



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:

Gutachter 1

JAHR

Fassung 25. Oktober 2014

© JAHR 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- \LaTeX 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. Schwergewichtige 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	15
3.2.1. Imperative Modellierung	16
3.2.2. Deklarative Modellierung	20
4. Modellierungswerkzeuge	23
4.1. Modellierungswerkzeuge	23
4.1.1. Signavio	24

4.1.2. Declare	26
5. Anforderungserhebung	31
5.1. Vergleichskriterien	31
5.1.1. Erfüllung der Modellierungsgrundsätze	32
5.1.2. Weitere Vergleichskriterien	35
6. Imperative und deklarative Modellierung für SE-Prozessmodelle	37
6.1. Scrum	38
6.1.1. Analyse Scrum	38
6.1.2. Imperative Modellierung Scrum	41
6.1.3. Deklarative Modellierung Scrum	43
6.1.4. Vergleich	45
6.2. Open Unified Process (Open UP)	47
6.2.1. Analyse Open UP	49
6.2.2. Imperative Modellierung Open UP	53
6.2.3. Deklarative Modellierung Open UP	66
6.2.4. Vergleich	74
6.3. V-Modell XT	77
6.3.1. Analyse V-Modell XT	77
6.3.2. Imperative Modellierung V-Modell	87
6.3.3. Deklarative Modellierung V-Modell	91
6.3.4. Vergleich	95
6.3.5. Vergleich	95
7. Validierung	97
8. Related Work	99
8.1. Modellierung von Software-Engineering Prozessmodellen	99
8.1.1. Analyse und Überführung von Softwareentwicklungsprozessen in die standardisierte BPMN Notation	99

8.2. Verständlichkeit von Prozessmodellierungssprachen	100
8.2.1. Investigating expressiveness and understandability of hierarchy in declarative business process models	100
8.3. Vergleich von Prozessmodellierungssprachen	101
8.3.1. Declarative versus Imperative Process Modeling Languages: An Empirical Investigation	101
8.3.2. Imperative versus Declarative Process Modeling Languages: An Empirical Investigation	101
9. Zusammenfassung und Ausblick	103
A. BPMN Notation	105
B. ConDec Notation	109

1

Einleitung

1.1. Einleitung

1.1.1. Motivation

1.1.2. Zielstellung

1.1.3. Aufbau der Arbeit

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 vorgestellt. Anschließend gibt es eine Einführung in die drei Softwareprozessmodelle Scrum, Open Unified Process und V-Modell-XT.

2.1. Software Engineering

Heutzutage werden immer mehr Systeme von Software kontrolliert. Dies macht Software Engineering zu einer der bedeutendsten Technologien, welche ebenfalls eine sehr schnelle Entwicklung macht [?]. 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 [?]. Software Engineering besteht aus mehreren Schichten (Abbildung 2.1):

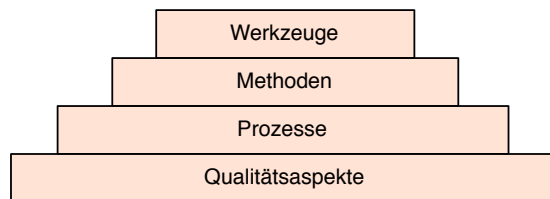


Abbildung 2.1.: Schichten des Software Engineering [?]

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

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 somit

ihre Anforderungen erst während der Entwicklung. Somit muss man mit Änderungen der Anforderungen an ein System während dessen Entwicklung rechnen. Aus diesem Grund ist es wichtig Ziele beim Software Engineering zu haben, um die Auswirkungen solche Änderungen einzudämmen [?]. Abbildung 2.2 zeigt die von [?] definierten Ziele, Prinzipien und den Prozess des Software Engineering, welche nachfolgend genauer erläutert werden:

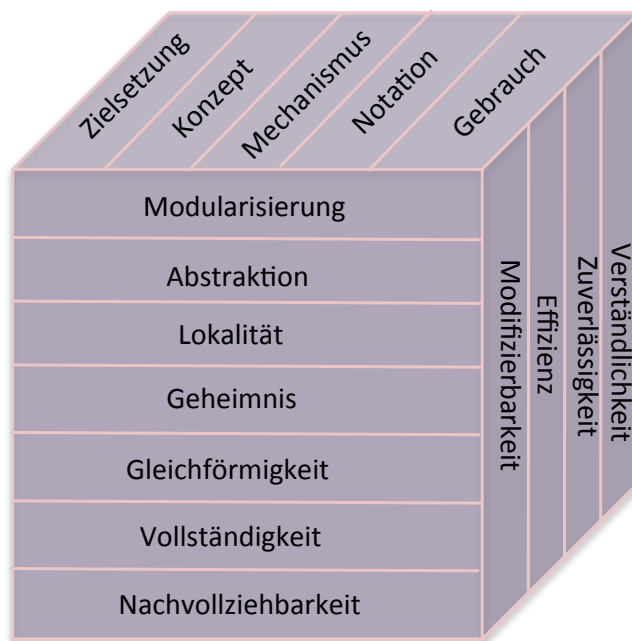


Abbildung 2.2.: Ziele, Prozess und Prinzipien des Software Engineering [?]

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. Beim *Geheimnisprinzip* geht es darum, dass unwesentliche Informationen unzugänglich gemacht werden. Es bezieht sich somit auf das Definieren und Durchsetzen von Zugriffsbeschränkungen. Das *Lokalitätsprinzip*

2. Prozessmodelle

verlangt das räumlich zusammenhängende Ablegen von zusammengehörenden Informationen. Konsistenz wird durch das *Gleichförmigkeitsprinzip* gewährleistet. Durch 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 [?].

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 festgelegt. 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 festgelegt, 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 Benutzerhandbuches, beschrieben werden [?].

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.

2.2. Softwareprozessmodelle

Für das Verständnis, die Schaffung oder Unternehmung etwas Großen 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 [?].

Hierbei stützten sie sich auf Modelle, welche als Stellvertreter für das was verstanden, geschaffen, unternommen oder betrieben werden soll, ansehen kann [?].

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 [?].

Modell leitet sich von dem lateinischen Begriff „modelus“ ab und kann mit „Regel, Form, Muster, Vorbild“ übersetzt werden [?]. Der Begriff Prozess stammt von dem lateinischen Wort "processus" ab und lässt sich mit "Fortgang oder Verlauf" übersetzen [?, ?].

Ein Softwareprozess stellt eine Abfolge von Schritten dar, welche zur Herstellung von Software notwendig sind [?, ?]. Mit Hilfe eines Softwareprozessmodelles lässt sich der organisatorische Rahmen zur Herstellung von Software beschreiben [?]. Ein Softwareprozessmodell stellt somit ein Modell für die Entwicklung eines Software-Systems dar [?]. Die einzelnen Abschnitte eines Softwareprozesses werden hierbei als Phasen bezeichnet [?]. Diese werden unterscheiden 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 die während des Entwicklungsprozesses zu erstellenden Dokumente und Unterlagen festgelegt [?].

2. Prozessmodelle

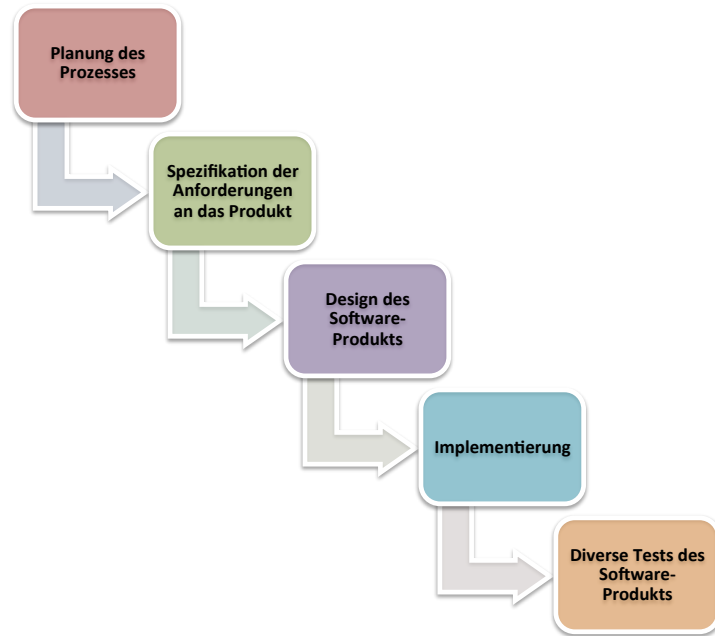


Abbildung 2.3.: Phasen Softwareprozess nach [?]

2.2.1. Software-Projekttypen

Software-Projekte lassen sich in drei Gruppen einteilen (Abbildung 6.38):

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 Erfahrung 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 [?, ?].

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

Eine Schnittstelle zwischen diesen beiden Projekttypen bilden die *Mittelschweren Pro-*

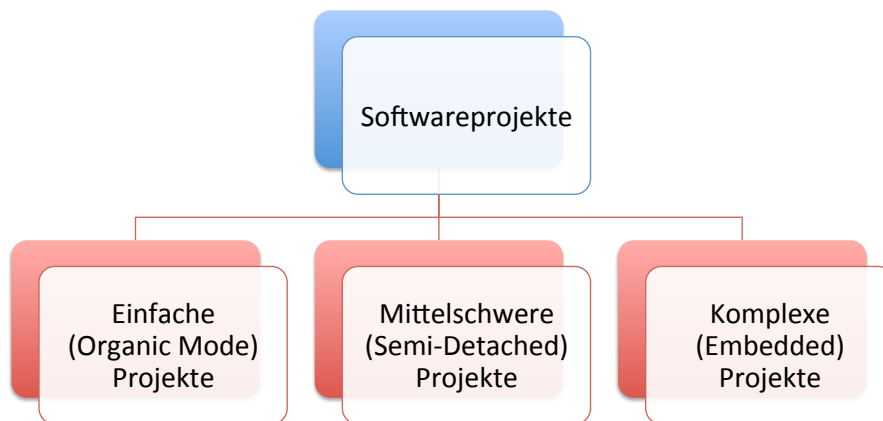


Abbildung 2.4.: Software-Projekttypen nach [?]

jekte. Hier sind die Software-Entwicklungsteams mittelgroß und bestehen aus erfahrenen und unerfahrenen Mitgliedern. Teilweise sind nicht alle Aspekte des Produktes schon im Vorherein bekannt und die Anzahl der Code-Zeilen ist groß [?, ?].

2.2.2. Schwergewichtige und leichtgewichtige Prozessmodelle

Aus der eben erfolgten Einteilung von Software-Projekten lässt sich eine Einteilung von Software-Prozessmodellen in *leichtgewichtige* und *schwergewichtige Prozessmodelle* ableiten [?].

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*[?].

Eine sehr formale und dokumentenlastige Vorgehensweise kommt bei den *schwergewichtigen Prozessmodellen* zum Einsatz. Es findet eine ausführliche Dokumentation in allen Entwicklungsphasen statt und der Ablauf des Prozesses ist genau vorgegeben.

2. Prozessmodelle

Bei Software-Produkten, welche bei einer möglichen Fehlfunktion Menschenleben in Gefahr bringen ist beispielsweise eine Vorgehensweise mit einem *schwergewichtigen Prozessmodell* sinnvoll. Ihr Einsatz ist besonders in *schweren Projekten* sinnvoll, aber auch in *mittelschweren Projekten* [?].

3

Modellierung

Kapitel 3 liefert einen Überblick über die Grundlagen der Modellierung. Zunächst werden in Kapitel 3.1 die Grundlagen der Prozessmodellierung vorgestellt. Hierbei wird auf die Grundsätze ordnungsgemäßer Modellierung eingegangen. Anschließend werden in Kapitel 3.2 Prozessmodellierungssprachen vorgestellt. Einerseits wird hier auf imperative Modellierungssprachen eingegangen und es wird ein kurzer Einblick in die Prozessmodellierungssprache BPMN gegeben. Andererseits werden deklarative Prozessmodellierungssprachen vorgestellt und es erfolgt ein detaillierter Einblick in die deklarative Prozessmodellierungssprache ConDec.

3. Modellierung

3.1. Prozessmodellierung

Prozessmodellierung hat den Zweck, Prozesse zu beschreiben [?]. 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 [?].

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.



Abbildung 3.1.: Ziele der Prozessmodellierung nach [?]

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 Kundenzufriedenheit 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.

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 [?].

Der *Grundsatz der Richtigkeit* besitzt zwei verschiedene Ausprägungen: Eine syntaktische und eine semantische. 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 korrekte Abbildung der Struktur des Metamodells, als auch des dort beschriebenen Verhaltens geachtet werden [?, ?].

Die syntaktische *Richtigkeit* eines Modells wird durch die Einhaltung der Notationsregeln der dem Modell zugrunde liegenden Prozessmodellierungssprache erreicht [?, ?].

3. Modellierung

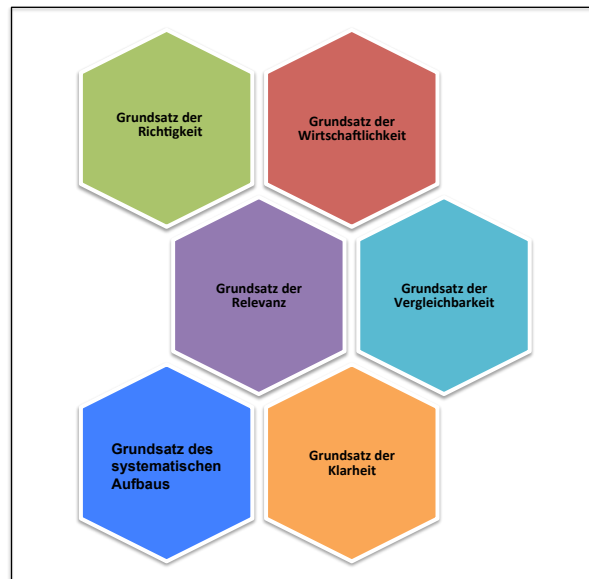


Abbildung 3.2.: Grundsatz ordnungsgemäßer Modellierung nach [?]

Modelle werden üblicherweise in getrennten Sichten modelliert, um die Komplexität so gering wie möglich zu halten. Z.B. werden die 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 Aufbaus* 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 Inputdaten eines Prozessmodells einen Verweis auf bestehende Datenmodelle enthalten [?, ?, ?, ?].

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 [?, ?]. Auf der anderen Seite sollen 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 [?, ?, ?].

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

Durch den *Grundsatz der Klarheit* soll sichergestellt werden, dass das Modell für den Adressaten verständlich ist. Es muss also auf 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, hervorgerufen durch parallele Gateways und XOR Gateways wirkt sich negativ auf die Verständlichkeit von Prozessmodellen aus. Ebenso allgemein, die Anzahl von Verbindungen zwischen Aktivitäten. [?, ?, ?, ?, ?, ?, ?].

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. [?, ?, ?, ?].

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

3. Modellierung

Prozessmodelle mit mathematischen Modellen und Formalismen führt 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 [?, ?].

Hierfür existierten eine Reihe verschiedener Prozessmodellierungssprachen, deren Vor- und Nachteile stark diskutiert werden. Der derzeit am meisten diskutierte Unterschied ist der zwischen imperativen und deklarativen Prozessmodellierungssprachen [?].

Die ursprüngliche Unterscheidung zwischen imperativen und deklarativen Sprachen stammt aus der Programmierung. Während imperative Programmierung angibt, "Wie etwas zu tun ist", folgt deklarative Programmierung dem Ansatz "sag was benötigt wird und lass das System herausfinden, wie es erreicht werden kann"[?].

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 [?].

Ähnlich wie die imperative Programmierung, folgt auch die imperative Modellierung einem "Inside-Out-Ansatz". Alle Ausführungsalternativen eines Prozesses sind somit im Prozess 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 [?]. 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 [?]. Die BPMN-Elemente

lassen sich anhand der fünf Kategorien *Flussobjekte*, *verbindende Objekte*, *Daten*, *Artefakte* und *Swimlanes* einteilen. Abbildung 3.3 zeigt die Einteilung und die wichtigsten Prozess-Elemente von BPMN, welche nachfolgend genauer erläutert werden [?].

In der Kategorie **Swimlanes** befinden sich *Pools* und *Lanes*. *Pools* stellen eine Art Container für den Prozess dar. Ein *Pool* stellt einen Prozessteilnehmer dar. 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 *Lanes* unterteilt werden. *Lanes* können untergeordnete Organisationseinheiten, Partnerrollen (z.B. Vertrieb, Projektleitung, Marketing), oder auch verschiedene Bestandteile eines Systems sein [?, ?, ?].

Aktivitäten, *Ereignisse* 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. Weiterhin gibt es auch noch *Zwischenereignisse*.

Aktivitäten stellen Arbeitseinheiten dar und stellen einen Oberbegriff für Aufgaben, Unterprozesse und Aufruf-Aktivitäten dar. Beschriftet werden sie mit einer Objekt-Verb-Verbindung (z.B. Lieferung überprüfen) [?].

Mit Hilfe von *Gateways* lässt sich der Prozessablauf kontrollieren und steuern, da durch diese Verzweigungen und Zusammenführungen von Sequenzflüssen dargestellt werden. [?, ?]. 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 [?].

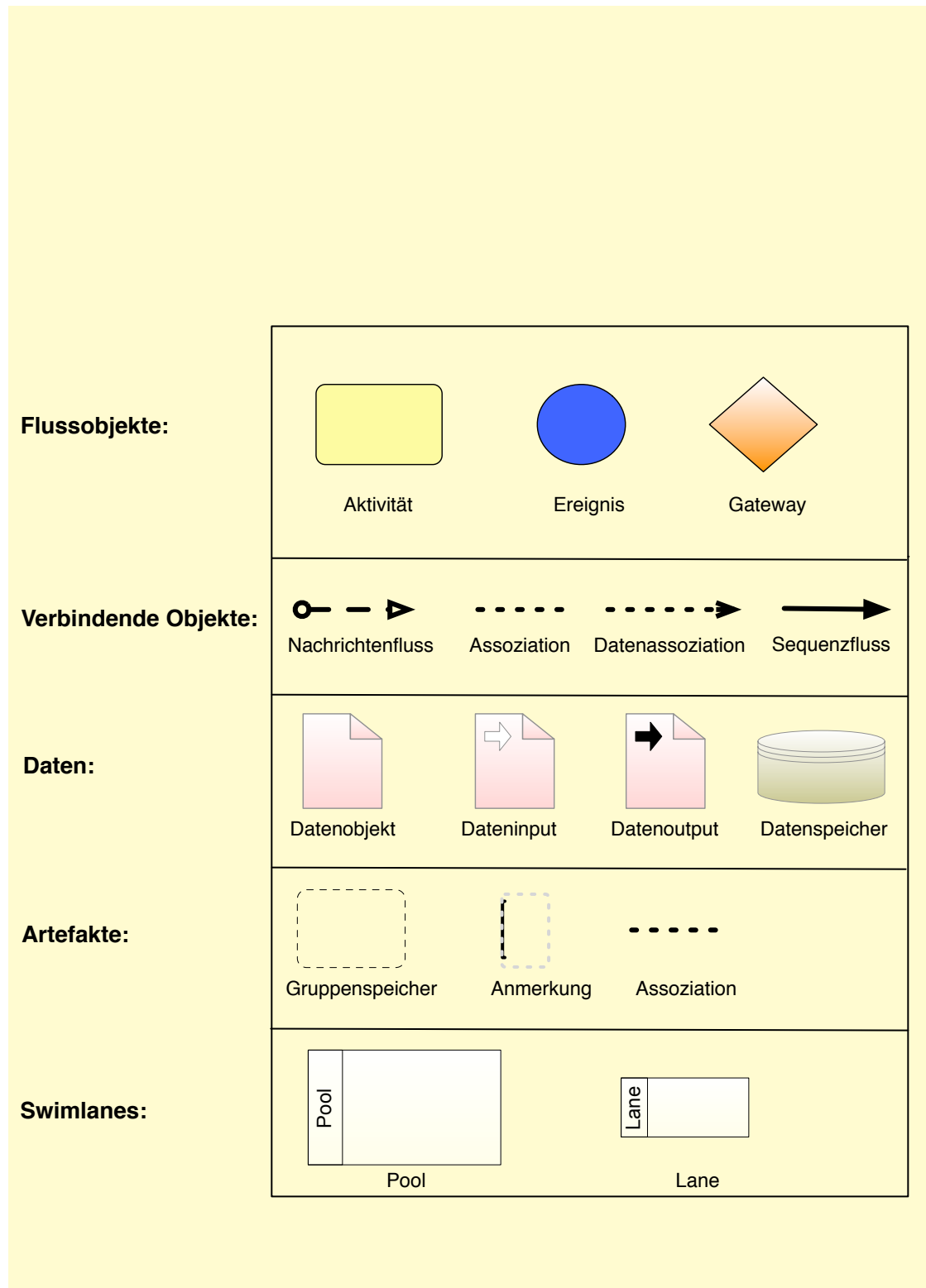


Abbildung 3.3.: BPMN-Elemente Übersicht nach [?]

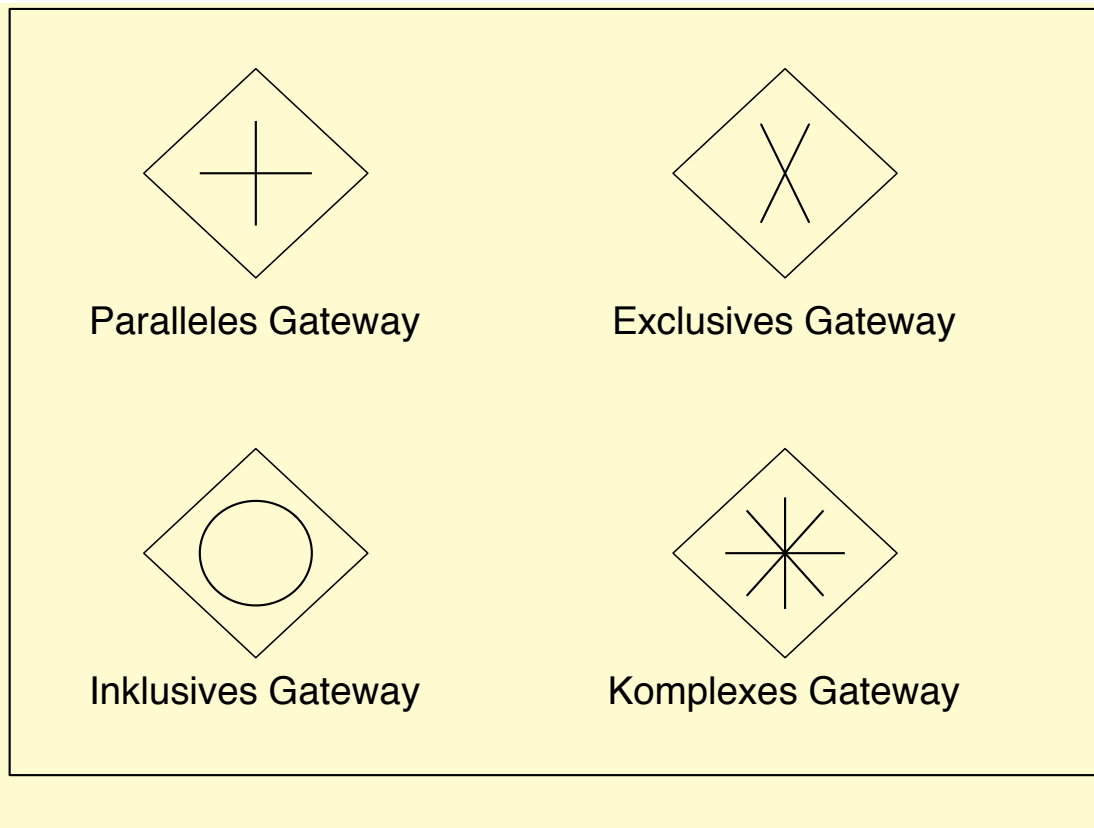


Abbildung 3.4.: BPMN-Gateways

Nachrichtenfluss, *Assoziation*, *Datenassoziation* und *Sequenzfluss* bilden zusammen die Kategorie **Verbindende Objekte**. Ein *Sequenzfluss* dient dazu die Reihenfolge der Aktivitäten im Prozess darzustellen. 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 Outputs von Aktivitäten aufgezeigt [?].

In der Kategorie **Daten** befinden sich *Datenobjekte*, *DatenInput*, *DatenOutput* und *Datenspeicher*. *Datenobjekte* geben hierbei an, welche Daten von den Aktivitäten benötigt werden, bzw. von diesen erzeugt werden [?]. Sie stellen somit Information dar, welche durch den Prozess fließen. Bei einem *DatenInput* handelt es sich um einen externen

3. Modellierung

Input 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 [?]. 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 [?]. Die Kategorie **Artefakte** beinhaltet *Gruppierung*, *Anmerkung* und *Assoziation*. Diese ergänzen den Prozess um zusätzliche Informationen, haben jedoch keinerlei Einfluss auf diesen [?]. Eine *Gruppierung* kann hierbei zur Dokumentation oder für Analysezwecke benutzt werden. Durch eine *Annotation* können dem Leser zusätzliche Informationen in Textform bereit gestellt werden [?].

3.2.2. Deklarative Modellierung

Die deklarative Modellierung folgt im Gegensatz zur imperativen Modellierung einem "Outside-In-Ansatz"[?]. Deklarative Sprachen legen den Ablauf nicht im Vorhinein fest. [?]. Sie sind somit sehr flexibel [?]. Zu Beginn befinden sich nur die Aktivitäten im Prozessmodell und erlauben jegliches Ausführungsverhalten. Erst wenn Constraints zum Modell hinzugefügt werden, werden schrittweise Ausführungsalternativen verworfen [?]. 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 oder eine Mindestzeitverzögerung zwischen zwei Aktivitäten handeln. Terminierungsconstraints hingegen geben an, wann eine korrekte Terminierung möglich ist. Z.B. kann hier vorgeschrieben werden, dass eine Aktivität mindestens einmal ausgeführt werden muss oder dass der Aktivität A Aktivität B folgen muss [?]. 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 bei Aktivität C legt fest, dass diese mindestens einmal ausgeführt werden muss, aber beliebig oft ausgeführt werden kann und das Constraint zwischen A und B legt fest, dass

Aktivität B Aktivität A vorausgehen muss. Abgesehen von diesen Bedingungen, können die Aktivitäten sowohl beliebig oft, als auch in beliebiger Reihenfolge ausgeführt werden.

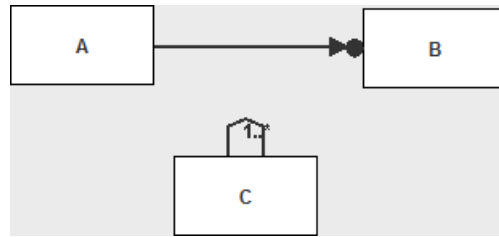


Abbildung 3.5.: Deklarativer Beispiel-Prozess

ConDec

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

In ConDec gibt es die vier verschiedenen 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 Abhang A verfügbar.

3. Modellierung

Constraint	Erläuterung
Existente 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

4

Modellierungswerkzeuge

Dieses Kapitel stellt die in dieser Arbeit für die Prozessmodellierung benutzten Modellierungswerkzeuge vor. Nach einer kurzen allgemeinen Einführung in Modellierungswerkzeuge, wird das Modellierungswerkzeug Signavio vorgestellt, welches zur imperativen Modellierung von Prozessen mit BPMN in dieser Arbeit herangezogen wird. Anschließend erfolgt eine Einführung in das Modellierungswerkzeug Declare, mit welchem die deklarativen Prozessmodelle in der Prozessmodellierungssprache ConDec in der vorliegenden Arbeit erstellt werden.

4.1. Modellierungswerkzeuge

Ein Modellierungswerkzeug ist ein Softwaresystem, mit dessen Hilfe sich Prozessmodelle erstellen, ausführen und monitoren lassen. Teilweise bietet ein Modellierungswerkzeug

4. Modellierungswerkzeuge

noch weitere Funktionen wie z.B. 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 [?].

4.1.1. Signavio

Bei Signavio handelt es sich um ein webbasiertes Prozessmodellierungstool, welches auch das kollaborative Modellieren von Prozessen mit den Modellierungsstandards BPMN und EPC zulässt. Der 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 das Modellierungswerkzeug Signavio mitgeladen und kann somit im Web-Browser ausgeführt werden [?].

Abbildung 4.1 zeigt den *Signavio Process Editor*.

4.1. Modellierungswerkzeuge

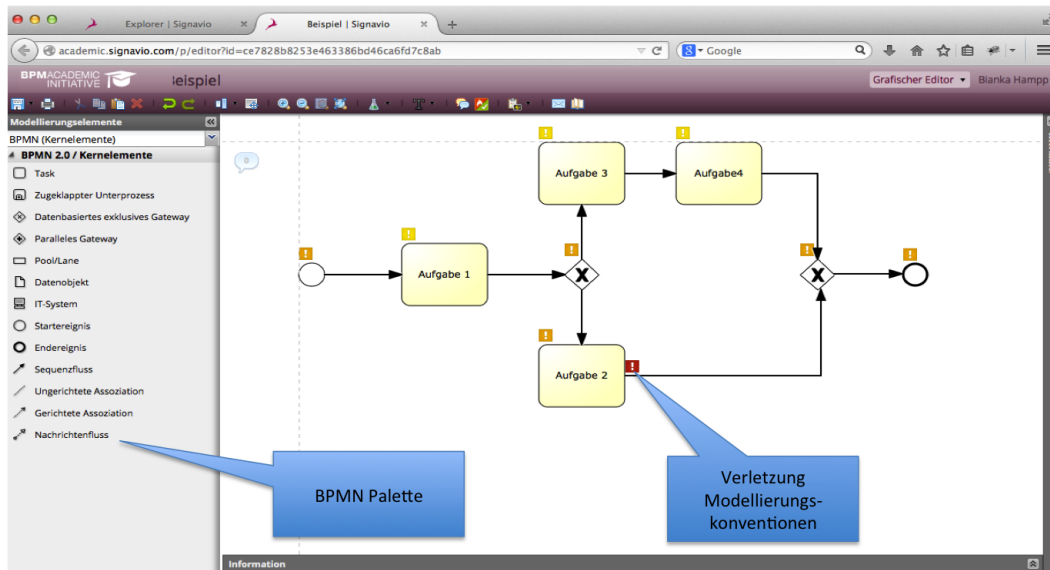


Abbildung 4.1.: Signavio Process Editor

Links 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 und Diagrammlayout zu überprüfen. Die Modelle können alle als PDF exportiert werden.

In Abbildung 4.2 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 wobei XOR-Verzweigungen nach wie vor vom Benutzer ausgewählt werden müssen.

4. Modellierungswerkzeuge

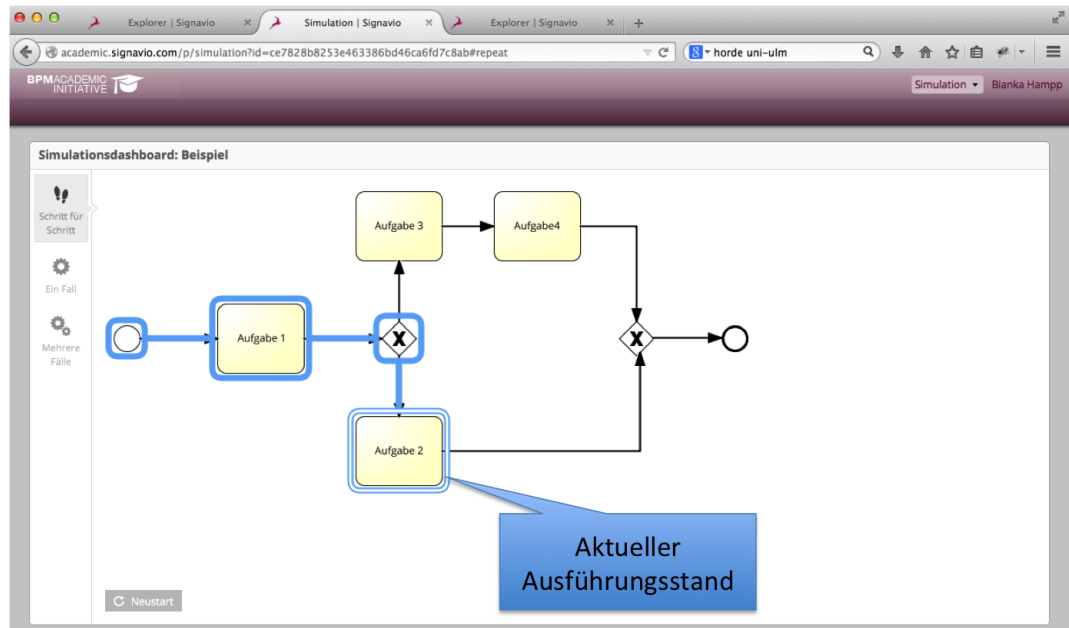


Abbildung 4.2.: Signavio Simulation

4.1.2. 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:

- Modellentwicklung
- Modellüberprüfung (Suche nach Fehlern in Modellen)
- automatisierte Modellausführung
- wechselnde Modelle zur Laufzeit
- Analyse der bereits durchgeführten Prozesse
- Prozess Dekomposition

Abbildung 4.3 zeigt die Systemarchitektur von Declare.

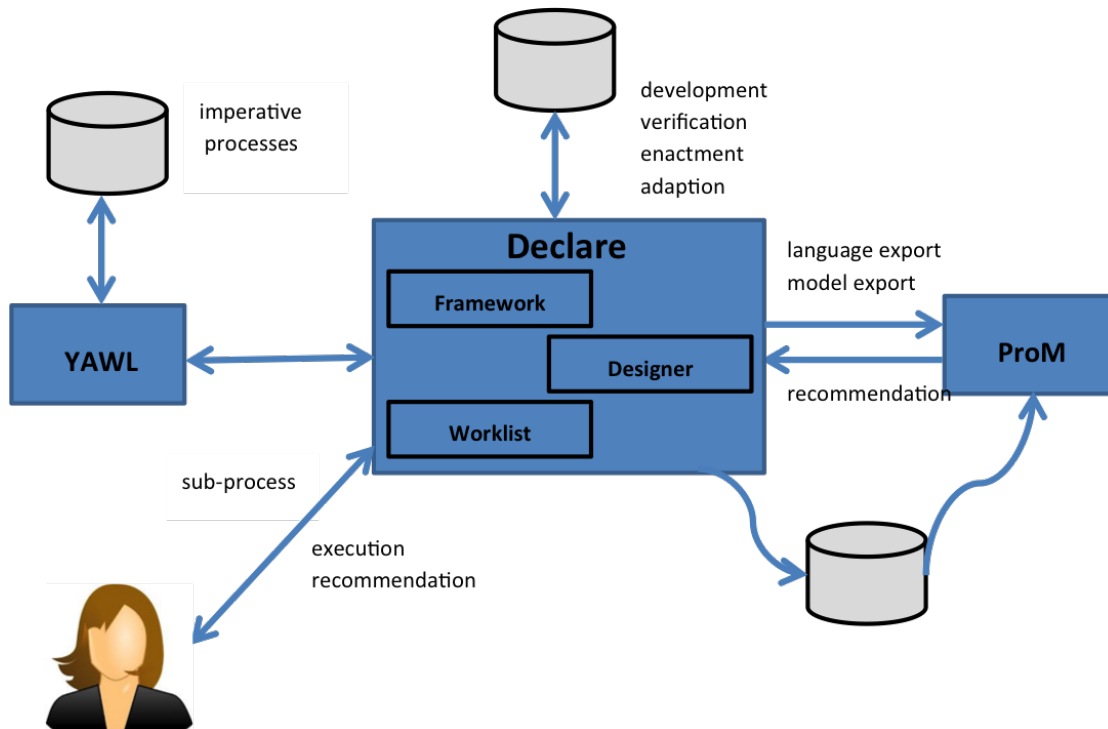


Abbildung 4.3.: Declare Systemarchitektur

Hieraus wird ersichtlich, dass *Declare* mit den beiden Systemen *YAWL* und *ProM* kooperiert. Bei *YAWL* handelt es sich hierbei 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 [?].

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 4.4). Das *Framework* ist für das Prozess-Enactment zuständig. Außerdem übernimmt es die

4. Modellierungswerkzeuge

Kommunikation mit *YAWL* und *ProM* und das Ändern der Prozessmodelle zur Laufzeit (Abbildung 4.5). Die Prozessausführung wird von *Worklist* durchgeführt. Hier können die Nutzer ihre zuvor erstellten Prozesse ausführen und können die von *ProM* erstellten Empfehlungen sehen (Abbildung 4.6). Alle Modelle können als Bilddateien exportiert werden.

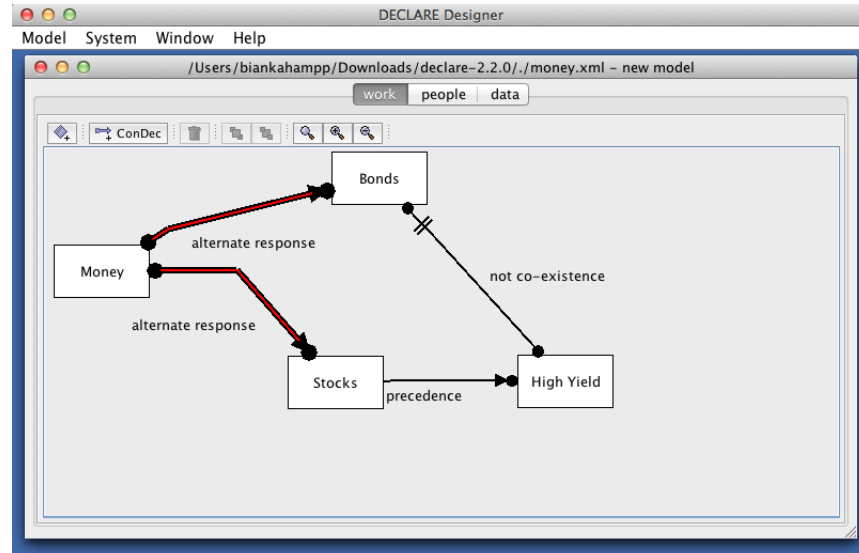


Abbildung 4.4.: Declare Designer

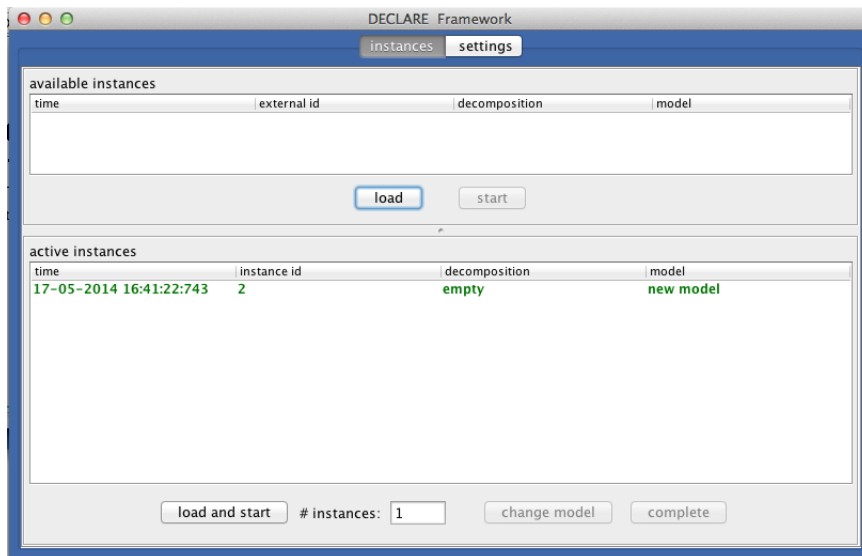


Abbildung 4.5.: Declare Framework

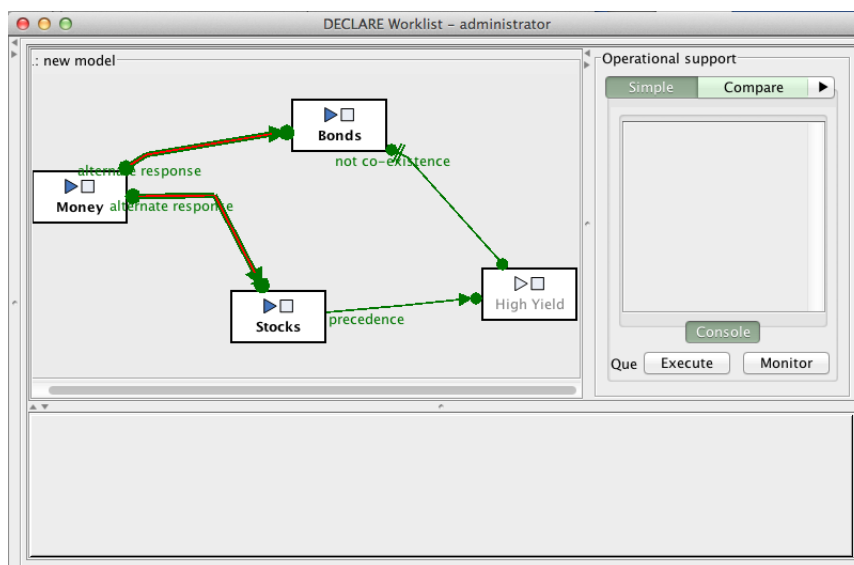


Abbildung 4.6.: Declare Worklist

5

Anforderungserhebung

In diesem Kapitel werden die Anforderungen an den in Kapitel 6 folgenden Vergleich der imperativen und deklarativen Modellierung für SE-Prozessmodelle erhoben.

5.1. 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 zwischen deklarativen und imperativen Prozessmodellierungssprachen im Kontext von Software-Engineering Prozessmodellen durchgeführt werden. Hierbei sollen die **Unterschiede und Ähnlichkeiten** der beiden Prozessmodellierungssprachen herausgearbeitet und deren **Eignung für die Modellierung von unterschiedlich großen**

5. Anforderungserhebung

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

Hierfür sollen die imperativen und deklarativen Prozessmodelle, welche für die drei Software-Engineering Prozessmodelle Scrum, Open Up und V-Modell-XT erstellt werden im Hinblick auf verschiedene Vergleichskriterien untersucht werden. Da es sich bei Scrum um ein leichtgewichtiges Software-Engineering Prozessmodell, beim V-Modell XT um ein schwergewichtiges Software-Engineering Prozessmodell und bei Open Up um ein Software-Engineering Prozessmodell 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 Prozessmodelle. 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 [?].

5.1.1. Erfüllung der Modellierungsgrundsätze

Zum einen 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 [?]. Hierbei werden die erstellten Prozessmodelle auf *Klarheit, Richtigkeit, Wirtschaftlichkeit, Relevanz, Vergleichbarkeit* und *systematischen Aufbau* verglichen.

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 Prozessmodel-

lierungssprache erstellt wurden. Denn fehlende Verständlichkeit eines Prozessmodells führt dazu, dass das Prozessmodell wenig Nutzen bringt. Insbesondere soll hier auch die Anzahl an Verbindungen zwischen Aktivitäten verglichen werden, da sich diese mit steigender Anzahl negativ auf die Verständlichkeit auswirken.

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 Software-Engineering Prozessmodells. 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 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.

Richtigkeit

Die Richtigkeit der Prozessmodelle soll ebenfalls verglichen werden. Hierbei soll die semantische und syntaktische Richtigkeit der Prozessmodelle untersucht werden. 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 beschriebene Verhalten besser abbildet, als die andere. Insbesondere wird hier untersucht, ob es Grenzen in der Darstellbarkeit der abzubildenden Aspekte des Metamodells gibt. Hierbei soll verglichen werden, wie gravierend sich diese Grenzen der Darstellbarkeit auf die Verständlichkeit des Prozessmodells auswirken.

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 soll auch untersucht werden, ob sich die Aufwände für die Modellierung zwischen den beiden Prozessmodellierungssprachen in Abhängigkeit der Größe des zugrunde liegenden Metamodells wesentlich voneinander unterscheiden.

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.

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. Außerdem wird hier die Größe der jeweiligen Prozessmodelle als Vergleichskriterium herangezogen. Hier wird

die Anzahl 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.

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 Modellierung gezogen werden und eventuelle Grenzen der Prozessmodellierungssprache aufgezeigt werden.

5.1.2. Weitere Vergleichskriterien

Wartbarkeit

Bei diesem Kriterium soll untersucht werden, ob sich die Prozessmodelle, welche mit einer imperativen, bzw. deklarativen Prozessmodellierungssprache erstellt wurden, im Hinblick auf ihre Wartbarkeit unterscheiden. Hierbei soll festgestellt werden, ob es bei einer der Modellierungssprachen erhebliche Schwierigkeiten in Bezug auf die Änderung/Anpassung von existierenden Prozessmodellen gibt.

6

Imperative und deklarative Modellierung für SE-Prozessmodelle

In diesem Kapitel wird der Vergleich zwischen imperativer und deklarativer Modellierung für SE-Prozessmodelle durchgeführt. Als Erstes wird dieser Vergleich in Kapitel 6.1 für das SE-Prozessmodelle Scrum durchgeführt. Hierfür wird zunächst in Kapitel 6.1 das der Modellierung zugrunde liegende Modell, das SE-Prozessmodell Scrum, vorgestellt und in Kapitel 6.1.1 für die Modellierung analysiert. Danach erfolgt in Kapitel 6.1.2 die imperative Modellierung von in der Prozessmodellierungssprache BPMN und anschließend die deklarative Modellierung in der Prozessmodellierungssprache ConDec in Kapitel 6.1.3. Danach erfolgt in Kapitel 6.1.4 der Vergleich zwischen den beiden Modellen.

Der zweite SE-Prozess, welcher in diesem Kapitel in imperativer und deklarativer Prozessmodellierungssprache verglichen werden soll, ist der Open Unified Process (Open Up). Auch hier erfolgt zunächst eine kurze Einführung in den Open Up in Kapitel 6.2,

6. Imperative und deklarative Modellierung für SE-Prozessmodelle

bevor dieser in Kapitel 6.2.1 analysiert wird, damit er in den Kapiteln 6.2.2 und 6.2.3 in imperativer, bzw. deklarativer Prozessmodellierungssprache modelliert werden kann. Hiernach erfolgt in Kapitel 6.2.4 der Vergleich zwischen den Prozessmodellen.

Zuletzt wird noch das V-Modell XT modelliert und verglichen. Eine Einführung in das V-Modell XT erfolgt in Kapitel 6.3. In Kapitel 6.3.1 wird dieses als Vorbereitung für die Modellierung analysiert und in den Kapiteln 6.3.2 und 6.3.3 in imperativer und deklarativer Prozessmodellierungssprache modelliert. Der Vergleich hierzu erfolgt in Kapitel 6.3.4.

6.1. 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 [?, ?].

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 [?, ?]. Einen ersten Überblick über das Scrum-Prozessmodell gibt Abbildung 6.1:

Der genaue Ablauf im Scrum Prozessmodell wird nachfolgend genau analysiert.

6.1.1. 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 [?].

Der *Product Owner* ist verantwortlich, die Interessen aller am Projekt beteiligten Personen zu vertreten. Neben der Budgetierung des Projektes erstellt er ebenfalls Re-

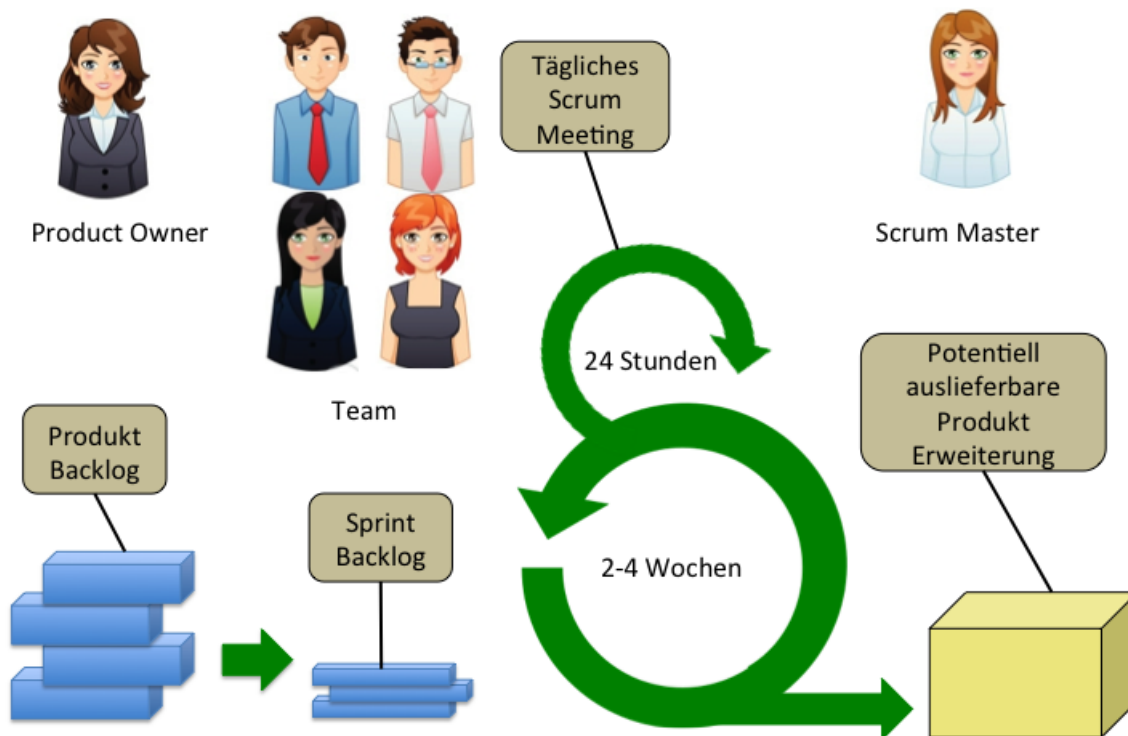


Abbildung 6.1.: Scrum Überblick nach [?]

6. Imperative und deklarative Modellierung für SE-Prozessmodelle

leasepläne und erstellt den Produkt-Backlog, welcher eine Liste mit funktionalen und nicht-funktionalen Anforderungen darstellt [?, ?, ?]. 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 [?, ?, ?]. Der *Product Owner* ist ebenfalls zuständig für das Annehmen, bzw. Ablehnen der Arbeitsergebnisse [?].

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 [?, ?]. Alle Teammitglieder sind gemeinsam für den Erfolg eines jeden *Sprints* und des gesamten Projektes verantwortlich [?].

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 [?]. 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 [?]. Die Dauer eines Sprints beträgt zwei bis vier Wochen. Am Ende eines jeden Sprints muss das *Team* ein lauffähiges Produkt abliefern [?]. Vor jedem Sprint findet ein *Sprint Planning Meeting* statt, welches sich aus zwei Teilen zusammensetzt [?]. Im ersten Teil findet eine Planung des nächsten *Sprints* statt [?]. Hierfür präsentiert der *Product Owner* dem *Team* eine Liste der Product-Backlog-Elemente mit der aktuell höchsten Priorität [?, ?, ?]. Diese Liste wird *Sprint-Backlog* genannt [?]. 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 [?]. Das *Team* kann sich die Aufgaben eigenverantwortlich aufteilen und muss sich anschließend dem *Product Owner* verpflichten.

ten, die *Tasks* bis zum Abschluss des *Sprints* zu erledigen [?, ?, ?]. Das *Team* trifft sich während des *Sprints* täglich in einem 15-minütigen Meeting, dem *täglichem Scrum-Meeting*. Hier redet das *Team* über seinen Fortschritt und eventuelle Probleme bei ihrer Arbeit [?]. Hier muss jedes Teammitglied die nachfolgenden drei Fragen beantworten [?]:

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

6.1.2. Imperative Modellierung Scrum

Abbildung 6.2 zeigt die imperative Modellierung von Scrum. Parallel zu allen anderen Aktivitäten des Teams und des Product Owners muss der Scrum-Master stets den Scrum-Prozess managen.

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 dargestellt ist, so wird zuerst das Sprint Planning Meeting durchgeführt. Dies ist hier als Unterprozess in Abbildung 6.3 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 6.2) statt 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 XOR-Gateway immer wieder zurück zu 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

6. Imperative und deklarative Modellierung für SE-Prozessmodelle

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.

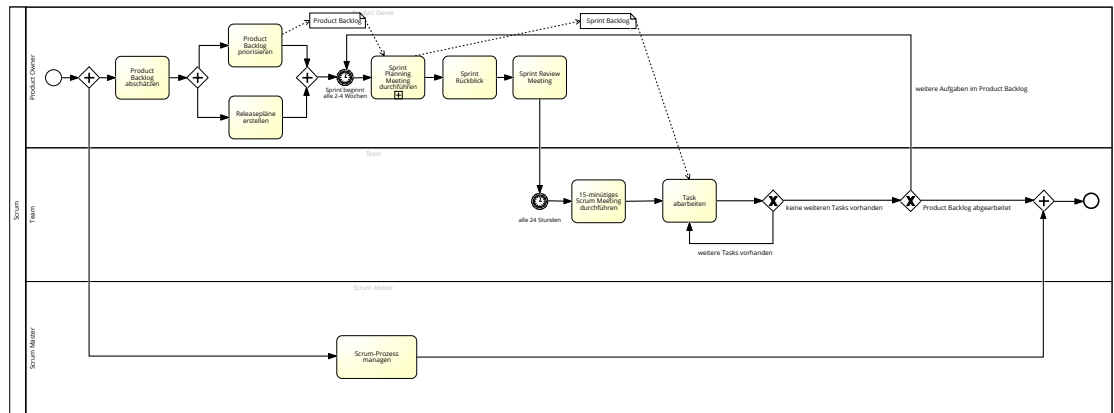


Abbildung 6.2.: Imperative Modellierung Scrum

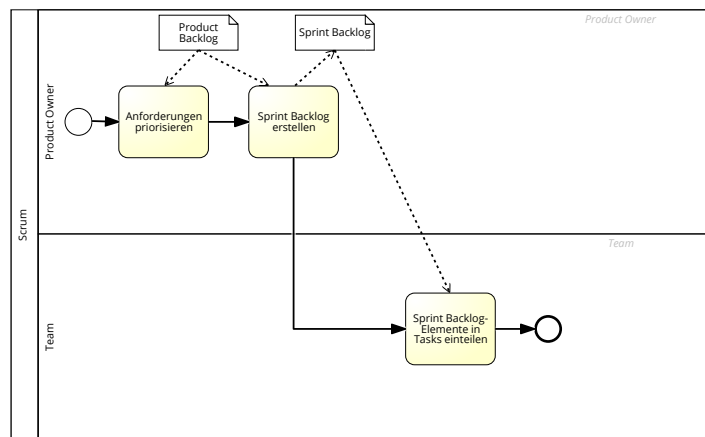


Abbildung 6.3.: Imperative Modellierung Scrum Unterprozess

6.1.3. Deklarative Modellierung Scrum

Abbildung 6.4 zeigt die deklarative Modellierung von Scrum.

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

Nach deren Ausführung muss die Aktivität *alle 2-4 Wochen Sprint Planning Meeting durchfuehren* erfolgen. Der zugehörige Unterprozess ist in Abbildung 6.5 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.

6. Imperative und deklarative Modellierung für SE-Prozessmodelle

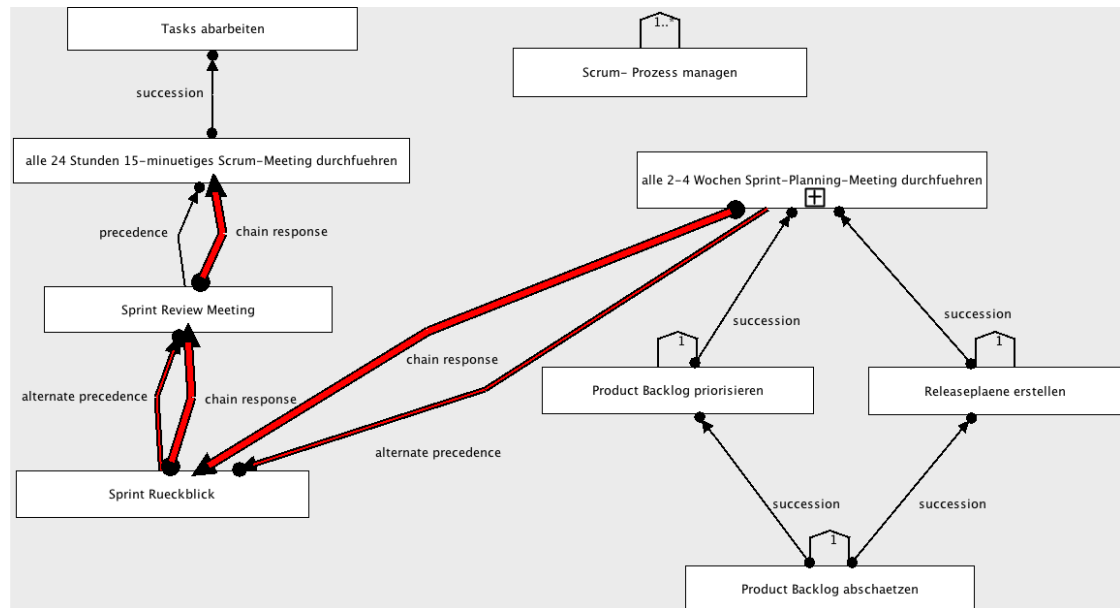


Abbildung 6.4.: Deklarative Modellierung Scrum

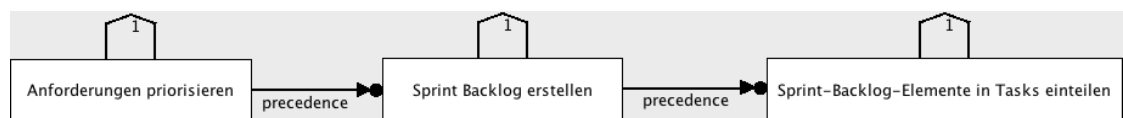


Abbildung 6.5.: Deklarative Modellierung Scrum-Unterprozess Sprint-Planning-Meeting durchfuehren

6.1.4. Vergleich

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

Wie Abbildung 6.6 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.

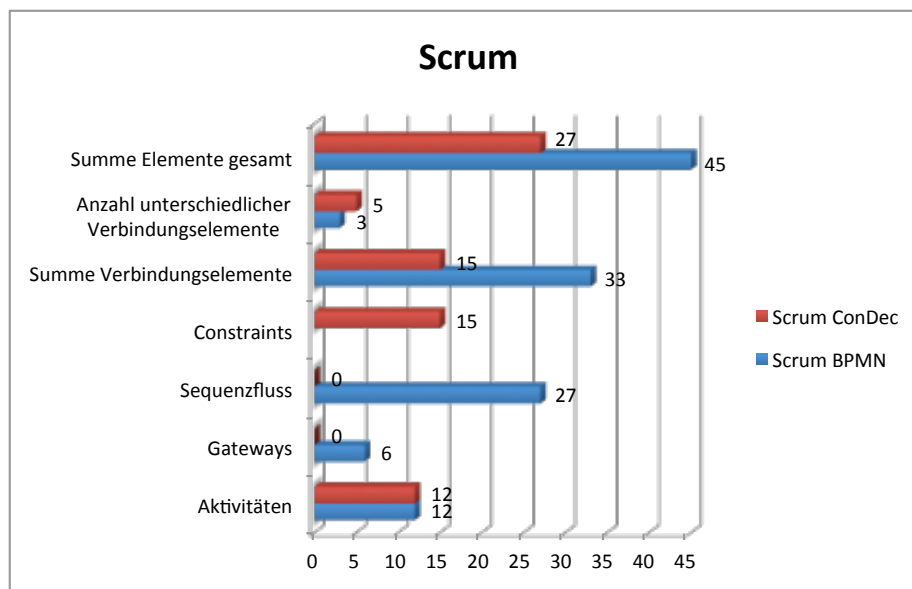


Abbildung 6.6.: Vergleich der Anzahl der Elemente Scrum

Die Anzahl der Verbindungselemente, welche zur Beschreibung des Ablaufes notwendig sind, unterscheiden sich jedoch, wie Abbildung 6.6 zeigt.

So werden in BPMN 6 Gateways und 27 Sequenzflusselmente benötigt, also insgesamt 33 Verbindungselemente um den Ablauf des Metamodells darzustellen. Bei Verwendung von ConDec werden 15 Verbindungselemente benötigt. In BPMN werden jedoch nur 3 verschiedene Verbindungselemente benötigt, während es in ConDec 5 verschiedene

6. Imperative und deklarative Modellierung für SE-Prozessmodelle

Verbindungselemente sind.

Weiterhin werden in BPMN 4 Artefakte zum Beschreiben des Metamodells benötigt, in ConDec gibt es jedoch keine Modellierungselemente hierfür, weshalb es hier im deklarativen Modell keine Artefakte gibt.

Genau wie bei den Artefakten gibt es in ConDec keine Möglichkeiten verschiedene Rollen innerhalb eines Prozessmodells darzustellen. Aus diesem Grund gibt es in BPMN 3 Rollen und in ConDec keine.

In Bezug auf die *Klarheit*, welche in Kapitel 5 als Anforderungskriterium festgelegt wurde, haben beide Modellierungssprachen Stärken und Schwächen. Die Anzahl der Aktivitäten unterscheidet sich nicht, jedoch die Anzahl der Flusselemente. Eine größere Anzahl an Verbindungselementen kann sich negativ auf die Verständlichkeit auswirken. Hier ist die Anzahl aber bei BPMN mehr als doppelt so hoch als bei ConDec. Die Anzahl verschiedener Verbindungselemente kann sich ebenfalls negativ auf die Verständlichkeit auswirken. In BPMN werden hier 3 und in ConDec 5 verschiedene Verbindungselemente benötigt. Damit ist die Anzahl hier bei ConDec nur etwas höher als bei BPMN.

Die *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 selbst abzubilden, müssen diese Informationen weggelassen werden. Die Rollen und Artefakte können zwar in Declare abgebildet werden, jedoch ist dies auf Papier/Bild nicht sichtbar. 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.

Wirtschaftlichkeit lässt sich mit beiden Modellierungssprachen einhalten. Die Aufwände zum Erstellen der Modelle unterscheiden sich in den beiden verwendeten Sprachen

nicht voneinander, da sie beide ungefähr die gleiche Größe haben.

Beide Prozessmodelle können mit minimal relevanten Informationen erstellt werden. Bei keiner der beiden Modellierungssprachen war es notwendig, weitere Informationen zum Modell hinzuzufügen, um dessen Qualität zu erhöhen. Somit kann *Relevanz* bei beiden Prozessmodellen eingehalten werden.

Nur BPMN bietet die Möglichkeit, Artefakte im Prozessmodell abzubilden 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.

Da bei BPMN mehr Verbindungselemente benötigt werden, weißt das mit BPMN erstellte Modell auch insgesamt mehr Elemente auf. Während bei ConDec insgesamt 27 Elemente benötigt werden, werden zur Darstellung des gleichen Sachverhaltes bei BPMN 45 Elemente benötigt. Die *Vergleichbarkeit* ist dadurch nicht ganz gewährleistet, da die Anzahl der Elemente bei BPMN fast doppelt so hoch ist wie bei ConDec.

6.2. 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 [?]. Er ist Teil des Eclipse Process Frameworks. Open Up ist ein iterativer, inkrementeller und minimaler Prozess, aber dennoch vollständig und erweiterbar [?, ?]. Der Prozess ist minimal gehalten, da 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 [?]. Das

6. Imperative und deklarative Modellierung für SE-Prozessmodelle

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 [?].

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 [?].

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 [?]. Einen ersten Überblick über Open Up gibt Abbildung 6.7.

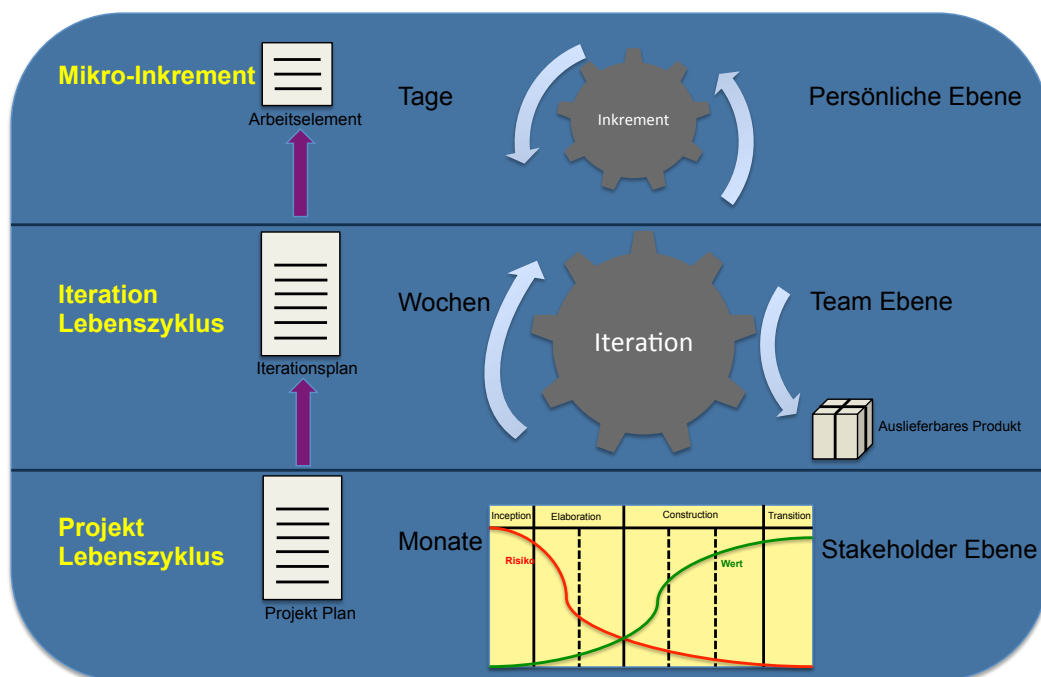


Abbildung 6.7.: Open Up Überblick nach [?]

Der Open UP wird im Folgenden analysiert.

6.2.1. Analyse Open UP

Auf der persönlichen Ebene teilen sich die Teammitglieder ihre Arbeit in *Mikro-Inkrement* 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 nachvollzogen werden. Die Teammitglieder teilen ihre Fortschritte täglich miteinander, was die Arbeitstransparenz und das Vertrauen erhöht und die Teamarbeit fördert [?].

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 [?]. In jeder Iteration wird ein Iterationsplan angefertigt, der vorgibt, was in dieser Iteration geliefert werden muss und auf welchen sich das Team verpflichten muss [?].

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 vier Phasen ein, über welche Abbildung 6.8 einen Überblick gibt [?].

In jeder Phase finden eine oder mehrere Iterationen statt und werden mit einem Meilenstein abgeschlossen [?]. Tabelle 6.1 zeigt die Abläufe in den Iterationen in den einzelnen Phasen und die zugehörigen Zielstellungen.

Vorlagenmodell Iterationen	Zielsetzung der Phase
----------------------------	-----------------------

6. Imperative und deklarative Modellierung für SE-Prozessmodelle

<p>Inception Phase Iteration</p> <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
<p>Elaboration Phase Iteration</p> <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

6.2. Open Unified Process (Open UP)

<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 6.1.: Iterationen und Zielstellungen der Phasen in Open UP [?]

Abbildung 6.9 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.

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 [?].

Der *Entwickler* entwickelt einen Teil des Systems und muss hierbei sicherstellen, dass

6. Imperative und deklarative Modellierung für SE-Prozessmodelle

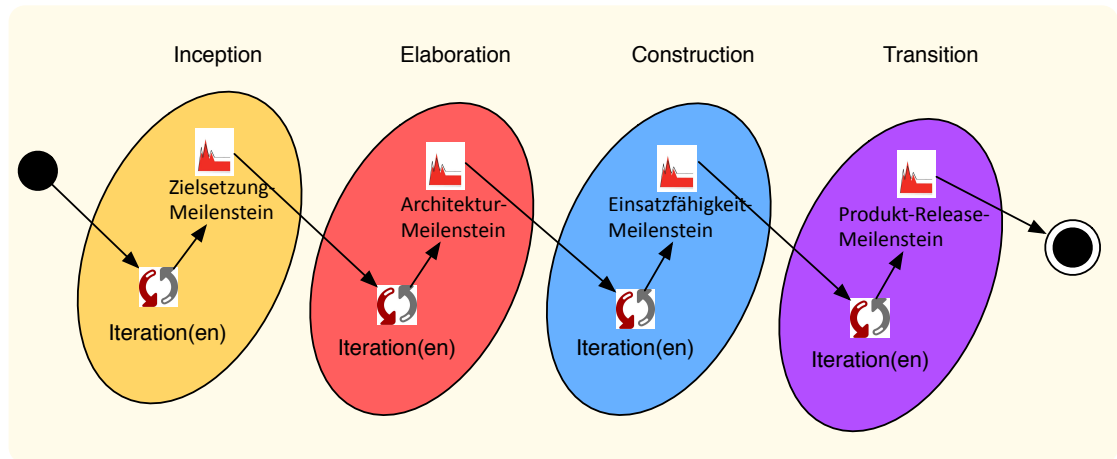


Abbildung 6.8.: Phasen Open UP nach [?]

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

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

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 [?].

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

Ein Artefakt ist etwas, das hergestellt, modifiziert oder durch eine Task verwendet wird. Rollen sind für die Erstellung und Aktualisierung von Artefakten verantwortlich. Artefakte

6.2. Open Unified Process (Open UP)

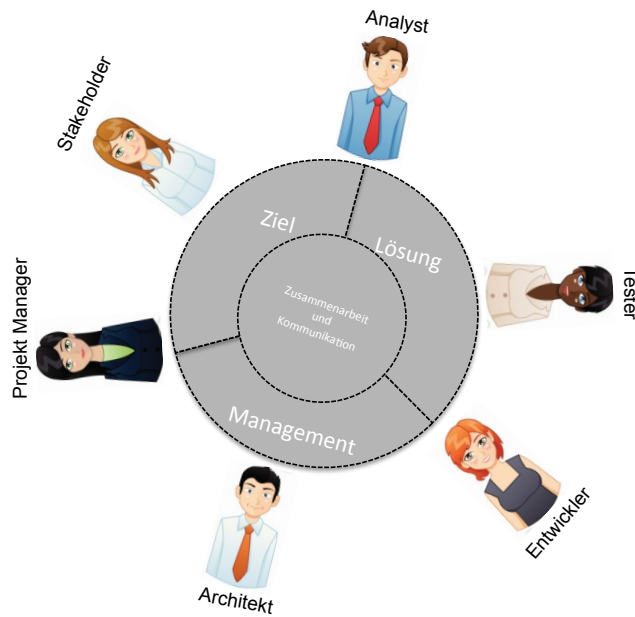


Abbildung 6.9.: Rollen in Open UP nach [?]

stellen eine Versionskontrolle während des gesamten Projektlebenszyklus dar. Die 17 Artefakte in OpenUP 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 [?].

6.2.2. Imperative Modellierung Open UP

Phasen Open UP

In Abbildung 6.10 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 zurück führt. Diese kann sodann erneut durchlaufen werden.

6. Imperative und deklarative Modellierung für SE-Prozessmodelle

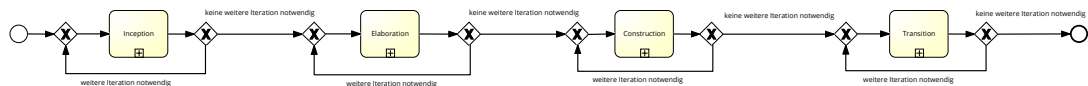


Abbildung 6.10.: Phasen Open UP- imperativ

Abbildung 6.11 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.

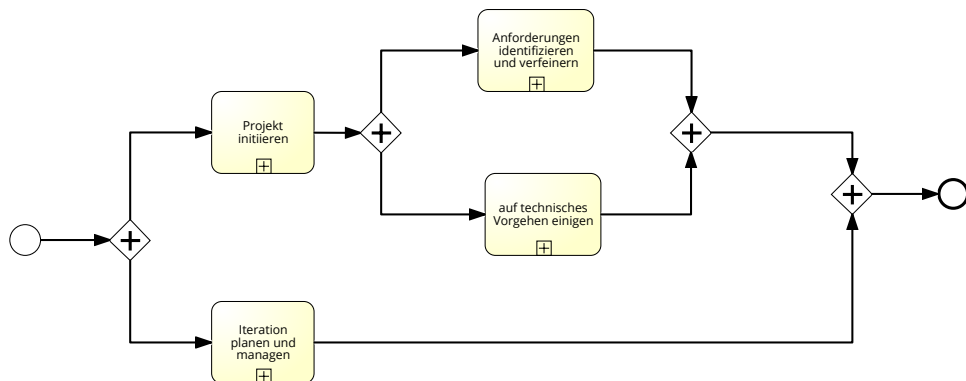


Abbildung 6.11.: Phasen Open UP Unterprozess Inception- imperativ

In Abbildung 6.12 ist die imperative Modellierung der Phase Elaboration abgebildet. Die sechs Aktivitäten *Anforderungen identifizieren und verfeinern*, *Architektur entwickeln*, *Lösungsincrement entwickeln*, *Lösung testen*, *Iteration planen und managen* sowie *weitere Aufgaben erledigen* werden parallel zueinander ausgeführt.

Die imperative Modellierung der Phase Construction kann Abbildung 6.13 entnommen werden. Hier werden die sechs Aktivitäten *Anforderungen identifizieren und verfeinern*, *Lösungsincrement entwickeln*, *Lösung testen*, *Iteration planen und managen*, *weitere*

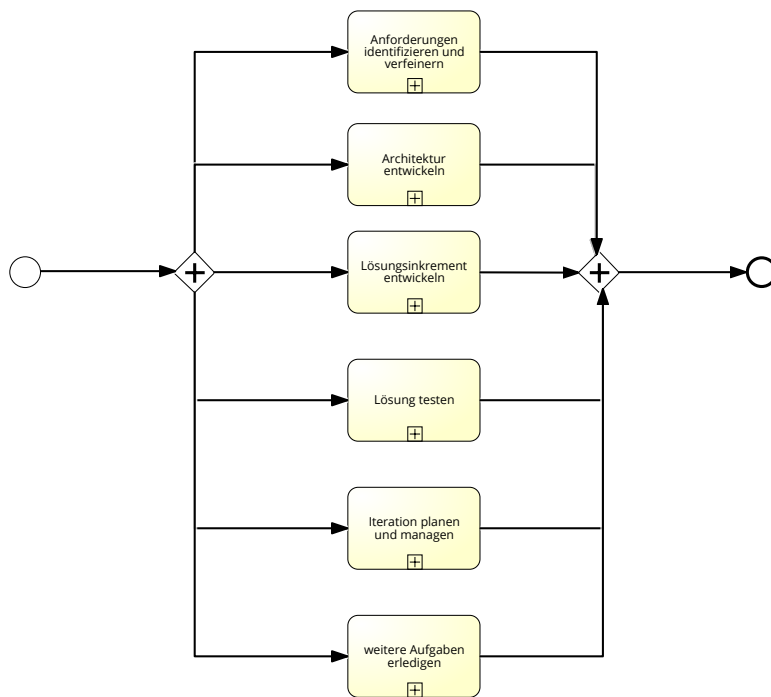


Abbildung 6.12.: Phasen Open UP Unterprozess Elaboration- imperativ

6. Imperative und deklarative Modellierung für SE-Prozessmodelle

Aufgaben erledigen und Produktdokumentation und Training erstellen nebeneinander parallel ausgeführt.

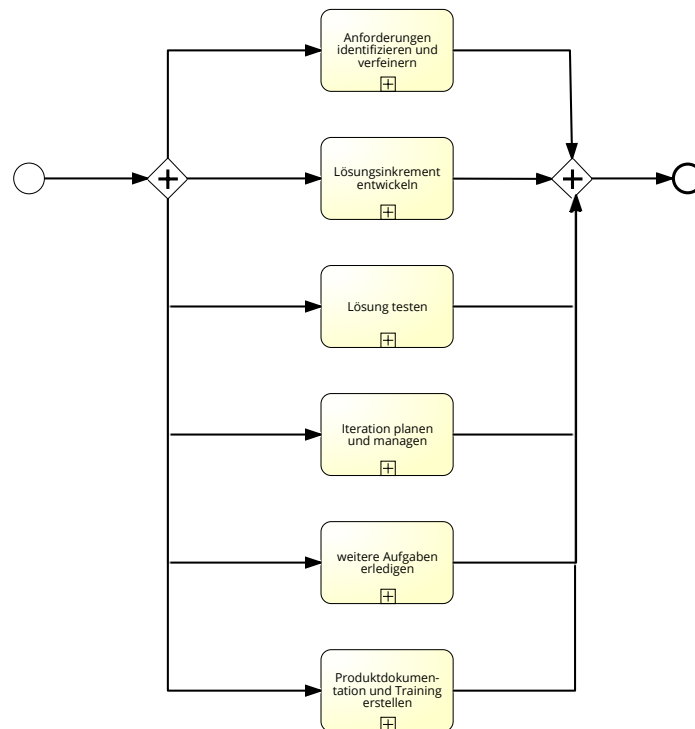


Abbildung 6.13.: Phasen Open UP Unterprozess Construction- imperativ

Abbildung 6.14 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

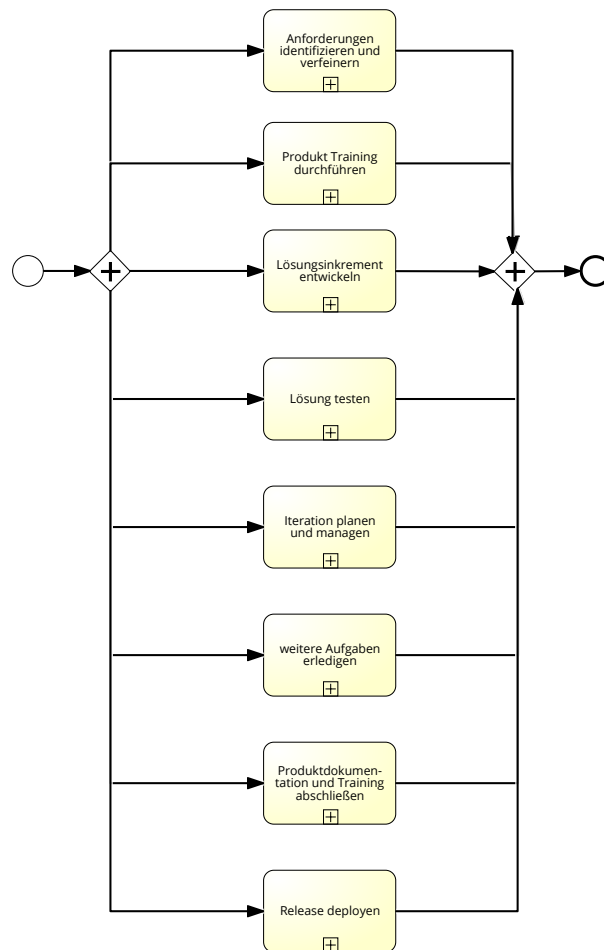


Abbildung 6.14.: Phasen Open UP Unterprozess Transition- imperativ

6. Imperative und deklarative Modellierung für SE-Prozessmodelle

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 *Lösungsinkrement entwickeln* modelliert.

Lösungsinkrement entwickeln

Im Unterprozess *Lösungsinkrement 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 6.26 ist die imperative Modellierung von *Lösungsinkrement entwickeln*

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.

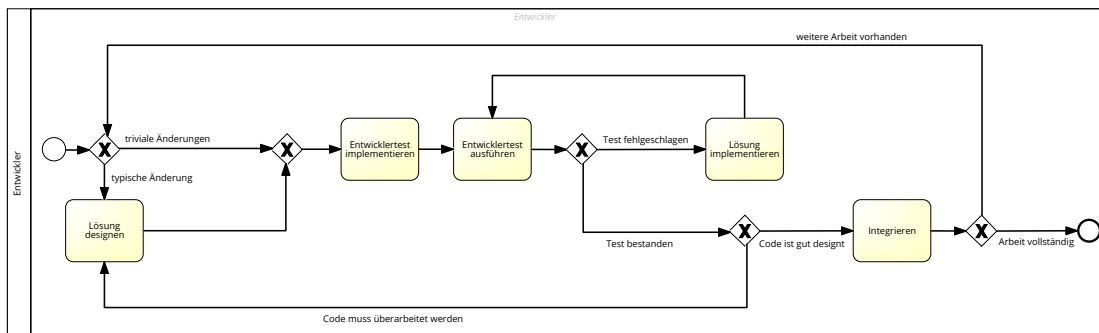


Abbildung 6.15.: Lösung inkrement entwickeln imperativ

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

6. Imperative und deklarative Modellierung für SE-Prozessmodelle

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 6.16 zeigt die imperative 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 6.17 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.

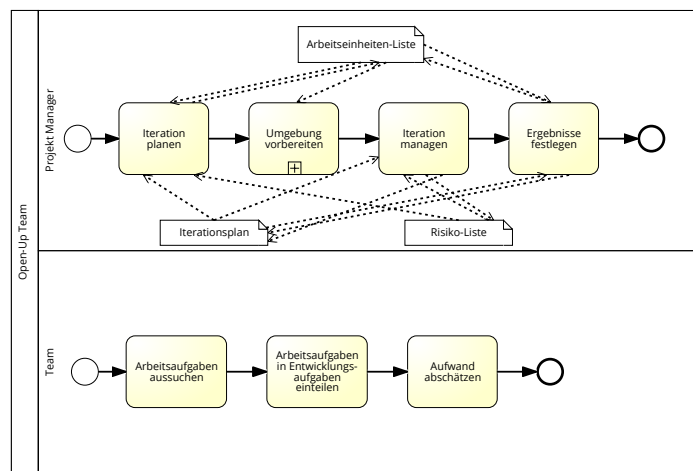


Abbildung 6.16.: Iteration planen und managen imperativ -Inception

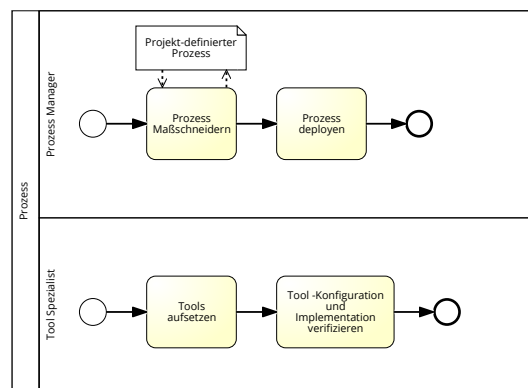


Abbildung 6.17.: Iteration planen und managen imperativ -Inception Unterprozess Umgebung vorbereiten

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 6.18 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*

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

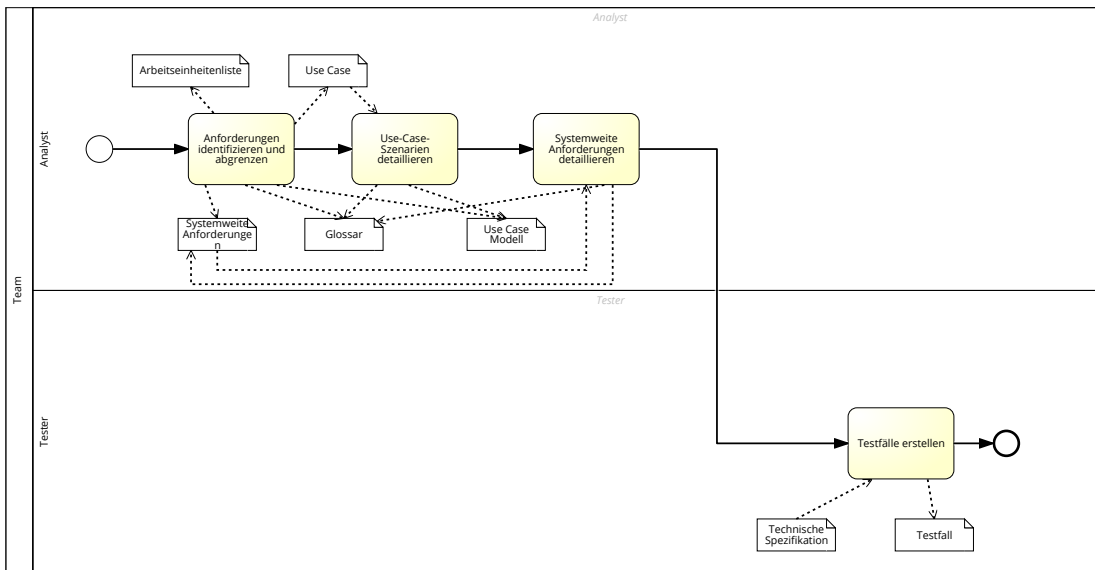


Abbildung 6.18.: Anforderungen identifizieren und verfeinern-Elaboration

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.

6. Imperative und deklarative Modellierung für SE-Prozessmodelle

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 6.19 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*.

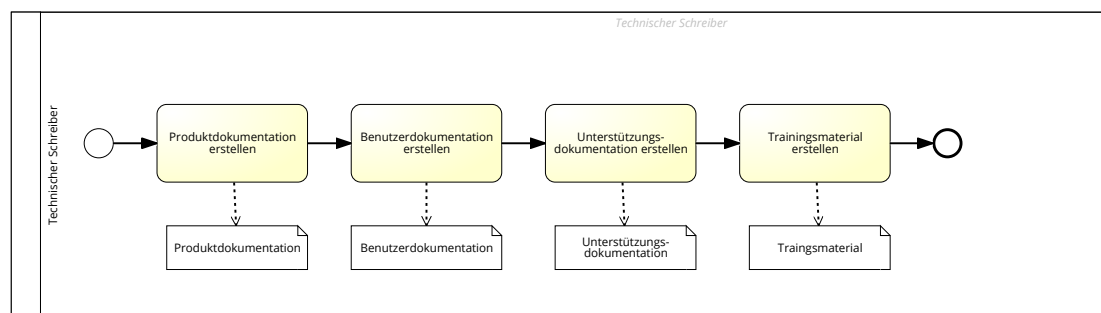


Abbildung 6.19.: Produktdokumentation und Training erstellen

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 releast werden kann.

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

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 6.20 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 *Releasemitteilungen übermitteln* ausgeführt. Falls das Deployment nicht erfolgreich ist, muss zunächst die Aktivität *Backoutplan ausführen* erledigt werden und erst danach die Aktivität *Releasemitteilungen übermitteln* ausgeführt werden.

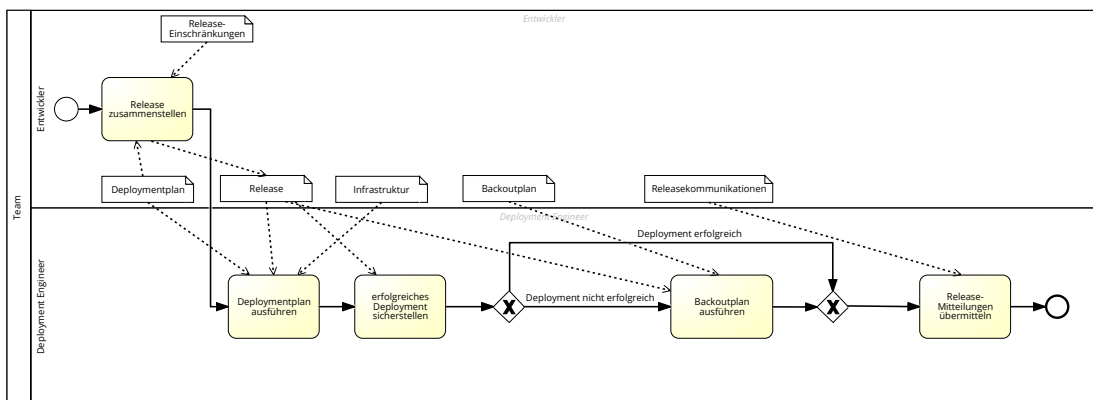


Abbildung 6.20.: Release deployen-Transition

6.2.3. Deklarative Modellierung Open UP

Phasen des Open UP

In Abbildung 6.21 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 *succession* 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.

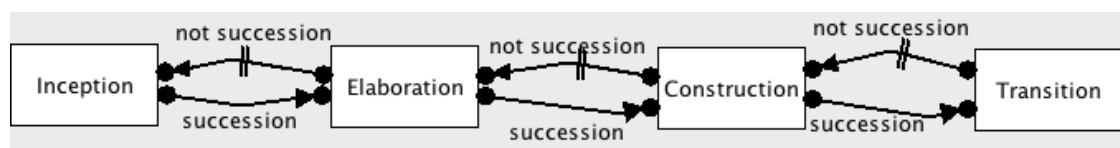


Abbildung 6.21.: Phasen Open UP- deklarativ

Abbildung 6.22 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 6.23 ist die deklarative Modellierung der Phase Elaboration abgebildet. Die sechs Aktivitäten *Anforderungen identifizieren und verfeinern*, *Architektur entwickeln*, *Lösungsincrement entwickeln*, *Lösung testen*, *Iteration planen und managen* sowie *weitere Aufgaben erledigen* werden parallel zueinander ausgeführt. Aus diesem Grund befindet sich lediglich das Constraint *Exactly 1* an jeder Aktivität, da sie inner-

6.2. Open Unified Process (Open UP)

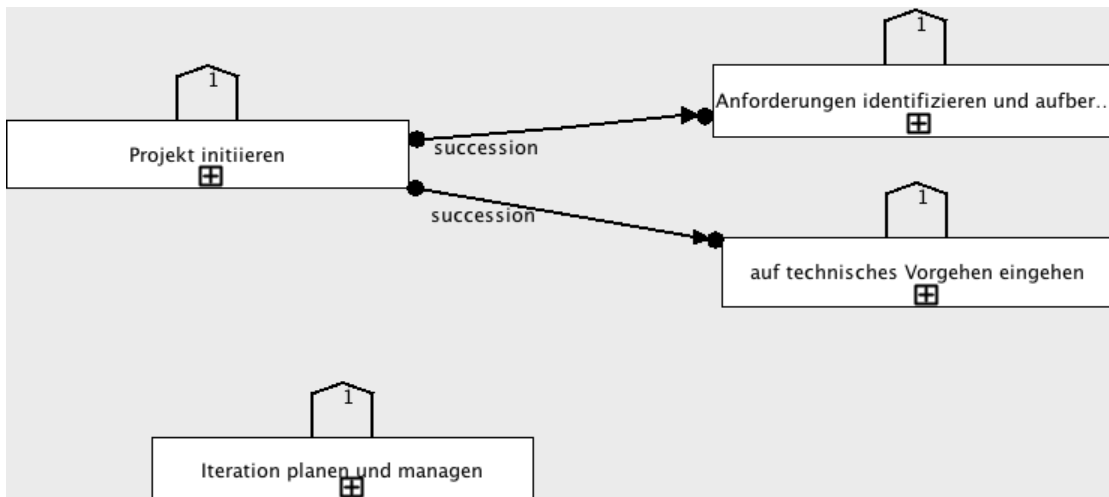


Abbildung 6.22.: Phasen Open UP Unterprozess Inception- deklarativ

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 6.24 entnommen werden. Hier werden die sechs Aktivitäten *Anforderungen identifizieren und verfeinern*, *Lösungsincrement entwickeln*, *Lösung testen*, *Iteration planen und managen*, *weitere Aufgaben erledigen* und *Produktdokumentation und Training erstellen* nebeneinander parallel ausgeführt.

Abbildung 6.25 kann die deklarative Modellierung der Phase Transition entnommen werden.

Die Phasen *Anforderungen identifizieren und verfeinern*, *Produkt Training durchführen*, *Lösungsincrement 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

6. Imperative und deklarative Modellierung für SE-Prozessmodelle

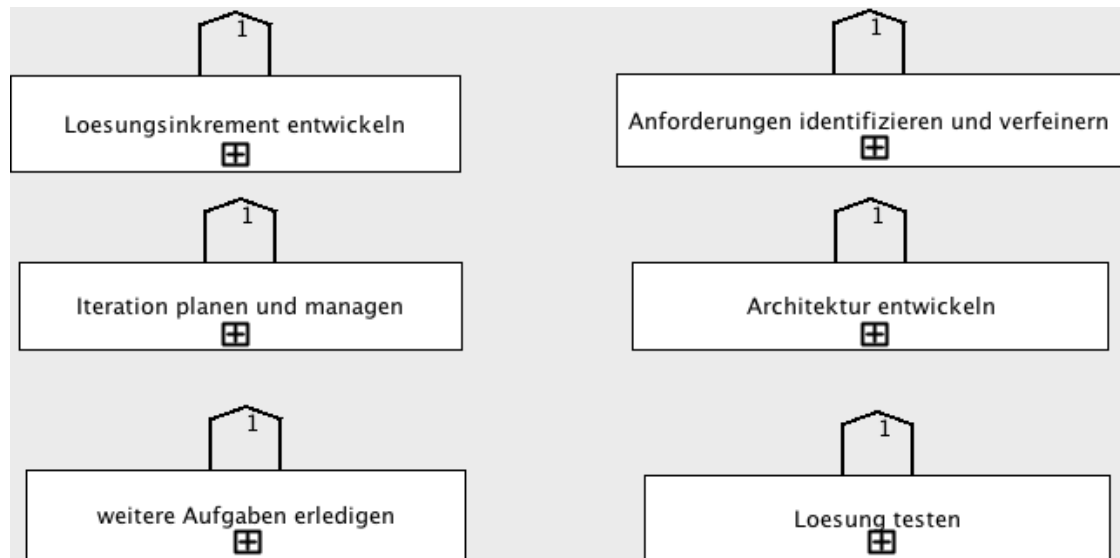


Abbildung 6.23.: Phasen Open UP Unterprozess Elaboration- deklarativ

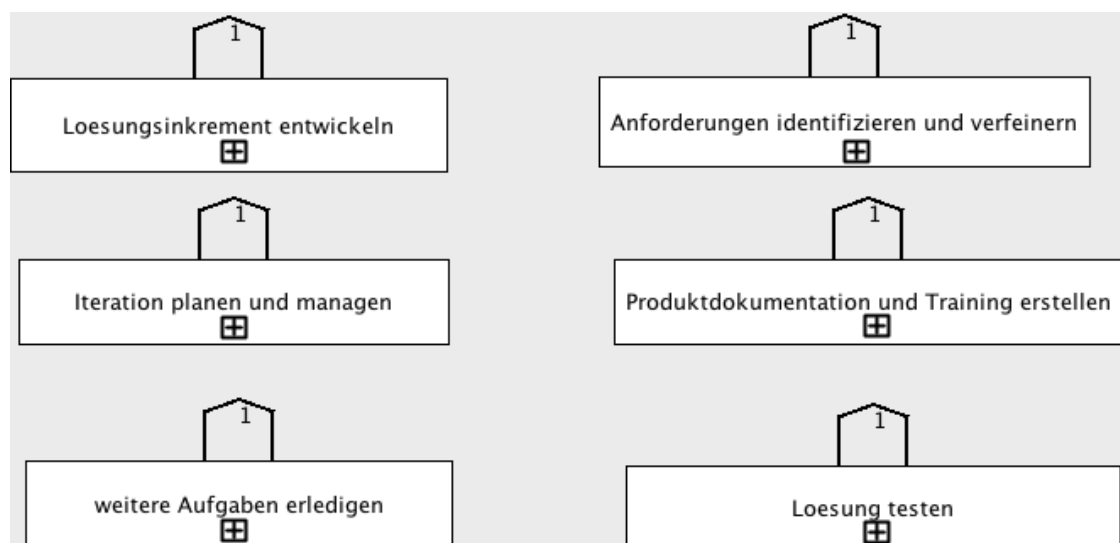


Abbildung 6.24.: Phasen Open UP Unterprozess Construction- deklarativ

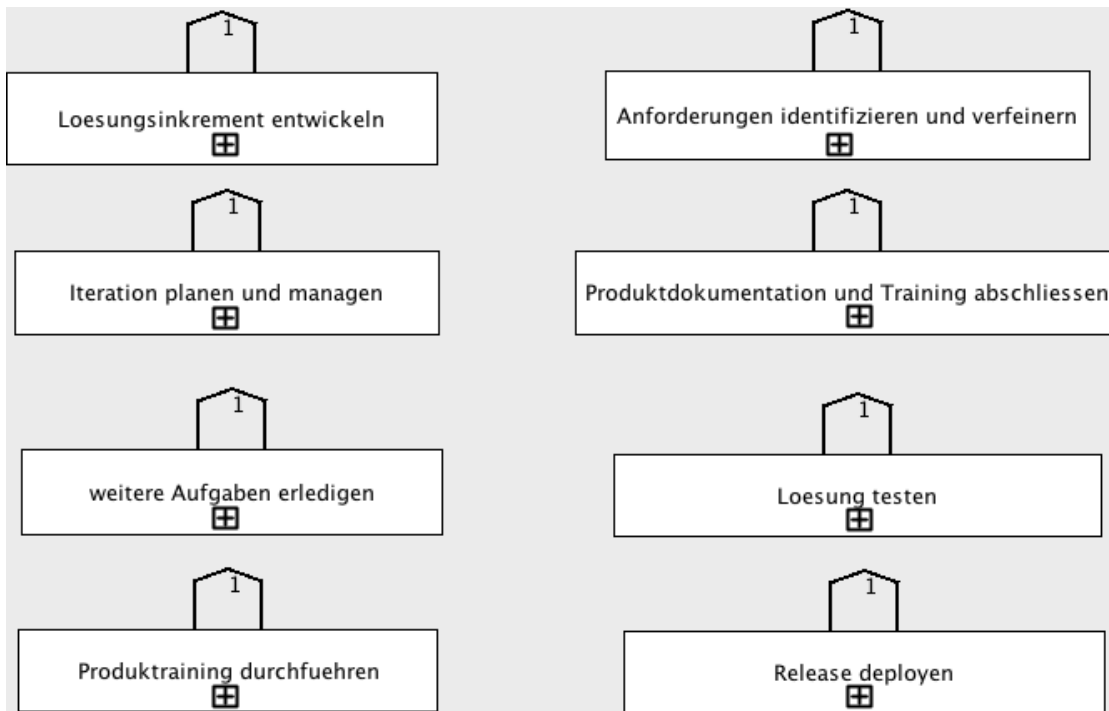


Abbildung 6.25.: Phasen Open UP Unterprozess Transition- deklarativ

6. Imperative und deklarative Modellierung für SE-Prozessmodelle

Unterprozesse aus jeder Phase den Rahmen der Arbeit sprengen würde.

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 6.26 entnommen werden.

Falls eine *Lösung designt* 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.

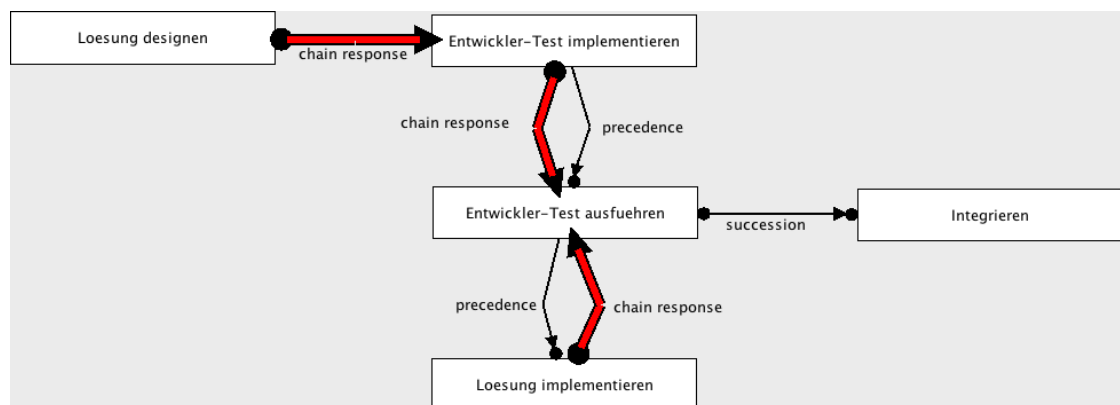


Abbildung 6.26.: Develop Solution Increment deklarativ

6.2. Open Unified Process (Open UP)

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

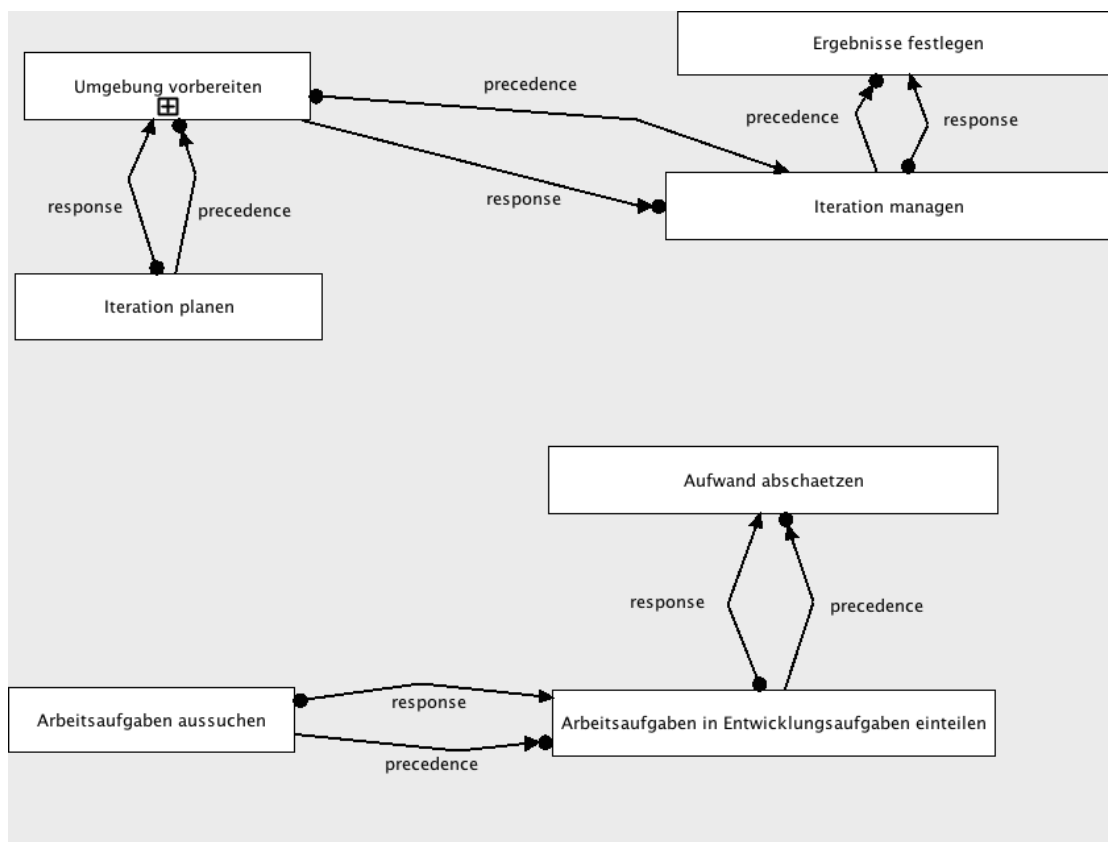


Abbildung 6.27.: Plan and manage iteration deklarativ

Abbildung 6.29 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 muss die Aktivität *Deploymentplan*

6. Imperative und deklarative Modellierung für SE-Prozessmodelle

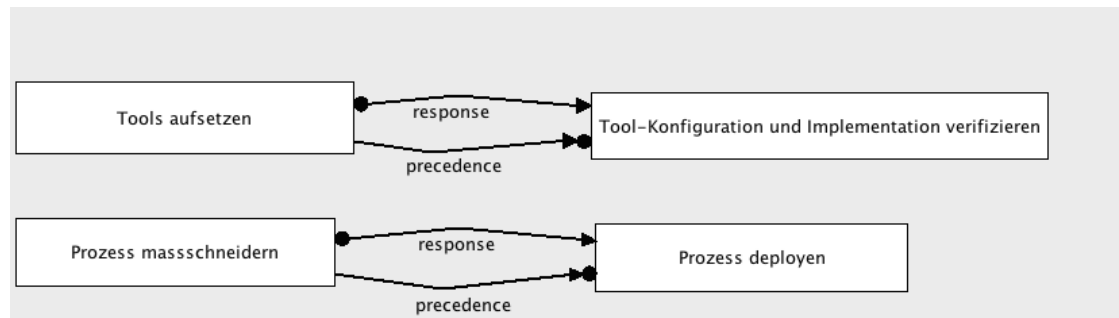


Abbildung 6.28.: Plan and manage iteration deklarativ

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.

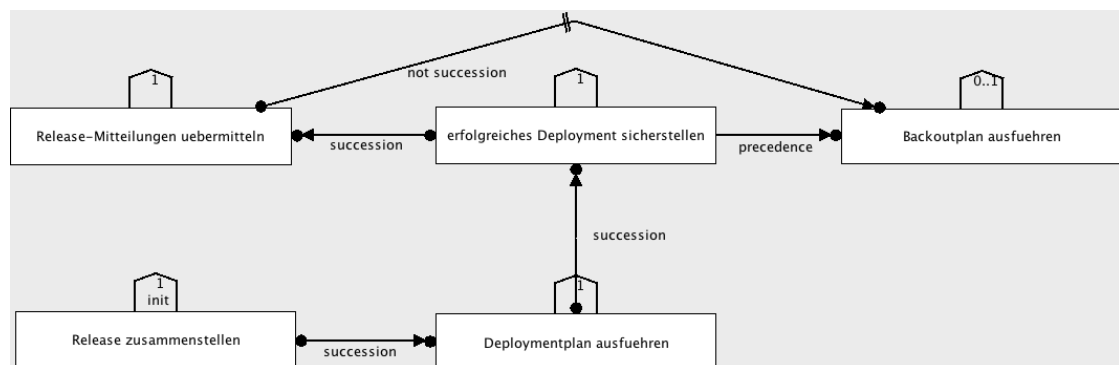


Abbildung 6.29.: Deploy Release-Transition

In Abbildung 6.30 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 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.

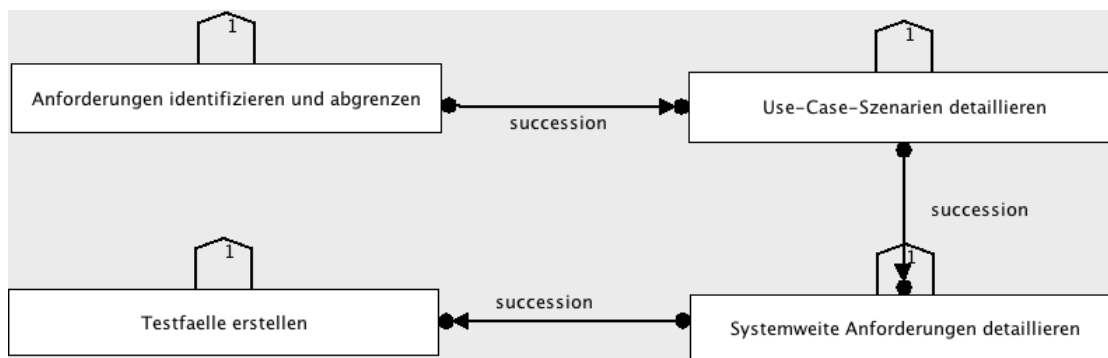


Abbildung 6.30.: Identify and refine requirements-Elaboration

Abbildung 6.31 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

6. Imperative und deklarative Modellierung für SE-Prozessmodelle

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

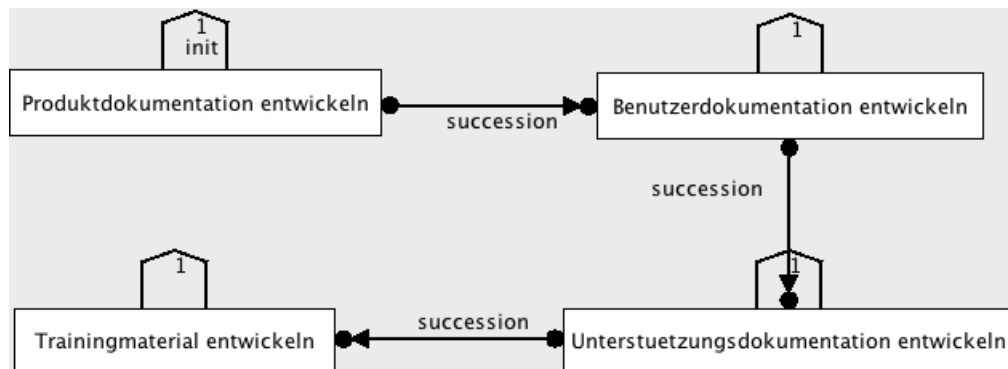


Abbildung 6.31.: Develop Product Documentation-Construction

6.2.4. Vergleich

Abbildung 6.32 zeigt die Auswertung der Elemente im Modell Phasen des Open UP. Während bei BPMN vier Aktivitäten, acht Gateways und 25 Verbindungselemente, also gesamt 29 Elemente für das Modell benötigt werden, werden in ConDec nur vier Aktivitäten und sechs Constraints, also insgesamt zehn Elemente benötigt.

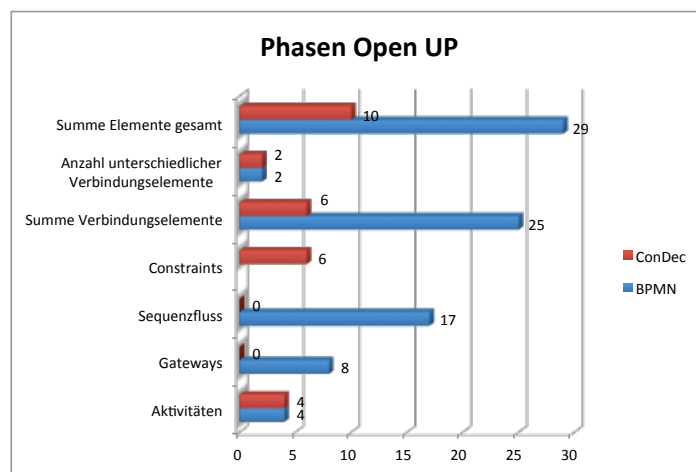


Abbildung 6.32.: Phasen Open UP

Phasen Open UP-Inception

Die Anzahl der Elemente zur Darstellung der Phase Inception in ConDec und BPMN können Abbildung 6.33 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).

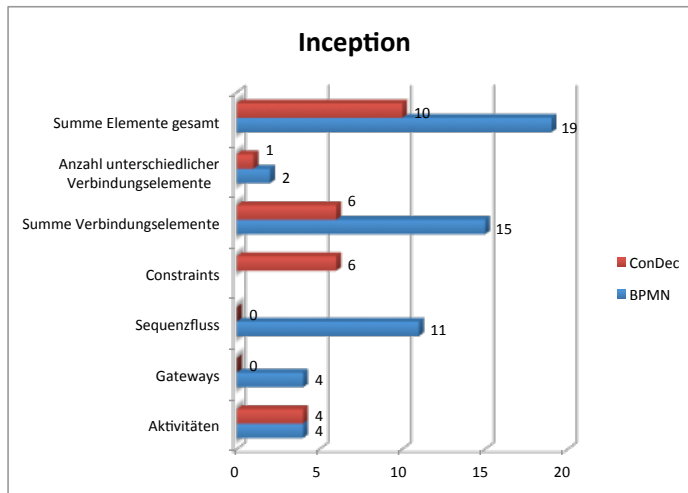


Abbildung 6.33.: Open UP-Inception

Abbildung 6.34 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 benötigt. In ConDec sind zur korrekten Darstellung 21 Constraints notwendig. Somit werden in BPMN insgesamt 26 Elemente und in ConDec 32 Elemente verwendet.

Wird die komplette Phase Inception mit allen Unterprozessen(Phasen Open UP, Inception, Iteration planen und managen) betrachtet, werden in Summe zur Darstellung in BPMN 70 Elemente und in ConDec 52 Elemente benötigt (Abbildung 6.35).

6. Imperative und deklarative Modellierung für SE-Prozessmodelle

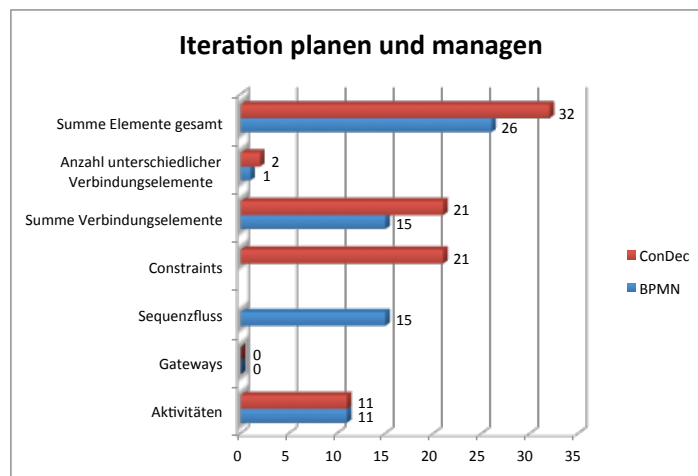


Abbildung 6.34.: Open UP-Iteration planen

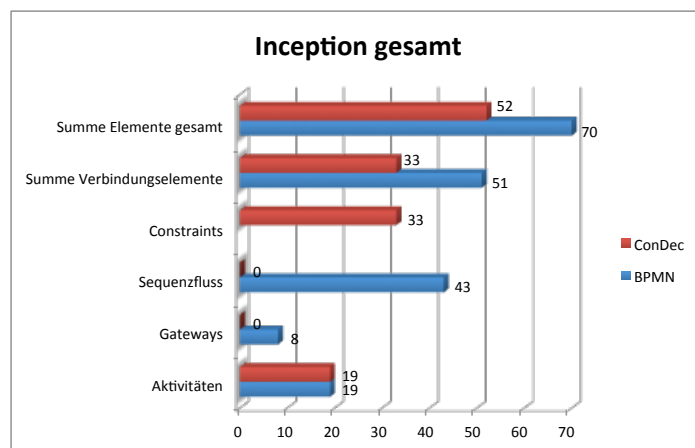


Abbildung 6.35.: Open UP-Inception gesamt

6.3. V-Modell XT

Das V-Modell XT zählt zu den schwergewichtigen Prozessmodellen [?]. Es wird als Entwicklungsstandard für die Durchführung von IT-Vorhaben in der öffentlichen Verwaltung in Deutschland herangezogen [?]. Beschrieben werden im V-Modell XT die Abläufe im Verlauf eines Entwicklungsprojektes über Produkte, Rollen und Aktivitäten [?]. Es wird somit ganz genau geregelt, *Wer*, *Wann*, *Was* in einem Projekt zu tun hat [?]. Die Vorgehensbausteine ermöglichen neben einer Modularisierung der Abläufe auch eine flexible Zusammenstellung, wodurch das V-Modell XT auf die jeweils eigene Situation angepasst werden kann. [?, ?].

6.3.1. Analyse V-Modell XT

Abbildung 6.36 zeigt die Grundstruktur des V-Modell XT, welche im Folgenden detailliert erläutert wird.

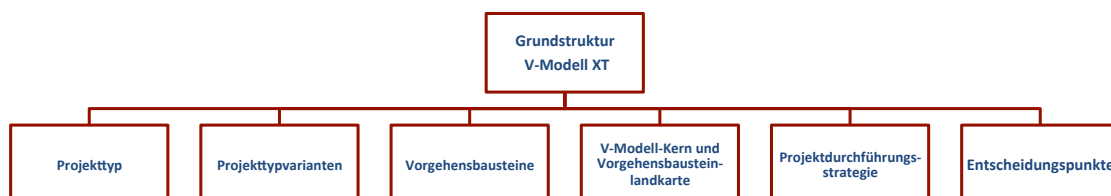


Abbildung 6.36.: Grundstruktur V-Modell XT nach [?]

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 6.38 gibt einen ersten Überblick über die verschiedenen Projekttypen im V-Modell XT [?].

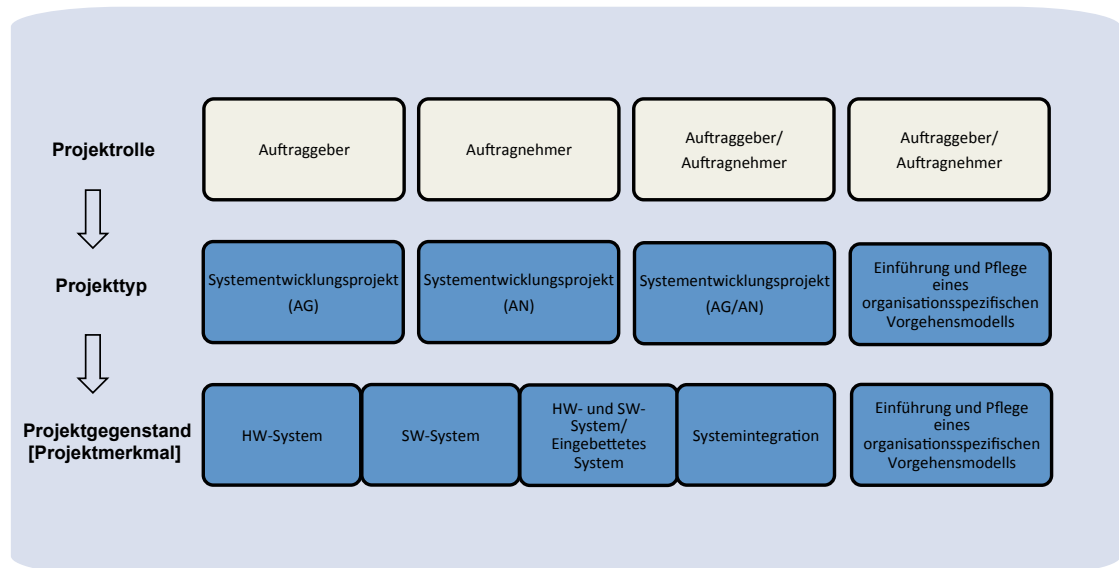


Abbildung 6.37.: Projekttypen V-Modell XT nach [?]

Es existieren somit drei verschiedene Projekttypen: *Systementwicklungsprojekt eines Auftraggebers*, *Systementwicklungsprojekt eines Auftragnehmers* und *Einführung und Pflege eines organisationsspezifischen Vorgehensmodells* [?].

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* [?, ?].

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 [?, ?].

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 [?, ?].

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 Prozessmodelles notwendig und etwaige Verbesserungsmöglichkeiten sind zu erfassen und durchzuführen [?, ?].

Wie aus Abbildung 6.38 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 [?, ?].

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 6.38 sind die verschiedenen Projekttypvarianten des V-Modell XT aufgelistet und es wird gezeigt, mit welchen Merkmalen die zugehörigen Projekttypvarianten ausgewählt werden können [?]. Hieraus ist ersichtlich, dass für den Projekttyp *Einführung und Pflege eines organisationsspezifischen Vorgehensmodells* nur eine einzige Projekttypvariante existiert [?].

Für den Projekttyp *Systementwicklungsprojekt (AG)* existieren zwei verschiedene Projekttypvarianten. 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 er-

6. Imperative und deklarative Modellierung für SE-Prozessmodelle

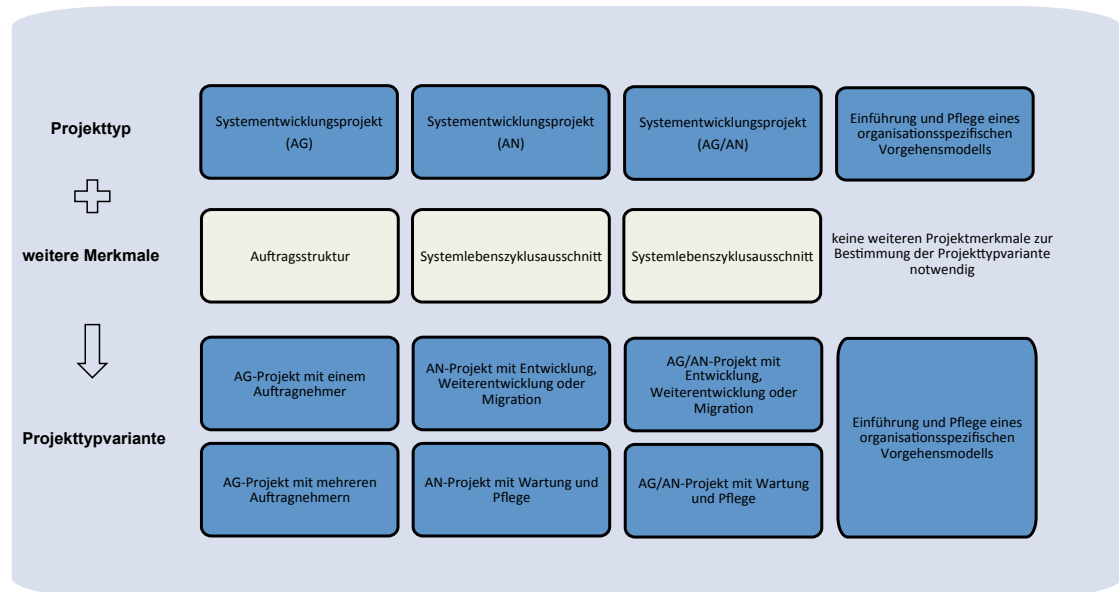


Abbildung 6.38.: Zuordnung der Projekttypvarianten zu den Projekttypen des V-Modell XT [?]

gibt sich die Projekttypvariante *Systementwicklungsprojekt (AG)- Projekt mit mehreren Auftragnehmern* [?].

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

Vorgehensbausteine

Modulare, aufeinander aufbauende Vorgehensbausteine bilden den Kern des V-Modell XT. Vorgehensbausteine sind selbständig 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 6.39

gibt einen Überblick über diese [?, ?].

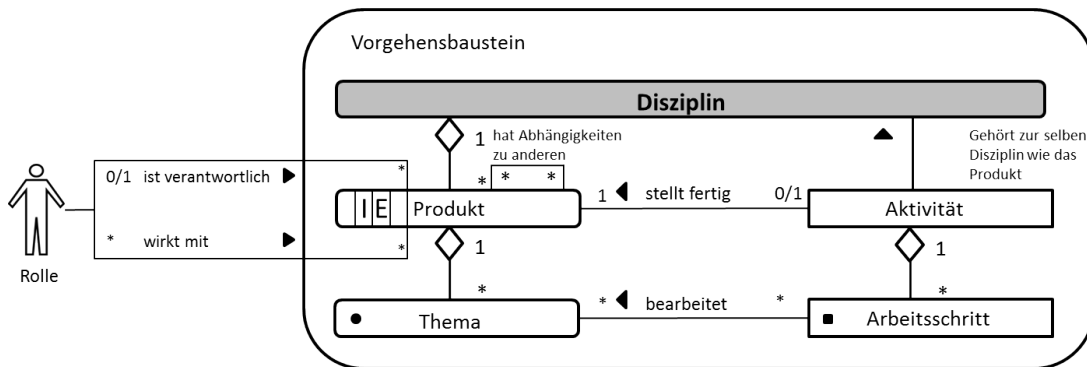


Abbildung 6.39.: Vorgehensbausteine V-Modell XT nach [?]

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 [?].

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 [?].

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 [?].

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 [?]. Hierdurch kann also ein individuelles V-Modell für ein Projekt erstellt werden [?]. Hierfür ist es notwendig, die Vorgehensbausteine für ein V-Modell-Projekt nach den Vorgaben des Projekttyps auszuwählen und festzulegen [?].

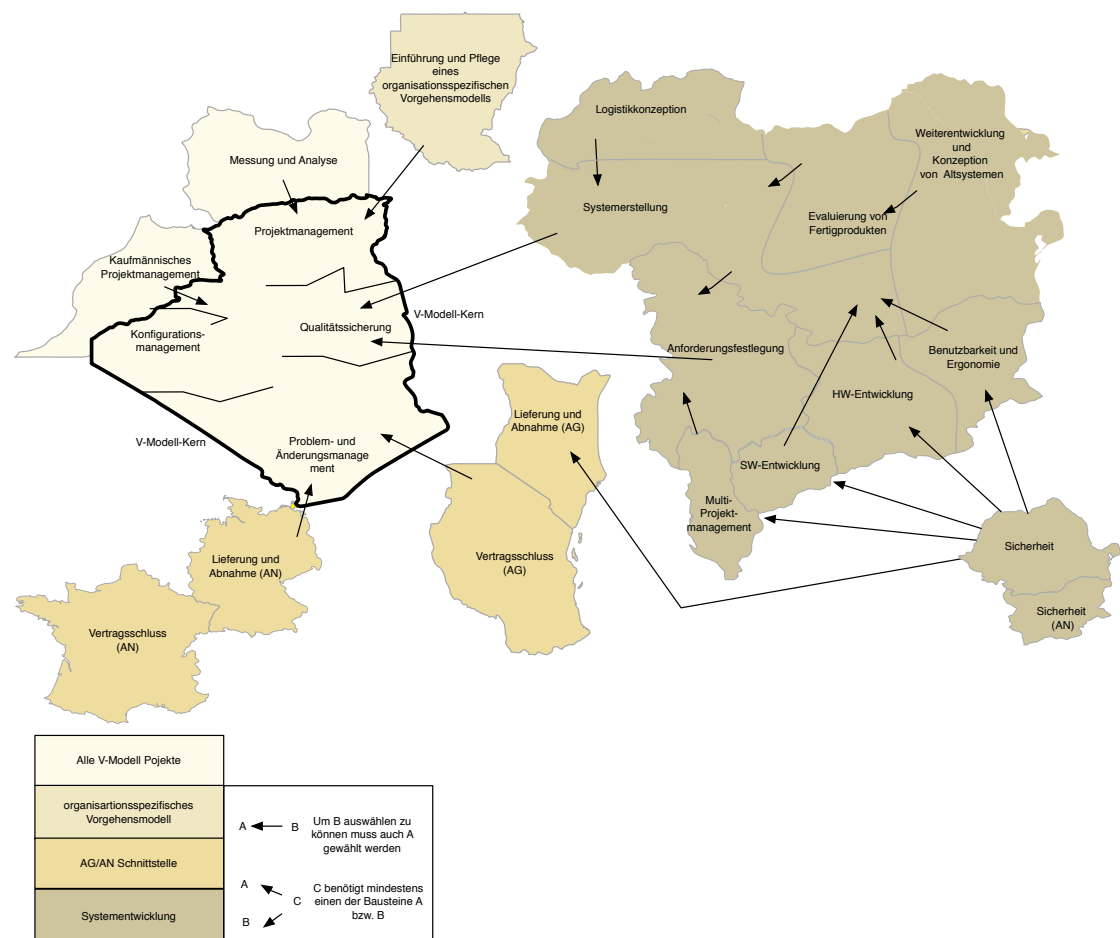


Abbildung 6.40.: V-Modell-Kern und Vorgehensbausteinlandkarte nach [?]

Wie Abbildung 6.40 zeigt, können die Vorgehensbausteine in die vier Bereiche *Alle V-Modell-Projekte*, *Organisationsspezifisches Vorgehensmodell*, *AG/AN-Schnittstelle* und

Systementwicklung eingeteilt werden [?].

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 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 [?].

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 [?].

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* [?].

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 [?].

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 [?].

Entscheidungspunkte

Abbildung 6.41 zeigt, dass die in der Projektdurchführungsstrategie vorgegebenen Projektfortschrittsstufen bei Erreichen durch Entscheidungspunkte markiert werden, welche einen Meilenstein im Projektablauf 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 6.42 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 [?].

Demnach gelten die Entscheidungspunkte *Projekt genehmigt*, *Projekt definiert*, *Iteration geplant* und *Projekt abgeschlossen* für alle Projekttypen und Projektdurchführungsstrategien [?].

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 [?].

Die Entscheidungspunkte für die Arbeitgeber/Arbeitnehmer Schnittstelle setzen sich

aus *Projekt ausgeschrieben*, *Angebot abgegeben*, *Projekt beauftragt*, *Lieferung durchgeführt*, *Abnahme erfolgt* und *Projektfortschritt überprüft* zusammen [?].

Bei der Entwicklung eines organisationsspezifischen Vorgehensmodells kommen die Entscheidungspunkte *Vorgehensmodell analysiert*, *Verbesserung Vorgehensmodell konzipiert* und *Verbesserung Vorgehensmodell realisiert* zum Einsatz [?].

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

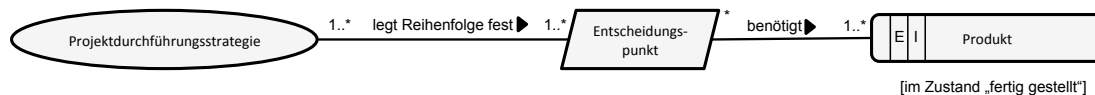


Abbildung 6.41.: Entscheidungspunkte V-Modell XT nach [?]

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.

6. Imperative und deklarative Modellierung für SE-Prozessmodelle

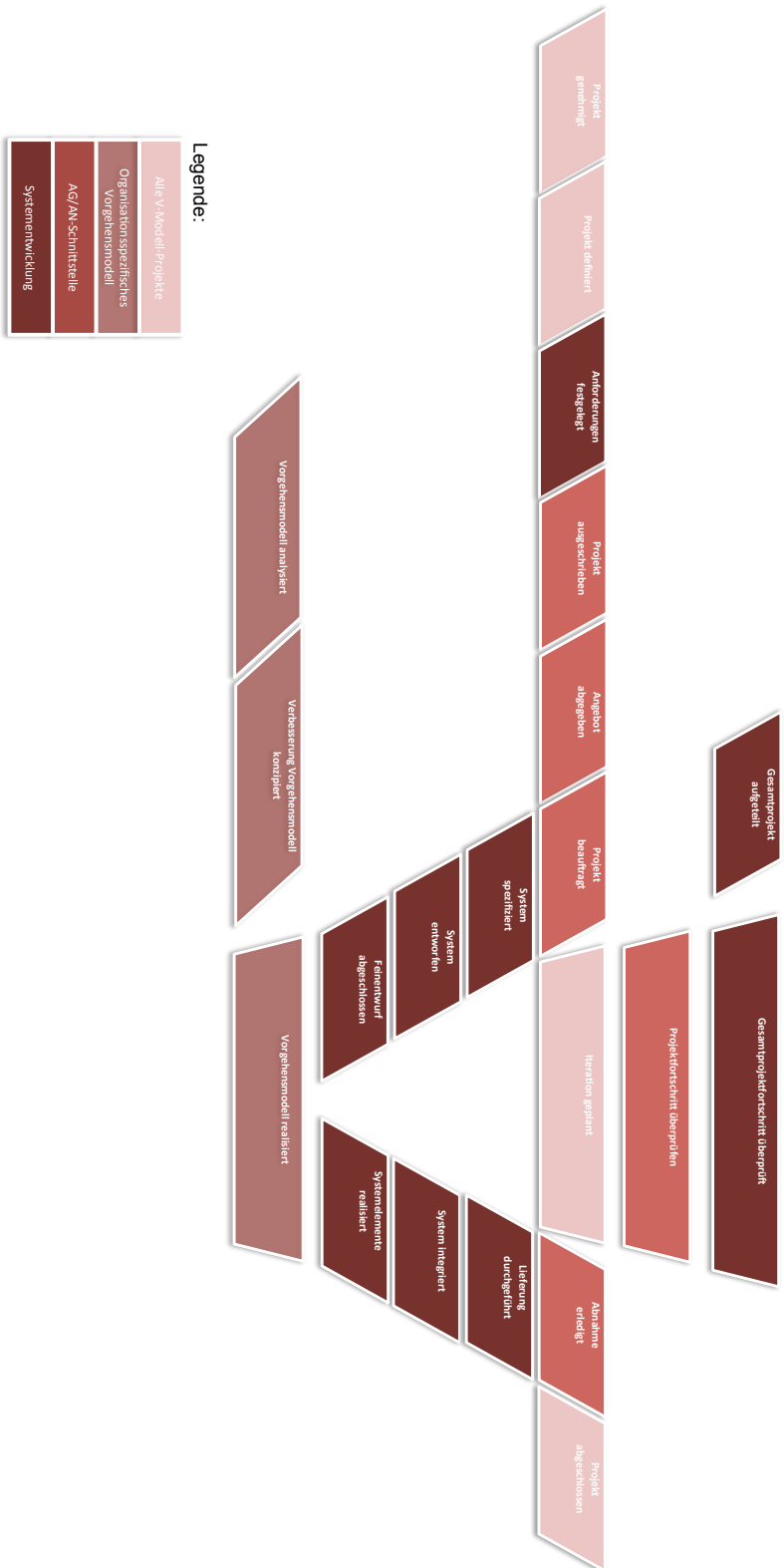


Abbildung 6.42.: Entscheidungspunkte für die Projektdurchführungsstrategie nach [?]

6.3.2. Imperative Modellierung V-Modell

Systementwicklungsprojekt AG/AN

Die imperative Modellierung von *Systementwicklungsprojekt AG/AN* zeigt Abbildung 6.49.

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 *Komponenten-basierte Entwicklung durchgeführt* oder eine *Inkrementelle Entwicklung durchgeführt* werden soll. dies wird durch das XOR-Gateway beschrieben, welche 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*.

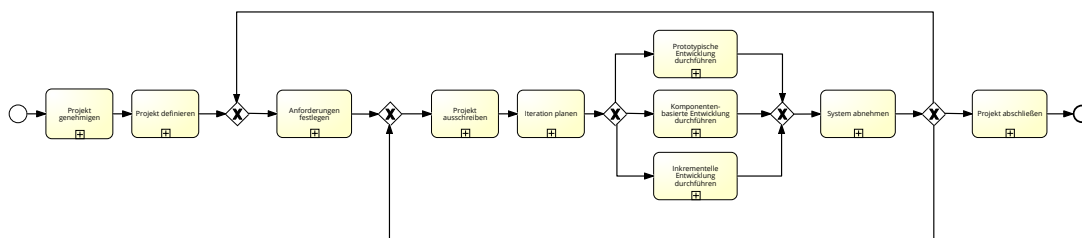


Abbildung 6.43.: Systementwicklungsprojekt AG/AN V-Modell - imperativ

Inkrementelle Entwicklung durchführen

In Abbildung 6.44 ist die imperative Modellierung des Unterprozesses *Inkrementelle Entwicklung durchführen* abgebildet.

6. Imperative und deklarative Modellierung für SE-Prozessmodelle

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

Hiernach wird der *Feinentwurf entworfen* und *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.

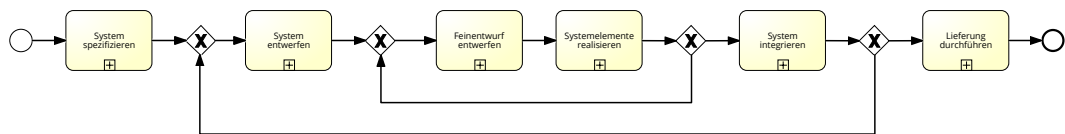


Abbildung 6.44.: Unterprozess Inkrementelle Entwicklung durchführen V-Modell - imperativ

System entwerfen

Abbildung 6.45 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*, *Integrations- und Prüfkonzept Hardware (HW) erstellen*, *Integrations- und Prüfkonzept Software (SW) erstellen* und *Migrationskonzept erstellen* werden hier nacheinander ausgeführt.

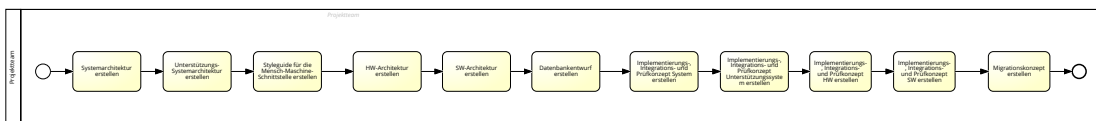


Abbildung 6.45.: System entwerfen V-Modell - imperativ

System spezifizieren

Am Anfang des Unterprozesses *System spezifizieren* (Abbildung 6.46) 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-*

6. Imperative und deklarative Modellierung für SE-Prozessmodelle

samtprojektfortschritt ermittelt

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

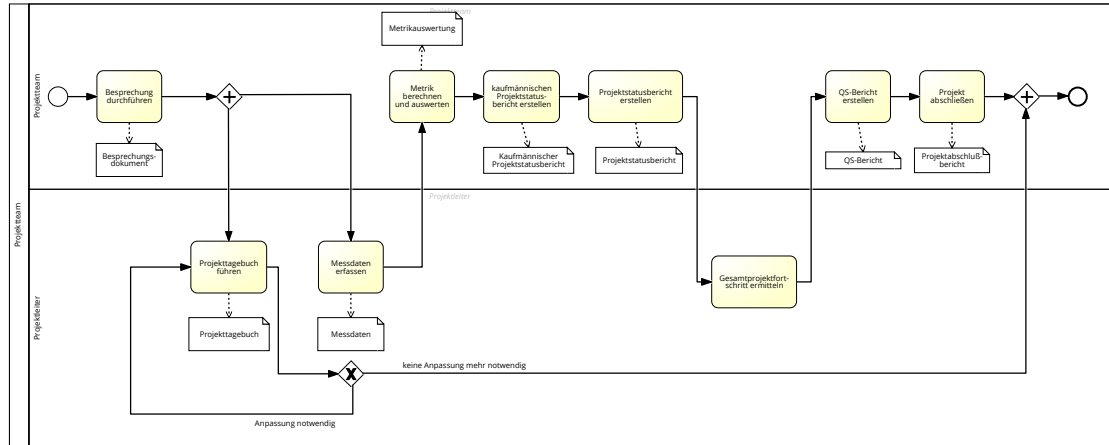


Abbildung 6.46.: System spezifizieren-imperativ

6.3.3. Deklarative Modellierung V-Modell

Die deklarative Modellierung von *Systementwicklungsprojekt AG/AN* zeigt Abbildung 6.47.

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 6.48 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 *1 of 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 6.49 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

6. Imperative und deklarative Modellierung für SE-Prozessmodelle

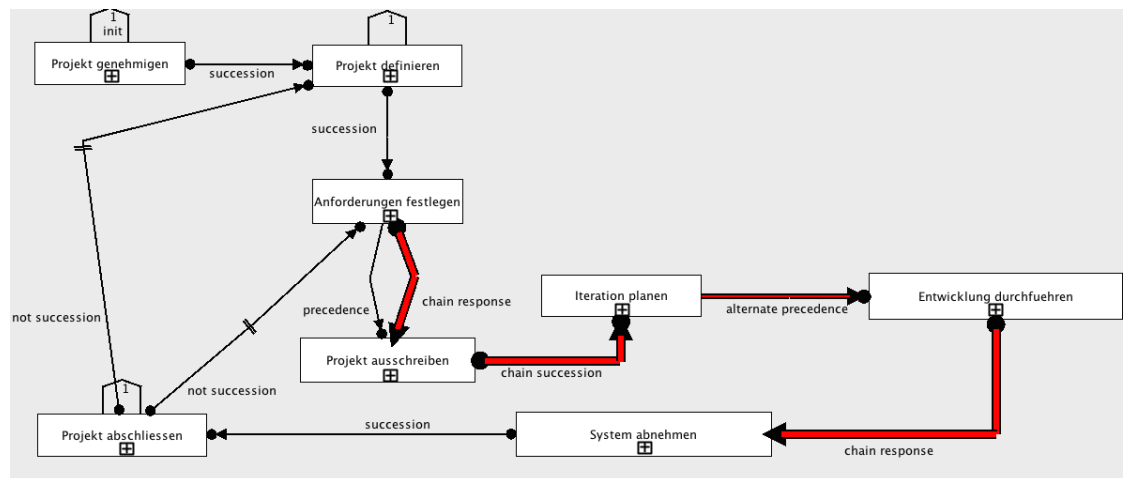


Abbildung 6.47.: Systementwicklungsprojekt AG/AN V-Modell - deklarativ

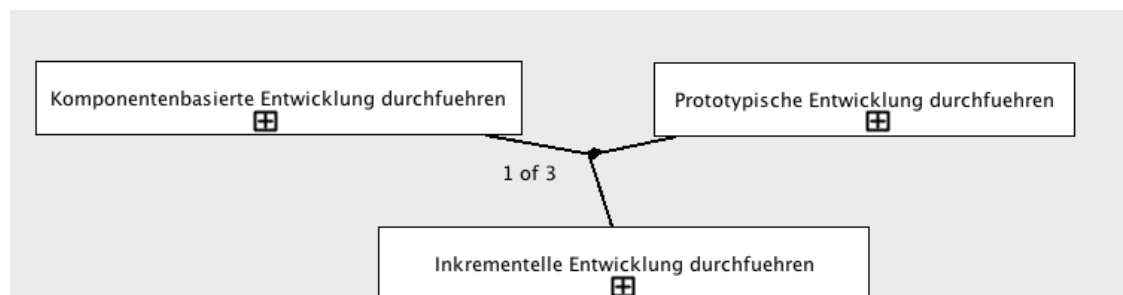


Abbildung 6.48.: Systementwicklungsprojekt AG/AN V-Modell Unterprozess Entwicklung durchfuehren - deklarativ

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.

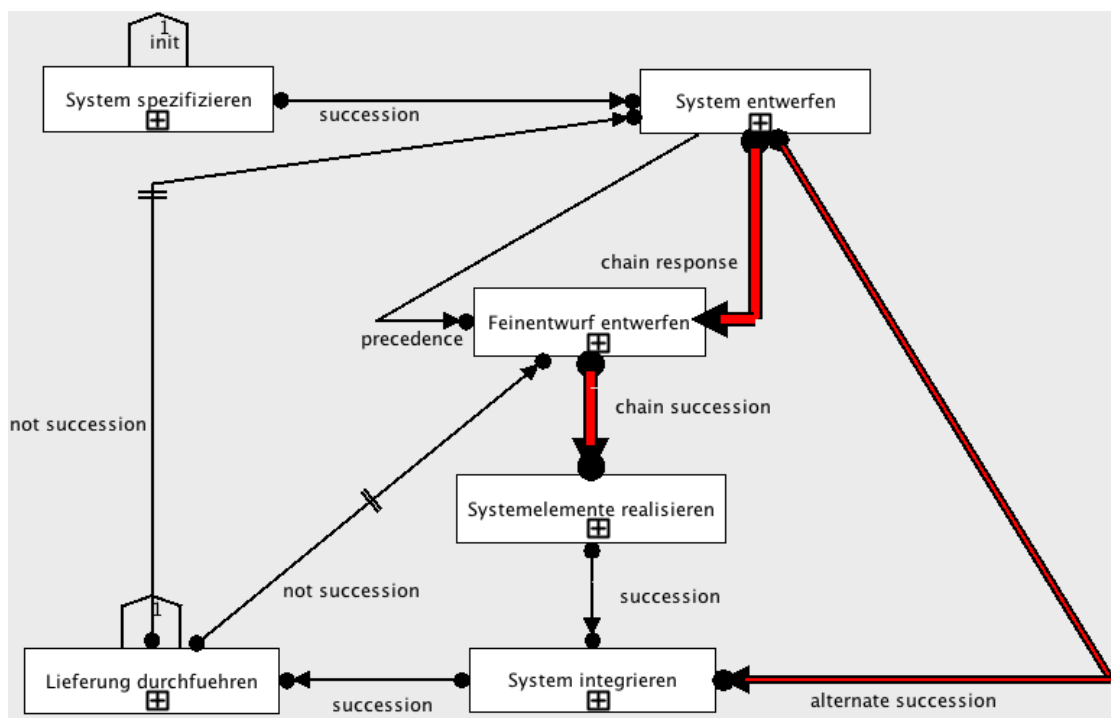


Abbildung 6.49.: Unterprozess Inkrementelle Entwicklung durchführen V-Modell - imperativ

System entwerfen

Abbildung 6.50 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.

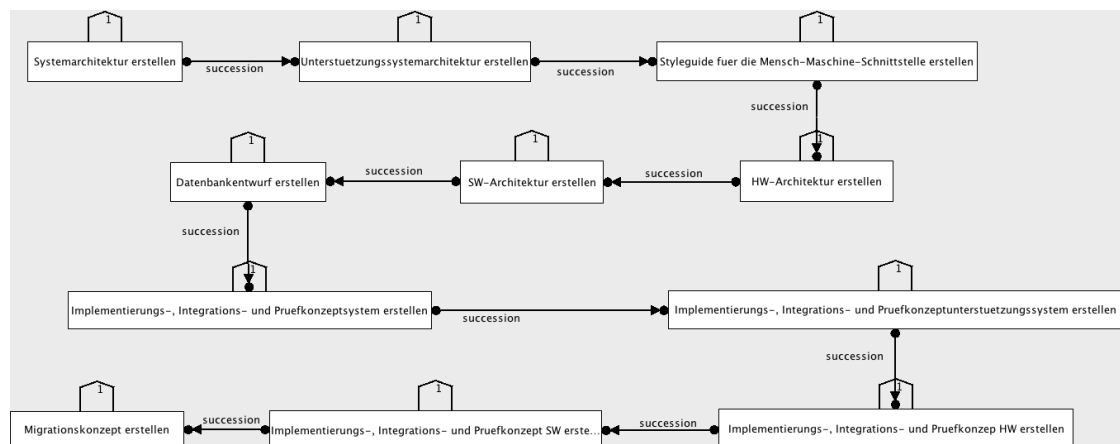


Abbildung 6.50.: System entwerfen - deklarativ

System spezifizieren

Am Anfang von System spezifizieren (Abbildung 6.51) muss eine *Besprechung durchgeführt* werden. Dies wird durch das Constraint *init* sichergestellt.

Bei jeder Änderung muss die Aktivität *Projekttagbuch fuehren* 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.

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.

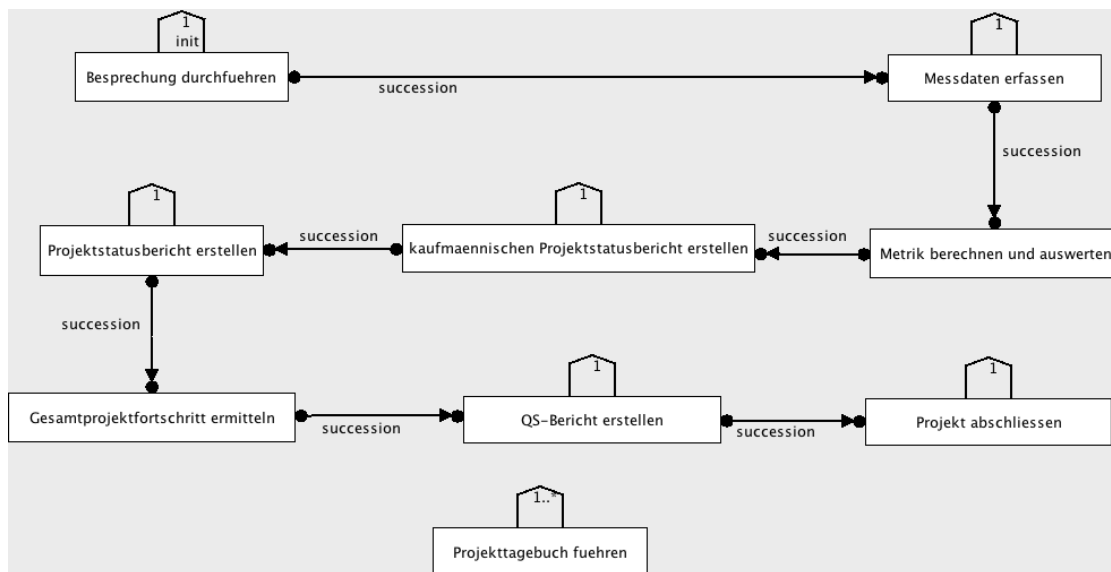


Abbildung 6.51.: System spezifizieren- deklarativ

6.3.4. Vergleich

6.3.5. Vergleich

7

Validierung

8

Related Work

8.1. Modellierung von Software-Engineering Prozessmodellen

8.1.1. Analyse und Überführung von Softwareentwicklungsprozessen in die standardisierte BPMN Notation

Die Bachelorarbeit [?] beschäftigt sich mit der Analyse und der Überführung von Softwareentwicklungsprozessen in die Prozessmodellierungssprache BPMN. Da es hierbei zu Problemen bei der Darstellung von Zuständigkeiten, Ergebnissen und Abhängigkeiten kommen kann, wird BPMN um weitere Elemente zur Darstellung dieser Sachverhalte erweitert. Die Softwareentwicklungsprozesse werden dann mit diesen Erweiterungen modelliert.

8. *Related Work*

In der vorliegenden Arbeit werden die imperativen und deklrativen Prozessmodellierungssprachen zwar auf ihre Grenzen in der Darstellbarkeit bestimmter Sachverhalte untersucht, jedoch werden sie nicht um weitere Elemente erweitert. Die vorliegende Arbeit beschäftigt sich explizit mit dem Vergleich der beiden Prozessmodellierungssprachen und auch mit ihren Grenzen in der Darstellbarkeit. Jedoch werden die Grenzen der Darstellbarkeit nicht so detailliert wie in [?] betrachtet, sondern auf einer niedrigeren Ebene.

8.2. Verständlichkeit von Prozessmodellierungssprachen

8.2.1. Investigating expressiveness and understandability of hierarchy in declarative business process models

In [?] wird die Verständlichkeit von deklarativen Prozessmodellen im Hinblick auf die Verwendung von hierarchischen Unterprozessen untersucht. Es wird gezeigt, dass der Einsatz von Hierarchie in deklarativen Prozessmodellen zwar deren Ausdrucksfähigkeit erhöht, aber nicht beliebig überall im Modell eingesetzt werden kann. Eine durchgeführte Studie zum Validieren der Erkenntnisse kam zu dem Ergebnis, dass Hierarchie beim Verstehen der Prozessmodelle unterstützt, jedoch waren die Ergebnisse nicht schlüssig. Die Arbeit [?] fokussiert auf den Einsatz von Unterprozessen als Unterstützung bei der Verständlichkeit von deklarativen Prozessmodellen. In der vorliegenden Arbeit 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. Die vorliegende Arbeit fokussiert auf den Vergleich von deklarativen und imperativen Prozessmodellen im Allgemeinen und nicht im Hinblick auf die Verwendung von Unterprozessen.

8.3. Vergleich von Prozessmodellierungssprachen

8.3.1. Declarative versus Imperative Process Modeling Languages: An Empirical Investigation

Der Artikel [?] 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. Es werden verschiedene imperative und deklarative Prozessmodellierungssprachen betrachtet, wie z.B. Petrinetze, Business Process Execution Language (BPEL), ConDec, Linear-Time Temporal Logic (LTL)...Die Arbeit kommt zu dem Schluß, dass sequentielle Informationen durch das Nutzen von imperativen Prozessmodellierungssprachen und umständliche Informationen durch das Nutzen von deklarativen Prozessmodellierungssprachen verständlicher darzustellen sind.

[?] 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.

8.3.2. Imperative versus Declarative Process Modeling Languages: An Empirical Investigation

[?] untersucht aufbauend auf den Erkenntnissen von [?] die Verständlichkeit von imperativen und deklarativen Prozessmodellierungssprachen anhand einer Studie. Als imperative Prozessmodellierungssprache dient in diesem Artikel ebenfalls BPMN und als deklarative Prozessmodellierungssprache ebenfalls ConDec.

8. Related Work

Hierfür wurden die teilnehmenden Probanden sowohl in BPMN, als auch in ConDec eingehend trainiert. Anschließend wurden sie in zwei Gruppen eingeteilt und ihnen wurden jeweils zwei imperative und zwei deklarative Prozessmodelle vorgelegt. Am ende stellten sich die imperativen Prozessmodelle als verständlicher heraus. Jedoch ist das Ergebnis durch das vorhergehende intensive Training und die damit einhergehende starke Vertrautheit der Probanden mit BPMN und ConDec kritisch zu betrachten.

Die vorliegende Arbeit untersucht die Verständlichkeit von deklarativen und imperativen Prozessmodellen nicht nur wie [?] im Allgemeinen, sondern auch speziell im Hinblick auf Unterschiede in der Verständlichkeit bei großen und kleinen Prozessmodellen. Auch wurde in der Studie 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.

9

Zusammenfassung und Ausblick


















BPMN Notation

A. BPMN Notation

	Start			Zwischen				Ende
	Standard	Ereignis-Teilprozess unterbrechend	Ereignis-Teilprozess Nicht-unterbrechend	Eingetreten	Angeheftet unterbrechend	Angeheftet Nicht-unterbrechend	Ausgelöst	Standard
Blanko: Umtypisierte Ereignisse; Blanko-Zwischenevents 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 registriert werden.								
Fehler: Auslösen und Behandeln von definierten Fehlern								
Eskalation: Meldung an den nächsthöheren Verantwortlichen								
Terminierung: Löst die sofortige Beendigung des Prozesse 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.								
Mehrfach/Parallel: Eintreten aller Ereignisse.								

Abbildung A.1.: BPMN Ereignisse

Aufgabe	Eine Aufgabe ist eine Arbeitseinheit. Die Aufgabe wird als zusätzlicher Teilprozess markiert durch ein zusätzliches 
Transaktion	Eine Transaktion ist eine Gruppe von Aktivitäten, die logisch zusammen gehören. Ein Transaktionsprotokoll kann angegeben werden.
Ereignis-Teilprozess	Eine Ereignis-Teilprozess wird in einem anderen Teilprozess platziert. Er wird durch ein Startereignis ausgelöst und kann abhängig vom Ereignistyp den umgebenden Teilprozess abbrechen oder parallel dazu ausgeführt werden.
Aufruf- Aktivität	Eine Aufruf-Aktivität repräsentiert einen Teilprozess oder eine Aufgabe, welche global definiert sind und im aktuellen Prozess wiederverwendet werden. Der Aufruf eines separaten Teilprozesses wird gekennzeichnet durch ein zusätzliches 

Markierungen Sie beschreiben das Ausführungsverhalten von Aktivitäten		Aufgaben-Typen Sie beschreiben den Charakter einer Aufgabe	
	Teilprozess		Senden
	Schleife		Empfangen
	Parallele Mehrfachausführung		Benutzer
	Sequentielle Mehrfachausführung		Manuell
	Ad-Hoc		Geschäftsregel
	Kompensation		Service
			Skript




		
Sequenzfluss definiert die Abfolge der Ausführung	Standardfluss wird durchlaufen, wenn alle anderen Bedingungen nicht zutreffen	Bedingter Fluss enthält eine Bedingung, die definiert, wann er

Abbildung A.2.: BPMN Übersicht

A. BPMN Notation

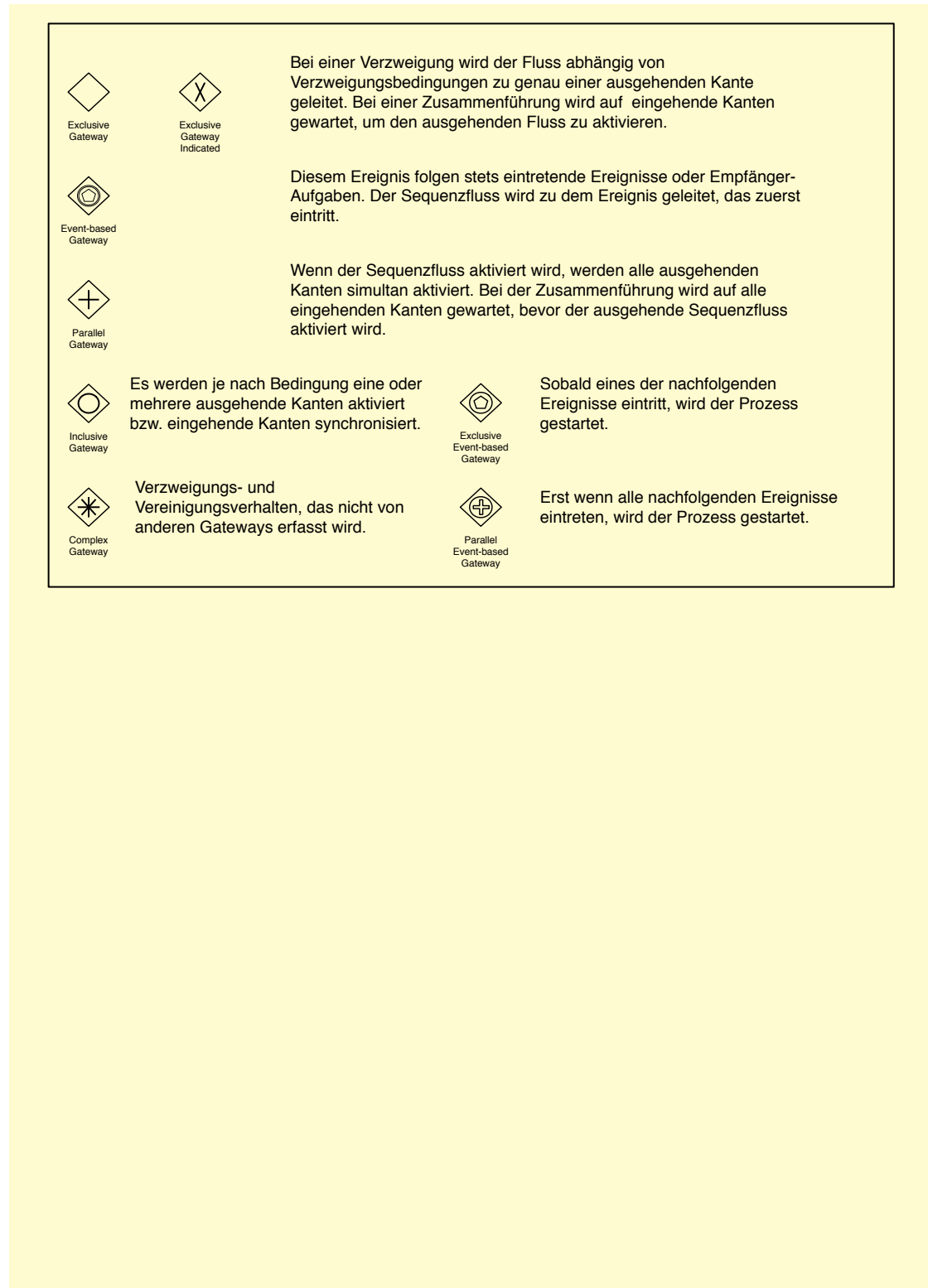


Abbildung A.3.: BPMN Gateways

B

ConDec Notation

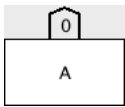
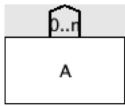
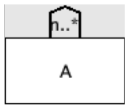
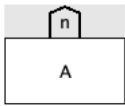
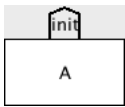
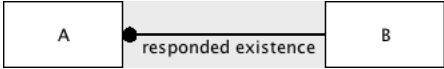
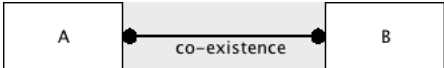
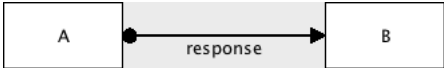
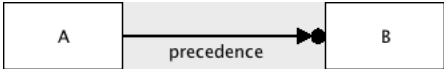
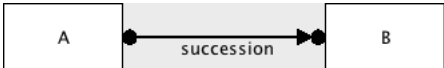
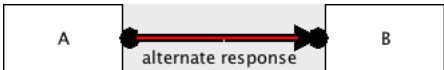
Notation	Erläuterung
	absence (A) Aktivität A darf nicht ausgeführt werden
	absence (n+1, A) Aktivität A kann höchstens n mal ausgeführt werden, aber nicht n+1-mal
	existence (n, A) Aktivität A muss mindestens n-mal ausgeführt werden
	exactly (n, A) Aktivität A muss genau n-mal ausgeführt werden
	init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

Tabelle B.1.: ExistenceConstraints ConDec

Constraint	Erläuterung
	<p>responded existence</p> <p>Falls A ausgeführt wird, muss B entweder davor oder danach ebenfalls ausgeführt werden.</p> <p>Beispiel: Korrekt: [A,B]; [B,A]; Inkorrekt:[A];</p>
	<p>co-existence</p> <p>A und B kommen in einem Pfad immer zusammen vor.</p> <p>Beispiel: Korrekt: [A,B]; Inkorrekt: [A]; [B];</p>
	<p>response</p> <p>Falls A ausgeführt wird, muss B danach ebenfalls ausgeführt werden.</p> <p>Beispiel: Korrekt: [B,A,A,A,C,B]; Inkorrekt: [B,A,A,A,C]</p>
	<p>precedence</p> <p>Falls B ausgeführt wird, muss vorher A ausgeführt werden.</p> <p>Beispiel: Korrekt: [A,C,B,B,A]; Inkorrekt:[C,B,B,A]</p>
	<p>succession</p> <p>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.</p> <p>Beispiel: Korrekt: [A,C,A,B,B]; Inkorrekt: [A,C,B,B]</p>
	<p>alternate response</p> <p>Verlangt, dass nach einer Aktivität A Aktivität B ausgeführt wird, jedoch darf vor Aktivität B nicht eine weitere Aktivität A ausgeführt werden.</p> <p>Beispiel: Korrekt: [B,A,C,B,A,B] ; Inkorrekt: [B,A,C,A,B,A,B]</p>

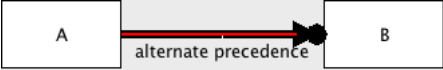
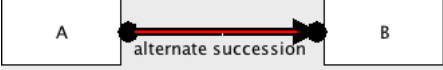
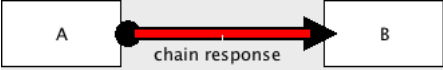
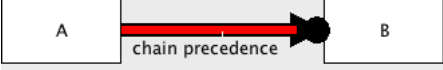
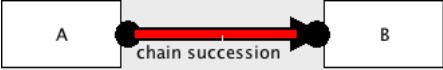
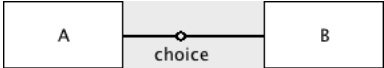
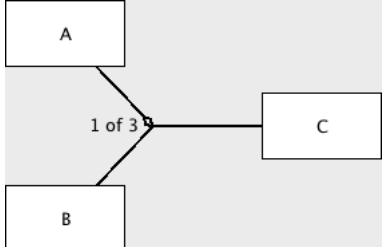
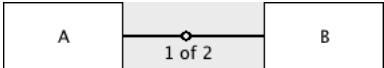
	<p>alternate precedence Verlangt, dass jeder Instanz von Aktivität B eine Instanz der Aktivität A vorausgeht. Die nächste Instanz einer Aktivität B kann somit nicht vor der nächsten Instanz von Aktivität A ausgeführt werden. Beispiel: Korrekt: [A,C,B,A,B,A]; Inkorrekt: [A,C,B,B,A]</p>
	<p>alternate succession 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]</p>
	<p>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]</p>
	<p>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]</p>
	<p>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]</p>

Tabelle B.2.: Relation Constraints ConDec

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

B. ConDec Notation

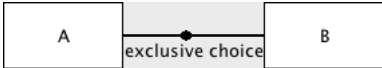
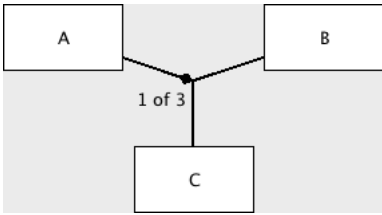
	<p>alternate precedence</p> <p>Verlangt, dass jeder Instanz von Aktivität B eine Instanz der Aktivität A vorausgeht. Die nächste Instanz einer Aktivität B kann somit nicht vor der nächsten Instanz von Aktivität A ausgeführt werden.</p> <p>Beispiel: Korrekt: [A,C,B,A,B,A]; Inkorrekt: [A,C,B,B,A]</p>
	<p>alternate succession</p> <p>Stellt eine Kombination aus alternate response und alternate precedence dar.</p> <p>Beispiel: Korrekt: [A,C,B,A,B,A,B]; Inkorrekt: [C,B,A,A,B]</p>

Tabelle B.3.: Choice Constraints ConDec

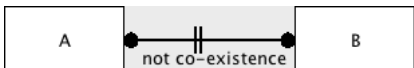


	<p>not co-existence</p> <p>Verlangt, dass falls Aktivität A ausgeführt wird, darf Aktivität B nicht mehr ausgeführt werden und umgekehrt.</p> <p>Beispiel: Korrekt: [A,C,A,A] ; Inkorrekt: [A,C,A,A,B]</p>
	<p>not succession</p> <p>Verlangt, dass falls Aktivität A ausgeführt wird, darf Aktivität B nicht 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.: Negation Constraints ConDec

[?, ?]

Abbildungsverzeichnis

2.1. Schichten des Software Engineering [?]	4
2.2. Ziele, Prozess und Prinzipien des Software Engineering [?]	5
2.3. Phasen Softwareprozess nach [?]	8
2.4. Software-Projekttypen nach [?]	9
3.1. Ziele der Prozessmodellierung nach [?]	12
3.2. Grundsatz ordnungsgemäßer Modellierung nach [?]	14
3.3. BPMN-Elemente Übersicht nach [?]	18
3.4. BPMN-Gateways	19
3.5. Deklarativer Beispiel-Prozess	21
4.1. Sigantio Process Editor	25
4.2. Sigantio Simulation	26
4.3. Declare Systemarchitektur	27
4.4. Declare Designer	28
4.5. Declare Framework	29
4.6. Declare Worklist	29
6.1. Scrum Überblick nach [?]	39
6.2. Imperative Modellierung Scrum	42
6.3. Imperative Modellierung Scrum Unterprozess	42
6.4. Deklarative Modellierung Scrum	44
6.5. Deklarative Modellierung Scrum-Unterprozess Sprint-Planning-Meeting durchführen	44

Abbildungsverzeichnis

6.6. Vergleich der Anzahl der Elemente Scrum	45
6.7. Open Up Überblick nach [?]	48
6.8. Phasen Open UP nach [?]	52
6.9. Rollen in Open UP nach [?]	53
6.10. Phasen Open UP- imperativ	54
6.11. Phasen Open UP Unterprozess Inception- imperativ	54
6.12. Phasen Open UP Unterprozess Elaboration- imperativ	55
6.13. Phasen Open UP Unterprozess Construction- imperativ	56
6.14. Phasen Open UP Unterprozess Transition- imperativ	57
6.15. Lösungsincrement entwickeln imperativ	59
6.16. Iteration planen und managen imperativ -Inception	60
6.17. Iteration planen und managen imperativ -Inception Unterprozess Umge- bung vorbereiten	61
6.18. Anforderungen identifizieren und verfeinern-Elaboration	63
6.19. Produktdokumentation und Training erstellen	64
6.20. Release deployen-Transition	65
6.21. Phasen Open UP- deklarativ	66
6.22. Phasen Open UP Unterprozess Inception- deklarativ	67
6.23. Phasen Open UP Unterprozess Elaboration- deklarativ	68
6.24. Phasen Open UP Unterprozess Construction- deklarativ	68
6.25. Phasen Open UP Unterprozess Transition- deklarativ	69
6.26. Develop Solution Increment deklarativ	70
6.27. Plan and manage iteration deklarativ	71
6.28. Plan and manage iteration deklarativ	72
6.29. Deploy Release-Transition	72
6.30. Identify and refine requirements-Elaboration	73
6.31. Develop Product Documentation-Construction	74
6.32. Phasen Open UP	74
6.33. Open UP-Inception	75
6.34. Open UP-Iteration planen	76
6.35. Open UP-Inception gesamt	76

6.36. Grundstruktur V-Modell XT nach [?]	77
6.37. Projekttypen V-Modell XT nach [?]	78
6.38. Zuordnung der Projekttypvarianten zu den Projekttypen des V-Modell XT [?]	80
6.39. Vorgehensbausteine V-Modell XT nach [?]	81
6.40. V-Modell-Kern und Vorgehensbausteinlandkarte nach [?]	82
6.41. Entscheidungspunkte V-Modell XT nach [?]	85
6.42. Entscheidungspunkte für die Projektdurchführungsstrategie nach [?]	86
6.43. Systementwicklungsprojekt AG/AN V-Modell - imperativ	87
6.44. Unterprozess Inkrementelle Entwicklung durchführen V-Modell - imperativ	88
6.45. System entwerfen V-Modell - imperativ	89
6.46. System spezifizieren-imperativ	90
6.47. Systementwicklungsprojekt AG/AN V-Modell - deklarativ	92
6.48. Systementwicklungsprojekt AG/AN V-Modell Unterprozess Entwicklung durchführen - deklarativ	92
6.49. Unterprozess Inkrementelle Entwicklung durchführen V-Modell - imperativ	93
6.50. System entwerfen - deklarativ	94
6.51. System spezifizieren- deklarativ	95
A.1. BPMN Ereignisse	106
A.2. BPMN Übersicht	107
A.3. BPMN Gateways	108

Tabellenverzeichnis

3.1. Constraints ConDec	22
6.1. Iterationen und Zielstellungen der Phasen in Open UP [?]	51
B.1. ExistenceConstraints ConDec	110
B.2. Relation Constraints ConDec	112
B.3. Choice Constraints ConDec	114
B.4. Negotiation Constraints ConDec	114

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