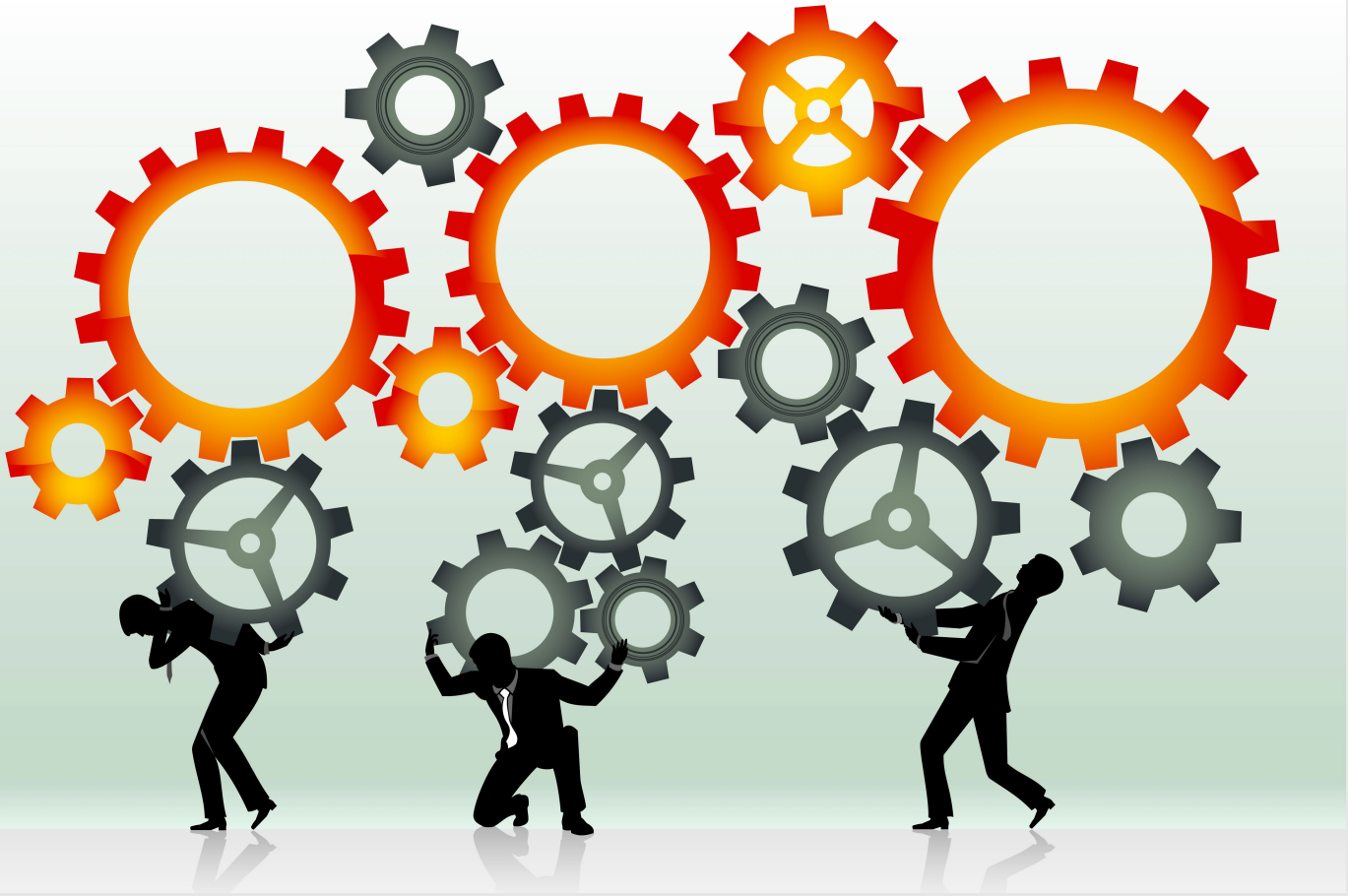
TU Clausthal



Michael Deynet

Kontextsensitiv lernende Sequenzvorhersage

zur erfahrungsbasierten Unterstützung bei der Softwareprozessausführung

SSE-Dissertation 7

Software
Systems
Engineering

Institut für Informatik
Lehrstuhl von Prof. Dr. Andreas Rausch

Kontextsensitiv lernende Sequenzvorhersage zur erfahrungsbasierten Unterstützung bei der Softwareprozessausführung

Dissertation

zur Erlangung des Grades eines
Doktors der Naturwissenschaften

vorgelegt von
Michael Deynet
aus Birkenfeld

genehmigt von der Fakultät für
Mathematik/Informatik und Maschinenbau
der Technischen Universität Clausthal

Tag der mündlichen Prüfung
07. Dezember 2012

Vorsitzender der Promotionskommission:
Prof. Dr. Jürgen Dix

Hauptberichterstatter:
Prof. Dr. Andreas Rausch

Berichterstatter:
Prof. Dr. Dieter Rombach

Titelbild: ©iStockphoto.com/draco77

D104
Dissertation Clausthal, SSE-Dissertation 7, 2012

Für meine Familie

KURZFASSUNG

Die Komplexität von IT-Systemen steigt stetig. Es ist zu beobachten, dass hierdurch deren Entwicklung mit „klassischen“ Verfahren des Software-Engineering immer schwerer beherrschbar wird. Deshalb ist derzeit ein Trend zu beobachten, bei dem versucht wird, Methoden des Software-Engineering zu verbessern, indem Verfahren des Maschinellen Lernens angepasst und integriert werden.

Diese Arbeit hat das Ziel, durch Maschinelles Lernen Verbesserungen im Software-Engineering zu erzielen. Sie ist auf der Software-Engineering-Landkarte im Bereich Prozessmodelle/Vorgehensmodelle, speziell Process Enactment für IT-Projekte, einzuordnen.

Motiviert wird diese Arbeit durch die teils großen Differenzen zwischen den dokumentierten und gelebten Prozessen während der IT-Entwicklung. Sie sieht den Projektmitarbeiter im Vordergrund und geht davon aus, dass bei einer Unterstützung zur Durchführung „seiner“ Arbeit (also „seines gelebten Prozesses“) am Ende ein besseres IT-Produkt fertiggestellt wird. Die Arbeit realisiert diese Unterstützung, indem die Vorgehensweise des Projektmitarbeiters beobachtet wird und mögliche nächste Schritte vorschlagen werden.

Hierfür wird ein Verfahren des Maschinellen Lernens (Bereich: Sequence Learning, Sequence Prediction) erarbeitet. Das hier vorgestellte Verfahren beobachtet die Arbeit, also die ausgeführten Prozessschritte und die dazugehörigen Kontextinformationen eines Projektmitarbeiters, und erlernt aus diesen Beobachtungssequenzen Strukturen, also methodisches Vorgehen. Mit den gelernten Informationen ist es möglich, konkrete und kontextspezifische Vorschläge für das weitere Vorgehen im Projekt zu machen. Somit kann, im Vergleich zu bisherigen Sequenzvorhersageverfahren, für unterschiedliche Kontexte unterschiedliches methodisches Vorgehen erlernt und vorgeschlagen werden.

Als Grundlage zur Anwendung dieses Verfahrens werden ein einfaches, regelbasiertes Prozessmodell und dessen Semantik vorgestellt. Dieses Prozessmodell enthält die relevanten Beschreibungselemente, um die o.g. Nutzerunterstützung umzusetzen.

Zur Bewertung des vorgestellten Verfahrens wird dieses einerseits mit synthetischen Beispieldaten evaluiert. Diese Daten werden aus einer Methodik, die durchgängig in der gesamten Arbeit zur Veranschaulichung heran gezogen wird, abgeleitet. Die Methodik ist dabei mit den Beschreibungsmitteln des o.g. Prozessmodells beschrieben. Andererseits wird die vorgestellte Vorgehensweise zur Unterstützung des Projektmitarbeiters auch mit realen Prozessdaten evaluiert. Bei beiden Evaluierungen wird die hier vorgestellte Methode mit dem zugrunde liegenden Kernverfahren verglichen. Die vorgestellte Vorgehensweise schneidet bei beiden Evaluierungen besser ab als das verglichene Kernverfahren. Das bedeutet, dass, sofern in den Prozessdaten kontextspezifisches Vorgehen enthalten ist, dieses – im Gegensatz zum Kernverfahren – auch gelernt wird. Das vorgestellte Verfahren ist also geeignet, kontextspezifisches, methodisches Vorgehen durch Beobachtung zu erlernen und vorzuschlagen. Die Anwendungsbereiche für dieses Verfahren sind vielfältig: Integriert in eine werkzeuggestützte Arbeitsumgebung lässt sich hiermit eine Unterstützung für Projektmitarbeiter realisieren. Diese Unterstützung sieht dann so aus, dass nächste Arbeitsschritte eines Projektes kontextspezifisch vorgeschlagen werden. Auf der anderen Seite lässt sich das Verfahren auch in eine Prozessmanagementumgebung integrieren, mit der projekt- oder organisationsweite Prozessbeschreibungen erstellt und gepflegt werden.

DANKSAGUNG

Dies ist die mit Sicherheit umfangreichste und aufwendigste wissenschaftliche Arbeit, die ich bisher erstellt habe. Eine Vielzahl an Personen hat mich bei der Fertigstellung der Arbeit direkt und indirekt unterstützt. Da die vollständige Nennung aller Personen schwer möglich ist, werde ich mich auf die Hervorhebung einiger weniger beschränken.

An erster Stelle danke ich meinem Doktorvater Prof. Dr. Andreas Rausch, der es mir durch die Möglichkeit der „flexiblen Arbeitsgestaltung" erst ermöglicht hat, diese Arbeit fertig zu stellen. Auch hat er mit seiner teils „peniblen" Liebe zum Detail und den sich daraus ergebenden Diskussionen bei der Erarbeitung des hier vorliegenden Ergebnisses geholfen.

Für die Übernahme des Zweitgutachtens möchte ich Prof. Dr. Dieter Rombach danken.

Dirk Niebuhr danke ich für die Übernahme des fachlichen Reviews vor Abgabe der Arbeit. In gleichem Maß bedanke ich mich bei meiner Frau Stefanie Deynet sowie meinem Schwiegervater Franz-Peter Schäfer für das Korrektorat.

Des Weiteren danke ich meiner Frau Stefanie und meinen Schwiegereltern für die Übernahme vieler „sozialer Verantwortung", wie zum Beispiel der Betreuung unserer Kinder. In dieser Zeit konnte ich sehr produktiv an der Dissertation arbeiten.

Abschließend bedanke ich mich bei meiner Mutter, die durch ihre Mühen in meinen frühen Jahren maßgeblich für meinen Bildungsweg verantwortlich ist.

Vielen Dank! Michael

Inhalt

1 EINLEITUNG

„Wir bauen Software wie Kathedralen: Zuerst bauen wir – dann beten wir." [1]

Dieses Zitat spiegelt die Problematik bei der Entwicklung von IT-Produkten wider. Es gibt viele bekannte Beispiele, bei denen IT-Produkte nicht wie gewünscht fertiggestellt wurden. Dies liegt teilweise an der sehr hohen Komplexität heutiger Systeme. Die Problematik wird weiterhin noch durch einen hohen Druck verstärkt, der auf IT-Projekten und deren Projektmitarbeitern lastet (Stichwort: schneller, günstiger, besser) [2].

Die Bemühungen, diesen Problemen in der IT-Entwicklung zu begegnen, sind vielfältig: Neue Technologien, Beschreibungstechniken für die Spezifikation, Implementierung und Test (wie zum Beispiel die UML), Architekturframeworks, Werkzeuge (z.B. Versions- und Anforderungsmanagement) und methodisches Vorgehen (z.B. OOA, OOD) sind nur einige Bereiche, die kontinuierlich verbessert wurden und noch werden mit dem eigentlichen Ziel, bessere Produkte (schneller und günstiger) zu entwickeln.

Nun stellt sich die Frage, wie Personen (die Projektmitarbeiter) all diese Technologien, Werkzeuge und Verfahren gezielt und integriert anwenden können, um möglichst gute Produkte zu entwickeln. Die Antwort versuchen Prozessmodelle[1] zu geben, indem sie generisch und idealisiert beschreiben, wer wann was zu tun hat, und damit versuchen, die o.g. Technologien, Werkzeuge und Verfahren ineinander zu integrieren. Diese Modelle können in IT-Projekten angewendet werden. Ein konkreter Ablauf in einem Projekt entspricht dann – so die Theorie – dem im Prozessmodell allgemein beschriebenen Vorgehen.

Es ist jedoch zu beobachten, dass es eine – teilweise große – Diskrepanz zwischen den in Prozessmodellen beschriebenen und in den IT-Projekten konkret durchgeführten Abläufen gibt [3–5]. Aussagen von Projektmitarbeitern, angesprochen auf die in ihrer Organisation dokumentierten Prozesse, wie „So arbeiten wir im Projekt aber nicht" (z.B. beschrieben in [6]) machen dies deutlich. Es gibt viele weitere Beispiele, die genau diese Problematik adressieren. Ein weiteres sei hier erwähnt und befindet sich im Bereich der Dokumentation und Konsistenz von Architekturen. In Prozessmodellen findet man in diesem Bereich häufig ein sehr restriktives Vorgehen (z.B. Designphase vor Implementierungsphase). Die Aussagen wie „Das müssen wir noch gerade ziehen" oder „Hier müssen wir noch nacharbeiten, um konsistent mit der Architektur zu werden" machen im Grunde nichts anderes deutlich, als dass sich hier der durchgeführte Prozess vom spezifizierten deutlich unterscheidet. Kabbaj et al. formulieren dies in [5] treffend, indem sie beschreiben, dass in der Software-Entwicklung „Konsistenz die Ausnahme und nicht die Regel ist".

Die Gründe hierfür sind sicherlich vielfältig. Einige werden in [7] angedeutet:

- Aufgrund der hohen Komplexität heutiger Software-Systeme und der daraus resultierenden Fülle an Aktivitäten, Artefakten, Stakeholdern und Interaktionen sind mögliche Abläufe und Entscheidungen nicht oder nur schlecht vorherzusehen.

[1] Auch „Vorgehensmodelle", „Prozessbeschreibungen" genannt
[2] Werkzeuge und Methoden haben dagegen einen geringen Einfluss.
[3] „Process Deviation Mangement Systems"
[4] Ein (unseres Erachtens nach) negatives Beispiel, wie effizient solche kontextsensitiven Verfahren in anderen Domänen sind, kann im Bereich der orts- und personensensitiven

- Die Projektmitarbeiter entscheiden viele Dinge nach ihren früheren Erfahrungen, die sie gesammelt haben, nach ihrem Bauchgefühl oder auch nach Vorlieben. Deshalb sind der Prozess und die daraus resultierenden Ergebnisse unsicher.

- Die Gründe (Einflüsse), die bestimmte Entscheidungen in IT-Projekten herbeigeführt haben, sind meist nicht klar ersichtlich.

- Die Einflüsse der „Umwelt", also Situationen und Parameter, die nicht offensichtlich entscheidungsrelevant sind, sind nicht zu unterschätzen.

Diese Probleme machen es schwierig, Prozessmodelle zu erstellen und zu pflegen, die die konkreten Abläufe in IT-Projekten widerspiegeln. Die eigentlichen Ursachen, die zu den oben genannten Punkten führen, sind insbesondere:

- Die Komplexität von IT-Systemen nimmt stetig zu und

- die Aktivitäten, die zur Erstellung eines Software-Systems nötig sind, werden größtenteils von Menschen durchgeführt.

Durch die zunehmende Komplexität von Software-Systemen wird deren Entwicklung mit „klassischen" Verfahren des Software-Engineering immer schwerer beherrschbar. Deshalb ist derzeit ein Trend zu beobachten, bei dem versucht wird, Verfahren aus dem Maschinellen Lernen in die Domäne des Software Engineering zu integrieren und anzuwenden. Die Vielzahl an internationalen Konferenzen und Workshops, die Themen des Software-Engineering mit Verfahren des Maschinellen Lernens (bzw. Künstliche Intelligenz) verknüpfen, machen dies deutlich.

Im Bereich der IT-Prozesse ist die Konsequenz der o.g. Problematik klar: Entweder gibt es die oben beschriebene Differenz zwischen Prozessmodell und ausgeführtem Prozess, oder man muss die Projektmitarbeiter derart einschränken, dass ihre Arbeitsweise konform zu dem spezifizierten Prozessmodell ist. Beide Lösungen sind nicht ernsthaft umsetzbar: Das Leben mit der Differenz zwischen Prozessmodell und Prozess stellt im Grunde genommen die Motivation einer prozessgetriebenen IT-Entwicklung infrage. Die Konsequenz wäre, dass man ganz auf Prozesse verzichten könnte. Dies ist jedoch keine Option, da durch die zunehmende Komplexität der IT-Produkte deren Entwicklung immer schwerer beherrschbar wird und genau dies eine Motivation für prozessgetriebene Entwicklungen ist. Die Einschränkung des Projektmitarbeiters ist auch keine sinnvolle Lösung: DeMarco [8] und Curtis et al. [9] argumentieren, dass bei einer IT-Entwicklung das Ergebnis *maßgeblich* von den Individuen abhängt und davon, wie diese arbeiten[2]. Eine Einschränkung der Individuen (also der Projektmitarbeiter) würde deshalb letztendlich zu schlechteren IT-Produkten führen.

Die o.g. Problematik der Diskrepanz zwischen spezifizierten und ausgeführten Prozessen ist bereits bekannt. Heutige, „klassische" Lösungen hierfür sind:

- Die Inhalte in den Prozessmodellen entsprechen einer „Prozessdokumentation". Diese Dokumentation (wie z.B. beim V-Modell XT) kann als Anleitung oder zur Orientierung genutzt werden. Die Modelle zeigen typischerweise eine starke Produktzentrierung. Abläufe stehen hier eher im Hintergrund und haben beschreibenden Charakter. Eine Werkzeugunterstützung im Prozessbereich ist damit aber nicht möglich (auch schon wegen der unpräzisen Semantik der Beschreibungssprachen).

[2] Werkzeuge und Methoden haben dagegen einen geringen Einfluss.

- Prozessbeschreibungssprachen (also die Sprachen, um Prozessmodelle mit Inhalt zu füllen) können so gestaltet werden, dass eine flexible Ausführung möglich ist. Hier gibt es grundsätzlich verschiedene Ansätze:

    - Regelbasierte Sprachen geben keinen direkten Ablauf vor. Die Ausführung eines Prozessschrittes ist lediglich an Bedingungen geknüpft. Der Ablauf während der Prozessausführung ist frei wählbar (abhängig von den Bedingungen). Das Problem bei diesen Ansätzen ist, dass eine prozessgetriebene Anleitung/Hilfestellung des Projektmitarbeiters nicht möglich ist, da die nächsten anzuwendenden Arbeitsschritte nicht explizit im Prozessmodell beschrieben sind.

    - Andere Ansätze sehen eine Anpassung des Prozessmodells vor. Hier ist zwischen drei Ebenen zu unterscheiden:

        a) Auf der ersten Ebene sind Ansätze anzusiedeln, die sich mit der *organisationsweiten Anpassung* von Prozessmodellen beschäftigen. Die Motivation ist hier vornehmlich das Prozessmodell-Management.

        b) Auf der zweiten Ebene bieten Ansätze Operationen an, um organisationsweite (generische) Prozessmodelle initial *projektspezifisch anzupassen*. Ziel ist es hier, das Prozessmodell auf die Bedürfnisse des Projektes (z.B. Prozessanforderungen des Auftraggebers) auszurichten.

        c) Die dritte Ebene beschäftigt sich mit der *Anpassung* von Prozessmodellen *während der Ausführung*. Motiviert wird dies durch die Tatsache, dass es (auch während eines IT-Projektes) permanent Änderungsbedarf an den spezifizierten Abläufen gibt.

        Alle diese Ansätze (a-c) haben das Problem, dass bei einer oftmals nur temporären oder spontanen Änderung des Ablaufs das Prozessmodell angepasst werden muss (und zwar bevor der geänderte Ablauf durchgeführt wird!) [4], [5], [10]. Zum einen unterstützen Prozessmodelle in solchen Situationen den Projektmitarbeiter nicht, sondern verlangen einen Mehraufwand (in Form der Modellanpassung). Zum anderen gehören temporäre/spontane oder auch projektmitarbeiterspezifische Abläufe nicht in ein (projektweit oder organisationsweit gültiges) Prozessmodell.

    - Weitere Ansätze[3] bieten die Chance, generelle, idealisierte Abläufe im Prozessmodell zu beschreiben. Weiterhin gibt es auch die Möglichkeit, etwaige Abweichungen zu definieren. Eine flexible Ausführung ist so machbar. Als Unterstützung des Projektmitarbeiters (in Form einer Anleitung/Orientierung) werden die im Prozessmodell beschriebenen generellen Abläufe herangezogen. Das Problem bei diesen Ansätzen ist, dass die Unterstützung des Projektmitarbeiters nur dem idealisierten, generischen (Standard-)Prozess folgt. Er wird also nicht bei seiner spezifischen Arbeit unterstützt. Der eigentliche Nutzen für ihn ist hier – ähnlich wie bei den oben beschriebenen regelbasierten Modellen – gering.

Alle oben vorgestellten Ansätze nutzen „klassische Verfahren“ des Software Engineering, sind jedoch keine gut funktionierenden Lösungen für das Problem der Diskrepanz zwischen spezifizierten und ausgeführten Prozessen. Deshalb setzen wir den o.g. Trend, mit lernenden

[3] „Process Deviation Mangement Systems“

Verfahren Verbesserungen im Software Engineering zu erreichen, fort und versuchen somit, die existierenden Unterschiede zwischen Prozessmodell und ausgeführtem Prozess zu minimieren.

Diese Arbeit folgt der o.g. Annahme, dass die Individuen (die Projektmitarbeiter) maßgeblich für das Ergebnis einer Produktentwicklung verantwortlich sind. Deshalb sieht sie den Projektmitarbeiter im Fokus und geht davon aus, dass er nicht eingeschränkt werden sollte. Stattdessen haben wir in der Arbeit die Idee, dass ein Prozessmodell mit einer geeigneten Werkzeuglandschaft den Projektmitarbeiter unterstützen sollte. Diese Unterstützung darf sich jedoch nicht nur auf das im Prozessmodell idealisierte Vorgehen beschränken, sondern muss flexibel sein und individuelle und „umweltabhängige" Einflüsse (Kontexte) berücksichtigen.

Unsere Arbeit lässt sich mit den aktuellen Trends vergleichen, die heutige IT-Systeme (z.B. Smartphones und Tablet-Computer) mit ihren Bedien- und Nutzungskonzepten setzen. Wir meinen hiermit die Möglichkeiten, die diese Geräte bieten, um intuitiv zu arbeiten und dabei sogar auf eine Bedienungsanleitung verzichten können. Diese Systeme werden „Kontextsensitive Systeme" genannt und zeichnen sich dadurch aus, dass sie ihre Dienste „optimal und ohne explizite Eingabe der Nutzer erbringen" [11]. Sie arbeiten situationsgemäß und beziehen den „aktuellen Kontext" aus dem aktuellen Zustand des Systems und einer Menge von Sensorinformationen (z.B. GPS). So können diese Systeme gezielt und für den Kontext sinnvolle Informationen einblenden (z.B. Navigationsvorschläge). Der Benutzer ist sich i.A. der Technik, die dahinter steckt, nicht bewusst.[4] Diese Arbeit schlägt genau in diese Kerbe und bietet erste Grundsteine, um ein intuitives Arbeiten in IT-Projekten zu ermöglichen, indem lernende Verfahren mit Prozessmodellen integriert werden, um die immer komplexer werdende Entwicklung von Systemen beherrschbar zu machen.

1.1 THEMATISCHE EINORDNUNG

Diese Arbeit ist im Gebiet des „Software Engineering" der Informatik anzusiedeln. In Abbildung 1 ist die Software-Engineering Landkarte[5] dargestellt. Links und rechts befinden sich die Themen, die sich mit der Anwendung in IT-Projekten beschäftigen und sich an den grundlegenden Phasen während der IT-Entwicklung orientieren: Das Thema „Software-Engineering Management" beschäftigt sich beispielsweise mit der Anwendung von Management-Aktivitäten (z.B. Planung, Controlling). Andere Bereiche befassen sich mit Anforderungen (z.B. Prozesse und Methoden zur Erhebung dieser) oder mit dem Design von Software-Architekturen bis hin zum Testen und zur Pflege von Software-Systemen. Im mittleren Bereich der Landkarte sind querschnittliche Themen des Software-Engineering enthalten. Diese Themen wirken sich teilweise auf mehrere der o.g. Bereiche aus, die sich mit der Anwendung beschäftigen. Hier sei insbesondere auf das Thema „Software-Engineering Prozess" hingewiesen: Dieses Thema befasst sich nicht mit konkreten Prozessen für die Software-Entwicklung. Konkrete Prozesse werden durch die Anwendungsthemen links und rechts auf der Landkarte abgedeckt. „Software-Engineering Prozess" geht vielmehr auf generelle Prozess-Themen ein: Zum Beispiel auf die Beschreibung von Prozessen (Meta-Modelle, Prozessbeschreibungssprachen), deren Ausführung (die Semantik dieser Sprachen) oder auch das Management, das Messen oder die Verbesserung von IT-Prozessen im Allgemeinen. Viele Themen des Software-Engineering nutzen Verfahren aus anderen Bereichen der Informatik. Mit „Related Disciplines of Software Engineering" sind

[4] Ein (unseres Erachtens nach) negatives Beispiel, wie effizient solche kontextsensitiven Verfahren in anderen Domänen sind, kann im Bereich der orts- und personensensitiven Werbung beobachtet werden.

[5] Das „IEEE Software Engineering Body of Knowledge" [12] versucht, Software-Engineering in kleinere Teilgebiete zu klassifizieren und den Stand der Technik zu erfassen. Diese Klassifikation wurde in dieser Arbeit als Grundlage für die Software-Engineering Landkarte genommen.

diese Verfahren gemeint. Einige Themenbereiche der Informatik sind in Abbildung 2, rechts dargestellt. Diese Klassifizierung ist an [13] angelehnt.

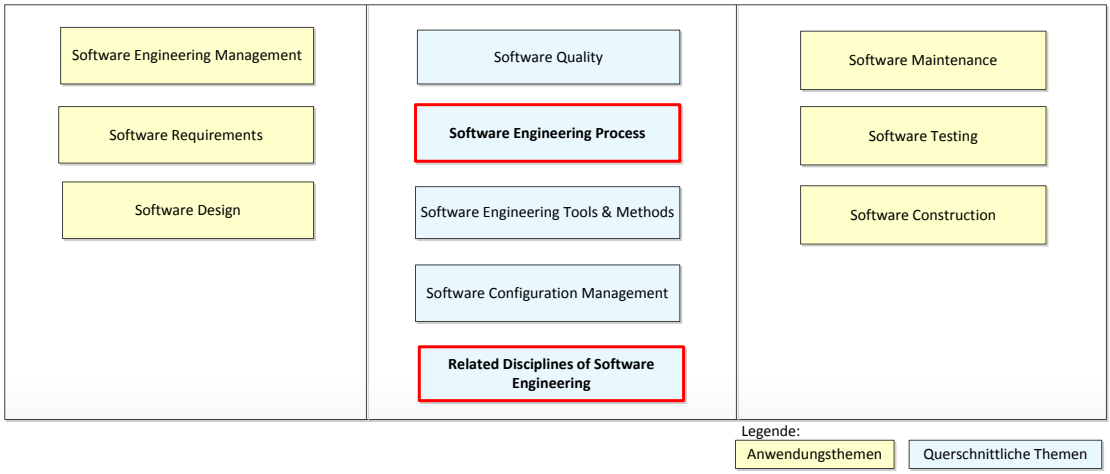


Abbildung 1: Die Software-Engineering Landkarte (abgeleitet aus [12])

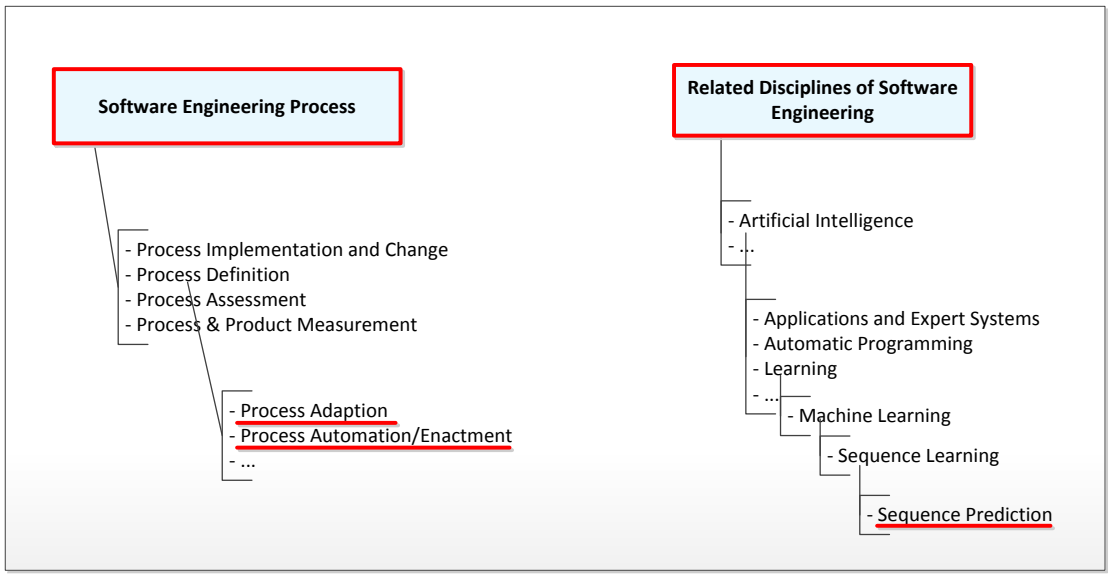


Abbildung 2: In dieser Arbeit adressierte Bereiche der Software-Engineering Landkarte (links: abgeleitet aus [12], rechts: abgeleitet aus [13]

Die vorliegende Arbeit adressiert die in Kapitel 1.1 beschriebene Differenz zwischen spezifizierter Ablaufbeschreibung im Prozessmodell und tatsächlichem Ablauf während eines IT-Projektes. Der Projektmitarbeiter steht hier insbesondere im Fokus. Das Prozessmodell, sowie Werkzeuge, die dieses interpretieren, haben das Ziel, den Projektmitarbeiter bei seiner Arbeit zu unterstützen. Deshalb ist dieses Werk im Bereich „Software-Engineering Prozess“ der Software-Engineering Landkarte einzuordnen. Eine Detailsicht des Bereichs „Software-Engineering Prozess“ ist in Abbildung 2 dargestellt. Diese Arbeit liefert Beiträge zu „Process Adaption“ sowie „Process Automation/Enactment“. Mit „Process Adaption“ ist die Anpassung von Prozessmodellen an „lokale Bedürfnisse“ gemeint. „Process Automation/Enactment“ befasst sich mit dem Ausführen von diesen Prozessmodellen. Hierunter fällt auch das

Ausführen/Interpretieren zur Anleitung, Orientierung oder Hilfestellung des Projektmitarbeiters. Diese Unterstützung des Projektmitarbeiters liegt im Fokus dieser Arbeit. Hierfür werden Verfahren aus dem Bereich des Maschinellen Lernens angewendet und angepasst. Im Speziellen sind dies Verfahren für die Sequenzvorhersage[6] (siehe Abbildung 2, rechts).

Im folgenden Kapitel werden die Ziele dieser Arbeit ausführlich erläutert.

1.2 ZIELSETZUNG UND ERGEBNISSE

Diese Arbeit greift die in der Einleitung beschriebene Problematik auf. Das Kernziel ist es, die wesentlichen Elemente, die für eine prozessgetriebene Unterstützung des Projektmitarbeiters bei seiner (Projekt-)Arbeit nötig sind, zu konzipieren und spezifizieren. Wichtig ist dabei die Annahme, dass es sinnvoll ist, den Projektmitarbeiter bei seinen Arbeiten nicht oder nur in sinnvollem Maß einzuschränken. Die Kernmotivation der Anwendung eines prozessgetriebenen Ansatzes in IT-Projekten liegt deshalb in der Unterstützung (Hilfestellung) des Projektmitarbeiters. Diese Unterstützung soll durch ein lernendes Verfahren erreicht werden, da wir hierdurch Verbesserungen im Software Engineering erwarten. Hieraus leiten sich folgende weitere Ziele ab:

- Um eine flexible Ausführung eines Prozessmodells zu bewerkstelligen, muss als Erstes eine adäquate **Prozessbeschreibungssprache** konzipiert und spezifiziert werden. Dies ist die Grundlage für eine Unterstützung (Hilfestellung, z.B. in Form von Vorschlägen, welche Aktivitäten als nächstes ausgeführt werden) des Projektmitarbeiters. Dabei beinhaltet die Spezifikation der Sprache sowohl die Syntax, als auch deren Ausführungssemantik.

- Mit einer Prozessbeschreibungssprache, mit der eine flexible, den Projektmitarbeiter nicht behindernde Ausführung möglich ist und bei der der konkrete Ablauf konform zur spezifizierten Prozessbeschreibung im Prozessmodell ist, ist bisher keine Nutzerunterstützung umgesetzt. Deshalb ist es ein weiteres Ziel, eine sinnvolle **Nutzerunterstützung** für den Projektmitarbeiter zu konzipieren und zu spezifizieren. Hierbei ist relevant, dass der Projektmitarbeiter bei *seiner* Arbeit unterstützt wird. Eine Hilfestellung in Form einer Bereitstellung eines idealisierten Standardablaufs ist nicht ausreichend. Weiterhin sollen zur Bereitstellung der Nutzerunterstützung keine zusätzlichen Arbeiten für den Projektmitarbeiter entstehen. Dies wäre nämlich eine zusätzliche Belastung und Behinderung des Mitarbeiters. Vielmehr muss die Nutzerunterstützung automatisiert an die (auch umwelt- oder kontextspezifische) Arbeitsweise des Projektmitarbeiters angepasst werden. Diese Nutzerunterstützung soll mit einem Verfahren aus dem Bereich des Maschinellen Lernens realisiert werden.

Die wesentlichen Ergebnisse dieser Arbeit sind dann:

- Konzeption und Spezifikation einer Prozessbeschreibungssprache, bei der kein expliziter Ablauf einzelner Arbeitsschritte vorgegeben wird. Die Arbeitsschritte stehen also „flach nebeneinander“ und können – abhängig von deren Vor- und Nachbedingungen – flexibel gestartet und beendet werden (ähnlich wie regelbasierte Sprachen). Dabei werden Kernelemente, die für die weitere Nutzerunterstützung nötig sind, integriert. Dies umfasst insbesondere Kontextinformationen (z.B. Umwelteinflüsse).

[6] Engl.: „Sequence Prediction“

- Konzeption und Spezifikation einer Nutzerunterstützung: Diese Nutzerunterstützung wird in Form einer „Erfahrungsdatenbank“ realisiert. Das bedeutet, dass die Arbeitsweise des Projektmitarbeiters beobachtet wird. Die Beobachtungen (z.B. Arbeitsschritte und Kontextinformationen) werden genutzt, um die Erfahrungsdatenbank aufzubauen. Diese enthält kontextabhängige Abläufe, die der Arbeitsweise des Projektmitarbeiters entsprechen. Zum Aufbau der Erfahrungsdatenbank wird ein Verfahren aus dem Maschinellen Lernen, speziell Sequenzvorhersage, genutzt.

- Evaluierung der Nutzerunterstützung mit einem synthetischen Beispiel, sowie mit realen Projektdaten.

Weiterhin wird ein beispielhafter Prozessausschnitt (Methodik) als Anwendungsbeispiel definiert. Dieser Prozessausschnitt wird durchgängig in der gesamten Arbeit genutzt, um die Konzepte und Verfahren anschaulich zu beschreiben.

1.3 AUFBAU DER ARBEIT

Im folgenden Kapitel 2 wird der Stand der Technik zu IT-Prozessen und zu Verfahren für die Sequenzvorhersage (Bereich Maschinelles Lernen) dargelegt. Das Kapitel ist in zwei große Teile unterteilt:

Der erste Teil befasst sich mit IT-Prozessen. Als Erstes werden Prozessbeschreibungen, also das idealisiert beschriebene Vorgehen in Prozessmodellen im Allgemeinen, vorgestellt. Danach werden Verfahren beschrieben, die eine Ausführung von Prozessmodellen vorsehen. Diese definieren also eine klar definierte Syntax und Ausführungssemantik. Der Fokus liegt dabei auf Beschreibungssprachen, mit denen eine für den Projektmitarbeiter flexible Ausführung möglich ist. Abschließend werden Verfahren vorgestellt, die sich mit der Anpassung von Prozessmodellen beschäftigen. Diese Anpassung ist nötig, um konkrete Abläufe konsistent zu den Prozessmodellen zu halten, und sie ist essentiell, um eine adäquate Nutzerunterstützung zu gewährleisten.

Der zweite Teil von Kapitel 2 beschreibt den Stand der Technik im Bereich Maschinelles Lernen und Sequenzvorhersage. Der Fokus liegt hier auf Verfahren für die Sequenzvorhersage, die aus einer ähnlichen Domäne wie diese Arbeit stammen (Erlernen von Abläufen von Menschen, z.B. intelligente Benutzeroberflächen). Verfahren aus anderen Domänen (z.B. DNA-Sequenzierung) werden hier nicht vorgestellt.

Das darauf folgende Kapitel 3 stellt einen Prozessausschnitt als Anwendungsbeispiel vor. Hier wird eine Methodik beschrieben, um ausgehend von den Anforderungen eine Systemarchitektur zu entwerfen und die darin enthaltenen Komponenten zu spezifizieren. Dieses Anwendungsbeispiel wird im weiteren Verlauf der Arbeit konsequent als Beispiel genutzt. Weiterhin werden in diesem Kapitel mögliche unterschiedliche projektmitarbeiter- und kontextspezifische Abläufe bei der Anwendung der Methodik beschrieben.

Kapitel 4 beschreibt den in dieser Arbeit umgesetzten Gesamtansatz. Hier wird als Erstes das Problem, das sich bei Umsetzung einer projektmitarbeiter- und kontextspezifischen Nutzerunterstützung mit traditionellen Prozessansätzen ergibt, anschaulich dargestellt. Ausgehend von diesen Punkten werden daraufhin eine Prozessbeschreibungssprache und die Nutzerunterstützung konzipiert.

Ausgehend von der Konzeption in Kapitel 4 erfolgt die Spezifikation der Prozessbeschreibungssprache in Kapitel 5. Es umfasst einerseits die präzise Beschreibung der Syntax, andererseits auch die Definition der Ausführungssemantik.

Kapitel 6 stellt die Nutzerunterstützung während der Prozessausführung vor. Hier wird ein maschinelles Lernverfahren für die Sequenzvorhersage, das auch Kontexteinflüsse berücksichtigt, vorgestellt und spezifiziert.

In Kapitel 7 erfolgt eine Evaluierung auf Basis der Beispielmethodik aus Kapitel 3: Aus der Methodik werden mehrere Beispielsequenzen abgeleitet. Das Verfahren zur Nutzerunterstützung aus Kapitel 6 wird mit diesen Beispielsequenzen evaluiert.

Das darauf folgende Kapitel 8 beschreibt die Evaluierung mit realen Projektdaten. Diese Daten stammen aus der IT-Abteilung eines großen deutschen Unternehmens.

Kapitel 9 fasst die Ergebnisse der Arbeit zusammen und gibt einen Ausblick auf weiterführende Themen.

2 STAND DER TECHNIK: SOFTWAREPROZESSE UND SEQUENZVORHERSAGE

Dieses Kapitel befasst sich mit dem Stand der Technik und ist in zwei große Bereiche unterteilt:

Der erste Teil (Kapitel 2.1) beschäftigt sich mit Softwareprozessen. Hier wird einführend auf Softwareprozesse im Allgemeinen eingegangen. Weiterhin werden Prozessbeschreibungssprachen, also Sprachen, um IT-Prozesse generalisiert und idealisiert zu beschreiben, vorgestellt sowie deren Ausführung (Enactment) behandelt. Abschließend werden verfügbare Techniken vorgestellt, um Prozessbeschreibungen anzupassen. Dies kann im Großen, beispielsweise organisationsübergreifend, oder auch im Kleinen, also während der Ausführung geschehen.

In dieser Arbeit versuchen wir – stark vereinfacht ausgedrückt – mit Hilfe von maschinellen Lernverfahren Verbesserungen im Software Engineering zu erzielen. Das Kapitel 2.2 geht auf den Stand der Technik in dem Bereich des maschinellen Lernens ein. Der erste Teil dieses Unterkapitels beschäftigt sich mit Maschinellem Lernen im Allgemeinen. Der zweite Teil stellt existierende Sequenzvorhersage-Verfahren vor. Der Fokus hier liegt auf solche Verfahren, die sich mit Benutzerinteraktionen beschäftigen (z.B. Vorhersage von Operationen, die ein Benutzer an einer grafischen Oberfläche ausführt – hiermit können beispielsweise kontextabhängige Menüs realisiert werden).

2.1 SOFTWAREPROZESSE – BESCHREIBUNG UND ENACTMENT

Dieses Kapitel befasst sich mit dem Stand der Technik für Prozesse für IT-Projekte und ist in drei Teile untergliedert:

Der erste, einleitende Teil (Kapitel 2.1.1) geht grundlegend auf Prozessbeschreibungen[7], also mit allgemeingültigen Beschreibungen von Vorgehen in IT-Projekten ein. In diesem Kapitel werden dann auch die Unterschiede zwischen diesen Prozessbeschreibungen und Prozessbeschreibungssprachen erläutert. Weiterhin werden die genannten Begriffe mit der Anwendung/Ausführung einer Prozessbeschreibung, also dem eigentlichen Prozess, in Bezug gesetzt.

Das Kapitel 2.1.2 befasst sich mit der Ausführung, dem Enactment[8] von Vorgehensmodellen. In diesem Kontext werden verschiedene Paradigmen für das Enactment von Vorgehensmodellen vorgestellt und es wird auf entsprechende Werkzeuge verwiesen.

Im Software Engineering ist es eine wichtige Erkenntnis, dass Prozessbeschreibungen nicht fix und starr sind, sondern permanent angepasst werden müssen [12]. Diese Anpassung geschieht derzeit auf zwei Ebenen: Auf der Organisationsebene werden organisationsübergreifendes Vorgehen, Werkzeuge sowie Standards in Prozessbeschreibungen integriert. Auf der Projektebene, also im Rahmen eines einzelnen IT-Projekts, werden diese Prozessbeschreibungen an die projektspezifischen Bedürfnisse angepasst. Der Stand der Technik in Bezug auf die Anpassbarkeit von Prozessbeschreibungen wird in Kapitel 2.1.3 ab Seite 17 erläutert.

[7] Andere Begriffe hierfür sind: Vorgehensmodelle, Prozessmodelle

[8] In dieser Arbeit wird das „Ausführen“ von Vorgehensmodellen als „Enactment“ bezeichnet. Das Wort „Enactment“ wird benutzt, da hiermit nicht (nur) das (automatische) Ausführen eines Prozessschrittes durch eine Maschine (Computer) gemeint ist (wie dies im Bereich der Produktionsprozesse zu finden ist). In IT-Projekten führen dagegen „ein Symbiose von Menschen und Computern“ [14] Prozessschritte aus.

2.1.1 PROZESSBESCHREIBUNGEN IM ALLGEMEINEN

Eine Prozessbeschreibung gibt vor, wer wann was in einem IT-Projekt zu tun hat. Prozessbeschreibungen werden keineswegs einmal initial erstellt und danach nicht mehr verändert. Vielmehr werden sie permanent im Großen (auf Organisationsebene) und im Kleinen (Projektebene) angepasst und aktualisiert (siehe hierzu Kapitel 2.1.3). Um diese Veränderungen an den Modellen sinnvoll vornehmen zu können, haben sich Prozessbeschreibungssprachen in der Welt der Vorgehensmodelle etabliert.

Prozessbeschreibungssprachen bieten dem Nutzer, im Allgemeinen dem Prozessingenieur auf Organisationsebene und dem Projektleiter auf Projektebene, die Möglichkeit, Vorgehensweisen (in IT-Projekten) modellhaft und allgemein gültig zu beschreiben.

In Prozessbeschreibungen werden, neben den Ablaufbeschreibungen[9], meistens noch Rollen und Produkte[10] beschrieben. Im V-Modell XT (siehe [15]) gibt es beispielsweise die Rolle „SW-Entwickler", die Produkte vom Typ „SW-Modul" erstellt. Die Aktivität, in der der Ablauf zum Erstellen des Produktes beschrieben ist, heißt „SW-Modul realisieren". Die „Grundelemente" wie z.B. „Aktivität" oder „Produkt" werden von Prozessbeschreibungssprachen unterstützt.

Prozessbeschreibungssprachen fokussieren i.A. auf bestimmte Beschreibungselemente: Manche Prozessbeschreibungen sind produktorientiert, wie zum Beispiel das V-Modell XT, andere sind aktivitätsorientiert.

Prozessbeschreibungen werden in IT-Projekten angewendet, d.h. sie werden ausgeführt. Eine ausgeführte Prozessbeschreibung (oder eine Teilsequenz davon) wird als Prozess bezeichnet. Ein solcher Prozess entspricht – so die Theorie – der Prozessbeschreibung, der er zugrunde liegt. Das Ausführen von Vorgehensmodellen wird auch als „Enactment" (engl. process enactment) bezeichnet. In realen IT-Projekten entspricht der Prozess jedoch selten dem Vorgehensmodell. Tom DeMarco beschreibt dies in seinem Buch „Peopleware":

> *„There is a big difference between Methodology and methodology." [8]*

Die Methodik[11], die bei DeMarco groß geschrieben ist, entspricht in dieser Arbeit dem Begriff Prozessbeschreibung/Vorgehensmodell und wird als Versuch beschrieben, ein allgemeingültiges Vorgehen in einer Organisation zu etablieren, bei dem alle bedeutenden Entscheidungen nicht von den Projektmitarbeitern getroffen werden, sondern von der Methodik (dem Vorgehensmodell) vorgegeben sind. Auf der anderen Seite lässt sich die Methodik, die bei DeMarco klein geschrieben ist, mit dem in dieser Arbeit verwendeten Begriff „Prozess" vergleichen und besteht sinngemäß aus „einem auf die [in einem Projekt] bevorstehenden Arbeiten maßgeschneiderten Plan sowie Fähigkeiten [des Projektmitarbeiters], diesen Plan auszuführen" [8].

Rombach erklärt in [3] die teils großen Unterschiede zwischen Vorgehensmodell und ausgeführtem Prozess mit der Tatsache, dass Vorgehensmodelle oftmals hauptsächlich dem Projektcontrolling dienen und sehr stark in andere (nicht für die Entwicklung/Ingenieurstätigkeit bestimmte) Prozessbeschreibungen integriert sind. Daraus resultieren sehr generische Beschreibungen für die IT-Entwicklung, die schließlich für den Projektmitarbeiter schlecht anwendbar sind.

---

[9] Oftmals auch „Aktivitäten" genannt

[10] Auch „Artefakte" genannt

[11] Andere Arbeiten differenzieren stärker zwischen den Begriffen Vorgehensmodell und Methodik: In [16] ist ein Vorgehensmodell neben Beschreibungstechniken, Systemmodell, Technologien, Architekturen, Managementpraktiken und Werkzeugunterstützung nur ein Teil einer Methodik.

Ein weiterer Punkt, der die Unterschiede zwischen Prozessbeschreibung und Prozess erklärt, ist, dass Prozessbeschreibungen „das Vorgehen“ für den Nutzer anschaulich beschreiben. Das heißt, diese Vorgehensmodelle entsprechen eher einer „Vorgehensdokumentation“ als einer generischen Beschreibung von konkreten Abläufen. Als Beispiel sei hier die V-Modell XT Aktivität „Systemarchitektur erstellen“ genannt. Diese ist in Abbildung 3 dargestellt. Betrachten wir nun lediglich die zwei Arbeitsschritte, die links mittig in der Abbildung enthalten sind: „Architektursichten identifizieren“ und „Architektursichten erarbeiten“. Diese beiden Arbeitsschritte werden – laut Abbildung – parallel ausgeführt. Das V-Modell XT schreibt zum erstgenannten Arbeitsschritt: „Ausgehend von den identifizierten Architekturtreibern ist mit der Auswahl geeigneter Architektursichten fortzufahren. […]“ [15]. Zum Arbeitsschritt „Architektursichten erarbeiten“ schreibt das V-Modell XT: „[…] Für jede Sicht werden mit Hilfe geeigneter Beschreibungssprachen die relevanten Architekturaspekte erarbeitet. […]“ [15]. Diese beiden textuellen Beschreibungen spezifizieren das Vorgehen wesentlich detaillierter, als die „formal“ modellierte Aktivität: Gemeint ist hier, dass für jede Architektursicht, die im erstgenannten Arbeitsschritt identifiziert wurde, ein Arbeitsschritt durchgeführt werden muss, um diese Sicht zu erarbeiten. Die Reihenfolge (z.B. erst eine Architektursicht identifizieren, dann diese erarbeiten, … vs. Alle Architektursichten identifizieren, dann alle erarbeiten) ist dabei nicht relevant. Aus der formalen Aktivitätsbeschreibung (Abbildung 3) ist dies so nicht erkennbar, und deshalb ist es auch nicht durch ein Softwarewerkzeug interpretierbar.

In der IT-Entwicklung gibt es nicht DAS Vorgehensmodell. Hierfür ist das Vorgehen in Projekten zu unterschiedlich und hängt von zu vielen Faktoren ab, beispielsweise von der Domäne, in der das Projekt durchgeführt wird (z.B. eingebettete Systeme vs. Informationssysteme), oder auch von dem Unternehmen und dessen eingeführten Werkzeugen und Standards. Des Weiteren spielt die Größe des IT-Projektes eine große Rolle für die angewendete Prozessbeschreibung. In einem großen Projekt ist beispielsweise ein stringentes Controlling sinnvoll. Einige Abläufe in einem solchen Prozess sind deshalb völlig anders als in einem kleinen Projekt, in dem ein agiles Vorgehensmodell angewendet wird. Als Beispiel für diese unterschiedlichen Abläufe bei großen und kleinen Projekten seien hier die Änderungsprozesse in den entsprechenden Vorgehensmodellen genannt: Ein Änderungsprozess in einem phasenorientierten Vorgehensmodell, z.B. dem Wasserfallmodell ist schwergewichtiger und „träger“. Deshalb werden Änderungen im Allgemeinen wesentlich später umgesetzt als bei einem agilen Modell. Denn es ist sogar ein Ziel des agilen Vorgehens, Änderungen schnell umzusetzen: „Reagieren auf Veränderung [ist wichtiger] […] als das Befolgen eines Plans“ [17].

Des Weiteren besitzen Vorgehensmodelle unterschiedliche Granularitätsstufen: Es gibt einerseits Modelle, die sehr präzise einzelne Arbeitsschritte im Projektverlauf beschreiben. Durch diese präzise Beschreibung ist es dann möglich, einzelne Schritte zu (teil-)automatisieren. Voraussetzung hierfür ist, dass die Sprache, mit der das Vorgehensmodell beschrieben ist, maschinenlesbar und -interpretierbar ist; siehe hierzu Kapitel 2.1.2. Auf der anderen Seite gibt es Vorgehensmodelle, die nur sehr abstrakt unterschiedliche Phasen und deren durchzuführende Reihenfolge beschreiben.

Prozessbeschreibungen spezifizieren und dokumentieren üblicherweise das Vorgehen mit Phasen und legen eine Durchführungsreihenfolge fest. Als Beispiel sei hier das Wasserfallmodell [18] genannt. Ein Überblick über die Phasen ist in Abbildung 4 dargestellt. Zu jeder dieser Phasen gibt es eine Menge von feingranulareren Aktivitäten und Produkten/Artefakten, die zu bearbeiten oder erstellen sind. In Abbildung 5 sind die Produkte beispielhaft dargestellt, die zur Phase „Program Design“ erstellt werden. Hier gibt es Produkte, die in der Phase genau einmal erstellt werden (z.B. Test Plan), andere Produkte, wie zum Beispiel „Final Design“, können öfter vorkommen.

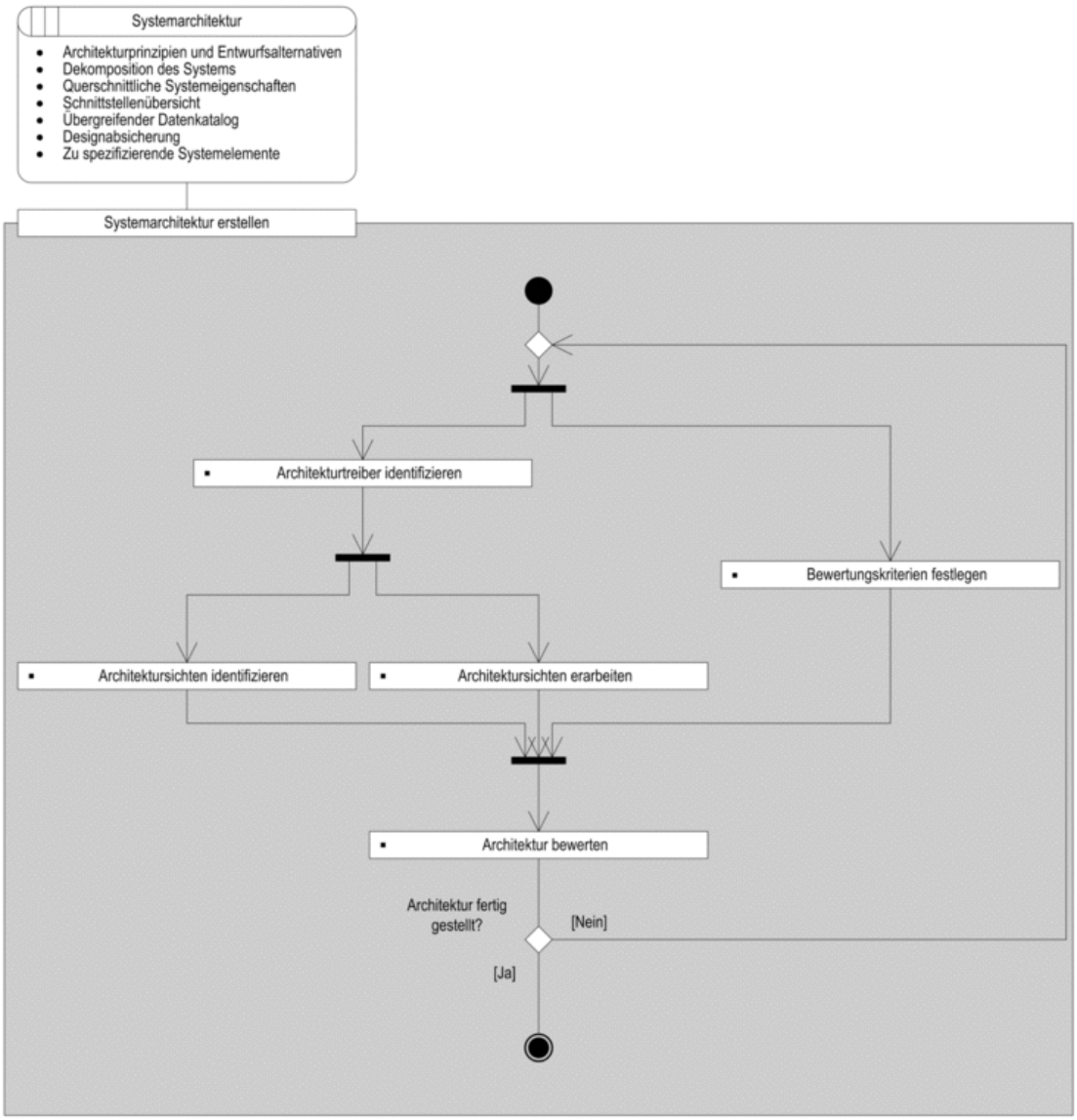


Abbildung 3: Aktivität „Systemarchitektur erstellen" des V-Modell XT (VMXT 2012)

Oftmals werden Prozessbeschreibungen anhand der Anordnung ihrer Phasen klassifiziert. So gibt es beispielsweise die V-förmigen Modelle. Die Phasen im linken Schenkel beschreiben dabei die Dekomposition/Entwurf des Systems, Spezifizierung und Implementierung. Die Phasen des rechten Schenkels befassen sich mit Integration, Test bis hin zur Abnahme. Ein im deutschsprachigen Raum bekanntes Modell in dieser Klasse ist das V-Modell XT (siehe [15], [19]). Ein Überblick über die Phasen zur Systemerstellung beim V-Modell XT ist in Abbildung 6 dargestellt[12]. Hinter diesen sind Produkte/Artefakte definiert, die zum Abschließen der Phase fertig gestellt werden müssen. Hinter den Produkten sind wiederum Aktivitäten hinterlegt, also die Ablaufbeschreibungen, die zum Erstellen eines Produktes durchgeführt werden müssen.

[12] Genaugenommen sind in dieser Abbildung keine Phasen dargestellt, sondern Enscheidungspunkte. Das Glossar des V-Modell XT schreibt zu Entscheidungspunkten: „In einem Entscheidungspunkt wird über das Erreichen einer Projektfortschrittsstufe entschieden." [20]. Das bedeutet, dass die eigentlichen Phasen zwischen zwei Entscheidungspunkten liegen(z.B. „System entwerfen" zwischen „System spezifiziert" und „System entworfen").

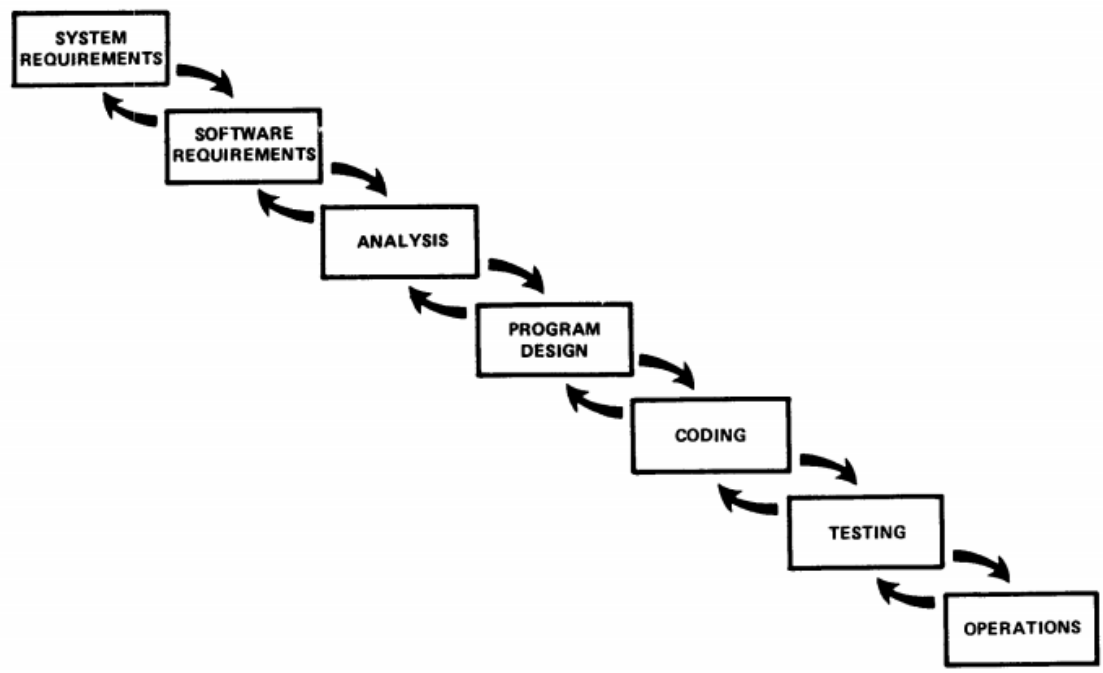


Abbildung 4: Überblick über das Wasserfallmodell [18]

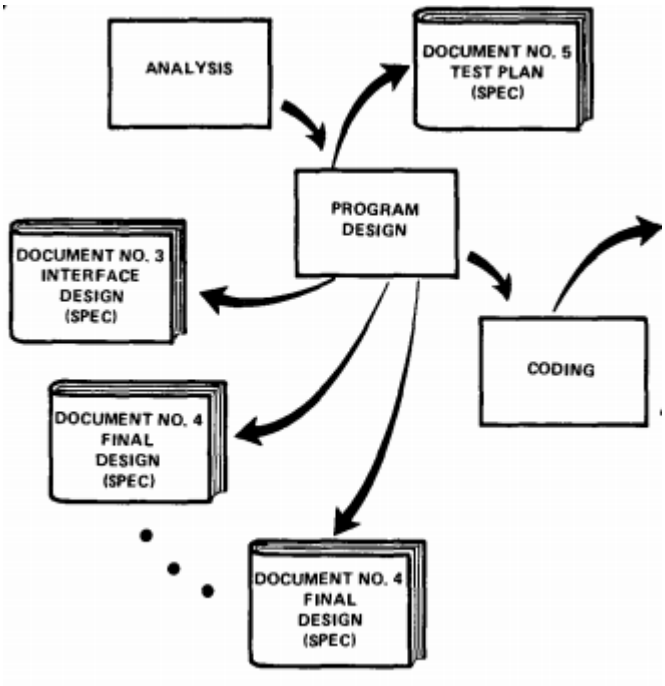


Abbildung 5: Zu erstellende Produkte beim „Program Design" (Wasserfallmodell) [18]

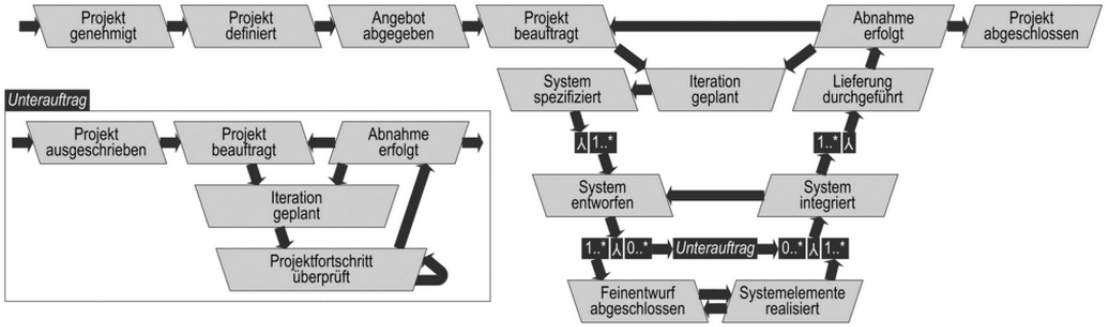


Abbildung 6: Inkrementelle Systementwicklung beim V-Modell XT [20]

2.1.2 AUSFÜHRUNG VON SOFTWAREPROZESSEN

Um Prozessmodelle werkzeugbasiert ausführen zu können, muss die zugrunde liegende Prozessbeschreibungssprache eine klar definierte Syntax und Semantik definieren. In diesem Kapitel werden solche Prozessbeschreibungssprachen behandelt, die diese Anforderungen erfüllen. Der Fokus liegt hier auf Ansätzen, die den Projektmitarbeiter während der Prozessausführung unterstützen, da dies im Kern dieser Arbeit (Kapitel 5 und 6) behandelt wird.

Vor ca. 25 Jahren veröffentlichte Leon Osterweil die Publikation „Software Processes are Software too“ [21]. Dies war die erste Publikation im Bereich des Software Engineering, die sich mit der Ausführung von Entwicklungsprozessen beschäftigte. Hierin vertritt Osterweil die These, dass Software-Entwicklungsprozesse mit einer geeigneten Sprache beschrieben werden können, sodass eine Ausführungsumgebung sie interpretieren kann. Durch geeignete Softwarewerkzeuge ließe sich somit eine Unterstützung für die Projektmitarbeiter erreichen.

Osterweils Publikation trat eine regelrechte Welle von Forschungsarbeiten los, die sich mit der Entwicklung von Prozessbeschreibungssprachen befassten, die für die Ausführung von Software-Prozessen geeignet sind. Es ist zu beachten, dass sich die Arbeiten in der ersten Hälfte der Forschungsaktivitäten in diesem Bereich sehr stark mit der Semantik und damit der Ausführung von diesen Sprachen beschäftigt haben, da eine sehr starke und integrierte Werkzeugunterstützung angestrebt wurde. Bei den Arbeiten in den letzten Jahren findet man – insbesondere bei den eingeführten und etablierten Industriestandards – kaum eine Prozessausführung im eigentlichen Sinne mehr. Der Fokus bei den neueren Arbeiten liegt eher im Bereich Prozessdokumentation, -management und Controlling.

In dieser Arbeit werden nicht alle verfügbaren Arbeiten aufgeführt, die sich mit der Ausführung von Vorgehensmodellen beschäftigen, da dies den Rahmen sprengen würde. Arbeiten, die einen guten Überblick über verfügbare Prozessbeschreibungssprachen geben, sind: [22–27]. Der Fokus in diesem Kapitel liegt auf Ansätzen, die sich nicht nur mit der reinen Ausführung von Prozessen beschäftigen, sondern die eine nutzerspezifische Unterstützung anstreben.

Prozesse können auf verschiedene Arten beschrieben werden. Unterschiedlichen Prozessbeschreibungssprachen liegen deshalb unterschiedliche Paradigmen zugrunde. So gibt es beispielsweise imperativen Ansätze: Diese basieren auf imperativen Programmiersprachen. Als Beispiel sei hier Osterweils Ansatz APPL/A [28] genannt, eine Prozessbeschreibungssprache, die auf Ada [29] basiert.

Graphbasierte Ansätze sind eine weitere Möglichkeit, um Prozesse zu beschreiben. Hierbei repräsentieren die Knoten Aktivitäten bei der Prozessausführung. Die Kanten geben eine Ablaufreihenfolge vor. Damit ist die Reihenfolge der Aktivitäten während der Prozessausführung

größtenteils vorbestimmt – bis auf Verzweigungen, deren konkreter Ablauf während der Ausführung durch den Projektmitarbeiter bestimmt werden kann. Graphbasierte Prozessbeschreibungen eignen sich besonders gut zur Prozessdokumentation, da hier die Ablaufreihenfolge genutzt werden kann, um eine Methodik strukturiert zu erklären. Das Problem bei graphbasierten Ansätzen ist, dass sich hieraus eine strikte Abfolge von Aktivitäten ergibt, die oftmals nicht in diesem Maße gewünscht ist. Deshalb unterscheiden sich die gelebten Prozesse teils sehr stark von der Prozessbeschreibung. SPADE [30], [31], SoftPM [32], [33] und EPOS [34], [35] sind Beispiele für graphbasierte Prozessbeschreibungssprachen. SoftPM wird im Kapitel 2.1.5 auf Seite 20, SPADE und EPOS werden in Kapitel 2.1.6 auf den Seiten 22 und 22 vorgestellt.

Die o.g. Probleme treten bei regelbasierten Ansätzen nicht auf. Bei diesen Ansätzen ist eine Ablaufreihenfolge der Aktivitäten nicht explizit dokumentiert. Die Aktivitäten besitzen jedoch Vor- und Nachbedingungen. Eine Aktivität kann während der Prozessausführung nur gestartet werden, wenn alle ihre Vorbedingungen erfüllt sind. Entsprechend kann eine Aktivität nur beendet werden, wenn alle Nachbedingungen erfüllt sind. Der Vorteil bei diesen Ansätzen ist, dass der Projektmitarbeiter die Arbeitsschritte flexibel ausführen kann. Er ist somit nicht an eine vorher festgelegte Ablaufreihenfolge gebunden. Nachteile sind einerseits die schlechte Lesbarkeit der Prozessbeschreibungen sowie die fehlende Möglichkeit, die Prozessbeschreibung als Leitfaden für eine mögliche Reihenfolge der Aktivitäten zu nutzen. Es wäre zwar denkbar, die Vor- und Nachbedingungen so restriktiv zu formulieren, dass sich hieraus eindeutige Abläufe für Leitfäden generieren lassen. Dies ist jedoch nicht im Sinne der regelbasierten Sprachen, da man hierdurch die Vorteile dieser Sprachen verliert (z.B. Flexibilität).

Marvel ist eine solche Prozessbeschreibungssprache, die auf Vor- und Nachbedingungen basiert. Ein Beispielausschnitt einer Marvel-Prozessbeschreibung ist in Abbildung 7 dargestellt. In diesem Beispiel sind zwei Prozessschritte aufgeführt: „archive“ und „all_arch“. Ein Prozessschritt besteht grob aus drei Elementen: Vorbedingungen, durchzuführende Aktion und Nachbedingungen. Marvel unterscheidet zwischen Bedingungen für Automatisierung und Konsistenz. Konsistenzbedingungen müssen durch den Projektmitarbeiter erfüllt werden. Im Bereich der Automatisierungsbedingungen wird der Projektmitarbeiter durch den bereitgestellten „Chaining“-Mechanismus unterstützt: Möchte der Projektmitarbeiter eine Aktivität starten, deren Automatisierungs-Vorbedingungen nicht erfüllt sind, so wird eine Aktivität (oder mehrere Aktivitäten) ausgeführt, die dazu führen, dass diese Vorbedingungen erfüllt werden. Zur Auflösung der Bedingungen baut Marvel hierfür zwei statische Tabellen auf. Ein Beispiel für solche Tabellen ist in Abbildung 8 dargestellt. Die obere Tabelle beschreibt zu jedem Prozessschritt die Vor- und Nachbedingungen (Automatisierungsbedingungen), die erfüllt sein müssen, um ihn zu starten bzw. zu beenden. Die untere Tabelle führt alle Bedingungen auf und wird zweifach verwendet: Einerseits wird sie genutzt, falls der Benutzer einen Prozessschritt starten will, bei dem eine Bedingung nicht erfüllt ist. Hier sind die Prozessschritte aufgeführt, die gestartet werden müssen, um die Bedingung zu erfüllen (in der Tabelle: Spalte „Backward Ptrs“). Andererseits wird diese Tabelle genutzt, um automatisiert Aktionen zu starten, sobald eine Aktion vom Benutzer beendet wurde (und die Nachbedingungen erfüllt sind). Dies ist in der Spalte „Forward Ptrs“ beschrieben.

Merlin [37] ist ebenfalls eine regelbasierte Prozessbeschreibungssprache. Junkermann et al. stellen dabei eine Ausführungsumgebung bereit, die personen- und rollenspezifische Arbeitskontexte (working context) bereitstellt. Hierbei werden Projektmitarbeiter unterstützt, indem die Ausführungsumgebung alle (für die Rolle relevanten) Produkte und ihre Abhängigkeiten darstellt. Zu jedem Produkt werden ebenfalls die möglichen Aktivitäten aufgezeigt. Der Unterschied zu anderen Prozessbeschreibungssprachen ist also der, dass dem Projektmitarbeiter nur ein Ausschnitt des Produkt- und Aktivitätsmodells dargestellt wird. Die in den Arbeiten von Junkermann et al. vorgestellten Kontexte unterscheiden sich jedoch von den Kontexten, die in dieser Arbeit vorgestellt werden in einem wichtigen Punkt: Während in Merlin ein Kontext mit einer Rolle assoziiert ist (also: Jede Rolle hat genau einen Kontext), ist in dieser Arbeit ein Kontext mit einem Produkt assoziiert. Das bedeutet, dass bei einem Projektmitarbeiter

in dieser Arbeit unterschiedliche Kontexte berücksichtigt werden, abhängig von dem Produkt, dass er gerade bearbeitet.

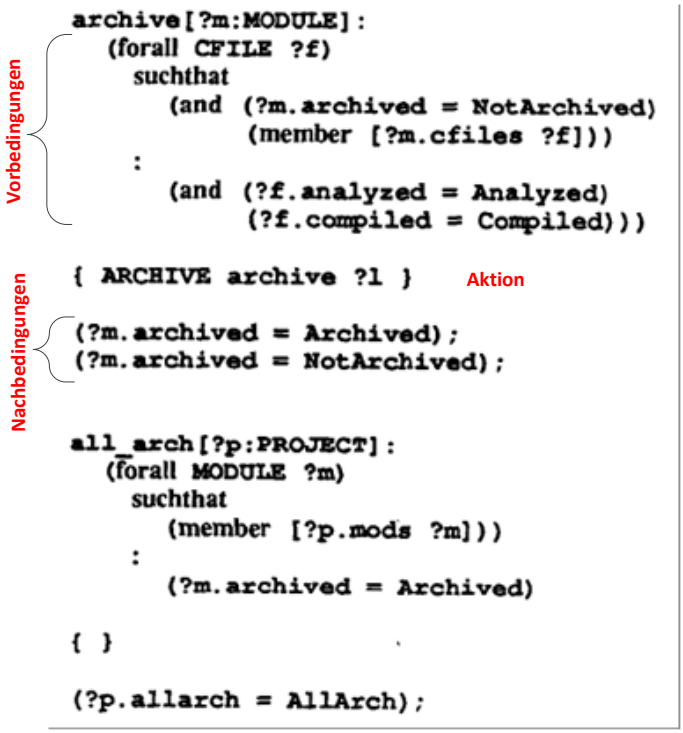


Abbildung 7: Marvel-Beispiel [36]

GRAPPLE [38] ist eine Prozessbeschreibungssprache, die auf Vor- und Nachbedingungen basiert. Sie integriert ein Plangenerierungsverfahren aus dem Bereich der Künstlichen Intelligenz. Hiermit können während der Prozessausführung Pläne (vergleichbar mit einem Projektplan) generiert werden, um von einem Zustand in einen gewünschten anderen Zustand (z.B. Endzustand=Projektende) zu gelangen.

Bisher wurden Ausführungsparadigmen von Prozessbeschreibungssprachen behandelt. Vorgehensmodelle werden jedoch auch angepasst und erweitert. Das folgende Unterkapitel 2.1.3 beschäftigt sich nun mit dieser Anpassung.

Activity	Precondition	Postcondition
ARCHIVE	NotArchived Analyzed Compiled	Archived NotArchived
ALL_ARCH	Archived	AllArch NotAllArch
ANALYZE		Analyzed NotAnalyzed
COMPILE		Compiled NotCompiled

Predicate	Backward Ptrs	Forward Ptrs
Archived	ARCHIVE	ARCHIVE ALL_ARCH
NotArchived	ARCHIVE	
AllArch	ALL_ARCH	
NotAllArch	ALL_ARCH	
Analyzed	ANALYZE	ARCHIVE
Compiled	COMPILE	ARCHIVE

Abbildung 8: Chaining-Tabelle bei Marvel [36]

2.1.3 ANPASSUNG VON VORGEHENSMODELLEN

Vorgehensmodelle werden regelmäßig angepasst und verändert. Dies kann z.B. im Bereich der Prozessverbesserung stattfinden, bei dem ausgeführte Prozesse analysiert werden, um dann entweder ein neues Vorgehensmodell zu erstellen oder um Verbesserungen an einem eingeführten Modell vorzunehmen. Dies nennt Ternité in [39] „Variantenbildung“ und identifiziert entsprechend zwei Schritte: „Initiale Entwicklung des ursprünglichen Vorgehensmodells“ und „Anpassung des ursprünglichen Vorgehensmodells“ (beides in [39], Seite 21 (übersetzt)). Es ist auch eine Anpassung von Vorgehensmodellen ohne vorherige Prozessanalyse möglich. In [12] wird die Einführung eines neuen Werkzeugs als Beispiel hierfür genannt. Dies wird häufig auch als Prozessevolution bezeichnet.

Weiterhin werden Vorgehensmodelle beim Anwenden in einem IT-Projekt projektspezifisch angepasst und konkretisiert. Dies ist zum Beispiel nötig, um Prozessanforderungen des Auftragnehmers umzusetzen, kann aber auch eine generelle „Feinabstimmung“ des Vorgehensmodells sein (z.B. Anpassung an die Projektgröße).

Worin liegen die Unterschiede zwischen diesen beiden Anpassungsebenen (organisationsspezifisch vs. projektspezifisch)? Zum einen werden die Anpassungen von verschiedenen Rollen innerhalb einer Organisation durchgeführt: Der Prozessingenieur passt eine Prozessbeschreibung organisationsspezifisch an, und der Projektleiter ist verantwortlich für die projektspezifische Anpassung. Rollenspezifisch ergeben sich hier auch andere Anforderungen an die Anpassung von Prozessbeschreibungen: Beispielsweise ist für den Prozessingenieur ein Variantenmanagement für Vorgehensmodelle essentiell, ein Projektleiter benötigt dies für seine Anpassungen nicht. Weiterhin verwenden die oben genannten beiden Rollen auch unterschiedliche Werkzeuge. Beim V-Modell XT [19] verwendet der Prozessingenieur zum Beispiel den „V-Modell XT Editor“ zum Anpassen des Vorgehensmodells. Für den Projektleiter ist der „V-Modell XT Projektassistent“ vorgesehen. Beim Rational Unified Process [40] ist dies ähnlich: Der Rational Method Composer [41] ist für beide Rollen vorgesehen. Dem Projektleiter wird mit dem integrierten „Anpassungseditor“ jedoch eine spezielle, vereinfachte Sicht geliefert, die auf die Anpassungen des Projektleiters zugeschnitten ist.

Das folgende Unterkapitel 2.1.4 befasst sich mit dem Stand der Technik im Bereich der erstgenannten organisationsspezifischen Anpassungen von Vorgehensmodellen. Das Unterkapitel 2.1.5 beschreibt den Stand der Technik bezüglich der projektspezifischen Anpassung von Vorgehensmodellen.

2.1.4 ORGANISATIONSSPEZIFISCHE ANPASSUNG VON VORGEHENSMODELLEN

Es gibt zwei Standards, die verfügbar sind und die die organisationsspezifische Anpassung von Vorgehensmodellen über Änderungsoperationen unterstützen [42]: Das **V-Modell XT** [19] und **SPEM** [43]. Beide stellen Operationen bereit, um Änderungen an Vorgehensmodellen zu beschreiben. Das V-Modell XT nennt diese Operationen „Änderungsoperationen", SPEM bezeichnet dies als „Variabilitätsoperationen". Bei SPEM sind die Operationen rein technischer Natur: Hier gibt es z.B. die Operationen „extends" und „replaces". „Extends" spezialisiert dabei ein Vorgehensmodellelement, d.h. das spezialisierende Element erhält alle Eigenschaften des spezialisierten Elements. „Replaces" ersetzt ein existierendes Element vollständig durch ein neues. Das V-Modell XT bietet eine Reihe fachlicher Operationen an. Als Beispiel sei hier das Umbenennen eines Produktes genannt. [44–46] beschreiben die Änderungsoperationen des V-Modell XT im Detail.

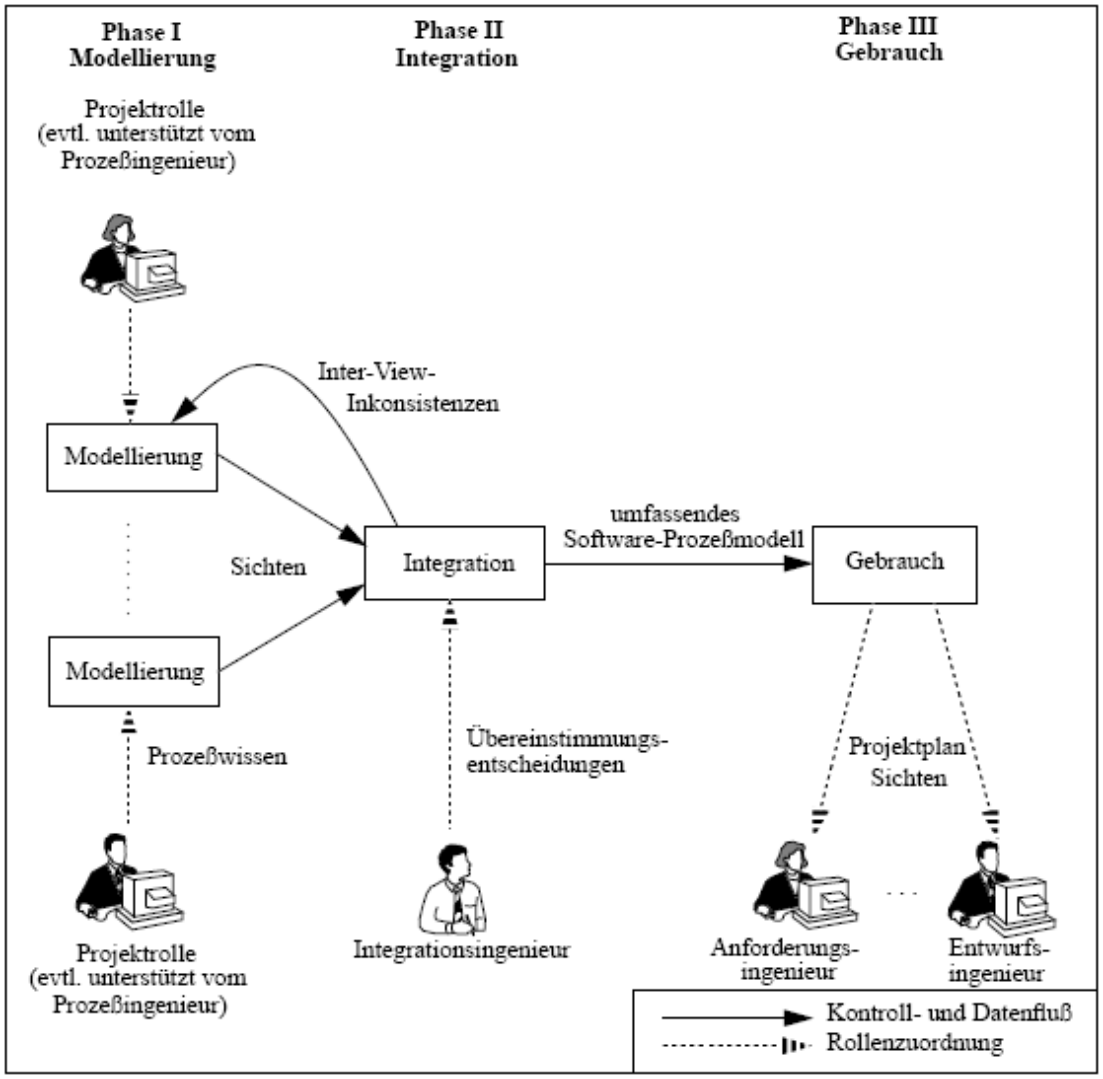


Abbildung 9: Multi-View-Prozess in MVP-L [47]

MVP-L (Multi View Process Language) [47–50] ist ein Prozessframework in dem die Arbeiten von Rombach et al. integriert sind. Motiviert sind die Arbeiten durch die Annahme, dass eine geringe Differenz zwischen Prozessbeschreibung und ausgeführter Prozess sinnvoll ist. Grundidee hinter MVP-L ist, dass sich durch die Kombination von Prozessunterstützung (explizite Prozessbeschreibung und Werkzeugunterstützung zur Ausführung) und Verfahren zur Messung und Bewertung einfacher organisationsweite Prozessverbesserungen durchführen lassen.

Eine Prozessbeschreibung in MVP-L besteht initial aus einzelnen Views, also Sichten auf ein Projekt (siehe Abbildung 9, Phase I). Die einzelnen Sichten werden von den Projektmitarbeitern unabhängig voneinander erstellt und können dann in einer Integrationsphase (siehe in Abbildung 9, Phase II) zu einem projektweiten Softwareprozessmodell integriert werden. Hierfür beschreibt MVP-L Konsistenzbeziehungen, die zwischen den einzelnen Sichten gelten müssen, um die Integration durchzuführen (z.B. in [47]). In der dritten Phase (siehe Abbildung 9) kann das integrierte Modell in einem Projekt ausgeführt werden. Nach Beendigung eines Projektes bietet MVP-L Verfahren zum Messen und Bewerten von Prozessbeschreibungen an, um organisationsweite Prozessverbesserungen durchzuführen.

2.1.5 INITIALE PROJEKTSPEZIFISCHE ANPASSUNG VON VORGEHENSMODELLEN

Die bisher beschriebenen Anpassungen von Vorgehensmodellen finden im Allgemeinen auf Organisationsebene statt. Das bedeutet, die Anpassungen verändern direkt das organisationsweit eingeführte Vorgehensmodell. Es gibt jedoch auch Änderungen „im Kleinen“: Um Vorgehensmodelle in IT-Projekten anwenden zu können, müssen diese an die „lokalen“ Gegebenheiten des Projektes angepasst werden. Als Beispiel können hier Anpassungen an die Projektgröße oder Prozessanforderungen des Auftraggebers genannt werden. In [12] wird dies als „Prozessadaption“ (process adaption) bezeichnet. Münch nennt dies „Projektplanung“ (speziell: „Vorgehensplanung“), siehe [51].

Das **V-Modell XT** bietet für die oben beschriebenen Änderungen das „Tailoring“ an. Dieser Mechanismus ist im Werkzeug „V-Modell XT Projektassistent“ integriert. Der Projektassistent lädt ein organisationsspezifisches Vorgehensmodell und bietet eine Reihe an Projektmerkmalen an (siehe Abbildung 10 und [44], [45]). Mit diesen Projektmerkmalen ist eine fachliche Klassifizierung des Projektes möglich. Nach Festlegung der Projektmerkmale können a) ein projektspezifisches Vorgehensmodell (in Form einer XML-Datei), b) projektspezifische Dokumentvorlagen und c) ein initialer Projektplan generiert werden.

Das projektspezifische Vorgehensmodell ist eine Teilmenge des organisationsspezifischen Vorgehensmodells, d.h. nicht relevante Teile sind entfernt. Die relevanten Teile werden durch das Belegen der Parameter im Projektassistent festgelegt. Das projektspezifische V-Modell XT lässt sich nachfolgend durch ein Werkzeug auslesen und nutzen.

Für jeden Dokumenttyp (z.B. Lastenheft, SW-Spezifikation) wird eine Dokumentvorlage generiert. Diese enthält einen Vorschlag für die Strukturierung. Dieser Aufbau wird wiederum aus den Projektparametern abgeleitet. Die Dokumentvorlagen lassen sich für die Erstellung der Dokumentexemplare (z.B. SW-Spezifikation für Modul X) nutzen. Weiterhin wird auch ein Vorschlag für die Struktur einer Produktbibliothek gemacht.

Der Projektassistent erlaubt es, eine rudimentäre Projektplanung durchzuführen. Hier können Entscheidungspunktexemplare (siehe Abbildung 3; hier ist ein Ablauf des V-Modell XT auf Typebene dargestellt) eingeplant werden. Ein Teil einer solchen Planung ist in Abbildung 12 dargestellt. Der mit dem Projektassistenten initial erstellte Projektplan lässt sich weiter mit gängigen Projektplan-Werkzeugen verarbeiten und präzisieren. Zum Beispiel lassen sich

Aktivitätsexemplare (zum Beispiel des Aktivitätstyps „Systemarchitektur erstellen“, siehe Abbildung 3 auf Seite 12) in die Planung integrieren.

Das spezifische V-Modell XT und der Projektplan[13] bilden gemeinsam das projektspezifische Vorgehensmodell. Bei dem Modell sind gegenüber dem (nicht projektspezifischen) organisationsweiten Modell nicht relevante (durch die fachlichen Tailoring-Parameter festgelegte) Inhalte entfernt. Der Projektplan detailliert die Ablaufreihenfolge im Projekt, die im V-Modell XT nur generisch beschrieben ist. Jedoch können die Aktivitätstypen an sich nicht projektspezifisch erweitert oder angepasst werden. D.h. hier ist es nicht vorgesehen, projektspezifisches Vorgehen in den Aktivitätsbeschreibungen zu berücksichtigen.

In diese Kerbe schlägt **SoftPM**. Dies ist eine auf Petri-Netzen basierte Sprache zum Erstellen von Prozessmodellen. Eine Übersicht über SoftPM ist hier zu finden: [32], [33], [52], [53]. Mit SoftPM ist es möglich, projektspezifische Änderungsoperationen auf dem organisationsweiten Vorgehensmodell (ähnlich wie die Änderungsoperationen beim V-Modell XT zur organisationsspezifischen Anpassung, siehe Kapitel 2.1.4) zu beschreiben. SoftPM bietet vier Änderungsoperationen, die auf ein bestehendes Prozessmodell angewendet werden können: ProcessEntityAddition, ProcessEntityDeletion, ProcessEntitySplitting und ProcessEntity-Merging. Mit den ersten beiden Operationen können Prozessentitäten (z.B. Aktivitäten, Artefakte/Produkte) hinzugefügt und gelöscht werden, die letzten beiden ermöglichen ein Teilen oder Zusammenlegen von Entitäten. Ein Beispiel der Operation ProcessEntityMerging ist in Abbildung 11 dargestellt. Oben in der Abbildung ist ein Teil des organisationsspezifischen Vorgehensmodells dargestellt, unten befindet sich das nach Anwendung der Operation resultierende projektspezifische Modell.

Der Rational Unified Process (**RUP**) [40] ist ein kommerzielles Vorgehensmodell und verfügt über ein ähnliches Verfahren wie SoftPM: Mit dem Rational Method Composer [41] ist es analog möglich, mit Operationen auf dem organisationsspezifischen RUP projektspezifische Anpassungen vorzunehmen. Eine Dokumentation des Rational Method Composer ist online verfügbar: [54].

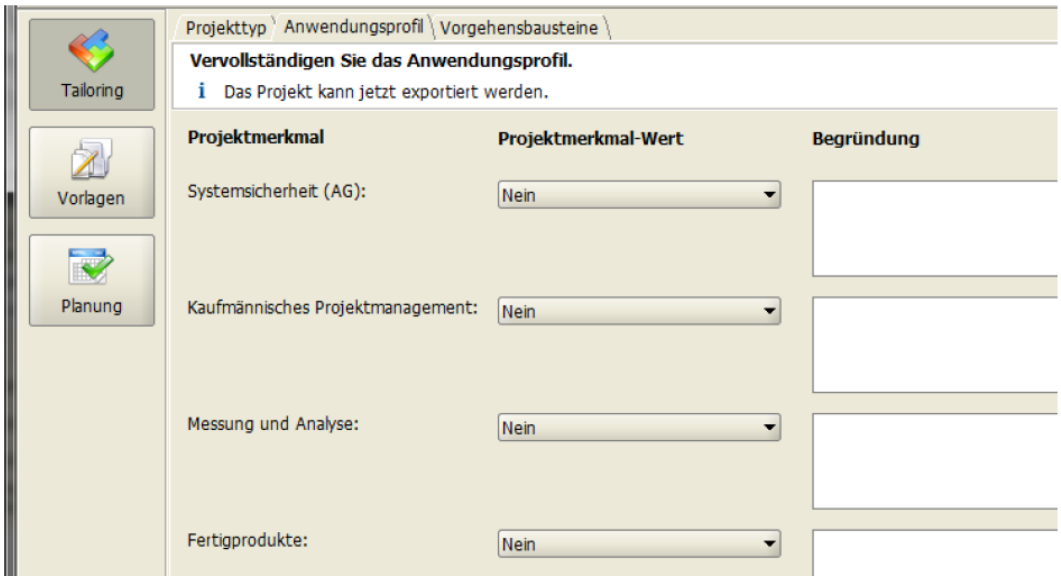


Abbildung 10: Tailoring beim V-Modell XT Projektassistent [45]

[13] Der Projektplan ist im V-Modell XT ein Produkt (wie beispielsweise auch ein Dokument „Sicherheitskonzept“ o.ä.). Diese Arbeit sieht beim V-Modell XT den Projektplan jedoch als integralen Bestandteil an. Hier wird nämlich das Vorgehensmodell projektspezifisch angepasst (zum Beispiel wird eine Projektdurchführungsstrategie verfeinert, indem sie ausgeplant wird).

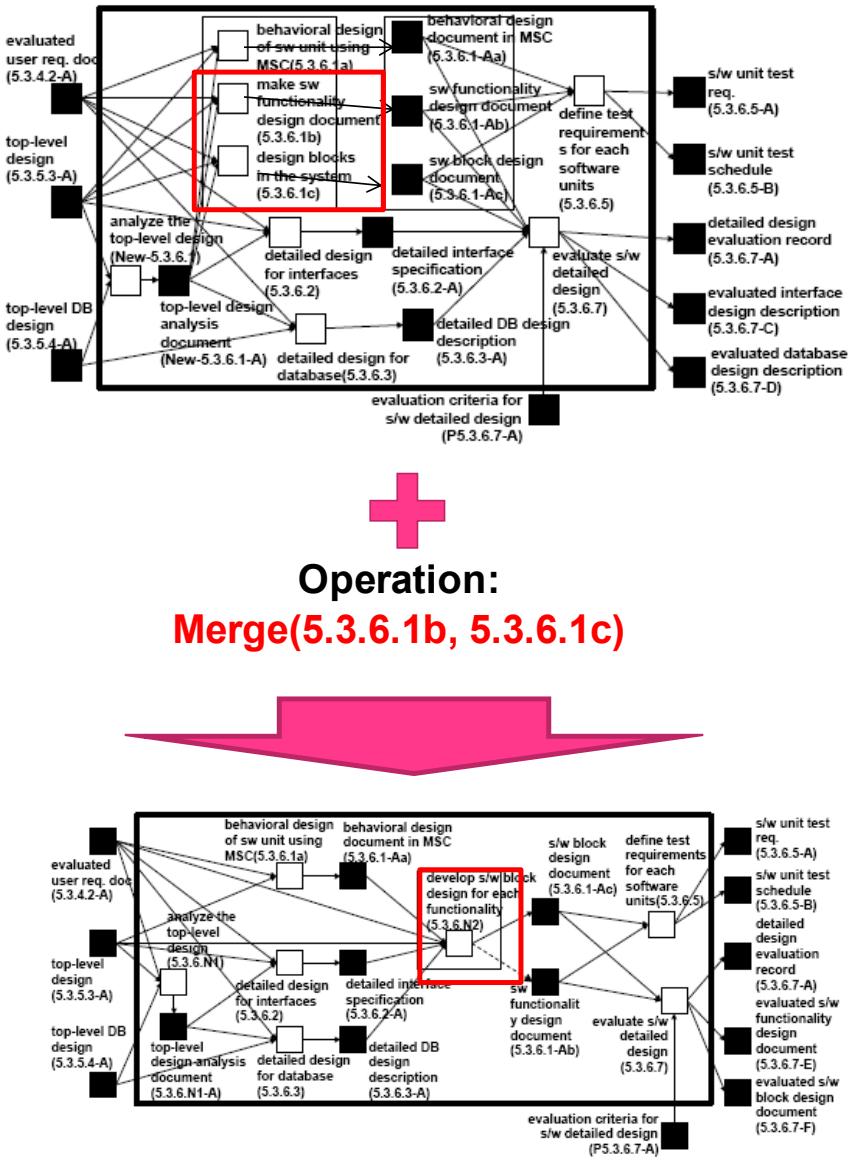


Abbildung 11: Beispiel einer projektspezifischen Anpassung bei SoftPM [53]

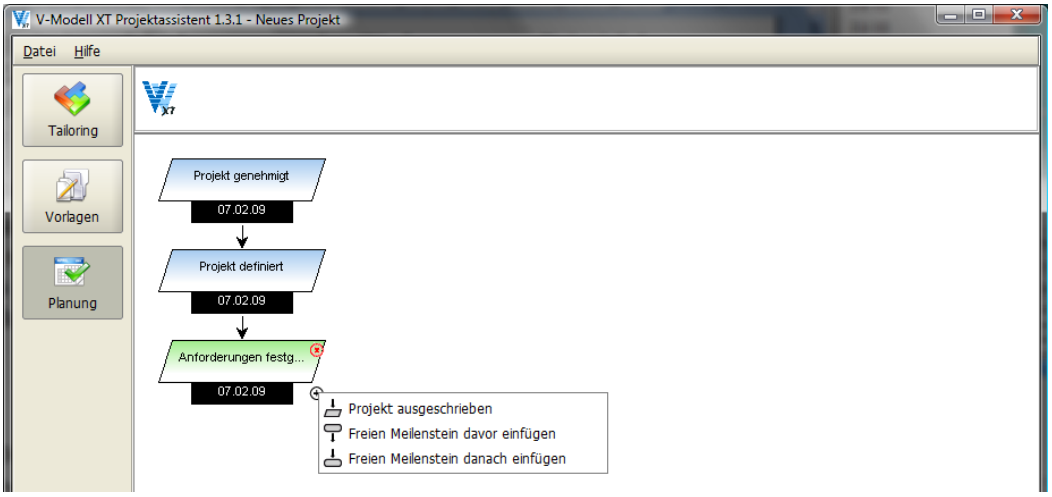


Abbildung 12: Projektplanung beim V-Modell XT [45]

2.1.6 ANPASSUNG VON VORGEHENSMODELLEN WÄHREND DER AUSSFÜHRUNG

Vorgehensmodelle während ihrer Ausführung anzupassen geschieht quasi analog zu der projektspezifischen Anpassung, jedoch müssen weitere Dinge beachtet werden: Es muss beispielweise festgelegt werden, was bei einer Änderung eines Ausschnitts des Vorgehensmodells geschehen soll, der gerade im Projekt ausgeführt bzw. instanziiert ist. Wenn also z.B. der Ablauf der Aktivitätsbeschreibung „Komponente spezifizieren" geändert wird, muss klar sein, was mit den instanziierten Aktivitäten „Komponente X spezifizieren" und „Komponente Y spezifizieren" geschehen soll. Diese befinden sich nämlich in einem eindeutigen Zustand, der nicht verloren gehen sollte. SPADE und EPOS nehmen sich dieses Themas an und werden in diesem Kapitel beschrieben.

Auf der anderen Seite gibt es auch Ansätze, die bewusst die Arbeiten des Projektmitarbeiters in den Vordergrund rücken und das Vorgehensmodell als eine Art Anleitung sehen. Spontane Änderungen des aktuellen Ablaufs sind hier – innerhalb gewisser Grenzen – möglich. PROSYT und DM_PSEE bieten eine solche „tolerante Ausführung" und werden ab Seite 23 beschrieben.

SPADE ist eine Prozessbeschreibungssprache, die auf Petri-Netzen beruht. Ein Beispiel eines Vorgehensmodellausschnitts ist in Abbildung 13 dargestellt. Es ist u.a. ein Interpreter, der die Ausführung von Vorgehensmodellen erlaubt, enthalten. Die Konzepte zu SPADE sind hier zu finden: [30], [31].

Bandinelli und Fugetta haben in ihren Arbeiten um SPADE festgestellt, dass Änderungen des Vorgehensmodells während der Ausführung, also während eines IT-Projektes, nötig sein können. Unkritisch sind Änderungen an Aktivitäten im Vorgehensmodell (zum Beispiel in Abbildung 13), zu denen es keine Instanzen im ausgeführten Prozess gibt. Diese können einfach geändert werden. Zu den Aktivitäten, zu denen es Instanzen gibt, bieten Bandinelli und Fugetta zwei Strategien an: „lazy strategy" und „eager strategy". Die erstgenannte Möglichkeit beruht darauf, dass sich Änderungen am Vorgehensmodell nicht auf laufende Aktivitäten auswirken. Lediglich zukünftig gestartete Aktivitäten werden also mit den Änderungen ausgeführt. Die „eager strategy" lässt Änderungen an instanziierten Aktivitäten zu. Hierfür wird der Zustand der laufenden Aktivitäten gesichert. Mit der Beschreibung der „neuen", geänderten Aktivität müssen Regeln zur Zustandsüberführung formuliert werden. Diese Regeln werden genutzt, um die gesicherten Zustände der „alten" Aktivitätsinstanzen auf die „neuen" Aktivitätsinstanzen zu überführen.

EPOS [34], [35], [55], [56] stellt eine Sprache für die Beschreibung von Vorgehensmodellen inkl. einer Ausführungsumgebung zur Verfügung. Bei den Vorgehensmodellen wird zwischen einer organisationsweiten, generischen Vorgehensbeschreibung (generic model) und einer projektspezifischen, geplanten Vorgehensbeschreibung (instance model) unterschieden. Die generische Prozessbeschreibung wird dabei mittels Vor- und Nachbedingungen beschrieben. Die projektspezifische Beschreibung kann mit einem netzbasierten Vorgehensplan verglichen werden.

Mit der bereitgestellten Softwarenunterstützung ist es möglich, auf verschiedene Weisen Anpassungen an Vorgehensmodellen durchzuführen. EPOS bietet hierfür eine Versionierung der Modellelemente sowie einen Transaktionsmechanismus. Neben der Anpassung des organisationsweiten Vorgehensmodells ist es möglich, das projektspezifische Modell während der Ausführung zu ändern. EPOS stellt Änderungsoperationen zur Verfügung, die entweder auf der Typebene (generic model), oder auf der Instanzebene (instance model) ausgeführt werden. Bei Änderung der Vorgehensmodellinstanzen kann entschieden werden, wie sich diese Änderung auf die Typ-/Instanzebene auswirkt. Zum Beispiel:

a) Bei Änderung einer Instanz wird der entsprechende Typ nicht geändert. Das heißt, hier ergibt sich eine Inkonsistenz zwischen Instanzmodell und Typmodell, die nicht aufgelöst wird.

b) Bei Änderung einer Instanz werden der entsprechende Typ sowie alle Instanzen dieses Typs geändert.

c) Bei Änderung einer Instanz kann entschieden werden, ob der entsprechende Typ sowie alle Instanzen dieses Typs, oder nur alle Instanzen dieses Typs (und der Typ an sich nicht) geändert werden.

EPOS wird kritisiert [23], da hier viele verschiedene heterogene Paradigmen und Konstrukte kombiniert werden, ohne eine klare, formale Semantik zu beschreiben. Dies macht es schwierig, ein klares Bild von den Fähigkeiten und Funktionalitäten von EPOS zu erhalten.

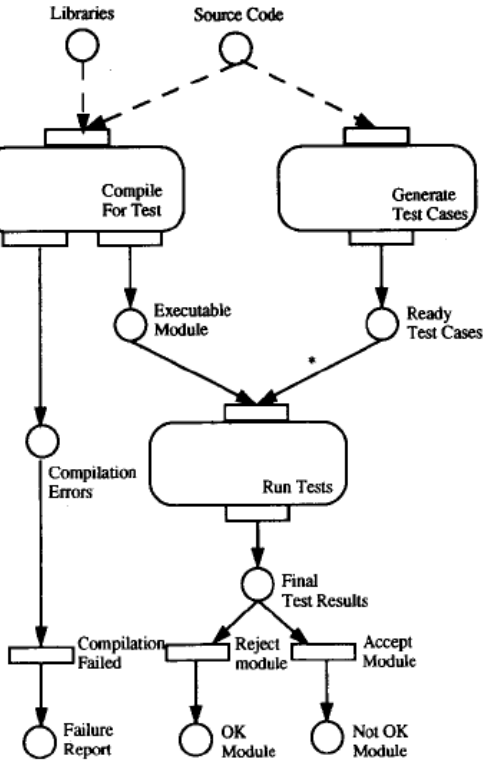


Abbildung 13: Ausschnitt eines SPADE-Modells [30]

Die Änderungsverfahren von SPADE und EPOS setzen voraus, dass Projektmitarbeiter ihre Arbeitweisen / ihr Vorgehen (während eines Projektes) nicht ändern, bevor das zugrunde liegende Vorgehensmodell geändert wurde. Kabbaj et al. [4] sowie Cugola [57] greifen diese Problematik auf und beschreiben, dass sich das Änderungsverfahren von SPADE bei großen, projektweiten Änderungen eigne, nicht jedoch bei kleinen, manchmal nur temporären Differenzen zwischen Prozessbeschreibung und ausgeführtem Prozess.

Viele Sprachen für Vorgehensmodelle sind aktivitätszentriert: Aktivitäten stehen im Mittelpunkt und sind die zentralen Elemente. **PROSYT** [10], [57], [58] stellt eine Sprache für Vorgehensmodelle bereit, bei denen Produkte die zentralen Einheiten für die Beschreibung eines Prozesses sind. Zu den einzelnen Produkttypen (z.B. Produkttyp „Spezifikation“) können Attribute definiert werden (z.B. „edit-Time“, „run-test-Time“). Weiterhin können Zustände beschrieben werden, die sich logisch aus den Belegungen der Attribute ableiten lassen (z.B. „tested: run-test-Time>edit-Time“). Des Weiteren können eine Menge von Operationen beschrieben werden. Diese Operationen dienen dem Erstellen des Produktes, dem sie angehören

(z.B. Operation „deliver“). Jede Operation kann dabei einen benötigten Produktzustand (Guard „tested“) enthalten. Um diese Operation auszuführen, muss sich das zugrunde liegende Produkt in diesem Zustand befinden. Weiterhin beinhaltet eine Operation beispielsweise eine Rolle, die diese Operation ausführt. Eine Abfolge der Operationen zur Erstellung eines Produktes wird nicht explizit beschrieben.

Um „spontanen“ Inkonsistenzen zwischen dem Vorgehensmodell und dem ausgeführten Prozess zu begegnen, werden von Cugola [57] eine Menge von „Abweichungspunkten“ (deviation) zwischen Vorgehensmodell und Prozess definiert. Es gibt z.B. die „a) User deviation“, die erfüllt ist, wenn eine Operation von einer Rolle ausgeführt wird, die nicht im Vorgehensmodell vorgesehen ist. Ein weiteres Beispiel ist die „b) State deviation“. Diese Abweichung tritt ein, wenn eine Operation ausgeführt wird, wobei der benötigte Produktzustand (guard) nicht erfüllt ist. Die Anzahl und Art dieser Abweichungspunkte sind fest definiert. PROSYT definiert eine Ordnung über diese Punkte (z.B. „a) User deviation“ < „b) State deviation“). Weiterhin gibt es eine feste Menge an „deviation handling policies“. Diese beschreiben Möglichkeiten, was zu tun ist, wenn eine Abweichung von einem bestimmten Typ erkannt wird. Als Beispiel sei hier „1) abort (Abbruch der Operation)“ oder „2) Inform user and continue“ genannt. Zu jeder Rolle und zu jedem Abweichungspunkt im Vorgehensmodell ist es nun möglich, diese einer „deviation handling policy“ zuzuordnen, zum Beispiel: (Architekt, a → 2). Will eine Rolle nun eine Operation starten und wird eine Abweichung festgestellt, so wird die entsprechende „policy“ ausgeführt. Sind mehrere Abweichungen vorhanden (z.B. a) und b)), ist die Abweichung mit der höchsten Ordnung ausschlaggebend für das Verhalten.

DM_PSEE steht für „Deviation Management PSEE (Process-centered Software Engineering Environment)” [4], [5]. Die Arbeiten von Kabbaj, Lbath und Coulette beschreiben ein Meta-Modell für Vorgehensmodelle (siehe hierzu [59]), das auf SPEM [43] beruht. Ein Beispielausschnitt eines Vorgehensmodells ist in Abbildung 14 dargestellt. Die Motivation der Arbeiten ist, dass es in IT-Projekten permanent Unterschiede zwischen ausgeführtem Prozess und Vorgehensmodell gibt und dies auch sinnig ist. Diese Unterschiede sind oftmals temporär und werden ad-hoc entschieden. Deshalb ist es nicht sinnvoll, Änderungen direkt im Vorgehensmodell zu übernehmen, wie dies beispielsweise bei SPADE (s.o.) zu finden ist. Folglich werden die Abläufe der Aktivitäten im Vorgehensmodell nur als Vorschlag während der Prozessausführung gesehen. Der Projektmitarbeiter kann in einem gewissen Rahmen die Reihenfolge der Aktivitäten selbst bestimmen. Hierfür ist es möglich, „Abweichungsregeln“ (deviation rules) zu formulieren und mit Vorgehensmodellelementen (z.B. Aktivitäten) zu verknüpfen. Diese Abweichungsregeln geben vor, inwieweit ein ausgeführter Prozess von dem Vorgehensmodell (der Prozessbeschreibung) abweichen darf. In den Abweichungsregeln werden Abweichungen des ausgeführten Prozesses zu dem Vorgehensmodell definiert, und hierzu können Toleranzaussagen beschrieben werden. Diese Aussagen beschreiben also inwieweit eine solche Abweichung toleriert wird oder nicht. Folgende Abweichungen können z.B. formuliert werden: a) „Starten einer Aktivität, deren Vorbedingung nicht erfüllt ist“ oder b) „Nicht-einbeziehen einer Rolle, die an einer Aktivität mitwirkt“. Zu a) könnte folgende Toleranzaussage formuliert werden: „a) wird toleriert, wenn die Eingabeprodukte im Zustand „vorgelegt“ sind und mindestens einen Erfüllungsgrad von 90% besitzen“.

Die Unterschiede zwischen DM_PSEE und PROSYT sind nun die, dass DM_PSEE mehr Flexibilität bietet: Die Abweichungen sind nicht fest definiert wie bei PROSYT, sondern können flexibel beschrieben werden und können an verschiedene Prozesselemente (z.B. Artefakte, Produkte) angehängt werden. Des Weiteren definiert DM_PSEE eine Reihenfolge der Aktivitäten. Diese Reihenfolge wird für die Nutzerunterstützung (guiding) während der Prozessausführung genutzt.

In dieser Arbeit werden mit Hilfe von maschinellen Lernverfahren (speziell: Sequenzvorhersageverfahren) Verbesserungen im Bereich des Software Engineering erzielt. Das folgende Unterkapitel 2.2 beschäftigt sich mit dem Stand der Technik zu diesem Thema.

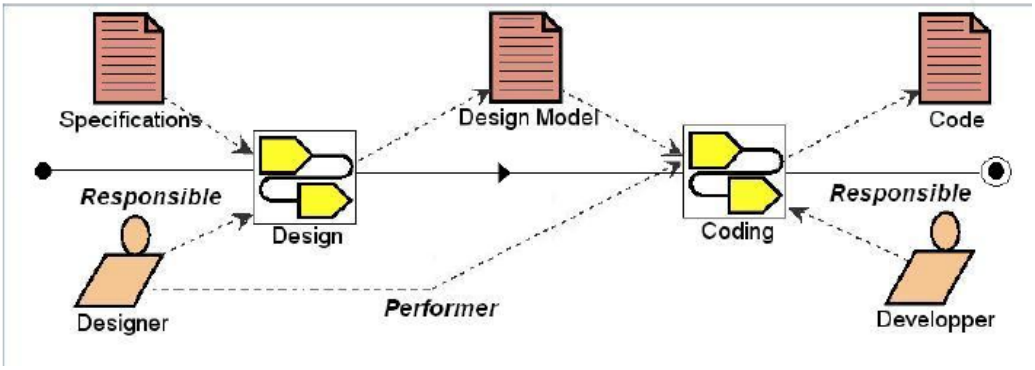


Abbildung 14: Auschnitt eines Vorgehensmodells bei DM_PSEE [4]

2.2 MASCHINELLES LERNEN UND SEQUENZVORHERSAGE

Maschinelles Lernen bedeutet, ein Computersystem so zu entwickeln, dass es mit Hilfe von Beispiel- bzw. Erfahrungsdaten aus der Vergangenheit Wissen ableiten kann [60]. Ein solches System lernt also nicht „einfach auswendig“, sondern versucht, die Beispieldaten zu verallgemeinern, um so gute Vorhersagen zu treffen. Zur Anwendung eines maschinellen Lernverfahrens ist es deshalb notwendig, dass die Erfahrungsdaten nicht zufällig verteilt sind, sondern eine Struktur besitzen. Der Reiz des Maschinellen Lernens besteht darin, dass wir – die Menschen – diese Struktur nicht notwendigerweise kennen oder erkennen müssen.

Es gibt viele Szenarien, in denen sich die Anwendung von maschinellen Lernverfahren bewährt hat. Alpaydın nennt hier u.a. Probleme, die sich „zeit- oder umweltabhängig“ ändern [60]. Genau ein solches Problem wird in dieser Arbeit adressiert: Der Prozessausschnitt als Anwendungsbeispiel, der in Kapitel 3 dargestellt wird, beschreibt unterschiedliches, „umweltabhängiges“ Vorgehen für die Dekomposition und den Entwurf eines Software-Systems. Unser Problem ist dabei, den Projektmitarbeiter zu unterstützen, indem dieses umweltabhängige Vorgehen erkannt wird. Dabei kann der Projektmitarbeiter während seiner Arbeit unterstützt werden, indem die nächsten Arbeitsschritte vorgeschlagen werden. Die Umwelt wird in dieser Arbeit konsequent als „Kontext“ bezeichnet.

Das in Kapitel 6 auf Seite 99 vorgestellte Verfahren zur Nutzerunterstützung bedient sich des Maschinellen Lernens. Es nutzt ein Verfahren zur Sequenzvorhersage und passt dieses domänenspezifisch an. Prozesse für IT-Projekte sind dabei unsere Domäne. Wir erweitern das Verfahren, um relevante Informationen (das sind unsere Kontexte) aus den Beispieldaten zu nutzen, um bessere Vorhersagen zu treffen.

Das folgende Kapitel 2.2.1 gibt einen kurzen Überblick über das Maschinelle Lernen und kategorisiert es. Kapitel 2.2.2 beschäftigt sich dann mit einer speziellen Klasse des Maschinellen Lernens, der Sequenzvorhersage. Ein solches Verfahren wird in dieser Arbeit erweitert. Dieses Kapitel gibt einen Überblick über bereits vorhandene Arbeiten.

2.2.1 MASCHINELLES LERNEN – EINORDNUNG UND ÜBERBLICK

Es gibt viele Kategorien, um das weite Feld des Maschinellen Lernens zu klassifizieren. Beispielsweise kann zwischen „Batch-Lernen“ und „Kontinuierlichem Lernen“ unterschieden werden. Bei Ersterem sind alle Erfahrungsdaten direkt vorhanden und können eingesetzt werden. Beim „Kontinuierlichem Lernen“ stehen nicht alle Beispieldaten direkt zur Verfügung, sondern fallen sukzessive an. Eine Klassifizierung, die in der Literatur oft gefunden wird, besteht aus den folgenden drei Kategorien[14]:

- Überwachtes Lernen,
- Bestärkendes Lernen und
- Unüberwachtes Lernen.

Beim **Überwachten Lernen** bestehen die Erfahrungsdaten aus Ein- und Ausgabepaaren. Das bedeutet, dass zu jeder Eingabe in den Erfahrungsdaten eine korrekte Ausgabe verfügbar ist. Das Ziel des Überwachten Lernens ist dabei, eine Abbildung der Ein- auf die Ausgabewerte zu finden.

[14] Diese Klassifizierung und die beschreibenden Beispiele sind aus [60] entnommen.

Ein bekanntes Beispiel für Überwachtes Lernen ist die **Klassifikation**: Das Ziel ist hier, durch eine Menge von Eingabewerten auf eine Klasse schließen zu können. Ein einfaches Beispiel, bei dem *eine* Klasse anhand von positiv/negativ Beispielen zu erlernen ist, ist die Klassifikation von Autos. Ziel ist es, zu erlernen, ob ein Auto in die Klasse „Familienwagen" einzuordnen ist oder nicht. Nehmen wir an, dass eine domänenspezifische Analyse ergeben hätte, dass die Merkmale „Preis" und „Motorleistung" die relevanten Parameter seien, um eine Klassifikation vornehmen zu können. Die Beispieldaten bestehen dann aus einer Menge von Eingabedaten (Preis, Motorleistung) sowie Ausgabedaten. Die Ausgabedaten sind dabei die Informationen, ob die Eingabedaten zur Klasse „Familienwagen" gehören oder nicht (z.B. Familienwagen=ja). Das Ziel ist es nun, eine Beschreibung zu finden, die auf alle positiven (Familienwagen=ja) und auf keine negativen Beispiele zutrifft.

Eine weitere Klasse an Problemen, die zum Überwachten Lernen gehören, sind die **Regressionsprobleme**. Die Eingabedaten sind hierbei vergleichbar mit den Eingabedaten bei den Klassifikationsproblemen. Die Ausgabedaten sind bei diesen Problemen jedoch Zahlenwerte. Ein Beispiel für ein solches Problem ist die Vorhersage des Preises eines gebrauchten Autos: Eingaben sind, wie bei dem Klassifikationsbeispiel, Attribute wie „Neupreis", „Marke", „Baujahr". Die Ausgabe ist dann jedoch der Preis des Autos.

Bei manchen Problemstellungen ist die unmittelbare Ausgabe eines Systems nicht relevant. Jedoch ist die Taktik, also eine Sequenz von Ausgaben, wichtig. Eine einzelne Ausgabe kann deshalb nicht unbedingt als richtig oder falsch bezeichnet werden. Einzelne Ausgaben sind dann richtig, wenn sie Teil einer Taktik sind, die zum Ziel führt. Ein Lernen von solchen Problemen nennt man **Bestärkendes Lernen**. Ein klassisches Beispiel für diese Problemstellungen sind Brettspiele. Die Ausgaben sind dabei die einzelnen Züge. Entscheidend für das Ziel, nämlich das Spiel zu gewinnen, ist dabei nicht ein einzelner Zug, sondern die Sequenz von Zügen bis zum Ziel.

Beim **Unüberwachten Lernen** bestehen die Erfahrungsdaten nur aus Eingabedaten. Die korrekten Ausgabedaten stehen nicht zur Verfügung oder sind nicht bekannt. Ziel ist es, Regelmäßigkeiten in den Eingabedaten zu erkennen und zu erlernen. Ein Beispiel für Unüberwachtes Lernen ist das **Clustering**. Das Ziel besteht hierin, Cluster (Häufungen) von Eingabedaten zu erkennen. Eine Beispielanwendung wäre hier das Clustering von Kunden einer Firma, um eine „natürliche Gruppierung" zu erhalten.

Das **Erlernen von Assoziationen** ist ein weiteres Gebiet des Unüberwachten Lernens. Ein Beispiel wäre hier die Warenkorbanalyse eines Supermarktes: Ein Kunde, der X kauft, kauft auch üblicherweise Y. Ziel beim Erlernen von Assoziationen ist es, die Beziehungen zwischen X und Y zu finden. Dies wird üblicherweise durch das Erlernen von bedingten Wahrscheinlichkeiten realisiert: $P(Y|X)$. X ist dabei das Produkt, das der Kunde schon gekauft hat und Y ist das Produkt, das durch X bedingt ist. Wird beispielsweise eine bedingte Wahrscheinlichkeit $P(Chips|Bier) = 0{,}7$ erlernt, so bedeutet dies: „70 Prozent aller Kunden, die Bier kaufen, kaufen auch Chips" [60].

Ein weiteres Gebiet des Unüberwachten Lernens ist die **Sequenzanalyse**[15]. Die Anwendungsgebiete sind hier sehr vielfältig: Spracherkennung und DNA-Sequenzierung sind nur zwei Beispiele, bei denen die Sequenzanalyse Anwendung findet.

Die Sequenzanalyse ist ein weites Feld mit vielfältigen Problemstellungen. In [61] wird die Sequenzanalyse in folgende Gebiete unterteilt: Sequenz-Erkennung, Sequenz-Generierung, Sequenz-Vorhersage und Zielorientierte Sequenz-Vorhersage.

[15] Englisch: Sequence Learning

Bei der **Sequenz-Erkennung** soll, basierend auf der erlernten Erfahrung, erkannt werden, ob eine Eingabe-Sequenz valide ist oder nicht: $s_i, s_{i+1}, \dots, s_j \rightarrow$ ja oder nein. Mit s_x werden dabei die einzelnen Sequenzschritte bezeichnet.

Die **Sequenz-Generierung** schlägt, basierend auf der Erfahrung und einer Eingabe-(Teil-)Sequenz, eine nachfolgende Sequenz vor: $s_i, s_{i+1}, \dots, s_j \rightarrow s_{j+1}, s_{j+2}, \dots$

Die **Sequenz-Vorhersage** versucht, den nächsten Sequenzbeschritt bei einer gegeben Teil-Sequenz vorherzusagen: $s_i, s_{i+1}, \dots, s_j \rightarrow s_{j+1}$. Hier wiederum gibt es vielfältige Möglichkeiten: Bei $i = 1$ wird die gesamte Sequenz-Historie für den Vorschlag des nächsten Sequenz-Schrittes verwendet. Auf der anderen Seite wird bei $i = j$ nur der letzte beobachtete Sequenzschritt für die Vorhersage betrachtet.

Bei der **Zielorientierten Sequenz-Vorhersage** wird, wie bei der obigen Sequenz-Vorhersage, der nächste Sequenz-Schritt vorhergesagt. Neben der Eingabe-Sequenz wird jedoch auch ein angestrebter Ziel-Sequenzschritt s_z angegeben: $s_i, s_{i+1}, \dots, s_j; s_z \rightarrow s_{j+1}$.

In dieser Arbeit wird ein für Software-Prozesse spezifisches Verfahren für die Sequenz-Vorhersage (insb. in Kapitel 6 auf Seite 99) vorgestellt. Im folgenden Kapitel 2.2.2 werden deshalb vorhandene Verfahren für die Sequenz-Vorhersage aufgezeigt.

2.2.2 SEQUENZVORHERSAGE

Dieses Kapitel beschäftigt sich mit dem Stand der Technik in Bezug auf Verfahren für die Sequenzvorhersage. Die Domänen für die Anwendung von Sequenzvorhersage sind vielfältig: Einige Verfahren werden beispielsweise für die Vorhersage von (textuellen) UNIX-Kommandos genutzt [62–64]. Andere Ansätze versuchen intelligente, grafische Benutzeroberflächen zu schaffen [65], [66], wieder andere wollen Verbesserungen in der Gebäudeautomatisierung („Smart Home“) erzielen [67–73]. Des Weiteren gibt es Verfahren, die angefangene, geschriebene Sätze vervollständigen [74], [75] (als Anwendung wird hier beispielsweise ein E-Mail-Programm in einem Call-Center genannt, das den Mitarbeitern hilft, schnell Standard-E-Mails zu verfassen), oder Ausfälle von Hardware[16] (z.B. Telekommunikationssysteme) vorhersagen [76]. Eine weitere, sehr bekannte Anwendung ist die „autocomplete“-Funktion, beschrieben in [77], die beispielweise bei Google implementiert ist: Beginnt ein Benutzer etwas in eine Eingabemaske zu schreiben, werden sukzessive Vorschläge zur Vervollständigung gemacht.

Im Folgenden werden einige Verfahren zur Sequenzvorhersage vorgestellt. Im Mittelpunkt stehen dabei Verfahren für die Benutzerinteraktion (z.B. grafische Benutzeroberflächen), da sich die dieser Arbeit zugrunde liegende Domäne hiermit am ehesten vergleichen lässt. Ein guter Überblick über einige dieser Verfahren ist in [78] zu finden.

IPAM [62], [63] ist ein Verfahren, welches ein Markov-Modell erster Ordnung einsetzt. Das bedeutet, es nutzt für die Vorhersage nur den letzten beobachteten Sequenzschritt. Hierfür speichert IPAM eine Liste von bedingten (z.B. $P(y|x)$) und nicht bedingten Wahrscheinlichkeiten (z.B. $P(x)$) und aktualisiert diese. Die Aktualisierung der Wahrscheinlichkeiten erfolgt rekursiv. Nehmen wir an, wir hätten eine Sequenz $x_1, \dots, x_i, x_{i+1}$ beobachtet und wollten die letzte Beobachtung x_{i+1} lernen. Folgende Wahrscheinlichkeiten würden dann aktualisiert werden:

[16] Die Algorithmen zur Vorhersage von Hardware-Ausfällen unterscheiden sich grundlegend von den anderen Verfahren zur Sequenz-Vorhersage, denn hier ist es wichtig, seltene Ereignisse vorherzusagen und nicht – wie bei den anderen Verfahren häufige Vorkommnisse.

$$P(y|x_i) = \begin{cases} \alpha \cdot P(y|x_i) + (1-\alpha) \text{ falls } y = x_{i+1} \\ \alpha \cdot P(y|x_i) \text{ sonst} \end{cases} \quad (1)$$

Die Konstante α ist ein Wert zwischen null und eins. Das bedeutet, bedingte Wahrscheinlichkeiten, die auf die aktuelle Beobachtung passen, steigen in der Schranke [0,1] an, Wahrscheinlichkeiten, die nicht passen, fallen entsprechend.

Der Nachteil bei diesem Verfahren ist, dass die Vorhersage immer nur durch die letzte Beobachtung bedingt ist.

ONISI [65] ist ein Sequenz-Vorhersage-Verfahren, das versucht, diesen Nachteil auszuräumen. ONISI stammt von Gorniak und Poole und ist ein „Online-Verfahren", d.h. es beschreibt nicht explizit den Aufbau einer Erfahrungs-Basis (wie z.B. bei IPAM), sondern berechnet diese Erfahrung „online".

Gorniak und Poole behaupten, dass die Betrachtung des letzten Sequenz-Schrittes nicht ausreiche, um gute Vorhersagen zu machen. Ihr Verfahren berechnet deshalb für jede mögliche Vorhersage ein Ranking, welches sich aus den k längsten Sequenzen der Historie berechnet, die mit der unmittelbaren Beobachtung übereinstimmen (k ist eine Konstante, Gorniak und Poole geben $k = 5$ als sinnvollen Wert an). Laut Gorniak und Poole gibt es jedoch auch Aktionen, die nur vom letzten Zustand (d.h. von der letzten Beobachtung) abhängen. Auch dies wird in dem Ranking berücksichtigt. Der Sequenzschritt mit dem höchsten Ranking kann dann als Vorhersage dienen.

ONISI ist also ein Verfahren, das einerseits lange Sequenzen erkennt, andererseits auch richtige Vorhersagen trifft, die nur von der letzten Beobachtung abhängen. Das Verfahren kann jedoch nicht Sequenzen beliebiger Länge lernen.

Jacobs und **Blockeel** [64] schreiben, dass Verfahren, die sich auf lange Sequenzen beschränken, nicht immer die beste Vorhersage bieten. Eine „ideale" Länge an Sequenzen, die gelernt werden, kann nicht vorher festgelegt werden. Deshalb baut ihr Algorithmus ein Markov-Modell mit gemischter Ordnung auf.

Das Verfahren von Jacobs und Blockeel baut auf IPAM (siehe Seite 28) auf und fügt einen weiteren Schritt ein, falls eine Vorhersage korrekt war. In diesem Fall werden neue bedingte Wahrscheinlichkeiten aufgenommen. Die Motivation dahinter ist, dass bei einer korrekten Vorhersage angenommen wird, dass die Wahrscheinlichkeitsverteilung, die für die aktuelle Historie angewendet wurde, sinnvoll ist. Deshalb werden für die Wahrscheinlichkeiten, die auf die aktuelle Historie passen, neue in die Liste eingefügt. Diese werden dann „nach vorne" verlängert und zwar um den korrekt vorhergesagten Sequenzschritt (war z.B. die Vorhersage a_{i+1} korrekt, werden $P(x|y \circ a_{i+1})$ hinzugefügt für alle möglichen x, wobei y der aktuellen Historie entspricht). Diese neuen Wahrscheinlichkeiten werden dann mit vorhandenen Einträgen in der Liste approximiert.

Das bedeutet, dass das Verfahren bei einer korrekten Vorhersage längere Sequenzen „spekulativ" lernt und die neuen Wahrscheinlichkeiten approximiert. Durch die Aktualisierung der Wahrscheinlichkeiten über das dahinter liegende IPAM-Verfahren werden diese dann sukzessive angepasst, d.h. bei „guten" Sequenzen erhöht sich die Wahrscheinlichkeit, bei „schlechten" vermindert sie sich. In Kapitel 6.1 auf Seite 99 wird dieses Verfahren detaillierter vorgestellt.

Ein weiteres Verfahren, welches ein Markov-Modell gemischter Ordnung aufbaut, ist **ActiveLeZi** [67–70]. ActiveLeZi ist ein Wörterbuch-basiertes Verfahren und baut auf LZ78[17]

[17] LZ78 ist ein Verfahren, welches zur Datenkompression eingesetzt wird.

[79], [80] auf. Das Wörterbuch, also erkannte Sequenzen, werden hier in einem Suffix-Baum gespeichert. Kern von ActiveLeZi ist das effiziente Aufbauen dieses Suffix-Baumes. Die Anzahl der beobachteten Wörter wird nach jeder Beobachtung aktualisiert. Mit Hilfe des Suffix-Baumes lassen sich bedingte Wahrscheinlichkeiten für die Vorhersage schätzen.

Künzer et al. [81] argumentieren, dass in den Erfahrungsdaten, die zum Training genutzt werden, Fehler enthalten sein können. Als Beispiel nennen die Autoren unnötige Schritte in einer Sequenz oder auch falsche Schritte, die der Benutzer ausgeführt hat. Diese Trainingsdaten sind dann für die zukünftige Vorhersage wenig nützlich, da sich die entsprechenden Teilsequenzen unterscheiden. Sie haben deshalb den **LEV-Algorithmus** entwickelt. Dieser baut ein Markov-Modell höherer Ordnung auf. Für die Vorhersage versucht das Verfahren zuerst, eine exakte Übereinstimmung der aktuellen Historie mit dem Markov-Modell zu finden. Wird keine exakte Übereinstimmung gefunden, wird versucht, eine ähnliche Sequenz im Markov-Modell zu finden. Als Ähnlichkeitsmaß wird die Levenshtein-Distanz [82] genutzt.[18] Zunächst wird versucht, eine Sequenz mit der Distanz eins zu finden. Falls auch dies scheitert, wird mit der Distanz zwei fortgefahren, usw.

Im Bereich Smart Home/Gebäudeautomatisierung gibt es das Verfahren **TMM** (Task-based Markov Model) [72], [83], [84], das sich ebenfalls mit der Sequenzvorhersage beschäftigt. Die Sequenzschritte, die in dieser Domäne beobachtet und gelernt werden sollen, sind „Events", die ein Benutzer in einem Gebäude ausführt. Ein solches Event besitzt mehrere Attribute: Schaltet ein Benutzer beispielsweise einen Schalter, so setzt sich das Event aus der ID des Schalters, der ID des Raumes, der Aktion (ein- oder ausschalten) sowie Datum- und Zeitstempel zusammen. Diese Events unterscheiden sich immer (da sich z.B. der Zeitstempel immer unterscheidet), können also mit normalen Sequenzvorhersage-Verfahren nicht gelernt werden. Weiterhin gibt es Events, die eigentlich einander entsprechen. Hier sind manche Attribute nicht relevant oder unterscheiden sich nur marginal. Als Beispiel seien hier zwei Events (an zwei aufeinander folgenden Tagen) zum Einschalten des Lichtes genannt: Der Datums-Stempel ist hierbei nicht relevant. Beim Zeitstempel kann eine geringe Differenz toleriert werden. Das bedeutet, zwei Events, deren Zeitstempel sich nur um x Minuten unterscheiden, können zusammengefasst werden.

Um dieser Problematik entgegenzuwirken, teilt TMM die Lernsequenz als Erstes in Teilsequenzen, Partitionen genannt. Hierfür werden Partitionierungsregeln definiert, die domänenspezifisch formuliert sind. Beispielsweise wird eine Event-Sequenz in zwei Partitionen unterteilt, wenn die Differenz dem Zeitstempel von zwei aufeinander folgenden Events >P Minuten entspricht[19]. Nachdem die Partitionierung erfolgt ist, wird ein Clustering der Partitionen vorgenommen. Das Ziel dahinter ist, ähnliche Partitionen zu finden, die dann einem „abstrakten Task" entsprechen. Nun wird ein verborgenes Markov-Modell (engl. Hidden-Markov-Model) aufgebaut, wobei die abstrakten Tasks den versteckten Zuständen (engl. hidden states) des Markov-Modells entsprechen.

Eine unserer Zielsetzungen aus Kapitel 1.2 ist, ein Verfahren zu konzipieren und zu spezifizieren, das kontextabhängige Abläufe erlernt. Wichtig in diesem Zusammenhang ist, dass in IT-Projekten die Gründe (Kontexte), die zu bestimmten Entscheidungen und letztendlich zu bestimmten Abläufen führen, nicht klar ersichtlich sind und auch nicht durch Regeln im Vorfeld beschrieben werden können. Dies ist bereits in Kapitel 1 ausführlich beschrieben. Deshalb eignen sich die in diesem Kapitel vorgestellten Sequenzvorhersageverfahren nicht, um das in Kapitel 1.2 definierte Ziel zu erreichen. Teilweise geht das zuletzt vorgestellte Verfahren TMM in eine ähnliche Richtung. Es eignet sich jedoch ebenfalls nicht für unsere Anwendung, da

[18] Die Levenshtein-Distanz ist ein Maß, das angibt, wie viele Operationen (Einfüge-, Lösch- und Ersetz-Operationen) minimal nötig sind, um eine Sequenz in eine andere zu überführen.
[19] Diese Regeln werden also nicht gelernt, sondern sind vorher durch Domänenwissen formuliert worden.

beispielsweise die Regeln für die Zerlegung einer Sequenz in Partitionen domänenspezifisch festgelegt sind (z.B. Zeitdifferenz für zwei aufeinander folgende Ereignisse > T). Diese Entscheidungen können in unserer Domäne nicht im Vorfeld getroffen werden.

3 PROZESSAUSSCHNITT ALS ANWENDUNGSBEISPIEL

Dieses Kapitel beschreibt einen Ausschnitt einer beispielhaften Entwicklungsmethodik für IT-Projekte. Die Methodik wird durchgängig in dieser Arbeit verwendet, um die dargestellten Konzepte und Verfahren anschaulich und durchgängig zu beschreiben. Abbildung 15 stellt diese Methodik im Überblick dar.

3.1 BESCHREIBUNG DER METHODIK

Im Projektablauf ist die hier dargestellte Methode nach der Fertigstellung des Pflichtenheftes anzuwenden. Das bedeutet, dass die Anforderungen an das zu entwickelnde System, die sich im Pflichtenheft wiederfinden, bereits existieren. Die Anforderungen können semi-formell, also z.B. textuell oder mittels Skizzen beschrieben sein. Relevant ist hierbei nur, dass die Anforderungen identifizierbar sind, also eine eindeutige ID besitzen.

Die nächsten Schritte im Projekt sind dann die Entwicklung einer Systemarchitektur, die Spezifikation der Systemelemente sowie deren Implementierung. Der weitere Verlauf im Projekt stellt sich folgendermaßen dar:

Als Erstes werden Komponenten – die Elemente der Systemarchitektur – identifiziert. In Abbildung 15 sind beispielsweise „K1“ und „K2“ als identifizierte Komponenten dargestellt.

Daraufhin werden die Anforderungen des Pflichtenhefts auf die Elemente der Architektur abgebildet (siehe Abbildung 15). Dies kann eine n:m-Abbildung sein. Eine Anforderung kann also auf mehrere Komponenten abgebildet werden, und eine Komponente kann mehrere Anforderungen umsetzen. In der Abbildung ist die Anforderung „Anf. 1“ auf „K1“ abgebildet und die Anforderungen „Anf. 2“ und „Anf. 3“ sind auf „K2“ abgebildet.

Sobald alle relevanten Anforderungen auf eine Komponente abgebildet sind, kann als nächster Schritt diese dann spezifiziert werden. Für jede Komponente wird also eine Spezifikation erstellt. Hier werden dann die Schnittstellen der Komponente detailliert beschrieben. In unserem Fall bedeutet dies, dass für jede textuelle Anforderung mindestens eine Schnittstellenfunktion beschrieben wird. In unserem Beispiel wird die Schnittstelle „IF1“, die die Anforderung „Anf. 1“ umsetzt, durch die Schnittstellenfunktion „foo(…)“ beschrieben.

Nachdem eine Spezifikation erstellt ist, kann die Implementierung dieser Komponente konzipiert und umgesetzt werden. Hierfür werden die Klassen der Komponente und deren Schnittstellen identifiziert und diese Klassen dann implementiert. Im Beispiel sind die Klassen „X“, „Y“ und „Z“ identifiziert und implementiert.

Entsprechend der obigen Beschreibung besteht die Methodik letztendlich aus vier zu unterscheidenden Arbeitsschritten:

1. Komponente identifizieren
2. Anforderung auf Komponente mappen
3. Komponente spezifizieren
4. Komponente implementieren.

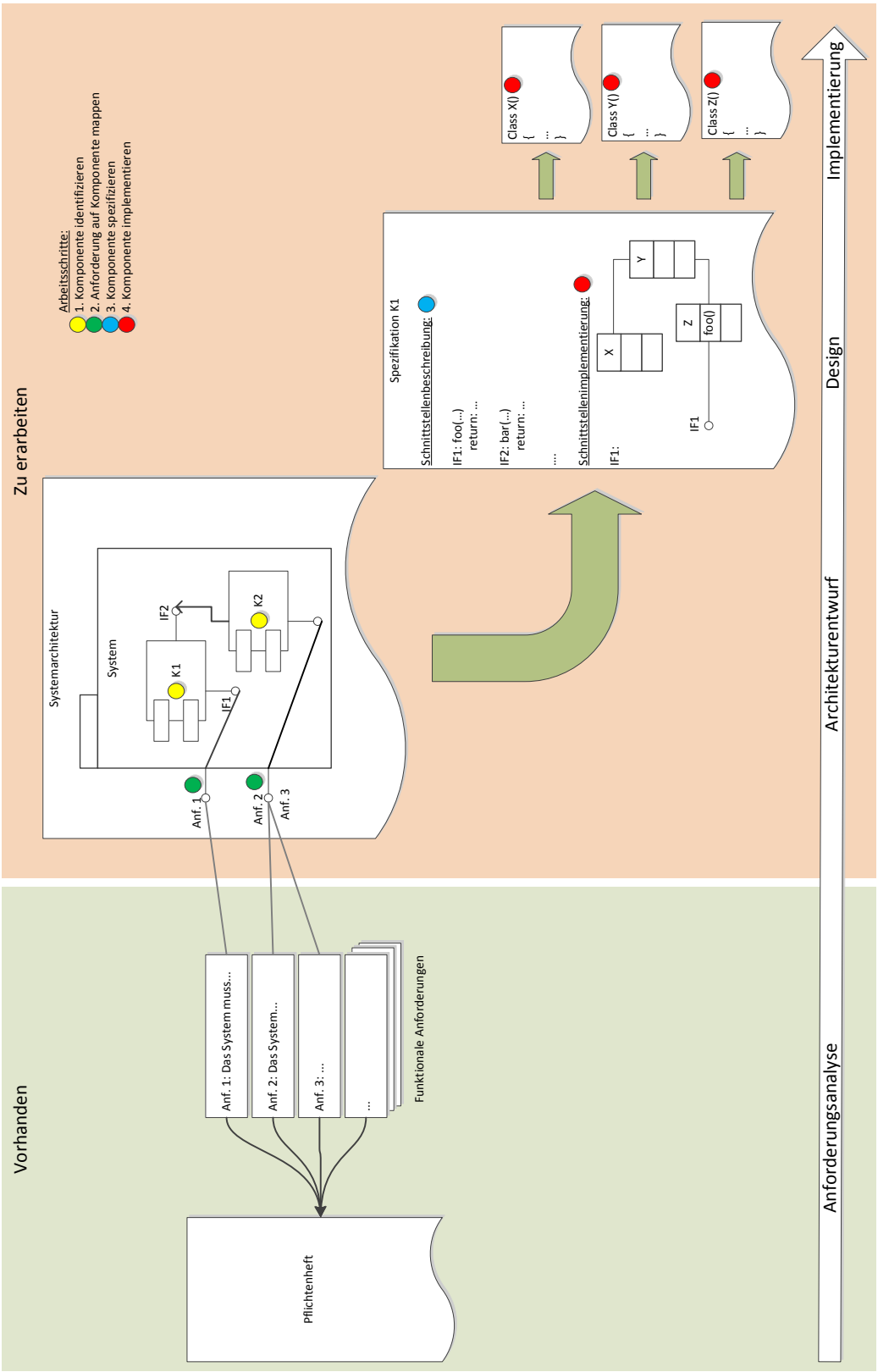


Abbildung 15: Durchgängig in dieser Arbeit verwendete Methodik

3.2 MÖGLICHE PROZESSABLÄUFE

In Kapitel 3.1 wurde die Methodik mit einer Systematik beschrieben, die für die Erklärung und Vorstellung sinnvoll ist. Begonnen wurde mit der Beschreibung der schon existierenden Anforderungen an das System, daraufhin wurde der Architekturentwurf beschrieben. Nach dem Entwurf wurde das Design, also die Detaillierung der Anforderungen in Form von Schnittstellenbeschreibungen, deren Implementierungsentwurf und die Implementierung dargestelt.

Dies entspricht jedoch nicht unbedingt dem tatsächlichen Ablauf bei der Anwendung der Methodik: Der Architekturentwurf zum Beispiel erfolgt nicht zeitlich streng vor dem Design der Schnittstellen. Das heißt, oftmals wird erst ein Teil der Systemarchitektur erstellt, und dieser Teil wird dann umgesetzt. Danach werden weitere Teile der Architektur erstellt usw. Auch denkbar ist, dass zuerst eine (prototypische) Implementierung erfolgt nachdem die Anforderungen fixiert sind, und diese dann später in einer Systemarchitektur nachdokumentiert werden.

Der genaue Ablauf der Methodik ist hier nicht festgelegt. Dies ist auch gar nicht erwünscht, denn es bleibt dem Projektmitarbeiter überlassen, in welcher Reihenfolge er die einzelnen Schritte durchführt. Relevant ist jedoch, dass am Ende alle nötigen Arbeitsschritte ausgeführt worden sind. Dies entspricht unserer Aussage und Motivation aus Kapitel 1, dass es sinnvoll ist, den Projektmitarbeiter nicht einzuschränken, sondern zu unterstützen, und dass letztendlich damit bessere Produkte fertiggestellt werden.

Bei der beispielhaften Anwendung der hier vorgestellten Methodik im weiteren Verlauf dieser Arbeit gibt es einen Projektmitarbeiter, der zwei Arten von Komponenten im System entwirft und pflegt. Zum einen existieren komplexe Komponenten. Dies könnten zum Beispiel Hardware-nahe Komponenten sein, die u.a. auch Treiber implementieren. Auf der anderen Seite sind Komponenten vorhanden, deren Klassen eine hohe Kopplung haben. Dies bedeutet, die Klassen innerhalb dieser Komponente haben viele Schnittstellen untereinander. Von diesen zwei Arten von Komponenten hängt das weitere Vorgehen, also der Ablauf/Prozess im Projekt ab.

So wird bei komplexen Komponenten eher eine prototypische Entwicklungsmethodik angewendet. Dies bedeutet, nachdem eine Komponente identifiziert ist, wird zunächst eine Anforderung auf diese Komponente abgebildet. Danach wird für diese Anforderung die Spezifikation erstellt, um dann die Klassen für die zuvor abgebildete Anforderung zu implementieren. Ein beispielhafter Prozessablauf ist in Abbildung 16 dargestellt. Das dazu passende Produktmodell ist in Abbildung 17 abgebildet. In der Abbildung 16 erkennt man, dass die Arbeitsschritte 1 bis 4 bzw. 2 bis 4 immer sequentiell durchlaufen werden: In den ersten beiden Teilsequenzen ist die Identifikation jeweils einer Komponente enthalten (Arbeitsschritt „Komponente identifizieren“). Die restlichen Teilsequenzen entwickeln diese beiden Komponenten weiter, indem weitere Anforderungen abgebildet werden. Danach folgt die Spezifikation etc. (Arbeitsschritte 2 bis 4). Abbildung 17 zeigt das nach dem Durchlauf der in Abbildung 16 beschriebenen Arbeitsschritte resultierende Datenmodell.

Auf der anderen Seite werden bei den Komponenten mit hoher Kopplung bei Anwendung der Methodik ein „breiter“ Entwurf und eine „breite“ Implementierung gewählt. Nachdem hier eine Komponente identifiziert ist, werden alle passenden Anforderungen auf diese Komponente abgebildet, um danach die Komponente im Ganzen zu spezifizieren und dann zu implementieren. Der Prozessablauf zu dieser Methodik ist in Abbildung 19 dargestellt, das dazugehörige Produktmodell ist in Abbildung 18 zu sehen. In Abbildung 19 ist zu erkennen, dass sich die Abfolge an Arbeitsschritten von der Abfolge bei der prototypischen Methodik unterscheidet: Während bei der prototypischen Methodik die Arbeitsschritte sequentiell durchlaufen werden (z.B. 1→2→3→4), wird hier derselbe Arbeitsschritt mehrfach hintereinander ausgeführt (z.B. 1→2→2→2→3→...). Abbildung 18 zeigt das resultierende

Datenmodell nach dem Durchlauf der Arbeitsschritte. Strukturell unterscheidet es sich nicht von dem Datenmodell bei der prototypischen Entwicklung. Der „Weg dahin“, also der durchgeführte Prozess, ist jedoch ein völlig anderer.

Die oben beschriebenen Abläufe sind als exemplarische Prozesse zu verstehen. Wir wollen hier, bzw. in einer Prozessbeschreibung, diese Abläufe nicht explizit festlegen. Es gibt jedoch einige Bedingungen, die bei der Anwendung der Methodik gelten müssen.

Anhand der hier vorgestellten möglichen Durchführungsvarianten der Methodik werden unsere Ideen, Konzepte und Verfahren in den nachfolgenden Kapiteln durchgängig erklärt.

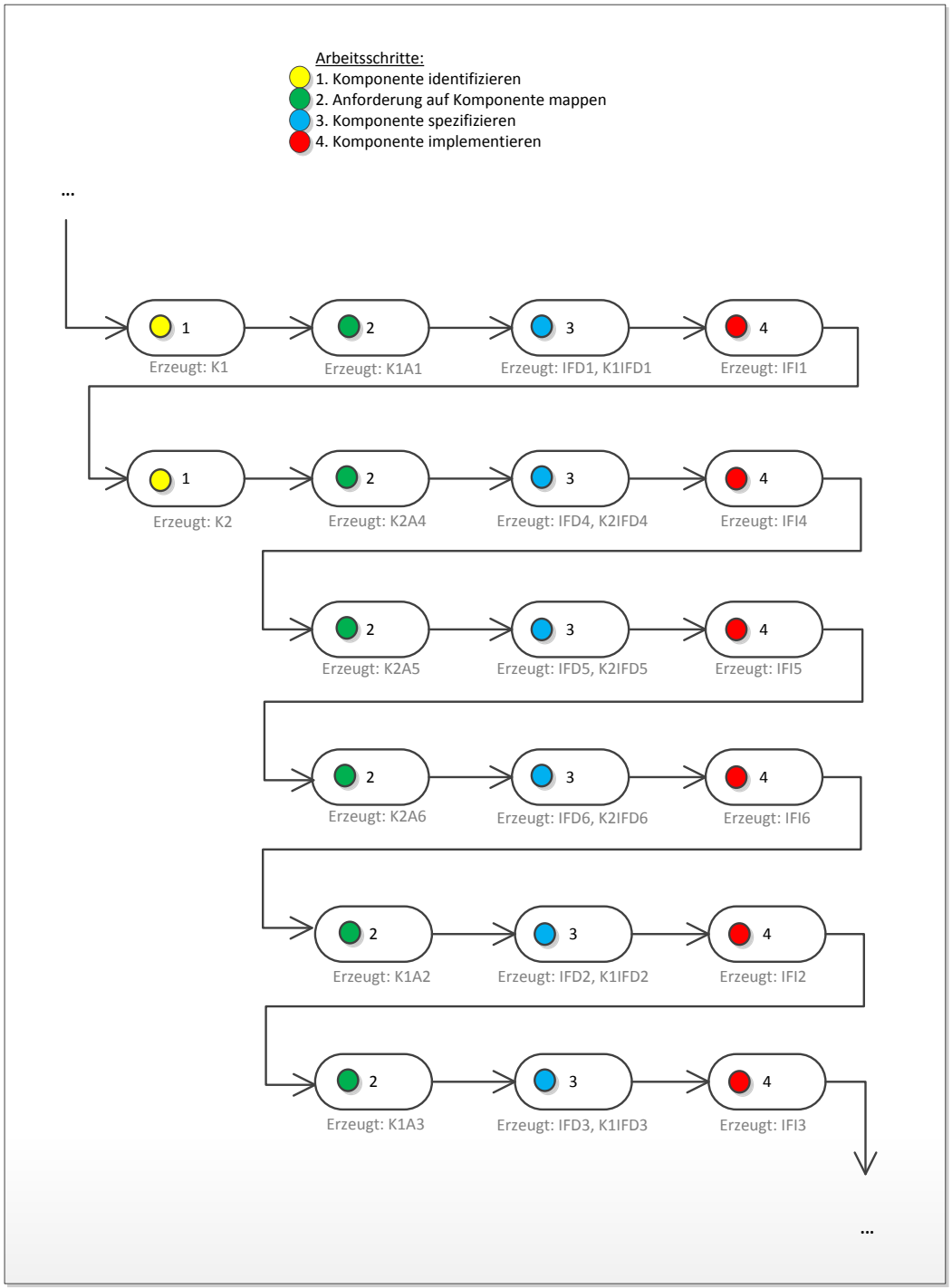


Abbildung 16: Beispielhafte Anwendung der Methodik: Prototypische Entwicklung

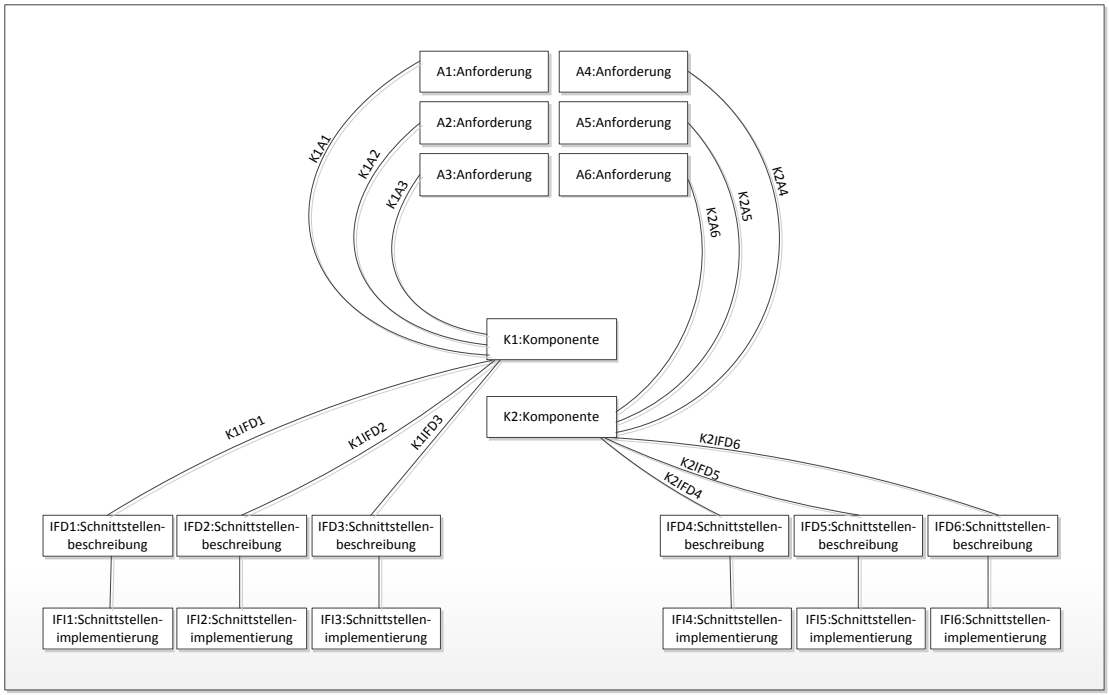


Abbildung 17: Produktmodell bei prototypischer Entwicklung

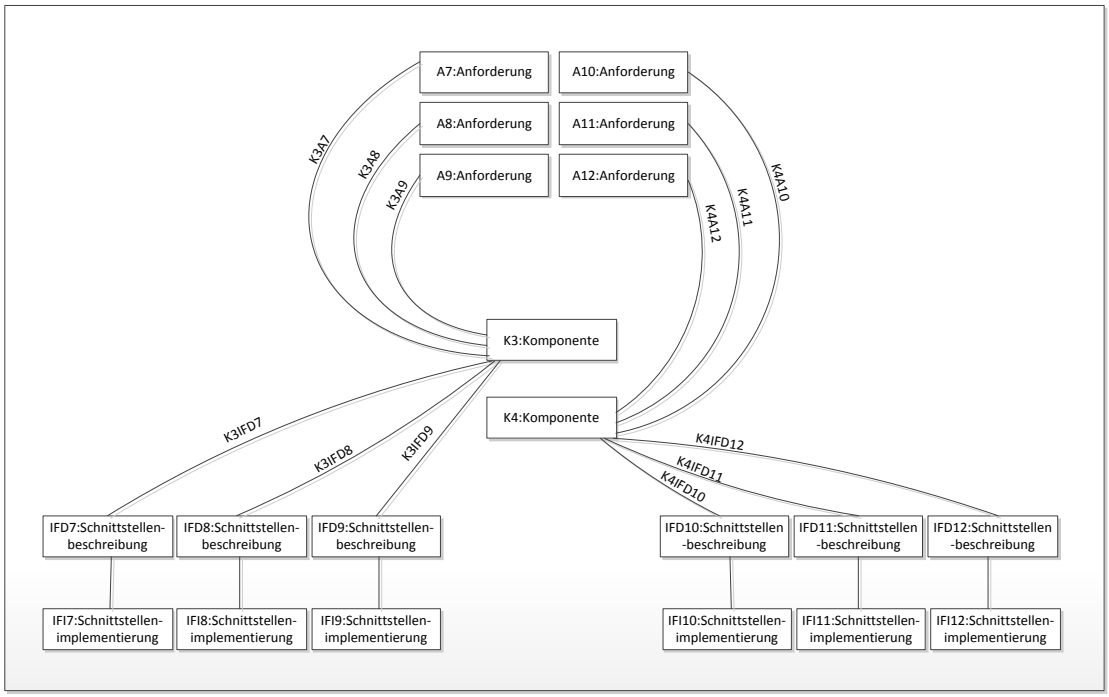


Abbildung 18: Produktmodell beim breiten Entwurf

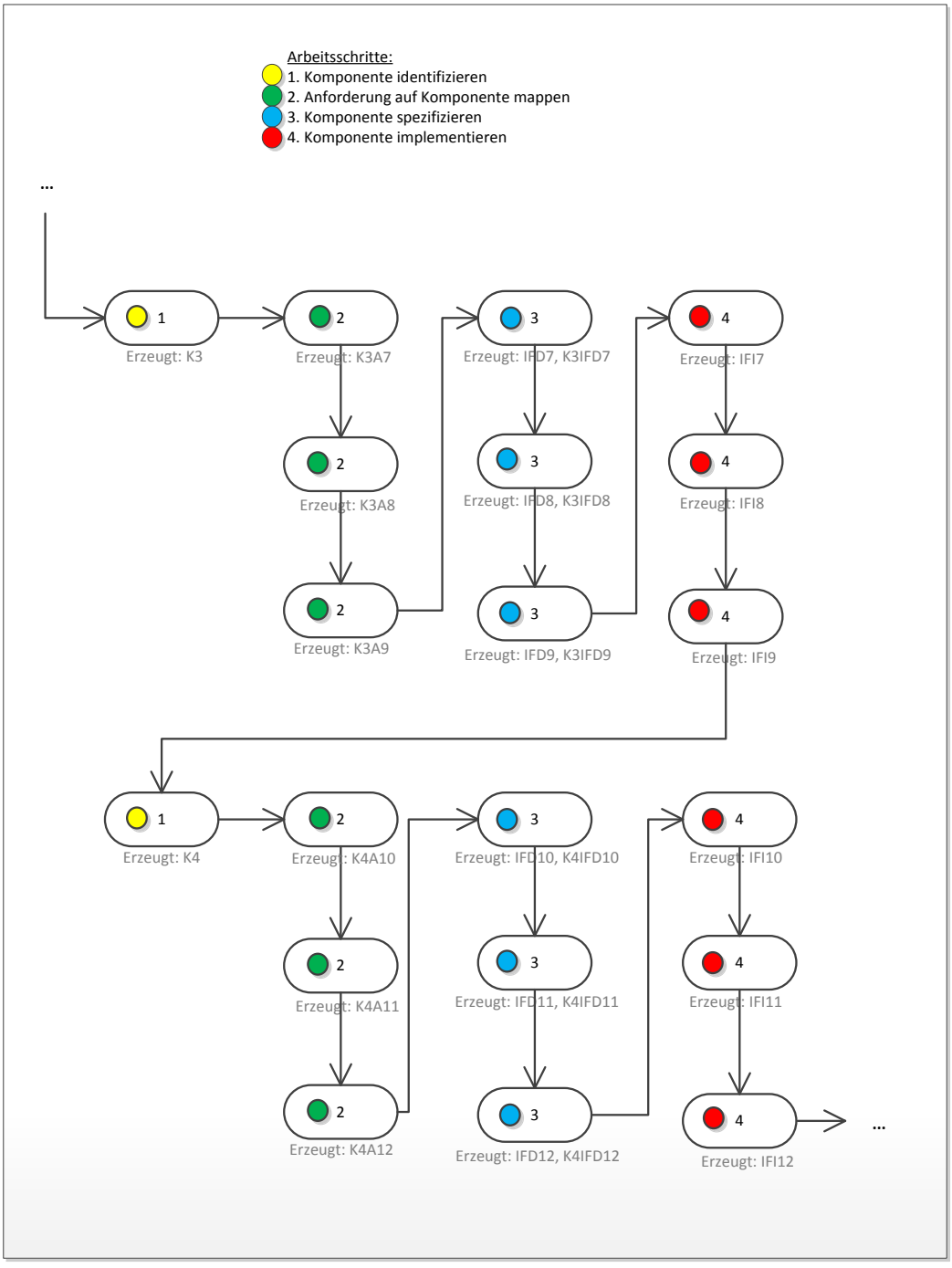


Abbildung 19: Beispielhafte Anwendung der Methodik: Breiter Entwurf

4 GESAMTANSATZ

Dieses Kapitel stellt die konzeptionellen Ideen für eine Prozessausführung in IT-Projekten vor, insbesondere unter Berücksichtigung der projektmitarbeiter- und kontextspezifischen Einflussfaktoren – siehe hierzu Kapitel 1, in dem dies grundlegend motiviert wird. Es ist die Grundlage für die Kapitel 5 und 6. In Kapitel 5 wird unsere Prozessbeschreibungssprache detailliert spezifiziert, das sechste Kapitel beschreibt das Verfahren für die Projektmitarbeiterunterstützung im Detail.

In Kapitel 4.1 werden die eigentlichen Probleme beschrieben und weshalb sich diese nicht mit bisher vorhandenen Ansätzen lösen lassen. Kapitel 4.2 beschreibt die grundlegende Idee zur Lösung dieser Probleme. Daraus lassen sich Anforderungen an eine Prozessbeschreibungssprache ableiten. Diese werden in Kapitel 4.3 dargestellt. Das abschließende Kapitel 4.4 beschreibt die eigentliche Konzeption und eine mögliche Werkzeugunterstützung.

4.1 DAS PROBLEM

Im Kapitel 3 wurde ein Prozessausschnitt einer Entwurfsmethodik als Anwendungsbeispiel gezeigt. Dieses wird durchgängig in der gesamten Arbeit aufgegriffen. Kapitel 3.2 beschrieb mögliche projektmitarbeiter- und kontextspezifische Abläufe/Prozesse dieser Methodik.

Dieses Kapitel beschreibt Probleme, die bei bisherigen Prozessbeschreibungen bei der Implementierung der oben genannten Entwurfsmethodik, insbesondere der projektmitarbeiter- und kontextspezifischen Abläufe, auftauchen. Als Beispiel wird hier eine generische, netzbasierte Prozessbeschreibungssprache, ähnlich wie UML-Aktivitäten, verwendet.

Mit regelbasierten Prozessbeschreibungssprachen lassen sich relativ leicht Freiheiten modellieren, die nötig sind, um projektmitarbeiter- und kontextspezifische Einflüsse zu berücksichtigen. Eine prozessbeschreibungskonforme Ausführung ist somit möglich, jedoch keine (werkzeugbasierte) Unterstützung des Projektmitarbeiters. Der Projektmitarbeiter hat immer die Menge von Arbeitsschritten zur Auswahl, deren Vorbedingungen erfüllt sind. Damit ist ein Nutzen einer vorliegenden Prozessbeschreibung, nämlich die Unterstützung während der Projektarbeit, nicht möglich. Hier wäre es nämlich wünschenswert, wenn der Projektmitarbeiter nicht alle möglichen Arbeitsschritte (unsortiert) zur Auswahl hat, sondern ihm kontextspezifisch die wahrscheinlichsten Schritte vorgeschlagen werden.

Für andere Verfahren, wie zum Beispiel Deviation Management Systeme, beschrieben in Kapitel 2.1.6 auf Seite 22 gilt das gleiche: Hier ist es ebenfalls möglich, Freiheiten während der Prozessausführung zu modellieren, jedoch beschränkt sich hier die Nutzerunterstützung nur auf den spezifizierten Standardablauf.

In Abbildung 20 ist unsere durchgängig verwendete Methodik aus Kapitel 3 modelliert. In Kapitel 3 wurden zwei mögliche Abläufe für zwei verschiedene Arten von Komponenten identifiziert. Diese sind in Abbildung 16 auf Seite 36 und Abbildung 19 auf Seite 38 dargestellt. Die hier vorgestellte Modellierung unterstützt diese zwei möglichen Abläufe (prototypische Entwicklung/breiter Entwurf). Zu sehen sind hier die vier zuvor beschriebenen Arbeitsschritte und deren mögliche Kantenübergänge. In Abbildung 20 ist die Prozessbeschreibung eines Projektmitarbeiters skizziert, der bereits vier Komponenten entworfen und entwickelt hat. Die Komponenten K1 und K2 werden prototypisch entwickelt, bei K3 und K4 wird eine „breite Entwurfsmethodik“ angewendet. Die Bedingungen an den Verzweigungen beschreiben, an welcher Komponente der Projektmitarbeiter gerade arbeitet (also: in welchem Kontext er zurzeit arbeitet). Abbildung 20 ist nur als Skizze der Prozessbeschreibung zu sehen. Der breite Entwurf

beispielsweise wiederholt die Arbeitsschritte 2, 3 und 4 mehrfach hintereinander. Nach einer individuellen Anzahl an Wiederholungen wird erst der nächste Schritt ausgeführt. In Abbildung 19 werden diese Arbeitsschritte zum Beispiel dreimal hintereinander ausgeführt. Die Übergänge von Arbeitsschritt 2 nach 3 und 3 nach 4 sind ohne Bedingungen versehen, da diese Abfolgen mit allen Komponenten durchlaufen werden kann.

Mit dieser Prozessbeschreibung und einem durchgängigen Werkzeug könnte eine in unserem Sinne vorteilhafte Unterstützung des Projektmitarbeiters erfolgen. Die hier skizzierte Vorgehensweise birgt jedoch mehrere Probleme:

- Der Projektmitarbeiter oder derjenige, der die Prozessbeschreibung anpasst muss sich konkrete Gedanken machen, wie er in welchem Fall arbeitet. Das heißt, er muss sich überlegen, welches Vorgehen er in welchem Kontext (z.B. bei welcher Komponente) wählt. Die möglichen Vorgehensweisen zu identifizieren ist schon schwierig. Noch komplexer wird es allerdings, diese Vorgehensweisen konkreten Kontexten (z.B. Komponenten) zuzuordnen.

- Alle möglichen Vorgehensweisen muss dieser Projektmitarbeiter dann in die Prozessbeschreibung integrieren, und zwar *bevor* er eine solche Vorgehensweise anwendet.

- Eine Unterstützung des Projektmitarbeiters bei der Identifikation der möglichen Vorgehensweisen und der dazugehörigen relevanten Kontextinformationen ist so nicht möglich.

Des Weiteren ergeben sich folgende untergeordnete Probleme:

- Im Beispiel hier müssten nach jeder neuen Erstellung einer Komponente die Kantenübergänge angepasst werden. Dies führt zu einer permanenten Aktualisierung der Prozessbeschreibung. Hierdurch kann zu Problemen während der Ausführung eines Prozesses kommen (z.B. Inkonsistenzen).

- Mit der Zeit entwickelt sich die Prozessbeschreibung zu einer hoch-komplexen Ablaufbeschreibung, deren Wartung nur noch schwer möglich ist.

- Die Informationen, mit denen hier im Beispiel die Prozessbeschreibung modelliert wurde (z.B. Komponenteninstanzen; denkbar sind auch Spezifikationsinstanzen oder Klasseninstanzen, die einen spezifischen Kontext ausmachen, der dann einen bestimmten Ablauf zur Folge hat), gehören nicht in eine Prozessbeschreibung. Hierin sollte ein generalisiertes, allgemeingültiges Vorgehen beschrieben werden.

- Temporäre Änderungen der Vorgehensweise (z.B. Kollege ist in Urlaub) erhöhen zusätzlich den Wartungsaufwand der Prozessbeschreibung.

- Bei dem Anpassungsintervall einer Prozessbeschreibung sprechen wir unseres Erachtens nach von einem mittelfristigen Zeithorizont (Tage bis Wochen). Die hier skizzierte Vorgehensweise erfordert jedoch eine viel kurzfristigere Anpassung (ad-hoc-Anpassung, Online-Anpassung).

Das nachfolgende Kapitel 4.2 beschreibt die grundlegende Idee, um projektmitarbeiter- und kontextspezifische Abläufe zu gewährleisten und zugleich die o.g. Probleme zu lösen.

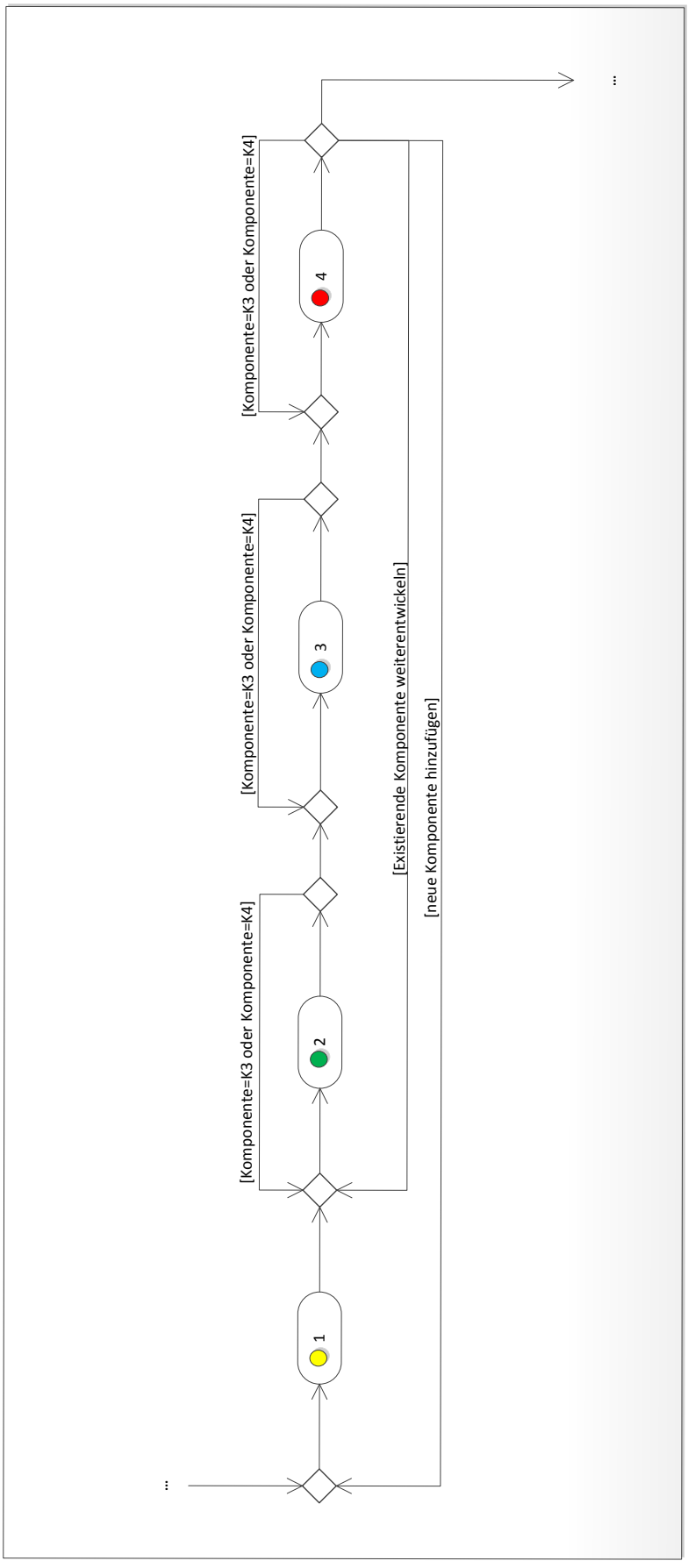


Abbildung 20: Skizze einer netzbasierten Prozessbeschreibung passend zu den Abläufen in Kapitel 3.2

4.2 DIE IDEE

Um die o.g. Probleme zu beheben wird folgende Idee verfolgt: Ein Projektmitarbeiter kann Arbeitsschritte im Projekt flexibel ausführen. Das heißt, er kann die konkrete Vorgehensweise (Abfolge von Arbeitsschritten) für die Bearbeitung von Aufgaben selbst bestimmen. Diese Arbeitsschritte werden üblicherweise mit einem Software-Werkzeug durchgeführt. Dieses Werkzeug beobachtet die Vorgehensweise des Projektmitarbeiters und leitet aus den Beobachtungen Vorschläge für die nächsten durchzuführenden Arbeitsschritte ab. Hierunter kann man sich eine sortierte Auswahlliste mit Schritten vorstellen, die den bisherigen Beobachtungen entsprechen. Diese Vorschläge werden mit einem maschinellen Lernverfahren bereit gestellt.

Der Unterschied zu bisher existierenden Verfahren ist, dass der Projektmitarbeiter sich hier nicht darüber im Klaren sein muss, welche Vorgehensweise er in welchem Kontext ausführt und er muss diese Vorgehensweisen auch nicht in das Prozessmodell integrieren. Bei einer Änderung der Vorgehensweise (z.B. temporäre Änderung) oder einer neuen Vorgehensweise muss diese nicht vorher in die Prozessbeschreibung eingefügt werden, sondern wird automatisiert erkannt. Durch diesen Automatismus wird der Projektmitarbeiter unterstützt, da er a) sich gar nicht erst um die Hinterlegung der Vorgehensweisen in das Prozessmodell kümmern muss (d.h. er kann sich auf „seine“ Arbeit konzentrieren) und b) ein Werkzeug kann gezielt und kontextspezifisch nächste Arbeitsschritte vorschlagen, wobei der Projektmitarbeiter diese Vorschläge befolgen kann oder nicht. Ein weiterer (technischer) Vorteil ist, dass sich hiermit die projektmitarbeiter- und kontextspezifischen Vorgehensweisen vom eigentlichen Prozessmodell trennen lassen und somit diese teils sehr spezifischen Informationen nicht in dem Prozessmodell enthalten sind, das eher generalisierte Informationen enthält (z.B. projektweit gültiges Vorgehen).

Mit einer Prozessbeschreibungssprache, die diese Ideen umsetzt und die in dieser Arbeit vorgestellt wird werden keine (wie beispielsweise in Abbildung 20 dargestellt) Abläufe beschrieben. Hier wird also kein netzbasierter oder imperativer Ansatz verfolgt. Stattdessen stehen die einzelnen Arbeitsschritte in einer Prozessbeschreibung „einzeln“ nebeneinander. Vorgehende oder nachfolgende Arbeitsschritte werden z.B. nicht durch Kanten beschrieben (siehe hierzu Abbildung 21). Die Arbeitsschritte beinhalten jedoch, neben der Aktion, die durchgeführt wird, Vor- und Nachbedingungen. Diese legen fest, ob ein Arbeitsschritt prinzipiell gestartet oder beendet werden kann. Die Vor- und Nachbedingungen sind i.A. so formuliert, dass sie einen Zustand des Produktmodells beschreiben (informell z.B. „Es gibt mindestens eine Komponente, deren Spezifikation nicht fertiggestellt ist.“).

Alle Arbeitsschritte, deren Vorbedingungen erfüllt sind, sind prinzipiell startbar und lauffähig. Entsprechend können alle Arbeitsschritte, die gestartet sind und deren Nachbedingungen erfüllt sind, beendet werden. Zu jedem Zeitpunkt kann also eine Menge von Arbeitsschritten bestimmt werden, die gestartet und die beendet werden können. Eine Menge startbarer Arbeitsschritte ist beispielhaft in Abbildung 22 dargestellt.

Während der Projektarbeit kann ein Projektmitarbeiter frei aus diesen beiden Mengen wählen, welche Arbeitsschritte er starten und welche er beenden will (Abbildung 23). Im oberen Bereich der Abbildung ist eine Abfolge dargestellt, in der der Projektmitarbeiter Arbeitsschritte startet und beendet. Im unteren Bereich sind die aktuell gestarteten Arbeitsschritte aufgeführt.

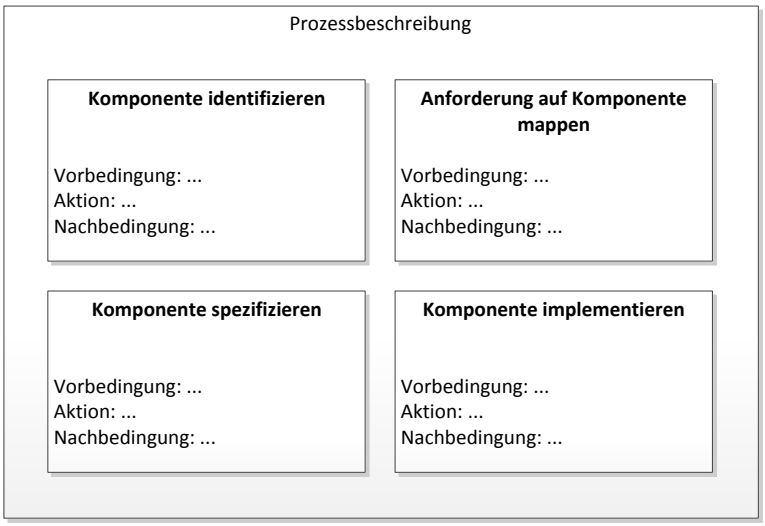


Abbildung 21: Beispielhafte Prozessbeschreibung

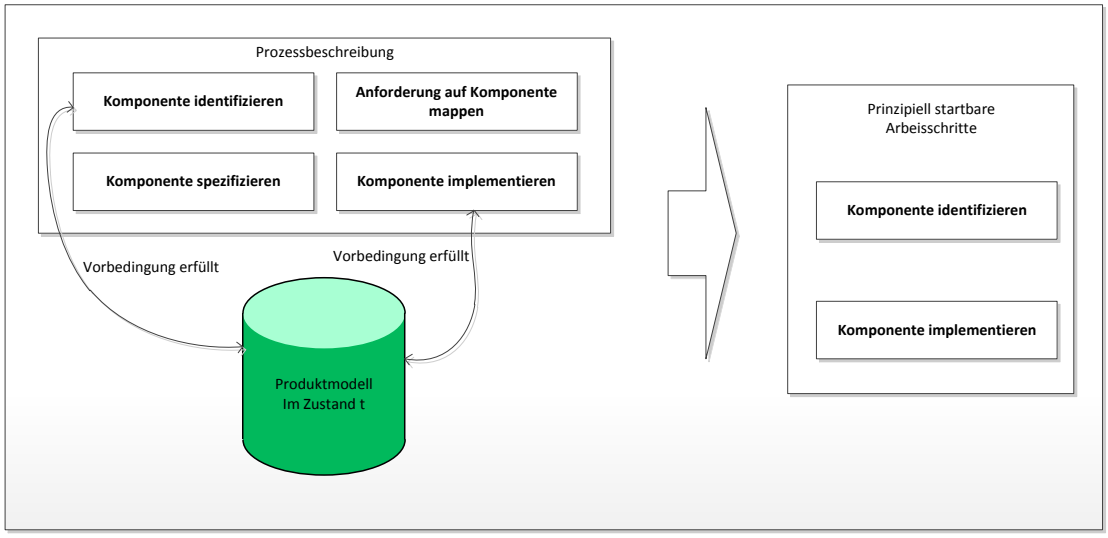


Abbildung 22: Erfüllte Vorbedingungen bestimmen prinzipiell startbare Arbeitsschritte

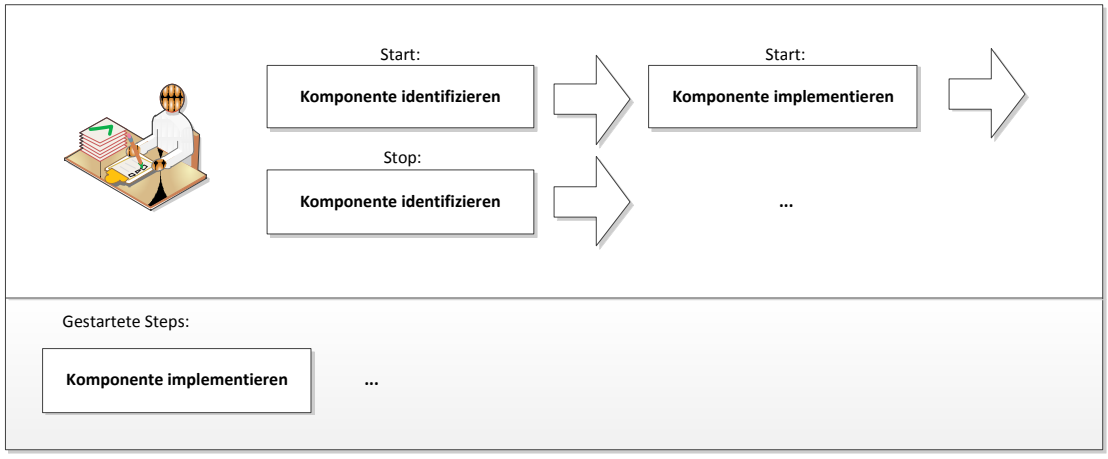


Abbildung 23: Starten und Beenden von Arbeitsschritten durch den Projektmitarbeiter

Weiterhin wichtig ist, dass die Projektarbeit werkzeugbasiert durchgeführt wird. Das heißt, es gibt ein durchgängiges Werkzeug (Beispiel: siehe Abbildung 24), das

a) die vorliegende Prozessbeschreibung interpretiert und ausführt, also eine Ablaufkontrolle bereitstellt (z.B. alle Arbeitsschritte, deren Vorbedingung erfüllt sind, werden angeboten) und

b) für die eigentliche Projektarbeit, also das Ausführen der in den Arbeitsschritten beschriebenen Aktionen, verwendet wird. Hierunter kann man sich beispielsweise ein (UML) Case-Tool vorstellen, mit dem man Operationen auf dem Produktmodell ausführen kann (z.B. Hinzufügen einer Referenz zwischen einer Anforderung und einer Komponente).

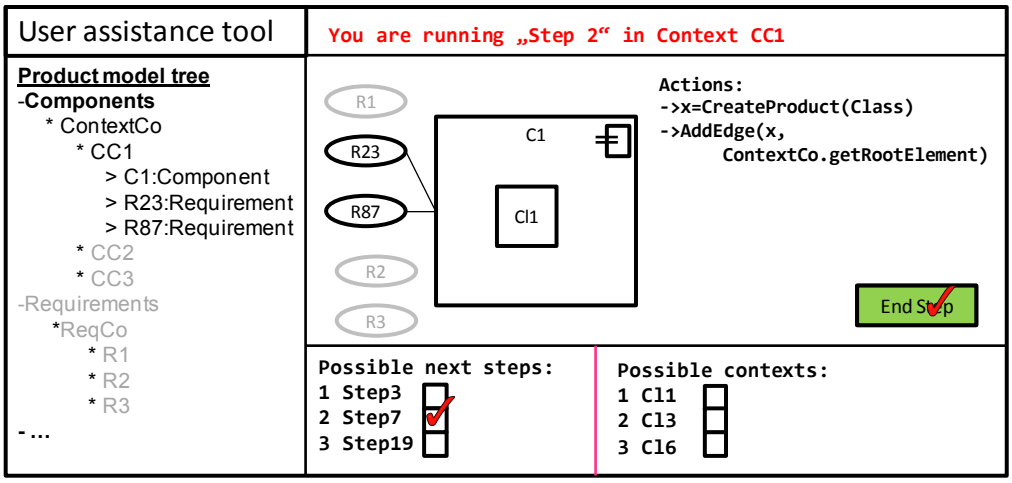


Abbildung 24: Screenshot-Prototyp für durchgängige Werkzeugunterstützung

Die Idee ist nun, dass die Arbeit des Projektmitarbeiters, also die Reihenfolge der Arbeitsschritte, die der Projektmitarbeiter ausführt, beobachtet wird. Zusätzlich werden auch die Kontextinformationen zu jedem Arbeitsschritt aufgezeichnet.

Das Ziel ist es dann, diese projektmitarbeiter- und kontextspezifischen Beobachtungen (Abläufe von Arbeitsschritten mit dazugehörenden Kontexten) zu lernen und in einer Erfahrungsdatenbank zu hinterlegen (siehe Abbildung 25). Wenn zum Beispiel ein Projektmitarbeiter bei einer bestimmten Komponente k1 (Kontext: k1) ein spezielles Vorgehen anwendet, so soll dieser spezifische Prozess mit den relevanten Kontextinformationen hinterlegt werden (Abbildung 27, „Prozessablaufschnipsel“).

Mit diesem gelernten Wissen und der aktuellen Arbeit des Projektmitarbeiters (letzter, „aktueller“ Prozessausschnitt mit allen Kontextinformationen) können dann die gelernten und am besten passenden Vorgehen („Prozessablaufschnipsel“) dem Projektmitarbeiter vorgeschlagen werden.

Der Projektmitarbeiter kann dann einen beliebigen Arbeitsschritt aus dieser Liste wählen und ausführen (siehe Abbildung 26).

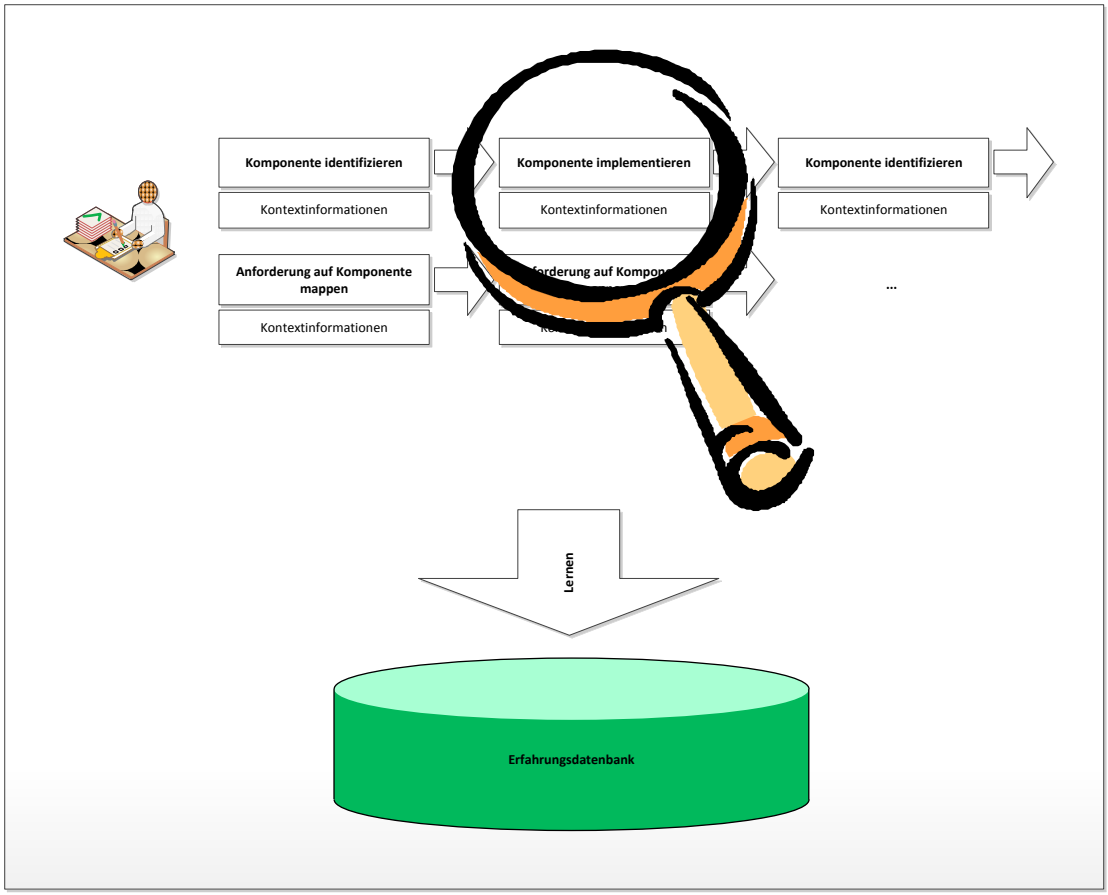


Abbildung 25: Beobachten und Erlernen der Arbeitsweise des Projektmitarbeiters

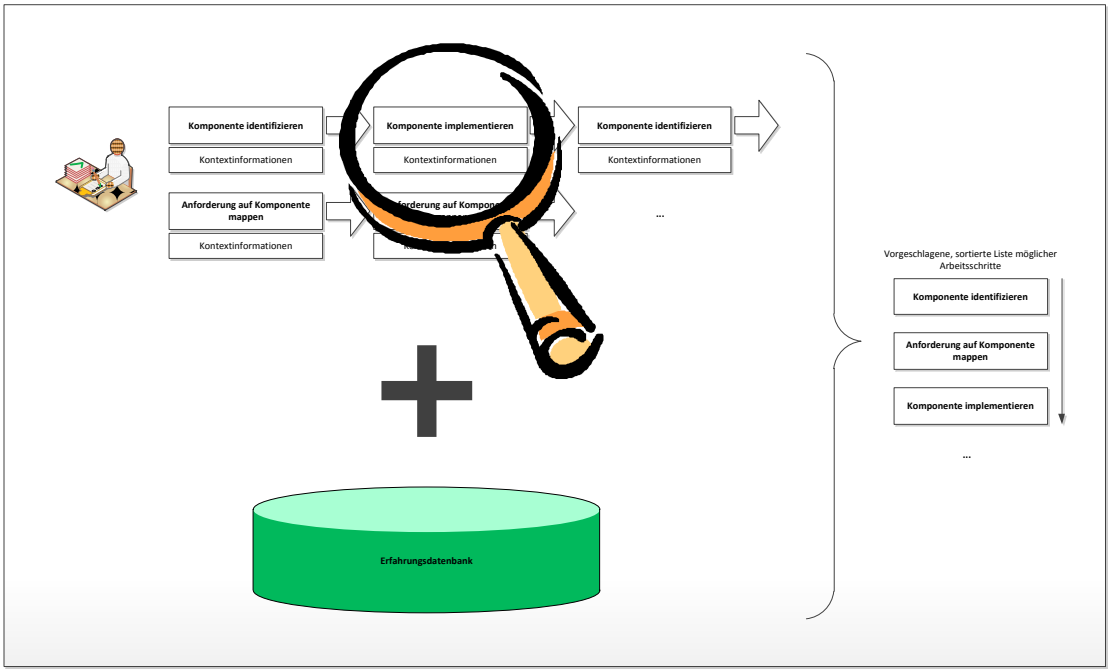


Abbildung 26: Vorschlag für nächste Arbeitsschritte

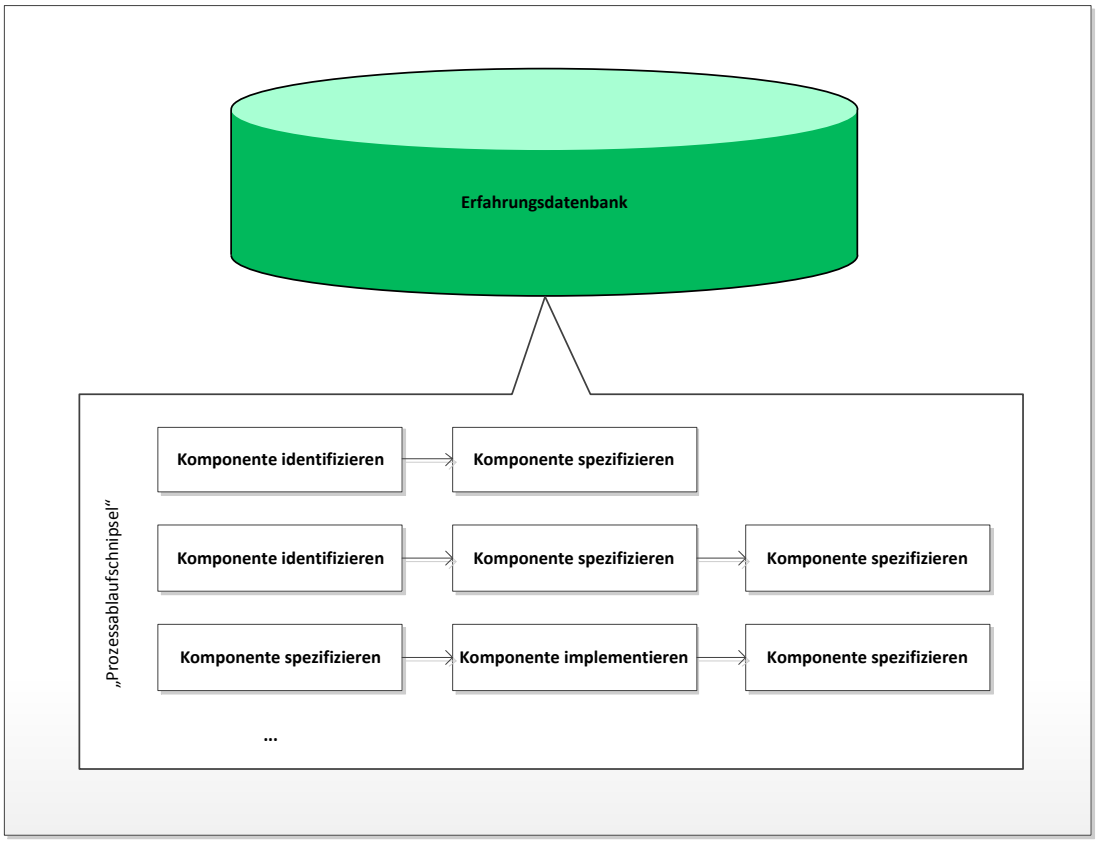


Abbildung 27: Erlernte Arbeitsweise des Projektmitarbeiters

4.3 ANFORDERUNGEN AN DAS PROZESSMODELL

Der Fokus dieser Arbeit liegt in der Prozessausführung (Process enactment) bei IT-Projekten. Insbesondere sollen die projektmitarbeiter- und kontextspezifischen Einflüsse berücksichtigt werden, sodass Grundlagen geschaffen werden, die es ermöglichen, eine Werkzeugbasierte Unterstützung des Projektmitarbeiters während der Projektarbeit zu gewährleisten. Die Anforderungen an ein Prozessmodell sind dann folgende:

- Nutzerspezifische, flexible Ausführung von Arbeitsschritten
- Berücksichtigung von Kontexten während der Prozessausführung
- Klar definierte Ausführungssemantik

Das Prozessmodell muss es ermöglichen, dass Projektmitarbeiter Arbeitsschritte flexibel ausführen können. Somit ist es möglich, dass Mitarbeiter ihre spezifische Arbeitsweise im Projekt durchführen. Eine grundlegende Methodik – zum Beispiel im Rahmen von organisations- oder projektspezifischen Vorgaben – muss jedoch integriert werden können.

Weiterhin muss es die Möglichkeit geben, Kontexte zu definieren und während der Prozessausführung zu nutzen. Somit können die o.g. kontextspezifischen Einflüsse bei der Unterstützung des Projektmitarbeiters berücksichtigt werden.

Eine klar definierte Ausführungssemantik ist nötig, um das erarbeitete Konzept und Verfahren in ein durchgängiges Werkzeug zu implementieren.

Einzelne Arbeitsschritte werden während der Prozessausführung innerhalb kurzer Zeit durchgeführt. Eine manuelle (z.B. kontextspezifische) Adaption der Prozessbeschreibung an die Bedürfnisse des Projektmitarbeiters erfordert immer Zeit. Hierdurch kommt es immer wieder zu Differenzen zwischen spezifiziertem und ausgeführtem Prozess. Deshalb ist eine weitere Anforderung an das Prozessmodell, einen

- Automatismus bei der projektmitarbeiter- und kontextspezifischen Adaption des Prozessmodells

bereitzustellen. Dieser Automatismus wird „Online" die Arbeit des Projektmitarbeiters beobachten und erlernen.

Im folgenden Kapitel 4.4 wird die hier erarbeitete Prozessbeschreibungssprache konzipiert. Das Kapitel 5 ab Seite 55 beschäftigt sich dann mit der detaillierten Spezifikation der Beschreibungssprache, inkl. deren Semantik. Kapitel 6 ab Seite 99 stellt dann den o.g. Automatismus zur Adaption des Prozessmodells während der Prozessausführung vor.

4.4 KONZEPTION DER PROZESSBESCHREIBUNGSSPRACHE

In diesem Kapitel wird das Konzept einer Software-Prozessbeschreibungssprache vorgestellt, die es erlaubt, die in Kapitel 4.2 beschriebene und in Kapitel 6 detailliert spezifizierte Nutzerunterstützung zu realisieren. Die hier vorgestellte Sprache stellt einen Ausschnitt einer produktiv nutzbaren Prozessbeschreibungssprache dar. Es fehlen zum Beispiel Produktzustände („in Bearbeitung", „vorgelegt", „fertig gestellt"), ein Rollenmodell, oder die Möglichkeit Arbeitsschritte hierarchisch zu gruppieren[20]. Eine Erweiterung der Sprache oder eine Integration in eine bestehende Prozessbeschreibungssprache ist jedoch möglich.

Die Prozessbeschreibungssprache unterscheidet grob zwischen verschiedenen Bereichen. Diese sind in Abbildung 28 skizziert und werden im Folgenden kurz erläutert.

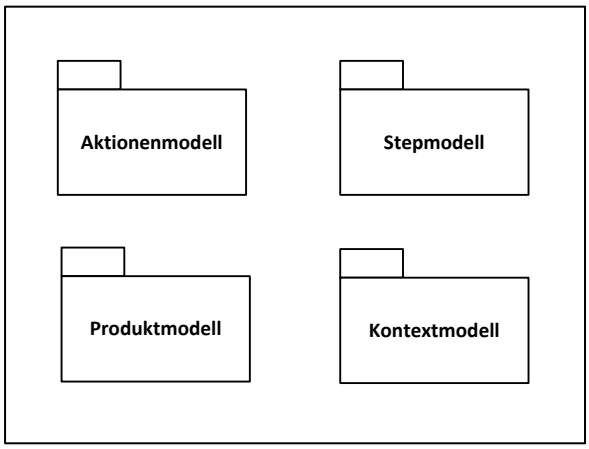


Abbildung 28: Bereiche der Prozessbeschreibungssprache

Das Produktmodell dient zur Beschreibung von Produkten, also z.B. Dokumenten oder anderen Artefakten. Das Kontextmodell wird genutzt, um für einen Prozessablauf relevante

[20] Eine Komposition von Arbeitsschritten ist mit der in dieser Arbeit vorgestellten Sprache zunächst einmal nicht möglich, da dies zur Lösung des eigentlichen Problems nicht nötig ist.

Gegebenheiten zu beschreiben. Eine Untergruppe des Kontextmodells ist das Produktkontextmodell. Im Gegensatz zum Kontextmodell greift dieses spezifisch in das Produktmodell und wird genutzt, um „Schnitte“ durch dieses zu ziehen. Produktkontexte können zum Beispiel eine fachliche Klassifizierung des Produktmodells darstellen. Das Aktionenmodell beschreibt Aktionen (also Änderungen) auf dem Produkt- oder Produktkontextmodell (z.B. Hinzufügen eines Dokuments). Das Stepmodell enthält Steps, also Arbeitsschritte, die ein Projektmitarbeiter ausführt. Ein Step enthält eine Aktion aus dem Aktionenmodell und greift auch auf Elemente der anderen Modelle zu.

Bei der hier vorgestellten Prozessbeschreibungssprache handelt es sich um eine regelbasierte Sprache. Das bedeutet, es werden keine Abfolgen von Arbeitsschritten, Steps genannt, beschrieben, wie dies zum Beispiel bei UML-Aktivitäten oder Petri-Netzen der Fall ist. Steps haben jedoch Vor- und Nachbedingungen. Eine mögliche Abfolge von Steps während eines Projektes ergibt sich durch wahr werdende Vorbedingungen. Alle Steps, deren Vorbedingungen zu einem Zeitpunkt t erfüllt sind, sind prinzipiell lauffähig und können gestartet werden. Ein vom Benutzer gestarteter Step kann beendet werden, sobald die Nachbedingung erfüllt ist. Das bedeutet, es gibt zu jedem Zeitpunkt während eines Projektes eine Menge von Steps, die ein Benutzer starten kann, und eine Menge von Steps, die er beenden kann.

Abbildung 29 zeigt mögliche Prozesssequenzen, die Projektmitarbeiter ausführen können. Ein einfaches Bespiel ist in der Abbildung oben dargestellt: Ein Benutzer kann einen Step starten (dessen Vorbedingungen erfüllt sind), diesen bearbeiten und ihn wieder beenden, sofern die Nachbedingungen erfüllt sind. Daraufhin startet er den nächsten Step, bearbeitet diesen und beendet ihn wieder usw. Eine komplexere Abfolge ist in Abbildung 29 unten dargestellt: Hier startet ein Projektmitarbeiter den Step „Komponente spezifizieren“, beendet diesen jedoch nicht direkt, sondern startet weitere Steps, die er ebenfalls bearbeitet. Erst am Ende der Abfolge wird der Step „Komponente spezifizieren“ beendet.

Das folgende Unterkapitel beschreibt die Konzeption des Produktmodells.

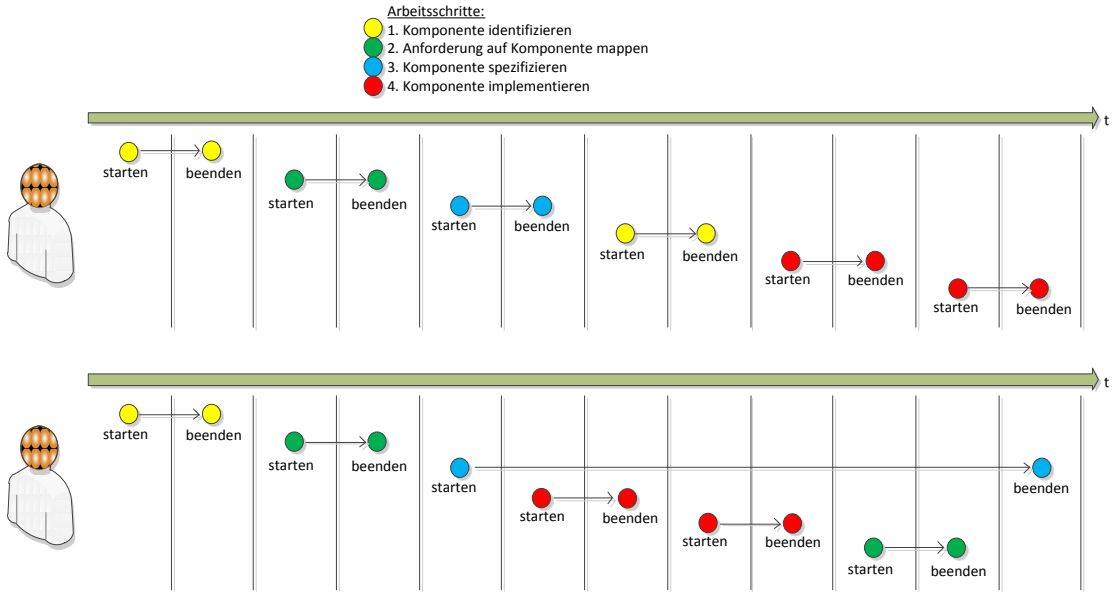


Abbildung 29: Mögliche Stepabfolgen (Prozesse)

4.4.1 PRODUKTMODELL

Die Prozessbeschreibungssprache besitzt ein globales und getyptes Objektmodell. Bei Erstellung einer Prozessbeschreibung ist es also möglich, Typen von Objekten zu definieren (z.B. „Lastenheft“, „Anforderung“, „Komponente“) und deren Beziehung zueinander zu beschreiben (das „Lastenheft“ enthält Elemente vom Typ „Anforderung“). Ein einfaches Beispiel eines Typmodells ist in Abbildung 30 dargestellt. Ein passendes Beispiel eines Objektmodells ist in Abbildung 31 zu sehen.

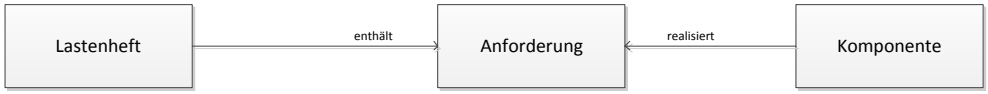


Abbildung 30: Einfaches Beispiel eines Typmodells

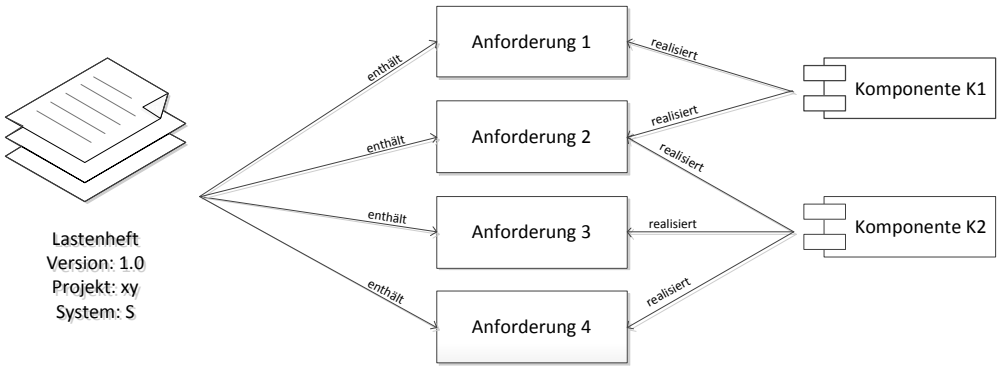


Abbildung 31: Einfaches Beispiel eines Objektmodells

Das folgende Unterkapitel 4.4.3 beschreibt die Konzeption des Kontextmodells.

4.4.2 KONTEXTMODELL

Es ist möglich, Kontexttypen in der Prozessbeschreibung zu hinterlegen. Ein Kontexttyp ist dabei eine Information, die für den Prozessablauf relevant ist. Als Kontexttyp wäre beispielweise „Programmiersprache“ oder auch „Verwendetes Framework“ denkbar. Während der Prozessausführung können diese Kontexttypen dann belegt sein (z.B. Programmiersprache=Java).

Ein spezialisierter Bereich des Kontextmodells sind die Produktkontexte. Diese greifen in das Produktmodell und ziehen Schnitte durch dieses. Diese Produktkontexte werden im folgenden Unterkapitel vorgestellt.

4.4.3 PRODUKTKONTEXT

Ein Produktkontexttyp ist eine Unterklasse von Kontexttypen und beschreibt dabei eine Teilmenge des Produktmodells. Die Motivation dahinter ist eine Form der Nutzerunterstützung, nämlich dass man zu einem Element im Produktmodell alle relevanten dazugehörenden Elemente beschreiben möchte. Wenn ein Projektmitarbeiter beispielsweise ein Element des Produktmodells bearbeitet, kann ein Unterstützungswerkzeug, den relevanten Kontext darstellen. Nehmen wir zum Beispiel an, in unserem Projekt gibt es mehrere Objekte vom Typ

„Komponente“, des Weiteren besitzt jede Komponente eine Menge von Schnittstellen sowie eine Menge von Klassen, die diese Schnittstellen implementieren. Mit Hilfe von Kontexttypen ist es möglich, in der Prozessbeschreibung zusammenhängende Teile im Objektmodell (während der Prozessausführung) zu beschreiben. Jeder Ausführungskontexttyp spezifiziert genau einen Objekttyp als ausgewiesenes Wurzelelement. Zu jedem Objekt dieses Typs gibt es zur Laufzeit genau einen Ausführungskontext.

In Abbildung 32 oben ist beispielhaft ein Typmodell mit Kontexttypen dargestellt. Als Typen gibt es hier die Systemarchitektur, die Komponenten enthalten kann. Eine Komponente realisiert Anforderungen und besitzt Schnittstellen (IF). Außerdem sind zwei Kontexttypen spezifiziert: Der Systemkontext „SystemC“ beschreibt die Systemarchitektur als Wurzelelement und beinhaltet alle anderen Elemente des Typmodells. Der Komponentenkontext „KompC“ besitzt als Wurzelelement den Typ „Komponente“ und enthält Anforderungen und die Schnittstellen der Komponente.

Abbildung 32 unten beschreibt ein mögliches passendes Objektmodell mit Ausführungskontexten während eines Projektes. Hier gibt es ein Objekt vom Typ „Systemarchitektur“ und deshalb einen Ausführungskontext vom Typ „SystemC“. Dieser beinhaltet hier alle vorhandenen Objekte vom Typ „Anforderung“, „Komponente“ und „IF“. Da zwei Komponenten in dem Objektmodell existieren, existieren zwei Ausführungskontexte vom Typ „KompC“. Diese enthalten verschiedene Objekte von den Typen „Anforderung“ und „IF“.

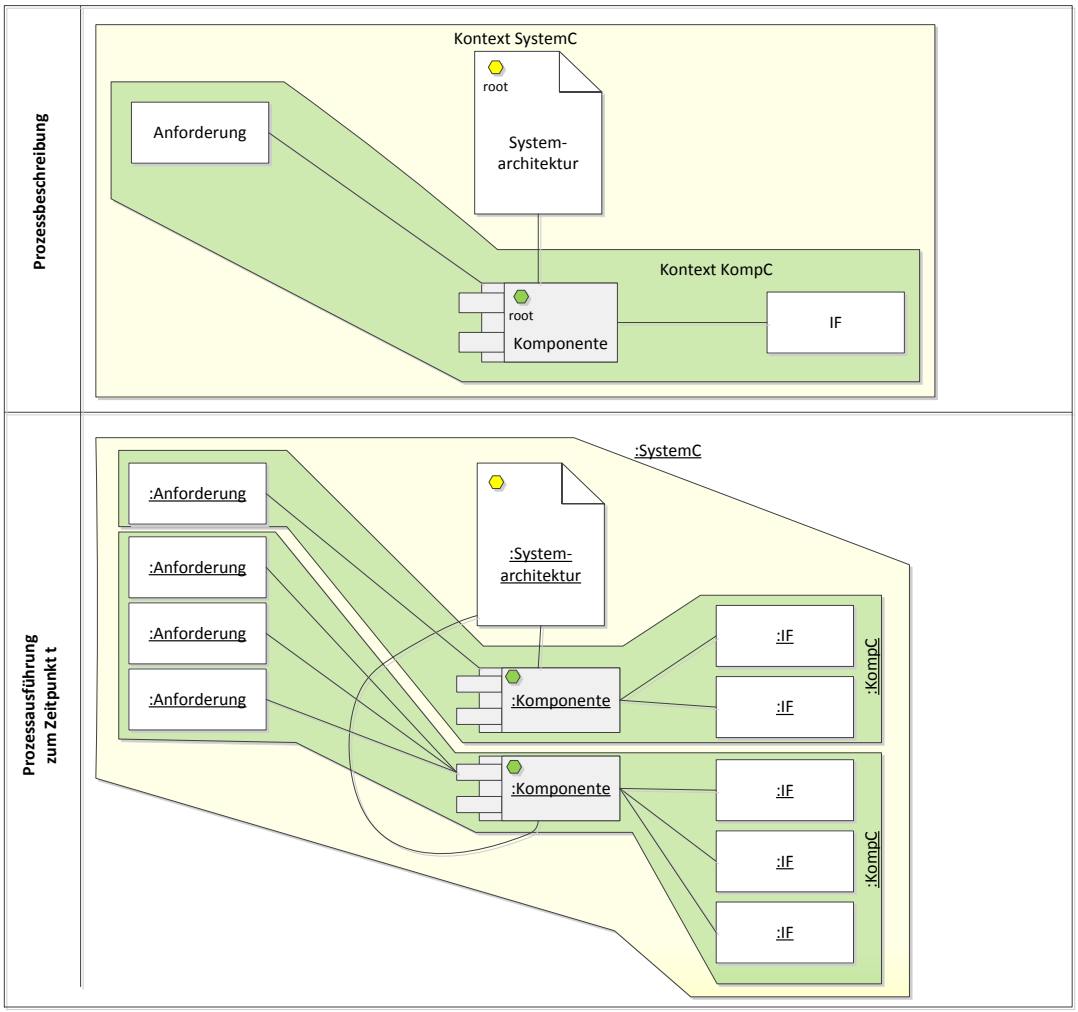


Abbildung 32: Oben: Typmodell mit Kontexttypen, unten: Objektmodell mit Kontextexemplaren

4.4.4 SEMANTIK

In diesem Teil wird die Semantik zur Ausführung der Prozessbeschreibungssprache beschrieben.

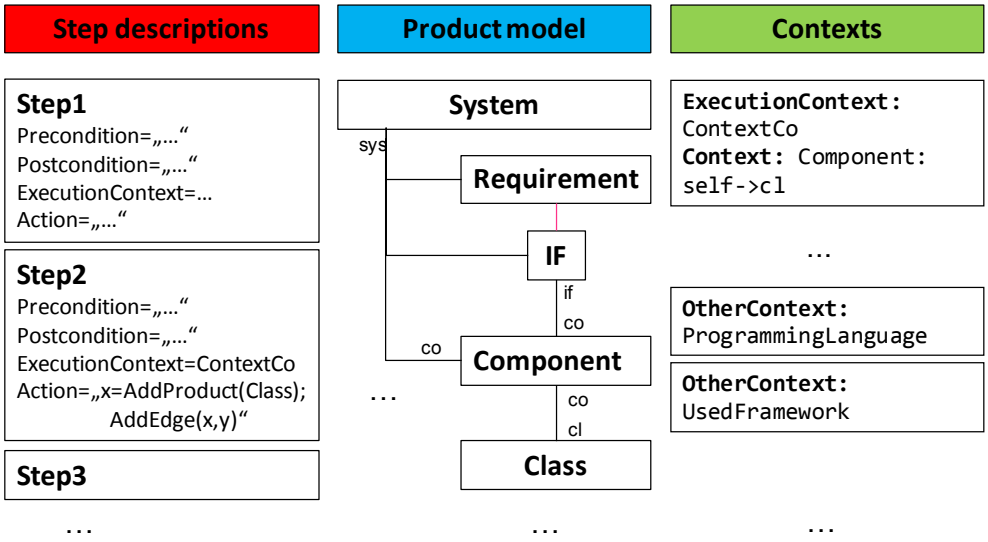


Abbildung 33: Beispielhafte Skizze einer Prozessbeschreibung

Abbildung 33 zeigt eine beispielhafte Skizze einer möglichen Prozessbeschreibung. Diese Prozessbeschreibung besteht aus einer Menge von Steps (links dargestellt). Jeder Step hat neben Vorbedingungen (Preconditions), Nachbedingungen (Postconditions) und einem Ausführungskontext (ExecutionContext) Aktionen (Actions), die Operationen auf dem Produktmodell beschreiben.

In der Mitte von Abbildung 33 ist ein Produkttypmodell skizziert. Als Produkttypen gibt es z.B. das „System“, sowie Anforderungen (Requirement), Komponenten einer Architektur (Component) und deren Schnittstellen (IF). Als weitere, unterhalb der Komponenten liegende Architekturelemente, sind Klassen (Class) in dieser Abbildung erkennbar. Weiterhin sind im Produktmodell die Beziehungen zu den einzelnen Elementen dargestellt: Eine Komponente kann zum Beispiel Schnittstellen besitzen (Verbindung zwischen Component und IF) oder auch Klassen enthalten.

Auf der rechten Seite von Abbildung 33 sind die Kontexte skizziert. Oben ist ein Ausführungskontext „ContextCo“ dargestellt. Zu jeder zu einem Zeitpunkt während der Laufzeit existierenden Komponente (also einem Exemplar von Component) gibt es einen Ausführungskontext. Dies wird in Abbildung 33 mit „Context: Component“ beschrieben. Jedes Exemplar einer Komponente besitzt weiterhin eine Menge von Klassen. Das Ausführungskontextexemplar „ContextCo“ eines Komponentenexemplars enthält weiterhin alle Klassen(exemplare), die von der Komponente referenziert werden (also eine Referenz „cl“ besitzen, wie in Abbildung 33 im Produktmodell dargestellt). Dies wird in der Abbildung mit „self->cl“ beschrieben.

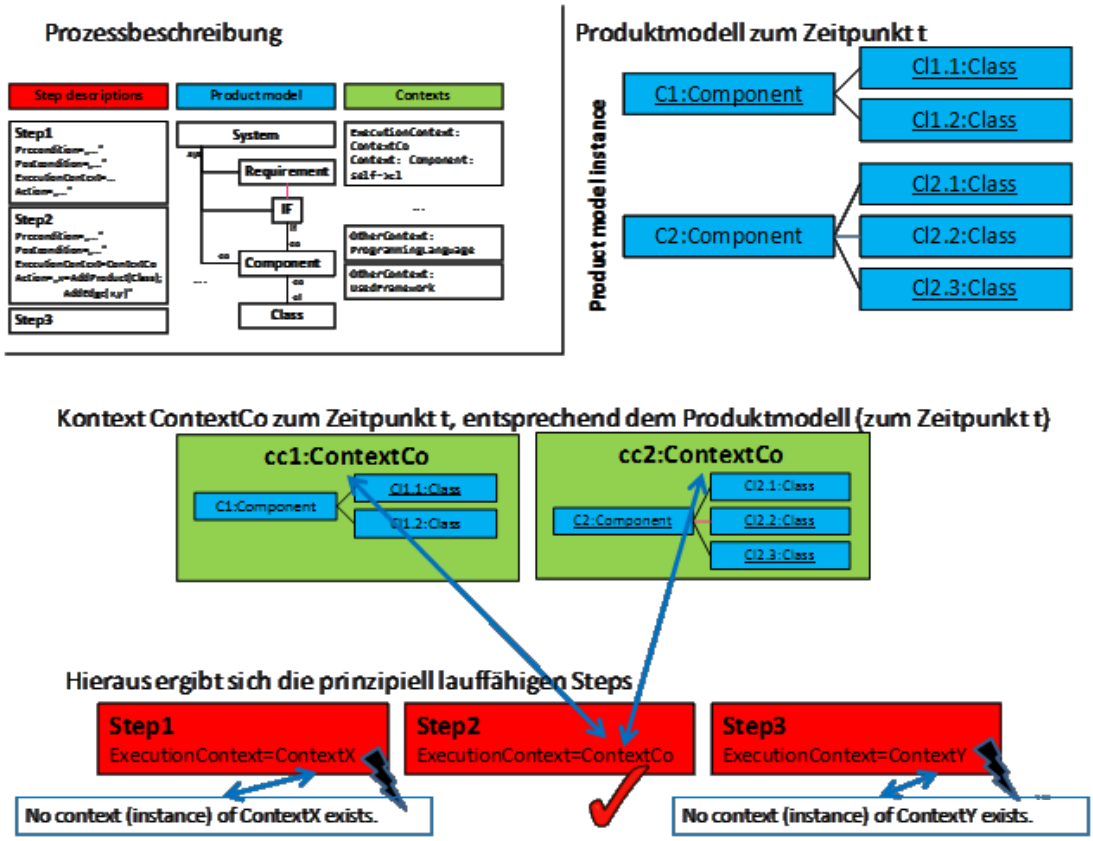


Abbildung 34: Produktmodell, Kontexte und prinzipiell lauffähige Steps zu einen Ausführungszeitpunkt t

In Abbildung 34 ist oben rechts ein Produktexemplarmodell während der Prozessausführung zu einem Zeitpunkt t dargestellt. Aus diesem Produktmodell wird zur Laufzeit eine Menge an Kontextexemplaren gebildet bzw. aktualisiert. Es werden genau zu den in der Prozessbeschreibung spezifizierten Kontexten Kontextexemplare gebildet, zu denen es entsprechende Produktexemplare im Produktmodell gibt. Da in diesem Produktmodell zwei Komponenten C1 und C2 existieren, sind somit auch zwei Kontextexemplare (siehe Abbildung 34, Mitte) vorhanden. Somit ergibt sich, dass nur Step 2 lauffähig ist (falls mindestens eine Precondition bzgl. der beiden Kontextexemplare erfüllt ist). Dieser Step 2 kann somit prinzipiell vom Projektmitarbeiter gestartet werden.

Abbildung 35 zeigt oben eine mögliche Abfolge von Steps, wie diese vom Benutzer gestartet werden könnten. Nach der Bearbeitung eines Steps hat der Projektmitarbeiter dann die Möglichkeit, eine Menge anderer Steps zu starten. Nach Step 2 würde beispielweise ein für Step 3 nötiges Ausführungskontextexemplar existieren. Ebenfalls wäre die für Step 3 nötige Precondition erfüllt und damit wäre Step 3 prinzipiell startbar. Der Projektmitarbeiter startet dann im nachfolgenden Schritt diesen Step.

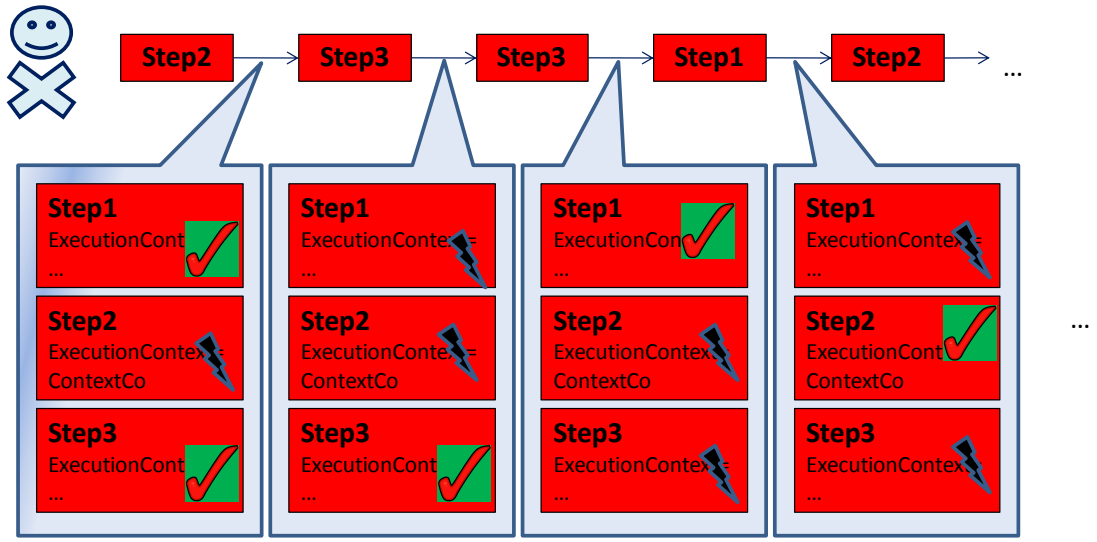


Abbildung 35: Mögliche Stepabfolge

5 SOFTWAREPROZESSBESCHREIBUNGSSPRACHE

Dieses Kapitel beschreibt die Software-Prozessbeschreibungssprache, die bereits in Kapitel 4 konzipiert wurde, im Detail. Einen Überblick über die Spezifikation der Beschreibungssprache gibt Abbildung 36. Es existieren vier grundlegende Bereiche der Sprache:

a) Das Produktmodell,

b) das Produktkontextmodell,

c) das Aktionenmodell und

d) das Stepmodell.

Das Produktmodell beschreibt Produkttypen und deren Beziehungen zueinander. Produkttypen sind dabei Klassen von Produkten mit gleichen Merkmalen. Produkte sind häufig Dokumente (z.B. Lastenheft), können jedoch auch andere Artefakte (z.B. Anforderung, Datei) sein. Der Produkttyp „Lastenheft" beschreibt beispielsweise die Gemeinsamkeiten aller Lastenheftexemplare. Häufig werden hierunter die Kapitelstruktur und der Inhalt der jeweiligen Kapitel der Dokumente verstanden. Produkttypen stehen häufig in Beziehung zueinander. Zum Beispiel haben die Produkttypen „Lastenheft" und „Anforderung" eine Beziehung: Ein Lastenheft enthält eine Menge von Anforderungen. Diese Beziehungen werden über Kanten beschrieben. Weiterhin werden in dem Produktmodell Produktexemplare behandelt. Dies sind konkrete Instanzen zu dem o.g. Produkttypen. Das Lastenheft für Projekt „X" ist beispielsweise ein Produktexemplar. In diesem sind konkrete Anforderungen an ein zu entwickelndes System enthalten. Dementsprechend werden Kantenexemplare beschrieben – dies sind Instanzen zu Kantentypen und setzen Produktexemplare in Beziehung zueinander. Zum Beispiel enthält das Produktexemplar „Lastenheft Projekt X" das Anforderungsexemplar „NF1: Das System muss wartungsfrei sein".

Ein Produktkontext kann als ein Container von einer Menge von Produkten verstanden werden. Durch Produktkontexte können „Schnitte", also Teilmengen durch ein Produktmodell gebildet werden. Die Motivation dahinter ist die Nutzerunterstützung während der Ausführung: Der Projektmitarbeiter kann einerseits direkt unterstützt werden, indem zu einem Produktexemplar alle relevanten anderen Exemplare dargestellt werden. Andererseits werden Produktkontexte genutzt, um kontextspezifisches Vorgehen zu Erlernen und Vorschläge für die Prozessausführung zu machen. Die Konzeption der Produktkontexte ist in Kapitel 4.4.3 beschrieben. Die eigentliche Projektmitarbeiterunterstützung ist in Kapitel 6 detailliert spezifiziert. Für jedes Produkt kann es verschiedene Produktkontexte geben. Hierdurch wird es möglich, „fachliche Klassifizierungen" vorzunehmen. Es ist so zum Beispiel möglich, alle sicherheitsrelevanten Aspekte einer Komponente zu beschreiben. Dies könnten alle sicherheitsrelevanten Anforderungen sowie sicherheitskritische Subkomponenten und deren Implementierung sein. Es ist jedoch auch möglich, zu einer Komponente *alle* relevanten Informationen in einem Produktkontext zu bündeln. Das Produktkontextmodell unterscheidet analog zum Produktmodell zwischen Produktkontexttypen und Produktkontextexemplaren.

Jeder Produktkontexttyp beschreibt einen Produkttyp als „Wurzelelement". Zu jedem Produktexemplar dieses Produkttyps gibt es genau ein Produktkontextexemplar des Produktkontexttyps. Zu einem solchen Produktkontextexemplar können weitere Produktexemplare hinzugefügt/entfernt werden. Die möglichen Produkttypen zu diesen Exemplaren werden ebenfalls in den Produktkontexttypen festgelegt.

Das Produktkontextmodell umfasst Produktkontexttypen und Produktkontextexemplare. Mit Produktkontexttypen werden Produktkontexte im Allgemeinen, also auf Typebene, beschrieben. Hiermit wird also definiert, zu welchen Produkttypen es Produktkontexte gibt. Weiterhin wird beschrieben, welche Produkttypen in die Produktkontexte mit aufgenommen werden können.

Das Aktionenmodell beschreibt Änderungen auf dem Produktmodell und Produktkontextmodell. Hierbei handelt es sich um „fachliche“ Aktionen auf den Modellen, zum Beispiel das Hinzufügen einer Anforderung zu dem Produktmodell. Diese Aktionen werden durch eine Abfolge von (atomaren) Operationen beschrieben. Es wird dabei zwischen Operationen auf dem Produktmodell und Operationen auf dem Produktkontextmodell unterschieden. Die Operationen auf dem Produktmodell umfassen das Hinzufügen und Entfernen von Produktexemplaren sowie das Hinzufügen und Entfernen von Kantenexemplaren. Mit Hilfe der Produktkontextoperationen ist es möglich, Produktexemplare einem Produktkontextexemplar hinzuzufügen oder ein Produktexemplar aus einem Produktkontextexemplar zu entfernen. Die Beschreibung der Operationen erfolgt mittels Variablen, also ungebundenen Platzhaltern für Produkt-/Kantenexemplare und Produktkontextexemplare. Zur Laufzeit müssen diese Platzhalter mittels Variablenbelegungen an Produkt-/Kantenexemplare gebunden sein. Die konkrete Variablenbelegung während der Laufzeit erfolgt dann über die Arbeitsschritte (Steps), die im nächsten Absatz beschrieben werden. Unser Variablenkonzept beinhaltet nur globale Variablen. Dies ist ausreichend, weil wir keine Hierarchien bzw. Kompositionen von Arbeitsschritten modellieren (s.u.) und deshalb auch nicht die Übergabe von Variablen behandeln müssen (z.B. call-by-value, call-by-reference).

Das Stepmodell beschreibt Steps, also Arbeitsschritte, die der Projektmitarbeiter durchführt. Jeder Step wird unter einem ausgewiesenen Produktkontext ausgeführt. Das Starten eines Steps ist nur möglich, wenn es ein entsprechendes Produktkontextexemplar gibt. Weiterhin beschreibt ein Step Vor- und Nachbedingungen. Die Vorbedingungen müssen erfüllt sein, um einen Arbeitsschritt starten zu können. Entsprechend müssen die Nachbedingungen erfüllt sein, um einen gestarteten Step beenden zu können. Ein Arbeitsschritt beschreibt weiterhin eine Aktion, dies ist eine Abfolge von Operationen aus dem Aktionenmodell. In unserer Prozessbeschreibungssprache stehen Steps „flach nebeneinander“. Das bedeutet, in unserer Sprache können keine Kompositionen von Steps modelliert werden. Deshalb hat der Projektmitarbeiter die freie Wahl, welche Arbeitsschritte er startet und beendet (abhängig von den Vor- und Nachbedingungen). Ein Arbeitsschritt enthält die nötigen Variablenbelegungen, um die hinterlegte Aktion, die nur über ungebundene Variablen verfügt, auszuführen.

Im Folgenden werden diese vier Modelle im Detail beschreiben.

5.1 PRODUKTMODELL

Das Produktmodell der hier vorgestellten Software-Prozessbeschreibungssprache ist in Abbildung 37 in UML-Notation dargestellt. Hier ist zu beachten, dass dies kein vollständiges Modell für den produktiven Einsatz darstellt. Es sind lediglich relevante Elemente zur Unterstützung des Projektmitarbeiters enthalten. Andere nötige Elemente für den produktiven Einsatz – zum Beispiel Produktzustände (z.B. enthalten in [85]) – können jedoch problemlos erweitert werden. Das ist in dieser Arbeit jedoch nicht vorgesehen.

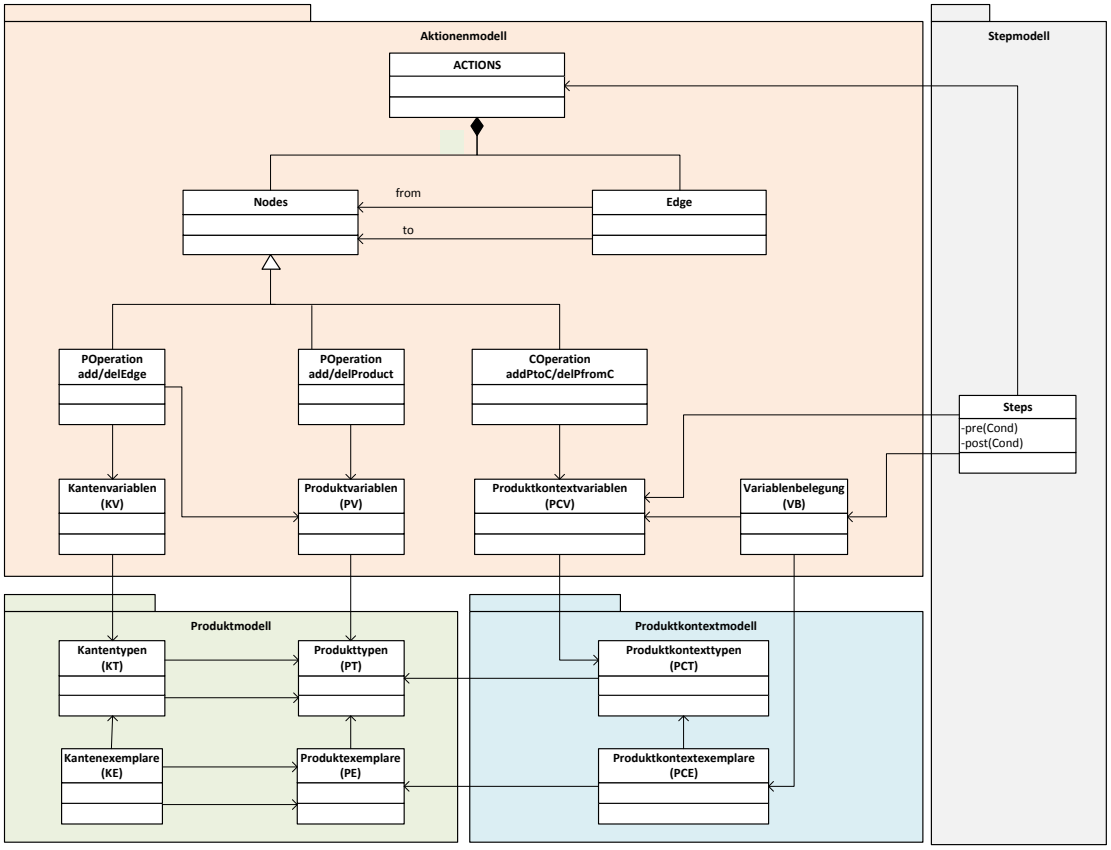


Abbildung 36: Überblick über die Software-Prozessbeschreibungssprache

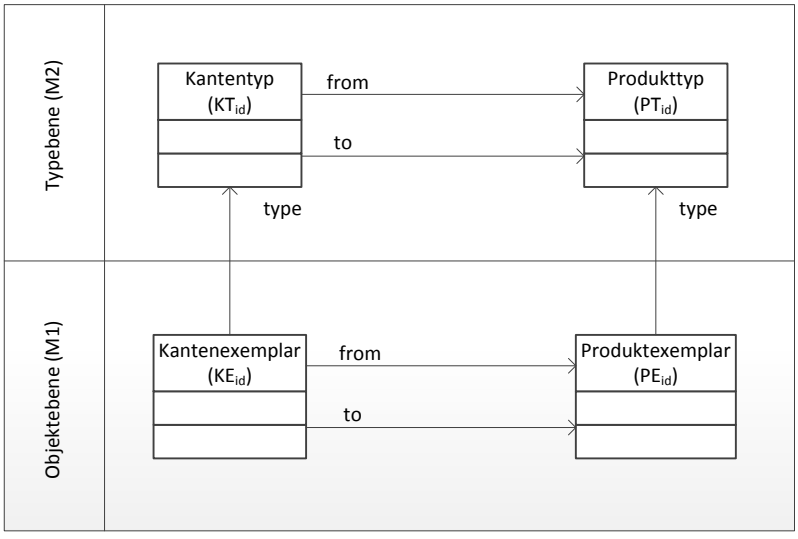


Abbildung 37: Produktmodell in UML-Notation

Die Modelle in dieser Arbeit unterscheiden zwei Klassen von Elementen: Typelemente und Exemplarelemente. Dies ist ein bewährtes Konzept, das zum Beispiel auch bei den Mustern „Item-Item Descriptor“ [86], [87] und „Exemplartyp“ [88] Anwendung findet.

Deshalb werden die Modellelemente hier auf zwei Ebenen aufgeteilt: Die Typebene und die Objektebene. Die Elemente der Typebene entsprechen den Beschreibungselementen beim „Item-Item Descriptor Pattern“, die Elemente der Objektebene den Objektelementen.

Das Produktmodell besteht auf der Typebene aus zwei Elementen: Produkttyp und Kantentyp. „Lastenheft“ oder „Anforderung“ könnte zum Beispiel ein Produkttyp sein. Mittels Kantentypen können Produkttypen in Verbindung zueinander gesetzt werden. Kantentypen werden hier explizit modelliert und mit den Referenzen „von“ und „zu“ zu den jeweiligen Produkttypen verbunden. Ein Kantentyp mit dem Namen „ist enthalten in“ wäre hier denkbar und verbindet den Produkttyp „Anforderung“ mit dem Produkttyp „Lastenheft“.

Auf der Objektebene gibt es, entsprechend der Typebene, ebenfalls zwei Elemente: Produktexemplar und Kantenexemplar. Ein Produktexemplar ist beispielsweise ein konkretes Lastenheft (eines konkreten Projektes) oder eine konkrete, reale Anforderung an ein System. Diese Produktexemplare können – ensprechend den Vorgaben auf der Typebene – mittels Kantenexemplaren in Verbindung zueinander gesetzt werden. Gibt es z.B. die Produkttypen „Lastenheft“ und „Anforderung“ sowie einen Kantentyp „ist enthalten in“ von „Anforderung“ zu „Lastenheft“, so kann es die Produktexemplare „Lastenheft für Projekt X“, „Anforderung 1“ und „Anforderung 2“ geben. Weiterhin könnte es Kantenexemplare geben, die von „Anforderung 1“ und „Anforderung 2“ auf das „Lastenheft Projekt X“ verweisen. Dies entspräche den Vorgaben auf der Typebene.

In der obigen Beschreibung besitzen die Kanten zwischen zwei Modellelementen ein eigenes Element: Kantentyp/-exemplar. Das bedeutet, dass eine Kante zwischen zwei Elementen durch ein Element vom Kantentyp/-exemplar modelliert wird. Diese Kantenelemente besitzen die Referenzen „von“ und „zu“ zu den jeweiligen Produktelementen. Diese Modellierung ist in Abbildung 38 (M1, rechte Seite) dargestellt. Der Einfachheit halber kann von dieser technischen Modellierung abgesehen, und die Kanten können nur mittels Pfeilen skizziert werden (siehe Abbildung 38, M1, links).

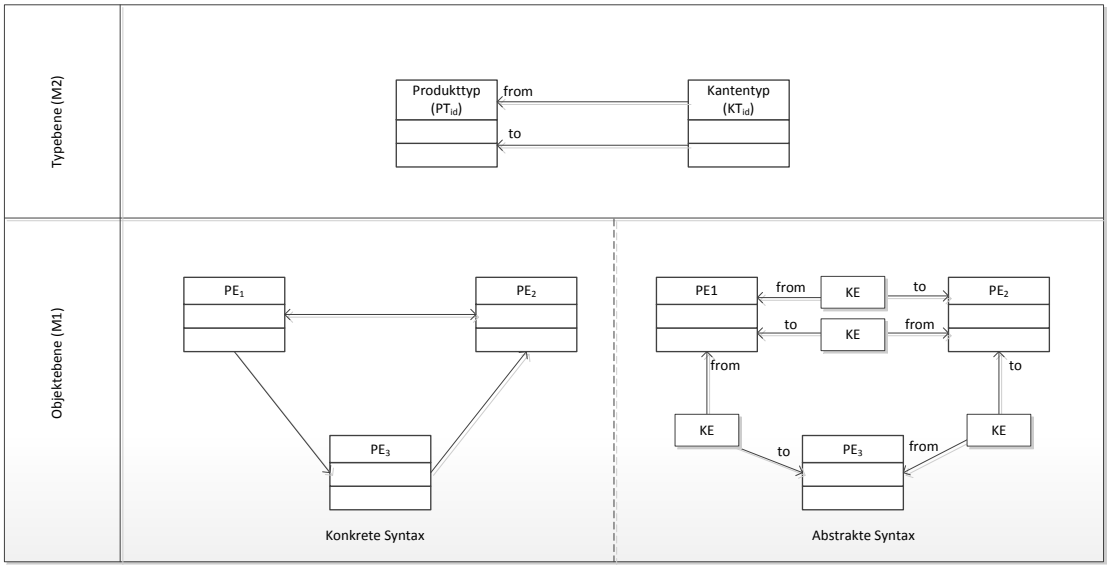


Abbildung 38: Produktmodell – Technische Modellierung der Kanten und ihre grafische Darstellung

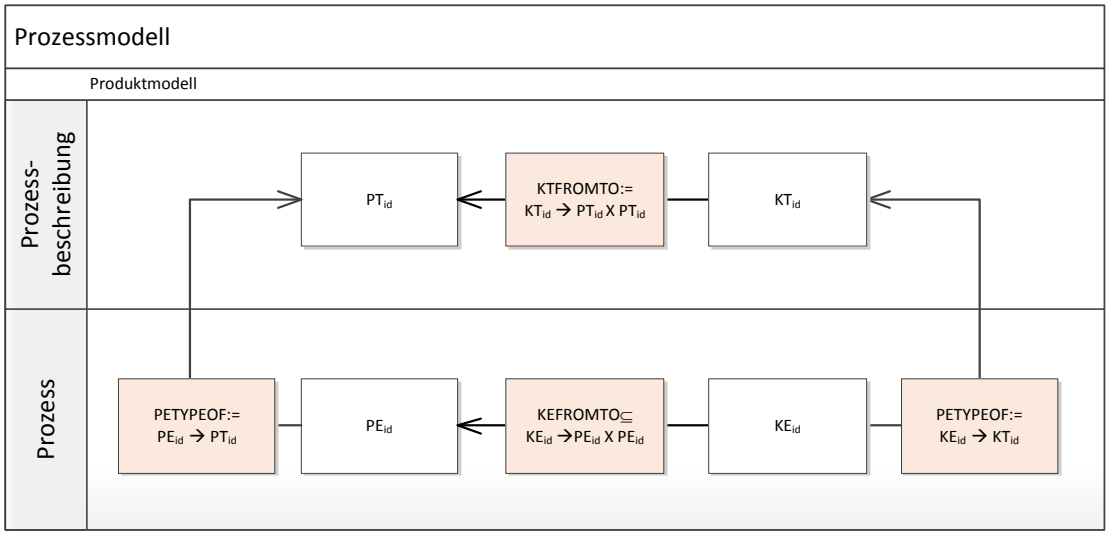


Abbildung 39: Produktmodell in algebraischer Repräsentation

Das Produktmodell wird nun im Folgenden mit einer algebraischen Spezifikationstechnik detailliert beschrieben. Eine Repräsentation dieses Modells ist in Abbildung 39 dargestellt.

Als Erstes definieren wir die Elemente von Abbildung 37 als Mengen. Die Mengenrepräsentation des Modells von Abbildung 37 ist in Abbildung 39 dargestellt:

Sei PT_{id} die Menge aller möglichen Produkttypen. *(2)*

Sei KT_{id} die Menge aller möglichen Kantentypen. *(3)*

Sei PE_{id} die Menge aller möglichen Produktexemplare. *(4)*

Sei KE_{id} die Menge der möglichen Kantenexemplare. *(5)*

Nun beschreiben wir die Referenzen des UML-Modells (Abbildung 37) mit Hilfe von Funktionen, die über die oben beschriebenen Mengen definiert sind. Zunächst spezifizieren wir die „from/to"-Referenzen auf der Typebene zwischen Kantentypen und Produkttypen.

Jeder Kantentyp beschreibt eine unidirektionale Referenz zwischen zwei Produkttypen. Diese werden durch die Menge $KTFROMTO$ beschrieben:

Sei $KTFROMTO :=_{DEF} KT_{id} \rightarrow PT_{id} \times PT_{id}$ die Menge aller möglichen Kantentypen zwischen Produkttypen. *(6)*

Sei $k \in KT_{id}$ ein beliebiger Kantentyp. Dann beschreibt $from_k$ den ersten Produkttyp und to_k den zweiten Produkttyp dieses Kantentyps.

Des Weiteren besitzt jedes Produktexemplar einen Produkttyp:

Sei $PETYPE :=_{DEF} PE_{id} \rightarrow PT_{id}$ die Menge aller möglichen Zuordnungen zwischen Produktexemplaren und Produkttypen. *(7)*

Sei $p \in PE_{id}$ ein beliebiges Produktexemplar. Dann beschreibt $type_p$ den Produkttyp von p.

Ebenso besitzt jedes Kantenexemplar einen Kantentyp:

Sei $KETYPE :=_{DEF} KE_{id} \rightarrow KT_{id}$ die Menge aller möglichen Zuordnungen zwischen Kantenexemplaren und Kantentypen. *(8)*

Sei $ke \in KE_{id}$ ein beliebiges Kantenexemplar. Dann beschreibt $type_{ke}$ den Kantentyp von ke

Entsprechend den Kantentypen beschreibt jedes Kantenexemplar eine unidirektionale Referenz zwischen zwei Produktexemplaren. Dabei müssen die Produkttypen der referenzierten Produktexemplare mit den Produkttypen, die der Kantentyp beschreibt, übereinstimmen:

Sei $KEFROMTO :=_{DEF} \{(ke, pe_a, pe_b) \in KE_{id} \rightarrow PE_{id} \times PE_{id} | type_{pe_a} = from_{type_{ke}} \wedge type_{pe_b} = to_{type_{ke}}\}$ die Menge aller möglichen Kantenexemplare. *(9)*
Sei $ke = (id, pe1, pe2) \in KEFROMTO$ ein beliebiges Element. Dann beschreibt $from_{ke}$ das erste Produktexemplare $pe1$ und to_{ke} das zweite Produktexemplar $pe2$.

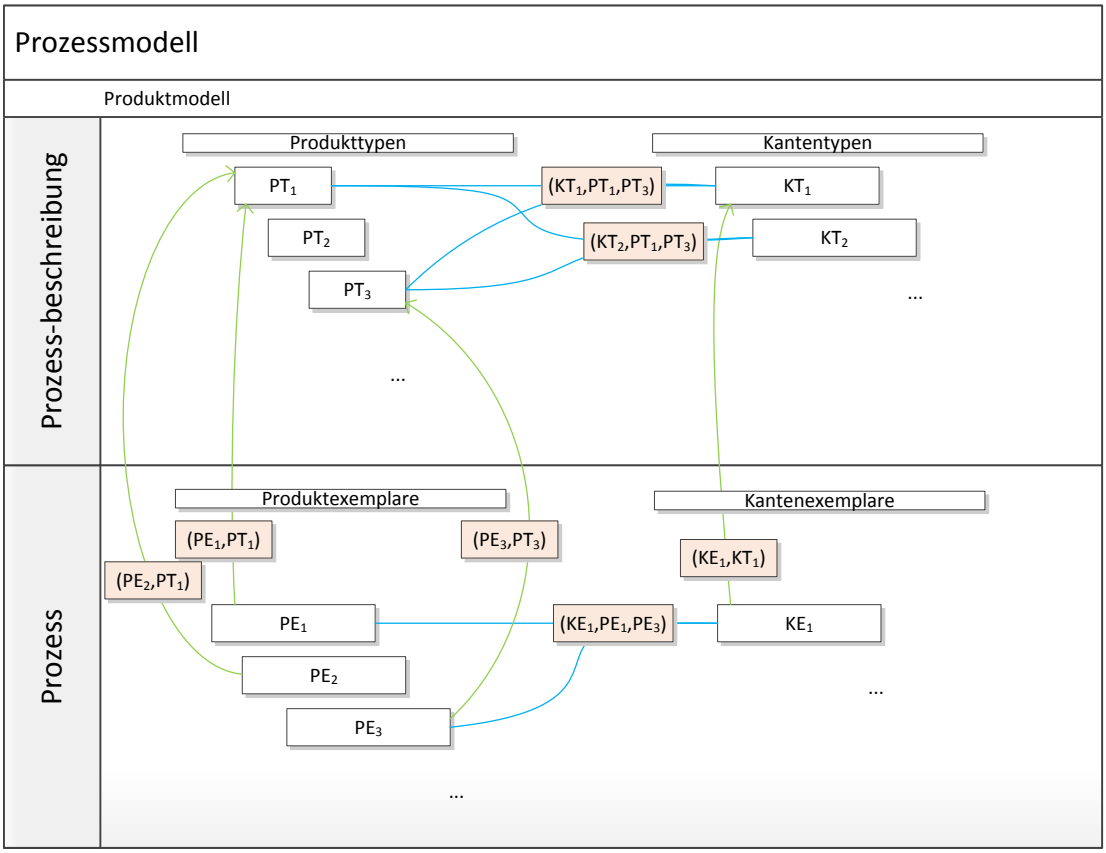


Abbildung 40: Beispiel eines Produktmodells

Ein abstraktes Beispiel eines Produktmodells ist in Abbildung 40 dargestellt. Entsprechend Abbildung 39 sind im oberen Bereich Produkttypen und Kantentypen visualisiert, im unteren Bereich befinden sich die Produkt- und Kantenexemplare. Folgende Notation wird dabei verwendet: Eine ID eines Typs/Exemplars trägt den Namen der Abkürzungsbeschreibung (PT=Produkttyp, PE=Produktexemplar, etc. – siehe Abbildung 39 und Beschreibung auf Seite

57) gefolgt von einer Zahl. Diese Elemente werden in Grau dargestellt. „PT_1“ beschreibt z.B. eine ID, also die Repräsentation eines Produkttyps.

In Abbildung 40 sind auch die Funktionen, also die Referenzen zwischen den Modellelementen (Produkttypen/-exemplare, Kantentypen/-exemplare) dargestellt. Diese Funktionen sind in roter Farbe abgebildet. (KT_1, PT_1, PT_3) beschreibt beispielsweise die Referenz, die mit der Funktion $KTFROMTO$ beschrieben wird. KT_1 ist dabei das Argument und PT_1 und PT_3 der Funktionswert ($KTFROMTO(KT_1) = (PT_1, PT_3)$ oder: $from_{KT_1} = PT_1$ und $to_{KT_1} = PT_3$).

Das Beispiel-Produktmodell aus Abbildung 40 beschreibt die „technische“ Modellierung eines Modells, also die abstrakte Syntax. Die grafische Repräsentation, also die konkrete Syntax, ist der Vollständigkeit halber in Abbildung 41 dargestellt. Hier wird UML-Notation zur Repräsentation verwendet.

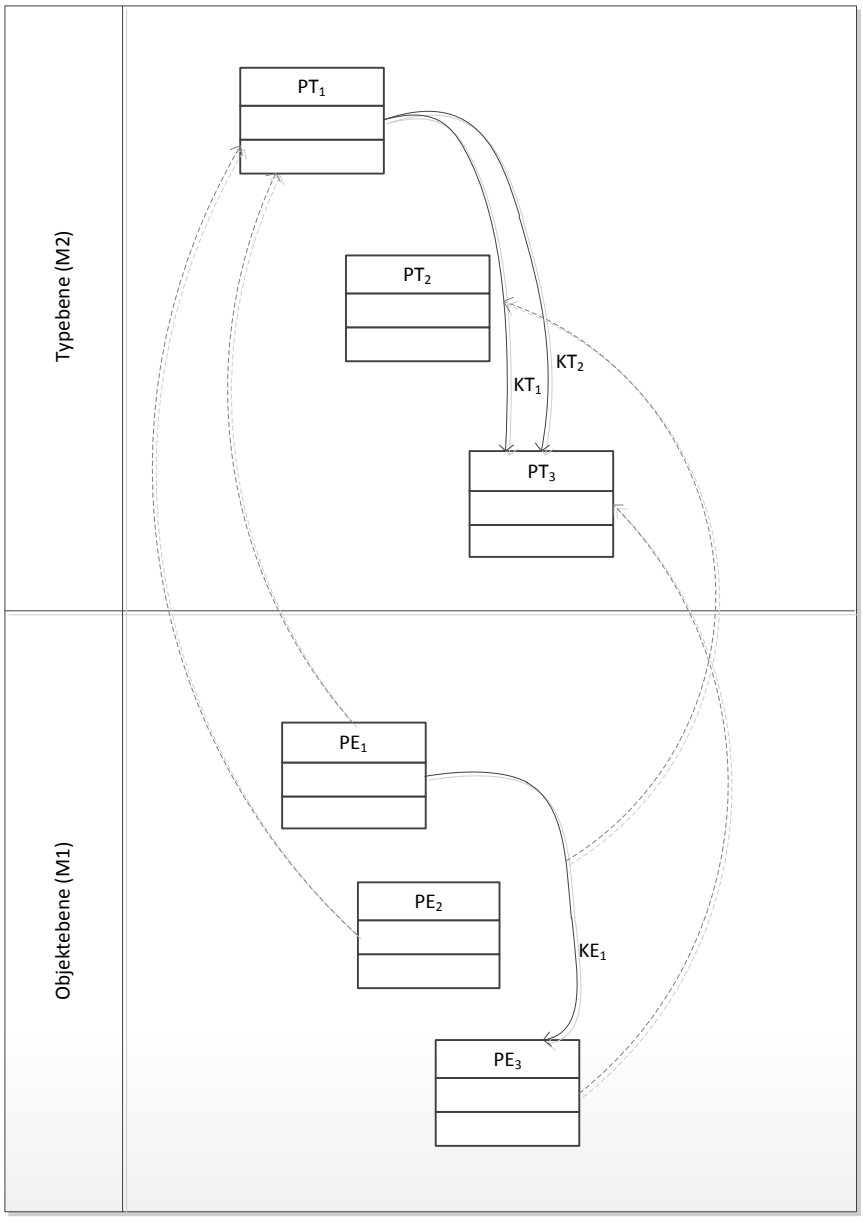


Abbildung 41: Beispiel des Produktmodells in UML-Notation

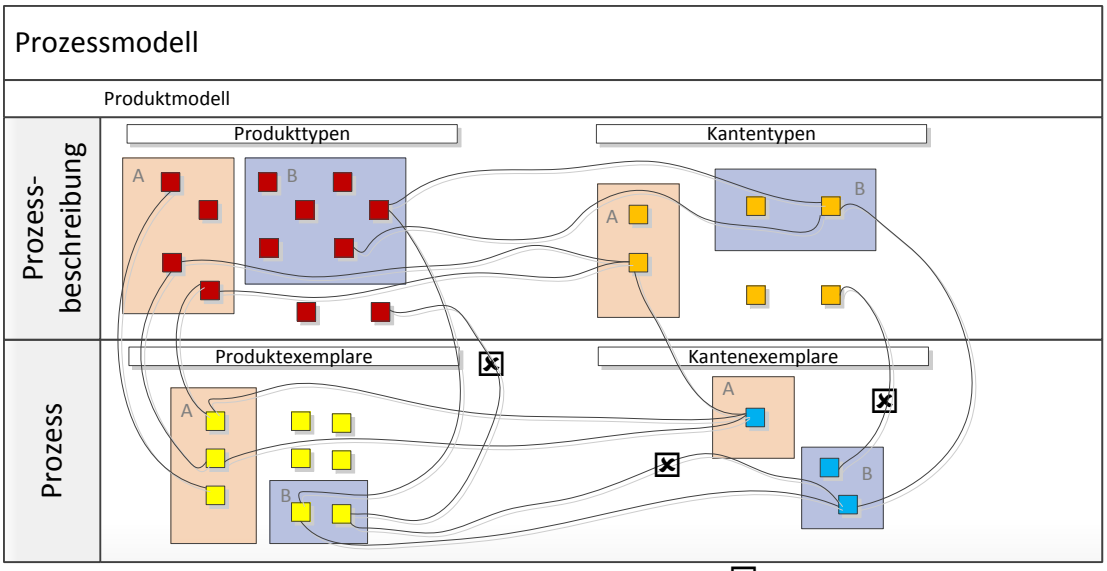


Abbildung 42: Konformes Produktmodell A, nicht-konformes Modell B

Bis jetzt wurden die Bereiche der Produkttypen/-exemplare und Kantentypen/-exemplare isoliert voneinander betrachtet. Um ein konformes Produktmodell zu erhalten, gibt es jedoch Einschränkungen, die für jedes vollständige Produktmodell gelten müssen.

Die Menge aller möglichen, konsistenten Produktmodelle, bestehend aus der Menge der Produkttypen, Produktexemplare, Kantentypen und Kantenexemplare, ist dann wie folgt definiert:

$$\boxed{PM} :=_{DEF} \{(\boxed{PT_{PM}} \subseteq PT_{id}, \boxed{PE_{PM}} \subseteq PE_{id}, \boxed{KT_{PM}} \subseteq KT_{id}, \boxed{KE_{PM}} \subseteq KE_{id}, \boxed{KTFROMTO_{PM}} \subseteq KTFROMTO, \boxed{PETYPEOF_{PM}} \subseteq PETYPEOF, \boxed{KETYPEOF_{PM}} \subseteq KETYPEOF, \boxed{KEFROMTO_{PM}} \subseteq KEFROMTO)\} \quad (10)$$

mit:

- Zu jedem Produktexemplar gibt es genau einen Produkttyp:

$$\forall pe \in \boxed{PE_{PM}} : \exists! \, t_{id} \in \boxed{PT_{PM}} \text{ mit } type_{pe} = t_{id}$$

- Für jedes der beiden Kantenenden pro Kantentyp gibt es genau einen Produkttyp:

$$\forall kt \in \boxed{KT_{PM}} : \exists! \, pt_x, pt_y \in \boxed{PT_{PM}} \text{ mit } from_{kt} = pt_x \text{ und } to_{kt} = pt_y$$

- Zu jedem Kantenexemplar gibt es genau einen Kantentyp und für jedes der beiden Kantenenden pro Kantenexemplar gibt es ein Produktexemplar:

$$\forall k \in \boxed{KE_{PM}} : \exists! \, kt \in \boxed{KT_{PM}} \text{ mit } type_k = kt.$$

$$\text{und } \exists! \, m_x, m_y \in \boxed{PE_{PM}} \text{ mit } from_k = m_x \text{ und } to_k = m_y$$

- Jedes Kantenexemplar besitzt zwei Referenzen auf Produktexemplare, und jedes Kantenexemplar hat einen Kantentyp. Die Produkttypen

dieser Produktexemplare müssen mit den im Kantentyp spezifizierten Produkttypen übereinstimmen:

$$\forall \mathrm{ke} \in \boxed{\mathrm{KE_{PM}}}: \mathrm{from}_{\mathrm{type_{ke}}} = \mathrm{type}_{\mathrm{from_{ke}}} \text{ und } \mathrm{to}_{\mathrm{type_{ke}}} = \mathrm{type}_{\mathrm{to_{ke}}}.$$

Abbildung 42 zeigt beispielhaft eine Menge von Produktelementen (Produkttypen, Produktexemplaren, Kantentypen und Kantenexemplaren) und zeigt Mengen, die ein konsistentes Produktmodell repräsentieren (A), sowie Mengen, die ein nicht-konsistentes Produktmodell darstellen (B).

Nachfolgend werden noch einige Klassifizierungen aufgeführt, die wir im späteren Verlauf der Arbeit verwendet werden. Diese Klassifizierungen beziehen sich auf ein Teilmodell (z.B. $\boxed{PE}$) eines Produktmodells $\boxed{PM}$.

Die folgende Klassifizierung liefert zu einem Produkttyp die Menge der passenden Produktexemplare:

$$getPE_{\boxed{PE}-pt} :=_{DEF} \{Pe \subseteq \boxed{PE} | \forall pe \in \boxed{PE} \text{ mit } pt = type_{pe} \Rightarrow pe \in Pe; \text{ sonst: } pe \notin Pe\} \quad (11)$$

Diese Klassifizierung beschreibt zu einem Produktexemplar alle Kantenexemplare, die auf das angegebene Produktexemplar referenzieren:

$$getKE_{\boxed{KE}-pe} :=_{DEF} \{Ke \subseteq \boxed{KE} | \forall ke \in \boxed{KE} \text{ mit } from_{ke} = pe \vee to_{ke} = pe \Rightarrow ke \in Ke; \text{ sonst: } ke \notin Ke\} \quad (12)$$

In Abbildung 43 ist das Produktmodell der Beispielmethodik, beschrieben in Kapitel 3, in konkreter Syntax dargestellt. Im oberen Bereich befinden sich die Produkt- und Kantentypen, im unteren Bereich die Produkt- und Kantenexemplare. Im linken Bereich der Typebene befindet sich das Lastenheft, das Anforderungen enthalten kann. Rechts ist die Systemarchitektur zu finden, in der das System in Komponenten dekomponiert ist. Komponenten besitzen Schnittstellen (IF), die Anforderungen umsetzen. Zu einer Komponente existiert eine Spezifikation, die Schnittstellenbeschreibungen (IFD) und deren Implementierung (IFI) enthält. Auf der Objektebene gibt es gemäß Abbildung 43 ein konkretes Lastenheft, das drei Anforderungen enthält. Des Weiteren existiert bisher nur eine Systemarchitektur, die keine Inhalte (z.B. Komponenten) enthält.

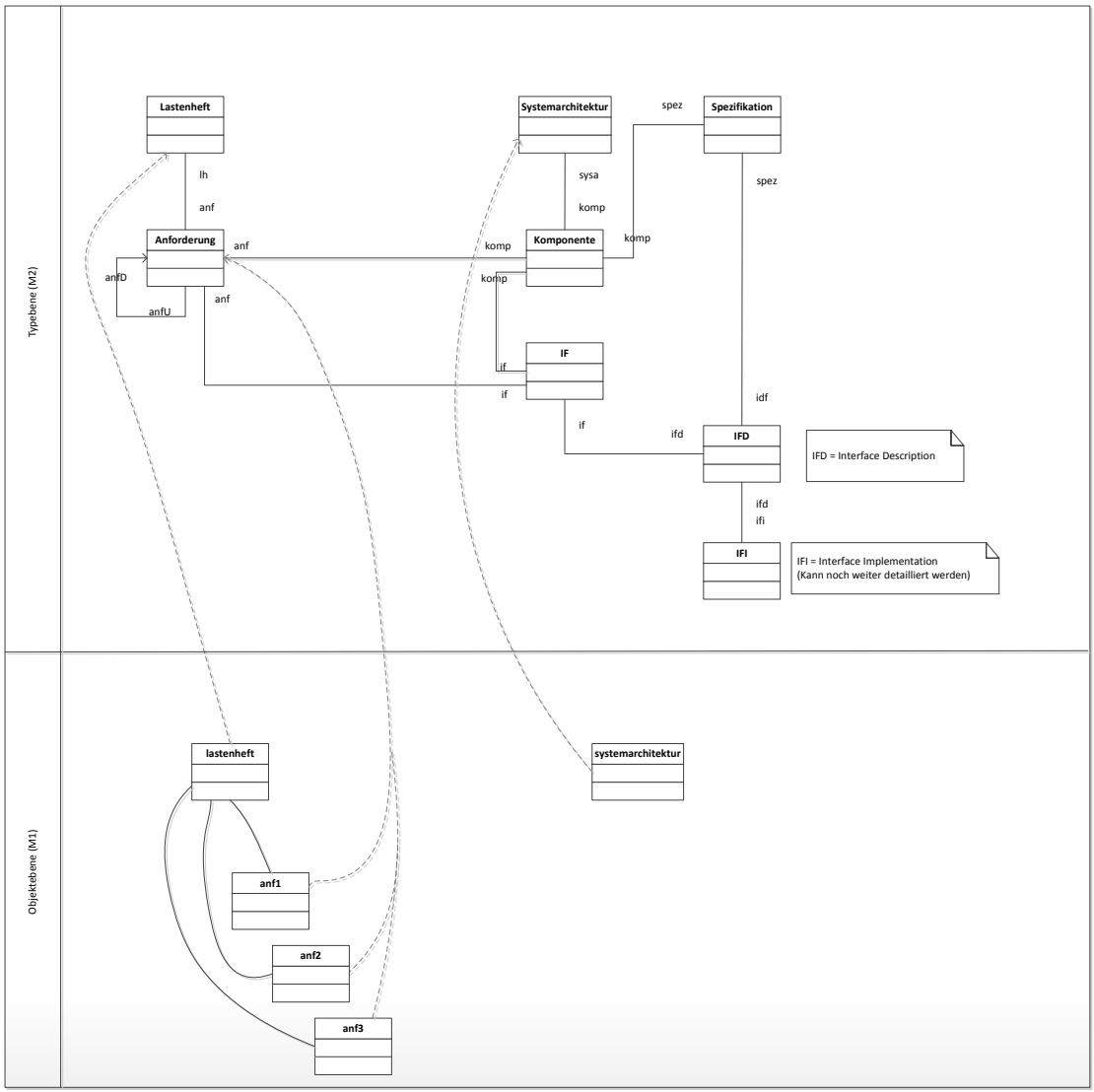


Abbildung 43: Produktmodell der Beispielmethodik in UML-Notation

Im Folgenden ist das in Abbildung 43 dargestellte Produktmodell in Form der vorher erarbeiteten abstrakten Syntax in Mengendarstellung aufgeführt:

- $\boxed{pm_a}$ = (`// Produkttypen (PT)`

  {Lastenheft,Anforderung,IF,Systemarchitektur,Komponente,

  Spezifikation,IFD,IFI},

  `// Produktexemplare (PE)`

  {lastenheft,systemarchitektur,anf1,anf2,anf3},

  `// Kantentypen (KT)`

  {lh-anf,anf-lh,anfD-anfU,anf-komp,komp-anf,

anf-if,if-anf,sysa-komp,komp-sysa,komp-if,if-komp,komp-spez,

spez-komp,spez-ifd,ifd-spez,if-ifd,ifd-if,ifd-ifi,ifi-ifd},

// Kantenexemplare (KE)

{lh-anf1,anf1-lh,lh-anf2,anf2-lh,lh-anf3,anf3-lh}

// Kantentypen From/To (KTFROMTO)

{(lh-anf,Lastenheft,Anforderung),(anf-lh,Anforderung,Lastenheft),

(anfD-anfU,Anforderung,Anforderung),

(anf-komp,Anforderung,Komponente),

(komp-anf,Komponente,Anforderung),

(anf-if,Anforderung,IF),(if-anf,IF,Anforderung),

(sysa-komp,Systemarchitektur,Komponente),

(komp-sysa,Komponente,Systemarchitektur),

(komp-if,Komponente,IF),(if-komp,IF,Komponente),

(komp-spez,Komponente,Spezifikation),

(spez-komp,Spezifikation,Komponente),(spez-ifd,Spezifikation,IFD),

(ifd-spez,IFD,Spezifikation),(if-ifd,IF,IFD),(ifd-if,IFD,IF),

(ifd-ifi,IFD,IFI),(ifi-ifd,IFI,IFD)}

// Produktexemplare TypeOf (PETYPEOF)

{(lastenheft,Lastenheft),(systemarchitektur,Systemarchitektur),

(anf1,Anforderung),(anf2,Anforderung),(anf3,Anforderung)}

// Kantenexemplare TypeOf (KETYPEOF)

{(lh-anf1,lh-anf),(anf1-lh,anf-lh),

(lh-anf2,lh-anf),(anf2-lh,anf-lh),

(lh-anf3,lh-anf),(anf3-lh,anf-lh)}

// Kantenexemplare From/To (KEFROMTO)

{(lh-anf1,lastenheft,anf1),(anf1-lh,anf1,lastenheft),

(lh-anf2,lastenheft,anf2),(anf2-lh,anf2,lastenheft),

(lh-anf3,lastenheft,anf3),(anf3-lh,anf3,lastenheft)})

In Abbildung 43 sind die Kanten der Einfachheit halber bidirektional dargestellt. Diese Bidirektionalität wird in der Mengenspezifikation mittels zweier unidirektionalen Kanten modelliert. Bei den Kantentypen existiert zum Beispiel jeweils eine Kante von „Lastenheft“ nach „Anforderung“ und umgekehrt.

5.2 PRODUKTKONTEXTMODELL

Produktkontexte machen es möglich, „Schnitte“ durch das Produktmodell zu ziehen, also Teilmengen zu bilden. Dies erlaubt es, fachliche Gegebenheiten (z.B. alle sicherheitsrelevanten Elemente, wie zum Beispiel Anforderungen, Schnittstellen und Implementierungen einer Komponente) zu bündeln. Hierdurch ist eine Unterstützung des Projektmitarbeiters, der z.B. ein bestimmtes Elementexemplar bearbeitet, möglich, indem ein Werkzeug auf die relevanten (im Kontext enthaltenen) Elemente fokussiert. Des Weiteren werden die Produktkontexte genutzt, um kontextspezifisches Vorgehen (also Strukturen in der Ausführungsreihenfolge von Arbeitsschritten) zu Erlernen und vorzuschlagen. Das Verfahren hierzu wird im Kapitel 6 auf Seite 99 ausführlich beschrieben. Ein Produktkontext bezieht sich immer auf ein Produkt als „Wurzelelement“ und kann weitere Elemente aufnehmen.

Das Produktkontextmodell ist in Abbildung 44 in UML-Notation dargestellt. Auch hier wird, analog zu dem Produktmodell, beschrieben in Kapitel 5.1, zwischen Typebene und Objektebene unterschieden. Ein Produktkontexttyp besitzt zwei Referenzen auf den Produkttyp des Produktmodells: „rootEl“ beschreibt das Wurzelelement, „otherEl“ die weiteren Elemente, die ein Produktkontext enthalten kann (siehe hierzu die Konzeption der Produktkontexte in Kapitel 4.4.3 auf Seite 49 und die Einleitung dieses Kapitels ab Seite 55). Dementsprechend besitzt das Produktkontextexemplar ebenfalls zwei Referenzen auf das Produktexemplar des Produktmodells.

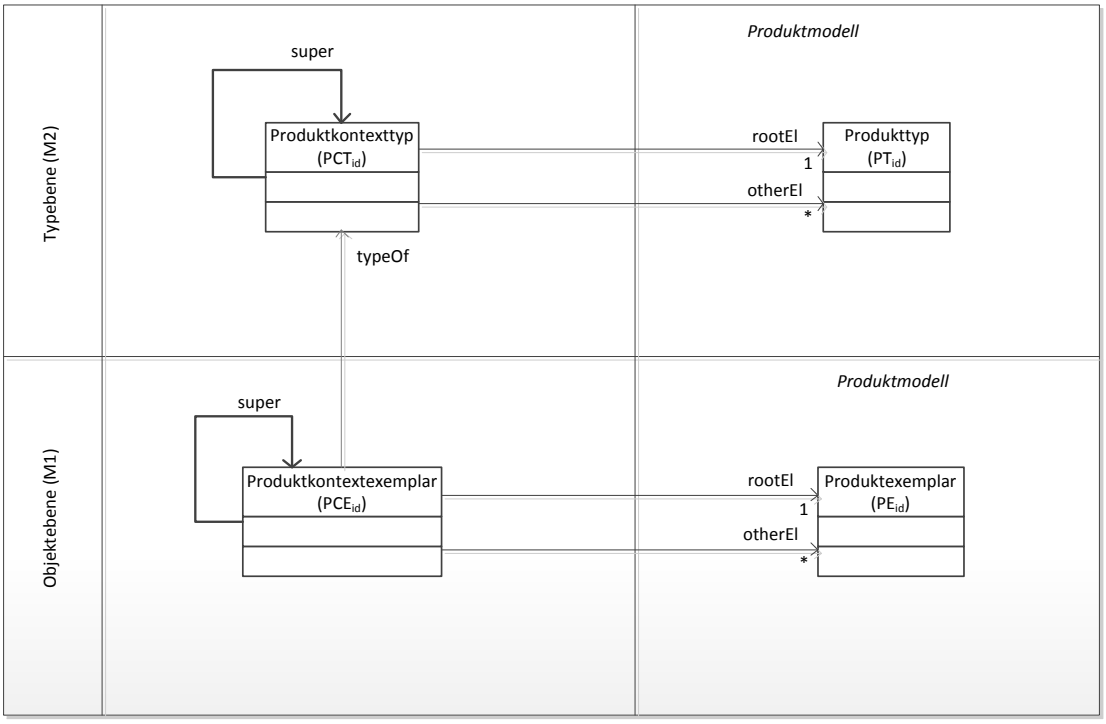


Abbildung 44: Produktkontextmodell in UML-Notation

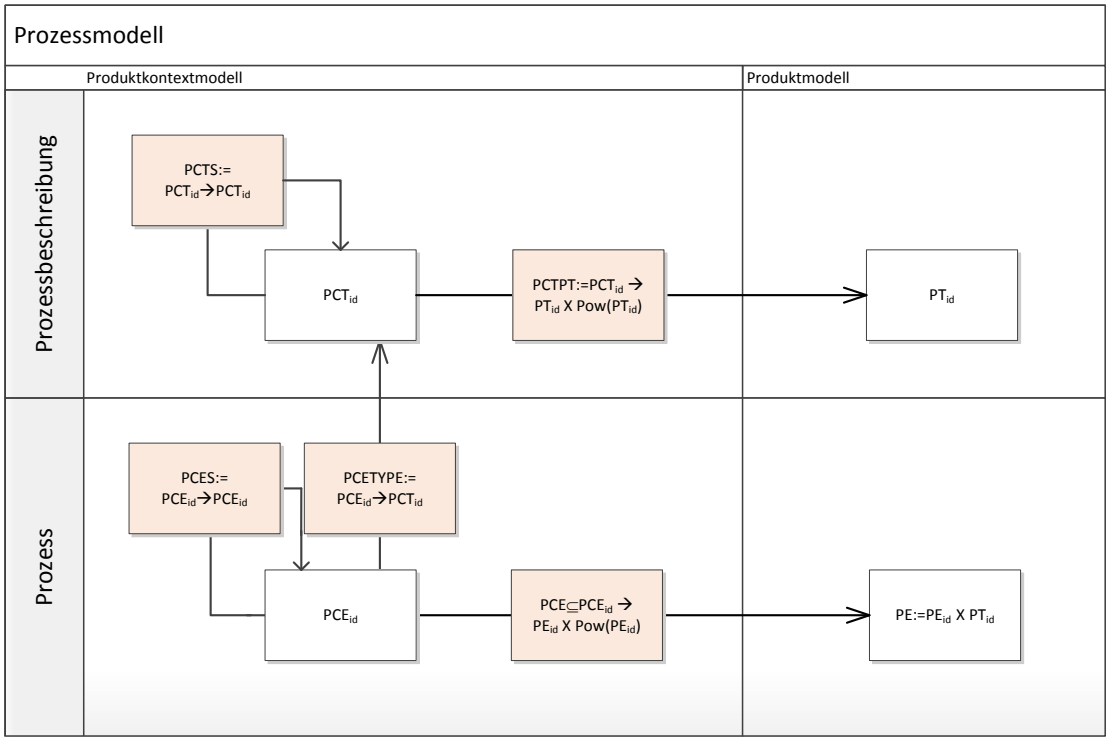


Abbildung 45: Produktkontextmodell in algebraischer Repräsentation

Das Produktkontextmodell ist in Abbildung 45 in algebraischer Form dargestellt. Im Folgenden wird das Produktkontextmodell detailliert spezifiziert.

Wir beginnen mit der Beschreibung der Elemente des Produkkontextmodells, dargestellt in Abbildung 44. Die referenzierten Elemente des Produktmodells sind in der Abbildung ebenfalls dargestellt. Die Referenzen zwischen Produkttyp und Produktexemplar sind der Einfachheit halber in der Abbildung nicht aufgeführt.

Sei PCT_{id} die Menge der möglichen Produktkontexttypen. *(13)*

Sei PCE_{id} die Menge der möglichen Produktkontextexemplare. *(14)*

Nun beschreiben wir die Referenzen des Produktkontextmodells in Form von Funktionen. Die folgende Funktion beschreibt die möglichen Referenzen von Produktkontexttypen zu den Produkttypen:

Sei $PCTPT :=_{DEF} PCT_{id} \rightarrow PT_{id} \times Pow(PT_{id})$. *(15)*

Sei $p \in PCT_{id}$ ein beliebiger Produktkontexttyp. Dann beschreibt $rootEl_p$ den definierten Produkttyp PT_{id} und $otherEl_p$ die zuvor beschriebene Potenzmenge $Pow(PT_{id})$.

Im Produktkontextmodell ist es möglich, eine Komposition (bzw. Vererbung) von Produktkontexten zu beschreiben. Dies wird mit der Referenz $super$ realisiert. Motiviert wird diese Komposition, indem hierdurch kontextspezifisches Vorgehen besser erlernt werden kann. Zum besseren Verständnis sei hier ein kurzes Beispiel erwähnt: In einem Projekt besteht das zu entwickelnde System u.a. aus der Komponente „GUI“. Diese enthält feingranulare Komponenten. Alle diese Komponenten werden mit dem gleichen Vorgehen entwickelt. Des Weiteren gibt es zu jeder der genannten Komponenten einen eigenen Kontext. Die Kontexte der

feingranularen Komponenten besitzen jeweils eine o.g. $super$-Referenz auf die GUI-Komponente. Hierdurch wird ein fachlicher Zusammenhang zwischen den Komponenten modelliert, der dann bei einer Nutzerunterstützung in Form von Arbeitsschritt-Vorschlägen (motiviert in Kapitel 4.2 und spezifiziert in Kapitel 6) genutzt werden kann.

Nun definieren wir eine Funktion zur Beschreibung dieser Referenz auf der Typebene:

Sei $PCTS \coloneqq_{DEF} PCT_{id} \rightarrow PCT_{id}$. *(16)*

Sei $pct \in PCT_{id}$ ein beliebiger Produktkontexttyp. Dann beschreibt $super_{pct}$ den Super-Typ des Produktkontexttyps pct.

Bisher wurden nur die Referenzen der Typebene des Produktkontextmodells beschrieben. Im Folgenden werden die Referenzen der Produktkontextexemplare definiert. Zuerst beschreiben wir die Exemplar-Typ-Zuordnung durch die Funktion $PCETYPE$:

Sei $PCETYPE \coloneqq_{DEF} PCE_{id} \rightarrow PCT_{id}$ die Funktion, die die Zuordnung der Produktkontextexemplare zu den Produktkontexttypen beschreibt. *(17)*

Sei $pce \in PCE_{id}$ ein beliebiges Produktkontextexemplar. Dann beschreibt $type_{pce}$ den Produktkontexttyp des Produktkontextexemplars pce.

Im Folgenden definieren wir die Referenzen der Produktkontextexemplare zu den Produktexemplaren:

Sei $PCEPE \coloneqq_{DEF} PCE_{id} \rightarrow PE_{id} \times Pow(PE_{id})$. *(18)*

Sei $pce \in PCE_{id}$ ein beliebiges Produktkontextexemplar. Dann beschreibt $rootEl_{pce}$ das definierte Produktexemplar und $otherEl_{pce}$ die beschriebene Potenzmenge $Pow(PE_{id})$.

Für alle $pce \in PCE_{id}$ muss Folgendes gelten:

- $type_{rootEl_{pce}} = rootEl_{type_{pce}}$
- $\forall oe \in otherEl_{pce}: \exists z \in otherEl_{type_{pce}}$ mit $z = type_{oe}$

Entsprechend dem Typmodell des Produktkontextmodells ist es auf der Objektebene auch möglich, Kompositionen von Produktkontextexemplaren zu beschreiben. Die folgende Funktion $PCES$ dient der Beschreibung dieses „Super-Exemplars“:

Sei $PCES \coloneqq_{DEF} PCE_{id} \rightarrow PCE_{id}$. *(19)*

Sei $pce \in PCE_{id}$ ein beliebiges Produktkontextexemplar. Dann beschreibt $super_{cep}$ das Super-Exemplar des Produktkontextexemplars pce.

Für alle $pce \in PCE_{id}$ muss Folgendes gelten:

- $type_{super_{pce}} = super_{type_{pce}}$

Ein synthetisches Beispiel eines Produktkontextmodells ist in Abbildung 46 dargestellt. Entsprechend der Abbildung 45 sind die Produktkontexttypen oben links, die Produktkontextexemplare unten links dargestellt. Die Notation der Elemente des Modells entspricht der des Produktmodells (siehe hierzu die Beschreibung auf Seite 60). Die Typen und Exemplare des Produktmodells sind entsprechend rechts zu finden. Die Elemente des Produktmodells orientieren sich an dem Beispiel des Produktmodells (siehe Abbildung 40 und Beschreibung auf Seite 60). Zur Veranschaulichung ist dieses Beispiel in Abbildung 47 in UML-Notation dargestellt. Die Referenz $super$ zwischen PCT_2 und PCT_1 (siehe Abbildung 47) ist in

Abbildung 46 beispielsweise modelliert als (PCT_2, PCT_1) und ist ein Element der Menge $PCTS$ (siehe (16)). Die Typisierung des Elements PCE_1 wird beschrieben durch (PCE_1, PCT_1) und ist Element der Menge $PCETYPE$ (siehe (17)).

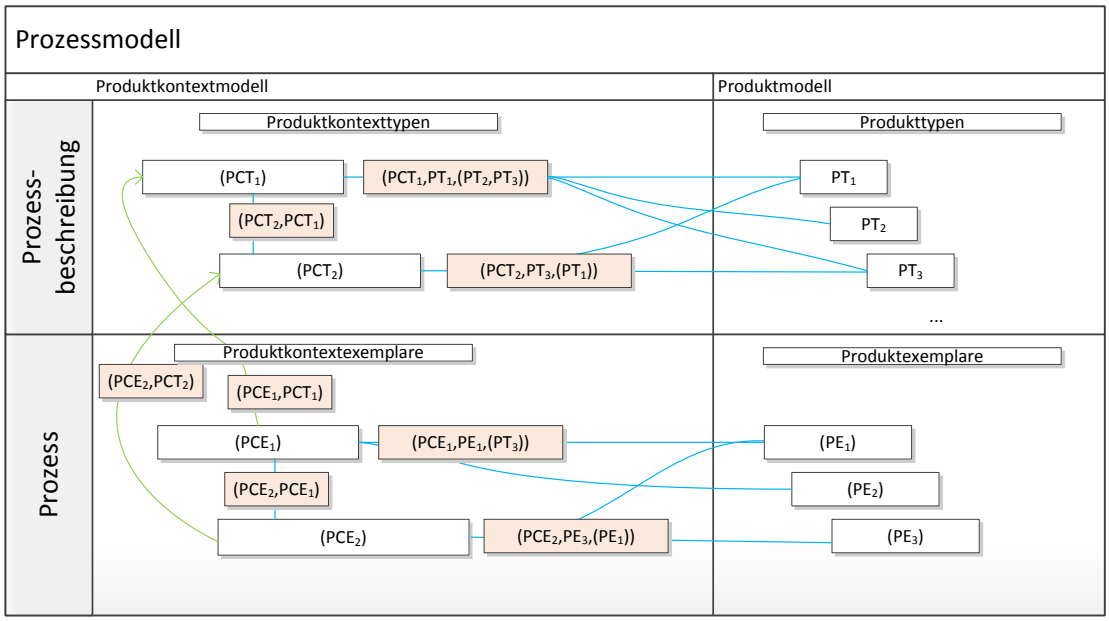


Abbildung 46: Beispiel eines Produktkontextmodells

Nun definieren wir einige Klassifizierungen, die im Verlauf der Arbeit verwendet werden.

Die folgende Klassifizierung liefert zu einem Produktkontexttyp pct die Menge der passenden Produktkontextexemplare:

$$getPCE_{PCE-pct} :=_{DEF} \{Pce \subseteq PCE | \forall pce \in PCE \text{ mit } type_{pce} = pct \Rightarrow pce \in Pce; \text{ sonst: } pce \notin Pce\} \quad (20)$$

Des Weiteren benötigen wir eine Klassifizierung, die uns zu einem gegebenen Produktexemplar pe und einem gegebenen Produktkontexttyp pct das dazugehörige Produktkontextexemplar liefert:

$$getPCE_{PCE-pe-pct} :=_{DEF} \{\forall pce \in PCE | rootEl_{pce} = pe \text{ und } type_{pce} = pct\} \quad (21)$$

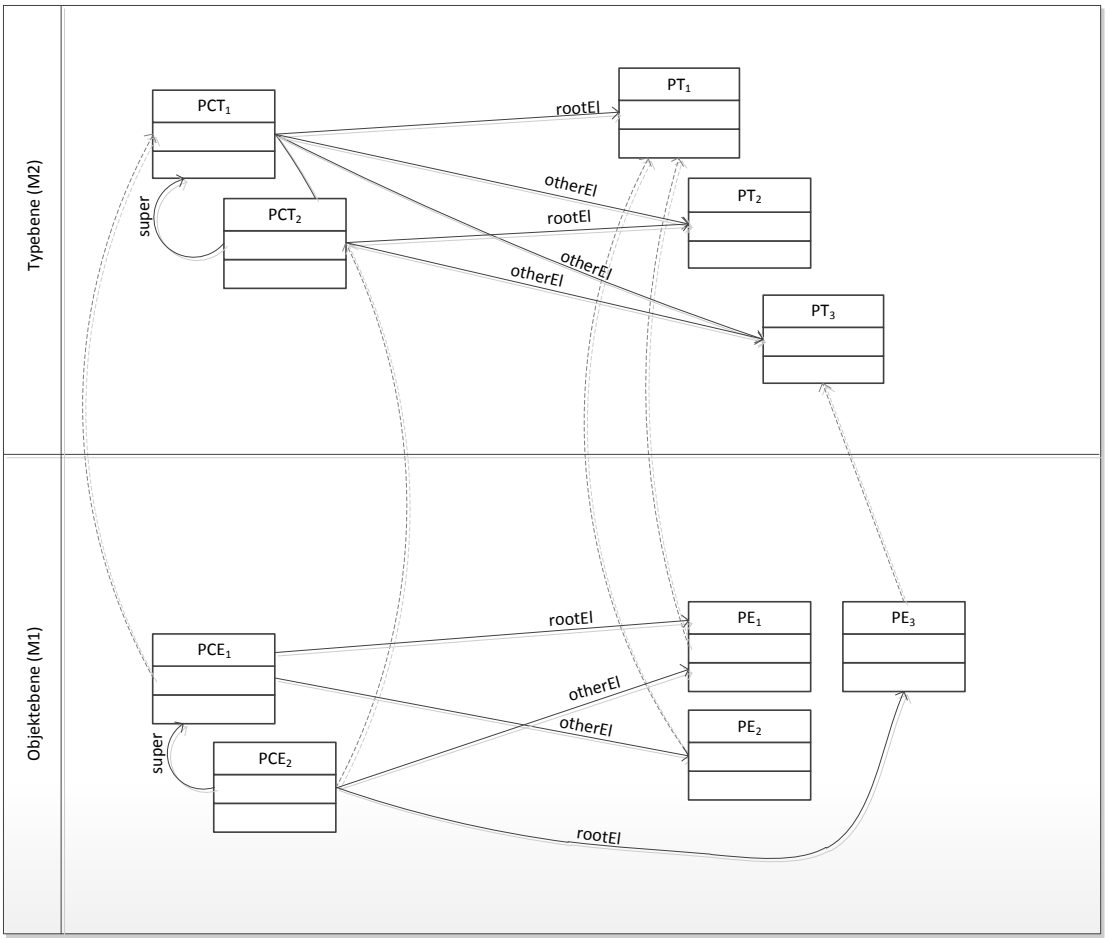


Abbildung 47: Beispiel eines Produktkontextmodells in UML-Notation

Bis jetzt wurden die Bereiche der Produktkontexttypen und Produktkontextexemplare weitgehend isoliert voneinander betrachtet. Um ein konformes Produktkontextmodell zu erhalten, gibt es jedoch Einschränkungen, die für jedes vollständige Produktkontextmodell gelten müssen.

Die Menge aller möglichen, konsistenten Produktkontextmodelle, bestehend aus der Menge der Produktkontexttypen und Produktkontextexemplare, ist dann im Folgenden definiert.

Die Menge aller möglichen Produktkontexte zu einem gegebenen Produktmodell $\boxed{PM}$ wird beschrieben durch:

$$\boxed{PC}_{\boxed{PM}} :=_{DEF} \{(\boxed{PCT_{PC}} \subseteq PCT_{id}, \boxed{PCE_{PC}} \subseteq PCE_{id}, \boxed{PCTPT_{PC}} \subseteq PCTPT, \boxed{PCTS_{PC}} \subseteq PCTS, \boxed{PCETYPEOF_{PC}} \subseteq PCETYPEOF, \boxed{PCEPE_{PC}} \subseteq PCEPE, \boxed{PCES_{PC}} \subseteq PCES)\} \text{ mit:} \quad (22)$$

- Zu jedem Produktkontextexemplar gibt es genau einen Typ:
  $\forall pce \in \boxed{PCE_{PC}} : \exists! pct \in \boxed{PCT_{PC}}$ mit $type_{pce} = pct$

- Für jeden Produkttyp, der von einem Produktkontexttyp referenziert wird ($rootEl$ oder $otherEl$), gibt es genau einen Produkttyp im Produktmodell:

$$\forall pct \in \boxed{PCT_{PC}} : \exists! pt \in \boxed{PT_{PM}} \text{ mit } pt = rootEl_{pct} \text{ und}$$
$$\forall u \in otherEl_{pct} : \exists! u_{\boxed{PT}} \in \boxed{PT_{PM}} \text{ mit } u = u_{\boxed{PT}}$$

- Für jeden Produktkontexttyp gibt es genauso viele Produktkontextexemplare wie Produktexemplare vom (im Produktkontexttyp mit rootEl) spezifizierten Produkttyp:

  $$\forall pct \in \boxed{PCT_{PC}} : \text{Sei } L = getPE_{\boxed{PE}-rootEl_{pct}} \text{ und}$$
  $$M = getPCE_{\boxed{PCE_{PC}}-pct} \Rightarrow |L| = |M|.$$
  Und: $\forall l \in L \exists! m \in M$ mit $l = rootEl_m$

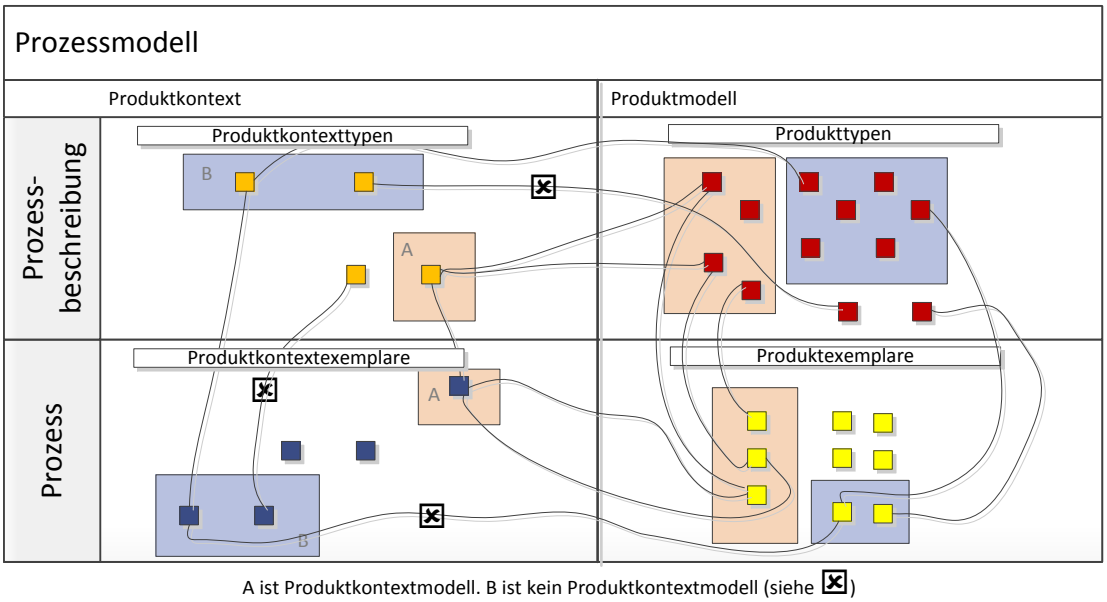


Abbildung 48: A: Beispiel eines Produktkontextmodells; B: nicht-konformes Modell

Abbildung 48 zeigt beispielhaft Produktkontextelemente (Produktkontexttypen, Produktkontextexemplare) sowie Produktelemente (Produkttypen, Produktexemplare). Das Produktmodell orientiert sich an dem in Abbildung 42 vorgestellten Produktmodell. Weiterhin sind in dieser Abbildung Mengen dargestellt, die ein konsistentes Produktkontextmodell beinhalten (A), sowie Mengen, deren Elemente nicht konsistent zueinander sind (B). B ist beispielsweise kein Produktkontextmodell, weil ein Produktkontextexemplar auf einen Produktkontexttyp referenziert, welches nicht selbst Teil dieses Modells ist. Weiterhin referenziert ein Produktkontexttyp einen Produkttyp, der im entsprechenden Produktmodell nicht vorhanden ist.

In Abbildung 49 ist das Produktkontextmodell der Beispielmethodik, beschrieben in Kapitel 3 und passend zum Produktmodell auf Seite 64 (Abbildung 43), dargestellt. Im oberen linken Bereich der Abbildung sind die Produktkontexttypen abgebildet, unten links sind die Produktkontextexemplare beschrieben. Auf der rechten Seite befindet sich ein Ausschnitt des Produktmodells, das bereits im Kapitel 5.1 vorgestellt wurde.

Auf der Ebene der Produktkontextypen existiert der System-Kontext „SystemC“ als übergeordneter Kontext. Die Systemarchitektur ist das Wurzelelement, da es genau ein Objekt dieses Typs in einem Projekt gibt. Der Systemkontext kann alle weiteren dargestellten Produkttypen enthalten. Des Weiteren existieren für Anforderungen, Komponenten und

Spezifikationen jeweils Kontexte, d.h. für jedes dieser Elemente existiert ein Produktkontext, dessen Referenz $rootEl$ auf den jeweiligen Produkttyp referenziert. Auf der Ebene der Produktkontextexemplare existieren ein Systemkontext (da es genau ein Objekt „systemarchitektur“ gibt) sowie drei Anforderungskontexte entsprechend den drei existenten Objekten vom Typ „Anforderung“.

Im Folgenden ist dieses Beispiel in der zuvor spezifizierten Mengendarstellung beschrieben:

$\boxed{pc_a}_{\boxed{pm_a}}$ = (

//Produktkontexttypen (PCT)

{SystemC,AnfC,KompC,SpezC},

// Produktkontextexemplare (PCE)

{systemc,anf1c,anf2c,anf3c},

// Produkttypen der Produktkontexttypen (PCTPT)

{(SystemC,Systemarchitektur,(Anforderung,Komponente,IF,Spezifikation,IFD,IFI)),

(AnfC,Anforderung,(∅)),

(KompC,Komponente,(Anforderung,IF)),

(SpezC,Spezifikation,(IFD,IFI,IF))},

// Super-Typen der Produktkontexttypen (PCTS)

{(SystemC,∅),

(AnfC,SystemC),

(KompC,SystemC),

(SpezC,KompC)},

// Typen der Produktexemplare (PCETYPEOF)

{(systemc,SystemC),

(anf1c,AnfC),(anf2c,AnfC),(anf3c,AnfC)},

// Produktexemplare der Produktkontextexemplare (PCEPE)

{(systemc,systemarchitektur,(anf1,anf2,anf3)),

(anf1c,anf1,∅),(anf2c,anf2,∅),(anf3c,anf3,∅)},

// „Super-Exemplare“ der Produktkontextexemplare (PCES)

{(systemc,∅),

(anf1c,systemc),(anf2c,systemc),(anf3c,systemc)},

)

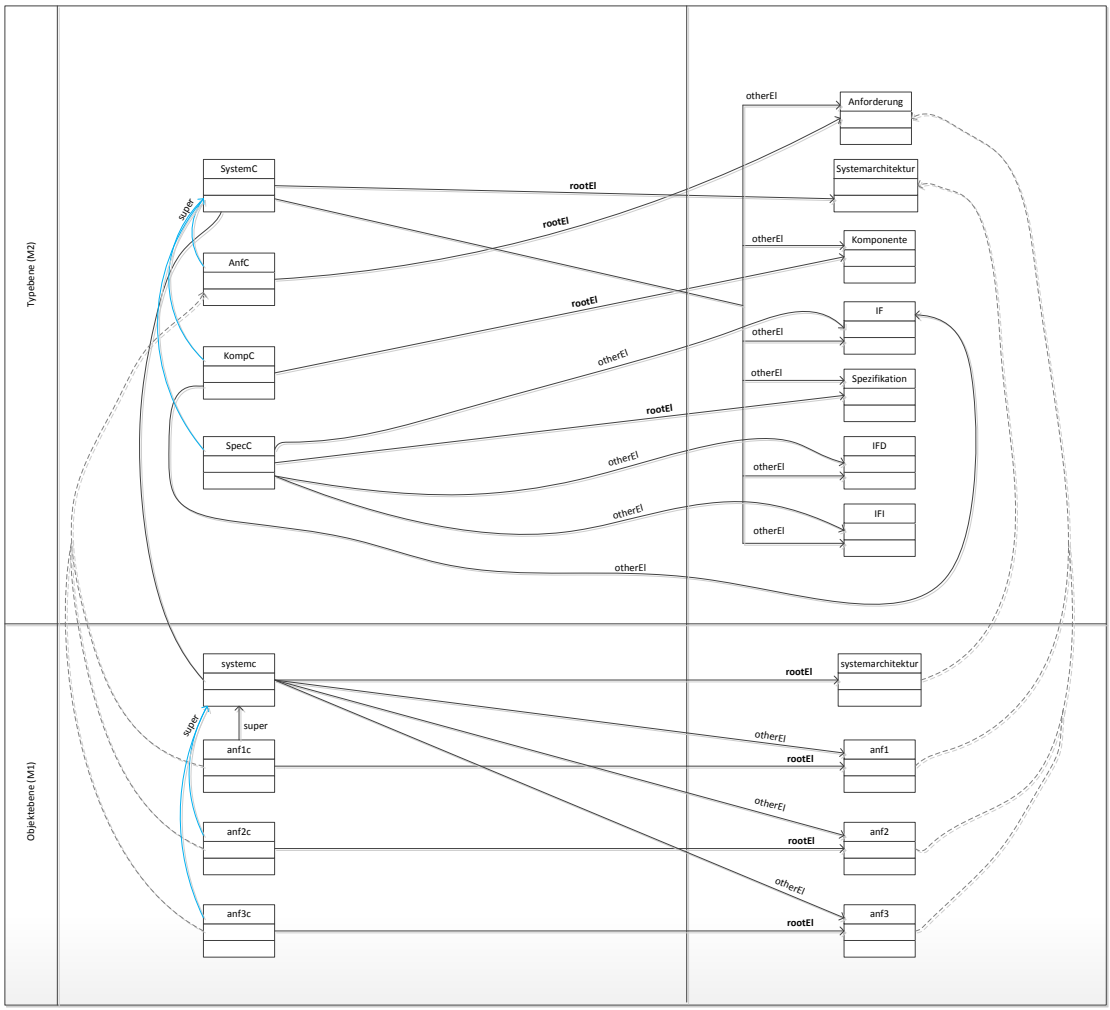


Abbildung 49: Produktkontextmodell der Beispielmethodik in UML-Notation

5.3 AKTIONENMODELL

Im Aktionenmodell werden Änderungen auf dem Produktmodell sowie dem Produktkontextmodell beschrieben. Diese Änderungen werden „Aktionen“ genannt und setzen sich aus atomaren Operationen auf den Modellen zusammen. Aktionen haben fachlichen Charakter; denkbar ist z.B. eine Aktion „Anforderung hinzufügen“. Diese setzt sich dann aus mehreren Operationen auf den Modellen zusammen, zum Beispiel dem Hinzufügen eines Produktexemlars vom Typ „Anforderung“ sowie dem Hinzufügen von Kantenexemplaren/Referenzen zur Verschaltung des Exemplars mit anderen Elementen des Produktmodells.

Zur Beschreibung der Aktionen werden als Erstes Variablen eingeführt. Diese werden in einer Prozessbeschreibung definiert und dann zur Prozesslaufzeit an konkrete Objekte gebunden. Die konkrete Variablenbelegung geschieht dann während der Laufzeit über die Steps – diese werden in Kapitel 5.4 beschrieben. Danach werden die (atomaren) Operationen auf dem Produktmodell und Produktkontextmodell spezifiziert. Vorgestellt werden dabei vier Operationen auf dem Produktmodell: Hinzufügen und Entfernen von Produktexemplaren sowie Hinzufügen und Entfernen von Kantenexemplaren. Auf dem Produktkontextmodell werden zwei Operationen vorgestellt, nämlich das Hinzufügen und Entfernen von Produktexemplaren aus einem Produktkontextexemplar.

5.3.1 VARIABLEN.

In Abbildung 51 ist das Variablenkonzept in UML-Notation dargestellt. Hierbei wird unterschieden zwischen Produkt-, Kanten- und Produktkontextvariablen. Eine Produktvariable referenziert einen Produkttyp und ist ein Platzhalter für ein Produktexemplar. Kantenvariablen und Produktkontextvariablen sind entsprechend aufgebaut und referenzieren Kantentypen bzw. Produktkontexttypen. Während der Prozesslaufzeit, also wenn Produkt-, Kanten- und Produktkontextexemplare im Prozessmodell existieren, können Variablen über Variablenbelegungen an diese Exemplare gebunden werden. Ist dies erfüllt, so können die in der Prozessbeschreibung spezifizierten Operationen auf dem Produktmodell und Produktkontextmodell ausgeführt werden. Ein Beispiel mit Variablen und deren Belegung ist in Abbildung 50 dargestellt. Hier sind drei Variablen „anfvar1" bis „anfvar3" als Platzhalter für drei Anforderungen spezifiziert (in der Abbildung oben: Typebene). Diese Variablen sind zur Laufzeit folgendermaßen belegt: „anfvar1" referenziert die Anforderung „anf2" (ein Produktexemplar von Produkttyp „Anforderung"), die Variablen „anfvar2" und „anfvar3" referenzieren beide die Anforderung „anf3".

Nun folgt die algebraische Spezifikation der Variablen. Das Variablenkonzept ist ebenfalls in Abbildung 52 dargestellt:

Sei PV_{id} die Menge der möglichen Produktvariablen. *(23)*

Sei KV_{id} die Menge der möglichen Kantenvariablen *(24)*

Sei PCV_{id} die Menge der möglichen Produktkontextvariablen. *(25)*

Die o.g. Mengen beschreiben bisher nur die möglichen Variablen-IDs. In unserem Ansatz werden die Variablen getypt. Das bedeutet, dass beispielsweise eine Produktvariable einen Produkttyp referenziert. Diese Referenz hat Auswirkungen auf die spätere Belegung der Variablen: Eine Produktvariable kann nur mit einem Produktexemplar belegt werden, dessen Typ dem Produkttyp der Variablen entspricht. Bei den Kantenvariablen und Produktkontextvariablen wird dies analog gehandhabt.

Diese Typisierung der Variablen wird mit den folgenden drei Funktionen beschrieben:

Sei $PVPT :=_{DEF} PV_{id} \rightarrow PT_{id}$ die Typisierung der Produktvariablen. *(26)*

Sei $pv \in PV_{id}$ eine beliebige Produktvariable. Dann beschreibt $type_{pv} = PVPT(pv)$

Sei $KVKT :=_{DEF} KV_{id} \rightarrow KT_{id}$ die Typisierung der Kantenvariablen *(27)*

Sei $kv \in KV_{id}$ eine beliebige Kantenvariable. Dann beschreibt $type_{kv} = KVKT(kv)$

Sei $PCVPCT :=_{DEF} PCV_{id} \rightarrow PCT_{id}$ die Typisierung der Produktkontextvariablen *(28)*

Sei $pcv \in PCV_{id}$ eine beliebige Produktkontextvariable. Dann beschreibt $type_{pcv} = PCVPCT(pcv)$

Für jeden der drei oben genannten Variablentypen (siehe (23) bis (25)) gibt es eine Menge an möglichen Variablenbelegungen. *Zur Laufzeit* hat jede Variable genau eine Belegung.

Sei PVB_{id} die Menge der möglichen Produktvariablenbelegungen. *(29)*

Sei KVB_{id} die Menge der möglichen Kantenvariablenbelegungen. *(30)*

Sei $PCVB_{id}$ die Menge der möglichen Produktkontextvariablenbelegungen. *(31)*

Die unter (29) bis (31) dargestellten Mengen beschreiben lediglich die IDs der Variablenbelegungen. Die folgenden drei injektiven Funktionen stellen die Zuordnungen Variablenbelegungen – Variablen her (zu einer Variablenbelegung gibt es zur Laufzeit maximal eine Variable, deshalb injektiv):

Sei $PVBPV :=_{DEF} PVB_{id} \rightarrow PV_{id}$ die Zuordnung Produktvariablenbelegung – Produktvariable. *(32)*

Sei $pvb \in PVB_{id}$ eine beliebige Produktvariablenbelegung. Dann beschreibt $var_{pvb} = PVBPV(pvb)$.

Sei $pv \in PV_{id}$ eine beliebige Produktvariable. Dann beschreibt $vb_{pv} = PVBPV^{-1}(pv)$ die Umkehrfunktion von $PVBPV$.

Sei K$VBKV :=_{DEF} KVB_{id} \rightarrow KV_{id}$ die Zuordnung Kantenvariablenbelegung – Kantenvariable. *(33)*

Sei $kvb \in KVB_{id}$ eine beliebige Kontextvariablenbelegung. Dann beschreibt $var_{kvb} = KVBKV(kvb)$.

Sei $kv \in KV_{id}$ eine beliebige Kantenvariable. Dann beschreibt $vb_{kv} = KVBKV^{-1}(kv)$ die Umkehrfunktion von $KVBKV$.

Sei $PCVBPCV :=_{DEF} PCVB_{id} \rightarrow PCV_{id}$ die Zuordnung Produktkontextvariablenbelegung – Produktkontextvariable. *(34)*

Sei $pcvb \in PCVB_{id}$ eine beliebige Produktkontextvariablenbelegung. Dann beschreibt $var_{cvbp} = PCVBPCV(pcvb)$.

Sei $pcv \in PCV_{id}$ eine beliebige Produktkontextvariable. Dann beschreibt $vb_{pcv} = PCVBPCV^{-1}(pcv)$ die Umkehrfunktion von $PCVBPCV$.

Bisher wurden die Variablen-IDs sowie die Variablenbelgungs-IDs und die Zuordnungen dieser beiden Mengen beschrieben. Zur Laufzeit wird eine Variablenbelegung an ein konkretes Exemplar im Objektmodell gebunden. Damit ist zur Laufzeit definiert, welches Exemplar eine Variable repräsentiert.

Die tatsächliche Belegung der Variablen zur Laufzeit wird über die folgenden drei Funktionen beschrieben:

Sei $PVBPE :=_{DEF} PVB_{id} \rightarrow PE_{id}$. *(35)*

Zur einfachen Handhabung bedeutet $ex_{pvb} = PVBPE(pvb)$.
Dabei muss der Produkttyp des Produktexemplars (PE_{id}) der Typisierung der Produktvariablen entsprechen: $\forall pvb \in PVB_{id}: type_{ex_{pvb}} = type_{var_{pvb}}$:

Sei $KVBKE :=_{DEF} KVB_{id} \rightarrow KE_{id}$ *(36)*

Zur einfachen Handhabung bedeutet $ex_{kvb} = KVBKE(kvb)$.
Dabei muss der Kontexttyp des Kontextexemplars (KE_{id}) der Typisierung der

Kontextvariablen entsprechen: $\forall kvb \in KVB_{id}: type_{ex_{kvb}} = type_{var_{kvb}}$

Sei $PCVBPCE :=_{DEF} PCVB_{id} \rightarrow PCE_{id}$ *(37)*

Zur einfachen Handhabung bedeutet $ex_{pcvb} = PCVBPCE(pcvb)$.
Dabei muss der Produktkontexttyp des Produktkontextexemplars (PCE_{id}) der Typisierung der Produktkontextvariablen entsprechen: $\forall pcvb \in PCVB_{id}: type_{ex_{pcvb}} = type_{var_{pcvb}}$.

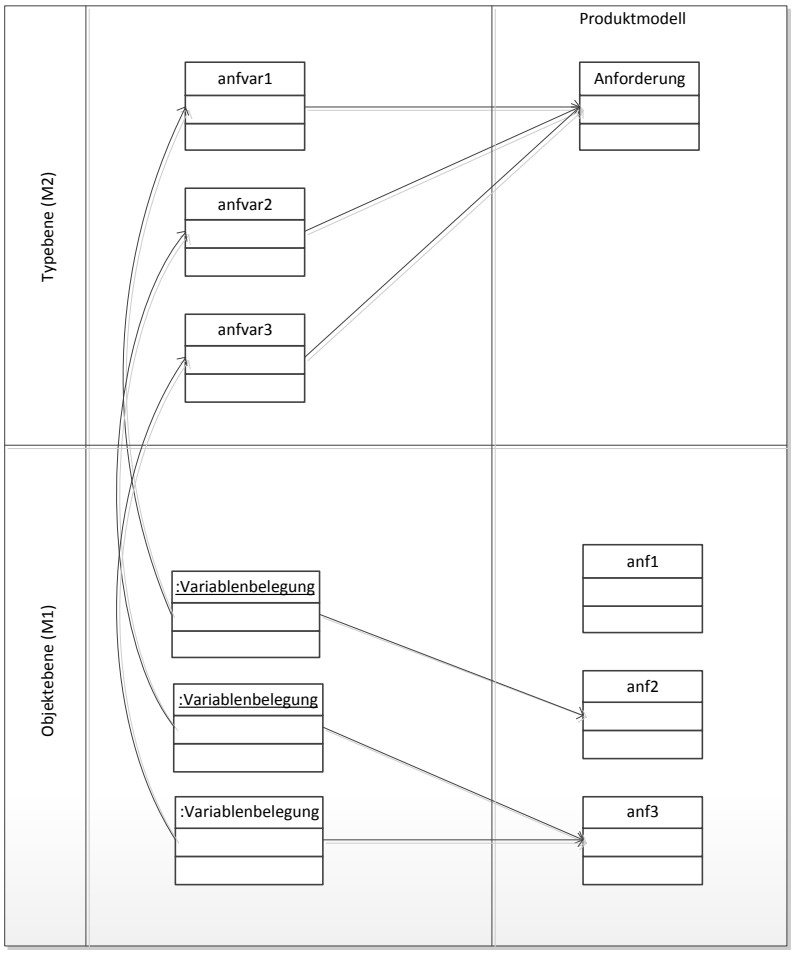


Abbildung 50: Beispiel Variablen in UML-Notation

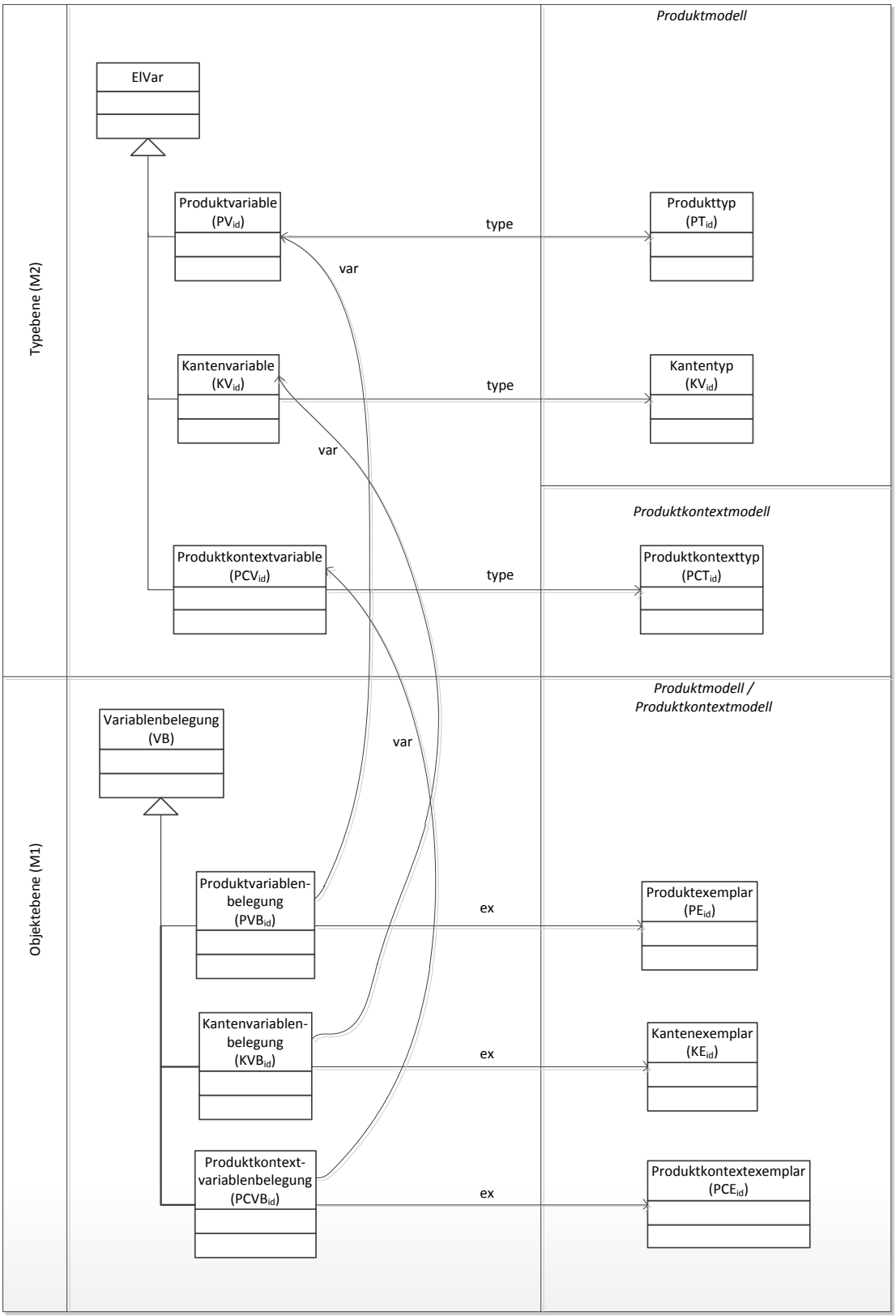


Abbildung 51: Variablenkonzept in UML-Notation

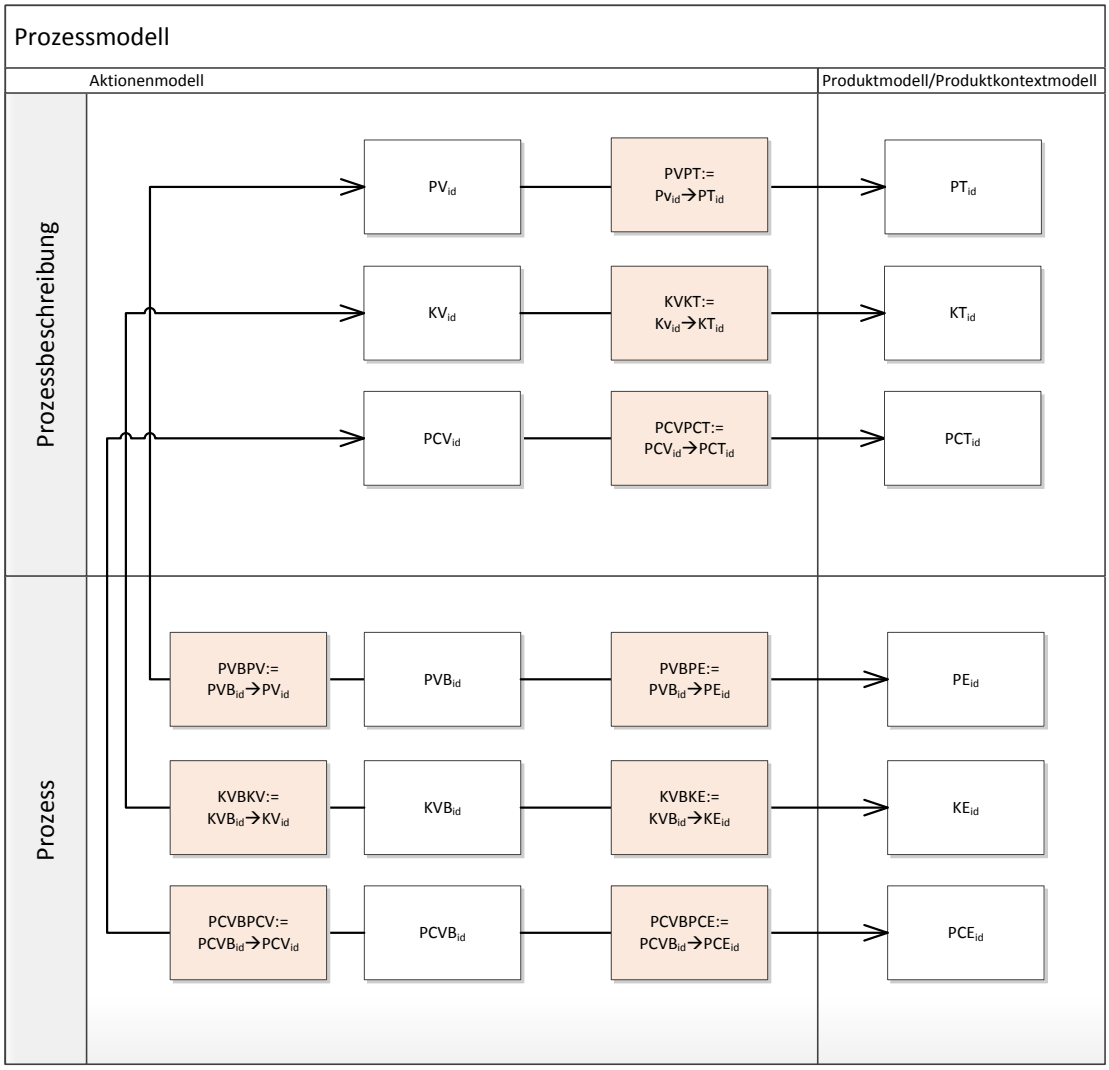


Abbildung 52: Variablenkonzept in algebraischer Repräsentation

5.3.2 OPERATIONEN

Nach der Einführung der Variablen im letzten Abschnitt können nun Operationen, also die atomaren Operationen auf dem Produkt- und Produktkontextmodell, definiert werden. Die Operationen sind in Abbildung 53 in UML-Notation dargestellt. Als Produktoperationen gibt es die Möglichkeit, mit $addProduct$ ein Produktexemplar dem Produktmodell hinzuzufügen. Auf der anderen Seite löscht $delProduct$ ein Produktexemplar aus dem Produktmodell. Des Weiteren gibt es mit $addEdge$ und $delEdge$ Möglichkeiten, um Kantenexemplare zwischen zwei Produktexemplaren hinzuzufügen und zu entfernen. Bei den Produktkontextoperationen fügt $addPtoC$ das über die Produktvariable referenzierte Produktexemplar dem referenzierten Produktkontextexemplar hinzu. Auf der anderen Seite entfernt $delPfromC$ das referenzierte Produktexemplar aus dem Produktkontextexemplar.

Zur einfachen Handhabung definieren wir folgende Mengen:

$$pop :=_{DEF} \{addProduct, delProduct, addEdge, delEdge\} \quad (38)$$

$$cop :=_{DEF} \{addPtoC, delPfromC\} \quad (39)$$

Mit Hilfe dieser zwei Mengen werden die Operationen auf dem Produktmodell und Produktkontextmodell beschrieben.

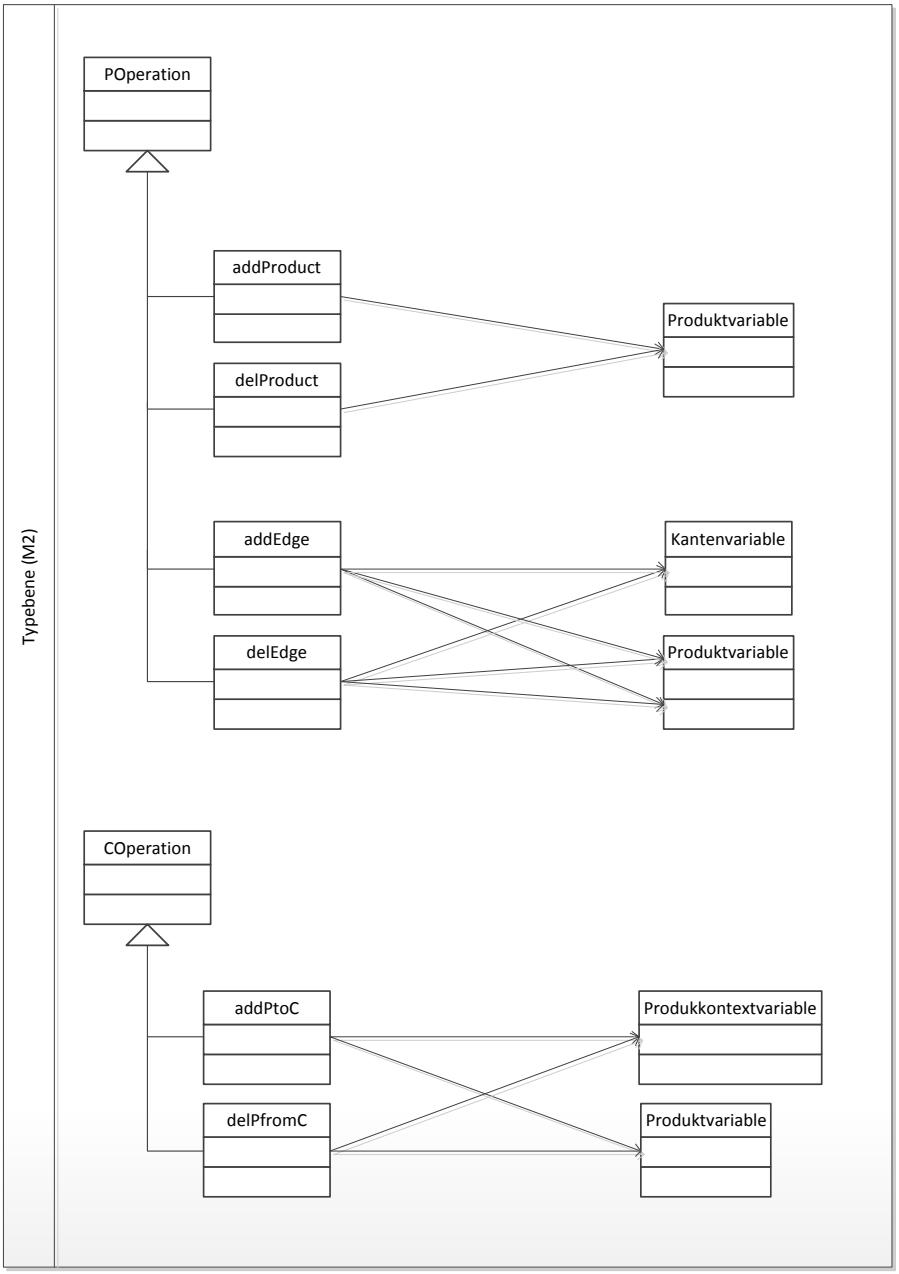


Abbildung 53: Mögliche Operationen auf dem Produkt- und Produktkontextmodell

Im Folgenden werden die zuvor beschriebenen Operationen auf dem Produktmodell detailliert beschrieben. Hierbei wird grundsätzlich zwischen dem Hinzufügen ($addProduct$) und Löschen ($delProduct$) von Produkten und dem Hinzufügen ($addEdge$) und Löschen ($delEdge$) von Kanten unterschieden, weil sich bei diesen zwei Klassen von Operationen grundsätzlich der Definitionsbereich der Funktionsköpfe unterscheidet. Für die Unterscheidung der einzelnen Operationstypen wird die Menge pop (siehe (38)) verwendet.

- $POperation_{pop}(\boxed{pm_o}, pvar) :=_{DEF} \boxed{pm_1}$ *(40)*
  - $pop = addProduct$:
    - $\exists el \in PE$ mit $el \notin \boxed{PE_{pm_0}}$ und $el \in \boxed{PE_{pm_1}}$
    - $type_{el} \in \boxed{PT_{pm_0}}$ und $type_{el} \in \boxed{PT_{pm_1}}$
    - $(el, type_{pvar}) \notin \boxed{PETYPEOF_{pm_0}}$ und $(el, type_{pvar}) \in \boxed{PETYPEOF_{pm_1}}$
    - $type_{el} = type_{pvar}$
    - $\boxed{PT_{pm_0}} = \boxed{PT_{pm_1}}$ und $\boxed{KT_{pm_0}} = \boxed{KT_{pm_1}}$ und
      $\boxed{KE_{pm_0}} = \boxed{KE_{pm_1}}$ und
      $\boxed{KTFROMTO_{pm_0}} = \boxed{KTFROMTO_{pm_1}}$ und
      $\boxed{PETYPEOF_{pm_0}} = \boxed{PETYPEOF_{pm_1}}$ und
      $\boxed{KETYPEOF_{pm_0}} = \boxed{KETYPEOF_{pm_1}}$ und
      $\boxed{KEFROMTO_{pm_0}} = \boxed{KEFROMTO_{pm_1}}$
    - $\left|\boxed{PE_{pm_0}}\right| + 1 = \left|\boxed{PE_{pm_1}}\right|$
    - $\boxed{PE_{pm_0}} \setminus \boxed{PE_{pm_1}} = \emptyset$
    - $\left|\boxed{PETYPEOF_{pm_0}}\right| + 1 = \left|\boxed{PETYPEOF_{pm_1}}\right|$
    - $\boxed{PETYPEOF_{pm_0}} \setminus \boxed{PETYPEOF_{pm_1}} = \emptyset$
    - *Zur Laufzeit*:
      $\exists! pvb \in PVB_{id}$ mit $var_{pvb} = pvar$ und $ex_{pvb} = el$
  - $pop = delProduct$:
    - $\exists el \in PE$ mit $el \in \boxed{PE_{pm_0}}$ und $el \notin \boxed{PE_{pm_1}}$
    - $type_{el} \in \boxed{PT_{pm_1}}$ und $type_{el} \in \boxed{PT_{pm_0}}$
    - $(el, type_{pvar}) \notin \boxed{PETYPEOF_{pm_1}}$ und $(el, type_{pvar}) \in \boxed{PETYPEOF_{pm_0}}$
    - $type_{el} = type_{pvar}$
    - $\boxed{PT_{pm_0}} = \boxed{PT_{pm_1}}$ und $\boxed{KT_{pm_0}} = \boxed{KT_{pm_1}}$ und
      $\boxed{KE_{pm_0}} = \boxed{KE_{pm_1}}$ und
      $\boxed{KTFROMTO_{pm_0}} = \boxed{KTFROMTO_{pm_1}}$ und
      $\boxed{PETYPEOF_{pm_0}} = \boxed{PETYPEOF_{pm_1}}$ und
      $\boxed{KETYPEOF_{pm_0}} = \boxed{KETYPEOF_{pm_1}}$ und

$$\boxed{KEFROMTO_{pm_0}} = \boxed{KEFROMTO_{pm_1}}$$

- $\left|\boxed{PE_{pm_0}}\right| = \left|\boxed{PE_{pm_1}}\right| + 1$
- $\boxed{PE_{pm_1}} \setminus \boxed{PE_{pm_0}} = \emptyset$
- $\left|\boxed{PETYPEOF_{pm_0}}\right| = \left|\boxed{PETYPEOF_{pm_1}}\right| + 1$
- $\boxed{PETYPEOF_{pm_1}} \setminus \boxed{PETYPEOF_{pm_0}} = \emptyset$
- $getKE_{\boxed{KE_{pm_0}}-el} \notin \boxed{KE_{pm_1}}$
- $\boxed{KE_{pm_1}} \setminus \boxed{KE_{pm_0}} = \emptyset$
- *Zur Laufzeit*:

  $\exists!\, pvb \in PVB_{id}$ mit $var_{pvb} = pvar$ und $ex_{pvb} = el$

- $POperation_{pop}(\boxed{pm_o}, edgevar, pvar1, pvar2) :=_{DEF} \boxed{pm_1}$ *(41)*
  - $pop = addEdge$:
    - $\exists el \in KE$ mit $el \notin \boxed{KE_{pm_0}}$ und $el \in \boxed{KE_{pm_1}}$
    - $\exists p1, p2 \in PE$ mit $p1, p2 \in \boxed{PE_{pm_0}}$ und $p1, p2 \in \boxed{PE_{pm_1}}$
    - $type_{el} \in \boxed{KT_{pm_0}}$ und $type_{el} \in \boxed{KT_{pm_1}}$
    - $(el, type_{egdevar}) \notin \boxed{KETYPEOF_{pm_0}}$ und $(el, type_{edgevar}) \in \boxed{KETYPEOF_{pm_1}}$
    - $type_{edgevar} = type_{el}$
    - $\boxed{PT_{pm_0}} = \boxed{PT_{pm_1}}$ und $\boxed{KT_{pm_0}} = \boxed{KT_{pm_1}}$ und

      $\boxed{PE_{pm_0}} = \boxed{PE_{pm_1}}$ und

      $\boxed{KTFROMTO_{pm_0}} = \boxed{KTFROMTO_{pm_1}}$ und

      $\boxed{PETYPEOF_{pm_0}} = \boxed{PETYPEOF_{pm_1}}$
    - $\left|\boxed{KE_{pm_0}}\right| + 1 = \left|\boxed{KE_{pm_1}}\right|$
    - $\boxed{KE_{pm_0}} \setminus \boxed{KE_{pm_1}} = \emptyset$
    - $\left|\boxed{KETYPEOF_{pm_0}}\right| + 1 = \left|\boxed{KETYPEOF_{pm_1}}\right|$
    - $\boxed{KETYPEOF_{pm_0}} \setminus \boxed{KETYPEOF_{pm_1}} = \emptyset$
    - $(el, type_{edgevar}) \notin \boxed{KETYPEOF_{pm_0}}$ und $(el, type_{edgevar}) \in \boxed{KETYPEOF_{pm_1}}$

- $\left|\boxed{KEFROMTO_{pm_0}}\right| + 1 = \left|\boxed{KEFROMTO_{pm_1}}\right|$
- $\boxed{KEFROMTO_{pm_0}} \setminus \boxed{KEFROMTO_{pm_1}} = \emptyset$
- $(el, p1, p2) \notin \boxed{KEFROMTO_{pm_0}}$ und $(el, p1, p2) \in \boxed{KEFROMTO_{pm_1}}$
- *Zur Laufzeit*:
  $\exists! pvb1 \in PVB_{id}$ und $pvb2 \in PVB_{id}$ und $kvb \in KVB_{id}$ mit $var_{kvb} = edgevar$ und $var_{pvb1} = pvar1$ und $var_{pvb2} = pvar2$ und $ex_{kvb} = el$ und $ex_{pvb1} = p1$ und $ex_{pvb2} = p2$ und $from_{el} = p1$ und $to_{el} = p2$

- $pop = delEdge$:
  - $\exists el \in KE$ mit $el \in \boxed{KE_{pm_0}}$ und $el \notin \boxed{KE_{pm_1}}$
  - $\left(el, type_{egdevar}\right) \in \boxed{KETYPEOF_{pm_0}}$ und $(el, type_{edgevar}) \notin \boxed{KETYPEOF_{pm_1}}$
  - $(el, from_{el}, to_{el}) \in \boxed{KEFROMTO_{pm_0}}$ und $(el, from_{el}, to_{el}) \notin \boxed{KEFROMTO_{pm_1}}$
  - $\left|\boxed{KETYPEOF_{pm_0}}\right| = \left|\boxed{KETYPEOF_{pm_1}}\right| + 1$
  - $\left|\boxed{KEFROMTO_{pm_0}}\right| = \left|\boxed{KEFROMTO_{pm_1}}\right| + 1$
  - $\left|\boxed{KE_{pm_0}}\right| = \left|\boxed{KE_{pm_1}}\right| + 1$
  - $\boxed{PT_{pm_0}} = \boxed{PT_{pm_1}}$ und $\boxed{KT_{pm_0}} = \boxed{KT_{pm_1}}$ und
    $\boxed{PE_{pm_0}} = \boxed{PE_{pm_1}}$ und
    $\boxed{KTFROMTO_{pm_0}} = \boxed{KTFROMTO_{pm_1}}$ und
    $\boxed{PETYPEOF_{pm_0}} = \boxed{PETYPEOF_{pm_1}}$
  - $\boxed{KE_{pm_1}} \setminus \boxed{KE_{pm_0}} = \emptyset$
  - $\boxed{KETYPEOF_{pm_1}} \setminus \boxed{KETYPEOF_{pm_0}} = \emptyset$
  - $\boxed{KEFROMTO_{pm_1}} \setminus \boxed{KEFROMTO_{pm_0}} = \emptyset$
    - *Zur Laufzeit*:
      $\exists! kvb \in KVB_{id}$ mit $var_{kvb} = edgevar$ und $ex_{kvb} = el$

Des Weiteren definieren wir Operationen auf dem Produktkontextmodell. Hierbei wird zwischen dem Hinzufügen ($addPtoC$) und Entfernen ($delPfromC$) von Produktexemplaren aus einem Kontextexemplar (Referenz $otherEl$) unterschieden. Für die Unterscheidung der einzelnen Operationstypen wird die Menge cop (siehe (39)) verwendet.

- $COperation_{cop}\left(\boxed{pc_{0_{pm}}}, cvar, elvar\right) :=_{DEF} \boxed{pc_{1_{pm}}}$ *(42)*
    - $cop = addPtoC$:
        - Sei $c \in \boxed{PCE_{pc_0}}$ und $c \in \boxed{PCE_{pc_1}}$
          (c ist ein existierendes Produktkontextexemplar.)
        - Sei $(c, r, otherOld) \in$
          $\boxed{PCEPE_{pc_0}}$ und $(c, r, otherNew) \in \boxed{PCEPE_{pc_1}}$
        - Sei $el \in \boxed{PE_{pm}}$
          (el ist ein Produktexemplar.)
        - $\boxed{PCT_{pc_0}} = \boxed{PCT_{pc_1}}$
          (Die Produktkontexttypen ändern sich nicht.)
        - $\boxed{PCE_{pc_0}} = \boxed{PCE_{pc_1}}$
          (Die Menge der Produktkontextexemplare ändert sich nicht.)
        - $\boxed{PCTPT_{pc_0}} = \boxed{PCTPT_{pc_1}}$
        - $\boxed{PCTS_{pc_0}} = \boxed{PCTS_{pc_1}}$
        - $\boxed{PCETYPEOF_{pc_0}} = \boxed{PCETYPEOF_{pc_1}}$
        - $\boxed{PCES_{pc_0}} = \boxed{PCES_{pc_1}}$
        - $otherNew = otherOld \cup \{el\}$
        - $type_{el} \in otherEl_{type_c}$.
          (Der Elementtyp von el passt zu dem spezifizierten Kontexttyp.)
        - *Zur Laufzeit*:
          $\exists! pcvb \in PCVB_{id}$ mit
          $var_{pcvb} = cvar$ und $ex_{pcvb} = c$ und
          $\exists! pvb \in PVB_{id}$ mit $var_{pvb} = elvar$ und
          $ex_{pvb} = el$
    - $cop = delPfromC$:
        - Sei $c \in \boxed{PCE_{pc_0}}$ und $c \in \boxed{PCE_{pc_1}}$
          (c ist ein existierendes Produktkontextexemplar.)
        - Sei $(c, r, otherOld) \in \boxed{PCEPE_{pc_0}}$

und $(c, r, otherNew) \in \boxed{PCEPE_{pc_1}}$

- Sei $el \in \boxed{PE_{pm}}$

  (el ist ein Produktexemplar.)
- $\boxed{PCT_{pc_0}} = \boxed{PCT_{pc_1}}$
  (Die Produktkontexttypen ändern sich nicht.)
- $\boxed{PCE_{pc_0}} = \boxed{PCE_{pc_1}}$
  (Die Menge der Produktkontextexemplare ändert sich nicht.)
- $\boxed{PCTPT_{pc_0}} = \boxed{PCTPT_{pc_1}}$
- $\boxed{PCTS_{pc_0}} = \boxed{PCTS_{pc_1}}$
- $\boxed{PCETYPEOF_{pc_0}} = \boxed{PCETYPEOF_{pc_1}}$
- $\boxed{PCES_{pc_0}} = \boxed{PCES_{pc_1}}$
- $otherNew = otherOld \backslash \{el\}$
- $type_{el} \in otherEl_{type_c}$.
  (Der Elementtyp von el passt zu dem spezifizierten Kontexttyp.)
- *Zur Laufzeit*:

  $\exists! pcvb \in PCVB_{id}$ mit $var_{pcvb} = cvar$

  und $ex_{pcvb} = c$ und

  $\exists! pvb \in PVB_{id}$ mit $var_{pvb} = elvar$
  und $ex_{pvb} = el$

5.3.3 AKTIONEN

Bisher wurden die atomaren Operationen auf dem Produkt- und Produktkontextmodell definiert. Diese werden nun „zusammengeschaltet“, was in dieser Arbeit als „Aktionen“ bezeichnet wird. Das Zusammenschalten der Operationen lehnt sich weitestgehend an Ablaufbeschreibungen, wie zum Beispiel UML Aktivitäten oder Petri Netze, an.

Das bedeutet, eine Aktion besteht aus einer Menge von Operationen (siehe (40) bis (42)), die über gerichtete Kanten und Kontrollstrukturen verbunden sind. Als Kontrollstrukturen können einerseits explizite Start- und Endpunkte verstanden werden. Auf der anderen Seite sind hiermit Verzweigungselemente gemeint. Dies können entweder Entscheidungselemente ($branch$, $merge$), oder Parallelisierungselemente ($split$, $join$) sein. Bei den Entscheidungselementen ($branch$) wird eine der möglichen direkt nachfolgenden Operationen ausgeführt. Entscheidungen können dann über die $merge$-Elemente wieder zusammengeführt werden. Bei den Parallelisierungselementen ($split$) werden alle direkt nachfolgenden Operationen parallel gestartet. Parallele Abläufe werden dann über $join$-Elemente wieder synchronisiert. Die UML [89] beschäftigt sich z.B. mit solchen Ablaufbeschreibungen. Diese werden als „UML Aktivitäten“ (Activities) bezeichnet. Für weitergehende Beschreibungen sei hier exemplarisch auf [90] verwiesen.

Im Folgenden wird die zuvor informelle Beschreibung der Ablaufbeschreibungen in Mengenschreibweise spezifiziert. Diese Spezifikation ist ebenfalls in Abbildung 54 dargestellt.

Als Erstes definieren wir die Menge der Kontrollknoten. Hierbei wird zwischen den Kontrollknotentypen und Kontrollknotenexemplaren unterschieden:

Sei $CNodeTypes :=_{DEF} \{n_{start}, n_{end}, n_{split}, n_{join}, n_{branch}, n_{merge}\}$ die Menge der verfügbaren Kontrollknotentypen. *(43)*

Dabei bezeichnet:

- n_{start} die Startknoten
- n_{end} die Endknoten
- n_{split} die Parallelisierungsknoten
- n_{join} die Synchronisationsknoten
- n_{branch} die Verzweigungsknoten
- n_{merge} die Verbindungsknoten

Sei $CNode_{id}$ die Menge der möglichen Kontrollknotenexemplare. *(44)*

Jedes Kontrollknotenexemplar hat einen Typ ($CNodeTypes$). Dieser wird durch die Funktion $CNodeTypeOf$ beschrieben:

Sei $CNodeTypeOf :=_{DEF} CNode_{id} \rightarrow CNodeTypes$ die Typisierung der Kontrollknotenexemplare. *(45)*

Der Einfachheit halber definieren wir nun die Menge möglicher Knoten in einer Aktion, bestehend aus den Produkt- und Produktkontextoperationen sowie einer Menge von Kontrollknoten:

Sei $NODES :=_{DEF} POperation \cup COperation \cup CNode$ *(46)*

Diese Knoten können nun über Kanten verbunden werden. Dies wird mit der folgenden Menge $EGDE$ repräsentiert:

Sei $EDGE :=_{DEF} NODES \times NODES$ die Menge möglicher Kanten zwischen Knoten (Produkt-/Produktkontextoperationen und Kontrollknoten). *(47)*

Nun kann die Menge möglicher Operationen definiert werden als:

Sei $ACTIONS :=_{DEF} \{(Nodes \subseteq NODES, Edge \subseteq EDGE)\}$ die Menge aller möglichen Aktionen, bestehend aus einer Menge von Knoten, die über Kanten verbunden sind. *(48)*

Die Darstellungen von Aktionen in UML-Notation ist in Abbildung 55 dargestellt.

Mit den vorstehend definierten Aktionen ist es nun möglich, mit einer Menge von atomaren Operationen auf dem Produktmodell und Produktkontextmodell fachliche Aktionen zu beschreiben. In dieser Arbeit erfolgt jedoch keine Beschreibung der Semantik der hier vorgestellten Aktionen. Dies hat vornehmlich den Grund, dass das im nächsten Kapitel 6 vorgestellte Verfahren zur Nutzerunterstützung Aktionen als atomare Einheiten zur Nutzerunterstützung sieht, und deshalb nur Sequenzen von Aktionen erlernt werden. Der Ablauf einer Aktion an sich wird, wie oben beschrieben, explizit modelliert und muss nicht erlernt werden. Deshalb ist die Semantik der Aktionen an dieser Stelle nicht relevant. Trotzdem sei hier exemplarisch auf Literatur verwiesen, die sich mit der Semantik von Abläufen beschäftigt: [91–94].

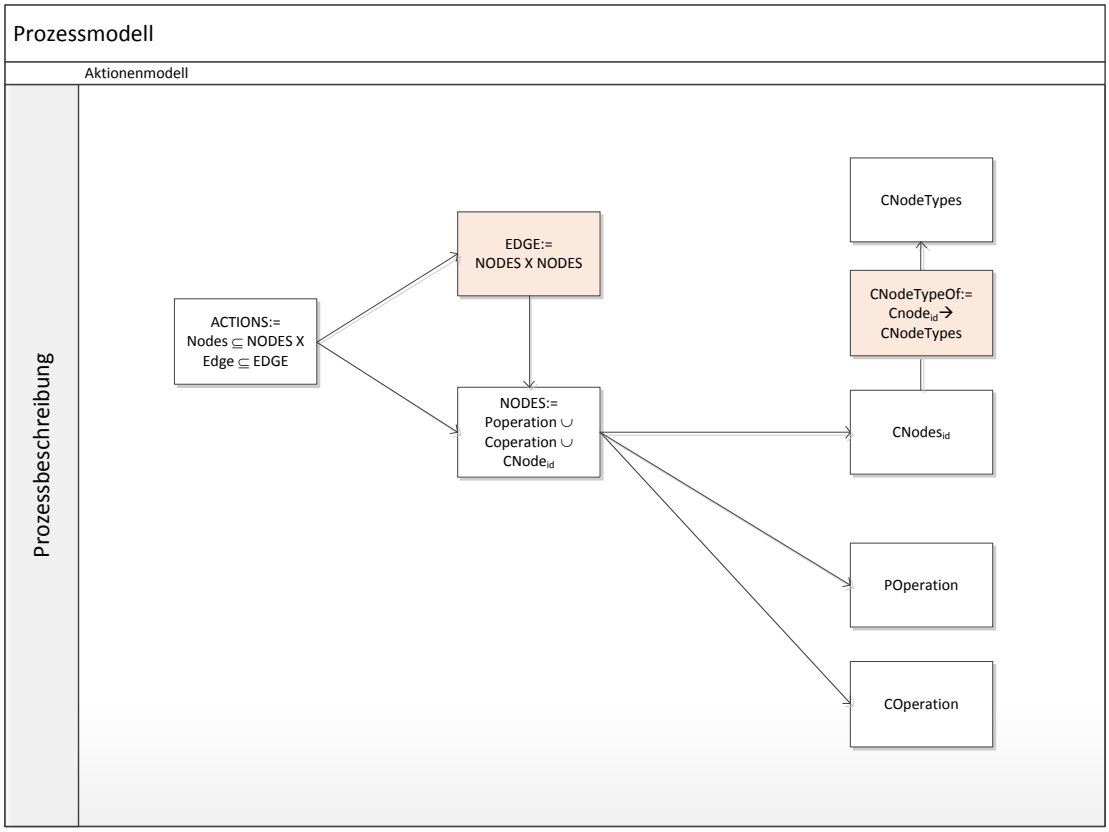


Abbildung 54: Aktionen in algebraischer Repräsentation

Für die semantische Beschreibung der Prozessbeschreibungssprache in Kapitel 5.5 ist es jedoch wichtig, dass festgestellt werden kann, wann eine Aktion vollständig beendet ist bzw. beendet werden kann. Normalerweise ist dies der Fall, wenn ein Kontrollknoten vom Typ n_{end} erreicht wird. Hierzu sei Folgendes vereinbart:

Eine gestartete Aktion a kann beendet werden $\Rightarrow end_a = 1$ sonst 0. *(49)*

Im Folgenden werden die Operationen der Beispielmethodik, beschrieben in Kapitel 3, vorgestellt. Weitere Teile der Modellierung der Beispielmethodik wurden bereits auf den Seiten 63 (Produktmodell) und 71 (Produktkontextmodell) dargestellt.

Dieses Beispiel beschreibt die Operationen auf dem Produktmodell, die nötig sind, um den Step „Komponente identifizieren“ durchzuführen. Als Erstes beschreiben wir die dafür nötigen Operationen (pm entspricht hierbei dem globalen Produktmodell, pc entspricht dem globalen Produktkontextmodell; unterstrichen: ungebundene Variablen, die zur Prozess-Laufzeit belegt werden). Der Einfachheit halber wird hier auf eine komplexe Mengendarstellung verzichtet. Stattdessen werden zuerst die Operationen dieser Menge aufgeführt, danach werden die Kontrollknoteninstanzen genannt, abschließend werden die Kanten modelliert.

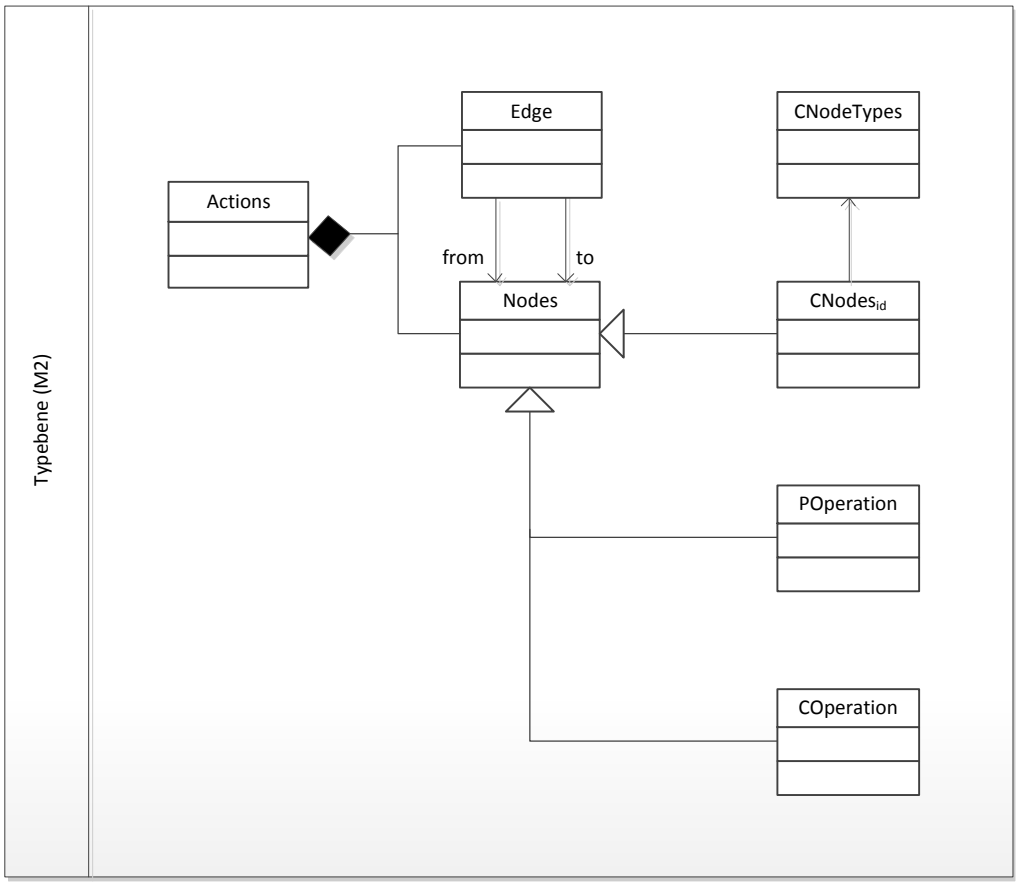


Abbildung 55: Aktionen in UML-Notation

Nachfolgend ist die Menge der Operationen ($KIAop$) für die Aktion $KompIdentifizierenA$ beschrieben:

- $addKomp(\underline{kname}) :=_{DEF}$ *(50)*

  $POperation_{addProduct}\left(pm, (\underline{kname}, \text{Komponente})\right)$

- $addSpez\left(\underline{kname\text{-}spez}\right) :=_{DEF}$

  $POperation_{addProduct}(pm, \left(\underline{kname\text{-}spez}, \text{Spezifikation}\right))$

- $addEdgeAnfKomp(\underline{ename}, \underline{kname}, \underline{aname}) :=_{DEF}$
  $POperation_{addEdge}(pm,$
  $(\underline{ename}, \text{anf-komp}), (\underline{aname}, \text{Anforderung}),$
  $(\underline{kname}, \text{Komponente}))$

- $addEdgeKompAnf(\underline{ename2}, \underline{kname}, \underline{aname}) :=_{DEF}$
  $POperation_{addEdge}(pm,$
  $(\underline{ename2}, \text{komp-anf}), (\underline{kname}, \text{Komponente}),$

($aname$,Anforderung))

- $addEdgeKompSpez\left(\underline{ename3}, \underline{kname}, \underline{kname\text{-}spez}\right) :=_{DEF}$ $POperation_{addEdge}(pm,$ ($\underline{ename3}$,komp-spez), ($\underline{kname}$,Komponente), ($\underline{kname\text{-}spez}$,Spezifikation))
- $addEdgeSpezKomp\left(\underline{ename4}, \underline{kname}, \underline{kname\text{-}spez}\right) :=_{DEF}$ $POperation_{addEdge}(pm,$ ($\underline{ename4}$,spez-komp),($\underline{kname\text{-}spez}$,Spezifikation), ($\underline{kname}$,Komponente))
- $addAnfToKC(\underline{kcname}, \underline{aname}) :=_{DEF}$ $COperation_{addPtoC}(pc, \underline{kcname}, \underline{aname})$

Nun folgt die Menge der Kontrollknoteninstanzen $KIAcnodes$:

$$KIAcnodes = ((start, n_{start}), (end, n_{end}), (s1, n_{split}), (s2, n_{split}), (s3, n_{split}), (j1, n_{join}), (j2, n_{join})) \quad (51)$$

Die Menge der Kanten $KIAegdes$ wird beschrieben als:

- $KIAedges = (\big((start, n_{start}), (s1, n_{split})\big), \big((s1, n_{split}), (addKomp)\big), \big((addKomp), (s2, n_{split})\big), \big((s2, n_{split}), (addEdgeAnfKomp)\big), \big((s2, n_{split}), (addEdgeKompAnf)\big), \big((s2, n_{split}), (addAnfToK)\big), \big((s2, n_{split}), (j1, n_{join})\big), \big((s1, n_{split}), (addSpez)\big), \big((addSpez), (j1, n_{join})\big),$ (52)

$$\Big((j1, n_{join}), (s3, n_{split})\Big),$$

$$\Big((s3, n_{split}), (addEdgeSepzKomp)\Big),$$

$$\Big((s3, n_{split}), (addEdgeKompSpez)\Big),$$

$$\Big((addEdgeSpezKomp), (j2, n_{join})\Big),$$

$$\Big((addEdgeKompSpez), (j2, n_{join})\Big),$$

$$\Big((addEdgeAnfKomp), (j2, n_{join})\Big),$$

$$\Big((addEdgeKompAnf), (j2, n_{join})\Big),$$

$$\Big((addAnhfToK), (j2, n_{join})\Big),$$

$$\Big((j2, n_{join}), (end, n_{end})\Big))$$

Durch diese Beschreibung wird sichergestellt, dass zuerst neue benötigte Modellelemente hinzugefügt werden und danach Kanten zwischen diesen Modellelementen erzeugt werden.

Die Aktion $KompIdentifizierenA$ kann nun beschrieben werden als:

$$KompIdentifizierenA = (KIAop \cup KIAcnodes, KIAedges) \quad (53)$$

In Abbildung 56 ist diese Aktion als UML-Aktivität beschrieben.

Der Einfachheit halber beschreiben wir im Folgenden die Aktion $KompIdentifizierenA$ mit den obigen Operationen als:

- $KompIdentifizierenA(\underline{kname}, \underline{kname\text{-}spez}, \underline{ename}, \underline{ename2}, \underline{ename3}, \underline{ename4}, \underline{aname}, \underline{kcname})$ *(54)*

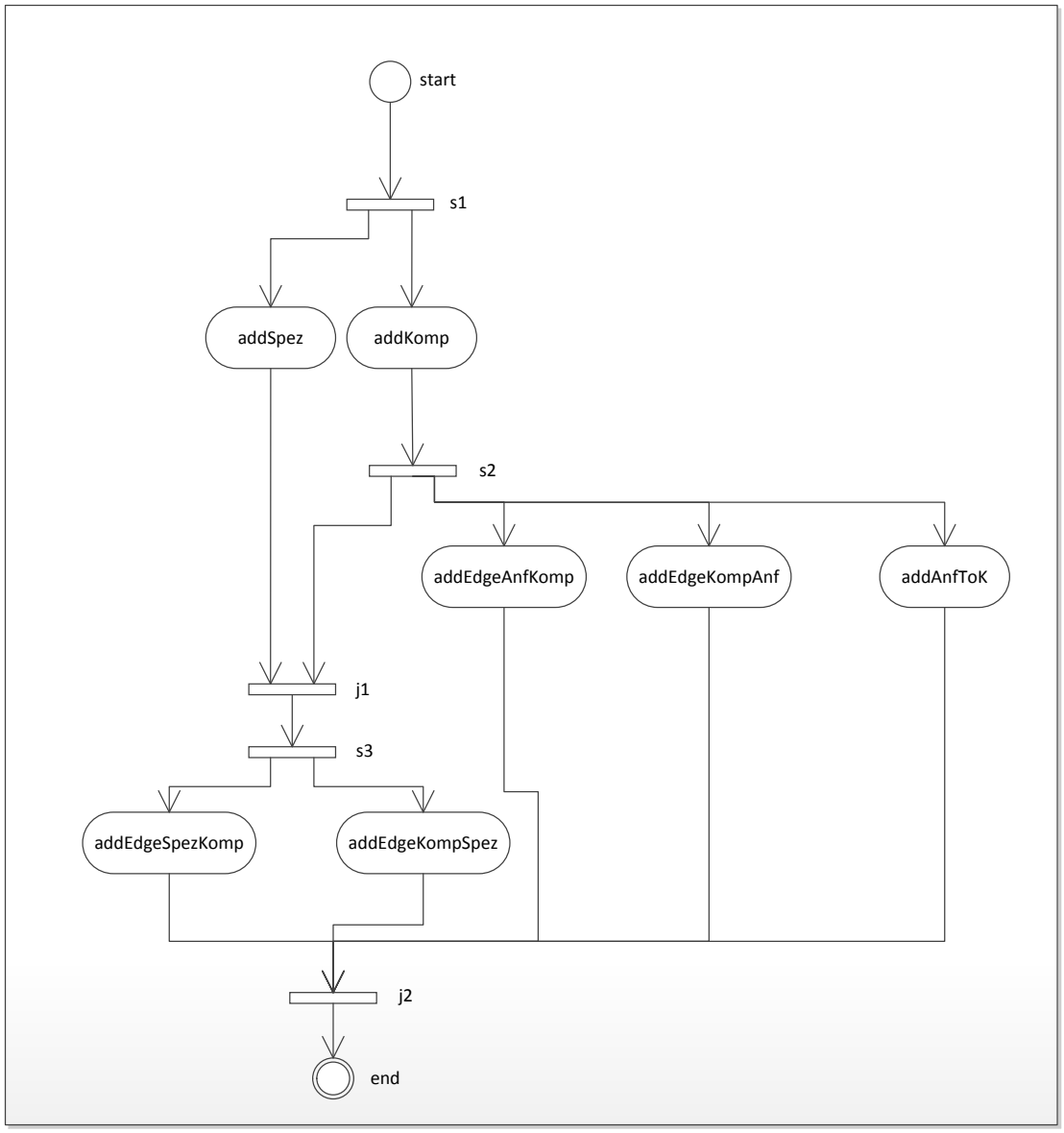


Abbildung 56: Aktion KompIdentifizierenA als UML-Aktivität

5.4 STEPS

Ein wichtiges Element unserer Software-Prozessbeschreibungssprache ist der Step. Ein Step (oder auch „Arbeitsschritt“) ist hauptsächlich das Element, das der Projektmitarbeiter „sieht“ und mit dem er arbeitet. Abbildung 57 zeigt den prinzipiellen Aufbau von Steps in UML-Notation.

Ein Step kann als Container verstanden werden, der als zentrales Element eine Aktion (vgl. Kapitel 5.3 ab Seite 73) enthält. In einer Prozessbeschreibungssprache sind vorhandene Steps nicht miteinander gekoppelt, d.h. es existiert keine explizit beschriebene Ablaufreihenfolge. Auch kann man keine Kompositionen von Steps modellieren. Deshalb genügt auch die in Kapitel 5.3.1 vorgestellte Variablenspezifikation, indem es nur globale Variablen gibt. Der Benutzer, also der Projektmitarbeiter, kann Steps starten und beenden. Um die Menge der startbaren Steps einzuschränken, gibt es die Möglichkeit, Vor- und Nachbedingungen zu beschreiben. Ein Arbeitsschritt kann nur vom Projektmitarbeiter gestartet werden, wenn alle Preconditions erfüllt sind. Ein bereits gestarteter Step kann vom Projektmitarbeiter nur beendet werden, wenn alle

Postconditions erfüllt sind. Zur Beschreibung der Vor- und Nachbedingungen ließe sich zum einen eine OCL-ähnliche Sprache (siehe [95]) integrieren. Mit OCL ist es möglich, sehr präzise Bedingungen zu formulieren. Auf der anderen Seite wäre es ebenfalls möglich, lediglich Bedingungen bzgl. der Produktzustände zu beschreiben. Die Prozessbeschreibungssprachen Marvel [36], [96] und Grapple [38] haben beispielsweise solche Vor- und Nachbedingungen integriert. In dieser Arbeit wird auf eine Modellierung der Vor- und Nachbedingungen verzichtet, da diese für die Nutzerunterstützung im folgenden Kapitel 6 nur von untergeordneter Bedeutung sind. Im Bereich der regelbasierten Prozessbeschreibungssprachen gibt es einige Ansätze, die sich mit diesen Bedingungen beschäftigen und auch versuchen, den Projektmitarbeiter bei der Erfüllung der Vor- und Nachbedingungen zu unterstützen. An dieser Stelle sei auf das Kapitel 2 verwiesen, das auch diese Ansätze vorstellt.

Weiterhin besitzt ein Step einen Produktkontext. Bei der Prozessbeschreibung wird hierfür eine Produktkontextvariable festgelegt (siehe Abbildung 57, Produktkontextvariable (PCV), Assoziationsname $pcvIn$). Während der Prozesslaufzeit ist diese Produktkontextvariable an ein Produktkontextexemplar gebunden. Ein Step wird somit unter einem bestimmten Kontext ausgeführt. Dies impliziert, dass mindestens ein Ausführungskontextexemplar von dem in der Stepbeschreibung spezifizierten Ausführungskontext zur Projektlaufzeit existent sein muss, um diesen Step starten zu können (neben den Preconditions, die ebenfalls erfüllt sein müssen).

Nach dem Starten eines Steps wird der Projektmitarbeiter eine Aktion, bestehend aus mehreren Operationen auf dem Produktmodell sowie dem Produktkontextmodell ausführen, die in der Stepbeschreibung spezifiziert ist. Üblicherweise sind dies Operationen auf dem Produktexemplarmodell. Hierfür müssen die in der Prozessbeschreibung (speziell: im Aktionenmodell) beschriebenen Variablen korrekt belegt sein. Dabei stellt ein Step zur Laufzeit die Verbindung zwischen den in den Aktionen spezifizierten Variablen und den (Produkt-) Exemplaren her.
Zum Einen kann eine Beschreibung der Belegung der Variablen schon bei der Prozessbeschreibung (also während der „Spezifikationszeit“ des Prozesses) feststehen. Bei Ausführung eines Steps könnte zum Beispiel eine Variable mit dem Wurzelelement des Produktkontextexemplars, unter dem dieser Step ausgeführt wird (pcvIn), belegt sein. Auf der anderen Seite gibt es Belegungen, die erst zur Laufzeit des Steps feststehen. Dies können zum Einen schon existierende Exemplare im Produktmodell sein (z.B. eine Anforderung, die der Benutzer explizit auswählt) andererseits aber auch ein (z.B. vom Benutzer) neu hinzugefügtes Exemplar (z.B. ein Komponentenexemplar beim Step „Komponente identifizieren“).

Durch Ausführen der Operationen kann sich eine Menge von Ausführungskontexten, die zur Projektlaufzeit existieren, ändern. Zu existierenden Produktkontextexemplaren können zum Beispiel neue existierende Produktexemplare hinzugefügt werden oder auch gänzlich neue Produktkontextexemplare erzeugt werden. Diese Menge ist für die Unterstützung des Projektmitarbeiters relevant (das Verfahren hierfür wird in Kapitel 6 vorgestellt). Deshalb werden, neben dem Ausführungskontext, unter dem ein Step ausgeführt wird, auch die Ausführungskontexte, die sich durch die durchzuführenden Aktionen ändern oder neu erzeugt wurden, beschrieben. Dies ist in Abbildung 57, Assoziationsname „$pcvOut$“, dargestellt.

Abschließend spezifiziert ein Step eine Menge von Kontexten. Ein Kontext beschreibt eine bestimmte Situation. Es könnte zum Beispiel einen Kontext „Programmiersprache“ geben, der zur Steplaufzeit belegt ist mit „Programiersprache=JAVA“.

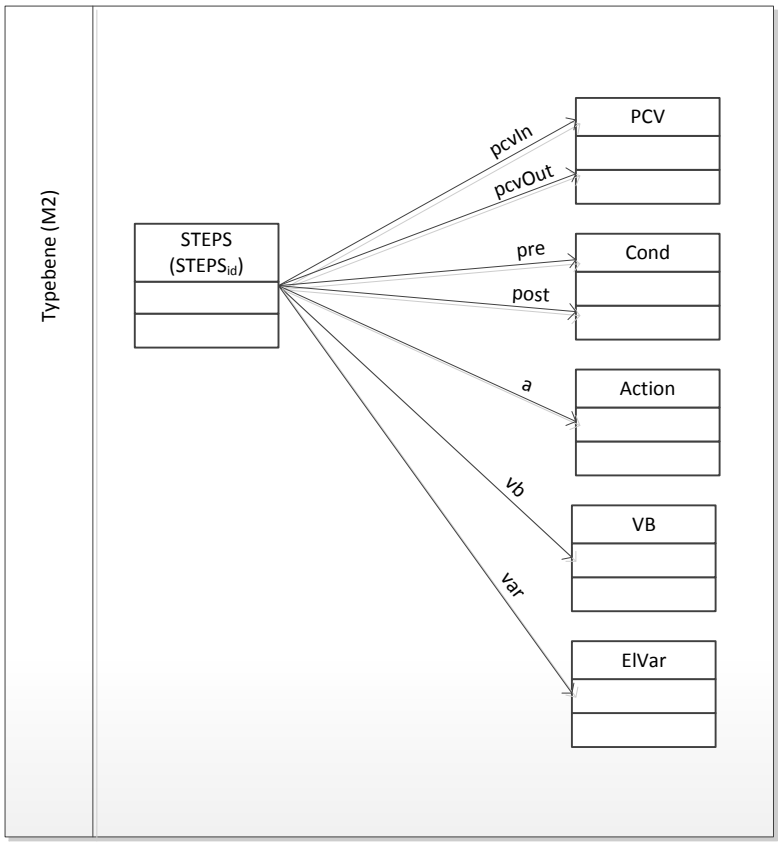


Abbildung 57: Steps

Die Menge aller Steps ist wie folgt definiert:

Sei $STEPS_{id}$ die Menge aller möglichen Steps. *(55)*

Die Assoziationen, zu sehen in Abbildung 57, werden im Folgenden definiert. Als Erstes beginnen wir mit der Referenz, die die Eingabekontextvariable $pcvIn$ beschreibt:

Sei $STEPSPCVIN \coloneqq_{DEF} STEPS_{id} \rightarrow PCV$ die Menge, die die Eingabekontextvariable der Steps repräsentiert. (56)

Sei $s \in STEPS_{id}$ ein beliebiges Element. Dann beschreibt $pcvIn_s$ die Produktkontextvariable.

Entsprechend der Eingabekontexte gibt es auch die Ausgabekontextvariablen, die sich ändernde oder neue Kontexte beschreiben:

Sei $STEPSPCVOUT \coloneqq_{DEF} STEPS_{id} \rightarrow \{PCV\}$ die Menge, die die Ausgabekontextvariablen der Steps repräsentiert. *(57)*

Sei $s \in STEPS_{id}$ ein beliebiges Element. Dann beschreibt $pcvOut_s$ die Menge der entsprechenden Produktkontextvariable.

Die folgenden zwei Mengen beschreiben die Vor- und Nachbedingungen der Steps:

Sei $STEPSCONDPRE \coloneqq_{DEF} STEPS_{id} \rightarrow \{Cond\}$ die Menge, die die Vorbedingungen der Steps repräsentiert. *(58)*

Sei $s \in STEPS_{id}$ ein beliebiges Element. Dann beschreibt pre_s die Menge der

entsprechenden Bedingungen.

Sei $STEPSCONDPOST \coloneqq_{DEF} STEPS_{id} \rightarrow \{Cond\}$ die Menge, die die Nachbedingungen der Steps repräsentiert. *(59)*

Sei $s \in STEPS_{id}$ ein beliebiges Element. Dann beschreibt $post_s$ die Menge der entsprechenden Bedingungen.

Die Variablenbelegungen werden mit der folgenden Funktion modelliert:

Sei $STEPSVB \coloneqq_{DEF} STEPS_{id} \rightarrow Pow(VB)$ die Menge, die die Variablenbelegungen der Steps repräsentiert. Diese sind bereits bei der Prozessbeschreibung festgelegt. *(60)*

Sei $s \in STEPS_{id}$ ein beliebiges Element. Dann beschreibt vb_s die Menge der entsprechenden Belegungen.

Es gibt Variablen, deren Belegung nicht zur Spezifikationszeit der Prozessbeschreibung feststehen. Diese müssen zur Laufzeit belegt werden:

Sei $STEPSVAR \coloneqq_{DEF} STEPS_{id} \rightarrow Pow(ElVar)$ die Menge, die die Variablen der Steps repräsentiert, die erst zur Prozesslaufzeit (z.B. durch den Benutzer) festgelegt werden. *(61)*

Sei $s \in STEPS_{id}$ ein beliebiges Element. Dann beschreibt var_s die Menge der entsprechenden Variablen.

Die durchzuführenden Aktionen der Steps werden mit der Funktion $STEPSACTION$ beschrieben:

Sei $STEPSACTION \coloneqq_{DEF} STEPS_{id} \rightarrow Actions$ die Menge, die die auszuführenden Aktionen der Steps repräsentiert. *(62)*

Sei $s \in STEPS_{id}$ ein beliebiges Element. Dann beschreibt a_s die auszuführende Aktion.

Im Folgenden wird der Step „Komponente identifizieren“ der Beispielmethodik beschrieben. Die vorherigen Teile der Methodik sind auf den Seiten 63 (Produktmodell), 71 (Produktkontextmodell) und 86 (Aktionenmodell) ersichtlich.

$KomponenteIdentifizierenStep \in STEPS$ mit:

- $pcvIn_{KomponenteIdentifizierenStep} = (\underline{acname})$
- $pre_{KomponenteIdentifizierenStep} = \emptyset$
- $vb_{KomponenteIdentifizierenStep} = ((\underline{aname}, el_{\underline{acname}}), (\underline{kcname}, pce_{el_{\underline{kname}}}))$
- $var_{KomponenteIdentifizierenStep} = (\underline{kname}, \underline{kname\text{-}spez}, \underline{ename}, \underline{ename2}, \underline{ename3}, \underline{ename4})$
- $a_{KomponenteIdentifizierenStep} = (KompIdentifizierenA)$
- $pcvOut_s = (\underline{kname}, \underline{kname\text{-}Spez})$
- $post_{KomponenteIdentifizierenStep} = \emptyset$

5.5 SEMANTIK DER PROZESSBESCHREIBUNGSSPRACHE

In diesem Unterkapitel wird die Semantik zur Ausführung der Prozessbeschreibungssprache bei einem gegebenen Modell (z.B. Prozessbeschreibung, Produktmodell, …) beschrieben. Es befasst sich also mit der Beschreibung, welche Arbeitsschritte (Steps) zu einem bestimmten Zeitpunkt prinzipiell vom Projektmitarbeiter gestartet und beendet werden können.

Die Menge der start- und stoppbaren Steps ist abhängig von der Prozessbeschreibung, also der Menge der definierten Steps inkl. Aktionen (und darin enthaltenen Operationen) und dem „aktuellen Zustand" des Projektes. Dieser aktuelle Zustand wird durch das Produktmodell, das Produktkontextmodell und das Aktionenmodell (insb. die Variablen und deren aktuelle Belegung) repräsentiert. Abhängig von diesen Modellen wird im Folgenden die Semantik der Prozessbeschreibungssprache, also welche Steps gestartet und beendet werden können, beschrieben. Zuerst werden diese Modelle zur Prozesslaufzeit definiert, danach werden darauf basierend Mengen und Funktionen vorgestellt, die die Möglichkeiten zum Starten und Beenden von Steps zu einem gegebenen Zeitpunkt (und Modellzustand) beschreiben.

Sei

$\boxed{PM}$ das aktuelle Produktmodell *(63)*

$\boxed{PC_{PM}}$ das aktuelle Produktkontextmodell *(64)*

$\boxed{VB}$ die aktuelle Variablenbelegungen *(65)*

$\boxed{ElVar}$ die Variablen *(66)*

$\boxed{STEPS}$ die Menge der Steps der Prozessbeschreibung *(67)*

zur Prozesslaufzeit.

Nun definieren wir die Menge $STARTEDSTEPS$. Diese beschreibt die Menge aller möglichen gestarteten Steps. Die Elemente dieser Menge setzten sich aus Elementen der Menge $STEPS$ sowie aus Elementen der Menge der Produktkontextexemplare PCE zusammen. Ein Step kann dabei nur mit einem Produktkontextexemplar gestartet werden, dessen Typ dem spezifizierten Produktkontexttyp ($pcvIn$) entspricht (siehe Kapitel 5.4 auf Seite 90).

Sei

$STARTEDSTEPS :=_{DEF} \{(s, pce) \in STEPS \times PCE | type_{pcvIn_s} = type_{pce}\}$ die Menge möglicher gestarteter Steps. *(68)*

Sei $ss = (s, pce) \in STARTEDSTEPS$ ein beliebiges Element. Dann bezeichnet s_{ss} den Step und pce_{ss} das Produktkontextexemplar.

Dann beschreibt

$\boxed{STARTEDSTEPS} \subseteq STARTEDSTEPS$ *(69)*

die Menge der gestarteten Steps zur Prozesslaufzeit. Zum Beginn eines Prozesses ist dies die leere Menge.

Nun definieren wir die Menge der startbaren Steps. Das sind genau die Steps, zu deren spezifiziertem Produktkontexttyp ($pcvIn$) mindestens ein Produktkontextexemplar mit demselben Typ existiert. Des Weiteren darf dieser Step mit dem Produktkontextexemplar noch

nicht gestartet sein (d.h. nicht in der Menge $\boxed{STARTEDSTEPS}$ enthalten sein), und die Vorbedingung muss erfüllt sein.

Die Menge der startbaren Steps beschreibt sich dann wie folgt:

$$STSTEPS :=_{DEF} \{s \in \boxed{STEPS} | \exists pce \in \boxed{PCE_{PC}} \text{ mit } type_{pce} = type_{pcvIn_s} \text{ und } (s, pce) \notin \boxed{STARTEDSTEPS} \text{ und } pre_s \text{ ist erfüllt}\} \quad (70)$$

Nun können wir die Funktion zum Starten eines Steps beschreiben. Die folgende Funktion $startStep$ nimmt als Argument die Menge der bisher gestarteten Steps ($\boxed{STARTEDSTEPS_0}$) sowie den zu startenden Step (s). Die Rückgabe ist die aktualisierte Menge der gestarteten Steps ($\boxed{STARTEDSTEPS_1}$).

- $startStep(\boxed{STARTEDSTEPS_0}, s) :=_{DEF} \boxed{STARTEDSTEPS_1}$ (71)
  - $s \in STSTEPS_0$
  - $ex_{pcvIn_s} \neq \emptyset$
  - $\boxed{STARTEDSTEPS_1} = \boxed{STARTEDSTEPS_0} \cup (s, ex_{pcvIn_s})$

Vor dem Starten eines Steps muss ein Produktkontextexemplar ausgewählt werden. Die Variable $pcvIn$ des Steps muss also mit einem Exemplar belegt sein.

Ist ein Step in den Zustand „gestartet“ übergegangen, steht die Variablenbelegung, die über vb zur Spezifikationszeit des Steps definiert wurde, fest. Die übrigen Variablen, definiert über var müssen vom Benutzer belegt werden. Dies kann bereits vor dem Starten des Steps geschehen, direkt nach dem Starten oder auch erst während dessen Ausführung, bevor eine Variable für eine Operation auf dem Produktmodell verwendet wird. Die Reihenfolge der auszuführenden Operationen ist über die spezifizierte Ablaufreihenfolge (siehe Kapitel 5.3.3 auf Seite 84) bestimmt.

Ein gestarteter Step kann wieder beendet werden, wenn die in der entsprechenden Aktion definierten Operationen durchgeführt wurden und deren Nachbedingung erfüllt ist.

Die Menge der stoppbaren Steps beschreibt sich dann wie folgt:

$$STOSTEP :=_{DEF} \{s \in \boxed{STARTEDSTEPS} | end_{a_{s_s}} = 1 \text{ und } post_s \text{ ist erfüllt}\} \quad (72)$$

Im Folgenden wird das Ausführen des Steps „Komponente identifizieren“ der Beispielmethodik beispielhaft ausgeführt.

Seien $\boxed{PM}$ und $\boxed{PC}_{\boxed{PM}}$ wie in den obigen Beispielen beschrieben. Ein Ausschnitt der Variablenbeschreibung $\boxed{ElVar}$ und der Variablenbelegung $\boxed{VB}$ (vor dem Starten des Steps) ist dann wie folgt:

$$\boxed{ElVar} = (\{(\underline{aname}, \text{Anforderung}), (\underline{kname}, \text{Komponente}), (\underline{kname\text{-}spez}, \text{Spezifikation}), (\underline{ename}, \text{anf-komp}), \quad (73)$$

$(\underline{ename2}$,komp-anf$)$, $(\underline{ename3}$,komp-spez$)$,

$(\underline{ename4}$,spez-komp$)$, $(\underline{acname}$,AnfC$)$, $(\underline{kcname}$,KompC$)\})$

Die Variablenbelegung sieht dann wie folgt aus:

$$\boxed{VB} = (\underline{aname},\text{anf1}), \quad (74)$$

$$(\underline{kname}, (\text{komp1,Komponente})),$$

$$\left(\underline{kname\text{-}spez}, (\text{spez-Komp1,Spezifikation})\right),$$

$$(\underline{ename}, (\text{anf1-komp1,anf1,komp1,anf-komp})),$$

$$(\underline{ename2}, (\text{komp1-anf1,komp1,anf1,komp-anf})),$$

$$(\underline{ename3}, (\text{komp1-spez-Komp1,komp1,spez-Komp1,komp-spez})),$$

$$(\underline{ename4}, (\text{spez-Komp1-komp1,spez-Komp1,komp1,spez-komp})),$$

$$(\underline{acname},\text{anf1c}),$$

$$(\underline{kcname},(\text{komp1c,KompC,}\emptyset\text{,systemc}))\})$$

Hinweis: Bei einer „echten" Prozessausführung in der Praxis würde die endgültige Variablenbelegung erst nach dem Starten eines Steps feststehen. Beispielsweise würde nach dem Starten des Steps „Komponente identifizieren" das Objekt „komp1" vom Typ „Komponente" vom Benutzer erzeugt werden.

Mögliche Steps sind:

$$\boxed{STEPS} = \{KomponenteIdentifizierenStep\} \quad (75)$$

Die Menge der gestarteten Steps ist leer:

$$\boxed{STARTEDSTEPS} = \emptyset \quad (76)$$

Nach dem Starten des Steps „Komponente identifizieren" mit obiger Variablenbelegung sehen die einzelnen Modelle wie folgt aus (neu hinzugefügte Elemente sind blau dargestellt):

- $\boxed{pm_a}$ = (//Produkttypen (77)

  {Lastenheft,Anforderung,IF,Systemarchitektur,System,

  Komponente,Spezifikation,IFD,IFI},

  // Produktexemplare

  {(lastenheft,Lastenheft),

  (systemarchitektur,Systemarchitektur),

  (system,System),

(anf1,Anforderung),(anf2,Anforderung),(anf3,Anforderung),
(komp1,Komponente),(spez-Komp1,Spezifikation)},
// Kantentypen
{(lh-anf,Lastenheft,Anforderung),
(anf-lh,Anforderung,Lastenheft),
(anfD-anfU,Anforderung,Anforderung),
(anf-komp,Anforderung,Komponente),
(komp-anf,Komponente,Anforderung),
(anf-if,Anforderung,IF),(if-anf,IF,Anforderung),
(sysa-komp,Systemarchitektur,Komponente),
(komp-sysa,Komponente,Systemarchitektur),
(komp-if,Komponente,IF),(if-komp,IF,Koponente),
(komp-spez,Komponente,Spezifikation),
(spez-komp,Spezifikation,Komponente),
(spez-ifd,Spezifikation,IFD),(ifd-spez,IFD,Spezifikation),
(if-ifd,IF,IFD),(ifd-if,IFD,IF),
(ifd-ifi,IFD,IFI),(ifi-ifd,IFI,IFD)},
// Kantenexemplare
{(lh-anf1,lastenheft,anf1,lh-anf),
(anf1-lh,anf1,lastenheft,anf-lh),
(lh-anf2,lastenheft,anf2,lh-anf),
(anf2-lh,anf2,lastenheft,anf-lh),
(lh-anf3,lastenheft,anf3,lh-anf),
(anf3-lh,anf3,lastenheft,anf-lh),
(anf1-komp1,anf1,komp1,anf-komp),
(komp1-anf1,komp1,anf1,komp-anf),
(komp1-spez-Komp1,komp1,spez-Komp1,komp-spez),
(spez-Komp1-komp1,spez-Komp1,komp1,spez-komp),
})

- $\boxed{pc_a}_{\boxed{pm_a}}$ = (*(78)*

{ //Produktkontexttypen
(SystemC,Systemarchitektur,
(Anforderung,Komponente,IF,Spezifikation,IFD,IFI),∅),
(AnfC,Anforderung,(System),SystemC),
(KompC,Komponente,(Anforderung,IF),SystemC),
(SpezC,Spezifikation,(IFD,IFI,IF),KompC)},
// Produktkontextexemplare
{(systemc,SystemC,systemarchitektur,(anf1,anf2,anf3),∅),
(anf1c,AnfC,anf1,system),
(anf2c,AnfC,anf2,system),
(anf3c,AnfC,anf3,system),
(komp1c,KompC,anf1,systemc)
})

6 NUTZERUNTERSTÜTZUNG BEI PROZESSAUSFÜHRUNG

In diesem Kapitel wird die Nutzerunterstützung während der Prozessausführung erläutert. Die grundsätzliche Idee hierzu ist bereits in Kapitel 4.2 ab Seite 42 beschrieben.

In der in Kapitel 5 spezifizierten Prozessbeschreibungssprache stehen die Arbeitsschritte, Steps genannt, „flach" nebeneinander. Das bedeutet, in einer Prozessbeschreibung werden keine Abläufe direkt spezifiziert. Der Projektmitarbeiter kann Steps starten und beenden - vgl. hierzu die Konzeption der Prozessbeschreibungssprache in Kapitel 4.4 ab Seite 47 und die eigentliche Spezifikation in Kapitel 5.4 ab Seite 90 inklusive der Produktkontextinformationen in Kapitel 5.2 ab Seite 66. Das in diesem Kapitel vorgestellte Verfahren für die Nutzerunterstützung beobachtet die letzten von Benutzer gestarteten Steps und schlägt die wahrscheinlich nächsten Steps, die der Projektmitarbeiter starten wird, vor. Hierfür wird eine Informationsbasis – hier genannt LookupDB – aufgebaut und erkannte Step-Sequenzen werden gespeichert.

Das hier vorgestelltes Verfahren zur Nutzerunterstützung baut auf einem existierenden Verfahren zur Sequenzvorhersage auf. Das folgende Kapitel 6.1 stellt das zugrunde liegende Verfahren vor. Kapitel 6.2 gibt einen Überblick über unser Verfahren zur Nutzerunterstützung, wie es beispielsweise angewendet wird und welche Informationen berücksichtigt werden. Danach folgt im Kapitel 6.3 die präzise Spezifikation unseres Lernverfahrens.

6.1 ZUGRUNDE LIEGENDE LERNVERFAHREN

Das in dieser Arbeit vorgestellte Verfahren zur Nutzerunterstützung erweitert die Sequenzvorhersage-Verfahren IPAM [62], [63] und das Verfahren von Jacobs und Blockeel [64]. Das zuletzt genannte Verfahren baut direkt auf IPAM auf. [78] stellt u.a. diese beiden Verfahren vor und vergleicht diese.

IPAM wurde bereits im Stand der Technik in Kapitel 2.2.2 auf Seite 28 kurz vorgestellt. Es wurde von Davison und Hirsh entwickelt. Die Anwendungsdomäne für dieses Verfahren ist die Vorhersage von Unix-Kommandos. Es verwendet ein Markov-Modell erster Ordnung, d.h. Vorhersagen basieren nur auf der letzten Beobachtung. Hierfür speichert IPAM nicht-bedingte und bedingte Wahrscheinlichkeiten in einer Datenbank. Nach jeder Beobachtung werden die Einträge in der Datenbank aktualisiert:

$$P'(x|y) = \begin{cases} \alpha \cdot P(x|y) + (1-\alpha), & \text{für } x = \text{aktuelle Beobachtung}, y = \text{letzte Beobachtung} \\ \alpha \cdot P(x|y), & \text{für } x \neq \text{aktuelle Beobachtung}, y = \text{letzte Beobachtung} \end{cases} \quad (79)$$

Entsprechend geschieht die Aktualisierung der nicht-bedingten Wahrscheinlichkeiten:

$$P'(x) = \begin{cases} \alpha \cdot P(x) + (1-\alpha), & \text{für } x = \text{aktuelle Beobachtung} \\ \alpha \cdot P(x), & \text{für } x \neq \text{aktuelle Beobachtung} \end{cases} \quad (80)$$

Ist eine zu aktualisierende bedingte Wahrscheinlichkeit noch nicht in der Datenbank enthalten, wird diese hinzugefügt und initial angenähert mit:

$$P'(x|y) = P(x) \quad (81)$$

Bei einer Beobachtung nicht enthaltene nicht-bedingte Wahrscheinlichkeiten werden mit 0 initialisiert.

Das Verfahren von Jacobs und Blockeel (vorgestellt im Stand der Technik in Kapitel 2.2.2 auf Seite 29) basiert auf IPAM, speichert jedoch ein Markov-Modell höherer Ordnung. Hierfür fügt es in das bestehende Verfahren einen Zwischenschritt ein. Nach jeder korrekten Vorhersage werden neue, „längere" bedingte Wahrscheinlichkeiten der Datenbank hinzugefügt: Sei $\dots y_1 y_0 x$ die aktuelle Beobachtung und die Vorhersage von x war richtig ($P(x|y_2 y_1 y_0)$ hatte beispielweise die höchste Wahrscheinlichkeit). In diesem Fall werden neue Einträge in die Datenbank hinzugefügt. Sei hierfür C die Menge der Suffixe $\dots y_1 y_0$ mit $P(a|c) > 0$ für alle $c \in C$. Sei weiterhin l der längste Suffix von $\dots y_1 y_0 x$ mit $P(a|l) > 0$. Folgende neue Wahrscheinlichkeiten werden der Datenbank dann hinzugefügt:

$$P(z|c \circ x) = P(z|l), \text{ für alle } c \in C \text{ und } z \in \text{bisherige Beobachtungen} \quad (82)$$

Die Motivation in diesem zusätzlichen Schritt ist folgende: Falls eine Vorhersage richtig war, wird angenommen, dass die Wahrscheinlichkeitsverteilung (entspricht C) der „letzten" Beobachtungssequenz richtig ist, korrekte Teilsequenzen also richtig erkannt wurden. In diesem Fall werden diese Teilsequenzen spekulativ „nach vorne" verlängert. Die daraus resultierenden, neu hinzugenommenen bedingten Wahrscheinlichkeiten werden approximiert (mit l).

6.2 ÜBERBLICK

Zentrale Voraussetzung für die Anwendung des hier vorgestellten Verfahrens zur Nutzerunterstützung ist die werkzeuggestützte Ausführung von Arbeitsschritten des Projektmitarbeiters. Dies ist erforderlich, weil das Verfahren auf einen Ausschnitt der zuletzt durchgeführten Arbeitsschritte (Steps) (siehe hierzu die Konzeption der Prozessbeschreibungssprache in Kapitel 4.4 ab Seite 47 als Überblick) und Produktkontexte (siehe Kapitel 4.4.3 auf Seite 49) zugreift, um spezifische Prozesse zu erlernen und Vorschläge für kommende Arbeitsschritte – also die eigentliche Nutzerunterstützung – zu machen.

6.2.1 SEQUENZEN

Relevant für die Anwendung des hier vorgestellten Verfahrens sind die beobachteten Sequenzen von gestarteten Arbeitsschritten (Steps) des Projektmitarbeiters. Ein Beispiel einer solchen Step-Sequenz ist in Abbildung 58 dargestellt. Diese Abbildung beschreibt eine mögliche Sequenz von Arbeitsschritten des beispielhaften Prozessausschnittes aus Kapitel 3. Unten in der Abbildung ist die Repräsentation der Arbeitsschrittsequenz mittels IDs, passend zur eigentlichen Sequenz, dargestellt. Diese IDs sind Repräsentanten der Elemente der Menge $STEPS$ (siehe (56) auf Seite 92 und Abbildung 57). Nachfolgend wird in diesem Kapitel nur noch die Darstellung mit den IDs verwendet.

Neben den Arbeitsschritten (Steps) sind die beobachteten Produktkontexte ebenfalls relevant für dieses Verfahren. Deshalb werden zu jedem ausgeführten Arbeitsschritt auch die dazu gehörenden und beobachteten Produktkontextexemplare erfasst. Eine beispielhafte Stepsequenz mit dazugehörigen Kontextinformationen ist in Abbildung 59 oben dargestellt. Hier werden die Kontexte schon als IDs (entsprechend der Step-IDs in Abbildung 58) angegeben. Die Produktkontext-IDs repräsentieren die Elemente der Menge PCE_{id} (siehe (14) auf Seite 67). In der Abbildung 59 unten ist diese Sequenz in reiner ID-Darstellung abgebildet. Die Elemente der ersten Spalte in dieser Abbildung (PC IN, PC OUT) bezeichnen den „Ort", an dem das Produktkontextexemplar bei Ausführung des Steps (1. Zeile in der Abbildung) beobachtet wurde. „PC IN" bezeichnet dabei das Produktkontextexemplar, mit dem der Step ausgeführt wurde. Das entspricht der Belegung der Eingabekontextvariablen $pcvIn_s$ (siehe (56) auf Seite 92). Mit anderen Worten: In Abbildung 59, während Step 1 ausgeführt wird, gilt: $ex_{vb_{pcvIn_1}} = 12$. vb liefert die Variablenbelegung der Variablen $pcvIn_1$ (siehe (34) auf Seite 75) und ex liefert das

Produktkontextexemplar dieser Variablenbelegung (siehe (37) auf Seite 76). Entsprechend bezeichnet in Abbildung 59 „PC OUT“ das Produktkontextexemplar, das durch die Ausgabekontextvariable $pcvOut$ des Steps repräsentiert wird (siehe (56) auf Seite 92).

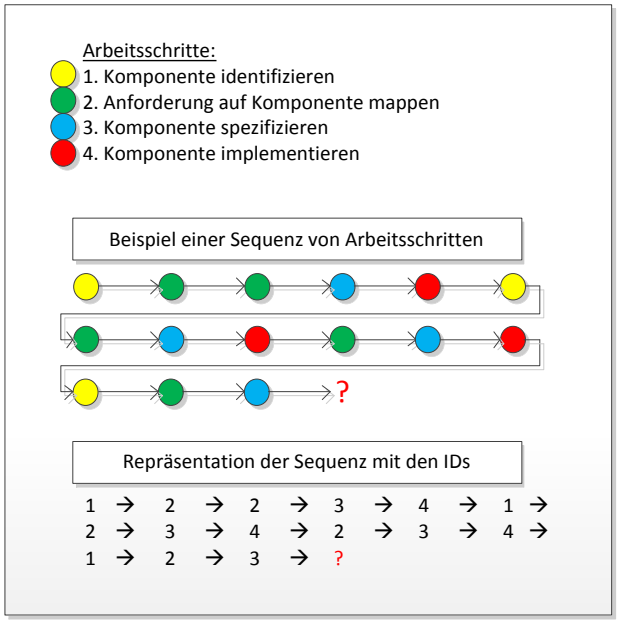


Abbildung 58: Beispiel Arbeitsschrittsequenz

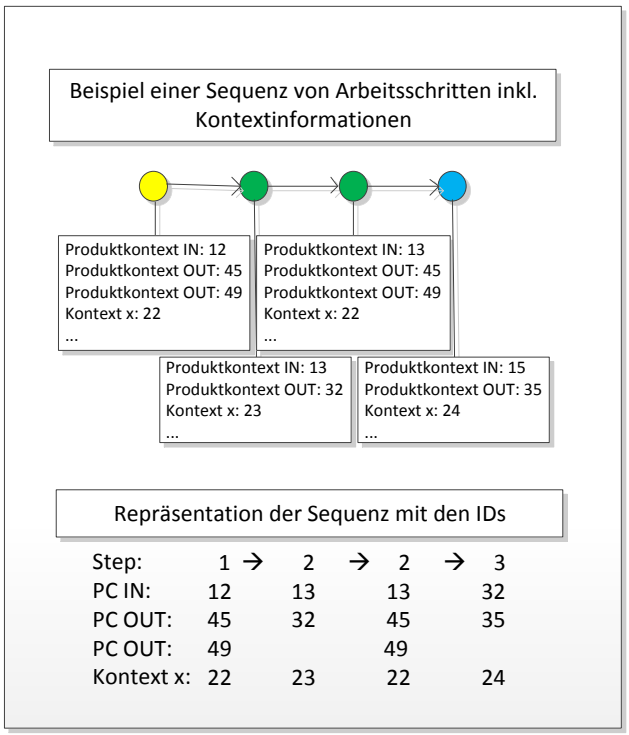


Abbildung 59: Beispiel Arbeitsschrittsequenz mit Kontextinformationen

6.2.2 VERFAHREN IM ÜBERBLICK

Ein Überblick über das Verfahren zur Unterstützung der Projektmitarbeiter ist in Abbildung 60 dargestellt. Motiviert wird das Verfahren bereits in Kapitel 4 ab Seite 39. Zentraler Teil dieses Verfahrens ist die LookupDB. Diese Datenbank speichert die erkannten und gelernten Beobachtungen während der Prozessausführung. Jedes Element in der LookupDB beinhaltet u.a. eine Teilsequenz von Steps (Bedingung/Cond), die in der aktuellen Stepsequenz beobachtet werden muss, damit das LookupDB-Element ein möglicher Kandidat für eine Vorhersage ist. Des Weiteren beinhaltet jedes Element eine Stepvorhersage (Prediction) und eine Wahrscheinlichkeit (P). Diese Parameter beschreiben die bedingte Wahrscheinlichkeit P, dass bei einer beobachteten Stepsequnz (Cond) als nächstes der Step „Prediction" gestartet wird (P(Prediction|Cond)). Beispielsweise beschreibt das LookupDB-Element mit der ID=5 in der Abbildung, dass, wenn die Stepsequenz 1→2→3 beobachtet wird, die Wahrscheinlichkeit, dass Step 1 gestartet wird, gleich 0,9 ist. Weiterhin werden in der LookupDB zu jedem beobachteten Step (in der Teilsequenz Cond) Produktkontextwahrscheinlichkeiten gespeichert.

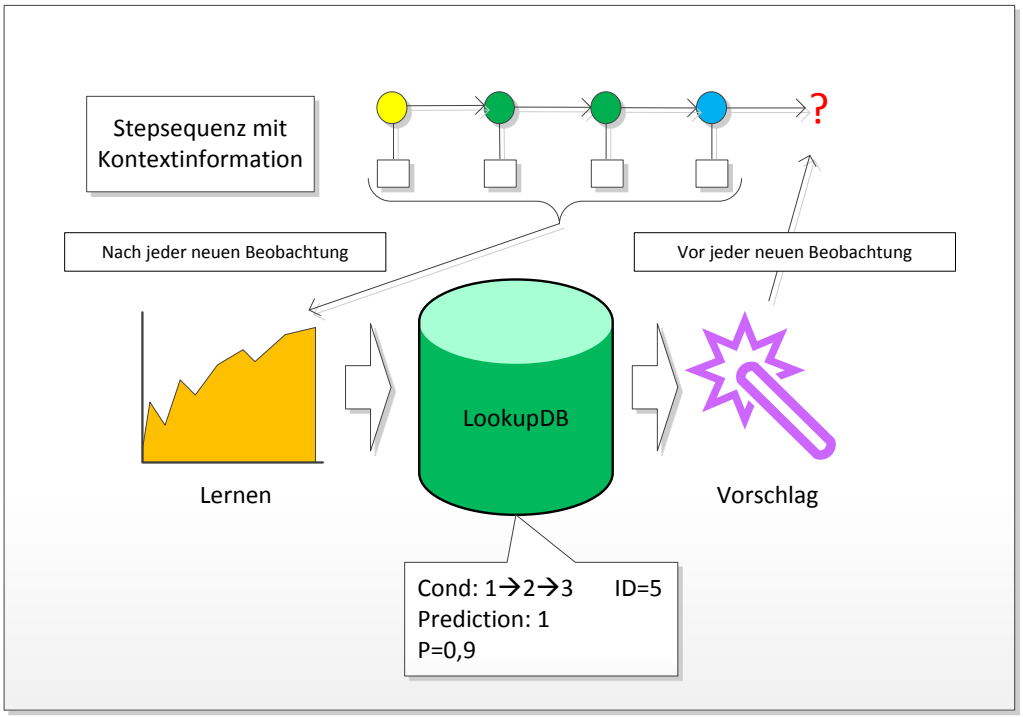


Abbildung 60: Elemente der Nutzerunterstützung im Überblick

Für das Vorschlagen eines Arbeitsschrittes, in Abbildung 60 rechts dargestellt, werden alle Elemente der LookupDB, deren Bedingung (Cond) mit der aktuellen Beobachtung übereinstimmt, verwendet. Für jedes dieser Elemente wird eine „aktuelle Wahrscheinlichkeit" berechnet. Hierfür werden die Wahrscheinlichkeit P des LookupDB-Elements und die gelernten Kontext-Wahrscheinlichkeiten (deren Kontexte aktuell beobachtet wurden) für die Berechnung verwendet. Das Element aus der LookupDB mit der höchsten aktuellen Wahrscheinlichkeit (oder n Elemente mit den n höchsten Wahrscheinlichkeiten) wird für die Vorhersage verwendet und die entsprechenden Steps (Prediction des jeweiligen Elements) werden vorgeschlagen.

Nach jeder Vorhersage und dem tatsächlichen Ausführen eines Steps durch den Projektmitarbeiter lernt das hier vorgestellte Verfahren diesen Step, indem es alle bedingten Wahrscheinlichkeiten und Kontextwahrscheinlichkeiten aller passenden Elemente der LookupDB aktualisiert.

In den folgenden Kapiteln 6.2.2.1 bis 6.2.2.3 wird das Verfahren informell erklärt, im darauf folgenden Kapitel 6.3 folgt eine detaillierte Beschreibung.

6.2.2.1 LookupDB

Die LookupDB dient der Speicherung der gelernten Prozesse, also Sequenzen von Steps inkl. der beobachteten Kontexte. Systematisches Vorgehen wird erkannt, indem Wahrscheinlichkeiten in den Elementen der LookupDB aktualisiert werden. Je häufiger eine Sequenz erkannt wird, umso höher werden die Wahrscheinlichkeiten der entsprechenden LookupDB-Elemente, und umso eher werden diese bei Bedarf vorgeschlagen.

Die Idee unseres Verfahrens ist die folgende: Methodisches Vorgehen wird über sich wiederholende Strukturen der Step-Sequenzen erkannt und gelernt. Zusätzlich werden zu jeder dieser gelernten Strukturen die beobachteten Kontexte erfasst und die beobachteten Wahrscheinlichkeiten gespeichert. Hierdurch wird es möglich, gelernte Teilsequenzen kontextspezifisch vorzuschlagen.

Abbildung 61 zeigt ein Beispiel eines Elements der LookupDB. Folgende Elemente sind dabei enthalten:

- **Cond (Condition)** beschreibt eine Abfolge von Steps, die beobachtet werden müssen, sodass das LookupDB-Element gültig ist. Entspricht Cond der aktuellen Beobachtung von Steps, so wird für dieses LookupDB-Element die „aktuelle Wahrscheinlichkeit" berechnet und mit den anderen gültigen LookupDB-Elementen verglichen.

- **Prediction** beschreibt einen Step, der den beobachteten Steps (Cond) folgt. Dieser Step wird, falls das LookupDB-Element relevant ist, also eine hohe „aktuelle Wahrscheinlichkeit" besitzt, vorgeschlagen.

- **P** beschreibt eine geschätzte Wahrscheinlichkeit, dass bei der Beobachtung von Cond der vorhergesagte Step (Prediction) eintritt. Die bisherigen drei Elemente können auch als bedingte Wahrscheinlichkeit verstanden werden: $P(\text{Prediction}|\text{Cond})$.

- **Context Relevance:** Hier werden die Produktkontextrelevanz-Werte gespeichert. Zu jeder Kontextklasse (z.B. PC IN, PC OUT, Kontext x, …, siehe hierzu Kapitel 6.2.1, insb. ab Seite 100) werden die Produktkontexte erfasst, die beobachtet wurden. Zu diesen werden Wahrscheinlichkeiten hinterlegt. In Abbildung 61 ist beispielsweise dargestellt, dass bei dem beobachteten Step 1 (siehe Cond: S1) der Kontext mit der ID 12 als Produktkontext IN (PC IN) beobachtet wurde. Als Wahrscheinlichkeit, dass dieser Kontext auftrat, wurde der Wert 0,7 gemessen.

- **Context Prediction:** Diese Informationen werden verwendet, um den Eingabekontext (PC IN) des nächsten Steps vorzuschlagen. Dies entspricht dem Produktkontextexemplar, das die Eingabekontextvariable $pcvIn$ eines Steps referenziert (siehe (56) auf Seite 92). In diesem Bereich werden relative Positionen zu beobachteten Kontexten gespeichert und mit Wahrscheinlichkeiten belegt. Dies ist dadurch motiviert, dass ein Projektmitarbeiter im Allgemeinen möglichst wenig Kontextwechsel durchführen will und die Wahrscheinlichkeit, dass ein Kontext, unter dem ein neuer Step gestartet wird, in der letzten Vergangenheit auftaucht, groß ist. Beispiel: Ein Projektmitarbeiter erzeugt ein neues Produktkontextexemplar, z.B. vom Typ Komponentenkontext, dieser wird dann unter PC OUT – ein Exemplar

das durch $pcvOut$ bei der Stepbeschreibung referenziert wird, siehe (56) auf Seite 92 – beobachtet. Der Projektmitarbeiter führt dann den Step „Komponente spezifizieren“ unter diesem neu hinzu gefügten Produktkontextexemplar aus (also: PC IN des „neuen“ Steps ist PC OUT des „alten“ Steps), um die Komponente zu spezifizieren).

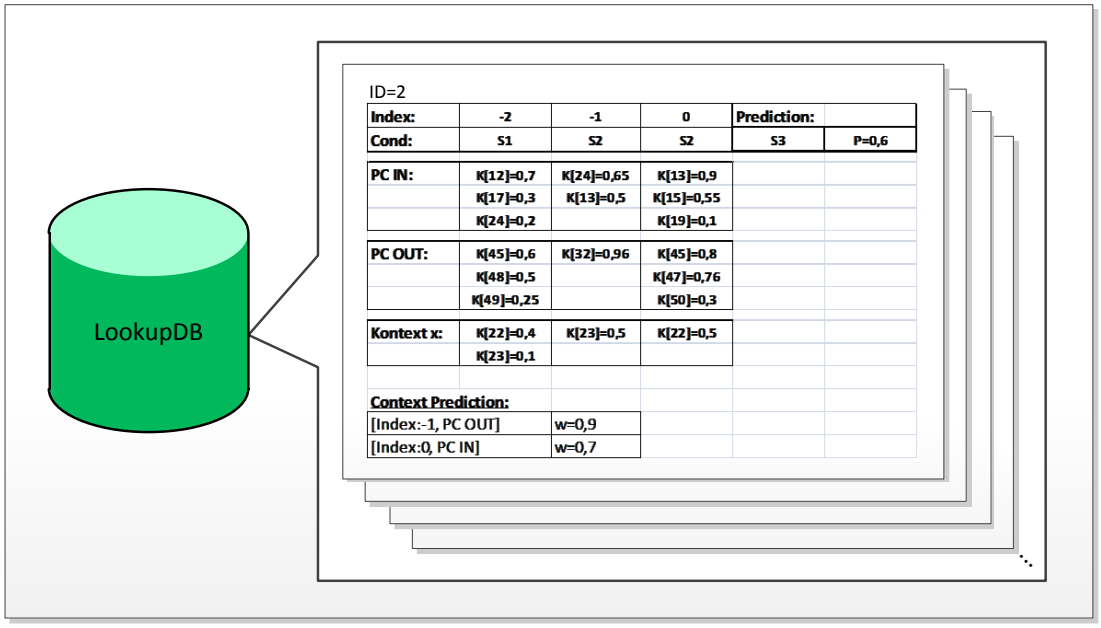


Abbildung 61: Beispiel eines Elements der LookupDB

6.2.2.2 Vorhersage

Die unserem Verfahren zugrunde liegende Sequenzvorhersage von Jacob/Blockeel, beschrieben in Kapitel 6.1, nutzt die gelernten bedingten Wahrscheinlichkeiten, um Arbeitsschritt-Vorhersagen zu treffen. Hierfür werden alle „passenden“ bedingten Wahrscheinlichkeiten herangezogen. Das sind diejenigen, deren Bedingung (bei $P(x|\dots y_0)$ wäre dies „$\dots y_0$“) mit der aktuellen Beobachtung übereinstimmt. x und y_* sind dabei die Arbeitsschritte, die vorhergesagt und beobachtet werden bzw. wurden. Das bedeutet, methodisches Vorgehen, also die Strukturen über die Steps, wird in Form der bedingten Wahrscheinlichkeiten gelernt. Die höchste bedingte Wahrscheinlichkeit wird dann für die nächste Vorhersage gewählt. Die Entscheidung, welche Arbeitsschritte vorgeschlagen werden, beruht alleine auf der zuletzt beobachteten Sequenz der Steps.

Unser Verfahren soll Arbeitsschritte kontextspezifisch vorschlagen. Hierfür enthält die LookupDB die Kontextrelevanzwerte. Zu jeder gelernten Struktur der Arbeitsschritte in Form einer bedingten Wahrscheinlichkeit werden also relevante Kontexte erfasst. Für die Auswahl einer bedingten Wahrscheinlichkeit zur Vorhersage wird dann nicht nur der eigentliche Wahrscheinlichkeitswert herangezogen, sondern eine „aktuelle Wahrscheinlichkeit“ berechnet. Diese setzt sich zusammen aus dem o.g. Wahrscheinlichkeitswert und den Kontextrelevanzwerten, deren korrespondierende Kontexte in der aktuellen Beobachtung enthalten sind. Mit anderen Worten: Enthält die aktuelle Beobachtung Kontextinformationen, die in der Vergangenheit bei gleichen Stepsequenzen erkannt wurden, so wird die entsprechende bedingte Wahrscheinlichkeit eher für Vorschläge herangezogen, als wenn die aktuellen Kontextinformationen nicht enthalten sind.

Das Vorgehen für den Vorschlag eines Steps ist dann das folgende: Es werden die Elemente der LookupDB herangezogen, deren Cond mit der aktuellen Beobachtung übereinstimmt. Für jedes dieser Elemente wird eine aktuelle, kontextspezifische Wahrscheinlichkeit ermittelt. Die Elemente mit der höchsten aktuellen Wahrscheinlichkeit werden dann als Vorschlag für den nächsten Step gewählt.

In die Berechnung der aktuellen, kontextspezifischen Wahrscheinlichkeit fließen einerseits die Gesamtwahrscheinlichkeit P des LookupDB-Elements ein, andererseits die Kontextrelevanzwerte der Kontexte, die in der aktuellen Beobachtung erkannt wurden. Aus diesen für die aktuelle Beobachtung relevanten Produktkontextwerten wird der Mittelwert gebildet, um so eine „Gesamtkontextrelevanz" zu erhalten.

Eine Beschreibung des Verfahrens für den Vorschlag von Steps ist in Abbildung 62 dargestellt.

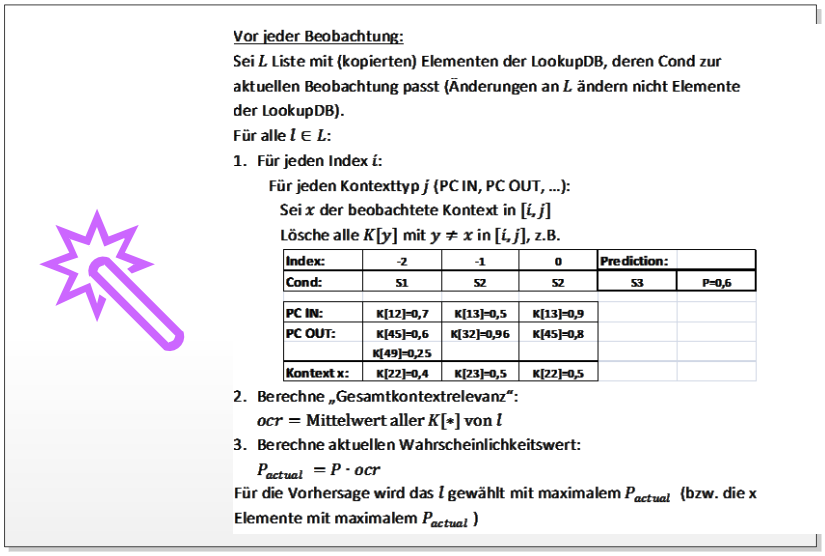
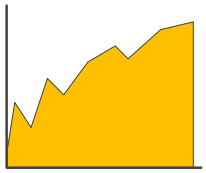


Abbildung 62: Informelle Beschreibung des Algorithmus zur Vorhersage eines Steps

6.2.2.3 Lernen

Die grundsätzliche Idee für das Lernen, also das Aktualisieren der LookupDB ist folgende: Methodisches Vorgehen, d.h. Teilsequenzen von Arbeitsschritten, wird wie beim Standardverfahren von Jacobs/Blockeel gelernt, d.h. hier werden bedingte Wahrscheinlichkeiten angelegt und sukzessive aktualisiert. Die Bedingung (Cond) und die Vorhersage (Prediction) entsprechen dabei einer Teilsequenz von Arbeitsschritten (z.B. $P(Prediction|Cond)$). Um kontextspezifisches, methodisches Vorgehen zu erlernen, erweitern wir unsere Teilsequenzen/bedingten Wahrscheinlichkeiten. Zu jedem beobachteten Arbeitsschritt der Teilsequenz werden die beobachteten Kontextinformationen aufgeführt und die Wahrscheinlichkeit des Auftretens (seit dem Hinzufügen der Teilsequenz zu der LookupDB) hinterlegt. Folgende Information befindet sich beispielsweise in der LookupDB: Bei einer bestimmten Teilsequenz x der LookupDB wurde der erste Arbeitsschritt (in dieser Sequenz) mit einer Wahrscheinlichkeit y unter dem Kontext z ausgeführt.

Nach jeder Beobachtung eines Steps und deren Kontextinformationen wird also die LookupDB mit ihren Einträgen aktualisiert. Die informelle Beschreibung, wie dies vonstatten geht, ist in Abbildung 63 dargestellt und wird im Folgenden beschrieben.



Nach jeder Beobachtung:

Für alle Elemente in der LookupDB, deren Cond zur aktuellen Beobachtung (ohne den letzten beobachteten Step – dieser wird gelernt) and deren Prediction zum letzten beobachteten Step passt:

1. Aktualisiere P:
   $P = \alpha \cdot P + (1 - \alpha)$
2. Aktualisiere Kontext-Relevanz:
   Für jeden Index i:
   Für jeden Kontexttyp j (PC IN, PC OUT, ...):
   Sei α der beobachtete Kontext in [i,j] (z.B. [-2, PC IN]: Kontext 12)
   Aktualisiere Kontext-Relevanz mit:
   $K[\alpha] = \frac{\#\text{ bisher beobachtete } \alpha \text{ s in [i,j]}}{\#\text{ bisher beobachtete Kontexte in [i,j]}}$
3. Aktualisiere Kontext-Vorhersage:
   Sei c der zu lernende Kontext (PC IN des letzten beobachteten Steps):
   Sei l eine Liste mit Elementen [Index: x,Kontextklasse] in denen c beobachtet wurde.
   Für alle l: Aktualisiere w: $w = \alpha \cdot w + (1 - \alpha)$
4. Falls letzte Vorhersage korrekt war:
   Füge neue Elemente zur LookupDB hinzu, siehe x

Abbildung 63: Informelle Beschreibung des Algorithmus zum Aktualisieren der LookupDB

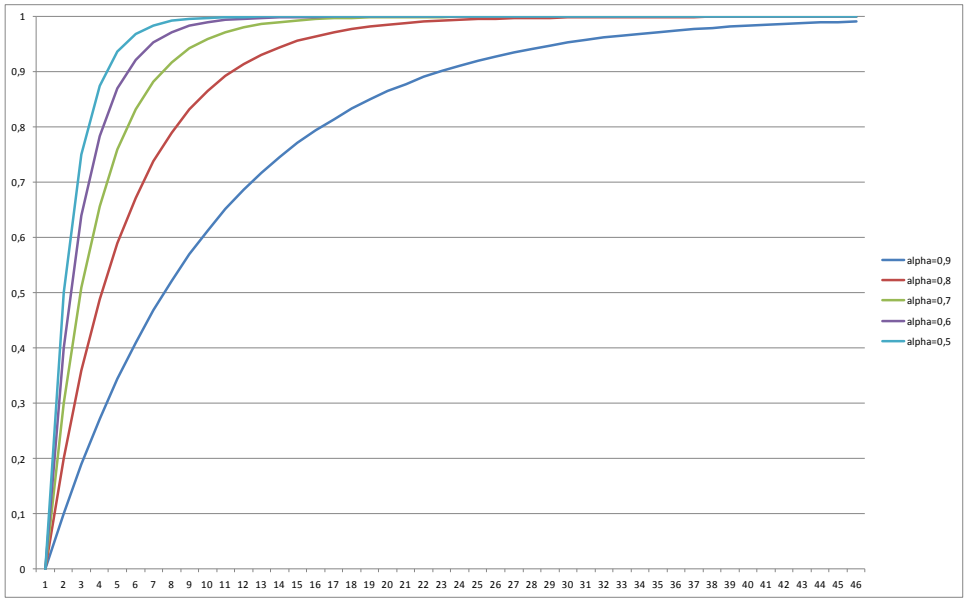


Abbildung 64: Wahrscheinlichkeitskurven mit verschiedenen α-Konstanten

Ziel ist es, den letzten Step (mit Produktkontexten) in der Beobachtung zu lernen. Hierfür werden alle LookupDB-Elemente, deren Condition zu der letzten Beobachtungssequenz ohne das letzte Element passt und deren Prediction dem letzten Element der Beobachtung entspricht, aktualisiert. Ein Beispiel hierzu ist in Abbildung 65 dargestellt.

Für jedes dieser Elemente wird daraufhin die geschätzte Wahrscheinlichkeit P aktualisiert. Dies geschieht mit der Funktion wie in Abbildung 63 (unter 1.) beschrieben. α ist hierbei eine Konstante im Intervall $(0,1)$. Die Funktion ist rekursiv und wird bei jedem Lernen mit dem bisherigen Wahrscheinlichkeitswert aufgerufen. Sie beginnt mit der Wahrscheinlichkeit $P = 0$ und nähert sich sukzessive dem maximalen Wert 1 an. Die Geschwindigkeit der Annäherung hängt von der Konstante α ab. Verschiedene Wahrscheinlichkeitskurven für verschiedene α sind in Abbildung 64 beispielhaft dargestellt. Je größer die Konstante ist, desto träger, d.h. langsamer ändert sich der Wahrscheinlichkeitswert. Wird ein kleiner Wert für α gewählt (z.B. 0,6), so wachsen die Wahrscheinlichkeitswerte der beobachteten Teilsequenzen schnell. Dies führt zu durchgängig höheren Wahrscheinlichkeitswerten im Vergleich mit einem hohen α-Wert. Dadurch, dass die Wahrscheinlichkeitswerte auch schneller wachsen, hat dies Einfluss auf die Geschwindigkeit des Lernens neuer Teilsequenzen. Wird eine neue Teilsequenz hinzugefügt, besitzt diese „schnell“ einen hohen Wahrscheinlichkeitswert. Da dieser Wert auch zur Berechnung der aktuellen Wahrscheinlichkeit berücksichtigt wird, wird die entsprechende Teilsequenz früher für Vorschläge herangezogen als bei einem großen α-Wert.

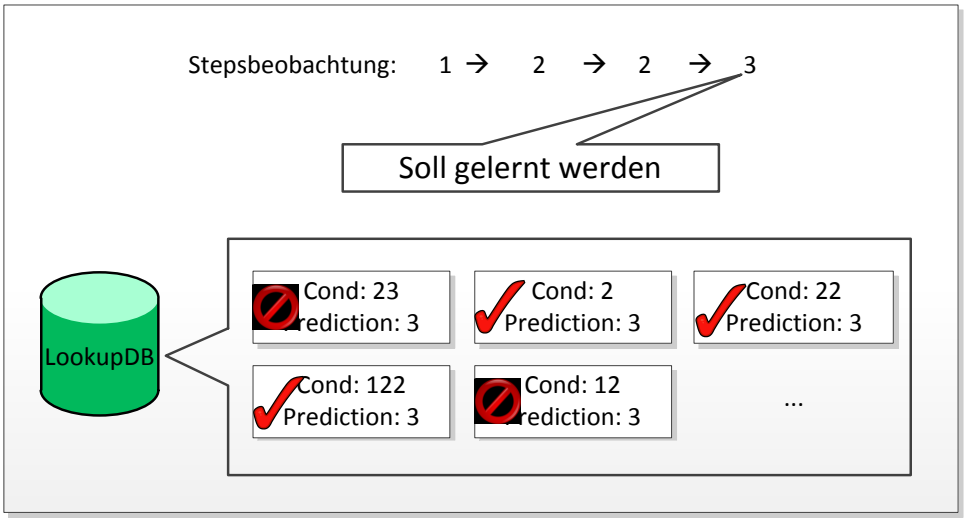


Abbildung 65: Auswahl passender LookupDB-Elemente entsprechend der Stepbeobachtung

Nach Aktualisierung der Gesamtwahrscheinlichkeitswerte P werden die Werte zu jeder Produktkontextwahrscheinlichkeit aktualisiert (siehe Abbildung 63, 2.). Hier wird zu jeder Produktkontextwahrscheinlichkeit (also die Wahrscheinlichkeit, dass ein bestimmtes Kontextexemplar k an einer bestimmten relativen Position i an einer bestimmten Position j auftritt (z.B. PC IN, siehe hierzu Kapitel 6.2.1 auf Seite 100)) die Anzahl der bisher (seit das LookupDB-Element existiert) beobachteten Kontextexemplare k in $[i,j]$ sowie die Gesamtanzahl der beobachteten Kontexte in $[i,j]$ gemessen.

Des Weiteren werden die Werte für die Kontextvorhersage aktualisiert (siehe Abbildung 63, 3.): Hier wird der Kontext PC IN des letzten beobachteten Steps (unter diesem Kontext wurde dieser Step ausgeführt) gelernt. Dabei werden die Gewichte zur Kontextvorhersage für den zu lernenden Kontext an den relativen Positionen aktualisiert, an denen dieser Kontext (PC IN des letzten Steps) beobachtet wurde.

Ein Beispiel für das Aktualisieren der Kontextvorhersage ist in Abbildung 66 dargestellt. Gelernt werden soll, dass Step 3 hier im Kontext 32 ausgeführt wurde. Dieser Kontext 32 wurde bereits bei dem ersten Step 2 (Index: -1) bei PC OUT beobachtet. Diese relative Position wird gelernt und für die Kontextvorhersage verwendet.

Der Vorteil beim Erlernen der relativen Positionen zur Kontextvorhersage ist, dass ein Step nicht immer im gleichen Kontext ausgeführt wird, obwohl die Abfolge von Steps bei verschiedenen Prozessdurchläufen gleich ist. Als Beispiel sei hier der Ablauf „Komponente identifizieren“ → „Komponente spezifizieren“ genannt. Im ersteren Step wird ein neuer Komponentenkontext erzeugt (PC OUT, siehe Seite 100), unter diesem wird dann der darauf folgende Step „Komponente spezifizieren“ ausgeführt (PC IN).

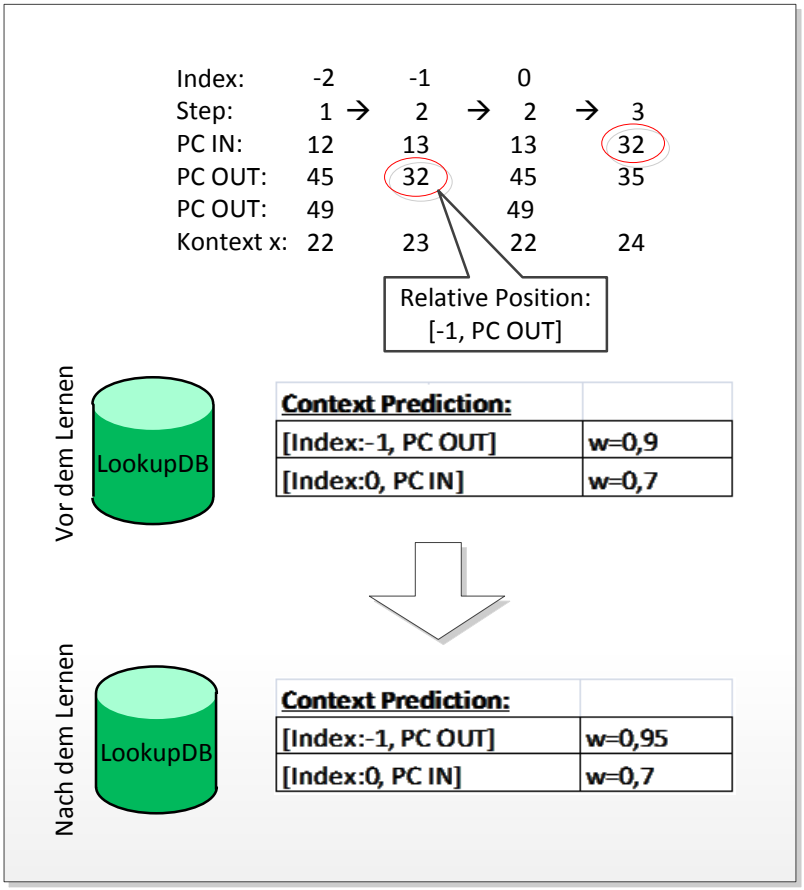


Abbildung 66: Beispiel Aktualisierung Kontextvorhersage

Sofern die letzte Vorhersage korrekt war, werden neue Elemente zu der LookupDB gemäß dem Verfahren, beschrieben in [64], hinzugefügt. Hierfür werden die Elemente der LookupDB gewählt, deren Cond zur Beobachtung (bis auf das letzte zu lernende Element) passen (siehe Abbildung 67, C). Da die Vorhersage korrekt war, wird angenommen, dass die Wahrscheinlichkeiten P dieser Menge von Elementen korrekt verteilt sind – also die Sequenzen richtig gelernt wurden. Diese Menge von Elementen wird dupliziert, und die Bedingungen (Conds) der neu hinzugefügten Elemente werden „nach vorne“ verlängert, nämlich um die aktuelle Beobachtung. Die Wahrscheinlichkeit P für den jeweiligen vorhergesagten Step (Prediction) wird approximiert (siehe Abbildung 67, l). Ein beispielhafter Teilschnitt einer LookupDB, jeweils vor dem Hinzufügen und nach dem Hinzufügen neuer Elemente, ist in Abbildung 68 dargestellt.

Im Folgenden Kapitel 6.3 werden alle Bereiche der Nutzerunterstützung – die LookupDB als Basis für die Erfahrungswerte, das Verfahren zum Trainieren der Erfahrungswerte sowie das Verfahren zum Vorschlagen des nächsten Steps – im Detail vorgestellt. Die LookupDB wird dabei im Kapitel 6.3.1, die Vorhersage im Kapitel 6.3.2 und das Lernen im Kapitel 6.3.3 beschrieben.

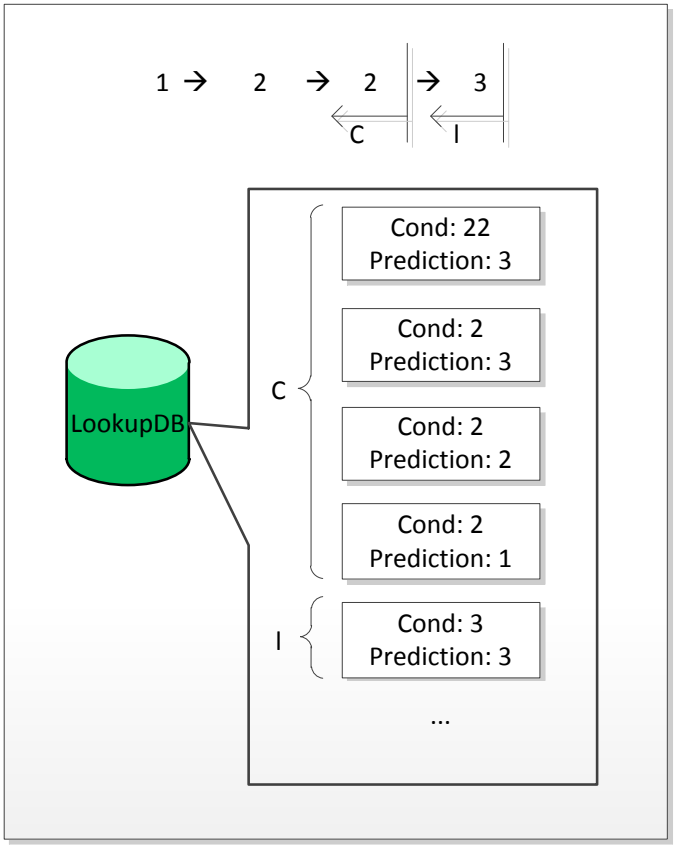


Abbildung 67: C und l – anschaulich erklärt

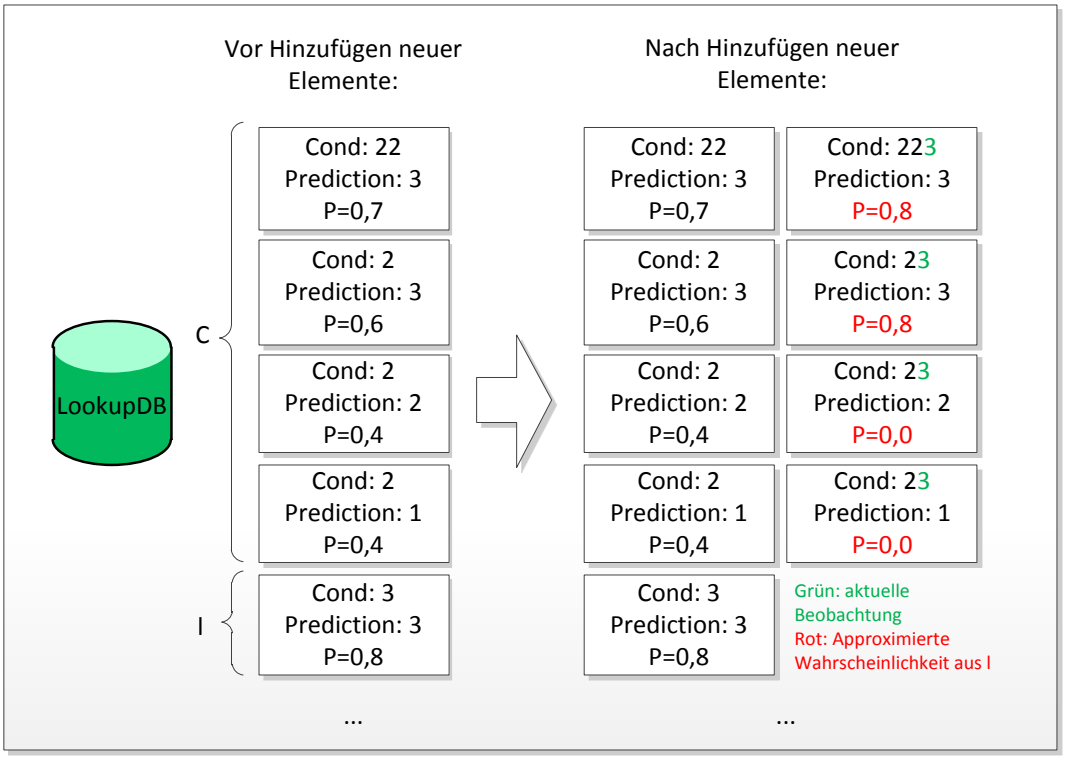


Abbildung 68: Hinzufügen neuer LookupDB-Elemente

6.3 NUTZERUNTERSTÜTZUNG IM DETAIL

In diesem Unterkapitel wird unser Verfahren zur Nutzerunterstützung im Detail vorgestellt. [97], [98] bieten gute Überblicke über dieses Verfahren.

Für die Nutzerunterstützung müssen wir auf die Historie der gestarteten Steps und deren Kontextinformationen zugreifen können. Hierfür benötigen wir eine Indexmenge I:

$$\text{Sei } I :=_{DEF} [-n, \dots, 0] \text{ Indexmenge.} \tag{83}$$

Die folgende Funktion beschreibt die letzten beobachteten Steps zu einem Zeitpunkt während der Prozessausführung:

$$observation_s :=_{DEF} I \rightarrow STEPS \tag{84}$$

Des Weiteren benötigen wir Funktionen, um auf die Produktkontextinformationen – also die Produktkontextexemplare, die von den Produktkontextklassen $pcvIn$ und $pcvOut$ des jeweiligen Steps referenziert wurden (siehe hierzu (56) auf Seite 92) – der letzten Steps während der Ausführung zugreifen zu können:

$$observation_{pcvIn} :=_{DEF} I \rightarrow PCE \tag{85}$$

$$observation_{pcvOut} :=_{DEF} I \rightarrow Pow(PCE) \tag{86}$$

Diese Funktionen liefern zu einer relativen Position den beobachteten Step (Index s) sowie dessen Kontextinformationen (Indizes $pcvIn$ und $pcvOut$). Abbildung 59 zeigt eine Arbeitsschrittsequenz mit dazugehörigen Kontextinformationen. Die o.g. Funktionen liefern zu der Beobachtung in dieser Abbildung Folgendes: $observation_s(0) = 3$, $observation_s(-1) = 2$, $observation_s(-2) = 2$ etc. und $observation_{pcvIn}(0) = 32$.

Um methodisches Vorgehen, also Teilsequenzen von Steps, lernen zu können, ist es bei unserem Verfahren nicht nötig, die gesamte Historie der Beobachtungen (über die Funktionen $observation_*$) verfügbar zu haben. Wichtig ist nur, dass mindestens so viele Informationen (Anzahl der Steps und zugehörige Kontextinformationen) der Historie verfügbar sind, wie die längste gelernte Teilsequenz[21]. Denn nur so ist es möglich zu entscheiden, ob eine Teilsequenz zu der aktuellen Beobachtung passt oder nicht.

Nun können wir uns die LookupDB definieren.

6.3.1 LOOKUPDB

In Abbildung 69 ist die LookupDB in UML-Notation dargestellt. Abbildung 70 zeigt die LookupDB in algebraischer Repräsentation.

[21] Dies ist natürlich nur möglich, falls die entsprechende Anzahl an Steps bereits beobachtet wurde. Dies ist der Fall, wenn beim Start eines Prozesses die Erfahrung neu aufgebaut wird, d.h. bei Prozessbeginn nicht auf frühere Erfahrungsdaten zurück gegriffen wird.

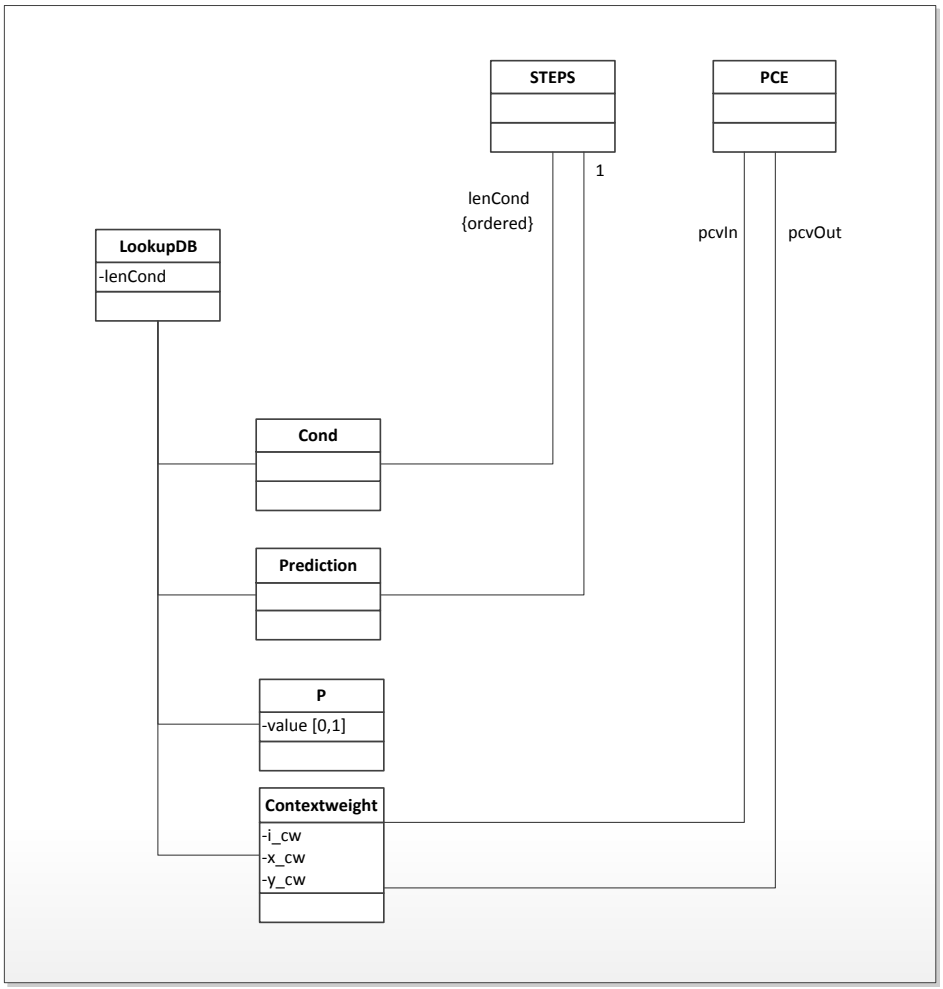


Abbildung 69: LookupDB in UML-Notation

Als Erstes definieren wir uns IDs der Elemente der Lookup-DB als Indexmenge:

Sei $LOOKUPDB :=_{DEF} \{0,1,2,\dots\}$ Indexmenge. *(87)*

Jedes Element dieser Menge repräsentiert ein Element der Lookup-DB. Diese LookupDB-Elemente besitzen eine Stepabfolge, die für die Vorhersage beobachtet wurde (Cond). In Cond sind also die Elemente der gelernten Teilsequenzen enthalten, bis auf das letzte Element. Dieses wird zur Vorhersage herangezogen und wird deshalb separat hinterlegt. Dies kann auch mit einer bedingten Wahrscheinlichkeit verglichen werden: Bei $P(Prediction|Cond)$ ist Cond genau die o.g. Abfolge von Steps. Zur Beschreibung der Menge aller möglichen Abfolgen bedienen wir uns wieder der Indexmenge I:

Sei $COND :=_{DEF} \{LOOKUPDB \times I \times STEPS\}$ die Menge der möglichen Stepabfolgen der Elemente der Lookup-DB. Das sichtbare LookupDB-Element mit der ID=2 aus Abbildung 61 hat z.B. folgendes Element: (2,-1,2). Das bedeutet: An der relativen Position -1 wurde Step 2 beobachtet. Beachte: Elemente der LookupDB können unterschiedlich lange Beobachtungssequenzen haben. *(88)*

Sei $ldb \in LOOKUPDB$ beliebiges Element. Dann beschreibt $cond_{ldb,i}$ den beobachteten Step des Elements ldb an der Position i.

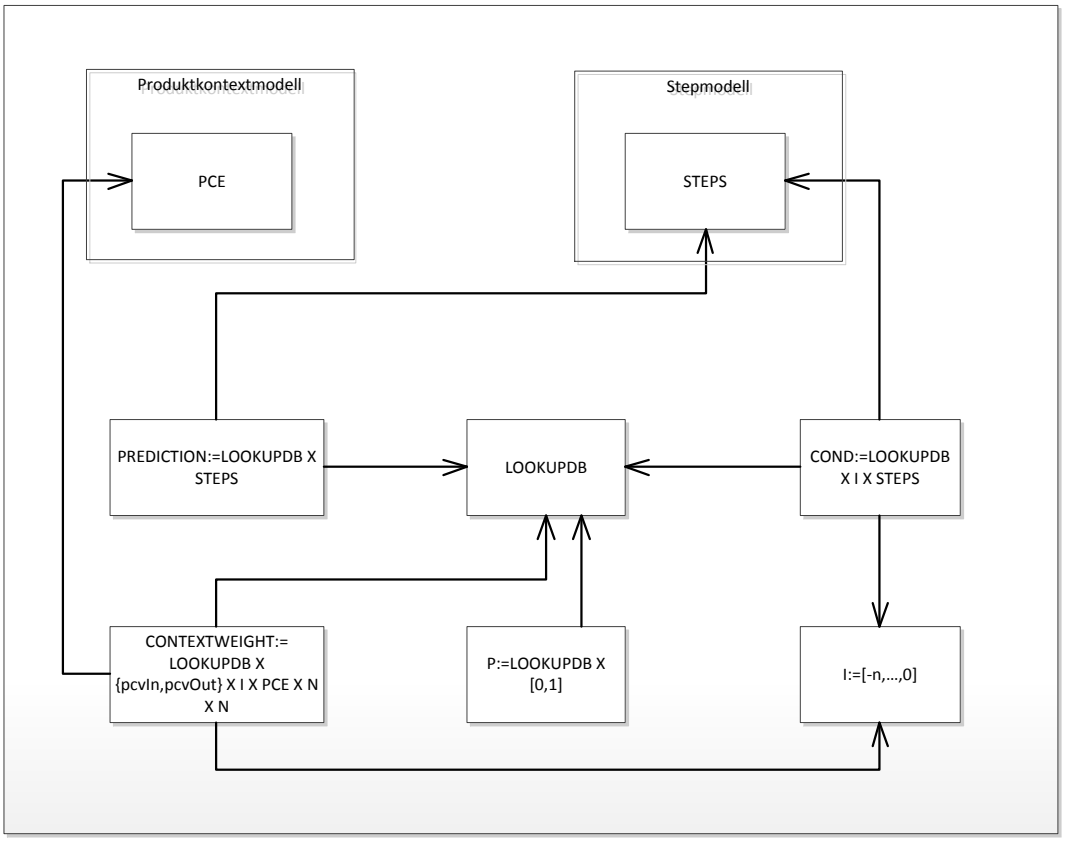


Abbildung 70: LookupDB in algebraischer Repräsentation

Zur einfachen Handhabung definieren wir eine Menge, die die Länge der Step-Beobachtungssequenzen der Elemente der LookupDB beschreibt:

Sei $LENCOND :=_{DEF} \{LOOKUPDB \times \mathbb{N}\}$ die Menge möglicher Zuordnungen von Elementen der LookupDB zu Längen der Beobachtungssequenzen. Für das LookupDB-Element aus Abbildung 61 gibt es z.B. folgendes Element in der Menge $LENCOND$: (2,3). *(89)*

Sei $ldb \in LOOKUPDB$ ein beliebiges Element. Dann beschreibt $lencond_{ldb}$ die Länge der Sequenz des Elements ldb.

Jedes Element der LookupDB besitzt einen Step, der die Vorhersage beschreibt. Dies entspricht bei einer bedingten Wahrscheinlichkeit $P(Prediction|Cond)$ der $Prediction$. Hierfür definieren wir die folgende Menge:

Sei $PREDICTION :=_{DEF} \{LOOKUPDB \times S\}$ die Menge möglicher Zuordnungen von Elementen der LookupDB zu Stepvorhersagen. Das Element aus Abbildung 61 wird z.B. beschrieben durch: (2,3). *(90)*

Sei $ldb \in LOOKUPDB$ ein beliebiges Element. Dann beschreibt $prediction_{ldb}$ die Stepvorhersage des Elements ldb.

Des Weiteren hat jedes LookupDB-Element eine Wahrscheinlichkeit P:

Sei $P :=_{DEF} \{LOOKUPDB \times [0,1]\}$ die Menge der möglichen Zuordnungen von Elementen der LookupDB zu Wahrscheinlichkeiten. $(2, 0.6)$ repräsentiert das Element der LookupDB in Abbildung 61. *(91)*

Sei $ldb \in LOOKUPDB$ ein beliebiges Element. Dann beschreibt p_{ldb} die Wahrscheinlichkeit des Elements ldb.

Die Kontextrelevanzwerte werden wie folgt als Menge repräsentiert:

Sei $CONTEXTWEIGHT :=_{DEF} \{LOOKUPDB \times \{pcvIn, pcvOut\} \times I \times PCE \times \mathbb{N} \times \mathbb{N}\}$ die Menge möglicher Kontextrelevanzwerte. Die Kontextrelevanzwerte an Index 0 für „pcvIn“ in Abbildung 61 wären zum Beispiel: $(2, pcvIn, 0{,}13{,}9{,}10)$ und $(2, pcvIn, 0{,}19{,}1{,}10)$. Die letzte Menge beschreibt folgendes: Bei dem LookupDB-Element mit der ID 2 wurde als Eingabe-Kontext ($pcvIn$) das Kontextexemplar mit der ID 19 in einem (1) von 10 Fällen beobachtet. *(92)*

Sei $cw \in CONTEXTWEIGHT$ ein beliebiges Element. Dann beschreibt id_{cw} die LookupDB-ID, cc_{cw} den „Ort“, an dem das Kontextexemplar beobachtet wurde (hier: pcvIn oder pcvOut), i_{cw} die relative Position der Sequenz, pce_{cw} das beobachtete Kontextexemplar, x_{cw} die Anzahl der beobachteten Kontextexemplare pce an der Position i und dem Ort cc sowie y_{cw} die Gesamtanzahl an (beliebigen) Kontextexemplaren an Position i und Ort cc. Beachte: x/y beschreibt die Wahrscheinlichkeit, dass pce an Position cc und i relevant ist.

Ein beispielhafter Ausschnitt einer LookupDB mit dessen algebraischer Repräsentation ist in Abbildung 71 dargestellt.

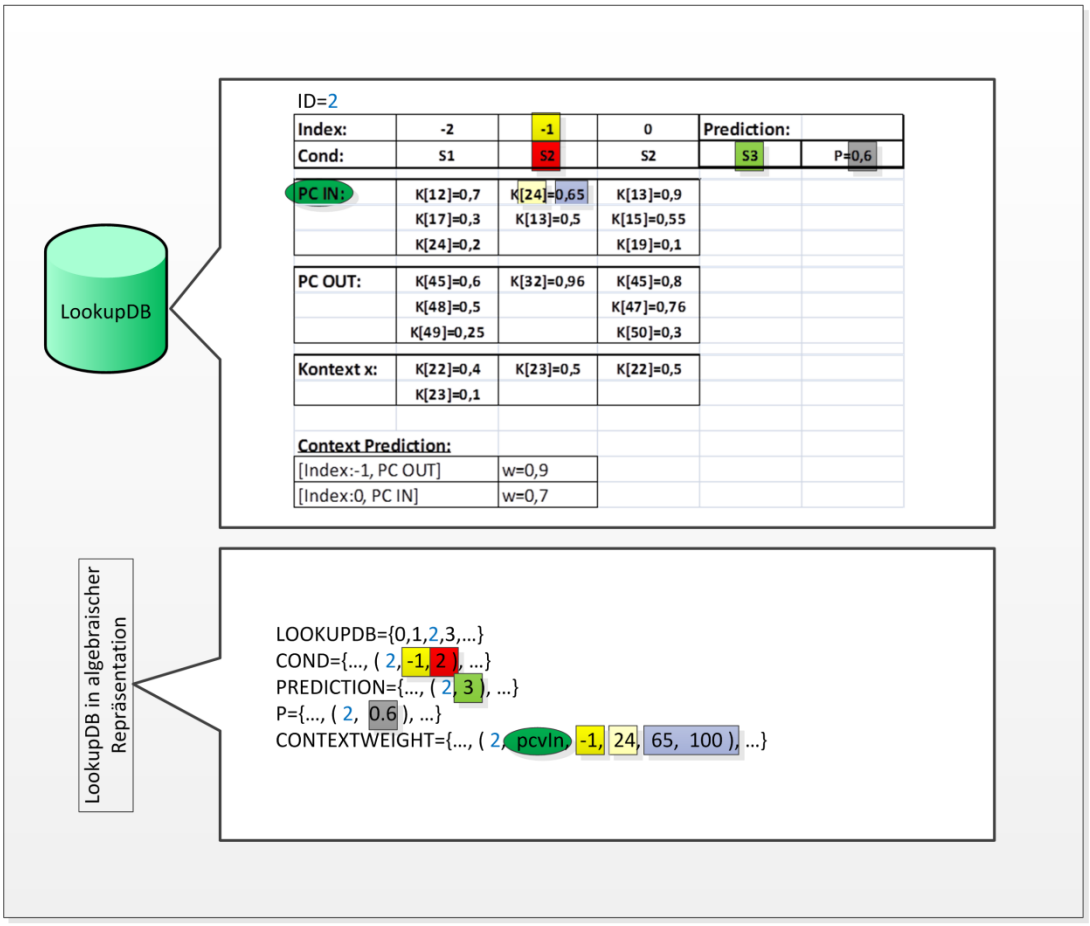


Abbildung 71: Beispielhafter Ausschnitt einer LookupDB

Zur einfachen Handhabung definieren wir die folgenden zwei Funktionen. Die erste sucht ein bestimmtes Element aus der Menge $CONTEXTWEIGHT$:

$$getCW :=_{DEF} LOOKUPDB \times \{pcvIn, pcvOut\} \times I \times PCE \rightarrow CONTEXTWEIGHT \tag{93}$$

$$getCW(ldb, cc, i, pce) := cw \text{ mit } cw \in CONTEXTWEIGHT \text{ und } id_{cw} = ldb \text{ und } cc_{cw} = cc \text{ und } i_{cw} = i \text{ und } pce_{cw} = pce$$

Ein Beispiel für diese Funktion ist in Abbildung 72 dargestellt.

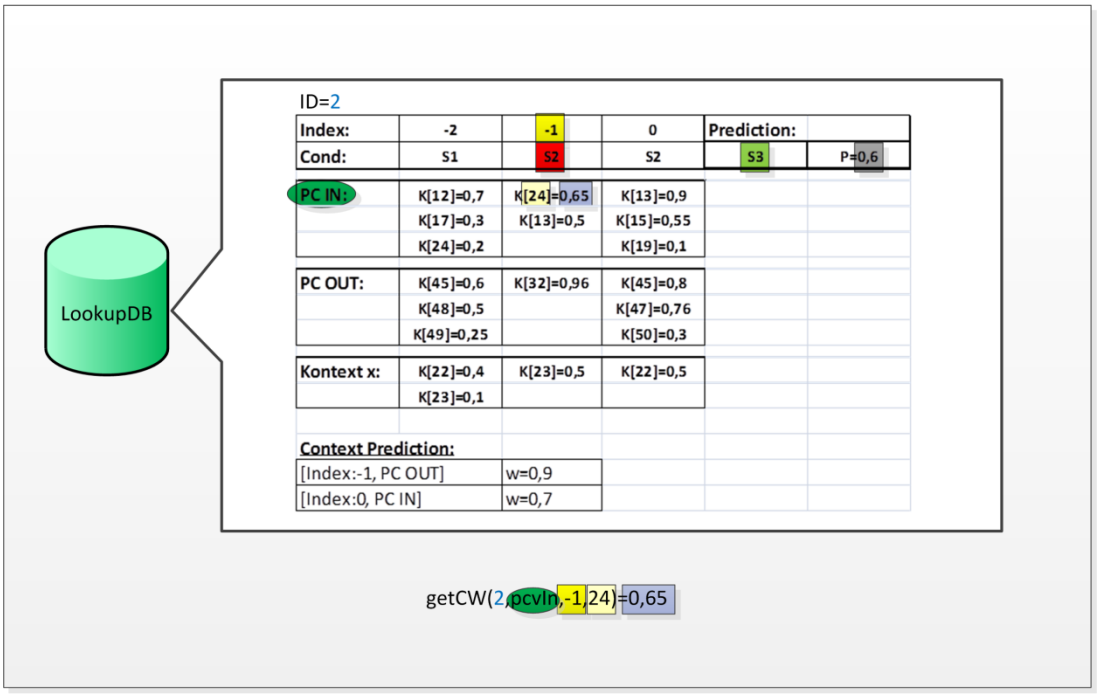


Abbildung 72: Beispiel für Funktion getCW

Nun definieren wir eine Funktion, die 1 zurück liefert, falls das übergebene Element der LookupDB zu der aktuellen Beobachtung passt. Dies ist der Fall, wenn die beobachtete Stepsequenz der gespeicherten Sequenz $cond$ entspricht:

$$match :=_{DEF} LOOKUPDB \times \mathbb{N} \rightarrow \{0,1\} \tag{94}$$

$$match(ldb, offset) := \begin{cases} 1, \text{ falls } \forall q \in [-lencond_{ldb}, 0]: cond_{ldb,q} = observation_s(q - offset) \\ 0, \text{ sonst} \end{cases}$$

Ein Beispiel dieser Funktion ist in Abbildung 73 dargestellt.

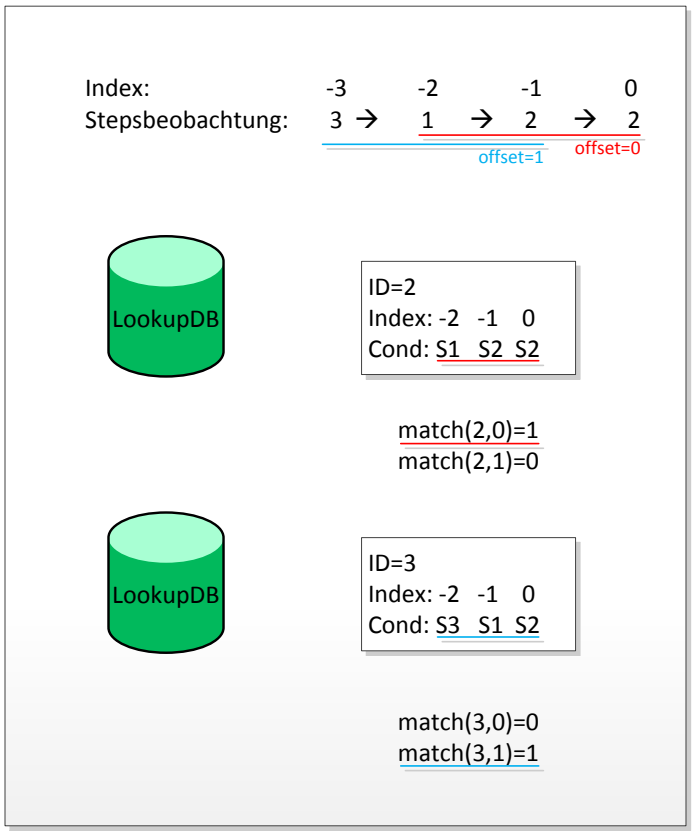


Abbildung 73: Beispiel der Funktion match

Im folgenden Unterkapitel 6.3.2 wird das Verfahren für die Vorhersage des nächsten Steps detailliert vorgestellt.

6.3.2 VORHERSAGE

Dieses Unterkapitel beschreibt das Verfahren, wie aus einer bestehenden LookupDB und einer aktuellen Beobachtungssequenz eine Vorhersage getroffen werden kann. Hierfür definieren wir vorerst zwei Funktionen:

$$getAMC :=_{DEF} \{(I \times \{pcvIn, pcvOut\} \times PCE \times [0,1]\} \rightarrow [0,1] \quad (95)$$

$getAMC(\{(i, cc, pce, w)\}) :=$ Mittelwert von allen $w > \theta$ (θ ist Schwellwert)

$$f :=_{DEF} LOOKUPDB \rightarrow \{(I \times \{pcvIn, pcvOut\} \times PCE \times [0,1]\} \quad (96)$$

$f(ldb) := \{(i, cc, c, w)\}$ mit: $i \in [-lc_{ldb}, 0]$ und $cc \in \{pcvIn, pcvOut\}$

und $c := observation_{cc}(i)$ für alle i und cc und $w := \frac{x_{cw}}{y_{cw}}$ mit

$cw := getCW(ldb, cc, i, c)$

Die Funktion f nimmt einen Eintrag ldb aus der LookupDB und gibt eine Menge von Elementen der Form (i, cc, c, w) zurück. w entspricht dem Kontextrelevanzwert des LookupDB-Elements ldb für jede Position i und für die möglichen cc ($pcvIn$ und $pcvOut$)

sowie für das entsprechende Kontextexemplar c der Beobachtung. Ein Beispiel hierfür ist in Abbildung 74 dargestellt.

Die Funktion $getAMC$ berechnet für eine Menge von Elementen der Form (i, cc, c, w) das arithmetische Mittel der enthaltenen w. Werden diese beiden Funktionen kombiniert aufgerufen, ($getAMC(f(ldb))$) so erhält man einen „Parameter“, inwieweit das Element ldb der LookupDB (und die darin enthaltenen Kontextrelevanzwerte) auf die aktuelle Beobachtung „passt“. Eine Beispielrückgabe für diese Funktion ist in Abbildung 75 dargestellt.

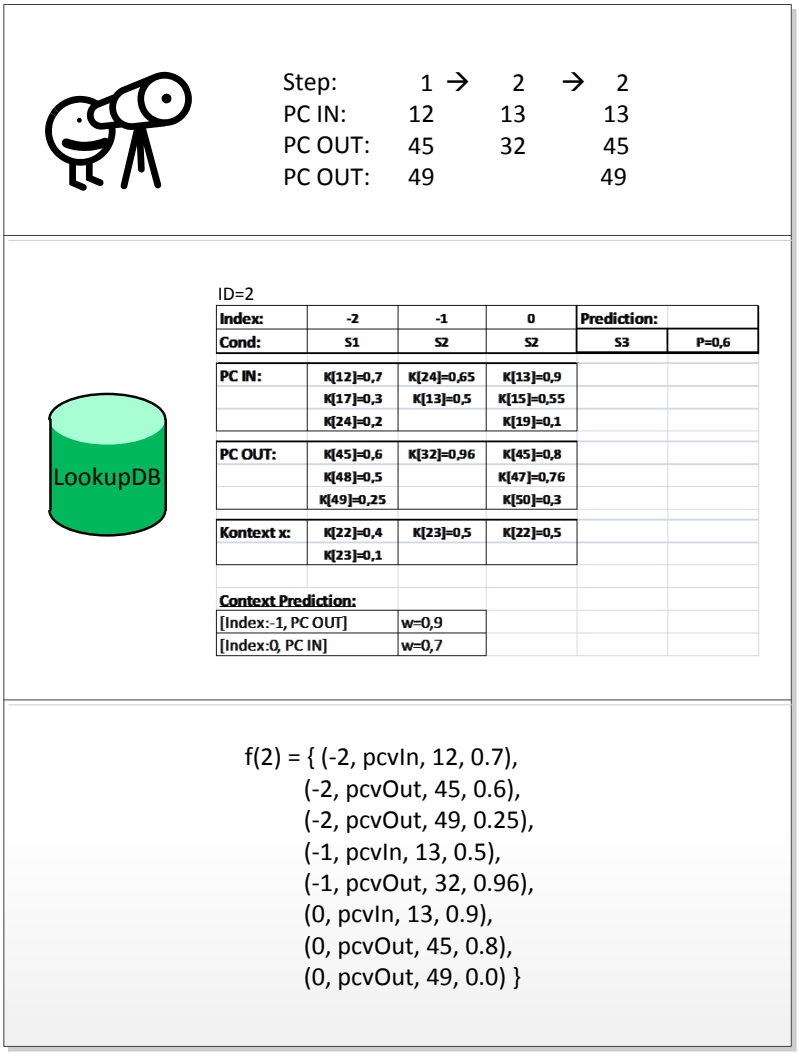


Abbildung 74: Beispielrückgabe der Funktion f

Die folgende Funktion berechnet zu einem Element der LookupDB einen aktuellen Wahrscheinlichkeitswert. Dieser beruht auf der Wahrscheinlichkeit P des Elements selbst sowie auf dem Parameter, der über die Funktion $getAMC(f(ldb))$ ermittelt wird und die aktuelle Beobachtung berücksichtigt:

$$getActualP :=_{DEF} LOOKUPDB \rightarrow [0,1] \qquad (97)$$

$$getActualP(ldb) := getAMC\big(f(ldb)\big) * prob_{ldb}$$

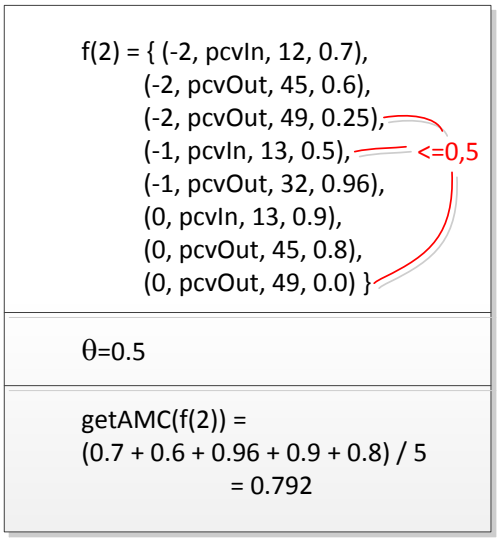


Abbildung 75: Beispielrückgabe der Funktion getAMC

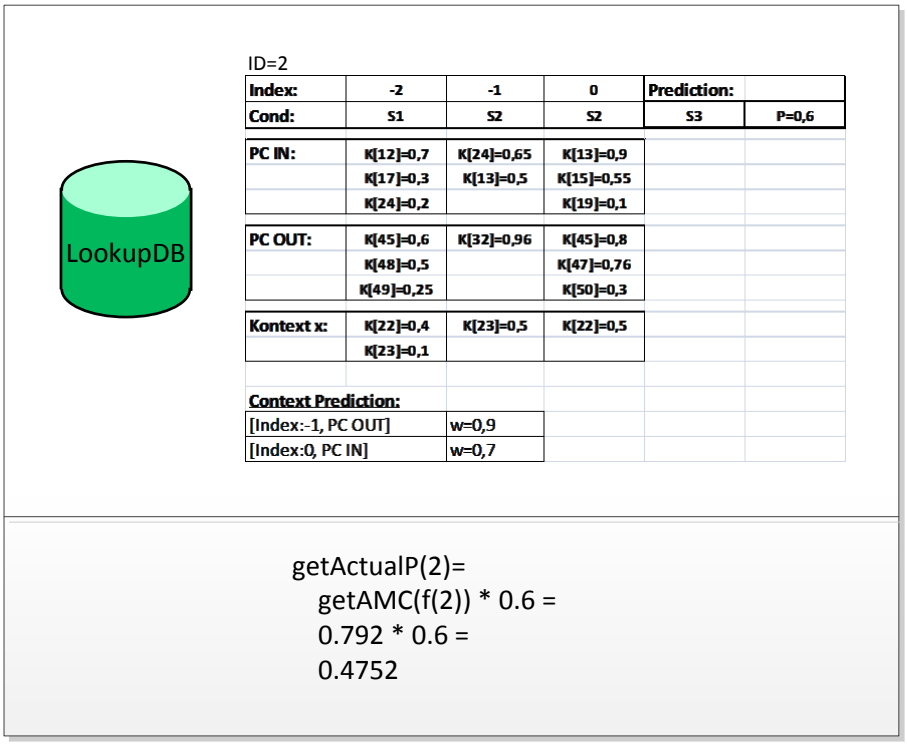


Abbildung 76: Beispielrückgabe der Funktion getActualP

Um nächste Steps vorzuschlagen, definieren wir eine Ordnungsrelation über die Elemente der LookupDB:

$$\geq :=_{DEF} \{(a,b) | a,b \in LOOKUPDB, getActualP(a) \geq getActualP(b)\} \quad (98)$$

Diese Ordnungsrelation hängt von dem aktuellen Wahrscheinlichkeitswert ab, den die Funktion $getActualP$ liefert.

Nun können wir eine Funktion beschreiben, die zu einer Menge von Elementen der LookupDB die wahrscheinlich nächsten Steps vorhersagt:

$$makePrediction :=_{DEF} Pow(LOOKUPDB) \rightarrow STEPS \qquad (99)$$

$$makePrediction(ldb) := s$$

Sei $l \in ldb$ mit $l \geq m$ für alle $m \in ldb$. Dann ist $s := s_{ldb}$

Alternativ wäre es auch möglich, dass diese Funktion eine Menge von Steps zurück liefert und nicht nur den Step mit der größten aktuellen Wahrscheinlichkeit. Dies wären dann beispielsweise die fünf Steps mit der größten aktuellen Wahrscheinlichkeit, die die Funktion $getActualP$ liefert. Diese Menge von Steps lässt sich aus der Ordnung, definiert in (98), bestimmen. An dieser Stelle wird darauf jedoch verzichtet, da der Prototyp, mit dem die Evaluierungen aus Kapitel 7 und 8 durchgeführt wurden, nur den wahrscheinlichsten Step vorschlägt. Der Grund hierfür ist, dass die Evaluierungssequenzen nur eine begrenzte Menge an unterschiedlichen Steps enthalten (deutlich unter 10).

Um den nächsten Step vorherzusagen, wird die Funktion $makeprediction$ aufgerufen mit einer Menge von Elementen $ldb \in LOOKUPDB$ für die gilt: $match(ldb, 0) = 1$.

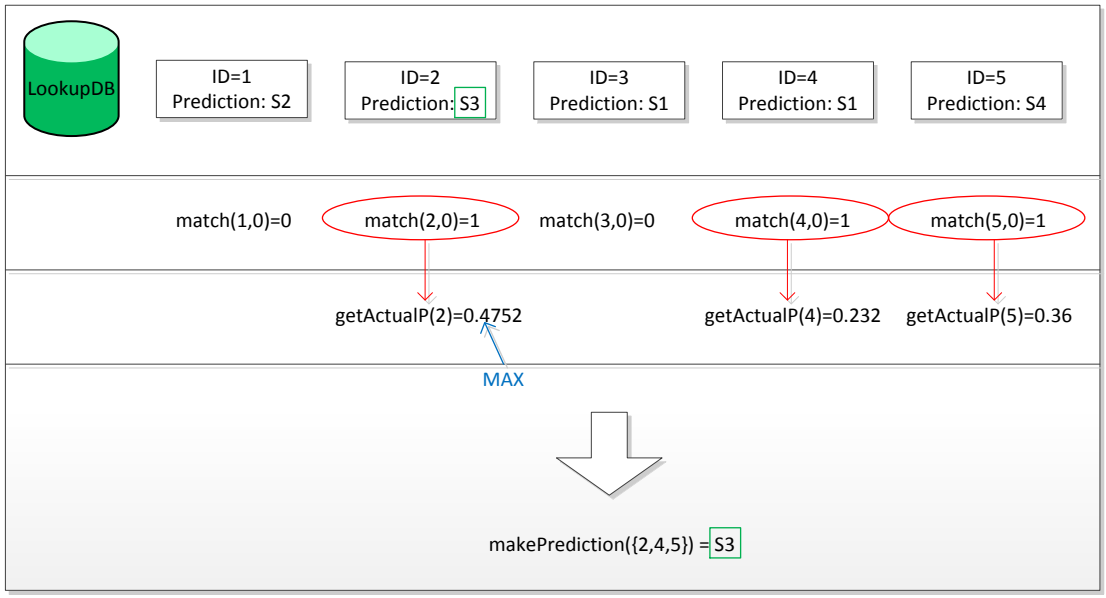


Abbildung 77: Beispiel der Funktion makePrediction

6.3.3 LERNEN

Nach der Vorhersage eines Steps wird der Projektmitarbeiter einen Step auch tatsächlich starten. Nach diesem Starten wird die LookupDB aktualisiert, d.h. das Verfahren wird durch die Erfahrungsdaten angepasst. (Zu beachten: $observation_s(0)$ beschreibt den Step, für den eine Vorhersage gemacht wurde und den wir nun lernen wollen.)

Falls kein Element $ldb \in LOOKUPDB$ existiert mit

- $len_{ldb} = 1$,
- $cond_{ldb,0} = observation_s(-1)$ und
- $prediction_{ldb} = observation_s(0)$,

wird ein neues Element zu der LookupDB hinzugefügt:

$$addEntry :=_{DEF} \{LOOKUPDB\} \times \{COND\} \times \{LENCOND\} \times \{PREDICTION\} \times \{P\} \rightarrow \{LOOKUPDB\} \times \{COND\} \times \{LENCOND\} \times \{PREDICTION\} \times \{P\} \quad (100)$$

$$addEntry(ldb_0, cond_0, lc_0, pre_0, p_0) :=_{DEF} (ldb_1, cond_1, lc_1, pre_1, p_1)$$

a. $ldb_1 \backslash ldb_0 = \emptyset, cond_1 \backslash cond_0 = \emptyset, lc_1 \backslash lc_0 = \emptyset, pre_1 \backslash pre_0 = \emptyset, p_1 \backslash p_0 = \emptyset$

b. $|ldb_0| + 1 = |ldb_1|, |cond_0| + 1 = |cond_1|, |lc_0| + 1 = |lc_1|, |pre_0| + 1 = |pre_1|, |p_0| + 1 = |p_1|$

c. $n \notin ldb_0$ und $n \in ldb_1$

d. $\big(n, 0, observation_s(-1)\big) \notin cond_0$ und $\big(n, 0, observation_s(-1)\big) \in cond_1$

e. $(n, 1) \notin lc_0$ und $(n, 1) \in lc_1$

f. $\big(n, observation_s(0)\big) \notin pre_0$ und $\big(n, observation_s(0)\big) \in pre_1$

g. $(n, 1 - \alpha) \notin p_0$ und $(n, 1 - \alpha) \in p_1$

Anschaulich geschrieben wird die o.g. Funktion aufgerufen, falls nach einer Beobachtung eine entsprechende Teilsequenz der Länge 2 nicht existiert. Mit anderen Worten: Wird beispielsweise $\dots a \rightarrow b$ beobachtet, so würde obige Funktion die bedingte Wahrscheinlichkeit $P(b|a)$ der LookupDB hinzufügen.

Die Änderungen an den o.g. Mengen durch die Funktion $addEntry$ ist in Abbildung 78 beispielhaft dargestellt.

Sei $LDB :=_{DEF} \{l \in LOOKUPDB | match(l, 1) = 1$ und $(l, observation_s(0)) \in PREDICTION\}$. Für jedes Element $el \in LDB$ werden die folgenden zwei Funktionen aufgerufen:

Die Funktion $updateP$ aktualisiert die Wahrscheinlichkeitswerte P. Diese Aktualisierungsfunktion beruht auf dem Kernverfahren IPAM. Dieses wurde bereits in Kapitel 6.1 auf Seite 99 vorgestellt.

$$updateP :=_{DEF} \{P\} \times LOOKUPDB \rightarrow \{P\} \quad (101)$$

$updateP(p_0, el) := p_1$ mit:

a. $(el, pp) \in p_0$ und $(el, pp) \notin p_1$ und $\big(el, \alpha * pp + (1 - \alpha)\big) \in p_1$ und $p_0 \backslash \{(el, pp)\} = p_1 \backslash \{(el, \alpha * pp + (1 - \alpha))\}$

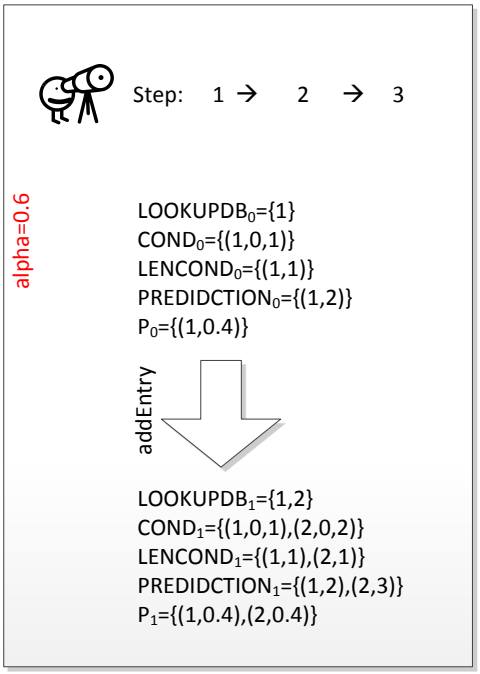


Abbildung 78: Beispiel der Funktion addEntry

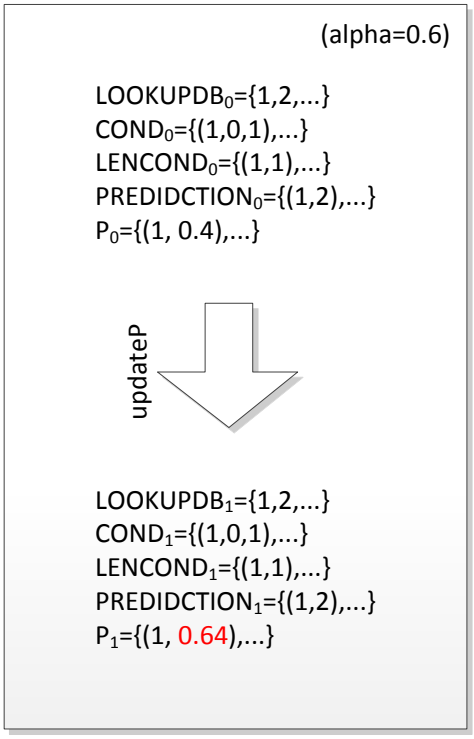


Abbildung 79: Beispiel der Funktion updateP

Ein Aufruf dieser Funktion ist in Abbildung 79 beispielhaft dargestellt. Die zweite Funktion aktualisiert die Kontextrelevanzwerte:

$$updateContextweight :=_{DEF} \{CONTEXTWEIGHT\} \times LOOKUPDB \rightarrow \{CONTEXTWEIGHT\} \quad (102)$$

$$updateContextweight(cw_0, el) := cw_1$$

a. Für alle $(ldb, cc, i, c, x, y) \in cw_0$ mit $ldb \neq el \Rightarrow (ldb, cc, i, c, x, y) \in cw_1$

b. Für alle $(ldb, cc, i, c, x, y) \in cw_0$ mit $ldb = el$ und $c = observation_{cc}(i)$
$\Rightarrow (ldb, cc, i, c, x, y) \notin cw_1$ und $(ldb, cc, i, c, x + 1, y + 1) \in cw_1$

c. Für alle $(ldb, cc, i, c, x, y) \in cw_0$ mit $ldb = el$ und $c \neq observation_{cc}(i)$
$\Rightarrow (ldb, cc, i, c, x, y) \notin cw_1$ und $(ldb, cc, i, c, x, y + 1) \in cw_1$

Diese Funktion aktualisiert alle zum LookupDB-Element el „passenden" Kontextrelevanzwerte. Entspricht der darin enthaltene Kontext c der aktuellen Beobachtung an der richtigen Stelle, so wird die Wahrscheinlichkeit erhöht (siehe b; $x = x + 1$ und $y = y + 1$). Entspricht der enthaltene Kontext nicht der aktuellen Beobachtung, wird die Wahrscheinlichkeit vermindert (siehe c; $y = y + 1$).

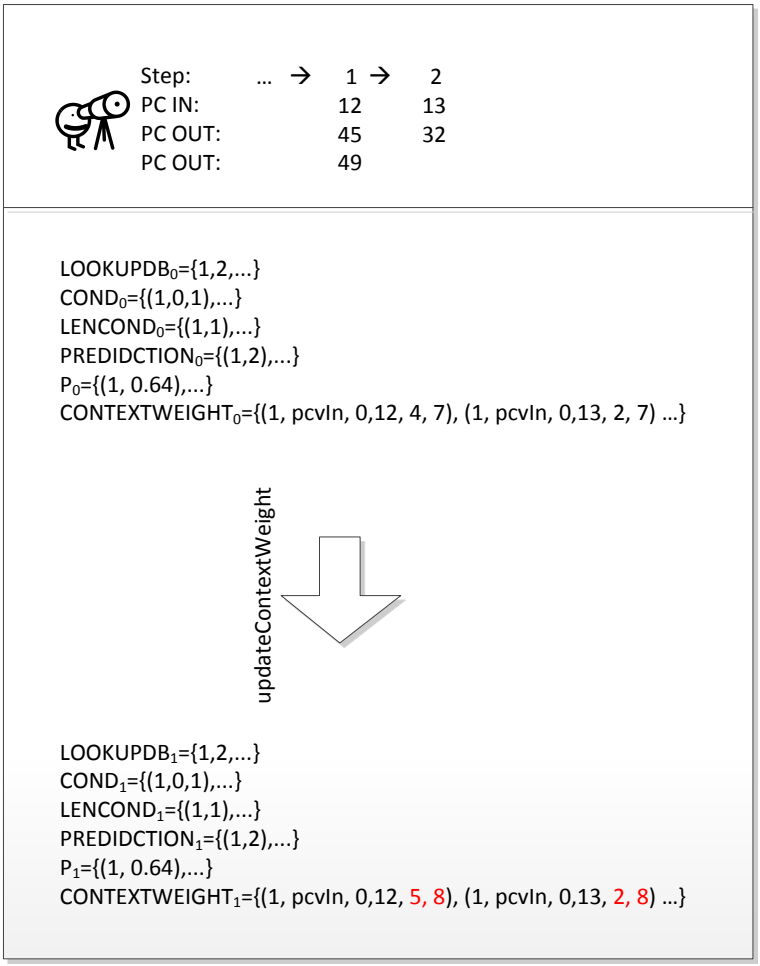


Abbildung 80: Beispiel der Funktion updateContextWeight

Falls die letzte Vorhersage korrekt war, werden neue Elemente zu der LookupDB hinzugefügt. Hierfür definieren wir zunächst die zwei Teilmengen Q und L:

$$Q :=_{DEF} \{q \in LOOKUPDB | cond_{q,i} = observation_s(i-1), \quad \forall i \in [lenCond_q, 0]\} \tag{103}$$

$$L :=_{DEF} \{l \in LOOKUPDB | cond_{l,i} = observation_s(i), \forall\, i \in [lenCond_l, 0]\} \tag{104}$$

Seien $ll \subseteq L$ die Elemente mit der längsten Sequenz ($lenCond$) und $P_{ll} > 0$. Nun können wir uns die Funktion $updateLOOKUPDB$ definieren, die neue Elemente zu der LookupDB hinzufügt. Diese Funktion entspricht dem Update-Algorithmus des Kernverfahrens von Jacob/Blockeel, vorgestellt in Kapitel 6.1 auf Seite 99.

$$updateLOOKUPDB \tag{105}$$

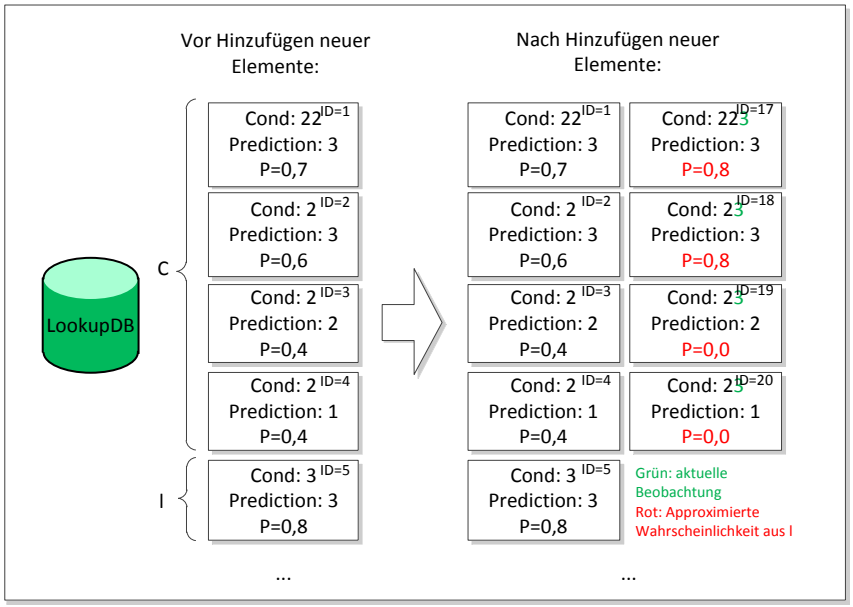
$$\in (\{LOOKUPDB\} \times Q \times \{ll\} \times \{COND\} \times \{LENCOND\} \times \{PREDICTION\} \times \{P\} \times \{CONTEXTWEIGHT\})$$

$$\rightarrow (\{LOOKUPDB\} \times \{COND\} \times \{LENCOND\} \times \{PREDICTION\} \times \{P\} \times \{CONTEXTWEIGHT\})$$

$$updateLOOKUPDB(LDB_0, q, ll, c_0, lc_0, pre_0, p_0, cw_0) := (LDB_1, c_1, lc_1, pre_1, p_1, cw_1)$$ mit:

a. $\{0,1,\dots,n\} \in LDB_0$ und $\{0,1,\dots,n,\dots m\} \in LDB_1$ mit $|LDB_0| + |q| = |LDB_1|$
b. Sei *ldiff* $= LDB_1 \backslash LDB_0$. $\forall qq \in q\ \exists!\ ldiffel \in lediff$ mit:
   - $(q, i, s) \in c_0$ und $(ldiffel, i-1, s) \in c_1$ für alle i und s; und: $(ldiffel, 0, oberservation(0)) \in c_1$
   - $(q, l) \in lc_0$ und $(ldiffel, l+1) \in lc_1$
   - $(q, s) \in pre_0$ und $(ldiffel, s) \in pre_1$ für alle s
   - $(q, p) \in p_0$ und $(ll, p2) \in p_0$ und $(ldiffel, p2) \in p_1$

Die obige Funktion fügt neue Einträge zu der LookupDB hinzu. Sie verwendet hierfür die vorhandenen Einträge, die die aktuelle Beobachtung ($observation_s(0)$) korrekt vorhergesagt haben, und erweitert die darin enthaltenen Stepsequenzen. Die Wahrscheinlichkeiten der neuen Einträge werden geschätzt, indem hierfür die Wahrscheinlichkeiten der Menge ll verwendet werden. Ein Beispiel für die Arbeitsweise dieser Funktion ist in Abbildung 81, konsistent zu Abbildung 68 auf Seite 109, dargestellt.



$LOOKUPDB_0$={1, 2, 3, 4, 5, ...}
$COND_0$={(1, 0, 2), (1, -1, 2), (2, 0, 2), (3, 0, 2), (4, 0, 2), (5, 0, 3),...}
$LENCOND_0$={(1, 2), (2, 1), (3, 1), (4, 1), (5, 1),...}
$PREDIDCTION_0$={(1, 3), (2, 3), (3, 2), (4, 1), (5, 3),...}
P_0={(1, 0.7), (2, 0.6), (3, 0.4), (4, 0.4), (5, 0.8),...}

updateLOOKUPDB

$LOOKUPDB_0$={1, 2, 3, 4, 5, ..., 17, 18, 19, 20}
$COND_0$={(1, 0, 2), (1, -1, 2), (2, 0, 2), (3, 0, 2), (4, 0, 2), (5, 0, 3), ...,
(17, 0, 3), (17, -1, 2), (17, -2, 2), (18, 0, 3), (18, -1, 2), (19, 0, 3), (19, -1, 2), (20, 0, 3), (20, -1, 2)}
$LENCOND_0$={(1, 2), (2, 1), (3, 1), (4, 1), (5, 1),..., (17, 3), (18, 2), (19, 2), (20, 2)}
$PREDIDCTION_0$={(1, 3), (2, 3), (3, 2), (4, 1), (5, 3), ..., (17, 3), (18, 3), (19, 2), (20, 1)}
P_0={(1, 0.7), (2, 0.6), (3, 0.4), (4, 0.4), (5, 0.8), ..., (17, 0.8), (18, 0.8), (19, 0.0), (20, 0.0)}

Abbildung 81: Beispiel der Funktion updateLOOKUPDB

7 BEWERTUNG UND ERGEBNISSE

Dieses Kapitel beschreibt die Bewertung des in Kapitel 6 spezifizierten Verfahrens zur Unterstützung des Projektmitarbeiters, indem es mit synthetischen Beispieldaten evaluiert wird. Die Beispieldaten sind Ablaufsequenzen von Arbeitsschritten, wie sie Projektmitarbeiter in einem IT-Projekt durchführen könnten. Die dahinter liegende Methodik, also die generellen Vorgehensweisen, ist aus der Methodik in Kapitel 3 abgeleitet. Das hier evaluierte Verfahren wird mit dem zugrunde liegenden Kernverfahren von Jacobs und Blockeel [64] verglichen. Dieses Kernverfahren wurde bereits in Kapitel 6.1 vorgestellt.

Das folgende Unterkapitel beschreibt kurz die zugrunde liegende Methodik, die zur Ableitung der Testdaten gedient hat, sowie die eigentlichen Testdaten in Form von Ablaufsequenzen. In Kapitel 7.2 wird die prototypische Implementierung unseres Verfahrens kurz vorgestellt. Mit dieser Implementierung wurde die synthetische Evaluierung durchgeführt, deren Ergebnisse in Kapitel 7.3 dargestellt werden.

7.1 METHODIK UND ABLAUFSEQUENZEN

In Kapitel 3 wird eine Entwurfsmethodik für IT-Projekte vorgestellt, die durchgängig in der gesamten Arbeit aufgegriffen wird. Die beschriebene Methodik fokussiert auf den Systementwurf und die Spezifikation von Systemelementen. Die Anforderungen an ein System sind bereits vorhanden. Ziel ist es, eine Systemarchitektur, bestehend aus Systemelementen (z.B. Komponenten), zu erstellen und die Anforderungen auf diese Elemente abzubilden. Jedes identifizierte Systemelement wird nachfolgend spezifiziert. In diesen Spezifikationen werden die Schnittstellen der Systemelemente detailliert beschrieben und implementiert. Nachdem eine Spezifikation zu einem Systemelement erstellt worden ist, kann dieses implementiert werden.

Die oben beschriebene Methodik wurde in vier Arbeitsschritte unterteilt:

1. Komponente identifizieren
2. Anforderung auf Komponente mappen
3. Komponente spezifizieren
4. Komponente implementieren

Die Beschreibung der Methodik (oben und auch in Kapitel 3.1) gibt keinen konkreten Ablauf vor, in welcher Reihenfolge die Arbeitsschritte durchzuführen sind. Solch eine Einschränkung ist auch gar nicht gewollt, da in dieser Arbeit die Motivation verfolgt wird, dass der Projektmitarbeiter in seiner Arbeitsweise nicht eingeschränkt, sondern unterstützt wird (siehe hierzu Kapitel 1). Deshalb wurden in Kapitel 3.2 auch zwei mögliche Prozessabläufe beispielhaft skizziert. Diese beispielhaften Prozessabläufe sind:

1. Prototypischer Ablauf: Ein Projektmitarbeiter wendet dies meist bei komplexen Komponenten an. Nachdem eine (komplexe) Komponente identifiziert ist, wird eine Anforderung auf diese Komponente abgebildet. Danach wird eine entsprechende Spezifikation erstellt und der entsprechende Teil der Komponente implementiert. Nach dieser Implementierung wird die nächste Anforderung abgebildet, spezifiziert und implementiert usw. Beim prototypischen Ablauf wird für jede Anforderung ein „Durchstich“ bis zur Implementierung gemacht.

2. Breiter Entwurf: Diese Ablaufstrategie wird bei Komponenten, deren Klassen eine hohe Kopplung haben, angewendet. Nachdem eine Komponente identifiziert ist, werden alle relevanten Anforderungen auf die Komponente abgebildet (Arbeitsschritt „Anforderung auf Komponente mappen“ wird mehrmals hintereinander ausgeführt). Nachdem alle relevanten Anforderungen identifiziert sind, werden die entsprechenden Spezifikationen erstellt („Komponente spezifizieren“ wird mehrmals hintereinander ausgeführt). Danach wird die Komponente im Ganzen implementiert („Komponente implementieren“ wird ebenfalls mehrfach hintereinander ausgeführt).

Zur Bewertung unseres in Kapitel 6 vorgestellten Verfahrens zur Nutzerunterstützung werden aus diesen zwei möglichen Abläufen drei Szenarien abgeleitet. Jedes Szenario besteht dabei aus einer Sequenz an Arbeitsschritten und den dazugehörigen Kontextinformationen. Kontexte und Produktkontexte werden in den Kapiteln 4.4.2 und 4.4.3 ab Seite 49 im Überblick vorgestellt. In Kapitel 5.2 ab Seite 66 sind Produktkontexte im Detail spezifiziert. Die drei Sequenzen unterscheiden sich dabei folgendermaßen:

1. Entwurf, Spezifikation und Implementierung von Komponenten mit hoher Kopplung. In diesem Szenario werden nur Komponenten mit hoher Kopplung entwickelt, d.h. hier wird lediglich der breite Entwurf angewendet.

2. Entwurf, Spezifikation und Implementierung von komplexen Komponenten. In diesem Szenario werden nur komplexe Komponenten entwickelt. Hier wird also der prototypische Ablauf angewendet.

3. Entwurf, Spezifikation und Implementierung von komplexen Komponenten und Komponenten mit hoher Kopplung. Dieses Szenario entspricht am ehesten einem realen Projektablauf. Hier werden beide Arten von Komponenten entwickelt. Die Reihenfolge der Komponenten-Arten ist dabei zufällig gewählt.

Zu allen Arbeitsschrittsequenzen der drei Szenarien werden zusätzlich die Kontextinformationen beschrieben. Dies ist dadurch begründet, dass unser Verfahren, welches in Kapitel 6 vorgestellt wird, diese Kontextinformationen nutzt und verarbeitet (siehe z.B. den entsprechenden Bereich der LookupDB (92), Seite 113 und die Funktion für die Vorhersage (95), (96), Seite 116). Die Arbeitsschritte und Kontexte bei den Sequenzen werden mit eindeutigen IDs kodiert (siehe Abbildung 82). In diesem Beispiel repräsentiert die ID mit der Nummer 38 den Arbeitsschritt „Komponente identifizieren“, Nummer 39 repräsentiert „Anforderung auf Komponente mappen“ und Nummer 40 repräsentiert „Komponente spezifizieren“.

Arbeitsschritte	38	39	39	39	40	40	40	...
Kontext	48	49	49	49	49	49	49	...
Kontext	49				50	51	52	...
Kontext	33	34	34	34	34	34	34	...

Abbildung 82: Teil einer Beispielsequenz

Alle drei Szenarien lassen sich in Teilsequenzen unterteilen (siehe hierfür Abbildung 83 bis Abbildung 84). Die ersten beiden Teilsequenzen unterscheiden sich von den restlichen: In diesen wird jeweils eine Komponente erzeugt (Arbeitsschritt „Komponente identifizieren“), die dann weiterentwickelt wird („Anforderung auf Komponente mappen“, „Komponente spezifizieren“, „Komponente implementieren“). Die nachfolgenden Teilsequenzen entwickeln dann die anfangs

identifizierten Komponenten weiter. Hier wird der Arbeitsschritt „Komponente identifizieren“ nicht mehr ausgeführt. Die Farben der Teilsequenzen in den Abbildungen markieren die Komponente, die entwickelt bzw. weiterentwickelt wird. Bei den unterschiedlichen Farben sind insbesondere auch unterschiedliche Kontextinformationen enthalten.

Bei Teilsequenzen mit gleicher Farbe unterscheiden sich manche Kontextinformationen nicht (z.B. in Abbildung 83: Arbeitsschritt 39 hat immer Kontext [49, 34]), andere Kontexte werden bei jedem Durchlauf neu erzeugt und verwendet. „49“ spiegelt beispielsweise den Komponentenkontext einer Komponente wider, auf die eine Anforderung abgebildet wird. Im Arbeitsschritt „Komponente spezifizieren“ hingegen wird eine neue Spezifikation erstellt. Deshalb wird hier ein neuer (Spezifikations-)Kontext erzeugt. Dieser wird in den Abbildungen mit „neuer Kontext“ umschrieben. Die darauf folgenden Arbeitsschritte „Komponente implementieren“ nutzen die „neuen“ Kontexte. Dies wird auch in den Abbildungen kenntlich gemacht.

In Abbildung 83, links ist das Szenario 1, also der Entwurf, die Spezifikation und die Implementierung von Komponenten mit hoher Kopplung dargestellt. In den detaillierten Ausschnitten erkennt man, dass immer mehrere Anforderungen auf die Komponente abgebildet werden (Arbeitsschritt „Anforderung auf Komponente mappen“ (ID: 39) wird mehrmals hintereinander ausgeführt). Danach werden für die neu abgebildeten Anforderungen Spezifikationen erstellt (mehrmaliges Ausführen von Arbeitsschritt „Komponente spezifizieren“). Abschließend werden die spezifizierten Elemente implementiert (mehrmaliges Ausführen von „Komponente implementieren“).

Das Szenario 2, also Entwurf, Spezifikation und Implementierung von komplexen Komponenten ist in Abbildung 83, rechts dargestellt. Der Unterschied zu Szenario 1 ist, dass hier jede Anforderung einzeln spezifiziert und implementiert wird. Das Szenario beinhaltet also nicht das mehrmalige, sequentielle Ausführen der gleichen Arbeitsschritte, wie dies in Szenario 1 der Fall ist.

Abschließend ist in Abbildung 84 das Szenario 3 abgebildet. Dieses Szenario beinhaltet wie die beiden vorherigen Szenarien die Entwicklung und Weiterentwicklung von zwei Komponenten. Der Unterschied ist jedoch, dass eine Komponente eine hohe Kopplung hat. Hier wird also der breite Entwurf angewendet (wie Szenario 1). Die andere Komponente ist komplex, deshalb wird diese prototypisch entwickelt (wie Szenario 2).

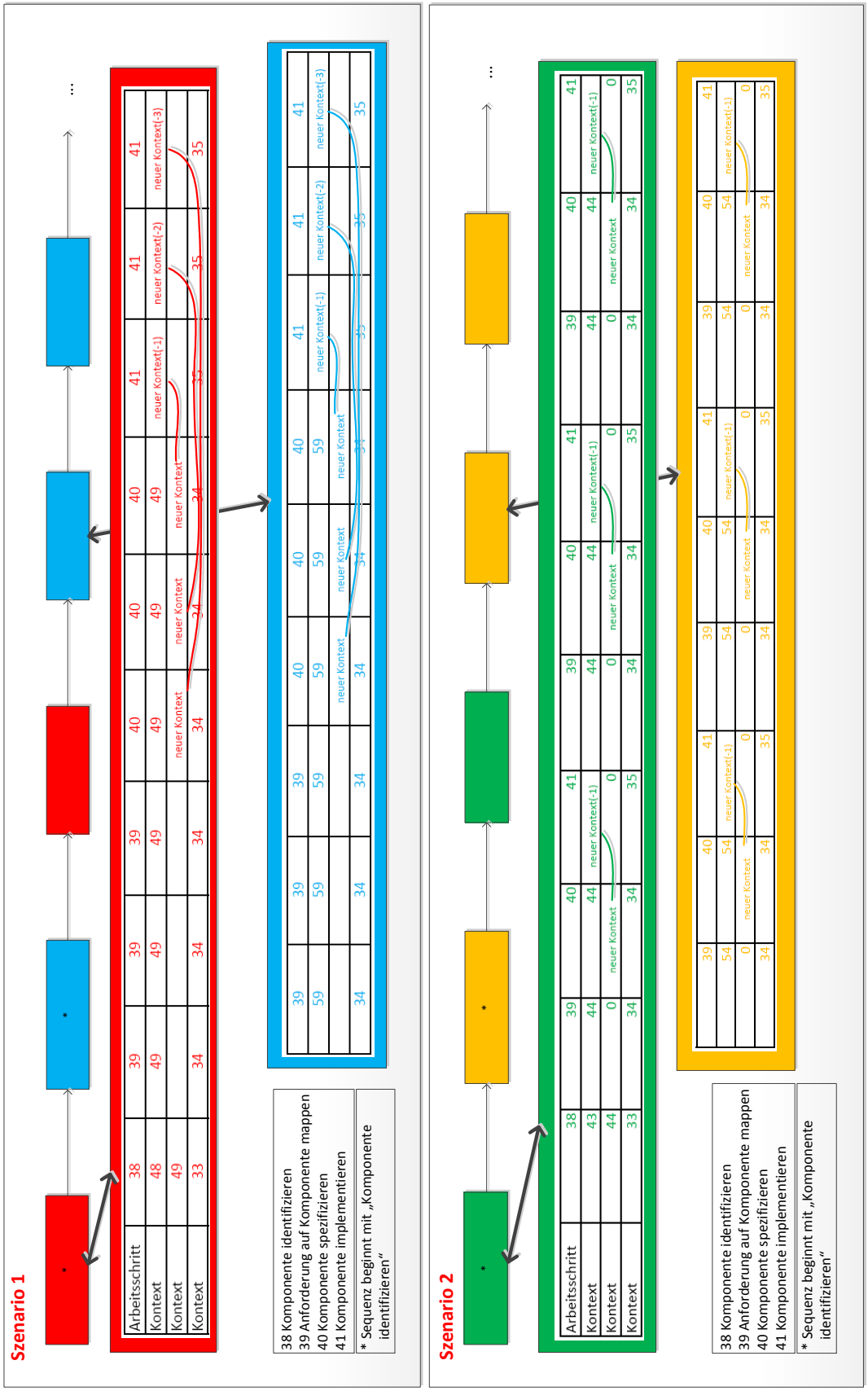


Abbildung 83: Szenario 1: Entwurf, Spezifikation und Implementierung von Komponenten mit hoher Kopplung und Szenario 2: Entwurf, Spezifikation und Implementierung von komplexen Komponenten

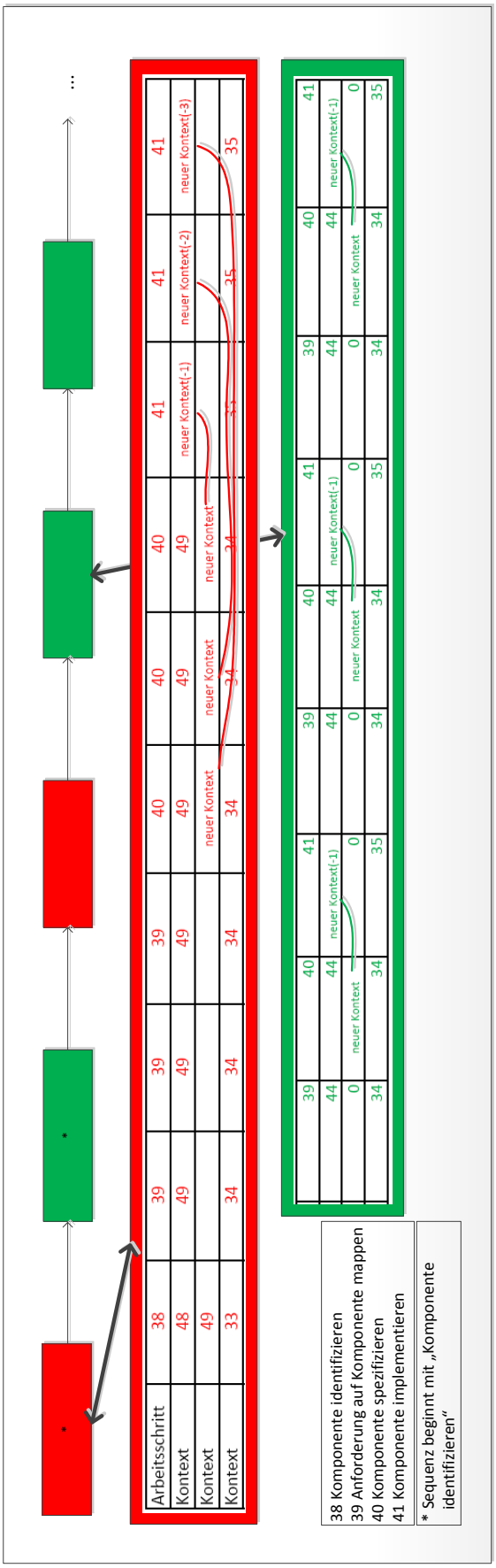


Abbildung 84: Szenario 3: Mischung aus Szenario 1 und 2

7.2 PROTOTYP

Das in Kapitel 6 vorgestellte und spezifizierte Verfahren zur Nutzerunterstützung wurde prototypisch implementiert. Das Ziel ist hier insbesondere die Bewertung des Verfahrens mit den oben beschriebenen Szenarien. Bei der Implementierung des Prototyps wurde auf eine spezifikationsnahe Implementierung Wert gelegt. Die Datenstrukturen, also die in Kapitel 6.3 beschriebenen Mengen und Operationen auf den Datenstrukturen in Form von Funktion wurden wie spezifiziert implementiert. Dabei ist die Implementierung als Prototyp zur Bewertung des Verfahrens anzusehen, da keine Optimierung der Datenstrukturen vorgenommen wurde. Dies ist beispielhaft in Abbildung 85 dargestellt: Die Funktion $getCW$ liefert ein Element der Menge $contextweight$. In der Implementierung, in der Abbildung rechts dargestellt, wird die Menge $contextweight$ sequentiell durchsucht, bis das gesuchte Element gefunden wurde. Für eine produktive Implementierung würden die Datenstrukturen optimiert werden, um eine effizientere Suche zu ermöglichen.

Zur Bewertung unseres Verfahrens nimmt der Prototyp eine Beispielsequenz als Eingabe und führt jeden Arbeitsschritt mit den dazugehörenden Kontextinformationen aus. Nach jeder Eingabe aktualisiert der Prototyp die Erfahrungsdatenbank und schlägt einen Arbeitsschritt, der als nächstes gestartet werden soll, vor. Der vorgeschlagene Arbeitsschritt wird dann mit der nächsten Eingabe verglichen. Für die Auswertung wird zu jedem Arbeitsschritt gespeichert, ob er korrekt vorgeschlagen wurde oder nicht.

Dieser Prototyp schlägt immer nur einen Arbeitsschritt vor. Das Konzept (siehe Kapitel 4 auf Seite39) und die Spezifikation, also die detaillierte Beschreibung des Verfahrens (siehe Kapitel 6 auf Seite 99), sehen es jedoch vor, mehrere „wahrscheinliche“ Arbeitsschritte vorzuschlagen. Da die Beispielsequenzen aber nur aus vier Arbeitsschritten bestehen, wurde hiervon in der Implementierung abgesehen. Das spezifizierte Verfahren erlaubt es jedoch problemlos, mehrere „wahrscheinliche“ Arbeitsschritte vorzuschlagen (siehe (98) und (99) auf Seite 118).

Spezifikation (aus Kap. 7)

$LOOKUPDB :=_{DEF} \{0,1,2,\dots\}$

...

$getCW :=_{DEF} LOOKUPDB \times \{pcvIn, pcvOut\} \times I \times PCE \rightarrow CONTEXTWEIGHT$¶

$getCW(ldb, cc, i, pce) := cw$ mit $cw \in CONEXTWEIGHT$ und $id_{cw} = ldb$ und $cc_{cw} = cc$ und $i_{cw} = i$ und $pce_{cw} = pce$¶

...

Prototypische Implementierung

```
self.LOOKUPDB=[]
self.cond=[]
self.lenCond=[]
self.prediction=[]
self.P=[]
self.contextweight=[]

        ...

def getCW(self,ldb,cc,i,pce):
    if ldb not in self.LOOKUPDB:
        return -1
    for cc_cw in self.contextweight:
        if cc_cw[0]==ldb and cc_cw[1]==cc and cc_cw[2]==i and cc_cw[3]==c:
            return cc_cw
    return -1
        ...
```

Abbildung 85: Ausschnitt der Spezifikation und Implementierung

Das folgende Unterkapitel beschreibt die Anwendung der oben beschriebenen Szenarien mit dem Prototypen zur Bewertung unseres Verfahrens zur Nutzerunterstützung.

7.3 DURCHFÜHRUNG UND ERGEBNISSE

Die drei in Kapitel 7.1 beschriebenen Szenarien wurden mit dem in Kapitel 7.2 dargestellten Prototyp ausgeführt und die Ergebnisse mit dem Sequenzvorhersage-Verfahren von Jacobs und Blockeel verglichen. Das Jacobs/Blockeel-Verfahren wurde bereits im Stand der Technik in

Kapitel 2.2.2 auf Seite 29 im Überblick sowie im Kapitel 6.1 auf Seite 99 im Detail vorgestellt. In [64] wurde das Verfahren von den Autoren publiziert. Unser Verfahren (im Folgenden „MD“ genannt) nutzte für den Aufbau der Erfahrung die Arbeitsschritt-Sequenzen mit den dazugehörigen Kontextinformationen (siehe Abbildung 82). Der Algorithmus von Jacobs und Blockeel (im Folgenden „JB“ genannt) nutzt hingegen nur die Sequenzen von Arbeitsschritten. Die Motivation dahinter ist klar: Jacobs/Blockeel kann die Kontextinformationen nicht sinnvoll interpretieren. Die einzige Möglichkeit wäre, einen Arbeitsschritt mit Kontextinformationen (z.B. [38,48,49,33], siehe Abbildung 82) als vorzuschlagenden Schritt (also als atomares Ereignis) zu sehen. Das Ergebnis der Vorhersage ist dabei offensichtlich viel schlechter, da sich viele dieser Ereignisse unterscheiden (weil sie z.B. einen neuen Kontext erzeugen, also eine „neue“ ID beobachtet wird), obwohl der eigentliche Arbeitsschritt gleich bleibt.

Zu jedem Szenario wurden zwei Graphen erstellt: Der eine (in den folgenden Abbildungen oben dargestellt), zeigt die Gesamtanzahl korrekter Vorhersagen nach jedem Arbeitsschritt. Unten in den Abbildungen wird für jeden Schritt beschrieben, wie hoch der Prozentsatz korrekter Vorhersagen ist. Die Ergebnisse von Szenario 1 sind in Abbildung 86, von Szenario 2 in Abbildung 87 und von Szenario 3 in Abbildung 88 dargestellt.

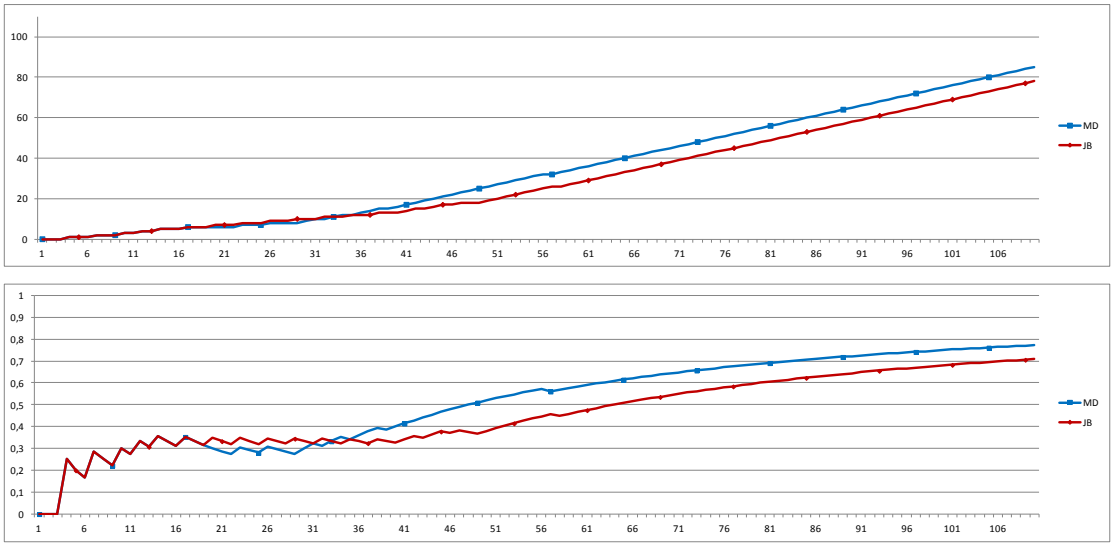


Abbildung 86: Ergebnisse Szenario 1 (oben: Gesamtanzahl korrekter Vorhersagen, unten: % korrekter Vorhersagen)

Szenario 1 (hohe Kopplung, breiter Entwurf) folgt einer schwierig zu erlernenden Methodik, da es viele gleichförmige Sequenzen (z.B. $39 \rightarrow 39 \rightarrow 39$) gibt. Hier dauert es lange, bis beide Verfahren die Wahrscheinlichkeitsverteilung richtig angepasst haben. Besonders schwierig sind die Übergänge von einer gleichförmigen Teilsequenz zur nächsten Teilsequenz (z.B. $39 \rightarrow 39 \rightarrow 39 \rightarrow 40$) zu erlernen. Das liegt vor allem daran, dass die bedingten Wahrscheinlichkeiten eine Mindestlänge haben müssen, um sicher richtig vorhersagen zu können. Die Bedingung der Wahrscheinlichkeit $P(40|39 \rightarrow 39)$ ist beispielsweise nicht lang genug, um sicher korrekte Vorhersagen zu treffen. Denn diese Wahrscheinlich passt auf die Beobachtung $41 \rightarrow 39 \rightarrow 39$, liefert jedoch in diesem Fall ein falsches Ergebnis, da hier der Arbeitsschritt 39 folgt. D.h. hier benötigen wir eine Wahrscheinlichkeit mit mindestens der Länge 3 (z.B. $P(40|39 \rightarrow 39 \rightarrow 39)$), um die Methodik richtig zu erlernen. Dabei „dauert“ es eine bestimmte Zeit, bis die Wahrscheinlichkeiten in dieser Länge initial hinzugefügt werden. Weiterhin muss die Verteilung dieser Wahrscheinlichkeiten korrekt gelernt werden.

Beide Verfahren lernen die Methodik, sodass nach ca. 60 Schritten nur noch korrekte Vorhersagen gemacht werden. Unser Verfahren ist dabei besser und hat nach 110 Arbeitsschritten 85 Schritte (oder 77%) korrekt vorhergesagt (Vergleich JB: 78 Schritte oder 71%, siehe Tabelle 1 und Tabelle 2).

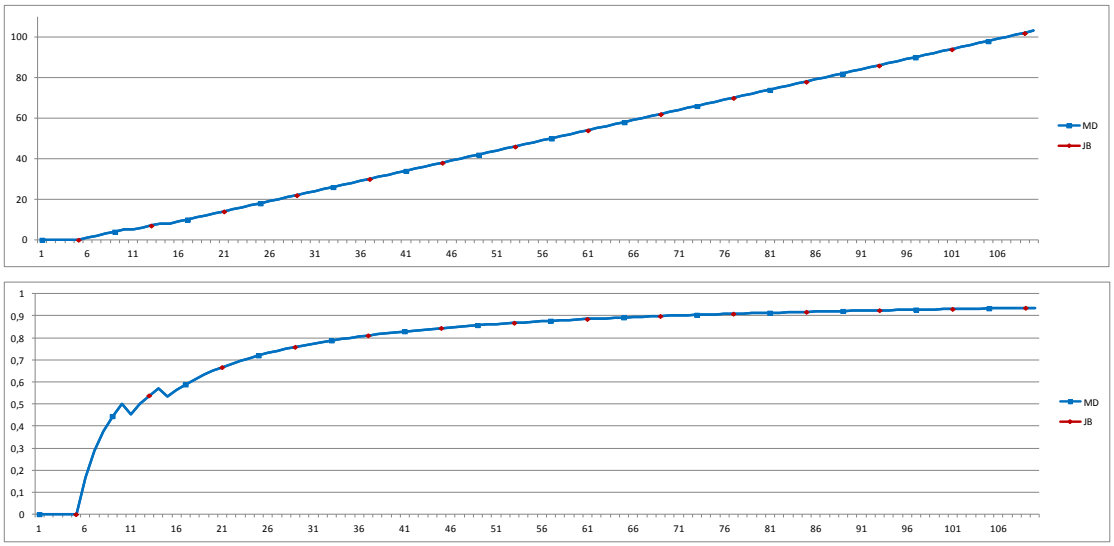


Abbildung 87: Ergebnisse Szenario 2 (oben: Gesamtanzahl korrekter Vorhersagen, unten: % korrekter Vorhersagen)

Szenario 2 (komplex, prototypische Methodik) ist in Abbildung 87 dargestellt. Die Abfolge ist hier für beide Verfahren einfach zu erlernen, da die Teilsequenzen keine direkt nacheinander wiederkehrenden Arbeitsschritte enthalten (wie in Szenario 1). Deshalb sagen hier beide Verfahren gleich voraus. In den Kontextinformationen (die unser Verfahren nutzt) sind in diesem Szenario auch keine entscheidungsrelevanten Mehrinformationen enthalten. Beide Verfahren schlagen deshalb mit 103 Arbeitsschritten 93% richtig vor.

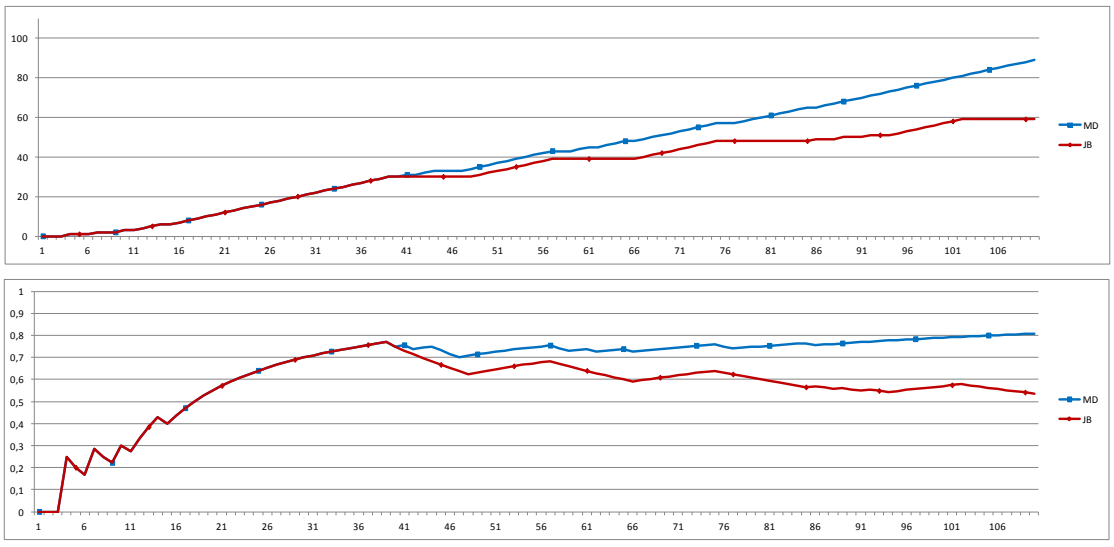


Abbildung 88: Ergebnisse Szenario 3 (oben: Gesamtanzahl korrekter Vorhersagen, unten: % korrekter Vorhersagen)

Abbildung 88 beschreibt das Szenario 3, das am ehesten einem realen Projektablauf entspricht. Hier sind die Teilsequenzen aus Szenario 1 und 2 zufällig vermischt. Die Kontextinformationen der Arbeitsschritte beinhalten einen richtigen Mehrwert, da auf Grundlage von bestimmten Kontexten unterschiedliche Arbeitsschritte vorzuschlagen sind. Dies ist auch deutlich am Ergebnis zu erkennen: Nach ca. 85 Arbeitsschritten hat unser Verfahren beide angewendeten Methoden richtig erlernt und schlägt dann immer den richtigen Arbeitsschritt vor. Das Verfahren von Jacobs und Blockeel kann dagegen die beiden Methoden nicht richtig erlernen und konvergiert gegen 55% richtiger Vorschläge. Unser Verfahren hat dagegen nach 110 Arbeitsschritten schon 89 (81%) richtig vorgeschlagen.

Anzahl korrekt vorhergesagter Arbeitsschritte (nach 110 Beobachtungen)		
Szenario	**MD**	**JB**
1	85	78
2	103	103
3	89	59

Tabelle 1: Anzahl korrekte Vorschläge der drei Szenarien

% korrekt vorhergesagter Arbeitsschritte (nach 110 Beobachtungen)		
Szenario	**MD**	**JB**
1	77%	71%
2	93%	93%
3	81%	53%

Tabelle 2: Prozent korrekter Vorschläge der drei Szenarien

Zusammenfassend ist zu sagen, dass unser Verfahren, insbesondere bei dem „realen Projektszenario“, wesentlich besser abschneidet, indem es die zur Verfügung stehenden Kontextinformationen sinnvoll nutzt. Unser Verfahren kann nach einer Lernphase sicher zwischen den beiden angewendeten Methoden unterscheiden. Es lernt dabei nicht nur schneller als das Verfahren von Jacobs/Blockeel, sondern kann auch eine größere Anzahl an Strukturen erkennen, sodass der Prozentsatz korrekter Vorschläger höher liegt.

8 EVALUIERUNG

Dieses Kapitel beschreibt die Evaluierung des hier vorgestellten Ansatzes zur Unterstützung des Projektmitarbeiters aus Kapitel 6. Die Evaluierung erfolgt mit realen Prozessdaten der IT-Entwicklung eines großen deutschen Konzerns. Dies ist der Unterschied zu der Evaluierung „im Kleinen“ aus Kapitel 7. Hier wurde die Beispielmethodik aus Kapitel 3 mit der in Kapitel 5 vorgestellten Prozessbeschreibungssprache beschrieben. Hieraus wurden dann synthetische Prozessdaten abgeleitet. Unsere Prozessbeschreibungssprache wurde so entworfen, dass der Projektmitarbeiter bei der Ausführung seiner Arbeit Freiheiten hat. Des Weiteren enthalten die während der Ausführung aufgezeichneten Prozessinformationen – insbesondere im Bereich der Produktkontexte – relevante Informationen, um gute Vorhersagen zu treffen. Das bedeutet, dass bei den synthetischen Prozessdaten die Unterschiede im Ablauf durch die Kontextinformationen erkannt werden können (und auch werden – siehe hierzu Kapitel 7).

Bei den „echten“ Prozessdaten, die in diesem Kapitel verwendet werden, ist dies nicht klar: Hier stellen sich insbesondere zwei Fragen: Hängen die Unterschiede im Prozessablauf überhaupt von Kontextinformationen ab? D.h., kann die Anwendung unseres Verfahrens aus Kapitel 6 überhaupt Vorteile bringen? Nachgelagert ist die zweite Frage, die die Evaluierungsdaten betrifft: Sind die relevanten Kontextinformationen überhaupt erfasst worden?

Der dieser Evaluierung zugrunde liegende Datensatz wurde bewusst von einer Organisation gewählt, bei der die spezifizierten Prozesse sehr stringent eingehalten werden. Die Prozesse sind hier auch in einer netzbasierten Spezifikationssprache hinterlegt. Unterschiede im durchgeführten Prozess kann es deshalb nur an Verzweigungspunkten geben. Das bedeutet, dass weniger Abweichungen, die gelernt werden können, in den Prozessen existieren, als dies bei einer Prozessbeschreibungssprache der Fall ist, bei der eine vollkommen flexible Ausführung möglich ist. Zum Vergleich: Unsere Sprache aus Kapitel 5 hat keine Reihenfolge für Arbeitsschritte angegeben. Die Arbeitsschritte stehen hier „flach nebeneinander“, und der Projektmitarbeiter kann Arbeitsschritte flexibel starten und beenden. Deshalb bieten diese Evaluierungsdaten, bei denen das „andere Extrem“, nämlich stringente Abfolge von Arbeitsschritten, vorhanden ist, eine gute Basis, um die Leistungsfähigkeit unseres Verfahrens zu zeigen.

Ein weiterer Unterschied von dieser Evaluierung zu der synthetischen ist, dass der zugrunde liegende Datensatz Prozessabläufe mehrerer Mitarbeiter enthält.[22] Der Fokus im vorderen Teil der Arbeit war die Unterstützung eines Projektmitarbeiters. Dennoch eigenen sich die Evaluierungsdaten sehr gut um festzustellen, ob Unterschiede in Abläufen kontextspezifisch sind. Mit anderen Worten: Wir können hiermit feststellen, ob die Kontextinformationen relevant für die konkreten Abläufe sind. Mit dieser Evaluierung prüfen wir also, ob sich unser Lernverfahren eignet, um aus dem bisherigen Vorgehen methodische Abläufe zu lernen, um einen Projektmitarbeiter zu unterstützen.

Das folgende Kapitel 8.1 gibt einen Überblick über die Prozessdaten sowie den zugrunde liegenden Entwicklungsprozess des Unternehmens, aus dem die Daten stammen. Der Umfang der Evaluierungsdaten (also die Anzahl der Prozessschritte und die Anzahl der enthaltenen Kontextinformationen pro Prozessschritt) hat eine andere Dimension im Vergleich zu den synthetischen Prozessdaten aus Kapitel 7. Um die Evaluierung effizient durchführen zu können, sind deshalb Überlegungen nötig, um diese „Informationsberge“ bewältigen zu können. Das Resultat, welches dann in der prototypischen Implementierung umgesetzt wurde, wird in Kapitel 8.1 beschrieben. Abschließend werden im Kapitel 8.2 die Ergebnisse der Evaluierung vorgestellt und diskutiert.

[22] Das Unternehmen hat aus datenschutzrechtlichen Gründen die Projektmitarbeiter-Informationen aus den Evaluierungsdaten entfernt.

Der den Evaluierungsdaten zugrunde liegende Entwicklungsprozess ist im Ausschnitt in Abbildung 89 als UML-Aktivitätsdiagramm abgebildet. Der Prozessausschnitt ist verfremdet dargestellt: Die Aktivitätsbeschreibungen sind nicht aufgeführt. Stattdessen sind Platzhalter in Form von IDs abgebildet. Die Abbildung ist in zwei Bereiche aufgeteilt: Der grün hinterlegte Bereich beschreibt die Arbeitsschritte, die zwingend ausgeführt werden müssen. Der rote Bereich gibt optionale Arbeitsschritte vor. Die Arbeitsschritte 1 und 6 sind für jede Teilsequenz zwingend auszuführen. Die anderen, 2-5, sind optionale Arbeitsschritte. Die Modellierung in Abbildung 89 beschreibt weiterhin folgende Einschränkungen: Vor oder nach Arbeitsschritt 1 (oder vor und nach Schritt 1) kann optional Arbeitsschritt 3 ausgeführt werden. Weiterhin können nach Schritt 6 die Arbeitsschritte 2, 4 und/oder 5 ausgeführt werden, wobei die Reihenfolge hierbei nicht relevant ist.

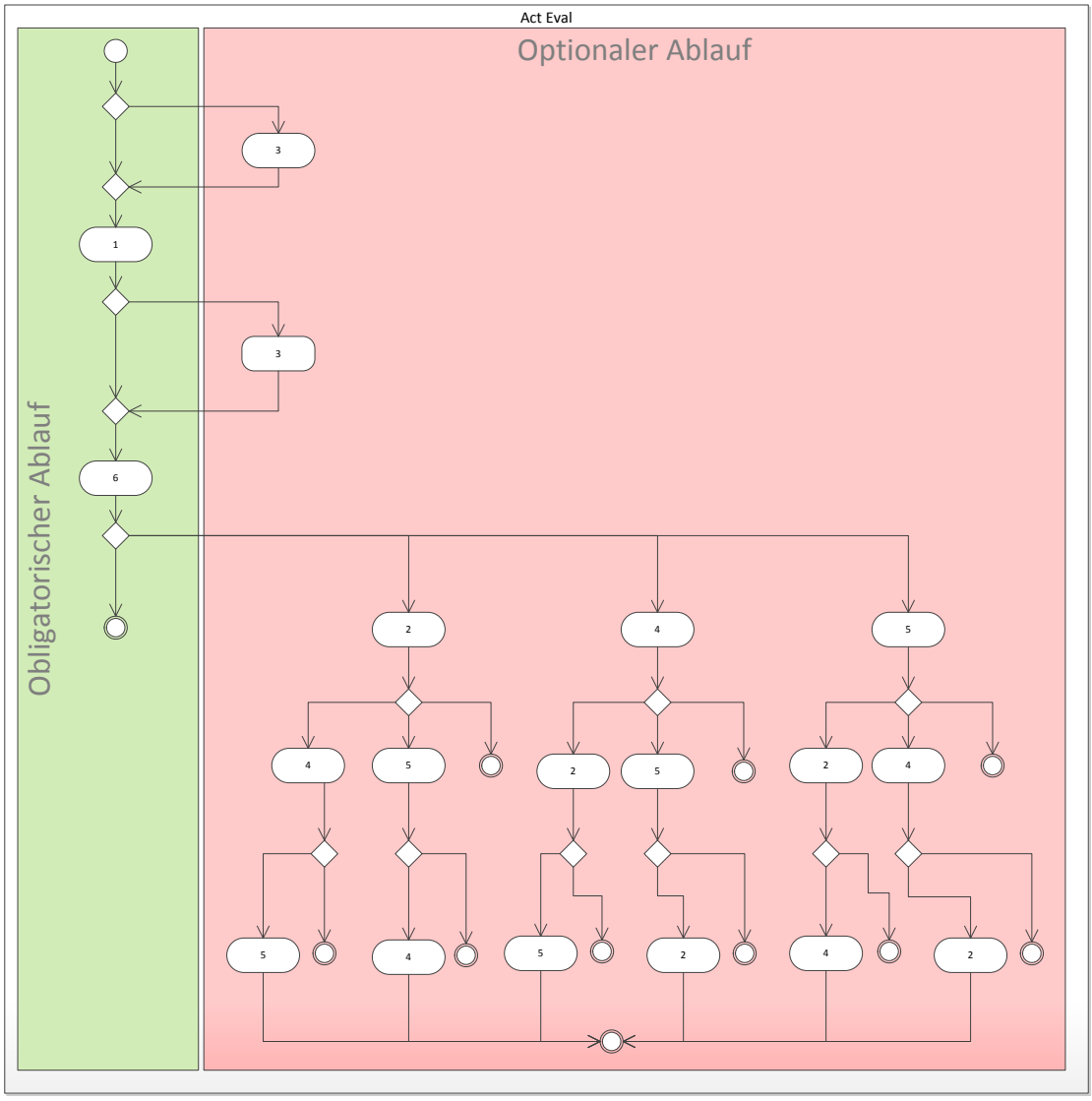


Abbildung 89: Prozessbeschreibung, die den Evaluierungsdaten zugrunde liegt

Abbildung 90 (unten) zeigt die Informationen, die für die Evaluierung unseres Verfahrens verwendet werden. In dieser Abbildung ist auch dargestellt, wie sich diese aus den Quelldaten zusammensetzen. Die Arbeitsschritte (in der Abbildung markiert: Arbeitsschritt mit der ID 6) entsprechen dem Task-Typ, die darunter liegenden Kontextinformationen werden aus der Modul-Spalte abgeleitet. Bei den Kontexten gibt es vier Kontextklassen (vgl. hierzu (56) auf Seite 92 oder die Seiten 103 und 110). Diese tragen die IDs 1-4 (dargestellt in Abbildung 90 unten am Anfang der eckigen Klammern). Die Abbildung der Spalteninformationen auf die Kontextklassen ist in der Abbildung oben rechts skizziert.

Die zur Verfügung gestellten Rohdaten wurden komplett nach dem oben dargestellten Muster in ein für den Prototypen interpretierbares Format (entsprechend Abbildung 90 unten konvertiert. Die gesamte Testsequenz hat eine Länge von etwa 5600 Einzelschritten. Diese Menge stellt eine andere Dimension dar als die Testsequenzen der synthetischen Evaluierung aus Kapitel 7. Um diese Evaluierung effizient durchführen zu können, sind deshalb „Begrenzungen“ der Datenstruktur, die die gelernte Erfahrung darstellt, nötig. Dies wird im folgenden Unterkapitel dargestellt.

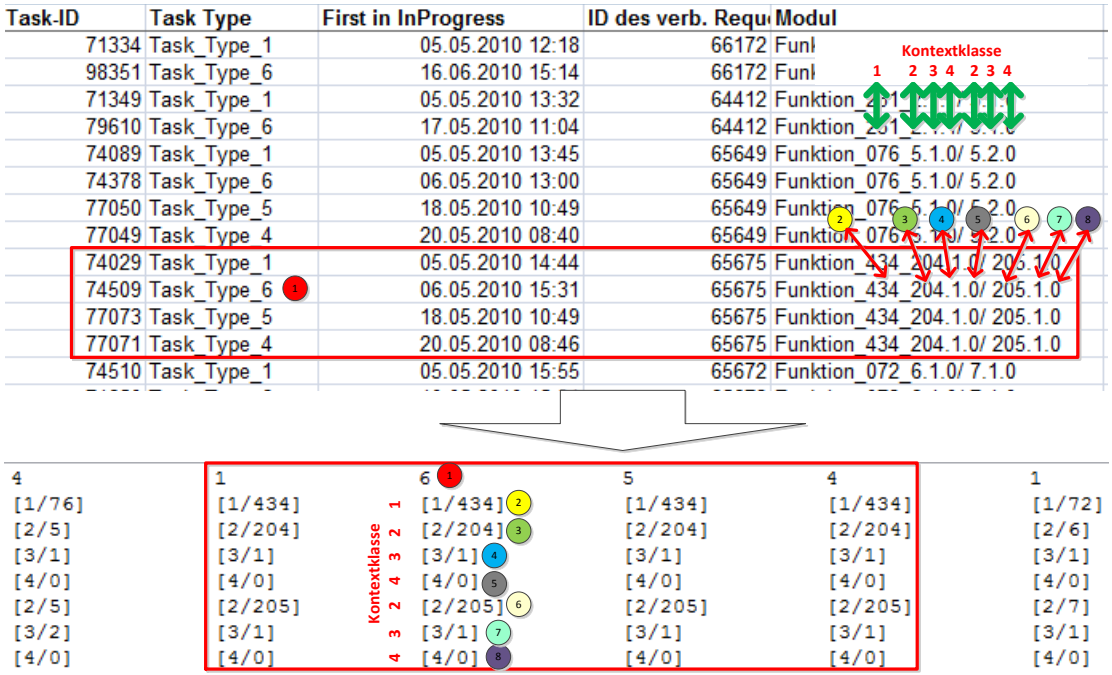


Abbildung 90: Ausschnitt der Prozesdaten (oben); entsprechender Ausschnitt der Testsequenz

8.1 PROTOTYP

Das in diesem Kapitel evaluierte und in Kapitel 6 (ab Seite 99) beschriebene Verfahren zur Nutzerunterstützung basiert auf einem Sequenzvorhersage-Verfahren von Jacobs und Blockeel [64]. Zu Erinnerung: Unser Verfahren lernt Teilsequenzen in Form von bedingten Wahrscheinlichkeiten. An jede bedingte Wahrscheinlichkeit werden die beobachteten Kontextinformationen angehängt. Schlägt unser Verfahren einen richtigen Arbeitsschritt vor, wird angenommen, dass die Wahrscheinlichkeitswerte für den relevanten Ausschnitt der Sequenz richtig verteilt sind. Die Teilsequenzen dieser bedingten Wahrscheinlichkeiten werden dann „nach vorne“ verlängert und die Wahrscheinlichkeiten werden angenähert.

Macht unser Verfahren viele „gute“ Vorschläge, so wächst unsere Datenstruktur. Hieraus resultieren zwei Probleme: Erstens werden die Teilsequenzen in Form von bedingten Wahrscheinlichkeiten immer länger. Zweitens werden die Teilsequenzen immer „tiefer“: Wird

eine existierende Teilsequenz erneut beobachtet, werden auch die Kontextwahrscheinlichkeiten aktualisiert und neu beobachtete Kontexte angehängt.

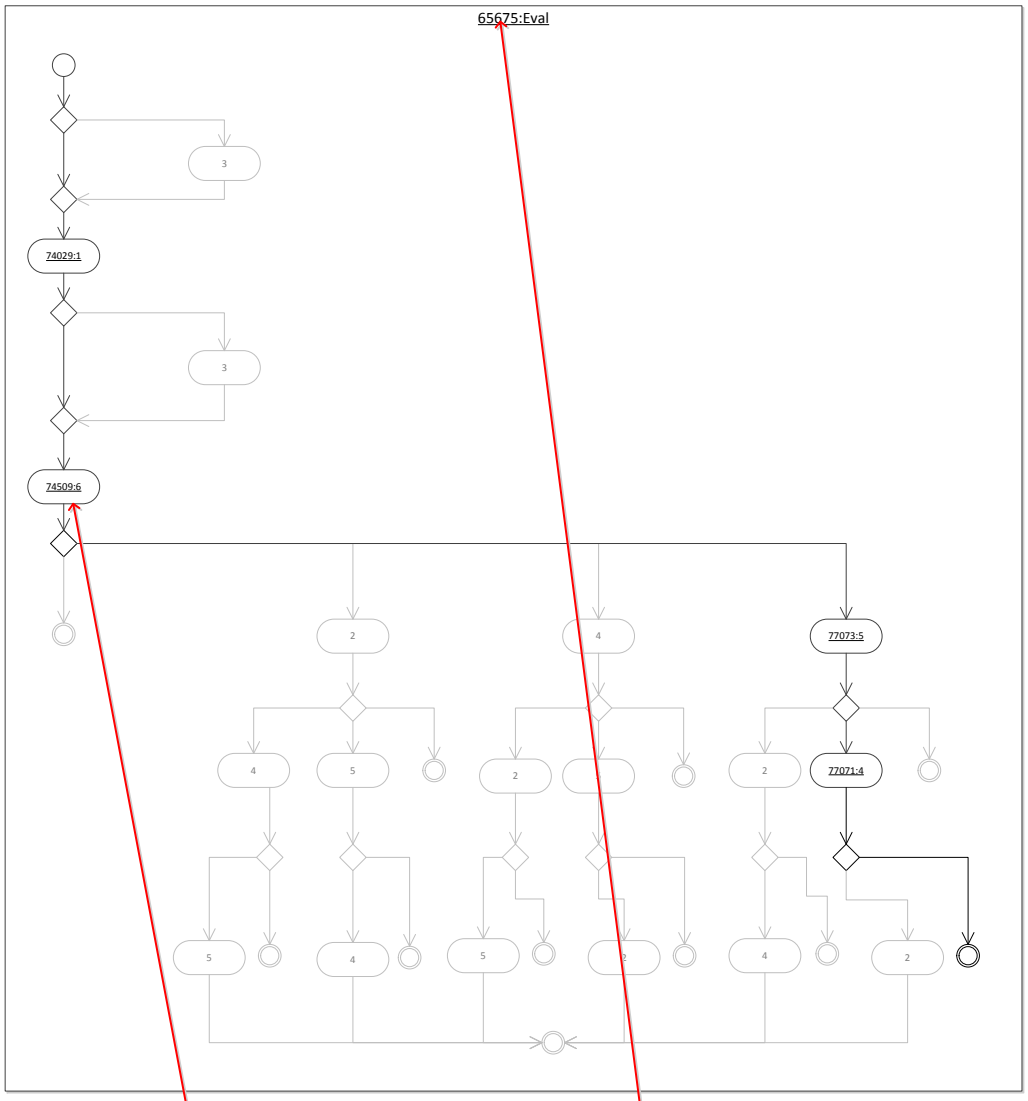
Ersteres Problem hat prinzipiell jedes Sequenzvorhersage-Verfahren. Eine Lösung hierfür ist, dass die Länge der gelernten Sequenzen auf einen festen maximalen Wert beschränkt wird. In unserem Fall lässt sich die maximale Länge der Teilsequenzen leicht feststellen: In der der Evaluierung zugrunde liegenden Prozessbeschreibung haben zusammenhängende Teilsequenzen die maximale Länge 7 (siehe Abbildung 89, z.B. $3 \rightarrow 1 \rightarrow 3 \rightarrow 6 \rightarrow 2 \rightarrow 5 \rightarrow 4$). Deshalb wurden für unsere Evaluierung die Teilsequenzen auf die maximale Länge 8 (nicht 7, da so die Übergänge zwischen den Teilsequenzen auch präzise gelernt werden können) beschränkt. Dennoch werden mit dem Standardverfahren von Jacobs und Blockeel Sequenzen (in der Länge bis 8) „spekulativ" hinzugefügt, die anschließend nicht mehr beobachtet werden und deshalb nicht relevant sind. Um diese Elemente zu erfassen, wurde in unserer Implementierung ein „Timer" für die Teilsequenzen (bzw. die bedingten Wahrscheinlichkeiten) hinzugefügt. Wird die Wahrscheinlichkeit einer Teilsequenz aktualisiert (wird diese Teilsequenz also wieder beobachtet), wird der entsprechende Timer inkrementiert. Wurde nach einer festen Anzahl an Beobachtungen ein Timer einer Sequenz niemals inkrementiert, so wird angenommen, dass diese Teilsequenz nicht relevant ist und aus der Datenstruktur entfernt.[23]

Ein Ausschnitt der Prozessdaten in Rohform ist in Abbildung 91 unten dargestellt. Jede Zeile entspricht hier dem Start eines Arbeitsschrittes. Die Arbeitsschritte sind in der Spalte „Task Type" beschrieben. Links daneben (Task-ID) ist eine eindeutige Identifikation, die den gestarteten Arbeitsschritt repräsentiert, hinterlegt. Bei jedem Start eines Arbeitsschrittes wird diese ID neu vergeben. Die Spalte „First in InProgress" gibt das Datum und die Uhrzeit an, wann der Arbeitsschritt gestartet wurde. Die nächste Spalte „ID des verb. Requests" beschreibt die ID der Prozessausschnitt-Instanz: Arbeitsschritte einer zusammengehörenden Teilsequenz haben die gleiche ID in dieser Spalte. Aus diesen beiden Spalten lässt sich die Abfolge der Arbeitsschritte bestimmen. In Abbildung 91 ist die Sortierung in der richtigen Reihenfolge bereits erfolgt. Die letzte Spalte („Modul") beschreibt die für die Ausführung der Arbeitsschritte relevanten Architekturelemente (Module).

Für die Evaluierung unseres Verfahrens sind die Informationen in den Spalten „Task Type" sowie „Modul" relevant. In „Task Type" ist die Sequenz der Arbeitsschritte beschrieben, für die dazugehörenden Kontextinformationen wird die Spalte „Modul" verwendet. Die übrigen Spalten (z.B. „Task-ID") enthalten eindeutige und einmalige Informationen. Hieraus lassen sich mit unserem Verfahren keine Strukturen erkennen, deshalb werden diese Informationen auch nicht für die Evaluierung verwendet.

Das zweite oben genannte Problem beschränkt sich auf unser Verfahren und betrifft die Kontextwahrscheinlichkeiten. Bei jeder beobachteten Teilsequenz werden alle Kontextwahrscheinlichkeiten aktualisiert (siehe (102) auf Seite 122). Deshalb ist hier eine einfache Lösung, in regelmäßigen Abständen alle diese Kontextwahrscheinlichkeiten aus der Datenstruktur zu entfernen, die einen festgelegten Schwellwert unterschreiten. Die Motivation dahinter ist, dass diese Kontexte nur temporär (oder auch nur einmalig) beobachtet wurden und deshalb für gute Vorhersagen nicht relevant sind.

[23] Hier genügt es übrigens nicht, alle Teilsequenzen, deren Wahrscheinlichkeiten einen Schwellwert unterschreiten aus der Datenstruktur zu entfernen, da die Wahrscheinlichkeiten nicht beobachteter Teilsequenzen (präziser: Bedingte Wahrscheinlichkeiten, deren Bedingungen nicht beobachtet werden) auch nicht aktualisiert werden.



Task-ID	Task Type	First in InProgress	ID des verb. Requ	Modul
71334	Task_Type_1	05.05.2010 12:18	66172	Funktion_256_3.1.1/ 4.1.0
98351	Task_Type_6	16.06.2010 15:14	66172	Funktion_256_3.1.1/ 4.1.0
71349	Task_Type_1	05.05.2010 13:32	64412	Funktion_251_2.1.1/ 3.1.0
79610	Task_Type_6	17.05.2010 11:04	64412	Funktion_251_2.1.1/ 3.1.0
74089	Task_Type_1	05.05.2010 13:45	65649	Funktion_076_5.1.0/ 5.2.0
74378	Task_Type_6	06.05.2010 13:00	65649	Funktion_076_5.1.0/ 5.2.0
77050	Task_Type_5	18.05.2010 10:49	65649	Funktion_076_5.1.0/ 5.2.0
77049	Task_Type_4	20.05.2010 08:40	65649	Funktion_076_5.1.0/ 5.2.0
74029	Task_Type_1	05.05.2010 14:44	65675	Funktion_434_204.1.0/ 205.1.0
74509	Task_Type_6	06.05.2010 15:31	65675	Funktion_434_204.1.0/ 205.1.0
77073	Task_Type_5	18.05.2010 10:49	65675	Funktion_434_204.1.0/ 205.1.0
77071	Task_Type_4	20.05.2010 08:46	65675	Funktion_434_204.1.0/ 205.1.0
74510	Task_Type_1	05.05.2010 15:55	65672	Funktion_072_6.1.0/ 7.1.0
		5:34		Funk
		.0:51		Funktion_072_6.1.0/ 7.1.0
		20.05.2010 09:00	65672	Funktion_072_6.1.0/ 7.1.0
74390	Task_Type_1	05.05.2010 16:36	65673	Funktion_069_11.1.0/ 12.1.0
74949	Task_Type_6	11.05.2010 13:12	65673	Funktion_069_11.1.0/ 12.1.0
79070	Task_Type_5	18.05.2010 10:51	65673	Funktion_069_11.1.0/ 12.1.0
79069	Task_Type_4	20.05.2010 09:04	65673	Funktion_069_11.1.0/ 12.1.0
71350	Task_Type_1	05.05.2010 17:46	62312	Funktion_250_3.1.0/ 4.1.0
75271	Task_Type_6	07.05.2010 09:33	62312	Funktion_250_3.1.0/ 4.1.0
77240	Task_Type_5	30.08.2010 13:35	62312	Funktion_250_3.1.0/ 4.1.0

Arbeitsschritt (Instanz)
Arbeitsschritt (Typ)
Ausführungsreihenfolge
Aktivität (Instanz)
Kontextinformationen

Abbildung 91: Ausschnitt der Prozessdaten in Rohform (unten); entsprechender Ablauf in der Prozesbeschreibung (oben)

8.2 DURCHFÜHRUNG UND ERGEBNISSE

Unser Verfahren wurde mit der Evaluierungssequenz, vorgestellt in Kapitel 8.1, getestet und mit dem zugrunde liegenden Kernverfahren (siehe Kapitel 6.1 auf Seite 99) verglichen. Dabei ist zu beachten, dass die in Kapitel 8.1 beschriebenen Einschränkungen und Optimierungen der Datenstruktur auch beim Kernverfahren übernommen wurden. So können die Ergebnisse der Anwendung beider Verfahren auch verglichen werden.

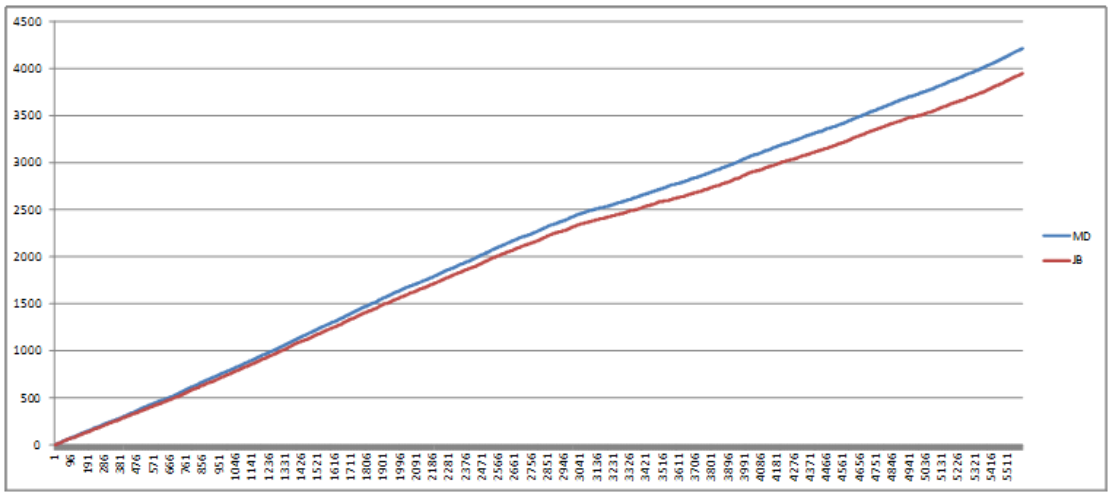


Abbildung 92: Anzahl korrekter Vorschläge nach jedem Prozessschritt

Abbildung 92 zeigt die Gesamtanzahl korrekter Vorschläge nach jedem Prozessschritt der beiden Verfahren. Der prozentuale Anteil korrekter Vorschläge nach jedem Prozessschritt beider Verfahren ist in Abbildung 93 dargestellt. Aus diesen beiden Grafiken lässt sich erkennen, dass unser Verfahren bessere Ergebnisse liefert als das Kernverfahren, welches die Kontextinformationen ignoriert. Das Sequenzvorhersage-Verfahren von Jacobs/Blockeel hat am Ende der Sequenz 70% (Anzahl: 3949) der Arbeitsschritte korrekt vorgeschlagen. Unser Verfahren dagegen schafft es am Ende auf 75% (Anzahl: 4215) richtiger Vorschläge. Die Ergebnisse sind der Vollständigkeit halber in Tabelle 3 und Tabelle 4 dokumentiert.

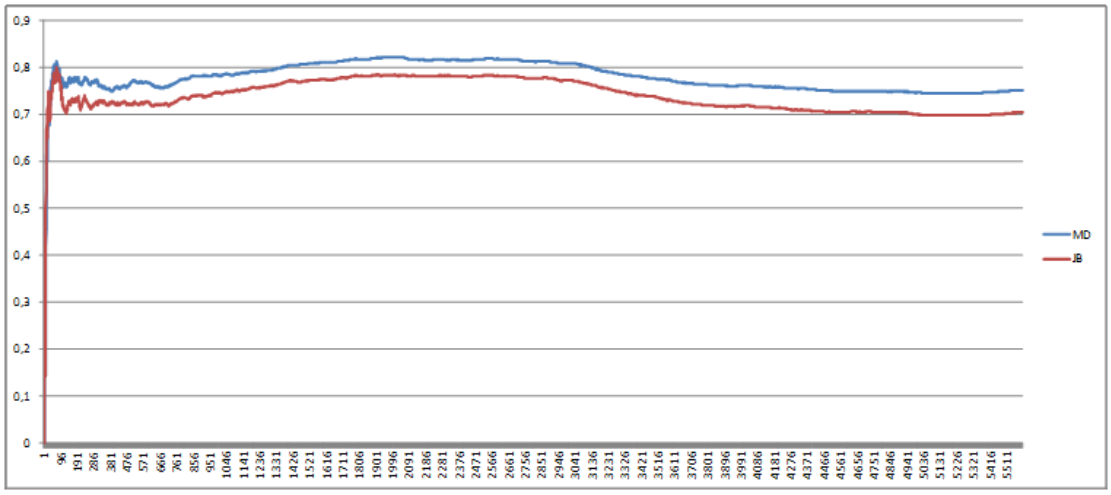


Abbildung 93: Prozent korrekter Vorschläge nach jedem Prozessschritt

Noch entscheidender ist jedoch, dass unser Verfahren eine (mindestens gleichbleibend) höhere Vorschlagsrate für korrekte Vorschläge hat, als das Verfahren von Jacobs/Blockeel. Dies ist auch in Abbildung 94 erkennbar: Hier werden die beiden Verfahren verglichen, indem man die Differenz der Anzahl korrekter Vorschläge beider Verfahren nach jedem Prozessschritt angibt. Diese Grafik zeigt deutlich, dass über lange Sequenzen unser Verfahren eine höhere Rate für korrekte Vorschläge besitzt. Des Weiteren ist es sogar so, dass unser Verfahren diesen Vorsprung über längere Sequenzen hinweg ausbauen kann. Abbildung 95 stellt die Differenz der Prozentwerte korrekter Vorschläge der beiden Verfahren dar. Die ebenfalls dargestellte Trendlinie steigt stetig. Hieran ist zu erkennen, dass unser Verfahren den Vorsprung mit der Zeit sogar leicht ausbaut. Das bedeutet, unser Verfahren lernt (auch über die gesamte Evaluierungssequenz hinweg) schneller als das Kernverfahren von Jacobs/Blockeel.

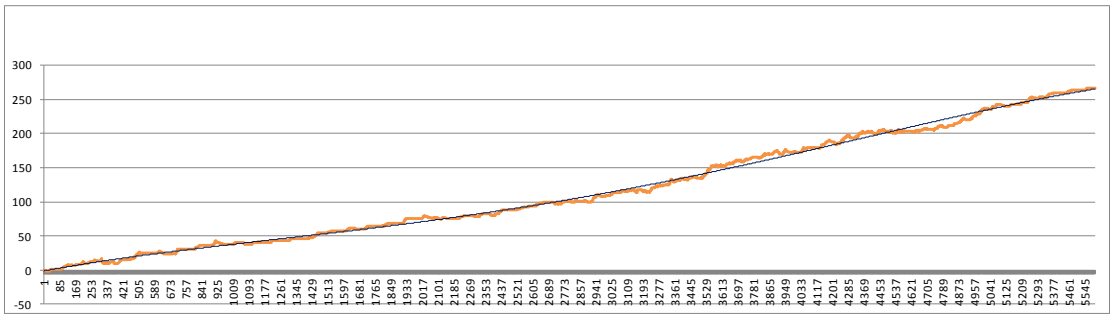


Abbildung 94: Differenz Anzahl korrekter Vorschläge (MD) und Anzahl korrekter Vorschläge (JB)

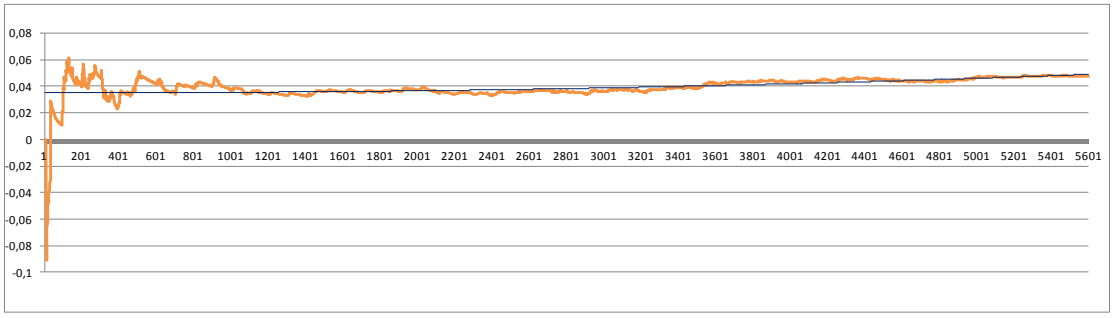


Abbildung 95: Differenz % korrekter Vorschläge (MD) und % korrekter Vorschläge (JB)

Abschließend ist zu sagen, dass unser Verfahren in der Evaluierungssequenz bei den darin enthaltenen Kontextinformationen Strukturen gelernt hat, die zu einer (auch dauerhaft) besseren Vorschlagsrate im Vergleich zu dem Verfahren von Jacobs/Blockeel geführt hat. Unser Verfahren liefert mit der Zeit sogar immer bessere Ergebnisse.

Zu beachten ist auch, dass die Differenz korrekter Vorschläge beider Verfahren bei ca. 5% liegt (im Vergleich: Beim „real-life"-Szenario bei der synthetischen Evaluierung in Kapitel 7 wurde eine Differenz von 28% gemessen). Damit können die beiden am Anfang dieses Kapitels gestellten Fragen mit „ja" beantwortet werden. Die Unterschiede in den Prozessabläufen hängen offensichtlich auch von den (erfassten) Kontextinformationen ab und werden auch erlernt. Offen bleibt dagegen, ob wirklich *alle* relevanten Kontextinformationen in der Evaluierungssequenz erfasst wurden.

Ein weiterer Punkt wurde schon am Anfang dieses Kapitels angesprochen: Die Evaluierungsdaten enthalten die Prozessabläufe mehrerer Projektmitarbeiter. Es ist durchaus

vorstellbar, dass es zu noch besseren Ergebnissen gekommen wäre, wenn die Abläufe einzelner Mitarbeiter evaluiert worden wären oder auch der Kontext „Projektmitarbeiter“ hinzugefügt worden wäre.

Es steht jedoch außer Frage: Das Kernverfahren hat mit ca. 70% Trefferrate schon sehr gute Ergebnisse erzielt. Dies ist darauf zurückzuführen, dass die Organisation, aus der die Daten stammen, wie einleitend in Kapitel 8 auf Seite 135 beschrieben, einen sehr stringent eingehaltenen Entwicklungsprozess hat und dieser durch eine netzbasierte Sprache beschrieben ist. Zum Vergleich: Die Rate des Kernverfahrens bei Szenario 3 der synthetischen Evaluierung in Kapitel 7 erzielte dagegen 53% korrekte Vorschläge (unser Verfahren im Vergleich: 81%). Trotz dieser stringenten Prozesse in der Organisation schafft es unser Verfahren dennoch, bessere Ergebnisse zu erzielen als das Kernverfahren von Jacobs/Blockeel.

Anzahl korrekt vorhergesagter Arbeitsschritte (nach 5604 Beobachtungen)		
Evaluierung	**MD**	**JB**
	4215	3949

Tabelle 3: Anzahl korrekte Vorhersagen bei der Evaluierung

% korrekt vorhergesagter Arbeitsschritte (nach 5604 Beobachtungen)		
Evaluierung	**MD**	**JB**
	75%	70%

Tabelle 4: Prozent korrekter Vorschläge bei der Evaluierung

Motivation. Motiviert wurde diese Arbeit auf der einen Seite durch die (teils große) Diskrepanz zwischen spezifizierten Prozessmodellen und tatsächlich durchgeführten Prozessen in der IT-Entwicklung. Derzeit entsprechen also die generisch und idealisiert beschriebenen Abläufe in den Modellen nicht dem tatsächlich in IT-Projekten konkret angewendeten Vorgehen.

Auf der anderen Seite sind Menschen – die Projektmitarbeiter – und ihre individuelle Arbeit maßgeblich für den Erfolg eines Projektes verantwortlich. Andere Elemente, die in die Entwicklung involviert sind, wie zum Beispiel Werkzeuge oder Beschreibungssprachen, spielen für den Projektverlauf und das Ergebnis (im Vergleich zu den Menschen) nur eine untergeordnete Rolle.

Deshalb sieht diese Arbeit den Projektmitarbeiter und seine (individuelle) Arbeit im Vordergrund. Dies hat einige Konsequenzen für Prozessmodelle: Ein „ideales" Prozessmodell sollte die Projektmitarbeiter während ihrer Arbeit unterstützen und nicht einschränken. Das hat zur Folge, dass die o.g. Diskrepanz zwischen Modell und Ablauf möglichst gering gehalten werden muss.

Heutige Lösungen, um dies zu bewerkstelligen, zielen darauf ab, dass Menschen (Projektmitarbeiter) das Prozessmodell anpassen. Dies kann im organisationsweiten Fokus geschehen. Hier geht es also darum, Anpassungen eines organisationweiten Prozessmodells vorzunehmen. Weiterhin gibt es Verfahren, um diese organisationsweit gültigen Modelle initial projektspezifisch anzupassen. Die dritte Klasse an Lösungen beschäftigt sich mit der Anpassung eines (projektspezifischen) Modells während der Ausführung.

Alle diese bisherigen Lösungen haben den Nachteil, dass die Anpassung des Prozessmodells erfolgen muss, *bevor* es zu der Abweichung zwischen diesem und dem Prozess kommt. Weiterhin muss diese Abweichung manuell (also beispielsweise von dem Projektmitarbeiter) vorgenommen werden. Der Nutzen ist, speziell bei temporären oder projektmitarbeiterspezifischen Änderungen, mehr als fraglich.

Gesamtansatz. Deshalb wurde in dieser Arbeit ein Konzept vorgestellt, das sich von anderen prozessgetriebenen Ansätzen und deren zugrunde liegenden Prozessmodellen (z.B. netzbasierte Ansätze) unterscheidet: Da der Projektmitarbeiter und seine Arbeit im Vordergrund stehen, soll das Prozessmodell einerseits flexibel ausführbar sein, andererseits soll es den Mitarbeiter bei seiner Arbeit unterstützen, insbesondere auch bei temporären oder kontextspezifischen Vorgehensweisen. Um dies zu bewerkstelligen, sieht es unser Konzept vor, dass ein Werkzeug das Prozessmodell ausführt und die Arbeitsweise des Projektmitarbeiters beobachtet. Mit Hilfe dieser Informationen werden dann Erfahrungsdaten aufgebaut, mit denen es möglich ist, den Projektmitarbeiter individuell und (kontext)spezifisch während seiner Arbeit zu unterstützen. Das bedeutet, dass in unserem Konzept das Vorgehen des Mitarbeiters automatisch gelernt wird. Die Erfahrung wird genutzt, um dem Mitarbeiter Vorschläge für die nächsten anzuwendenden Arbeitsschritte zu machen (und nicht etwa um ihn einzuschränken).

Das Konzept sieht also zwei Dinge vor: Einerseits eine Prozesssprache, die eine flexible Ausführung zulässt, andererseits wird während der Ausführung Wissen mit einem Lernverfahren aufgebaut. Ein Werkzeug, das beide Elemente umsetzt, kann mit der vorhandenen Erfahrung den Projektmitarbeiter gezielt in seiner Arbeit unterstützen.

Softwareprozessbeschreibungssprache. Hierfür haben wir eine Prozessbeschreibungssprache spezifiziert, die diesen Anforderungen gerecht wird. Diese Sprache setzt dabei die relevanten Konzepte um, wie zum Beispiel die Möglichkeit der flexiblen Ausführung, und beschreibt die Elemente, die nötig sind, um einer (kontext)spezifischen Unterstützung des Projektmitarbeiters gerecht zu werden. Beispielhaft wurde hier der

Produktkontext spezifiziert, der genutzt werden kann, um kontextspezifische, methodische Erfahrungen aufzubauen. Bei der hier vorgestellten Sprache wurden die Syntax, aber auch die grundlegende Ausführungssemantik beschrieben. Zu beachten ist jedoch, dass die vorgestellte Sprache nur die Kernelemente für unser eigentliches Ziel enthält: Flexible Ausführung und Unterstützung des Projektmitarbeiters. Weitere Elemente, um die Sprache produktiv einsetzen zu können, müssten ergänzt werden, bzw. müssten unsere Konzepte in eine bestehende Prozessbeschreibungssprache integriert werden. Beispielhaft kann hier die Komposition von Arbeitsschritten oder auch die Einführung von Produktzuständen genannt werden.

Verfahren für die Unterstützung des Projektmitarbeiters. Die spezifizierte Prozessbeschreibungssprache setzt vornehmlich eines unserer Ziele um, nämlich die flexible Ausführung des Prozessmodells. Weiterhin werden die Grundlagen gelegt, um methodische, kontextspezifische Erfahrung für Projektmitarbeiter aufzubauen. Unser Verfahren für die Unterstützung des Mitarbeiters nutzt diese Informationen, die während der Prozessausführung anfallen, um die Erfahrungsdaten aufzubauen und auszuwerten, mit dem eigentlichen (zweiten) Ziel, den Projektmitarbeiter während seiner Arbeit spezifisch zu unterstützen.

Das hier vorgestellte Verfahren ist ein Sequenzvorhersage-Verfahren aus dem Gebiet des Maschinellen Lernens. Hier wurde auf ein Standardverfahren für die Sequenzvorhersage zurückgegriffen und dieses derart erweitert, dass kontextspezifische Abläufe schneller gelernt und diese auch sicher unterschieden werden können.

Bewertung des Verfahrens. Für die Bewertung unseres Maschinellen Lernverfahrens wurden synthetische Projektabläufe (Szenarien) erstellt. Diese Ablaufsequenzen entsprechen einer zuvor beschriebenen Beispielmethodik, die auch in der gesamten Arbeit Anwendung findet. Unser Lernverfahren wurde mit dem zugrunde liegenden Kernverfahren verglichen. Insbesondere bei dem Szenario, das unterschiedliche kontextspezifische Teilsequenzen enthielt, wurde eine wesentlich bessere Vorschlagsgenauigkeit erzielt. Unser Verfahren hat nicht nur schneller gelernt, es konnte sogar – nach einer Lernphase – die unterschiedlichen Teilsequenzen sicher unterscheiden. Dies war bei dem Kernverfahren – schon prinzipbedingt – nicht möglich.

Evaluierung. Die synthetische Evaluierung kann die vorher erarbeiteten Konzepte und Ideen bestätigen oder widerlegen. Was sie jedoch nicht kann, ist, die grundlegende Idee, auf die unser Lernverfahren aufbaut, nämlich spezifisches Vorgehen anhand von Kontextinformationen zu erlernen, überprüfen. Mit anderen Worten: Hängen spezifische Teilsequenzen in einem realen Projektablauf von Kontexten ab, bzw. ist es möglich, diese zu erfassen? Deshalb wurde unser Verfahren mit realen IT-Projektdaten eines deutschen Unternehmens evaluiert. Zu beachten ist jedoch, dass die den Prozessdaten zugrunde liegende Prozessbeschreibung nicht den Konzepten unserer Beschreibungssprache entspricht. Die Sprache, die den Evaluierungsdaten zugrunde liegt, ist beispielsweise eine netzbasierte Sprache. Deshalb ist eine flexible Ausführung nur teilweise, nämlich an den Verzweigungspunkten, möglich.

Unser Lernverfahren hat trotz der völlig anderen Beschreibungssprache, die der Testsequenz zugrunde lag, bessere Vorschläge geliefert als das Kernverfahren. Die Differenz der Vorhersageraten war zwar nicht so groß wie bei der synthetischen Evaluierung, jedoch deutlich erkennbar. Mit der Zeit konnte unser Verfahren diese Differenz sogar ausbauen, was bedeutet, dass unser Verfahren schneller „besser wird“ als das Kernverfahren.

Die Gründe, warum unser Verfahren nicht wesentlich besser abschneidet als das Kernverfahren, können mit Sicherheit vielfältig sein: Auf der einen Seite wäre es etwa möglich, dass nicht alle relevanten Kontextinformationen erfasst wurden. Auf der anderen Seite muss auch deutlich gemacht werden, dass das Kernverfahren mit ca. 70% Vorschlagsrate schon sehr gute Ergebnisse liefert. Damit liegt die Vermutung nahe, dass sich die Teilsequenzen der Evaluierungsdaten nicht so stark unterscheiden, wie dies beispielsweise bei dem o.g. Szenario der synthetischen Evaluierung der Fall war. Dies ist auch nicht weiter verwunderlich, da die den

Evaluierungsdaten zugrunde liegende Prozesssprache eine netzbasierte Sprache ist und deshalb schon weniger „Freiraum“ in der Ausführung besteht.

Ausblick. Weiterführende Arbeiten kann man grob in zwei Bereiche einteilen: *Evaluierung* und *Integration*.

Im Bereich der **Evaluierung** wären Untersuchungen interessant, die auf Evaluierungsdaten beruhen, deren zugrunde liegende Prozessbeschreibung unsere Sprachkonzepte umsetzt. So könnte beurteilt werden, welche Vorteile wirklich z.B. mit dem Konzept der flexiblen Ausführung erreicht werden. Des Weiteren wäre eine Untersuchung interessant, bei der Evaluierungsdaten eines gesamten Projektes vorliegen. Zu Erinnerung: Unsere Evaluierungen haben sich immer nur auf einen begrenzten Prozessausschnitt beschränkt. Zusätzlich wären weitere Untersuchungen sinnvoll, um konkrete Klassen von Kontexten, die die Entscheidung des Vorgehens in einem Projekt beeinflussen, zu identifizieren.

Der Bereich **Integration** kann wiederum unterteilt werden in *technische Integration* und *fachliche Integration*. Technische Integration bedeutet insbesondere die Einbettung unserer Konzepte und Verfahren in ein Werkzeug, welches in einem Projekt genutzt wird. Dies ist auch zwingend nötig, um eine o.g. Evaluierung durchzuführen. Man kann sich hierunter beispielsweise ein Case-Tool vorstellen, mit dem UML-Modelle erstellt und gepflegt werden. Weitere Arbeiten in diesem Bereich könnten die Integration unserer Konzepte in eine umfassende, produktiv einsetzbare Prozessbeschreibungssprache sein. Beispielsweise stehen bei unserer Prozessbeschreibungssprache die Arbeitsschritte eines Projetmitarbeiters „flach nebeneinander“. Der Projektmitarbeiter wählt durchzuführende Arbeitsschritte aus und startet diese. Es ist jedoch durchaus sinnvoll, bestimmte Dinge in einer Prozessbeschreibung starr vorzugeben (z.B. Für jede fertiggestellte Komponente gibt es eine Qualitätssicherung). Hier stellt sich die Frage, wie dies sinnvoll mit der hier vorgestellten Prozessbeschreibungssprache kombiniert werden kann. Eine Antwort könnte sein, diese Sachverhalte in Vor- und Nachbedingungen zu kombinieren. Eine andere Möglichkeit wäre, Hierarchien in die Prozessbeschreibungssprache aufzunehmen. Auf oberster Ebene könnten dann „starre Vorgaben“ gemacht werden, die untere Ebene könnte flexibel ausführbar sein.[24]

Ein weiterer Bereich, der zu der technischen Integration gehört, betrifft die Umsetzung und Integration von dem hier vorgestellten Sequenzvorhersageverfahren. In diesem Bereich fällt beispielsweise die Bestimmung der möglichen Kontexte (beispielsweise könnte man sich die Frage stellen, ob es Sinn ergibt, den Wochentag als Kontext aufzunehmen) oder auch die maximale Länge der Teilsequenzen, die gelernt werden können.

Unter *fachlicher Integration* wird dagegen die Einbettung unserer Ideen und Konzepte in den „Gesamtlebenszyklus der Softwareprozesse“ verstanden. Zum Hintergrund: Unser Lernverfahren sieht sich die Arbeitsweise *eines* Projektmitarbeiters, der in *einem* Projekt arbeitet an. Das Verfahren lernt spezifisches Vorgehen (ohne Vorwissen) und kann damit Vorschläge für zukünftige Vorgehensweisen bieten. Derzeit wird also der Prozess-Lebenszyklus auf einen Anwender (in einem Projekt) beschränkt. In IT-Projekten und Organisationen gibt es jedoch mehr „Informationen“, die bei der Unterstützung genutzt werden könnten: Beispielsweise ändern sich die projektspezifischen Softwareprozessbeschreibungen noch während eines Projektes permanent. Teilweise werden diese Änderungen direkt an projektspezifischen Prozessmodellen vorgenommen (z.B. projektspezifisches V-Modell XT XML), teilweise werden die Änderungen auch in das Dokument „Projekthandbuch“ integriert. Wo man diese Änderungen auch einfügt ist nicht relevant. Wichtig ist nur, dass diese Änderungen die Prozessbeschreibung betreffen. Als

[24] Das V-Modell XT bietet einen ähnlichen Ansatz: Ablaufvorgaben werden explizit nur auf „hoher Ebene“, im Bereich der Entscheidungspunkte gemacht. Der feingranulare Ablauf vor dem Erreichen eines Entscheidungspunktes, also wann welches Produkt bzw. Thema behandelt wird, ist explizit nicht festgelegt.

Beispiel sei hier eine Änderung eines Projekthandbuchs, in der der Tag der wöchentlichen Projektbesprechung von Montag auf Mittwoch verschoben wird, genannt. Dies beschreibt nichts anderes als eine Änderung einer Prozessbeschreibung!

Ein Ziel für weitere Arbeiten in diesem Bereich könnte deshalb sein, die durch unsere Verfahren genutzte Erfahrung von (mehreren) Projektmitarbeitern zu nutzen, um die projektspezifische Prozessbeschreibung anzupassen. Hieraus könnten weiterhin Prozessdokumentationen erzeugt werden. Die Erfahrung der Projektmitarbeiter könnte auch genutzt werden, um neue Mitarbeiter (in einem Projekt) einzuführen und zu unterstützen.

Eine weitere Möglichkeit der *fachlichen Integration* ist die Einbettung unserer Arbeit in den „großen Zyklus" des Softwareprozesslebenszyklus. Hiermit meinen wir die Rückführung der Erfahrung eines Projektes in das organisationsweit gültige Prozessmodell. Auch hier kann unsere gelernte Erfahrung genutzt werden, um beispielsweise neue Vorgehensweisen, Techniken oder Verfahren zu integrieren.

10 REFERENZEN

[1] CHROUST, GERHARD: *Modelle der Software-Entwicklung*. München und Wien, Oldenbourg, 1992, ISBN 3-48621-878-6

[2] KELLNER, M. I ; MADACHY, R. J ; RAFFO, D. M: Software process simulation modeling: why? what? how? In: *Journal of Systems and Software* Bd. 46 (1999), Nr. 2, S. 91–105

[3] ROMBACH, D.: Integrated software process and product lines. In: *Unifying the Software Process Spectrum* (2006), S. 83–90

[4] KABBAJ, M. ; LBATH, R. ; COULETTE, B.: A Deviation Management System for Handling Software Process Enactment Evolution. In: *Making Globally Distributed Software Development a Success Story* (2008), S. 186–197

[5] MOHAMMED, K. ; REDOUANE, L. ; BERNARD, C.: A deviation-tolerant approach to software process evolution. In: *Ninth international workshop on Principles of software evolution in conjunction with the 6th ESEC/FSE joint meeting*, 2007, S. 75–78

[6] DERNIAME, JEAN-CLAUDE ; KABA, BADARA ; WASTELL, DAVID: The human dimension of the software process. In: *Software Process: Principles, Methodology, and Technology*, *Lecture Notes in Computer Science*. Bd. 1500, Springer US, 1999, ISBN 978-3-540-65516-9, S. 165–199

[7] RUS, IOANA ; BIFFL, STEFAN ; HALLING, MICHAEL: Systematically combining process simulation and empirical data in support of decision analysis in software development. In: *Proceedings of the 14th international conference on Software engineering and knowledge engineering*. New York, USA, ACM, 2002, ISBN 1-58113-556-4, S. 827–833

[8] DEMARCO, TOM ; LISTER, TIMOTHY: *Peopleware: Productive Projects and Teams*. 2. Aufl., Dorset House, 1999, ISBN 0932633439

[9] CURTIS, BILL ; HEFLEY, BILL ; MILLER, SALLY: *People Capability Maturity Model (P-CMM) Version 2.0, 2. Aufl.*, 2009

[10] CUGOLA, G. ; NITTO, E. DI ; GHEZZI, C. ; MANTIONE, M.: How To Deal With Deviations During Process Model Enactment. In: *17th International Conference on Software Engineering (ICSE)*. Seattle, Washington, USA, 1995, ISBN 0-89791-708-1, S. 265

[11] *Webseite KIT - Pervasive Computing Systems - Lehrstuhl für Pervasive Computing Systems (PCS) - Vorlesung: Kontextsensitive Systeme*. URL http://pcs.tm.kit.edu/209.php. - abgerufen 2012-08-03

[12] BOURQUE, P. ; DUPUIS, R.: *Guide to the software engineering body of knowledge*, IEEE Press, 2004, ISBN 0-7695-2330-7

[13] *Webseite ACM Computing Reviews*. URL http://www.computingreviews.com/. - abgerufen 2012-05-31

[14] ARMENISE, P. ; BANDINELLI, S. ; GHEZZI, C. ; MORZENTI, A.: Software processes representation languages: Survey and assessment. In: *Proceedings of the 4th International Conference on Software Engineering and Knowledge Engineering*, 1992, ISBN 0-81862-830-8, S. 455–462

[15] V-Modell XT Dokumentation, Version 1.3. Verfügbar: www.vmodellxt.de (2012)

[16] RAUSCH, ANDREAS: *Componentware: Methodik des evolutionären Architekturentwurfs*. München, TU München, 2001

[17] BECK, KENT ; BEEDLE, MIKE ; VAN BENNEKUM, ARIE ; COCKBURN, ALISTAIR ; CUNNINGHAM, WARD ; FOWLER, MARTIN ; GRENNING, JAMES ; HIGHSMITH, JIM ; HUNT, ANDREW ; U. A.:*Manifest für agile Softwareentwicklung*. URL http://www.agilemanifesto.org/iso/de/. - abgerufen 2012-08-27

[18] ROYCE, W.: Managing the development of large software systems. In: *Proceedings of IEEE WESCON*. Bd. 26, 1970

[19] *Webseite V-Modell XT*. URL www.vmodellxt.de. - abgerufen 2012-08-27

[20] V-Modell XT Dokumentation, Version 1.2. Verfügbar: www.vmodellxt.de (2006)

[21] OSTERWEIL, L.: Software processes are software too. In: *Proceedings of the 9th international conference on Software Engineering*, 1987, ISBN 0-89791-216-0, S. 2–13

[22] KLINGLER, CAROL DIANE: A case study in process definition. In: *Proceedings of the conference on TRI-Ada '93*. Seattle, Washington, United States, ACM, 1993, ISBN 0-89791-621-2, S. 65–79

[23] ARMENISE, P. ; BANDINELLI, S. ; GHEZZI, C. ; MORZENTI, A.: A survey and assessment of software process representation formalisms. In: *International Journal of Software Engineering and Knowledge Engineering* Bd. 3 (1993), S. 401–401

[24] CURTIS, BILL ; KELLNER, MARC I. ; OVER, JIM: Process modeling. In: *Commun. ACM* Bd. 35 (1992), Nr. 9, S. 75–90

[25] FINKELSTEIN, ANTHONY ; KRAMER, JEFF ; HALES, MATTHEW: Process Modelling: A Critical Analysis. In: *Integrated Software Reuse: management and techniques* (1992), S. 137–148

[26] KELLNER, M.I. ; ROMBACH, H.D.: Session Summary Comparisons of Software Process Descriptions. In: *Proceedings of the 6th International Software Process Workshop: Support for the Software Process*. Hakodate, Japan, IEEE Computer Society Press, 1990, S. 7–18

[27] ACUNA, S. T ; FERRÉ, X.: Software process modelling. In: *Proc. World Multiconf. on Systemics, Cybernetics and Informatics, Orlando, Florida, USA* (2001), S. 237–242

[28] SUTTON, S. M ; HEIMBIGNER, D. ; OSTERWEIL, L.J.: *APPL/A: a prototype language for software-process programming* (Technical Report Nr. CU-CS-448-89). Boulder, University of Colorado at Boulder, 1989

[29] FELDMANN, HARRY: *Programmieren mit Ada: Ein einführendes Lehrbuch*. Braunschweig [u.a.], Vieweg, 1992, ISBN 3528052058 9783528052058

[30] BANDINELLI, S.C. ; FUGGETTA, A. ; GHEZZI, C.: Software process model evolution in the SPADE environment. In: *IEEE Transactions on Software Engineering* Bd. 19 (1993), Nr. 12, S. 1128–1144

[31] BANDINELLI, SERGIO ; FUGGETTA, ALFONSO ; GHEZZI, CARLO ; LAVAZZA, LUIGI: SPADE: an environment for software process analysis, design, and enactment. In: *Software process modelling and technology*, Research Studies Press Ltd., 1994, ISBN 0-86380-169-2, S. 223–247

[32] MIN, S. -Y. ; LEE, H. -D. ; BAE, D. -H.: SoftPM: a software process management system reconciling formalism with easiness. In: *Information and Software Technology* Bd. 42 (2000), Nr. 1, S. 1–16

[33] WANG, QING ; XIAO, JUNCHAO ; LI, MINGSHU ; NISAR, M. ; YUAN, RONG ; ZHANG, LEI: A Process-Agent Construction Method for Software Process Modeling in SoftPM. In: *Software Process Change*, 2006, S. 204–213

[34] JACCHERI, L. ; LARSEN, J.O. ; CONRADI, R.: Software Process Modeling and Evolution in EPOS. In: *Proceedings of the Fourth International Conference on Software Engineering and Knowledge Engineering*, 1992, S. 574–581

[35] NGUYEN, M. N ; WANG, A. I ; CONRADI, R.: Total Software Process Model Evolution in EPOS. In: *Submitted paper for 4th ICSP*, 1996

[36] KAISER, G.E. ; BARGHOUTI, N.S. ; SOKOLSKY, M.H.: Preliminary experience with process modeling in the MARVEL software development environment kernel. In:

Proceedings of the Twenty-Third Annual Hawaii International Conference on System Sciences. Bd. 2, 1990, S. 131–140

[37] JUNKERMANN, G. ; PEUSCHEL, B. ; SCHÄFER, W. ; WOLF, S.: *MERLIN: Supporting cooperation in software development through a knowledge based environment*, *Advanced Software Development*. Bd. Software process modelling and technology, Research Studies Press Ltd., 1995, ISBN 0-86380-169-2

[38] HUFF, K.E. ; LESSER, V.R.: A plan-based intelligent assistant that supports the software development process. In: *ACM SIGSOFT Software Engineering Notes*. Bd. 13, 1989, S. 97–106

[39] TERNITÉ, THOMAS: *Variability of Development Models - An approach for the adaption of development models*. Clausthal-Zellerfeld, TU Clausthal, 2010

[40] *Webseite IBM - Rational Unified Process (RUP)*. URL http://www-01.ibm.com/software/awdtools/rup/. - abgerufen 2012-05-03

[41] *Webseite IBM - RUP - Rational Method Composer - Software*. URL http://www-01.ibm.com/software/awdtools/rmc/. - abgerufen 2012-05-03

[42] BARTELT, CHRISTIAN ; FISCHER, EDWARD ; TERNITÉ, THOMAS: Paradigmen zur Variabilitätsbeschreibung von Vorgehensmodellen. In: *INFORMATIK 2009 - Im Focus das Leben*, *GI-Lecture Notes in Informatics*, 2009, ISBN 978-3-88579-248-2, S. 3507–3521

[43] *Webseite SPEM*. URL http://www.omg.org/spec/SPEM/2.0/. - abgerufen 2012-05-07

[44] FRIEDRICH, JAN ; KUHRMANN, MARCO ; TERNITÉ, THOMAS: Erweitertes Tailoring und verbesserte organisationsspezifische Anpassung mit dem neuen V-Modell XT Metamodell. In: *Proceedings des 15. Workshop der Fachgruppe WI-VM der Gesellschaft für Informatik e.V. (GI), Vorgehensmodelle und der Product Life-cycle*. Aachen, Germany, Shaker Verlag, 2008, ISBN 978-3-8322-7123-7, S. 226–240

[45] KUHRMANN, MARCO ; TERNITÉ, THOMAS: Das V-Modell XT - Neuerungen für Anwender und Prozessingenieure. In: *Proceedings des 16. Workshop der Fachgruppe WI-VM der Gesellschaft für Informatik e.V. (GI), Vorgehensmodelle und Implementierungsfragen*. Aachen, Germany, Shaker Verlag, 2009, ISBN 978-3-8322-8109-0, S. 97–108

[46] TERNITÉ, THOMAS ; KUHRMANN, MARCO: *Das V-Modell XT 1.3 Metamodell* (Technischer Bericht Nr. TUM-I0905). München, TU München, 2009

[47] MÜNCH, J.: *Inter-View-Konsistenzen von MVP-L-Modellen (Diplomarbeit)*. Kaiserslautern, TU Kaiserslautern, Diplomarbeit, 1995

[48] LOTT, C. M. ; ROMBACH, H. D.: Measurement-based guidance of software projects using explicit project plans. In: *Information and Software Technology* Bd. 35 (1993), Nr. 6-7, S. 407–419

[49] ROMBACH, H. D. ; BRÖCKERS, A. ; LOTT, C. M. ; VERLAGE, M.: Entwicklungsumgebungen zur Unterstützung qualitätsorientierter Projektpläne. In: *Softwaretechnik-Trends: Mitteilungen der GI-Fachgruppen 'Software-Engineering'und 'Requirements-Engineering* Bd. 13 (1993), Nr. 3, S. 1–8

[50] BRÖCKERS, A. ; LOTT, C. M. ; VERLAGE, M. ; ROMBACH, H. D.: *MVP-L language report version 2* (Technischer Bericht Nr. 265/95). TU Kaiserslautern, 1995

[51] MÜNCH, J.: Anpassung von Vorgehensmodellen im Rahmen ingenieurmässiger Softwarequalitätssicherung. In: *Tagungsband des 6. Workshops der Fachgruppe 5.1.1 (GI)*. Kaiserslautern, 1999

[52] YOON, KYUNG-A ; MIN, SANG-YOON ; BAE, DOO-HWAN: Model-Based Project Process Analysis Using Project Tracking Data. In: *Software Engineering Research and Applications*, 2004, S. 148–167

[53] YOON, IL-CHUL: Tailoring and Verifying Software Process. In: *Eighth Asia-Pacific Software Engineering Conference (APSEC'01)*. Macao, IEEE Press, 2001, S. 202–209

[54] *Webseite Dokumentation Rational Unified Process (RUP)*. URL http://publib.boulder.ibm.com/infocenter/rmchelp/v7r5m1/index.jsp?topic=%2Fcom.ibm.rmc.help.doc%2Ftopics%2Fa_product_overview.html. - abgerufen 2012-05-03

[55] JACCHERI, M. L ; CONRADI, R.: Techniques for process model evolution in EPOS. In: *IEEE Transactions on Software Engineering*, *12*. Bd. 19, 1993, S. 1145–1156

[56] NGUYEN, M.N. ; WANG, A.I. ; CONRADI, R.: Total software process model evolution in EPOS: experience report. In: *Proceedings of the 19th international conference on Software engineering*, 1997, S. 390–399

[57] CUGOLA, G.: Tolerating deviations in process support systems via flexible enactment of process models. In: *IEEE Transactions on Software Engineering* Bd. 24 (1998), Nr. 11, S. 982 –1001

[58] CUGOLA, G. ; GHEZZI, C.: Design and implementation of PROSYT: a distributed process support system. In: *IEEE Proceedings of the 8th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*. Stanford, 1999, S. 32–39

[59] LBATH, RÉDOUANE ; COULETTE, BERNARD ; CRÉGUT, XAVIER: A Multi-Agent Approach to a SPEM-based Modeling and Enactment of Software Development Processes. In: CHU, W. C. ; JUZGADO, N. J. ; WONG, W. E. (Hrsg.): *7th Int. Conf. on Software Engineering and Knowledge Engineering (SEKE)*. Taipei, Taiwan, 2005, ISBN 1-891706-16-0, S. 241–246

[60] ALPAYDIN, ETHEM: *Maschinelles Lernen*. München, Oldenbourg, 2008, ISBN 978-3-486-58114-0

[61] SUN, R ; GILES, CL: Sequence learning: from recognition and prediction to sequential decision making. In: *IEEE Intelligent Systems* Bd. 16 (2001), Nr. 4, S. 67–70

[62] DAVISON, B. D ; HIRSH, H.: Predicting sequences of user actions. In: *Notes of the AAAI/ICML 1998 Workshop on Predicting the Future: AI Approaches to Time-Series Analysis*, 1998

[63] DAVISON, B. D ; HIRSH, H.: Probabilistic Online Action Prediction. In: *Proceedings of the AAAI Spring Symposium on Intelligent Environments*, 1998

[64] JACOBS, N. ; BLOCKEEL, H.: Sequence prediction with mixed order Markov chains. In: *In Proceedings of the Belgian/Dutch Conference on Artificial Intelligence*, 2002

[65] GORNIAK, P. ; POOLE, D.: Predicting future user actions by observing unmodified applications. In: *Proceedings of the National Conference on Artificial Intelligence*, 2000, S. 217–222

[66] STUMPF, S. ; BAO, X. ; DRAGUNOV, A. ; DIETTERICH, T.G. ; HERLOCKER, J. ; JOHNSRUDE, K. ; LI, L. ; SHEN, J.: Predicting user tasks: I know what you're doing. In: *20th National Conference on Artificial Intelligence (AAAI-05), Workshop on Human Comprehensible Machine Learning*, 2005

[67] GOPALRATNAM, K. ; COOK, D. J: Active LeZi: An incremental parsing algorithm for sequential prediction. In: *Proceedings of the Florida Artificial Intelligence Research Symposium*, 2003, S. 38–42

[68] GOPALRATNAM, K. ; COOK, D. J: Online sequential prediction via incremental parsing: The Active LeZi algorithm. In: *IEEE Intelligent Systems* Bd. 22 (2007), Nr. 1, S. 52–58

[69] DAS, S. K ; COOK, D. J ; BATTACHARYA, A. ; HEIERMAN III, E. O ; LIN, T. Y: The role of prediction algorithms in the MavHome smart home architecture. In: *Wireless Communications, IEEE* Bd. 9 (2003), Nr. 6, S. 77–84

[70] BHATTACHARYA, A. ; DAS, S. K: LeZi-update: an information-theoretic approach to track mobile users in PCS networks. In: *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, 1999, ISBN 1581131429, S. 1–12

[71] JAKKULA, V. ; COOK, D. J.: Anomaly detection using temporal data mining in a smart home environment. In: *Methods of information in medicine* Bd. 47 (2008), Nr. 1, S. 70–75

[72] RAO, S. P ; COOK, D. J: Improving the Performance of Action Prediction through Identification of Abstract Tasks. In: *Proceedings of the 16th International FLAIRS-2003 Conference*, 2003, S. 43–47

[73] JAKKULA, V. ; COOK, D. J: Learning temporal relations in smart home data. In: *Proceedings of the second International Conference on Technology and Aging*. Kanada, 2007

[74] BICKEL, S. ; HAIDER, P. ; SCHEFFER, T.: Learning to complete sentences. In: *Machine Learning: ECML 2005* (2005), S. 497–504

[75] GRABSKI, K. ; SCHEFFER, T.: Sentence completion. In: *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, 2004, ISBN 1581138814, S. 433–439

[76] WEISS, G. M ; HIRSH, H.: Learning to predict rare events in event sequences. In: *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, 1998, S. 359–363

[77] *Welie.com - Patterns in Interaction Design*. URL http://www.welie.com/. - abgerufen 2012-09-05

[78] HARTMANN, M. ; SCHREIBER, D.: Prediction algorithms for user actions. In: *Proceedings of Lernen Wissen Adaption, ABIS* (2007), S. 349–354

[79] ZIV, J. ; LEMPEL, A.: A universal algorithm for sequential data compression. In: *IEEE Transactions on Information Theory* Bd. 23 (1977), Nr. 3, S. 337–343

[80] ZIV, J. ; LEMPEL, A.: Compression of individual sequences via variable-rate coding. In: *IEEE Transactions on Information Theory* Bd. 24 (1978), Nr. 5, S. 530–536

[81] KÜNZER, A. ; OHMANN, F. ; SCHMIDT, L.: Antizipative modellierung des benutzerverhaltens mit Hilfe von Aktionsvorhersage-Algorithmen. In: *MMI-Interaktiv* Bd. 7 (2004), S. 61–83

[82] LEVENSHTEIN, V. I: Binary codes capable of correcting deletions, insertions, and reversals. In: *Soviet physics doklady*. Bd. 10, 1966, S. 707–710

[83] COOK, D. J ; YOUNGBLOOD, M. ; HEIERMAN III, E. O ; GOPALRATNAM, K. ; RAO, S. ; LITVIN, A. ; KHAWAJA, F.: MavHome: An agent-based smart home. In: *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, 2003, S. 521–524

[84] RAO, S. P ; COOK, D. J: Predicting inhabitant action using action and task models with application to smart homes. In: *International Journal on Artificial Intelligence Tools* Bd. 13 (2004), Nr. 1, S. 81–100

[85] GNATZ, MICHAEL ; MARSCHALL, FRANK ; POPP, GERHARD ; RAUSCH, ANDREAS ; SCHWERIN, WOLFGANG: *Enabling a Living Software Development Process with Process Patterns* (Technischer Bericht Nr. TUM-I0310). München, Technische Universität Müchen, 2003

[86] COAD, P.: Object-oriented patterns. In: *Communications of the ACM* Bd. 35 (1992), Nr. 9, S. 152–159

[87] COAD, P ; MAYFIELD, M ; NORTH, D: *Object models : strategies, patterns and applications*. Upper Saddle River, Yourdon Press, 1997, ISBN 0138401179 9780138401177

[88] BALZERT, HELMUT: *Lehrbuch der Software-Technik. 1, Basiskonzepte und requirements engineering*. Heidelberg [u.a.], Spektrum, Akad. Verl., 2009, ISBN 3827417058 9783827417053

[89] *OMG Unified Modeling Language (OMG UML) - Superstructure, Version 2.4.1*. URL http://www.omg.org/spec/UML/2.4/Superstructure

[90] RUPP, CHRIS ; QUEINS, STEFAN ; SOPHISTEN, DIE: *UML 2 glasklar: Praxiswissen für die UML-Modellierung*. 4., aktualisierte und erweiterte. Aufl., Carl Hanser Verlag GmbH & CO. KG, 2012, ISBN 3446430571

[91] ESHUIS, R. ; WIERINGA, R.: *A formal semantics for UML activity diagrams – Formalising workflow models* (Technischer Bericht Nr. CTIT-01-04). Enschede, University of Twente, 2001

[92] TRICKOVIC, IVANA: Formalizing Activity Diagram of UML by Petri Nets. In: *Journal of Mathematics, Novi. Sad 30*, *3*. (2000), S. 161–171

[93] STÖRRLE, HARALD ; HAUSMANN, JAN HENDRIK: Towards a Formal Semantics of UML 2.0 Activities. In: *P. Liggesmeyer, K. Pohl, and M. Goedicke, editors, Software Engineering*. Bd. 64, 2005, S. 117–128

[94] STÖRRLE, HARALD: Semantics of UML 2.0 Activities. In: *IEEE Symposium on Visual Languages and Human Centric Computing*, 2004, S. 235–242

[95] *OMG Object Constraint Language (OCL), Version 2.3.1*. URL http://www.omg.org/spec/OCL/2.3.1

[96] BARGHOUTI, NASER S. ; KAISER, GAIL E.: Scaling Up Rule-Based Software Development Environments. In: *Proceedings of the 3rd European Software Engineering Conference*, Springer-Verlag, 1991, ISBN 3-540-54742-8, S. 380–395

[97] DEYNET, MICHAEL: User-Centric Process Descriptions. In: *Proceedings of the 3rd International Conference on Software Technology and Engineering*. Kuala Lumpur, Malaysia, ASME Press, 2011, ISBN 978-0-7918-5979-7, S. 209–214

[98] DEYNET, MICHAEL: Predicting User Actions in Software Processes. In: *4th Workshop on Intelligent Techniques in Software Engineering at the European Conference on Machine Learning and Principles and Practices of Knowledge Discovery in Databases (ECML-PKDD)*. Athen, Ägypten, 2011

11 ABBILDUNGSVERZEICHNIS