



ulm university universität
ulm

Universität Ulm | 89069 Ulm | Germany

**Fakultät für
Ingenieurwissenschaften
und Informatik**
Institut für Datenbanken
und Informationssysteme

Vergleich der Anwendbarkeit von deklarativen und imperativen Prozessmodellierungsansätzen im Kontext von Softwareentwicklungsprozessen

Masterarbeit an der Universität Ulm

Vorgelegt von:

Bianka Hampp

bianka.hampp@uni-ulm.de

Gutachter:

Manfred Reichert

Vera Künzle

Betreuer:

Gregor Grambow

2014

Fassung 23. November 2014

© 2014 Bianka Hampp

This work is licensed under the Creative Commons. Attribution-NonCommercial-ShareAlike 3.0 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/3.0/de/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.

Satz: PDF-L^AT_EX 2_ε

Kurzfassung

Abstract

Inhaltsverzeichnis

1. Einleitung	1
1.1. Einleitung	1
1.1.1. Motivation	1
1.1.2. Zielstellung	1
1.1.3. Aufbau der Arbeit	1
2. Prozessmodelle	3
2.1. Software Engineering	4
2.1.1. Ziele, Prozess und Prinzipien des Software Engineering	4
2.2. Softwareprozessmodelle	7
2.2.1. Software-Projekttypen	8
2.2.2. Schwer- und Leichtgewichtige Prozessmodelle	9
3. Modellierung	11
3.1. Prozessmodellierung	12
3.1.1. Ziele der Prozessmodellierung	12
3.1.2. Grundsätze ordnungsgemäßer Modellierung	13
3.2. Prozessmodellierungssprachen	16
3.2.1. Imperative Modellierung	17
3.2.2. Deklarative Modellierung	21
3.3. Modellierungswerzeuge	23
3.3.1. Signavio	24
3.3.2. Declare	25

Inhaltsverzeichnis

4. Anforderungserhebung	29
4.1. Vergleichskriterien	29
4.1.1. Erfüllung der Modellierungsgrundsätze	30
5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse	37
5.1. Scrum	38
5.1.1. Analyse Scrum	40
5.1.2. Imperative Modellierung Scrum	41
5.1.3. Deklarative Modellierung Scrum	44
5.1.4. Vergleich	46
5.2. Open Unified Process (Open UP)	50
5.2.1. Analyse Open UP	50
5.2.2. Imperative Modellierung Open UP	56
5.2.3. Deklarative Modellierung Open UP	69
5.2.4. Vergleich	78
5.3. V-Modell XT	87
5.3.1. Analyse V-Modell XT	88
5.3.2. Imperative Modellierung V-Modell	98
5.3.3. Deklarative Modellierung V-Modell	102
5.3.4. Vergleich	107
5.4. Vergleich insgesamt	112
6. Validierung	117
6.1. Forschungsfragen	117
6.2. Design der Studie	118
6.2.1. Durchführung der Studie	121
6.2.2. Fazit der Studie	132
7. Verwandte Arbeiten	133
7.1. Modellierung von Software-Engineering Prozessmodellen	133
7.2. Verständlichkeit von Prozessmodellierungssprachen	134

Inhaltsverzeichnis

7.3. Vergleich von Prozessmodellierungssprachen	135
8. Zusammenfassung und Ausblick	137
A. BPMN Notation	139
B. ConDec Notation	143

1

Einleitung

1.1. Einleitung

1.1.1. Motivation

1.1.2. Zielstellung

Die vorliegende Arbeit verfolgt mehrere Ziele.

1.1.3. Aufbau der Arbeit

Eine Übersicht über den Aufbau dieser Arbeit gibt Abbildung. Zunächst werden in Kapitel 2 und 3 grundlegende Begriffe erläutert, welche für das Verständnis der vorliegenden Arbeit notwendig sind.

1. Einleitung

Kapitel 2 liefert zunächst einen Einblick in Prozessmodelle. In Kapitel 2.1 wird der Begriff Software Engineering eingeführt und es werden die Ziele, der Prozess sowie die Prinzipien des Software Engineering vorgestellt. Anschließend werden in Kapitel 2.2 Softwareprozesse erläutert. Hierfür werden Software-Projekttypen sowie Schwergewichtige und Leichtgewichtige Prozessmodelle definiert.

In Kapitel 3 erfolgt eine Einführung in die Grundlagen der Modellierung. Zum Einen werden Prozessmodellierung, die Ziele der Prozessmodellierung sowie die Grundsätze ordnungsgemäßer Modellierung vorgestellt. Zum Anderen erfolgt eine allgemeine Einführung in Prozessmodellierungssprachen und vor allem werden die in dieser Arbeit verwendete Imperative und Deklarative Modellierung erklärt. Des Weiteren werden noch die in der vorliegenden Arbeit verwendeten Modellierungswerzeuge vorgestellt.

Die Anforderungserhebung erfolgt in Kapitel 4. Hier werden in Kapitel 4.1 die Vergleichskriterien für die beiden Prozessmodellierungssprachen erläutert.

Die imperative und deklarative Modellierung für Software Engineering Prozessmodelle erfolgt in Kapitel 5. Hier wird das Software Engineering Prozessmodell Scrum in Kapitel 5.1 zunächst eingeführt, anschließend analysiert, imperativ und deklarativ modelliert und die beiden Modellierungen werden miteinander verglichen. In Kapitel 5.2 wird zunächst der Open Unified Process (Open UP) vorgestellt, es folgt eine Analyse desselben und es werden imperative und deklarative Modelle des Open UP erstellt und miteinander verglichen. Das V-Modell XT wird in Kapitel 5.3 erläutert, analysiert, imperativ und deklarativ modelliert und die jeweiligen Modelle werden miteinander verglichen. In Kapitel 5.4 erfolgt ein insgesamter Vergleich der imperativen und deklarativen Modelle.

Die Validierung der Ergebnisse aus Kapitel 5 wird in Kapitel 6 durchgeführt. In Kapitel 7 widmet sich verwandten Arbeiten zur vorliegenden Arbeit. Zunächst werden in Kapitel 7.1 verwandte Arbeiten zur Modellierung von Software Engineering Prozessmodellen gegenüber der vorliegenden Arbeit abgegrenzt. Weiterhin werden in Kapitel 7.2 Arbeiten über die Verständlichkeit von Prozessmodellierungssprachen und in Kapitel 7.3 Arbeiten über den Vergleich von Prozessmodellierungssprachen dargelegt und der Thematik dieser Arbeit gegenüber gestellt.

Kapitel 8 fasst die gesamt Arbeit nochmals zusammen und gibt weiterhin einen Ausblick auf zukünftige Forschung in dieser Thematik.

2

Prozessmodelle

In Kapitel 2 werden grundlegende Konzepte des Software Engineering vorgestellt, welche notwendig sind, um den Inhalt dieser Arbeit besser zu verstehen. Zunächst wird in Kapitel 2.1 der Begriff Software Engineering definiert und die Ziele, der Prozess und die Prinzipien des Software Engineering werden erläutert. Weiterhin wird in Kapitel 2.2 der Begriff Softwareprozessmodell erklärt. Hierbei werden Software- Projekttypen sowie schwergewichtige und leichtgewichtige Prozessmodelle beschrieben. Anschließend gibt es eine Einführung in die drei Softwareprozessmodelle Scrum, Open Unified Process und V-Modell-XT.

2. Prozessmodelle

2.1. Software Engineering

Heutzutage werden immer mehr Systeme von Software kontrolliert. Dies macht Software Engineering zu einer der bedeutendsten Technologien [Pun07]. Unter Software versteht man laut Duden die "Gesamtheit aller Programme, die auf einem Computer eingesetzt werden können". Das Wort Engineering, welches sich laut Duden von dem lateinischen Wort Ingenium (=schöpferische) Begabung; Erfindungsgabe) ableitet, wird heutzutage mit Ingenieurwesen, bzw. technische Entwicklung übersetzt. Software Engineering umfasst somit die Gesamtheit der Aktivitäten zur Analyse, Konzeption, Entwicklung und Implementierung einer softwaretechnischen Lösung [Spe98]. Software Engineering besteht aus mehreren Schichten (Abbildung 2.1):



Abbildung 2.1.: Schichten des Software Engineering [Pun07]

Somit sind für Software Engineering ein diszipliniertes Qualitätsmanagement sowie eine Prozessschicht vorhanden, um die termingerechte Ablieferung von Software zu gewährleisten. In der Methoden-Schicht wird sodann die Implementierung unter Zuhilfenahme von Anforderungsanalysen, Design und Programmierung durchgeführt. Hierbei werden Werkzeuge zur Automatisierung in Software- Dokumentenprozessen benutzt [Pun07].

2.1.1. Ziele, Prozess und Prinzipien des Software Engineering

Das Hauptziel in der Software Entwicklung ist, dass die Lösungen mit den Anforderungen übereinstimmen. Vollständige und konsistente Anforderungserhebungen sind, insbesondere für große Systeme, selten. Sowohl die Nutzer, als auch die Entwickler haben ein oftmals unvollständiges Verständnis des eigentlichen Problems und erheben ihre Anforderungen erst während der Entwicklung. Somit muss man mit Änderungen der

2.1. Software Engineering

Anforderungen an ein System während dessen Entwicklung rechnen. Aus diesem Grund ist es wichtig, Ziele beim Software Engineering zu haben, um die Auswirkungen solcher Änderungen einzudämmen [DB93]. Abbildung 2.2 zeigt die von [RGI75] definierten Ziele, Prinzipien und den Prozess des Software Engineering, welche nachfolgend genauer erläutert werden:

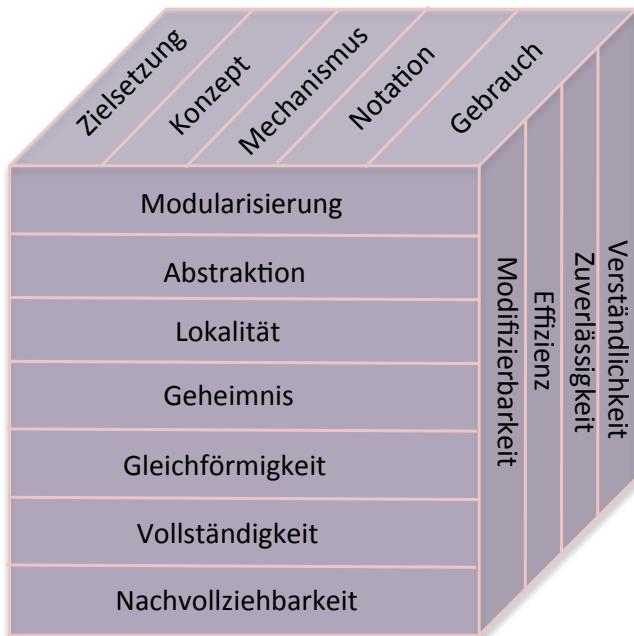


Abbildung 2.2.: Ziele, Prozess und Prinzipien des Software Engineering [RGI75]

Prinzipien des Software Engineering

Das *Modularisierungsprinzip* gibt eine geeignete Strukturierung für Softwaresysteme an. Das *Abstraktionsprinzip* soll dabei helfen, sich von unwichtigen Details, welche für die zu entwickelnde Lösung irrelevant sind, zu lösen. Das *Lokalitätsprinzip* verlangt das räumlich zusammenhängende Ablegen von zusammengehörenden Informationen. Das *Geheimnisprinzip* bezieht sich auf das Definieren und Durchsetzen von Zugriffsbeschränkungen. Konsistenz wird durch das *Gleichförmigkeitsprinzip* gewährleistet. Durch

2. Prozessmodelle

das *Vollständigkeitsprinzip* wird sichergestellt, dass nichts vergessen wurde. Das Prinzip der *Nachvollziehbarkeit* stellt sicher, dass Informationen, welche zur Überprüfung der Korrektheit benötigt werden, detailliert dargelegt werden [RGI75].

Prozess des Software Engineering

Wie Abbildung 2.2 entnommen werden kann, besteht der Prozess des Software Engineering aus 5 Schritten: Im ersten Schritt *Zielsetzung* werden die Anforderungen an ein System erhoben. Anschließend erfolgt im Schritt *Konzept* die Ableitung der Software-Architektur, um die zuvor erhobenen Anforderungen zu erfüllen. Des Weiteren werden die Komponenten des Software-Systems festgelegt. Im dritten Schritt *Mechanismus* erfolgt sodann die Implementierung des Software-Systems. Im darauffolgenden Schritt *Notation* wird die Kommandosprache definiert, die ein Benutzer verwendet, um die Funktionalitäten des Software-Systems aufzurufen. Im letzten Schritt *Gebrauch* muss noch die Bedienung des Systems, z.B. in Form eines Benutzerhandbuchs, beschrieben werden [RGI75].

Ziele des Software Engineering

Modifizierbarkeit ist das wohl schwierigste Ziel des Software Engineering. Hierbei geht es darum, dass es manchmal notwendig ist, Teile des zu entwickelnden Systems zu ändern, während andere Teile unverändert bleiben, aber dennoch das gewünschte neue Ergebnis erreicht wird. Auf die *Effizienz* der jeweiligen Aktivitäten sollte immer geachtet werden, da dieses Ziel des Software Engineering häufig vernachlässigt wird. Bei dem Ziel *Zuverlässigkeit* geht es darum, einerseits Fehler bei der Konzeption, im Design und der Implementierung zu vermeiden, andererseits muss auch Fehlverhalten bei der Ausführung und der Leistung verhindert werden [RGI75].

2.2. Softwareprozessmodelle

Für das Verständnis, die Schaffung oder Unternehmung von etwas Großem, fertigen Menschen in der Regel ein vereinfachtes Bild davon an, bzw. nehmen Maß, fertigen eine Skizze oder einen Plan an oder orientieren sich an einem Vorbild, bzw. bauen sich eines. Dies geschieht normalerweise mit Papier und Schreibzeug, anderen Materialien oder einem Computer. Besonders für die Lösung von komplexen wissenschaftlichen Problemen oder für die Erfüllung großer Führungs- und Konstruktionsaufgaben ist dies unumgänglich [HM08].

Hierbei stützen sie sich auf Modelle, welche als Stellvertreter für die Sache, die verstanden, geschaffen, unternommen oder betrieben werden soll, angesehen werden kann [HM08].

Insbesondere die heutzutage von Softwareentwicklern zu erstellenden Softwareprodukte zeichnen sich durch ein hohes Maß an Komplexität und Umfang aus. Neben den Erwartungen von Kunden hinsichtlich Qualität müssen Softwaresysteme ebenfalls termingerecht und innerhalb eines vorgegebenen Budgetrahmens erstellt werden. Effektive und effiziente Softwareprozessmodelle gewinnen somit immer mehr Bedeutung [GBBK10]. Modell leitet sich von dem lateinischen Begriff „modelus“ ab und kann mit „Regel, Form, Muster, Vorbild“ übersetzt werden [HM08]. Der Begriff Prozess stammt von dem lateinischen Wort "processus" ab und lässt sich mit "Fortgang oder Verlauf" übersetzen [Koc11, Sta06].

Ein Softwareprozess ist eine Abfolge von Schritten, welche zur Herstellung von Software notwendig sind [MM12, Stö05]. Mit Hilfe eines Softwareprozessmodells lässt sich der organisatorische Rahmen zur Herstellung von Software beschreiben [Kö00]. Ein Softwareprozessmodell stellt somit ein Modell für die Entwicklung eines Software-Systems dar [Han10]. Die einzelnen Abschnitte eines Softwareprozesses werden hierbei als Phasen bezeichnet [Stö05]. Diese werden unterschieden in (Abbildung 2.3):

In einem Softwareprozessmodell werden nicht nur die durchzuführenden Aktivitäten definiert, sondern auch die Rollen und Qualifikationen der Mitarbeiter, welche die jeweiligen Aktivitäten durchführen sollen, bzw. für diese verantwortlich sind. Des Weiteren werden

2. Prozessmodelle

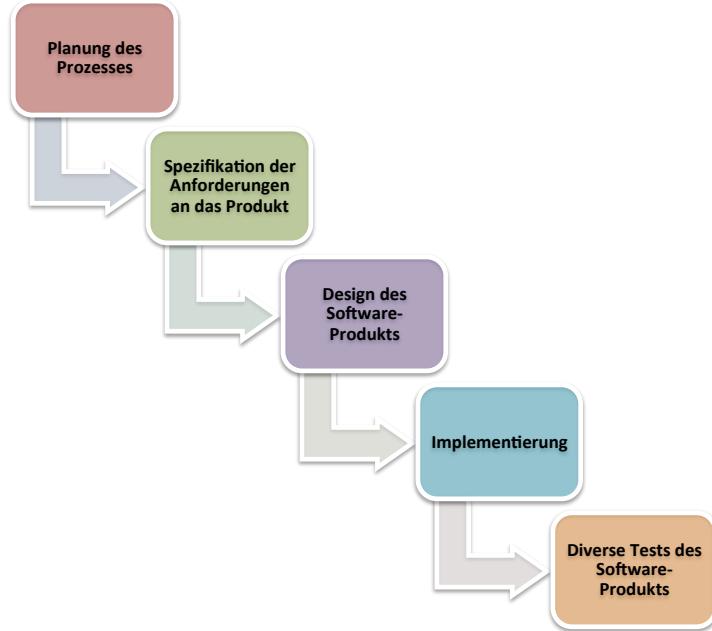


Abbildung 2.3.: Phasen Softwareprozess nach [Han10]

die während des Entwicklungsprozesses zu erstellenden Dokumente und Unterlagen festgelegt [Han10].

2.2.1. Software-Projekttypen

Software-Projekte lassen sich in drei Gruppen einteilen (Abbildung 5.45):

Bei den *Einfachen Projekten* sind relativ kleine Teams am Entwicklungsprozess beteiligt und bei den Teammitgliedern besteht räumliche Nähe. Jedes Teammitglied weist eine hohe methodische und fachliche Erfahrenheit auf und kennt sich in dem späteren Einsatzgebiet der Software gut aus. Die Anzahl der Code-Zeilen bei der zu entwickelnden Software ist meist gering [Boe81, Han10].

Bei den *Komplexen Projekten* handelt es sich um Software-Projekte, welche in den meisten Fällen stark durch behördliche Auflagen reguliert sind. Die Software muss einerseits eine hohe Zuverlässigkeit aufweisen und andererseits sind nachträgliche Änderungen fast nicht mehr möglich. Im Gegensatz zu den *Einfachen Projekten* ist das

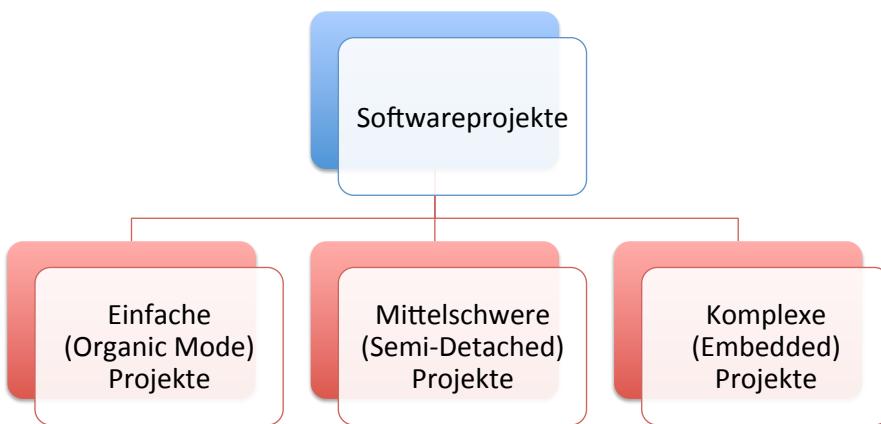


Abbildung 2.4.: Software-Projekttypen nach [Boe81]

Entwicklungsteam hier groß, besteht sowohl aus erfahrenen, als auch aus unerfahrenen Entwicklern und die Anzahl der Code-Zeilen ist ebenfalls groß [Boe81, Han10].

Eine Schnittstelle zwischen diesen beiden Projekttypen bilden die *Mittelschweren Projekte*. Hier sind die Software-Entwicklungsteams mittelgroß und bestehen aus erfahrenen und unerfahrenen Mitgliedern. Teilweise sind nicht alle Aspekte des Produktes schon im Vornherein bekannt und die Anzahl der Code-Zeilen ist groß [Boe81, Han10].

2.2.2. Schwerge wichtige und Leichtgewichtige Prozessmodelle

Aus der eben erfolgten Einteilung von Software-Projekten lässt sich eine Einteilung von Software-Prozessmodellen in *Leichtgewichtige* und *Schwerge wichtige Prozessmodelle* ableiten [Han10].

Leichtgewichtige Prozessmodelle eignen sich eher für kleine Teams, bei denen keine detaillierte Anforderungserhebung stattfindet, da die Kommunikation sowohl innerhalb des Teams, als auch mit dem Kunden auf Grund der kleinen Teamgröße gut funktioniert. Da viele Informationen hier informell über kurze Kommunikationswege weitergegeben werden, ist eine ausführliche Dokumentation derer nicht notwendig. Der Einsatz von *Leichtgewichtigen Prozessmodellen* eignet sich sehr gut für *Einfache Projekte* und teilweise auch für *Mittelschwere Projekte* [Han10].

2. Prozessmodelle

Eine sehr formale und dokumentenlastige Vorgehensweise kommt bei den *Schwerge wichtigen Prozessmodellen* zum Einsatz. Es findet eine ausführliche Dokumentation in allen Entwicklungsphasen statt und der Ablauf des Prozesses ist genau vorgegeben. Bei Software-Produkten, welche bei einer möglichen Fehlfunktion Menschenleben in Gefahr bringen, ist beispielsweise eine Vorgehensweise mit einem *Schwer gewichtigen Prozessmodell* sinnvoll. Ihr Einsatz ist besonders in *Schweren Projekten* vorzuziehen, aber auch in *Mittelschweren Projekten* [Han10].

3

Modellierung

Kapitel 3 liefert einen Überblick über die Grundlagen der Modellierung. Zunächst werden in Kapitel 3.1 die Grundlagen der Prozessmodellierung erläutert. Hierbei wird auf die Grundsätze ordnungsgemäßer Modellierung eingegangen. Anschließend werden in Kapitel 3.2 Prozessmodellierungssprachen diskutiert. Einerseits werden imperative Modellierungssprachen erklärt und es wird ein kurzer Einblick in die Prozessmodellierungssprache BPMN gegeben. Andererseits werden deklarative Prozessmodellierungssprachen vorgestellt und es erfolgt ein Einblick in die deklarative Prozessmodellierungssprache ConDec. In Kapitel 3.3 werden die in dieser Arbeit für die imperative und deklarative Modellierung verwendeten Modellierungswerkzeuge vorgestellt.

3. Modellierung

3.1. Prozessmodellierung

Prozessmodellierung hat den Zweck, Prozesse zu beschreiben [FLM⁺09]. Ein Prozessmodell ist eine vereinfachte Darstellung eines Prozesses und besteht aus einer Abfolge von Tätigkeiten, welche chronologisch-sachlogisch angeordnet sind. Der Umfang und Detaillierungsgrad der Prozessmodelle kann sich je nach Zweck und Zielsetzung unterscheiden [Koc11].

3.1.1. Ziele der Prozessmodellierung

Mit der Modellierung von Prozessen werden verschiedene Ziele verfolgt. Eine erste Übersicht über die Ziele der Prozessmodellierung gibt Abbildung 3.1 [Koc11].



Abbildung 3.1.: Ziele der Prozessmodellierung nach [Koc11]

Bei der *Transparenz* geht es darum, dass alle Beteiligten am Prozess einsehen können, von wem welche Aufgaben durchgeführt werden. Weiterhin verfolgt die Prozessmodellierung das Ziel, durch *Fehlervermeidung* die Qualität, Termintreue und Kundenzu-

3.1. Prozessmodellierung

friedenheit zu erhöhen. Durch die Modellierung eines Prozesses kann dieser genau analysiert werden und hierdurch können Einsparungspotenziale von *Kosten* aufgedeckt werden. Indem die Abläufe in einem Unternehmen als Prozesse dargestellt werden, ist es möglich, eine *personenunabhängige Verfügbarkeit des Wissens* zu erreichen, da das Wissen hierdurch allen Personen zugänglich gemacht wird, unabhängig davon, ob sie am Prozess beteiligt sind oder nicht. Die Prozessmodellierung führt zu einer *erleichterten Einarbeitung neuer Mitarbeiter*. Durch die Darstellung der Tätigkeiten der einzelnen Mitarbeiter in Prozessmodellen wird ihnen ihr Beitrag zum Erfolg des Unternehmens vor Augen geführt, was eine *erhöhte Mitarbeitermotivation* zur Folge hat. Nach deren Erstellung gibt es verschiedene *Auswertungsmöglichkeiten* für die Prozessmodelle. Durch die Modellierung von Prozessen werden etwaige Schwachstellen, wie z.B. Doppelarbeiten und Prozessverzögerungen offengelegt, wodurch eine *Prozessoptimierung* möglich ist. Mit Hilfe von *Simulationen* der Prozessmodelle lassen sich eventuelle Engpässe rechtzeitig erkennen. Die Voraussetzung für die *Zertifizierung* nach DIN EN ISO 9000:1000 sind Prozessmodelle als Dokumentation. Basis für die Entwicklung von Softwaresystemen bilden Prozessmodelle, weshalb sie als *Basis für die informationstechnische Unterstützung* dienen [Koc11].

3.1.2. Grundsätze ordnungsgemäßer Modellierung

Bei der Gestaltung eines Modells sollten grundlegende Prinzipien beachtet werden, um die Qualität eines Modells zu sichern. Hierfür gibt es die Grundsätze ordnungsgemäßer Modellierung, über deren Prinzipien Abbildung 3.2 einen Überblick gibt [Fre07].

Der *Grundsatz der Richtigkeit* besitzt zwei verschiedene Ausprägungen: Eine syntaktische und eine semantische. Die syntaktische *Richtigkeit* eines Modells wird durch die Einhaltung der Notationsregeln der dem Modell zugrunde liegenden Prozessmodellierungssprache erreicht [BRS95, Bec12].

Ein Modell wird als semantisch korrekt, oder auch formal korrekt bezeichnet, wenn es dem ihm zugrunde liegenden Metamodell gegenüber vollständig und konsistent ist, d.h. es gibt den abzubildenden Sachverhalt korrekt wieder. Hierbei muss einerseits auf die

3. Modellierung



Abbildung 3.2.: Grundsatz ordnungsgemäßer Modellierung nach [BRS95]

3.1. Prozessmodellierung

korrekte Abbildung der Struktur des Metamodells, als auch des dort beschriebenen Verhaltens geachtet werden [BRS95, Bec12].

Modelle werden üblicherweise in getrennten Sichten modelliert, um die Komplexität so gering wie möglich zu halten. Bespielsweise werden Prozesse in einem Prozessmodell, die Daten aber in einem Datenmodell modelliert. Werden bei einer Modellierung mehrere Sichten (z.B. Organisationssicht, Datensicht, Funktionssicht) modelliert, müssen diese auch ineinander integriert werden. Beim *Grundsatz des systematischen Aufbau* geht es darum, bei der Modellierung auch auf die anderen Sichten zu achten, um eine spätere konsistente Integration der verschiedenen Sichten zu gewährleisten. Insbesondere ist zu vermeiden, dass die gleichen Informationsobjekte mehrmals mit jeweils verschiedenen Begriffen verwendet werden. Weiterhin sollten die Eingabedaten eines Prozessmodells einen Verweis auf bestehende Datenmodelle enthalten [BRS95, Fre07, Bec12, Koc11].

Der *Grundsatz der Relevanz* besagt, dass alle Elemente und Verknüpfungen eines Modells, ohne die der Nutzen des Modells sinken würde, für die Modellierung relevant sind [BRS95, Rei09]. Auf der anderen Seite sollten aber auch nur diejenigen Teile der Realität in das Modell aufgenommen werden, die wirklich notwendig sind. Es sollte somit darauf geachtet werden, nur so viele Informationen ins Modell zu bringen wie minimal benötigt werden [BRS95, Fre07, Rei09].

Durch den *Grundsatz der Klarheit* soll sichergestellt werden, dass das Modell für den Adressaten verständlich ist. Es muss also bei der Modellierung auf Strukturiertheit, Verständlichkeit und Anschaulichkeit geachtet werden. Insbesondere sollte das Modell ohne besondere methodische Kenntnisse verständlich sein. Somit sollte die Modellierung entweder von links nach rechts oder von oben nach unten verlaufen, wobei darauf zu achten ist, dass sich Flusslinien und Kanten hierbei so wenig wie möglich überkreuzen. Weiterhin sollte die Anzahl der Elemente auf das Nötigste reduziert werden. Vor allem die Anzahl an Verzweigungen innerhalb eines Prozessmodells wirkt sich negativ auf die Verständlichkeit von Prozessmodellen aus. Ebenso hat eine hohe Anzahl

3. Modellierung

von Verbindungen zwischen Aktivitäten einen negativen Einfluß auf das Verständnis [Lei12, BRS95, Fre07, Rei09, Bec12, Koc11, MRC, Pes08].

Der *Grundsatz der Wirtschaftlichkeit* sagt aus, dass die Modellierung kosteneffektiv durchzuführen ist [Lei12]. Es gilt also abzuwägen, ob der Aufwand, der für die Modellierung notwendig ist, auch einen entsprechenden Nutzen bringt [Fre07, BRS95].

Wird in unterschiedlichen Modellen der gleiche Sachverhalt abgebildet, so sollten letztendlich auch vergleichbare Modelle entstehen, unabhängig von der verwendeten Modellierungssprache. Dies besagt der *Grundsatz der Vergleichbarkeit*. Insbesondere ist auf einen einheitlichen Abstraktionsgrad der Prozessmodelle zu achten [Lei12, BRS95, Fre07, Rei09].

3.2. Prozessmodellierungssprachen

Die Modellierung eines Prozesses mit natürlicher Sprache bringt einige Nachteile mit sich, wie z.B. fehlende Eindeutigkeit, schwer zu überprüfende Vollständigkeit und teilweise Widersprüche. Mögliche Folgen davon können unterschiedliche Interpretationen, Missverständnisse und falsche Schlussfolgerungen sein. Eine reine Beschreibung der Prozessmodelle mit mathematischen Modellen und Formalismen führt jedoch oftmals zu einer Verminderung der intuitiven Verständlichkeit der Prozessmodelle. Aus diesem Grund ist es sinnvoll den Prozess graphisch als Diagramm mit einer Prozessmodellierungssprache darzustellen, da diese eine Schnittstelle zwischen formaler Exaktheit und intuitiver Verständlichkeit darstellen [Tho09, Kir06].

Hierfür existieren eine Reihe verschiedener Prozessmodellierungssprachen, deren Vor- und Nachteile intensiv diskutiert werden. Ein viel diskutierter Unterschied ist der zwischen imperativen und deklarativen Prozessmodellierungssprachen [FLM⁺09].

Die ursprüngliche Unterscheidung zwischen imperativen und deklarativen Sprachen stammt aus der Programmierung. Während imperative Programmierung angibt, "Wie

3.2. Prozessmodellierungssprachen

etwas zu tun ist", folgt deklarative Programmierung dem Ansatz "sag was benötigt wird und lass das System herausfinden, wie es erreicht werden kann" [PWZ⁺12].

3.2.1. Imperative Modellierung

Imperative Programmierung wird als zustandsbehaftete Programmierung bezeichnet, da das Ergebnis einer Komponente nicht nur von ihren Argumenten abhängt, sondern auch von internen Parametern, was auch als ihr "Zustand" bezeichnet wird [FLM⁺09].

Ähnlich wie die imperative Programmierung, folgt auch die imperative Modellierung einem "Inside-Out-Ansatz". Alle Ausführungsalternativen eines Prozesses sind somit in diesem spezifiziert und alle weiteren Ausführungsalternativen müssen explizit hinzugefügt werden. Bei der imperativen Modellierung werden Prozesse mit Operatoren und elementaren Aktivitäten modelliert. Hierbei können Sequenz, Parallelität und Synchronisation beschrieben werden [Kas98]. Bei einer imperativen Modellierungssprache liegt der Fokus auf den ständigen Veränderungen der Prozess-Objekte.

BPMN

Die *Business Process Modelling Notation (BPMN)* wurde von der *Business Process Management Initiative* entwickelt und 2004 veröffentlicht. Seit 2005 wird sie von der *Object Management Group* standardisiert und weiterentwickelt [KBL13]. Die BPMN-Elemente lassen sich anhand der fünf Kategorien *Swimlanes*, *Flussobjekte*, *verbindende Objekte*, *Daten* und *Artefakte* einteilen. Abbildung 3.3 zeigt die Einteilung und die wichtigsten Prozess-Elemente von BPMN, welche nachfolgend genauer erläutert werden [GL12].

In der Kategorie **Swimlanes** befinden sich *Pools* und *Lanes*. *Pools* stellen eine Art Container für den Prozess dar. Ein *Pool* ist ein Prozessteilnehmer. Ein Prozessteilnehmer ist z.B. eine Organisationseinheit oder eine selbstständige Geschäftseinheit. Werden in einem Prozessmodell mehrere *Pools* verwendet, so können hiermit Kollaborationen zwischen verschiedenen Prozessteilnehmern dargestellt werden. Ein *Pool* kann in mehrere

3. Modellierung

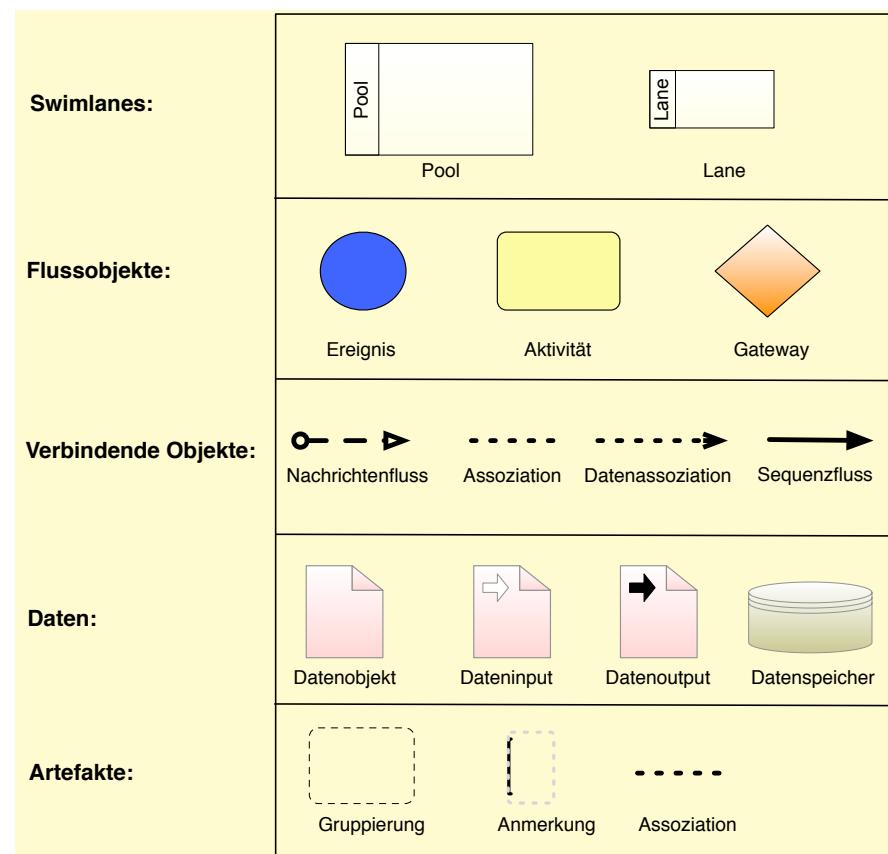


Abbildung 3.3.: BPMN-Elemente Übersicht nach [GL12]

3.2. Prozessmodellierungssprachen

Lanes unterteilt werden. *Lanes* können untergeordnete Organisationseinheiten, Partnerrollen (z.B. Vertrieb, Projektleitung, Marketing) oder auch verschiedene Bestandteile eines Systems sein [GL12, Pit10, All13].

Ereignisse, *Aktivitäten* und *Gateways* befinden sich in der Kategorie **Flussobjekte**. Start und Ende von Prozessen werden in BPMN durch *Ereignisse* beschrieben. Diese werden in *Startereignisse* und *Endereignisse* unterschieden und geben somit den Beginn und das Ende eines Prozesses an. Weiterhin gibt es auch noch *Zwischenereignisse*. Hierdurch können beispielsweise Pausen im Prozess modelliert werden. Der Prozess stoppt in diesem Fall solange, bis ein bestimmtes Ereignis eintritt [All13].

Aktivitäten stellen Arbeitseinheiten dar und sind ein Oberbegriff für Aufgaben, Unterprozesse und Aufruf-Aktivitäten. Aufgaben sind Tätigkeiten, welche nicht weiter unterteilt werden können, während ein Unterprozess eine Aufgabe darstellt, welche in weitere Tätigkeiten unterteilt werden kann. Beschriftet werden sie mit einer Objekt-Verb-Verbindung (z.B. Lieferung überprüfen) [GL12].

Mit Hilfe von *Gateways* lässt sich der Prozessablauf kontrollieren und steuern, da durch diese Verzweigungen und Zusammenführungen von Sequenzflüssen dargestellt werden. [GL12, All13]. Hierbei werden *Exklusive Gateways* zur Modellierung alternativer Pfade, *Parallele Gateways* zur Modellierung parallel ablaufender Pfade, *Inklusive Gateways* zur Modellierung der Auswahl eines oder mehrerer Pfade und *Komplexe Gateways* zur Modellierung komplexer Regeln bei Verzweigungen und Zusammenführungen, unterschieden [All13].

3. Modellierung

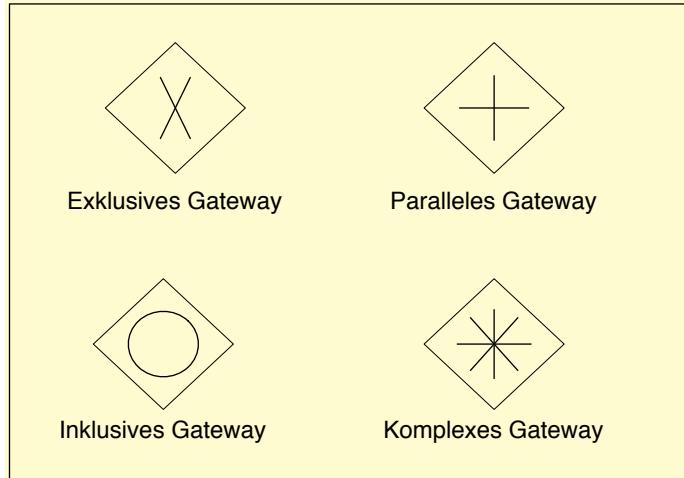


Abbildung 3.4.: BPMN-Gateways

Nachrichtenfluss, Assoziation, Datenassoziation und Sequenzfluss bilden zusammen die Kategorie **Verbindende Objekte**. Ein *Nachrichtenfluss* wird dazu verwendet, den Nachrichtenfluss zwischen zwei getrennten Prozessteilnehmern, z.B. aus zwei verschiedenen Unternehmen, darzustellen. Mit Hilfe einer *Assoziation* können Daten, Text und andere Artefakte mit Flussobjekten verknüpft werden. Hiermit werden die In- und Ausgabes von Aktivitäten aufgezeigt. Ein *Sequenzfluss* dient dazu die Reihenfolge der Aktivitäten im Prozess festzulegen [Whi04].

In der Kategorie **Daten** gibt es sich *Datenobjekte*, *DatenInput*, *DatenOutput* und *Datenspeicher*. *Datenobjekte* geben hierbei an, welche Daten von den Aktivitäten benötigt werden, bzw. von diesen erzeugt werden [Whi04]. Sie stellen somit Informationen dar, welche durch den Prozess fließen. Bei einem *Dateninput* handelt es sich um einen externen Eingabe für den ganzen Prozess, der von einer Aktivität gelesen wird. Ein *Datenoutput* hingegen wird als Ergebnis eines ganzen Prozesses erzeugt. Somit handelt es sich bei *Dateninput*, bzw. *Datenoutput* um Eingangs-, bzw. Ausgangsprozessschnittstellen [Mou14a]. Ein *Datenspeicher* kann für den indirekten Austausch von Daten zwischen zwei verschiedenen Prozessteilnehmern verwendet werden. Hierfür ist es notwendig, dass beide Prozessteilnehmer Zugriff auf den *Datenspeicher* haben [All13].

Die Kategorie **Artefakte** beinhaltet *Gruppierung*, *Anmerkung* und *Assoziation*. Diese

ergänzen den Prozess um zusätzliche Informationen, haben jedoch keinerlei Einfluss auf diesen [GL12]. Eine *Gruppierung* kann hierbei zur Dokumentation oder für Analysezwecke benutzt werden. Durch eine *Anmerkungen* können dem Leser zusätzliche Informationen in Textform bereit gestellt werden [Whi04]. Mit Hilfe einer *Assoziation* lassen sich Datenobjekte mit Aktivitäten und Prozessen verknüpfen [Mou14a].

Eine Übersicht über alle Elemente der BPMN Notation kann Anhang A entnommen werden.

3.2.2. Deklarative Modellierung

Die deklarative Modellierung folgt im Gegensatz zur imperativen Modellierung einem "Outside-In-Ansatz" [Lic12]. Das heißt, deklarative Sprachen legen den Ablauf nicht im Vorhinein fest [PWZ⁺12] und sie sind somit sehr flexibel [RW12]. Zu Beginn befinden sich nur die Aktivitäten im Prozessmodell und erlauben jegliches Ausführungsverhalten. Erst wenn Constraints zum Modell hinzugefügt werden, werden schrittweise Ausführungsalternativen verworfen [PWZ⁺12]. Constraints lassen sich hierbei in die beiden verschiedenen Kategorien **Ausführungsconstraints** und **Terminierungsconstraints** einteilen. Die Ausführungsconstraints geben Einschränkungen für die Ausführung von Aktivitäten an. Hierbei kann es sich z.B. um die Anzahl möglicher Ausführungen für eine Aktivität oder eine Mindestzeitverzögerung zwischen zwei Aktivitäten handeln. Terminierungsconstraints hingegen führen auf, wann eine korrekte Terminierung (Beendigung) des Prozesses möglich ist. Es kann hier z.B. vorgeschrieben werden, dass eine Aktivität mindestens einmal ausgeführt werden muss oder dass der Aktivität A Aktivität B folgen muss. Bevor dies nicht geschehen ist, ist kein korrektes Ende des Prozesses möglich [RW12]. Abbildung 3.5 zeigt ein Beispiel für ein deklaratives Prozessmodell. Es besteht aus den drei Aktivitäten A,B und C sowie aus zwei Constraints: Das Constraint zwischen A und B legt fest, dass Aktivität B Aktivität A vorausgehen muss und das Constraint bei Aktivität C legt fest, dass diese mindestens einmal ausgeführt werden muss, aber beliebig oft ausgeführt werden kann. Abgesehen von diesen Bedingungen, können die Aktivitäten sowohl beliebig oft, als auch in beliebiger Reihenfolge ausge-

3. Modellierung

führt werden. Es wäre z.B. [A,B,C,C,A,B,C] eine korrekte Ausführungsreihenfolge. Die Ausführungsreihenfolgen [C,B,C,A] oder [A,B,A,B] wären jedoch inkorrekt, da bei der ersten Ausführungsreihenfolge B vor A ausgeführt wird und somit das Constraint zwischen diesen beiden Aktivitäten verletzt würde. Bei der zweiten Ausführungsreihenfolge wird Aktivität C nicht ausgeführt, wodurch das Constraint verletzt wird, so dass diese mindestens einmal auszuführen ist [RW12].

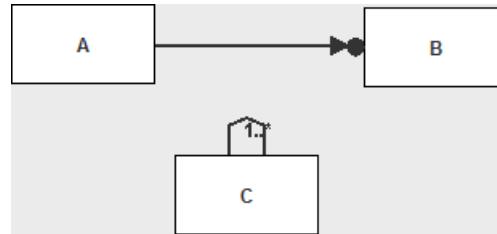


Abbildung 3.5.: Deklarativer Beispiel-Prozess [PA06]

ConDec

Die deklarative Modellierungssprache ConDec wurde erstmals unter dem Namen DecSerFlow veröffentlicht [FMR⁺10]. Mit ConDec lassen sich einerseits sehr strenge Modelle erstellen, welche den gesamten Prozess im Detail vorgeben und andererseits sehr leichtgewichtige Modelle, welche zwar angegeben, welche Arbeit getan werden muss, aber nicht wie sie ausgeführt werden muss [PA06].

In ConDec gibt es die vier verschiedenen Arten von Constraints: *Existence*, *Choice*, *Relation* und *Negation*. Tabelle 3.1 zeigt die Bedeutung der verschiedenen Constraints.

Eine Übersicht über die genaue Notation von ConDec ist in Anhang B verfügbar.

Constraint	Erläuterung
Existence Constraints	Ein-stellige Kardinalitäts-Constraints. Sie geben an, wie oft eine Aktivität ausgeführt werden kann, bzw. muss.
Choice Constraints	N-stellige Constraints. Sie geben die Notwendigkeit der Ausführung von Aktivitäten an, die zu einer Reihe möglicher Alternativen gehören, unabhängig von anderen Constraints.
Relation Constraints	Zwei-stellige Constraints. Sie geben vor, dass eine gewisse Aktivität ausgeführt werden muss, falls eine andere Aktivitäten ausgeführt wird. Es können auch qualitative zeitliche Constraints zwischen diesen beiden Aktivitäten verlangt werden.
Negation Constraints	Stellt die negative Version der Relation Constraints dar. Sie verbieten explizit die Ausführung einer gewissen Aktivität, wenn eine andere Aktivität ausgeführt wird.

Tabelle 3.1.: Constraints ConDec [PA06]

3.3. Modellierungswerkzeuge

Ein Modellierungswerkzeug ist ein Softwaresystem, mit dessen Hilfe sich Prozessmodelle erstellen lassen. Teilweise bietet ein Modellierungswerkzeug noch weitere Funktionen wie z.B. das Ausführen und Monitoring der Prozesse, Simulationen und die Analyse von Prozessmodellen an. Die Ausführung der Prozessschritte kann hierbei durch die jeweilige Person, welche für die Aktivität zuständig ist, ausgeführt werden. Für die Prozessmodellierung in der vorliegenden Arbeit kommt das Modellierungswerkzeug Signavio für die imperative Modellierung mit BPMN und Declare für die deklarative Modellierung mit ConDec zum Einsatz. Diese beiden Modellierungswerkzeuge werden nachfolgend vorgestellt [Gad12].

3. Modellierung

3.3.1. Signavio

Bei Signavio handelt es sich um ein webbasiertes Prozessmodellierungstool, welches auch das kollaborative Modellieren von Prozessen mit den Modellierungsstandards BPMN und EPC zulässt. Ein grosser Vorteil von Signavio besteht darin, dass es nicht auf dem Rechner installiert werden muss, sondern direkt im Web-Browser ausgeführt werden kann. Die Prozessmodelle werden in einem zentralen Repository gespeichert und sind für die Benutzer entsprechend ihren Zugriffsrechten aufrufbar. Prozessmodelle besitzen alle eine eigene eindeutige URL und können über diese im Web-Browser aufgerufen werden. Hierbei wird auch gleich die Modellierungsumgebung mitgeladen und kann somit im Web-Browser ausgeführt werden [MRW12]. Abbildung 3.6 zeigt den *Signavio Process Editor*.

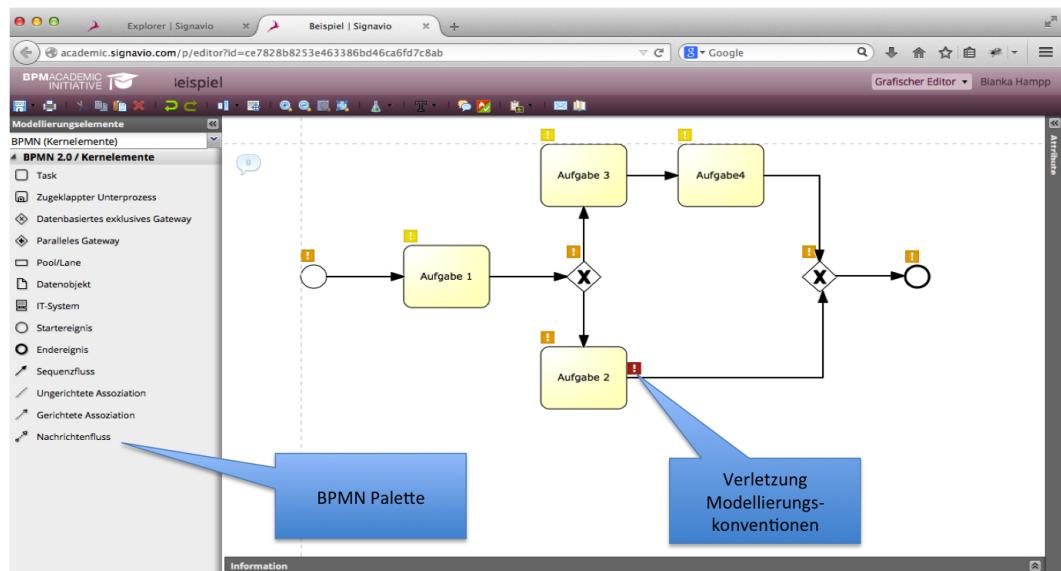


Abbildung 3.6.: Siganvio Process Editor (Screenshot Siganvio)

Links in Abbildung 3.6 ist die BPMN Palette zu sehen. Die einzelnen Elemente können per *Drag and Drop* in das Arbeitsdokument gezogen werden. Signavio verfügt über Modellierungskonventionen. Mit diesen ist es möglich, das Modell auf die Einhaltung von Modellierungsrichtlinien, wie z.B. Notationsumfang, Benennung, Prozessstruktur

3.3. Modellierungswerkzeuge

und Diagrammlayout zu überprüfen. Die Modelle können alle als PDF exportiert werden. In Abbildung 3.7 ist die Simulations-Sicht von Signavio zu sehen. Hier kann der Benutzer den Prozessablauf simulieren. Dies kann einerseits mit Benutzerinteraktion Schritt für Schritt erfolgen oder auch im Ganzen durch den Simulator gesteuert werden, wobei XOR-Verzweigungen nach wie vor vom Benutzer ausgewählt werden müssen.

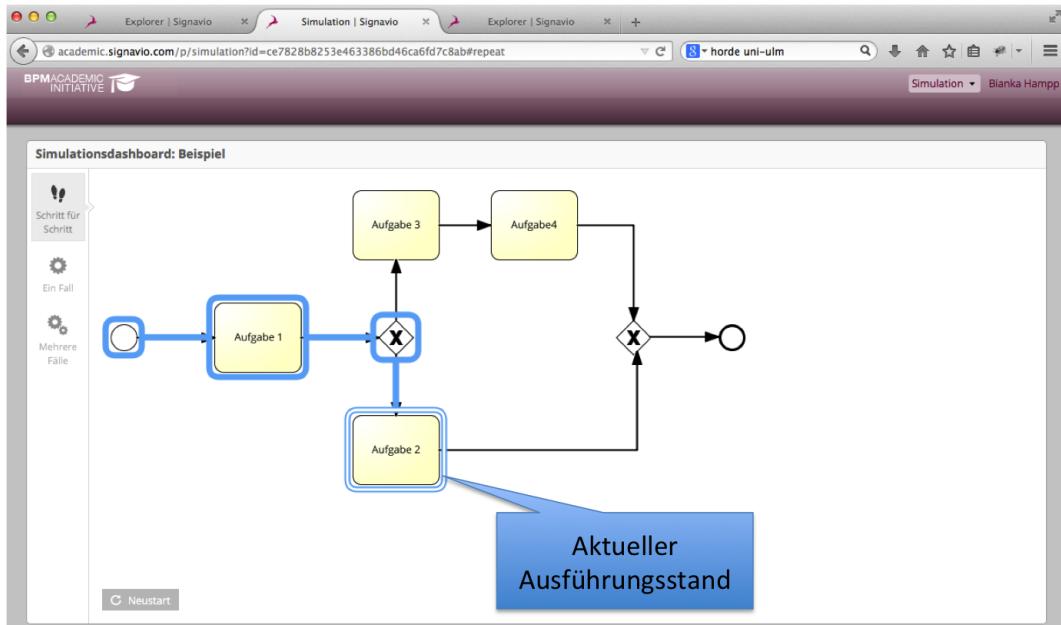


Abbildung 3.7.: Siganvio Simulation (Screenshot Signavio)

3.3.2. Declare

Declare wurde als Constraint-basiertes Workflow-Management-System entwickelt. Es wird für die Entwicklung von Prozessmodellen, welche auf deklarativen Sprachen basieren, benutzt. Declare bietet die folgenden Funktionen an [PSA07]:

- Modellentwicklung
- Modellüberprüfung (Suche nach Fehlern in Modellen)
- automatisierte Modellausführung
- Modelle können zur Laufzeit geändert werden

3. Modellierung

- Analyse der bereits ausgeführten Prozesse
- Prozess Dekomposition

Abbildung 3.8 zeigt die Systemarchitektur von *Declare*.

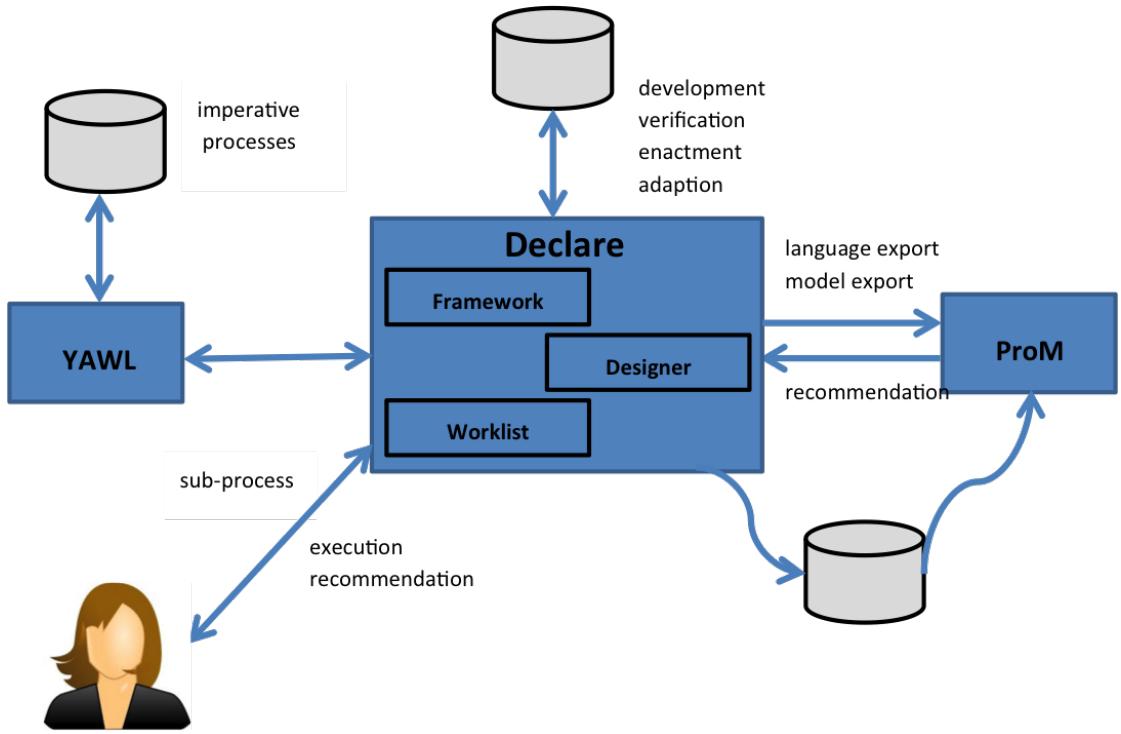


Abbildung 3.8.: Declare Systemarchitektur nach [PSA07]

Hieraus wird ersichtlich, dass *Declare* mit den beiden Systemen *YAWL* und *ProM* kooperiert. Bei *YAWL* handelt es sich um ein Workflow-Management System, welches auf strukturierte Workflows spezialisiert ist. Dies wirkt sich auf die Zusammenarbeit mit *Declare* in der Art aus, dass die strukturierten Teile des Prozesses von *YAWL* abgehandelt werden, während die unstrukturierten Teile von *Declare* übernommen werden. Bei *ProM* handelt es sich um ein Prozess-Mining-Tool. Hier werden bereits ausgeführte Prozesse von *Declare* analysiert und darauf aufbauend werden dem Nutzer während der Prozessausführung Empfehlungen gegeben [PSA07].

3.3. Modellierungswerkzeuge

Weiterhin besteht *Declare* selbst aus drei Komponenten *Framework*, *Designer* und *Worklist*. Beim *Designer* handelt es sich um ein Modellierungstool, welches für Systemeinstellungen und die Prozessmodell-Entwicklung verwendet wird (Abbildung 3.9). Das *Framework* ist für das Prozess-Enactment (Prozessausführung) zuständig. Außerdem übernimmt es die Kommunikation mit *YAWL* und *ProM* und das Ändern der Prozessmodelle zur Laufzeit (Abbildung 3.10). Die Prozessausführung wird von *Worklist* durchgeführt. Hier können die Nutzer ihre zuvor erstellten Prozesse ausführen und können die von *ProM* erstellten Empfehlungen sehen (Abbildung 3.11). Alle Modelle können als Bilddateien exportiert werden [PSA07].

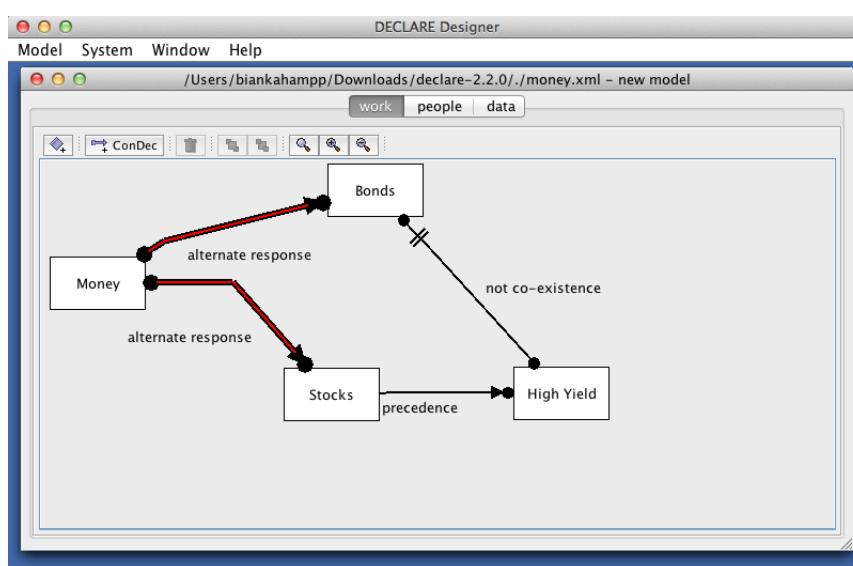


Abbildung 3.9.: Declare Designer (Screenshot aus Declare)

3. Modellierung

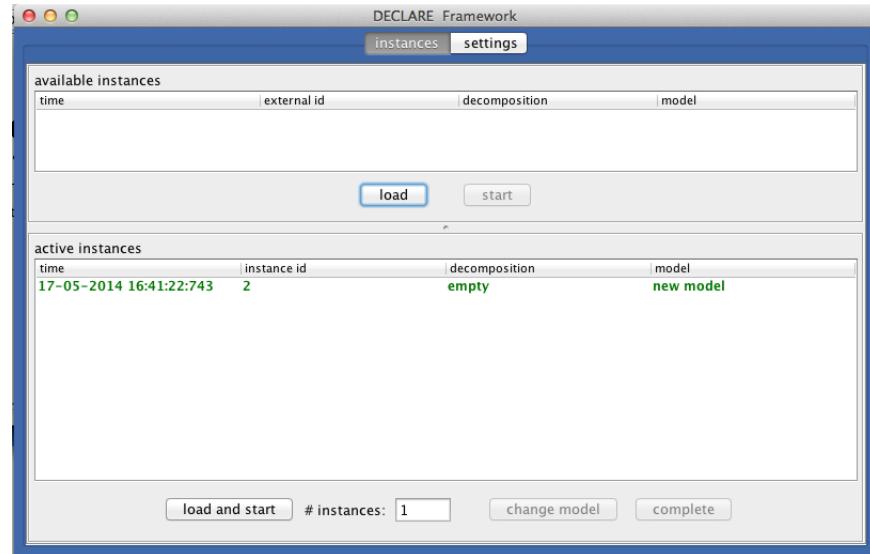


Abbildung 3.10.: Declare Framework (Screenshot aus Declare)

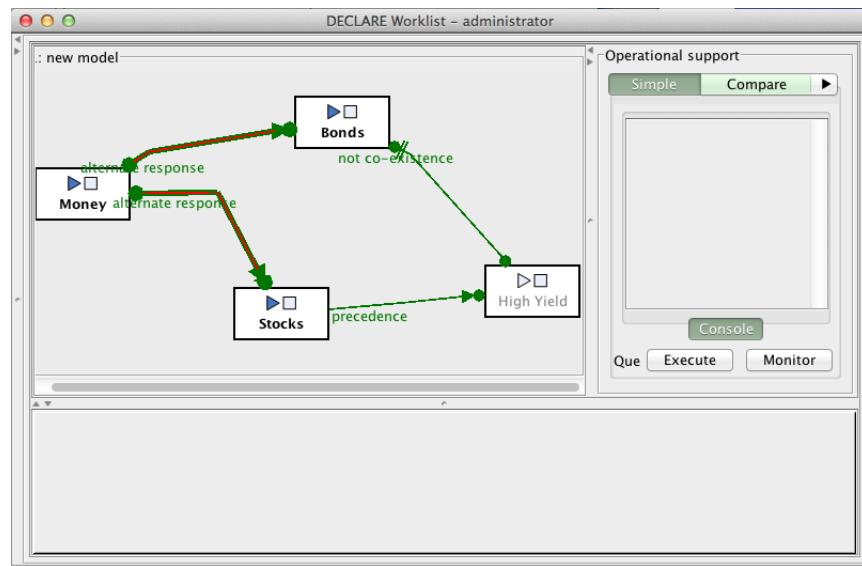


Abbildung 3.11.: Declare Worklist (Screenshot aus Declare)

4

Anforderungserhebung

In diesem Kapitel werden die Anforderungen an den in Kapitel 5 folgenden Vergleich der imperativen und deklarativen Modellierung für SE-Prozessmodelle erhoben. Hierfür werden in Kapitel 4.1 die Vergleichskriterien vorgestellt und erläutert.

4.1. Vergleichskriterien

Bisher gibt es nur wenige Arbeiten, welche sich mit deklarativen Prozessmodellierungssprachen und insbesondere mit dem Vergleich von imperativen und deklarativen Prozessmodellierungssprachen beschäftigen. Aus diesem Grund soll in der vorliegenden Arbeit ein Vergleich der Anwendbarkeit zwischen deklarativen und imperativen Prozessmodellierungssprachen im Kontext von Softwareentwicklungsprozessen durchgeführt werden. Hierbei soll die Eignung der beiden Prozessmodellierungssprachen für die

4. Anforderungserhebung

Modellierung von unterschiedlich großen Prozessmodellen beurteilt werden. Weiterhin sollen die Stärken und Grenzen der beiden Modellierungssprachen aufgezeigt werden und es soll dadurch herausgefunden werden, ob eine der beiden Modellierungssprachen über eine bessere Eignung zur Modellierung verfügt als die andere [LK06].

Hierfür sollen die imperativen und deklarativen Prozessmodelle, welche für die drei Softwareentwicklungsprozesse Scrum, Open UP und V-Modell-XT erstellt werden im Hinblick auf verschiedene Vergleichskriterien untersucht werden. Da es sich bei Scrum um ein leichtgewichtiges Softwareentwicklungsprozessmodell, beim V-Modell XT um ein schwergewichtiges Softwareentwicklungsprozessmodell und bei Open UP um ein Softwareentwicklungsprozessmodell handelt, welches sich in der Mitte zwischen leichtgewichtig und schwergewichtig befindet, eignen sich diese drei besonders gut, zum Vergleichen der imperativen und deklarativen Modellierung für unterschiedlich große Metamodelle. Außerdem liegen den in imperativer und deklarativer Modellierungssprache zu erstellenden Prozessmodellen so jeweils die gleichen Metamodelle zugrunde, was eine objektive Bewertung für den Vergleich gewährleistet [LK06].

4.1.1. Erfüllung der Modellierungsgrundsätze

Es sollen die imperativen und deklarativen Modellierungssprachen im Hinblick auf deren Erfüllung der in Kapitel 3.1.1 vorgestellten *Grundsätze ordnungsgemäßer Modellierung* untersucht werden, da durch deren Einhaltung die Qualität, Klarheit und Konsistenz der Prozessmodelle gesichert wird [Fre07]. Somit lässt sich hierdurch die Eignung der beiden Prozessmodellierungssprachen sehr gut überprüfen. Denn falls eine von beiden Prozessmodellierungssprachen die Modellierungsgrundsätze wesentlich schlechter einhalten kann, als die andere, so ist sie zum Modellieren deutlich weniger geeignet, da die hierdurch entstandenen Prozessmodelle geringere Qualität, Klarheit und Konsistenz aufweisen. Hierfür werden nachfolgend die erstellten Prozessmodelle im Hinblick auf *Richtigkeit, systematischen Aufbau, Relevanz, Klarheit, Wirtschaftlichkeit und Vergleichbarkeit* verglichen. Hierfür werden nachfolgend für jeden Modellierungsgrundsatz verschiedene Kriterien festgelegt, mit deren Hilfe die Einhaltung der Modellierungsgrund-

sätze für die jeweilige Modellierungssprache überprüft wird.

Richtigkeit

Die Richtigkeit der Prozessmodelle soll verglichen werden. Hierbei soll die syntaktische und semantische Richtigkeit der Prozessmodelle untersucht werden.

Die sysntaktische Korrektheit wird dahingegend untersucht, ob sich die jeweiligen Modelle unter Einhaltung der Modellierungsregeln der jeweiligen Prozessmodellierungssprache erstellen lassen. Dies wird mit Hilfe der Modellierungstools Signavio und Declare durchgeführt. Beide Programme verfügen über eine automatische Überprüfung der syntaktischen Korrektheit der dort erstellten Modelle.

Bei der semantischen Korrektheit der Prozessmodelle wird verglichen in wie weit die mit deklarativer bzw. imperativer Prozessmodellierungssprache erstellten Prozessmodelle dem zugrunde liegenden Metamodell gegenüber vollständig und konsistent sind. Denn falls wesentliche Aspekte des Metamodells nicht darstellbar sind, leidet der Nutzen des Prozessmodells erheblich. Es wird somit überprüft, ob eine der beiden Prozessmodellierungssprachen die Struktur des Metamodells und das dort beschriebe Verhalten besser abbildet, als die andere. Insbesondere wird hier untersucht, ob es Grenzen in der Darstellbarkeit der abzubildenden Aspekte des Metamodells gibt [BRS95, Bec12].

Systematischer Aufbau

Um den systematischen Aufbau der imperativen und deklarativen Prozessmodelle zu vergleichen, werden die Prozessmodelle dahingehend untersucht, in wie weit sie die Integration anderer Sichten in das Prozessmodell unterstützen und sie Verweise auf bestehende Datenmodelle zulassen. Da nicht alle Informationen, wie z.B. Daten und Funktionen in einem Prozessmodell abgebildet werden können, ist die Integration anderer Sichten in das Prozessmodell sehr wichtig, um wirklich alle Informationen aus dem Metamodell abbilden zu können. Hier können Rückschlüsse auf die Eignung zur Mo-

4. Anforderungserhebung

dellierung gezogen werden und eventuelle Grenzen der Prozessmodellierungssprache aufgezeigt werden [BRS95, Fre07, Bec12, Koc11].

Relevanz

Beim Vergleich der Relevanz der Prozessmodelle werden die mit BPMN bzw. ConDec modellierten Prozessmodelle dahingehend verglichen in wie weit es möglich ist die Prozessmodelle mit den minimal relevanten Informationen zu erstellen. Es soll also untersucht werden, ob bei einer der beiden Prozessmodellierungssprachen mehr Informationen im Prozessmodell abgebildet werden müssen, als bei der anderen, um die Qualität des Prozessmodells zu sichern. Weiterhin soll auch andersrum untersucht werden, ob bei einer der beiden Prozessmodellierungssprachen alle minimal relevanten Informationen des Metamodells dargestellt werden können. Hier kann wiederum die Eignung der beiden Prozessmodellierungssprachen sehr gut verglichen werden, da falls mit einer Prozessmodellierungssprache nicht alle minimal relevanten Informationen des Metamodells abgebildet werden können, ist diese nicht zum Modellieren geeignet.[BRS95, Fre07, Rei09].

Klarheit

Die Prozessmodelle, welche jeweils in imperativer und deklarativer Prozessmodellierungssprache erstellt werden, sollen im Hinblick auf ihre Klarheit untersucht werden. Hierbei soll festgestellt werden, ob es wesentliche Unterschiede bei der Verständlichkeit der Prozessmodelle gibt, wenn diese in imperativer, bzw. deklarativer Prozessmodellierungssprache erstellt wurden. Denn fehlende Verständlichkeit eines Prozessmodells führt dazu, dass das Prozessmodell wenig Nutzen bringt. Insbesondere soll hier bei den imperativen Modellen die Anzahl an Verbindungen bzw. bei den deklarativen Modellen die Anzahl an Constraints zwischen Aktivitäten verglichen werden, da sich diese mit steigender Anzahl negativ auf die Verständlichkeit auswirken[GL06a, Pes08, HBZ⁺14]. Hier soll auch insbesondere die Anzahl unterschiedlicher Gateways/Constraints betrachtet werden. Da alle Gateways in BPMN und alle Constraints in ConDec eine

unterschiedliche Semantik haben und diese jeweils verstanden werden muss, kann sich eine hohe Anzahl an verschiedenen Gateways, bzw. Constraints negativ auf die Verständlichkeit auswirken. Es existieren Studien über den geistigen Aufwand, welcher für das Verständnis von BPMN-Notationselementen notwendig ist. Den einzelnen Notationselementen werden verschiedene geistige Gewichtungen zugewiesen. Ein Sequenzfluss hat auf Grund des weniger hohen geistigen Aufwandes beim Verstehen eine geistige Gewichtung von 1. Das Exklusive Gateway hat eine geistige Gewichtung von 2, falls es nur zwei ausgehende Kanten hat. Bei drei oder mehr ausgehenden Kanten hat es eine geistige Gewichtung von 3. Das parallele Gateway hat eine geistige Gewichtung von 4. Einem Inklusiven Gateway wird sogar eine geistige Gewichtung von 7 zugeschrieben [GL06a, Pes08].

Leider existieren derzeit noch keine Metriken über den geistigen Aufwand beim Verstehen der einzelnen Constraints bei ConDec. Jedoch gibt es bereits Studien, welche sich anderweitig mit dem Verstehen von deklarativen Prozessmodellen auseinandersetzt haben. Da die Existenz-Constraints relativ einfach zu verstehen sind, werden sie beim Vergleich in Kapitel 6 mit den Sequenzflusselementen gleichgesetzt und haben somit auch in etwa eine geistige Gewichtung von 1. Da die Constraints in ConDec genau wie die Gateways in BPMN Patterns darstellen, müssten sie ebenfalls alle Werte für die geistige Gewichtung zwischen 2 und 7 annehmen. Da die Constraints jedoch nicht direkt den Gateways in BPMN zugeordnet werden können und sich somit die geistigen Gewichtungen der Gateways nicht auf die Constraints übertragen lassen wird im Vergleich in Kapitel 6 nur die jeweilige Anzahl an Gateways und Constraints im Modell für den Vergleich herangezogen. Zudem wird auch die Anzahl an unterschiedlichen Gateways/Constraints betrachtet. Denn sowohl bei BPMN, als auch bei ConDec wurde in Studien herausgefunden, dass gerade die Kombination von vielen verschiedenen Gateways/Constraints einen sehr großen negativen Einfluß auf das Verstehen hat [GL06a, Pes08, HBZ⁺14]. Weiterhin soll hier untersucht werden, ob es wesentliche Unterschiede in der Verständlichkeit der imperativen und deklarativen Prozessmodelle gibt, in Abhängigkeit der Größe des zugrunde liegenden Softwareentwicklungsprozessmodells. Hierbei kann die Eignung der jeweiligen Modellierungssprache sehr gut festgestellt werden, da sie im Falle von schwerer/fehlender Verständlichkeit nicht zum Modellieren geeignet ist. Falls es

4. Anforderungserhebung

Unterschiede in der Verständlichkeit der Prozessmodelle in Abhängigkeit der Größe des zugrunde liegenden Metamodells gibt, lassen sich hierbei Rückschlüsse auf die Eignung der Prozessmodellierungssprache in Bezug auf große/kleine Metamodelle ziehen [Lei12, BRS95, Fre07, Rei09, Bec12, Koc11, MRC, Pes08].

Wirtschaftlichkeit der Prozessmodelle

Hier soll untersucht werden, ob sich der Aufwand für die Modellierung bei den beiden Modellierungssprachen erheblich voneinander unterscheidet, da wenn die Erstellung eines Prozessmodells mit einem zu hohen Aufwand für die Erstellung verbunden ist, obwohl der spätere Nutzen des Prozessmodells erheblich geringer ist, ist die Modellierung nicht sinnvoll. Hier kann die Eignung zur Modellierung der Prozessmodellierungssprachen sehr gut verglichen werden, denn falls der Aufwand für die Modellierung für eine der beiden Prozessmodellierungssprachen weitaus höher ist, als für die andere, eignet sich die Prozessmodellierungssprache mit dem sehr viel höherem Aufwand nicht für die Modellierung.

Hier wird einerseits die Anzahl der Elemente insgesamt, welche zur Modellierung der Prozesse notwendig sind miteinander verglichen. Weiterhin wird die Anzahl von Gateways in BPMN mit der Anzahl von Constraints in ConDec miteinander verglichen und auch die Anzahl unterschiedlicher Gateways/Constraints. Da bei der Verwendung von Gateways/Constraints für den Modellierer ein höherer geistiger Aufwand notwendig ist und somit auch ein größerer Aufwand für das Modellieren, kann sich dies negativ auf die *Wirtschaftlichkeit* auswirken [Fre07, BRS95, Lei12].

Vergleichbarkeit

Bei der Vergleichbarkeit der Prozessmodelle wird untersucht, ob die in imperativer, bzw. deklarativer Prozessmodellierungssprache erstellten Prozessmodelle, welchen die gleichen Metamodelle zugrunde liegen, trotzdem vergleichbare Prozessmodelle darstellen. Es wird hier somit insbesondere untersucht, ob die Abstraktionsgrade der Prozessmodelle sich wesentlich voneinander unterscheiden. Weiterhin wird die Vergleichbarkeit in

4.1. Vergleichskriterien

Bezug auf das Ausführungsverhalten der imperativen und deklarativen Prozessmodelle durch Ausführung der Modelle in den Modellierungstools Siganvio und Declare nach der Modellierung überprüft. Hier werden die möglichen Pfade im jeweiligen Modell durchlaufen und somit wird sichergestellt, dass das Verhalten der Modelle gleich ist. Außerdem wird hier die Größe der jeweiligen Prozessmodelle als Vergleichskriterium herangezogen. Hier wird die Gesamtanzahl der notwendigen Elemente zur Darstellung des Prozessmodells verglichen. Es soll festgestellt werden, ob bei Verwendung einer imperativen oder deklarativen Prozessmodellierungssprache wesentlich mehr Elemente zur Darstellung des gleichen Prozesses notwendig sind indem die Anzahl der verwendeten Aktivitäten und Patterns verglichen wird [Lei12, BRS95, Fre07, Rei09].

Eine Übersicht über die Modellierungsgrundsätze und die jeweiligen Vergleichskriterien bietet Abbildung 4.1.

Modellierungsgrundsatz		Vergleichskriterien
Richtigkeit	syntaktisch	Einhaltbarkeit der Notationsregeln
	semantisch	Grenzen der Darstellbarkeit
Systematischer Aufbau		Unterstützung der Integration von anderen Sichten
Relevanz		Erstellung mit minimal relevanten Informationen möglich
Klarheit		Verständlichkeit, Anzahl Sequenzfluss/Existenz Constraints, Anzahl Gateways/Constraints, Anzahl unterschiedlicher Gateways/Constraints, Summe Elemente gesamt
Wirtschaftlichkeit		Aufwand für die Modellierung, Anzahl Sequenzfluss/ Existenz Constraints, Anzahl Gateways/Constraints, Anzahl unterschiedlicher Gateways/Constraints, Summe Elemente gesamt
Vergleichbarkeit		Ausführungsverhalten (Siganvio, Declare), Summe Elemente gesamt, Grenzen Darstellbarkeit

Abbildung 4.1.: Übersicht Vergleichskriterien

5

Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

In diesem Kapitel wird ein Vergleich der Anwendbarkeit zwischen imperativer und deklarativer Modellierung für Softwareentwicklungsprozesse gezogen. Als Erstes wird dieser Vergleich in Kapitel 5.1 für das Softwareentwicklungsprozess Scrum durchgeführt. Hierfür wird zunächst in Kapitel 5.1 das der Modellierung zugrunde liegende Modell, das Softwareentwicklungsprozess Scrum, vorgestellt und in Kapitel 5.1.1 für die Modellierung analysiert. Danach erfolgt in Kapitel 5.1.2 die imperative Modellierung in der imperativen Prozessmodellierungssprache BPMN und anschließend die deklarative Modellierung in der deklarativen Prozessmodellierungssprache ConDec in Kapitel 5.1.3. Danach erfolgt in Kapitel 5.1.4 der Vergleich zwischen den beiden Modellen.

Der zweite Softwareentwicklungsprozess, welcher in diesem Kapitel in imperativer und deklarativer Prozessmodellierungssprache verglichen werden soll, ist der Open Unified

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

Process (Open UP). Auch hier erfolgt zunächst eine kurze Einführung in den Open UP in Kapitel 6.2, bevor dieser in Kapitel 5.2.1 analysiert wird, damit er in den Kapiteln 5.2.2 und 5.2.3 in imperativer, bzw. deklarativer Prozessmodellierungssprache modelliert werden kann. Hiernach erfolgt in Kapitel 5.2.4 der Vergleich zwischen den imperativen und deklarativen Prozessmodellen.

Zuletzt werden noch für das V-Modell XT die Prozessmodelle erstellt und verglichen. Eine Einführung in das V-Modell XT erfolgt in Kapitel 5.3. In Kapitel 5.3.1 wird für dieses als Vorbereitung für die Modellierung eine Analyse durchgeführt und in den Kapiteln 5.3.2 und 5.3.3 erfolgt die Modellierung in imperativer und deklarativer Prozessmodellierungssprache. Der Vergleich hierzu erfolgt in Kapitel 5.3.4.

Kapitel 5.4 widmet sich dem Vergleich zwischen allen Modellen insgesamt. Hier werden die Ergebnisse der Vergleiche aus den vorigen drei Kapiteln zusammengefasst und es werden allgemeine Schlüsse gezogen.

5.1. Scrum

Der Begriff Scrum stammt aus dem Artikel "The New New Product Development Game", welchen Hirotaka Takeuchi und Ikujiro Nonaka im Harvard Business Review 1986 veröffentlicht haben. Sie beschrieben einen ganzheitlichen Ansatz, bei dem kleine, funktionsübergreifende Teams zusammen an einem gemeinsamen Ziel arbeiten. Dies verglichen sie mit der Scrum-Formation beim Rugby [Pha12, TN86].

Bei Scrum handelt es sich um ein agiles Prozessmodell, welches seit Anfang 1990 für komplexe Entwicklungen verwendet wird. Agile Prozessmodelle werden den leichtgewichtigen Prozessmodellen zugeordnet [Han10, Lac12]. Einen ersten Überblick über das Scrum-Prozessmodell gibt Abbildung 5.1:

Der genaue Ablauf im Scrum Prozessmodell wird nachfolgend genau analysiert.

5.1. Scrum

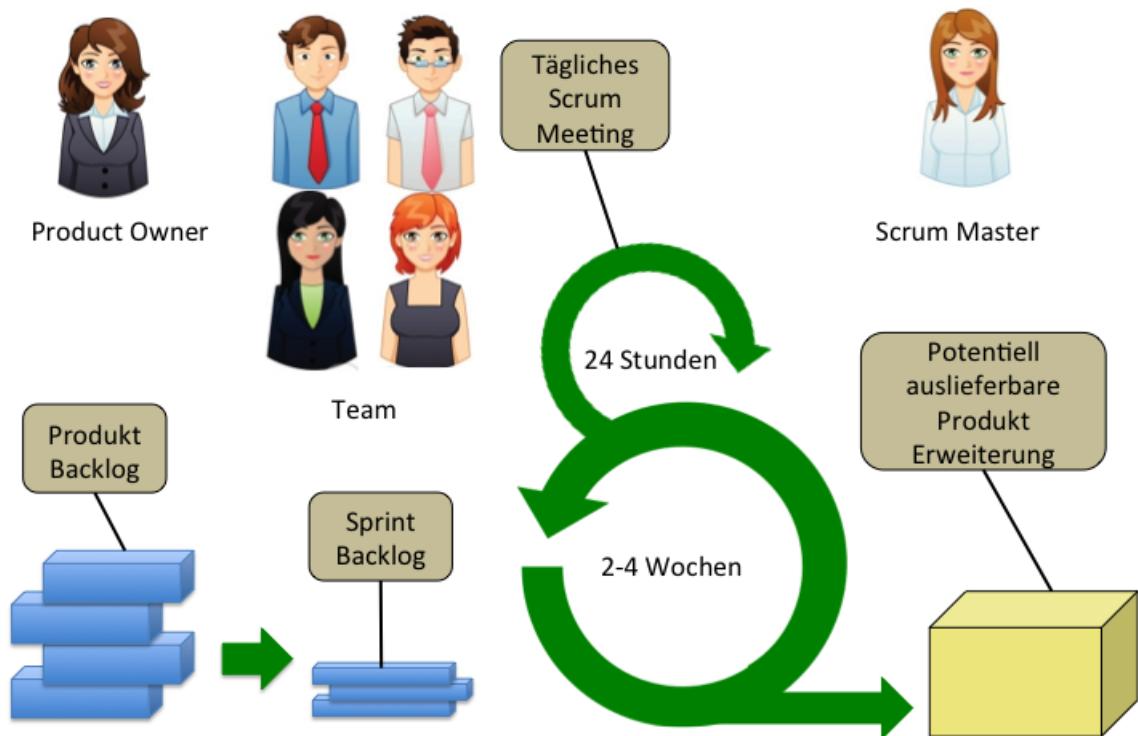


Abbildung 5.1.: Scrum Überblick nach [Mou14b]

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

5.1.1. Analyse Scrum

Im Scrum-Prozessmodell gibt es nur drei verschiedene Rollen: Den *Product Owner*, das *Team* und den *Scrum Master*. Sämtliche Verantwortlichkeiten innerhalb eines Projektes werden hierbei auf diese drei Rollen aufgeteilt [Sch04].

Der *Product Owner* ist verantwortlich, die Interessen aller am Projekt beteiligten Personen zu vertreten. Neben der Budgetierung des Projektes erstellt er ebenfalls Releasepläne und erstellt den Produkt-Backlog, welcher eine Liste mit funktionalen und nicht-funktionalen Anforderungen darstellt [Sch04, Pic10, Sch07]. Weiterhin priorisiert er die Aufgaben, welche von den Entwicklern im Sprint erledigt werden sollen, so dass die aktuell nützlichsten Elemente die höchste Priorität haben. Er erstellt eine Liste dieser Elemente, welche *Sprint-Backlog* genannt wird [Wol11a, Sch07, Pic10]. Der *Product Owner* ist ebenfalls zuständig für das Annehmen, bzw. Ablehnen der Arbeitsergebnisse [Ecl14].

Die *Teams* bestehen bei Scrum für gewöhnlich aus fünf bis neun Mitgliedern und verwalten sich selbst. Ihre Tätigkeiten müssen erfolgreich sein, liegen aber in ihrer eigenen Verantwortung [PQ11, Wol11b]. Alle Teammitglieder sind gemeinsam für den Erfolg eines jeden *Sprints* und des gesamten Projektes verantwortlich [Pic10].

Der *Scrum-Master* ist für den gesamten Scrum-Prozess verantwortlich. Dies schließt die Vermittlung von Scrum-Inhalten (z.B. Schulungen) und die Implementation von Scrum in die Unternehmenskultur ein [Pic10]. Er überwacht die Sprint-Tasks, um sicher zu gehen, dass der Sprint erfolgreich verläuft.

Bei Scrum wird die Entwicklung in mehrere kurze Zyklen, also Iterationen eingeteilt. Eine einzelne Iteration wird bei Scrum *Sprint* genannt [Wol11a]. Die Dauer eines Sprints beträgt zwei bis vier Wochen. Am Ende eines jeden Sprints muss das *Team* ein lauffähiges Produkt abliefern [Wol11b]. Vor jedem Sprint findet ein *Sprint Planning Meeting*

statt, welches sich aus zwei Teilen zusammensetzt [Pic10]. Im ersten Teil findet eine Planung des nächsten *Sprints* statt [Lac12]. Hierfür präsentiert der *Product Owner* dem *Team* eine Liste der Product-Backlog-Elemente mit der aktuell höchsten Priorität [Sch04, Sch07, Pic10]. Diese Liste wird *Sprint-Backlog* genannt [Wol11b]. Das *Team* hat die Möglichkeit, Fragen bezüglich Inhalt, Zweck, Bedeutung und Absichten der *Sprint-Backlog*-Elemente zu stellen. Anschließend werden die einzelnen Elemente aus dem *Sprint-Backlog* in sogenannte *Tasks* aufgeteilt, welche jeweils eine ideale Bearbeitungszeit von zwei bis vier Stunden haben, aber niemals länger als zwei Tage dauern sollten [Wol11b]. Das *Team* kann sich die Aufgaben eigenverantwortlich aufteilen und muss sich anschließend dem *Product Owner* verpflichten, die *Tasks* bis zum Abschluss des *Sprints* zu erledigen [Wol11b, Kei10, Pic10]. Das *Team* trifft sich während des *Sprints* täglich in einem 15-minütigen Meeting, dem *täglichen Scrum-Meeting*. Dabei redet das *Team* über seinen Fortschritt und eventuelle Probleme bei seiner Arbeit [Kei10]. Hier muss jedes Teammitglied die nachfolgenden drei Fragen beantworten [Wol11b]:

1. Was habe ich seit gestern erreicht?
2. Was werde ich heute erreichen?
3. Was blockiert mich?

5.1.2. Imperative Modellierung Scrum

Abbildung 5.2 zeigt die imperiative Modellierung von Scrum.

Im Prozess gibt es die drei verschiedenen Rollen *Product Owner*, *Team* und *Scrum Master*, was im Prozessmodell durch drei verschiedene Swimlanes dargestellt ist, welche die jeweilige Rolle repräsentieren.

Parallel zu allen anderen Aktivitäten des Teams und des *Product Owners* muss der *Scrum-Master* stets den Scrum-Prozess managen. Dies wird im Prozessmodell durch das parallele Gateway dargestellt.

Der *Product Owner* schätzt als erste Aktivität den Product Backlog ab. Anschließend priorisiert er den Product Backlog und erstellt parallel dazu die Releasepläne.

Wenn alle zwei bis vier Wochen ein neuer Sprint beginnt, was hier durch ein Zeitereignis

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

dargestellt ist, so wird zuerst das Sprint Planning Meeting durchgeführt. Dies ist hier als Unterprozess in Abbildung 5.3 dargestellt. Zunächst priorisiert der Product Owner die Anforderungen, welche während des Sprints erledigt werden müssen und erstellt danach den Sprint Backlog. Anschließend teilt sich dann das Team selbstständig die Sprint Backlog-Elemente in Tasks ein.

Im Anschluss findet ein Sprint-Rückblick statt (Abbildung 5.2) und der Product Owner führt ein Sprint Review Meeting durch.

Das Team führt während des Sprints täglich ein 15-minütiges Scrum Meeting durch und jedes Teammitglied arbeitet eine Task nach der anderen ab. Dies wird hier als Schleife dargestellt: Solange noch weitere Tasks vorhanden sind, führt das Exklusive Gateway immer wieder zurück zur Aktivität *Tasks abarbeiten*. Erst wenn keine weiteren Tasks mehr vorhanden sind, führt der Prozess weiter zum nächsten Entscheidungspunkt.

Sind noch weitere Aufgaben im Product Backlog vorhanden, die noch erledigt werden müssen, so beginnt ein weiterer Sprint, was hier durch eine Rückschleife dargestellt ist. Ist jedoch schon der komplette Product Backlog abgearbeitet, so endet der Prozess hier.

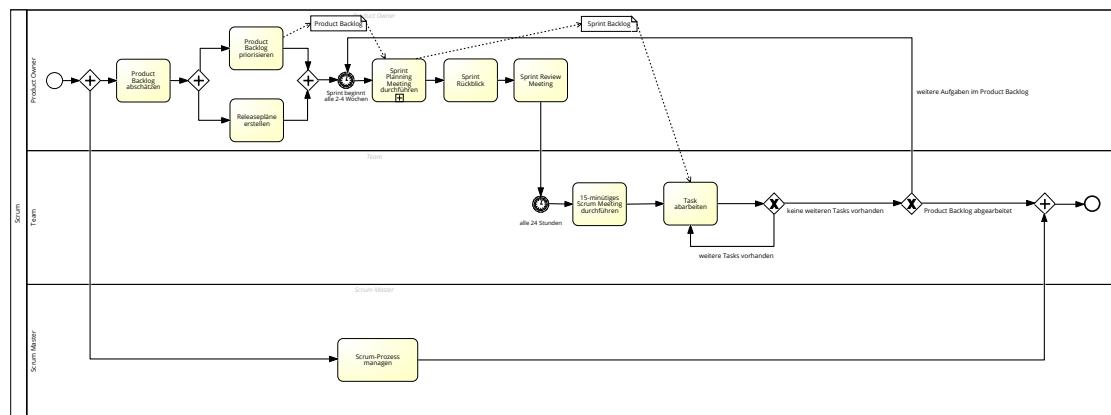


Abbildung 5.2.: Imperative Modellierung Scrum

5.1. Scrum

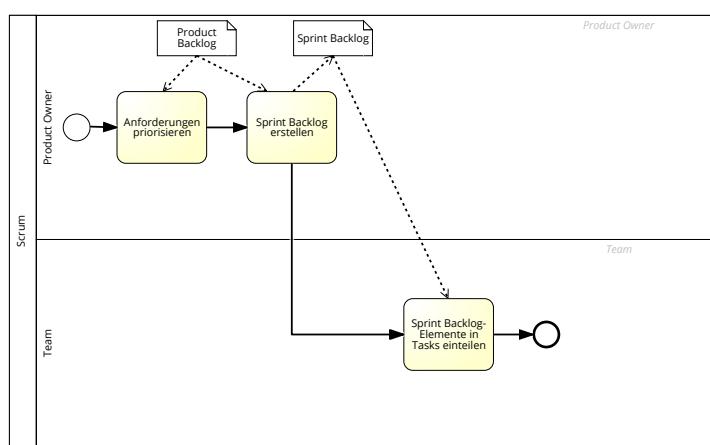


Abbildung 5.3.: Imperative Modellierung Scrum Unterprozess

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

5.1.3. Deklarative Modellierung Scrum

Abbildung 5.4 zeigt die deklarative Modellierung von Scrum.

Der Prozess beginnt mit der Aktivität *Product Backlog abschätzen*. Dies ist hier durch das init-Constraint gekennzeichnet. Weiterhin wird diese Aktivität im Prozess genau einmal ausgeführt was durch das Exactly (1)- Constraint dargestellt ist. Das Constraint *succession* gibt an, dass die Aktivität *Product Backlog abschätzen* vor den Aktivitäten *Product Backlog priorisieren* und *Releasepläne erstellen* ausgeführt werden müssen und dass die Aktivitäten *Product Backlog priorisieren* und *Releasepläne erstellen* auf jeden Fall nach *Product Backlog abschätzen* durchgeführt werden müssen. *Product Backlog priorisieren* und *Releasepläne erstellen* werden ebenfalls genau einmal ausgeführt, was durch das Exactly (1)- Constraint festgelegt wird.

Nach deren Ausführung muss die Aktivität *alle 2-4 Wochen Sprint Planning Meeting durchführen* erfolgen. Der zugehörige Unterprozess ist in Abbildung 5.5 zu finden. Hier sind die Aktivitäten *Anforderungen priorisieren*, *Sprint Backlog erstellen* und *Sprint-Backlog-Elemente in Tasks einteilen* durch das Constraint *precedence* miteinander verbunden, um die Einhaltung deren Reihenfolge nacheinander zu gewährleisten. Außerdem dürfen diese Aktivitäten pro Ausführung des Unterprozesses, also pro Prozessinstanz nur einmal ausgeführt werden.

Nach der Ausführung der Aktivitäten des Unterprozesses Sprint-Planning-Meeting durchführen, muss im Anschluß die Aktivität *Sprint Rückblick* durchgeführt werden. Dies wird durch das Constraint *chain response* sichergestellt. Eine erneute Ausführung von *alle 24 Stunden 15-minütiges Scrum-Meeting durchführen* ist erst nach Durchführung von *Sprint Rückblick* möglich (Constraint *alternate precedence*). Hierdurch wird eine Schleife modelliert, welche den immer wiederkehrenden Sprint simuliert.

Die Aktivitäten *alle 24 Stunden 15-minütiges Scrum-Meeting durchführen* und *Tasks abarbeiten* können während des Sprints so oft wie nötig durchgeführt werden. Die Aktivitäten *alle 24 Stunden 15-minütiges Scrum-Meeting durchführen* und *Tasks abarbeiten* müssen jedoch nebeneinander ausgeführt werden, was durch das Constraint *successi-*

5.1. Scrum

on beschrieben ist.

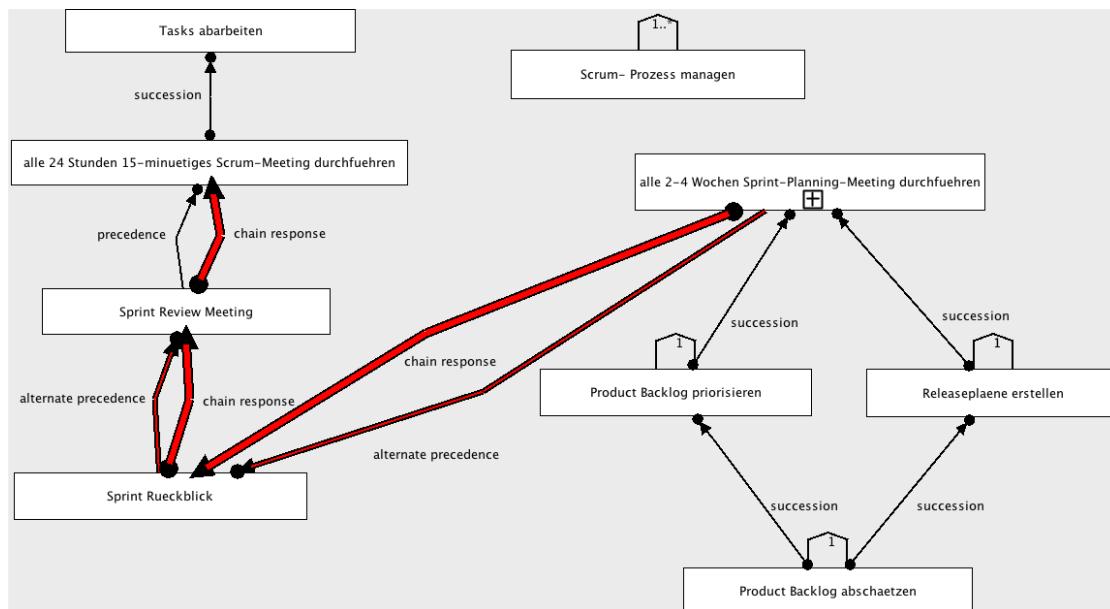


Abbildung 5.4.: Deklarative Modellierung Scrum



Abbildung 5.5.: Deklarative Modellierung Scrum-Unterprozess Sprint-Planning-Meeting durchfuehren

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

5.1.4. Vergleich

Der Vergleich zwischen den in der deklarativen Prozessmodellierungssprache ConDec und dem in der imperativen Prozessmodellierungssprache BPMN erstellten Scrum Prozessmodell wird im Folgenden anhand der in Kapitel 5 definierten Anforderungen durchgeführt.

Wie Abbildung 5.6 entnommen werden kann, unterscheidet sich die Anzahl der Aktivitäten zwischen den in BPMN und ConDec modellierten Prozessmodellen nicht voneinander. In jedem Prozessmodell gibt es 12 Aktivitäten. Damit handelt es sich hier um ein großes Modell (großes Modell hier definiert als Modell mit > 5 Aktivitäten).

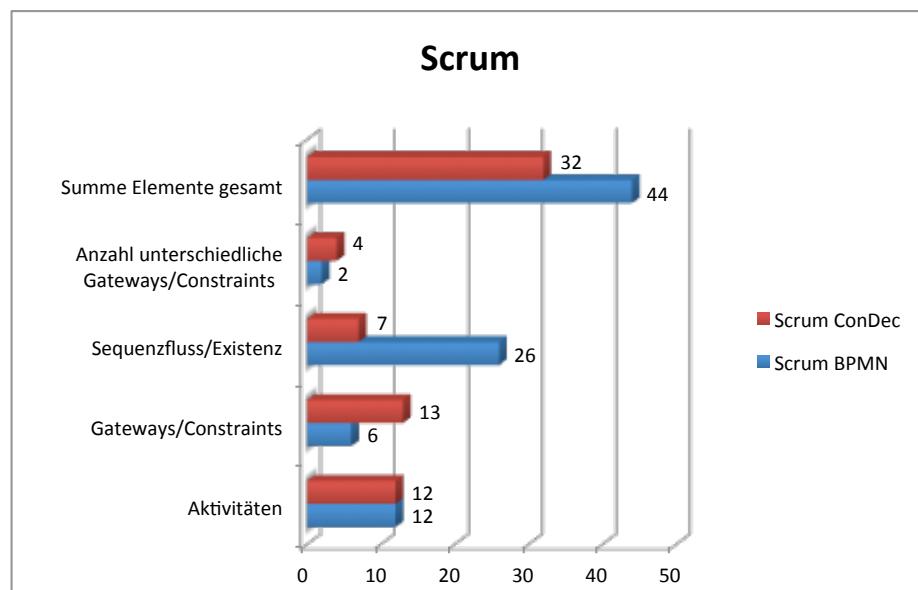


Abbildung 5.6.: Vergleich der Anzahl der Elemente Scrum

Zudem braucht es in BPMN sechs Gateways und 26 Sequenzflusselemente, um den Ablauf des Metamodells darzustellen. Bei Verwendung von ConDec werden 13 Constraints (vier unterschiedliche) und sieben Existenz Constraints zur Darstellung der Abfolge der Aktivitäten benötigt. In BPMN werden sechs Gateways (zwei unterschiedliche) verwendet.

5.1. Scrum

Die syntaktische *Richtigkeit* kann bei beiden Modellierungssprachen eingehalten werden, da bei beiden Prozessmodellierungssprachen die Notationsregeln zur korrekten Darstellung des Prozessablaufes eingehalten werden können. Dies hat das Testen der beiden Modelle in Signavio, bzw. Declare bestätigt.

Die semantische *Richtigkeit* lässt sich mit BPMN in Bezug auf Rollen und Artefakte besser einhalten, als mit ConDec. Da es bei ConDec keine Möglichkeit gibt, Rollen und Artefakte im Prozessmodell zu visualisieren fehlen diese Informationen. Die Rollen und Artefakte können zwar in Declare abgebildet werden, jedoch gibt es in ConDec hierfür keine Notationselemente, um Rollen und Artefakte im Prozessmodell selbst sichtbar zu machen. Aus diesem Grund müssen diese Informationen beim Modellieren weggelassen werden, was zur Folge hat, dass das im Metamodell beschriebene Verhalten nicht vollständig abgebildet werden kann und somit leidet auch der Nutzen des Modells.

In Anbetracht der syntaktischen Richtigkeit sind BPMN und ConDec gleich gut zur Modellierung geeignet. Bei Betrachtung der semantischen Richtigkeit ist BPMN die geeigneteren Modellierungssprache.

Nur BPMN bietet die Möglichkeit, Artefakte im Prozessmodell zu visualisieren und lässt somit die Integration anderer Sichten in das Modell zu. Somit kann der Modellierungsgrundsatz des *systematischen Aufbaus* nur von BPMN eingehalten werden. Da ConDec dies nicht zulässt, schmälert es die Eignung von ConDec zum Modellieren in Bezug auf Softwareprozessmodelle. Hierdurch können wichtige Informationen aus dem Metamodell nicht abgebildet werden.

Lediglich die mit BPMN erstellten Prozessmodelle können mit minimal relevanten Informationen erstellt werden. Der Grund dafür ist wiederum die fehlende Visualisierungsmöglichkeit von Rollen und Artefakten. Somit kann die *Relevanz* nur von BPMN eingehalten werden.

Bei Untersuchung der *Klarheit* lässt sich feststellen, dass sich die Anzahl der Sequenzflusselemente/Existenz Constraints zwischen BPMN (26) und ConDec (7) unterscheidet. Ebenfalls gibt es eine Differenz bei der Anzahl der Gateways (6)/Constraints (13) und

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

der Anzahl der unterschiedlichen Gateways (2)/Constraints (4) zwischen BPMN und ConDec. BPMN weist insgesamt eine höhere Anzahl an Elementen (44) auf als ConDec (32).

Wie in Kapitel 5 bereits erwähnt, kann sich eine größere Anzahl an Gateways/Constraints negativ auf die Verständlichkeit auswirken. Hier ist die Anzahl bei ConDec mehr als doppelt so hoch wie bei BPMN. In BPMN werden insgesamt zwei verschiedene Gateways (ein exklusives Gateway und ein paralleles Gateway) verwendet zur Darstellung von Verzweigungen. In ConDec werden vier verschiedene Constraints zur Darstellung verwendet. Da das Verständnis der Constraints in ConDec an sich relativ schwierig ist, ist das Verständnis eines Prozessmodells mit vielen verschiedenen Constraints relativ schwer. Da die Anzahl der unterschiedlichen Constraints hier bei ConDec doppelt so hoch ist wie die Anzahl der Gateways bei BPMN und das mit ConDec modellierte Prozessmodell dadurch sehr viel komplexer ist, als das mit BPMN modellierte Prozessmodell, ist hier BPMN trotz der höheren Anzahl an Elementen insgesamt das verständlichere Modell.

Somit ist hier BPMN in Bezug auf die *Klarheit* geeigneter zum Modellieren als ConDec.

Die *Wirtschaftlichkeit* unterschiedet sich nicht bei den beiden Modellierungssprachen. Das mit BPMN erstellte Modell weist 26 Sequenzflusselemente auf und das mit ConDec erstellte Modell sieben Existenz Constraints. Bei ConDec hingegen werden mehr Constraints (13, 4 verschiedene) benötigt als Gateways (6, 2 verschiedene) bei BPMN. Bei BPMN müssen mehr Elemente (44) zum Modellieren des gleichen Sachverhaltes verwendet werden wie bei ConDec (32). Die Tatsache, dass in ConDec vier unterschiedliche Constraints verwendet werden und in BPMN nur zwei unterschiedliche Gateways, macht den geistigen Aufwand für das Modellieren der beiden Prozesse in ConDec höher, als in BPMN.

Da BPMN zwar mehr Elemente insgesamt beim Modellieren benötigt und die Modellierung mit ConDec einen größeren geistigen Aufwand fordert wird hier BPMN als Prozessmodellierungssprache bevorzugt.

5.1. Scrum

Die *Vergleichbarkeit* in Betracht des Ausführungsverhaltens der beiden Modelle wurde durch Testausführungen in den Modellierungstools Signavio und Declare gewährleistet. Bei BPMN weist das erstellte Modell insgesamt mehr Elemente auf. Während bei ConDec insgesamt 32 Elemente benötigt werden, werden zur Darstellung des gleichen Sachverhaltes bei BPMN 44 Elemente verwendet.

Bei ConDec müssen Informationen wie Rollen und Artefakte weggelassen werden, während sie in BPMN dargestellt werden können.

Die *Vergleichbarkeit* kann zwar in Bezug auf das Ausführungsverhalten von beiden Sprachen eingehalten werden. BPMN weist jedoch insgesamt mehr Elemente auf und ConDec kann die *Vergleichbarkeit* in Bezug auf die Darstellbarkeit von Rollen und Artefakten nicht einhalten. Somit liegt hier keine der beiden Prozessmodellierungssprachen vorne.

Abbildung 5.7 zeigt die Ergebnisse des Vergleichs von BPMN und ConDec nochmals in der Zusammenfassung. Somit liegt BPMN bei den Grundsätzen *Richtigkeit, systematischer Aufbau, Klarheit* und *Vergleichbarkeit* vorne, bei den Grundsätzen *Wirtschaftlichkeit* und *Relevanz* liegen BPMN und ConDec gleich auf.

Modellierungsgrundsatz		Geeignete Modellierungssprache
Richtigkeit	syntaktisch	BPMN, ConDec
	semantisch	BPMN
Systematischer Aufbau		BPMN
Relevanz		BPMN
Klarheit		BPMN
Wirtschaftlichkeit		BPMN
Vergleichbarkeit		BPMN, ConDec

Abbildung 5.7.: Zusammenfassung Vergleich Scrum

5.2. Open Unified Process (Open UP)

Der Open Unified Process, kurz Open UP ist eine frei zugängliche Variante des Rational Unified Process, welcher ein sehr bekannter Entwicklungsprozess ist [HM10]. Er ist Teil des Eclipse Process Frameworks. Open UP ist ein iterativer, inkrementeller und minimaler Prozess, aber dennoch vollständig und erweiterbar [Gau06, EHS10]. Der Prozess ist minimal gehalten, weil er nur die wesentlichen Inhalte einbezieht. Trotzdem ist er vollständig, da er als Prozess benutzt werden kann, um ein Softwaresystem zu entwickeln. Er ist außerdem auch erweiterbar, da er als Grundlage herangezogen werden kann und mit weiteren Prozessfragmenten aufgestockt und nach Belieben zugeschnitten werden kann [WPR07]. Das Konzept des Open UP ist es, den Prozess zu vergrößern, sich aber auf das Minimum, welches für das Projekt benötigt wird, zu beschränken, anstatt zu versuchen große, überladene Prozesse zu verstehen und diese dann zu verkleinern [AL12].

Open UP ist auf kleine Teams ausgerichtet, bei welchen bei der Zusammenarbeit räumliche Nähe besteht. Die Teammitglieder haben hierbei die Freiheit, ihre eigenen Entscheidungen bezüglich ihren aktuellen Aufgaben und Prioritäten zu treffen, um die Anforderungen der Stakeholder zu erfüllen. Das Team trifft sich täglich, um über den aktuellen Status zu reden [COR09].

Es werden Rollen, Aufgaben, Artefakte und Ebenen in Open UP definiert. Dies soll ermöglichen, dass verschiedene Sichten, die sich in ihrem Detaillierungsgrad unterscheiden auf das Projekt möglich sind [Fre]. Einen ersten Überblick über Open UP gibt Abbildung 5.8.

Der Open UP wird im Folgenden analysiert.

5.2.1. Analyse Open UP

Auf der persönlichen Ebene teilen sich die Teammitglieder ihre Arbeit in *Mikro-Inkreme*nte ein. Diese stellen das Ergebnis von Stunden, bzw. wenigen Tagen Arbeit dar. Die Arbeit entwickelt sich somit ein Mikro-Inkrement weiter und der Fortschritt kann Tag für Tag

5.2. Open Unified Process (Open UP)

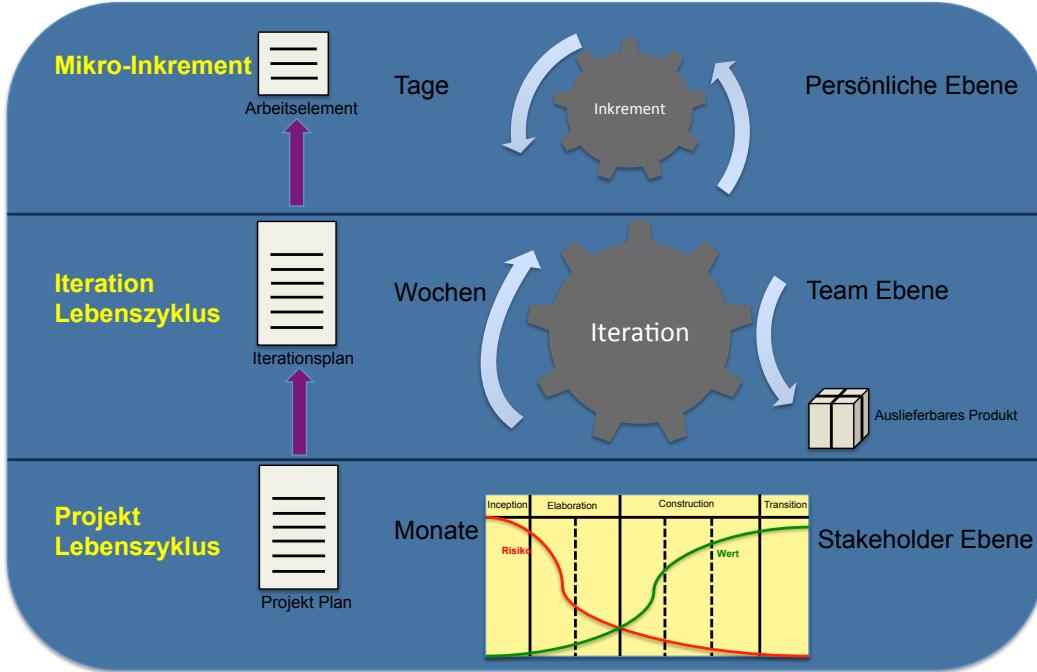


Abbildung 5.8.: Open Up Überblick nach [Ric07]

nachvollzogen werden. Die Teammitglieder teilen ihre Fortschritte täglich miteinander, was die Arbeitstransparenz und das Vertrauen erhöht und die Teamarbeit fördert [Ric07].

Auf der Team-Ebene wird das Projekt in Iterationen unterteilt, welche einen Zeitraum von mehreren Wochen umfassen, mit dem Ziel am Ende eines Iterationszyklus ein funktionierendes Softwareinkrement zu haben. Dieses Inkrement stellt eine Version des Softwaresystems dar welche zusätzliche oder verbesserte Funktionalitäten besitzt als die vorherige Version [EHS10]. In jeder Iteration wird ein Iterationsplan angefertigt, der vorgibt, was in dieser Iteration geliefert werden muss und auf welchen sich das Team verpflichten muss [Fre].

Auf Stakeholder-Ebene wird diesen durch den *Projektlebenszyklus* die Möglichkeit gegeben, die Projektfinanzierung, den Umfang, das Risiko und andere Aspekte des Prozesses zu kontrollieren. Der Open UP teilt den *Projektlebenszyklus* in die vier Phasen *Inception*,

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

Elaboration, Construction und Transition ein, über welche Abbildung 5.9 einen Überblick gibt [Ric07].

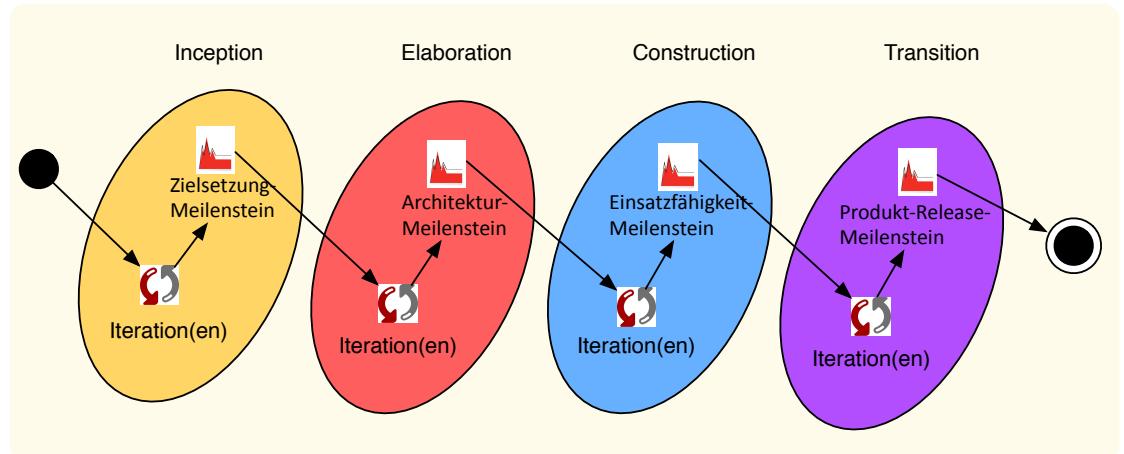


Abbildung 5.9.: Phasen Open UP nach [Ric07]

In jeder Phase finden eine oder mehrere Iterationen statt und werden mit einem Meilenstein abgeschlossen [EHS10]. In der Phase *Inception* ist dies der Zielsetzung- Meilenstein, in der Phase *Elaboration* der Architektur- Meilenstein, in der Phase *Construction* der Einsatzfähigkeit- Meilenstein und in der Phase *Transition* der Produkt- Release- Meilenstein. Tabelle 5.1 zeigt die Abläufe in den Iterationen in den einzelnen Phasen und die zugehörigen Zielstellungen.

Vorlagenmodell Iterationen

Zielsetzung der Phase

5.2. Open Unified Process (Open UP)

Inception Phase Iteration	<ul style="list-style-type: none">• Iteration starten• Iteration planen und verwalten• Anforderungen festlegen und verfeinern <ul style="list-style-type: none">• Verstehen, was zu bauen ist• Die wichtigsten Systemfunktionen verstehen• Mindestens eine mögliche Lösung bestimmen• Kosten, Zeitplan und Risiken verstehen, welche mit dem Projekt verbunden sind
Elaboration Phase Iteration	<ul style="list-style-type: none">• Iteration planen und verwalten• Anforderungen erheben und verfeinern• Architektur definieren• Lösung entwickeln• Testlösung• Laufende Aufgaben <ul style="list-style-type: none">• Ein detaillierteres Verständnis der Anforderungen einholen• Architektur designen, implementieren und validieren• Wesentliche Risiken mindern und genauen Zeitplan und Kostenschätzungen erstellen

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

<p>Construction Phase Iteration</p> <ul style="list-style-type: none"> • Iteration planen und verwalten • Anforderungen erheben und verfeinern • Lösung entwickeln • Testlösung • Laufende Aufgaben 	<ul style="list-style-type: none"> • Komplettes Produkt iterativ entwickeln, welches am Ende bereit ist an seine Nutzer ausgeliefert zu werden • Entwicklungskosten minimieren und einen gewissen Grad an Parallelität erzielen
<p>Transition Phase Iteration</p> <ul style="list-style-type: none"> • Iteration planen und verwalten • Lösung entwickeln • Testlösung • Laufende Aufgaben 	<ul style="list-style-type: none"> • Beta-Test, um zu überprüfen, dass die Erwartungen der Benutzer erfüllt sind • Zustimmung der Stakeholder einholen, dass Bereitstellung abgeschlossen ist

Tabelle 5.1.: Iterationen und Zielstellungen der Phasen in Open UP [Ric07]

Abbildung 5.10 gibt einen Überblick über die verschiedenen Rollen in Open UP. Die Rolle *Analyst* stellt den Kunden und Endnutzer dar. Die Aufgaben des *Analysten* bestehen aus dem Sammeln von Informationen von den Stakeholdern, um das Problem, welches es zu lösen gilt, zu verstehen. Weiterhin erstellt er Anforderungen und setzt Prioritäten für diese [COR09].

Der *Tester* ist für sämtliche Testaktivitäten verantwortlich. Diese umfassen die Ermittlung, Festlegung, Umsetzung und Durchführung der erforderlichen Tests sowie die Protokollierung und Analyse der Ergebnisse [COR09]. Der *Entwickler* entwickelt einen Teil des Systems und muss hierbei sicherstellen, dass dieser in die Gesamtarchitektur

5.2. Open Unified Process (Open UP)

passt. Er muss eventuell Prototypen des User-Interface anfertigen und anschließend die Komponenten implementieren, testen und integrieren [COR09].

Der *Architekt* ist für die Definition der Software-Architektur verantwortlich, d.h. er trifft alle wichtigen technischen Entscheidungen, die die gesamte Entwicklung und Umsetzung des Systems betreffen [COR09].

Der *Projekt Manager* führt die Planung des Projektes durch, koordiniert die Zusammenarbeit zwischen allen Beteiligten und achtet darauf, dass das Projektteam die Erfüllung der Projektziele stets im Auge behält [COR09].

Die Rolle des *Stakeholders* schließt alle Interessengruppen ein, deren Ansprüche durch das Projekt erfüllt werden müssen.

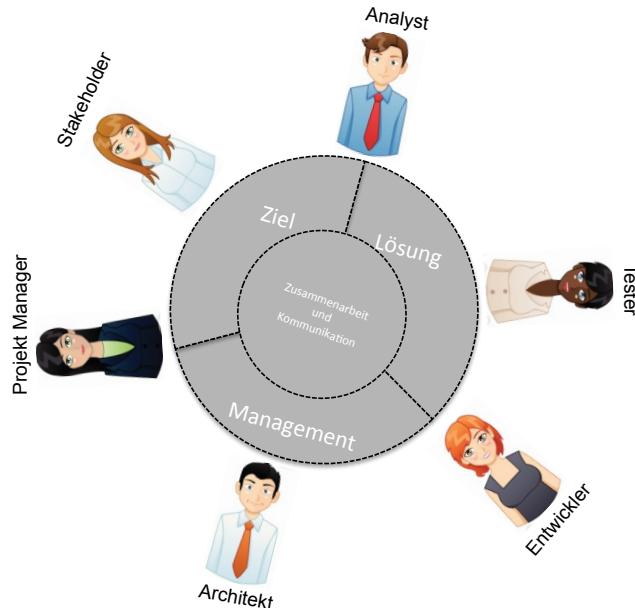


Abbildung 5.10.: Rollen in Open UP nach [Spa14]

Eine Task bezeichnet in Open UP die Arbeitseinheit einer Rolle, welche von dieser durchgeführt werden soll. Insgesamt gibt es 18 Tasks, welche von den verschiedenen Rollen entweder als Primär-Darsteller (der Verantwortliche für die Durchführung der Aufgabe) oder als zusätzlicher Darsteller (Unterstützung und Bereitstellung von Informationen, die

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

in der Task- Ausführung verwendet werden), durchgeführt werden. Hierdurch wird der kollaborative Charakter von Open UP gefestigt [Ric07].

Ein Artefakt ist etwas, das hergestellt, modifiziert oder durch eine Task verwendet wird. Rollen sind für die Erstellung und Aktualisierung von Artefakten verantwortlich. Artefakte stellen eine Versionskontrolle während des gesamten Projektlebenszyklus dar. Die 17 Artefakte in Open UP gelten als die wesentlichen Artefakte, welche ein Projekt verwenden sollte, um produkt- und projektbezogene Informationen zu erfassen. Die Informationen müssen hierbei nicht mit formalen Artefakten festgehalten werden. Dies kann auch informell, z.B. durch White-Boards oder Meeting-Notizen geschehen. Es können die Open UP Artefakte oder eigene Artefakte verwendet werden [Ric07].

5.2.2. Imperative Modellierung Open UP

Phasen Open UP

In Abbildung 5.11 sind die vier Phasen des Open UP modelliert. Da jede Phase in Iterationen mehrmals durchlaufen werden kann, gibt es nach jeder Phase ein XOR-Gateway, welches im Falle einer weiteren notwendigen Iteration zum Anfang der Phase zurückführt. Diese kann sodann erneut durchlaufen werden.

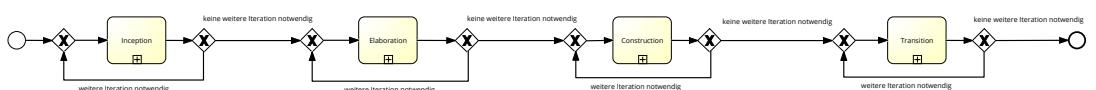


Abbildung 5.11.: Phasen Open UP- imperativ

Abbildung 5.12 zeigt die imperative Modellierung der Phase Inception. Die Aktivität *Projekt planen und managen* kann parallel zu allen anderen Aktivitäten des Modells ausgeführt werden.

Nach Ausführung der Aktivität *Iteration planen* werden die Aktivitäten *Anforderungen identifizieren und aufbereiten* und *auf technisches Vorgehen einigen* parallel zueinander ausgeführt.

5.2. Open Unified Process (Open UP)

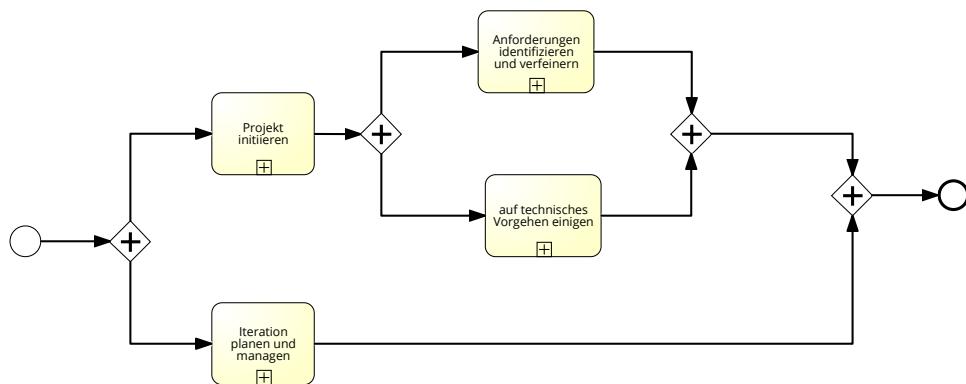


Abbildung 5.12.: Phasen Open UP Unterprozess Inception- imperativ

In Abbildung 5.13 ist die imperative Modellierung der Phase Elaboration abgebildet. Die sechs Aktivitäten *Anforderungen identifizieren und verfeinern*, *Architektur entwickeln*, *Lösungssinkrement entwickeln*, *Lösung testen*, *Iteration planen und managen* sowie *weitere Aufgaben erledigen* werden parallel zueinander ausgeführt.

Die imperative Modellierung der Phase Construction kann Abbildung 5.14 entnommen werden. Hier werden die sechs Aktivitäten *Anforderungen identifizieren und verfeinern*, *Lösungssinkrement entwickeln*, *Lösung testen*, *Iteration planen und managen*, *weitere Aufgaben erledigen* und *Produktdokumentation und Training erstellen* nebeneinander parallel ausgeführt.

Abbildung 5.15 kann die imperative Modellierung der Phase Transition entnommen werden.

Die Phasen *Anforderungen identifizieren und verfeinern*, *Produkt Training durchführen*, *Lösungssinkrement entwickeln*, *Lösung testen*, *Iteration planen und managen*, *weitere Aufgaben erledigen*, *Produktdokumentation und Training abschließen* sowie *Release für die Produktion freigeben* werden parallel zueinander ausgeführt.

Im weiteren Verlauf wird aus jeder der vier Phasen Inception, Elaboration, Construction und Transition des Open UP jeweils ein Unterprozess modelliert, da die Abbildung aller Unterprozesse aus jeder Phase den Rahmen der Arbeit sprengen würde.

Somit wird für die Phase Inception der Unterprozess *Iteration planen und managen*, für

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

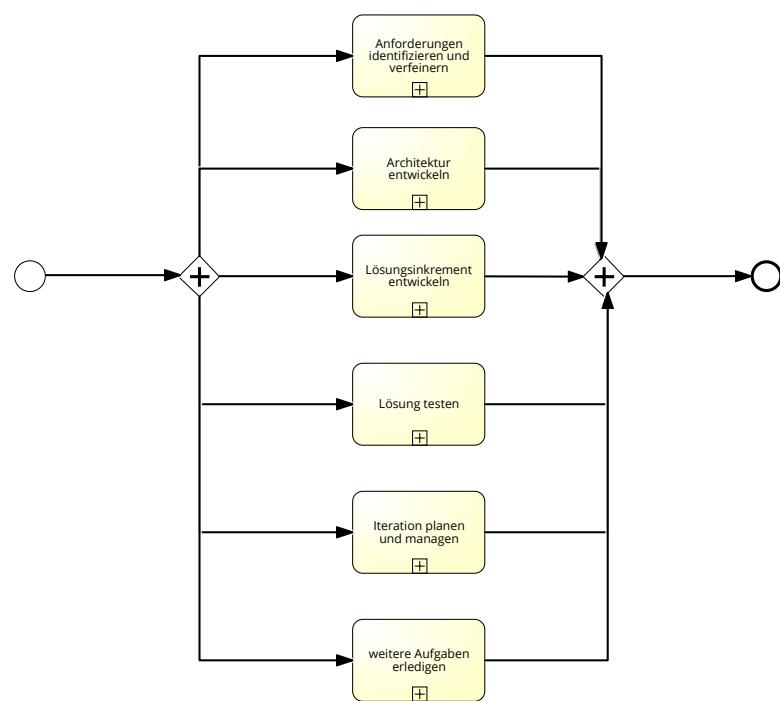


Abbildung 5.13.: Phasen Open UP Unterprozess Elaboration- imperativ

5.2. Open Unified Process (Open UP)

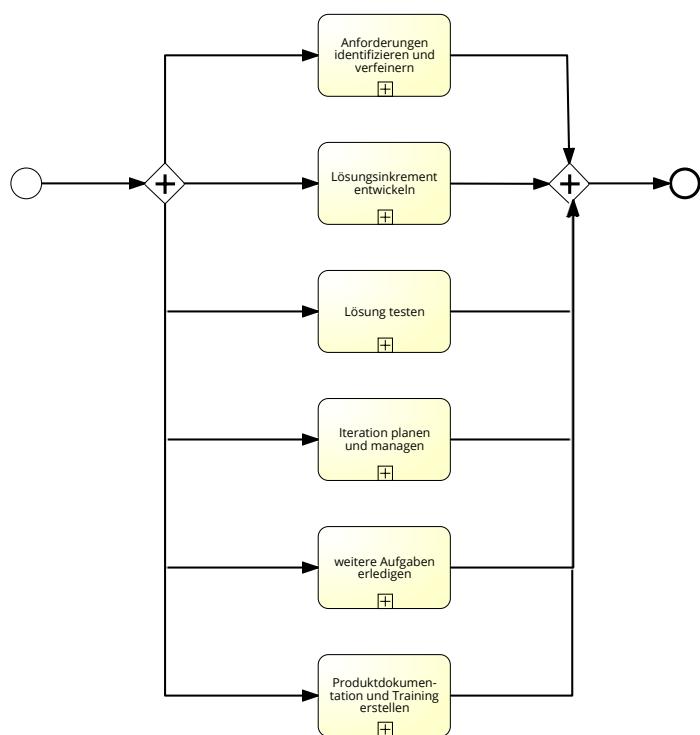


Abbildung 5.14.: Phasen Open UP Unterprozess Construction- imperativ

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

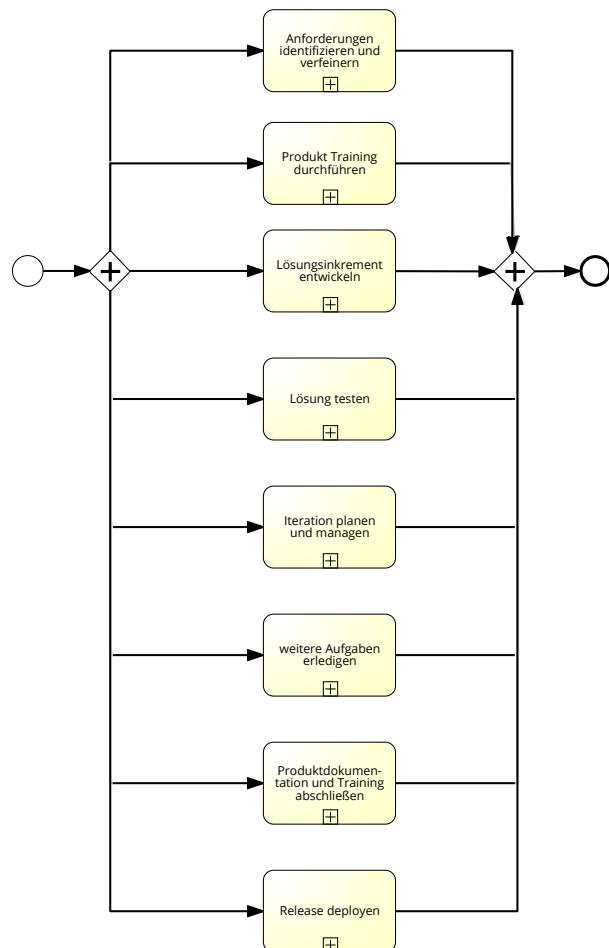


Abbildung 5.15.: Phasen Open UP Unterprozess Transition- imperativ

5.2. Open Unified Process (Open UP)

die Phase Elaboration der Unterprozess *Anforderungen identifizieren und verfeinern*, für die Phase Construction der Unterprozess *Release deployen* und für die Phase Transition der Unterprozess *Produktdokumentation und Training erstellen* modelliert. Außerdem wird der in den drei Phasen Elaboration, Construction und Transition wiederkehrende Unterprozess *Lösungssinkrement entwickeln* modelliert.

Lösungssinkrement entwickeln

Im Unterprozess *Lösungssinkrement entwickeln* geht es um das Design, die Implementierung, das Testen und die Integration der Lösung für eine Anforderung in einem bestimmten Kontext. Sie tritt genauso viele Male auf, wie es Arbeitsaufgaben gibt, die in einer Iteration entwickelt werden müssen. Handelt es sich um eine typische Veränderung wird zunächst eine Lösung designt und anschließend ein Entwicklertest implementiert. Bei einer trivialen Änderung an der bestehenden Implementierung kann diese auch direkt in der bestehenden Architektur vorgenommen werden.

Sobald die Fragen der technischen Umsetzung geklärt sind, werden Entwicklertests implementiert, um die Implementierung zu verifizieren. Anschließend werden diese Entwicklertests ausgeführt.

Falls bei der Ausführung der Tests Fehler ersichtlich werden, muss eine Lösung für diesen Fehler implementiert werden und die Entwicklertests müssen erneut ausgeführt werden. Dies wird solange wiederholt, bis alle Tests bestanden sind.

Auch wenn alle Tests bestanden werden, sollte der Entwurf an dieser Stelle nochmals überdacht werden. Falls hier beschlossen wird, dass der Code überarbeitet werden muss, muss im Prozess zurückgegangen werden und erneut eine Lösung designt werden, da eine Änderung des Codes die Implementation und die Entwicklertests beeinflussen könnte.

Da es am Besten ist die Implementierungsteile so klein wie möglich zu halten, sollte zunächst eine kleine Design-Lösung für einen Teil der Arbeitsaufgabe entwickelt werden. Anschließend sollte dies für weitere kleine Teile solange wiederholt werden, bis die gesamte Arbeitsaufgabe implementiert ist.

In Abbildung 5.27 ist die imperative Modellierung von *Lösungssinkrement entwickeln*

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

abgebildet.

Die XOR-Verknüpfung am Anfang führt im Falle einer trivialen Änderung zur sofortigen Ausführung der Aktivität *Entwicklertest implementieren*. Falls es sich jedoch um eine typische Änderung handelt, muss zuvor die Aktivität *Lösung designen* ausgeführt werden. Im Anschluss an *Entwicklertest implementieren* muss die Aktivität *Entwicklertest ausführen* durchgeführt werden.

Hiernach wird im Falle eines fehlgeschlagenen Tests zunächst eine *Lösung implementiert* und anschließend erneut der *Entwicklertest ausgeführt*.

Wenn der Test bestanden ist muss am XOR-Gateway entschieden werden, ob der Code gut designt ist. Falls nein, muss erneut eine Lösung designt werden. Falls doch, kann der Code integriert werden. Ist die Arbeit vollständig erledigt, so ist der Prozess beendet.

Wenn jedoch noch weitere Arbeit vorhanden ist, beginnt er von vorne.

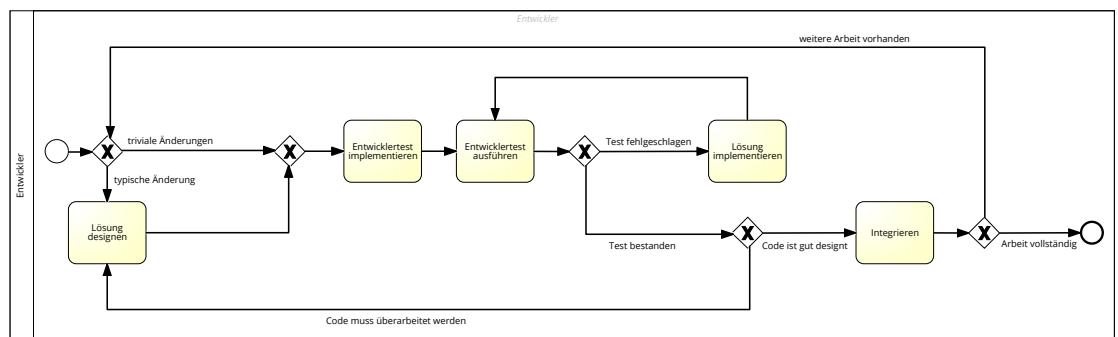


Abbildung 5.16.: Lösungsinkelement entwickeln imperativ

Iteration planen und managen- Inception

Die Aktivität *Iteration planen und managen* wird während des gesamten Projektlebenszyklus ausgeführt. Ihr Ziel ist es, Risiken und Probleme früh genug zu identifizieren, damit diese entschärft werden können, um die Ziele für die Iteration festzulegen und das Team dabei zu unterstützen, diese zu erreichen.

Die Iteration wird durch den Projektmanager und das Team gestartet. Hier findet die Priorisierung der Arbeit für eine gegebene Iteration statt. Der Projektmanager, die Stakeholder und die Teammitglieder einigen sich darauf, was während der Iteration zu

5.2. Open Unified Process (Open UP)

entwickeln ist.

Die Teammitglieder melden sich für die Arbeitsaufgaben, die während der Iteration entwickelt werden müssen. Anschließend teilt sich jedes Teammitglied seine Arbeitsaufgaben selbstständig in Arbeitseinheiten ein und schätzt den Aufwand hierfür ab.

Während der Iteration trifft sich das Team regelmäßig, um den aktuellen Stand der Arbeit und eventuelle Probleme zu besprechen.

Abbildung 5.17 zeigt die imperativen Modellierung von *Iteration planen und managen*.

Vom Projektmanager sind hierbei nacheinander die Aktivitäten *Iteration planen*, *Umgebung vorbereiten*, *Iteration managen* und *Ergebnisse festlegen* durchzuführen und das Team muss nacheinander die Aktivitäten *Arbeitsaufgaben aussuchen*, *Arbeitsaufgaben in Entwicklungsaufgaben einteilen* sowie *Aufwand abschätzen* ausführen. Hierbei gehen jeweils die Artefakte *Arbeitseinheiten-Liste*, *Iterationsplan* und *Risiko-Liste* in verschiedenen Aktivitäten als Input ein und kommen eventuell verändert als Output wieder heraus.

Die Aktivität *Umgebung vorbereiten* ist als Unterprozess in Abbildung 5.18 dargestellt. Hier müssen vom Projektmanager die Aktivitäten *Prozess Maßschneidern* und *Prozess deployen* sequentiell erledigt werden, während der Tool Spezialist die Aufgaben *Tools aufsetzen* und *Tool-Konfiguration und Implementation verifizieren* zu erledigen hat.

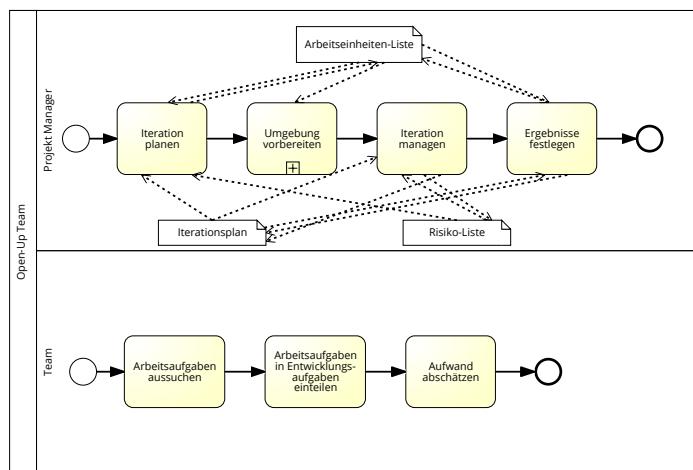


Abbildung 5.17.: Iteration planen und managen imperativ -Inception

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

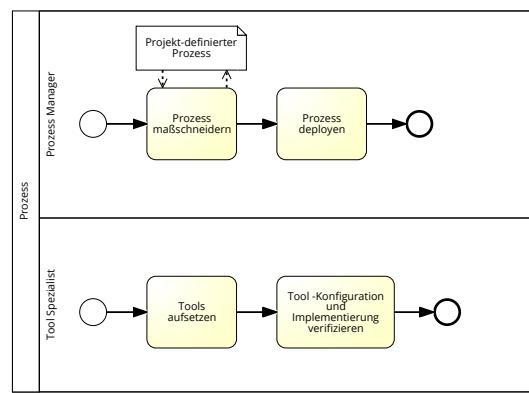


Abbildung 5.18.: Iteration planen und managen imperativ -Inception Unterprozess Umgebung vorbereiten

Anforderungen identifizieren und verfeinern

Der Unterprozess *Anforderungen identifizieren und verfeinern* beschreibt die Aufgaben, welche durchzuführen sind, um die Anforderungen eines Systems zu sammeln, zu analysieren und zu validieren bevor die Implementierung und die Validierung stattfinden. Sie wird in Zusammenarbeit mit Stakeholdern und dem gesamten Entwicklungsteam ausgeführt, um sicher zu gehen, dass klare, konsistente, korrekte und nachprüfbare Anforderungen vorhanden sind.

In der Phase Elaboration liegt der Fokus hierbei auf der Definition der Lösung. Hierfür müssen diejenigen Anforderungen gefunden werden, welche für die Stakeholder am wichtigsten sind die besonders herausfordernd oder sogar riskant sind oder eine große Bedeutung für die Architektur haben.

Dafür ist es notwendig, zunächst die funktionalen und nicht-funktionalen Anforderungen an das System zu erheben. Genau diese Anforderungen stellen dann die Basis für die Kommunikation und die Übereinstimmung zwischen den Stakeholdern und dem Entwicklungsteam dar, in Bezug auf was das System können muss, um die Wünsche der Stakeholder zu erfüllen.

Weiterhin müssen die Use-Case-Szenarien und die systemweiten Anforderungen ausführlich genug beschrieben werden, um sicher zu gehen, dass die Anforderungen richtig verstanden wurden und dass diese mit den Erwartungen der Stakeholder übereinstimmen.

Zudem müssen Testfälle und Testdaten für die Anforderungen entwickelt werden, um ein gemeinsames Verständnis für die spezifischen Bedingungen, die die Lösung erfüllen muss, zu erreichen. In Abbildung 5.19 ist die imperative Modellierung von *Anforderungen identifizieren und verfeinern* abgebildet.

Zunächst muss der Analyst die *Anforderungen identifizieren und abgrenzen*, bevor er anschließend die *Use-Case-Szenarien detaillieren* kann. Daraufhin muss er die *Systemweiten Anforderungen detaillieren*, damit der Tester anschließend die *Testfälle erstellen* kann.

Hier gehen bei den verschiedenen Aktivitäten die Artefakte *Arbeitseinheitenliste*, *Use*

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

Case, Glossar, Systemweite Anforderungen, Use case Modell, Technische Spezifikation und Testfall als Input hinein, bzw. als Output heraus.

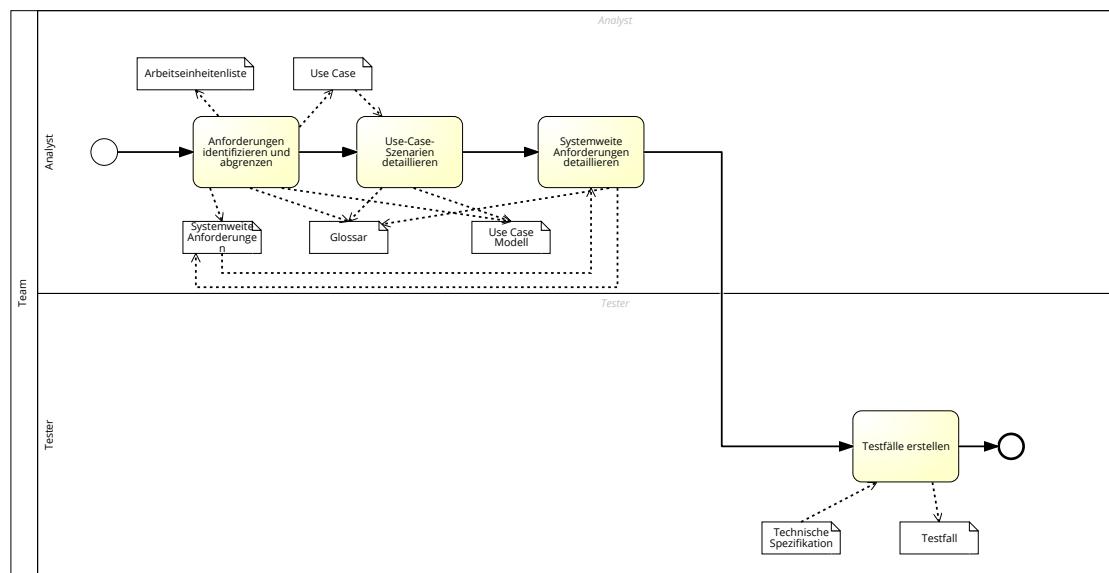


Abbildung 5.19.: Anforderungen identifizieren und verfeinern-Elaboration

Produktdokumentation und Training erstellen-Construction

Das Ziel des Unterprozesses *Produktdokumentation und Training erstellen* ist es, die Produktdokumentation und Trainingsmaterial vorzubereiten. Da die Produktdokumentation oftmals erst nach Abschluss der Entwicklungstätigkeiten erstellt wird, muss sichergestellt werden, dass die Funktionen die während einer Release entwickelt werden klar dokumentiert werden, solange die Funktionalität noch frisch in den Köpfen der Teammitglieder vorhanden ist.

Hierfür ist es notwendig, dass genug Informationen über die Funktionen, die in einer bestimmten Release entwickelt wurden, dokumentiert werden um dem Kunden während der gesamten Lebenszeit des Produkts nützlich zu sein.

Weiterhin müssen den Endnutzern nützliche Informationen bereit gestellt werden in Form von Benutzerhandbüchern, Tutorials, häufig gestellte Fragen (FAQs), Online-Hilfedateien, Installationsanweisungen und Betriebsabläufe.

5.2. Open Unified Process (Open UP)

Zudem muss sichergestellt werden, dass diejenigen, die mit der Unterstützung des Systems beauftragt sind, genug Informationen über das Produkt haben, um ihre Arbeit effektiv durchzuführen, nachdem das Produkt produktiv gegangen ist.

Außerdem muss die Einführung des Produkts zu ermöglicht werden und dessen ordnungsgemäße Verwendung gewährleistet werden.

Die imperative Modellierung von *Produktdokumentation und Training erstellen* kann Abbildung 5.20 entnommen werden.

Hier sind vom technischen Schreiber nacheinander die Aktivitäten *Produktdokumentation erstellen*, *Benutzerdokumentation erstellen*, *Unterstützungsdokumentation erstellen* und *Trainingsmaterial erstellen* auszuführen. Aus den jeweiligen Aktivitäten entstehen sodann die Artefakte *Produktdokumentation*, *Benutzerdokumentation*, *Unterstützungsdokumentation* und *Trainingsmaterial*.

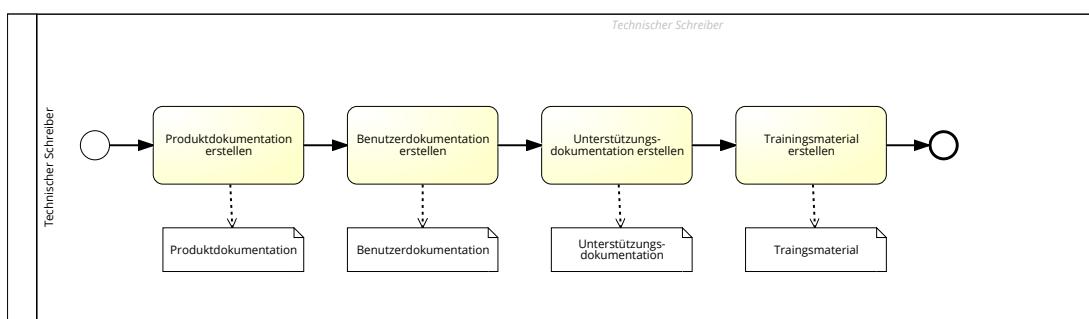


Abbildung 5.20.: Produktdokumentation und Training erstellen - Construction

Release deployen-Transition

Das Ergebnis dieses Unterprozesses ist die Release eines Sets von integrierten Komponenten in der Integrationsumgebung.

Hierfür ist es notwendig ein komplettes, bereitstellungsfähiges Paket zu erstellen, welches vom Deployment Engineer in die Bereitstellungsumgebung released werden kann.

Außerdem muss sichergestellt werden, dass der Roll-Out aus klaren, geprüften und wiederholbaren Anweisungen besteht und das Risiko eines Bereitstellungsfehlers muss

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

minimiert werden.

Zudem muss sichergestellt werden, dass eine Release zu keinen ungewollten Unterbrechungen im Ablauf in der Produktionsumgebung führt.

Falls eine Release Probleme verursacht oder sie von den Stakeholdern als untauglich empfunden wird, muss diese Release von der Produktionsumgebung so schnell wie möglich entfernt werden.

Zusätzlich muss dafür gesorgt werden, dass Informationen über eine anstehende Release weitest möglich verteilt werden.

Abbildung 5.21 zeigt die imperative Modellierung von *Release deployen*.

Somit muss der Entwickler zunächst die *Release zusammenstellen*, bevor der Deployment Engineer nacheinander die Aktivitäten *Deploymentplan ausführen* und *erfolgreiches Deployment sicherstellen* ausführt. Falls das Deployment erfolgreich ist, wird gleich anschließend die Aktivität *Releasemittelungen übermitteln* ausgeführt. Falls das Deployment nicht erfolgreich ist, muss zunächst die Aktivität *Backoutplan ausführen* erledigt werden und erst danach die Aktivität *Releasemittelungen übermitteln* ausgeführt werden.

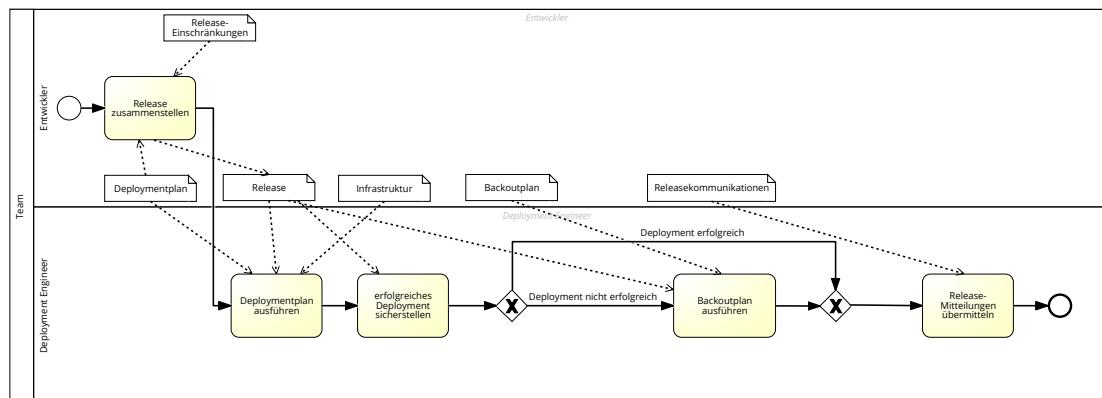


Abbildung 5.21.: Release deployen-Transition

5.2.3. Deklarative Modellierung Open UP

Phasen des Open UP

In Abbildung 5.22 sind die vier Phasen des Open UP deklarativ modelliert. Jede Phase kann in Iterationen mehrmals durchlaufen werden. Aus diesem Grund sind die vier Phasen durch das Constraint *succesion* miteinander verbunden. Hierdurch wird gewährleistet, dass jede Phase so oft ausgeführt werden kann, wie nötig, aber das ebenfalls die Reihenfolge eingehalten wird. Z.B. kann die Phase Elaboration so erst durchlaufen werden, nachdem die Phase Inception durchlaufen wurde.

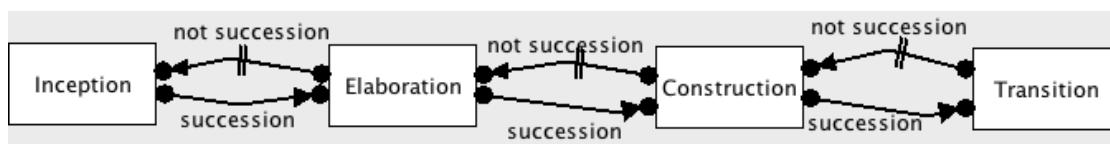


Abbildung 5.22.: Phasen Open UP- deklarativ

Abbildung 5.23 zeigt die deklarative Modellierung der Phase Inception. Die Aktivität *Projekt planen und managen* kann parallel zu allen anderen Aktivitäten des Modells ausgeführt werden.

Nach Ausführung der Aktivität *Iteration planen* werden die Aktivitäten *Anforderungen identifizieren und aufbereiten* und *auf technisches Vorgehen einigen* parallel zueinander ausgeführt. Aus diesem Grund sind die Aktivitäten *Anforderungen identifizieren und aufbereiten* und *auf technisches Vorgehen einigen* mit der Aktivität *Projekt planen und managen* durch das Constraint *succession* verbunden, da sie erst nach deren Ausführung ausgeführt werden dürfen und auch ausgeführt werden müssen.

In Abbildung 5.24 ist die deklarative Modellierung der Phase Elaboration abgebildet. Die sechs Aktivitäten *Anforderungen identifizieren und verfeinern*, *Architektur entwickeln*, *Lösungskrement entwickeln*, *Lösung testen*, *Iteration planen und managen* sowie *weitere Aufgaben erledigen* werden parallel zueinander ausgeführt. Aus diesem Grund befindet sich lediglich das Constraint *Ecactly 1* an jeder Aktivität, da sie inner-

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

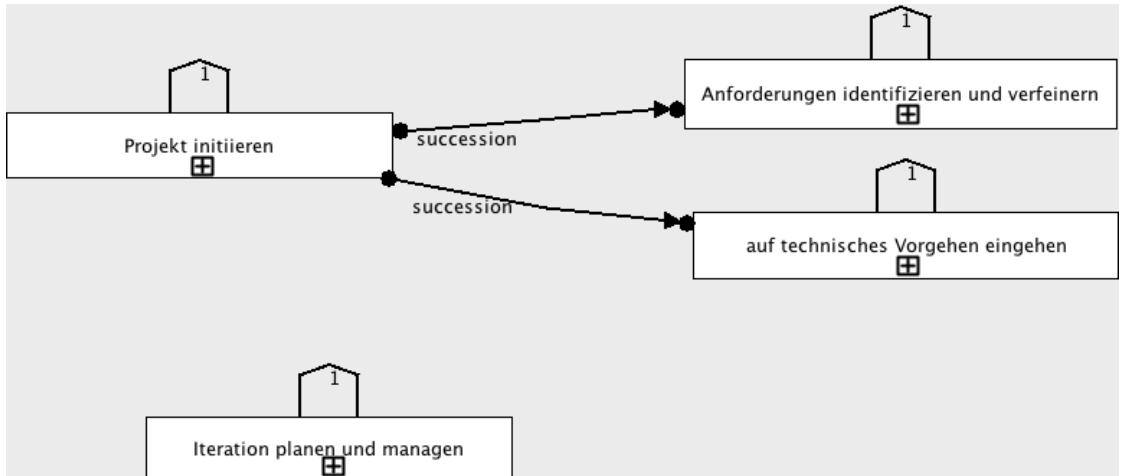


Abbildung 5.23.: Phasen Open UP Unterprozess Inception- deklarativ

halb einer Prozessinstanz nur einmal ausgeführt werden dürfen, dies aber in beliebiger Reihenfolge. Im Falle einer weiteren Iteration der Phase Elaboration wird eine neue Prozessinstanz aufgerufen.

Die deklarative Modellierung der Phase Construction kann Abbildung 5.25 entnommen werden. Hier werden die sechs Aktivitäten *Anforderungen identifizieren und verfeinern*, *Lösungssinkrement entwickeln*, *Lösung testen*, *Iteration planen und managen*, *weitere Aufgaben erledigen* und *Produktdokumentation und Training erstellen* nebeneinander parallel ausgeführt.

Abbildung 5.26 kann die deklarative Modellierung der Phase Transition entnommen werden.

Die Aktivitäten *Anforderungen identifizieren und verfeinern*, *Produkt Training durchführen*, *Lösungssinkrement entwickeln*, *Lösung testen*, *Iteration planen und managen*, *weitere Aufgaben erledigen*, *Produktdokumentation und Training abschließen* sowie *Release für die Produktion freigeben* werden parallel zueinander ausgeführt.

Im weiteren Verlauf wird aus jeder der vier Phasen Inception, Elaboration, Construction und Transition des Open UP jeweils ein Unterprozess modelliert, da die Abbildung aller Unterprozesse aus jeder Phase den Rahmen der Arbeit sprengen würde.

5.2. Open Unified Process (Open UP)

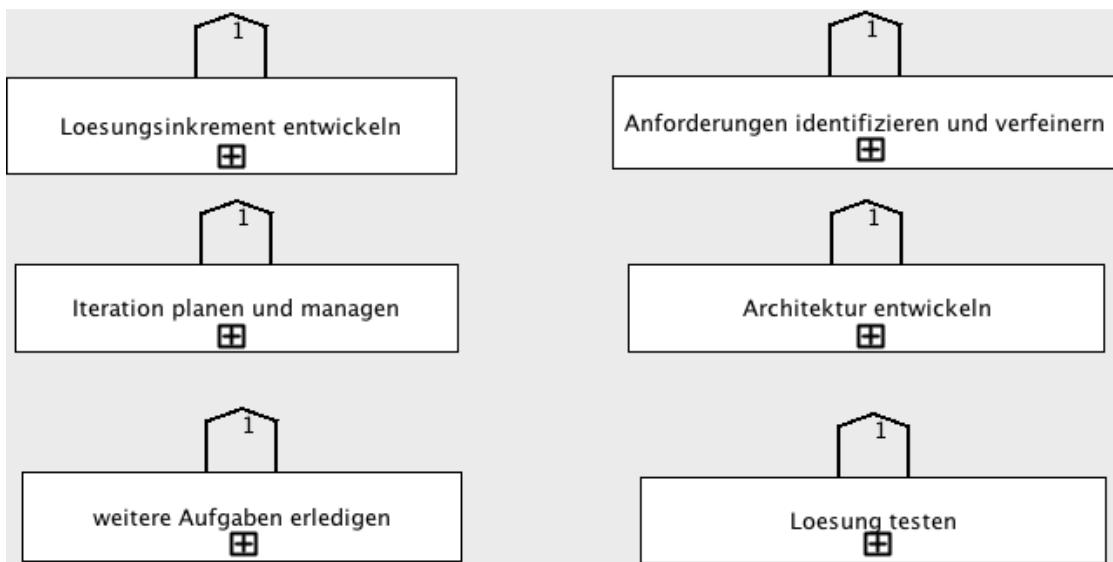


Abbildung 5.24.: Phasen Open UP Unterprozess Elaboration- deklarativ

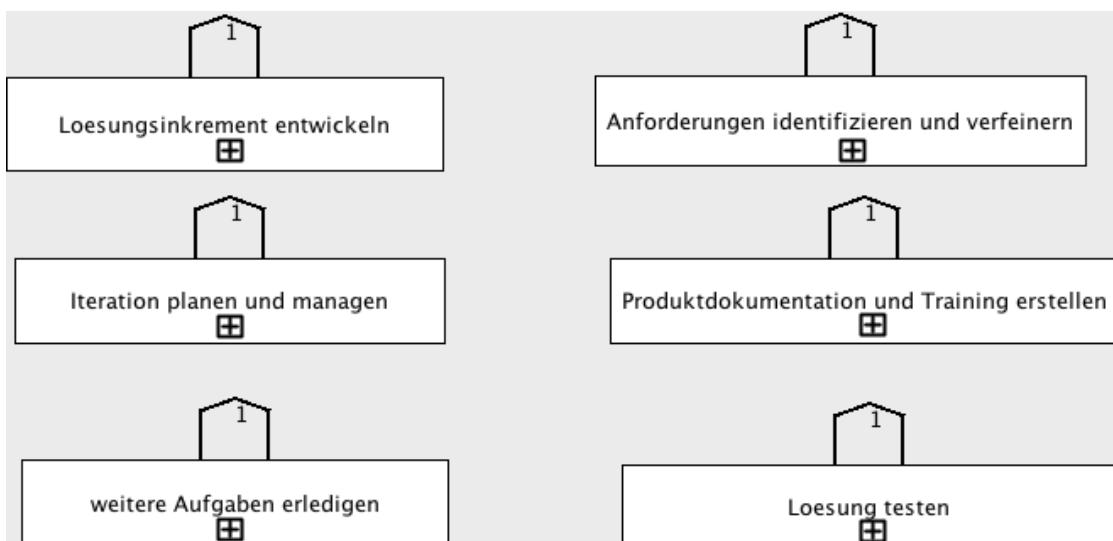


Abbildung 5.25.: Phasen Open UP Unterprozess Construction- deklarativ

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

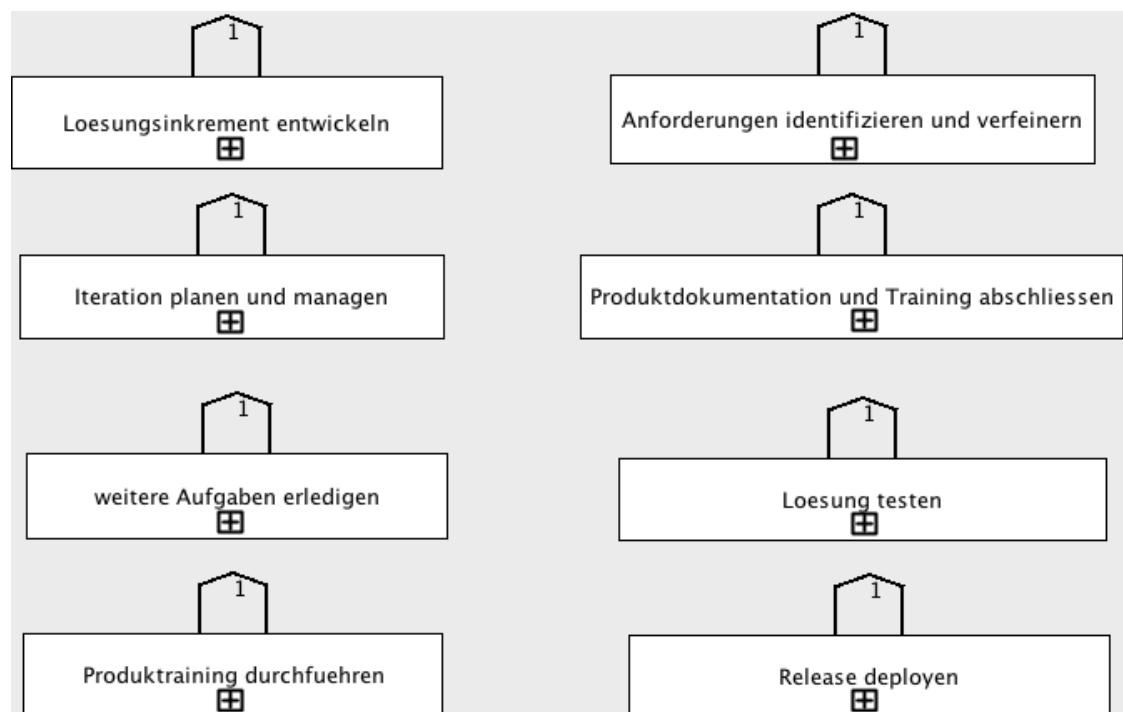


Abbildung 5.26.: Phasen Open UP Unterprozess Transition- deklarativ

5.2. Open Unified Process (Open UP)

Somit wird für die Phase Inception der Unterprozess *Iteration planen und managen*, für die Phase Elaboration der Unterprozess *Anforderungen identifizieren und verfeinern*, für die Phase Construction der Unterprozess *Release deployen* und für die Phase Transition der Unterprozess *Produktdokumentation und Training erstellen* modelliert. Außerdem wird der in den drei Phasen Elaboration, Construction und Transition wiederkehrende Unterprozess *Lösungsincrement entwickeln* modelliert.

Die deklarative Modellierung von Develop Solution Increment kann Abbildung 5.27 entnommen werden.

Falls eine *Lösung designet* wird, muss danach der *Entwicklertest implementiert* werden. Dies ist durch das Constraint *chain response* zwischen diesen beiden Aktivitäten verlangt. Wenn der Entwicklertest implementiert wird, muss er danach auch ausgeführt werden und er kann nur ausgeführt werden, falls er vorher implementiert wurde (Constraints *chain response* und *precedence*).

Bevor die Lösung implementiert werden kann, muss vorher der Entwicklertest ausgeführt werden (Constraint *precedence*) und nach der Implementierung der Lösung muss nochmals der Entwicklertest ausgeführt werden (Constraint *chain response*).

Vor dem *Integrieren* muss der Entwicklertest ausgeführt worden sein, was durch das Constraint *precedence* vorgegeben wird.

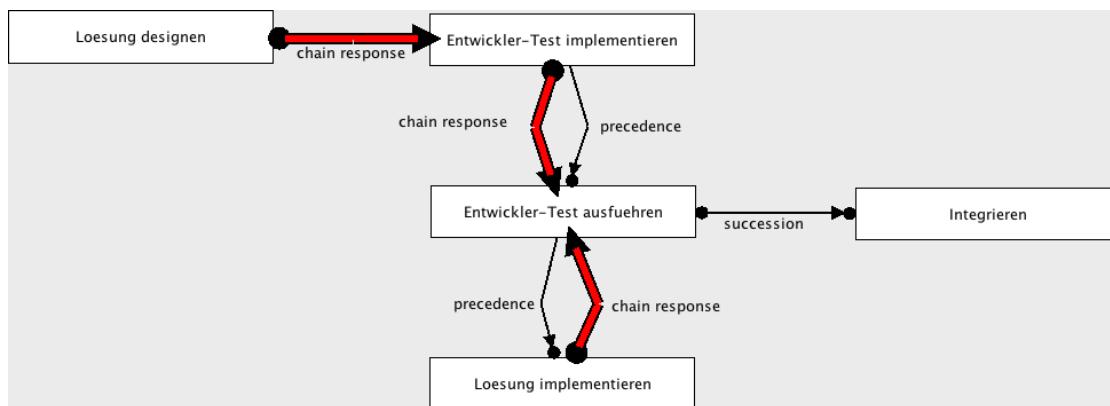


Abbildung 5.27.: Lösungsincrement entwickeln- deklarativ

Die deklarative Modellierung von Plan and manage iteration findet sich in Abbildung *IterationPlanenDec*. Gestartet werden kann mit den Aktivitäten Iteration planen oder Ar-

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

beitsaufgaben aussuchen, da diese unabhängig voneinander ausgeführt werden können und von zwei verschiedenen Personen ausgeführt werden können.

danach können entweder die Aktivitäten *Umgebung vorbereiten* oder *Arbeitsaufgaben in Entwicklungsaufgaben einteilen* ausgeführt werden. Diese sind jeweils mit ihrer Vorgängeraktivität durch das Constraint *succession* verbunden und müssen deshalb auf die Ausführung ihres Vorgängers warten und nach dessen Ausführung ausgeführt werden. Das gleiche Ausführungsverhalten gilt auch für die anderen Aktivitäten im Prozess.

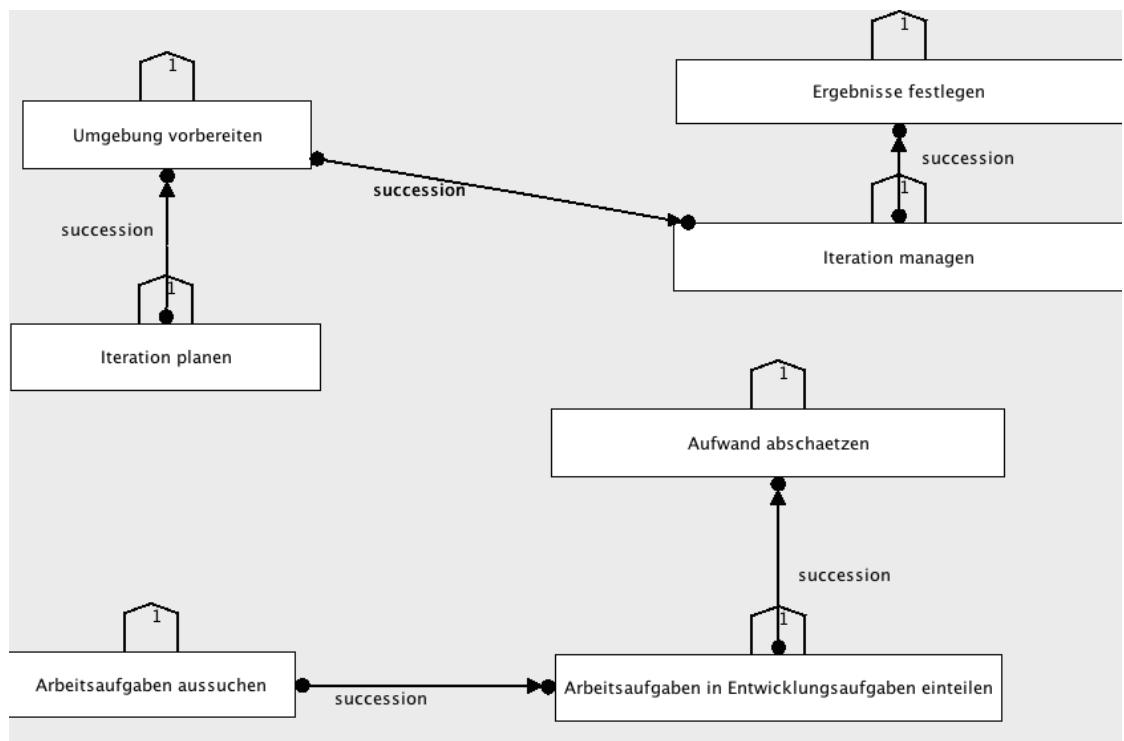


Abbildung 5.28.: Iteration planen und managen-Inception deklarativ

In Abbildung 5.30 ist die deklarative Modellierung von Identify and Refine Requirements dargestellt.

Zu Beginn muss die Aktivität *Anforderungen identifizieren und abgrenzen* ausgeführt werden, was durch das init-Label dargestellt ist. Im Anschluss muss die Aktivität *Use-Case-Szenarien detaillieren* ausgeführt werden, was durch das Constraint *chain response* festgelegt ist. Das Constraint *precedence* legt hingegen fest, dass bevor *Use-Case-Szenarien detaillieren* ausgeführt werden kann, zunächst *Anforderungen identifizieren*

5.2. Open Unified Process (Open UP)

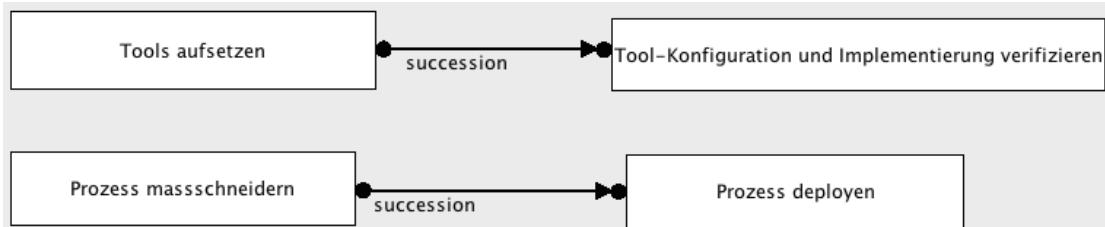


Abbildung 5.29.: Iteration planen und managen- Inception Unterprozess Umgebung vorbereiten- deklarativ

und abgrenzen bearbeitet werden muss. Die gleichen Constraints gelten zwischen Use-Case-Szenarien detaillieren und Systemweite Anforderungen detaillieren sowie zwischen Systemweite Anforderungen detaillieren und Testfaelle erstellen. Alle Aktivitäten werden genau einmal ausgeführt, was jeweils durch das 1-Label dargestellt ist.

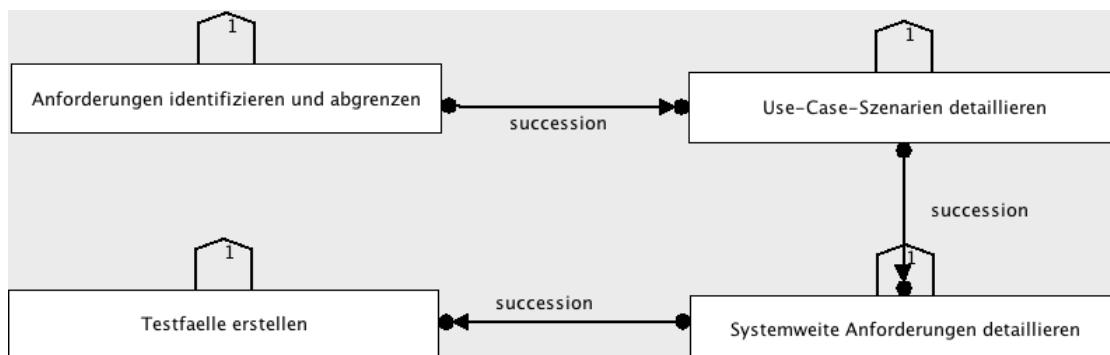


Abbildung 5.30.: Anforderungen identifizieren und verfeinern-Elaboration

Abbildung 5.31 zeigt die deklarative Modellierung von Develop Product Documentation. Zu Beginn muss die Aktivität *Produktdokumentation entwickeln* ausgeführt werden, was durch das init-Label dargestellt ist. Im Anschluss muss die Aktivität *Benutzerdokumentation entwickeln* ausgeführt werden. Dies ist durch das Constraint *chain response* festgelegt ist. Das Constraint *precedence* legt hingegen fest, dass bevor *Benutzerdokumentation entwickeln* ausgeführt werden kann, zunächst *Produktdokumentation entwickeln* bearbeitet werden muss. Die gleichen Constraints gelten zwischen *Benutzerdokumentation entwickeln* und *Unterstützungsdokumentation entwickeln* sowie zwischen *Unterstützungsdokumentation entwickeln* und *Trainingsmaterial entwickeln*. Alle

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

Aktivitäten werden genau einmal ausgeführt, was jeweils durch das 1-Label dargestellt ist.

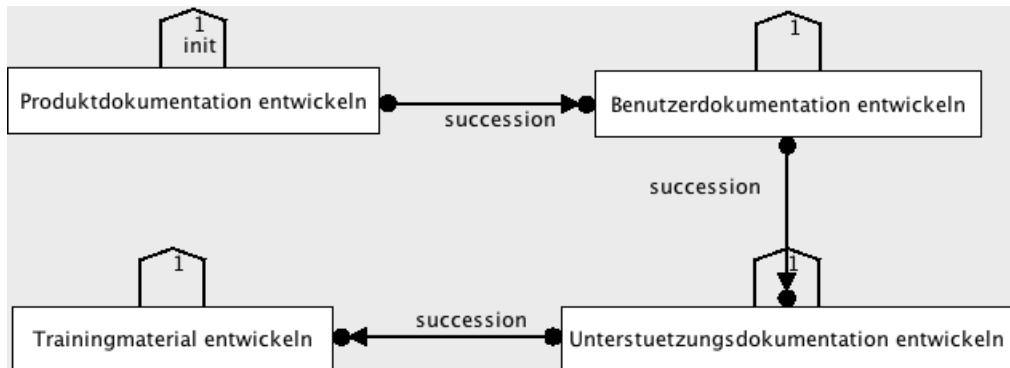


Abbildung 5.31.: Produktdokumentation und Training erstellen-Construction

Abbildung 5.32 zeigt die deklarative Modellierung von Deploy Release.

Zu Beginn muss die Aktivität *Release zusammenstellen* bearbeitet werden, was durch das init-Constraint vorgegeben ist. Danach wird die Aktivität *Deploymentplan ausfuehren* durchgeführt werden (Constraint response), aber erst nachdem *Release zusammenstellen* durchgeführt wurde (Constraint precedence).

Die gleichen Constraints gelten zwischen den Aktivitäten *Deploymentplan ausfuehren* und *erfolgreiches Deployment sicherstellen*.

Nach Abschluss der Aktivität *erfolgreiches Deployment sicherstellen* kann entweder die Aktivität *Backoutplan ausfuehren* ausgeführt werden, welche optional ist (0..1 Constraint) oder die Aktivität *Release-Mitteilungen uebermitteln*, welche auf jeden Fall ausgeführt werden muss.

Nach Ausführung von *Release-Mitteilungen uebermitteln* darf *Backoutplan ausfuehren* nicht mehr durchgeführt werden. Dies stellt das Constraint *not succession* sicher.

5.2. Open Unified Process (Open UP)

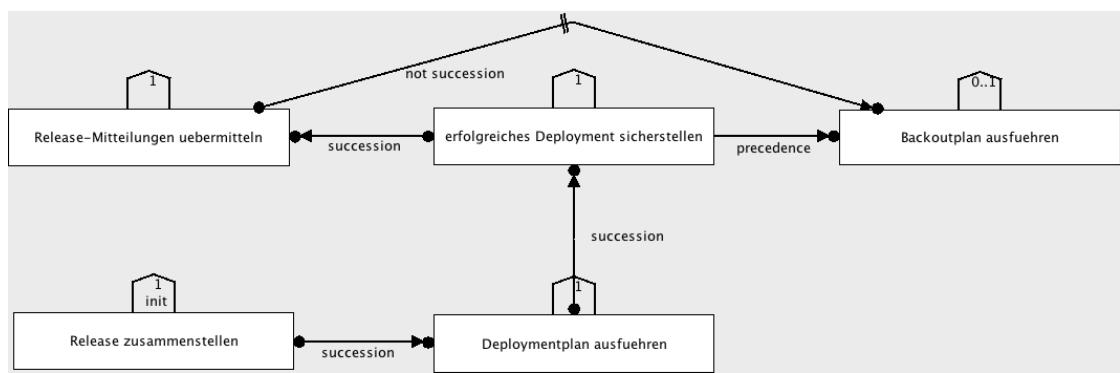


Abbildung 5.32.: Release deployen-Transition

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

5.2.4. Vergleich

Abbildung 5.33 zeigt die Auswertung der Elemente im Modell Phasen des Open UP. Während bei BPMN vier Aktivitäten, acht Gateways (keine unterschiedlichen) und 17 Sequenzflusselemente, also gesamt 29 Elemente für das Modell benötigt werden, werden in ConDec nur vier Aktivitäten und sechs Constraints (zwei verschiedene), also insgesamt zehn Elemente verwendet.

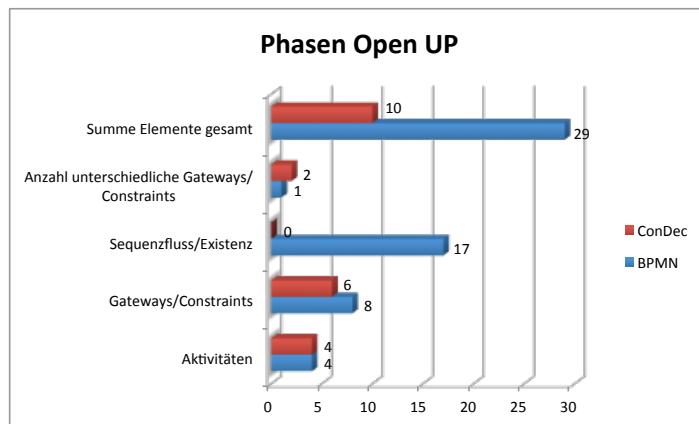


Abbildung 5.33.: Phasen Open UP

Phasen Open UP-Inception

Die Anzahl der Elemente zur Darstellung der Phase Inception in ConDec und BPMN können Abbildung 5.34 entnommen werden. BPMN benötigt somit insgesamt 19 Elemente zur Darstellung dieser Phase (vier Aktivitäten, vier Gateways und 15 Verbindungselemente), ConDec nur zehn (vier Aktivitäten, sechs Constraints). Bei BPMN wird nur eine Sorte an Gateways verwendet und bei ConDec werden zwei unterschiedliche Constraints benötigt.

Zur Darstellung der Phase Elaboration bedarf es in BPMN insgesamt 22 Elementen und in ConDec 14 Elementen wie Abbildung 5.35 entnommen werden kann. Weiterhin werden jeweils sechs Aktivitäten in beiden Prozessmodellierungssprachen verwendet.

5.2. Open Unified Process (Open UP)

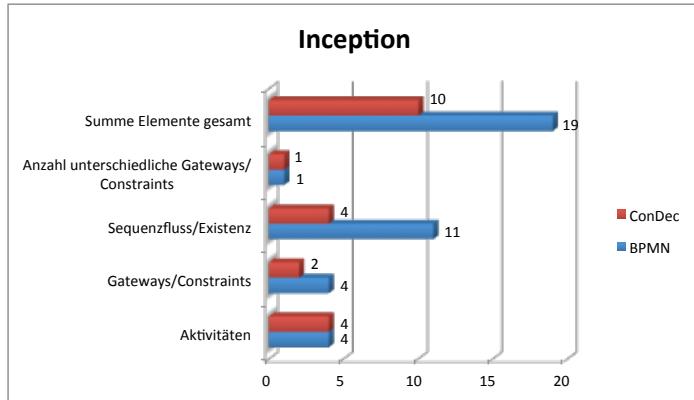


Abbildung 5.34.: Open UP-Inception

In BPMN sind zwei Gateways (keine unterschiedlichen) und 14 Sequenzflusselemente, also insgesamt 16 Verbindungselemente notwendig. ConDec hingegen braucht acht Existenz-Constraints.

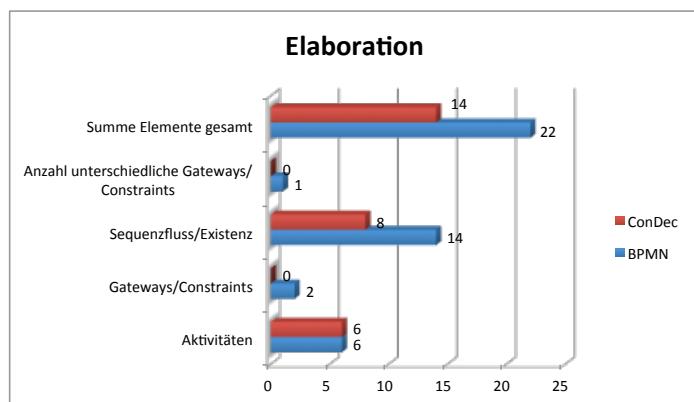


Abbildung 5.35.: Open UP-Elaboration

Abbildung 5.36 kann die Anzahl der Elemente zur Darstellung der Phase Construction entnommen werden. Demnach benötigt es in BPMN 24 Elemente und in ConDec 12 Elemente zur Darstellung des Prozesses. Sowohl in BPMN, als auch in ConDec werden jeweils sechs Aktivitäten benötigt. In BPMN sind vier Gateways (keine unterschiedlichen) und 14 Sequenzflusselemente erforderlich. In ConDec werden sechs

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

Existenz-Constraints zur Darstellung des Ablaufes verwendet.

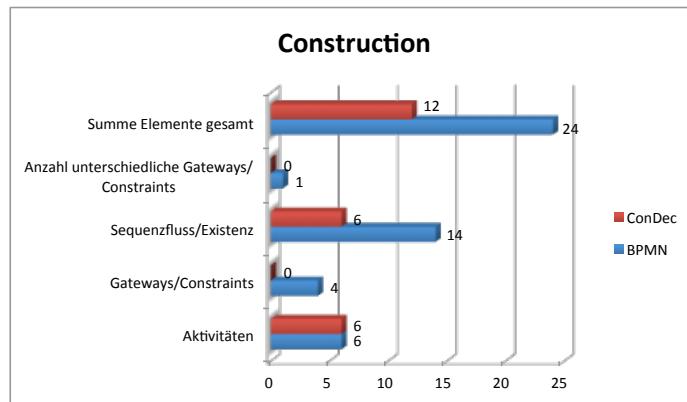


Abbildung 5.36.: Open UP-Construction

Die Anzahl der notwendigen Elemente zur Darstellung der Phase Transition kann Abbildung 5.37 entnommen werden. Somit bedarf es jeweils acht Aktivitäten. BPMN benötigt zwei Gateways (keine unterschiedlichen) und 18 Sequenzflusselemente zur korrekten Modellierung. In ConDec braucht es insgesamt acht Existenz-Constraints zur Darstellung des Ablaufes.

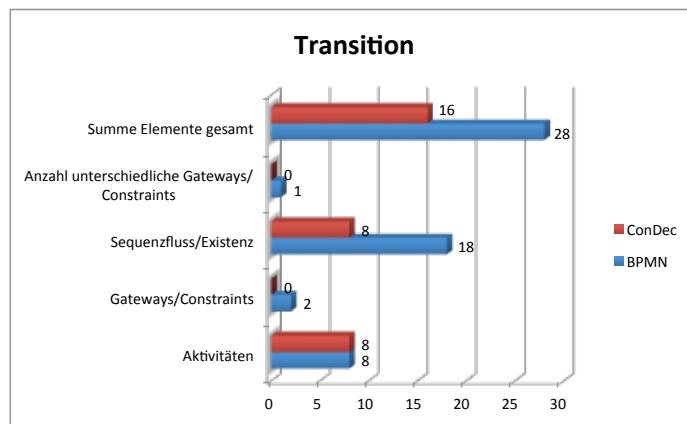


Abbildung 5.37.: Open UP-Transition

5.2. Open Unified Process (Open UP)

In Abbildung 5.38 sind die Anzahl der Elemente zur Darstellung des Prozesses Lösungssinkrement entwickeln abgebildet. Es werden sowohl in BPMN, als auch in ConDec jeweils fünf Aktivitäten verwendet. Weiterhin werden in BPMN 15 Sequenzflusselemente und fünf Gateways gebraucht. Hierbei handelt es sich ausschließlich um XOR-Gateways. Das mit ConDec erstellte Modell benötigt insgesamt sechs Constraints. Hierbei werden insgesamt drei unterschiedliche Constraints benutzt. Insgesamt werden in BPMN 25 Elemente zum Modellieren eingesetzt und in ConDec 11 Elemente.

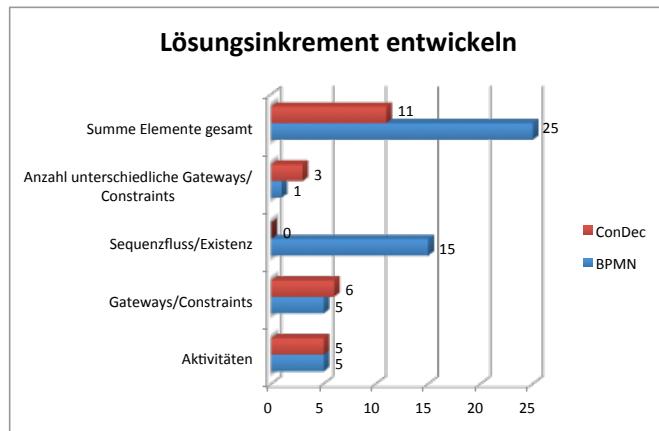


Abbildung 5.38.: Lösungssinkrement entwickeln

Abbildung 5.39 kann die Anzahl der Elemente, welche jeweils zur Darstellung des Unterprozesses Iteration planen und managen notwendig sind entnommen werden. Sowohl in BPMN, als auch in ConDec werden somit jeweils 11 Aktivitäten benötigt. In BPMN werden keine Gateways und 15 Sequenzflüsse verwendet. In ConDec sind zur korrekten Darstellung 21 Constraints notwendig. Somit werden in BPMN insgesamt 26 Elemente und in ConDec 32 Elemente gebraucht.

In Abbildung 5.40 sind die Anzahl der Elemente zur Darstellung von Anforderungen identifizieren und verfeinern abgebildet. Demnach werden sowohl in BPMN, als auch in ConDec jeweils vier Aktivitäten benötigt. Weiterhin sind keine Gateways und 5 Sequenzflusselemente in BPMN zur Darstellung nötig. In ConDec werden drei Constraints

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

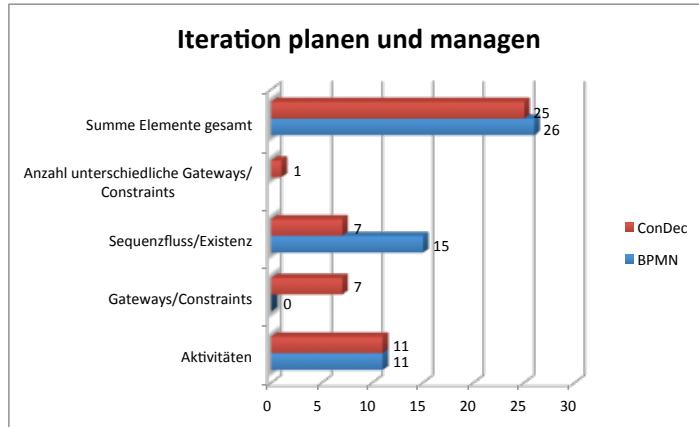


Abbildung 5.39.: Open UP-Iteration planen

verwendet, aber keine unterschiedlichen. Insgesamt werden zur korrekten Abbildung des Metamodells in BPMN neun Elemente und in ConDec 11 Elemente eingesetzt.

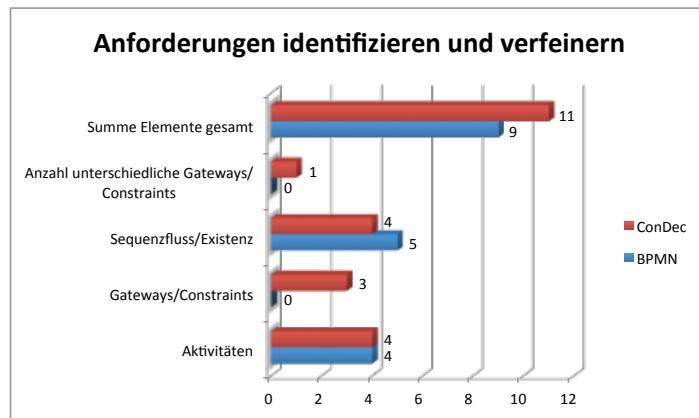


Abbildung 5.40.: Anforderungen identifizieren

Die Anzahl der Elemente zur Darstellung des Prozesses Produktdokumentation entwickeln kann Abbildung 5.41 entnommen werden. Es werden somit jeweils vier Aktivitäten zur Darstellung genutzt. Weiterhin werden in BPMN keine Gateways und sieben Sequenzflusselemente gebraucht. In ConDec sind zur Darstellung sieben Constraints notwendig jedoch keine unterschiedlichen Constraints. Insgesamt werden in BPMN

5.2. Open Unified Process (Open UP)

neun Elemente und in ConDec 11 Elemente verwendet.

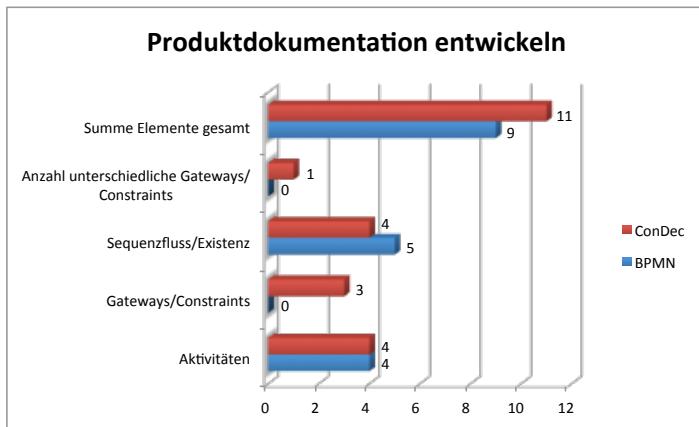


Abbildung 5.41.: Produktdokumentation erstellen

Wie viele Elemente zur Modellierung des Prozesses Release deployen benötigt werden, ist in Abbildung 5.42 aufgeführt. Jeweils fünf Aktivitäten werden in BPMN und ConDec eingesetzt. In BPMN sind zwei Gateways (keine verschiedenen) und neun Sequenzflusselemente zur Darstellung des Ablaufes notwendig. In ConDec werden hierfür fünf Constraints (drei unterschiedliche) und fünf Existenz-Constraints benötigt. Somit sind insgesamt 16 Elemente in BPMN und 15 Elemente in ConDec zur Darstellung notwendig.

Der Grundsatz der syntaktischen *Richtigkeit* kann von beiden Prozessmodellierungssprachen eingehalten werden, denn alle imperativen und deklarativen Modelle lassen sich unter Einhaltung der Modellierungsgrundsätze der jeweiligen Modellierungssprachen erstellen.

Bei dem Grundsatz der semantischen *Richtigkeit* tritt bei ConDec wiederum das Problem auf, dass Rollen und Artefakte nicht darstellbar sind. Zwar ist dies bei den Phasen des Open UP kein Problem, da hier noch keine Rollen und Artefakte zugeordnet werden, jedoch bei den anderen Prozessen des Open UP (Iteration planen und managen, Anforderungen identifizieren und verfeinern, Produktdokumentation entwickeln und Release

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

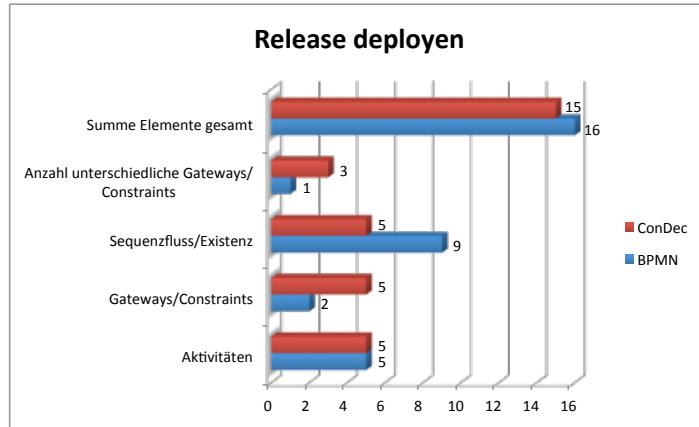


Abbildung 5.42.: Release deployen

deployen). Da hier keine Rollen visualisierbar sind, kann das jeweils im Metamodell enthaltene Verhalten in den ConDec-Modellen nicht vollständig wiedergegeben werden. Hierdurch leidet der Nutzen des Metamodells.

Somit ist BPMN in Bezug auf die *Richtigkeit* die geeigneter Modellierungssprache.

Der Grundsatz des *systematischen Aufbaus* kann wiederum nur von BPMN eingehalten werden, da ConDec keine Möglichkeit bietet, Artefakte im Prozessmodell selbst zu visualisieren.

Auch hier weist BPMN eine bessere Eignung zur Modellierung auf.

Die *Relevanz* lässt sich nur bei BPMN einhalten, denn es ist wieder bei den in ConDec erstellten Modellen nicht möglich, Rollen und Artefakte zu visualisieren.

In Bezug auf die *Klarheit* fällt gerade bei Betrachtung der Modelle der Phasen Inception, Elaboration, Construction und Transition auf, dass bei Modellen bei denen mehrere Aktivitäten parallel zueinander ablaufen in BPMN deutlich mehr Elemente zur Darstellung des gleichen Sachverhaltes notwendig sind, als in ConDec. Bei allen vier Modellen sind doppelt so viele, teilweise auch mehr als doppelt so viele Elemente zur Darstellung in BPMN notwendig als in ConDec. Dies liegt an der Anzahl der Sequenzflusselemente und

5.2. Open Unified Process (Open UP)

Gateways. BPMN weißt eine hohe Anzahl an eingehenden und ausgehenden Kanten an den parallelen Gateways auf. Da die Parallelität in ConDec nicht durch Constraints dargestellt werden muss, werden in ConDec nur die Existenz-Constraints benötigt, um die Aktivitäten auf einmalige Ausführung zu beschränken. Da sich die hohe Anzahl an Elementen in BPMN negativ auf die Verständlichkeit auswirken kann und in ConDec nur die leicht verständlichen Existenz-Constraints verwendet werden, sind die ConDec-Modelle zur Darstellung von parallelen Abläufen die leichter verständlichen Modelle.

Das Gleiche gilt für die Darstellung der Phasen des Open UP. Hier werden zur Darstellung des Prozesses in BPMN drei-mal so viele Elemente benötigt wie in ConDec. Dies liegt hier an der hohen Anzahl der XOR-Gateways, die beim Modellieren des Prozesses notwendig sind. Somit ist auch hier das Modell, welches mit ConDec erstellt wurde das leichter verständliche.

Beim Prozess Lösungssinkrement entwickeln werden bei BPMN mehr als doppelt so viele Elemente benötigt wie bei ConDec. Es werden fünf Gateways in BPMN verwendet und bei ConDec werden sechs Constraints benötigt. Bei ConDec werden drei verschiedene Constraints verwendet. Dies erhöht die Komplexität im Gegensatz zum BPMN-Modell deutlich. Daher kann zusammenfassend gesagt werden, dass die Komplexität bei beiden Prozessmodellen in etwa gleich groß ist, da BPMN zwar deutlich mehr Elemente insgesamt aufweist, die Kombination von drei verschiedenen Constraints das ConDec-Modell jedoch komplexer macht.

Im Prozess Iteration planen und managen gibt es keine Verzweigungen. In ConDec werden sieben Constraints zur Darstellung verwendet. Die Anzahl der Elemente insgesamt ist bei beiden Modellen ungefähr gleich groß (ein Elemente mehr bei BPMN). Auf Grund der geringeren Komplexität des mit BPMN erstellten Modelles, da dort keine Gateways eingesetzt werden müssen, ist dieses das verständlichere.

Bei den Prozessen Anforderungen identifizieren und verfeinern und Produktdokumentation entwickeln weißen die ConDec-Modelle eine leicht höhere Anzahl an Elementen insgesamt auf. Außerdem werden hier auch in ConDec jeweils drei Constraints verwendet und in BPMN keine Gateways. Dies macht die ConDec-Modelle komplexer als die BPMN-Modelle, weswegen die mit BPMN erstellten Modelle verständlicher sind.

Beim Prozess Release deployen werden fast gleich viele Elemente in ConDec, wie

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

auch in BPMN zur Darstellung des Prozesses benötigt (ein Element mehr in BPMN). In ConDec werden insgesamt fünf Constraints (drei verschiedene) zur Darstellung des Sachverhaltes benötigt, in BPMN nur zwei Gateways. Somit liegt hier das BPMN-Modell in Bezug auf die Komplexität unter dem ConDec-Modell und ist somit das verständlichere.

Somit liegt ConDec bei der Darstellung der Phasen, in denen viele parallele Abläufe dargestellt werden bei der *Klarheit* vorne, während bei den anderen Modellen, bei denen es Schleifen oder Abläufe ohne Verzweigungen zum Darstellen gibt BPMN vorne liegt.

Die *Wirtschaftlichkeit* lässt sich auch beim Open UP bei den Prozessmodellen Inception, Elaboration, Construction und Transition sowie bei den Phasen des Open UP mit ConDec besser einhalten, als mit BPMN. Da hier bei BPMN deutlich mehr Elemente benötigt werden als bei ConDec und da bei diesen Modellen zur Darstellung in BPMN Gateways verwendet werden müssen und bei ConDec keine Constraints, sind die ConDec Modelle hier weniger komplex und somit mit weniger geistigem Aufwand erstellbar, als die BPMN-Modelle.

Bei den anderen Modellen des Open UP jedoch gibt es bei den mit ConDec erstellten Modellen deutlich mehr Constraints und auch unterschiedliche Constraints, als Gateways bei BPMN. Aus diesem Grund sind hier genau gegensätzlich die ConDec- Modelle komplexer und benötigen somit einen höheren geistigen Aufwand zum Erstellen.

Somit können hier einerseits mit ConDec bei der Darstellung der Phasen des Open UP und andererseits bei den anderen Modellen des Open UP die BPMN-Modelle die *Wirtschaftlichkeit* besser erfüllen und somit kann hier keine der beiden Sprachen als geeigneter angesehen werden.

Die Abläufe der Diagramme wurden in Signavio und Declare getestet und somit ist hier ihre *Vergleichbarkeit* gewährleistet.

Bei der Darstellung der Phasen des Open UP werden doppelt so viele Elemente in ConDec, wie in BPMN verwendet. Hier ist deshalb die *Vergleichbarkeit* nicht ganz gewährleistet. Bei den anderen Modellen des Open UP unterscheidet sich die Anzahl der Elemente kaum voneinander, wodurch die Vergleichbarkeit hier gewährleistet ist.

Bei ConDec muss wieder auf die Darstellung von Artefakten und Rollen verzichtet werden, wodurch die *Vergleichbarkeit* der Modelle von ConDec behindert wird. Somit weisen hier beide Prozessmodellierungssprachen Stärken und Schwächen auf.

Eine Zusammenfassung, welche Modellierungssprache bei welchem Grundsatz eher überzeugt hat, kann Abbildung 5.43 entnommen werden.

Modellierungsgrundsatz		Geeignete Modellierungssprache
Richtigkeit	syntaktisch	BPMN, ConDec
	semantisch	BPMN
Systematischer Aufbau		BPMN
Relevanz		BPMN
Klarheit		BPMN (Schleifen und Verzweigungen, gerade Abläufe), ConDec (Parallele Abläufe)
Wirtschaftlichkeit		BPMN (Schleifen und Verzweigungen, gerade Abläufe), ConDec (Parallele Abläufe)
Vergleichbarkeit		BPMN, ConDec

Abbildung 5.43.: Übersicht Vergleich Open UP

5.3. V-Modell XT

Das V-Modell XT zählt zu den schwergewichtigen Prozessmodellen [Han10]. Es wird als Entwicklungsstandard für die Durchführung von IT-Vorhaben in der öffentlichen Verwaltung in Deutschland herangezogen [KLS11]. Beschrieben werden im V-Modell XT die Abläufe im Verlauf eines Entwicklungsprojektes über Produkte, Rollen und Aktivitäten [FHK08]. Es wird somit ganz genau geregelt, *Wer*, *Wann*, *Was* in einem Projekt zu tun hat [Bun04]. Die Vorgehensbausteine ermöglichen neben einer Modularisierung der

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

Abläufe auch eine flexible Zusammenstellung, wodurch das V-Modell XT auf die jeweils eigene Situation angepasst werden kann. [FHKS08, Zö12].

5.3.1. Analyse V-Modell XT

Abbildung 5.44 zeigt die Grundstruktur des V-Modell XT, welche im Folgenden detailliert erläutert wird.

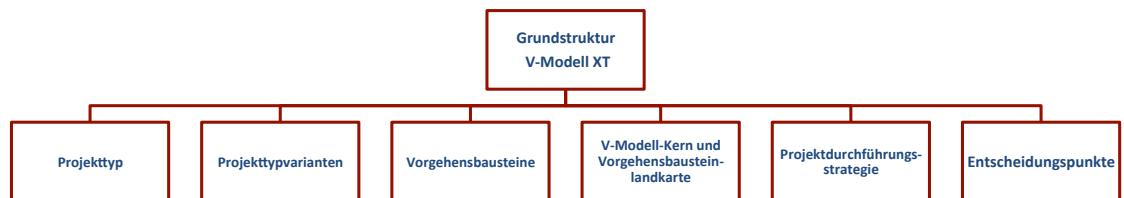


Abbildung 5.44.: Grundstruktur V-Modell XT nach [Bun04]

Projekttypen

Nicht alle V-Modell-Projekttypen laufen nach exakt demselben Schema ab. Auf Grund ihrer charakteristischen Eigenschaften lassen sie sich demnach in unterschiedliche Projekttypen einteilen. Abbildung 5.45 gibt einen ersten Überblick über die verschiedenen Projekttypen im V-Modell XT [Bun04].

5.3. V-Modell XT

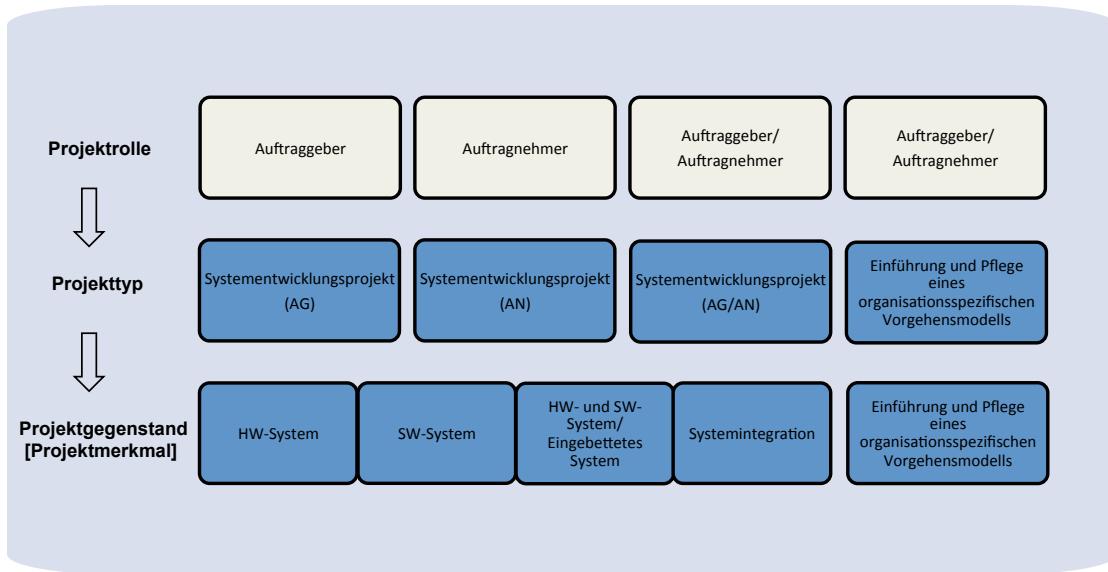


Abbildung 5.45.: Projekttypen V-Modell XT nach [Bun04]

Es existieren somit vier verschiedene Projekttypen: *Systementwicklungsprojekt eines Auftraggebers*, *Systementwicklungsprojekt eines Auftragnehmers*, *Systementwicklungsprojekt eines Auftraggebers/Auftragnehmers* und *Einführung und Pflege eines organisationsspezifischen Vorgehensmodells* [HRB⁺08].

Es werden drei verschiedene Projektrollen unterschieden, welche dem jeweiligen Projekttyp entsprechen: In der Rolle *Auftragnehmer* wird ein vom *Auftraggeber* spezifiziertes System entwickelt. Die Systementwicklung wird an einen oder mehrere *Arbeitnehmer* weiter gegeben, wenn man sich in der Rolle *Arbeitgeber* befindet. Das System wird selbst entwickelt in der Rolle *Auftraggeber/Auftragnehmer* [Bra10, Bun04].

Beim *Systementwicklungsprojekt eines Auftraggebers* wird die Entwicklung des Projektgegenstandes im Projektverlauf ausgeschrieben und der Auftragnehmer trifft eine Auswahl anhand der eingehenden Angebote. Der Auftragnehmer, welcher für die Entwicklung des Projektgegenstandes ausgewählt wurde, entwickelt den Projektgegenstand, welcher dann vom Auftragnehmer abgenommen wird [HRB⁺08, Bun04].

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

Umgekehrt wird beim *Systementwicklungsprojekt eines Auftragnehmers* im Laufe des Projektes ein Angebot erstellt und bei Auswahl durch den Auftraggeber ein Projektgegenstand entwickelt, welcher abschließend an den Auftraggeber ausgeliefert und von diesem abgenommen wird [HRB⁺08, Bun04].

Bei *Einführung und Pflege eines organisationsspezifischen Vorgehensmodells* geht es um Projekte, welche Prozessmodelle z.B. das V-Modell einführen und verbessern wollen. Für diesen Zweck ist eine Analyse des vorherigen Prozessmodells notwendig und etwaige Verbesserungsmöglichkeiten sind zu erfassen und durchzuführen [HRB⁺08, Bun04]. Wie aus Abbildung 5.45 ersichtlich ist, kann es sich im V-Modell XT beim Projektgegenstand um ein Hardware (HW)-System, ein Software (SW)-System, ein eingebettetes System oder eine Systemintegration handeln [Bra10, Bun04].

Projekttypvarianten

Für jeden der Projekttypen, gibt es im V-Modell XT mindestens eine passende Projekttypvariante. Diese bestimmt die Rahmenbedingungen für mögliche Abläufe eines Projektes. In Abbildung 5.46 sind die verschiedenen Projekttypvarianten des V-Modell XT aufgelistet und es wird gezeigt, mit welchen Merkmalen die zugehörigen Projekttypvarianten ausgewählt werden können [Bun04].

Für den Projekttyp *Systementwicklungsprojekt (AG)* existieren zwei verschiedene Projekttypvarianten, welche je nach *Auftragsstruktur* ausgewählt werden. Falls der Auftraggeber mit nur einem Auftragnehmer zusammen arbeitet, ergibt sich die Projekttypvariante *Systementwicklungsprojekt (AG)- Projekt mit einem Auftragnehmer*. Arbeitet der Auftraggeber mit mehreren Auftragnehmern zusammen, ergibt sich die Projekttypvariante *Systementwicklungsprojekt (AG)- Projekt mit mehreren Auftragnehmern* [Bun04].

Bei den Projekttypen *Systementwicklungsprojekt (AN)* und *Systementwicklungsprojekt (AG/AN)* wird die Unterscheidung anhand des Systemlebenszyklusausschnitt des Projektes durchgeführt. Somit wird in den Systemlebenszyklusausschnitten Entwicklung, Weiterentwicklung und Migration eine andere Projekttypvariante gewählt, als in Wartung und Pflege [Bun04].

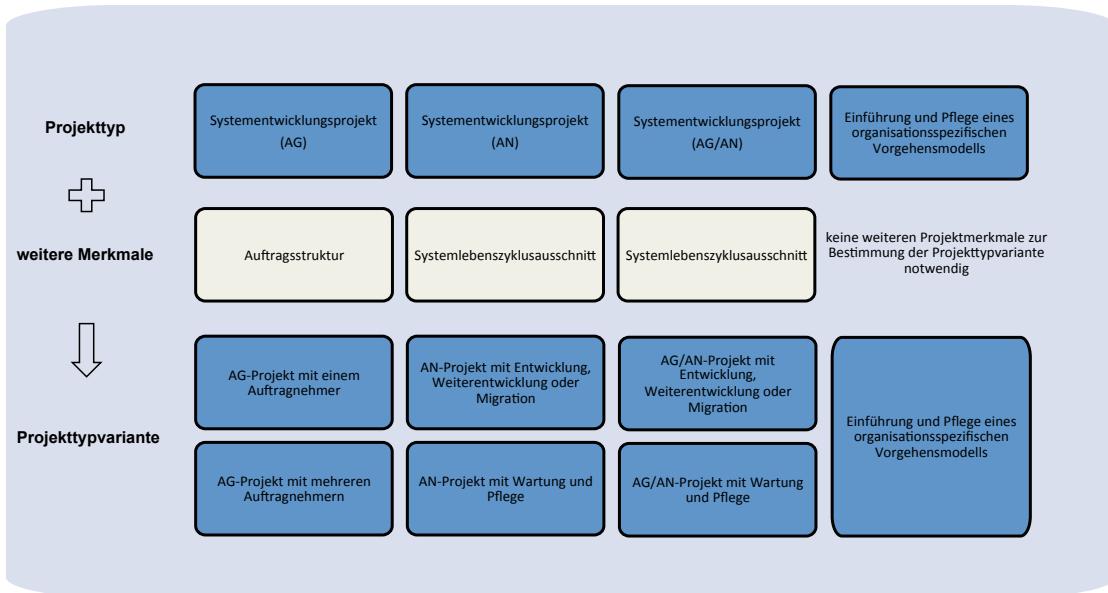


Abbildung 5.46.: Zuordnung der Projekttypvarianten zu den Projekttypen des V-Modell XT [Bun04]

Für den Projekttyp *Einführung und Pflege eines organisationsspezifischen Vorgehensmodells* existiert nur eine einzige Projekttypvariante, weshalb hier keine weiteren Merkmale zur Bestimmung der Projekttypvariante notwendig sind [Bun04].

Vorgehensbausteine

Modulare, aufeinander aufbauende Vorgehensbausteine bilden den Kern des V-Modell XT. Vorgehensbausteine sind selbständige entwickelbare und änderbare Einheiten und bestehen aus Aktivitäten, Produkten und Rollen. Sie geben einerseits vor, „Was“ in einem Projekt zu tun ist, also welche Produkte zu erstellen sind und andererseits „Wer“, also welche konkrete Rolle für das jeweilige Produkt verantwortlich ist. Abbildung 5.47 gibt einen Überblick über diese [RF08, Bun04].

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

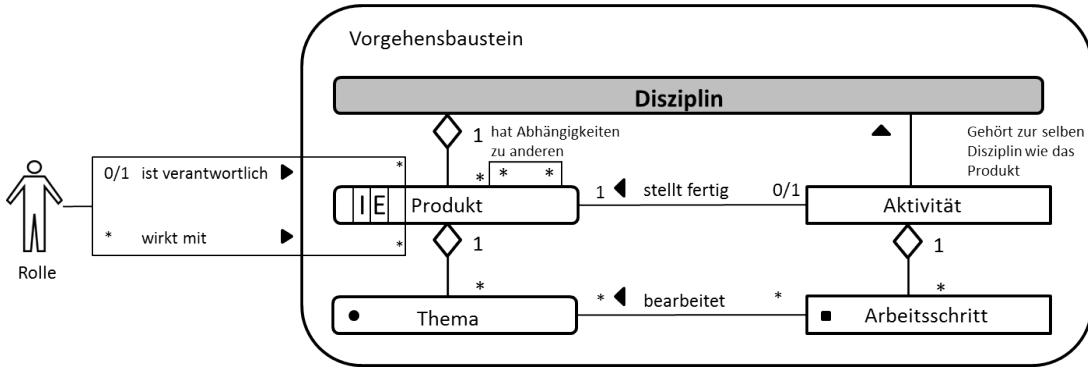


Abbildung 5.47.: Vorgehensbausteine V-Modell XT nach [Bun04]

Ergebnisse und Zwischenergebnisse werden Produkte genannt. Komplexe Produkte können in ein oder mehrere Themen gegliedert werden und inhaltlich zusammengehörende Produkte können zu einer Disziplin zusammengefasst werden. Produkte können hierbei auch voneinander abhängig sein, sowohl innerhalb eines Vorgehensbausteins, als auch zwischen verschiedenen Vorgehensbausteinen [Bun04].

Jedes Produkt wird von genau einer Aktivität fertig gestellt. Aktivitäten legen auch fest, wie die einzelnen Produkte zu bearbeiten sind. Sie bestehen aus einer oder mehreren Teilaktivitäten, sogenannten Arbeitsschritten. Diese stellen eine Art Arbeitsanleitung dar und bearbeiten eine oder mehrere Themen [Bun04].

Durch Rollen werden eine Menge von Aufgaben und Verantwortlichkeiten gekapselt, wodurch das V-Modell XT unabhängig von organisatorischen Rahmenbedingungen bleibt. Eine Zuordnung von Personen, bzw. Organisationseinheiten zu einer Rolle erfolgt erst zu Beginn eines Projektes. Es wird jedem Produkt genau eine Rolle als Verantwortlicher zugewiesen, weitere Rollen können am Produkt als Mitwirkende mitarbeiten [Bun04].

V-Modell-Kern und Vorgehensbausteinlandkarte

Um ein spezifisches Projekt an ein V-Modell-Projekt anzupassen, ist für jeden Projekttyp und jede Projekttypvariante genau vorgegeben, welche Vorgehensbausteine jeweils anzuwenden sind [Bun04]. Hierdurch kann also ein individuelles V-Modell für ein Pro-

5.3. V-Modell XT

jekt erstellt werden [Hei07]. Hierfür ist es notwendig, die Vorgehensbausteine für ein V-Modell-Projekt nach den Vorgaben des Projekttyps auszuwählen und festzulegen [Bun04].

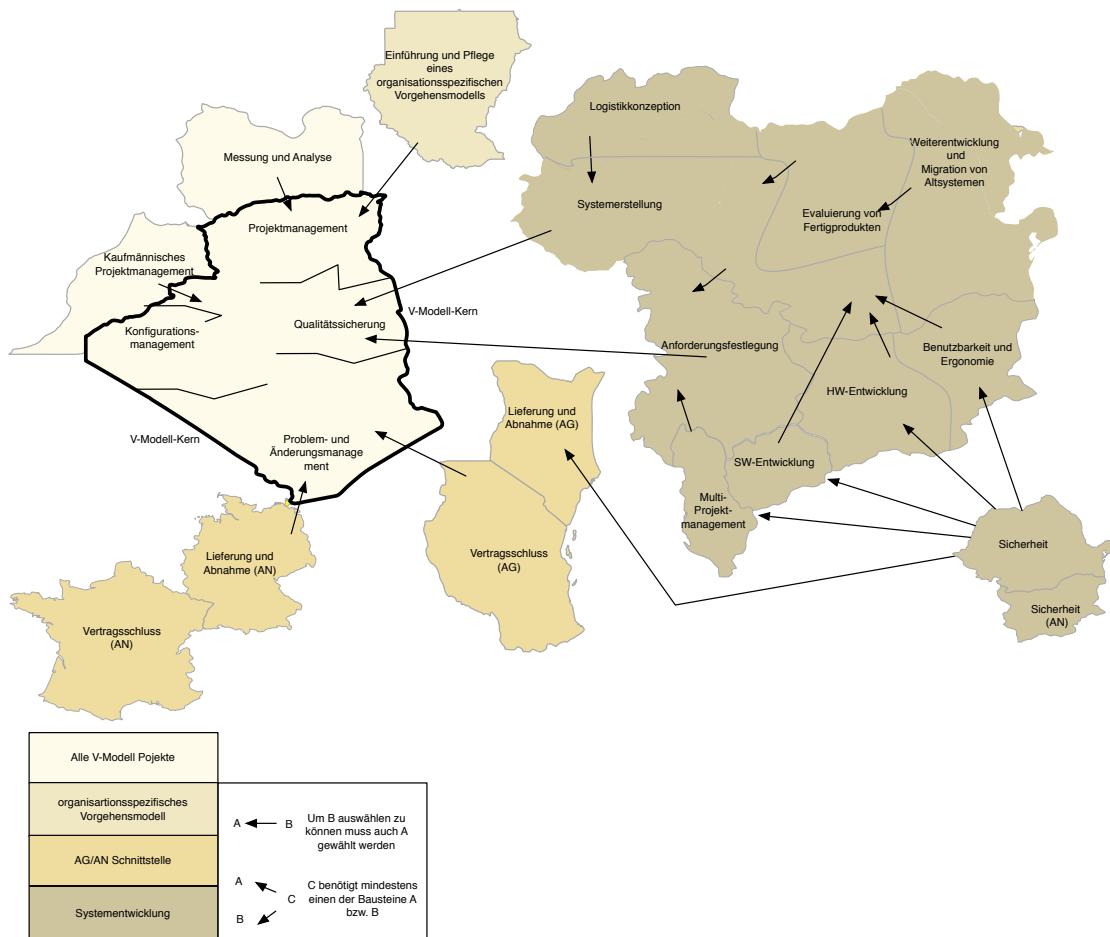


Abbildung 5.48.: V-Modell-Kern und Vorgehensbausteinlandkarte nach [Bun04]

Wie Abbildung 5.48 zeigt, können die Vorgehensbausteine in die vier Bereiche *Alle V-Modell-Projekte*, *Organisationsspezifisches Vorgehensmodell*, *AG/AN-Schnittstelle* und *Systementwicklung* eingeteilt werden [Bun04].

Im Bereich *Alle V-Modell-Projekte* finden sich diejenigen Vorgehensbausteine, welche in jedem V-Modell-Projekt herangezogen werden können. Zudem gibt es den V-Modell-Kern, in welchem sich die Vorgehensmodelle finden, die in jedem V-Modell-Projekt

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

unerlässlich sind: *Projektmanagement, Konfigurationsmanagement, Problem- und Änderungsmanagement* und *Qualitätssicherung*. Zusätzlich zu diesen verpflichtenden Vorgehensbausteinen können in jedem Projekt noch *Kaufmännisches Projektmanagement*, welches bei der Integration des Projektmanagements in das kaufmännische Management hilft und *Messung und Analyse*, welches Verfahren für die organisationsweite und projektübergreifende Erfassung und Auswertung von Kennzahlen bereitstellt, verwendet werden [Bun04].

Ist der Zweck eines Projektes die Entwicklung eines *Organisationsspezifischen Vorgehensmodells*, so muss der Vorgehensbaustein *Einführung und Pflege eines organisationsspezifischen Vorgehensmodells* hinzugenommen werden. In diesem finden sich Verfahren und Richtlinien für die Einführung eines Vorgehensmodells innerhalb einer Organisation sowie die damit einhergehende Etablierung eines stetigen Verbesserungsprozesses [Bun04].

Wenn ein Projekt die Entwicklung eines Systems zum Ziel hat, so wird der Bereich *Systementwicklung* herangezogen. In diesem befinden sich die Vorgehensbausteine *Anforderungsfestlegung, Systemerstellung, HW-Entwicklung, SW-Entwicklung, Logistikkonzeption, Weiterentwicklung und Migration von Altsystemen, Evaluierung von Fertigprodukten, Benutzbarkeit und Ergonomie, Sicherheit sowie Sicherheit (AN)* und *Multi-Projektmanagement* [Bun04].

Im Bereich *AG/AN-Schnittstelle* befinden sich die Vorgehensbausteine für die Kommunikation zwischen Arbeitgeber und Arbeitnehmer: *Lieferung und Abnahme (AG), Lieferung und Abnahme (AN), Vertragsschluss (AG) und Vertragsschluss (AN)*. Hier finden sich Regelungen über den Vertrag zwischen Arbeitgeber und Arbeitnehmer sowie über Lieferung und Abnahme des Entwicklungsgegenstandes [Bun04].

Projektdurchführungsstrategie

Die Vorgehensbausteine im V-Modell XT geben zwar an, welche Produkte jeweils zu erstellen und welche Aktivitäten durchzuführen sind, sie geben jedoch hierbei nicht vor, in welcher Reihenfolge dies geschehen soll. Damit das Projekt trotzdem geplant und

gesteuert werden kann, gibt es im V-Modell eine Projektdurchführungsstrategie welche auf den jeweiligen Projekttyp und die Projekttypvariante abgestimmt ist. Hier wird somit die Reihenfolge der Produkte und Aktivitäten festgelegt, also das “Wann“ festgelegt. Außerdem werden hier zu erreichende Projektfortschrittsstufen vorgegeben [Bun04].

Entscheidungspunkte

Abbildung 5.49 zeigt, dass die in der Projektdurchführungsstrategie vorgegebenen Projektfortschrittsstufen bei Erreichen durch Entscheidungspunkte markiert werden, welche einen Meilenstein im Projektlauf darstellen. Um den Entscheidungspunkt zu erreichen, muss eine vorgegebene Menge an Produkten fertig gestellt werden. Hier entscheidet das Projektmanagement über das Erreichen der Projektfortschrittsstufe und das Freigeben des nächsten Projektabschnitts. Die Entscheidungspunkte, welche im V-Modell XT erreicht werden müssen können Abbildung 5.50 entnommen werden. Diese werden wie im V-Modell-Kern in die vier Bereiche *Alle V-Modell-Projekte*, *Organisationsspezifisches Vorgehensmodell*, *AG/AN-Schnittstelle* und *Systementwicklung* unterschieden [Bun04]. Demnach gelten die Entscheidungspunkte *Projekt genehmigt*, *Projekt definiert*, *Iteration geplant* und *Projekt abgeschlossen* für alle Projekttypen und Projektdurchführungsstrategien [Bun04].

Bei der Systementwicklung werden die Entscheidungspunkte *Anforderungen festgelegt*, *System spezifiziert*, *System entworfen*, *Feinentwurf abgeschlossen*, *Systemelemente realisiert* und *System integriert* verwendet. Falls das Projekt vor der Anforderungserhebung in mehrere Teilmodelle aufgeteilt werden soll, werden zusätzlich die Entscheidungspunkte *Gesamtprojekt aufgeteilt* und *Gesamtprojektfortschritt überprüft* hinzugenommen [Bun04].

Die Entscheidungspunkte für die Arbeitgeber/Arbeitnehmer Schnittstelle setzen sich aus *Projekt ausgeschrieben*, *Angebot abgegeben*, *Projekt beauftragt*, *Lieferung durchgeführt*, *Abnahme erfolgt* und *Projektfortschritt überprüft* zusammen [Bun04].

Bei der Entwicklung eines organisationsspezifischen Vorgehensmodells kommen die Entscheidungspunkte *Vorgehensmodell analysiert*, *Verbesserung Vorgehensmodell konzi-*

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

piert und Verbesserung Vorgehensmodell realisiert zum Einsatz [Bun04].

Die Entscheidungspunkte legen das “Wann“ und “Was“ fest, d.h. wann welche Produkte fertig gestellt sein müssen.

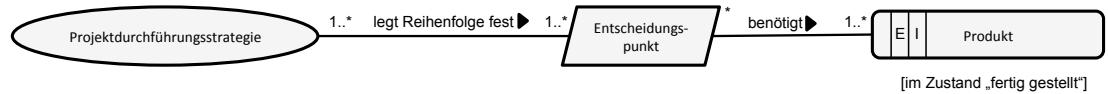


Abbildung 5.49.: Entscheidungspunkte V-Modell XT nach [Bun04]

Im Folgenden werden Teile des V-Modells XT modelliert da das ganze V-Modell in dieser Arbeit nicht modelliert werden kann. Aus diesem Grund wird zum Einen das *Systementwicklungsprojekt AG/AN* modelliert. Weiterhin wird ein hierzu gehöriger Unterprozess *Inkrementelle Entwicklung* und die hierzu gehörenden Unterprozesse *System entwerfen* und *System spezifizieren* modelliert.

5.3. V-Modell XT

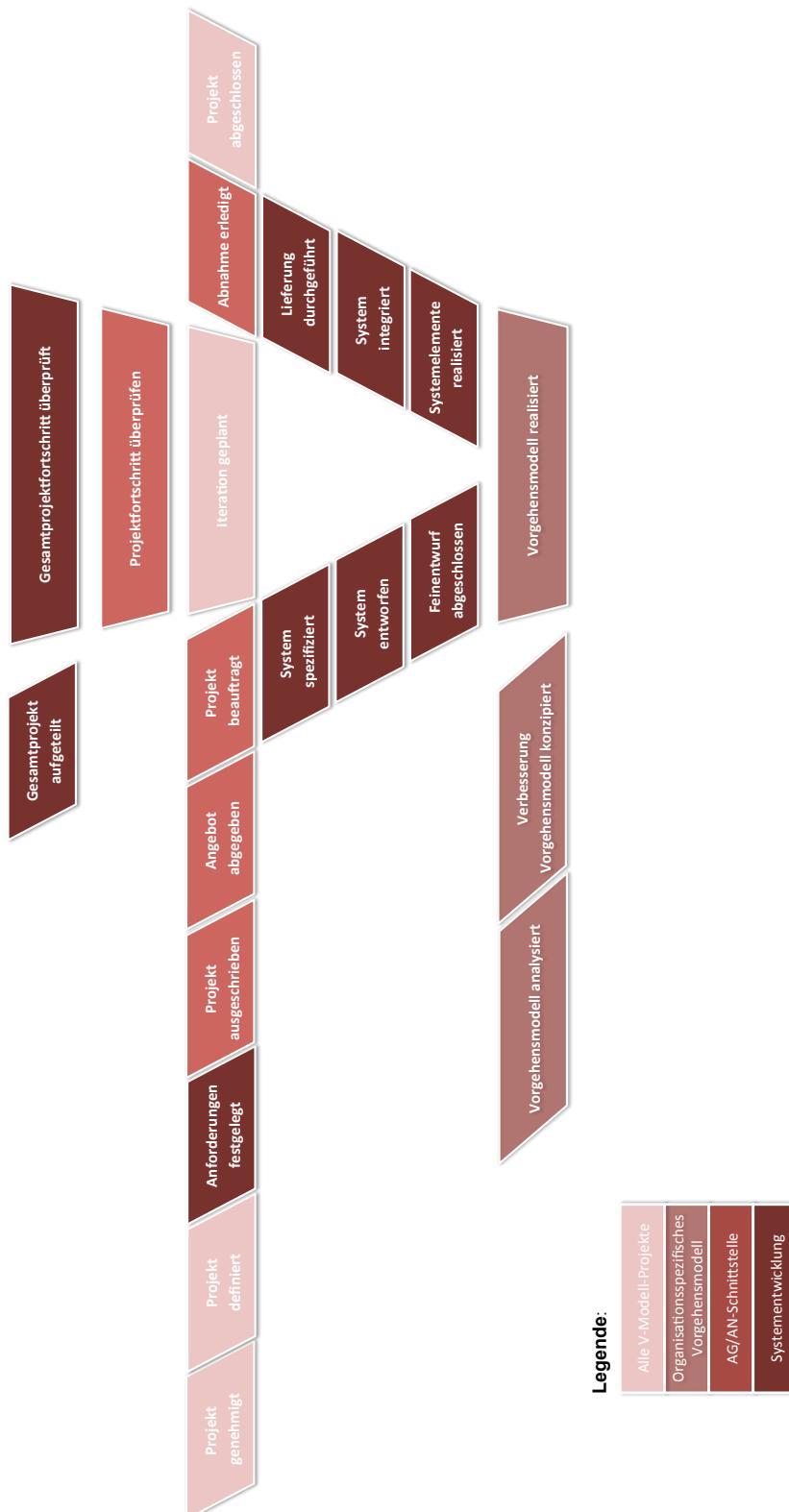


Abbildung 5.50.: Entscheidungspunkte für die Projektdurchführungsstrategie nach [Bun04]

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

5.3.2. Imperative Modellierung V-Modell

Systementwicklungsprojekt AG/AN

Die imperative Modellierung von *Systementwicklungsprojekt AG/AN* zeigt Abbildung 5.57.

Zunächst muss ein Projekt genehmigt und definiert werden. Dies ist durch die einander folgenden Aktivitäten *Projekt genehmigen* und *Projekt definieren* dargestellt.

In der nachfolgenden Aktivität müssen sodann die *Anforderungen festgelegt werden*, bevor die *Iteration geplant* werden kann.

Hiernach muss entschieden werden, ob eine *Prototypische Entwicklung durchgeführt*, eine *Komponentenbasierte Entwicklung durchgeführt* oder eine *Inkrementelle Entwicklung durchgeführt* werden soll. Dies wird durch das XOR-Gateway beschrieben, welches nur eine Alternative zulässt.

Anschließend wird das *System abgenommen*.

An dieser Stelle wird entschieden, ob erneut zu *Anforderungen festlegen* zurückgekehrt wird und der Prozess ab dieser Aktivität erneut startet oder ob zu *Projekt ausschreiben* zurückgekehrt wird und der Prozess ab hier erneut startet. Ansonsten endet der Prozess mit der Aktivität *Projekt abschließen*.

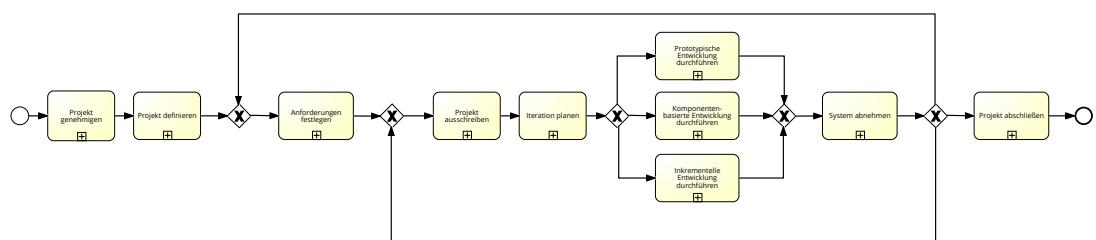


Abbildung 5.51.: Systementwicklungsprojekt AG/AN V-Modell - imperativ

Inkrementelle Entwicklung durchführen

In Abbildung 5.52 ist die imperative Modellierung des Unterprozesses *Inkrementelle Entwicklung durchführen* abgebildet.

Zu Beginn muss das *System spezifiziert* werden und anschließend wird das *System entworfen*.

Hier nach wird der *Feinentwurf entworfen* und es werden die *Systemelemente realisiert*. Diese beiden Aktivitäten können so oft wie nötig durchgeführt werden, was durch das XOR-Gateway beschrieben ist.

Im nächsten Schritt wird das *System integriert* und es beginnt eine neue Iteration bei der Aktivität *System entwerfen*.

Falls keine weitere Iteration mehr notwendig ist, wird die *Lieferung durchgeführt* und der Unterprozess endet hier.

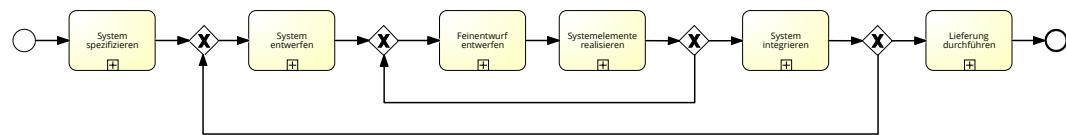


Abbildung 5.52.: Unterprozess Inkrementelle Entwicklung durchführen V-Modell - imperativ

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

System entwerfen

Abbildung 5.53 zeigt die imperative Modellierung des Unterprozesses *System entwerfen*. Die Aktivitäten *Systemarchitektur erstellen*, *Unterstützungssystemarchitektur erstellen*, *Styleguide für die Mensch-Maschine-Schnittstelle erstellen*, *HW-Architektur erstellen*, *SW-Architektur erstellen*, *Datenbankentwurf erstellen*, *Implementierungs-, Integrations- und Prüfkonzept Unterstützungssystem erstellen*, *Implementierungs-, Integrations- und Prüfkonzept Hardware (HW) erstellen*, *Implementierungs-, Integrations- und Prüfkonzept Software (SW) erstellen* und *Migrationskonzept erstellen* werden hier nacheinander ausgeführt.

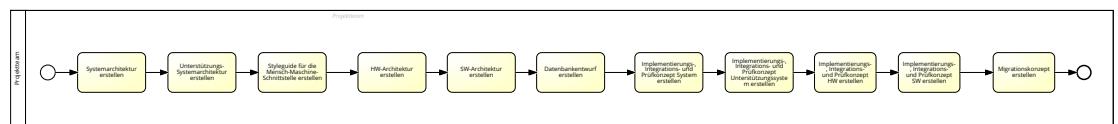


Abbildung 5.53.: System entwerfen V-Modell - imperativ

System spezifizieren

Am Anfang des Unterprozesses *System spezifizieren* (Abbildung 5.54) muss eine *Besprechung durchgeführt* werden. Hieraus entsteht das Artefakt *Besprechungsdokument*.

Parallel zu allen anderen Aktivitäten ist bei Bedarf bei jeder Änderung das *Projekttagbuch zu führen*. Dies wird durch das Parallel-Gateway sichergestellt und die XOR-Schleife stellt sicher, dass die Aktivität so oft durchgeführt wird, wie Anpassungen notwendig sind.

Im nächsten Schritt werden die *Messdaten erfasst*.

In der nachfolgenden Aktivität wird die *Metrik berechnet und ausgewertet*, woraus das Artefakt *Metrikauswertung* entsteht.

Anschließend erfolgt die Durchführung der Aktivität *Kaufmännischen Projektstatusbericht erstellen*, wobei das Artefakt *Kaufmännischer Projektstatusbericht* als Artefakt herauskommt.

Bei der nächsten Aktivität wird der *Projektstatusbericht erstellt* und danach wird der Ge-

samtprojektfortschritt ermittelt

Die Aktivität *QS-Bericht erstellen* bringt dann das Artefakt *QS-Bericht* hervor und die nachfolgende Aktivität *Projekt abschließen* den *Projektab schlußbericht*.

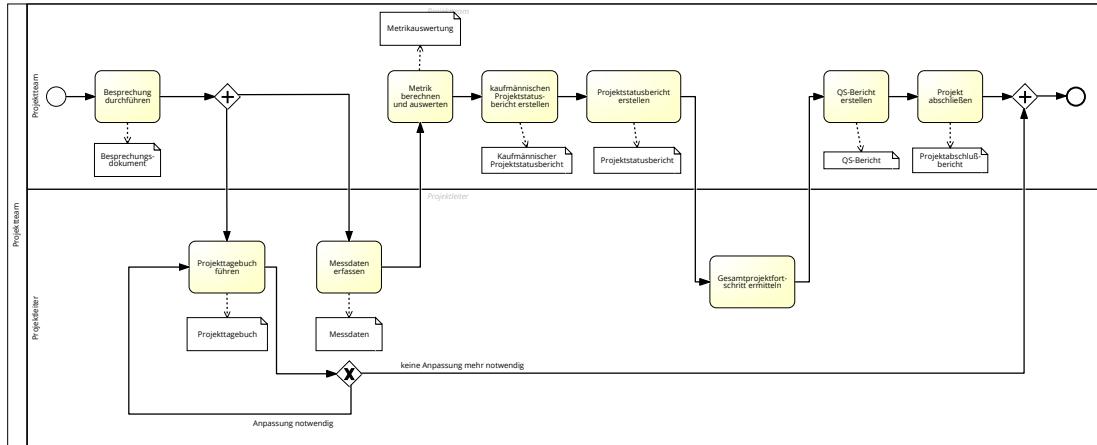


Abbildung 5.54.: System spezifizieren-imperativ
Anzahl unterschiedliche Gateways/Constraints

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

5.3.3. Deklarative Modellierung V-Modell

Die deklarative Modellierung von *Systementwicklungsprojekt AG/AN* zeigt Abbildung 5.55.

Zunächst muss ein Projekt genehmigt und definiert werden. Dies ist durch die aufeinander folgenden Aktivitäten *Projekt genehmigen* und *Projekt definieren* dargestellt, welche durch das Constraint *succession* verbunden sind.

In der nachfolgenden Aktivität müssen sodann die *Anforderungen festgelegt werden*, bevor die *Iteration geplant* werden kann.

Hiernach muss entschieden werden, ob eine *Prototypische Entwicklung durchgeführt*, eine *Komponentenbasierte Entwicklung durchgeführt* oder eine *Inkrementelle Entwicklung durchgeführt* werden soll. Dies wird hier durch einen extra Unterprozess *Entwicklung durchführen* (Abbildung 5.56) dargestellt, da dieser Sachverhalt auf Grund der Schleifen im Prozess ohne Unterprozess nicht darstellbar war. Im Unterprozess kann dann eine der drei Aktivitäten ausgeführt werden, was das Constraint *1of 3* vorgibt.

Anschließend wird das *System abgenommen*.

An dieser Stelle wird entschieden, ob erneut zu *Anforderungen festlegen* zurückgekehrt wird und der Prozess ab dieser Aktivität erneut startet oder ob zu *Projekt ausschreiben* zurückgekehrt wird und der Prozess ab hier erneut startet. Ansonsten endet der Prozess mit der Aktivität *Projekt abschließen*.

Inkrementelle Entwicklung durchführen

In Abbildung 5.57 ist die deklarative Modellierung des Unterprozesses *Inkrementelle Entwicklung durchführen* abgebildet.

Zu Beginn muss das *System spezifiziert* werden, weshalb diese Aktivität mit dem Constraint *init* beschrieben ist und anschließend wird das *System entworfen*, was durch das Constraint *succession* zwischen diesen beiden Aktivitäten sichergestellt wird.

Hiernach wird der Feinentwurf entworfen und Systemelemente realisiert. Diese beiden Aktivitäten können so oft wie nötig durchgeführt werden. Es kommt nur darauf an, dass zuerst die Aktivität *Feinentwurf entwerfen* und anschließend die Aktivität *Systemelemente realisieren* durchgeführt wird, weshalb das Constraint *chain succession* sich zwischen

5.3. V-Modell XT

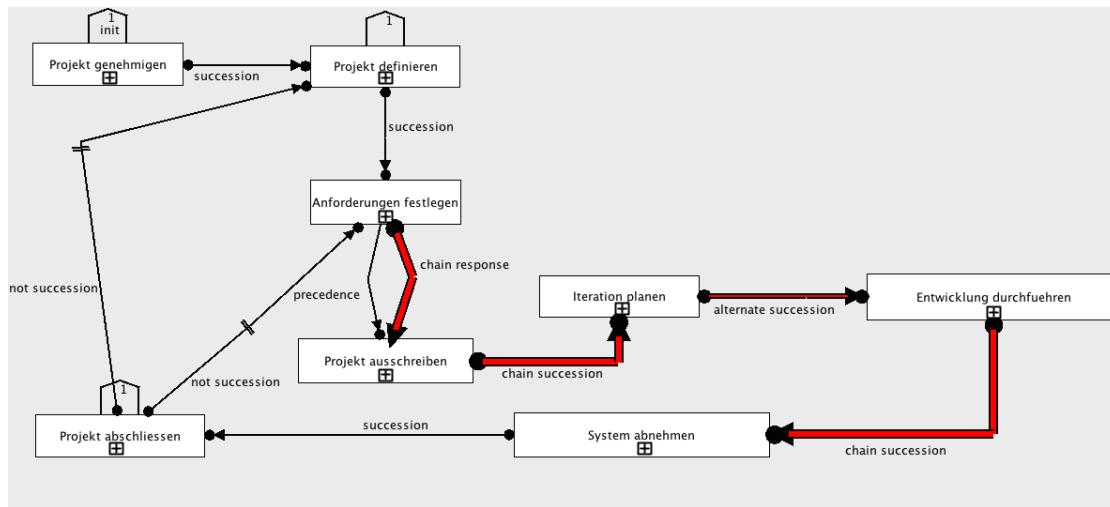


Abbildung 5.55.: Systementwicklungsprojekt AG/AN V-Modell - deklarativ

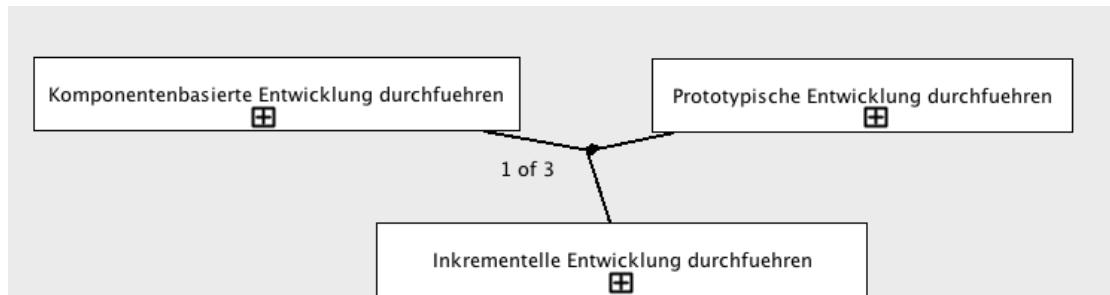


Abbildung 5.56.: Systementwicklungsprojekt AG/AN V-Modell Unterprozess Entwicklung durchfuehren - deklarativ

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

diesen beiden Aktivitäten befindet.

Im nächsten Schritt wird das System integriert und es beginnt eine neue Iteration bei der Aktivität *System entwerfen*. Aus diesem Grund befindet sich hier das Constraint *alternate succession* zwischen den Aktivitäten *System integrieren* und *System entwerfen*, da eine erneute Ausführung von *System entwerfen* erst nach Ausführung der Aktivität *System integrieren* möglich ist.

Falls keine weitere Iteration mehr notwendig ist, wird die *Lieferung durchgeführt* und der Unterprozess endet hier, da durch die Constraints zu *System entwerfen* und *Feinentwurf entwerfen* keine weitere Aktivität mehr ausgeführt werden kann.

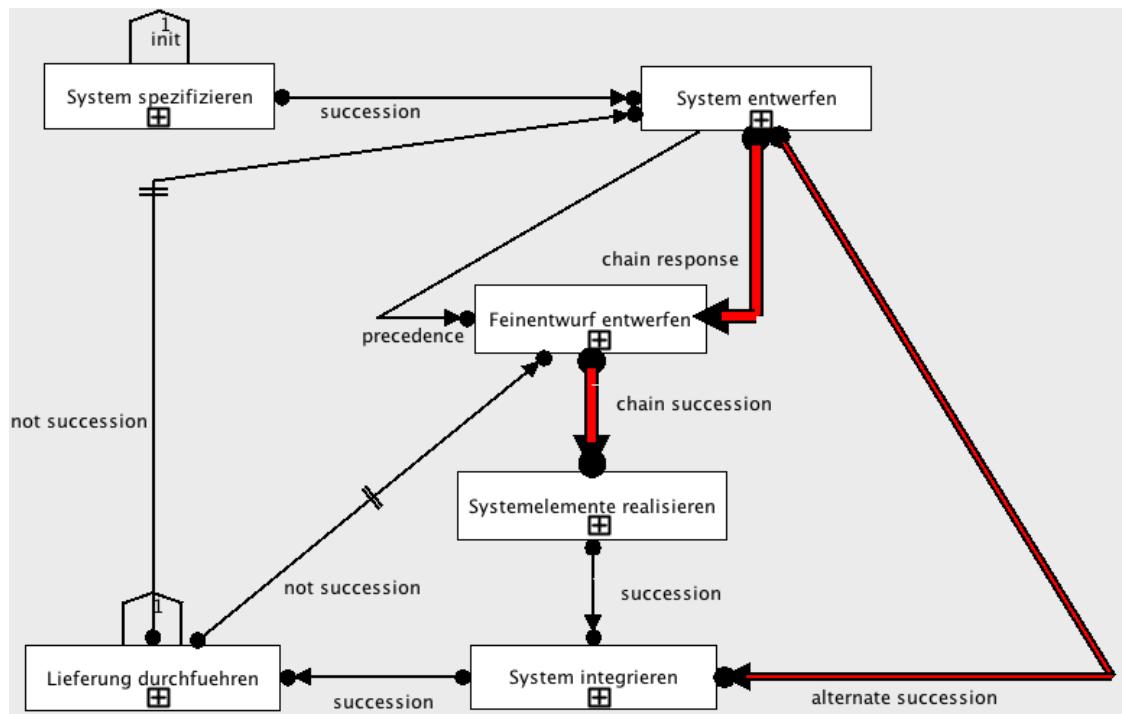


Abbildung 5.57.: Unterprozess Inkrementelle Entwicklung durchführen V-Modell - imperativ

System entwerfen

Abbildung 5.58 zeigt die deklarative Modellierung von System entwerfen.

Die Aktivitäten *Systemarchitektur erstellen*, *Unterstützungssystemarchitektur erstellen*, *Styleguide für die Mensch-Maschine-Schnittstelle erstellen*, *HW-Architektur erstellen*, *SW-Architektur erstellen*, *Datenbankentwurf erstellen*, *Implementierungs-, Integrations- und Prüfkonzept Unterstützungssystem erstellen*, *Integrations- und Prüfkonzept Hardware (HW) erstellen*, *Integrations- und Prüfkonzept Software (SW) erstellen* und *Migrationskonzept erstellen* werden hier nacheinander ausgeführt. Da die vorherige Aktivität immer Voraussetzung für das Ausführen der nachfolgenden Aktivität ist, und die nachfolgende Aktivität nach der vorherigen ausgeführt werden muss, ist zwischen allen Aktivitäten jeweils *succession* als Constraint eingefügt.

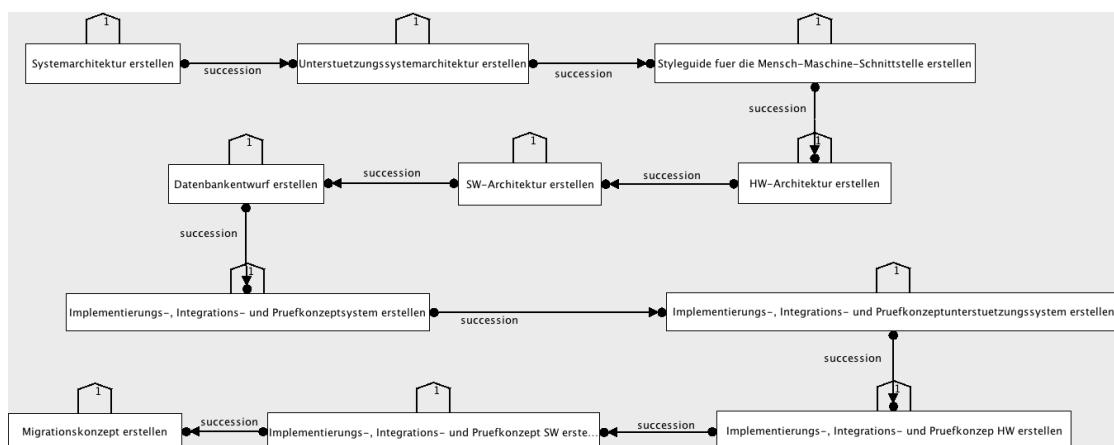


Abbildung 5.58.: System entwerfen - deklarativ

System spezifizieren

Am Anfang von System spezifizieren (Abbildung 5.59) muss eine *Besprechung durchgeführt* werden. Dies wird durch das Constraint *init* sichergestellt.

Bei jeder Änderung muss die Aktivität *Projekttagbuch führen* ausgeführt werden. Aus diesem Grund ist diese durch kein Constraint mit einer anderen Aktivität verbunden, da sie jederzeit ausgeführt werden kann und so oft wie nötig.

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

Im nächsten Schritt werden die *Messdaten erfasst*. Da ab hier alle Aktivitäten nacheinander auszuführen sind, sind dies jeweils durch das Constraint *succession* verbunden. In der nachfolgenden Aktivität wird die *Metrik berechnet und ausgewertet*.

Anschließend erfolgt die Durchführung der Aktivität *Kaufmännischen Projektstatusbericht erstellen*.

Bei der nächsten Aktivität wird der *Projektstatusbericht erstellt* und danach wird der *Gesamtprojektfortschritt ermittelt*.

Danach werden noch die Aktivitäten *QS-Bericht erstellen* und *Projekt abschließen* ausgeführt.

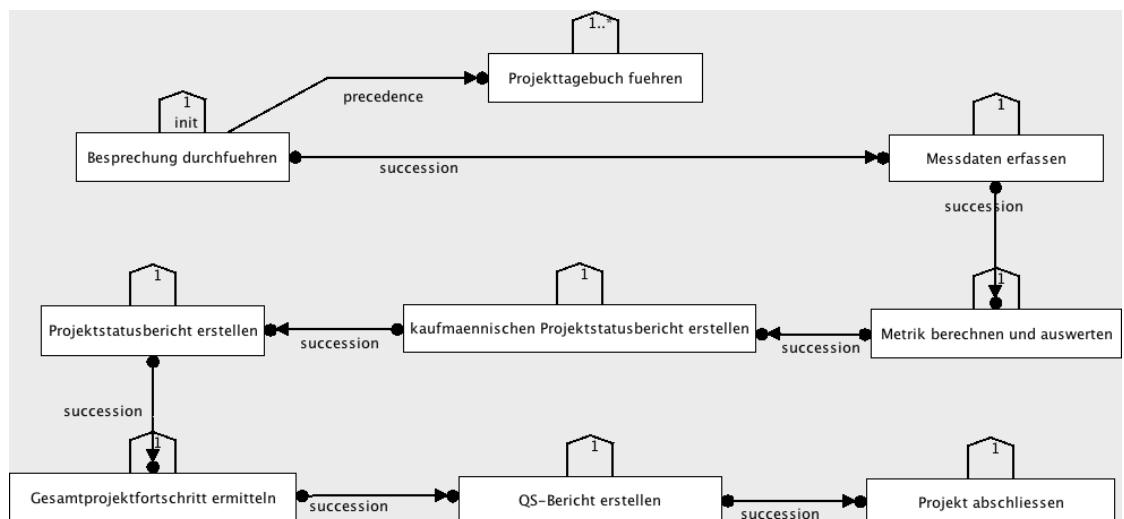


Abbildung 5.59.: System spezifizieren- deklarativ

5.3.4. Vergleich

Abbildung 5.60 zeigt die Zahl notwendigen Elemente zur Darstellung des Prozesses Systementwicklungsprojekt AG/AN. In ConDec (11 Aktivitäten) ist somit einer Aktivität mehr zur Darstellung nötig als in BPMN (10 Aktivitäten). In BPMN werden vier Gateways und 20 Sequenzflusselemente verwendet. In ConDec hingegen braucht es 11 Constraints. In ConDec werden sieben unterschiedliche Constraints im Modell verwendet in BPMN eines. Somit sind zur Darstellung des Sachverhaltes insgesamt 34 BPMN Elemente 25 ConDec Elemente notwendig.

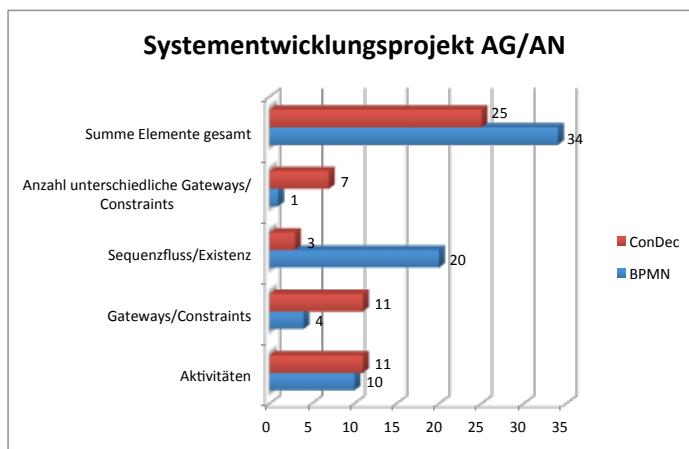


Abbildung 5.60.: Systementwicklungsprojekt AG/AN

Die Anzahl der Elemente zur Darstellung des Prozesses Inkrementelle Entwicklung kann Abbildung 5.61 entnommen werden. Es werden jeweils sechs Aktivitäten verwendet. Weiterhin werden in BPMN vier Gateways und 13 Sequenzflüsse zur Darstellung des Prozessablaufes benötigt. In ConDec sind hierfür neun Constraints und zwei Existenz-Constraints notwendig. BPMN benötigt ein Gateway, also keine verschiedenen Gateways und ConDec braucht sechs unterschiedliche Constraints. Somit ergeben sich in Summe 23 BPMN Elemente und 17 ConDec Elemente zur Darstellung des Sachverhaltes.

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

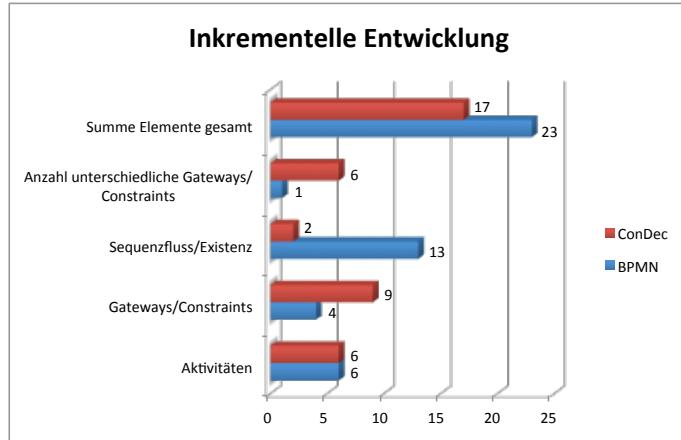


Abbildung 5.61.: Inkrementelle Entwicklung

Abbildung 5.62 zeigt die Anzahl der Elemente, welche zur Darstellung des Prozesses System entwerfen notwendig sind. Demnach werden in beiden Prozessmodellierungs-sprachen jeweils 11 Aktivitäten verwendet. In BPMN werden keine Gateways und 12 Sequenzflusselemente benötigt. ConDec braucht zur Darstellung des Sachverhaltes 10 Constraints und 11 Existenz-Constraints jedoch keine unterschiedlichen Constraints. In Summe sind somit in BPMN 23 Elemente und in ConDec 32 Elemente zur Darstellung notwendig.

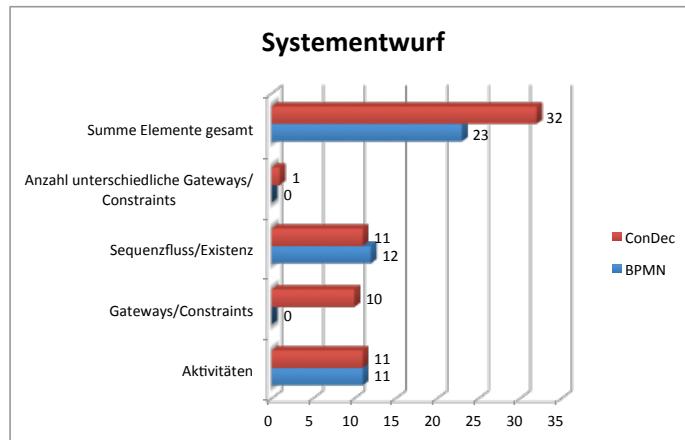


Abbildung 5.62.: System entwerfen

In Abbildung 5.63 ist die Anzahl der Elemente zur Darstellung des Prozesses System spezifizieren abgetragen. Demnach werden neun Aktivitäten verwendet sowohl in BPMN als auch in ConDec. Es werden drei Gateways und 15 Sequenzflüsse in BPMN benötigt. In ConDec werden acht Constraints und neun Existenz-Constraints zur Darstellung verwendet. Weiterhin gibt es zwei unterschiedliche Gateways in BPMN und zwei unterschiedliche Constraints in ConDec. Insgesamt ergeben sich somit 27 unterschiedliche Elemente in BPMN und 26 unterschiedliche Elemente in ConDec.

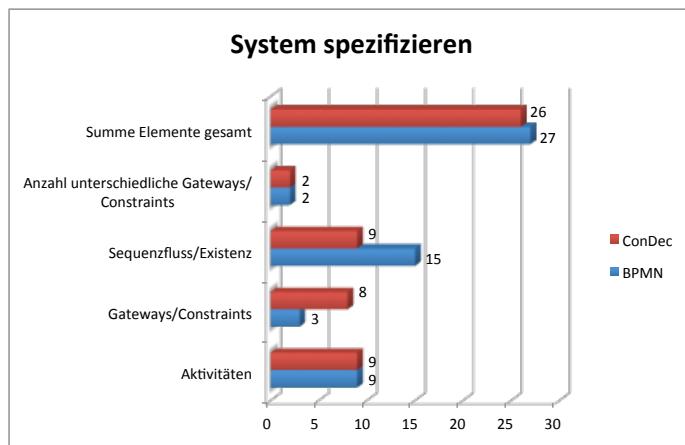


Abbildung 5.63.: System spezifizieren

Sowohl BPMN, als auch ConDec können die syntaktische *Richtigkeit* jedoch einhalten, da in beiden Modellierungssprachen die Notationsregeln bei der Modellierung der Prozessmodelle eingehalten werden können.

Die semantische *Richtigkeit* kann von ConDec in Bezug auf die Darstellung von Rollen und Artefakten nicht eingehalten werden. Hier werden Grenzen der Darstellbarkeit in ConDec erreicht.

Der Grundsatz des *systematischen Aufbaus* kann bei ConDec wiederum nicht eingehalten werden, da keine Artefakte visualisiert werden können.

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

Der Grundsatz der *Relevanz* kann wieder nur von BPMN eingehalten werden, da es bei den ConDec-Modellen keine Visualisierungsmöglichkeiten für Rollen und Artefakte gibt.

Beim Grundsatz der *Klarheit* gibt es Unterschiede zwischen den Modellen. Zur Darstellung der beiden Prozesse Systementwicklungsprojekt AG/AN und Inkrementelle Entwicklung werden in ConDec jeweils viele unterschiedliche Constraints benötigt (acht bei Systementwicklungsprojekt AG/AN und sechs bei Inkrementelle Entwicklung). Dies wirkt sich sehr negativ auf die Verständlichkeit aus, da sechs bis acht unterschiedliche, teilweise sehr komplexe Constraints verstanden werden müssen. Auch wenn BPMN bei diesen Modellen eine höhere Anzahl an Elementen insgesamt aufweist, weisen die beiden ConDec-Modelle eine höhere Komplexität auf, weswegen die BPMN-Modelle verständlicher sind.

Beim Prozess Systementwurf hingegen weist ConDec insgesamt eine höhere Anzahl an Elementen auf. Weiterhin werden hier bei ConDec 10 Constraints verwendet, in BPMN hingegen keine Gateways. Aus diesem Grund ist hier das mit BPMN erstellte Modell das weniger komplexe und somit auch das verständlichere.

Beim Prozess System spezifizieren werden in BPMN drei Gateways verwendet und in ConDec acht Constraints. Bei beiden handelt es sich hierbei um zwei unterschiedliche Gateways/Constraints. Da ConDec hier eine höhere Anzahl an Constraints aufweist, als BPMN Gateways, ist das mit BPMN erstellte Modell hier das verständlichere.

Somit lässt sich insgesamt hier festhalten, das beim Grundsatz der *Klarheit* BPMN hier die geeigneteren Modellierungssprache ist, da sie hier weniger komplexe Modelle erzeugt.

Die Anzahl der Elemente insgesamt unterscheidet sich bis auf das Modell Systementwicklungsprojekt AG/AN nicht stark voneinander. Die BPMN-Modelle weisen eine höhere Anzahl an Sequenzflusselementen auf, als die ConDec Modelle Existenz- Constraints aufweisen. Der Grundsatz der *Wirtschaftlichkeit* kann hier von BPMN besser eingehalten werden, da die ConDec Modelle deutlich mehr Constraints und auch verschiedene Constraints enthalten und hier somit ein größerer geistiger Aufwand für die Erstellung der Modelle aufgebracht werden muss.

5.3. V-Modell XT

Bei Ausführung der Prozesse in den Modellierungstools Signavio und Declare weisen beide das gleiche Ausführungsverhalten auf, wodurch hier die *Vergleichbarkeit* gewährleistet ist.

Die Anzahl der Elemente in den Prozessmodellen unterscheidet sich teilweise. Bei BPMN werden bei den Prozessen Systementwicklungsprojekt AG/AN deutlich mehr Elemente zur Darstellung des gleichen Sachverhaltes verwendet wie bei ConDec. Beim Prozess Systementwurf werden bei ConDec mehr Elemente benötigt wie bei BPMN und bei System spezifizieren werden in etwa gleich viele Elemente verwendet.

Die *Vergleichbarkeit* kann jedoch in Bezug auf Artefakte und Rollen, außer bei den Phasen des Open UP (hier werden noch keine Artefakte und Rollen verwendet) nicht eingehalten werden.

Somit weisen bei der *Vergleichbarkeit* beide Prozessmodellierungssprachen Stärken und Schwächen auf.

Abbildung 5.64 gibt nochmal eine zusammenfassende Übersicht über die Ergebnisse des Vergleichs.

Modellierungsgrundsatz		Geeignete Modellierungssprache
Richtigkeit	semantisch	BPMN
	syntaktisch	BPMN, ConDec
Systematischer Aufbau		BPMN
Relevanz		BPMN
Klarheit		BPMN
Wirtschaftlichkeit		BPMN
Vergleichbarkeit		BPMN, ConDec

Abbildung 5.64.: Übersicht Vergleich V-Modell

5.4. Vergleich insgesamt

BPMN und ConDec können die syntaktische *Richtigkeit* bei allen Modellen einhalten, da in beiden Modellierungssprachen die Notationsregeln bei der Modellierung der Metamodelle eingehalten werden können, um das dort beschriebene Verhalten korrekt wieder zu geben. Somit sind in bezug auf die syntaktische *Richtigkeit* beide Prozessmodellierungssprachen gleich geeignet.

Die semantische *Richtigkeit* kann von ConDec in Bezug auf die visuelle Darstellung von Rollen und Artefakten im Prozessmodell selbst bei keinem Modell eingehalten werden, bei welchem solche Informationen notwendig wären. Die Grenzen der Darstellbarkeit von ConDec werden hier erreicht. Da Rollen und Artefakte jedoch für das insgesamte Verständnis eines Prozessmodells sehr wichtige Elemente sind, ist BPMN in Hinsicht auf die semantische *Richtigkeit* die geeignetere Sprache zum Modellieren.

Der Grundsatz des *systematischen Aufbaus* kann bei ConDec bei keinem Prozessmodell eingehalten werden, da nirgendwo Artefakte visualisiert werden können. Da Artefakte jedoch für das Verständnis eines Modelles und die Darstellung von Schnittstellen sehr wichtige Elemente sind, ist BPMN in Hinsicht auf den *systematischen Aufbau* die geeignetere Prozessmodellierungssprache.

Die *Relevanz* kann ebenfalls nur von BPMN eingehalten werden, da sich die ConDec-Modelle nicht mit den minimal relevanten Informationen erstellen lassen. Auch hier ist das Problem wiederum die fehlende Visualisierungsmöglichkeit von Rollen und Artefakten. Somit ist auch hier BPMN die geeignetere Prozessmodellierungssprache.

Beim Grundsatz der *Klarheit* gibt es Unterschiede bei der Eignung zwischen den beiden Prozessmodellierungssprachen abhängig vom abzubildenden Verhalten des Metamodells. Hier weisen beide Modelle Stärken und Schwächen auf.

Bei der Darstellung von Metamodellen, bei denen viele Aktivitäten parallel ablaufen, wie z.B. bei den Phasen des Open UP benötigt BPMN deutlich mehr Elemente zur Darstellung.

5.4. Vergleich insgesamt

lung, als ConDec. Auch sind in BPMN Gateways zur Darstellung notwendig, während bei ConDec lediglich leichter verständliche Existenz-Constraints eingesetzt werden. Dies ist sowohl der Fall bei den kleineren Modellen (< fünf Aktivitäten), als auch bei den größeren Modellen (\geq fünf Aktivitäten). Somit sind hier die BPMN Modelle komplexer als die ConDec Modelle und ConDec ist hierfür die geeigneter Prozessmodellierungssprache. Ein anderes Bild zeichnet sich bei Prozessmodellen ab, bei welchen viele Verzweigungen oder aber auch Abläufe ohne Verzweigungen und Parallelität, also gerade Abläufe dargestellt werden müssen. Hier sind zwar bei BPMN oftmals auch mehr Elemente insgesamt zur Darstellung notwendig, jedoch werden bei BPMN bei der Darstellung von Abläufen ohne Verzweigung keine Gateways und bei der Darstellung von Abläufen mit vielen Verzweigungen auch maximal zwei verschiedene Gateways benötigt. ConDec braucht zur Darstellung von Abläufen ohne Verzweigung und Parallelität trotzdem Constraints, wenn auch wenig unterschiedliche Constraints. Zur Darstellung von Abläufen mit vielen Verzweigungen/Schleifen und auch eventuell noch gleichzeitig parallelen Abläufen werden bei ConDec viele unterschiedliche Constraints eingesetzt. Bei kleinen Modellen (< fünf Aktivitäten), wie z.B. beim Modell Phasen des Open UP ist ConDec das weniger komplexe Modell. Beim Prozess Lösungskrement entwickeln ist nur ein kleiner Unterschied in der Komplexität vorhanden. Bei größeren Modellen (\geq fünf Aktivitäten) sind jedoch die ConDec Modelle deutlich komplexer. Dies lässt sich bei den Modellen Scrum sowie bei den Modellen des V-Modell Systementwicklungsprojekt AG/AN und Inkrementelle Entwicklung durchführen erkennen. Hier müssen bei den ConDec Modellen vier bis sieben unterschiedliche Constraints zur Darstellung des Ablaufes verwendet werden. In BPMN hingegen werden bei diesen Modellen maximal zwei unterschiedliche Gateways verwendet. Daher ist hier bei Modellen mit mehr als fünf Aktivitäten und vielen Verzweigungen und eventuell noch parallelen Aktivitäten BPMN die geeigneter Modellierungssprache. Hingegen bei Prozessen mit vielen parallelen Aktivitäten ist ConDec besser zum Modellieren geeignet.

Bei Betracht der *Wirtschaftlichkeit* weisen beide Modellierungssprachen Stärken und Schwächen auf. Bei Modellen mit vielen parallelen Abläufen, wie z.B. den Phasen des Open UP müssen in BPMN mehr als doppelt so viele Elemente verwendet werden, wie

5. Imperative und deklarative Modellierung für Softwareentwicklungsprozesse

bei ConDec, wodurch hier ein höherer Aufwand bei der Modellierung mit BPMN entsteht, als bei ConDec.

Bei großen Modellen (\geq fünf Aktivitäten) mit vielen Verzweigungen/Schleifen und noch dazu parallelen Aktivitäten werden bei ConDec sehr viel mehr Constraints bei der Modellierung benötigt, als Gateways bei BPMN. Aus diesem Grund ist bei diesen Modellen bei der Modellierung mit ConDec ein größerer geistiger Aufwand notwendig, als bei BPMN und daher ist bei Modellen mit vielen Verzweigungen/Schleifen BPMN die geeigneter Modellierungssprache, wenn die *Wirtschaftlichkeit* betrachtet wird.

Bei allen Prozessen lässt sich durch Ausführung in den Modellierungstools Signavio und Declare das gleiche Ausführungsverhalten beobachten, wodurch hier die *Vergleichbarkeit* gewährleistet ist.

Die Anzahl der Elemente in den Prozessmodellen unterscheidet sich teilweise. Bei BPMN werden bei vielen Prozessen deutlich mehr Elemente zur Darstellung des gleichen Sachverhaltes verwendet wie bei ConDec.

Die *Vergleichbarkeit* kann jedoch in Bezug auf Artefakte und Rollen, außer bei den Phasen des Open UP (hier werden noch keine Artefakte und Rollen verwendet) nicht eingehalten werden.

Somit weisen bei der *Vergleichbarkeit* beide Prozessmodellierungssprachen Stärken und Schwächen auf.

Abbildung 5.65 gibt nochmal eine zusammenfassende Übersicht über die Ergebnisse des Vergleichs.

5.4. Vergleich insgesamt

Modellierungsgrundsatz		Geeignete Modellierungssprache
Richtigkeit	semantisch	BPMN
	syntaktisch	BPMN, ConDec
Systematischer Aufbau		BPMN
Relevanz		BPMN
Klarheit		BPMN (bei großen Modellen (mit >=fünf Aktivitäten) mit vielen Verzweigungen/Schleifen oder Prozesse mit geraden Abläufen), ConDec (bei Prozessen mit vielen parallelen Aktivitäten)
Wirtschaftlichkeit		BPMN (bei großen Modellen (mit >=fünf Aktivitäten) mit vielen Verzweigungen/Schleifen oder Prozesse mit geraden Abläufen), ConDec (bei Prozessen mit vielen parallelen Aktivitäten)
Vergleichbarkeit		BPMN, ConDec

Abbildung 5.65.: Übersicht Vergleich Allgemein

6

Validierung

In diesem Kapitel werden die Ergebnisse des Vergleichs der ConDec - und BPMN-Modelle aus Kapitel 5 mit Hilfe einer Studie validiert. Im Rahmen der Studie ist es nur möglich die Grundsätze der *Klarheit* und der *Vergleichbarkeit* zu testen.

6.1. Forschungsfragen

Forschungsfrage 1a:

Ist die Punktsumme bei den Verständnisfragen insgesamt bei BPMN oder bei ConDec höher?

Forschungsfrage 1b:

6. Validierung

Ist die Punktsumme bei den Verständnisfragen bei den kleinen Modellen bei BPMN oder bei ConDec höher?

Forschungsfrage 1c:

Ist die Punktsumme bei den Verständnisfragen bei den großen Modellen bei BPMN oder bei ConDec höher?

Forschungsfrage 1d:

Gibt es Unterschiede im Ergebnis in Abhängigkeit des Hintergrundwissens der Versuchsobjekte über Prozessmodellierung?

Forschungsfrage 2a:

Werden bei den Meinungsfragen insgesamt die Modelle von BPMN oder von ConDec präferiert?

Forschungsfrage 2b:

Werden bei den Meinungsfragen bei den kleinen Modellen die von BPMN oder von ConDec präferiert?

Forschungsfrage 2c:

Werden bei den Meinungsfragen bei den großen Modellen die von BPMN oder von ConDec präferiert?

6.2. Design der Studie

Abbildung 6.1 kann die Struktur der Umfrage entnommen werden. Bei der Studie wurden zwei Fragebögen eingesetzt. Die 32 Teilnehmer der Studie wurden deswegen zufalls-

6.2. Design der Studie

bedingt in zwei Gruppen (je 16 Personen) eingeteilt. Zunächst wurden den Probanden allgemeine demographische Fragen gestellt (Geschlecht, Alter, Hintergrundwissen zu imperativer und deklarativer Modellierung, Hintergrundwissen zu den Software Engineering Prozessmodellen Scrum, Open UP und V-Modell XT).

Anschließend wurden den Teilnehmern Verständnisfragen zu ausgewählten Modellen gestellt. Es wurden die vier Modellpaare *Lösungssinkrement entwickeln* (Open UP), *Scrum*, *Systementwicklungsprojekt AG/AN* und *Phasen Open UP -Inception* aus Kapitel 5 ausgewählt. Hierbei wurde darauf geachtet, dass es sich um zwei kleine Modelle (≤ 5 Aktivitäten) und zwei große Modelle (> 5 Aktivitäten) handelt. Gruppe 1 startete mit einem deklarativen Prozess und Gruppe 2 mit dem entsprechenden imperativen Prozess. Somit wurden von jeder Gruppe zwei imperative und zwei deklarative Prozesse bearbeitet.

Im letzten Teil des Fragebogens wurden den Probanden noch vier Modellpaare direkt gegenüber gestellt und sie wurden nach Ihrem präferierten Modell (deklarativ oder imperativ) gefragt und mussten in einem Freitextfeld den Grund für Ihre Entscheidung angeben. Auch hier wurden den Teilnehmern wiederum zwei kleine (≤ 5 Aktivitäten) und zwei große (> 5 Aktivitäten) Modelle gezeigt. Hier wurden die Modelle *System spezifizieren* (V-Modell XT), *Phasen des Open UP*, *Inkrementelle Entwicklung durchführen* (V-Modell XT) und *Release deployen* ausgewählt.

Die semantische Gleichheit der Modellpaare wurde wie bereits in Kapitel 5 erwähnt durch das Testen von validen Pfaden durch die jeweiligen Modelle sichergestellt.

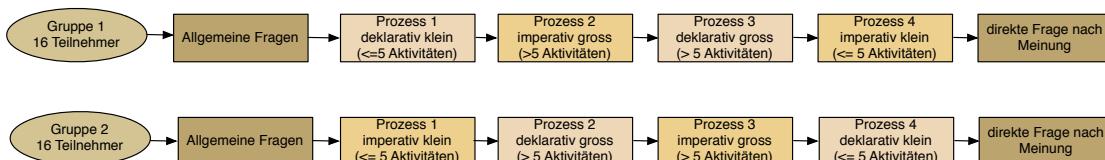


Abbildung 6.1.: Struktur der Umfrage

6. Validierung

Verständnisfragen

Zu jedem Modell wurden den Teilnehmern jeweils acht Verständnisfragen gestellt. Diese zielten auf das Verständnis der möglichen Reihenfolge der Aktivitäten, mögliche Start- und Endaktivitäten, allgemeine Informationen aus dem Modell sowie parallele Abläufe von Aktivitäten, sich ausschließende Aktivitäten und die Anzahl möglicher Ausführungen von Aktivitäten.

Die Teilnehmer konnten bei der Beantwortung der Fragen zwischen vier Antwortmöglichkeiten wählen: *Ja*, *Nein*, *Geht nicht aus Modell hervor* und *Unentschlossen*.

Umfragewerkzeug und Durchführung

Zur Durchführung wurde das Fragebogenwerkzeug *Limesurvey* verwendet. Der entsprechende Link zum Fragebogen sowie eine kleine Legende zur Notationsübersicht von Declare und BPMN wurde den Teilnehmern per E-Mail zugeschickt. Die entsprechenden Antworten der Probanden wurden automatisch von *Limesurvey* gespeichert. Weiterhin war es dort möglich die jeweilige Zeit, welche die Probanden zur Bearbeitung Verständnisfragen benötigt haben mit zu messen. Die gespeicherten Daten können aus *Limesurvey* für verschiedene externe Anwendungen exportiert werden (z.B als Excel, CSV oder für SPSS).

Auswertung

Für jede richtige Antwort wurde ein Punkt vergeben. Für jede falsche Antwort gab es null Punkte. Auch *Unentschlossen* wurde als falsche Antwort gewertet. Die einzelnen Punkte wurden dann pro Frage aufsummiert, so dass ein maximaler Wert pro Frage von 1 möglich ist.

6.2.1. Durchführung der Studie

Teilnehmer

Es wurden 32 Studenten und Doktoranden aus dem Bereich Informatik/Medieninformatik befragt. 12 Teilnehmer waren weiblich und 20 männlich (Abbildung 6.2). Die allgemeinen demographischen Daten der Probanden können Abbildung 6.4 entnommen werden. Diese hatten unterschiedliches Hintergrundwissen zum Thema Prozessmodellierung. Wie Abbildung 6.3 entnommen werden kann, hatten sieben Studienteilnehmer weder in imperativer noch in deklarativer Modellierung Erfahrung. 18 Probanden hatten nur in imperativer Modellierung Erfahrung, jedoch nicht in deklarativer und sieben weitere Teilnehmer hatten in beiden Modellierungssprachen Erfahrung. Die Versuchsobjekte wurden bewusst nach unterschiedlichem Hintergrundwissen zum Thema Prozessmodellierung ausgewählt um zu prüfen, inwiefern sich die Ergebnisse bei den Verständnisfragen zwischen Personen mit viel und wenig Hintergrundwissen zum Thema Prozessmodellierung unterscheiden.



Abbildung 6.2.: Geschlechterverteilung

6. Validierung

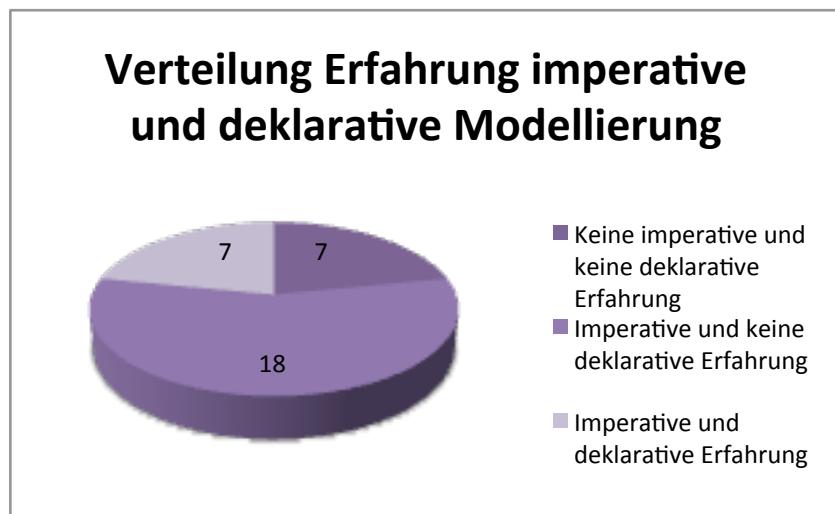


Abbildung 6.3.: Verteilung Erfahrung imperative und deklarative Modellierung

Allgemeine demographische Daten	Minimum	Maximum	Durchschnitt
Alter	23	37	27,325
Selbst eingeschätzte Erfahrung imperative Modellierung (1-5)	1	5	3,15
Jahre Erfahrung in imperativer Modellierung	0	10	3,175
Anzahl modellierte imperative Modelle im letzten Jahr	0	200	13,65
Selbst eingeschätzte Erfahrung deklarative Modellierung (1-5)	1	5	1,85
Jahre Erfahrung in deklarativer Modellierung	0	6	1,1
Anzahl modellierte deklarative Modelle im letzten Jahr	0	10	1,175
Selbst eingeschätztes Wissen Scrum	1	4	2,6
Selbst eingeschätztes Wissen Open UP	1	5	1,55
Selbst eingeschätztes Wissen V-Modell XT	1	4	2,175

Abbildung 6.4.: Allgemeine demographische Daten

Ergebnisse Verständnisfragen

Abbildung 6.5 zeigt die Ergebnisse der Verständnisfragen zum Modell *Open UP: Lösungssinkrement entwickeln*. Die Ergebnisse variieren hier zwischen den deklarativen und imperativen Modellen. Während die Ergebnisse teilweise gleich sind, bzw. nur wenig voneinander abweichen, liegen die Ergebnisse der deklarativen Modelle bei den Fragen 5 und 6 deutlich unter den Ergebnissen der imperativen Modelle.

Frage 5 lautete: *Nach Ausführung der Aktivität „Integrieren“ endet der Prozess in jedem Fall sofort*. Hier wurde von den Teilnehmern die imperative XOR-Verknüpfung besser verstanden, als die deklarative Darstellung des Ablaufes. Auch bei Frage 6 (*Als erste Aktivität im Prozess kann die Aktivität „Entwickeltest implementieren“ ausgeführt werden*) war den Probanden die imperative XOR-Darstellung, wohl in Verbindung mit dem BPMN Startsymbol als klaren Einstiegspunkt klarer, als die entsprechende deklarative Darstellung.

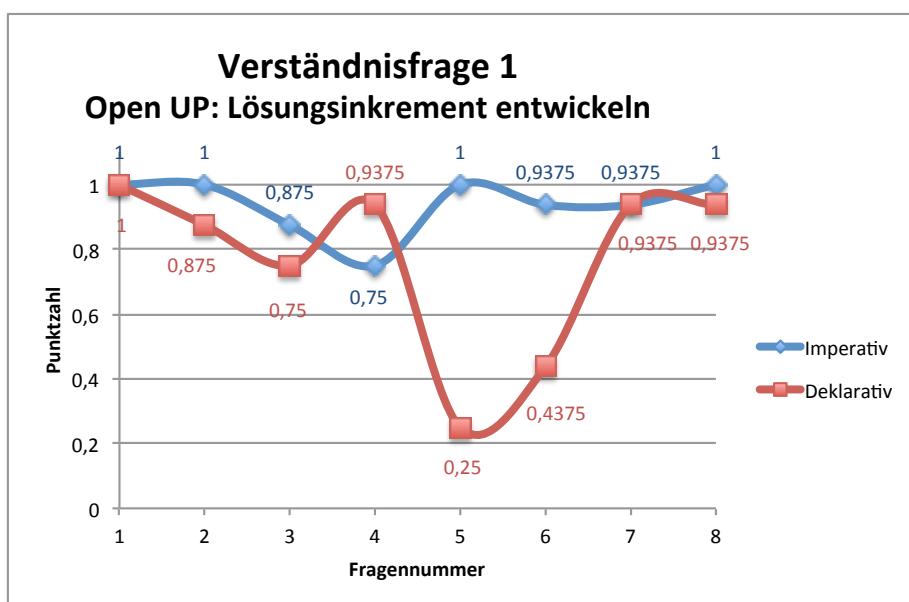


Abbildung 6.5.: Ergebnisse Verständnisfrage 1 aller Teilnehmer

6. Validierung

Die Ergebnisse der Verständnisfrage 2 zum Modell *Scrum* von allen Teilnehmern kann Abbildung 6.6 entnommen werden. Bei Scrum handelte es sich um ein großes Modell (>5 Aktivitäten), welches sowohl viele Verzweigungen, als auch viele parallele Aktivitäten aufweist. Auch hier weichen die Ergebnisse zwischen den imperativen und deklarativen Modellen voneinander ab.

Nur bei der ersten Frage (*Ein Scrum Meeting dauert 15 Minuten*) schneidet der deklarative Prozess besser ab, als der imperativen. Diese allgemeine Information aus dem Prozess befand sich bei beiden Prozessen in der Beschriftung der Aufgabe *15-minütiges Scrum Meeting durchführen*. Der imperative Scrum-Prozess weist insgesamt mehr Elemente auf, als der deklarative. Daher fiel es wohl den Probanden hier einfacher die Übersicht über allgemeine Informationen zu behalten.

Bei den Fragen 2 und 8 hat das deklarative Modell eine sehr schlechte Punktzahl erreicht. Bei Frage 2 (*Die Aktivität "Task abarbeiten" kann beliebig ausgeführt werden*) konnten die Teilnehmer der XOR-Verknüpfung im imperativen Modell welche eine Rückschleife auf die Aktivität *Task abarbeiten* mehr folgen, als der Darstellung der Aktivität im deklarativen Modell.

Das gleiche gilt für Frage 8 (*Nach Beendigung der Aufgabe "Task abarbeiten" endet der Prozess sofort*). Auch hier wurde die Verzweigung und das damit mögliche Zurückkehren zur Aufgabe *Sprint-Planning-Meeting durchführen* durch eine XOR-Verknüpfung im imperativen Modell dargestellt und war somit für die Teilnehmer klarer verständlich.

Beim Modell *V-Modell: Systementwicklungsprojekt AG/AN* der Verständnisfrage 3 lagen die Ergebnisse des deklarativen Modell bis auf Frage 3 immer unter denen des imperativen Modells. Starke Abweichung gab es bei den Fragen 2, 4, 5, 7.

Bei Frage 2 (*Die Aktivitäten "Prototypische Entwicklung durchführen", "Komponentenbasierte Entwicklung durchführen" und "Inkrementelle Entwicklung durchführen" können parallel zueinander ausgeführt werden*) war den Teilnehmern, welche das deklarative Modell bearbeiten mussten, die Notation des Constraints *Exclusive Choice 1 of 3* nicht ganz klar. Sechs der 16 Probanden kreuzten hier entweder *Ja* oder *Unentschlossen* an.

Sowohl Frage 4 (*Nach Ausführung der Aktivität "System abnehmen" kann die Aktivität*

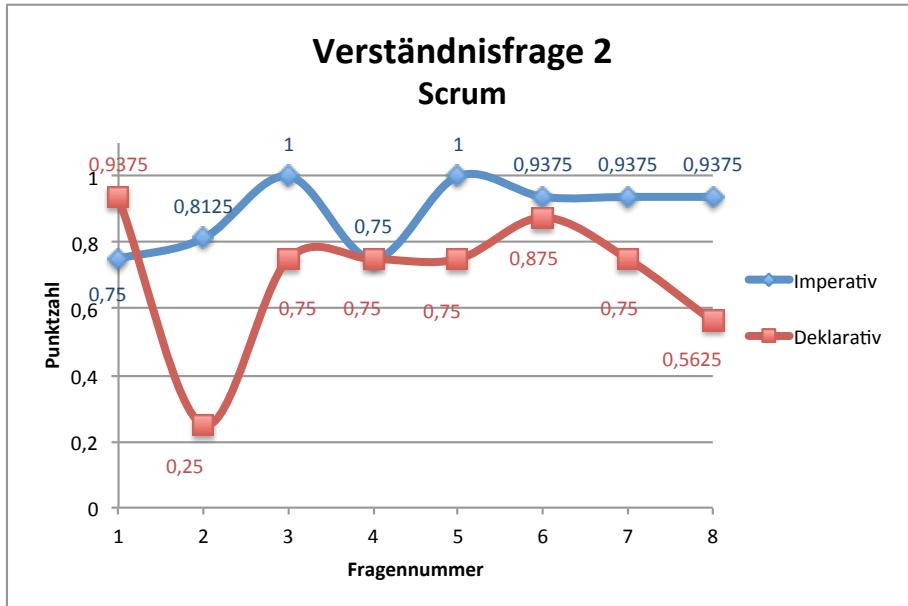


Abbildung 6.6.: Ergebnisse Verständnisfrage 2 aller Teilnehmer

“Anforderungen festlegen“ ausgeführt werden, als auch Frage 5 (Nach Ausführung der Aktivität “System abnehmen“ kann die Aktivität “Projekt ausschreiben“ ausgeführt werden) zielten wieder auf Verzweigungen des Prozesses ab und waren den Teilnehmern mit dem imperativen Prozess klarer.

Frage 7 (Nach Ausführung der Aktivität “Projekt abschließen“ endet der Prozess) wurde von den Probanden, welchen das imperative Modell gezeigt wurde richtiger beantwortet. Hier war im imperativen Modell durch das BPMN-Ende-Symbol den Teilnehmern das Ende des Prozesses wohl klarer als die Darstellung durch das Constraint *not succession* im deklarativen Modell.

Die Ergebnisse des Prozesses *Open UP: Inception* weichen zwischen den deklarativen und imperativen Modellen nicht stark voneinander ab.

Lediglich bei Frage 6 (Die Aktivität “Iteration planen und managen“ kann beliebig oft ausgeführt werden) und Frage 7 (Die Aktivitäten “Anforderungen identifizieren und verfeinern“ und “auf technisches Vorgehen einigen“ können beliebig oft ausgeführt werden) war den Teilnehmern wohl teilweise die Funktion des Existenz (1) Constraints nicht ganz

6. Validierung

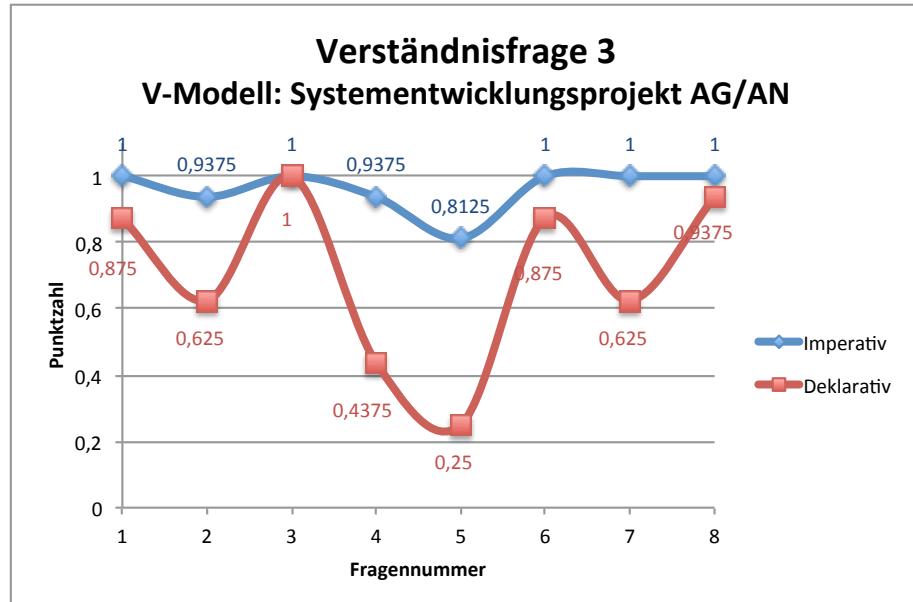


Abbildung 6.7.: Ergebnisse Verständnisfrage 3 aller Teilnehmer

klar oder wurde übersehen.

6.2. Design der Studie

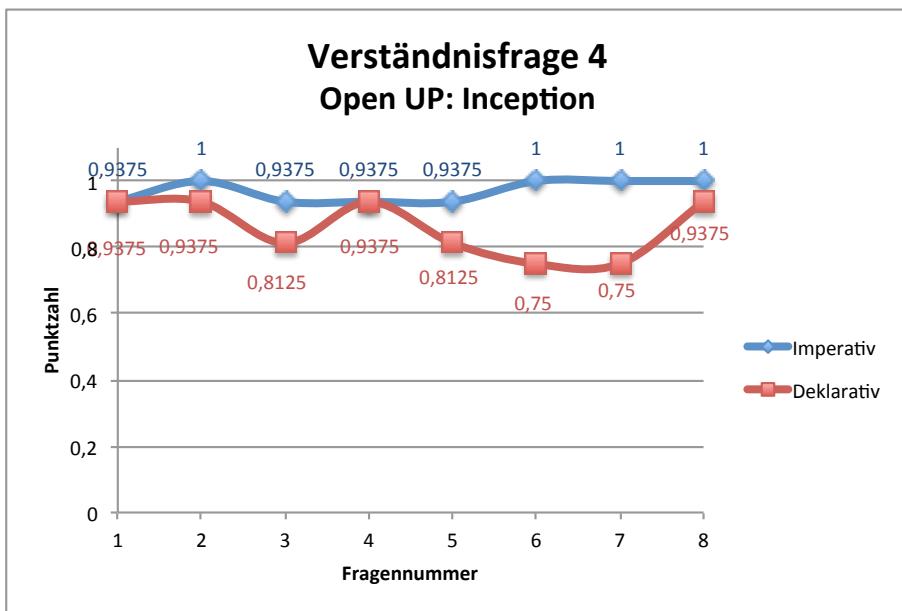


Abbildung 6.8.: Ergebnisse Verständnisfrage 4 aller Teilnehmer

6. Validierung

Ergebnisse Meinungsfragen

Abbildung 6.9 zeigt, dass 31 der 32 Befragten beim Modell *System spezifizieren* das imperative Modell bevorzugen. Nur eine Person zog das deklarative Modell vor. Hierbei handelt es sich um einen Teilnehmer, welcher weder in imperativer, noch in deklarativer Prozessmodellierung Erfahrung aufweist. Als Begründung für den Vorzug des deklarativen Modells gab der Befragte an, das Modell sei kompakter, jedoch sei auch mehr Verständnis notwendig.

Die Probanden, welche das imperative Modell bevorzugten gaben verschiedene Gründe hierfür an. Unter anderem nannten sie als Grund die klarere Struktur des BPMN-Modells, die vielen unterschiedlichen/komplexen Elemente im deklarativen Modell oder auch die klare Rollenverteilung durch die Swimlanes. Einige der Befragten gaben auch ihre besseren Kenntnisse in imperativen Prozessmodellierungssprachen als Grund an.

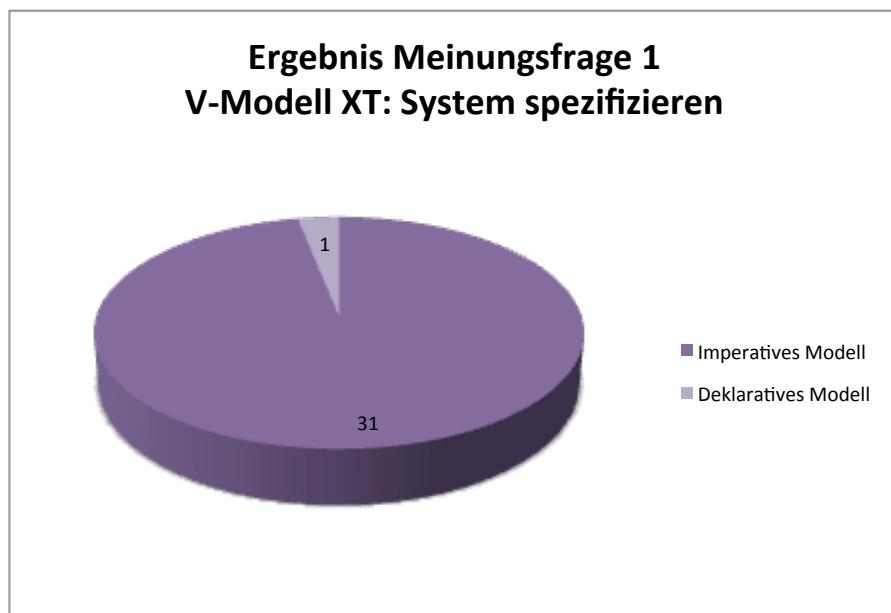


Abbildung 6.9.: Ergebnisse Meinungsfrage 1 aller Teilnehmer

Ebenfalls beim Modell *Phasen Open UP* bevorzugt eine deutliche Mehrheit (26 von 32 Befragten) das imperative Modell, wie Abbildung 6.10 entnommen werden kann.

6.2. Design der Studie

Hierbei verfügte nur einer der sechs Personen, welche das deklarative Modell bevorzugte auch über Erfahrung in deklarativer Modellierung. Die anderen Probanden verfügten entweder über keine Erfahrungen in beiden Modellierungssprachen (zwei) oder nur über Erfahrungen in imperativer Modellierung (drei). Als Grund für ihre Wahl gaben die Probanden beispielsweise an, dass das deklarative Modell kompakter sei und dass es klarer sei, dass nur bei Erfolg die nächste Aktivität ausgeführt wird.

Die Personen, welche das imperative Modell präferierten gaben an, dass sie den Ablauf mit den Schleifen im imperativen Modell klarer finden und dass sie dem Sequenzfluss besser folgen könnten.

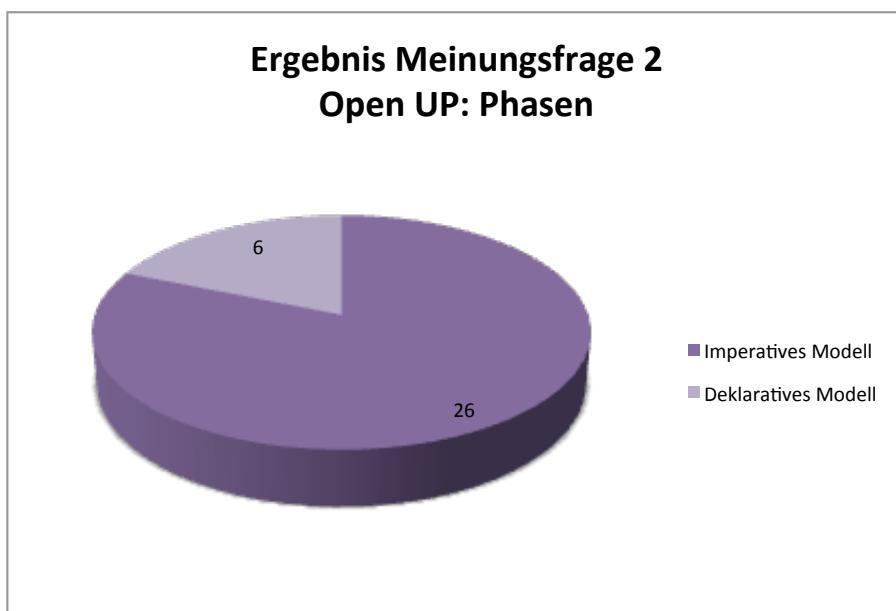


Abbildung 6.10.: Ergebnisse Meinungsfrage 2 aller Teilnehmer

Die Ergebnisse des dritten Modellpaars *Inkrementelle Entwicklung* zeigt Abbildung 6.11. Demnach präferierten nur zwei Personen (eine Person mit Erfahrung sowohl in imperativer, als auch in deklarativer Modellierung, eine Person ohne imperative und deklarative Modellierungserfahrung) das deklarative Modell und 30 Probanden ziehen das imperative Modell vor.

Als Grund für den Vorzug des imperativen Modells wurde die Menge an unterschiedli-

6. Validierung

chen Symbolen beim imperativen Modell genannt.

Die 30 Personen, welchen das imperative Modell besser gefiel gaben an, dass sie die imperative Notation verständlicher finden, der Ablauf im imperativen Modell klarer erkennbar sei, sie keinen Anhaltspunkt haben, wo im deklarativen Modell gestartet bzw. geendet wird und die vielen verschiedenen Constraints im deklarativen Modell es erschweren den Ablauf nachzuvollziehen.

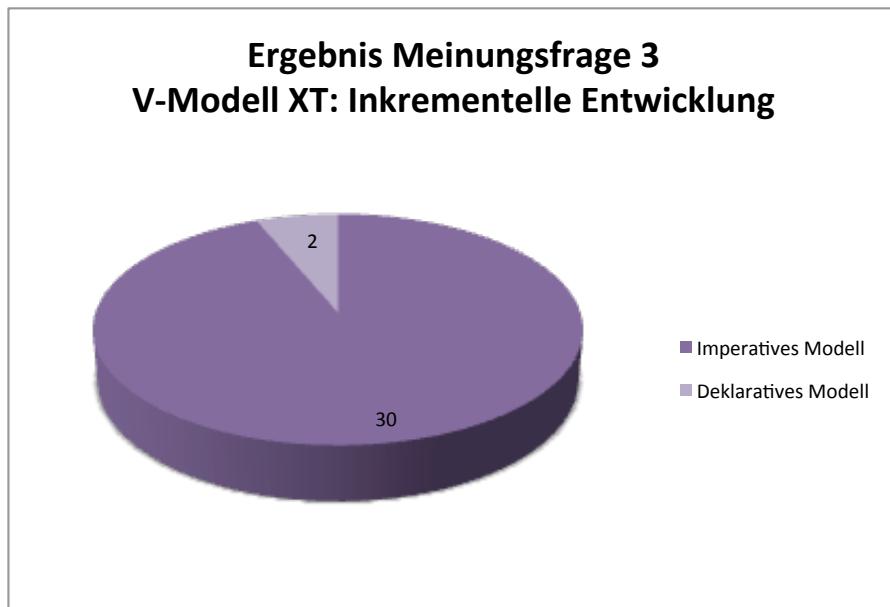


Abbildung 6.11.: Ergebnisse Meinungsfrage 3 aller Teilnehmer

Beim Modell *Open UP: Release deployen* präferierten neun Personen das deklarative Modell und 23 Teilnehmer das imperative Modell. Von den neun Teilnehmern, welche das deklarative Modell präferierten hatte nur einer Erfahrung in deklarativer Modellierung, einer hatte weder in deklarativer noch in imperativer Modellierung Erfahrung und sieben hatten nur in imperativer Modellierung Erfahrung.

Als Begründung für die Wahl des deklarativen Modelles wurde die Übersichtlichkeit desselbigen genannt und zwar auf Grund der fehlenden Artefakte im Modell. Es wurde bemängelt, die vielen Artefakte würden das imperative Modell unübersichtlich machen. Die Probanden, welche das imperative Modell besser fanden gaben als Gründe den

6.2. Design der Studie

Klaren Anfang und das klare Ende des Prozesses an, die bessere Verständlichkeit der Optionalität der Aktivität *Backoutplan ausführen* und die fehlenden Artefakte im deklarativen Modell.

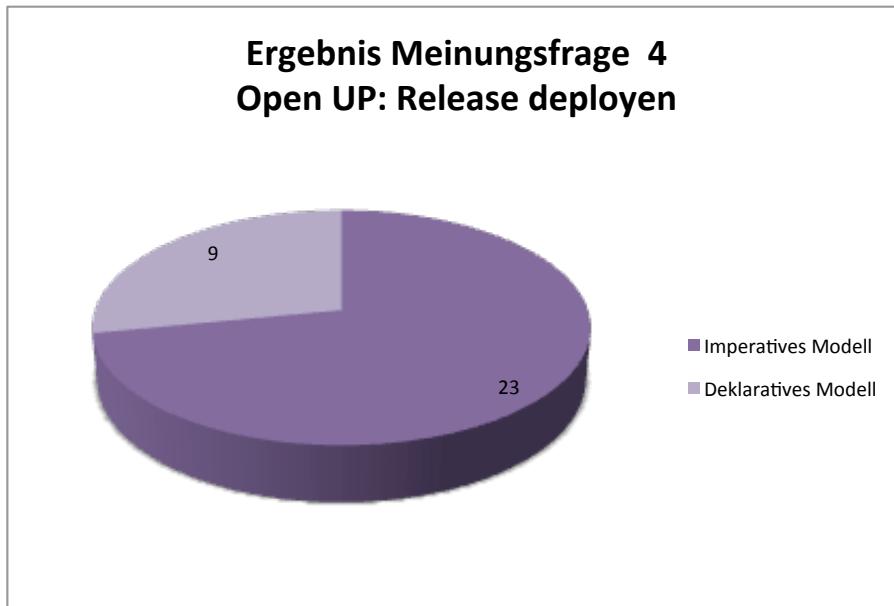


Abbildung 6.12.: Ergebnisse Meinungsfrage 4 aller Teilnehmer

6. Validierung

6.2.2. Fazit der Studie

Durch die Studie konnten die Ergebnisse des Vergleichs aus Kapitel 5 größtenteils belegt werden.

Bei Modellen, welche viele Verzweigungen/Schleifen beinhalten war für die Teilnehmer BPMN verständlicher. Dies zeigte sich beim kleinen Prozess *Open UP: Lösungskreislauf entwickeln* und noch deutlicher beim großen Modell *V-Modell: Systementwicklungsprojekt AG/AN*. Bei diesen beiden Prozessen sind in ConDec deutlich mehr Constraints, vor allem viele verschiedene Constraints zur korrekten Darstellung des Ablaufs notwendig als Gateways in BPMN. Dadurch haben die ConDec-Modelle eine deutlich höhere Komplexität, als die BPMN-Modelle.

Hier fällt besonders auf, dass Fragen bezüglich der Reihenfolge der Aktivitäten bei ConDec bei direkt aufeinander folgenden Aktivitäten größtenteils richtig beantwortet wurden. Jedoch bei Abläufen, bei denen es durch Verzweigungen im Prozessablauf zu einem Sprung kommt wurden nur die Fragen zu den BPMN-Modellen größtenteils richtig beantwortet. Bei ConDec wurden diese Fragen häufig falsch beantwortet.

7

Verwandte Arbeiten

In diesem Kapitel werden verwandte Arbeiten der vorliegenden Arbeit vorgestellt. Es werden Arbeiten zu den Themen Modellierung von Software-Engineering Prozessmodellen, Verständlichkeit von Prozessmodellierungssprachen und Vergleich von Prozessmodellierungssprachen vorgestellt und gegenüber der Thematik der vorliegenden Arbeit abgegrenzt.

7.1. Modellierung von Software-Engineering Prozessmodellen

Es gibt schon einige Arbeiten, welche sich mit der Modellierung von Software Engineering Prozessmodellen in BPMN beschäftigen. Die Bachelorarbeit [Men14] beschäftigt sich mit der Analyse und der Überführung von Softwareentwicklungsprozessen in die

7. Verwandte Arbeiten

Prozessmodellierungssprache BPMN und der Erweiterung von BPMN bei eventuellen Grenzen der Darstellbarkeit.

Weiterhin gibt es drei Arbeiten, bzw. Blögeinträge, welche sich mit der Modellierung von Scrum [CO13], Open UP [Bru07] und [Bre14] V-Modell XT in BPMN beschäftigen. Hier werden jeweils Teile der drei Software-Engineering Prozessmodelle analysiert und anschließend in BPMN modelliert.

Im Gegensatz zur vorliegenden Arbeit werden in diesen Arbeiten entweder jeweils nur ein Software Engineering Prozessmodell modelliert oder die Modellierung wird nur in der Prozessmodellierungssprache BPMN durchgeführt. In der vorliegenden Arbeit hingegen werden Teile von allen drei Software Engineering Prozessmodellen in zwei verschiedenen Prozessmodellierungssprachen modelliert. Der Fokus der vorliegenden Arbeit liegt auf der Beurteilung der Eignung zur Modellierung der beiden Prozessmodellierungssprachen und auf deren Vergleich.

7.2. Verständlichkeit von Prozessmodellierungssprachen

Verwandte Arbeiten zur Thematik Verständlichkeit von Prozessmodellierungssprachen werden im Folgenden vorgestellt.

Es existieren bereits einige Arbeiten, welche die Verständlichkeit von Prozessmodellen untersuchen. [MRC] z.B. untersucht die Verständlichkeit von Ereignisgesteuerten Prozessketten (EPK), während in [GL06b] die Komplexität und Verständlichkeit von BPMN und UML-Diagrammen untersucht werden. [RM11] untersucht die Verständlichkeit von Ereignisgesteuerten Prozessketten (EPK) und BPMN im Hinblick auf die Komplexität der XOR-, OR und UND-Verzweigungen.

In [ZSH⁺13] wird die Verständlichkeit von deklarativen Prozessmodellen im Hinblick auf die Verwendung von hierarchischen Unterprozessen untersucht. Eine weitere Arbeit, welche sich mit der Verständlichkeit von deklarativen Prozessmodellen beschäftigt ist [HBZ⁺14]. Hier wird das Vorgehen von Systemanalysten beim Verstehen von deklarati-

7.3. Vergleich von Prozessmodellierungssprachen

ven Modellen untersucht.

In der vorliegenden Arbeit werden einige Erkenntnisse der vorgestellten Arbeiten beim Modellieren und beim Vergleich der Prozessmodelle beachtet. Beispielsweise wurden Unterprozesse beim Modellieren verwendet um komplexe Prozessmodelle übersichtlicher darzustellen. Diese wurden aber beim Modellieren von sowohl BPMN als auch ConDec bei den gleichen Sachverhalten verwendet.

Außerdem wurden die Ergebnisse von [HBZ⁺14] zum schwierigeren Verständnis von Patterns in BPMN bei der Durchführung des Vergleichs herangezogen.

Im Gegensatz zu den hier vorgestellten Arbeiten liegt der Fokus des Vergleiches in dieser Arbeit nicht ausschließlich auf der Verständlichkeit der beiden Prozessmodellierungssprachen sondern vor allem auf deren Eignung zur Modellierung. Die Untersuchung der Verständlichkeit nimmt zwar in der vorliegenden Arbeit ebenfalls eine große Rolle ein. Jedoch wird hier explizit die Verständlichkeit von zwei verschiedenen Prozessmodellierungssprachen untersucht.

7.3. Vergleich von Prozessmodellierungssprachen

Dieser Abschnitt widmet sich Arbeiten aus dem Bereich Vergleich von Prozessmodellierungssprachen und dem Abgrenzen dieser Arbeiten gegenüber der vorliegenden Arbeit. Die Arbeit [RD07] untersucht Unterschiede in der Verständlichkeit zwischen Ereignissteuerten Prozessketten (EPK) und BPMN.

Der Artikel [FLM⁺09] beschäftigt sich mit dem Unterschied zwischen imperativen und deklarativen Prozessmodellierungssprachen und arbeitet deren Stärken und Schwächen heraus. Der Vergleich baut auf den Unterschieden von imperativen und deklarativen Programmiersprachen auf.

[PWZ⁺12] untersucht aufbauend auf den Erkenntnissen von [FLM⁺09] die Verständlichkeit von imperativen und deklarativen Prozessmodellierungssprachen anhand einer Studie. Als imperative Prozessmodellierungssprache dient in diesem Artikel ebenfalls

7. Verwandte Arbeiten

BPMN und als deklarative Prozessmodellierungssprache ebenfalls ConDec. [FLM⁺09] betrachtet im Wesentlichen die Stärken und Schwächen von imperativen und deklarativen Prozessmodellierungssprachen im Allgemeinen. Die vorliegende Arbeit untersucht die Stärken und Schwächen von imperativen und deklarativen Prozessmodellierungssprachen im Hinblick auf deren Eignung zur Modellierung in Bezug auf unterschiedlich große Prozessmodelle. Hier liegt der Fokus nicht auf sequentiellen und umständlichen Informationen sondern es werden die Stärken und Schwächen der imperativen und deklarativen Prozessmodellierungssprachen in Bezug auf den gleichen Sachverhalt und damit auch die gleichen abzubildenden Informationen betrachtet. Die vorliegende Arbeit untersucht die Verständlichkeit von deklarativen und imperativen Prozessmodellen nicht nur wie [PWZ⁺12] im Allgemeinen sondern auch speziell im Hinblick auf Unterschiede in der Verständlichkeit bei großen und kleinen Prozessmodellen. In der durchgeföhrten Studie in [PWZ⁺12] wurden die Teilnehmer vorher sowohl in der imperativen, als auch in der deklarativen Prozessmodellierung geschult. In der Studie der vorliegenden Arbeit wurde darauf geachtet Probanden mit unterschiedlichem Hintergrundwissen zu imperativen und deklarativen Prozessmodellen zu befragen. Das Wissen der Teilnehmer der Studie reichte hier von sehr großem Wissen bis überhaupt kein Wissen.

8

Zusammenfassung und Ausblick

A

BPMN Notation

A. BPMN Notation

	Start	Zwischen		Ende
	Standard	Ereignis-Teilprozess unterbrechend	Ereignis-Teilprozess Nicht-unterbrechend	
	Eingetreten	Angehettet unterbrechend	Angehettet Nicht-unterbrechend	Ausgelöst
Blanko: Umtypisierte Ereignisse; Blanko-Zwischenereignisse können einen Statuswechsel kennzeichnen	○			○
Nachricht: Empfang und Versand von Nachrichten	✉	✉	✉	✉
Zeit: Periodische zeitliche Ereignisse, Zeitpunkte oder Zeitspannen	🕒	🕒	🕒	
Bedingung: Reaktion auf veränderte Bedingungen und Bezug auf Geschäftsregeln	☒	☒	☒	
Link: Zwei zusammengehörige Link-Ereignisse repräsentieren einen Sequenzfluss			➡	➡
Signal: Signal über mehrere Prozesse. Auf ein Signal kann mehrfach reagiert werden.	△	△	△	▲
Fehler: Auslösen und Behandeln von definierten Fehlern		⚡		⚡
Eskalation: Meldung an den nächsthöheren Verantwortlichen	↗	↗	↗	↗
Terminierung: Löst die sofortige Beendigung des Prozesses aus				●
Kompensation: Behandeln oder Auslösen einer Kompensation		◀◀		◀◀
Abbruch: Reaktion auf abgebrochene Transaktionen oder Auslösen von Abbrüchen			✗	
Mehrfach: Eintreten eines von mehreren Ereignissen; Auslösen aller Ereignisse.	pentagon	pentagon	pentagon	pentagon
Mehrfach/Parallel: Eintreten aller Ereignisse.	⊕	⊕	⊕	

Abbildung A.1.: BPMN Ereignisse

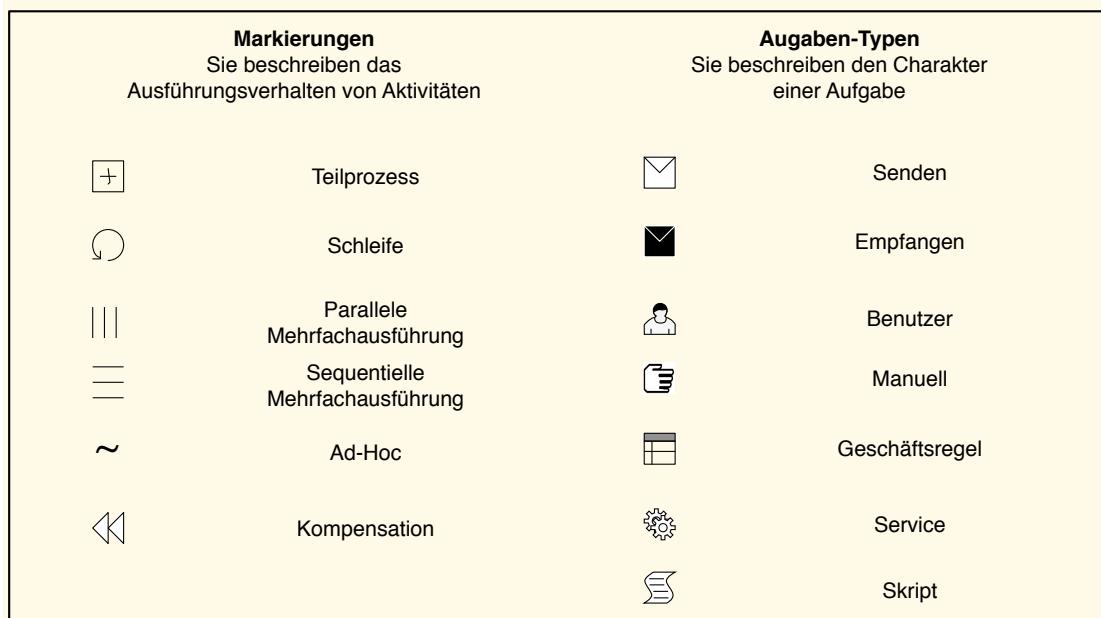
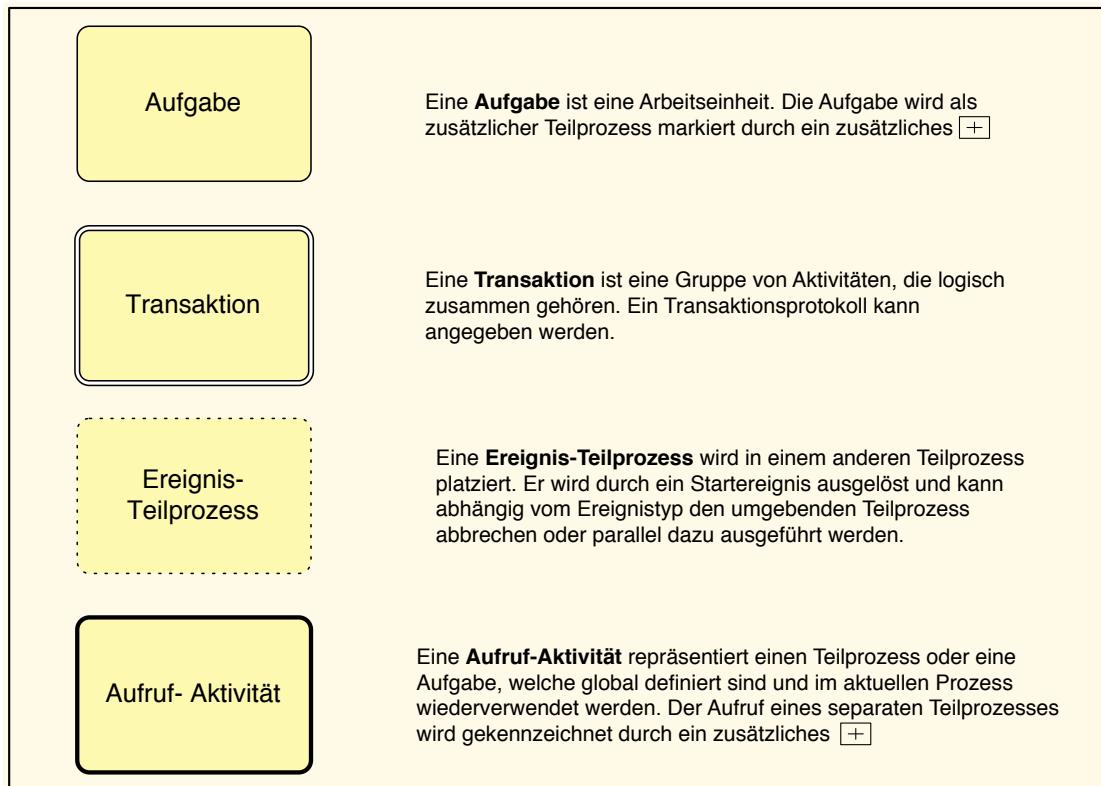


Abbildung A.2.: BPMN Übersicht

A. BPMN Notation

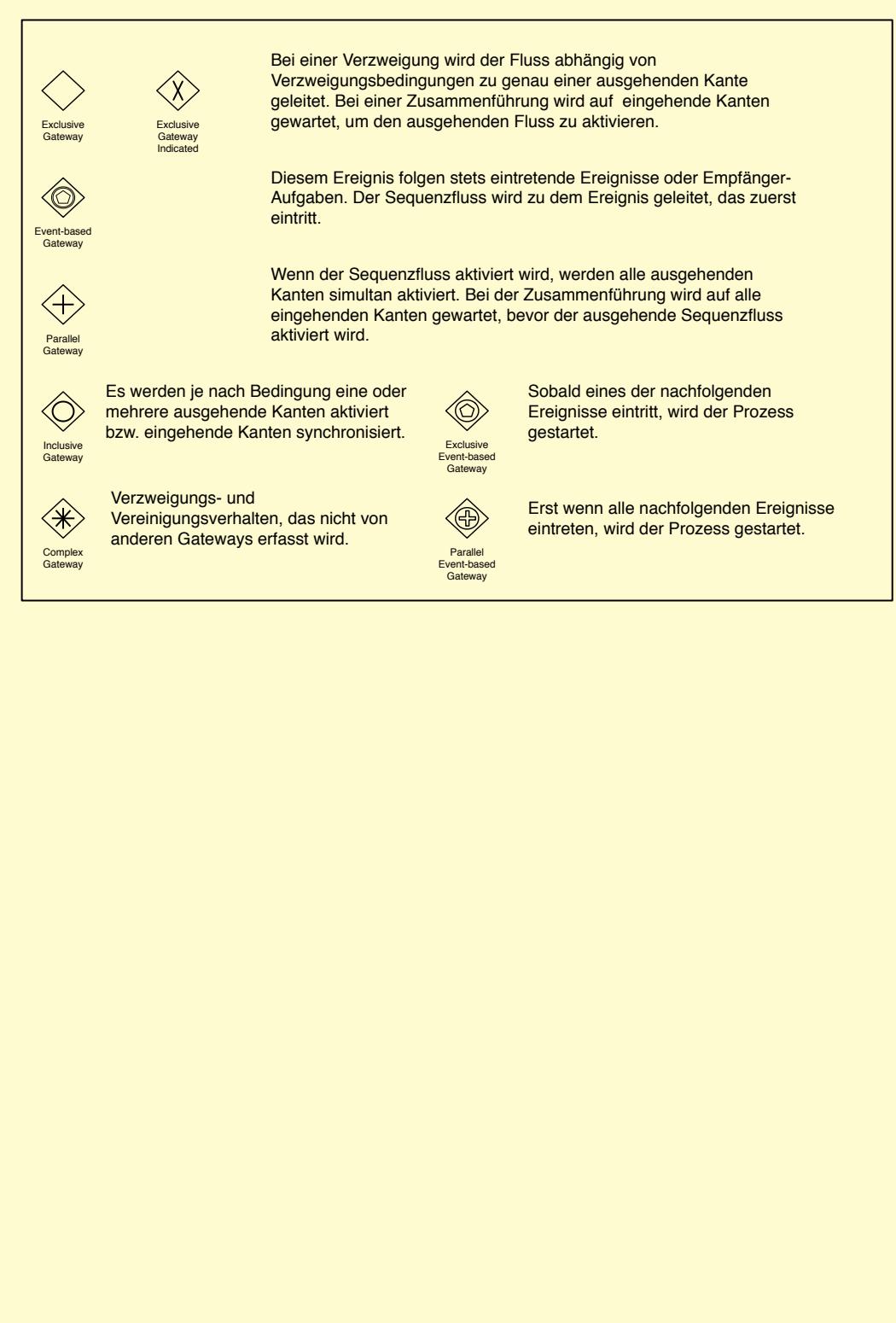


Abbildung A.3.: BPMN Gateways

B

ConDec Notation

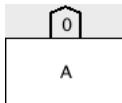
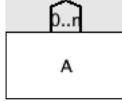
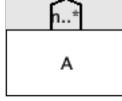
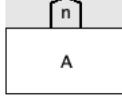
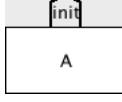
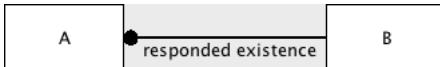
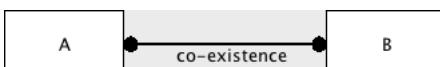
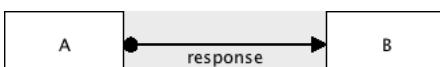
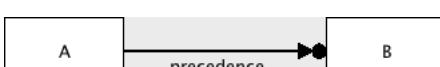
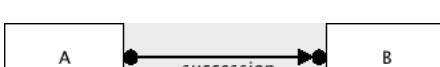
Constraint	Erläuterung
	absence (A) Aktivität A darf nicht ausgeführt werden
	absence (n+1, A) Aktivität A kann höchstens n-mal ausgeführt werden, aber nicht n+1-mal
	existence (n, A) Aktivität A muss mindestens n-mal ausgeführt werden
	exactly (n, A) Aktivität A muss genau n-mal ausgeführt werden
	init (A) Aktivität A muss als erste Aktivität ausgeführt werden

Tabelle B.1.: ExistenceConstraints ConDec

Constraint	Erläuterung
	responded existence Falls A ausgeführt wird, muss B entweder davor oder danach ebenfalls ausgeführt werden. Beispiel: Korrekt: [A,B]; [B,A]; Inkorrekt:[A];
	co-existence A und B kommen in einem Pfad immer zusammen vor. Beispiel: Korrekt: [A,B]; Inkorrekt: [A]; [B];
	response Falls A ausgeführt wird, muss B danach ebenfalls ausgeführt werden. Beispiel: Korrekt: [B,A,A,A,C,B]; Inkorrekt: [B,A,A,A,C]
	precedence Falls B ausgeführt wird, muss vorher A ausgeführt werden. Beispiel: Korrekt: [A,C,B,B,A]; Inkorrekt:[C,B,B,A]
	succession Verlangt, dass die beiden Constraints precedence und response zwischen den Aktivitäten A und B eingehalten werden. Somit muss jede Aktivität A von Aktivität B gefolgt werden und für jede Aktivität B muss eine Aktivität A vorhanden sein. Beispiel: Korrekt: [A,C,A,B,B]; Inkorrekt: [A,C,B,B]

B. ConDec Notation

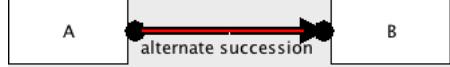
	alternate response Verlangt, dass nach einer Aktivität A Aktivität B ausgeführt wird, jedoch darf vor Aktivität B nicht eine weitere Aktivität A ausgeführt werden. Beispiel: Korrekt: [B,A,C,B,A,B] ; Inkorrekt: [B,A,C,A,B,A,B]
	alternate precedence Verlangt, dass jeder Instanz von Aktivität B eine Instanz der Aktivität A vorausgeht. Die nächste Instanz einer Aktivität B kann somit nicht vor der nächsten Instanz von Aktivität A ausgeführt werden. Beispiel: Korrekt: [A,C,B,A,B,A]; Inkorrekt: [A,C,B,B,A]
	alternate succession Stellt eine Kombination aus alternate response und alternate precedence dar. Beispiel: Korrekt: [A,C,B,A,B,A,B]; Inkorrekt: [C,B,A,A,B]
	chain response Verlangt, dass die nächste Aktivität, welche nach Aktivität A ausgeführt wird, Aktivität B ist. Beispiel: Korrekt: [B,A,B,C,A,B]; Inkorrekt: [B,A,C,A,B]
	chain precedence Verlangt, dass Aktivität A unmittelbar bevor Aktivität B ausgeführt wird. Beispiel: Korrekt: [A,B,C,A,B,A]; Inkorrekt: [A,B,C,B,A]
	chain succession Stellt eine Kombination aus chain response und chain precedence dar und verlangt, dass Aktivität A und Aktivität B jeweils nebeneinander ausgeführt werden. Beispiel: Korrekt: [A,B,C,A,B,A,B]; Inkorrekt: [A,B,C,A,B,A,B,A,C]

Tabelle B.2.: Relation Constraints ConDec

	exclusive choice Entweder A oder B kann ausgeführt werden. Beispiel: Korrekt: [A]; [B] Inkorrekt: [A,B]
	exclusive choice Entweder A oder B oder C kann ausgeführt werden. Beispiel: Korrekt: [A]; [B]; [C]; Inkorrekt: [A,B]; [A,C]; [B,C]

Tabelle B.3.: Choice Constraints ConDec

Constraint	Erläuterung
	choice Mindestens eine der beiden Aktivitäten A oder B muss ausgeführt werden. Beispiel: Korrekt: [A]; [B]; Inkorrekt: []
	choice 1 of 3 Mindestens eine der drei Aktivitäten A,B oder C muss ausgeführt werden.. Beispiel: Korrekt: [A]; Inkorrekt: []
	1 of 2 Entweder A oder B muss mindestens einmal ausgeführt werden. Beispiel: Korrekt: [A,C,A,B,B]; Inkorrekt: [A,C,B,B]

B. ConDec Notation

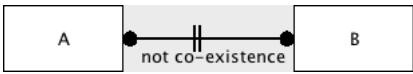
	<p>not co-existence</p> <p>Verlangt, dass falls Aktivität A ausgeführt wird, darf Aktivität B nicht mehr ausgeführt werden und umgekehrt.</p> <p>Beispiel: Korrekt: [A,C,A,A] ; Inkorrekt: [A,C,A,A,B]</p>
	<p>not succession</p> <p>Verlangt, dass falls Aktivität A ausgeführt wird, darf Aktivität B nicht danach ausgeführt werden.</p> <p>Beispiel: Korrekt: [B,B,C,A,C,A,A] ; Inkorrekt: [A,C,B]</p>
	<p>negation chain succession</p> <p>Verlangt, dass die Aktivitäten A und B nicht nebeneinander ausgeführt werden.</p> <p>Beispiel: Korrekt: [B,A,C,B,A] ; Inkorrekt: [B,A,B,A]</p>

Tabelle B.4.: Negotiation Constraints ConDec

[Mon10, AP06]

Abbildungsverzeichnis

2.1. Schichten des Software Engineering [Pun07]	4
2.2. Ziele, Prozess und Prinzipien des Software Engineering [RGI75]	5
2.3. Phasen Softwareprozess nach [Han10]	8
2.4. Software-Projekttypen nach [Boe81]	9
3.1. Ziele der Prozessmodellierung nach [Koc11]	12
3.2. Grundsatz ordnungsgemäßer Modellierung nach [BRS95]	14
3.3. BPMN-Elemente Übersicht nach [GL12]	18
3.4. BPMN-Gateways	20
3.5. Deklarativer Beispiel-Prozess [PA06]	22
3.6. Siganvio Process Editor (Screenshot Siganvio)	24
3.7. Siganvio Simulation (Screenshot Signavio)	25
3.8. Declare Systemarchitektur nach [PSA07]	26
3.9. Declare Designer (Screenshot aus Declare)	27
3.10. Declare Framework (Screenshot aus Declare)	28
3.11. Declare Worklist (Screenshot aus Declare)	28
4.1. Übersicht Vergleichskriterien	35
5.1. Scrum Überblick nach [Mou14b]	39
5.2. Imperative Modellierung Scrum	42
5.3. Imperative Modellierung Scrum Unterprozess	43
5.4. Deklarative Modellierung Scrum	45

Abbildungsverzeichnis

5.5. Deklarative Modellierung Scrum-Unterprozess Sprint-Planning-Meeting durchführen	45
5.6. Vergleich der Anzahl der Elemente Scrum	46
5.7. Zusammenfassung Vergleich Scrum	49
5.8. Open Up Überblick nach [Ric07]	51
5.9. Phasen Open UP nach [Ric07]	52
5.10. Rollen in Open UP nach [Spa14]	55
5.11. Phasen Open UP- imperativ	56
5.12. Phasen Open UP Unterprozess Inception- imperativ	57
5.13. Phasen Open UP Unterprozess Elaboration- imperativ	58
5.14. Phasen Open UP Unterprozess Construction- imperativ	59
5.15. Phasen Open UP Unterprozess Transition- imperativ	60
5.16. Lösungssinkrement entwickeln imperativ	62
5.17. Iteration planen und managen imperativ -Inception	63
5.18. Iteration planen und managen imperativ -Inception Unterprozess Umgebung vorbereiten	64
5.19. Anforderungen identifizieren und verfeinern-Elaboration	66
5.20. Produktdokumentation und Training erstellen - Construction	67
5.21. Release deployen-Transition	68
5.22. Phasen Open UP- deklarativ	69
5.23. Phasen Open UP Unterprozess Inception- deklarativ	70
5.24. Phasen Open UP Unterprozess Elaboration- deklarativ	71
5.25. Phasen Open UP Unterprozess Construction- deklarativ	71
5.26. Phasen Open UP Unterprozess Transition- deklarativ	72
5.27. Lösungssinkrement entwickeln- deklarativ	73
5.28. Iteration planen und managen-Inception deklarativ	74
5.29. Iteration planen und managen- Inception Unterprozess Umgebung vorbereiten- deklarativ	75
5.30. Anforderungen identifizieren und verfeinern-Elaboration	75
5.31. Produktdokumentation und Training erstellen-Construction	76
5.32. Release deployen-Transition	77

Abbildungsverzeichnis

5.33. Phasen Open UP	78
5.34. Open UP-Inception	79
5.35. Open UP-Elaboration	79
5.36. Open UP-Construction	80
5.37. Open UP-Transition	80
5.38. Lösungssinkrement entwickeln	81
5.39. Open UP-Iteration planen	82
5.40. Anforderungen identifizieren	82
5.41. Produktdokumentation erstellen	83
5.42. Release deployen	84
5.43. Übersicht Vergleich Open UP	87
5.44. Grundstruktur V-Modell XT nach [Bun04]	88
5.45. Projekttypen V-Modell XT nach [Bun04]	89
5.46. Zuordnung der Projekttypvarianten zu den Projekttypen des V-Modell XT [Bun04]	91
5.47. Vorgehensbausteine V-Modell XT nach [Bun04]	92
5.48. V-Modell-Kern und Vorgehensbausteinlandkarte nach [Bun04]	93
5.49. Entscheidungspunkte V-Modell XT nach [Bun04]	96
5.50. Entscheidungspunkte für die Projektdurchführungsstrategie nach [Bun04]	97
5.51. Systementwicklungsprojekt AG/AN V-Modell - imperativ	98
5.52. Unterprozess Inkrementelle Entwicklung durchführen V-Modell - imperativ	99
5.53. System entwerfen V-Modell - imperativ	100
5.54. System spezifizieren-imperativ	101
5.55. Systementwicklungsprojekt AG/AN V-Modell - deklarativ	103
5.56. Systementwicklungsprojekt AG/AN V-Modell Unterprozess Entwicklung durchführen - deklarativ	103
5.57. Unterprozess Inkrementelle Entwicklung durchführen V-Modell - imperativ	104
5.58. System entwerfen - deklarativ	105
5.59. System spezifizieren- deklarativ	106
5.60. Systementwicklungsprojekt AG/AN	107
5.61. Inkrementelle Entwicklung	108

Abbildungsverzeichnis

5.62. System entwerfen	108
5.63. System spezifizieren	109
5.64. Übersicht Vergleich V-Modell	111
5.65. Übersicht Vergleich Allgemein	115
6.1. Struktur der Umfrage	119
6.2. Geschlechterverteilung	121
6.3. Verteilung Erfahrung imperative und deklarative Modellierung	122
6.4. Allgemeine demographische Daten	122
6.5. Ergebnisse Verständnisfrage 1 aller Teilnehmer	123
6.6. Ergebnisse Verständnisfrage 2 aller Teilnehmer	125
6.7. Ergebnisse Verständnisfrage 3 aller Teilnehmer	126
6.8. Ergebnisse Verständnisfrage 4 aller Teilnehmer	127
6.9. Ergebnisse Meinungsfrage 1 aller Teilnehmer	128
6.10. Ergebnisse Meinungsfrage 2 aller Teilnehmer	129
6.11. Ergebnisse Meinungsfrage 3 aller Teilnehmer	130
6.12. Ergebnisse Meinungsfrage 4 aller Teilnehmer	131
A.1. BPMN Ereignisse	140
A.2. BPMN Übersicht	141
A.3. BPMN Gateways	142

Tabellenverzeichnis

3.1. Constraints ConDec [PA06]	23
5.1. Iterationen und Zielstellungen der Phasen in Open UP [Ric07]	54
B.1. ExistenceConstraints ConDec	144
B.2. Relation Constraints ConDec	146
B.3. Choice Constraints ConDec	147
B.4. Negotiation Constraints ConDec	148

Literaturverzeichnis

- [AL12] AMBLER, S.W. ; LINES, M.: *Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise*. Pearson Education, 2012
- [All13] ALLWEYER, T.: *BPMN 2.0 - Business Process Model and Notation: Einführung in den Standard für die Geschäftsprozessmodellierung*. Books on Demand, 2013
- [AP06] AALST, W.M.P. ; PESIC, M.: DecSerFlow: Towards a Truly Declarative Service Flow Language. In: BRAVETTI, Mario (Hrsg.) ; NÚÑEZ, Manuel (Hrsg.) ; ZAVATTARO, Gianluigi (Hrsg.): *Web Services and Formal Methods*. Springer Berlin Heidelberg, 2006 (Lecture Notes in Computer Science), S. 1–23
- [Bal07] BALDUINO, RICARDO : *Introduction to OpenUP (Open Unified Process)*. <http://www.eclipse.org/epf/general/OpenUP.pdf>. Version: August 2007
- [Bec12] BECKER, J.: *Prozessmanagement: Ein Leitfaden Zur Prozessorientierten Organisationsgestaltung*. Springer Berlin Heidelberg, 2012
- [Boe81] BOEHM, Barry W.: *Software Engineering Economics*. 1981
- [Bra10] BRACK, T.: *Das V-Modell XT 1.2.1 im Umfeld der Qualitätssicherung nach ISO 9001:2000*. Diplom.de, 2010
- [Bre14] BREGENZER, Stefan: *Projektdurchführungstrategie als strategisches Prozessmodell in BPMN*. <http://blog.milsystems.de/2012/08/projektdurchfuehrungstrategie-als-strategisches-prozessmodell-in-bpmn/>. Version: august 2014

Literaturverzeichnis

- [BRS95] BECKER, Jörg ; ROEMANN, Michael ; SCHÜTTE, Reinhard: Grundsätze ordnungsmäßiger Modellierung. In: *Wirtschaftsinformatik* 37 (1995), Nr. 5, S. 435–445
- [Bru07] BRUNNER, Maximilian: *Fallstudie zur Modellierung von Software-Entwicklungsprozessen auf Basis des Software Process Engineering Metamodel 2.0*, Diplomarbeit, Lehrstuhl für Informatik 2, FAU, Diss., 2007
- [Bun04] BUNDESVERWALTUNG, Beratungsstelle der Bundesregierung für Informationstechnik in d. ; DEUTSCHLAND, Bundesrepublik (Hrsg.): V-Modell XT. Version:2004. <http://ftp.uni-kl.de/pub/v-modell-xt/Release-1.2/Dokumentation/pdf/V-Modell-XT-Teill1.pdf>. 2004. – Forschungsbericht
- [CO13] CAMPOS, Andre L. N. ; OLIVEIRA, Toacy C.: Software Processes with BPMN: An Empirical Analysis. In: *PROFES'13*, 2013, S. 338–341
- [COR09] CORP, IBM: *OpenUP Process Version 1.5.0.4*. <http://epf.eclipse.org/wikis/openup/>. Version:2009
- [DB93] DOUG, Bryan ; BOOCH, Grady: *Software Engineering with ADA*. 3. Addison Wesley Pub Co Inc, 1993
- [Ecl14] ECLIPSE: *Eclipse Scrum Library*. http://www.eclipse.org/epf/downloads/praclib/praclib_downloads.php. Version:July 2014
- [EHS10] EL-HAIK, Basem ; SHAOUT, Adnan: *Software Design For Six Sigma: A Roadmap For Excellence*. Hoboken, N.J. : Wiley, 2010
- [FHK08] FRIEDRICH, Jan ; HAMMERSCHALL, Ulrike ; KUHRMANN, Marco ; SIHLING, Marc: *Das V-Modell XT*. Heidelberg : Springer, 2008
- [FLM⁺09] FAHLAND, Dirk ; LÜBKE, Daniel ; MENDLING, Jan ; REIJERS, Hajo ; WEBER, Barbara ; WEIDLICH, Matthias ; ZUGAL, Stefan: Declarative versus imperative process modeling languages: The issue of understandability. In: *Enterprise, Business-Process and Information Systems Modeling*. Springer, 2009, S. 353–366

- [FMR⁺10] FAHLAND, Dirk ; MENDLING, Jan ; REIJERS, Hajo A. ; WEBER, Barbara ; WEIDLICH, Matthias ; ZUGAL, Stefan: Declarative versus imperative process modeling languages: the issue of maintainability. In: *Business Process Management Workshops*, Springer, 2010, S. 477–488
- [Fre] FREUDENREICH, Robert: Evaluierung der Potentiale des Eclipse Process Frameworks. In: *Modellbasierte Softwareentwicklung*
- [Fre07] FREUND, T.: *Software Engineering durch Modellierung wissensintensiver Entwicklungsprozesse*. GITO, 2007
- [Gad12] GADATSCH, A.: *Grundkurs Geschäftsprozess-Management*. Vieweg+Teubner Verlag, 2012
- [Gau06] GAU, Thosten: UMA und EPF: Einführung und Anwendung in der Praxis. In: *Objekt Spektrum, (November/Dezember 2006-6)* (2006), S. 42–47
- [GBBK10] GRECHENIG, Thomas ; BERNHART, Mario ; BREITENEDER, Roland ; KAPPEL, Karin: *Softwaretechnik*. München : Pearson Studium, 2010
- [GL06a] GRUHN, Volker ; LAUE, Ralf: Adopting the cognitive complexity measure for business process models. In: *Cognitive Informatics, 2006. ICCI 2006. 5th IEEE International Conference on* Bd. 1 IEEE, 2006, S. 236–241
- [GL06b] GRUHN, Volker ; LAUE, Ralf: Complexity metrics for business process models. In: *9th international conference on business information systems (BIS 2006)* Bd. 85, 2006, S. 1–12
- [GL12] GÖPFERT, J. ; LINDENBACH, H.: *Geschäftsprozessmodellierung mit BPMN 2.0: Business Process Model and Notation*. Oldenbourg Wissenschaftsverlag, 2012
- [Han10] HANSER, Eckhart: *Agile Prozesse*. Heidelberg : Springer, 2010
- [HBZ⁺14] HAISJACKL, Cornelia ; BARBA, Irene ; ZUGAL, Stefan ; SOFFER, Pnina ; HADAR, Irit ; REICHERT, Manfred ; PINGGERA, Jakob ; WEBER, Barbara: Understanding Declare models: strategies, pitfalls, empirical results. In: *Software & Systems Modeling* (2014), S. 1–28

Literaturverzeichnis

- [Hei07] HEINRICH, G.: *Allgemeine Systemanalyse*. Oldenbourg, 2007
- [HM08] HESSE, Wolfgang ; MAYR, Heinrich C.: Modellierung in der Softwaretechnik: eine Bestandsaufnahme. In: *Informatik-Spektrum* 31 (2008), Nr. 5, S. 377–393
- [HM10] HAUBER, Rudolf ; MUTH, Bertil: Architekturprozesse–Systeme systematisch entwickeln. In: URL: www.sigs-datacom.de (Zugriffsdatum: 22. 03. 2013) (2010)
- [HRB⁺08] HÖHN, Reinhard ; RAUSCH, A. ; BROY, M. ; HÖPPNER, Stephan ; BERGNER, K. ; PETRASCH, R. ; BIFFL, S. ; WAGNER, R. ; HESSE, W.: *Das V-Modell XT: Grundlagen, Methodik und Anwendungen*. Heidelberg : Springer, 2008
- [Kas98] KASCHEK, Roland: Prozeßontologie als Faktor der Geschäftsprozeßmodellierung. In: *Modellierung*, 1998
- [KBL13] KRALLMANN, H. ; BOBRIK, A. ; LEVINA, O.: *Systemanalyse im Unternehmen: Prozessorientierte Methoden der Wirtschaftsinformatik*. Oldenbourg Wissenschaftsverlag, 2013
- [Kei10] KEITH, Clinton: *Agile game development with Scrum: Description based on print version record*. Upper Saddle River, N.J. : Addison-Wesley, 2010
- [Kir06] KIRCHER, H.: *IT: Technologien, Lösungen, Innovationen*. Springer, 2006
- [KLS11] KUHRMANN, Marco ; LANGE, Christian ; SCHNACKENBURG, André: In: O'CONNOR, Rory (Hrsg.) ; PRIES-HEJE, Jan (Hrsg.) ; MESSNARZ, Richard (Hrsg.): *EuroSPI*. Heidelberg : Springer, 2011, S. 49–60
- [Koc11] KOCH, Susanne: *Einführung in das Management von Geschäftsprozessen*. Heidelberg : Springer, 2011
- [Kö00] KÖLMEL, Bernhard: *Softwareprozessverbesserungsprojekte*. Norderstedt : Books on Demand GmbH, 2000
- [Lac12] LACEY, Mitch: *The Scrum field guide: practical advice for your first year*. [S.I.] : Addison-Wesley Professional, 2012

- [Lei12] LEIMEISTER, J.M.: *Dienstleistungsengineering und -management*. Springer Berlin Heidelberg, 2012
- [Lic12] LICHTENEGGER, W.: *Methoden zur teilautomatischen Konstruktion von Ist-Prozessmodellen mittels Process Mining sowie zur Integration manuell konstruierter und automatisch generierter Ist-Prozessmodelle*. Logos Verlag Berlin, 2012
- [LK06] LIST, Beate ; KORHERR, Birgit: An evaluation of conceptual business process modelling languages. In: *Proceedings of the 2006 ACM symposium on Applied computing* ACM, 2006, S. 1532–1539
- [Men14] MENHORN, Nicole: *Analyse und Überführung von Softwareentwicklungsprozessen in die standardisierte BPMN Notation*. 2014
- [MM12] MISHRA, Jibitesh ; MOHANTY, Ashok: *Software Engineering*. New Delhi : Pearson Studium, 2012
- [Mon10] MONTALI, Marco: *Specification and verification of declarative open interaction models: A logic-based approach*. Bd. 56. Springer, 2010
- [Mou14a] MOUNTAIN GOAT SOFTWARE: *BPMN Offensive Berlin*. <http://www.bpmb.de/index.php/BPMNPoster>. Version: April 2014
- [Mou14b] MOUNTAIN GOAT SOFTWARE: *Scrum Overview*. <http://epf.eclipse.org/wikis/scrum/>. Version: April 2014
- [MRC] MENDLING, J. ; REIJERS, H.A. ; CARDOSO, J.: What Makes Process Models Understandable? In: ALONSO, G. (Hrsg.) ; DADAM, P. (Hrsg.) ; ROSEMANN, M. (Hrsg.): *International Conference on Business Process Management (BPM 2007)*, Springer-Verlag, Berlin (Lecture Notes in Computer Science), S. 48–63
- [MRW12] MENDLING, Jan ; RECKER, Jan C. ; WOLF, Johannes: Collaboration features in current BPM tools. In: *EMISA Forum* 32 (2012), January, Nr. 1, S. 48–65
- [PA06] PESIC, Maja ; AALST, Wil M. d.: A declarative approach for flexible business processes management. In: *Business Process Management Workshops*, Springer, 2006, S. 169–180

Literaturverzeichnis

- [Pes08] PESIC, M.: *Constraint-based Workflow Management Systems: Shifting Control to Users*, Eindhoven University of Technology, PhD Thesis, 2008
- [Pha12] PHAM, Andrew: *Scrum in action: agile software project management and development*. Boston, Mass. : Course Technology, 2012
- [Pic10] PICHLER, Roman: *Agile product management with Scrum: creating products that customers love. - Description based on print version record*. Upper Saddle River, N.J. : Addison-Wesley, 2010
- [Pit10] PITSCHEKE, J.: *Unternehmensmodellierung für die Praxis: Eine Einführung in die Darstellung von Unternehmensmodellen*. Books on Demand, 2010
- [PQ11] PRIES, Kim H. ; QUIGLEY, Jon M.: *Scrum project management*. Boca Raton, Fla. [u.a.] : Woodhead Publishing Limited, 2011
- [PSA07] PESIC, Maja ; SCHONENBERG, Helen ; AALST, Wil M. d.: Declare: Full support for loosely-structured processes. In: *Enterprise Distributed Object Computing Conference, 2007. EDOC 2007. 11th IEEE International*, IEEE, 2007, S. 287–287
- [Pun07] PUNTAMBEKAR, A.A.: *Software Engineering*. Pune : Technical Publications Pune, 2007
- [PWZ⁺12] PICHLER, Paul ; WEBER, Barbara ; ZUGAL, Stefan ; PINGGERA, Jakob ; MENDLING, Jan ; REIJERS, Hajo A.: Imperative versus declarative process modeling languages: An empirical investigation. In: *Business Process Management Workshops* Springer, 2012, S. 383–394
- [RD07] RECKER, Jan C. ; DREILING, Alexander: Does it matter which process modelling language we teach or use? an experimental study on understanding process modelling languages without formal education. (2007)
- [Rei09] REINSHAGEN, F.: *Konzepte einer komprimierten Informationsversorgung für die interne Führung und externe Performance-Kommunikation grosser Publikumsgesellschaften*. Logos-Verlag, 2009

- [RF08] RUF, W. ; FITTKAU, T.: *Ganzheitliches IT-Projektmanagement: Wissen, Praxis, Anwendungen*. Oldenbourg, 2008
- [RG175] Ross, Douglas T. ; GOODENOUGH, John B. ; IRVINE, CA: Software engineering: process, principles, and goals. In: *Computer* 8 (1975), Nr. 5, S. 17–27
- [RM11] REIJERS, Hajo A. ; MENDLING, Jan: A study into the factors that influence the understandability of business process models. In: *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on* 41 (2011), Nr. 3, S. 449–462
- [RW12] REICHERT, Manfred ; WEBER, Barbara: *Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies*. Springer, 2012
- [Sch04] SCHWABER, Ken: *Agile Project Management with Scrum*. MRedmond : Microsoft Press, 2004
- [Sch07] SCHWABER, Ken: *The enterprise and Scrum*. MRedmond : Microsoft Press, 2007
- [Spa14] SPARKLING CONSULTING: *Open Unified Process (OpenUP)*. <http://www.itpractices.org/Live/framework/openup>. Version: April 2014
- [Spe98] SPECKER, Adrian: *Kognitives Software Engineering*. Zürich : vdf Hochschulverlag AG, 1998
- [Sta06] STAUD, Josef: *Geschäftsprozessanalyse: Ereignisgesteuerte Prozessketten und objektorientierte Geschäftsprozessmodellierung für betriebswirtschaftliche Standardsoftware*. Heidelberg : Springer, 2006
- [Stö05] STÖRRLE, Harald: *UML 2 für Studenten*. München : Pearson Studium, 2005
- [Tho09] THOMAS, Oliver: *Fuzzy Process Engineering*. Gabler Verlag, 2009
- [TN86] TAKEUCHI, Hirotaka ; NONAKA, Ikujiro: The New New Product Development Game. In: *Harvard Business Review* (1986)
- [Whi04] WHITE, Stephen A.: Introduction to BPMN. In: *IBM Cooperation* 2 (2004)

Literaturverzeichnis

- [Wol11a] WOLF, Henning: *Die Kraft von Scrum: eine inspirierende Geschichte über einen revolutionären Projektmanagementansatz*. München : Addison-Wesley, 2011
- [Wol11b] WOLF, Henning: *Die Kraft von Scrum: eine inspirierende Geschichte über einen revolutionären Projektmanagementansatz*. München : Addison-Wesley, 2011
- [WPR07] WANG, Qing (Hrsg.) ; PFAHL, Dietmar (Hrsg.) ; RAFFO, David M. (Hrsg.): *Software Process Dynamics and Agility, International Conference on Software Process, ICSP 2007, Minneapolis, MN, USA, May 19-20, 2007, Proceedings*. Bd. 4470. Heidelberg : Springer, 2007 (Lecture Notes in Computer Science)
- [ZSH⁺13] ZUGAL, Stefan ; SOFFER, Pnina ; HAISJACKL, Cornelia ; PINGGERA, Jakob ; REICHERT, Manfred ; WEBER, Barbara: Investigating expressiveness and understandability of hierarchy in declarative business process models. In: *Software & Systems Modeling* (2013)
- [Zö12] ZÖRNER, Stefan: *Software-Architekturen Dokumentieren und Kommunizieren*. Carl Hanser Verlag, 2012

Name: Bianka Hampp

Matrikelnummer: MATRIKEL NR

Erklärung

Ich erkläre, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den

Bianka Hampp