) zu folgen, müssen wir *Variationspunkte* im Metamodell vorsehen. Einmal haben wir Variationspunkte durch definierte Soll-Bruchstellen im Gesamtkonzept definiert. Prozesskomponenten zusammen mit Konfigurationen auf der Vorgehensmodellebene liefern hier einen einfachen Mechanismus. Dieser ist jedoch zu grobgranular, sodass wir auch auf der Ebene einzelner Prozesselemente einen entsprechenden Mechanismus vorsehen müssen. In Abbildung 5.3 ist der hier konzipierte Mechanismus gezeigt, der eine erste Spezialisierung der Prozesselementabhängigkeit darstellt (vgl. hierzu auch Kapitel 4.3).

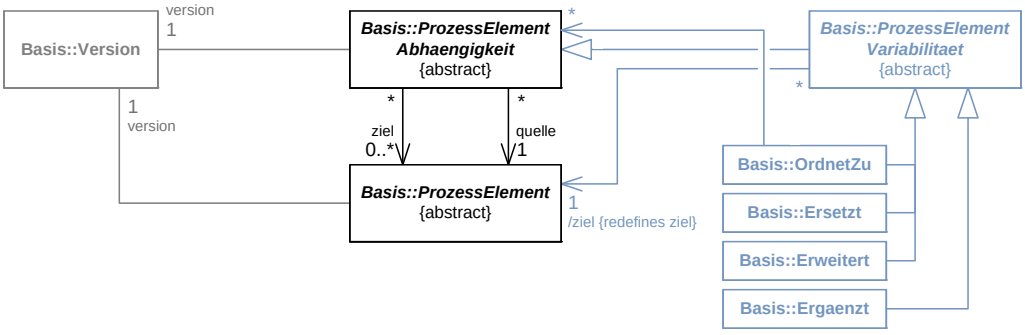


Abbildung 5.3.: Sicht des Pakets Basis: Variabilitätsanteile

Die gezeigten Operationen sind trotzdem noch sehr generisch. In Kapitel 4.3.3 haben wir uns bereits mit der Semantik der Variabilitätsoperationen auseinandergesetzt. Wir vertiefen im Abschnitt 5.1.4 diese Thematik weiter und diskutieren am Beispiel Fähigkeiten und Konsequenzen der Anwendung dieser Operationen.

Vorgehensmetamodell (vereinfacht). In Abbildung 5.4 geben wir eine vereinfachte Sicht des Vorgehensmetamodells[2] an. Diese zeigt die grundlegenden Klassen des Pakets *Basis* und deren Einbettung in das Konzept Prozesskomponente. Jede Prozesskomponente enthält somit eine Menge von Prozesselementen und Abhängigkeiten, die in einer Konfiguration zusammengefasst und in Beziehung gestellt werden. Die Konfiguration einer Prozesskomponente ist privat[3].

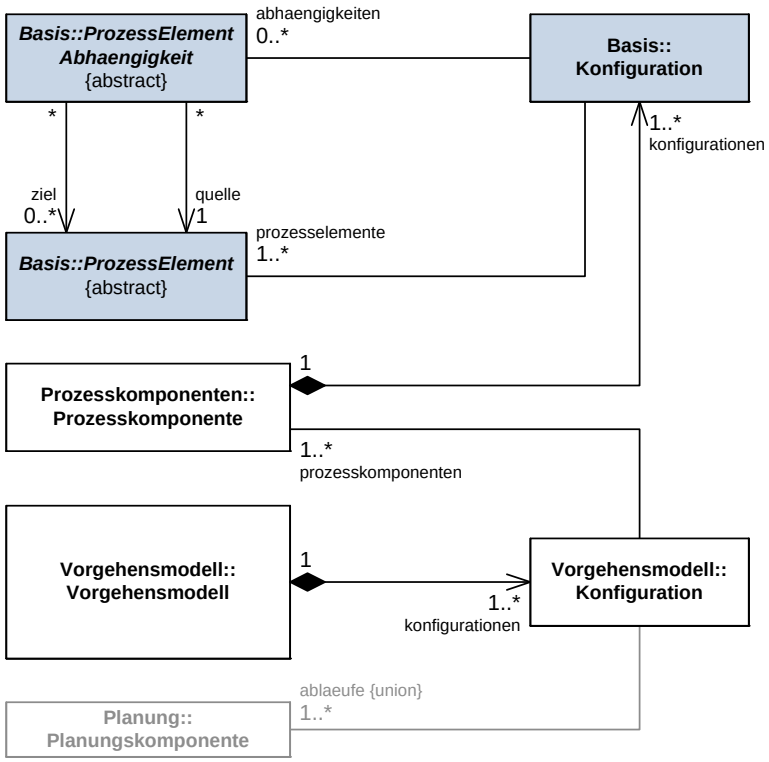


Abbildung 5.4.: Allgemeines Vorgehensmetamodell mit Basisklassen und -beziehungen

Analog zur Bildung der Prozesskomponenten wird auch bei der Erstellung von Vorgehensmodellen beziehungsweise Vorgehensmodellvarianten (die wir hier nicht unterscheiden) verfahren. Die Diskussion der Konfiguration von Prozesskomponenten führen wir jedoch später. Zunächst betrachten wir noch den Aspekt der Dokumentation der einzelnen Prozesselemente.

Dokumentation. Bereits Gnatz [Gna05] befasste sich mit der Modellierung von (strukturierten) Texten. Bartelt und Herold [BH06] griffen diese Problematik wieder auf, diskutierten sie jedoch im Kontext eines allgemeinen Variantenmanagements. Sie stellten heraus, dass hier der Mensch eingebunden werden müsse, wenn im Kontext von zum Beispiel informalen und unstrukturierten Texten auf Varianten von Modellen gearbeitet werden muss. Sotó et al. [SM06c] adressieren das Problem in ähnlicher Weise. In dieser Arbeit abstrahieren wir jedoch von der Problematik von Texten, Beschreibungen etc. und wählen einen pragmatischen Ansatz, indem wir eine *allgemeine Dokumentation* auf der Ebene von Prozesselementen mit modellieren. Jedes Prozesselement beschreibt sich weitgehend selbst (durch entsprechende Attribute nach dem Muster des V-Modells). Als Option sehen wir die Referenzierung weiterer Dokumentationseinheiten vor; ebenso wie die Möglichkeit, dass auch andere Elemente als Prozesselemente Dokumentationseinheiten referenzieren können. Hier ergibt sich die Möglichkeit, die

2 Ergänzend zu der hier gezeigten Modellierung verweisen wir auch auf Anhang C, wo wir eine Interpretation des Metamodells als DSL-Modell im Rahmen des Werkzeugkonzepts anbieten.

3 Wir sehen jedoch prinzipiell mehrere Konfigurationen pro Prozesskomponente vor, um ggf. erweiterten Ansprüchen hinsichtlich der Variationsfähigkeiten nachkommen zu können, vgl. Kapitel 4.3.2

Dokumentation[4] trotz der engen Bindung zum dokumentierten Element separat zu erstellen, zu verwalten und zu organisieren. Ein (entfernt) ähnliches Konzept finden wir im V-Modell XT mit den so genannten Textbausteinen als abstrakten Modellelementen und im MSF, wo die Prozessdokumentation komplett eigenständig und losgelöst von der Vorgehensmodellstruktur ist.

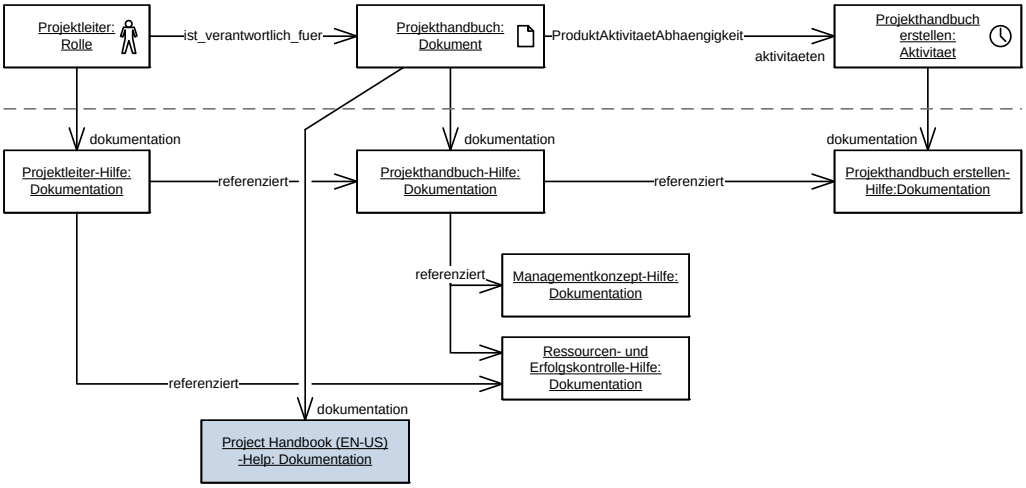


Abbildung 5.5.: Beispiel für die Trennung von Elementstruktur und Dokumentation eines Vorgehensmodells

Die Abbildung 5.5 zeigt als Vorgriff in Form einer Instanziierung des Metamodells (inkl. der Submodelle) ein Beispiel für die Separation von Elementstrukturen und der Dokumentation am Beispiel ausgewählter V-Modell XT Inhalte. Die Vorgehensmodellstruktur, also die Struktur der Prozesselemente, ist im oberen Teil der Abbildung zu sehen; der untere Teil zeigt eine separate Dokumentationsstruktur. So ist es beispielsweise möglich, eine minimale (abstrakte, generische) Dokumentation für ein Vorgehensmodell zu erstellen und diese dann entweder anzupassen oder problemspezifisch zu ergänzen, ohne dass die Elementstrukturen geändert werden müssen.

Vorteile einer separierten Dokumentation

Ist es beispielsweise erforderlich, dass ein Vorgehensmodell multilingual zur Verfügung gestellt werden muss, kann durch die Separation der Vorgehensmodell-Elementstruktur und der Dokumentation ein Mehr an Flexibilität erzielt werden. Die Elementstrukturen des Vorgehensmodells müssen hier nur einmal erstellt werden, während die Dokumentation mehrsprachig zur Verfügung gestellt werden kann.

Eine solche Option ist wünschenswert und notwendig. Im Kontext Englischübersetzung des V-Modell XT tritt der Bedarf ebenso akut zu Tage, wie im Kontext von CollabXT. Im letzteren sind darüber hinaus sogar komplette sprachspezifische Infrastrukturen erforderlich, was sich negativ auf Wartung und Pflege auswirkt.

Ergänzend dazu kann die Dokumentation im Rahmen eines Anpassungsprozesses einfach ergänzt werden. Je nach Formalisierungsgrad des betrachteten Vorgehensmodells können über den Beziehungstypen zwischen der Dokumentation und den dokumentierten Elementen *Konsistenzbedingungen* definiert werden, um beispielsweise eine gewisse „Mindestdokumentation" zu erzwingen.

4 Zu beachten ist beim Terminus *Dokumentation*, dass es sich nicht zwangsläufig um ein Dokument, wie im Produktmodell definiert (Abschnitt 5.1.2), handeln muss. Eine Dokumentation kann beispielsweise eine Checkliste oder eine allgemeine Verfahrensbeschreibung zum Beispiel für eine Aktivität sein.

5.1.2. Submodelle und Komponentenbildung

In diesem Abschnitt befassen wir uns mit der grundlegenden Strukturmodellierung von Prozesskomponenten, die als Bausteine unseres Vorgehensmodellkonzepts fungieren. Wir stützen uns dabei auf den Begriffsbildungen aus Kapitel 2.4 ab, um einen inhaltlichen Rahmen zu schaffen. Nach der Modellierung der Strukturen mit allen relevanten Entitätstypen konzentrieren wir uns auf die Abhängigkeitsstrukturen. Wir greifen dabei die Diskussionen aus Kapitel 4 wieder auf und konzentrieren uns neben internen Abhängigkeiten schwerpunktmäßig auf Abhängigkeiten zwischen Prozesskomponenten. Die Inhalte dieses Abschnitts liefern die Grundlagen für den Rest dieses Kapitels. Neben der Modellierung auf Basis der UML 2 geben wir in weiten Teilen eine entsprechende Formalisierung mit an. Wo für das Verständnis erforderlich geben wir auch Beispiele an, in denen wir auf Inhalte zurückgreifen, die wir auf Basis des V-Modell XT als Referenz erarbeitet haben.

Prozesskomponenten

Abbildung 5.6 zeigt eine vereinfachte Darstellung der Prozesskomponente[5]. Im Folgenden gehen wir detailliert auf Prozesskomponenten ein und nehmen die notwendigen Detaillierungen vor. Prozesskomponenten definieren wir als eigenständige Einheiten, die autonom verwendbar und (weiter-)entwickelbar sind [HR02, Sie04, VAC+05]. Gemäß unserer Modellierung in Kapitel 4 definieren wir Prozesskomponenten zunächst als Container für Prozesselemente. Wir konkretisieren hier jedoch und geben eine Prozesskomponente in einer Form an, wie sie für Vorgehensmodelle üblich ist (Abbildung 5.6).

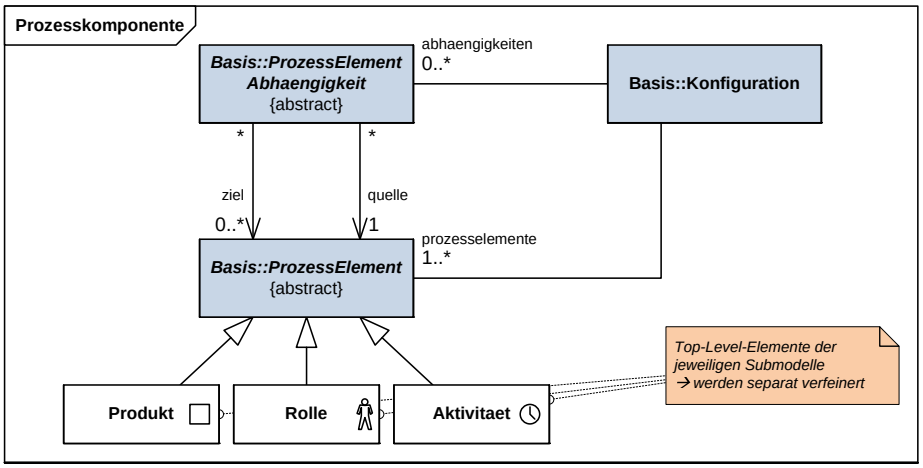


Abbildung 5.6.: Metamodell einer Prozesskomponente (ohne Verfeinerungen)

Wir orientieren uns bei der Definition von Prozesskomponenten am Konzept *Vorgehensbaustein* (Kapitel 3) des V-Modell XT. Das Method Plug-In von EPF/UMA stellt ebenfalls eine Option dar, jedoch sind wir hier bestrebt, möglichst einfache und flache Strukturen zu modellieren, da wir dies als Grundvoraussetzung für kontrollierbares Handling sehen. Nichtdestotrotz finden sich Bestandteile beider Konzepte hier wieder: die Integrationsdichte des V-Modells und die physische Modularität von EPF/UMA. Das Modell in Abbildung 5.6 zeigt insbesondere im oberen Teil die Nutzung der im letzten Abschnitt beschriebenen Basiskonzepte. Das Metamodell der internen Struktur einer Prozesskomponente ist durch die Vielzahl von *explizit* modellierten Beziehungen für die Konfiguration zunächst sehr komplex. Die Verschiebung der Komplexität in diesen

5 Die hier modellierte Prozesskomponente ist nur eine mögliche Struktur, die wir jedoch im Rahmen dieser Arbeit bereits vordefinieren. Nach dem hier vorgestellten Konzept lassen sich weitere Spezialisierungen beziehungsweise Verknüpfungen von Prozesselementen vornehmen und in ein Vorgehensmodell integrieren. Im Anhang B zeigen wir hierfür ein Beispiel.

Modellteil erlaubt uns aber im Folgenden einfacher strukturierte Modelle zu instanziieren. Wir verfeinern im Folgenden schrittweise und zeigen dies anhand von Beispielen.

In Abbildung 5.6 ist als Verfeinerungsaspekt für die Prozesselemente die *Ontologie für Vorgehensmodelle* (vgl. Kapitel 2.4) enthalten. Wir verwenden den Begriff wie Gnatz [Gna05] und orientieren uns auch strukturell an seiner Modellierung. Abbildung 5.7 zeigt die drei wesentlichen Submodelle für Produkte, Rollen und Aktivitäten – jeweils durch eine Klasse repräsentiert. Zwischen den Submodellen bestehen Beziehungen, die beispielsweise durch das Rollenmodell hergestellt werden können.

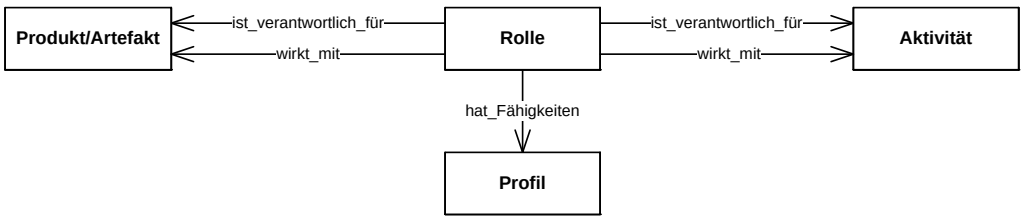


Abbildung 5.7.: Ontologie für Vorgehensmodelle im Kontext einer Prozesskomponente

Im Folgenden gehen wir auf die drei Submodelle vertiefend ein und verfeinern dadurch die Modellierung der Prozesskomponente. Nicht weiter verfeinern werden wir das Rollenmodell, das bereits in einfacher Form der Abbildung 5.7 entnommen werden kann. Wichtig sind uns an dieser Stelle nur die Beziehungstypen, die wir im Kontext der Modellierung der Submodelle ebenfalls für die Rollen betrachten wollen.

Produktmodell

Wir führen zunächst die beiden komplexen Submodelle für Produkte und Aktivitäten ein. Die Abbildung 5.8 zeigt die Verfeinerung des Metamodells aus Abbildung 5.4 für den Kontext der Produkte. Das Produktmodell wird aufbauend auf der zentralen

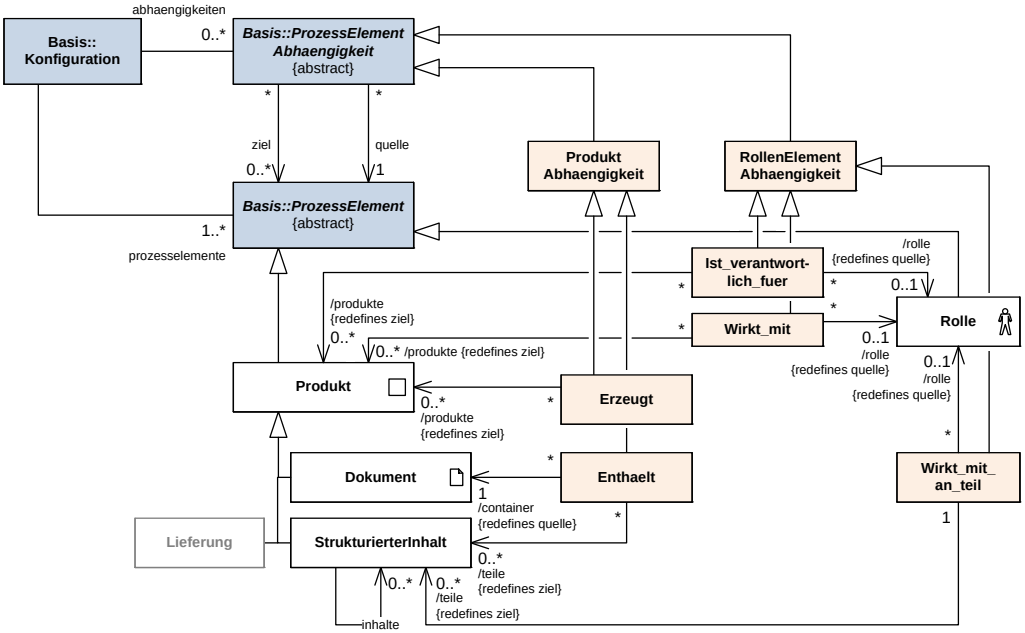


Abbildung 5.8.: Verfeinerung des Metamodells für den Kontext Produktmodell

Klasse `Produkt` (Kind von `ProzessElement`) modelliert. Über Produkten lassen wir

Beziehungstypen der Klasse `ProduktAbhaengigkeit`[6] (Kind von `ProzessElementAbhängigkeit`) zu. Unser Produktmodell sieht im Wesentlichen nur allgemeine Produkte (Klasse `Produkt`), Dokumente (Klasse `Dokument`) und Lieferungen (Klasse `Lieferung`) vor. Wir orientieren uns für die Modellierung des Produktmodells an Gnatz [Gna05] und EPF/UMA (Kapitel 3.2.3). Für Dokumente modellieren wir bereits einfache Strukturen (Klasse `StrukturierterInhalt` als Kind von `Produkt`). Eine ähnlich verfeinerte Modellierung wäre zum Beispiel für Lieferung ebenfalls möglich, jedoch führen wir sie hier nicht durch. Die Verknüpfungen zwischen Elementen des Produktmodells werden über die Produktabhängigkeiten modelliert. Weiterhin hat das Produkt Beziehungen zum Rollenmodell. Über einen eigenen Beziehungstyp `RollenElementAbhaengigkeit` modellieren wir hier analoge Verknüpfungsmechanismen. Über die innere Struktur der einzelnen Klassen treffen wir an dieser Stelle keine weitere Aussage, verfeinern jedoch noch die Betrachtung der Beziehungstypen. Im Anschluss daran geben wir ein kurzes Beispiel.

Beziehungstypen im Produktmodell. Wir gehen nun detaillierter auf die Beziehungstypen ein, die aus den Prozesselementabhängigkeiten abgeleitet werden und konkretisieren diese zunächst für das Produktmodell. Wir geben auch hier nur ein Minimalmodell an, das unseren unmittelbaren Anforderungen genügt[7]. Wir modellieren das Produktmodell derart, dass es abgeschlossen ist, d. h. dass jeder durch das Produktmodell gegebene Beziehungstyp ausschließlich über Elementen des Produktmodells definiert ist.

Muster für Elemente und Abhängigkeiten

Wir verwenden für die Modellierung grundsätzlich folgendes Muster: Zu allen betrachteten Elementtypen modellieren analoge Beziehungstypen (vgl. Descriptoren aus EPF/UMA, Kapitel 3.2.3). Derart modellierte Beziehungen sind abgeschlossen, d. h. sie referenzieren nur Elemente des jeweiligen Submodells. Für die Kopplung von verschiedenen Submodellen modellieren nach demselben Muster spezialisierte Beziehungstypen.

Produktabhängigkeiten. In unserem Modell sind Produktabhängigkeiten Spezialisierungen von Relationen zwischen Prozesselementen vom Typ Produkt. Produktabhängigkeiten lassen sich vielfältig modellieren, wie zum Beispiel im V-Modell XT [RH06] oder in [Gna05]. Um die mögliche Breite der Begriffsdefinition zu verdeutlichen, verweisen wir auf den folgenden Auszug aus dem V-Modell XT-Glossar:

Produktabhängigkeiten im V-Modell XT

Eine Produktabhängigkeit beschreibt eine Konsistenzbedingung zwischen zwei oder mehreren Produkten. Dabei kann eine Produktabhängigkeit sowohl innerhalb eines Vorgehensbausteins als auch zwischen Produkten verschiedener Vorgehensbausteine bestehen.

Man unterscheidet Tailoring-Produktabhängigkeiten, erzeugende [...], strukturelle [...] und inhaltliche Produktabhängigkeiten. Alle diese Arten von Produktabhängigkeiten können relevante Produktabhängigkeiten sein. [RH06]

Das V-Modell XT als produktorientiertes Vorgehensmodell misst den Produktabhängigkeiten einen großen Stellenwert bei. Wir abstrahieren und reduzieren die Menge der relevanten Abhängigkeitstypen für unser Modell derart, dass wir nur die Erzeugung und strukturelle Bezüge modellieren (vgl. Abbildung 5.8). Im Folgenden spezifizieren wir die beiden Beziehungstypen, die wir aus der allgemeinen Produktabhängigkeit (Klasse: `ProduktAbhaengigkeit`) ableiten.

6 Diesem Beziehungstyp hat das Konzept der Produktabhängigkeiten des V-Modell XT Pate gestanden. Wir verwenden dieses Konzept hier und überführen es nach konsequenter Weiterentwicklung gesamtheitlich auf das ganze Vorgehensmetamodell.

7 Bei einer Erweiterung eines der Submodelle eines Vorgehensmodells sollte auch eine Anpassung der Assoziationstypen in Betracht gezogen werden. Dies entspricht dann einer *strukturellen Anpassung* im Sinne von Kapitel 2.2.1.

Beziehunstyp `Erzeugt`: Für den Beziehungstyp `Erzeugt`, der als Klasse modelliert von `ProduktAbhaengigkeit` erbt, legen wir ein Verhalten fest, das sowohl der einfachen als auch der erweiterten Prozesselementabhängigkeit genügt. Abbildung 5.8 illustriert dies analog auch für den Beziehungstyp `Enthaelt`[8]. Für den Beziehungstyp `Erzeugt` und $x, y \in$ *Produkte* definieren wir:

$$\text{erzeugt}\,(x) = y, \text{bzw. erzeugt}\,(x, y) \tag{5.1}$$

Das Produkt x heißt der Terminologie des V-Modell XT folgend *erzeugendes Produkt*, y heißt *erzeugtes Produkt*. Eine „Selbsterzeugung", also *erzeugt* (x, x), verbieten wir. In Abbildung 5.8 ist darüber hinaus auch zu sehen, dass für den Beziehungstyp `Erzeugt` das Assoziationsende `produkte` die Kardinalität $0..*$ aufweist. Dies bedeutet, dass wir hier ein erweiterte Prozesselementabhängigkeit vorliegen haben und somit *erzeugt* auch auf eine Menge $Y \subseteq$ *Produkte* anwenden können, also:

$$\text{erzeugt}\,(x) = Y, \text{bzw. erzeugt}\,(x, \{y_1, \ldots, y_n\}) \tag{5.2}$$

Dies ist dann, wie in Kapitel 4.1.2 besprochen, gleichwertig einer Darstellung als Prädikatenmenge (wie wir sie auch in der Konfiguration verwenden können):

$$\begin{aligned}&\text{erzeugt}\,(x, y_1)\\ &\text{erzeugt}\,(x, y_2)\\ &\ldots\\ &\text{erzeugt}\,(x, y_{n-1})\\ &\text{erzeugt}\,(x, y_n)\end{aligned}$$

Beziehungstyp `Enthaelt`: Analog verfahren wir für den Beziehungstyp `Enthaelt`. Sei $x \in$ *Dokumente* und seien $y_1, \ldots, y_n \in$ *StrukturierterInhalt*, dann gilt:

$$\text{enthaelt}\,(x) = \{y_1, \ldots, y_n\}, \text{bzw. enthaelt}\,(x, \{y_1, \ldots, y_n\}) \tag{5.3}$$

Das Dokument x nennen wir *Container*. Auf der Grundlage von Formel 5.3 definieren wir dabei strukturierte Dokumente.

Innerhalb einer Prozesskomponente sind die Produktabhängigkeiten in der Regel vollständig. Den Aspekt der Abhängigkeiten über die Grenzen einer Prozesskomponente hinaus vertiefen wir im weiteren Verlauf dieses Kapitels noch.

Rollen und deren Einbindung im Produktmodell. Rollen sind die Elemente eines Vorgehensmodells, die Personen einbringen. Die Erfahrung hat gezeigt, dass es einerseits erforderlich ist, Personen, die eine Rolle besetzen, über ihre Befugnisse, andererseits aber auch über ihre Aufgaben zu informieren. Üblicherweise werden Rollen enger mit Aktivitäten in Beziehung gesetzt. Lediglich das V-Modell XT setzt konsequent auf eine Kopplung von Rollen und Produkten. Für Prozesskomponenten nach der Modellierung aus Abbildung 5.6 setzt sich die Menge der in einer Prozesskomponente zu betrachtenden Prozesselemente zunächst wie folgt zusammen:

$$Pe = \text{Rollen} \cup \text{Produkte} \cup \text{Aktivitaeten} \tag{5.4}$$

Die einzelnen Teilmengen sind dabei *disjunkt*. Da wir in dieser Arbeit sowohl Rollen, Produkte und Aktivitäten auf der selben Ebene konkretisieren und dabei insbesondere Produkte und Aktivitäten gleich behandeln, modellieren wir die Beziehungstypen für Rollen (`Ist_verantwortlich_fuer`, `Wirkt_mit` und `Wirkt_mit_an_teil`) analog, siehe Formeln 5.5 und 5.6. Die Beziehungstypen sind gerichtet von Elementen aus *Rollen* $\subseteq$ *Pe* zu Elementen entweder aus *Produkte* oder *Aktiviaeten*. Wir definieren daher:

$$\text{ist\_verantwortlich\_fuer} \subseteq \text{Rollen} \times (\text{Produkte} \cup \text{Aktivitaeten}) \tag{5.5}$$

$$\text{wirkt\_mit} \subseteq \text{Rollen} \times (\text{Produkte} \cup \text{Aktivitaeten}) \tag{5.6}$$

$$\text{wirkt\_mit\_an\_teil} \subseteq \text{Rollen} \times \text{StrukturierterInhalt} \tag{5.7}$$

8 In den Abbildungen der Metamodelle beziehungsweise deren Ausschnitte wie zum Beispiel Abbildung 5.8 sind die Klassen, die Beziehungstypen modellieren, immer farblich hervorgehoben.

Diese allgemeinen Definitionen genügen jedoch noch nicht vollständig, da eine Rolle nun mit Produkten und Aktivitäten in Beziehung stehen kann. Dies kann in einigen Szenarios durchaus sinnvoll sein, jedoch sind auch Vorgehensmodelle wie das V-Modell XT zu berücksichtigen, die eine Gewichtung von Rollen und deren Beziehungen zu Ergebnistypen vornehmen. Wir geben daher auch eine mögliche Verfeinerung der Assoziationen für typechte Teilmengen[9] an. Sei r eine Rolle, für die die Assoziationen `Ist_verantwortlich_fuer` und `Wirkt_mit` über Teilmengen von Prozesselementen definiert sind:

$$\text{ist\_verantwortlich\_fuer} \subseteq \text{Rollen} \times (\text{Produkte} \cup \text{Aktivitaeten}) \text{ und es gilt:}$$
$$\text{ist\_verantwortlich\_fuer}|_{\text{Produkte}} \subseteq \text{Rollen} \times \text{Produkte} \tag{5.8}$$
$$\text{ist\_verantwortlich\_fuer}|_{\text{Aktivitaeten}} \subseteq \text{Rollen} \times \text{Aktivitaeten} \tag{5.9}$$

und analog für die Assoziation `Wirkt_mit`:

$$\text{wirkt\_mit} \subseteq \text{Rollen} \times (\text{Produkte} \cup \text{Aktivitaeten}) \text{ und es gilt:}$$
$$\text{wirkt\_mit}|_{\text{Produkte}} \subseteq \text{Rollen} \times \text{Produkte} \tag{5.10}$$
$$\text{wirkt\_mit}|_{\text{Aktivitaeten}} \subseteq \text{Rollen} \times \text{Aktivitaeten} \tag{5.11}$$

dann gilt für ein $n+1$-stelliges Prädikat *ist_verantwortlich_fuer* $(r, \{p_1, \ldots, p_n\})$, dass die Elemente $p_1, \ldots, p_n$ nur von einem Typ sind. Also:

$$\forall p_i \in \text{ist\_verantwortlich\_fuer}|_{\text{Produkte}}\ (r, \{p_1, \ldots, p_n\}) : p_i \in \text{Produkte} \tag{5.12}$$
$$\forall p_i \in \text{ist\_verantwortlich\_fuer}|_{\text{Aktivitaeten}}\ (r, \{p_1, \ldots, p_n\}) : p_i \in \text{Aktivitaeten} \tag{5.13}$$

Beispiel: Instanziierung des Produktmodells. Wir geben ein kleines Beispiel an, mit dem wir eine Instanziierung des Produktmodells mit Inhalten des V-Modell XT zeigen (Abbildung 5.9). Im oberen Teil der Abbildung ist der Ausschnitt des V-Modell XT gezeigt, auf den wir uns beziehen. Im unteren Teil die Instanziierung des Produktmodells mit einigen weiter gehenden Elementen.

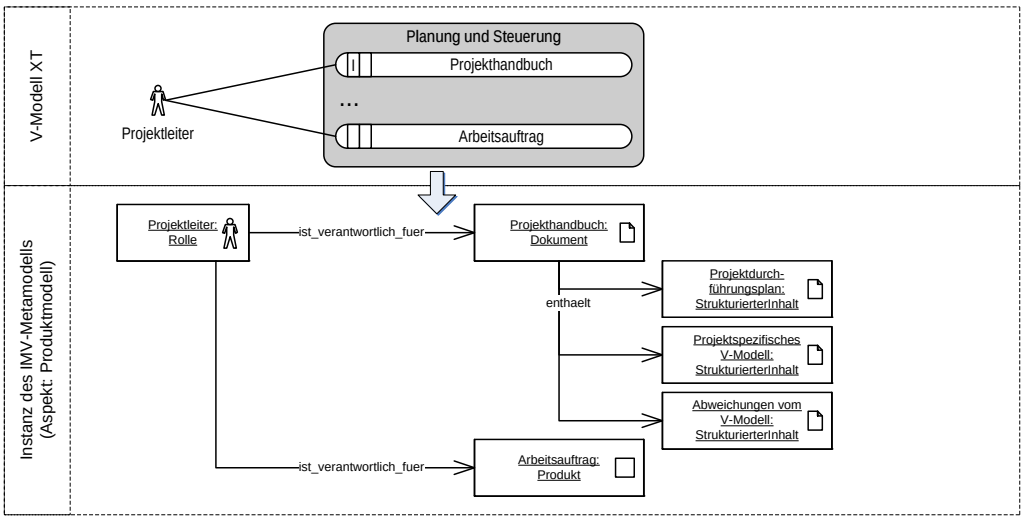


Abbildung 5.9.: Beispielhafte Anwendung des Produktmodells für Auszüge aus dem V-Modell XT

Im Beispiel ist mithilfe unseres Produktmodells ein Auszug aus dem V-Modell XT remodelliert. Zuerst ist das *Projekthandbuch* zu nennen, das wir als Dokument modellieren. Das Projekthandbuch kann laut V-Modell Themen enthalten, wie zum Beispiel *Projektdurchführungsplan*, *Projektspezifisches V-Modell* oder *Abweichungen vom V-Modell* etc. Diese Themen modellieren wir als strukturierte Inhalte und verknüpfen sie mithilfe der

9 Unter einer typechten Teilmenge verstehen eine Teilmenge von Prozesselementen, in der nur Elemente eines Typs enthalten sind, zum Beispiel ausschließlich Produkte.

Assoziation „*Enthaelt*". Ebenfalls betrachten wir das V-Modell-Produkt *Arbeitsauftrag*, welches wir nicht zwangsläufig als Dokument modellieren , sondern beispielsweise als Formular (siehe [KK07]).

Aktivitätsmodell

In unserem Vorgehensmetamodell definieren wir ebenfalls ein minimales Aktivitätsmodell (Abbildung 5.10). Das Aktivitätsmodell haben wir sehr einfach gehalten, jedoch kann es analog zum Produktmodell erweitert werden. Zentrale Klasse ist die `Aktivitaet` als Kind von `ProzessElement` (vgl. Abbildung 5.6). Für `Aktivitaet` definieren wir nur zwei Kinder `Aufgabe` und `Arbeitspaket`. Wir verweisen hier auf das Metamodell des V-Modell XT, das eine ähnliche explizite Abstufung in Aktivitäten und Teilaktivitäten vorsieht. Eine derartige Beschränkung wollen wir hier nicht einführen; auch betrachten wir alle Aufgaben als gleichberechtigt. Arbeitspakete definieren wir als Container für mehrere Aufgaben (Kardinalität: $2..*$).

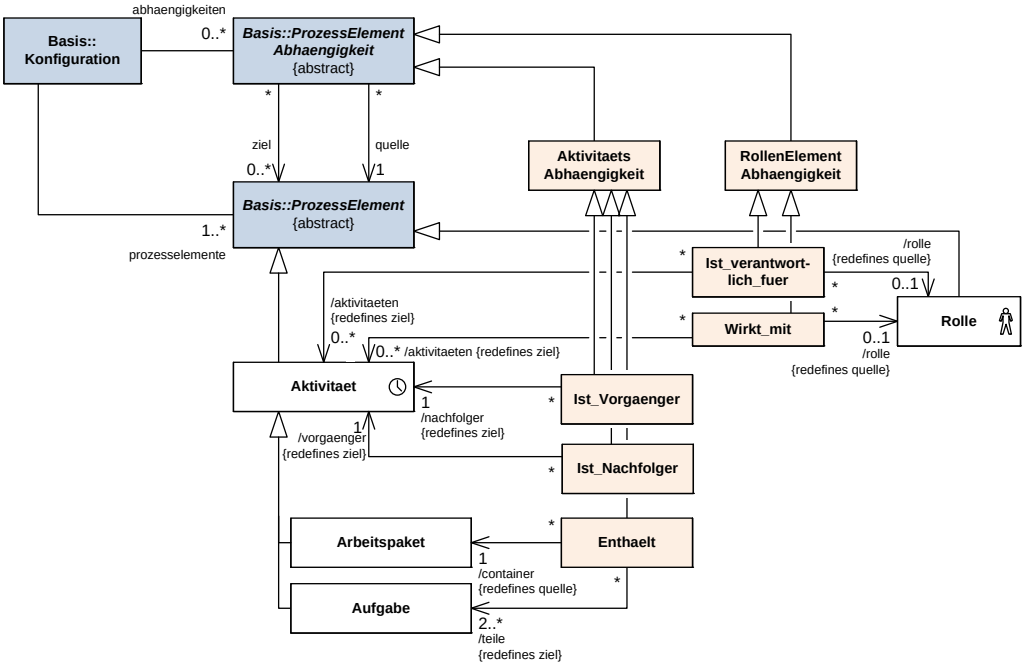


Abbildung 5.10.: Verfeinerung des Metamodells für den Kontext Aktivitätsmodell

Analog zum Produktmodell ist auch das Aktivitätsmodell abgeschlossen. Verknüpfungen zu Prozesselementen, die nicht vom Typ `Aktivitaet` oder spezieller sind, werden nur durch externe Konfiguration gestattet (wie zum Beispiel auch bei den Produkten gezeigt – Abbildung 5.10).

Beziehungstypen im Aktivitätsmodell. Auf ähnlicher Ebene wie Produktabhängigkeiten modellieren wir auch *Aktivitätsabhängigkeiten*. Vorgehensmodelle wie das V-Modell XT messen Aktivitäten vergleichsweise wenig Bedeutung zu, da sie Produkte favorisieren. Andere Vorgehensmodelle, wie beispielsweise OpenUP oder MSF (vgl. für beide Kapitel 2.1) favorisieren hingegen Aktivitäten und liefern hier eine entsprechend ausführliche Modellierung. Da wir in dieser Arbeit sowohl Aktivitäten als auch Produkte vom selben Basiskonzept ableiten, behandeln wir sie auch gleichberechtigt und definieren Aktivitätsabhängigkeiten (vgl. Abbildung 5.10). Für Aktivitäten modellieren wir folgende grundlegenden Beziehungstypen als Spezialisierung der Klasse `AktivitaetsAbhaengigkeit` (Kind von `ProzessElementAbhaengigkeit`):

Beziehungstyp `Enthaelt`: Den Beziehungstyp `Enthaelt` modellieren wir analog zum Beziehungstyp `Enthaelt` über Produkten[10]. Als Elemente für diese Beziehung stehen uns gemäß Abbildung 5.10 Aufgaben und Arbeitspakete zur Verfügung, wobei die Aufgaben den strukturierten Inhalten und die Arbeitspakete den Dokumenten entsprechen. Wir definieren die Beziehung für $x \in$ *Arbeitspakete* und $y_1, \ldots, y_n \in$ *Aufgaben* also wie folgt:

$$\text{enthaelt}(x) = \{y_1, \ldots, y_n\}, \text{bzw. enthaelt}(x, \{y_1, \ldots, y_n\}), n \geq 2 \tag{5.14}$$

Beziehungstypen `Ist_Nachfolger` und `Ist_Vorgaenger`: Für die beiden Beziehungstypen `Ist_Vorgaenger` und `Ist_Nachfolger` verlangen wir, dass es sich um gerichtete, einfache Assoziationen handelt. Für $x, y \in$ *Aktivitaeten* gilt also:

$$\text{ist\_vorgaenger}(x) = y, \text{bzw. ist\_vorgaenger}(x, y) \tag{5.15}$$

$$\text{ist\_nachfolger}(x) = y, \text{bzw. ist\_nachfolger}(x, y) \tag{5.16}$$

Wir können diese beiden Beziehungstypen so einfach gestalten, da wir Vorgänger-/Nachfolgerrelationen direkt über Aktivitäten definieren. Da wir Aktivitäten zum Beispiel auch als Arbeitspakete spezialisieren, führen wir damit implizit einen Kompositionsoperator für Aktivitäten ein. Betrachten wir folgendes Beispiel: Seien $p_1, p_2 \in$ *Arbeitspakete* und $a_1, a_2, a_3, a_4, a_5 \in$ *Aufgaben*. Gelte ferner folgende Konfiguration: *enthaelt* $(p_1, \{a_1, a_5\})$ und *enthaelt* $(p_2, \{a_2, a_3, a_4\})$, dann gilt:

$$\begin{aligned}&\text{ist\_nachfolger}(p_1, p_2) \Rightarrow \\ &\forall a_i \in \text{enthaelt}(p_2) . \forall x_j \in \text{enthaelt}(p_1) : \text{ist\_nachfolger}(x_j, a_i)\end{aligned} \tag{5.17}$$

Für den Beziehungstyp `Ist_Vorgaenger` können wir eine analoge Modellierung angeben. Relationen der Containerelemente gelten auch für die enthaltenen Kindelemente. Analog zu den Produktabhängigkeiten sind auch die Aktivitätsabhängigkeiten innerhalb einer Prozesskomponente in der Regel vollständig.

Beispiel: Instanziierung des Aktivitätsmodells. Auch für die Instanziierung des Aktivitätsmodells wollen wir ein kleines Beispiel angeben. Der Rückgriff auf das V-Modell XT ist hier jedoch nur eingeschränkt möglich, da die Aktivitäten des V-Modells erst bei der Plangenerierung mit dem Projektassistenten relevant werden.

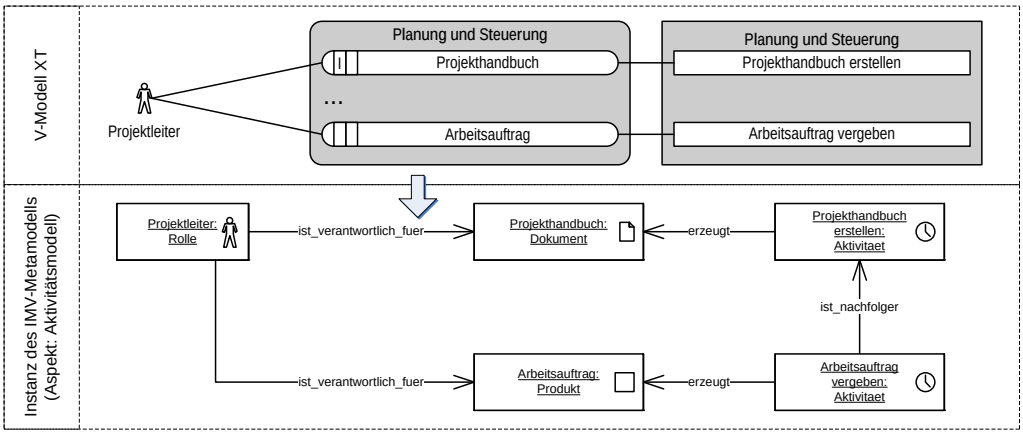


Abbildung 5.11.: Beispielhafte Anwendung des Aktivitätsmodells für Auszüge aus dem V-Modell XT

Das Beispiel greift die den beiden Produkten *Projekthandbuch* und *Arbeitsauftrag* zugeordneten Aktivitäten *Projekthandbuch erstellen* und *Arbeitsauftrag vergeben* auf. Ein direkte Verknüpfung der Aktivitäten sieht das V-Modell nicht vor. Abhängigkeiten und

10 Im Rahmen einer Implementierung muss hier natürlich eine Eindeutigkeit hergestellt werden. Im Rahmen des Werkzeugkonzepts in Anhang C gehen wir darauf näher ein.

Nachfolgerrelationen werden durch die Ausgestaltung der Entscheidungspunkte und der durch diese definierten Produktvolumina ermittelt. Direkte Verknüpfungen zwischen den Aktivitäten gibt es aber auch an dieser Stelle nicht. Die Überführung auf das IMV-Metamodell lässt beispielsweise eine Verknüpfung der beiden gezeigten Aktivitäten zu, sodass durch

$$\text{ist\_nachfolger}\,(\text{Arbeitsauftrag vergeben}, \text{Projekthandbuch erstellen})$$

ausgedrückt werden kann, dass zuerst das Projekthandbuch zu erstellen ist, bevor Arbeitsaufträge vergeben werden können. Dies stellt ein zu den erzeugenden Produktabhängigkeiten des V-Modells gleichwertiges, ablauforientiertes Konzept dar. Es kann zum Beispiel auch dazu verwendet werden, um weiter gehende Konsistenzbedingungen zu formulieren. Dies stellt einen Schritt in Richtung automatische, prüfbare Plangenerierung dar, wie sie auch Gnatz schon betrachtet hat. Die Integrationsoptionen dieser beiden Ansätze diskutieren wir in Anhang B.

Verknüpfung von Produkten und Aktivitäten

Die Beziehungstypen zwischen Prozesselementen erzeugen jeweils die Strukturen innerhalb der einzelnen Submodelle. Jedoch müssen die Submodelle auch untereinander in Beziehung gesetzt werden, um eine sinnvolle Ausgestaltung und letztendlich auch Anwendung eines Vorgehensmodells zu ermöglichen. In Abbildung 5.12 haben wir die Modellierung der zu betrachtenden Beziehungstypen vorgenommen.

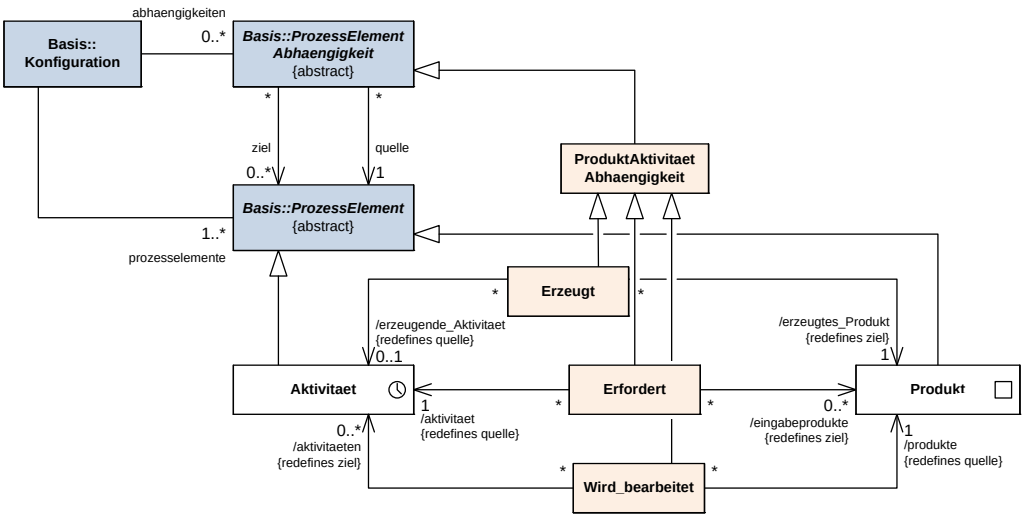


Abbildung 5.12.: Verfeinerung für Produkt- und Aktivitätsabhängigkeiten

Hierfür spezialisieren wir den Beziehungstyp `ProzessElementAbhaengigkeit` einmal für die Verknüpfung von Produkten und Aktivitäten (`ProduktAktivitaetAbhaengigkeit`) und weiterhin für die Verknüpfung eben dieser Elemente mit Rollen (`RollenElementAbhaengigkeit`, siehe weiter oben). Wir spezialisieren `ProduktAktivitaetAbhaengigkeit` in drei konkreten Beziehungstypen:

- Beziehung: `Erzeugt`
- Beziehung: `Erfordert`
- Beziehung: `Wird_bearbeitet`

Diese drei in dieser Arbeit definierten Beziehungstypen stellen einen minimalen Umfang dar, um Produkterstellung (`Erzeugt`), Produktfluss (`Erfordert`) und Produktbearbeitung (`Wird_bearbeitet`) zu ermöglichen. Von den grundlegenden Fähigkeiten orientieren wir uns wieder schwerpunktmäßig am V-Modell XT, schließen jedoch anders lautende Spezialisierungen auf der Grundlage anderer Vorgehensmodellphilosophien nicht aus.

Beziehungstyp: `Erzeugt` – Der Beziehungstyp `Erzeugt` legt eine „Konstruktionsoperation" für ein Produkt fest. Anders als die Produktabhängigkeit `Erzeugt` wird hier konkret festgelegt, welche (genau eine) Aktivität zur Erstellung eines Produkts führt. Die Produkterstellung wird somit beispielsweise im Kontext einer Projektplangenerierung zur Planungsgröße (vgl. [Mün01, Gna05]). Es gilt also für eine Aktivität $a \in \mathit{Aktivitaeten}$ und zwei Produkte $p_1, p_2 \in \mathit{Produkte} \wedge p_1 \neq p_2$:

$$\text{erzeugt}\,(a, p_1) \Rightarrow \neg\exists p_2 : \text{erzeugt}\,(a, p_2) \tag{5.18}$$

Aufgrund der Kompositionalität von Aktivitäten können wir jedoch auch mehrere Produkterzeugungen pro Aktivität modellieren, indem wir die Containereigenschaften des *Arbeitspakets* nutzen. Einzelne *Aufgaben* $af_1, \ldots, af_i$ (als spezielle Aktivitäten) können jeweils ein Produkt $p_1, \ldots, p_i$ erzeugen. Es gilt also:

$$\begin{aligned}
&\text{erzeugt}\,(af_1, p_1)\\
&\text{erzeugt}\,(af_2, p_2)\\
&\ldots\\
&\text{erzeugt}\,(af_{n-1}, p_{n-1})\\
&\text{erzeugt}\,(af_n, p_n)
\end{aligned}$$

Fassen wir diese in einem Arbeitspaket zusammen: $\mathit{enthaelt}\,(ap, \{af_1, \ldots, af_i\})$, so ist die Produkterzeugung im Rahmen des Arbeitspakets zu erbringen, womit eine (komposite) Aktivität auch mehrere Produkte erzeugen kann.

Anders herum muss ein Produkt nicht unbedingt von einer Aktivität erzeugt werden. Analog zum V-Modell XT sprechen wir dann von einem *externen* Produkt[11]. Ein Beispiel für ein derartiges Konstrukt findet sich beispielsweise dort, wo organisationsübergreifend Standardformulare zu erstellen sind, der jeweilige Erstellungsprozess jedoch spezifisch durch eine Anpassung geregelt werden muss.

Ein weiterer Fall, der berücksichtigt werden muss, ist die Instanziierung einer Beziehung zwischen einem *Arbeitspaket* und einem Produkt. Analog zur Vorgänger- und Nachfolgerrelation gilt auch hier, dass Beziehungen des Containers transitiv für die Containerelemente gelten. Für ein Arbeitspaket sind dann $n + 1$ Aktivitätstypen im Kontext Produkterzeugung zu betrachten, wobei einer durch das Arbeitspaket ausgezeichnet ist. Für die Erzeugung eines Produkts $p \in \mathit{Produkte}$ durch ein Arbeitspaket $A \in \mathit{Arbeitspakete}$ mit $\mathit{enthaelt}\,(A, \{a_1, \ldots, a_n\})$ gilt daher für $\mathit{erzeugt}\,(A, p)$:

$$\text{fertiggestellt}\,(p) \Leftrightarrow \text{abgeschlossen}\,(A) := \bigwedge_{\forall i} \text{abgeschlossen}\,(a_i) \tag{5.19}$$

Alle Aktivitäten (Aufgaben) a_i werden somit im Kontext eines Projektplans unter einem Sammelvorgang zusammengefasst. Das Prädikat *fertiggestellt* bildet den Produktstatus des gesamten Ergebnistyps auf einen boolschen Wert ab. Das Prädikat *abgeschlossen* bildet eine abschließende Bewertung von a_i auf einen boolschen Wert ab. Die Konjunktion verlangt, dass alle Aufgaben des Arbeitspakets erfolgreich abgeschlossen sind, um die Produkterzeugung ebenfalls abzuschließen. Gnatz [Gna05] (Seite 135 ff.) geht hierauf aus Sicht der Plangenerierung noch detaillierter ein.

Beziehungstyp: `Erfordert` – Der Beziehungstyp `Erfordert` dient dazu, Eingaben für Aktivitäten zu modellieren. In Kombination mit dem Beziehungstyp `Erzeugt` können wir damit auch *Produktflüsse* modellieren. Ein Produktfluss unterscheidet sich von einer erzeugenden Produktabhängigkeit insofern, als dass hier die Erstellung eines Produkts durch eine Aktivität erfolgt, die explizit andere Produkte als Eingabe erfordert. Ein Anwendungsfall findet sich beispielsweise im V-Modell XT beim Verfahren am Entscheidungspunkt. Dieses Verfahren zur Projektfortschrittsbestimmung ist im V-Modell nur semiformal und über viele Elemente verstreut

11 Für die Modellierung von Produkten hat das natürlich zur Folge, dass ein entsprechendes Attribut vorzusehen ist, das eine entsprechende Unterscheidung zwischen externen und nicht externen Produkten erlaubt.

definiert. Wir modellieren `Erfordert` wieder als $n+1$-stellige Assoziation mit $n \geq 1$ für $a \in \mathit{Aktivitaeten}, p_1, \ldots, p_n \in \mathit{Produkte}$:

$$\text{erfordert}\,(a, \{p_1, \ldots, p_n\}) \tag{5.20}$$

Über das Resultat von a sagen wir an dieser Stelle nichts aus. Durch Verknüpfung erzeugen wir dann einen Produktfluss für ein Produkt p_0:

$$\text{erzeugt}\,(a, p_0) \ \wedge \text{erfordert}\,(a, \{p_1, \ldots, p_n\}) \tag{5.21}$$

Als konkretes Beispiel beziehen wir uns auf den V-Modell Entscheidungspunkt *Projekt definiert*, zu dem mindestens ein *Projekthandbuch* (PHB), ein *QS-Handbuch* (QSHB) und ein *Projektplan* (PL) im Zustand „fertig gestellt" vorgelegt werden müssen. Aufgrund dieser Eingaben wird eine Projektfortschrittsentscheidung herbeigeführt, die als Ergebnis das Produkt *Projektfortschrittsentscheidung* (PFE) hat. Wir modellieren das wie folgt (vgl. Abbildung 5.13):

$$\text{erfordert}\,(\text{PFE herbeiführen}, \{\text{PHB}, \text{QSHB}, \text{PL}\})$$
$$\text{fertiggestellt}\,(\text{PHB})$$
$$\text{fertiggestellt}\,(\text{QSHB})$$
$$\text{fertiggestellt}\,(\text{PL})$$
$$\text{erzeugt}\,(\text{PFE herbeiführen}, \text{PFE})$$

Beziehungstyp: `Wird_bearbeitet` – Der Beziehungstyp `Wird_bearbeitet` dient uns dazu, planbare Operationen auf Produkten zu modellieren. Im V-Modell zum Beispiel dienen Aktivitäten ausschließlich der Fertigstellung von Produkten, d. h. Details zur Produktbearbeitung sind transparent. Aktivitätsorientierte Vorgehensmodelle wie MSF treffen andererseits sehr detaillierte Aussagen darüber, welche Aktivitäten ein Produkt im Laufe seines Lebenszyklus bearbeiten können. Beispielhaft sei hier ein Modell, zum Beispiel ein Entwurfsmodell genannt, dass erstellt, geprüft, aktualisiert und ggf. sogar bewiesen wird. Um dies auf der Metamodellebene bereits vorzusehen, definieren wir einen Beziehungstyp für Aktivitäten $a_1, \ldots, a_n$ und ein Produkt p wie folgt:

$$\text{wird\_bearbeitet}\,(p, \{a_1, \ldots, a_n\}) \tag{5.22}$$

An dieser Stelle ist es für uns dann auch unerheblich, zu welchem Aktivitätstyp $a_1, \ldots, a_n$ gehören.

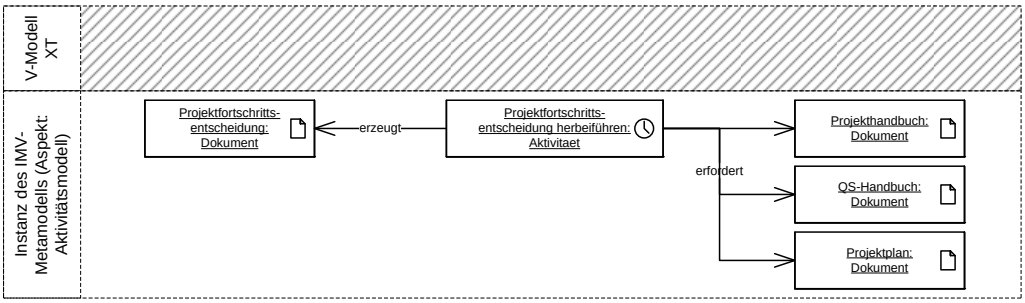


Abbildung 5.13.: Beispielhafter Produktfluss für einen V-Modell XT Entscheidungspunkt

Viele weitere Beziehungstypen zwischen Produkten und Aktivitäten sind denkbar, ähnlich wie diverse Einschränkungen, die für Spezialisierungen dieser beiden Konzepte denkbar sind. Um jedoch eine so hohe Komplexität im möglichen Abhängigkeitsgeflecht zu vermeiden, wie sie zum Beispiel bei EPF/UMA (Kapitel 3.2.3 und 3.4) zu finden ist, begnügen wir uns mit dieser kleinen Menge Beziehungstypen[12]. Die möglichen

12 Im Rahmen von Erweiterungen und Anpassungen besteht hier ggf. sowieso die Notwendigkeit der Anpassung oder Ergänzung.

Kombinationsmöglichkeiten zwischen Einzelelementen sind somit zwar eingeschränkt, können jedoch durch die Kombinierbarkeit der Einzelprädikate sehr schnell eine hohe Komplexität und Integration erreichen.

Zwischenstand. Alle Beziehungstypen, die wir bisher definiert haben dienen uns zur internen Konfiguration von Prozesskomponenten. Einige dieser Beziehungstypen sind jedoch im Rahmen der Komposition von Vorgehensmodellen aus Prozesskomponenten ebenfalls erforderlich. Hierfür fassen wir zunächst noch einmal die definierten Beziehungstypen zusammen (Tabelle 5.1) und widmen uns anschließend dem Themenkomplex der internen und externen Abhängigkeiten zur Bildung von Prozesskomponenten.

Name	Quelltyp	Zieltyp	Typ und Bemerkungen
`Erzeugt`	Produkt	Produkt	$1:n$ Beziehung
`Enthaelt`	Produkt	Produkt	$1:n$ über Spezialisierungen; Container und Hierarchien sind erforderlich, hier: Dokument enthält strukturierte Inhalte.
`Enthaelt`	Aktivität	Aktivität	$1:n$ über Spezialisierungen; Container und Hierarchien sind erforderlich, hier: Arbeitspakete enthalten Aufgaben (min. 2).
`Ist_Vorgaenger`	Aktivität	Aktivität	$1:1$ Beziehung; transitiv für Container und enthaltene Elemente
`Ist_Nachfolger`	Aktivität	Aktivität	$1:1$ Beziehung; transitiv für Container und enthaltene Elemente
`Ist_verantwortlich_fuer`	Rolle	Aktivität	$1:n$ Beziehung
`Ist_verantwortlich_fuer`	Rolle	Produkt	$1:n$ Beziehung
`Wirkt_mit`	Rolle	Aktivität	$1:n$ Beziehung
`Wirkt_mit`	Rolle	Produkt	$1:n$ Beziehung
`Wirkt_mit_an_teil`	Rolle	Produkt	$1:n$ Beziehung; für strukturierte Inhalte
`Erzeugt`	Aktivität	Produkt	$1:1$ Beziehung mit Ausnahme der externen Produkte
`Erfordert`	Aktivität	Produkt	$1:n$ Beziehung für die Modellierung von Produktflüssen
`Wird_bearbeitet`	Produkt	Aktivität	$1:n$ Beziehung

Tabelle 5.1.: Beziehungstypen des Vorgehensmetamodells (Zwischenstand und Übersicht)

Wie in Kapitel 4 definiert, stehen uns mit der *einfachen* und der *erweiterten Prozesselementabhängigkeit* ausreichend mächtige Konstrukte für die Modellierung der Abhängigkeitsstrukturen zur Verfügung. Die in Tabelle 5.1 zusammengestellten Beziehungstypen genügen, um sämtliche relevanten Beziehungen zwischen einzelnen Prozesselementen in einer Prozesskomponente auszudrücken. Gemäß unseren Überlegungen aus Kapitel 4.2.1 erhalten wir so einen *gerichteten Abhängigkeitsmultigraphen*. Wir vertiefen im Folgenden die Frage nach der Grenze zwischen der Konfiguration einer Prozesskomponente, ihren Schnittstellen sowie externen Abhängigkeiten und korrespondierenden

Konfigurationen von Vorgehensmodellen.

5.1.3. Prozesskomponenten und Vorgehensmodelle

Wir widmen uns nun der Zusammenstellung von Vorgehensmodellen auf Basis der gerade modellierten Prozesskomponenten. Gemäß unseren Vereinbarungen aus Kapitel 4 fixieren wir die Hierarchieebenen der Vorgehensmetamodellarchitektur und sehen *keine* Komposition von (elementaren) Prozesskomponenten zu Prozesskomponenten höherer Integration vor. In Abbildung 5.14 verfeinern wir das Modell aus Abbildung 5.4 und befassen uns nun mit einigen Fragestellungen der Konfiguration von Prozesskomponenten im Kontext eines Vorgehensmodells. Dabei betrachten wir in diesem Abschnitt zunächst nur die Beziehungstypen zwischen Prozesskomponenten und widmen uns dabei insbesondere den Schnittstellen, die externe Abhängigkeiten verursachen oder befriedigen.

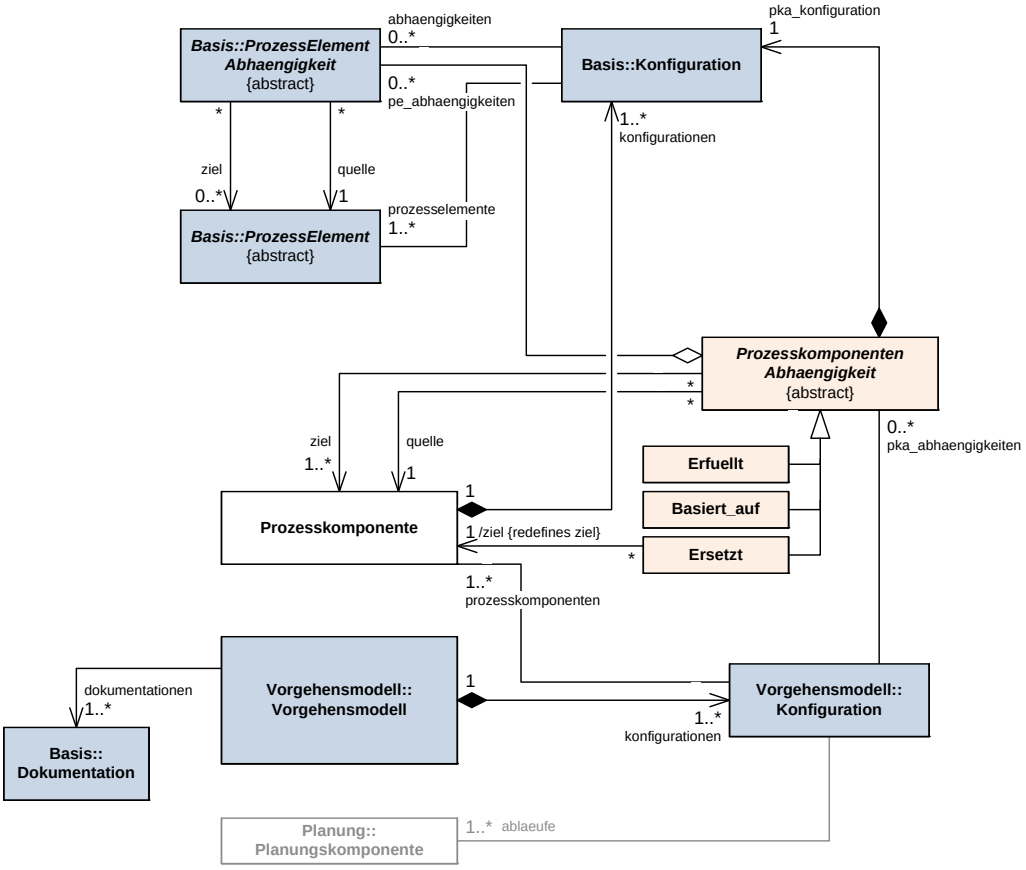


Abbildung 5.14.: Verfeinerung des Metamodells für Vorgehensmodelle mit Beziehungstypen

Beziehungstypen. Für Prozesskomponenten sehen wir analog zu Prozesselementen Beziehungstypen vor, die Relationen über der Menge der Prozesskomponenten definieren. In Kapitel 4.2.2 haben wir dafür entsprechende Relationen eingeführt. Die Beziehungstypen zwischen Prozesskomponenten werden dabei wieder auf Beziehungstypen zwischen Prozesselementen zurückgeführt. Die Beziehungstypen legen dabei Gültigkeiten und Semantik fest. In diesem Abschnitt beschreiben wir zunächst die Beziehungstypen für Prozesskomponenten näher und bauen im folgenden Abschnitt 6.2 mit der Betrachtung des Konfigurationsgesamtkonzepts darauf auf. Für Prozesskomponenten definieren wir drei Beziehungstypen:

- Beziehung: `Erfuellt`

- Beziehung: `Ersetzt`
- Beziehung: `Basiert_auf`

Mit diesen drei Beziehungstypen können wir alle relevanten Verknüpfungen zwischen Prozesskomponenten modellieren. Prozesskomponenten liegen uns als (weitgehend) abgeschlossene und vollständige Einheiten vor. Ihre interne Struktur ist definiert.

Schnittstellen von Prozesskomponenten. Durch die Konfiguration von Prozesselementen definieren Prozesskomponenten eine sehr detaillierte Schnittstelle, die alle Elemente und alle Beziehungen beschreibt. Dem Komponentenbegriff der Software Architektur (und hier insbesondere Siedersleben [Sie04]) folgend, können Schnittstellen für Komponenten *angeboten* oder *angefordert* sein. Bei der Verknüpfung von Prozesskomponenten müssen wir dies ebenfalls berücksichtigen. Wir nehmen folgende Fallunterscheidung vor:

1. Wir betrachten ausschließlich *konkrete* Prozesskomponenten. Dann existieren nur *angebotene*, jedoch keine *angeforderten* Schnittstellen und ein Vorgehensmodell kann durch einfache Konfiguration der Einzelkomponenten gebildet werden. Vorbedingungen existieren in diesem Fall nicht. Die Beziehungstypen `Basiert_auf` und `Ersetzt` sind anwendbar.
2. Wir betrachten auch *abstrakte* Prozesskomponenten, sodass es *angeforderte* Schnittstellen gibt. Als Vorbedingungen sind hier zunächst die angeforderten Schnittstellen eventuell genutzter Prozesskomponenten zu erfüllen. Dann sind insbesondere die Beziehungstypen `Erfuellt` und `Basiert_auf` zu berücksichtigen und zu realisieren.

Abbildung 5.14 zeigt den Ausschnitt des Metamodells, der die Beziehungstypen für Prozesskomponenten verfeinert. Für den allgemeinen Beziehungstyp (Klasse `ProzesskomponentenAbhaengigkeit`) gilt: Eine Konfiguration auf der Ebene des Vorgehensmodells referenziert alle für das jeweilige Vorgehensmodell relevanten Prozesskomponenten[13]. Zusätzlich enthält eine Konfiguration für Vorgehensmodelle eine Menge von Prozesskomponentenabhängigkeiten. Diese sind über der Menge der Prozesskomponenten definiert, die für ein Vorgehensmodell konfiguriert (also in der Konfiguration referenziert sind) werden. Eine Prozesskomponentenabhängigkeit referenziert alle beteiligten Prozesskomponenten und stellt zwischen diesen Beziehungen her. Da es sich hierbei um Beziehungen auf der Ebene von Prozesselementen handelt, werden hier neue Beziehungen zwischen den Prozesselementen der referenzierten Prozesskomponenten hergestellt.

Beispiel: Nehmen wir als Beispiel ein Auftraggeberprojekt (AG) des V-Modell XT: Das Produkt *Projekthandbuch* ist dort im Vorgehensbaustein Projektmanagement definiert. Das Produkt *Anforderungen (Lastenheft)* ist im Vorgehensbaustein Anforderungsfestlegung definiert. Es gibt eine Beziehung zwischen diesen beiden Vorgehensbausteinen, nämlich Anforderungsfestlegung basiert auf Projektmanagement. Weiterhin ist im *Projekthandbuch* ein Thema *Organisation und Vorgaben zum Anforderungsmanagement* enthalten, welches aber im Vorgehensbaustein Anforderungsfestlegung definiert ist, dem *Projekthandbuch* jedoch hinzugefüht wird.

Für unseren Kontext betrachten wir zwei Prozesskomponenten Projektmanagement und Anforderungsmanagement, die in einer *basiert_auf*-Beziehung stehen. In der betreffenden Konfiguration ist diese Beziehung hinterlegt und beide Prozesskomponenten (und nur diese beiden) können nun miteinander verknüpft werden, indem Beziehungen zwischen in ihnen enthaltenen Prozesselementen hergestellt werden. In der Konfiguration der Prozesskomponentenabhängigkeit zwischen Projektmanagement und Anforderungsmanagement ist also das Produkt (hier modelliert als Dokument) *Projekthandbuch* zu referenzieren und um den strukturierten Inhalt *Organisation und Vorgaben zum Anforderungsmanagement* zu erweitern.

Eine nach diesem Muster hergestellte Beziehung zwischen Elementen zweier oder mehrerer Prozesskomponenten ist auch nur für den Kontext dieser Prozesskomponentenabhängigkeit gültig. Wird die Prozesskomponentenabhängigkeit aus einer Konfiguration

13 An dieser Stelle wir auch die Unterscheidung zwischen Vorgehensmodell und Vorgehensmodellvariante hinfällig, da die Unterscheidung ausschließlich über unterschiedliche Konfigurationen getroffen wird. Wir sprechen auf dieser Ebene ausschließlich von Vorgehensmodellvarianten.

entfernt, werden auch sämtliche Verknüpfungen entfernt. Im Folgenden gehen wir detailliert auf die oben aufgeführten, speziellen Beziehungstypen ein:

Beziehung: `Basiert_auf` – Dieser Beziehungstyp dient im Wesentlichen der Nachnutzung und Wiederverwendung von Prozesskomponenten. Wir orientieren uns hier am gleichnamigen Konzept des V-Modell XT (Kapitel 3). Eine Prozesskomponente pk_0 kann auf n weiteren Prozesskomponenten basieren. Wir notieren sofort in vereinfachter Form:

$$\text{basiert\_auf}\left(pk_0, \{pk_1, \ldots, pk_n\}, \{pea_1, \ldots, pea_m\}\right) \tag{5.23}$$

Dies hat zur Folge, dass Inhalte aus pk_0 und Inhalte aus $pk_1, \ldots, pk_n$ sich referenzieren können. Zur Verfügung stehen hier sämtliche Prozesselemente und Prozesselementabhängigkeiten die wir in den vorangegangenen Abschnitten beschrieben haben. Dabei ist bei der Anwendung dieser Version darauf zu achten, dass gelten muss:

$$\forall pk \in \{pk_1, \ldots, pk_n\} : pk\,|_{G_\perp} = \emptyset \tag{5.24}$$

Keine durch `Basiert_auf` referenzierte Prozesskomponente darf *abstrakt* sein, d. h. keine referenzierte Prozesskomponente darf eine Schnittstelle anfordern. Wird durch eine Prozesskomponente eine Schnittstelle angefordert, ist anstelle einer Basierungs- eine *Erfüllungs*beziehung anzugeben.

Beziehung: `Erfuellt` – Eine Prozesskomponente darf explizit über *angeforderte Schnittstellen* verfügen, d. h. konkrete Funktionalität von einer spezialisierenden beziehungsweise ergänzenden Prozesskomponente anfordern. Für Prozesskomponenten $pk_0, \ldots, pk_n$ schreiben wir wieder:

$$\text{erfuellt}\left(pk_0, \{pk_1, \ldots, pk_n\}, \{pea_1, \ldots, pea_m\}\right) \tag{5.25}$$

wenn die Prozesskomponente pk_0 die angeforderten Schnittstellen der Prozesskomponenten $pk_1, \ldots, pk_n$ befriedigt. In den Prozesskomponenten $pk_1, \ldots, pk_n$ gibt es also Beziehungen, deren Ende nicht gültig ist (wir schreiben dafür einfach $\perp$, also zum Beispiel *enthaelt* $(a, \perp)$). Betrachten wir alle abstrakten Prozesskomponenten pk_i in einer solchen Verknüpfung, dann gilt:

$$G_{VgM,\perp} = \bigcup_{1 \le i \le n} pk_i\,|_{G_\perp} = (Pe_\perp, (PEA \cup EPEA)_\perp)$$

$G_{VgM,\perp}$ ist der Abhängigkeitsgraph aller in einer Konfiguration referenzierten Prozesskomponenten $pk_1, \ldots, pk_n$. Gemäß unserem Beispiel aus Kapitel 4.2.2 ist dieser Graph nicht (zwangsweise) zusammenhängend. Der Zusammenhang wird durch den Beziehungstyp `Erfuellt` in der Prozesskomponente pk_0 hergestellt. Anders als bei der einfachen Verknüpfung von Graphen durch neue Kanten zwischen den Teilgraphen wird die Verknüpfung hier durch Substitution der „ungültigen" Knoten $\perp$ der offenen Kanten durch entsprechende Knoten aus pk_0 hergestellt. Die offenen Kanten sind bestimmt durch $\rho \subseteq (PEA \cup EPEA)_\perp$, sodass für alle ρ gilt: $\rho = (x, \perp) \vee \rho = (x, Y) \wedge \perp \in Y$. Wir berücksichtigen also sowohl einfache, wie auch erweiterte Abhängigkeiten, wobei die erweiterten Abhängigkeiten wieder auf einfache Paare reduzierbar sind. Seien ferner die für die Substitution verfügbaren Prozesselemente aus pk_0 gegeben durch Pe_{pk_0}, dann gilt für eine konfigurierte `Erfuellt`-Beziehung:

$$\forall \rho . \exists \alpha \in Pe_{pk_0} : \text{subst}\,(\perp, \alpha) \tag{5.26}$$

Für alle existierenden offenen Kanten ρ muss bei der Konfiguration ein Knoten α zur Substitution angegeben werden. Die Operation *subst* übernimmt diese Zuordnung. Es entstehen hierbei *keine* neuen Kanten im Graph; vielmehr werden existierende Kanten mit gültigen Zielen vervollständigt. Die Beziehung `Erfuellt` zwischen n-Prozesskomponenten muss also sicher stellen, dass keine offenen Kanten mehr im resultierenden Gesamtabhängigkeitsgraph vorhanden sind.

Aufgrund der Typisierung der Prozesselementabhängigkeiten, können hier sehr feingranulare Anforderungen beschrieben werden. Beispielhaft verweisen wir auf den Beziehungstyp `Erfordert` zwischen Aktivitäten und Produkten. Eine Aktivität kann beispielsweise bestimmte Eingangsprodukte erfordern, die konkrete Belegung jedoch verlagern. Eine beispielhafte Formulierung für eine Aktivität aus einer Prozesskomponente, die ein Eingangsprodukt aus einer anderen Prozesskomponente benötigt, hat folgendes Aussehen:

$$\text{erfordert}\,(akt_0, \{p_1, \ldots, p_i, \bot_{akt_0}, p_j, \ldots, p_n\})$$
$$\ldots$$
$$\text{subst}\,(\bot_{akt_0}, p_\alpha)$$

Die Anwendung von *subst* $(\bot_{akt_0}, p_\alpha)$ nimmt hier eine konkrete Belegung vor. Eine äquivalente Schreibweise stellt die folgende Form dar:

$$\text{erfordert}\,(akt_0, \{p_1, \ldots, p_n\})$$
$$\text{erfordert}\,(akt_0, \{\bot\})$$
$$\ldots$$
$$\text{subst}\,(\bot_{akt_0}, p_\alpha)$$

Diese Form orientiert sich schon näher an der konkreten Umsetzung der angeforderten Schnittstelle für eine Prozesskomponente. Diese setzt sich aus einer parziellen Konfiguration zusammen. Eine Prozesskomponente selbst soll intern abgeschlossen sein. Sind Abhängigkeiten mit anderen Prozesskomponenten derart zu modellieren, dass Inhalte durch externe Prozesskomponenten beigesteuert werden müssen, so müssen diese Anforderungen separat formuliert werden. Die Anforderung wiederum entspricht der Spezialisierung eines verfügbaren Angebots (eine *bekannte* Schnittstelle wird angefordert). Dies wird gerade durch den Beziehungstyp *basiert_auf* realisiert. Eine angeforderte Schnittstelle einer Prozesskomponente pk_0 können wir daher wie folgt formulieren:

$$\text{basiert\_auf}\left(pk_0, \{\bot\}, \{pea_1, \ldots, pea_m\}\right)$$

Im Kontext dieser Prozesskomponentenabhängigkeit können wir nun Prozesselementabhängigkeiten der Form ρ instanziieren. Stehen nun zwei Prozesskomponenten in einer `Erfuellt`-Beziehung, so werden eben diese Prozesselementabhängigkeiten aufgelöst.

Ein konkretes Anwendungsfeld finden wir beispielsweise in Organisationen, in denen generische Verfahren beschrieben und durch Abteilungen konkret umzusetzen sind (zum Beispiel Berichte, Meldungen, Aufträge etc.). Wir können auf diese Art auch abstrakte Prozessschnittstellen definieren, die im Rahmen einer organisationsspezifischen Ausgestaltung zu konkretisieren sind.

Beziehung: `Ersetzt` – Der letzte zu betrachtende Beziehungstyp ist der Typ `Ersetzt`. Dieser Beziehungstyp dient dazu, einzelne Prozesskomponenten gegen Prozesskomponenten mit kompatiblen Schnittstellen (in der Regel Weiterentwicklungen, vgl. Abbildung 4.17, Seite 105) zu ersetzen. Der Beziehungstyp `Ersetzt` ist der einzige der hier besprochenen, der ausschließlich 2-stellig ist. Eine kumulative Ersetzung, d. h. eine Prozesskomponente ersetzt n andere, lassen wir *nicht* zu, um die Nachverfolgbarkeit zum Beispiel von Versionshistorien weiterhin zu gewährleisten. Wir definieren also:

$$\text{ersetzt}\,(pk_1, pk_0, \bot) \tag{5.27}$$

für zwei Prozesskomponenten pk_0 und pk_1 genau dann, wenn sichergestellt ist, dass:

$$pk_1\,|_G \supseteq pk_0\,|_G \tag{5.28}$$

Dies schließt natürlich nicht aus, dass die Schnittstelle beziehungsweise das allgemeine Leistungsangebot von pk_1 umfassender sein kann, als das von pk_0. Wir ziehen uns hier auf komponentenorientierte Techniken zurück, die für Software durch so genannte *Schnittstellenvererbung* (zum Beispiel [HR02]) realisiert werden.

Analog zu Tabelle 5.1 fassen wir in Tabelle 5.2 noch einmal die vorgestellten Beziehungstypen für Prozesskomponenten zusammen.

Name	Kardinalität	Bemerkungen
`Basiert_auf`	$1:n$	Anwendbar auf alle Prozesskomponenten, die nicht abstrakt sind. Die Nutzung der aggregierenden Prozesskomponente hat automatisch die Mitnutzung der aggregierten Prozesskomponenten zur Folge.
`Erfuellt`	$1:n$	Nachbedingung: Alle referenzierten Prozesskomponenten sind befriedigt. Die erfüllende Prozesskomponente ist nicht abstrakt.
`Ersetzt`	$1:1$	Kumulative Ersetzung ist verboten. Transitives Ersetzen ist jedoch möglich.

Tabelle 5.2.: Beziehungstypen des Vorgehensmetamodells für Prozesskomponenten

Beispiel: Instanziierung des Metamodells für Prozesskomponenten. Nachdem wir die einzelnen Beziehungstypen für Prozesskomponenten eingeführt haben, wollen wir noch ein kurzes Beispiel geben. Betrachten wir das System von Prozesskomponenten $pk_0, \ldots, pk_5$ in Abbildung 5.15 (oben).

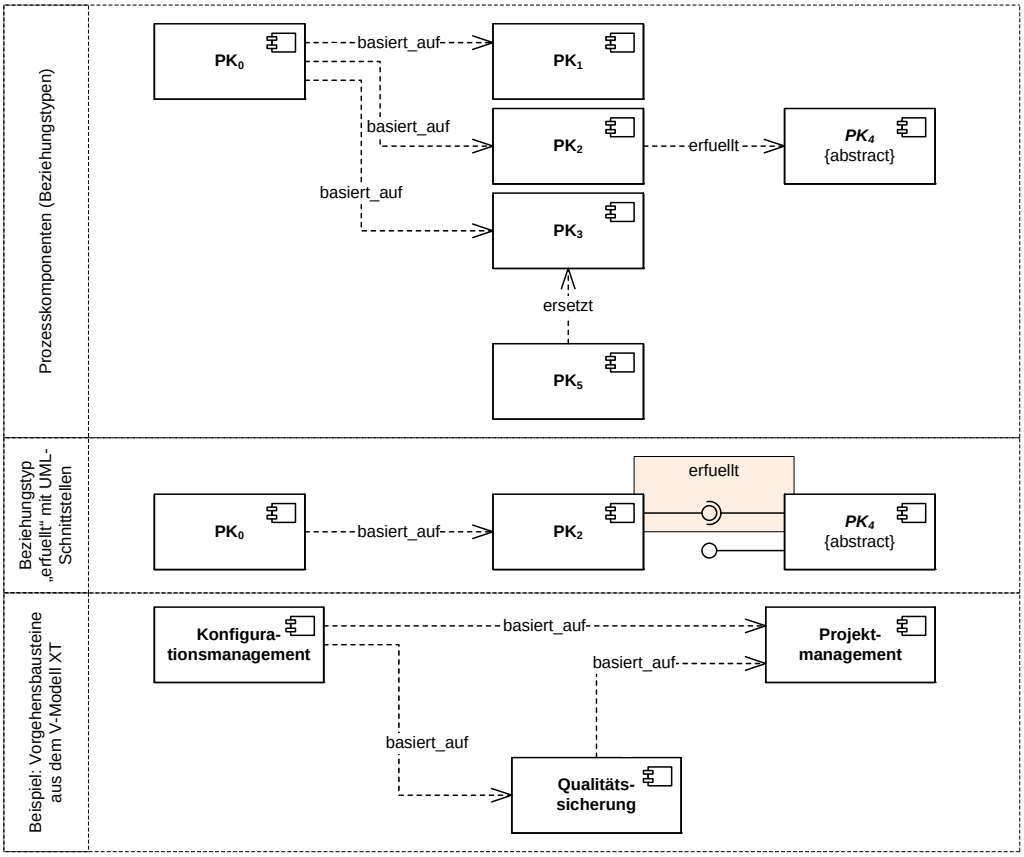


Abbildung 5.15.: Beispiel der Anwendung von Prozesskomponentenabhängigkeiten

pk_4 ist eine abstrakte Prozesskomponente, die eine angeforderte Schnittstelle definiert. Diese Prozesskomponente kann beispielsweise einen allgemeinen Qualitätssicherungsprozess abteilungsübergreifend definieren, der durch Abteilungen konkretisiert werden muss, zum Beispiel: allgemeine QS vs. QS für Software oder QS für Hardware

oder QS für integrierte Soft- und Hardware. Diese Spezialisierung nimmt in unserem Beispiel pk_2 vor. Im mittleren Teil der Abbildung haben wir dies in Form der UML 2-Notation für Komponenten und deren Schnittstellen noch einmal gesondert hervorgehoben. Es existiert hier also eine Beziehung *erfuellt* $(pk_2, pk_4, \{pea_1, \ldots, pea_m\})$ zwischen diesen beiden Prozesskomponenten. Sie wird durch ein Paar Schnittstellen realisiert, die wie oben diskutiert die Komplettierung ungültiger Kanten vornehmen. In der angeforderten Schnittstelle von pk_4 sind dabei die zu erfüllenden Abhängigkeiten ρ enthalten, während in der erfüllenden Schnittstelle diejenigen Prozesselemente enthalten sind, die für die Substitution der $\perp$-Knoten verwendet werden. Die Prozesskomponente pk_0 basiert auf drei weiteren Prozesskomponenten, sodass hier eine Beziehung: *basiert_auf* $(pk_0, \{pk_1, pk_2, pk_3\}, \{pea_1, \ldots, pea_m\})$ existiert. Für pk_3 ist darüber hinaus noch zu beachten, dass mit pk_5 eine äquivalente Ersetzungsmöglichkeit existiert, also eine Beziehung *ersetzt* $(pk_5, pk_3, \perp)$. Beispielhaft könnten wir für pk_3 annehmen, dass es die Prozesselemente enthält, die OO-Programmierung mit Java beschreiben, während pk_5 OO-Porgrammierung mit .NET beschreibt.

Im unteren Teil der Abbildung 5.15 finden wir eine beispielhafte Anwendung des V-Modell XT auf das Konzept der Prozesskomponente. Basiskomponente hier ist der Vorgehensbaustein *Projektmanagement* (PM), auf dem sowohl *Qualitätssicherung* (QS) als auch *Konfigurationsmanagement* (KM) aufsetzen. Konfigurationsmanagement basiert weiterhin auf Qualitätssicherung. In unserer Modellierung würde sich das wie folgt darstellen:

$$\text{Prozesskomponenten} = \{\text{PM}, \text{QS}, \text{KM}\}$$
$$\text{basiert\_auf}\,(\text{QS}, \{\text{PM}\}, \{\ldots\})$$
$$\text{basiert\_auf}\,(\text{KM}, \{\text{QS}, \text{PM}\}, \{\ldots\})$$

Bereits auf Grundlage von Abbildung 5.15 (oben) können wir schon zwei verschiedene Konfigurationen angeben, die wir im Kontext eines Vorgehensmodells bilden können. Die verschiedenen Beziehungstypen zwischen den einzelnen Prozesskomponenten sind dabei eine grobgranulare Sicht. In Abhängigkeit vom Beziehungstyp sind die Konfigurationen frei oder nur unter bestimmten Vor- und Nachbedingungen möglich (vgl. Tabelle 5.2). Wenden wir zum Beispiel die Option *ersetzt* $(pk_5, pk_3, \perp)$ nicht an, enthält unsere Vorgehensmodellvariante die Prozesskomponenten pk_0, pk_1, pk_2, pk_3 und pk_4 sowie alle enthaltenen Prozesselemente. Wenden wir hingegen *ersetzt* $(pk_5, pk_3, \perp)$ an, entfallen die Prozesselemente aus pk_3 vollständig und werden durch diejenigen aus pk_5 ersetzt, wobei sicherzustellen ist, dass die Beziehung *basiert_auf* $(pk_0, pk_3, \{pea_1, \ldots, pea_m\})$ geprüft werden muss, da durch:

$$\text{basiert\_auf}\,(pk_0, \text{ersetzt}\,(pk_5, pk_3), \{pea_1, \ldots, pea_m\})\ \text{gilt:}$$
$$\text{basiert\_auf}\,(pk_0, pk_5, \{pea_1, \ldots, pea_m\})$$

Im Sinne der Produktlinienentwicklung ist pk_3 ein Variationspunkt Die (inhaltliche) Konsistenz dieser Ersetzung muss im Nachhinein sichergestellt werden. Wir realisieren auf diese Weise die *konfigurative* Variabilität (vgl. Abbildung 4.17) und können somit Varianten über den Mengen der Prozesskomponenten erzeugen mit: $v_1 = \{pk_0, pk_1, pk_2, pk_3, pk_5\}$ oder $v_2 = \{pk_0, pk_1, pk_2, pk_5, pk_4\}$.

5.1.4. Variabilität in Vorgehensmodellen

Nachdem wir bereits die Beziehungstypen für Prozesselemente sowie Prozesskomponenten vorgestellt und schon weiter gehende Konfigurationen motiviert haben, greifen wir nun wieder die Beziehungstypen auf, die Operationen für die Variabilität realisieren. Diese gewinnen jetzt an Bedeutung, da wir auf der Ebene konfigurierter Prozesskomponenten nun Beziehungen zwischen den enthaltenen Prozesselementen herstellen können. Die Anwendung von Variabilitätsoperationen ist hier vorzunehmen – hier werden im Rahmen der Anwendung und Ausgestaltung eines Vorgehensmodelles auch Prozessingenieure Variabilitäten einfügen. Unterscheiden müssen wir jedoch an dieser

Stelle, welcher Typ *Anpassungsoperation*[14] also Variabilitätsoperation und welcher Typ *nur* als Erweiterungsoperation zu modellieren ist. Als Anpassungsoperationen verstehen wir alle Operationen, die in irgendeiner Form der strukturellen oder inhaltlichen Anpassung eines Vorgehensmodells dienen. Im Speziellen definieren wir dann im konstruktiven Prozess die Erweiterungsoperationen als solche, die rein additiv wirken. Das einfache Zusammenstellen von Prozesskomponenten wie in Abbilung 5.15 gezeigt, ist zum Beispiel eine Erweiterungsoperation.

Definition 5.1 (Erweiterungsoperation)**:**
Eine Erweiterungsoperation ist ein Standardverfahren zur Erstellung eines Vorgehensmodells, in dem *keine* strukturellen Änderungen (im Sinne von Substitution, Überschattung etc.) vorgenommen werden. Erweiterungsoperationen erzeugen ausschließlich neue Kanten im Abhängigkeitsgraphen.

Definition 5.2 (Variabilitätsoperation (auch Änderungsoperation))**:**
Eine Variabilitätsoperation ist ein Verfahren zur Manipulation der Knotenmengen in Abhängigkeitsgraphen im Sinne von Substitution, Überschattung etc.

Variabilitätsoperationen, als zweiter Typ von Anpassungsoperationen, können auf die Strukturen des Abhängigkeitsgraphen auch durch die Manipulation von Knoten wirken (vgl. Kapitel 4.3.3). Hierzu ist zunächst die Frage nach sinnvollen Variabilitätsoperationen zu klären. Abbildung 5.16 greift noch einmal die allgemeine Ontologie auf (vgl. Abbildung 5.7), auf der die Modellierung unserer Prozesskomponenten basiert.

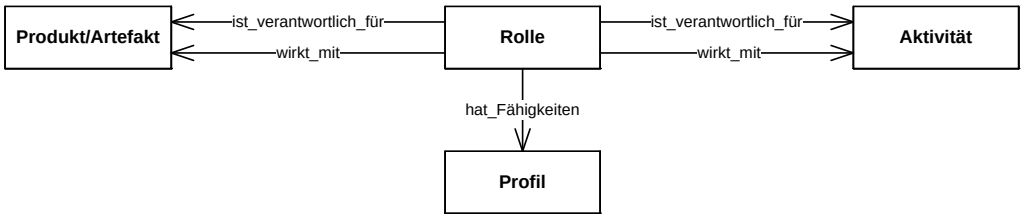


Abbildung 5.16.: Ontologie für Vorgehensmodelle im Kontext der Variabilität

Variabilitätsoperationen definieren wir nur im Kontext der Kombination von Prozesskomponenten[15]. Daher wirken Variabilitätsoperationen zunächst nur auf Prozesselementtypen und Abhängigkeitstypen, die im Kontext von Prozesskomponenten definiert sind. Wir zeigen hier nur eine kleine Auswahl exemplarischer Variabilitätsoperationen für Produkte und Aktivitäten, die wir der aktuellen V-Modell XT-Entwicklung entnommen haben. Ergänzend zu den Aussagen aus Kapitel 4.3.3 betrachten wir hier jedoch auch die Wirkung auf Eigenschaften[16] der betroffenen Prozesselemente. Die dabei für uns relevanten Eigenschaften sind diejenigen, die im Basismodell (Abbildung 5.2) für Prozesselemente definiert sind. Wir führen die Diskussionen im Folgenden anhand kleiner Anwendungsfälle und Szenarios.

Variation: Produktersetzung. Die Ersetzung von Produkten ist ein typischer Anwendungsfall, der im Rahmen einer organisationsspezifischen Anpassung auftritt. Ausgangspunkt ist hierbei ein standardmäßig definiertes (generisches) Produkt, das durch ein bereits etabliertes Produkt der anpassenden Organisation ersetzt werden soll. Ein typischer Vertreter für diese Art Szenario sind Produkte der Systementwicklung (vgl. [HKST07]). Abbildung 5.17 gibt die Anwendung der Variabilitätsoperation *ersetzt* für den Kontext Produkte beispielhaft an.

14 Diese Fragestellung und die daraus folgende Terminologie ist bei den Arbeiten am Variabilitätskonzept des V-Modells aufgekommen (vgl. Anhang A.3). Wir führen diese Begrifflichkeit hier ebenfalls weiter und verweisen für Detailinformationen auf den Anhang.

15 Sofern weitere Konzepte hinzukommen, zum Beispiel Ablaufkomponenten, können sich hier auch wieder neue Operationstypen ergeben

16 Dieses Verfahren wird auch bei der Verhaltensmodellierung der Variabilitätsoperationen bei der Überarbeitung des V-Modell XT angewendet – vgl. Anhang A.3

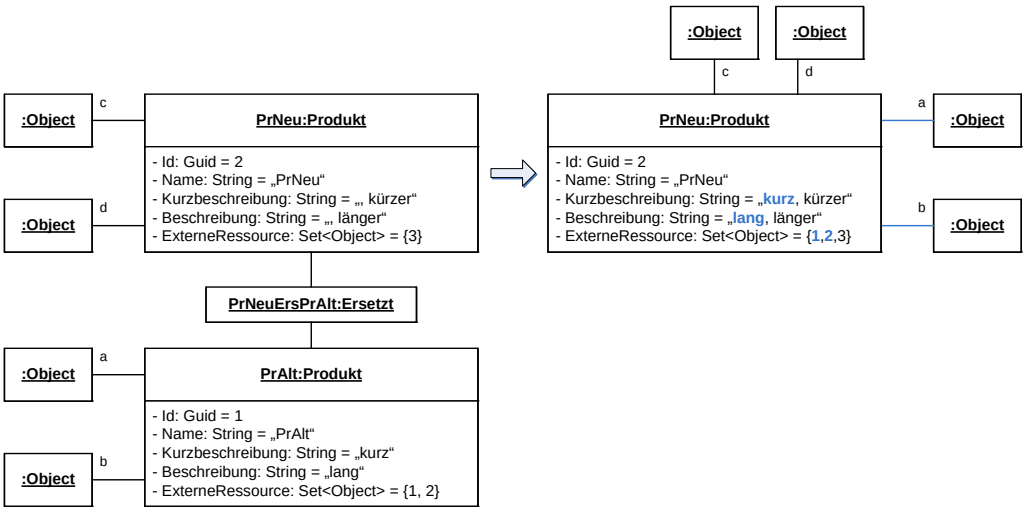


Abbildung 5.17.: Variationsoperation: Produktersetzung

Im Beispiel soll das Produkt *PrAlt* durch das Produkt *PrNeu* ersetzt werden. Wir haben im Kapitel 4.3.3 die Semantik der Operation so definiert, dass alle Referenzen auf das zu ersetzende Produkt erhalten bleiben, jedoch auf das neue Produkt umgebogen werden. Referenzen auf andere Prozesselemente, also Kanten im Abhängigkeitsgraph, haben wir der Einfachheit halber allgemein als *Object* typisiert. Da sie durch die Operationen nur insofern beeinflusst werden, als dass ihre Anfangs- beziehungsweise Endereferenzen aktualisiert werden, ist das Verhalten für alle Assoziationen zwischen Prozesselementen zunächst gleich. Abbildung 5.17 zeigt weiterhin die Verfahrensweise der Operation auf der inhaltlichen Ebene (Hervorhebungen in der Grafik). So werden hier alle Werte der „Nutzfelder"[17] miteinander verknüpft. Für Zeichenketten wird dabei eine Konkatenation $zkNeu := zkAlt \circ zkNeu$ durchgeführt; Mengen werden vereinigt: $mengeNeu := mengeAlt \cup mengeNeu$. Als Ergebnis verbleibt im Beispiel aus Abbildung 5.17 lediglich *PrNeu* im resultierenden Abhängigkeitsgraph. Jedoch sind die spezifizierten Attribute und die Referenzen entsprechend angepasst, sodass *PrNeu* nun gleichzeitig die Position von *PrAlt* einnehmen kann.

Neben der „echten" Ersetzung von Produkten, wie wir sie beispielsweise im Rahmen von [Kuh07] weitreichend modelliert haben, kann diese Operation auch für die *inhaltserhaltende Zusammenlegung* von Produkten verwendet werden. Wir kommen auf das eingangs aufgeführte Szenario zurück und beziehen uns auf [HKST07]. Im dortigen Beitrag skizzieren wir einen kompakten Systementwicklungsprozess, in dem verschiedene Produkte des V-Modells zusammengefasst werden müssen, um den kurzen Iterationen Rechnung zu tragen. Die V-Modell-Produkte *Systemarchitektur* und *SW-Architektur* werden dort als zusammenlegbar identifiziert[18]. Jedoch wird die *SW-Architektur* als maßgebend betrachtet. Die Anwendung der Operation:

$$\text{ersetzt}\,(\text{Systemarchitektur}, \text{SW-Architektur})$$

führt somit eine Zusammenfassung der Inhalte der beiden Produkte durch und aktualisiert die Graphenstruktur so, dass nun die *SW-Architektur* an die Stelle der *Systemarchitektur* tritt, gleichzeitig jedoch auch alle ihre Inhalte bereitstellt.

Variation: Produktspezialisierung. Die Spezialisierung eines Produktes ist der Ersetzung sehr ähnlich. Betrachten wir wieder die beiden Produkte *PrAlt* und *PrNeu*, so wird *PrNeu* analog zu gerade mit Informationen aus *PrAlt* angereichert. Der Unterschied zwischen der Ersetzung und der Spezialisierung ist in diesem Kontext, dass im Anschluss

17 Attribute, die nicht der Identifikation von Prozesselementen (in diesem Fall Produkten) dienen
18 Im Artikel jedoch nur vertreten durch die Zusammenlegung der Entscheidungspunkte; das Produkt Systemarchitektur entfällt dort einfach.

an die Operation *erweitert* sowohl *PrAlt* als auch *PrNeu* im resultierenden Abhängigkeitsgraphen enthalten sind.

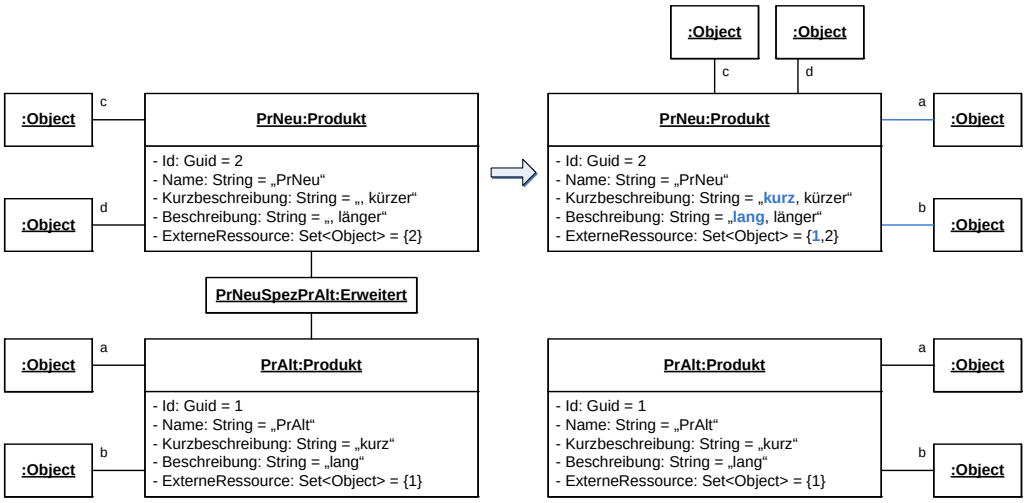


Abbildung 5.18.: Variationsoperation: Produktspezialisierung

Abbildung 5.18 zeigt analog zur Abbildung 5.17 das Ergebnis der Anwendung der Operation auf der Ebene konkreter Instanzen. Diese Operation erzeugt ein System polymorpher Objekte, die ausgehend von *PrAlt* spezialisiert werden. Das entspricht im Wesentlichen dem Konzept der Vererbung der objektorientierten Programmierung angewendet auf flache Datenstrukturen. Dementsprechend kann *PrNeu* in diesem Fall anstelle von *PrAlt* verwendet werden, jedoch auch daneben stehend.

Anwendungsfälle für diese Operation finden sich in Bereichen, in denen Teile der Arbeit sich stark ähneln beziehungsweise gleich sind und in bestimmte Felder (additiv) abweichen. Das Berichtswesen und die Qualitätssicherung sind solche Arbeitsfelder. Für das Berichtswesen lassen sich mit diesem Verfahren beispielsweise Berichtshierarchien abbilden, in denen es einen schlanken *Basisbericht* gibt, auf dem weiter gehenden, mit zusätzlichen Informationen angereicherte Berichte aufsetzen. Eine ähnliche Hierarchie können wir auch für die Qualitätssicherung konstruieren. Wir beziehen uns hier auf das V-Modell XT und seine Produkte in der Produktgruppe *Prüfung*. Wir betrachten die Produkttypen *Prüfspezifikation Dokument*, *Prüfspezifikation Systemelement* und *Prüfspezifikation Lieferung*. Kleinstes Element (im Sinne einfachste Struktur) ist die *Prüfspezifikation Dokument*, die lediglich die Themen *Prüfobjekt* und *Prüfkriterien* enthält. Die *Prüfspezifikation Systemelement* fügt dem zum Beispiel das Thema *Prüfstrategie* hinzu und ersetzt das Thema *Prüfkriterien* durch *Prüffälle*. Die *Prüfspezifikation Lieferung* führt keine Themenersetzung durch, ergänzt aber weitere Themen, zum Beispiel *Schutzvorkehrungen*. Analoges gilt dann auch für die Produkttypen der Prüfprotokolle.

Variation: Produkterweiterung. Während die letzten beiden Operationen Produkte ausgeblendet und ersetzt, beziehungsweise Hierarchien gebildet haben, ist es im Kontext einer organisationsspezifischen Anpassung von Fall zu Fall auch ausreichend, wenn existierende Produkte um fehlende Strukturen ergänzt werden können. Ein Szenario ist beispielsweise: Ein Standardvorgehen beschreibt ein generisches Produkt *A*, zu dem die Themen (strukturierte Inhalte) *1*, *2*, *3* und *4* gehören. Verantwortlich für das Produkt ist die Rolle α. Eine anpassende Organisation stellt fest, dass *A* bereits weitgehend den Anforderungen der Organisation entspricht, jedoch zusätzlich noch die Themen *5* und *6* sowie die mitwirkende Rolle β ergänzt werden müssen. Die Operation *ergaenzt* führt genau dies kompakt und zentral durch.

Abbildung 5.19 zeigt die Wirkungen der Operation am Beispiel eines Instanzendiagramms. *PrNeu* steuert zwei Abhängigkeiten zu anderen Prozesselementen sowie eine inhaltliche Referenz bei. Im resultierenden Abhängigkeitsgraph sind die Links auf

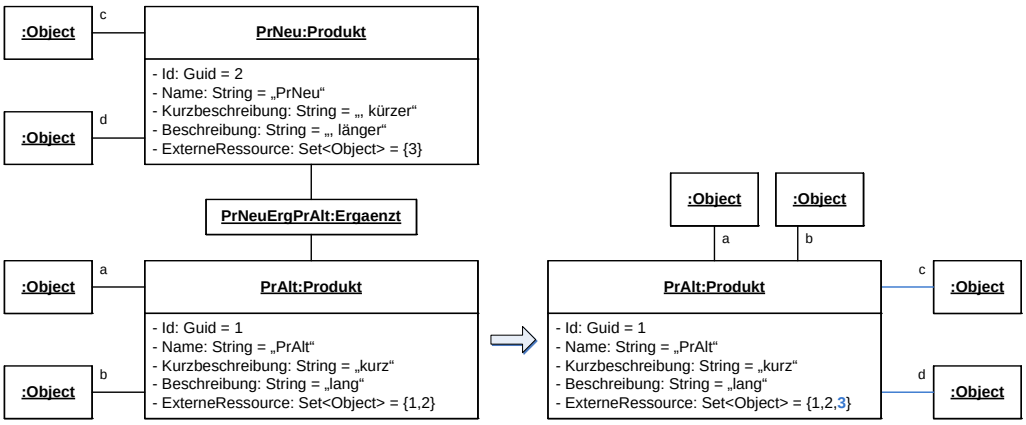


Abbildung 5.19.: Variationsoperation: Produkterweiterung

die abhängigen Elemente durch *PrAlt* übernommen worden. Eventuell neue Inhalte sind ebenfalls in *PrAlt* untergebracht (entweder via Konkatenation oder Mengenvereinigung). *PrNeu* ist nicht im resultierenden Abhängigkeitsgraphen enthalten.

Variation: Aktivitätsersetzung. Die Ersetzung von Aktivitäten sehen wir ebenfalls vor. Dies ist aus den Erfahrungen in der Anpassung des V-Modells eines der wesentlichen Felder, die bearbeitet werden müssen, sofern man die abstrakten Beschreibungen des V-Modells gegen konkretere tauschen möchte, die zum Beispiel durch integrierte Methoden beigesteuert werden [Kuh07].

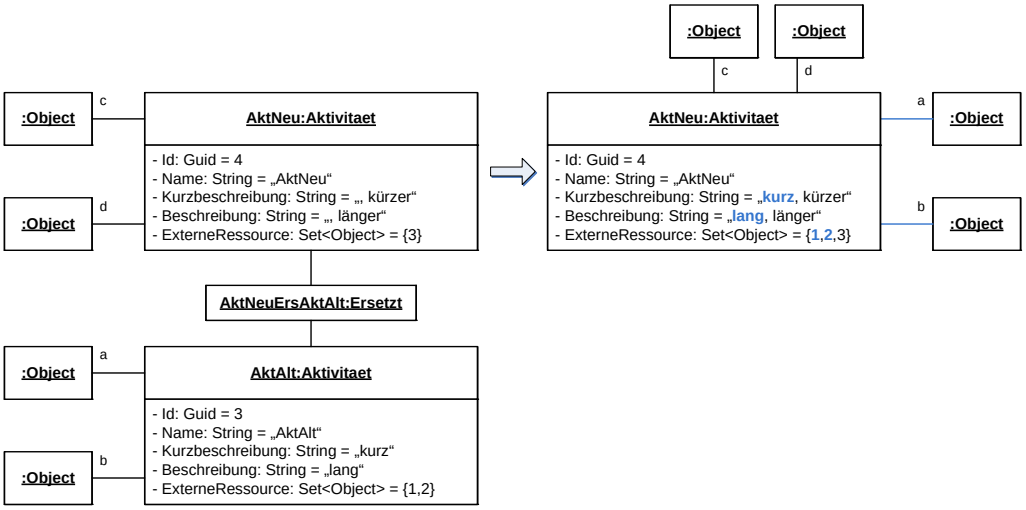


Abbildung 5.20.: Variationsoperation: Aktivitätsersetzung

Abbildung 5.20 zeigt eine beispielhafte Ersetzung einer Aktivität *AktAlt* durch einen Aktivität *AktNeu*. Semantisch ist diese Operation gleich der Ersetzung eines Produkts.

Variation: Aktivitätsausgestaltung. Bislang haben wir bei der Variation von Elementen (intuitiv) auf die Typverträglichkeit geachtet und zum Beispiel nur Produkte mit Produkten und Aktivitäten nur mit Aktivitäten kombiniert. In der Praxis wird dies auch den Regelfall darstellen, da ein großer Teil der Anforderungen durch die Zusammenlegung und/oder Anpassung von ausgewählten Einzelelementen abgedeckt ist. Nichtsdestotrotz ist der hier beschriebene und verwendete Ansatz mächtiger und eignet sich zum Beispiel auch dafür, im Rahmen einer organisationsspezifischen Anpassung weiter

gehende Verfeinerungen vorzunehmen. Hilfreich sind dabei die den jeweiligen Submodellen zugrunde liegenden Basiskonzepte.

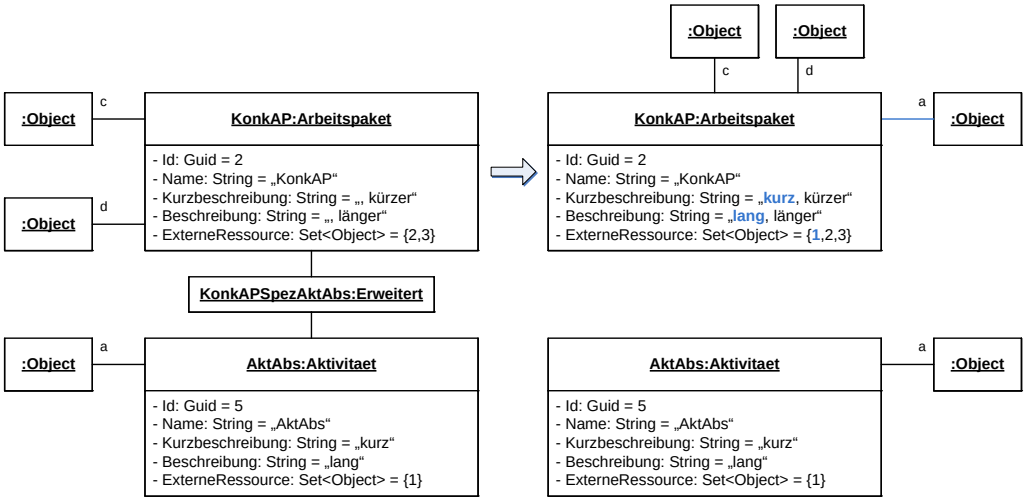


Abbildung 5.21.: Variationsoperation: Aktivitätsausgestaltung

Abbildung 5.21 zeigt ein solches Szenario, in dem wir mithilfe einer Variabilitätsoperation eine verfeinernde und erweiternde Ausgestaltung sowohl auf der inhaltlichen als auch auf der strukturellen Ebene ermöglichen. Unser Anwendungsfall stellt sich wie folgt dar: In einem anzupassenden Standardvorgehen ist die (generische) Aktivität *AktAbs* beschrieben, die in Abhängigkeit zu einem Produkt (hier repräsentiert durch einen anonymen Objektlink) steht. Im Rahmen der Anpassung soll die Produkterstellung jedoch noch einem organisationsinternen Verfahren erfolgen. Wir wenden die Operation *erweitert* auf der Aktivität mit einem *Arbeitspaket* an, welches das vorgegebene Verfahren durch eine Menge von *Aufgaben* abbildet. Gemäß der Modellierung in den Kapiteln 5.1.2 und 5.1.2 können wir so vorgehen, da wir Abhängigkeitsstrukturen auf einer Containerstruktur (Arbeitspaket) anwenden. Der Container trifft somit die Festlegungen für die enthaltenen Elemente. Für das Produkt, das nun auf der Grundlage eines Arbeitspakets erstellt wird, ist diese Änderung jedoch transparent.

Über dieses Verfahren können wir nun sinnvolle, methodische Ergänzungen transparent einsteuern. Im Rahmen eines Plattform- und Organisationsmodells haben wir damit ein Werkzeug an der Hand, das es gestattet, abstrakte/generische Vorgaben zu machen, die durch die Anpassung einfach und transparent ausgestaltet werden können. Dies definiert darüber hinaus auch eine Schnittstelle zwischen verschiedenen Organisationen, die auf demselben Plattformmodell eine Anpassung vorgenommen haben. Der Variationspunkt ist eindeutig bestimmt womit die beiden spezialisierenden Verfahren aufeinander abbildbar sind.

5.2. Modellierungssicht: Lebenszyklusmodell

In diesem Abschnitt setzen wir auf dem erarbeiteten Architekturmodell auf und definieren ein *Lebenszyklusmodell* über der Architektur. Wir orientieren uns dabei an den Entwicklungs- und Anpassungsprozessen, wie wir sie in Kapitel 2.2 analysiert haben. Die grundlegende Gestaltung des Modells führen ebenfalls auf den Solution Delivery Process (SDP) und die (Software-)Produktlinienentwicklung (PLE) aus Kapitel 2.3 zurück. Abbildung 5.22 zeigt das *Lebenszyklusmodell für Vorgehensmodelle und Vorgehensmodelllinien*. Es berücksichtigt die

- Entwicklung,
- Pflege und Bereitstellung sowie die
- Anwendung

von Vorgehensmodellen nach dem hierarchisch strukturierten Muster aus Kapitel 4. Dabei steht hier nicht mehr nur ein einzelnes Vorgehensmodell im Fokus, sondern eine Menge von Vorgehensmodellen. Dem PLE-Referenzmodell folgend, betrachten wir die vier Bereiche:

- Linienentwicklung,
- Management,
- Exemplarentwicklung und
- Anwendung.

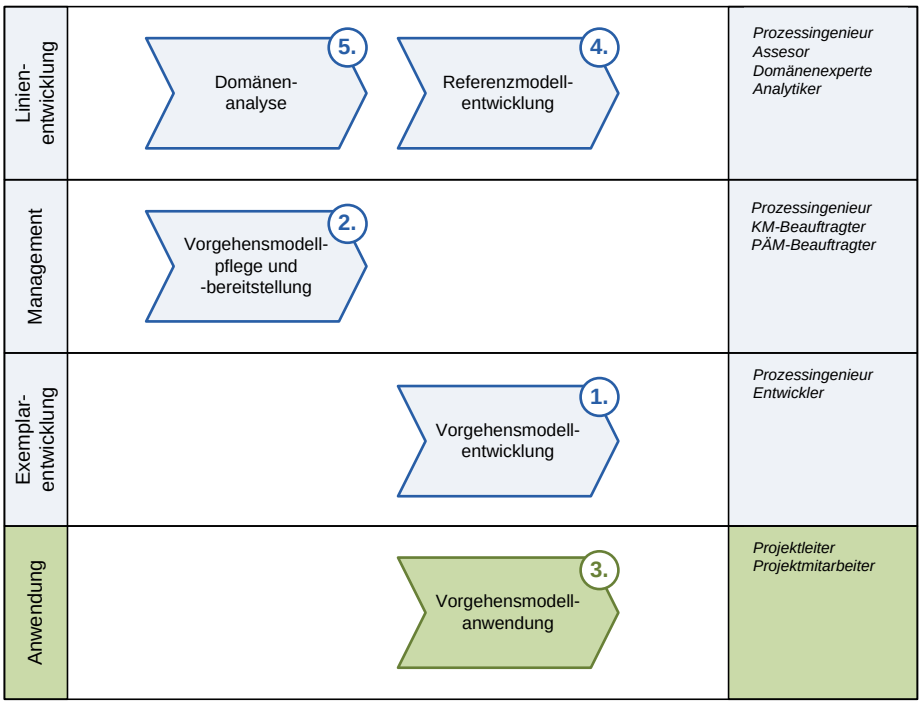


Abbildung 5.22.: Lebenszyklusmodell für Vorgehensmodelle und -linien als Menge von Prozessen nach dem Schema der (Software-)Produktlinienentwicklung (PLE)

Das Modell ist bewusst einfach gehalten und besteht im Wesentlichen nur aus den fünf gezeigten Prozessen. Diese sind *komposit* und können verschiedenartig verfeinert werden. Wir geben im Kontext dieser Arbeit eine mögliche Verfeinerung an. Gleichzeitig beschreiben wir auch die Schnittstellen zwischen den einzelnen Prozessen, um die verschiedenartige Detaillierung zu unterstützen und ein hohes Maß an Flexibilität zu erhalten.

Wir erfassen mit dem Lebenszyklusmodell drei wesentliche Bereiche, die wir zu Böckle et al. [BKPS04] (vgl. Abbildung 2.11) positionieren (Tabelle 5.3) und im Anschluss kurz beschreiben.

Bereits in Kapitel 2.3.2 wird auf die Notwendigkeit verschiedener Prozesse für den Kontext der *Linienentwicklung* hingewiesen. Wir definieren dies in unserem Ansatz mithilfe eines Entwicklungsprozesses für die Exemplarentwicklung, den wir in einen PLE-orientierten Ansatz integrieren. Als Schnittstelle hierfür dient uns ein expliziter *Management- und Bereitstellungsprozess*.

5.2.1. Verwaltung als zentraler Prozess

Im Zentrum des Lebenszyklusmodells in Abbildung 5.22 steht der Managementprozess (*2. Vorgehensmodellpflege und -bereitstellung*). Dieser ist die Anlaufstelle für die Neu- und Weiterentwicklung sowie die Anwendung von Vorgehensmodellen. Der zentrale Managementprozess arbeitet auf dem Metamodell, das wir im Rahmen dieser Arbeit vorgestellt haben. Konkret verwaltet er:

PLE	Lebenszyklusmodell	Bemerkung
Domain Engineering	Linienentwicklung	–
Application Engineering	Exemplarentwicklung	–
Repository Management	Management	Umfasst die Artefakte und die Prozesse zu deren Verwaltung.
–	Anwendung	Enthält die Anwendung des Vorgehensmodells zur Etablierung einer Feedbackschleife.

Tabelle 5.3.: Positionierung des Lebenszyklusmodells zum PLE-Referenzprozesses

- Vorgehensmodellkonfigurationen und
- Prozesskomponenten sowie
- alle sonstigen benötigten Elemente

als Entitäten und stellt ebenfalls die assoziierten Operationen auf den Entitäten bereit (zum Beispiel Konstruktions-/Konfigurations- oder Exportfunktionen). Der Managementprozess (*2. Vorgehensmodellpflege und -bereitstellung*, Abbildung 5.22) bekommt als Eingaben nicht nur ausschließlich Informationen aus *1. Vorgehensmodellentwicklung*, sondern auch aus den Prozessen der Linienentwicklung (*4.* und *5.*). Der Managementprozess stellt eine Menge möglicher Konfigurationen bereit, von denen der Hauptstrang eine ist.

5.2.2. Prozessschnittstellen

In Abbildung 5.22 haben wir bislang die notwendigen Prozesse gesammelt und im PLE-Referenzmodell positioniert. Die benannten Prozesse sind jedoch untereinander auch gekoppelt. Wir betrachten nun die Schnittstellen zwischen diesen Prozessen und gehen auch hier wieder vom zentralen Prozess *2. Vorgehensmodellpflege und -bereitstellung* aus. Abbildung 5.23 zeigt die Schnittstellen zwischen dem Management, der Exemplarentwicklung und der Anwendung von Vorgehensmodellen.

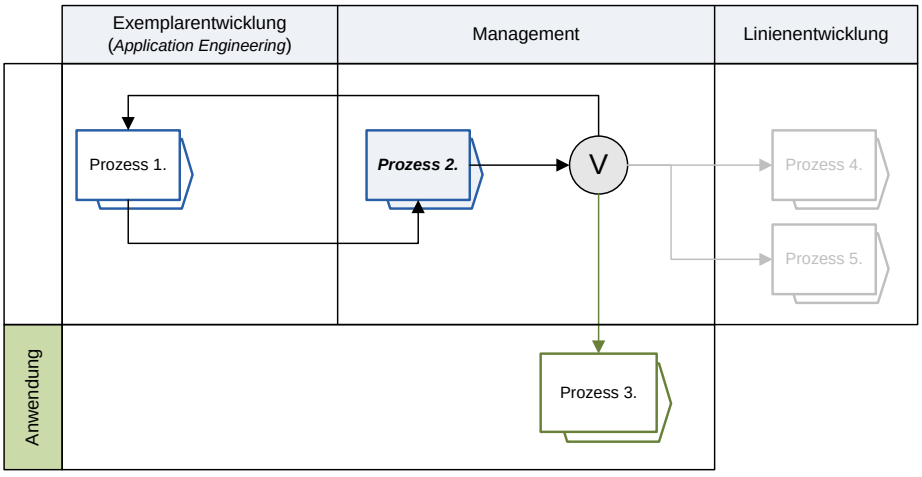


Abbildung 5.23.: Prozessschnittstellen zwischen Management, Exemplarentwicklung (Application Engineering) und Anwendung von Vorgehensmodellen

Die zentrale Rolle des Prozesses *2. Vorgehensmodellpflege und -bereitstellung* wird zum Beispiel beim Prozess *3. Vorgehensmodellanwendung* deutlich, der nicht direkt zur Ausführung gelangt, sondern Eingaben vom Prozess *2. Vorgehensmodellpflege und -bereitstellung* erfordert. Dies weist auf ein zentrales Konfigurationsmanagement hin, in dem Vor-

gehensmodelle beziehungsweise Teile davon verwaltet werden. Analog dazu verläuft die Entwicklung eines Vorgehensmodells. Der Prozess *1. Vorgehensmodellentwicklung* benötigt Eingaben aus dem Prozess *2. Vorgehensmodellpflege und -bereitstellung*, stellt ihm aber in jedem Fall seine Ergebnisse wieder als Eingabe zur Verfügung. Entwicklung und Weiterentwicklung[19] eines Vorgehensmodells finden also nur in den beiden Prozessen *1.* und *2.* statt.

Abbildung 5.24 zeigt die zweite Gruppe von Schnittstellen, die zu betrachten sind. Auch hier fungiert der Prozess *2. Vorgehensmodellpflege und -bereitstellung* als Anlauf- und Verteilerpunkt für die Prozesse *4. Referenzmodellentwicklung* und *5. Domänenanalyse*.

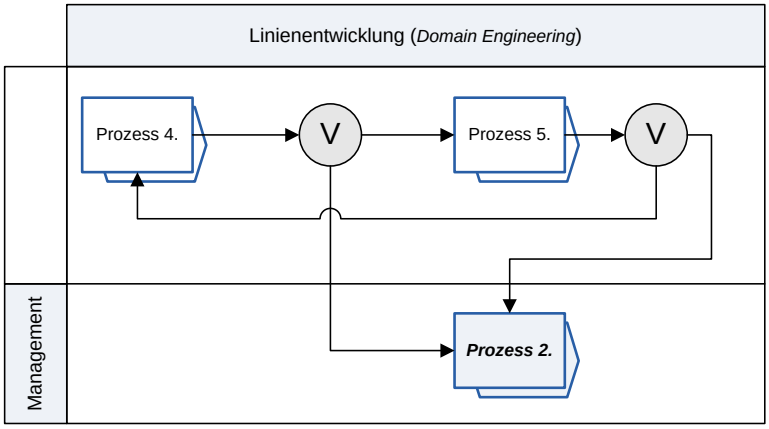


Abbildung 5.24.: Prozessschnittstelle zwischen Management und Linienentwicklung (Domain Engineering) von Vorgehensmodellen

Beide Prozesse erzeugen Eingaben für *2. Vorgehensmodellpflege und -bereitstellung* und erhalten über ihn selbst wieder Eingaben. Wir modellieren dadurch bereits auf dieser Ebene *Feedbackschleifen*, die kontinuierliche Informationsflüsse zwischen den einzelnen Prozessen des Lebenszyklusmodells ermöglichen.

Beispiel: Wird die Anwendung eines Vorgehensmodells abgeschlossen (*3. Vorgehensmodellanwendung*), d. h. wird ein Projekt abgeschlossen, stehen Erfahrungen mit dem eingesetzten Vorgehensmodell zur Verfügung. Diese werden in die Weiterentwicklung und Pflege zurückgeführt. Erfahrungen können dabei zum Beispiel auch aktualisierte Dokumentvorlagen oder aus Pragmatismus heraus optimierte Teilprozesse sein.

Feedback ist im Kontext einer Prozesslinie umständlicher zu handhaben, als bei nur einem einzelnen Vorgehensmodell. Das Feedback muss der gesamten Plattform zugeführt werden und wirkt sich somit auf alle aus der Plattform instanziierten Modelle aus. Hier ist dann zu unterscheiden, welchen Einfluss das Feedback hat. Mögliche Einflussgrößen können wir wie folgt klassifizieren:

- *Domänenbezogenes Feedback* – Hier sind grundlegende Änderungen an der Plattform zu erwarten. Die Konsequenzen sind hier am weitreichendsten.
- *Strukturbezogenes Feedback* – Hier sind Änderungen an Metamodellelementen also Strukturanpassungen zu erwarten.
- *Inhaltliches Feedback* – Hier sind Änderungen an konkreten Prozesskomponenten zu erwarten, beispielsweise durch Bugfixing oder sonstige Aktualisierungen.

Durch die so formulierten Feedback-Mechanismen sehen wir prinzipiell eine *proaktive Evolution* ([BKPS04], Seite 177ff.) der Gesamtplattform vor.

5.2.3. Prozess 1. Vorgehensmodellentwicklung

Die Entwicklung eines Vorgehensmodells ist ein äußerst komplexe Aufgabe, die neben den technischen Fähigkeiten und Fertigkeiten noch weitere *Soft Skills* von den Prozess-

19 Dies betrifft auch eventuelle Evaluierungen zum Beispiel mithilfe von Pilotprojekten, die wir im Prozess *1. Vorgehensmodellentwicklung* positionieren.

ingenieuren verlangt. Aus diesem Grund gibt es vielfältige Verfahren, eine Vorgehensmodellentwicklung durchzuführen. Dennoch lässt sie sich in der Regel immer auf die in Abbildung 5.25 gezeigten Schritte abstrahieren (vgl. Kapitel 2.3.3).

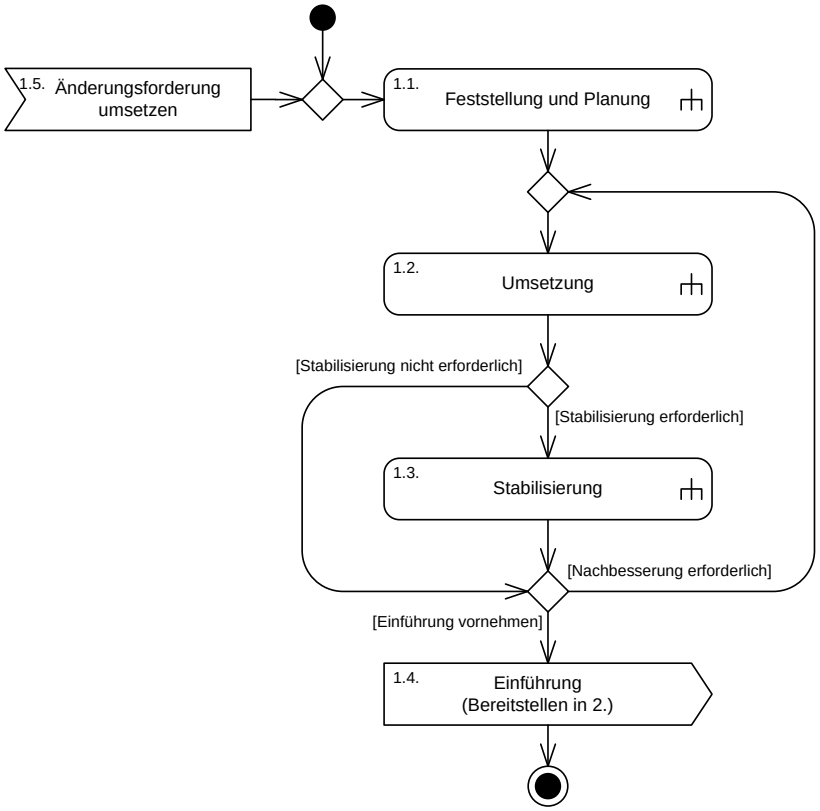


Abbildung 5.25.: Verfeinerung des Prozesses 1. Vorgehensmodellentwicklung

Der Prozess *1. Vorgehensmodellentwicklung* nimmt weiterhin einen besonderen Status ein, da er auch losgelöst vom Lebenszyklusmodell angewendet werden kann. Der Projekttyp *Einführung und Pflege eines organisationsspezifischen Vorgehensmodells* des V-Modell XT lässt sich zum Beispiel sehr gut durch diesen Prozess nachbilden.

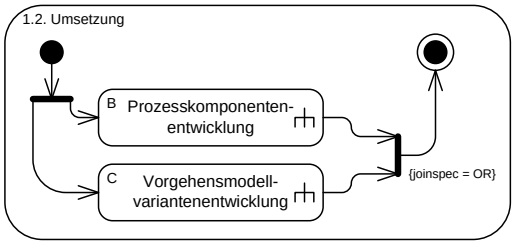


Abbildung 5.26.: Verfeinerung der Aktivität 1.2

Abbildung 5.26 zeigt die Verfeinerung der Aktivität zur Umsetzung von Entwicklungsarbeiten für ein Vorgehensmodell. Diese Aktivität ist hier für uns besonders interessant, da es im Wesentlichen diejenige ist, die Änderungen an Vorgehensmodellanteilen verursacht. Die Aktivität *1.1 Feststellung und Planung* dient im Wesentlichen der Bedarfsermittlung sowie dem Entwurf und der Umsetzungs- und Einführungsplanung. Weitere Informationen hierzu sind zum Beispiel [AEH+07] zu entnehmen. Die Aktivität *1.3 Stabilisierung* kann der Etablierung einer Pilotierungsphase dienen, in der Umsetzungen punktuell in Form von Zwischenergebnissen oder gesamtheitlich in Form von Vorabversionen geprüft werden. Für weitere Informationen zu Pilotierungen verweisen wir

zum Beispiel auf [NK05] oder Anhang A.1. Die Aktivität *1.4 Einführung (Bereitstellung)* bezieht sich im Wesentlichen auf zwei Punkte: Einerseits werden die entwickelten Vorgehensmodellanteile in ein zentrales Managementsystem zurückgespielt.

Andererseits kann bezogen auf einzelne Vorgehensmodelle ein einzelner Einführungsprozess begonnen werden. Dies ist dann sinnvoll, wenn noch kein Entwicklungsstandard eingeführt und somit noch kein Linienentwicklungsprozess etabliert ist. Dann ist es analog zur bisherigen Einführung des V-Modells möglich, das gegebene V-Modell herzunehmen, organisationsspezifisch anzupassen und dann in das Tagesgeschäft zu übernehmen (vgl. Kapitel 2.2.2).

Keine der gerade besprochenen Aktivitäten wirkt dabei jedoch strukturell auf ein Vorgehensmodell. Wir fokussieren daher die Aktivität *1.2 Umsetzung* und betrachten die sie verfeinernden Aktivitäten. Bevor wir das in den Abschnitten 5.2.6 bis 5.2.8 tun, betrachten wir noch die Schnittstelle des Prozesses (Tabelle 5.4).

Von	An	Beschreibung
2.		– Metamodell – Vorgehensmodellkonfigurationen (inkl. des Referenzmodells) – Prozesskomponenten – (sonstige durch das Metamodell beschriebene Elemente)
(extern)		– Konzeption für Vorgehensmodell – Anforderungen an Vorgehensmodellanpassung
	2.	– Vorgehensmodellkonfigurationen – Prozesskomponenten (neu, aktualisiert) – (sonstige durch das Metamodell beschriebene Elemente)

Tabelle 5.4.: Schnittstelle von Prozess 1. Vorgehensmodellentwicklung

Das Schema in Tabelle 5.4 stellt den Prozess ins Zentrum der Betrachtung und beleuchtet darauf aufbauend die Eingaben und Ausgaben sowie die korrespondierenden Quellen und Ziele. Wie zu sehen ist, ist der Austausch mit dem Managementprozess rein formal und mit den Ergebnissen dieser Arbeit beschreibbar. Prozess 2. liefert dabei die Metamodellbeschreibungen (vgl. Abschnitt 5.1 und Anhang B). Weitere Eingaben kommen aus sonstigen externen Quellen, worunter zum Beispiel auch ein Change Control Board zu verstehen ist. Mögliche Eingabe sind Konzepte und Anforderungen, die wir im Rahmen dieser Arbeit jedoch nicht weiter spezifizieren. Als Ergebnisse produziert dieser Prozess neue oder aktualisierte Prozesskomponenten sowie auch integrierte Vorgehensmodellkonfigurationen. Neben den bereits in dieser Arbeit vorgestellten Metamodellkonzepten, kann dieser Prozess auch aktualisierte Metamodellelemente sowohl als Eingabe bekommen wie auch als Ergebnis produzieren. Ein entsprechendes Beispiel liefert der Anhang B.

5.2.4. Prozess 2. Vorgehensmodellpflege und -bereitstellung

Die Pflege und Bereitstellung von Vorgehensmodellen im Kontext einer Prozesslinie ist kompliziert, da hier neben verschiedenen Vorgehensmodellkonfigurationen auch noch verschiedene Versionen und ein Referenzmodell zu beachten sind. Abbildung 5.27 zeigt die Verfeinerung des entsprechenden Prozesses.

Kurzbeschreibung. Dieser Prozess enthält verschiedene Aktivitäten. Startpunkt ist die Aktivität *2.1 Strategieauswahl, Planung und Releasemanagement*. Diese Aktivität dient unter anderem auch als unabhängige Managementaktivität im Kontext der Linienentwicklung. Sie kann zum Beispiel festlegen, ob der Entwicklungsstandard *proaktiv* oder *reaktiv* (vgl. Kapitel 2.3.2) oder eine Releaseplanung enthält, die festlegt, welche Entwicklungen zu welchem Release anstehen, beziehungsweise wann ein neues Release erscheint. Ein- und Ausgaben für diese Aktivität sind somit Anforderungen, Änderungsforderungen oder Problemmeldungen sowie Pläne. Diese Aktivität wird jedoch

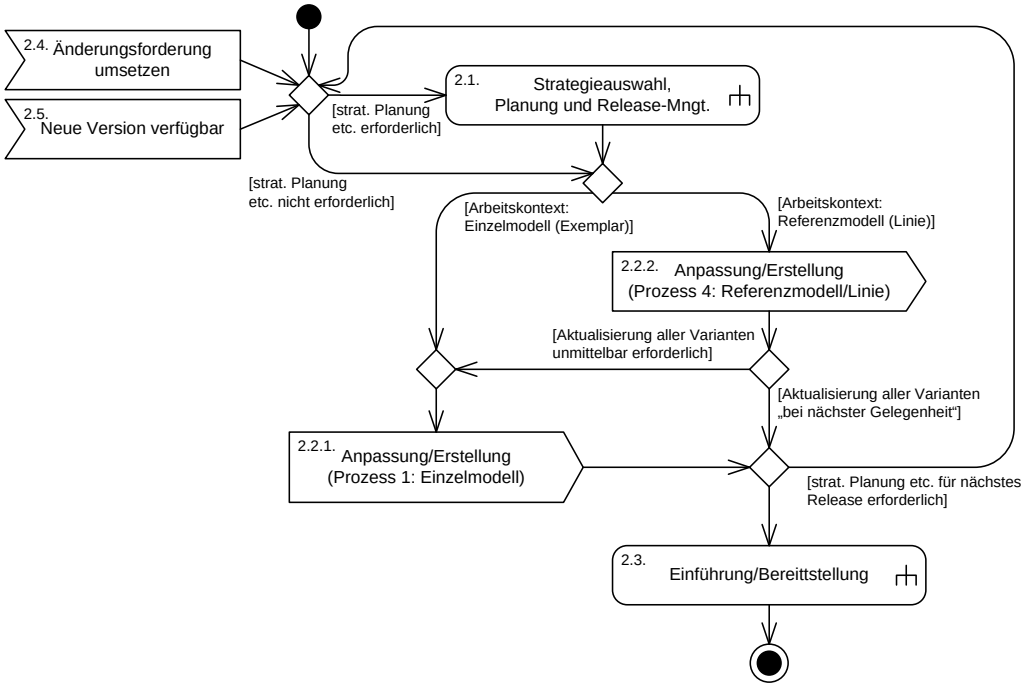


Abbildung 5.27.: Verfeinerung des Prozesses 2. Vorgehensmodellpflege und -bereitstellung

nur dann durchlaufen, wenn eine strategische Planung erforderlich ist. Es gibt jedoch auch Pflege- und Bereitstellungsprozesse, die unabhängig von einer solchen Planung laufen. Solche Prozesse sind beispielsweise kontinuierlich Pflege und Weiterentwicklung. Mögliche Auslöser für diesen Prozess (ohne strategische Anteile) sind durch die Signale *2.4 Änderungsforderung umsetzen* und *2.5 Neue Version verfügbar* gegeben. Wird beispielsweise im Rahmen des Prozesses *1. Vorgehensmodellentwicklung* eine neue Prozesskomponente erstellt, dann muss diese hier *nur* eingepflegt werden. Ein neues Release des Gesamtmodells ist hier nicht zwingend erforderlich.

Unabhängig von strategischen Entscheidungen wird in Abhängigkeit des *Arbeitskontextes* in entsprechende Unteraktivitäten verzweigt, die zum Beispiel Änderungsforderungen umsetzen. Arbeitskontexte sind entweder individuelle Vorgehensmodellvarianten (Exemplarentwicklung) oder der Entwicklungsstandard an sich (Linienentwicklung). Änderungen im Arbeitskontext des Entwicklungsstandards können dabei auch Änderungen/Anpassungen in einem oder mehreren Exemplaren verursachen, weshalb es einen entsprechenden Pfad gibt. Abschließend erfolgt eine Bereitstellung in der Prozessinfrastruktur. Während das Signal *2.4 Änderungsforderung umsetzen* in der Regel keine strategische Planung benötigt, ist sie für das Signal *2.5 Neue Version verfügbar* in der Regel obligatorisch. Signal *2.5* wird auch durch die Bereitstellung in der Aktivität *2.3* ausgelöst.

Beispiel: Um dies zu verdeutlichen, betrachten wir folgendes Beispiel: Aufgrund einer geänderten Anforderung in der Domäne (zum Beispiel ein neues Gesetz) ist es notwendig, eine der durch die Plattform bereitgestellten Vorgehensmodellkonfiguration anzupassen. In dieser Konfiguration werden einige Prozesskomponenten geändert, die gleichzeitig Teil des Referenzmodells sind. Die Anpassung der Konfiguration löst hier gleichzeitig eine Änderungsprozedur auf der Ebene des Referenzmodells aus, das die aktualisierten Prozesskomponenten einbindet. Eine Konsequenz ist mindestens eine Konsistenzprüfung des Referenzmodells.

Schnittstelle. Die Schnittstelle (Tabelle 5.5) dieses Prozesses ist sehr umfangreich. Sie muss hier anders als bei der Verwaltung von einzelnen Vorgehensmodellen nicht nur einen Pflegeprozess für ein Vorgehensmodell etablieren, sondern gleichzeitig mehrere

Konfigurationen unterstützen und dabei Vorgaben einer übergeordneten Plattform berücksichtigen.

Von	An	Beschreibung
1.		– Vorgehensmodellkonfigurationen – Prozesskomponenten (neu, aktualisiert) – (sonstige durch das Metamodell beschriebene Elemente)
4.		– Metamodell – Referenzmodell (Konfiguration etc.) – Prozesskomponenten (des Referenzmodells und allgemeine)
6.		– Beschreibung der Domäne, Normen, Vorgaben etc. – Prozessbeschreibungen – (Dokument-)Vorlagen
	1.	– Metamodell – Vorgehensmodellkonfigurationen (inkl. des Referenzmodells) – Prozesskomponenten – (sonstige durch das Metamodell beschriebene Elemente)
	3.	– Vorgehensmodellkonfiguration

Tabelle 5.5.: Schnittstelle von Prozess 2. Vorgehensmodellpflege und -bereitstellung

Die Schnittstelle zwischen diesem und dem Prozess *1. Vorgehensmodellentwicklung* besteht im Wesentlichen aus den bereits modellierten Größen (Abschnitt 5.1). Wie bereits zu Beginn eingeführt, fungiert dieser Prozess auch als zentraler Anlaufpunkt für die Bereitstellung eines organisationsspezifischen Vorgehensmodells. Dieses wird vom Prozess *3. Vorgehensmodellanwendung* als Eingabe benötigt und kann dort zum Beispiel analog zum V-Modell XT weiter angepasst werden (vgl. Kapitel 2.1.1 und 2.2.2). Nach Abschluss eines Projekts soll in unserem Modell das erworbene Wissen wieder zurück geführt werden. Hierzu verwenden ein *Feedback*-Konstrukt.

Feedback im Lebenszyklusmodell. Da Feedback in verschiedener Form vorliegen kann, widmen wir uns dem im Folgenden intensiver. Feedback kann in verschiedenen Größenordnungen vorliegen und orientiert sich in der Regel immer an konkreten Anwendungsfällen. Mögliche Formen von Feedback sind:

- Problemmeldungen und Änderungsforderungen (Change Requests)
- Feature Requests
- Angepasste Dokumentvorlagen
- Neu erstellte oder angepasste Prozessbeschreibungen
- Beispielunterlagen
- Erfahrungsberichte
- Projektspezifische Vorgehensmodelle
- Neue Prozesskomponenten
- ...

Der Prozess *2. Vorgehensmodellpflege und -bereitstellung* muss diese Eingaben sinnvoll verarbeiten. Für neue Elemente muss eine entsprechende Einpflegung vorgenommen werden.

5.2.5. Prozess 4. Referenzmodellentwicklung

Die Entwicklung eines *Referenzmodells* ist einer der zentralen Bestandteile einer Prozesslinie. Das Referenzmodell definiert gleichzeitig die Basisstruktur und die (minimal) gemeinsamen Inhalte aller Exemplare, die auf der Grundlage der Prozesslinie erstellt

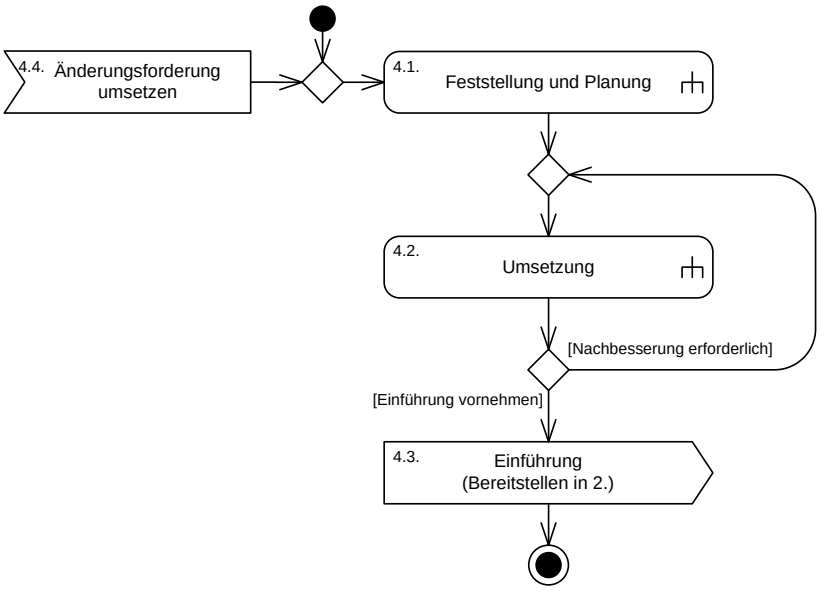


Abbildung 5.28.: Verfeinerung des Prozesses 4. Referenzmodellentwicklung

werden. Abbildung 5.28 zeigt den Prozess für die Modellierung eines Referenzmodells. Analog zum Prozess *1. Vorgehensmodellentwicklung* (Abschnitt 5.2.3) gibt es in diesem Prozess Aktivitäten, die auf die hier modellierten Strukturelemente wirken. Hier sind das die Aktivität *4.2 Umsetzung* und die sie verfeinernden Unteraktivitäten – siehe Abbildung 5.29.

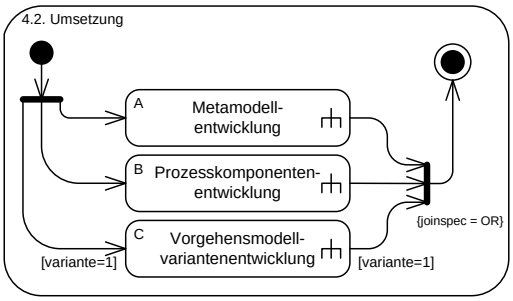


Abbildung 5.29.: Verfeinerung der Aktivität 4.2

Kurzbeschreibung. Die Entwicklung eines Referenzmodells ist ähnlich aufgebaut wie die Entwicklung eines einzelnen Vorgehensmodells. In der Tat haben wir im Kapitel 4.3 das Referenzmodell auch nur als ein (speziell ausgewiesenes) Vorgehensmodell von vielen möglichen bezeichnet. Für diesen Prozess haben wir jedoch eine andere Verfeinerung der Umsetzung vorliegen. Neben möglichen Arbeiten an Prozesskomponenten, muss immer mindestens eine Vorgehensmodellvariante bearbeitet werden. Das einfache Ändern von Prozesskomponenten ist nicht Aufgabe dieses Prozesses. Weiterhin können auch Arbeiten an Metamodellelementen vorgenommen werden. Die Verfeinerungen der Umsetzung charakterisieren die Referenzmodellentwicklung somit wie folgt: Im Rahmen der Referenzmodellentwicklung werden entweder neue Metamodellanteile erarbeitet und im Referenzmodell umgesetzt oder vorhandene Anteile zum Referenzmodell hinzugefügt (die Konfiguration zu bearbeiten).

Der Entwicklungs- und Einführungsprozess für das Referenzmodell ist wiederum ein kontinuierlicher Prozess. Das Referenzmodell bedarf kontinuierlicher Pflege und Weiterentwicklung und wird zu geplanten Zeitpunkten eingeführt (vgl. Abschnitt 5.2.4).

Schnittstelle. Die Schnittstelle dieses Prozesses ist in Tabelle 5.6 zu finden. Als Ausgaben liefert dieser Prozess das Metamodell und das Referenzmodell sowie dessen Bestandteile. Als Eingaben erhält dieser Prozess diejenigen Prozesskomponenten von Prozess 2., die Bestandteil des Referenzmodells sind. Weiterhin bekommt dieser Prozess alle Informationen aus der Domäne, die zur Erstellung und (inhaltlichen) Pflege des Referenzmodells erforderlich sind.

Von	An	Beschreibung
2.		– Prozesskomponenten
5.		– Beschreibung der Domäne, Normen, Vorgaben etc. – Prozessbeschreibungen – (Dokument-)Vorlagen
	2.	– Metamodell – Referenzmodell (Konfiguration etc.) – Prozesskomponenten (des Referenzmodells)
	5.	– Metamodell

Tabelle 5.6.: Schnittstelle von Prozess 4. Referenzmodellentwicklung

Wichtig ist an dieser Stelle auch die Weitergabe des Metamodells an den Prozess *5. Domänenanalyse*. Hiermit erhalten die Analytiker die Möglichkeit, Analysen und Entwürfe bereits in der Sprache des Entwicklungsstandards anzufertigen. Im Abschnitt 6.2 haben wir dies bereits einmal kurz gezeigt. Wir geben weitere Beispiele dafür in Kapitel 6 an.

5.2.6. Subprozess A. Metamodellentwicklung

Wir betrachten nun im Folgenden die Subprozesse, die die Umsetzungsaktivitäten verfeinern. Wir beginnen mit der Metamodellentwicklung, die in Abbildung 5.30 gezeigt ist.

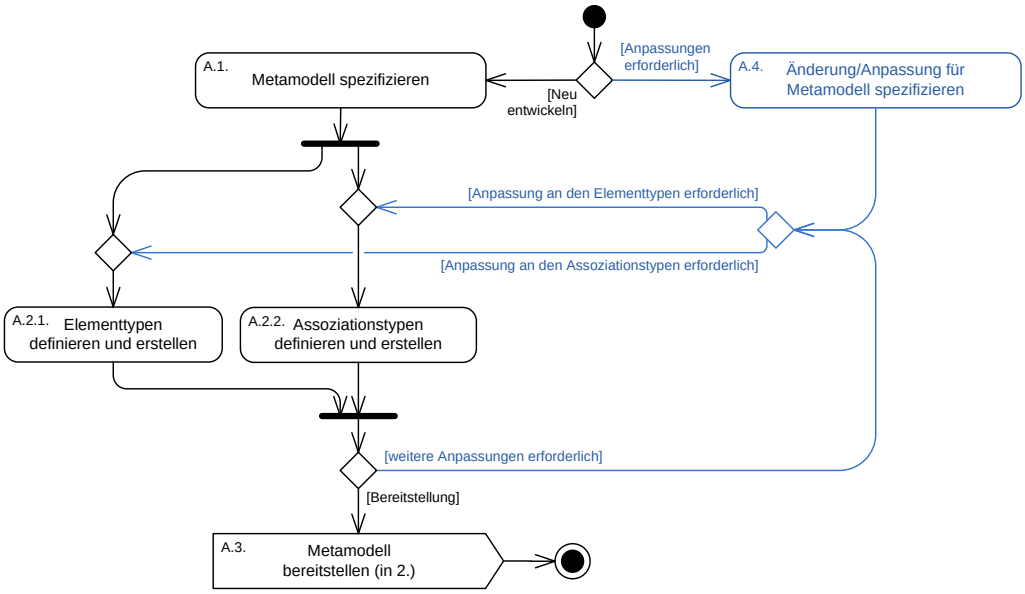


Abbildung 5.30.: Verfeinerung des Subprozess A. Metamodellentwicklung

Kurzbeschreibung. Die Metamodellentwicklung dient dazu, Struktur- und Beziehungstypen zu modellieren. Neben der Metamodellspezifikation (*A.1*) finden sich genau diese beiden Aufgaben hier wieder. Die Aktivität *A.2.1 Elementtypen definieren und*

erstellen dient dazu, die Entitäten des späteren Entwicklungsstandards zu definieren. Eingaben hierfür werden durch den Prozess *5. Domänenanalyse* bereitgestellt, da hier entsprechende Domänenmodelle erstellt werden müssen, die sich in späteren Prozessen wieder finden. Wie wir bereits auch in dieser Arbeit, insbesondere in den Kapiteln 3 und 5.1, mehrfach ausgeführt haben, ist die Erstellung eines passenden Beziehungsnetzes essenziell. Hierzu ist im Rahmen der Aktivität *A.2.2 Assoziationstypen definieren und erstellen* eine entsprechende Modellierung vorzunehmen. Auch die Eingaben für diese Aktivität sind aus der Domäne zur Verfügung zu stellen.

Wir unterscheiden bei der Metamodellentwicklung weiterhin zwischen der initialen Metamodellerstellung und einer kontinuierlichen Weiterentwicklung. Zu Beginn wird daher ermittelt, ob es sich um eine Neuentwicklung oder eine Anpassung handelt. Da die Operationen in den Aktivitäten gleich sind, unabhängig davon, ob eine Neuerstellung oder eine Änderung vorliegt, dient dieser zusätzliche Pfad dazu, die Aktivitäten selektiv nach Problem- oder Änderungslage anzuspringen. Der UML-Join muss daher auch mit einer *Oder*-Semantik ausgestattet sein. Wie die anderen Prozesse, die wir bislang vorgestellt haben, ist auch die Metamodellentwicklung prinzipiell ein kontinuierlicher Vorgang. Zyklen sind daher möglich. Nach Abschluss aller relevanten Arbeiten am Metamodell, wird dieses wieder auf der Plattform bereitgestellt.

Schnittstelle. Die Tabelle 5.7 zeigt die Schnittstellen des Subprozesses. Sie zeigt noch einmal, dass lediglich das Metamodell bearbeitet wird. Ausgaben des Metamodells an Prozess *5. Domänenanalyse* ermöglichen (ggf. werkzeugunterstützt) eine effizientere Domänenanalyse oder einen einfacheren Entwurf.

Von	An	Beschreibung
2.		– Metamodell
5.		– Beschreibung der Domäne, Normen, Vorgaben etc.
	2.	– Metamodell
	5.	– Metamodell

Tabelle 5.7.: Schnittstelle von Subprozess A. Metamodellentwicklung

Beispiel: Ein Beispiel für die punktuelle Anpassung des Metamodells eines umfassenden Standards ist bei den Metamodellversionen 1.2 und 1.2.1 des V-Modell XT zu finden. Hier wurde als marginale Änderung eine Versionsinformation eingeführt, die nur die Elementtypen ergänzt, jedoch auf die Beziehungstypen keinen Einfluss hat.

5.2.7. Subprozess B. Prozesskomponentenentwicklung

Die Entwicklung von Prozesskomponenten (Abbildung 5.31) findet nicht ausschließlich im Kontext eines konkreten Vorgehensmodells statt, sondern in der Regel übergeordnet auf der Ebene der Plattform. Prozesskomponenten beziehen sich somit auf verallgemeinerbare Problemstellungen, die eine Wiederverwendung erst ermöglichen. Nach der Erstellung stehen sie für die Konfiguration in Vorgehensmodellvarianten (auch dem Referenzmodell) zur Verfügung.

Kurzbeschreibung. Die Entwicklung von Prozesskomponenten ist sowohl fachlich als auch technisch sehr eng an das hier vorgestellte Metamodell (Abschnitt 5.1.2) gekoppelt. So werden nach einem Spezifikationsprozess die ermittelten Prozesselemente, die Dokumentation und die (internen) Abhängigkeiten festgelegt. Im Anschluss werden die benötigten Ressourcen allokiert (dies können zum Beispiel Beispieldokumente oder aber andere Prozesskomponenten sein, die referenziert werden müssen). Die fertige Prozesskomponente wird über die Plattform bereitgestellt und zur Nachnutzung angeboten.

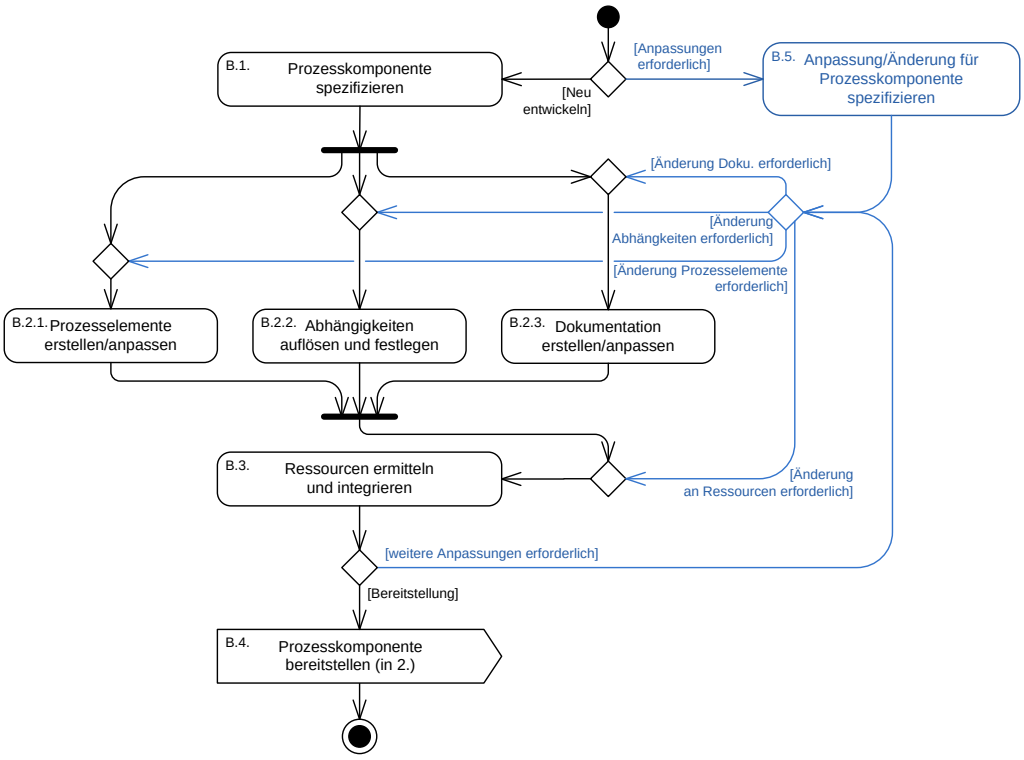


Abbildung 5.31.: Verfeinerung des Subprozesses B. Prozesskomponentenentwicklung

Analog zum gerade besprochenen Subprozess A. finden wir auch hier wieder den Alternativpfad über die Änderung/Aktualisierung. Auch hier können wieder Einzelaktivitäten selektiv angesprungen werden. UML-Joins müssen also wieder mit *Oder*-Semantik verwendet werden, um auch nach Minimaländerungen den Prozess geordnet verlassen und die Änderung bereitstellen zu können.

Schnittstelle. Die Schnittstelle dieses Prozesses (Tabelle 5.8) ist sehr schmal gehalten. Er benötigt als Eingaben lediglich das Metamodell und die Vorgaben aus der Domäne, um die Prozesskomponenten umzusetzen. Sollen Prozesskomponenten geändert oder aktualisiert werden, so erwartet dieser Prozess eine entsprechende Änderungsspezifikation. Als Ausgaben produziert dieser Prozess ausschließlich Prozesskomponenten, die einerseits der Referenzmodellentwicklung zur Verfügung stehen, andererseits auch direkt der Plattform zur Verfügung gestellt werden.

Von	An	Beschreibung
4.		– Metamodell – (Referenzinhalte)
5.		– Beschreibung der Domäne, Normen, Vorgaben etc. – Prozessbeschreibungen – (Dokument-)Vorlagen
	2.	– Prozesskomponenten
	4.	– Prozesskomponenten

Tabelle 5.8.: Schnittstelle von Subprozess B. Prozesskomponentenentwicklung

5.2.8. Subprozess C. Vorgehensmodellvariantenentwicklung

Ebenfalls Teil der Umsetzung ist der Subprozess *C. Vorgehensmodellvariantenentwicklung* (Abbildung 5.32), den wir nun noch behandeln wollen. Aufgrund des einheitlichen Konfigurationskonzeptes für Prozesskomponenten und Vorgehensmodelle (vgl. Abschnitt 5.1) fallen sofort Ähnlichkeiten zwischen diesem und dem Subprozess *B. Prozesskomponentenentwicklung* (Abbildung 5.31) auf. Im Unterschied zu diesem, werden hier jedoch keine neuen Prozesskomponenten mehr erstellt, sondern nur noch existierende konfiguriert. Ansonsten werden hier analog Abhängigkeiten aufgelöst, Dokumentation erstellt und Ressourcen allokiert. Zusätzlich können hier bereits Vorgaben hinsichtlich der Planung und Strukturierung eines Projekts (hier im Sinne des Konzepts Projektdurchführungsstrategie) gemacht werden. Dies setzt natürlich ein Planungsmodell voraus.

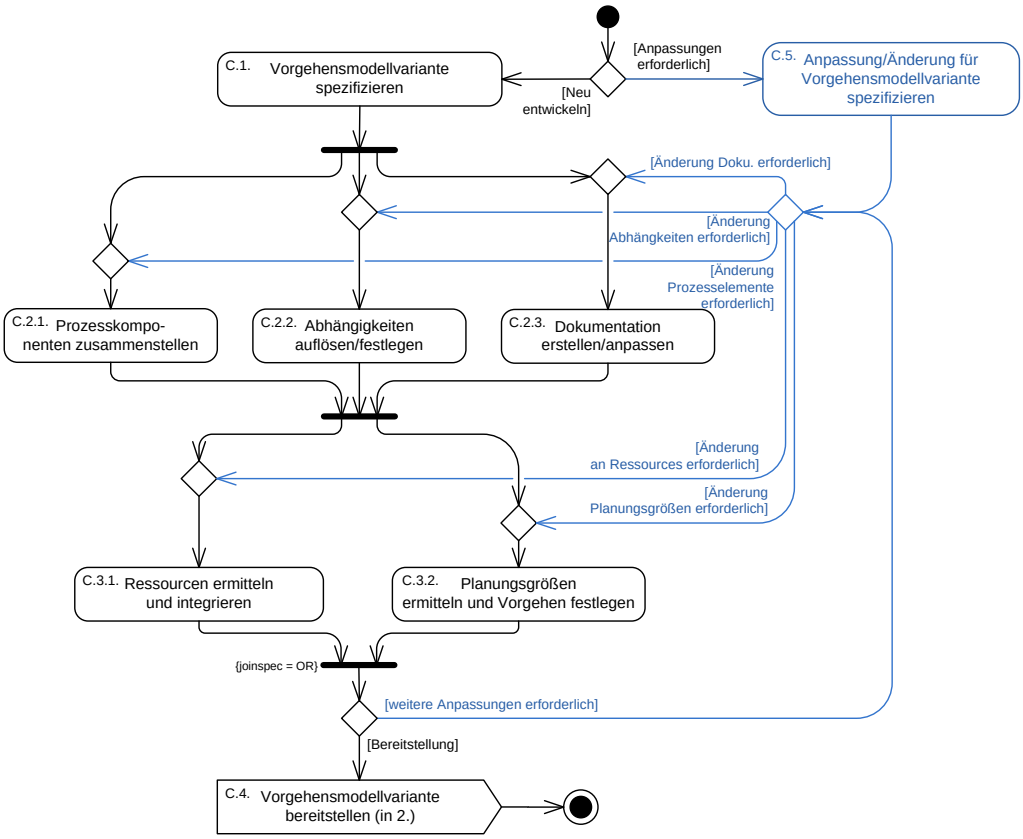


Abbildung 5.32.: Verfeinerung des Subprozess C. Vorgehensmodellvariantenentwicklung

Schnittstelle. Tabelle 5.9 zeigt die Schnittstelle des Subprozesses *C. Vorgehensmodellvariantenentwicklung*. Außer den Eingaben, die dieser Subprozess aus der Domänenanalyse bekommt, sind sämtliche Eingaben und Ergebnisse mit den Mitteln dieser Arbeit beschreibbar. Dies betrifft Prozesskomponenten, Metamodell (inkl. weiterer Metamodellelemente) sowie Konfigurationen.

Zustand eines Vorgehensmodells

Betrachten wir Vorgehensmodelle im Rahmen eines Lebenszyklusmodells, müssen wir mögliche Zustände eines Vorgehensmodells berücksichtigen. Jedes Mal, wenn eine Aktivität der gerade vorgestellten Modelle ausgeführt wird, ist in der Regel ein Vorgehensmodell, beziehungsweise eine Instanz davon entweder Eingabe- oder Ausgabeparameter. Beziehen wir uns jedoch auf das im Kapitel 4.1.2 gezeigte Hierarchiemodell

Von	An	Beschreibung
2.		– Prozesskomponenten
4.		– Metamodell – (Referenzinhalte)
5.		– Beschreibung der Domäne, Normen, Vorgaben etc. – Prozessbeschreibungen – (Dokument-)Vorlagen
	2.	– Vorgehensmodellvariante (in Form einer Konfiguration)

Tabelle 5.9.: Schnittstelle von Subprozess C. Vorgehensmodellvariantenentwicklung

(Abbildung 4.6) wird offensichtlich, dass ein Zustandsmodell im Wesentlichen nur *Prozesskomponenten* abdecken muss. Dies ergibt sich aus:

- Änderungen an Elementen, die in einer Prozesskomponente enthalten sind, implizieren eine Aktualisierung des Containers (der Prozesskomponente).
- Änderungen von Prozesskomponenten wiederum, schlagen auf alle Varianten durch, die die betreffenden Prozesskomponenten referenzieren.

Diese einfache Feststellung zeigt, dass der Zustand eines Vorgehensmodells oder einer Variante über die Zustände der enthaltenen Prozesskomponenten berechenbar ist. Wir führen also Zustände S (Abbildung 5.33, in Anlehnung an das Produktzustandsmodell des V-Modell XT) für Prozesskomponenten wie folgt ein:

$$S = \{\textit{In Bearbeitung}, \textit{Veröffentlicht}, \textit{Angepasst}, \textit{Veraltet}\}$$

In Bearbeitung: Bei der Bearbeitung von Prozesskomponenten und deren Inhalten ist die betreffende Prozesskomponente im Zustand in Bearbeitung. Für eine gegebene Prozesskomponente pk_1 und die enthaltenen Prozesselemente Pe_1 gilt dann:

$$\begin{aligned} &\textit{istInBearbeitung}\left(pk_1\right) = \text{true} \Leftrightarrow \\ &\exists\, pe \in Pe_1 :\ \textit{istInBearbeitung}\left(pe\right) = \text{true} \end{aligned}$$

, wobei *istInBearbeitung* eine Funktion von $S \rightarrow \{\text{true}, \text{false}\}$ ist.

Während eine Prozesskomponente somit automatisch im Zustand *In Bearbeitung* ist, wenn mindestens ein enthaltenes Element diesen Zustand hat, gilt der Umkehrschluss, dass die Prozesskomponente automatisch nicht im Zustand *In Bearbeitung* ist, wenn kein Element der Prozesskomponente in diesem Zustand ist. Der Zustandsübergang kann nur durch eine entsprechende Prüfung erfolgen.

Veröffentlicht: Die Veröffentlichung einer Prozesskomponente erfordert eine eigenständige Prüfung und ein Release, in dem sie verwendet wird. Analog zum letzten Zustand gilt für eine gegebene Prozesskomponente pk_1 und die enthaltenen Prozesselemente Pe_1:

$$\begin{aligned} &\textit{istVeröffentlicht}\left(pk_1\right) = \text{true} \Leftrightarrow \\ &\forall\, pe \in Pe_1 :\ \textit{istVeröffentlicht}\left(pe\right) = \text{true} \end{aligned}$$

Der Status *Veröffentlicht* ist für ein komplettes Release – also eine Version – eines Vorgehensmodells erforderlich. Der Zustand setzt sich transitiv fort, sodass wir ergänzend zu Kapitel 4 aussagen können, dass eine Version genau dann vorliegt, wenn alle im Vorgehensmodell referenzierten Prozesskomponenten im Zustand *Veröffentlicht* sind.

Angepasst: Aus dem Zustand *In Bearbeitung* führt der zweite mögliche Weg in diesen Zustand. Weiterhin kann dieser Zustand aus dem Zustand *Veröffentlicht* erreicht werden. In beiden Fällen zeigt er eine Prozesskomponente an, die den Bearbeitungsstatus hinter sich gelassen hat, aber nicht in einem Release im Sinne einer konsistenten Version ist. *Angepasst* signalisiert somit jede Form von Anpassung,

die wir in dieser Arbeit beschrieben haben. Analog zum letzten Zustand gilt für eine gegebene Prozesskomponente pk_1 und die enthaltenen Prozesselemente Pe_1:

$$istAngepasst\,(pk_1) = \text{true} \Leftrightarrow$$
$$\forall\, pe \in Pe_1 :\ istAngepasst\,(pe) = \text{true} \vee\ istVeröffentlicht\,(pe) = \text{true}$$

Veraltet: Der Zustand *Veraltet* dient als Aufbewahrungs- und Endzustand für eine Prozesskomponente. Er wird analog zum Zustand *In Bearbeitung* wie folgt festgestellt: Für eine gegebene Prozesskomponente pk_1 und die enthaltenen Prozesselemente Pe_1 gilt:

$$istVeraltet\,(pk_1) = \text{true} \Leftrightarrow$$
$$\exists\, pe \in Pe_1 :\ istVeraltet\,(pe) = \text{true}$$

Veraltet wird automatisch erreicht, wenn mindestens ein enthaltenes Element diesen Zustand erhält. Der Zustand kann aber auch ungeachtet der Zustände der Elemente direkt auf der Prozesskomponentenebene gesetzt werden.

Für alle Versionen (Releases) verlangen wir, dass alle in der Version referenzierten Prozesskomponenten im Zustand *Veröffentlicht* sind.

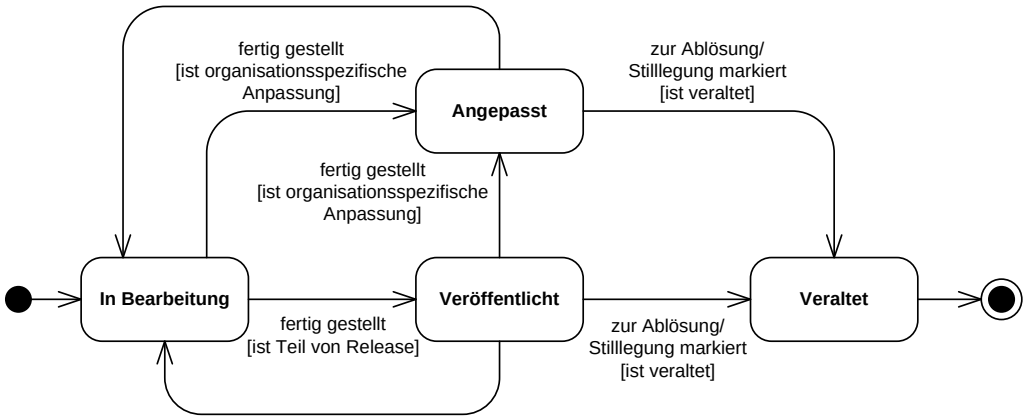


Abbildung 5.33.: Zustandsautomat für Prozesskomponenten

In Vorgehensmodellvarianten sind Prozesskomponenten im Zustand *Angepasst* möglich. Jedoch schließen wir Prozesskomponenten im Zustand *Veraltet* für ein Release aus. Für eine Vorgehensmodellvariante beziehungsweise eine organisationsspezifische Anpassung sind Prozesskomponenten im Zustand *Veraltet* jedoch durchaus denkbar, da hier möglicherweise Teilprozesse ähnlich zu so genannten *Legacy-Komponenten* Verwendung finden können.

5.3. Werkzeugkonzept und -unterstützung

Sowohl das Metamodell als auch das Lebenszyklusmodell weisen, trotz des erklärten Ziels der maximalen Einfachheit, bereits ein hohes Maß an Komplexität und Integration auf. Eine Unterstützung durch Werkzeuge sehen wir nicht nur als erwünscht, sondern auch als erforderlich an. In diesem Abschnitt konzipieren wir eine Werkzeugunterstützung, die insbesondere die Umsetzung des Metamodells und des Lebenszyklusmodells berücksichtigt. Im Anhang C gehen wir noch detaillierter darauf ein.

In Abbildung 5.34 ist eine Skizze der angestrebten Verteilungsarchitektur und der Hierarchie des Werkzeugs im System der Prozesskomponenten zu sehen. Wie der Abbildung zu entnehmen ist, liegt auch hier wieder die Plattformidee der Produktlinienentwicklung zugrunde. In der Bibliothek werden Prozesskomponenten und Vorgehensmodellkonfigurationen aufbewahrt. Außerdem ist bereits die Aufbewahrung anderer

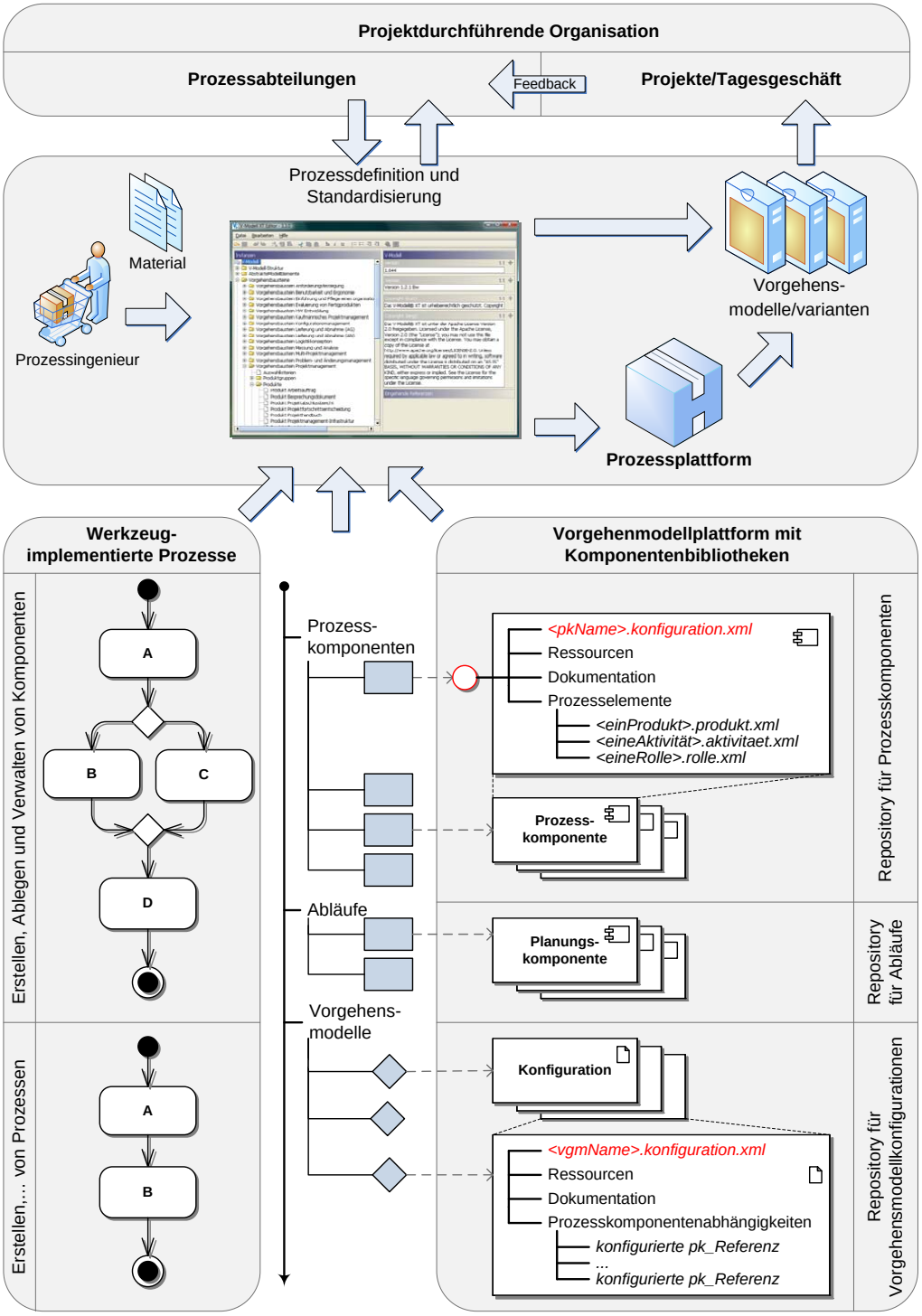


Abbildung 5.34.: Skizze des IMV-Werkzeugkonzepts

Komponententypen skizziert – hier am Beispiel eines *Abläufe*-Knotens. Weiterhin sind auch die Prozesse skizziert, die der Erstellung und Pflege der Plattforminhalte dienen.

Für die Prozesskomponenten ist in Abbildung 5.34 auch die interne Organisationsstruktur zu sehen. Die Konfiguration einer Prozesskomponente stellt als Schnittstelle den Zugang zu den Inhalten her. Wir haben dies in der Abbildung skizziert. In der Tat sind die Prozesskomponenten jedoch als geschlossene Dateien konzipiert. Werkzeuge sollten die interne Struktur vor dem Anwender verbergen. Die Abbildung zeigt die Elemente des Metamodells (Abschnitt 5.1.1): Prozesselemente, Dokumentation und Ressourcen. Über allem steht eine Konfiguration. Dies ist die Konfiguration der Prozesskomponenten; vergleichbar mit einer *Manifest*-Datei. Unter dem *Ressourcen*-Ast sind zum Beispiel Dokumentvorlagen oder ähnliches zu finden. Basierend auf den Komponenten der Plattform können dann Vorgehensmodelle konfigurativ erstellt werden. Die Abbildung5.34 skizziert dies durch die Referenzierung von Prozesskomponenten. Dabei werden gleichzeitig mithilfe der Prozesskomponentenabhängigkeiten die Verknüpfungen hergestellt.

Parallel dazu integriert das Werkzeugkonzept auch die Element des Lebenszyklusmodells in Form der in Abschnitt 5.2 beschriebenen Prozesse. Neben den Prozessen zur Erstellung von Prozesselementen und Prozesskomponenten sind hier insbesondere die Plattformprozesse relevant, die die Verwaltung von Prozesskomponenten und Vorgehensmodellen und Varianten realisieren. Diese ordnen sich dann insbesondere in den Prozess 2. (Abnschnitt 5.2.4) ein. Besonders wichtig sind hierbei Prozesse, wie zum Beispiel die Δ-Analyse [SM06c, SM06a] oder Unterstützung im Bereich der Ermittlung von allgemeinen Modellunterschieden [Kel07, TBWK07]. Auch die Versionskontrolle [OK02] im Rahmen des Konfigurationsmanagements ist hier besonders zu berücksichtigen. Aufgrund der gegebenen Fähigkeiten zur variablen Anpassung von Vorgehensmodellen, muss auf der Plattform mit Änderungen entsprechend umgegangen werden.

Die Prozesse und Strukturen sind in unserem Konzept in einem Autorenwerkzeug wie dem V-Modell XT Editor zu integrieren. Dieses stellt über das Metamodell die zu bearbeitenden Strukturen bereit. Anhand der beschriebenen Prozesse stellt es entsprechende Unterstützung für den Anwender bereit, um Arbeitsschritte für Verwaltung, Pflege und Export durchzuführen. Das Werkzeug stellt dabei die Abstraktion zur Organisation her. Hierüber wird der in Kapitel 2.3 motivierte Feedbackzyklus etabliert, wobei eine Prozessabteilung die Prozessdefinition und die Pflege übernimmt. Die Projektabteilungen übernehmen die Prozessbeschreibungen und liefern Feedback aus den Projekten.

Zusammenfassung

Wir haben aufbauend auf Kapitel 4 das abstrakte Basismodell auf der Grundlage der UML 2 konkretisiert. Wir führen die UML-Modellierung dabei anhand der Ergebnisdefinition aus Abbildung 1.2 durch und betrachten im Abschnitt 5.1 strukturelle Aspekte, insbesondere das *Vorgehensmetamodell*. Im Abschnitt 5.2 erstellen wir das *Lebenszyklusmodell* und schließen dann im Abschnitt 5.3 mit einem Werkzeugkonzept. Wir führen hier die Integration zu einem *Integrierten Modellierungsansatz für Vorgehensmodelle* (IMV) durch.

Im ersten Teil konkretisieren wir das abstrakte Basismodell und stellen im Abschnitt 5.1.1 zunächst die Metamodellanteile vor, die die Abbildung von:

- Prozesselementen und Prozesselementabhängigkeiten
- Prozesskomponenten und Prozesskomponentenabhängigkeiten
- Konfigurationen (im Kontext Prozesskomponente und Vorgehensmodell)

umsetzen. Mit diesen Metamodellanteilen realisieren wir ein *Framework*, das weitgehend ausgestaltet und erweitert werden kann (vgl. Abbildung C.1).

Diese Ausgestaltung nehmen wir im Abschnitt 5.1.2 vor. Unsere Modellierung setzt dabei die Terminologie aus Kapitel 2.4.1 um und orientiert sich stark am V-Modell XT-Konzept Vorgehensbaustein. Auf dieser Grundlage entwerfen wir detailliert *Prozesskomponenten* und konkretisieren im Anschluss die Submodelle von Vorgehensmodellen

auf der Basis von *Prozesselementen*. Bei der Definition von *Produkt-*, *Rollen-* und *Aktivitäts(sub)modellen* geben wir auch Definitionen von *Prozesselementabhängigkeiten* an. Die Abhängigkeitstypen detaillieren wir darüber hinaus noch, indem wir Einschränkungen, Constraints etc. angeben. Die Modellierung der einzelnen Strukturmodelle ist von Beispielen durchsetzt, die auf der Grundlage des V-Modell XT die unmittelbare Anwendung demonstrieren.

Nachdem wir die Strukturen innerhalb konkreter Prozesskomponenten definiert haben, beschreiben wir im Abschnitt 5.1.3 die weiter gehende Integration von Prozesskomponenten zu *Vorgehensmodellen* und die dafür notwendigen *Prozesskomponentenabhängigkeiten*. Auch hier geben wir wieder Beispiele an, die aktuelle Szenarios aus dem V-Modell XT aufgreifen mit *IMV* remodellieren.

Weiterhin betrachten wir im Abschnitt 5.1.4 konkrete Beispiele für die Anwendung der in Kapitel 4.3.3 beschriebenen Variabilitätsoperationen. Wir geben mit den Anwendungsfällen:

- Produktersetzung,
- Produktspezialisierung,
- Produkterweiterung,
- Aktivitätsersetzung und
- Aktivitätsausgestaltung

einige signifikante Beispielszenarios an, die insbesondere in der Motivation aus dem V-Modell-Kontext hohe Relevanz besitzen. Diese geben jedoch nur einen ersten Einblick, zeigen aber, wo weitere Operationen für andere Anpassungstypen – auch in Form umfassenderer graphentheoretischer Transformationsansätze – zu integrieren sind.

Abschnitt 5.2 enthält die Modellierung und Beschreibung des *Lebenszyklusmodells*. Dieses orientiert sich am so genannten *Referenzprozess für die Software Produktlinien-Entwicklung* (Kapitel 2.3.2). Das Lebenszyklusmodell führt insgesamt fünf allgemeine Prozesse auf, zwischen denen wir mögliche Pfade und Schnittstellen beschreiben. Darüber hinaus geben wir in den Abschnitten 5.2.3 bis 5.2.8 detaillierte Beschreibungen für die drei für diese Arbeit relevanten Prozesse an:

- Vorgehensmodellentwicklung
- Vorgehensmodellpflege und -bereitstellung
- Referenzmodellentwicklung

Wir halten die Beschreibung soweit möglich sehr allgemein und verweisen bei Schnittstellen und Aktivitäten zwischen und in diesen Prozessen nur auf Elemente, die im Rahmen dieser Arbeit (insbesondere Abschnitt 5.1) erstellt wurden. Alle Prozesse und Subprozesse liegen in Form von UML 2 Aktivitätsdiagrammen vor und sind so beschrieben, dass sie einfach instanziiert und ausgestaltet werden können. Eine mögliche Ausgestaltung der Prozesse findet sich zum Beispiel in Funktionen unterstützender Werkzeuge (siehe Abschnitt 5.3).

Wesentlich bei den Prozessen ist jedoch, dass die Pflege und Bereitstellung als *der* zentrale Prozess beschrieben wird. Dieser Prozess ist kontinuierlich zu absolvieren, womit einerseits eine zentral organisierte Verteilung von Vorgehensmodellen, Varianten und sonstigen Artefakten umgesetzt wird. Andererseits – und viel wichtiger – ist die Etablierung einer definierten *Feedback*-Schleife, sodass auf der Basis dieses Lebenszyklusmodells eine definierte, kontinuierliche Prozesspflege und Prozessverbesserung aufgebaut werden kann. Durch die ebenfalls zentrale Verteilung des Metamodells wird ein wichtiger Schritt in Richtung Konformität getan, da somit ein Entwicklungs-Framework bereitgestellt wird. An dieser Stelle sind dann viele Konzepte und Werkzeuge zur Verwaltung, zum Ablgeich und zur Rückführung von Varianten integrierbar. In der Skizze des Werkzeugkonzepts haben wir dies beschrieben.

Den Abschluss bildet die Konzeption einer Werkzeugunterstützung in Abschnitt 5.3. Hier werden die erarbeiteten Konzepte für die Umsetzung mithilfe von Werkzeuge aufbereitet und betrachtet.

6. Authoring, Tailoring und Anwendung

In den vorangegangenen Kapiteln haben wir ein formales Basismodell und darauf aufbauend ein Metamodell und ein Lebenszyklusmodell entwickelt. Wir haben bereits punktuelle Beispiele für die jeweiligen Konzepte aufgeführt. In diesem Kapitel widmen wir uns der Anwendung des im letzten Kapitel 5 vorgestellten *integrierten Modellierungsansatzes für Vorgehensmodelle* (IMV). Wir betrachten dabei zunächst allgemein die Anwendung und fokussieren danach die Organisations- und Projektebene. Auf der Organisationsebene beleuchten wir alle wesentlichen Schritte von der Entwicklung von Prozesselementen bis hin zur Erstellung von Vorgehensmodellvarianten. Da die Projektebene nicht im Fokus dieser Arbeit liegt, diskutieren wir hier nur Optionen und Potenziale des IMV-Ansatzes. Alle Beispiele, auf die wir uns in diesem Kapitel beziehen, stammen aus dem V-Modell XT.

Am Ende dieses Kapitels hat der Leser Einblick in den IMV-Ansatz bekommen und die wesentlichen Etappen exemplarisch durchlaufen. Die Optionen und Potenziale für die weiteren, dem Konstruktionsprozess nachgelagerten Lebensabschnitte eines Vorgehensmodells sind aufgezeigt.

6.1. Anwendung des integrierten Modellierungsansatzes

Für die Anwendung des *integrierten Modellierungsansatzes für Vorgehensmodelle* (IMV) ergänzen wir zunächst die Anwendungsfälle aus Kapitel 3.6.2 um diejenigen, die für die Implementierung und Anpassung eines Vorgehensmodells wichtig sind. Abbildung 6.1 zeigt sie mitsamt der Rollen, die hier relevant sind.

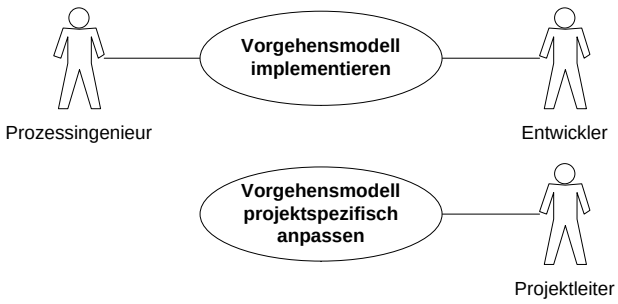


Abbildung 6.1.: Anwendungsfälle für die Umsetzung und Anpassung eines Vorgehensmodells

Der Anwendungsfall *Vorgehensmodell implementieren* ist über so genannte *Extension Points* (siehe [JRH+03], Seite 193ff.) mit den meisten anderen in Abbildung 3.23 verknüpfbar. Wir greifen ihn hier auf, um die Vorgehensweisen und Aufgaben in der Umsetzung der Vorgehensmodellanteile zu detaillieren. Wir führen weiterhin die Rolle des Entwicklers ein. Unter einem Vorgehensmodellentwickler fassen wir alle Personen zusammen, die schaffend und inhaltlich zu einem Vorgehensmodell beitragen. Nicht immer müssen sämtliche Implementierungsaufgaben von einem Prozessingenieur durchgeführt werden, jedoch obliegt ihm die Konzeption und Planung, was im Kapitel 3.6.2 entsprechend herausgestellt wird. Wir beschreiben zunächst die beiden für dieses Kapitel relevanten Anwendungsfälle näher.

7. Vorgehensmodell implementieren. In der Implementierung eines Vorgehensmodells, genauer in der Implementierung von Vorgehensmodellanteilen, werden spezifische Prozesselemente angelegt, mit Werten befüllt und mit anderen Prozesselementen in Beziehung gesetzt. Die wesentlichen Schritte und Aufgaben für diesen Anwendungsfall werden in den Subprozessen *B. Prozesskomponentenentwicklung* (Kapitel 5.2.7) und *C. Vorgehensmodellvariantenentwicklung* (Kapitel 5.2.8) festgelegt. Neben der Instanziierung von Prozesselementen können auch Dokumentationsanteile angefertigt oder Ressourcen inkludiert werden. Dieser Anwendungsfall produziert mindestens eine Prozesskomponente, beziehungsweise eine Prozesskomponentenversion.

Akteur: Prozessingenieur, Entwickler

Auslöser: Eine Anforderung im Kontext des Entwicklungsstandards (Domäne, Linie) oder im Kontext eines konkreten Vorgehensmodells (Exemplar) erfordert die Umsetzung definierter Prozessanteile durch Artefakte/Produkte, Aktivitätsbeschreibungen etc.

Ergebnis: Es liegt mindestens eine Prozesskomponente (Neu-/Initialentwicklung) oder eine Prozesskomponentenversion (Aktualisierung, Anpassung) vor.

Vorbedingungen: Es ist ein Metamodell verfügbar, dass den technischen Rahmen vorgibt. Es sind ggf. notwendige Vorgehensmodellanteile (Prozesskomponenten, Konfigurationen, Ressourcen etc.) verfügbar. Es sind Anforderungen für die Neuentwicklung/Aktualisierung verfügbar.

Nachbedingungen: Alle Anforderungen beziehungsweise Änderungsforderungen sind in Form von neuen/aktualisierten Prozesskomponenten umgesetzt. Ggf. steht eine neue Konfiguration für eine Vorgehensmodellvariante zur Verfügung.

Dieser Anwendungsfall adressiert somit explizit die entwicklungsbezogenen Anteile der Vorgehensmodellerstellung. Ergebnistypen sind wie bereits aufgeführt: Prozesselemente, die zu Prozesskomponenten zusammengefasst sind. Je nach Tragweite berücksichtigen wir hier auch schon Vorgehensmodellvarianten, ohne jedoch einen redundanten Anwendungsfall in Bezug auf die Auflistung in Kapitel 3.6.2 erzeugen zu wollen. Einen Einblick in den Entwicklungsprozess haben wir bereits kurz in Kapitel 6.2 gegeben. Vertiefen wollen wir diesen Anwendungsfall jedoch noch einmal im Abschnitt 6.2.3

8. Vorgehensmodell projektspezifisch anpassen. Die projektspezifische Anpassung eines Vorgehensmodells nimmt für generische Vorgehen weitgehende Anpassungen hinsichtlich definierter Parameter für verschiedene Projektkontexte vor. Bei der projektspezifischen Anpassung werden solche Parameter identifiziert und gemäß der vorliegenden Situation entsprechend mit Werten befüllt.

Akteur: Projektleiter

Auslöser: Ein neues Projekt soll auf der Grundlage eines Vorgehensmodells initialisiert und aufgesetzt werden. Das vorliegende Vorgehen ist generisch und muss in diesem Kontext den Projektspezifika angepasst werden.

Ergebnis: Projektspezifisches Vorgehensmodell.

Vorbedingungen: Es gibt ein Vorgehensmodell (entweder aus einem Entwicklungsstandard abgeleitet oder individuell) und entsprechende Anpassungsmechanismen.

Nachbedingungen: Das projektspezifische Vorgehensmodell ist konsistent und für den Fall der Ableitung aus einem Entwicklungsstandard konform zu diesem.

Die projektspezifische Anpassung steht nicht im Zentrum dieser Arbeit. Jedoch haben einige der Konzepte, die wir hier entwickelt haben, Konsequenzen für diesen Bereich, den wir in Kapitel 1.2 als *Tailoringstufe 2* bezeichnet haben. Wir gehen daher im Abschnitt 6.3 noch etwas darauf ein.

6.2. Modellierung von Vorgehensmodellstrukturen

Wir können mit dem Vorgehensmetamodell aus Kapitel 5.1 beliebige Strukturen auf der Basis von Prozesselementen und Prozesselementabhängigkeiten modellieren. Wei-

terhin können wir Prozesskomponenten eines Vorgehensmodells inklusive der relevanten Prozesselemente und deren Abhängigkeitsstrukturen sowie Konfiguration von Prozesskomponenten modellieren. Mit diesem Abschnitt betrachten wir die verschiedenen *Konfigurationsebenen* für Prozesselemente und Prozesskomponenten. Wir greifen dabei auf einen Auszug des V-Modell XT als Beispiel zurück, um V-Modell-Elemente mit unserem Modell zu remodellieren. Wir gehen dabei schrittweise vor und betrachten:

- Prozesskomponenten: Enthaltene Prozesselemente und deren Abhängigkeiten
- Vorgehensmodell: Enthaltene Prozesskomponenten und deren Abhängigkeiten
- Organisationsspezifische Anpassung: Verfeinerungen und Spezialisierungen

6.2.1. Modellierung von Prozesskomponenten

Wir wollen zunächst Vorgehensbausteine aus dem V-Modell herausgreifen und exemplarisch als Prozesskomponenten modellieren. Da hier ein Element realer Komplexität vorliegt, abstrahieren wir auch in diesem Beispiel, um es überschaubar zu halten. So verzichten wir auf eine vollständige Modellierung und greifen nur einige repräsentative Details heraus.

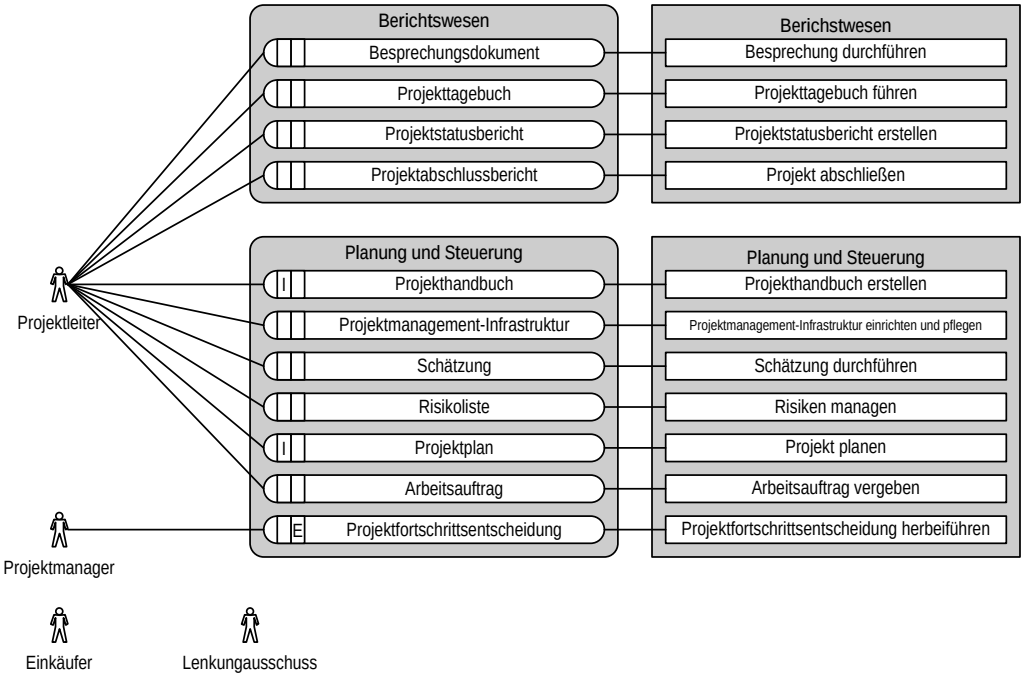


Abbildung 6.2.: Vorgehensbaustein Projektmanagement des V-Modell XT

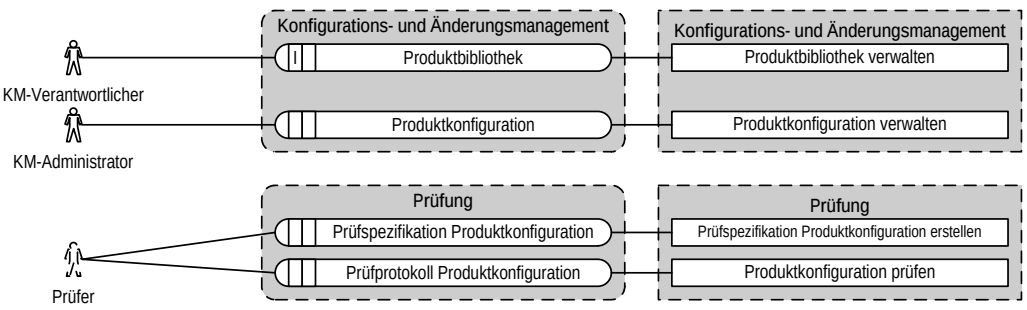


Abbildung 6.3.: Vorgehensbaustein Konfigurationsmanagement des V-Modell XT

Als Beispiel wählen wir die Vorgehensbausteine *Projektmanagement*, der in grober Struktur in Abbildung 6.2 zu sehen ist, und *Konfigurationsmanagement* (Abbildung 6.3). Wir greifen die Inhalte dieser Vorgehensbausteine auf und modellieren sie als Prozesskomponenten mit entsprechenden Prozesselementen und Beziehungstypen. Wir ermitteln zuerst auf der Basis der Produkte, Themen, Aktivitäten und Teilaktivitäten sowie der Rollen die zu modellierenden Prozesselemente und deren Abhängigkeiten. Diese resultieren dann in einer Konfiguration der Prozesskomponente. Wir instanziieren unser Metamodell und geben im Folgenden Modelle für Prozesskomponenten an. Wir vereinfachen dabei die grafische Darstellung und geben bei den Modellelementen explizit nicht mehr die Instanzinformationen an. Abbildung 6.4 zeigt die Prozesskomponente *Projektmanagement* in der selben Granularitätsstufe, wie sie die Abbildungen 6.2 und 6.3 des V-Modells haben. Alle Produkte und Aktivitäten des Vorgehensbausteins Projektmanagement sind enthalten. Die Beziehungstypen sind in unserer Modellierung benannt und bezeichnen somit Instanzen der in den vorangegangenen Abschnitten modellierten Beziehungstypen.

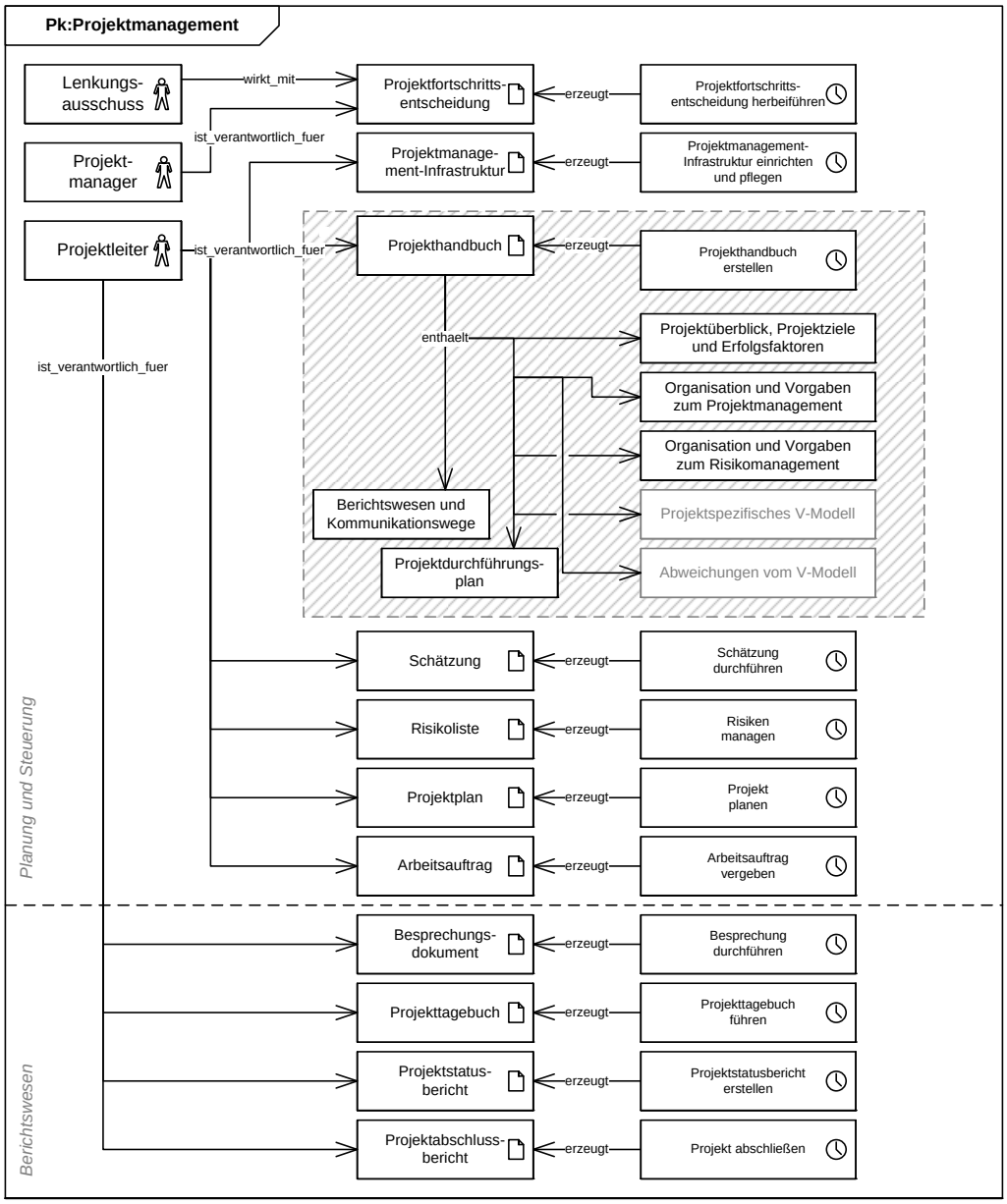


Abbildung 6.4.: Vorgehensbaustein Projektmanagement des V-Modell XT als Prozesskomponente modelliert

Für die grobgranulare Modellierung einer Prozesskomponente auf dieser Ebene genügen uns, wie in Abbildung 6.4 zu sehen, nur sehr wenige Beziehungstypen: *erzeugt*, *enthält*, *ist_verantwortlich_fuer* und *wirkt_mit*. Sie stellen die groben Zusammenhänge her. Im Metamodell haben wir jedoch weitere Beziehungstypen vorgesehen, die einmal die Strukturmodelle weiter detaillieren, andererseits zum Beispiel aber auch V-Modell-ähnliche Produktabhängigkeiten abbilden können. Eine Modellierung auf der Detaillierungsstufe von Abbildung 6.4 ist in der Regel zu grob. Daher greifen wir exemplarisch den hervorgehobenen Bereich heraus und detaillieren ihn in Abbildung 6.5 (auch hier verzichten wir wieder auf die Instanzinformationen).

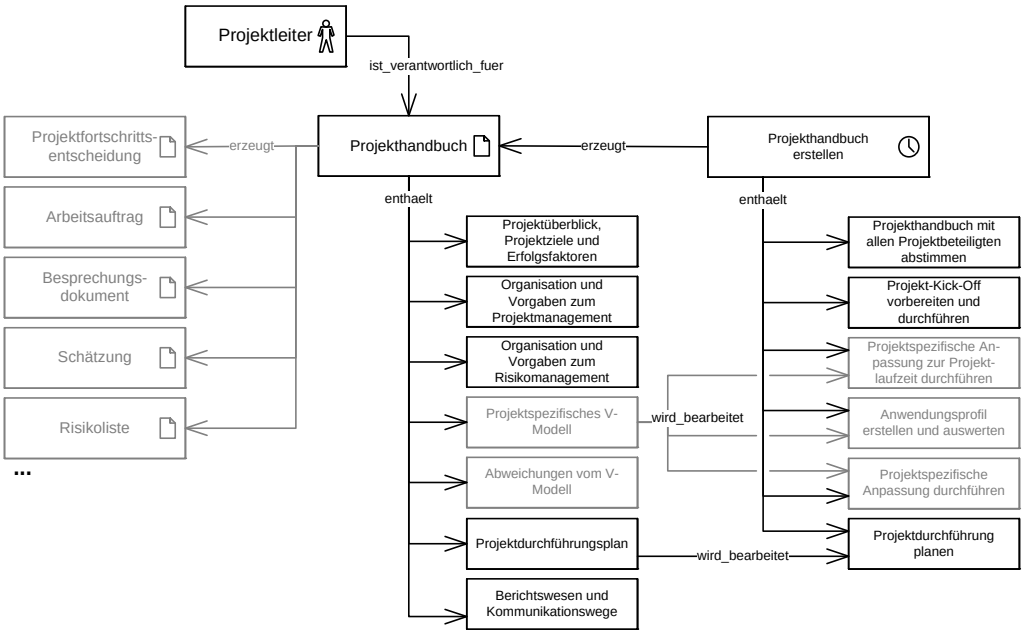


Abbildung 6.5.: Ausschnitt der Prozesskomponente Projektmanagement mit Detaillierung für das Projekthandbuch

Auch in der Verfeinerung ist zu sehen, dass wir mit vergleichsweise wenigen Beziehungstypen auskommen. Das Modell in Abbildung 6.5 zeigt jedoch auch insbesondere in der Verfeinerung der Modellierung des V-Modell-Themas *Projektspezifisches V-Modell*, dass hier weitere fein granulare Beziehungen bestehen – hier zu verschiedenen V-Modell-Teilaktivitäten. Die V-Modell-Elemente modellieren wir hier einmal als *Strukturierten Inhalt* und weiterhin als *Aufgaben*. Wir können dann gemäß der Modelle aus Abbildung 5.8 und Abbildung 5.10 komposite Strukturen bilden.

Wir sehen an dieser Stelle auch noch einmal die Notwendigkeit von Konsistenzbedingungen. Wir haben bisher auf eine Ausdetaillierung verzichtet, da uns die Black-Box Sicht im Wesentlichen genügt. Hier zeigt sich jedoch ein Anwendungsfall, in dem mit der Absicht der automatischen Plangenerierung einige Zusicherungen gemacht werden müssen. Wir betrachten den strukturierten Inhalt *Projektspezifisches V-Modell* und die assoziierten Aufgaben:

```
// Prozesselemente
Dokumente = Projekthandbuch,...
Aktivitaeten = Projekthandbuch erstellen,...
StrukturierterInhalt = Projektspezifisches V-Modell,...
Aufgaben = Projektspezifische Anpassungen zur Projektlaufzeit durchführen,
        Anwendungsprofil erstellen und auswerten,
        Projektspezifische Anpassung durchführen,...
// Beziehungen
erzeugt(Projekthandbuch erstellen, Projekthandbuch);
enthaelt(Projekthandbuch, Projektspezifisches V-Modell);
enthaelt(Projekthandbuch erstellen,
        {Projspezifische Anpassungen zur Projektlaufzeit durchführen,
         Anwendungsprofil erstellen und auswerten,
```

```
         Projektspezifische Anpassung durchführen});
wird_bearbeitet(Projektspezifisches V-Modell,
        {Projspezifische Anpassungen zur Projektlaufzeit durchführen,
         Anwendungsprofil erstellen und auswerten,
         Projektspezifische Anpassung durchführen});
```

Die Beziehung *wird_bearbeitet* wird hier zwischen Elementen instanziiert, die jeweils in Containern enthalten sind. Es muss daher (optimalerweise) in einem Werkzeug einen Mechanismus geben, der prüft, ob die Container auch miteinander in Beziehung stehen.

Der Einkäufer

In der Modellierung aus Abbildung 6.4 finden wir die Rolle *Einkäufer* des V-Modell XT nicht wieder. Dies hat den Grund, dass seine Positionierung primär nichts mit einem Bezug zum Projektmanagement zu tun hat, sondern mit Randbedingungen des V-Modell Tailorings. Hintergrund ist, dass die Rolle Einkäufer im V-Modell sowohl vom Auftraggeber- als auch vom Auftragnehmerprojekttyp benötigt wird. Die einzige (stabile) Schnittmenge an Vorgehensbausteinen zwischen diesen Projekttypen sind die des V-Modell Kerns. So wurde der Einkäufer als Designentscheidung dort untergebracht.

Im Kontext dieser Arbeit können wir mit unseren Strukturen eine Prozesskomponente *Einkauf* entwerfen, die neben der Rollendefinition alle weiteren Einkäufer-bezogenen Produkte und Aktiviäten enthält. Wird in einer anderen Prozesskomponente diese Funktionalität benötigt, können wir die Schnittstelle *Einkauf* anfordern, indem wir analog zu Abschnitt 5.1.3 folgende Anforderung für eine Prozesskomponente pk_0 formulieren:

$$\text{basiert\_auf}\left(pk_0, \{\text{Einkauf}\}, \{\ldots\}\right)$$

6.2.2. Konfiguration eines Vorgehensmodells

Analog zur Prozesskomponente *Projektmanagement* entwerfen wir die Prozesskomponenten *Konfigurationsmanagement*. Abbildung 6.6 zeigt die entsprechende Modellierung. Es fällt hier bereits auf, dass *Konfigurationsmanagement* Elemente aus anderen Prozesskomponenten nachnutzt.

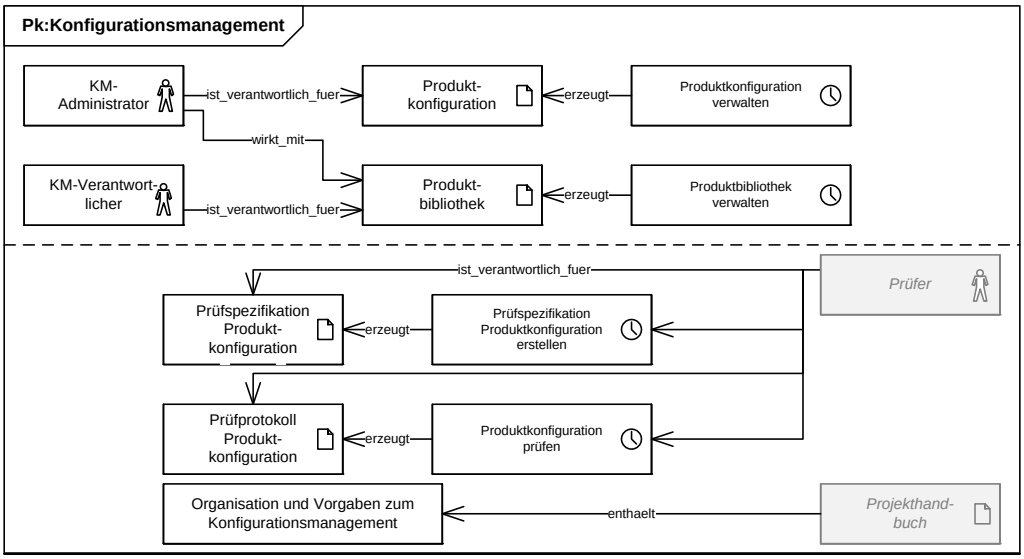


Abbildung 6.6.: Vorgehensbaustein Konfigurationsmanagement des V-Modell XT als Prozesskomponente modelliert

Konkret betrifft dies die Rolle *Prüfer*[1] und das Produkt *Projekthandbuch*. Für diese Elemente werden im Kontext der Erstellung und Verknüpfung der Prozesskomponenten Erweiterungsoperationen (vgl. Abschnitt 5.1.4) angewendet. Allein aus diesen Überlegungen heraus sind zwei Prozesskomponentenabhängigkeiten abzuleiten. Wir stellen dies in Abbildung 6.7 grafisch dar und erläutern die Zusammenhänge.

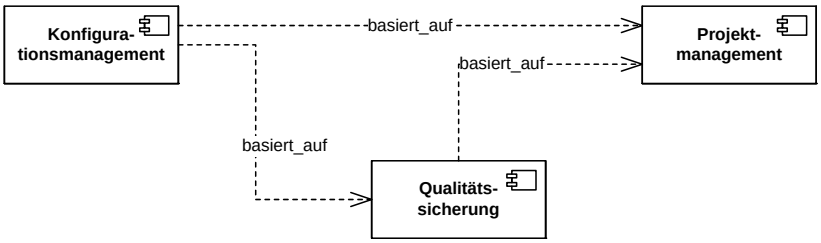


Abbildung 6.7.: Prozesskomponenten mit modellierten Abhängigkeiten

Die Konfiguration in Abbildung 6.7 modellieren wir wie folgt:

```
// Prozesskomponenten
Prozesskomponenten = Projektmanagement (PM), Qualitätssicherung (QS),
        Konfigurationsmanagement (KM)
// Prozesskomponentenabhängigkeiten
basiert_auf(QS, PM, ...); // hier nicht relevant...
basiert_auf(KM, PM, ...);
basiert_auf(KM, QS, ...); // vereinfacht: basiert_auf(KM, {PM, QS});
```

Dies macht deutlich, dass dadurch die in den beteiligten Prozesskomponenten enthaltenen Prozesselemente Beziehungen zueinander eingehen können[2]. Da nun alle Prozesselemente (egal welchen Typs) zur Verfügung stehen, können mithilfe der Spezialisierungen der Prozesselementabhängigkeiten neue Beziehungen hergestellt werden. Im Kontext von Abbildung 6.6 bedeutet dies:

```
basiert_auf(KM, PM, {enthaelt(Projekthandbuch,
        Organisation und Vorgaben zum Konfigurationsmanagement)});
basiert_auf(KM, QS, {ist_verantwortlich_fuer(Prüfer,
        {Prüfspezifikation Produktkonfiguration,
         Prüfprotokoll Produktkonfiguration})});
```

Die Abbildung 6.8 zeigt dies noch einmal am Beispiel eines UML Instanzendiagramms. Hervorgehoben sind hier einmal die Prozesskomponentenabhängigkeit mit ihrer Konfiguration, die wiederum die Prozesselementabhängigkeiten enthält. Die Prozesskomponentenabhängigkeit koppelt somit drei Prozesskomponenten. Da die Konfiguration des Abhängigkeitsgeflechts vollständig außerhalb der Prozesskomponenten liegt, sind diese vergleichsweise stabil. Prozesskomponenten können sich auf der einen Seite ändern, sofern ihre Schnittstelle sich nicht ändert. Als einzige Ausnahme ist in diesem Szenario die additive Änderung der Schnittstelle im Rahmen einer neuen Version zugelassen (vgl. Kapitel 4.3.2). Auf der anderen Seite, können pro Prozesskomponente mehrere Konfigurationen vorliegen, womit die Plattformidee grundlegend unterstützt wird. Prozesskomponenten sind somit nicht zwangsweise an Konfigurationen gebunden. Wie im letzten Abschnitt gezeigt, können sie jedoch vorkonfiguriert werden, um beispielsweise notwendige Abhängigkeiten zu anderen Prozesskomponenten zu beschreiben.

6.2.3. Organisationsspezifische Anpassung

Als Beispiel für die organisationsspezifische Anpassung greifen wir auf das V-Modell Bw zurück (vgl. Anhang A.1). Das V-Modell Bw ist eine echte Variante des Standard V-Modells. Einerseits fügt es neue Inhalte hinzu. Andererseits führt es auch Variationen

1 Die Rolle Prüfer ist im V-Modell im Vorgehensbaustein *Qualitätssicherung* definiert. Wir nehmen diesen hier ebenfalls als verfügbare Prozesskomponente an.

2 Im Kontext einer UML-basierten Modellierung entspricht dies einer *merge*-Assoziation auf Paketebene.

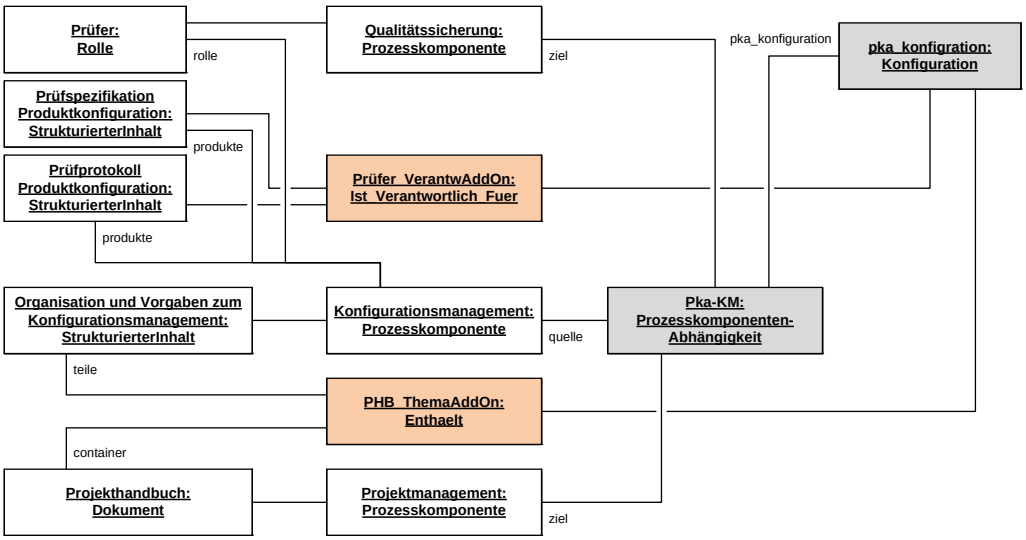


Abbildung 6.8.: Prozesskomponenten mit modellierten Abhängigkeiten als Instanz

ein. Wie im Anhang A beschrieben, sind gerade Variationen im Standard V-Modell problematisch gewesen und waren entscheidende Gründe für das Redesign des Metamodells (Anhang A.3). Wir greifen diesen Teilaspekt auf und diskutieren die Anpassung zum V-Modell Bw.

Gegenstand. In Abbildung 6.9 ist der Vorgehensbaustein *Bedarfsermittlung und Bedarfsdeckung in der Bundeswehr (CPM)* zu sehen, der die wesentlichen inhaltlichen Bestandteile des Bundeswehr-Modells enthält. Der Vorgehensbaustein definiert fünf neue Produkte (inklusive der passenden Aktivitäten) und drei Rollen.

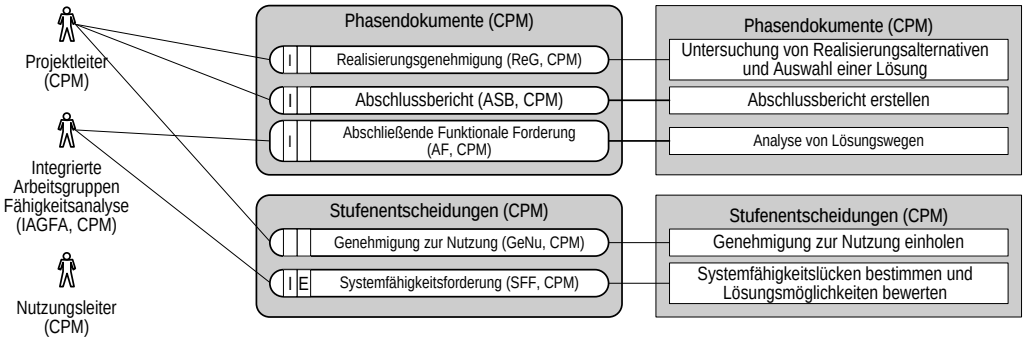


Abbildung 6.9.: Vorgehensbaustein für den V-Modell Bw Anteil

Im V-Modell Bw sind darüber hinaus noch neue Projektdurchführungsstrategien und neue Entscheidungspunkte definiert, die das Vorgehen und den Projektaufbau am etablierten CPM-Rahmenwerk[3] der Bundeswehr ausrichten. Das V-Modell Bw setzt im V-Modell XT direkt auf dem Vorgehensbaustein *Projektmanagement* auf und erweitert den Projekttyp *Systementwicklungsprojekt (AG)*. Um dies besser einordnen zu können, verwenden wir an dieser Stelle bereits die IMV-Komponentensicht und erhalten damit das in Abbildung 6.10 gezeigte Abhängigkeitsgeflecht.

3 CPM: Customer Product Management; Managementverfahren für die Entwicklung und Beschaffung im Geschäftsbereich des Verteidigungsministeriums (BMVg). Die Verfahrensanweisung ist Online unter: `http://www.bwb.org` verfügbar.

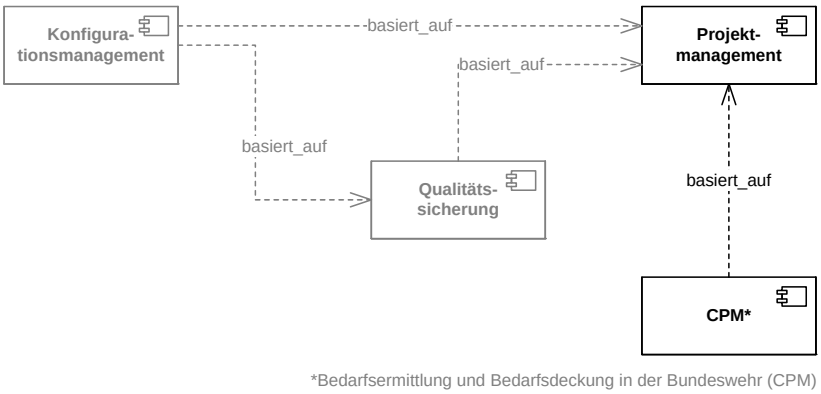


Abbildung 6.10.: System von Prozesskomponenten für das Anwendungsbeispiel

Adressiertes Problem. Es gibt verschiedene Herausforderungen, die im V-Modell Bw zu lösen sind. Viele davon lassen sich strukturell sehr einfach durch eine der in den Kapiteln 4.3.3 und 5.1.4 besprochenen Variabilitätsoperationen lösen. Ein Problem bleibt jedoch bestehen: Die phasenbezogene Rollenzuordnung. Abbildung 6.11 skizziert dieses Problem. Im V-Modell Bw sind Spezialisierungen der V-Modell XT Rolle *Projektleiter* vorzunehmen, da diese Rolle im Geschäftsbereich der Bundeswehr andere Aufgaben und Befugnisse hat. Daher wird zunächst eine erweiternde (alternativ eine ersetzende, siehe Andeutung in der Abbildung) Anpassung durch eine neue Rolle *Projektleiter (CPM)* definiert.

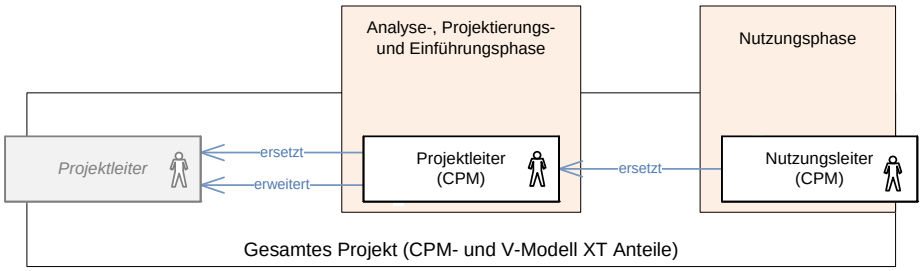


Abbildung 6.11.: Problem: Zeitabhängige Rollenzuordnung

Dem CPM liegt ein Phasenmodell zugrunde, das sich aus den Phasen: *Analyse-*, *Projektierungs-*, *Einführungs-* und *Nutzungsphase* zusammensetzt, wobei diese Abfolge der Risikominimierung dient (eine separate Projekttierungsphase muss nur durchlaufen werden, wenn hohe Risiken während der Analyse identifiziert werden). Für jede dieser Phasen gibt es einen Verantwortlichen, der im Sinne des V-Modells auch Verantwortung für zu erstellende Produkte übernehmen muss. Gemäß Abbildung 6.11 ist das über die Erweiterung des V-Modell Projektleiters realisierbar. In der Nutzung wird jedoch der *Projektleiter (CPM)* durch den *Nutzungsleiter (CPM)* abgelöst. Dieser stellt ebenfalls eine Erweiterung des V-Modell Projektleiters dar, übernimmt jedoch in der Nutzungsphase – und nur dort – alle Aufgaben und Verantwortungen des *Projektleiters (CPM)*. Wie die Abbildung 6.9 aus dem V-Modell Bw von 2006 zeigt, ist gerade diese Verantwortung im V-Modell *nicht* modellierbar gewesen.

Auch mit dem in Abbildung 6.11 gezeigten Teil des Lösungsansatzes ist dieses Problem noch nicht abschließend gelöst. In diesem Kapitel geben wir jedoch einige Ideen an. Die konkrete Umsetzung im Kontext der aktuellen V-Modell XT Entwicklung unterscheidet sich in einigen Aspekten, grundlegend steht hier jedoch dieselbe Idee einer Tailoring-abhängig angewendeten Variabilitätsoperation.

Prozesselemente und Prozesskomponenten definieren

Wir gehen (analog zu Kapitel 6.2.1) schrittweise vor und geben zuerst die Modellierung der Prozesselemente für das V-Modell Bw an.

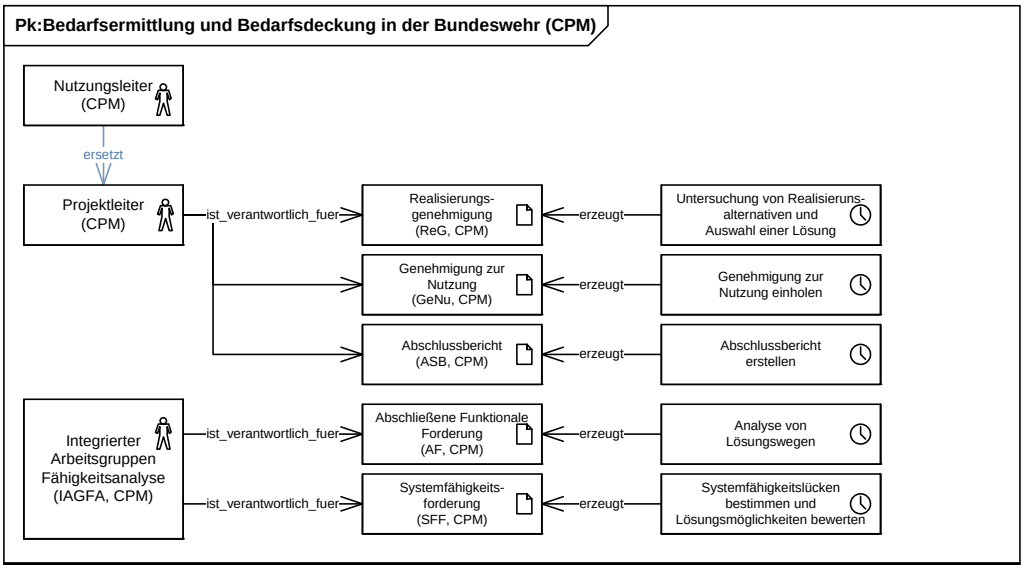


Abbildung 6.12.: Modellierte Prozesskomponente für den V-Modell Bw Anteil

Abbildung 6.12 zeigt die erste Verfeinerungsstufe, in der wir Rollen, Dokumente und Aktivitäten zusammenfassen. Wichtig ist dabei, dass wir zwar wissen, dass *Projektleiter (CPM)* und *Nutzungsleiter (CPM)* untereinander Abhängigkeiten haben, diese jedoch noch offen lassen. Die Beziehung ist daher in Abbildung 6.12 nur angedeutet. Wir gehen nun auf die Konfiguration ein, die die Beziehungen herstellt, wie sie in Abbildung 6.10 als Ziel aufgeführt ist. Dort sehen wir, dass die Prozesskomponente *CPM* mit der Prozesskomponente *Projektmanagement* (Abbildung 6.4) in einer *basiert_auf*-Beziehung stehen muss. Uns interessieren dabei im Wesentlichen die Rolle *Projektleiter* (im Sinne des V-Modells) und seine Verantwortlichkeiten. Diese gehen im Rahmen der Variantenbildung vollständig oder in Teilen auf den *Projektleiter (CPM)* über.

Beziehungen herstellen. Wir stellen zuerst die Beziehungen zwischen den Prozesskomponenten her. Wir orientieren uns dabei wieder am Beispiel aus Kapitel 6.2.2. Wir etablieren also eine Prozesskomponentenabhängigkeit vom Typ *basiert_auf* und geben darin entsprechend die notwendigen Verbindungen zum Projektmanagement an:

```
// Prozesskomponenten
Prozesskomponenten = Projektmanagement (PM), Bedarfsdeckung... (CPM),...
// Prozesskomponentenabhängigkeiten im V-Modell
basiert_auf(QS, PM, ...); // hier nicht relevant...
basiert_auf(KM, PM, ...);
basiert_auf(KM, QS, ...);
// Einbindung der Prozesskomponente CPM
basiert_auf(CPM, PM, ...);
```

Im aktuellen Stand sind keine Abhängigkeiten im Sinne struktureller Abhängigkeiten zum V-Modell Projektmanagement herzustellen. Auf der Ebene der Prozesskomponenten, auf der wir uns nun bewegen, sind daher nur die Rollen interessant. Primär interessiert uns dabei der *Projektleiter (CPM)*. Hier wählen wir folgenden Weg: Zunächst tragen wir der Anforderung des CPM Rechnung, dass der *Projektleiter (CPM)* alle Aufgaben des V-Modell XT-Projektleiters übernimmt. Wir wenden daher die `ersetzt`-Variabilitätsoperration an:

```
// Einbindung der Prozesskomponente CPM
// Schritt: 1
```

```
basiert_auf(CPM, PM,
        {ersetzt(Projektleiter (CPM), Projektleiter)});
```

Dies hat zur Folge, dass alle ein- und ausgehenden Kanten vom Prozesselement *Projektleiter* zum *Projektleiter (CPM)* umgehängt werden. Der den *Projektleiter* repräsentierende Knoten ist zwar noch im Graph enthalten, aber isoliert. Weiterhin ist es aber nicht zielführend, den V-Modell XT-Projektleiter ersatzlos zu streichen, wie verschiedene Feedbackrunden bei der Bundeswehr zeigten. Der *Projektleiter (CPM)* muss also nicht alle Aufgaben des V-Modell XT-Projektleiters wahrnehmen, was auf unser *IMV*-Model übersetzt heißt, dass nicht alle Kanten umgelenkt werden müssen. Wir nehmen also punktuell eine Korrektur der grobgranularen Variabilitätsoperation vor, indem wir selektiv einige Kanten wieder „zurück geben":

```
// Einbindung der Prozesskomponente CPM
// Schritt: 2
basiert_auf(CPM, PM,
        {ersetzt(Projektleiter (CPM), Projektleiter),
         ordnetZu(Projektleiter, Projektleiter (CPM), {<Abhängigkeiten>})});
```

Die Variabilitätsoperation `ordnetZu` übernimmt dies und korrigiert somit das Verhalten von `ersetzt`. Durch diese Kombination der Variabilitätsoperationen ist realisiert, dass die beiden Rollen *Projektleiter* (nach V-Modell XT) und *Projektleiter (CPM)* gleichzeitig und konfliktfrei im resultierenden V-Modell Bw enthalten sind.

Variation von Projektleiter und Nutzungsleiter. Durch die geschickte Konfiguration der beiden Projektleiterrollen von V-Modell und CPM, ist nun die V-Modell Bw-interne Variation der Rolle *Projektleiter (CPM)* durch die Rolle *Nutzungsleiter (CPM)* sehr einfach. Entscheidend ist hier die zeitliche Komponente, welche die Phasenzuordnung vornimmt. Die Phasenzuordnung können wir mithilfe der Prozesskomponenten nur mit Hilfskonstruktionen ausdrücken, indem wir zum Beispiel für jede Phase eine eigene Prozesskomponente bereitstellen. Dies ist jedoch nicht zielführend. Die Ersetzung des *Projektleiter (CPM)* muss daher durch ein Konstrukt vorgenommen werden, das Projektphasen beschreiben kann. Ein solches Konstrukt wird üblicherweise durch ein Planungs- beziehungsweise Ablaufmodell gegeben. Im Anhang B beschreiben wir mit der *Planungskomponente* ein solches Konstrukt auf der Basis von [Gna05].

Planungskomponenten werden analog zu Prozesskomponenten vorgefertigt und konfiguriert. Verwendet werden auch hier Konkretisierungen der Prozesselementabhängigkeiten, die analog in einer *Planungsabhängikeit* (analog zur Prozesskomponentenabhängigkeit) zusammengefasst werden. Eine Planungsabhängigkeit setzt eine Planungskomponente und mindestens eine Prozesskomponente miteinander in Beziehung:

```
// Verbindung von Planungs- und Prozesskomponenten, Schritt: 1
// Planungskomponente: Nutzungsphase (NPh)
// Prozesskomponenten: PM, CPM, ...
planungsAbhaenigkeit(NPh, CPM, {...});
```

Die Planungsabhängigkeit kann nun mithilfe aller im Vorgehensmetamodell definierten Prozesselementabhängigkeiten Verknüpfungen herstellen. Dies können zum Beispiel Abhängigkeiten zwischen Planungselementen und Ergebnissen oder Vorgängen sein. Aber auch nicht planungsorientierte Abhängigkeiten können gesetzt werden. Dementsprechend führen wir an dieser Stelle auch die Variation durch und ersetzen den *Projektleiter (CPM)* durch den *Nutzungsleiter (CPM)*:

```
// Verbindung von Planungs- und Prozesskomponenten, Schritt: 2
planungsAbhaenigkeit(NPh, CPM,
        {ersetzt(Nutzungsleiter (CPM), Projektleiter (CPM))});
```

Dies hat zur Folge, dass bei Einplanung der Nutzungsphase der *Projektleiter (CPM)*, der durch die weiter oben stehende Konfiguration korrekt in die V-Modell-Struktur eingebunden wurde, durch den *Nutzungsleiter (CPM)* ersetzt wird. Die phasenabhängige Variation wird hier durch die entsprechende Planungskomponente vorgenommen.

Vorgehensmodellvarianten erstellen

Wir geben nun basierend auf dem Lebenszyklusmodell (Kapitel 5.2) den Vorgang der Variantenerstellung noch einmal aus einer übergeordneten Perspektive an. Abbildung 6.13 vollzieht unsere Arbeitsschritte der letzten Abschnitte nach. Wir bilden die V-Modell Bw Variante ausgehend von einer Plattform. Diese enthält alle Prozesskomponenten (ggf. schon vorkonfiguriert im Rahmen eines Referenzmodells) sowie weitere Elemente, die im Rahmen eines konkreten Vorgehensmetamodells möglich sind. Beispielhaft haben wir hier Planungskomponenten aufgeführt. Auf der Plattform werden alle enthaltenen Elemente zentral verwaltet und gepflegt. Die dazu notwendigen Aufgaben variieren je nach konkreter Ausgestaltung des Metamodells. Sie sind generisch im Prozess *Vorgehensmodellpflege und -bereitstellung* (Kapitel 5.2.4) beschrieben.

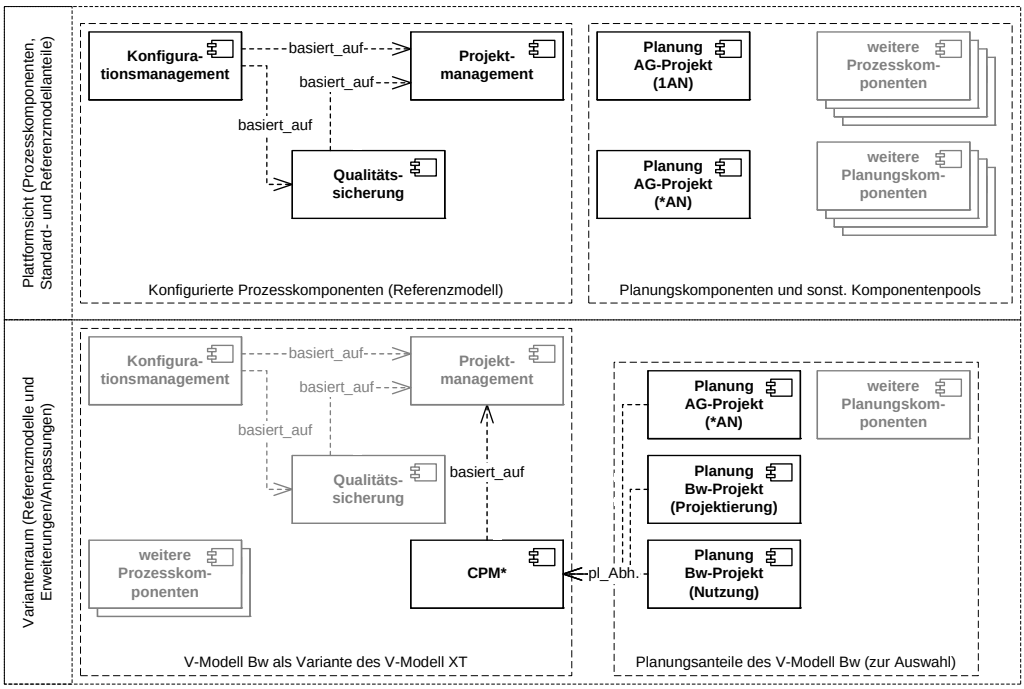


Abbildung 6.13.: Variantenbildung am Beispiel des V-Modell Bw

Aus dieser Plattform leiten wir eine Variante für das V-Modell Bw ab, indem wir das Referenzmodell als Grundlage nehmen und entsprechende Anpassungen und Erweiterungen vornehmen. Konkret werden nun eine neue Prozesskomponente und neue Planungskomponenten erstellt und soweit möglich konfiguriert. Der Prozess *Vorgehensmodellentwicklung* (Kapitel 5.2.3) beschreibt die dazu erforderlichen Schritte, Ein- und Ausgaben. Die Variantenbildung liefert als Ergebnis eine *Konfiguration* für das V-Modell Bw. Die neu erstellten Elemente sind wieder der Plattform und somit dem zentralen Pflege- und Bereitstellungsprozess zuzuführen. Mit der Ableitung der Variante ist eine *organisationsspezifische Anpassung* (Tailoring Stufe 1) des Referenzmodells erfolgt. Aufgrund der Nachnutzung der Komponenten des Referenzmodells und deren physischer Eigenständigkeit, kann nun auch die Variante V-Modell Bw von Aktualisierungen profitieren. Da Prozesskomponenten und Konfigurationen weitgehend entkoppelt sind, werden Änderungen transparent weitergereicht, sobald die Variante erneut bereitgestellt werden muss. Auf bereits instanziierte Varianten hat das zunächst Auswirkungen.

Sobald das organisationsspezifische Vorgehensmodell vorliegt, können daraus noch verschiedene Projektmodelle abgeleitet werden (Tailoring Stufe 2). Die Abbildung 6.14 zeigt dies für die weiter oben diskutierten Fälle des V-Modell Bw inner- und außerhalb der Nutzungsphase. Weiterhin wird die phasenabhängige Konfiguration des Projektmodells gezeigt.

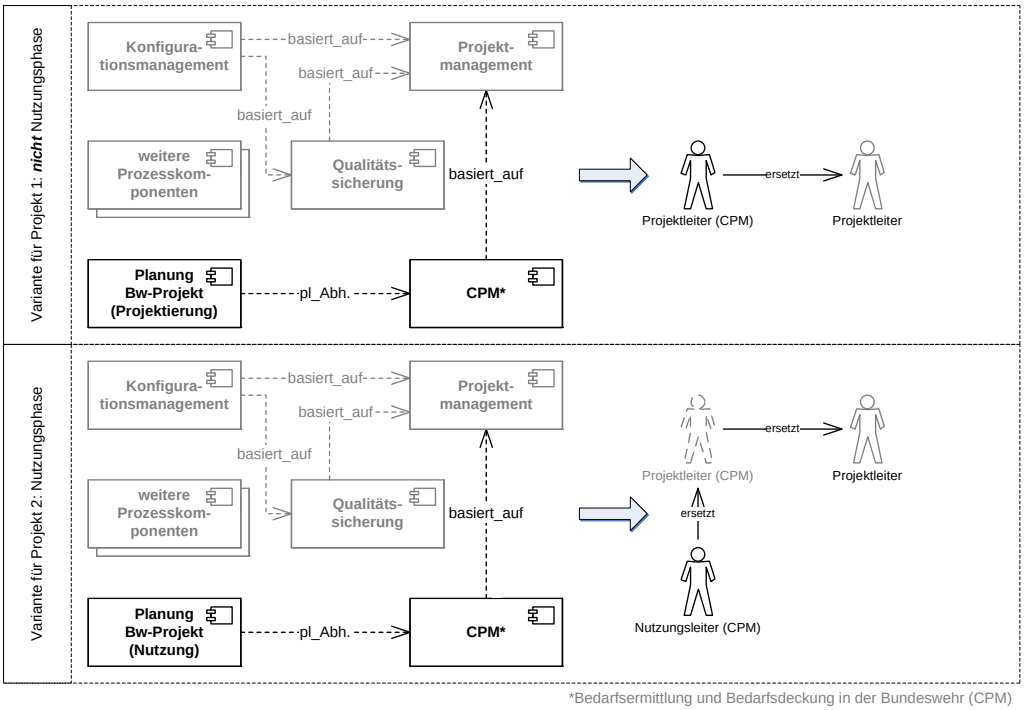


Abbildung 6.14.: Ableitung von Projektmodellen aus der V-Modell Bw Variante

6.3. Projektspezifische Anpassung

In Abbildung 6.14 sind bereits zwei konkrete Vorgehensmodelle zu sehen, die durch eine entsprechende *projektspezifische* Anpassung erzeugt wurden. Diese Anpassung im Kontext eines konkreten Projekts bezeichnen wir auch als *Tailoring der Stufe 2*. Das V-Modell XT unterstützt diese Stufe vollständig durch ein Werkzeug (vgl. Abschnitt 2.1.1). Die weitreichende Unterstützung ist deshalb möglich, weil es ein entsprechendes Anpassungskonzept für das V-Modell gibt. Mit dem *IMV*-Ansatz haben wir auch ein solches Anpassungskonzept verfügbar. In weiten Teilen lässt sich das *Konfigurationsmodell* über die organisationsspezifische Anpassung hinaus auch für projektspezifische Anpassung verwenden. Obwohl dies nicht im Fokus der vorliegenden Arbeit steht, geben wir einen kompakten Einblick in die projektspezifische Anpassung.

6.3.1. Parametrisierung

Eine wesentliche Methode zur projektspezifischen Anpassung ist die Parametrisierung eines Vorgehensmodells. Wir geben sofort ein konkretes Beispiel aus dem Tailoring-Konzept des V-Modell XT an, das diese Methode im V-Modell XT Projektassistenten umsetzt.

Projektmerkmale

Das V-Modell enthält das Konzept *Projektmerkmal*. Dieses wiederum enthält verschiedene Werte. Die Projektmerkmalswerte sind dabei Entscheidungskriterien für die An- und Abwahl von V-Modell-Anteilen im Rahmen des Tailorings. So legt beispielsweise das Projektmerkmal *Safety und Security* den Wertebereich *Ja*, *Nein* vor. Die Werte entscheiden dabei, ob der Vorgehensbaustein *Systemsicherheit* im projektspezifischen Vorgehen enthalten ist, oder nicht. Das V-Modell definiert mehrere Projektmerkmale mit verschiedenen Wertebereichen. Durch sie wird der mögliche Variantenraum für die Ableitung von projektspezifischen Vorgehen aus dem Standard V-Modell XT festgelegt.

Ein Projektmerkmal ist somit ein Parameter, der die Konfiguration eines Vorgehensmodells beeinflusst. In diesem Kontext betrachten wir auch unseren Ansatz. Für das Szenario in Abbildung 6.14 müssten wir daher einen Parameter *Phase* einführen, der als Wertebereich die Phasen des CPM abdeckt. In Abhängigkeit von der gewählten Phase können dann exklusive Teile der Konfiguration, wie zum Beispiel die Rollenzuordnung, ausgewählt werden, während andere Teile abgewählt werden. Wird also beispielsweise der Parameter *Phase = Nutzung* gesetzt, so hat dies zur Folge, dass die Planungsabhängigkeit für die Planungskomponente (Nutzung) ausgewählt wird, womit automatisch der *Projektleiter (CPM)* durch den *Nutzungsleiter (CPM)* ersetzt wird.

Die Parametrisierung erfordert passende Werkzeuge in der Art des V-Modell XT Projektassistenten. Grundlegende Eigenschaften sind bereits durch das in Kapitel 5.3 skizzierte Werkzeugkonzept beschrieben. Jedoch ist durch das dort beschriebene Konzept nur die Arbeit mit Vorgehensmodellen bis zur organisationsspezifischen Anpassung erfasst.

6.3.2. Skalierung

Die Skalierung eines Vorgehensmodells adressiert verschiedene Skalierungsgrößen, wie zum Beispiel Projektvolumen, Laufzeit oder Kritikalität. Ähnlich zur Parametrisierung sind auf der Grundlage eines gegebenen Vorgehensmodells mehrere projektspezifische Konfigurationen möglich. Skalierung und Parametrisierung unterscheiden sich hinsichtlich der Art der resultierenden Konfigurationen. Während die Parametrisierung in der Regel Einzelaspekte adressiert, wie zum Beispiel der oben diskutierte Rollenaustausch, zielen Skalierungen eines Vorgehensmodells auf konsistente Gesamtmodelle.

In [Kuh06] haben wir die Skalierung des V-Modell XT anhand typischer Anforderungen für kleine Projekt geprüft. Im *IMV*-Kontext stellt sich die Skalierung eines Vorgehensmodells in einer Variation des resultierenden Abhängigkeitsgraphen nach der Zusammenstellung der Vorgehensmodellvariante dar. Wir müssen im Rahmen einer projektspezifischen Anpassung zwei Ansatzpunkte berücksichtigen, um die entsprechenden Variationen im Rahmen der Ableitung eines Projektmodells verfügbar zu haben:

1. *Planung von skalierten Varianten* – Dieses Verfahren ist konstruktiv. Es umfasst die Definition von *Projektmodellvarianten*, die mithilfe von Parametern die Skalierungsstufen abbilden. Die Skalierung im Rahmen der projektspezifsichen Anpassung besteht somit im Wesentlichen in der Auswahl der entsprechenden Projektmodellvariante. Diese kann dann aber noch über möglicherweise definierte Parameter noch weiter angepasst werden[4].
2. *Änderung des Abhängigkeitsgraphen* – Dieses Verfahren setzt ein vollständig angepasstes Projektmodell voraus. Auf dieses muss dann eine Operation angewendet werden, die den vorliegenden Abhängigkeitsgraph mithilfe eines Transformationsgraphen entsprechend der Anforderungen des Projektes direkt verändert. Dieses Verfahren ist einfacher aus Sicht der Vorgehensmodellbereitstellung. Der Aufwand liegt jedoch in der Erstellung des Transformationsgraphen. Dieser eignet sich nicht für die Vorfertigung, da er sehr eng mit dem Abhängigkeitsgraphen verwoben ist.

Analog zur Parametrisierung erfordert auch die Skalierung eine passende Werkzeugunterstützung. Insbesondere das erste Verfahren mit der Vorfertigung der skalierten Varianten eignet sich dafür sehr gut. Es weist eine große Nähe zur Parametrisierung auf und ist mit ihr auch kombinierbar. Insbesondere eignet sich das in Kapitel 5.3 vorgestellte Werkzeugkonzept hierfür, da die Erstellung der Projektmodellvarianten mit den Mitteln der in dieser Arbeit entwickelten Mechanismen zur Vorgehensmodellkonfiguration erfolgen kann.

4 Ein ähnliches Konzept findet sich im neuen V-Modell XT Metamodell (Anhang A.3). Dieses ist eine Konsequenz der Konzentration auf die Konfigurations- und Variabilitätseigenschaften des V-Modells

Zusammenfassung

Im Abschnitt 6.2 haben wir die Anwendung des *integrierten Modellierungsansatzes für Vorgehensmodelle* (IMV) gezeigt. Anhand von Teilen des V-Modell XT haben wir die Modellierung von Prozesselementen und Prozesskomponenten gezeigt. Ferner haben wir die Anwendung der verschiedenen Abhängigkeiten für die Erstellung von Prozesskomponenten und die Konfiguration zu einem Vorgehensmodell behandelt.

Die Mächtigkeit von IMV haben wir am Beispiel der Variantenbildung diskutiert. Wir haben auf Basis des Standard V-Modell XT und des V-Modell Bw die Problematik phasenabhängiger Variationen im Vorgehensmodell beschrieben. Mithilfe unseres Ansatzes konnten wir durch Anwendung Konfigurationsmechanismen und der Variabilitätsoperationen die Variantenbildung vollständig beschreiben. Über die einfache, grob granulare Variation hinaus, haben wir weiterhin die Option fein granularer Anpassungen diskutiert.

Zusammenfassend haben wir noch einmal die Anwendung mit dem V-Modell XT als Beispiel im Gesamtkontext positioniert. Wir haben dabei die allgemeine Ableitung von Varianten auf eine Plattform beschrieben. Durch die Plattform stehen eine Menge Prozesskomponenten und weitere Elemente des Vorgehensmodells zur Verfügung. Die Plattform selbst enthält bereits vorkonfigurierte Prozesskomponenten sowie ein *Referenzmodell*. Auf der Basis des Referenzmodells sowie weiterer, zusätzlich verfügbarer Elemente (Prozess-, Planungskomponenten und sonstige) wird ein Variantenraum aufgespannt. Für das Beispiel V-Modell Bw haben wir dies gezeigt. Das V-Modell Bw ist eine Variante des V-Modell XT, die durch eine organisationsspezifische Anpassung entstanden ist. In abgeleiteten Varianten gibt es ggf. noch weiteren Anpassungsbedarf, wie wir gerade erläutert haben.

Mit dem Beispiel in diesem Kapitel haben wir die Anwendbarkeit des IMV-Ansatzes für die Vorgehensmodellkonstruktion auf der Plattformebene und die Fähigkeit zur Variantenbildung gezeigt. Im Abschnitt 6.3 haben wir die Teilaspekte Parametrisierung und Skalierung von Vorgehensmodellvarianten im Kontext projektspezifischer Anpassungen diskutiert. Die Parametrisierung ist von uns von Interesse, da sie notwendig ist, um die phasenabhängige Variation des Vorgehensmodells zu realisieren. Wir haben die Einführung eines Parametersystems am Beispiel der Rollenersetzung im V-Modell Bw motiviert. Neben der Parametrisierung haben wir noch die Skalierung in die Diskussion mit aufgenommen. Skalierung und Parametrisierung sind dicht zusammenhängend. Wir haben ein Verfahren motiviert, in dem Projektmodellvarianten, also Varianten einer Variante, mit den Mitteln des IMV-Ansatzes für die projektspezifische Anpassung vorgefertigt werden können. Die Integrationsmöglichkeit in das im Kapitel 5.3 entwickelte Werkzeugkonzept stellt hier eine direkte Verbindung zwischen Organsations- und Projektmodell in Aussicht.

7. Zusammenfassung und Ergebnisse

Der Optimierung von Prozessen wird heute im Allgemeinen das größte Potenzial bei der Herstellung der wirtschaftlichen Konkurrenzfähigkeit deutscher Unternehmen zugesprochen. Optimierte Prozesse helfen an vielen Stellen, Kosten zu sparen, indem zum Beispiel unnötige, redundante Aufgaben oder Wartezeiten durch fehlende Synchronisation vermieden werden. Weiterhin stellen definierte Prozesse Möglichkeiten bereit, um Ein- und Ausgabeprodukte einzelner Prozessschritte detailliert zu beschreiben. Damit können Qualitätsanforderung an die erwarteten Ergebnisse definiert werden. Mit qualitativ hochwertigen Prozessen wird versucht, Qualitätsmängeln entgegen zu wirken, womit ein Beitrag zur Vermeidung von zusätzlichen Aufwänden und Kosten zum Beispiel zur Problembehebung geleistet wird. Aus diesem Grund haben bereits viele Unternehmen in Vorgehensmodelle investiert. Sie haben ihre individuellen, organisationsspezifischen Vorgehensmodelle entweder neu oder auf der Basis eines Standardvorgehensmodells entwickelt, beziehungsweise prüfen die Optionen einer Einführung.

Aufgrund der Breite der verfügbaren Vorgehensmodelle für die Software-Entwicklung ist nach der Auswahl und Einführung eines Vorgehensmodells in vielen Bereichen eine Harmonisierung mit verschiedenen anderen Prozessen erforderlich. Die Anpassung/Variantenbildung von Vorgehensmodellen, die in diesem Zusammenhang durchgeführt wird, verläuft jedoch in weiten Bereichen nicht ausreichend strukturiert. Bei der Aktualisierung der Ausgangsprozesse, zum Beispiel durch eine neue Version des Ausgangsprozesses, entstehen hier zwangsläufig Konsistenzprobleme zwischen dem Ausgangsmodell und den eingesetzten Varianten: Beispielhaft sind hier Rechtsvorschriften und Normen zu nennen, auf die sich Vorgehensmodelle abstützen können. Ändert sich beispielsweise eine Rechtsvorschrift, ist eine Aktualisierung eines sich darauf beziehenden Vorgehensmodells ebenfalls nötig. Aus dem betreffenden Vorgehensmodell abgeleitete Varianten werden danach inkonsistent zum Ausgangsmodell und müssen ebenfalls aktualisiert werden – und so weiter. Verschiedene Lösungsansätze aus Forschung und Industrie versuchen die entstehenden Probleme mit ingenieurmäßigen Grundsätzen, wie zum Beispiel Standardisierung, Modularisierung, Plattformbildung und Wiederverwendung, zu lösen.

Kernthema dieser Arbeit ist die *methodische Konstruktion* von Vorgehensmodellen im Kontext von Standardentwicklungsprozessen. Ziel ist es dabei, vor der Erarbeitung oder Anpasssung eines spezifischen Vorgehensmodells anzusetzen. Wir beschreiben dazu ein Konzept zur *konfigurationsbasierten* und *modularen* Entwicklung von Vorgehensmodellen auf der Grundlage einer standardisierten Plattform. Die Plattform enthält einerseits *Strukturkonzepte* in Form eines Vorgehensmetamodells, andererseits *Lebenszykluskonzepte* in Form eines Prozessmodells, das die Entwicklung und Pflege von Vorgehensmodellen beschreibt. Diese Teile führen wir im *integrierten Modellierungsansatz für Vorgehensmodelle* (IMV) zusammen und bieten somit Prozessingenieuren eine methodische Unterstützung für die Erstellung und Pflege von Vorgehensmodellen an.

Ausgangslage

Am Markt sind vielfältige Vorgehensmodelle verfügbar. Die Bandbreite reicht dabei von national standardisierten und für öffentliche Aufträge verbindlich vorgeschriebenen Vorgehensmodellen wie dem V-Modell XT, bis zu firmeninternen, kompakten und somit spezifischen Entwicklungsprozessen. Über die gesamte Bandbreite hinweg sind in der Regel Anpassungsoptionen zu finden. Anpassungsoptionen können verschiedenartig ausgeprägt sein. Sie adressieren wie im Falle des V-Modell XT Ergebnistyporientierte Anpassungen (Tailoring) oder wie zum Beispiel beim Microsoft Solutions Framework die Anpassung und Ergänzung von Mikroprozessanteilen. Eine Zusammenstellung und Analyse ausgewählter Vorgehensmodelle unter dem Gesichtspunkt

der Anpassbarkeit bildet den technischen Grundstock und spiegelt den aktuellen Stand hinsichtlich der Modularität von Vorgehensmodellen wider.

Jedoch sind nicht nur die technischen Möglichkeiten relevant. Wie gerade angedeutet, sind die Anpassungsprozesse der jeweiligen Vorgehensmodelle ebenso wichtig, um Anpassungen zu strukturieren und nachvollziehbar zu machen. Für jedes der untersuchten Vorgehensmodelle lässt sich ein Anpassungsprozess finden und zumindest semi-formal beschreiben. Die auf diesen Prozessen aufbauenden Konzepte unterscheiden sich zum Teil sehr stark. Das Microsoft Solutions Framework bietet sehr fein granulare, physische Anpassungsgegenstände an (Work Items), stellt aber die Konsistenz des resultierenden, integrierten Vorgehens nach einer Anpassung nicht sicher. Das V-Modell XT hingegen strebt immer ein konsistentes, stimmiges, integriertes Vorgehensmodell an, erreicht dies zurzeit jedoch nur durch den Verzicht auf physische Modularisierung.

Eine Verknüpfung dieser beiden Fähigkeiten (fein granulare Änderungsoptionen bei gleichzeitiger Konsistenzsicherung) ist wünschenswert. Vereinzelt sind bereits Bestrebungen zu finden, *Prozess-Plattformen* in Anlehnung an Produklinienansätze aus der Software-Entwicklung zu definieren. Für den Aufbau einer Plattform, in der *Versionen/Releases* und *Varianten* von Vorgehensmodellen erstellt und gepflegt werden können, ist eine *physische Modularisierung* im Sinne einer *Komponentenbildung* erforderlich. Die untersuchten Vorgehensmodelle haben wir nicht nur hinsichtlich der Modularisierung untersucht, sondern auch Abhängigkeitsgeflechte, Kompositions- und Verteilungstechniken analysiert. Ein durchdachtes Verfahren zur Auflösung von Abhängigkeiten stellt ein zentrales Element einer Prozess-Plattform dar. Abhängige oder vorkonfigurierte Komponenten müssen auffindbar sein, um auftretende Änderungen auf Konsistenzverletzungen zu prüfen und somit die weitere Gültigkeit existierender Komponenten und Konfigurationen sicherzustellen. Dies erfordert somit nicht nur eine solide technische Grundlage zur Beschreibung eines Vorgehensmodells, sondern insbesondere im Kontext einer Prozess-Plattform auch die Beschreibung von Prozessen für die Verwaltung und Konfiguration von Komponenten und deren Verwendung zur Ableitung eines Vorgehensmodells. Darüber hinaus muss eine Prozess-Plattform auch die Pflege, die Weiterentwicklung und Variantenbildung adressieren.

Lösungskonzept

Auf den Analysen aufbauend haben wir Anforderungen an Bearbeitungs-, Tailoring- und Verteilungseinheiten modularer Vorgehensmodelle erstellt. Diese Anforderungen richten sich vornehmlich an das *Metamodell*, das der Plattform zugrunde liegt.

Physisch eigenständige Prozesskomponenten. Ein typisches Software-Entwicklungsprojekt fasst mehrere Disziplinen zusammen (Projektmanagement, Entwicklung, Test etc.). Physisch *eigenständige* und *konfigurierbare Prozesskomponenten* sind die Grundlage unseres Metamodells. Sie definieren die Einheiten zur Bearbeitung von Prozessinhalten und ermöglichen die strukturierte Beschreibung von Teilprozessen. Weiterhin stellen sie die Einheiten der Anpassung und der Verteilung dar. Die Konstruktion eines integrierten Vorgehens erfolgt mit unserem Ansatz additiv auf der Basis von Prozesskomponenten, die die Beschreibung der jeweils benötigten Teilprozesse/Teilvorgehen enthalten. Das Konzept ist am Tailoring des neuen V-Modell XT Metamodells orientiert.

Konfigurierte Vorgehensmodelle. Wir stellen einen *konfigurativen* Ansatz vor, in dem Vorgehensmodelle und Varianten anders als beim V-Modell XT nicht bereits vollständig vorliegen. Mit unserem Ansatz wird ein Vorgehensmodell individuell aus einer Menge Prozesskomponenten mittels *Konfiguration* zusammengestellt. Dies erfordert *Plattformeigenschaften* für einen Entwicklungsstandard, sodass Prozesskomponentenbibliotheken erstellt und gepflegt werden müssen.

Für die Konzeption einer solchen *Prozess-Plattform* muss ein Metamodell *Soll-Bruchstellen* definieren, womit eine individuelle Bearbeitung und Pflege einerseits, die flexible Zusammenstellung im *Baukasten-Prinzip* andererseits, ermöglicht wird. Eine Plattform für

Vorgehensmodelle definiert eine Menge von Kompositions- und Konfigurationsmustern. Mit den Prozesskomponenten setzen wir ein solches Konzept für die strukturellen Anteile einer Prozess-Plattform im Rahmen in dieser Arbeit um. In unserer Umsetzung sind Prozesskomponenten an den Vorgehensbausteinen des V-Modell XT angelehnt, bieten darüber hinaus aber die Möglichkeit, individuell erstellt, konfiguriert und verteilt zu werden.

Variantenbildung. Eine Prozess-Plattform muss weiterhin die Möglichkeit bieten, Konfigurationen im Sinne von Vorgehensmodellversionen oder *Releases* und *Varianten* zu erstellen. Um dabei jeweils die Konsistenz sicherzustellen, müssen Abhängigkeiten flexibel aber reproduzierbar verwaltet werden. Mit dem konfigurativen Ansatz dieser Arbeit greifen wir Ideen moderner Komponentensysteme aus der Softwareentwicklung auf und überführen sie auf unser Vorgehensmetamodell. Abhängigkeiten zwischen Prozesskomponenten werden in separaten Konfigurationen festgelegt. Die Konfiguration erfasst dabei neben der Auflösung der Abhängigkeiten auch Möglichkeiten zur fein granularen Anpassung im Rahmen einer *Variantenbildung*. Da wir Vorgehensmodelle und Varianten durch ihre Konfigurationen charakterisieren und diese separat vorhalten, erreichen wir eine hohe Unabhängigkeit der einzelnen Prozesskomponenten. Dies steigert die Wiederverwendbarkeit und ermöglicht kürzere Releasezyklen.

Lebenszyklusmodell. Die Anforderungen an das einer Prozess-Plattform zugrunde liegende Metamodell fokussieren sich somit auf die Bereitstellung *physischer Prozesskomponenten* und eines *definierten Konfigurationskonzepts*, welche wir im Rahmen dieser Arbeit entwickelt haben. Die strukturellen Eigenschaften müssen durch ein *Lebenszyklusmodell* ergänzt werden. Das Lebenszyklusmodell muss die Anforderungen aus der Praxis berücksichtigen, gleichzeitig aber den für Vorgehensmodelle neuen Aspekt der *Linienorientierung* durch die Prozess-Plattform beachten. Die Linienorientierung führen wir einerseits auf den Vorschlag integrierter *Produkt- und Prozesslinien* und weiterhin auf die *Produktlinien* aus der Software-Entwicklung zurück. Mit dieser Arbeit zeigen wir die Anwendung von Vorgehensmodellen im Kontext einer Prozess-Plattform. Wir zeigen die Auswahl von vorhandenen Prozesskomponenten und deren Konfiguration in Vorgehensmodellvarianten. Dies führen wir sowohl vom Standpunkt der Strukturen des Metamodells aus, als auch durch die Prozesse, die wir im Kontext des Lebenszyklusmodells beschrieben haben.

IMV. Die Strukturen des Metamodells, die plattformorientierten Prozesse des Lebenszyklusmodells sowie ein Werkzeugkonzept bilden den *integrierten Modellierungsansatz für Vorgehensmodelle* (IMV). Die strukturellen Anteile sind in Form eines Metamodells in UML modelliert. Dieses basiert auf einer einfachen graphenbasierten Theorie. Sie führt die Unterscheidung in *Strukturkompositions-* und *Abhängigkeitsgraphen* ein. Die Theorie beschreibt weiterhin die verschiedenen Verknüpfungsmöglichkeiten für die verwendeten Graphen. Auf dieser Theorie aufbauend definieren wir ein UML-Klassensystem, das grundlegende Elemente zur Bereitstellung eines *Prozess-Frameworks* definiert. Im Sinne eines Frameworks modellieren wir bereits beispielhafte Konkretisierungen für Vorgehensmodellstrukturen. Über diese Strukturen hinaus demonstrieren wir in dieser Arbeit auch die Anwendbarkeit für Metamodellerweiterungen. Die Theorie ist somit auch auf andere strukturorientierte Domänen überführbar.

Das Metamodell dient im Lebenszyklusmodell dazu, Ein- und Ausgaben von Prozessen zu beschreiben. Im Lebenszyklusmodell sind Erstellungs-, Pflege- und Bereitstellungsprozesse beschrieben. Sie bilden unter anderem auch den methodischen Rahmen für das beschriebene Werkzeugkonzept. Im für diese Arbeit erforderlichen Rahmen sind die vorgegebenen Prozesse durch UML-Aktivitätsbeschreibungen ausmodelliert.

Die praktischen Erfahrungen zeigen ein dichtes Abhängigkeitsgeflecht in Vorgehensmodellen. Die Analysen in dieser Arbeit unterstreichen das noch einmal. Das V-Modell enthält zum Beispiel jeweils über 100 Produkt- und Aktivitätsbeschreibungen. Dazu kommen *hunderte* Themen- und Teilaktivitätsbeschreibungen. Alle Elemente im V-

Modell XT sind miteinander verknüpft, üblicherweise in $n : m$-Beziehungen. Der IMV-Ansatz definiert ein *einfaches* Metamodell, das dem Prozessingenieur als Sprache zur Beschreibung von Vorgehensmodellen dient. Durch den einfachen und homogenen Aufbau des Metamodells, wird es dem Prozessingenieur ermöglicht, Vorgehensmodelle einfacher zu strukturieren. Die klare Trennung von Element- und Abhängigkeitstypen gestattet es, Abhängigkeiten separat von Elementen zu erstellen und zu pflegen. Die Prozesskonstruktion im IMV-Ansatz erfolgt somit flexibel durch die Festlegung von Abhängigkeiten zwischen den Elementen. Der hierarchische Aufbau des Metamodells mit drei definierten Ebenen beschränkt gleichzeitig die Komplexität so, dass weiterhin eigenständige Komponenten erstellt werden können. Die Festlegung von Abhängigkeiten zwischen den Elementen – die Konfiguration – bildet die Übergänge zwischen den Hierarchieebenen.

Dennoch steht Prozessingenieuren mit dem IMV-Ansatz ein hohes Maß an Flexibilität zur Verfügung. Da IMV als *Prozess-Framework* entworfen wurde, stehen alle Konzepte zur Erweiterung und Anpassung zur Verfügung. Anhand vieler Beispiele haben wir dies bereits in dieser Arbeit gezeigt, indem wir Teile des V-Modell XT mithilfe unseres Metamodells modelliert haben. Wir haben dabei sowohl Elementtypen als auch Strukturen berücksichtigt. Erweiterungen, zum Beispiel im Rahmen einer Prozesserstellung oder einer Anpassung, können durch Prozessingenieure ebenfalls vorgenommen werden. Durch die detailliert beschriebenen Basismechanismen von IMV ist in jedem Fall die nahtlose Integration möglich.

Anwendung und Erprobung

IMV enthält viele Konzepte und Ideen. Einige davon, insbesondere die Konfigurations- und Variabilitätskonzepte, sind für eine unmittelbare Überführung in die Praxis geeignet. In der aktuellen Weiterentwicklung des V-Modell XT fanden sowohl konfigurationsbasiertes Tailoring (Stufe 2, projektspezifische Anpassung) als auf das hier entwickelte Konzept der konfigurierbaren Variabilitätsoperation Anwendung. Diese beiden Punkte wurden im Rahmen des WEIT-Projektes (und dort im Task RO) bereits nach einer Anpassung und Abstimmung in das neue V-Modell XT Metamodell integriert. Zum Zeitpunkt der Erstellung dieser Arbeit findet der erste technische Durchstich statt.

Aber auch Ideen aus dem Lebenszyklusmodell haben auf die Weiterentwicklung des V-Modells gewirkt. So findet sich im neuen V-Modell XT Metamodell ein Konzept *Erweiterungsoperation*, das die bislang nur implizit beschriebenen Operationen des V-Modells (zum Beispiel Anlegen eines neuen Vorgehensbausteins) erfasst. Dies ist ein erster Schritt in der Entwicklung des V-Modells, eine vollständige Beschreibung aller erlaubten Operationen auf dem Modell zu erstellen. Eine solche Liste ist insbesondere für die Erstellung *konformer* und *zertifizierbarer* Vorgehensmodellderivate wünschenswert.

Ein weiteres Anwendungsfeld findet sich in der Prozessintegration. Der IMV-Ansatz gestattet es, verschiedene Prozesse und Konzepte miteinander zu integrieren. IMV-Strukturen treten dabei als *Adapter* zwischen den verschiedenen Prozessen auf. Eine frühe, prototypsiche Anwendung findet sich im Rahmen des Projekts *CollabXT*, indem wir die Systematisierung der Abbildung zwischen V-Modell XT und den Microsoft Prozessen und Prozessinfrastrukturen prototypisch vorgenommen haben. Ein praktisches Anwendungsfeld findet sich zum Beispiel bei der Integration von *Nicht-Entwicklungsprozessen* in einen Vorgehensstandard. Denkbar ist beispielsweise die Beschreibung eines Geschäftsprozessmodells auf der Basis von IMV und dessen Integration in ein Vorgehensmodell, um die Geschäftsprozessanalyse in einem Entwicklungsprojekt authentisch durchzuführen. Das Geschäftsprozess-Analysemodell wäre so direkt mit einem Architekturmodell verknüpfbar. Die Erstellung beider Modelle kann entsprechend in der Planung berücksichtigt werden.

Die Prozessintegration weist bereits hohe Analyseanteile auf. Daher positionieren wir den IMV-Ansatz auch als *Analysemodell* für Assessoren. Durch die Möglichkeit, nahezu beliebige Prozessstrukturen zu modellieren, ist IMV im Kontext der Prozesszertifizierung eine wertvolle Hilfe. Da das strukturelle Metamodell von IMV vergleichsweise

einfach gehalten ist, bietet es sich im Rahmen einer Zertifizierung oder Konformitätsfeststellung als Übersetzungmodell an. Der IMV-Ansatz kann somit die Automatisierung von Konformitätsprüfverfahren unterstützen.

Ausblick und offene Forschungsfragen

Die oben aufgeführten Punkte zeigen auf interessante Forschungsfragen, die insbesondere im Bereich der Automatisierung und Anwenderunterstützung liegen.

Automatische Konformitätsfeststellung. Die Feststellung, ob ein individuelles oder angepasstes Vorgehensmodell konform zu einem gegebenen Standard ist, wird heute ausschließlich durch Assessments getätigt. Hierbei kommen oft einfache Mittel wie Excel-Tabellen zum Einsatz, die Abbildungsmatrizen enthalten. Ein Schritt in Richtung Automatisierung wird gerade im V-Modell XT versucht, in dem diese Matrizen auf der Grundlage des V-Modell-Teils, der als Bezugspunkt dienen soll, erstellt werden. Die weitgehende Anwendung konfigurationsbasierter Ansätze verspricht hier eine höhere Automatisierbarkeit. Konfigurationsbasierte Vorgehensmodelle können anhand der Konfigurationsinformationen ausgewertet und eingestuft werden. Insbesondere Variationen von gegebenen Strukturen des Standardvorgehens sind somit nachvollziehbar, da sie explizit modelliert werden. Denkbar sind hier Lösungen auf der Basis von wissensorientierten Systemen wie PROLOG, in denen die Vorgehensmodellkonfigurationen als Wissensbasis verstanden werden können und Regeln über der Wissensbasis die Konformitätsfeststellung erbringen. Weiterhin sollte die Anwendung von Modelltransformationen untersucht werden, da sich das IMV-Metamodell durch seine Einfachheit als Übersetzungsmodell für verschiedene, zu prüfende Prozesse eignet. Prüfkriterien können durch Abbildungsvorschriften in IMV modelliert werden. Abbildungsvorschriften stellen die Verbindung zwischen den zu prüfenden Prozessen her.

Operationalisierung von Vorgehensmodellen. Im Rahmen des Projekts *CollabXT* haben wir bereits erste Schritte in Richtung automatische Werkzeugunterstützung eines Projekts auf der Basis projektspezifischer Vorgehensmodelle untersucht und konkrete Werkzeuge erstellt. Diese Werkzeuge zeigen einerseits das Potenzial und andererseits durch das rege Feedback auch den Bedarf an solchen Lösungen. Die in dieser Arbeit vorgestellten Konfigurationsmechanismen können bei der Einbindung von Werkzeugen wertvolle Daten liefern. So können zum Beispiel Ablaufbeschreibungen zusammengenommen mit Produkten und den assoziierten Rollen für die Generierung von Workflows verwendet werden. Gleichzeitig sind über Abhängigkeiten Analysen in einem Projekt möglich. Abhängigkeiten können ausgewertet und für die Berechnung von Trends verwendet werden. Auch kann die Wechselwirkung zwischen Prozess und Werkzeug optimiert werden. Ist heute in weiten Bereichen ein Update für ein Werkzeug erforderlich, wenn der integrierte Prozess angepasst wird, bietet die Generierung von Arbeitsumgebungen hier Vorteile, da die Werkzeuge eine höhere Stabilität aufweisen können. Dies kann mittelfristig Kosten bei der Werkzeugerstellung einsparen, womit weniger Kosten für Kunden anfallen.

A. Case Study: Anpassung des V-Modell XT

Aus den ursprünglichen Pilotprojekten des V-Modells ging recht schnell die Notwendigkeit der Anpassung an weitere Anwendungsdomänen und Werkzeuge hervor. In diesem Anhang stellen wir mit dem *V-Modell der Bundeswehr* eine zentral erstellte Anpassung des V-Modells vor. Auch in anderen Bereichen war die Anpassung von Vorgehensmodellen ein bestimmender Punkt. Gerade im Umfeld der Microsoft Technologien waren wir in den letzten Monaten und Jahren sehr aktiv. Abschnitt A.2 liefert hierzu Informationen, die im Umfeld des Projekts *CollabXT* entstanden. Die gemachten Erfahrungen verdeutlichten zunehmend auch die Notwendigkeit definierter Anpassungsverfahren, insbesondere im Kontext umfassenderer Vorgehensmodellplattformen. Im Abschnitt A.3 stellen wir hierzu das im Rahmen des *Task RO* für das V-Modell XT entwickelte Konzept vor, in das bereits einige der auch dieser Arbeit zugrunde liegenden Ideen mit einflossen.

A.1. Das V-Modell der Bundeswehr

Das V-Modell der Bundeswehr stellt neben dem V-Modell Witt [AEH+07] die bislang weitreichendste Anpassung des des V-Modells dar. Nach der Veröffentlichung des Standard V-Modells, sollte für den Kontext der Bundeswehr zunächst die Anwendbarkeit im Projekttagesgeschäft bei der Bundeswehr geprüft werden. Erklärte Ziele waren unter anderem:

- Bekanntmachung des V-Modells in der Bundeswehr
- Eignung des V-Modells für übliche Projekte in der Bundeswehr
- Positionierung etablierter Verfahren, Normen und Standards
- Eignungsfeststellung für die Einführung/Integration
- Erarbeitung von Anpassungen und Anwendungsempfehlungen

Zu diesem Zweck wurden über einen Zeitraum von über einem Jahr verschiedene Pilotprojekte bei der Bundeswehr begleitet und betreut. In [NK05] haben wir bereits einen kurzen Überblick über die betreuten Pilotprojekte gegeben und die Erfahrungen zusammengefasst. Im Kontext dieser Arbeit interessieren wir uns hauptsächlich für die resultierende Anpassung des V-Modell Bw. Wir gehen in diesem Abschnitt kurz auf die Kernfragen und -punkte der Anpassung ein und zeigen Konflikte, die das V-Modell Bw zu einem der Auslöser für den Task RO machen.

A.1.1. Anpassungen für die Bundeswehr

Die Anpassungen, die für das V-Modell im Kontext der Bundeswehr vorgenommen wurden, beziehen sich weitgehend auf die Integration des hausinternen Beschaffungsprozesses der Bundeswehr, dem *Customer Product Management* (CPM[1] (Stand: 12.11.2007)). CPM ist ein phasenorientierter Prozess, der ein Rüstungsprojekt in die Phasen:

- Analysephase
- Projektierungsphase
- Einführungsphase
- Nutzungsphase

1 CPM ist als Publikation über die Webseiten des Bundesamtes für Wehrtechnik und Beschaffung unter: `http://www.bwb.org/02DB022000000001/vwContentByKey/W26GXEDT127INFODE/File/CPM.pdf` verfügbar.

untergliedert. Jede dieser Phasen wird durch so genannte *Phasendokumente* abgeschlossen, die den Mittelfluss begründen und den Übergang in die nächste Phase ermöglichen.

Die Anpassung des V-Modell mit Hinblick auf die Integration des CPM wurde über einen langen Zeitraum konzipiert und führte zu folgendem Integrationskonzept: CPM und V-Modell bilden einen integrierten Ansatz, wobei die grundlegende Projektstruktur am CPM festgemacht wird. Der CPM steuert somit eine grundlegende *Ablaufstruktur* und wesentliche *Ergebnistypen* bei. die Anpassung des V-Modells sieht darüber hinaus noch einige zusätzliche Rollen vor, die im Kontext der Bundeswehr Relevanz besitzen. Die Ergebnistypen wurden im V-Modell-Kontext als Produkte modelliert. Zusammen mit den Aktiviäten und Rollen bilden sie den Vorgebensbaustein *Bedarfsermittlung und Bedarfsdeckung in der Bundeswehr (CPM)* wie in Abbildung A.1 gezeigt.

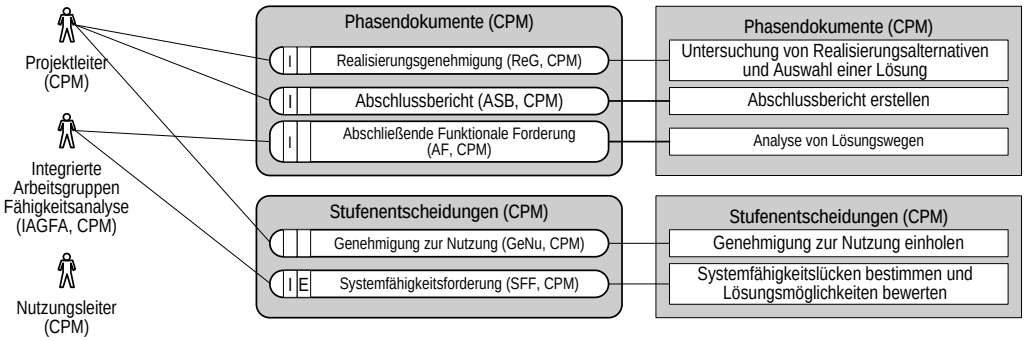


Abbildung A.1.: Vorgehensbaustein für den V-Modell Bw Anteil

Dieser Vorgehensbaustein ist jedoch nun ein Teil der Anpassung. Um das Phasenmodell CPM als Strukturierungsmittel für das V-Modell nutzbar zu machen, wurden die Phasen auf neue Projektdurchführungsstrategien abgebildet[2]. Für jede Phase gibt es also eine Projektdurchführungsstrategie, die mit einem der Phasendokumente (Abbildung A.1) abgeschlossen wird. Die Auswahl der PDS wird im V-Modell unter anderem durch das Tailoring geregelt. Für das V-Modell Bw wurde daher ein neuer *Projektmerkmalswert* (PMW) im Projektmerkmal *Projektrolle* eingeführt, der alle Standardvorgehen des V-Modells ausblendet, dafür jedoch die vier neuen Projektdurchführungsstrategien verfügbar macht[3]. Weitere Änderungen beziehen sich auf inhaltliche Fragen, wie zum Beispiel die Anwendungsempfehlung des Bundeswehr-Architekturstandards TABw anstelle des durch die KBSt empfohlenen SAGA-Standards. Diese Anpassungen/Erweiterungen konnten ebenso transparent wie ein neuer Vorgehensbaustein integriert werden.

Die Anpassungen, die für das V-Modell Bw erforderlich waren, beziehungsweise Features, die für die Bundeswehr interessant oder notwendig waren, fanden zur Version 1.2 des V-Modell XT zum Teil wieder den Weg zurück in das Kernmodell. Zu nennen wären hier beispielsweise die so genannten *Mustertexte*. Hierbei handelt es sich um Textbausteine, die während des Anpassungs- und Pflegeprozesses ermittelt und extra hinterlegt werden können. Im Rahmen des Tailorings mit dem Projektassistenten können diese Mustertexte ausgewählt und für das initiale Befüllen von Produkten mit vorgefertigten Inhalten verwendet werden. Diese stellen ein einfaches Mittel zur schnellen und unkomplizierten Bereitstellung und Nachnutzung von organisationsspezifischen Inhalten dar [Kuh06], die ohne eine langwierige organisationsspezifische Anpassung des gesamten Vorgehensmodells auskommen. Weiterhin ist durch die Notwendigkeit der Projektorganisation in der Bundeswehr (viele, unter Umständen zeitlich nicht synchrone Teilvorhaben) ein Konzept zur auftraggeberseitigen Multiprojekt-Organisation

2 Hier hat sich mittlerweile auch herausgestellt, dass die strikte Abbildung als neue PDS nicht immer trägt. Eine alternative Vorgehensweise wird gerade geprüft.

3 Dieses Vorgehen war im Stand V-Modell 1.1 eine technische Notwendigkeit. Mit den Arbeiten am Task RO wird hier ein eleganteres und robusteres Verfahren entwickelt.

(Vorgehensbaustein *Multiprojektmanagement*) erarbeitet worden. Dieser ist, wie das Konzept Mustertexte, ebenfalls wieder direkt im V-Modell Hauptentwicklungsstrang aufgenommen worden.

A.1.2. Konflikte, Zwischenlösungen und Optionen

Das V-Modell Bw ist eine sehr weitreichende und umfassende Anpassung, die jedoch auch Fragen aufwirft. So sind beispielsweise nicht alle Aspekte des in der Bundeswehr etablierten CPM adäquat abgebildet und an einigen Stellen mussten technische „Kunstgriffe" angewendet werden, um eine Abbildung zu ermöglichen. Wir gehen hier nur kurz auf drei prominente Vertreter ein. Diese zeigen aber nicht nur spezifische Fragestellungen bei der V-Modell/CPM-Integration, sondern zeigen auch Konfliktpotenzial für weitere Prozessintegrationsvorhaben (so zum Beispiel auch in [KT06] bereits andiskutiert). Mit den Arbeiten am Task RO werden diese Problemfelder auch adressiert.

Phasengebundene Rollenzuordnung: Der CPM sieht mehrere Phasen für die Projektstrukturierung vor. In allen diesen Phasen wird die Rolle des umfassenden Projektverantwortlichen als *Projektleiter* bezeichnet, mit Ausnahme der Nutzungsphase, wo diese Rolle die Bezeichnung *Nutzungsleiter* trägt. Ein Nutzungsleiter ist somit nur ein Projektleiter in der Nutzungsphase, was eine konditionale Zuordnung festlegt, die durch das V-Modell nicht unterstützt wird. Als Lösung wurde bis zum aktuellen Stand der Nutzungsleiter zwar modelliert, jedoch sind sämtliche Verantwortungsbeziehungen und -zuorndnungen auf den Projektleiter (CPM) gerichtet. Auf den Nutzungsleiter wird nur im Text verwiesen.

Modellierung von Verantwortungsbeziehungen: Diese gerade aufgedeckte Problematik zeigt auf eine weitere Schwierigkeit bei der Modellierung des V-Modell Bw. Jedem Produkt ist exakt und maximal eine *verantwortliche Rolle* eindeutig zugewiesen. Im V-Modell Bw ersetzt jedoch der *Projektleiter (CPM)* an diversen Stellen den Projektleiter nach V-Modell XT. Darüber hinaus muss die Verantwortung über sämtliche Produkte beim Eintritt in die Nutzungsphase an den *Nutzungsleiter (CPM)* übergehen. Diese Art der Modellierung wird durch das V-Modell nicht unterstützt. Zurzeit sind die Verantwortlichkeiten daher in einem separaten Anwendungsleitfaden geregelt.

Integration in das Tailoring via Projektmerkmalswert: Wie weiter oben bereits angesprochen handelt es sich hier um ein unglückliches Zusammenspiel von Erweiterung und Werkzeugen. Um die projekt- und inhaltsbezogene Verknüpfung der neuen Inhalte bei gleichzeitiger Einbindung in das Tailoring zu ermöglichen, werden im V-Modell so genannte Projektmerkmale verwendet. Im aktuellen Stand des V-Modell Bw greift nun jedoch die Besonderheit, dass die Projektmerkmale *Projektrolle* und *Projektgegenstand* als so genannte bestimmende Projektmerkmale ausgewertet werden. Um also einen neuen Projekttyp, wie das V-Modell Bw sich letztendlich darstellt, in das Tailoring zu integrieren, müssen diese Projektmerkmale angepasst werden. In diesem Fall betrifft es das Projektmerkmal *Projektrolle*, welches um den Wert *Auftraggeber in der Bundeswehr* erweitert wurde. Hierbei handelt es sich um eine technisch bedingte Problemstellung, die unter anderem auch auf das Tailoring-Konzept des V-Modells zurückzuführen ist. Daher wird auch das Tailoring im Rahmen des Task RO auf den Prüfstand gestellt.

Das V-Modell Bw zeigt sehr anschaulich die Tragweite und Herausforderungen aber auch die Möglichkeiten der Anpassung an organisationsspezifische Anforderungen. Ähnlich zu [KT06] konnten wir hier die praktische Umsetzung eines gelebten, umfassenden Prozesses beobachten und dabei wertvolle Erkenntnisse hinsichtlich der technischen und konzeptionellen Fragestellungen der Anpassung gewinnen. Im Rahmen von Task RO stellt das V-Modell Bw zusammen mit dem Standardmodell der KBSt den Bezugspunkt für alle Arbeiten zur Anpassung dar.

A.2. CollabXT – Das V-Modell im Microsoft-Umfeld

CollabXT ist ein Projekt, das auf der Basis abstrakter Übersetzungsvorschriften V-Modell Derivate für die automatische Bereitstellung von Projektarbeitsumgebungen umgestaltet [KK07, KDA07]. Im Umfeld dieses Projekts wurden verschiedene Vorabuntersuchungen durchgeführt, in denen die Integrierbarkeit der Microsoft Systeminfrastruktur (inklusive der implementierten Prozessplattform) mit dem V-Modell XT geprüft wurde [KT06, Kuh07]. Dort wurden unter anderem Fragen der Prozessintegration und der Automatisierung von Prozessen erörtert. Ziel der Vorabuntersuchungen war es, einen konkreten Leitfaden abzuleiten, der Aussagen zum allgemeinen Umgang mit V-Modell-Anpassungen macht. Hierzu wurde das Microsoft Solutions Framework analysiert und mit den Mitteln des V-Modell XT Metamodells remodelliert.

In diesem Abschnitt geben wir einen kompakten Überblick über die Resultate dieser Modellierung und geben insbesondere einen Einblick in den aus diesen Voruntersuchungen erkannten Bedarf. Weiterhin geben wir einen kurzen Einblick in die aus CollabXT resultierende Werkzeugumgebung und geben einen hier einen Einblick in die Potenziale hinsichtlich Operationalisierung. Dort interessieren uns insbesondere die Abhängigkeits- und Beziehungstypen sowie deren Auswertung im Rahmen der Generierung der Arbeitsumgebungen.

A.2.1. MSF-Integration im V-Modell XT

Die wesentlichen Vorabuntersuchungen waren durch die Notwendigkeit der flexiblen Methodenintegration in das V-Modell XT begründet. Das V-Modell schreibt zum Beispiel nicht explizit vor, wie Software konkret zu entwickeln ist. Dies wird ausdrücklich auf konkrete, im Projekt zu regelnde Verfahren ausgelagert, wobei die konkrete Schnittstelle zur Integration in den umfassenden V-Modell-Rahmen nicht eindeutig festgelegt wird. In [Kuh07] haben wir daher anhand der konkreten Methoden *Microsoft Solutions Framework for Agile Software-Development* verschiedene Integrationsverfahren geprüft und exemplarisch umgesetzt. Folgende Integrationsverfahren haben wir dabei identifiziert (aus [Kuh07]):

1. Integration des MSF als konkrete Software-Entwicklungsmethode in das V-Modell, das heißt also Ausgestaltung beziehungsweise Verfeinerung des V-Modells durch MSF-Inhalte.
2. Integration unter Beibehaltung der Zuständigkeitstrennung, das heißt: Das V-Modell bildet den Mangementrahmen für die Methode MSF.
3. Integration/Bereitstellung des V-Modell in die TFS Infrstruktur mithilfe eines Process Templates.

Wir haben in [Kuh07] für jede infrage kommende Methode eine entsprechende Diskussion geführt und beispielsweise Fragen hinsichtlich des Informationsverlustes bei der Überführung/Abbildung sowie konzeptionelle und technische Optionen diskutiert. Als Ergebnis der Analyse wurden erste Abbildungslisten und Abbildungsvorschriften ermittelt. Auch die Optionen der inhaltlichen Ersetzung bei entsprechender Überdeckung (vgl. [Kuh06]) wurden berücksichtigt. Analog zum V-Modell Bw (siehe letzter Abschnitt) wurden dann die notwendigen Strukturen im V-Modell erstellt. Diese umfassten Vorgehensbausteine und eine eigens angepasste Projektdurchführungsstrategien, die jedoch aufgrund der Vorgaben des V-Modells eindringlich auf Einhaltung beziehungsweise Konformität zur Systemarchitektur des V-Modells hin geprüft werden musste. In weiteren auf diesen Untersuchungen aufbauenden (studentischen) Arbeiten haben wir die Konzepte verfeinert und die bereits in [Kuh07] skizzierte umgekehrte Integration des V-Modells in die Infrastruktur des MSF weiter konkretisiert.

A.2.2. Werkzeugintegration und Operationalisierung

Die Konkretisierung der V-Modell-Integration in verschiedene Werkzeugumgebungen, insbesondere in die SharePoint Services und den Team Foundation Server (TFS), haben

wir unter dem Projekt *CollabXT* [KK07, KDA07] zusammengefasst. Abbildung A.2 zeigt das original entwickelte Prozessintegrationsverfahren für die TFS-Integration. Es ist zu sehen, dass weitgehend die standardmäßig mit dem V-Modell gelieferten Werkzeuge zu Einsatz kommen, um eine organisationsspezifische Anpassung des V-Modells vorzunehmen (vgl. auch Kapitel 2.2.2). Darauf aufbauend wird eine Generatorstufe implementiert, welche die V-Modell-Strukturen auf die Zielplattform überführt. Dort kann der resultierende Prozess mit den durch die Zielplattform zur Verfügung gestellten Werkzeugen ggf. noch weiter editiert werden. Für das hier gezeigte Beispiel MSF im TFS verweisen wir dazu auf Kapitel 2.2.2. In Konsequenz wird das V-Modell somit auf eine Plattform zur Ausführungsunterstützung eines konkreten Projekts überführt, womit die Werkzeugkette des V-Modell XT konsequent fortgesetzt wird.

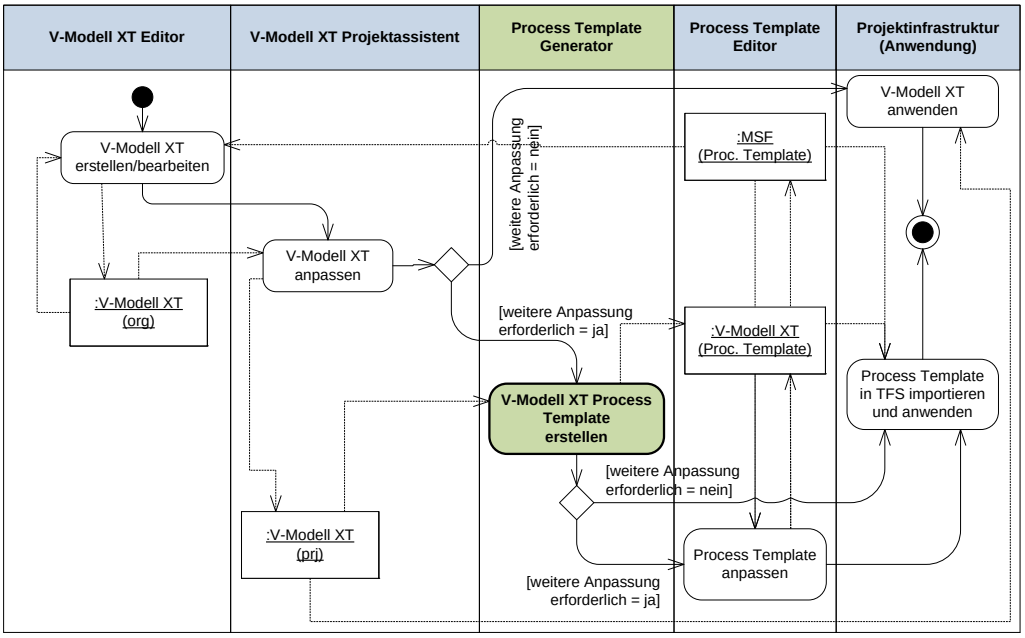


Abbildung A.2.: Prozessintegrationsverfahren für das V-Modell XT aus [Kuh07]

Für die Integration in SharePoint verläuft der Prozess analog. In beiden Szenarios und darüber hinaus verallgemeinert für weitere Plattformen sind die Strukturen geeignet und soweit möglich automatisch zu übersetzen. Dabei handelt es sich im Wesentlichen um eine Modelltransformation über den Metamodellen der betrachteten Vorgehensmodelle. Interessant ist aber auch der Punkt der *methodischen Anreicherung*. Hier haben wir mit CollabXT verschiedene Möglichkeiten vorgeschlagen, einen gegebenen abstrakten Prozess weiter zu konkretisieren. Daraus resultieren dann selbstverständlich wieder Fragen hinsichtlich Modellierungssprache, Konformität zum Ausgangsmodell, beziehungsweise zum Standardmodell und insbesondere Stabilität hinsichtlich von Änderungen am Vorgehensmodell.

A.2.3. Auswertung von Beziehungstypen

Als wesentlich hat sich hierbei die Möglichkeit zur Auswertung von Vorgehensmodellkonfigurationen herausgestellt. Insbesondere für die automatische Ermittlung von Elementen für die Erzeugung von Arbeitsumgebungen sind Abhängigkeiten zwischen den relevanten Elementen besonders wichtig. So zum Beispiel werden für den SharePoint-Anteil von CollabXT Produkt-Aktivitätsbeziehungen für die Vorinstanziierung und die Konfiguration des Aufgabenmanagements verwendet. Für den TFS-Anteil werden beispielsweise die Zuordnungen zwischen Aktivitäten, Produkten und Entscheidungspunkten ausgewertet und dadurch die Ergebnisse und Vorgänge einzelnen Projektfort-

schrittstufen zugeordnet. Im Kapitel 3.2.1 sind wir auf alle wesentlichen Beziehungstypen des V-Modell XT, die hier verwendet werden, eingegangen.

Im Kontext dieser Arbeit finden sich die Erkenntnisse hinsichtlich der Vorteile stark ausgeprägter Beziehungssysteme dort wieder, wo flexible Prozesskopplungen oder eine flexible Modellierung erforderlich sind.

A.3. Task RO – Anpassungskonzept für das V-Modell XT

Wie bereits in der Einleitung dieser Arbeit erwähnt, hat das V-Modell XT bereits in den letzten zwei Jahren einen erheblichen evolutionären Prozess durchlebt und entwickelt sich ständig weiter. Die Erstellung von organisationsspezifischen Anpassungen und einer englischen Sprachvariante in Kombination mit einen regelmäßigen Aktualisierungszyklus zeigt die Grenzen des aktuell vorliegenden Metamodells. Der *Task RO* (Aufteilung des V-Modells in Referenz- und Organisationsanteile) hat zur Aufgabe, das Metamodell des V-Modells zu aktualisieren und den Erfordernissen hinsichtlich Variantenbildung und regelmäßiger Aktualisierung anzupassen. Task RO ist ein aufwandsintensives Teilprojekt des V-Modell XT Entwicklungsprojekts *WEIT* und zum Zeitpunkt der Erstellung dieser Arbeit noch nicht abgeschlossen. Wir berichten hier in Auszügen über Ziele, aktuellen Entwicklungsstand und Synergieeffekte zwischen Task RO und dieser Arbeit.

A.3.1. Herausforderungen für die Weiterentwicklung des V-Modells

Die Frage bei der Weiterentwicklung und Pflege des V-Modell haben wir bereits zu Beginn dieser Arbeit aufgeführt (siehe Kapitel 1.1.2):

- Wie ist die (allgemeine) Weiterentwicklung des V-Modells sicherzustellen?
- Wie wirken sich Weiterentwicklungen (Versionen) auf die sich im Feld befindenden Varianten hinsichtlich Verbindlichkeit und Rechtssicherheit aus?
- Wie wird Feedback aus einer Variante oder Instanz dem „Hauptstrang" und allen weiteren Varianten zugänglich gemacht?
- Wie wird die Konformität einer Variante zum „Hauptstrang" sichergestellt?
- Ergänzend zu oben stehenden Fragen: Wie wird dann in Bezug auf Versionen verfahren?

Die Herausforderungen folgen unmittelbar und beziehen sich dabei primär auf Wartungsarbeiten und Pflegeaufgaben. So muss beispielsweise in Zukunft sichergestellt werden, das einmal gemachte Anforderungen nicht zeitintensiv mit jedem neuen Release des V-Modells von Hand synchronisiert werden müssen. Punktuelle Aktualisierungen müssen möglich sein, um beispielsweise auch die langwierigen und umfangreichen Releasebuilds kontrollierbar zu gestalten und schneller und unkomplizierte Minimalanpassungen durchführen zu können. Dazu ist die (physische) Aufteilung des V-Modells ebenfalls anzustreben.

Die Vision ist hierbei, das V-Modell XT zu einem Prozess Framework weiter zu entwickeln, in dem es einen zertifzierbaren, zentral qualitätsgesicherten Referenzanteil gibt, der kontinuierlich gepflegt und weiterentwickelt wird. Darauf aufbauend wird ein so genannter organisationsspezifischer Anteil aufgesetzt, der sämtliche Anpassungen des Referenzmodells im Kontext einer implementierenden Organisation enthält. Der Organisationsanteil enthält aber nicht nur Erweiterungen und Ergänzungen, sondern kann zusätzlich im Rahmen eines Variabilitätskonzeptes bestimmte Anpassungen der Referenzinhalte vornehmen. Da sämtliche Anpassungen und Variantionen strikt vom Referenzteil getrennt sind, vereinfacht sich einerseits die Wartung des Referenzanteils und andererseits erhöht sich somit tendenziell die Stabilität der organisationsspezifischen Anpassung, da sie Aktualisierungen des Referenzmodells im Idealfall transparent „durchreicht".

A.3.2. Überblick und Stand

Task RO ist zum Zeitpunkt der Erstellung dieser Arbeit selbst noch in Bearbeitung. Im aktuellen Stand liegt bereits ein neu gestaltetes Metamodell vor, dass statische, dynamische und Tailoringanteile modelliert. Darauf aufbauend definiert es ein Konzept für Anpassungsoperationen, unter dem Erweiterungs- und Variabilitätsoperationen zusammengefasst werden. Erweiterungsoperationen explizieren dabei die bislang bereits verfügbaren Optionen zur Ausgestaltung des V-Modells (zum Beispiel das Anlegen neuer Vorgehensbausteine). Variabilitätsoperationen realisieren die „nachträgliche" Rekonfiguration des V-Modells im Kontext des Tailorings und des Exports.

Zum Zeitpunkt der Erstellung dieser Arbeit wird gerade der erste technische Durchstich des neuen Metamodells erstellt und dieses damit auf Tragfähigkeit überprüft.

A.3.3. Synergien

Durch die Mitarbeit am Task RO finden sich einige Konzepte dieser Arbeit im V-Modell wider; andererseits sind Ideen dieser Arbeit auch aus diesem Kontext heraus motiviert. Wir geben nun drei signifikante Konzepte an, die in dieser Arbeit entwickelt und im Rahmen von Task RO auf das V-Modell XT überführt und dort angewendet wurden.

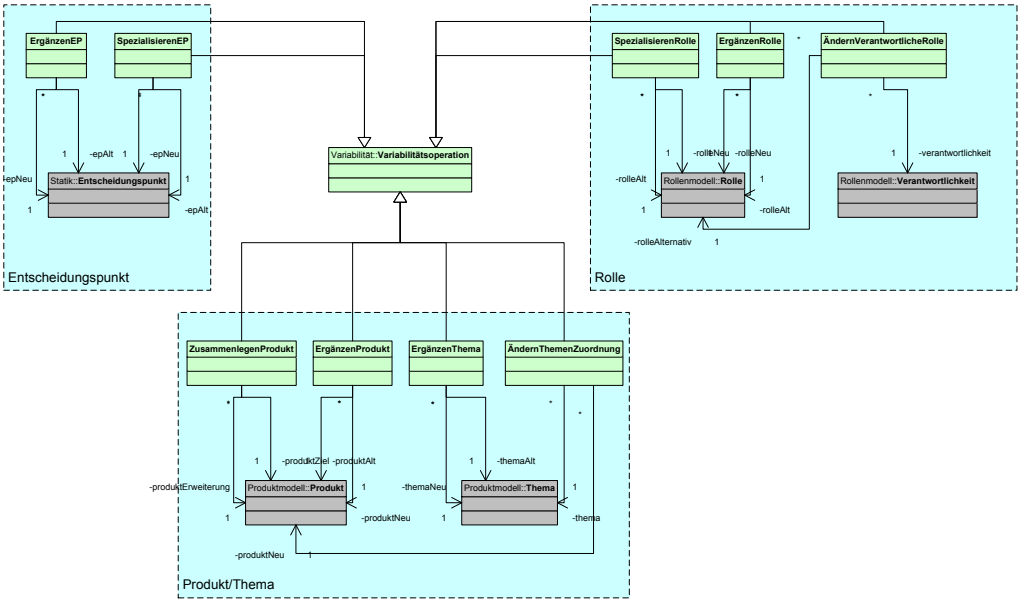


Abbildung A.3.: Ausschnitt des V-Modell XT Variabilitätsmodells (Vorabversion)

Konfigurationskonzept: Ein wesentliches Konzept dieser Arbeit ist ein *konfigurationsbasierter Ansatz* zur Erstellung von Vorgehensmodellen auf der Basis eines Entwicklungsstandards. Im neuen V-Modell Metamodell finden sich die Konfigurationsansätze bei der Definition der Projekttypen und der darauf aufbauenden Projekttypvarianten wieder. Diese bilden einen Rahmen für eine definierte Menge von Vorgehensbausteinen und Projektdurchführungsstrategien. Sie können durch die Projektmerkmale weiter parametrisiert werden.

Variablitätsoperationen: Das in dieser Arbeit entwickelte, auf den Prozesselementabhängkeiten aufbauende Konzept der Variabilitätsoperation ist zurzeit nahezu unverändert im neuen V-Modell Metamodell enthalten. Der Terminus *Variabilitätsoperation* ist dafür aus dem Projekt heraus als Bezeichner für die speziellen Formen der Prozesselementabhängigkeiten übernommen worden. Im neuen V-Modell Metamodell liegen die Variabilitätsoperationen jedoch bereits in konkreter typisierter Form vor, wie Abbildung A.3 andeutet.

Wartungs- und Pflegeprozesse: Die Wartungs- und Pflegeprozesse werden im Kontext von Task RO indirekt angewendet, wenn zugelassende Operationen auf dem Referenzmodell, insbesondere zu dessen Konkretisierung, definiert werden. Die Operationen im neuen V-Modell Metamodell, beziehungsweise im Kontext der V-Modell Weiterentwicklung sind jedoch bereits konkretisiert. Eine Abbildung auf die in Kapitel 5.2 beschriebenen Prozesse ist jedoch unmittelbar nach Anpassung der Terminologie möglich. Teile der oben bereits erwähnten *Erweiterungsoperationen* des neuen V-Modell Metamodells finden sich $1:1$ im hier entwickelten Lebenszyklusmodell wieder.

B. Case Study: Entwicklungsprozess eines Vorgehensmodells mit IMV

In diesem Anhang greifen wir das V-Modell XT wieder auf, das uns bereits durch die ganze Arbeit begleitet. Nachdem wir uns in den Kapiteln 4 bis 6 immer wieder punktuell auf das V-Modell bezogen haben, wenden wir nun unseren *IMV*-Ansatz an, um dem minimalen Basismodell aus Kapitel 5.1 weitere relevante Submodelle hinzuzufügen und zeigen dabei die Integration in das bereits vorhandene Modell. Bezogen auf die Prozesse der Vorgehensmodellentwicklung (vgl. Kapitel 5.2) betrachten wir hier also Änderungen, Anpassungen beziehungsweise Erweiterungen auf der Ebene des Metamodells. Wir stellen eine Erweiterung um ein Planungsmodell nach [Gna05] exemplarisch vor.

Metamodellerweiterung: Planungsmodell

Das von Gnatz in seiner Arbeit vorgestellte Ablauf- und Planungsmodell integrieren wir mit den Mitteln unseres Modells. Abbildung B.1 zeigt den planungsrelevanten Anteil und die entsprechende Integration mit unseren Mitteln.

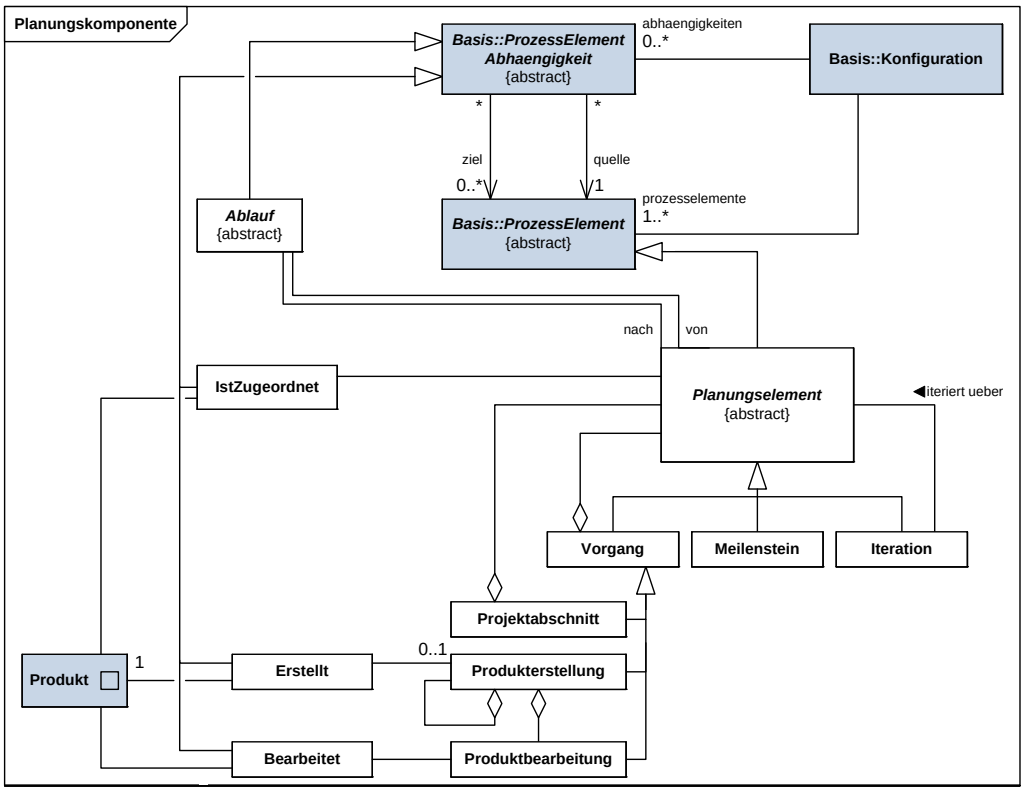


Abbildung B.1.: Erweiterung des Basismodells um Planungskomponenten

Zuerst definieren wir die Klasse `Planungselement` als das zentrale Prozesselement in der *Planungskomponente*. Die Planungskomponente sehen wir als Container analog zur Prozesskomponente (Kapitel 5.1.2) vor. Bezogen auf die Ontologie für Vorgehensmo-

delle (Kapitel 2.4.1) setzt die Planungskomponente das *Ablauf-Submodell* um. Unterhalb von `Planungsmodell` übernehmen wir die Modellierung von Gnatz.

Abläufe modelliert Gnatz durch die abstrakte Klasse `Ablauf`, die er durch `EA-Vorgangsfolge`, `EE-Vorgangsfolge`, `AA-Vorgangsfolge` und `AE-Vorgangsfolge` konkretisiert. Mithilfe des *IMV*-Ansatzes modellieren wir Abläufe durch eine Prozesselementabhängigkeit (Abbildung B.1 zeigt nur die Wurzelklasse, die Kinder zeigen wir aufgrund der Übersichtlichkeit nicht). Die Kopplung von konkreten Abläufen an Ergebnisse nimmt Gnatz in Anlehnung an das V-Modell XT produktorientiert vor. Wir modellieren dies analog zu Kapitel 5.1.2 über die speziellen Prozesselementabhängigkeiten: `IstZugeordnet`, `Erstellt` und `Bearbeitet`.

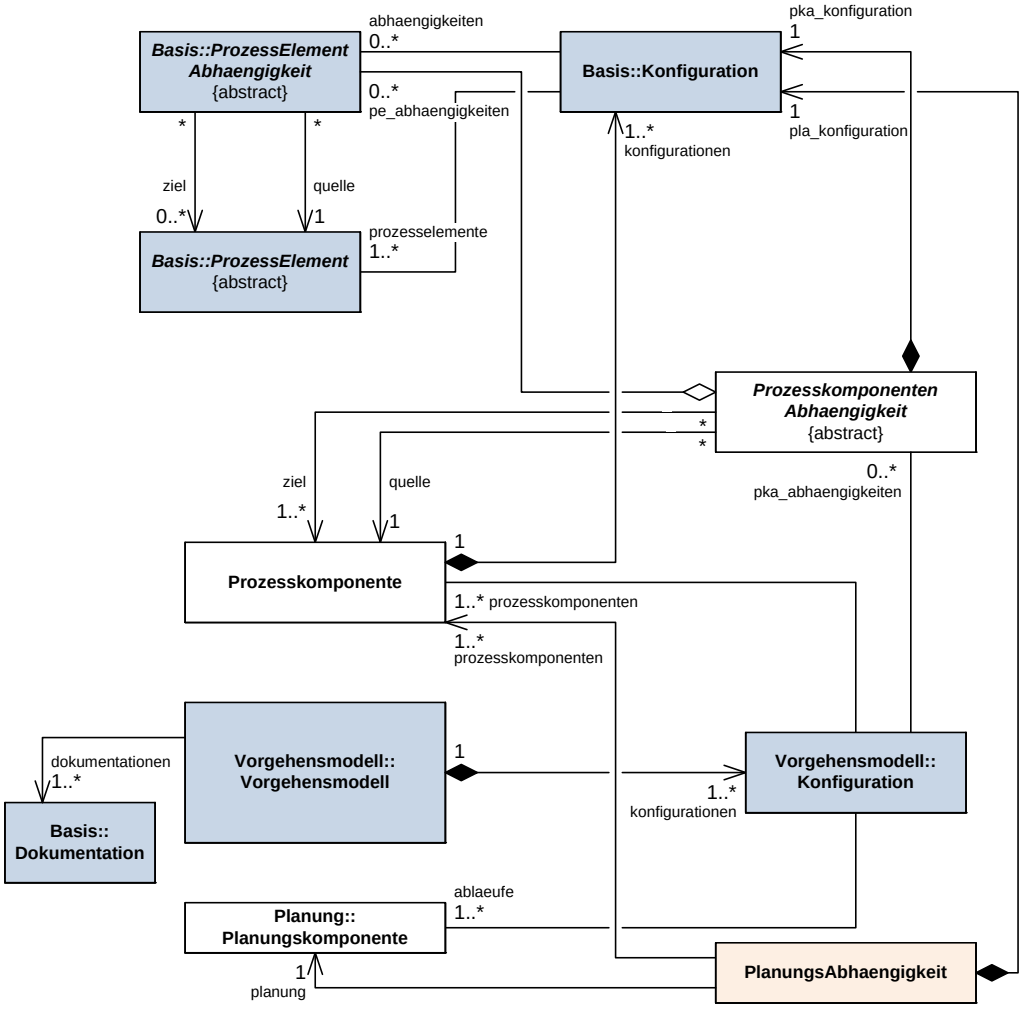


Abbildung B.2.: Integration der Planungskomponenten

Abbildung B.2 zeigt die Integration der Planungskomponenten in das Metamodell aus Kapitel 5.1. Parallel zur Prozesskomponentenabhängigkeit geben wir eine *Planungsabhängigkeit*. Diese verbindet eine Planungskomponente mit einer Menge Prozesskomponenten. Analog zur Prozesskomponentenabhängigkeit werden alle Prozesselemente (aus allen in dieser Abhängigkeit hinterlegten Konfiguration) zugreifbar. Die Ablaufelemente können so mit den Prozesselementen verbunden werden.

C. Werkzeugkonzept

Das in den Kapiteln 4 und 5 entwickelte Vorgehensmetamodell, dessen Architektur und das Lebenszyklusmodell legen eine breite Unterstützung durch verschiedene Werkzeuge nahe. Im Kapitel 5.3 haben wir ein entsprechendes Werkzeugkonzept entworfen. Für Teilbereiche (für die Komponentenerstellung und -konfiguration) ist bereits ein Prototyp in der Umsetzung, auf den wir kurz eingehen wollen.

C.1. Modellierungstechnik und Basismodell

Wir stellen zuerst die Modellierungstechnik anhand des Basismodells dar. Zur Umsetzung wird die .NET-Plattform verwendet. Visual Studio als Werkzeug bietet hierfür Designer für so genannte *Domänenspezifische Sprachen* (DSL) an, mit denen die dem Vorgehensmodell zugrunde liegenden Metamodelle technologieorientiert remodelliert werden können. In der Abbildung C.1 ist das für das *Basismodell* (vgl. Kapitel 5.1.1, Abbildungen 5.2 und 5.3) gezeigt.

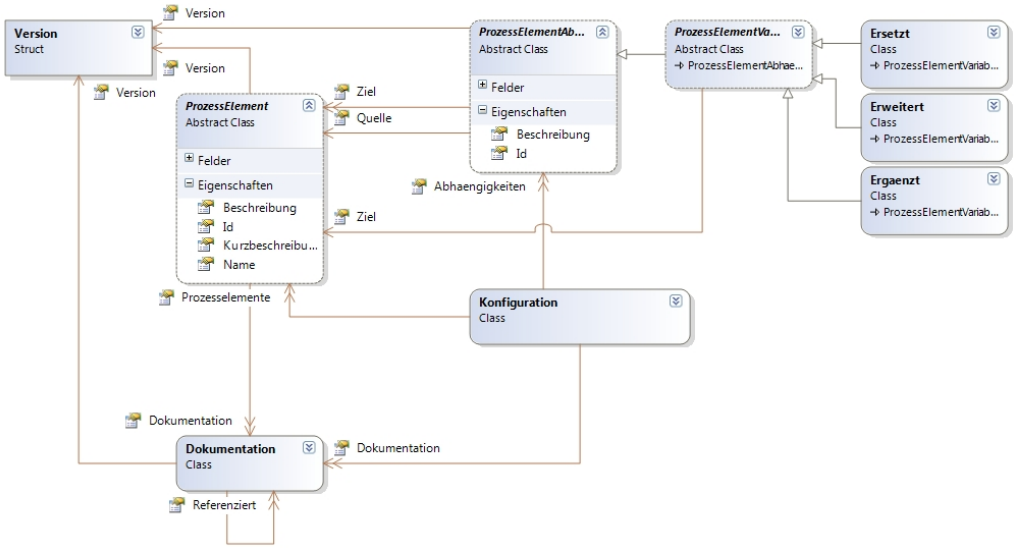


Abbildung C.1.: Basismodell des Vorgehensmetamodells in Visual Studio DSL modelliert

Dieses Basismodell stellt die Grundlage des Prozessframeworks dar, auf dem alle weiteren Element aufbauen. Da wir .NET als technologische Grundlage gewählt haben, entsteht hier nach dem Vorbild der .NET-Plattform eine objektorientierte Klassenbiliothek, die beliebig erweitert und angepasst werden kann.

C.2. Prozesskomponenten

Aufbauend auf dem Basismodell zeigt Abbildung C.2 eine Sicht auf die *Prozesskomponente*. Zu sehen sind neben den abstrakten Klassen des Basismodells bereits die Konfigurationsanteile von Prozesskomponenten (die Prozesskomponentenabhängigkeiten), sowie deren konkrete Ausprägungen aus Kapitel 5.1.2 in Abbildung 5.6.

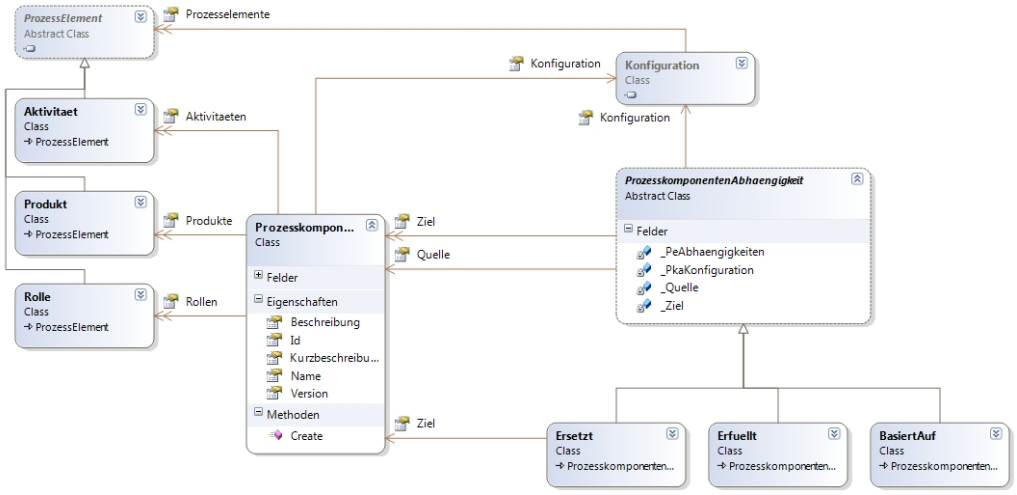


Abbildung C.2.: Prozesskomponentenmodell des Vorgehensmetamodells in Visual Studio DSL modelliert

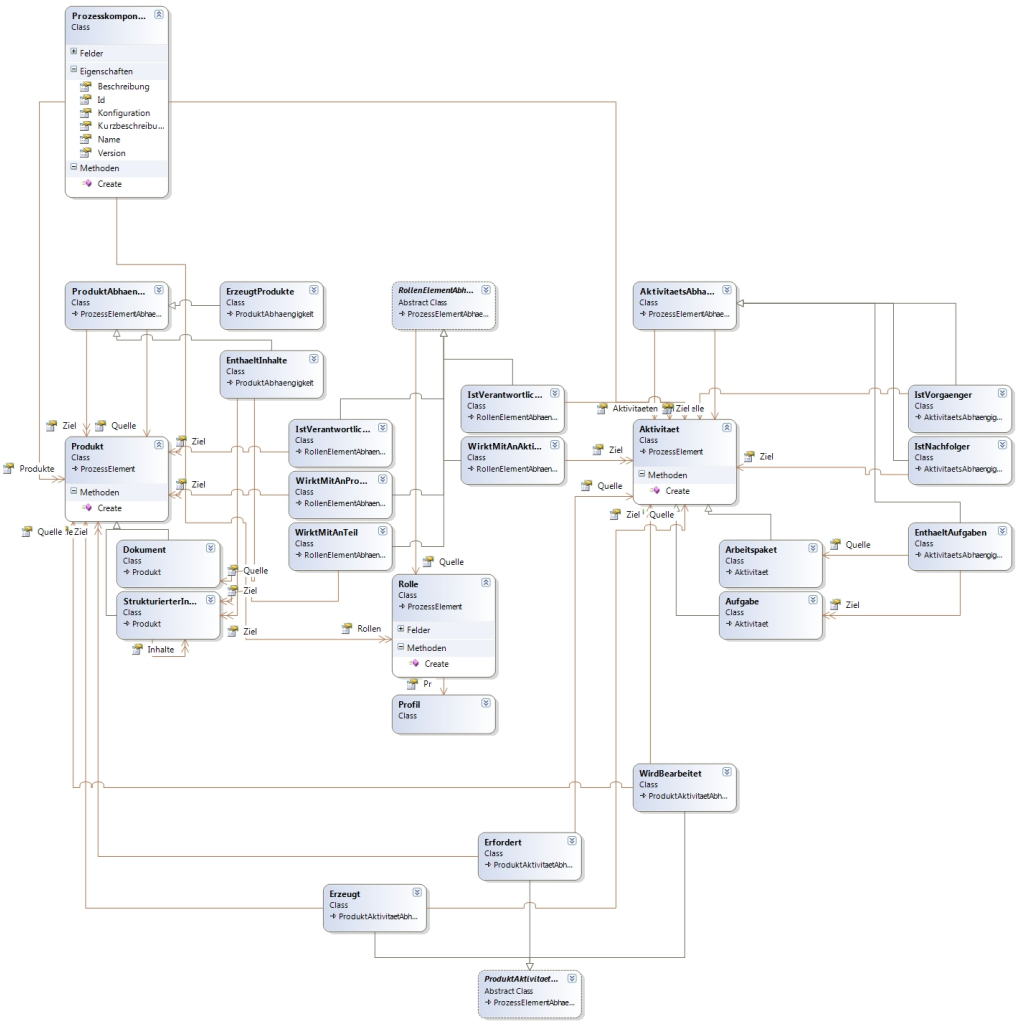


Abbildung C.3.: Prozesskomponentenmodell des Vorgehensmetamodells in Visual Studio DSL modelliert (detailliert)

In Abbildung C.2 ist ein vereinfachte Sicht gezeigt. Im Kapitel 5.1.2 haben wir schrittweise die strukturellen Submodelle eines Vorgehensmodells auf der Basis von Prozesselementen und Prozesselementabhängigkeiten ausgestaltet. Abbildung C.3 zeigt eine integrierte, vollständige Sicht auf das DSL-Klassenmodell. Es unterstreicht noch einmal die These der Arbeit, dass der Anspruch bei der Vorgehensmodellierung bei der Identifikation und Modellierung der Abhängigkeitsstrukturen liegt. Obwohl wir ein vergleichsweise einfaches Metamodell erstellt haben, ist die Integrationsstruktur über die Assoziationsklassen recht umfangreich.

C.3. Vorgehensmodellkonfiguration

Der letzte Aspekt der Strukturmodellierung, auf den wir hier eingehen wollen, ist der Metamodellanteil, der die Grundstruktur der Vorgehensmodellkonfiguration abdeckt (Kapitel 5.1.1, Abbildung 5.4).

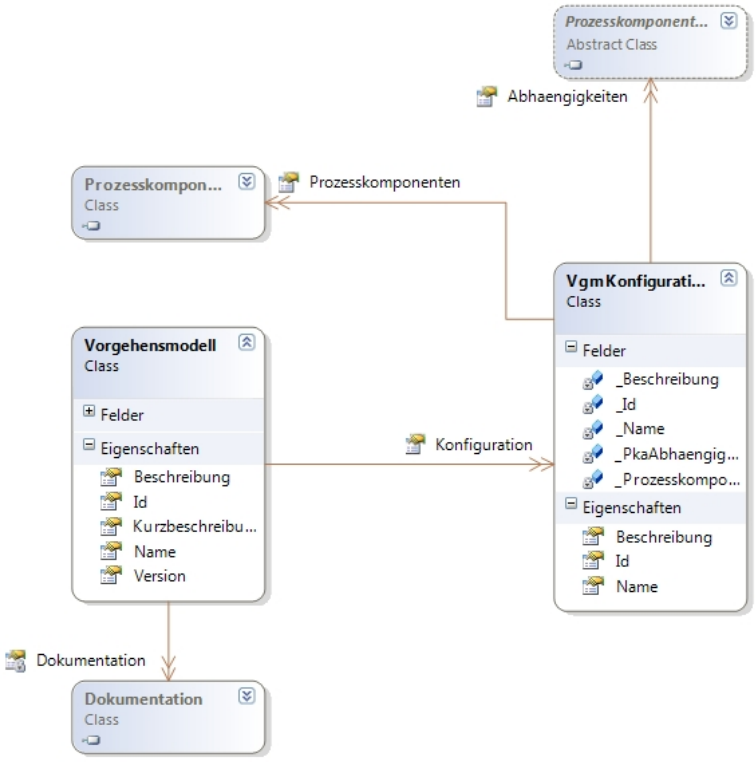


Abbildung C.4.: Vorgehensmetamodell in Visual Studio DSL modelliert

Abbildung C.4 zeigt diesen Aspekt des Metamodells, womit wir bis hier die Basistrukturen in Form von DSL-Klassendiagrammen modelliert haben. Aus den Modellen erzeugt das Visual Studio synchronisierte Codes für Klassen, Interfaces und sonstige Größen zur Programmierung der .NET Plattform. Somit ist das Visual Studio auch für die Metamodellierung verwendbar, sofern als Ergebnis Code entstehen soll.

Das so modellierte IMV-basierte Vorgehensmetamodell dient dem Werkzeug als Datenmodell. Sämtliche Metamodellanteile des Kapitels 5.1 sind hier erfasst. Aufbauend darauf wird das Werkzeug die Prozesse aus Kapitel 5.2 realisieren.

Literaturverzeichnis

[ABS04] AMELUNXEN, C., L. BICHLER und A. SCHÜRR: *Codegenerierung für Assoziationen in MOF 2.0*. In: *Proceedings Modellierung 2004*, Band P-45 der Reihe *Lecture Notes in Informatics*, Seiten 149–168, Bonn, 3 2004. Gesellschaft für Informatik.

[AEH+07] ARMBRUST, O., J. EBELL, U. HAMMERSCHALL, J. MÜNCH und D. THOMA: *Prozesseinführung und -reifung in der Praxis: Erfoolgsfaktoren und Erfahrungen*. In: *Proceedings des 14. Workshop der Fachgruppe WI-VM der Gesellschaft für Informatik e.V. (GI)*, Nummer ISBN: 978-3-8322-6111-5, Seiten 3–15. Shaker Verlag, apr 2007.

[AHM06] ANVIK, J, L. HIEW und G. C. MURPHY: *Who should fix this bug?* In: *ICSE '06: Proceeding of the 28th international conference on Software engineering*, Seiten 361–370, New York, NY, USA, 2006. ACM Press.

[AKRS06] AMELUNXEN, C., A. KÖNIGS, T. RÖTSCHKE und A. SCHÜRR: *MOFLON: A Standard-Compliant Metamodeling Framework with Graph Transformations*. In: RENSINK, A. und J. WARMER (Herausgeber): *Model Driven Architecture - Foundations and Applications: Second European Conference*, Band 4066 der Reihe *Lecture Notes in Computer Science (LNCS)*, Seiten 361–375, Heidelberg, 2006. Springer Verlag, Springer Verlag.

[AMK+06] ABE, S., O. MIZUNO, T. KIKUNO, N. KIKUCHI und M. HIRAYAMA: *Estimation of project success using Bayesian classifier*. In: *ICSE '06: Proceeding of the 28th international conference on Software engineering*, Seiten 600–603, New York, NY, USA, 2006. ACM Press.

[ARS05] AMELUNXEN, C., T. RÖTSCHKE und A. SCHÜRR: *Graph Transformations with MOF 2.0*. In: GIESE, H. und A. ZÜNDORF (Herausgeber): *Proc. 3rd International Fujaba Days 2005*, Band tr-ri-05-259, Seiten 25–31. Universität Paderborn, 9 2005.

[bAB05] AHMED BOULILA, N. BEN: *A Framework for Distributed Collaborative Software Design Meeting*. Doktorarbeit, Technische Universität München, 2005.

[Bal00] BALZERT, HELMUT: *Lehrbuch der Softwaretechnik Band 1/2*. Spektrum Akademischer Verlag, 2 Auflage, 2000. ISBN 3-8274-0480-0.

[BB01] BACHMANN, F. und L. BASS: *Managing Variability in Software Architectures*. In: *Symposium on Software Reusability*, 2001.

[BBM05] BHUTA, J., B. BOEHM und M. MEYERS: *Process Elements: Components of Software Process Architectures*. In: LI, MINGSHU, BARRY BOEHM und LEON J. OSTERWEIL (Herausgeber): *Unifying the Software Process Spectrum*, Band LNCS 3840 der Reihe *Lecture Notes in Computer Science*. Springer, 2005. International Software Process Workshop, SPW 2005, Beijing, China, May 25-27.

[BCC+97] BASS, L., P. CLEMENTS, S. COHEN, L. NORTHROP und J. WITHEY: *Product Line Practice Workshop Report*. Technischer Bericht CMU/SEI-97-TR-003, Software Engineering Institute, 1997.

[Bec03] BECK, K.: *Extreme Programming*. Nummer ISBN-13: 978-3827321398. Addison-Wesley, München, 2003.

[BEJ06] BUSCHERMÖHLE, R., H. ECKHOFF und B. JOSKO: *Success – Erfolgs- und Misserfolgsfaktoren bei der Durchführung von Hard- und Softwareentwicklungsprojekten in Deutschland*. Nummer ISBN 978-3-8142-2035-2. BIS-Verlag der Carl von Ossietzky Universität Oldenburg, 2006.

[Ber06] BERGNER, K.: *Plangenerierung im V-Modell XT 1.2*. In: HOCHBERGER, CHRISTIAN und RÜDIGER LISKOWSKY (Herausgeber): *Informatik 2006 - Bei-*

träge der 36. Jahrestagung der Gesellschaft für Informatik e.V.(GI), Band P-94 der Reihe *Lecture Notes in Informatics*. Gesellschaft für Informatik, oct 2006. available at http://www.gi-ev.de/LNI.

[BH06] BARTELT, C. und S. HEROLD: *Modellorientiertes Variantenmanagement*. In: MAYR, HEINRICH C. und RUTH BREU (Herausgeber): *Modellierung 2006*, Nummer P-82 in *Lecture Notes in Informatics (LNI)*, Seiten 172–182, Mar 2006.

[BHS00] BREU, R., W. HUBER und W. SCHWERIN: *Conformity and Integration of Software Processes*. In: *Proceedings of ICSSEA'2000, Volume 2, CNAM Paris, France, December 2000*, 200.

[Bic04] BICHLER, L.: *Codegeneratoren für MOF-basierte Modellierungssprachen*. Doktorarbeit, Universität der Bundeswehr München, 2004.

[BK06] BRABÄNDER, E. und J. KLÜCKMANN: *Geschäftsprozessmanagement als Grundlage für SOA*. Objekt Spektrum, (September/Oktober 2006 - 5):32–40, 2006.

[BKPS04] BÖCKLE, G., P. KNAUBER, K. POHL und K. SCHMID: *Software-Produktlinien – Methoden, Einführung und Praxis*. Nummer ISBN-13: 978-3898642576. dpunkt.verlag, 2004.

[BL95] BODENDIEK, R. und R. LANG: *Lehrbuch der Graphentheorie*, Band Band 1 und 2 der Reihe *Spektrum Hochschultaschenbuch*. Spektrum Akademischer Verlag, erste Auflage, 1995.

[BLPR06] BAUER, B., F. LAUTENBACHER, G. PALFINGER und S. ROSER: *AGILPRO: Modellierung, Simulation und Ausführung agiler Prozesse*. Objekt Spektrum, (Januar/Februar 2007 1):52–59, 2006.

[Boe05] BOEHM, B.: *The Future of Software Processes*. In: LI, MINGSHU, BARRY BOEHM und LEON J. OSTERWEIL (Herausgeber): *Unifying the Software Process Spectrum*, Band LNCS 3840 der Reihe *Lecture Notes in Computer Science*. Springer, 2005. International Software Process Workshop, SPW 2005, Beijing, China, May 25-27.

[Bos06] BOSCH, S.: *Algebra*. Nummer ISBN-13: 978-3540298809 in *Springer-Lehrbuch*. Springer, Sechste Auflage, 2006.

[BRSV98] BERGNER, K., A. RAUSCH, M. SIHLING und A. VILBIG: *A Componentware Methodology based on Process Patterns*. In: *GI-Workshop,Frankfurt am Main*, 1998.

[BS04] BROY, M. und R. STEINBRÜGGEN: *Modellbildung in der Informatik*. Nummer ISBN: 3-540-44292-8 in *Xpert.press*. Springer, 2004.

[BULL07] BADI ULMER, P., A. LORENZ und G. LORENZ: *Kennzahl-getriebenes Controlling zur Optimierung der Softwareentwicklung und -pflege – Ein Praxisbericht*. In: KOSCHKE, RAINER, OTTHEIN HERZOG, KARL-HEINZ RÖDIGER und MARC RONTHALER (Herausgeber): *Informatik 2007 - Beiträge der 37. Jahrestagung der Gesellschaft für Informatik e.V.(GI),*, Band P-110 der Reihe *Lecture Notes in Informatics*. Gesellschaft für Informatik, 2007.

[Bur02] BURGHARDT, M.: *Einführung und Projektmanagement - Definition, Planung, Kontrolle, Abschluss*. Publics Corporate Publishing, 4 Auflage, 2002.

[CDKT01] CHASTEK, G., P. DONOHOE, K. C. KANG und S. THIEL: *Product Line Analysis: A Practical Introduction*. Technischer Bericht, Software Engineering Institute, 2001.

[CHE04] CZARNECKI, K., S. HELSEN und U. EISENECKER: *Staged Configuration Using Feature Models*. In: *Proceedings Of Software Product Line Conference*, 2004.

[Chr92] CHROUST, G.: *Modelle der Software-Entwicklung*. Nummer ISBN: 3-486-21878-6. R. Oldenbourg Verlag München Wien, 1992.

[CM02] CHASTEK, G. und J. D. MCGREGOR: *Guidelines for Developing a Product Line Production Plan*. Technischer Bericht CMU/SEI-2002-TR-006, Software Engineering Institute, 2002.

[Coh99] COHEN, S.: *Guidelines for Developing a Product Line Concept of Operations*. Technischer Bericht CMU/SEI-99-TR-008, Software Engineering Institute, 1999.

[Die06] DIESTEL, R.: *Graphentheorie*. Nummer ISBN-13: 978-3540213918. Springer, Dritte Auflage, 2006.

[Dol06] DOLINS, S. B.: *Using the balanced scorecard process to compute the value of software applications*. In: *ICSE '06: Proceeding of the 28th international conference on Software engineering*, Seiten 881–884, New York, NY, USA, 2006. ACM Press.

[DW99] DRÖSCHEL, W. und M. WIEMERS: *Das V-Modell 97*. Oldenburg, 1999.

[epf07] *Das Eclipse Process Framework (EPF) Online Portal*. Online, `http://www.eclipse.org/epf`, Juli 2007.

[FD02] FÜERMANN, T. und C. DAMMASCH: *Prozessmanagement – Anleitung zur Steigerung der Wertschöpfung*. Nummer ISBN-13: 978-3446219168. Hanser, 2002.

[FHS01] FIRESMITH, D. und B. HENDERSON-SELLERS: *The OPEN Process Framework. An Introduction*. Nummer ISBN-13: 978-0201675108. Addison-Wesley Longman, 2001.

[Fis06] FISCHER, E.: *Entwicklung und Erprobung einer formalen Darstellung des V-Modell XT für die Prozesssteuerung*. Diplomarbeit, Technische Universität Dresden, 2006.

[FK07] FRITZSCHE, M. und P. KEIL: *Kategorisierung etablierter Vorgehensmodelle und ihre Verbreitung in der deutschen Software-Industrie*. Interner Bericht des Projekts IOSEW TUM-I0717, Technische Universität München, 2007.

[Fri06] FRIEDRICH, J.: *Technische und semantische Transformation von Vorgehensmodellen*. Diplomarbeit, Technische Universität München, 2006.

[Gau06] GAU, T.: *UMA und EPF: Einführung und Anwendung in der Praxis*. Objekt Spektrum, (November/Dezember 2006 - 6):42–47, 2006.

[GDMR04] GNATZ, MICHAEL, MARTIN DEUBLER, MICHAEL MEISINGER und ANDREAS RAUSCH: *Towards an Integration of Process Modeling and Project Planning*. In: *Proceedings of the 5th International Workshop on Software Process Simulation and Modeling (ProSim 2004). ICSE 2004*, 2004.

[GF06] GRONAU, N. und J. FRÖMMING: *KDML – Eine semiformale Beschreibungssprache zur Modellierung von Wissenskonversionen*. Wirtschaftsinformatik, 48(5):349–360, oct 2006.

[GFd98] GRISS, M. L., J. FAVARO und M. D'ALESSANDRO: *Integrated Feature Modeling with RSEB*. In: *Proceedings of Fifth International Conference on Software Reuse ICSR'98*, 1998.

[GHSY97] GRAHAM, I., B. HENDERSON-SELLERS und H. YOUNESSI: *The Open Process Specification*. ACM-Press. Addison-Wesley, 1997.

[GMP+01] GNATZ, M., F. MARSCHALL, G. POPP, A. RAUSCH und W. SCHWERIN: *Modular Process Patterns supporting an Evolutionary Software Development Process*. In: (EDITOR), V. AMBRIOLA (Herausgeber): *Proceedings of the Eight European Workshop on Software Process Technology 2001*, Nummer 2077 in *Lecture Notes in Computer Science*, Nov 2001.

[GMP+02a] GNATZ, M., F. MARSCHALL, G. POPP, A. RAUSCH, M. RODENBERG-RUIZ und W. SCHWERIN: *Proceedings of the 1st Workshop on Software Development Patterns*. Technischer Bericht TUM-I0213, Technische Universität München, 2002.

[GMP+02b] GNATZ, M., F. MARSCHALL, G. POPP, A. RAUSCH und W. SCHWERIN: *Towards a Tool Support for a Living Software Development Process*. In: *HICSS-35 Conference Proceedings*, 2002.

[GMP+03] GNATZ, M., F. MARSCHALL, G. POPP, A. RAUSCH und W. SCHWERIN: *Enabling a Living Software Development Process with Process Patterns*. Technischer Bericht TUM-I0310, Technische Universität Müchen, 2003.

[Gna05] GNATZ, M.: *Vom Vorgehensmodell zum Projektplan*. Doktorarbeit, Technische Universität München, 2005.

[Gna06] GNATZ, M.: *V-Modell XT: Meta-Modell und Konsistenzbedingungen*. Online, jan 2006. Version 1.1.

[GP06] GUCKENHEIMER, S. und J. J. PEREZ: *Software Enginnering with Microsoft Visual Studio Team System*. Addison Wesley, 2006.

[Gru05] GRUNDMANN, M.: *Prozessmodellintegration - Am Beispiel einer RUP-Erweiterung durch das V-Modell XT*. Diplomarbeit, Hochschule Reutlingen, 2005.

[GS04] GREENFIELD, J. und K. SHORT: *Software Factories*. Nummer ISBN: 978-0471202844. Wiley & Sons, 2004.

[GST06] GOSCHI, F., A. SAALMANN und M. TONNDORF: *Einführung eines organisationsspezifischen Vorgehensmodells nach dem V-Modell XT in einem RUP-geprägten Projektumfeld*. In: BISKUP, HUBERT und RALF KNEUPER (Herausgeber): *13. Workshop der Fachgruppe WI-VM der Gesellschaft für Informatik e.V. (GI) zum Thema: Nutzen und Nutzung von Vorgehensmodellen*. Shaker Verlag, mar 2006.

[Ham08] HAMMERSCHALL, U.: *Flexible Methodenintegration in Vorgehensmodelle – (to be published)*. Doktorarbeit, Technische Universität München, 2008.

[Hau06a] HAUMER, P.: *Eclipse Process Framework Composer – Part 1: Key Concepts*. 2006. Published online: http://www.eclipse.org/epf/general/EPFComposerOverviewPart1.pdf.

[Hau06b] HAUMER, P.: *Eclipse Process Framework Composer – Part 2: Authoring method content and processes*. 2006. Published online: http://www.eclipse.org/epf/general/EPFComposerOverviewPart2.pdf.

[HBH+06] HUANG, L., B. BOEHM, H. HU, J. GE, J. LÜ und C. QIAN: *Applying the Value/Petri process to ERP software development in China*. In: *ICSE '06: Proceeding of the 28th international conference on Software engineering*, Seiten 502–511, New York, NY, USA, 2006. ACM Press.

[HHV06] HESS, A., B. HUMM und M. VOSS: *Regeln für serviceorientierte Architekturen hoher Qualität*. Informatik Spektrum, 29(6):395–411, dec 2006.

[HJ04] HELLER, M. und D. JÄGER: *Graph-Based Tools for Distributed Cooperation in Dynamic Development Processes*. In: *Proceedings of 2nd International Workshop Applications of Graph Transformation with Industrial Relevance (AGTIVE'03)*, 2004.

[HKST07] HAMMERSCHALL, U., M. KUHRMANN, M. SIHLING und T. TERNITÉ: *Strategischer Vorteil - Das V-Modell XT an Unternehmen anpassen (Teil 2)*. iX - Magazin für professionelle Informationstechnik, (05/07):142–145, apr 2007. available at http://www.heise.de.

[HR02] HORN, E. und T. REINKE: *Softwarearchitektur und Softwarebauelemente*. Hanser, München, Wien, 2002.

[HRR98] HUBER, F., A. RAUSCH und B. RUMPE: *Component Interface Diagrams: Putting Components to Work*. Technischer Bericht TUM-I9831, Technische Universität München, Institut für Informatik, 1998.

[HS03] HARDING, K. und K. SZALL (Herausgeber): *Analyzing Requirements and Defining Microsoft. NET. MCSD Self-Paced Training Kit.: Analyzing Requirements and Deploying .NET Solution Architectures*. Nummer ISBN-13: 978-0735618947. Microsoft Press, 2003.

[HSW04] HELLER, M., A. SCHLEICHER und B. WESTFECHTEL: *Graph-Based Specification of a Management System for Evolving Development Processes*. In: *Proceedings of 2nd International Workshop Applications of Graph Transformation with Industrial Relevance (AGTIVE'03)*, 2004.

[Hum05] HUMPHREY, W. S.: *The Software Process: Global Goals*. In: LI, MINGSHU, BARRY BOEHM und LEON J. OSTERWEIL (Herausgeber): *Unifying the Soft-*

ware Process Spectrum, Band LNCS 3840 der Reihe *Lecture Notes in Computer Science*. Springer, 2005. International Software Process Workshop, SPW 2005, Beijing, China, May 25-27.

[HW06] HELLER, M. und R. WÖRZBERGER: *A Management System Supporting Interorganizational Cooperative Development Processes in Chemical Engineering*. In: *International Conference on Integrated Design and Process Technology (IDPT-2006)*, 2006.

[Int04] INTERNATIONAL, STANDISCH GROUP: *Chaos Reports*. Online, 2004. `http://www.standishgroup.com/chaos_resources/index.php`.

[Joi07] JOINT TECHNICAL COMMITTEE ISO/IEC JTC 1, SUBCOMMITTEE SC 7: *Software Engineering – Metamodel for Development Methodologies*. Nummer ISO/IEC 24744:2007. International Organization for Standardization, 2007.

[JRH+03] JECKLE, M., C. RUPP, J. HAHN, B. ZENGLER und S. QUEINS: *UML 2 glasklar*. Nummer ISBN: 978-3446225756. Hanser, Erste Auflage, 2003.

[JS02] JONES, L. G. und A. L. SOULE: *Software Process Improvement and Product Line Practice: Capability Maturity Model Integration (CMMI) and the Framework for Software Product Line Practice*. Technischer Bericht CMU/SEI-2002-TN-012, Software Engineering Institute, 2002.

[JS06] JIAMTHUBTHUGSIN, W. und D. SUTIVONG: *Portfolio management of software development projects using COCOMO II*. In: *ICSE '06: Proceeding of the 28th international conference on Software engineering*, Seiten 889–892, New York, NY, USA, 2006. ACM Press.

[KD07] KUHRMANN, M. und N. DIERNHOFER: *Software Lifecycle - Management und Entwicklung*. Schnell + Kompakt. entwickler.Press, erste Auflage, 2007.

[KDA07] KUHRMANN, MARCO, NORBERT DIERNHOFER und MARCUS ALT: *CollabXT – Prozessqualität durch Werkzeugunterstützung etablieren und steigern – Projektbericht*. In: KOSCHKE, RAINER, OTTHEIN HERZOG, KARL-HEINZ RÖDIGER und MARC RONTHALER (Herausgeber): *Informatik 2007 - Beiträge der 37. Jahrestagung der Gesellschaft für Informatik e.V.(GI),*, Band P-110 der Reihe *Lecture Notes in Informatics*, Seiten 309–311. Gesellschaft für Informatik, 2007.

[Kee04] KEETON, M.: *MSF, a pocket guide*. Nummer ISBN: 9-07721-216-7. Van Haren Publishing, 1 Auflage, jan 2004.

[Kel07] KELTER, U.: *Begriffliche Grundlagen von Modelldifferenzen*. In: *Vergleich und Versionierung von UML-Modellen (VVUM'07)*, 2007.

[Köh06] KÖHLER, P.: *Prince 2 - Das Projektmanagement-Framework*. Nummer ISBN: 3-540-29181-4 in *Xpert.press*. Springer, 2006.

[KHTS07] KUHRMANN, M., U. HAMMERSCHALL, T. TERNITÉ und M. SIHLING: *Individueller Standard - Das V-Modell XT an Unternehmen anpassen (Teil 1)*. iX - Magazin für professionelle Informationstechnik, (04/07):134–138, mar 2007.

[Kie06] KIEBUSCH, S.: *Ökonomisches Management von Prozess-Familien*. Objekt Spektrum, (März/April 2006 - 2):77–83, 2006.

[KK03] KROLL, P. und P. KRUCHTEN: *The Rational Unified Process Made Easy – A Practinioner's Guide to RUP*. Addison-Wesley, 2003.

[KK07] KALUS, G. und M. KUHRMANN: *CollabXT – Ein Ansatz zur automatischen Erzeugung von Kollaborationsportalen aus dem V-Modell XT*. In: *Proceedings des 14. Workshop der Fachgruppe WI-VM der Gesellschaft für Informatik e.V. (GI)*, Nummer ISBN: 978-3-8322-6111-5, Seiten 29–40. Shaker Verlag, apr 2007.

[KKNT04] KAZMAN, R., P. KRUCHTEN, R. L. NORD und J. E. TOMAYKO: *Integrating Software-Architecture-Centric Methods into the Rational Unified Process*. Technischer Bericht CMU/SEI-2004-TR-011, Software Engineering Institute, CMU, 2004.

[KMLO98] KNEUPER, R., G. MÜLLER-LUSCHNAT und A. OBERWEIS: *Vorgehensmodelle für die betriebliche Anwendungsentwicklung*. Nummer ISBN: 3-8154-2605-7. B.G. Teubner, Stuttgart – Leipzig, 1998.

[KMR06] KUHRMANN, M., J. MÜNCH und A. RAUSCH: *Metamodellbasierte Integration von Projekt Controlling Mechanismen in das V-Modell XT - Positionspapier*. In: HOCHBERGER, CHRISTIAN und RÜDIGER LISKOWSKY (Herausgeber): *Informatik 2006 - Beiträge der 36. Jahrestagung der Gesellschaft für Informatik e.V.(GI)*, Band P-94 der Reihe *Lecture Notes in Informatics*, Seiten 103–109. Gesellschaft für Informatik, oct 2006. available at http://www.gi-ev.de/LNI.

[Kön05a] KÖNIGS, A.: *Model Transformation with Triple Graph Grammars*. In: *Model Transformations in Practice Satellite Workshop of MODELS 2005, Montego Bay, Jamaica*, 2005.

[KN05b] KUHRMANN, M. und D. NIEBUHR: *Das V-Modell XT in der Praxis – IT-WiBe*. In: PETRASCH, ROLAND, REINHARD HÖHN, STEPHAN HÖPPNER, HERBERT WETZEL und MANUELA WIEMERS (Herausgeber): *12. Workshop der Fachgruppe WI-VM der Gesellschaft füur Informatik e.V. (GI) zum Thema: Entscheidungsfall Vorgehensmodelle*. Shaker Verlag, 2005.

[KNB05] KUHRMANN, M., D. NIEBUHR und C. BARTELT: *Anwendung des V-Modell XT - Stand und Erfahrungen aus der Pilotierungsphase*. In: CREMERS, ARMIN B., RAINER MANTHEY, PETER MARTINI und VOLKER STEINHAGE (Herausgeber): *Informatik 2005 - Beiträge der 35. Jahrestagung der Gesellschaft für Informatik e.V.(GI)*, Band P-67, Seiten 259–263. Gesellschaft für Informatik, sep 2005.

[Kne06] KNEUPER, R.: *CMMI*. Nummer ISBN: 978-3898643733. dpunkt.verlag, 2006.

[Kom06] KOMURO, M.: *Experiences of applying SPC techniques to software development processes*. In: *ICSE '06: Proceeding of the 28th international conference on Software engineering*, Seiten 577–584, New York, NY, USA, 2006. ACM Press.

[KP05] KLAPPHOLZ, D. und D. PORT: *M(in)BASE: An Upward-Tailorable Process Wrapper Framework for Identifying and Avoiding Model Clashes*. In: LI, MINGSHU, BARRY BOEHM und LEON J. OSTERWEIL (Herausgeber): *Unifying the Software Process Spectrum*, Band LNCS 3840 der Reihe *Lecture Notes in Computer Science*. Springer, 2005. International Software Process Workshop, SPW 2005, Beijing, China, May 25-27.

[KRS05] KINDLER, E., V. RUBIN und W. SCHÄFER: *Incremental Workflow Mining Based on Document Versioning Information*. In: LI, MINGSHU, BARRY BOEHM und LEON J. OSTERWEIL (Herausgeber): *Unifying the Software Process Spectrum*, Band LNCS 3840 der Reihe *Lecture Notes in Computer Science*. Springer, 2005. International Software Process Workshop, SPW 2005, Beijing, China, May 25-27.

[KS06] KÖNIGS, A. und A. SCHÜRR: *Tool Integration with Triple Graph Grammars - A Survey*. In: HECKEL, R. (Herausgeber): *Proceedings of the SegraVis School on Foundations of Visual Modelling Techniques*, Band 148 der Reihe *Electronic Notes in Theoretical Computer Science*, Seiten 113–150, Amsterdam, 2006. Elsevier Science Publ.

[KSG05] KÖPPEN, V., T. SCHWARZ und C. GERNERT: *Ein Transformationsansatz in der Geschäftsprozessmodellierung*. In: PETRASCH, ROLAND, REINHARD HÖHN, STEPHAN HÖPPNER, HERBERT WETZEL und MANUELA WIEMERS (Herausgeber): *12. Workshop der Fachgruppe WI-VM der Gesellschaft für Informatik e.V. (GI) zum Thema: Entscheidungsfall Vorgehensmodelle*. Shaker Verlag, 2005.

[KT06] KUHRMANN, M. und T. TERNITÉ: *Implementing the Microsoft Solutions Framework for Agile Sw-Development as Concrete Development-Method in the V-Modell XT*. International Transactions on Systems Science and Applicati-

ons (ITSSA), Special Issue Sections in ENASE 06, 1(ISSN 1751-147X):119 – 126, sep 2006.

[Kuh05] KUHRMANN, M.: *V-Modell XT Pilotprojekte bei der Bundeswehr*. Interner Projektbericht WEIT-III, EvaXT – unter Mitwirkung der Teammitglieder., nov 2005.

[Kuh06] KUHRMANN, M.: *Projektspezifische Anpassungen nach dem Tailoring des V-Modell XT durchführen*. In: BISKUP, HUBERT und RALF KNEUPER (Herausgeber): *13. Workshop der Fachgruppe WI-VM der Gesellschaft für Informatik e.V. (GI) zum Thema: Nutzen und Nutzung von Vorgehensmodellen*, Seiten 27–42. Shaker Verlag, mar 2006.

[Kuh07] KUHRMANN, M.: *Prozessintegration und -anpassung*. Forschungsbericht TUM-I0712, Technische Universität München, apr 2007.

[KWN05] KELTER, U., J. WEHREN und J. NIERE: *A Generic Difference Algorithm for UML Models*. In: *Proceedings of the Software Engineering 2005*, 2005.

[KZ96] KRUT, R. und N. ZALMAN: *Domain Analysis Workshop Report for the Automated Prompt & Response System Domain*. Technischer Bericht CMU/SEI-96-SR-001, Software Engineering Institute, 1996.

[LHSL06] LI, M., M. HUANG, F. SHU und J. LI: *A risk-driven method for eXtreme programming release planning*. In: *ICSE '06: Proceeding of the 28th international conference on Software engineering*, Seiten 423–430, New York, NY, USA, 2006. ACM Press.

[LKC06] LEE, J., J.-S. KIM und J.-H. CHO: *A series of development methodologies for a variety of systems in Korea*. In: *ICSE '06: Proceeding of the 28th international conference on Software engineering*, Seiten 612–615, New York, NY, USA, 2006. ACM Press.

[LKJ02] LEE, K., K. C. KANG und LEE J.: *Concepts and Guidelines of Feature Modeling for Product Line Software Engineering*. In: *Proceedings of the 7th International Conference on Software Reuse: Methods, Techniques, and Tools*, Band 2319 der Reihe *Lecture Notes In Computer Science*, Seiten 62 – 77, 2002.

[LVD06] LATOZA, T. D., G. VENOLIA und R. DELINE: *Maintaining mental models: a study of developer work habits*. In: *ICSE '06: Proceeding of the 28th international conference on Software engineering*, Seiten 492–501, New York, NY, USA, 2006. ACM Press.

[MBR06a] MIKLIZ, T., P. BUXMANN und A. RÖDDIGER: *Standortplanung für Anbieter von IT-Services*. Wirtschaftsinformatik, 48(6):397–406, dec 2006.

[MBR06b] MUTSCHLER, B., J. BUMILLER und M. REICHERT: *Designing an economic-driven evaluation framework for process-oriented software technologies*. In: *ICSE '06: Proceeding of the 28th international conference on Software engineering*, Seiten 885–888, New York, NY, USA, 2006. ACM Press.

[Mün99] MÜNCH, J.: *Anpassung von Vorgehensmodellen im Rahmen ingenieurmäßiger Softwarequalitätssicherung*. In: *6. Workshop der Fachgruppe WI-VM der Gesellschaft für Informatik e.V. (GI) zum Thema: „Vorgehensmodelle, Prozeßverbesserung und Qualitätsmanagement"*, 1999.

[Mün01] MÜNCH, J.: *Muster-basierte Erstellung von Software-Projektplänen*. Doktorarbeit, Technische Universität Kaiserslautern, 2001. PhD Theses in Experimental Software Engineering, Vol. 10, ISBN: 3-8167-6207-7, Fraunhofer IRB Verlag.

[Mün04] MÜNCH, J.: *Transformation-based Creation of Custom-tailored Software Process Models*. In: *Proceedings of the 5th International Workshop on Software Process Simulation and Modeling (ProSim 2004)*, Seiten 50–56, 2004.

[Mün05] MÜNCH, J.: *Goal-oriented Composition of Software Process Patterns*. In: *Proceedings of the 6th International Workshop on Software Process Simulation and Modeling (ProSim 2005)*, Seiten 164–168, 2005.

[MRD+04] MEISINGER, MICHAEL, ANDREAS RAUSCH, MARTIN DEUBLER, MICHAEL GNATZ, ULRIKE HAMMERSCHALL, INGA KÜFFER und SASCHA VOGEL:

Das V Modell 200x - ein modulares Vorgehensmodell. In: KNEUPER, RALF, ROLAND PETRASCH und MANUELA WIEMERS (Herausgeber): *11. Workshop der Fachgruppe WI-VM der Gesellschaft für Informatik e.V. (GI) zur Akzeptanz von Vorgehensmodellen*. Shaker Verlag, 2004.

[MRS06] MEISINGER, M., A. RAUSCH und M. SIHLING: *4everedit - Team-based Process Documentation Management*. Software Process Improvement and Practice, 2006.

[MSV97] MÜNCH, J., M. SCHMITZ und M. VERLAGE: *Tailoring großer Prozessmodelle auf der basis von MVP-L*. In: *4. Workshop der Fachgruppe WI-VM der Gesellschaft für Informatik e.V. (GI) zum Thema: „Vorgehensmodelle - Einführung, betrieblicher Einsatz, Werkzeug-Unterstützung und Migration"*, 1997.

[NK05] NIEBUHR, D. und M. KUHRMANN: *Projekt-Tüv*. iX - Magazin für professionelle Informationstechnik, 1(1/2006):126–129, dec 2005.

[NR69] NAUR, P. und B. RANDELL (Herausgeber): *NATO Konferenz: Working Conference on Software Engineering*. NATO, Oct. 1969. 7–11 Oct. 1968.

[NR05] NEJMEH, B. A. und W. E. RIDDLE: *A Framework for Coping with Process Evolution*. In: LI, MINGSHU, BARRY BOEHM und LEON J. OSTERWEIL (Herausgeber): *Unifying the Software Process Spectrum*, Band LNCS 3840 der Reihe *Lecture Notes in Computer Science*. Springer, 2005. International Software Process Workshop, SPW 2005, Beijing, China, May 25-27.

[OIM06] OGASAWARA, H., T. ISHIKAWA und T. MORIYA: *Practical approach to development of SPI activities in a large organization: Toshiba's SPI history since 2000*. In: *ICSE '06: Proceeding of the 28th international conference on Software engineering*, Seiten 595–599, New York, NY, USA, 2006. ACM Press.

[OK02] OHST, D. und U. KELTER: *A Fine-grained Version and Configuration Model in Analysis and Design*. In: *International Conference on Software Maintenance 2002 (ICSM2002)*, 2002.

[OMG05] OMG: *Software Process Engineering Metamodel Specification*. Technischer Bericht, Object Management Group, 2005.

[OMG06a] OMG: *Meta Object Facility (MOF) Core Specification –Version 2.0*. Meta Object Facility (MOF) Core Specification, Object Management Group, 2006.

[OMG06b] OMG: *Object Constraint Language (OCL) – Version 2.0*. OMG Available Specification, Object Managment Group, 2006.

[OMG07a] OMG: *Unified Modeling Language (UML): Infrastructure – Version 2.1.1*. Technischer Bericht, Object Management Group, 2007.

[OMG07b] OMG: *Unified Modeling Language (UML): Superstructure – Version 2.1.1*. Technischer Bericht, Object Management Group, 2007.

[OSKZ06] OESTEREICH, B., C. SCHRÖDER, M. KLINK und G. ZOCKOLL: *OEP - OOSE Engineering Process*. Nummer ISBN: 3-89864-407-3. dpunkt.verlag, 2006.

[Ost05] OSTERWEIL, L. J.: *Unifying Microprocess and Macroprocess Research*. In: LI, MINGSHU, BARRY BOEHM und LEON J. OSTERWEIL (Herausgeber): *Unifying the Software Process Spectrum*, Band LNCS 3840 der Reihe *Lecture Notes in Computer Science*. Springer, 2005. International Software Process Workshop, SPW 2005, Beijing, China, May 25-27.

[Pat06] PATZAK, M.: *Effektiver Umgang mit Änderungswünschen in Softwareprojekten*. Objekt Spektrum, (Januar/Februar 2007 - 1):32–35, dec 2006.

[PB05] POENSGEN, B. und B. BERTRAM: *Function-Point-Analyse - Ein Praxisbuch*. dpunkt.verlag, 2005.

[PPO05] PODOROZHNY, R. M., D. E. PERRY und L. J. OSTERWEIL: *Automatically Analyzing Software Processes: Experience Report*. In: LI, MINGSHU, BARRY BOEHM und LEON J. OSTERWEIL (Herausgeber): *Unifying the Software Process Spectrum*, Band LNCS 3840 der Reihe *Lecture Notes in Computer Science*. Springer, 2005. International Software Process Workshop, SPW 2005, Beijing, China, May 25-27.

[Rau01a] RAUSCH, A.: *Towards a Software Architecture Specification Language based on UML and OCL*. In: *Proceedings of the Workshop on Describing Software Architecture with UML, 23rd International Conference on Software Engineering*, 2001.

[Rau01b] RAUSCH, A.: *Using XML/XMI for Tool Supported Evolution of UML Models*. In: *Proceedings of the Thirty-Four Annual Hawaii International Conference on System Sciences*, 2001.

[RB02] RAUSCH, A. und M. BROY: *Evolutionary Design of Software Architectures*. In: *Technology for Evolutionary Software Development, a symposium organised by NATO's Research & Technology Organization (RTO)*, 2002.

[RBTK05] RAUSCH, A., C. BARTELT, T. TERNITÉ und M. KUHRMANN: *The V-Modell XT Applied – Model-Driven and Document-Centric Development*. In: *3rd World Congress for Software Quality, VOLUME III, Online Supplement*, Nummer 3-9809145-3-4, Seiten 131 – 138. International Software Quality Institute GmbH, sep 2005. available at http://www.isqi.org/isqi/deu/conf/wcsq/3/proc.php.

[RH06] RAUSCH, A., HÖHN R. BROY M. BERGNER K. und S. HÖPPNER: *Das V-Modell XT*. eXamen.press. Springer, 2006.

[Rie05] RIEGG, T.: *Analyse der Anwendbarkeit des V-Modell XT bei kleinen IT-Vorhaben*. Diplomarbeit, Berufsakademie Heidenheim, 2005.

[Riz06] RIZZO, S.: *Agiles Projekt- und Anforderungsmanagement: Der Live-Ansatz*. Objekt Spektrum, (März/April 2006 - 2):38–44, 2006.

[RK02] RÖTSCHKE, T. und R. KRIKHAAR: *Architecture Analysis Tools to Support Evolution of Large Industrial Systems*. In: *Proc. IEEE International Conference on Software Maintenance (ICSM 2002)*, Seiten 182–193, 10 2002.

[Rom05] ROMBACH, D.: *Integrated Software Process and Product Lines*. In: LI, MINGSHU, BARRY BOEHM und LEON J. OSTERWEIL (Herausgeber): *Unifying the Software Process Spectrum, International Software Process Workshop, SPW 2005, Beijing, China, May 25-27*, Lecture Notes in Computer Science, Seiten 83–90. Springer, 2005.

[RS95] REKERS, J. und A. SCHÜRR: *A Graph Grammar Approach to Graphical Parsing (2)*. In: NIJHOLT, A., G. SCOLLO und R. STEETSKAMP (Herausgeber): *Proc. Twente Workshop on Language Technology 10 joint with First AMAST Workshop on Language Processing*, Seiten 163–172, Enschede, 12 1995. Twente University.

[RS06] RÖTSCHKE, T. und A. SCHÜRR: *Temporal Graph Queries to Support Software Evolution*. In: CORRADINI, A., H. EHRIG, U. MONTANARI, L. RIBEIRO und G. ROZENBERG (Herausgeber): *International Conference on Graph Transformations*, Band 4178 der Reihe *Lecture Notes in Computer Science (LNCS)*, Seiten 291–305, Heidelberg, 2006. Springer Verlag.

[RT98] RUMPE, B. und V. THURNER: *Refining Business Processes*. In: H. KILOV, B. RUMPE und I. SIMMONDS (Herausgeber): *7th OOPSLA Workshop on Behavioral Semantics of OO Business and System Specifications*, 1998.

[Röt04] RÖTSCHKE, T.: *Adding Pluggable Meta Models to FUJABA*. In: GIESE, H., A. SCHÜRR und A. ZÜNDORF (Herausgeber): *Proc. 2nd International Fujaba Days*, Band tr-ri-04-253, Seiten 57–61. Universität Paderborn, 2004.

[SC06] SHEN, B. und C. CHEN: *The Design of a Flexible Software Process Language*. In: WANG, QING, DIETMAR PFAHL, DAVID M. RAFFO und PAUL WERNICK (Herausgeber): *Software Process Change, International Software Process Workshop and International Workshop on Software Process Simulation and Modelling, SPW 2006, Shanghai, China, May*, Lecture Notes in Computer Science, Seiten 186–194. Springer, 2006.

[Sch01] SCHÜRR, A.: *A New Type Checking Approach for OCL Version 2.0?* In: CLARK, T. und J. WARMER (Herausgeber): *Advances in Object Modeling with OCL*, Seiten 19–40. Springer Verlag, Heidelberg, 2001.

[Sch02] SCHWERIN, W.: *Models of Systems, Work Products, and Notations*. In: *Proceedings of Intl. Workshop on Model Engineering, ECOOP'2000, Cannes France, June 2000*, 2002.

[Sch04] SCHWABER, K.: *Agile Project Management with Scrum*. Microsoft Press, 2004.

[sei] *Software Product Lines*. Online: `http://www.sei.cmu.edu/productlines`. Visit: 2007-07-10.

[SG96] SHAW, M. und D. GARLAN: *Software Architecture – Perspectives on an emerging discipline*. Prentice Hall, 1996.

[Sie04] SIEDERSLEBEN, J.: *Moderne Software-Architektur – Umsichtig planen, robust bauen mit Quasar*. Nummer ISBN-13: 978-3898642927. dpunkt.verlag, 2004.

[SM06a] SOTO, M. und J. MÜNCH: *The DeltaProcess Approach for Analyzing Process Differences and Evolution*. Technischer Bericht IESE-Report No. 164.06/E, Fraunhofer Institut Experimentelles Software Engineering, October 2006. Submitted for Publication to Software Process: Improvement and Practice.

[SM06b] SOTO, M. und J. MÜNCH: *Pattern-Based Identification of Differences in Process Models*. Technischer Bericht IESE-Report No. 174.06/E, Fraunhofer Institut Experimentelles Software Engineering, November 2006. Submitted for Publication at SE 2007.

[SM06c] SOTO, M. und J. MÜNCH: *Process Model Difference Analysis for Supporting Process Evolution.* In: *EuroSPI*, Seiten 123–134, 2006.

[SMP06] SANGWAN, R., N. MULLICK und D. J. PAULISH: *Global Software Development Handbook (Applied Software Engineering)*. Nummer ISBN: 978-0849393846. Auerbach Publishers Inc., 2006.

[SRK06] SNELTING, G., T. ROBSCHINK und J. KRINKE: *Efficient Path Conditions in Dependence Graphs for Software Safety Analysis*. ACM Transactions on Software Engineering and Methodology, 15(4):411–457, oct 2006.

[SW00] SCHÜRR, A. und A. WINTER: *UML Packages for Programmed Graph Rewriting Systems*. In: *Proc. TAGT'98 – Theory and Application of Graph Transformations, Paderborn, Germany*, Band 1764 der Reihe *Lecture Notes in Computer Science (LNCS)*, Seiten 396–409, Heidelberg, 2000. Springer Verlag.

[SW06] STRUCKMANN, W und D. WÄTJEN: *Mathematik für Informatiker - Grundlagen und Anwendungen*. Elsevier Sprektrum Akademischer Verlag, 2006.

[TBWK07] TREUDE, C., S. BERLIK, S. WENZEL und U. KELTER: *Difference Computation of Large Models*. In: *6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 2007.

[Tho06] THOMAS, O.: *Ein Ordnungsrahmen für das Vorgehen zum Management von Referenzmodellen*. In: BISKUP, HUBERT und RALF KNEUPER (Herausgeber): *13. Workshop der Fachgruppe WI-VM der Gesellschaft für Informatik e.V. (GI) zum Thema: Nutzen und Nutzung von Vorgehensmodellen*. Shaker Verlag, mar 2006.

[Thu04] THURNER, V.: *Formal fundierte Modellierung von Geschäftsprozessen*. Doktorarbeit, Technischen Universität München, 2004. Logos Verlag Berlin, ISBN: 3-8325-0683-7.

[Tur06] TURNER, M.: *Microsoft Solutions Framework Essentials*. Nummer ISBN: 0-7356-2353-8. Microsoft Press, 2006.

[VAC+05] VOGEL, O., I. ARNOLD, A. CHUGHTAI, E. IHLER, U. MEHLIG, T. NEUMANN, M. VÖLTER und U. ZDUN: *Software Architektur: Grundlagen – Konzepte – Praxis*. Elsevier Spektrum Akademischer Verlag, 2005.

[vmx] *Das V-Modell XT Online Portal*. Online, `http://www.v-modell-xt.de/`.

[Wes01] WESTFECHTEL, B.: *Ein graphbasiertes Managementsystem für dynamische Entwicklungsprozesse*. Informatik Forschung und Entwicklung, 16:125–144, 2001.

[WG06] WEHRMANN, A. und D. GULL: *Ein COCOMO-basierter Ansatz zur Entscheidungsunterstützung bim Offshoring von Softwareentwicklungsprojekten*. Wirtschaftsinformatik, 48(6):407–417, dec 2006.

[WJG+06] WANG, Q., N. JIANG, L. GOU, X. LIU, M. LI und Y. WANG: *BSR: a statistic-based approach for establishing and refining software process performance baseline*. In: *ICSE '06: Proceeding of the 28th international conference on Software engineering*, Seiten 585–594, New York, NY, USA, 2006. ACM Press.

[WO06] WEILKIENS, T. und B. OESTEREICH: *UML 2 Zertifizierung: Fundamental, Intermediate und Advanced*. Nummer ISBN: 3-89864-424-3. dpunkt.verlag, Erste Auflage, 2006.

[YCC06] YE, C., S. C. CHEUNG und W. K. CHAN: *Publishing and composition of atomicity-equivalent services for B2B collaboration*. In: *ICSE '06: Proceeding of the 28th international conference on Software engineering*, Seiten 351–360, New York, NY, USA, 2006. ACM Press.

[zM06] ÖZTÜRK, M. und A. MEYN: *Function-Point-Messungen und standardisierte Aufwandschätzverfahren*. Objekt Spektrum, (September/Oktober 2006 - 5):76–82, 2006.