Operationalisierung des Projektcontrollings

Edward Fischer

Software Systems Engineering
TU Clausthal
Julius-Albert-Str. 4
38678 Clausthal-Zellerfeld
ef@tu-clausthal.de

Abstract: Der Einsatz eines Vorgehensmodells erfordert die stetige Anwendung der darin enthaltenen Regeln auf den aktuellen Projektzustand. Dies geschieht aufgrund fehlender Konzepte bisher per Hand – mit unakzeptablen Zeitaufwand. Dieser Beitrag stellt ein Lösungsansatz vor, mit dem automatisiert beantwortet werden kann, welche konkreten Dokumente (Produktexemplare) in einem Projekt jeweils als nächstes zu erstellen sind, und wie man dabei mit Veränderungen der Inhalte dieser Dokumente (Produktexemplarinhalte) umgeht.

1 Einleitung

1.1 Motivation

Vorgehensmodelle definieren wer, was und wann in einem Projekt zu tun hat. In aller Regel spielt der Projektleiter die zentrale Rolle bei der Projektdurchführung, dessen Aufgaben unter anderem die „Überwachung der Termine, des Erfüllungsgrads der Arbeitspakete und des Mittelabflusses“ umfassen. Konkret äußert sich dies in seiner Verantwortung für Produkte wie „Projektplan“, „Projektstatusbericht“ und „Schätzung“. Obgleich die zitierten Begrifflichkeiten dem V-Modell XT entstammen, sind sie in ähnlicher Form auch in anderen Vorgehensmodellen wie HERMES oder RUP enthalten. In jedem Falle aber muss der Projektleiter das gesamte Vorgehensmodell hinsichtlich zu berücksichtigender Produkte und dabei geltender Abhängigkeiten kennen. Eine solche Abhängigkeit kann bspw. festlegen, dass Produktexemplare zur Qualitätssicherung nur für diejenigen Produkttypen zu erzeugen sind, die im QS-Handbuch als zu prüfen aufgeführt sind – was sich maßgeblich auf die Planung und letztlich auch auf den Projektstatus auswirkt. Solche Abhängigkeiten können komplexerer Natur sein, so dass bspw. Spezifikationen nur für solche Teile eines Systems anzulegen sind, die als kritisch eingestuft wurden. Im Rahmen der Planung muss der Projektleiter die Abhängigkeiten auswerten, um entscheiden zu können, welche konkreten Produkte noch zu erstellen sind. Dabei muss er sowohl in die Produkt-exemplarinhalte hineinschauen als auch mit Veränderungen derselben umgehen. Dies alles erfolgt bislang größtenteils per Hand, was hinsichtlich des Umfangs an Produkttypen, Abhängigkeiten und der daraus erzeugten Produktexemplaren einen kostspieligen Aufwand als Folge hat.

1.2 Zielsetzung

Ein lohnendes Ziel ist es, den Projektleiter durch eine geeignete Operationalisierung des verwendeten Vorgehensmodells bei den mechanischen Aufgaben zu unterstützen. Gleichwohl geht es dabei jedoch nicht um eine Ausformalisierung eines Vorgehensmodells bis ins kleinste Detail. Gegenstand dieses Beitrags ist ein methodenneutrales Vorgehensmodell (konkret das V-Modell XT), wobei lediglich die Begriffe und Abhängigkeiten formalisiert werden, die im Vorgehensmodell bereits enthalten sind. Im Vordergrund stehen dabei die algorithmischen Fragen, wie festgestellt werden kann…

A) …ob Produktexemplarinhalte zum Vorgehensmodell konform sind? (eine Grundvoraussetzung für eine automatisierte Auswertung von Abhängigkeiten)
B) …welche Produktexemplare jeweils als nächstes zu erstellen sind?
C) …wie mit Änderungen von Produktexemplarinhalten – hinsichtlich der Produktabhängigkeiten – umzugehen ist?

2 Problemdefinition und verwandte Arbeiten

Bisherige Ansätze zur Unterstützung des Projektleiters bieten bereits eine indirekte Verwaltung von Produktexemplaren – indem diese aus den Inhalten der angewandten Methodik herausgeneriert werden [BTZ05,RBT+05]. Allerdings handelt es sich dabei um eine nachträgliche Dokumentation des bereits erledigten. Eine Planung, der jeweils als nächstes anzugehenden Arbeiten im Vorfeld, wird nicht unterstützt. Der Umgang mit Änderungen wird dabei ebenfalls ausgeklammert. Im Folgenden werden die oben aufgeworfenen, algorithmischen Fragestellungen anhand von Beispielen aus dem V-Modell XT näher erläutert.

2.1 Validitätsprüfung von Produktexemplarinhalten

Abbildung 1 zeigt das in der Motivation eingeführte Beispiel eines QS-Handbuches. Die Inhalte eines davon abgeleiteten Produktexemplars sind Referenzen auf Elemente des Vorgehensmodells. Eine solche Referenz durchbricht die Grenze der Modellebenen, und kann damit weder als Link noch als Assoziation im Rahmen des MOF[1]-Modellierungs-Frameworks typsicher dargestellt werden. Damit scheidet nicht nur UML aus, sondern sämtliche Modellierungsansätze, die auf einer strikten[2] Metamodellierung beruhen. Bei Programmiersprachen ist die Forderung nach Referenzen, die Ebenengrenzen durchbrechen dürfen, als Reflection [DM07] bekannt, offen bleibt aber, wie man die Mächtigkeit dieser Konzepte auf ein für Vorgehensmodelle erträgliches Maß beschränkt und mit der Metamodellierung in Einklang bringt.

[1] Meta Object Facility - http://www.omg.org/mof/
[2] Ebenengrenzen dürfen ausschließlich von Instanzierungsbeziehungen überquert werden [küh06]

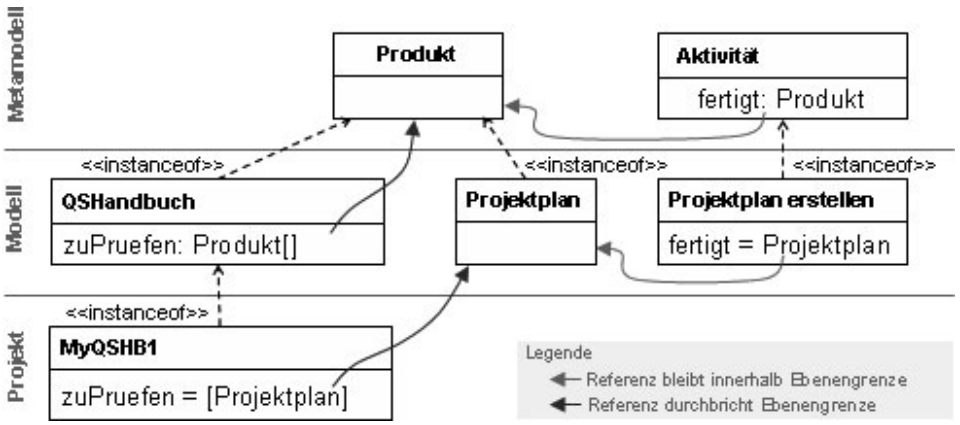

Abbildung 1: QS-Handbuch und Referenzen die Ebenengrenzen überqueren

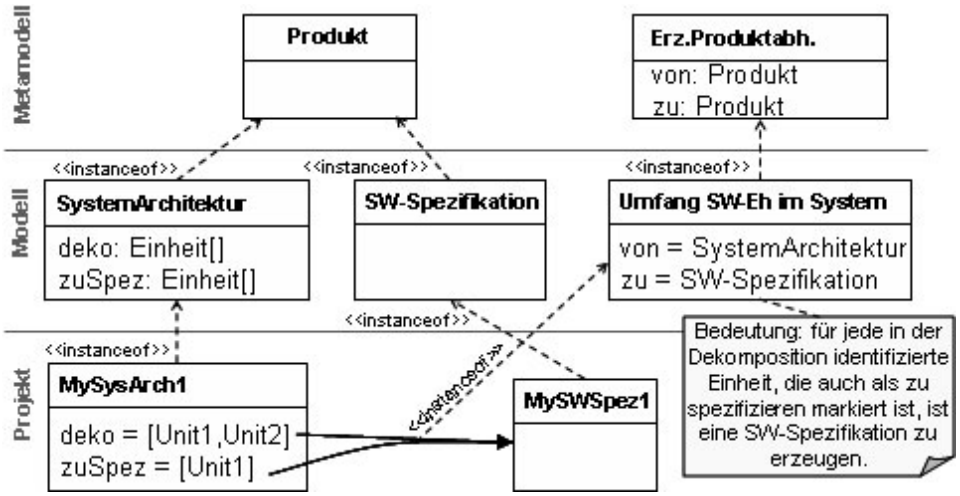

Abbildung 2: Produktabhängigkeitsexemplar

2.2 Folgen sich ändernder Produktexemplarinhalte

Abbildung 2 zeigt das ebenfalls in der Motivation angeschnittene Beispiel einer Systemarchitektur, für die für jede in der Dekomposition identifizierten Einheit (Teil eines Systems) eine SW-Spezifikation zu erstellen ist. Dies allerdings nur dann, wenn die entspr. Einheit in die „zu spezifizierenden" aufgenommen wurde. Sind die Spezifikationen einmal erzeugt, ergibt sich die Frage, wie mit Änderungen an den Inhalten der Systemarchitektur umzugehen ist. Werden neue Einheiten identifiziert oder bereits vorhandene doch als zu spezifizieren eingestuft, muss automatisch ermittelt werden können, welche Produkte schon erzeugt wurden und welche noch fehlen.

3 Lösungsansatz

In diesem Abschnitt wird ein Lösungsansatz vorgestellt, der eine Antwort auf die aufgezeigten Probleme bietet. Es handelt sich dabei um eine Modellierungssprache, die mit ihrem strukturellen Anteil sowohl das (Vorgehens-) Metamodell, das (Vorgehens-) Modell als auch die Projektebene umfasst, und somit die Frage nach validen Produktinhalten beantworten kann. Mit dem dynamischen Anteil der Modellierungssprache können Produktabhängigkeiten erklärt, ausgewertet und im Kontext sich ändernder Produktinhalte verwaltet werden. Für eine kurze und präzise Beschreibung der Struktur und Dynamik der wird eine algebraische Notation verwendet.

3.1 Struktur

Abbildung 3 fasst die Formalisierung der Struktur zusammen. Die zentrale Struktureinheit ist das **Element**. Jedes Element hat u.a. eine ID und besitzt eine Menge von **Eigenschaftsbeschreibungen** durch die es charakterisiert wird: ein Term wie

slot(zuPruefen, Projektplan)

drückt aus, dass das charakterisierte Element ein Schlüssel-Wert-Paar besitzt, wobei *zuPruefen* der Schlüssel, und *Projektplan* der Wert (hier: eine ID eines anderen Elements, wodurch ein Slot gleichzeitig die Funktionalität eines Links trägt) ist.

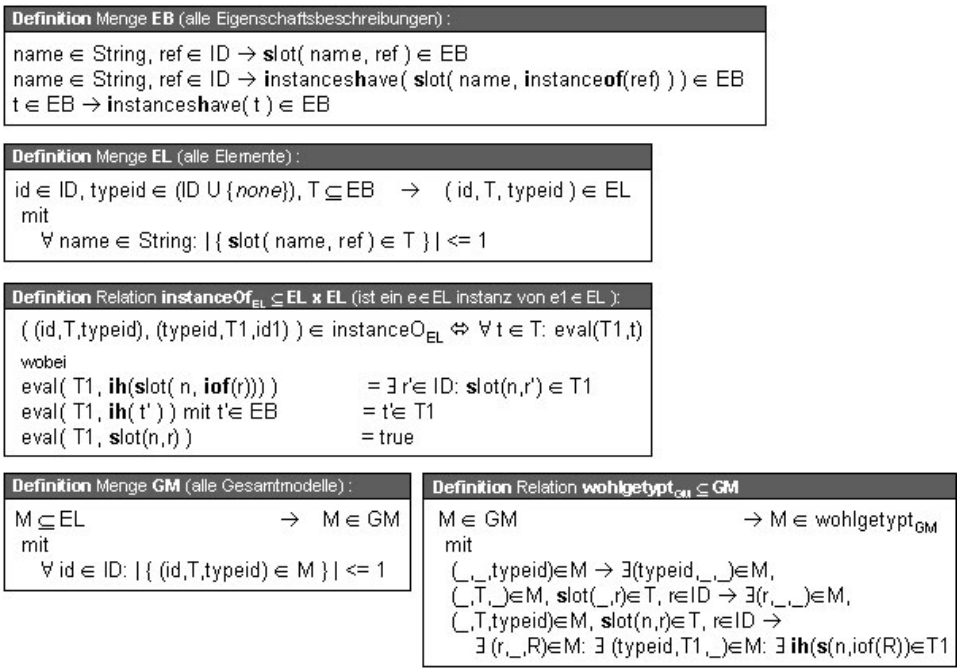

Abbildung 3: Formalisierung der Struktur

Eine Eigenschaftsbeschreibung wie

instances**h**ave(**s**lot(zuPruefen, Projektplan))

sagt aus, dass alle Instanzen des charakterisierten Elements das entsprechende Schlüssel-Wert-Paar besitzen. Schließlich legt ein Term wie

instances**h**ave(**s**lot(zuPruefen, **i**nstance**of**(Produkt))

fest, dass alle Instanzen des charakterisierten Elements als Wert des Schlüssels *zuPruefen* eine ID eines Elements haben müssen, welches Instanz von dem Element mit der ID *Produkt* ist. Da das referenzierte Element sich in einer beliebigen Ebene befinden kann, müssen i.Allg. alle Ebenen betrachtet werden. Zur einfacheren Darstellung werden in der Formalisierung sämtliche Ebenen zum **Gesamtmodell** verschmolzen. Ein Produktexemplar *e* ist valid bezüglich eines Gesamtmodells *GM*, sofern {e}∩GM zur **wohlgetypt**-Relation dazugehört.

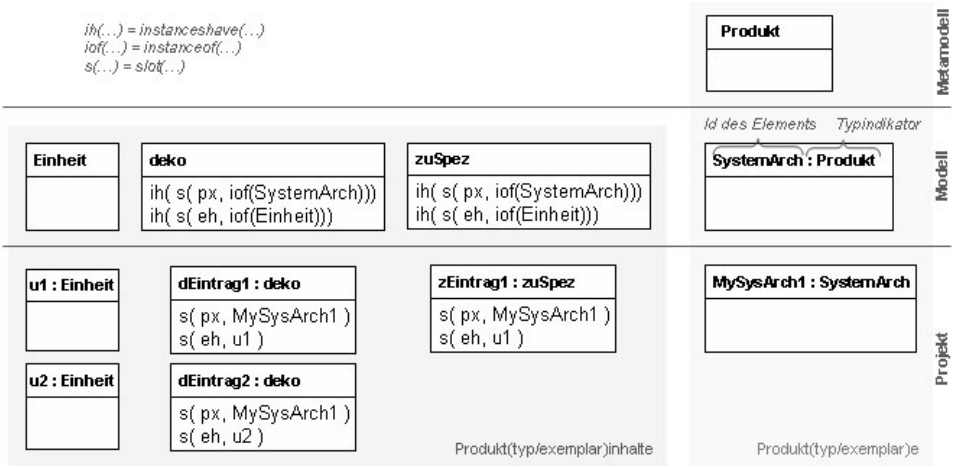

Abbildung 4: Anwendungsbeispiel für die Dynamik

3.2 Dynamik

Der Aufbau der Dynamik basiert auf Induktion. Eine Projektdurchführung beginnt mit dem Induktionsanfang – dem erzeugen initialer Produktexemplare. Produktabhängigkeiten wirken anschließend als Induktionsschritte, durch die jeweils neue Produktexemplare erzeugt werden, die ihrerseits den Induktionsschritt auslösen. Ein Induktionsschritt kann dabei als eine Implikation „T → P“ verstanden werden, wobei die linke Seite als „**Trigger**“, die rechte als „**Phantombild**“ bezeichnet sei. Ein Trigger beschreibt dabei eine Projektkonstellation wie bspw.: „es gibt eine Systemarchitektur A,

in deren Dekomposition ein Eintrag D gibt, der die Einheit E identifiziert, wobei diese Einheit auch im Thema zu spezifizierende Systemelemente durch den Eintrag Z als zu spezifizieren markiert ist“. Ein Phantombild beschreibt dagegen genau ein neu zu erzeugendes Produktexemplar, durch Angabe von Produkttyp sowie Schlüssel-Wert-Belegungen. Diese Beschreibung ist jedoch nicht vollständig, sondern stellt nur die Mindestanforderungen gemäß dem Vorgehensmodell dar. Abbildung 4 zeigt die Integration des Induktionskonzepts (formalisiert in Abb.5) mit der Struktur aus 3.1. Die Funktion **match** gibt dabei an, ob ein Trigger auslöst. Für jede Variablenbelegung, die zur Auslösung führt, ist jeweils ein Phantombild zu erzeugen. Dieser Vorgang wird im Folgenden als **Instanzierung einer Produktabhängigkeit** bezeichnet.

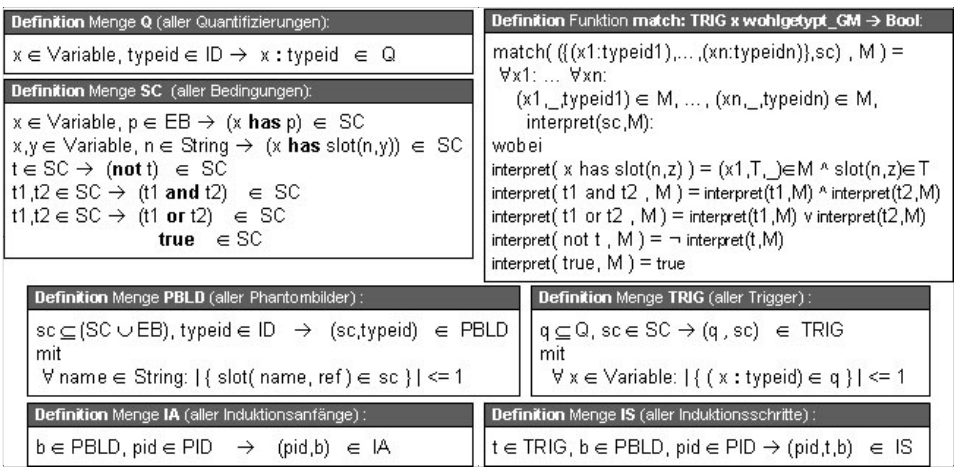

Abbildung 5: Formalisierung der Dynamik

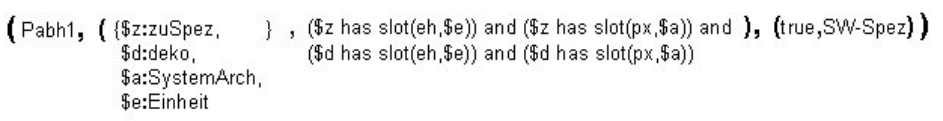

Abbildung 6: Beispiel einer Produktabhängigkeit

Um Änderungen von Produktinhalten zu berücksichtigen, ist es i.Allg. erforderlich, die gesamte Berechnung der Induktionskette von Beginn an zu wiederholen. Problematisch ist dies, dass zu den dabei berechneten Phantombildern nicht sofort neue Produktexemplare erstellt werden können – weil es diese evtl. bereits schon gibt. Folglich muss die Frage beantwortet werden können, ob ein bereits existierendes Produktexemplar auf ein Phantombild passt, so dass für jenes Phantombild kein neues Produktexemplar zu erzeugen ist. Da ein Phantombild nur bestimmte Eigenschaften eines Dokumentes beschreibt (die ID des Produktexemplars gehört da bspw. nicht dazu), ist die Antwort dazu nicht trivial. Erfreulicherweise kann man sich der Beantwortung dieser Frage entziehen, wenn Instanzierungen der Produktabhängigkeiten explizit verwaltet werden. Die Überlegung dabei ist, dass nur dann neue Produktexemplare zu erzeugen sind, wenn

es für einen Trigger eine neue, auslösende Variablenbelegung gibt. Sollte also bei einer Neuberechnung der Induktionskette eine auslösende Variablenbelegung gefunden werden, die bei einer vorherigen Berechnung bekannt war, ist entsprechend kein neues Produktexemplar zu erstellen. Sollen auch nicht mehr relevante Produktexemplare gefunden werden (Triggerbedingung nicht mehr erfüllt), ist bei einer Instazierung einer Produktabhängigkeit das dabei erzeugte Produktexemplar zu hinterlegen.

4 Zusammenfassung und Ausblick

In diesem Beitrag wurde ein Ansatz zur Integration von Metamodellierung, Reflection, Induktion als Beschreibungsgrundlage für die Dynamik eines Projektes und dem Umgang mit Veränderungen dabei, gezeigt. Dadurch kann automatisiert und im Kontext von Veränderungen beantwortet werden, ob Produktinhalte konform zum Vorgehensmodell sind und welche Produktexemplare jeweils als nächstes zu erstellen sind. Für die weitere Arbeit steht die Integration von inhaltlichen Produktabhängigkeiten und versionserzeugenden Produktabhängigkeiten (Bearbeitungszustandsmodell) im Focus. Erstere können als Konsistenzbedingungen für einen Projektzustand verstanden werden, während zweitere als eine Verfeinerung der Produktabhängigkeiten anzusehen sind – in Richtung einzelner Versionen eines Produktexemplars.

Literaturverzeichnis

[BTZ05] Christian Bartelt, Thomas Ternité, Matthias Zieger. *Modellbasierte Entwicklung mit dem V-Modell XT*. OBJEKTspektrum, 5, Mai 2005.

[Gna05] Michael Andreas Josef Gnatz. *Vom Vorgehensmodell zum Projektplan*. Dissertation, Technische Universität München, 2005

[RBT+05] Andreas Rausch, Christian Bartelt, Thomas Ternité, Marco Kuhrmann. *The V-Modell XT Applied - Model-Driven and Document-Centric Development*. In 3rd World Congress for Software Quality, Volume III, Online Supplement ISBN Nr. 3-9809145-3-4, 2005.

[Küh06] Thomas Kühne. *Matters of (Meta-)Modeling*. Journal on Software and Systems Modeling, Volume 5, Number 4, 369–385, December 2006.

[DM07] François-Nicola Demers, Jacques Malenfant. *Reflection in logic, functional and object-oriented programming: a Short Comparative Study*, 1995