UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

Institut für Softwaretechnik
ist

Ansätze zur Prüfung und Bewertung von Modellqualität im Bereich modellbasierter Lasten- und Pflichtenheftgenerierung

Bachelorarbeit

zur Erlangung des Grades Bachelor of Science (B.Sc.)
im Studiengang Informationsmanagement

vorgelegt von

Meike Weber

Erstgutachter:	Prof. Dr. Jürgen Ebert (Institut für Softwaretechnik, AG Ebert)
Zweitgutachter:	Mahdi Derakhshanmanesh (Institut für Softwaretechnik, AG Ebert)

Koblenz, im April 2014

ERKLÄRUNG

„Hiermit bestätige ich, dass die vorliegende Arbeit von mir selbständig verfasst wurde und ich keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet–Quellen – benutzt habe und die Arbeit von mir vorher nicht in einem anderen Prüfungsverfahren eingereicht wurde. Die eingereichte schriftliche Fassung entspricht der auf dem elektronischen Speichermedium (CD-Rom).

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☒	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☒	☐

……..
(Ort, Datum) (Unterschrift)

INHALTSVERZEICHNIS

ABKÜRZUNGSVERZEICHNIS

Abb.	Abbildung
BA	Bachelorarbeit
BPMN	Business Process Model and Notation
Bsp.	Beispiel
bspw.	beispielsweise
Def.	Definition
Doku.	Dokumentation
Engl.	Englisch
etc.	et cetera
FCM	Factor-Criteria-Metrics
GoM	Grundsatz ordnungsgemäßer Modellierung
GoMV	Grundsatz ordnungsgemäßer Modellvisualisierung
GQM	Goal Question Metric
i.S.v.	im Sinne von
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
MQ	Modellqualität
MS	Microsoft
Nr.	Nummer
OMG	Object Management Group
RE	Requirements Engineering
REQ	Anforderung
S.	Seite
SRS	Software Requirements Specification[1]
SQM	Software Quality Metrics
Tab.	Tabelle
u.a.	unter anderem
UAW	Unadjusted Actor Weights
UC	Anwendungsfall
UCP	Use Case Points
UUCP	Unadjusted Use Case Points
UUCW	Unadjusted Use Case Weights
UML	Unified Modeling Language
Z.	Zeile
z.B.	zum Beispiel

[1] Auf Deutsch: Anforderungsspezifikation

Die vorliegende Bachelorarbeit wurde in Zusammenarbeit mit der Firma MID GmbH erstellt, daher wird hier das Modellierungswerkzeug Innovator eingesetzt.

ZUSAMMENFASSUNG

Die vorliegende Arbeit beschäftigt sich mit Qualitätssicherungsansätzen für modellbasierte SRS (d.h. Lasten- und Pflichtenhefte), im Speziellen mit SRS-Modellen und SRS-Modellvisualisierungen (d.h. Diagrammen). Das Besondere an modellbasierten SRS-Dokumentationen ist, dass sie durch einen Dokumentationsgenerator aus dem Input SRS-Modell, SRS-Modellvisualisierungen und modellexternen Texten generiert werden. Um die Qualität der Dokumentation zu sichern, muss somit die Qualität der folgenden vier Faktoren gesichert werden: SRS-Modell, SRS-Modellvisualisierungen, modellexterne Texte und Dokumentationsgenerator. Ziel dieser Arbeit ist es, einen Qualitätsbegriff für SRS-Modelle und -Modellvisualisierungen zu definieren und ein Vorgehen zur Realisierung automatischer Qualitäts*prüfung*, *-messung* und *-bewertung* (d.h. Qualitätssicherung) im Modellierungswerkzeug Innovator aufzuzeigen.

ABSTRACT

This thesis deals with quality assurance of model-based SRS, in particular SRS-Models and SRS-Diagrams. The interesting thing about model-based SRS is that they are generated by a documentation generator based on the following input data: SRS-Model, SRS-Diagrams and texts external to the model. Therefore to assure the quality of the documentation the quality of their four factors must be assured, which are the SRS-Model, SRS-Diagrams, external texts and the documentation generator. The thesis' goal is to define a quality connotation for SRS-Models and -Diagrams and to show an approach for realizing automatically quality testing, measurement and assessment for the modelling tool Innovator.

1 EINLEITUNG

Seit vielen Jahren werden modellbasierte Techniken in der Softwareentwicklung eingesetzt. Bspw. wird Code aus Modellen generiert, und es werden modellbasierte Softwaretests durchgeführt. Ein weiteres Einsatzgebiet von Modellen stellt die modellbasierte Generierung von Anforderungsspezifikationsdokumenten dar. Hier wird ein Modell zur Dokumentation und Definition der Wünsche, Vorstellungen und Bedingungen (d.h. Anforderungen) des Auftraggebers an das zu erstellende Softwaresystem verwendet.

Die vorliegende Arbeit beschäftigt sich mit Möglichkeiten der Qualitätssicherung von modellbasierten Anforderungsspezifikationen[2] (engl.: Software Requirements Specification, kurz: SRS).

1.1 PROBLEMSTELLUNG UND MOTIVATION

Das *SRS-Modell* ist der zentrale Betrachtungsgegenstand dieser Arbeit. Neben ihm werden zusätzlich die Visualisierungen dieses Modells untersucht, die Diagramme. Zentral für diese Arbeit ist die Differenzierung zwischen dem Modell und den Modellvisualisierungen. Das Modell besteht aus Modellbestandteilen📖, welche durch Beziehungen und Abhängigkeiten miteinander verbunden sind und infolgedessen ein Modell bilden. Diagramme zeigen bestimmte Sichten📖 und Ausschnitte eines Modells, bspw. wird in Abb. 1-5 (S. 5) der Ausschnitt „Urlaubsanträge verwalten“ dargestellt. Dieses Diagramm zeigt als Sicht das Verhalten der Urlaubsverwaltungseinheit. Als *SRS-Modell* wird in der vorliegenden Arbeit ein Modell bezeichnet, dessen Inhalte den Zweck erfüllen, auf Grundlage dieses Modells eine modellbasierte SRS-Dokumentation generieren zu können.

Am Anfang eines klassischen Softwareentwicklungsprojekts steht – nachdem das Problem identifiziert wurde, welches es zu lösen gilt – die Erhebung der Anforderungen an die zu erstellende Lösung, die Anforderungsanalyse (engl.: Requirements Engineering, kurz: RE). Die Ergebnisse dieser Softwareentwicklungsphase werden in einer SRS festgehalten. Auftraggeberseitig wird in Deutschland das SRS-Dokument in Deutschland auch als Lastenheft, auftragnehmerseitig als Pflichtenheft bezeichnet (siehe Abschnitt 2.3).

In dieser Arbeit wird von dem nachstehenden Szenario als Problemstellung ausgegangen. Im Szenario werden die zentralen Aktivitäten für die Erstellung eines SRS-Modells erläutert. Ferner werden die wichtigsten Informationen, welche für die modellbasierte Lastenheftgenerierung im Modellierungswerkzeug Innovator hinterlegt werden müssen, benannt. Zur Verbildlichung werden Ausschnitte aus dem Modellierungswerkzeug und einer Beispiel-Dokumentation sowie beispielhafte Diagramme gezeigt. Das Szenario basiert auf dem V-Modell XT (siehe Abschnitt 2.3). Die SRS-Dokumentation geschieht modellbasiert.

[2] Bzw. Lasten- und / oder Pflichtenheften (siehe Abschnitt 2.3).

Das dieser Ausarbeitung zugrundeliegende Anwendungsszenario sieht wie folgt aus:

> *Ein Softwarehaus 🕮 (Auftragnehmer) soll von einer Firma (Auftraggeber) beauftragt werden, für die bestehende Systemlandschaft eines Großunternehmens[3] eine neue Systemkomponente (oder ein Teilsystem) zu entwickeln.*

Grundlage des Vertrages zwischen den beiden Parteien ist u.a. ein modellbasiertes Lastenheft. In diesem Dokument wird eine fachliche Lösung des Problems sowie alle Anforderungen an diese Softwarelösung beschrieben. Um das SRS-Dokument aus dem SRS-Modell zu generieren, hinterlegt ein Anforderungsanalytiker[4] als Vertreter des Auftraggebers alle relevanten Informationen in einem fachlichen Modell (siehe Abb. 1-1) in Innovator[5].

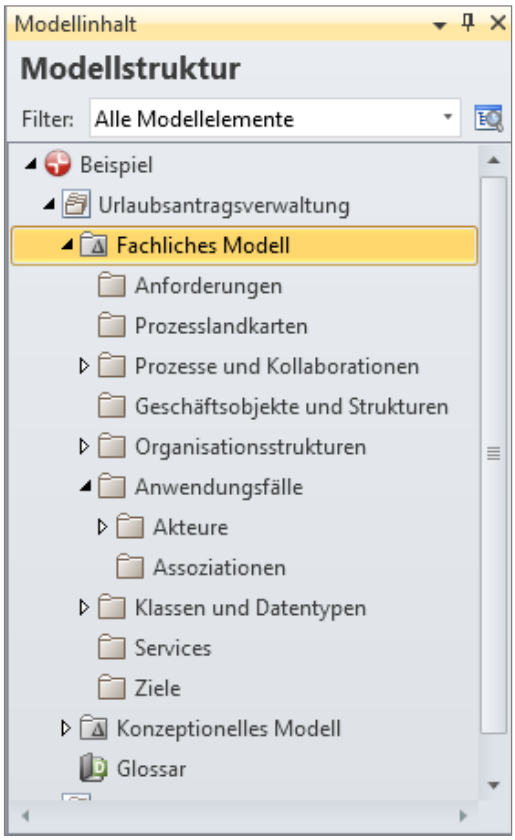


Abb. 1-1: Modellstruktur eines fachlichen Modells in Innovator.

Zu diesen Informationen zählen die Anforderungen bezüglich der angestrebten Softwarelösung. Sie sind der wichtigste Inhalt der SRS und somit auch des Lastenhefts. Eine Anforderung ist in Innovator ein Modellelement, welches in erster Linie den Text der Anforderung als Fließtext enthält und einen Namen trägt (z.B. „Muss – Urlaubsanträge verwalten (Vorgesetzter)“, siehe Abb. 1-3, S. 4). Des Weiteren hat eine Anforderung zusätzliche Eigenschaften z.B. eine Quelle, einen Ansprech-

[3] Def. Großunternehmen: Unternehmen mit über 250 Beschäftigten oder über 50 Millionen Euro Jahresumsatz (Statistisches Bundesamt).

[4] Der Anforderungsanalytiker fungiert als Schnittstelle zwischen den Stakeholdergruppen des Softwareentwicklungsprojektes (z.B. IT, Management und Fachabteilung). Er ist für das Erheben, die Prüfung, Abstimmung und Dokumentieren der Anforderungen zuständig. (Grechenig, et al., 2010)

[5] Wahlweise *Innovator for Business Analyst* oder *Innovator Enterprise Modelling Suite*.

partner, zugeordnete Stakeholder und eine Priorität. Diese Informationen werden in den Eigenschaftsfeldern hinterlegt (siehe Abb. 1-2, S. 3). Darüber hinaus können einer Anforderung untergeordnete Anforderungen zugeordnet werden. In Abb. 1-3 (S. 4) links unten ist z.B. die untergeordnete Anforderung „REQ7“ aufgeführt. Die wichtigsten Informationen im SRS-Modell sind die Zuordnung jeder Anforderung zum jeweiligen Modellbestandteil, welches durch sie beeinflusst oder bestimmt wird, z.B. ein Anwendungsfall oder ein Prozess. So können Auswirkungen einer Anforderungsänderung nachverfolgt werden: Auf welche Aspekte des SRS-Modells wirkt sich die jeweilige Änderung aus? In die andere Richtung können für jedes Modellelement alle relevanten Anforderungen ausgelesen werden. Dadurch kann in der generierten Dokumentation bspw. bei einem Anwendungsfall auf die zugehörigen Anforderungen verwiesen werden.

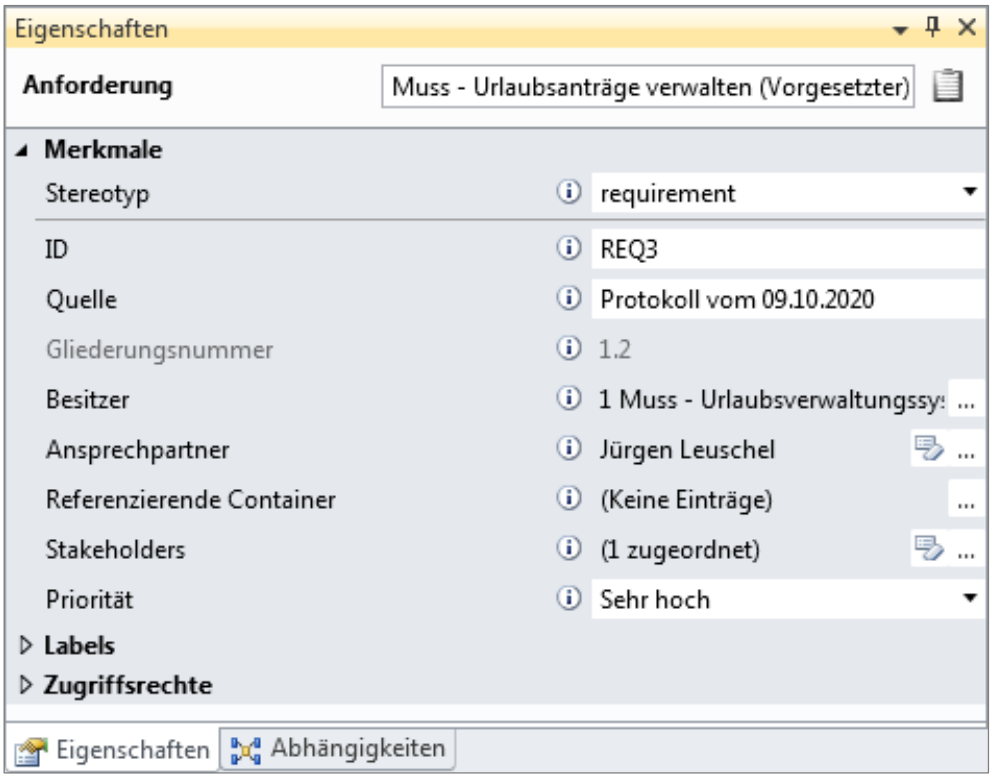


Abb. 1-2: Eigenschaftsfelder einer Anforderung in Innovator.

Neben den Anforderungen modelliert der Auftraggeber die gewünschte Funktionalität des Softwaresystems als Anwendungsfälle, setzt diese Fälle zueinander in Beziehung und ordnet ihnen die beteiligten Akteure zu (z.B. „Urlaubsanträge verwalten“, siehe Abb. 1-4, S. 4). Diese Zusammenhänge können in Anwendungsfalldiagrammen dargestellt werden. Um genauer das gewünschte Verhalten und die Interaktion des Systems mit den Akteuren zu beschreiben, modelliert er das Verhalten des Anwendungsfalls in einem BPMN-Kollaborationsdiagramm (z.B. „Urlaubsanträge verwalten“, siehe Abb. 1-5, S. 5). Die so definierte Kollaboration[6] hinterlegt er beim zugehörigen Anwendungsfall, um die Information im Modell festzuhalten, dass diese Kollaboration das Verhalten dieses Anwendungsfalls beschreibt.

[6] Eine BPMN-Kollaboration beschreibt die Interaktion zwischen mehreren Prozessen.

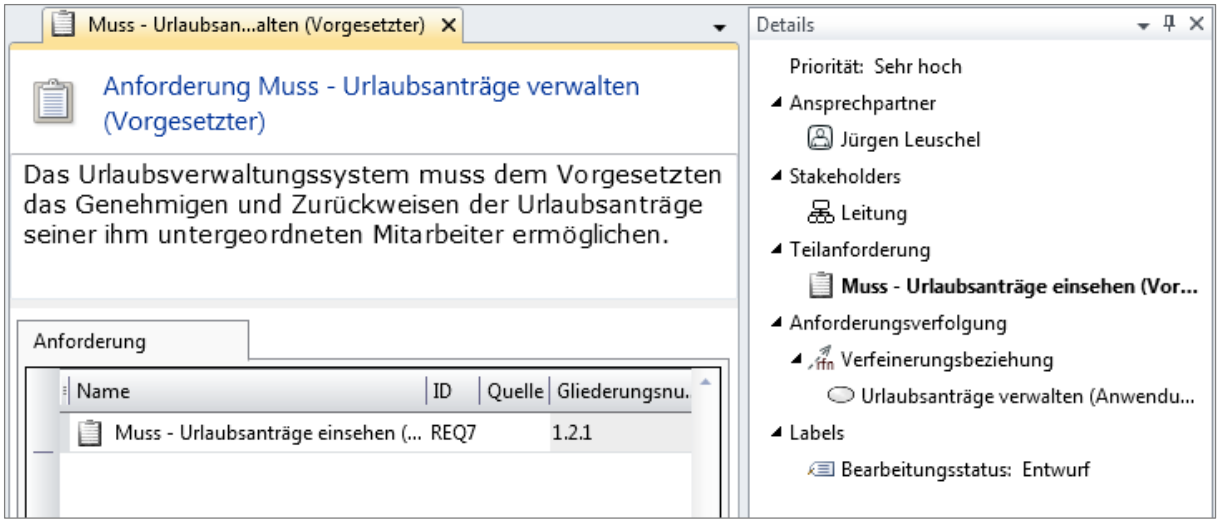


Abb. 1-3: Bsp. *Anforderung* „Muss - Urlaubsanträge verwalten (Vorgesetzter)“.

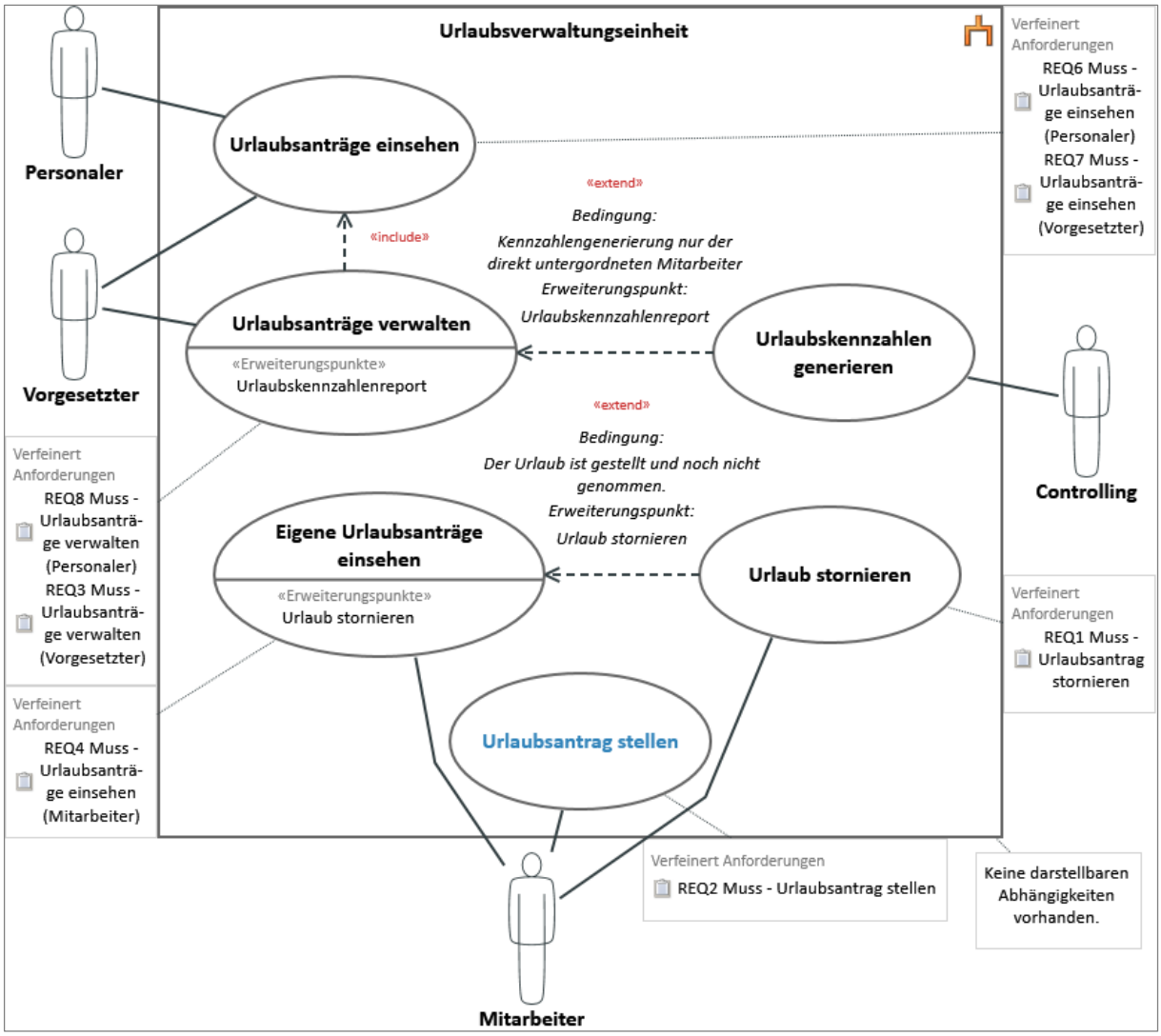


Abb. 1-4: Bsp. *Anwendungsfalldiagramm* zum Subsystem „Urlaubsverwaltungseinheit“.

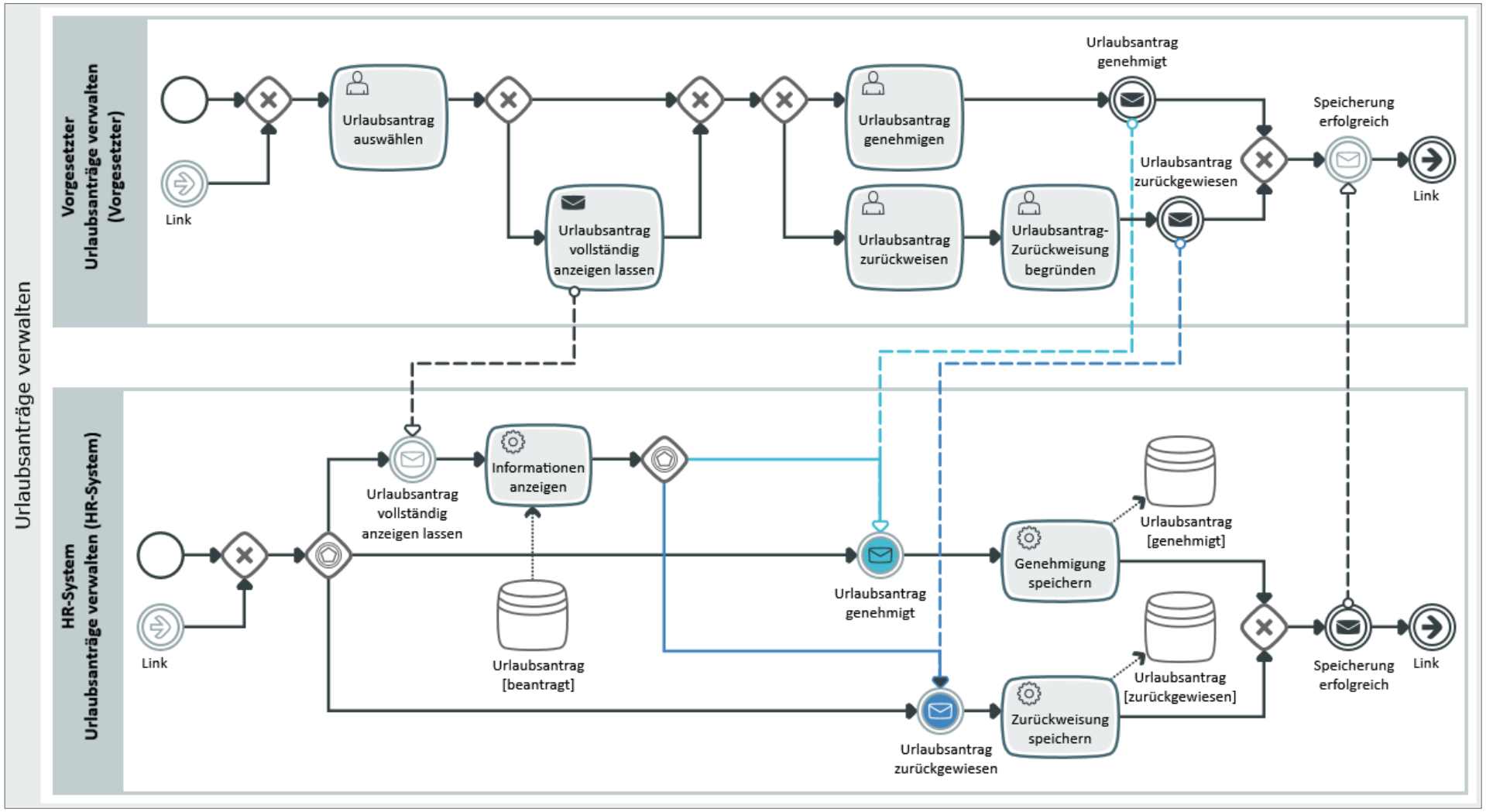


Abb. 1-5: Bsp. *Kollaborationsdiagramm* „Urlaubsanträge verwalten“.

Neben der in dieser Arbeit betrachteten Modellierung der Anforderungen, Anwendungsfälle, Kollaborationen und Prozesse werden im SRS-Modell weitere Informationen hinterlegt z.B. Datenbeschreibungen in Klassendiagrammen, Begriffsdefinitionen im Glossar, etc.

Da die neue Software in die bestehende Prozess- und Systemlandschaft des Auftraggebers integriert werden soll, müssen zusätzlich die Schnittstellen zu der bestehenden Software und Hardware oder auch zu bestehenden, am neuen System beteiligten oder für dieses relevanten Geschäftsprozesse beschrieben werden. Wir nehmen an, dass der Auftraggeber bereits seine aktuelle System- bzw. IT-Landschaft vollständig in einem Modell des Modellierungswerkzeugs hinterlegt und spezifiziert hat. Somit kann er aus seinem IT-Systemlandschaftsmodell die für den neuen Auftrag relevanten Informationen zu Schnittstellen etc. für sein SRS-Modell übernehmen, indem er die relevanten Modellbestandteile als Modellfragment in sein SRS-Modell importiert (siehe Abb. 1-6).

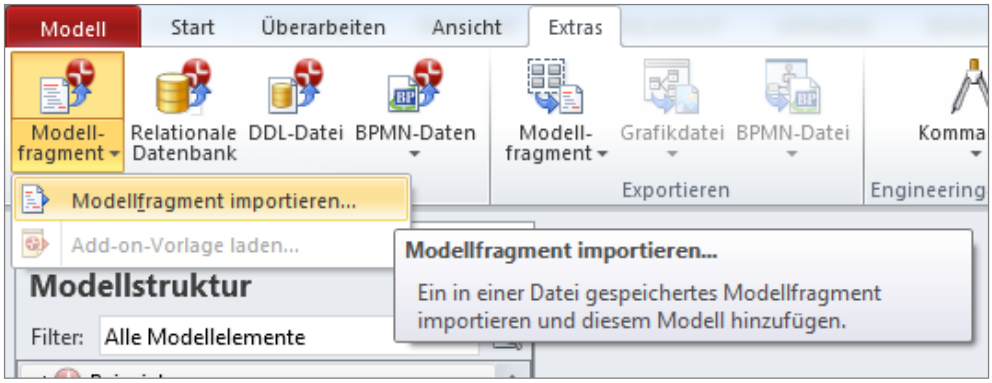


Abb. 1-6: Funktionalität Modellfragment-Import des Innovators.

Aus dem so entstandenen *SRS-Modell* kann zu jedem Zeitpunkt aus Innovator heraus eine Dokumentation mit standardisiertem Aufbau und gleichbleibender Struktur generiert werden: Die modellbasierte Anforderungsspezifikation.[7] Dies kann das Lasten- oder auch das Pflichtenheft sein (siehe Abschnitt 5.1.1, „Einsatz von Innovator für die Erstellung eines modellbasierten Lastenhefts nach V-Modell XT“). In Abb. 1-7 (S. 7) sind Ausschnitte einer auf diese Art generierten Dokumentation zu sehen.

(IT-Beauftragte der Bundesregierung, 2006; Rausch, et al., 2005; WEIT e.V., 2006)

Zusammenfassend ist ein Vorteil modellbasierter SRS, dass jederzeit aus den Informationen des Modells eine Dokumentation des aktuellen Projektstands mit geringem Aufwand erstellt werden kann. Auch unterschiedliche Arten von Dokumentationen mit bspw. unterschiedlicher Detaillierung oder Schwerpunktlegung sind möglich (durch unterschiedliche Dokumentationsausgabeformate und -arten).

Im Gegensatz zum Pflegen der Anforderungen im Anforderungsmanagementwerkzeug, der Diagramme im Modellierungswerkzeug und der Dokumentation im Texteditor, können durch die Zusammenführung und Pflege aller Informationen in einem einzigen Modell des Modellierungswerkzeugs Werkzeugbrüche innerhalb der

[7] Die Funktionalität Dokumentationsgenerierung des Innovators wird kurz in Abschnitt 5.1.1 erläutert.

InoX-Modell Beispiel

1 Anforderungen

(...)

1.2 Muss - Urlaubsanträge verwalten (Vorgesetzter)

... ist noch auszufüllen ...

Stereotypeigenschaften

Stereotypeigenschaft	Art	Wert
Priorität	Aufzählung	Sehr hoch

Das Urlaubsverwaltungssystem muss dem Vorgesetzten das Genehmigen und Zurückweisen der Urlaubsanträge seiner ihm untergeordneten Mitarbeiter ermöglichen.

ID

REQ3

Verfeinerung

- Urlaubsanträge verwalten (23)

(...)

4 Anwendungsfälle

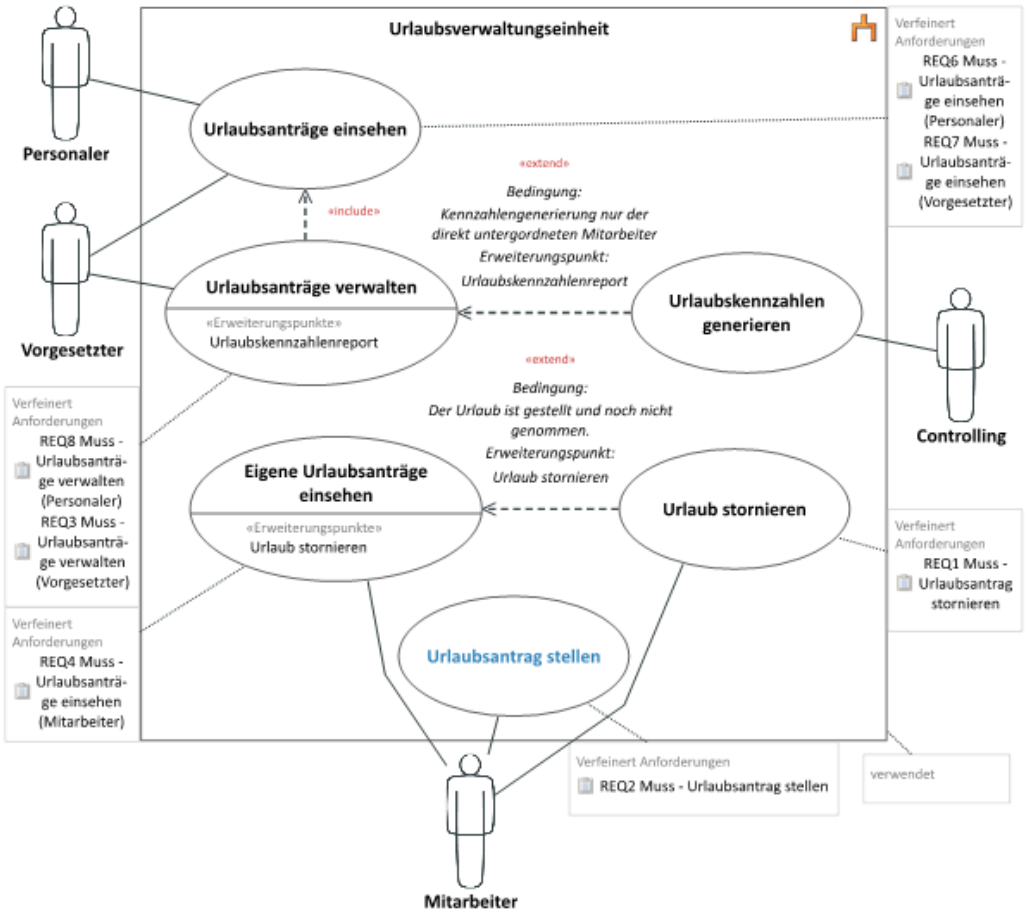


Abbildung 2: Anwendungsfalldiagramm 'Urlaubsverwaltung'

Abb. 1-7: Ausschnitte einer modellbasierten Bsp.-Dokumentation.

Anforderungsanalysephase vermieden werden. Des Weiteren ist es in diesem Werkzeug möglich, Softwaresysteme phasenübergreifend zu entwickeln (siehe Abschnitt 5.1). In jeder Phase kann auf dem bereits erarbeiteten Modell der vorherigen Phase aufgebaut werden. Daraus folgend können Dokumentationswerkzeugbrüche im Software-Lebenszyklus🕮 umgangen werden. U.a. aus diesen Vorzügen motiviert wird in dieser Arbeit das Thema modellbasierte SRS betrachtet.

In der Softwareentwicklung ist unumstritten, dass das Sichern der Softwarequalität in frühen Stadien der Entwicklung um ein Vielfaches günstiger ist als zu späteren Zeitpunkten. Frühe Fehler, die unentdeckt bleiben, werden von Phase zu Phase weitergereicht, weiterentwickelt und landen zuletzt im Softwareprodukt. Zum einen ist die Behebung des Fehlers in späteren Phasen mit einem höheren Aufwand verbunden und daher teurer, zum anderen potenziert sich dieser womöglich über die Phasen. Durch diese beiden Phänomene entsteht ein sogenannter Schneeballeffekt. Folglich ist die Qualitätssicherung modellbasierter SRS als Thema dieser Arbeit ein wichtiges Untersuchungsgebiet. (Rupp, et al., 2009)

Kostengünstiger ist nicht nur die frühe im Gegensatz zur späten Qualitätssicherung, sondern auch die „automatische“ (d.h. werkzeuggestützte) Softwareprüfung im Vergleich zur „manuellen“ (d.h. durch Menschen durchgeführte). Der automatische Prüfungsansatz ist hier kostengünstiger, da er mit einem geringen Zeitaufwand verbunden ist (siehe Abschnitt 3.2.3). Zudem sind automatische Prüfungen objektiv und identisch reproduzierbar, welches sie erstrebenswert macht (siehe Abschnitt 3.2.2). Diesem Aspekt trägt die vorliegende Arbeit Rechnung, indem speziell die automatische Prüfung, Messung und Bewertung von *SRS-Modellen* vorangetrieben und diesbezüglich ein Vorgehen erarbeitet wird.

1.2 Zielsetzung der Arbeit

Ziel dieser Arbeit ist es, einen Lösungsansatz zur automatischen Qualitätssicherung von *SRS-Modellen* und *-Modellvisualisierungen* (d.h. Diagrammen) aufzuzeigen. Unter automatischer Qualitätssicherung fassen wir im Kontext dieser Arbeit die folgenden Punkte zusammen: Automatische Qualitätsprüfung, -messung und -bewertung. Mit Prüfung ist die Ermittlung eines Wahrheitswertes bezüglich der Einhaltung einer Regel gemeint. Messen bedeutet, Eigenschaften numerisch auszudrücken, und Bewertung ist die Zuweisung von „Schulnoten“.

Im Rahmen der Arbeit sollen folgende Fragen geklärt werden, welche gleichzeitig unsere Teilziele festlegen:

- **Z1**: Welche im Betrachtungsgegenstand nachweisbaren Aspekte definieren die *Qualität des SRS-Modells* und *der -Modellvisualisierungen*?
- **Z2**: Welche dieser Aspekte können automatisch überprüft werden?
- **Z3**: Wie können Qualitätsdefizite des *SRS-Modells* und der *-Modellvisualisierungen* automatisch **gefunden** werden?
- **Z4**: Wie können Qualitätsdefizite des *SRS-Modells* und der *-Modellvisualisierungen* automatisch **gemessen** werden?
- **Z5**: Wie können mittels der Messergebnisse automatisch Aussagen über die *Qualität des SRS-Modells* und *der -Modellvisualisierungen* getroffen werden?

1.3 Methodisches Vorgehen und Aufbau der Arbeit

Um über Qualität sprechen zu können, muss im ersten Schritt der Begriff Qualität für die zu betrachtende Materie definiert werden. Hierzu wurde eine Literaturrecherche durchgeführt. Mindestens eine Qualitätsdefinition wird für jeden in dieser Arbeit betrachteten folgenden Punkte vorgestellt:

- Für ein Modell allgemein: GoM von Becker, et al. (2012)
- Für eine Modellvisualisierung: GoMV von Deelmann, et al. (2004)
- Für einen BPMN-Prozess und eine BPMN-Kollaboration: 3QM-Framework von Overhage, et al. (2012); eCH-0158 von eCH (2013);
- Für eine Anforderung: Pohl (2010)
- Für eine SRS: IEEE Std 830-1998 von IEEE (1998)

Zentral orientiert sich diese Arbeit dabei an den „Grundsätzen der ordnungsgemäßen Modellierung“ (GoM) von Becker, et al. (2012). Aus dieser und den sechs anderen Qualitätsdefinitionen folgern wir ein Qualitätssystem für die *Qualität von SRS-Modellen* und *-Modellvisualisierungen*, welches wir nach einer Kombination der Ansätze GQM[8] und FCM[9] definieren. Es gehen dabei nur jene Qualitätseigenschaften in das System ein, die direkt im Gegenstand der Betrachtung nachweisbar (d.h. Qualitätskriterien; vgl. Z1, siehe Abschnitt 1.2) und klar von den anderen Eigenschaften abgrenzbar sind. Für jene Qualitätskriterien der untersten Ebene des hierarchischen Systems, die automatisch überprüfbar sind, werden nach GQM Metriken für die Messung der Software-Qualitätseigenschaften abgeleitet (vgl. Z4). Mittels Messvorschriften werden diese definiert.

In dieser Arbeit wird nicht nur ein Qualitätssystem erarbeitet, sondern im zweiten Schritt ein Entwurf für eine Applikation und ein Vorgehen zur Entwicklung automatischer Prüfungen, Messungen und Bewertungen von Qualitätskriterien und -faktoren vorgestellt (vgl. Z3 – Z5). Eine praktische Umsetzung des theoretischen Qualitätssystems wird somit entworfen und die Umsetzung in Innovator anhand von Beispielen erläutert.

Insgesamt gliedert sich die vorliegende Arbeit in sechs Kapitel. Im anschließenden Kapitel (Kapitel 2) werden Grundlagen für das Verständnis der in der Arbeit verwendeten zentralen Begriffe und dem theoretischen Hintergrund geschaffen. Das dritte Kapitel führt in das Thema der Arbeit ein: Modellqualität, d.h. sowohl die *Qualität des Modells* als auch jene *der Modellvisualisierungen*. Das im Rahmen dieser Arbeit entwickelte Qualitätssystem als auch der Konzeptentwurf für die Modellierungswerkzeug-Applikation zur Qualitätsprüfung, -messung und -bewertung modellbasierter SRS wird in Kapitel 4 dargestellt. Im praxisorientierten vorletzten Kapitel (5) wird das Modellierungswerkzeug Innovator vorgestellt, im Speziellen die Funktionalität der Dokumentationsgenerierung sowie verschiedene Möglichkeiten der automatischen Qualitätssicherung in Innovator. Des Weiteren wird hier die Umsetzung der Qualitätsprüfung und -messung im Modellierungs-

[8] Siehe Abschnitt 3.3.1.1.

[9] Siehe Abschnitt 3.3.1.2.

werkzeug Innovator anhand von Beispielen erläutert. Das abschließende Kapitel 6 zieht das Fazit und gibt einen Ausblick.

2 GRUNDLAGEN

In diesem Kapitel werden Grundlagen vermittelt, auf deren Verständnis die folgenden Kapitel aufbauen. Es werden die Begriffe Modell und Modellierung thematisiert (2.1). In Abschnitt 2.2 werden die Modellierungssprachen UML und BPMN vorgestellt. In diesem Zusammenhang wird auf die Notation für Anwendungsfalldiagramme der UML und für Kollaborations- und Prozessdiagramme der BPMN eingegangen. Das in dieser Arbeit vertretene Verständnis und die Bestandteile von Lasten- als auch Pflichtenheften nach dem V-Modell XT werden unter Abschnitt 2.3 dargestellt. Abschließend wird in 2.4 geklärt, was unter der Bezeichnung „Modellbasierte Anforderungsspezifikation“ im Rahmen dieser Arbeit verstanden wird.

2.1 MODELL UND MODELLIERUNG

Das Modell ist das Produkt und die Modellierung der Prozess der Erstellung eines Modells (Barnert, et al., 2003). Die Definition des allgemeinen Modellbegriffs wird durch Herbert Stachowiaks Definition geprägt (Strahringer, 2014). Er definiert ein Modell durch die drei Merkmale *Abbildung*, *Verkürzung* und *Pragmatismus*. Ein Modell ist demnach ein *Abbild* eines Originals. Das Original kann ein real existierendes oder auch ein irreales Objekt oder System sein wie z.B. ein zu entwickelndes Softwaresystem. Dieses Original wird im Modell verkürzt, d.h. es werden nur Teile des Originals dargestellt (*Verkürzung*), jene die dem Modellierer als relevant erscheinen. Zudem stellt ein Modell ein Original nur unter bestimmten Einschränkungen dar (*Pragmatismus*). Diese Einschränkungen werden grob bezüglich der folgenden Punkte gemacht:

- Die Zielgruppe der intendierten Rezipienten,
- Ein bestimmter Zeitraum, zudem das Modell gültig ist und
- Ein Zweck, für den das Modell das Orginal beschreibt.

(Thomas, 2005)

MODELLE IN DER SOFTWAREENTWICKLUNG

Das allgemeine Verständnis des Modellbegriffs lässt sich direkt auf die Softwareentwicklung anwenden. Anstelle von „Verkürzung“ wird hier oft von Abstraktion gesprochen.

In der Softwareentwicklung beschreibt das Modell ein System, welches eine *„Sammlung untereinander verbundener Teile ist“* (Brügge, et al., 2004). Jede Art von Abstraktion eines Systems ist nach ihrer Definition ein Modell. Der gelebte Prozess der Urlaubsantragstellung ist somit das System und die Beschreibung dieses Prozesses ein Modell.

Da die Beschreibung aller relevanten Aspekte durch eine einzige Sicht oft nicht möglich ist, da Systeme vielschichtig und komplex sind, wird das zu betrachtende System🕮 aus unterschiedlichen Sichten beschrieben. Eine Sicht beschreibt einen Teilbereich des Modells (Brügge, et al., 2004). Es wird häufig unterschieden zwischen den beiden Arten von Sichten Struktur und Verhalten. Sie werden in unterschiedlichen Diagrammen betrachtet.

Zusätzlich wird der Sachverhalt des Originals im Modell auf unterschiedlichen Abstraktionsebenen studiert. Die Abstraktion kann umgesetzt werden durch:

- Detaillierungsstufungen (vertikale Abstraktion, z.B. detaillierte Beschreibung einer BPMN-Aktivität bzw. BPMN-Aufrufaktivität durch einen BPMN-Prozess) oder
- Verschiedene Sichten (horizontale Sichtenbildung, z.B. BPMN-Prozesse für die Beschreibung des Systemverhaltens und Klassendiagramme für die Strukturbeschreibung).

In der Softwareentwicklung gibt es unterschiedliche Modellarten:

- *Deskriptive* Modelle *„beschreiben ein Original“* (Fieber, et al., 2008), z.B. das Modell, welches den Prozess der aktuellen Urlaubsantragsgenehmigung beschreibt.
- *Präskriptive* Modelle dienen *„zur Erstellung eines Originals“* (ebd.), z.B. das Modell, welches die zukünftige Urlaubsantragsverwaltung und damit ein zu entwickelndes Softwaresystem beschreibt (siehe Abschnitt 1.1).

Für die Notation von Modellen gibt es verschiedene Modellierungssprachen. Die grafischen Notationen der Modellierungssprache UML sind eine mögliche Form Modelle zu notieren. Modelle können jedoch nicht nur grafisch, sondern auch textuell wie bspw. natürlichsprachlich oder formal notiert werden. Die jeweilige Modellierungssprache legt einen Wortschatz und teilweise Semantik (d.h. Bedeutung) für die Wörter des Wortschatzes fest. Darüber hinaus gibt es für viele Modellierungssprachen definierte Regeln, wie Wörter verwendet, diese zu Sätzen verbunden und Texte gebildet werden dürfen: Die Syntax.

Diagramme sind grafische Beschreibungen eines Modellausschnitts. Als Synonym für Diagramm wird in dieser Arbeit auch der Begriff Modellvisualisierung verwendet.

(Goll, 2012)

2.2 UML UND BPMN

UML und BPMN sind zwei grafische Modellierungssprachen. Sie werden in der Softwareentwicklung u.a. in der Phase der Anforderungsanalyse für die grafische Darstellung, Beschreibung und Definition des zu entwickelnden Softwaresystems, seiner Funktionalitäten, seines Verhaltens, seiner Schnittstellen etc. eingesetzt.

Die „Unified Modeling Language“ (UML) ist die Standardmodellierungssprache im Bereich der objektorientierten Modellierung. Sie wurde in den Jahren 1996 und 1997 von Booch, Jacobson und Rumbaugh entwickelt (Wikipedia, 2014). Die UML unterscheidet drei Kategorien von Diagrammen:

- *Struktur*diagramme: Klassen-, Objekt-, Komponenten-, Kompositionsstruktur-, Paket- und Verteilungsdiagramme,
- *Verhaltens*diagramme: Anwendungsfall-, Aktivitäts- und Zustandsdiagramme sowie
- *Interaktions*diagramme: Sequenz-, Kommunikations-, Zeitverlaufs- und Interaktionsübersichtsdiagramme.

In der Literatur werden die letzteren beiden Diagrammkategorien oft unter der Kategorie *Verhaltens*diagramme zusammengefasst (z.B. Rupp, et al., 2007). Von den dreizehn verschiedenen Diagrammen der UML sind die Anwendungsfalldiagramme für diese Arbeit relevant und werden daher kurz in Abschnitt 2.2.1 vorgestellt.

Im Gegensatz zu den vielen unterschiedlichen Diagrammen der UML gibt es in der „Business Process Model and Notation“ (BPMN) nur drei: Kollaborations-, Prozess- und Choreographie-Diagramme. Die BPMN wurde von der „Business Process Management Initiative“ (BPMI) mit dem Ziel entworfen, eine grafische Modellierungssprache zu entwickeln, die für alle Nutzer (d.h. Anforderungsanalytiker, technische Entwickler sowie Management) verständlich ist. Die BPMN gibt es seit 2004. Die aktuelle Version 2.0 ist 2011 erschienen. Im Vergleich zu den Diagrammen der UML lassen sich mit BPMN-Prozessdiagrammen wie mit dem UML-Aktivitätsdiagrammen Prozesse aus einer Abfolge von Aktivitäten (in UML: Aktionen) mit Daten- und Sequenzflüssen (in der UML: Kontrollflüsse) beschreiben. Für diese Arbeit sind die Kollaborations- und Prozessdiagramme der BPMN relevant, welche in Abschnitt 2.2.2 erklärt werden.

(OMG, 2011; Rupp, et al., 2007; White, 2004a; White, 2004b)

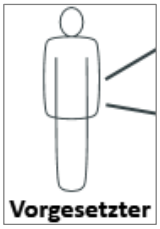
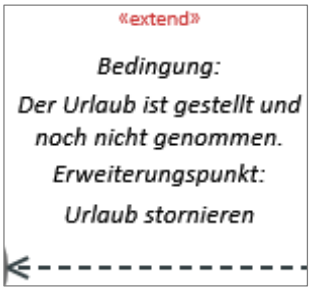
2.2.1 Anwendungsfalldiagramme der UML

Die Modellelemente der UML-Anwendungsfalldiagramme werden, zusätzlich zu der folgenden Erläuterung, auch in Tab. 2-1 (S. 14) gezeigt und kurz beschrieben.

Mit den Anwendungsfällen der Anwendungsfalldiagramme werden Funktionalitäten des Systems beschrieben, die vom System für Systemexterne (d.h. Akteure) zur Verfügung gestellten werden. Diese Funktionen können durch Akteure, die über Assoziationen dem jeweiligen Anwendungsfall zugeordnet sind, angestoßen werden. Akteure können Menschen, andere Systeme, Prozesse etc. sein. Akteure sind systemextern, Anwendungsfälle systemintern. Systeminternes wird innerhalb der Systemgrenze positioniert, welche in der Form eines Kastens dargestellt wird. Akteure befinden sich immer außerhalb der Systemgrenze.

Über Assoziationen hinaus, die dem jeweiligen Anwendungsfall seine Akteure zuordnen, können weitere Beziehungsarten mit Anwendungsfalldiagrammen modelliert werden:

- Zum einen Beziehungen zwischen Anwendungsfällen:
    - *Extend-Beziehung*: Ein Anwendungsfall wird, wenn die zugehörige Bedingung der Extend-Beziehung eingetreten ist, durch den verbundenen Anwendungsfall erweitert.
    - *Include-Beziehung*: Ein Anwendungsfall wird immer durch den verbundenen Anwendungsfall erweitert.
- Zum anderen Generalisierungen zwischen Akteuren oder zwischen Anwendungsfällen: Das Quellobjekt der Generalisierung ist eine Spezialisierung des Zielobjekts.

Objekte	
	*Anwendungsfall*: Beschreibt eine Funktionalität des Systems.
	*Akteur*: Beschreibt einen Beteiligten (ein Mensch, ein System oder Prozesse, etc.) des Anwendungsfallsystems.
Beziehungen	
	*Assoziation*: Beschreibt die Kommunikation eines Akteurs mit einem Anwendungsfall.
	*Extend-Beziehung*: Beschreibt, dass der Quellanwendungsfall den Zielanwendungsfall erweitert, wenn im Zielanwendungsfall die angegebene Bedingung eingetreten ist.
	*Include-Beziehung*: Beschreibt, dass der Zielanwendungsfall immer um den Quellanwendungsfall erweitert wird.
	*Generalisierung:* Beschreibt, dass das Quellobjekt eine Spezialisierung des Zielobjekts ist.

Tab. 2-1: Modellelemente der UML-Anwendungsfalldiagramme.

(Goll, 2012)

2.2.2 Kollaborations- und Prozessdiagramme der BPMN

Die grundsätzlichen Modellelemente von BPMN-Kollaborations- und -Prozessdiagrammen werden anhand des in Abb. 2-1 (S. 16) gezeigten Beispiels „Fahrkartenoptionen festlegen“ (ein Teilprozess von „Fahrkarte kaufen“) nachfolgend erklärt. Die für die Bachelorarbeit relevanten Modellelemente der BPMN werden zusätzlich in Tab. 2-2 (S. 17) gezeigt und kurz erläutert.

Beispiel für BPMN-Kollaborationsdiagramme

In unserem Beispiel in Abb. 2-1 (S. 16) interagiert ein Kunde mit einem Fahrkartenautomaten für den Nahverkehr. In dem Ausschnitt wird beschrieben, welche Optionen dem Kunden für die Auswahl der Fahrkartenart zur Verfügung stehen, sowie wann er welche Informationen gegenüber dem Softwaresystem (d.h. dem Fahrkartenautomaten) kommunizieren kann. Zunächst werden dem Kunden die Fahrkartenoptionen angezeigt. Er kann zwischen einem Einzel-, Mehrfach- und

Monatsticket auswählen. Nehmen wir an, er möchte ein Einzel-Ticket kaufen, so muss er diesen Wunsch über die Schnittstelle des Softwaresystems kundtun, z.B. über einen Touchscreen. Der Prozess läuft nicht weiter, solange der Kunde nicht seine gewünschte Fahrkartenart dem Fahrkartenautomaten mitteilt. Anschließend werden ihm die möglichen Zonen für die im vorherigen Schritt definierte Ticketart angezeigt. Hat der Kunde die Zone festgelegt, so ist dieser Teilprozess mit dem Endereignis beendet, dass die Fahrkartenoptionen festgelegt sind. Von dort aus wird der darüberliegende Prozess fortgesetzt.

BESTANDTEILE VON BPMN-KOLLABORATIONEN UND -PROZESSEN

Ein BPMN-Prozess beschreibt eine Abfolge von Aktivitäten (z.B. „Fahrkartenoptionen anzeigen“ in Abb. 2-1, S. 16) und Ereignissen (z.B. „Zone ausgewählt“). Welche Aktivität oder welches Ereignis nach welchem stattfindet, wird mit dem Sequenzfluss definiert. Die Interaktion von mehreren voneinander unabhängigen Prozessen wird in Kollaborationen (z.B. „Fahrkartenoptionen festlegen“) beschrieben. In den Abbildungen 2-2 und 2-3 (S. 16) ist der Unterschied zwischen Kollaborations- und dem Prozessdiagrammen zu sehen. Hier ist der gleiche Prozess einmal im Kontext einer Kollaboration (in Abb. 2-2) und einmal nur als Prozess (Abb. 2-3) dargestellt. Kollaborationsdiagramme veranschaulichen die Interaktion *mehrerer* Prozesse untereinander. Im Gegensatz dazu beschreiben Prozessdiagramme nur *einen* Prozess.

Die Kollaboration „Fahrkartenoptionen festlegen“ (in Abb. 2-1, S. 16) beinhaltet mehrere Kollaborationsbeteiligte (z.B. „Kunde“ und „Fahrkartenautomat“), die jeweils einen eigenen Prozess haben (z.B. „Fahrkartenoptionen abfragen“), der mit anderen Prozessen der Kollaboration interagiert. Ein Prozess wird in einem Pool modelliert. Ein Pool ist ein Container. Da pro Pool ein Prozess modelliert wird, ist mit dem Begriff Pool und Prozess das Gleiche gemeint.

In Abb. 2-1 wird der Prozess der Fahrkartenautomaten ausmodelliert, denn dieser interessiert uns, da er später im Softwaresystem umgesetzt werden soll. Den Prozess des Kunden stellen wir nicht dar (daher wird der Kollaborationsbeteiligte „Kunde“ in Abb. 2-1 zugeklappt gezeigt), denn wir können nicht wissen, wie dieser sich verhält. Zudem ist das Ziel des Modells, nicht dem Kunden ein bestimmtes Verhalten vorzuschreiben, sondern dem Fahrkartenautomaten. Wir zeigen lediglich die Interaktionsmöglichkeiten und somit die Benutzerschnittstelle des Kunden mit dem Softwaresystem auf. Soll dargestellt werden, dass mehrere oder unterschiedliche Parteien an **einem** Prozess beteiligt sind bzw. bestimmte Teilschritte ausführen oder für diese verantwortlich sind, wird der Prozess in Lanes aufgeteilt (siehe Abb. 2-2, S. 16). Die einzelne Lane wird mit dem Namen der zugehörigen Rolle bezeichnet. Um den jeweiligen Prozessschritt der Partei zuzuordnen, wird dieser innerhalb der zugehörigen Lane positioniert.

Aktivitäten beschreiben Prozessschritte, die an der entsprechenden Stelle im Prozess durchgeführt werden. Ereignisse illustrieren Zustände, die zu dem jeweiligen Zeitpunkt eintreten oder ausgelöst werden. Die Reihenfolge der Abfolge dieser Prozessschritte wird durch Sequenzflüsse dargestellt. In der Regel beginnt jeder Prozess mit einem Startereignis und endet mit einem Endereignis. Ereignisse, die

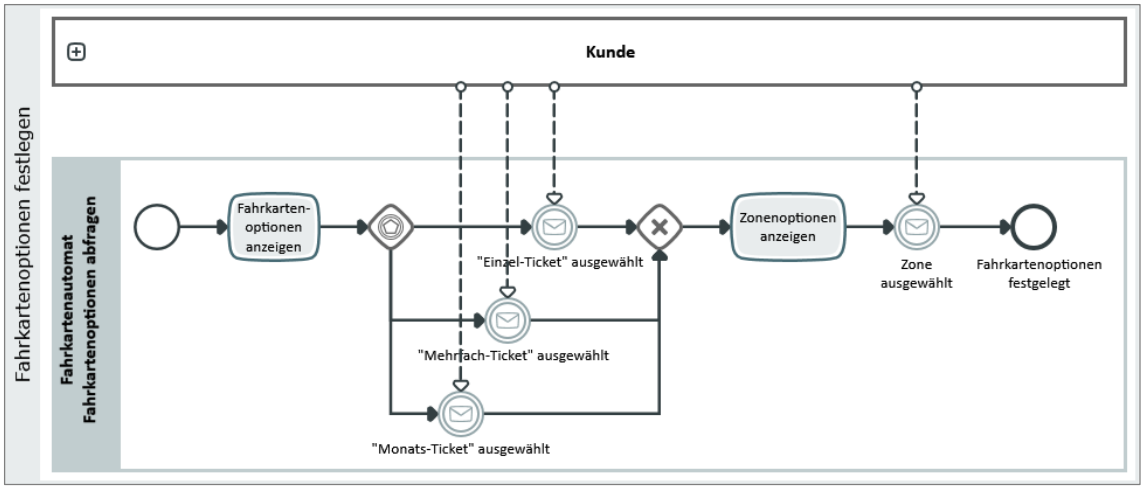


Abb. 2-1: Bsp. für eine BPMN-Kollaboration.

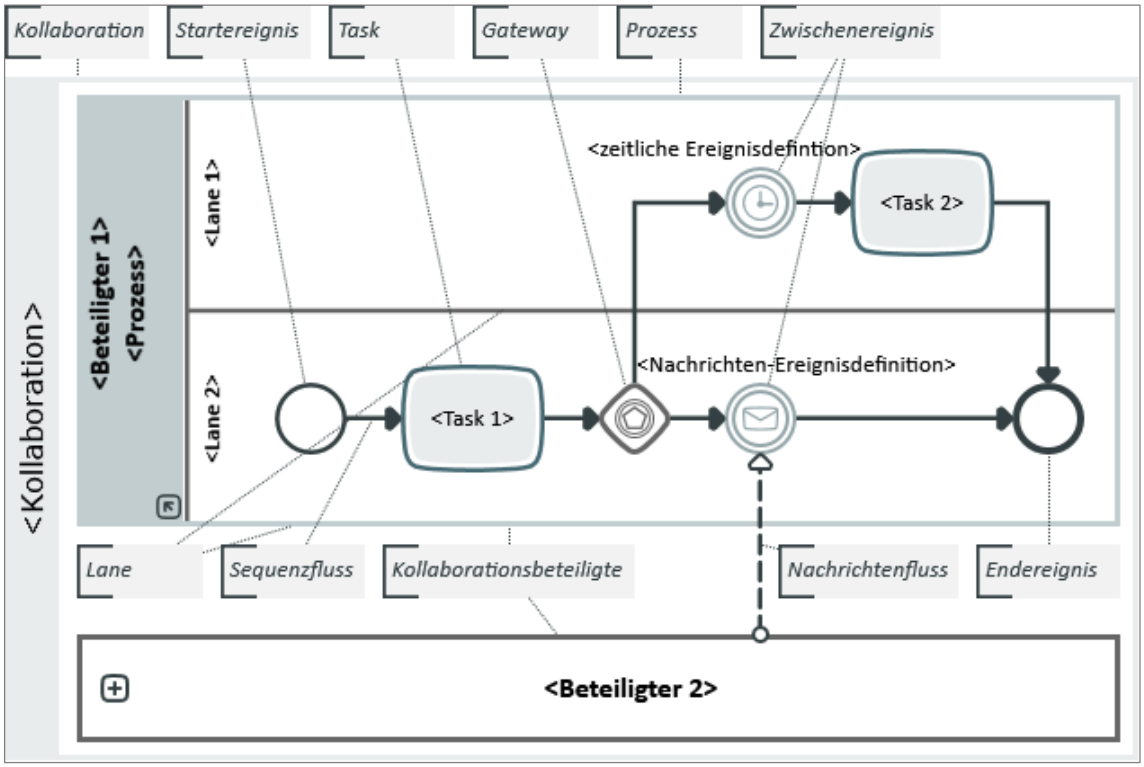


Abb. 2-2: Für die BA relevante BPMN-Modellbestandteile (Kollaborationsdiagramm).

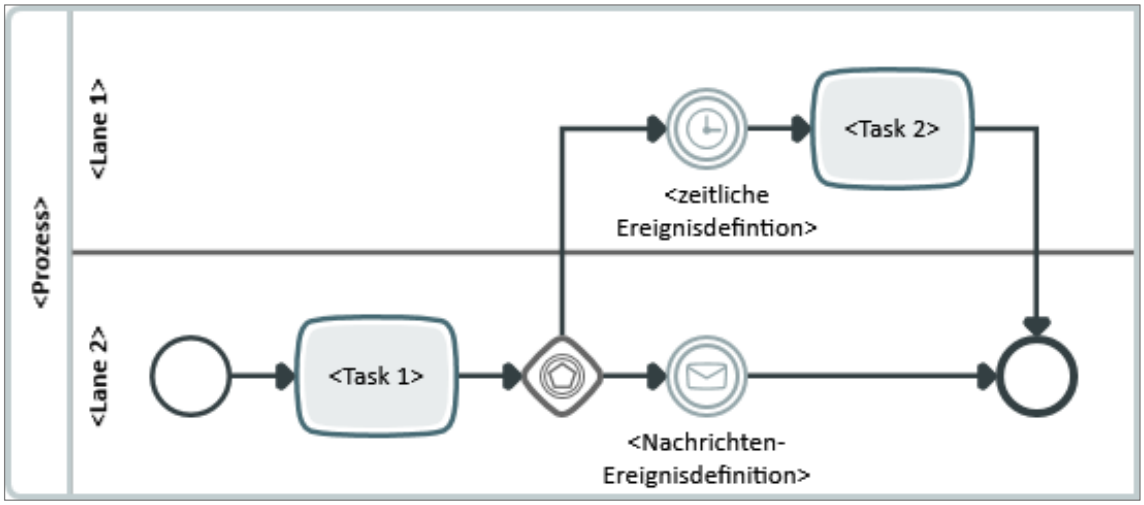


Abb. 2-3: Prozessdiagramm des Prozesses aus Abb. 2-2.

Aktivitäten	
	*Task*: Beschreibt einen einzelnen Schritt bzw. Aktivität des Prozesses.
	*Aufrufaktivität*: Beschreibt eine Aktivität, die aus mehreren Aktivitäten besteht, welche im unterliegenden Prozess beschrieben werden.
Ereignisse	
	*Startereignis*: Beschreibt oder benennt einen Startpunkt des Prozesses.
ausg.:  eintr.: 	*Zwischenereignis*: Beschreibt oder benennt ein ausgelöstes (kurz: ausg.) oder eintretendes (kurz: eintr.) Zwischenereignis.
	*Endereignis*: Beschreibt oder benennt einen Endpunkt des Prozesses.
Flüsse	
	*Sequenzfluss*: Beschreibt die Abfolge von Prozessschritten.
	*Nachrichtenfluss*: Beschreibt den Austausch von Informationen zwischen Prozessen.
Gateways	
	*Paralleles Gateway*: Beschreibt die Aufteilung des Sequenzflusses in mehrere *parallele* Sequenzflüsse, die *alle* durchlaufen werden.
	*Exklusives Gateway*: Beschreibt die Aufteilung des Sequenzflusses in mehrere *alternative* Sequenzflüssen, von denen nur *ein einziger* durchlaufen wird.
	*Inklusives Gateway*: Beschreibt die Aufteilung des Sequenzflusses in mehrere *alternative* Sequenzflüssen, von denen *einer* oder *mehrere* Sequenzflüsse durchlaufen werden.
	*Ereignisbasiertes Gateway*: Beschreibt die Aufteilung des Sequenzflusses in mehrere *alternative* Sequenzflüssen, von denen jene Sequenzflüsse durchlaufen werden, deren Ereignis eintritt.
Weitere:	*Komplexes / ereignisbasiert-exklusives / ereignisbasiert-paralleles Gateway.*

Tab. 2-2: Für diese Arbeit relevante Modellelemente der BPMN.

sowohl ein- als auch ausgehende Sequenzflusskanten haben, werden Zwischenereignisse genannt (z.B. „Zone ausgewählt“ in Abb. 2-1, S. 16). Ereignisse können in BPMN eine Ereignisdefinition einer bestimmten Ereigniskategorie tragen. Die Kategorien werden durch einheitlich definierte Symbole, die im Ereignisknoten zu sehen sind, unterschieden. Bspw. werden Nachrichten-Ereignisse durch einen Briefumschlag visualisiert. Ein Nachrichten-Ereignis beschreibt, dass Informationen von oder zu einem Prozessexternen übermittelt wurden. Ersteres ist ein eintretendes, letzteres ein ausgelöstes Ereignis. In Abb. 2-3 (S. 16) sind als Beispiel ein eintretendes Zwischenereignis mit zeitlicher und eines mit Nachrichten-Ereignisdefinition dargestellt.

Durch „Flüsse“ wird in der BPMN zum einen, wie bereits mehrfach erwähnt, die Abfolge der Prozessschritte veranschaulicht (Sequenzfluss), zum anderen gibt es in BPMN-Kollaborationen spezielle Nachrichtenflüsse. Letztere visualisieren die Übermittlung von Nachrichten zwischen zwei Prozessen.

Ein Sequenzfluss kann mit Gateways in mehrere Sequenzflüsse aufgespalten oder mehrere Sequenzflüsse können zu einem Sequenzfluss zusammengeführt werden. Die wichtigsten Verzweigungsarten sind: Parallele, exklusive und inklusive Gateways. Bei einem parallelen Gateway (i.S.v. einem Fork) wird jeder ausgehende Sequenzfluss durchlaufen. Gehen mehrere Sequenzflüsse in das parallele Gateway ein (i.S.v. einem Join), so wird der ausgehende Sequenzfluss erst gestartet, wenn an allen eingehenden Sequenzflüssen ein Token[10] anliegt. Das exklusive Gateway ist eine XOR-Verzweigung. Bei einem Fork kann nur einer der ausgehenden Sequenzflüsse passiert werden. Bei mehreren eingehenden Sequenzflüssen (i.S.v. einem Join) wird jedes auf einem der Sequenzflusskanten ankommende Token direkt weitergeleitet. Eine OR-Verzweigung wird in BPMN durch ein inklusives Gateway modelliert. Demnach können bei einem Fork ein oder mehrere vom Verzeigungsartenknoten ausgehende Sequenzflüsse durchlaufen werden. Abgesehen von den „klassischen“ Verzweigungsarten gibt es in der BPMN weitere Gateways wie bspw. das ereignisbasierte. Es wird eingesetzt um zu beschreiben, dass der Prozess erst weiterläuft, wenn eines der im Sequenzfluss nachfolgend modellierten Ereignisse eingetreten ist (siehe linke Raute in Abb. 2-1, S. 16).

(Freund, et al., 2012)

2.3 Anforderungsspezifikationen nach dem V-Modell XT

Eine Anforderungsspezifikation ist nach ISO/IEC/IEEE 29148:2011 *„eine strukturierte Kollektion der Anforderungen an die Software und ihre externen Schnittstellen“*[11]. IEEE 1012-2004 definiert zusätzlich, dass es sich bei der SRS um eine Dokumentation handelt.

In Deutschland gibt es die Unterscheidung der Anforderungsspezifikation des Auftraggebers, dem Lastenheft und des Auftragnehmers, dem Pflichtenheft. Im V-Modell XT wird u.a. zwischen den beiden Dokumenten „Anforderungen (Lastenheft)“ und „Gesamtspezifikation (Pflichtenheft)“ unterschieden. In dieser Arbeit werden die Begriffe Lasten- und Pflichtenheft nach der Definition des Vorgehensstandards V-Modell XT verwendet, welches nachstehend kurz vorgestellt wird.

Das V-Modell XT ist für zivile und militärische Vorhaben des Bundes der empfohlene Entwicklungsstandard, d.h. das Vorgehen muss für ein solches Projekt eingesetzt werden. Nur mit Begründung darf vom Vorgehen nach V-Modell XT abgewichen werden (IT-Stab des Bundesministerium des Innern, 2014). Das V-Modell definiert ein systematisches Vorgehen und regelt die Zusammenarbeit zwischen

[10] Ein Token ist ein fiktives Konstrukt zum Erklären des Ablaufs von Verhaltensdiagrammen. Der Token ist ein Zeiger auf den aktuellen Zustand oder Prozessschritt, in dem sich eine Instanz bspw. eines Prozesses befindet. (Staud, 2014)

[11] Übersetzung der Autorin, Original: *“structured collection of the requirements […] of the software and its external interfaces”* (IEEE, 2012)

Auftraggeber und Auftragnehmer für IT-Projekte von der Ausschreibung des IT-Projekts bis hin zum Projektabschlussbericht. Mittels „Tailoring“ kann dieses generische Vorgehensmodell auf kleine als auch große Projekte „zugeschnitten“ werden. Nach dem V-Modell und V-Modell-97 ist das V-Modell XT die dritte Version des Standards, beauftragt durch das Verteidigungs- und Innenministeriums, herausgegeben im Frühjahr 2005. Vorgehensmodelle wie das V-Modell XT stellen eine Zusammenfassung von bewährten Aktivitäten in einer definierten Reihenfolge dar. Sie halten somit erprobtes Wissen fest.

Nach dem V-Modell XT wird das Lastenheft zeitlich gesehen vor dem Pflichtenheft erstellt, daher stellen wir nachfolgend als erstes das Lastenheft (Abschnitt 2.3.1) und anschließend das Pflichtenheft (Abschnitt 2.3.2) vor.

(Rausch, et al., 2005; IT-Beauftragte der Bundesregierung, 2006)

2.3.1 LASTENHEFT

Im Lastenheft werden alle verbindlich gestellten *fachlichen* Anforderungen📖 an das zu erstellende System dokumentiert. Sie werden aus der Sicht des Auftraggebers verfasst.[12] *Fachliche* Anforderungen sind diejenigen, die von der Fachseite an das Softwareprodukt gestellt werden. Die späteren Nutzer und Anwender des Systems bilden diese Fachseite📖. Das Lastenheft ist an Auftraggeber und Auftragnehmer adressiert. Die Gruppe der Adressaten beinhaltet dabei die Fachbereiche IT, Management, Jura, Fachabteilungen und die Fachseite. Wie jede Software, so dient auch das Lastenheft der Kommunikation, in diesem Fall zwischen den Stakeholdern und den Systementwicklern (Sneed, et al., 2010).

Im Mittelpunkt des Lastenhefts stehen die fachliche Lösung der Problemstellung und die Anforderungen, welche die technische Lösung, die anschließend entwickelt wird, erfüllen muss. Das Lastenheft beschreibt, *was* das System können, nicht *wie* es dies leisten oder wie die jeweilige Funktionalität umgesetzt werden soll (letztere sind Fragestellungen, die in der anschließenden Entwurfsphase erst geklärt werden sollen). Ein Lastenheft besteht nach dem V-Modell XT aus den folgenden Punkten:

- Ausgangssituation und Zielsetzung,
- Funktionale Anforderungen,
- Nicht-Funktionale Anforderungen,
- Skizze des Lebenszyklus und der Gesamtsystemarchitektur,
- Sicherheitsrelevante Anforderungen, Risikoakzeptanz und Sicherheitsstufen,
- Lieferumfang und
- Abnahmekriterien.

Für die Ausschreibung des Softwareentwicklungsprojektes und die Vertragsgestaltung mit dem Auftragnehmer bildet das Lastenheft die Grundlage, zudem ist es nach dem V-Modell XT auch Vertragsbestandteil. In der Planungsphase der Softwareentwicklung wird auf Basis des Lastenhefts der Pro-

[12] Die Ermittlung und Dokumentation der fachlichen Anforderungen des Auftraggebers an das zu erstellende System muss nicht zwingend durch den Auftraggeber selbst erfolgen, sondern kann auch von einem externen Dienstleister durchgeführt werden.

jektplan entworfen, die Projektkalkulation durchgeführt sowie ein Auftragnehmer gesucht (d.h. die Ausschreibung wird aufbauend auf dem Lastenheft erstellt). Desweitern erstellt der Auftragnehmer aufbauend auf den Aussagen des Lastenhefts sein Pflichtenheft.

Der Zweck des Dokumentierens der Anforderungen ist nicht nur die Erstellung einer Dokumentation, welche Grundlage für weitere Dokumente und Prozesse ist, sondern auch der Erstellungsprozess selbst: Für das Dokument muss zunächst ermittelt werden, was das zu entwickelnde Softwaresystem können soll, d.h. wie die fachlichen Anforderungen der Stakeholder lauten. Diese Anforderungen oder auch Ziele, welche die Stakeholder bezüglich der Lösung ihres Problems haben, können bspw. im Widerspruch zueinander stehen. In diesem Fall muss ein Konsens gefunden werden oder sich eine der Anforderungen gegen die andere durchsetzen.

(Balzert, 2000; IT-Beauftragte der Bundesregierung, 2006; Rausch, et al., 2005)

2.3.2 Pflichtenheft

Wie bereits in Abschnitt 2.3.1 erwähnt, wird das Pflichtenheft aufbauend auf dem Lastenheft erstellt. Es ist eine genauere und ausführlichere Darstellung der bereits im Lastenheft thematisierten Inhalte. Darüber hinaus enthält das Pflichtenheft gegenüber dem Lastenheft drei zusätzliche Kapitel. Wie beim Lastenheft soll das Pflichtenheft nur beschreiben, *was* umgesetzt werden soll (Funktionen und Leistungsumfang), nicht *wie* (Entwurfs- und Implementierungsentscheidungen).

Abweichend von den Inhalten des Lastenhefts (siehe Abschnitt 2.3.1) ist im Pflichtenheft anstelle des Kapitels „Skizze des Lebenszyklus und der Gesamtsystemarchitektur“ das Kapitel „Lebenszyklusanalyse und Gesamtsystemarchitektur“ enthalten. Des Weiteren umfasst das Pflichtenheft die zusätzlichen Kapitel:

- Schnittstellenübersicht,
- Anforderungsverfolgung zu den Anforderungen (Lastenheft) und
- Anforderungsverfolgung.

Nach dem V-Modell XT wird das Pflichtenheft vom Auftragnehmer in Zusammenarbeit mit dem Auftraggeber erstellt. Für das Pflichtenheft werden alle Anforderungen aus dem Lastenheft übernommen und aufbereitet.

Das Pflichtenheft dient ebenfalls als juristisches Dokument zwischen Auftraggeber und Auftragnehmer. Es legt den Softwarelieferumfang fest. Das Pflichtenheft ist zum einen Grundlage für die in der Softwareentwicklung folgende Phase des Softwareentwurfs, zum anderen wird es als Ausgangspunkt für die Abnahme der fertigen Software in der Testphase verwendet.

(Balzert, 2000; IT-Beauftragte der Bundesregierung, 2006; Rausch, et al., 2005)

2.4 Modellbasierte Anforderungsspezifikation

In diesem Abschnitt soll der Begriff modellbasierte Anforderungsspezifikation erläutert werden. Als Einleitung werden zunächst unterschiedliche Stärkegrade der Einbeziehung von Modellen (d.h. *modellbasierte* und *modellgetriebene* Softwareentwicklung) voneinander abgegrenzt (2.4.1). Anschließend wird kurz beschrieben, wie modellbasierte SRS-Generierung funktioniert.

2.4.1 Softwareentwicklung mit Modellen

In der Theorie werden unterschiedliche Stärken der Einbindung von Modellen in den Prozess der Softwareentwicklung unterschieden: *Modellbasierte* und *modellgetriebene* Softwareentwicklung.

Modellgetrieben Softwareentwicklung

Modellgetriebene Softwareentwicklung (engl.: „Model-Driven Engineering“, kurz: MDE[13]) stellt nach France, et al. (2007) einen Softwareentwicklungsansatz dar. Hier werden abstrakte Modelle des Softwaresystems erstellt und systematisch in konkrete Implementation übersetzt. Ziel ist es, die große Lücke zwischen der Problemdefinition und der Implementation zu verkleinern oder sogar zu schließen. Die Vision von MDE ist, dass Modelle die primären Entwicklungsartefakte sind, und Softwareentwickler mittels computergestützter Techniken Modelle in lauffähige Systeme transformieren können. Des Weiteren sollen Modelle automatisch analysiert werden können. Das Modell beschreibt hierbei das System auf unterschiedlichen Abstraktionsebenen und aus unterschiedlichen Sichten. (France, et al., 2007)

Modellbasierte Softwareentwicklung

Im Gegensatz zur modelgetriebenen, werden im modellbasierten Softwareentwicklungsansatz Modelle weniger stark in den Softwareentwicklungsprozess eingebunden. Modelle sind hier selten das primäre Entwicklungsartefakt. Meistens fungieren sie als Unterstützung und Dokumentation der getroffenen Entscheidungen. Nach Bartelt, et al. (2005) werden bei der modellbasierten Softwareentwicklung alle wesentlichen Informationen in einem Modell hinterlegt (, im Gegensatz zu bspw. dokumentenbasierten Entwicklungsansätzen, bei denen die Informationen über mehrere Dokumente verstreut sind). Im Laufe des Projekts wird das Modell in einem Modellierungswerkzeug nach und nach erarbeitet und erweitert. Vorteile dieses Ansatzes sind bspw., dass bestimmte Aspekte des Modells automatisch überprüft werden können. (Bartelt, et al., 2005)

2.4.2 Modellbasierte SRS-Generierung

Im *SRS-Modell* sind alle Informationen, die zu einem Modellbestandteil oder einer Beziehung zwischen Bestandteilen des Modells im SRS-Dokument auftauchen, hinterlegt. Die Diagramme (bzw. *SRS-Modellvisualisierungen*) veranschaulichen bestimmte Aspekte des Modells visuell. Bei einer modellzentrierten SRS-Dokumentation strukturieren und gruppieren die Diagramme zudem die im Modell enthaltenen Informationen inhaltlich. Die *modellexternen SRS-Texte* sind längere Fließtexte, außerhalb des Modells verfasst, die Hintergrund- oder Zusatzinformationen zu dem zu entwickelnden System liefern, z.B. der Lieferumfang oder die Vision des Softwaresystems. Der *Dokumentationsgenerator* hat als Input das *SRS-Modell*, die *SRS-Modellvisualisierungen* sowie die hinterlegten *modellexternen Texte* (siehe Abb. 2-4). Wir nehmen an, dass er ausschließlich auf Grundlage dieses Inputs eine Dokumentation, das modellbasierte SRS-Dokument erstellt.

[13] Oder auch „Model-Driven Software Engineering“ (MDSE) oder „Model-Driven Development“ (MDD).

Wie Teile einer solchen Dokumentation aussehen können ist in Abb. 1-7 (S. 7, Abschnitt 1.1) zu sehen. Ausführlichere Informationen zu den Möglichkeiten der Dokumentationsgenerierung sind in Abschnitt 5.1.1 zu finden.

In dieser Arbeit werden im Folgenden die Qualität des SRS-Modells und der SRS-Diagramme genauer untersucht.

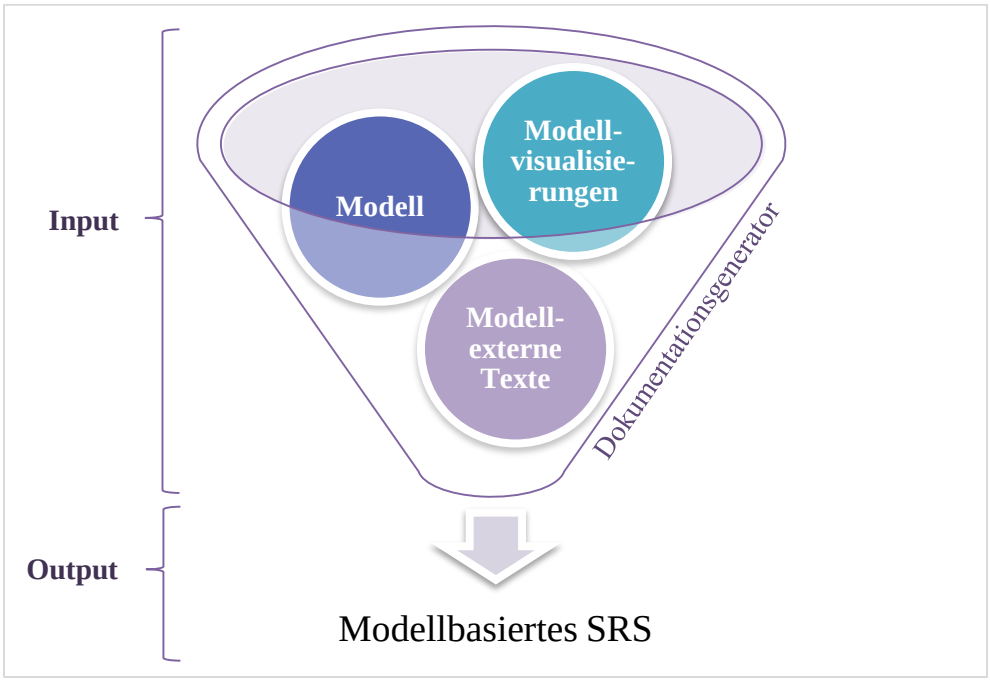


Abb. 2-4: Dokumentationserstellung modellbasierter SRS.

3 State of the Art – Modellqualität

In diesem Kapitel wird zunächst der Begriff Qualität allgemein gefasst und kurz auf den Begriff Qualitätsmodell (bzw. -system) eingegangen. Im anschließenden Abschnitt werden unterschiedliche Ansätze der Qualitätssicherung vor- und anschließend gegenübergestellt (3.2). Der für unsere Arbeit zentrale Qualitätssicherungs-Ansatz der Softwaremetrie wird in Abschnitt 3.3 gesondert behandelt. Abschließend werden verschiedene Qualitätsdefinitionen, die wir für unseren Ansatz einer Qualitätsdefinition und -bewertung von *SRS-Modellen* und *-Modellvisualisierungen* ausgewählt haben, betrachtet (3.4).

3.1 Qualität

Mehrere Normen im Bereich der Softwareentwicklung definieren den Begriff Qualität (IEEE 829-2008; ISO/IEC/IEEE 24765:2010; ISO/IEC 25010:2011; IEEE 829-2008). Von diesen Normen zeichnet sich die Norm ISO/IEC/IEEE 24765:2010 zu „Systems and software engineering“ durch ihre Ausführlichkeit und Eindeutigkeit gegenüber den anderen aus. Hier haben die „International Organization for Standardization“ (ISO), die „International Electrotechnical Commission“ (IEC) und das „Institute of Electrical and Electronics Engineers“ (IEEE) zusammen festgelegt, was unter Qualität zu verstehen ist: *„ability of a product, service, system, component, or process to meet customer or user needs, expectations, or requirements“*[14] (IEEE, 2012).

Aus dieser Definition schlussfolgernd müssen wir, um die Qualität eines SRS-Modells und von SRS-Diagrammen bewerten zu können, als erstes wissen, welche Wünsche, Erwartungen oder Anforderungen an das Modell und die Diagramme gestellt werden. Wie müssen die Qualitätsanforderungen aufstellen (siehe Abschnitt 3.4).

Qualitätsmodell

Um Qualitätsanforderungen zu differenzieren, werden in der Softwaretechnik Qualitätsmodelle verwendet. Diese bestehen aus Qualitätseigenschaften, welche hierarchisch in einem Qualitätsmodell strukturiert werden. (Fieber, et al., 2008). Im Folgenden wird für den Begriff Qualitätsmodell das Synonym *Qualitätssystem* verwendet, um Verwechslungsmöglichkeiten mit den anderen Begriffen „Modellqualität“ und „Qualität des SRS-Modells“(4.1.4) zu vermeiden:.

3.2 Ansätze der Qualitätssicherung

Für die Qualitätssicherung von Software📖 gibt es unterschiedliche manuelle sowie werkzeuggestützte Möglichkeiten. Im Allgemeinen wird bei Softwareprojekten eine Kombination aus mehreren Qualitätssicherungsmaßnahmen eingesetzt: Konstruktive und analytische, manuelle und werkzeuggestützte bzw. automatische, etc. Wir wollen in diesem Abschnitt die Unterschiede zwischen „manuellen“ und „automatischen“ Qualitätssicherungsansätzen herausarbeiten, um zu motivieren, wa-

[14] Übersetzung von Meike Weber: *Die Fähigkeit eines Produkts, Service, Systems, einer Komponente oder eines Prozesses Wünsche, Erwartungen oder Anforderungen der Kunden oder Benutzer zu erfüllen.*

rum in dieser Arbeit schwerpunktmäßig automatische Qualitätssicherung erarbeitet werden soll. Der Fokus der Betrachtung liegt hierbei auf analytischen Ansätzen.

3.2.1 „Manuelle“ Ansätze

Wird ein Ansatz zur Qualitätsprüfung oder konstruktiven Sicherung der Qualität hauptsächlich durch Menschen durchgeführt und nicht durch Werkzeuge, so bezeichnen wir diesen als „manuell“. Im Folgenden werden Modell- und Diagrammvorlagen als Beispiel für konstruktive, manuelle Ansätze der Qualitätssicherung und Reviews als Beispiel für analytische vorgestellt.

Konstruktive „manuelle“ Ansätze

Beispiele für manuelle konstruktive Qualitätssicherungsansätze sind Vorgehensmodelle für das Erstellen von Modellen sowie Modell- und Diagrammvorlagen.

Vorlagen sollen zu einer schnelleren und somit effizienteren Modellierung des eigenen Prozesses beitragen, indem Teile der Vorlage übernommen werden können. Ein Beispiel für eine Vorlage ist, ein Katalog an Standard-Anforderungen, die für jedes System dieser Art gelten und somit für das Projekt übernommen werden können. Zudem wird die Modellierung erleichtert, indem nicht auf einem „leeren“ Blatt Papier begonnen werden muss, sondern aufbauend auf einer bestehenden Modellierung nur diejenigen Sachverhalte abgeändert werden müssen, die sich im aktuellen Fall unterscheiden (siehe z.B. Abb. 3-1, eine Modellvorlage mit den Grundelementen und ihre Abfolge für den Kontext Antragstellung). Diese konstruktiven Ansätze können auch durch technische Werkzeuge unterstützt werden.

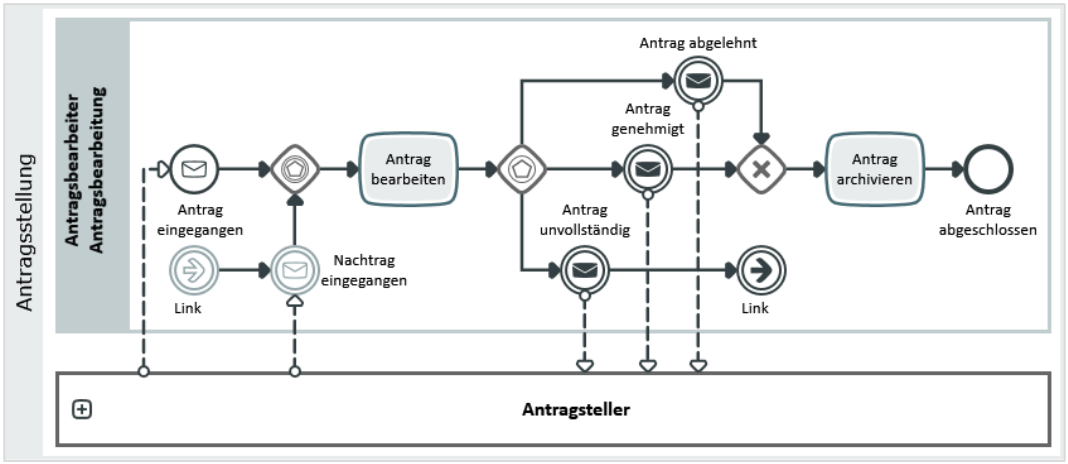


Abb. 3-1: Vorlage BPMN-Kollaboration „Antragsstellung“.

Analytische „manuelle“ Ansätze

Im Bereich der manuellen Qualitätssicherung gibt es unterschiedliche Ansätze, die unterschiedlich stark formal, sowie zeit- und ressourcenaufwendig sind. Reviews sind ein bekanntes Beispiel für diese stark unterschiedlichen Ausprägungen. Ein Review kann eher informell sein, z.B. eine einfache Stellungnahme, bis zu hochgradig strukturiert und definiert, wie eine Inspektion. Da Rupp, et al. (2004) Reviews für die analytische Qualitätssicherung von Anforderungen vorschlagen, stellen wir diese im Folgenden kurz als Beispiel für manuelle analytische Qualitätssicherungsansätze vor:

Rupp, et al. (2009) beschreiben die Qualitätsprüftechnik der **STELLUNGNAHME**. Hier wird das Prüfobjekt einem Kollegen mit der gleichen Fachlichkeit zur Überprüfung und Kommentierung gegeben. Er soll „Auffälligkeiten" im Dokument markieren, die der Autor dann anschließend bearbeitet. Hinter dem Ansatz steht das Vier-Augen-Prinzip wie auch die Erkenntnis, dass ein Externer mehr Fehler oder Auffälligkeiten entdeckt als der Autor selbst. Vorteil dieser Prüftechnik ist, dass sie keine Vorplanung und daher wenig Zeit benötigt, jedoch ist der Erfolg oder der Nutzen der Prüfung stark von den Fähigkeiten des Prüfers und den Schwerpunkten, die er bei der Prüfung setzt, abhängig.

Der **WALKTHROUGH** (bzw. das Assessment) ist eine andere Ausprägung der Klasse der Reviews. Er wird nach Rupp, et al. (2009) im RE eingesetzt, um unter allen *„Beteiligten ein gemeinsames Verständnis der Anforderungen [zu] entwickeln"* (Rupp, et al., 2009), kleinere Konflikte oder Unklarheiten zu klären und Probleme zu identifizieren. Ein Walkthrough wird vom Autor des Prüfobjekts im Beisein der „Prüfer" durchgeführt und geleitet. Schrittweise geht dieser den Prüfgegenstand durch und erläutert den Prozess zu dessen Entstehung. Die Prüfer können während des Walkthroughs Fragen stellen. Identifizierte Probleme werden, wenn sie entsprechend klein oder schnell zu klären sind, direkt gelöst und eingepflegt. Dadurch, dass der Autor selbst sein „Werk" erklärt und vorstellt, werden viele Schwächen und Fehler des Prüfgegenstands von ihm selbst aufgedeckt. Der Walkthrough ist ein Medium um kleine Verständnisprobleme und Konflikte direkt zu lösen und regt zum Gespräch und Wissensaustausch an. Nachteile dieser Reviewtechnik sind, dass der Autor von bestimmten Schwächen des Prüfgegenstands ablenken kann, da er die Prüfung leitet. Ferner wird nach Rupp, et al. (2009) bei dieser Reviewtechnik nicht kontrolliert, ob die identifizierten Probleme später behoben wurden.

Eine **INSPEKTION** ist eine klar definierte Vorgehensweise und ein Prozess zur Softwareprüfung. Hier prüfen Inspektoren anhand von Checklisten den Prüfgegenstand jeweils auf einen bestimmten Qualitätsaspekt. Dies kann in mehreren Zyklen geschehen. Zu einer Inspektion gehört eine Vorbesprechung, in der den Inspektoren kurz der Gegenstand der Prüfung erläutert wird. Anschließend bereiten sich die Inspektoren individuell auf die Reviewsitzung vor, indem sie mit Hilfe der an sie persönlich ausgehändigten Checklisten innerhalb einer vorgegebenen Zeit möglichst viele Auffälligkeiten bezüglich ihres Aufgabengebiets zu identifizieren versuchen. In der Reviewsitzung werden die Auffälligkeiten weitergegeben und protokolliert, sowie Verbesserungen für den Inspektionsprozess und die Checklisten festgehalten. Im Anschluss werden Prüfgegenstand als auch Inspektionsprozess und Checklisten entsprechend der Erkenntnisse aus der Reviewsitzung überarbeitet. Die Überarbeitung wird stichpunktartig vom Inspektionsleiter überprüft. Er ist auch derjenige, der die Inspektion bei Erfüllung der „festgesetzten Vorbedingungen" einleitet und zum Schluss überprüft, ob das sogenannte „Ende-Kriterium" erfüllt ist, sowie die Inspektion auswertet. Der Prozess der Inspektion ist klar definiert. Nach Rupp, et al. (2009) sind Vorteile der Inspektion, dass *„meist viele und tief sitzende Auffälligkeiten"* (Rupp, et al., 2009) entdeckt und die Prüfungen durch die Checklisten sehr einfach fokussiert werden können. Nachteilig ist, dass auch hier der Erfolg stark von den Fähigkeiten der Teilnehmer abhängt, sowohl dem

Inspektionsleiter als auch den Inspektoren. Zudem sind Inspektionen sehr zeit- und ressourcenintensiv und daher auch teuer.

(Rupp, et al., 2009)

3.2.2 „Automatische“ Ansätze

In diesem Abschnitt werden kurz unterschiedliche Ansätze der werkzeuggestützten oder -unterstützten bzw. „automatischen“ Qualitätssicherung vorgestellt.

Konstruktive „automatische“ Ansätze

Für konstruktive automatische Qualitätssicherung wird betrachtet, was Modellierungswerkzeuge in dem Bereich leisten können.

Modellierungswerkzeuge leisten konstruktive Qualitätssicherung dadurch, dass sie bspw. nur einen bestimmten Zeichenvorrat für einen bestimmten Diagrammtyp zur Verfügung stellen, nur bestimmte Beziehungen zwischen den Elementen zulassen oder den Modellierer zu der eindeutigen Benennung von Modellelementen zwingen (d.h. es werden keine Instanzen desselben Typs mit dem gleichen Namen in einem Modell zugelassen). Der Vorteil besteht darin, dass das Erstellen von fehlerhaften und somit qualitätsarmen Modellen nicht möglich ist. Dadurch wird viel Zeit und die damit verbundene Kosten gespart, da Fehler von vornherein nicht entstehen und somit das Modell nicht auf diese Fehler überprüft und verbessert werden muss.

Ein anderes Beispiel für automatische konstruktive Qualitätssicherung ist die Dokumentationsgenerierung. Durch die automatische Erstellung der Dokumentation können Fehler, die bei der Übertragung entstehen könnten vermieden werden und es wird viel Zeit im Gegensatz zur manuellen Erstellung der Dokumentation eingespart.

Analytische „automatische“ Ansätze

Ein Beispiel für die automatische analytische Qualitätssicherung durch Werkzeuge ist die Unterstützung oder Bereitstellung von Prüfungen. Die Prüfungen können auf im Werkzeug erstellte Objekte aufgerufen werden und überprüfen die Einhaltung von Regeln, z.B. die Überprüfung eines Modells auf die Einhaltung der Modellierungssprachen-Syntax oder projektspezifische Normen (siehe Abschnitt 5.1.2). Dem Benutzer werden nach der Überprüfung die im Prüfgegenstand gefundenen Mängel als Report ausgegeben. Auf Grundlage dessen kann er die entsprechenden Mängel beheben. Eine Überprüfung des Objekts kann somit jederzeit durch den Benutzer ausgelöst werden und hängt im Unterschied zu manuellen Ansätzen nicht von äußeren Einflüssen ab, wie bspw. bei der Stellungnahme, wo der Stellungsnahme-Gebende Zeit für die Überprüfung haben muss sowie die Überprüfung selbst einen gewissen Zeitraum in Anspruch nimmt. Des Weiteren ist jede Prüfung objektiv und reproduzierbar, d.h. wird mehrfach der gleiche Gegenstand geprüft, dann ergibt die Prüfung immer das gleiche Ergebnis. (Bartelt, et al., 2005)

3.2.3 Zusammenfassung und Gegenüberstellung

Alle manuellen Prüftechniken haben gemein, dass der Erfolg der Prüfung stark vom Können, der Erfahrung, der momentanen Konstitution und dem Wissen des

Prüfers abhängig ist. Die Prüfungen sind subjektiv, sie fallen daher unterschiedlich aus und sind zudem nicht identisch wiederholbar (d.h. reproduzierbar). Dies ist bei automatischen Ansätzen nicht der Fall.

Zudem sind manuelle Ansätze oft zeitaufwendiger als automatische. Werkzeuggestützte Qualitätssicherungsmaßnahmen wie automatische Prüfungen oder Metrikberechnung benötigen einmalig den Zeitaufwand ihrer Einrichtung bzw. Erstellung, später für die Prüfung jedoch nur noch einige Sekunden. Die anschließenden Aufwände für die Behebung der Fehler etc. sind bei beiden Herangehensweisen gleich hoch.

Nachteile bei automatischen Ansätzen sind zum einen, dass nicht alle Qualitätsaspekte über diese Art der Prüfung entdeckt und sichergestellt werden können, wie wir in Abschnitt 4.1 zeigen werden. Zum anderen entsteht beim Prüfprozess kein Austausch über die Inhalte des Prüfgegenstands wie es bspw. bei Inspektionen und Reviews der Fall ist.

Aus der soeben vorgestellten Diskussion und den darin genannten Vorteilen automatischer Qualitätssicherung folgend, liegt der Fokus der Arbeit auf der Automatisierung von Qualitätssicherung. Wir erarbeiten einen Ansatz für die automatische Prüfung, Messung und Bewertung der Qualität (siehe Abschnitt 4.2).

3.3 SOFTWAREMETRIE

Die Messung und Deutung von Softwareeigenschaften wird unter dem Titel Softwaremetrie betrachtet. In der Softwaretechnik werden sogenannte Softwaremetriken entworfen, um Eigenschaften von Software (in den meisten Fällen ist dies der Softwarecode) zu messen, vergleichen oder bewerten zu können. Hierbei wird eine Eigenschaft der Software als Zahlenwert, in diesem Kontext Maßzahl genannt, dargestellt. Durch Vergleichswerte bzw. Schwellwerte kann dem Zahlenwert eine Bedeutung zugeordnet werden, d.h. er wird interpretiert.

USE CASE POINTS

Die Use Case Points (UCP) sind ein Maß für die Größe von Anforderungsspezifikationen (Sneed, et al., 2010).[15] Sie sollen bei dem in dieser Arbeit vorgeschlagenen Vorgehen zur Entwicklung einer Modellqualitätsbewertung dazu genutzt werden, die Schwellwerte der übrigen Metriken relativ zur Größe der SRS definieren zu können und somit die *Qualität des Modells* und *der Modellvisualisierungen* bewertbar zu machen (siehe Abschnitt 4.2.4).

Die aktuellste Version der UCP sind die UCP 3.0 von Frohnhoff (2009). Anhand der Anzahl und Ausprägungen der Bestandteile des Anwendungsfallsystems und ihrer jeweiligen „Komplexität“ wird eine Zahl für die Größe berechnet. Bspw. werden die enthaltenen Anwendungsfälle in *einfach*, *mittel* und *komplex* aufgeteilt und ihnen dementsprechend Gewichte zugeordnet (siehe Tab. 3-1). Ein Anwendungsfall der Komplexität *einfach* hat nach Karner weniger als vier „Transaktionen“.

[15] Nach Frohnhoff (2009) berechnen die folgenden Wekzeuge bereits Use Case Points: Sparx Enterprise Architect, Cost Xpert Suite 3.5, Duvessa Estimate Easy Use Case.

Für den Begriff „Transaktion“ oder „Schritt“ gibt es nach Frohnhoff (2009) keine eindeutige Definition. Er definiert als Zählregel für Schritte folgendes:

„Als Schritte werden gezählt:

- *die Anzahl aller Aktionen in allen Szenarien; es handelt sich also um ein Aufsummieren über alle Szenarien, wobei Aktionen, die in mehreren Szenarien verwendet werden, nur einmal in die Zählung eingehen*
- *gleichwertig sowohl Anwenderaktionen als auch systemseitige Aktionen*
- *der Aufruf bzw. die Initialisierung eines Dialogs als ein Schritt*
- *der Aufruf einer Schnittstelle als ein Schritt“* (Frohnhoff, 2009)

In unserer Messung werden wir als *Schritte* die Anzahl der BPMN-Aktivitäten einer Kollaboration, die den Ablauf des Anwendungsfalls beschreibt, zählen (siehe Abschnitt 5.3). Weiterhin werden Frohnhoffs Definition[16] folgend nur die Aktivitäten auf der obersten Ebene gezählt, d.h. die direkt in der dem Anwendungsfall hinterlegten Kollaboration definiert sind.

Komplexität	**Definition**	**Gewichtung (Points)**
einfach	< 4 Anwendungsfall-Schritte	5
mittel	4 – 7 Anwendungsfall-Schritte	10
komplex	> 7 Anwendungsfall-Schritte	15

Tab. 3-1: Gewichtung von Anwendungsfällen nach Karner.

Die UCP setzen sich nach der Definition von Karner aus den vier Werten

- Unadjusted Use Case Weights (UUCW),
- Unadjusted Actor Weights (UAW),
- Technical Complexity Factor (TCF) und
- Environmental Factor (EF), wie folgt zusammen

➔ UCP = (UUCW + UAW) * TCF * EF.

UCCW zählt und gewichtet die Anwendungsfälle des Systems, UAW die Akteure (Menschen und andere Systeme). Die Summe aus UCCW und UAW ergibt die Unadjusted Use Case Points (UUCP). TCF und EF sind Faktoren zur Korrektur der UUCP. Die UUCP wurde im Rahmen dieser Arbeit für Innovator implementiert. In Abschnitt 5.3 erläutern wir diese Umsetzung.

SOFTWAREMAßE UND -METRIKEN

Die Verwendung des Begriffs „Softwaremetrik“ ist in der Literatur nicht einheitlich geprägt. Es gibt Autoren, die zwischen Maßen (bzw. Kennzahlen) und Metriken unterscheiden (z.B. Liggesmeyer, 2002; Sneed et al., 2010) und solche, die alle in Zahlen abgebildeten Softwareeigenschaften als Metriken bezeichnen (z.B. Lanza et al., 2006). In dieser Arbeit wird diese Unterscheidung nicht gemacht und somit

[16] *„Entscheidend für die Use Case Modellierung ist hierbei immer die Betrachtung der obersten Ebene der Zerlegung aus Sicht des Use Cases. Jede weitere Verfeinerung ist für die Schätzung nach UCP dann nicht mehr relevant. Werden in einer Spezifikation also z.B. in einer Beschreibung eines Anwendungsfalles Aktionen genannt, die wiederum aus Aktionen bestehen, so werden hier nur die Aktionen der obersten Ebene als Schritte gezählt.“* (Frohnhoff, 2009)

bei der Darstellung von Eigenschaften des Messgegenstands in Zahlen von Metriken gesprochen.

Lanza, et al. (2006) definieren den Begriff Metrik wie folgt:

> *„It is the mapping of a particular characteristic of a measured entity to a numerical value.“ (Lanza, et al., 2006)*

Demnach ist eine Metrik die Abbildung einer bestimmten Eigenschaft einer ausgemessenen Einheit auf einen numerischen Wert.

Messvorschriften definieren für Metriken, wie diese ermittelt werden sollen. Metriken können manuell oder automatisch erhoben werden.

ARTEN VON SOFTWAREMETRIKEN

Es gibt unterschiedliche Arten von Metriken, die unterschiedliche Aussagen über die Softwareeigenschaften treffen können:

- Was wird gemessen? – Kategorien nach Sneed, et al. (2010).
- Wofür wird gemessenen? – Kategorien nach Lanza, et al. (2006).
- Worüber treffen die Metriken aussagen? – Kategorien nach Liggesmeyer (2002).

Die Kategorien nach Sneed, et al. (2010) beschreiben, über welche Art von Softwareeigenschaften die jeweilige Metrik Aussagen trifft. Sie nennen sie die messbaren „Dimensionen der Substanz Software“. Daraus werden die folgenden drei Metrikarten abgeleitet:

- Quantitätsmetriken,
- Komplexitätsmetriken und
- Qualitätsmetriken.

Softwaremetriken, welche die **Größe** einer definierten Menge messen, gehören zur Gruppe der Quantitätsmetriken (z.B. Anzahl der Anforderungen). Sie treffen Aussagen über den Umfang der Software. Zählt die Metrik die **Beziehungen** zwischen Mengen und / oder Softwareelementen ist es nach Sneed, et al. (2010) eine Komplexitätsmetrik. *„Je mehr Beziehungen eine Menge hat, desto höher ist ihre Komplexität.“* (Sneed, et al., 2010). Wird die **Abweichung von einer Norm** gezählt, so handelt es sich um eine Qualitätsmetrik (z.B. Anzahl der Verstöße gegen eine Bezeichnungsregel für Anwendungsfälle). Genauer definiert der Softwarestandard ISO/IEC/ IEEE 24765:2010 eine Qualitätsmetrik wie folgt: *„Eine Funktion, deren Eingabe Softwaredaten und deren Ausgabe ein einziger numerischer Wert ist, welcher als Grad interpretiert werden kann, zu dem die Software ein gegebenes Qualitätsattribut besitzt.“*[17] (IEEE, 2012) In der folgenden Arbeit übernehmen wir die Unterscheidung von Metriken in Qualitäts- und Quantitätsmetriken nach Sneed, et al. (2010) und der ISO/IEC/ IEEE 24765:2010.

[17] Übersetzung von Meike Weber, Original:„a function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which the software possesses a given quality attribute” (IEEE, 2012).

QUALITÄTSANFORDERUNGEN AN METRIKEN

Liggesmeyer (2002) formuliert sechs Qualitätsanforderungen an Metriken:

- **EINFACHHEIT**: Das Ergebnis einer Metrik soll so einfach sein, dass der Aufwand für die Interpretation angemessen ist.
- **VALIDITÄT**: Die Metrik soll hinreichend stark mit der zu messenden Eigenschaft korrelieren.
- **STABILITÄT**: Die Metrik-Berechnung soll gleiche Werte liefern, auch wenn für die Metrik irrelevant Eigenschaften des Messobjekts verändert werden.
- **RECHTZEITIGKEIT**: Die Metrik soll ermittelbar sein zu einem Zeitpunkt, an dem auf den Messgegenstand noch eingewirkt werden kann, d.h. seine Eigenschaften änderbar sind.
- **ANALYSIERBARKEIT**: Die Maßzahlen sollen miteinander in Beziehung gesetzt und statistisch analysiert werden können. Ob dies möglich ist hängt von der Einheit der Metrik ab.
- **REPRODUZIERBARKEIT**: Wird eine Metrik für einen unveränderten Messgegenstand auf unterschiedliche Arten oder generell mehrfach ermittelt, so sollen die resultierenden Maßzahlen gleich sein. Dies setzt zum einen voraus, dass die Metriken objektiv sind und nicht durch den Messenden beeinflusst werden können, und zum anderen, dass es eine eindeutige Messvorschrift gibt.

Von Ebert (2013) wird über die Forderungen von Liggesmeyer (2002) hinausgehend gefordert:

- **EFFIZIENZ**, d.h. eine Metrik soll einfach messbar sein.
- **NORMIERUNG**, d.h. die Maßzahlen sind miteinander vergleichbar, da es eine sinnvolle Skala gibt.
- **NÜTZLICHKEIT**, d. h. die Maßzahl schafft einen Erkenntnisgewinn oder trifft Aussagen über eine Qualitätseigenschaft des Messgegenstands.

3.3.1 Auswählen von Metriken

Es gibt eine Vielzahl von Softwaremetriken. Für jede Sprache📖 können allein mindestens genauso viele Quantitätsmetriken📖, wie die Sprache Elemente hat, gebildet werden. Ebenfalls abhängig von der verwendeten Sprache gibt es Metriken, die die Komplexität (siehe Komplexitätsmetrik📖) und, bezogen auf Sollzahlen eines Projekts oder Richtlinien, die Qualität (siehe Qualitätsmetrik📖) von Software messen. Zusätzlich zu diesen Maßen gibt es eine große Menge an zusammengesetzten Metriken📖, die eine Kombination der elementaren Metriken darstellen. Auch von dieser Art gibt es eine Vielzahl von Metriken zu unterschiedlichen Fragestellungen.

Es wird in der Literatur davon abgeraten, alle möglichen, berechenbaren Metriken zu ermitteln, sondern systematische Vorgehensweisen vertreten, nach denen aus der Definition der bestimmenden Faktoren der Qualität passende Metriken abgeleitet werden. Basili, et al. (1994) nennen für das Herleiten von Metriken die Ansätze Goal Question Metric (GQM), Quality Function Deployment und Software Quality Metrics (SQM). Des Weiteren wird in der Literatur Function Criteria Metrics

(FCM) genannt. Im Folgenden werden die Ansätze GQM, SQM und FCM kurz erläutert.

3.3.1.1 GQM-ANSATZ

Bei dem von Basili, Calidiera und Rombach im Jahre 1994 entwickelten "Goal Question Metric"[18] (GQM) Ansatz werden die Metriken top-down erarbeitet. Ausgehend von einem Ziel (Goal) über eine Menge von Fragestellungen (Questions) werden die passenden Metriken (Metrics) ausgewählt und diese Metriken somit den Fragestellungen und mittelbar den Zielen zugeordnet.

Das Ziel (Goal) beschreibt in diesem Modell die konzeptuelle Ebene. *"A goal is defined for an object, for a variety of reasons, with respect to various models of quality, from various points of view, relative to a particular environment"*[19] (Basili, et al., 1994). Objekte können Artefakte📖, Prozesse📖 oder Ressourcen sein. Ein Beispiel für eine Zieldefinition, dargestellt in Form des von Basili, et al. (1994) entworfenen GQM Modells, ist in Tab. 3-2 zu finden.

GOAL	*Zielsetzung* (Purpose)	Erhöhen der
	Problem (Issue)	Übersichtlichkeit von
	Objekt (Object)	BPMN-Diagrammen
	Standpunkt (Viewpoint)	aus Sicht der Fachseite.

Tab. 3-2: Nach dem GQM-Modell aufgestellte Zieldefinition.

Auf der operationalen Ebene wird das Ziel in eine Menge von Fragestellungen übersetzt. Die Fragen beschreiben, wie das übergeordnete Ziel erreicht werden kann. Beispielhafte Fragestellungen zu dem in Tab. 3-2 formulierten Ziel sind in Tab. 3-3 zu sehen.

QUESTIONS	*Q 1*	*Wie groß ist das BPMN-Diagramm?*
	Q 2	*Wie viele unterschiedliche Typen von Notationselemente werden im BPMN-Diagramm verwendet?*

Tab. 3-3: Nach dem GQM-Modell aufgestellte Fragestellungen.

Zu guter Letzt wird jeder Fragestellung eine Menge von Metriken zugeordnet um die jeweilige Frage quantitativ zu beantworten. Auf die in Tab. 3-3 aufgeführte Fragestellung „Wie groß ist das Diagramm?" treffen die in Tab. 3-4 gelisteten Metriken Aussagen zur Diagrammgröße.

METRICS	*M 11*	Anzahl der Aktivitäten
	M 12	Anzahl der Pools
	M 13	Anzahl der Lanes
	M 14	Anzahl der Beteiligten

Tab. 3-4: Nach dem GQM-Modell aufgestellte Metriken zu Q1.

[18] (übersetzt ins Deutsche: „Ziel Fragestellung Metrik")

[19] Übersetzung von Meike Weber: „Ein Ziel ist für ein Objekt definiert, aus unterschiedlichen Gründen, bezogen auf unterschiedliche Qualitätsmodelle, aus unterschiedlichen Sichten, sich beziehend auf eine bestimmte Umgebung." (Basili, et al., 1994)

GQM bietet somit ein Schema für die strukturierte Definition von Zielen und zugehörigen Metriken.

3.3.1.2 SQM- UND FCM-ANSATZ

McCall, et al. (1977) stellen in ihrem dreiteiligen Band „Factors in Software Quality“ eine Vorgehensweise zur Quantifizierung von Softwarequalität vor. Unter Software verstehen sie alle Artefakte, die während des Softwareentwicklungsprozesses entstehen, vom Programmcode bis hin zur Anforderungsdokumentation. Wie in Abb. 3-2 dargestellt, besteht das SQM-Vorgehen zum Quantifizieren von Software aus fünf sequentiellen Schritten.

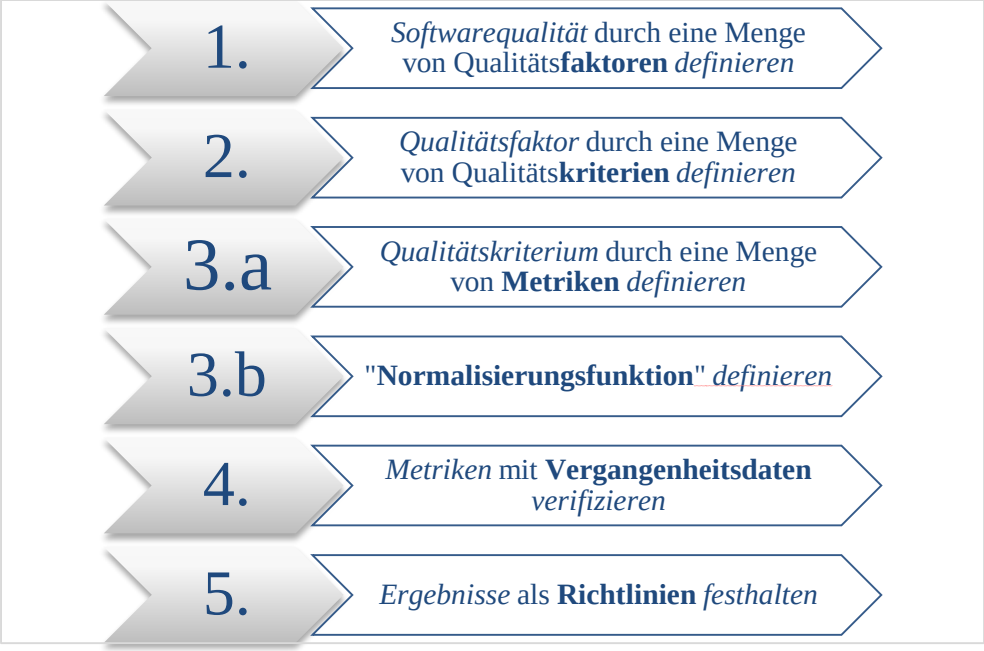


Abb. 3-2: Vorgehensschritte nach dem SQM-Ansatz von McCall, et al. (1977).

Als Erstes wird Softwarequalität definiert, indem die Qualitätsfaktoren ermittelt werden, die zusammen Softwarequalität umfassen (siehe Abb. 3-2). Qualitätsfaktoren sind allgemeine Qualitätsattribute wie beispielsweise „Korrektheit“. Im nächsten Schritt werden für jeden Qualitätsfaktor Qualitätskriterien identifiziert, die diesen definieren. Qualitätskriterien sind konkretere Qualitätsattribute wie z.B. „Konsistenz“. Anschließend werden für jedes Kriterium Metriken definiert. McCall, et al. verwenden in ihrem Beispiel auch Metriken mit nur zwei möglichen Ausprägungen („ja“ und „nein“). Ebenso wird im dritten Schritt eine „Normalisierungsfunktion“ aufgestellt. Im nachfolgenden Schritt werden die Metriken mit Hilfe von Vergangenheitsdaten verifiziert. Zum Abschluss werden die Ergebnisse in Richtlinien festgehalten.

SQM ähnelt GQM, nur dass bei diesem Ansatz an Stelle der ausführlich formulierten Ziele und Fragestellungen nur Qualitätsattribute verwendet werden. Auf der ersten Ebene wird eine Menge allgemeiner Qualitätsattribute (Qualitätsanforderung) definiert. Jede Qualitätsanforderung wird durch eine Menge von konkreteren Qualitätsanforderungen auf der zweiten Ebene definiert. Die Metriken werden der zweiten Ebene direkt zugeordnet. Auch bei diesem Ansatz bleibt offen, wie die Ziele und Metriken ausgewählt werden. Interessant ist bei SQM, dass eine Norma-

lisierungsfunktion zum Zusammenfassen der Metriken erwähnt wird, die es so ermöglicht, Aussagen über die Qualitätsanforderung zu treffen. Des Weiteren wird erklärt, dass die Metriken mit Hilfe von Vergangenheitsdaten verifiziert werden können.

Schritt 1 bis 3.a des SQM-Ansatzes sind in der Literatur auch unter dem Begriff „Factor-Criteria-Metrics“ (FCM) zu finden (z.B. in Scheible, 2010).

3.3.2 Deutung von Metriken

Der größte Zweifel an der Verwendung von Softwaremetriken scheint darin zu bestehen, inwiefern eine Softwareeigenschaft sinnvoll in einer Zahl dargestellt und dann der jeweiligen Zahl eine sinnvolle Bedeutung zugeordnet werden kann. Dem Thema der Deutung von Maßzahlen widmet sich dieser Abschnitt der Arbeit.

Lanza, et al. (2008) vertreten das Definieren von Schwellwerten für Metriken für die Deutung. Schwellwerte geben bspw. an, ab welchem Wert eine Maßzahl zu groß oder zu klein ist. Diese Werte teilen den Ergebnisbereich der Metriken in Bereiche ein und ordnen diesen Bereichen eine Bedeutung zu wie z.B. zu viel, ok und zu wenige oder hervorragend, gut, ausreichend und mangelhaft (siehe Abschnitt 4.2.4). Die Schwellwerte können auf unterschiedliche Arten definiert werden:

- Auf Grundlage von statistischen Informationen: Z.B. Mittelwert und Standardabweichung.
- Auf Grundlage von allgemein akzeptierter Bedeutung: Z.B. die Werte 0,25 / 0,33 / 0,5 / 0,66 / 0,75.

(Lanza, et al., 2006)

Aggregation mehrerer Metriken

Wie kann aus der Bewertung einer Qualitätseigenschaft oder einer Metrik die Bewertung eines Qualitätsfaktors abgeleitet werden, wenn von einem baumartigen Qualitätssystem📖 aus Qualitätseigenschaften und Metriken ausgegangen wird? Hierfür stehen unterschiedliche Ableitungsmöglichkeiten zur Verfügung:

Wurden für eine Metrik die Schwellwerte für jedes Qualitätslevel definiert, so ist der **einfachste** Bewertungsansatz für die übergeordnete Qualitätseigenschaft: Der übergeordnete Knoten im Qualitätssystem kann maximal so gut bewertet werden, wie die schlechteste Bewertung seiner untergeordneten Knoten. Folglich schlägt die schlechteste Bewertung im Bewertungsbaum bis zur Wurzel durch.

Eine andere **einfache** Möglichkeit ist, den hierarchisch übergeordneten Knoten mit dem Mittelwert der untergeordneten Bewertungen zu bewerten.

Soll bei der Bewertung eines Knotens mit einbezogen werden, dass nicht jeder der untergeordneten Knoten im Qualitätssystem gleich stark die Qualität des Modells beeinflusst, so kann eine Gewichtung der Metriken und Qualitätskriterien untereinander auf jeder Ebene vorgenommen werden. Dies ist ein **schwierigeres** Bewertungsverfahren, da entschieden werden muss, wie viel Gewicht bspw. ein *Original*

Verstoß gegen die Wortsyntax[20] gegenüber einem *Wiederholten Verstoß*[20] hat oder Verstöße gegen die Wortsyntax gegenüber jenem gegen Satz- oder Textsyntax. Es muss die Frage beantwortet werden: Ist ein Nicht-Einhalten einer Regel bezüglich des Textes im gleichen Maße qualitätsmindernd, wie wenn ein Wort der Modellierungssprache falsch verwendet wird? Womöglich gibt es für einen Knoten mehr als zwei direkt untergeordnete Knoten, welches die Erstellung von Gewichten zusätzlich erschwert. Overhage, et. al. (2012) beschreiben ein Vorgehen nach dem „Analytic Hierarchy Process“ (AHP) zur Erarbeitung von Gewichten für die Metrik- und Qualitätseigenschaften-Aggregation: Sie lassen mehrere Experten die Metriken und Qualitätseigenschaften, die zu einer Qualitätseigenschaft auf höherer Ebene zusammengefasst werden sollen, auf einer Skala[21] paarweise untereinander gewichten. Die paarweisen Vergleiche der unterschiedlichen Experten für ein Paar, werden anschließend durch die Bildung eines geometrischen Mittels vereint, wobei vorher sichergestellt wird, dass die Expertenmeinungen sich nicht grob widersprechen (genaueres siehe Overhage, et al., 2012). Um die Gewichtung mehrere Metriken untereinander aus den paarweisen Vergleichen abzuleiten, übertragen Overhage, et al. die Mittelwerte in eine Matrix und bestimmen die Gewichte anhand von Eigenwertberechnung.

Eine weitere Möglichkeit der Bewertung ist noch **komplexer**. Bei dem Bewertungsverfahren wird eine „Bewertungsformel“ aufgestellt. Diese geht auf eine der Kano-Klassifikation ähnliche Unterscheidung der Qualitätseigenschaften ein. Sie bezieht bei der Bewertung ein, ob die jeweilige untergeordnete Qualitätseigenschaft oder Metrik nur einen *qualitätsmindernden* Effekt auf den Bewertungsgegenstand hat (vgl. „Basismerkmal“), d.h. wird die Eigenschaft nicht erfüllt, verschlechtert es die Qualität und wird es erfüllt, trägt es jedoch nicht zu einer höheren Qualität bei. Alternativ können Knoten nur qualitätsfördernd (vgl. „Begeisterungseigenschaft“) oder sowohl qualitätsmindernd als auch -fördernd sein (vgl. „Leistungseigenschaft“).

3.3.3 Zusammenfassung

In diesem Abschnitt wurde in die Thematik der Softwaremetrie eingeführt. Zunächst wurde anhand der Metrik Use Case Points beispielhaft das Konzept der Metriken erklärt und anschließend auf die Begriffe Softwaremaße und -metriken eingegangen. Des Weiteren wurden die Metrikarten Qualitäts- und Quantitätsmetriken voneinander abgegrenzten und Qualitätsanforderungen für Metriken eingeführt. Wie Ansätze für das Auswählen von Metriken wurden in Abschnitt 3.3.1 vorgestellt, sowie für die Deutung und Aggregation von Metriken in Abschnitt 3.3.2.

[20] Siehe Abschnitt 4.1.4.1.

[21] Die von Overhage, et al. (2012) verwendete Skala für den paarweisen Vergleich geht von *„Kriterium A absolut dominierend“* (Overhage, et al., 2012) mit dem Wert neun über *„Beide Kriterien gleich“* (ebd.) mit dem Wert eins bis zu *„Kriterium B absolut dominierend“* (ebd.).

3.4 Qualität im Kontext von modellbasierten SRS

Im Kontext von *SRS-Modellen* gibt es mehrere Ebenen der Qualitätsbetrachtung. Wir unterscheiden in dieser Arbeit zwischen den Ebenen des in Abb. 3-3 abgebildeten sehr groben Metamodells (in blau), um die Betrachtungsebenen und somit die Teilabschnitte dieses Abschnitts zu gliedern: Zum einen kann das Modell als Gesamtobjekt untersucht werden, an das typspezifische Qualitätsanforderungen gestellt werden können. Zum anderen besteht ein solches Modell aus unterschiedlichen Teilmodellen, formuliert in unterschiedlichen Modellierungssprachen. Ein Teilmodell besteht abermals aus weiteren Teilmodellen oder aus Modellelementtypen, an die wiederum typspezifische Qualitätsanforderungen gestellt werden. Ist im Folgenden die Rede von Modellbestandteilen, so ist die Menge der Teilmodelle und Modellelemente gemeint.

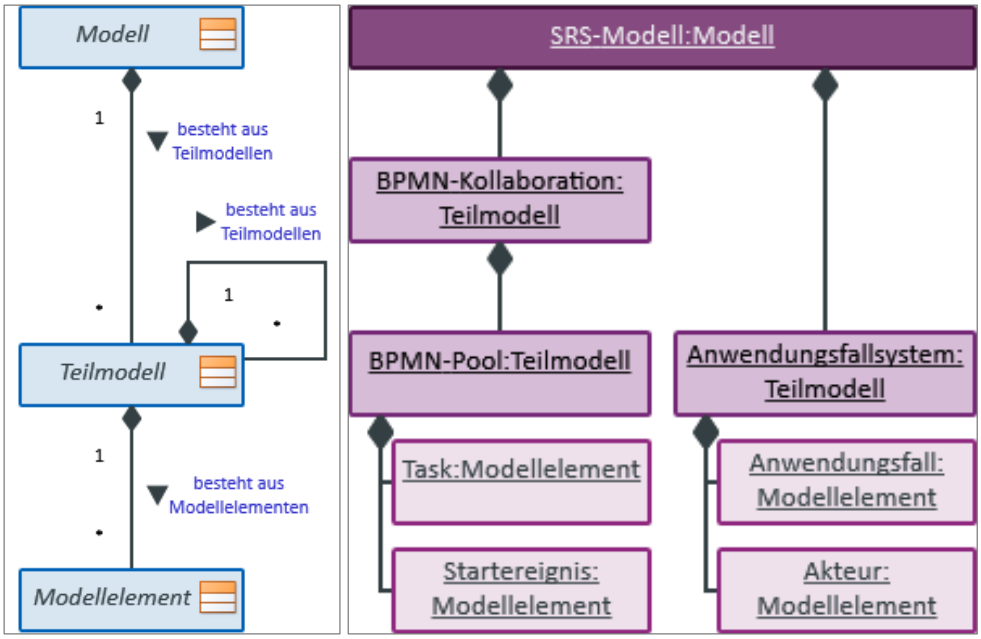


Abb. 3-3: Gegenstände der Qualitätsbetrachtung von *SRS-Modellen*, links das Metamodell und rechts Bsp.-Instanzen.[22]

In diesem Abschnitt werden Ansätze zur Definition des Qualitätsbegriffs für den Gesamtmodelltyp *SRS-Modell* (Abschnitt 3.4.1) betrachtet. Außerdem werden Qualitätskriterien für ausgewählte Teilmodelltypen und für einige Modellelementtypen des *SRS-Modells* vorgestellt (3.4.2). Zusätzlich zur Betrachtung der Qualität des Modells und der Modellbestandteile, wird der Qualitätsbegriff für Anforderungsspezifikationen allgemein besprochen (3.4.3). In Abschnitt 4.1 wird aufbauend auf diesen Definitionen ein Qualitätssystem für modellbasierte SRS hergeleitet.

Zentral orientiert sich die Arbeit an den Grundsätzen ordnungsgemäßer Modellierung (siehe Abschnitt 3.4.1.1). Um die Definitionen der Qualitätskriterien oder -an-

[22] Die Unterscheidung zwischen dem Modell sowie den Teilmodellen ist zum einen ein Konstrukt in Innovator (Dort ist ein Modell ein physisch abgeschlossenes System, aus der Sicht des Benutzers vergleichbar mit einem Word-Dokument.) Zum anderen wurde es gewählt, um in dieser Arbeit zwischen dem Gesamtobjekt Modell und seinen Teilen wie einem BPMN-Pool unterscheiden zu können.

forderungen nicht durch Paraphrasierung zu verfälschen, werden diese Definitionen fast ausschließlich wörtlich zitiert.

3.4.1 Qualität des Modells allgemein

Das Thema Modellqualität wird von einzelnen Autoren auf einer abstrakten Betrachtungsebene behandelt (z.B. Becker, et al., 2012; Deelmann, et al., 2004). Der wichtigste und bekannteste Ansatz hierzu sind die *Grundsätze ordnungsgemäßer Modellierung* (GoM), welche im Folgenden vorgestellt werden. Die *Grundsätze ordnungsgemäßer Modellvisualisierung* stellen eine Spezialisierung der GoM dar und werden nachfolgend ebenfalls besprochen.

3.4.1.1 Grundsätze ordnungsgemäßer Modellierung

Die Grundsätze ordnungsgemäßer Modellierung (GoM) sind namentlich und auch von der Intention her eine *„Analogie zu den Grundsätzen ordnungsgemäßer Buchführung“*[23] (Becker, et al., 2012). In den GoM wurde versucht, allgemeine Ziele und Kriterien für grafische Notationssprachen aufzustellen (speziell für den Bereich Informationsmodellierung, wobei die GoM auch auf andere Bereiche anwendbar sind). Die GoM wurden von Becker, Rosemann und Schütte erstmals im Jahre 1995 vorgestellt (Becker, 2012).

Diese Grundsätze sollen Ziele bereitstellen, die Modelle bewertbar machen sowie Modellierungskonventionen, die eine Verbesserung der Modellqualität bewirken sollen. Becker, et al. vertreten ihrer Aussage nach ein *„kundenorientiertes Modellqualitätsverständnis“* (Becker, et al., 2012). Sie definieren:

> *„Die Qualität des Modells ist umso höher zu bewerten, je geringer die Differenz zwischen den Anforderungen des Modelladressaten und der tatsächlichen Eignung des Modells zur Problemlösung ist.“* (ebd.)

Von den Autoren werden die folgenden sechs Grundsätze angeführt:

- *Grundsatz der Richtigkeit,*
- *Grundsatz der Relevanz,*
- *Grundsatz der Wirtschaftlichkeit,*
- *Grundsatz der Klarheit,*
- *Grundsatz der Vergleichbarkeit* und
- *Grundsatz des systematischen Aufbaus.*

Grundsatz der Richtigkeit

Der *Grundsatz der Richtigkeit* fordert *Syntaktische* und *Semantische Richtigkeit* des Modells:

> **GoM-1:** *„Ein Modell ist syntaktisch richtig, wenn es alle Regeln, die die Modellierungssprache vorgibt, einhält.“* (ebd.)

[23] Die Grundsätze ordnungsgemäßer Buchführung (GoB) bestehen aus allgemein anerkannten Regeln zur Führung von Handelsbüchern und zur Erstellung von Jahresabschlüssen. Es gibt jedoch keine eindeutige Festlegung dieser Grundsätze. Als die wichtigsten Grundsätze werden von Auer, et al. (2013) folgende genannt: Systematischer Aufbau der Buchführung, Unveränderlichkeit der Aufzeichnungen, Vollständigkeit und Richtigkeit, Verständlichkeit, Ordnungsmäßigkeit des Belegwesens. (Auer, et al., 2013)

Die *Semantische Richtigkeit* wird wie folgt definiert:

> **GoM-2:** *„Ein Modell ist dann semantisch richtig, wenn im Diskurs der Gutwilligen und Sachkundigen [Modellnutzer] eine Einigung erzielt worden ist."* (ebd.)
>
> Wobei erfüllt sein muss, dass:
>
> 1. Die Diskursteilnehmer jenes, was im Modell beschrieben werden soll, kennen, d.h. **sachkundig** sind.
> 2. Die Diskursteilnehmer zudem an einer Einigung interessiert, d.h. **gutwillig** sind.

Folglich wird ein Modell als semantisch richtig anerkannt, wenn diese *„Gutwilligen und Sachkundigen sich darauf verständigen, eine Repräsentation eines Objektsystems in einer bestimmten Art und Weise in einer Modellierungssprache zu dokumentieren"* (ebd.). Das Kriterium der semantischen Richtigkeit kann und sollte nach Becker, et al. nicht nur auf das Modell angewendet werden, sondern auch auf Teilbereiche des Modells oder einzelne Begriffe zutreffen.[24] Genauso wie für das Modell, muss für Teilmodelle oder einzelne Begriffe ein gemeinsames Verständnis entwickelt werden. Dadurch entsteht eine Sprachgemeinschaft. Dass das Modell etc. semantisch richtig ist, kann nur innerhalb dieser Sprachgemeinschaft sichergestellt werden.

GRUNDSATZ DER RELEVANZ

> **GoM-3:** Der *Grundsatz der Relevanz* fordert, dass *„nur die Sachverhalte modelliert werden [sollen], die für den zugrunde liegenden Modellierungszweck relevant sind."* (ebd.)

Um bewerten zu können, ob das Modellierte für den Modellierungszweck relevant ist, muss das Ziel der Modellierung, der Modellierungszweck, bekannt sein. Er soll expliziert werden. Im Zusammenhang des *Grundsatzes der Relevanz* wird externe und interne Minimalität des Modells gefordert:

> **GoM-4:** *„Es soll […] nichts in der Realwelt geben, was als zweckdienlich definiert worden ist und nicht im Modell vorhanden ist."* (ebd.)
>
> **GoM-5:** *„Auf der anderen Seite soll es aber auch nichts im Modell geben, was nicht sein entsprechendes Pendant in der Realwelt hat."* (ebd.)

GRUNDSATZ DER WIRTSCHAFTLICHKEIT

> **GoM-6:** Der *Grundsatz der Wirtschaftlichkeit* fordert, *„dass ein gegebenes Modellierungsziel mit minimalem Aufwand erreicht werden soll oder dass mit einem gegebenen Modellierungsaufwand ein Modell erreicht werden soll, das dem Modellierungszweck am nächsten kommt."* (ebd.)

Ersteres entspricht dem Minimal- und letzteres dem Maximalprinzip des Ökonomischen Prinzips.[25] Bei der Anwendung dieses Grundsatzes auf die Modelldetaillie-

---

[24] Dieses Verständnis der *syntaktischen Richtigkeit* ähnelt dem Syntax-Verständnis des *3QM-Frameworks*. Bei Overhage, et al. (2012) wird die Syntax in Text-, Satz- und Wortsyntax untergliedert.

[25] Das Ökonomische Prinzip ist eine *„rationale Verhaltensregel für wirtschaftliches Handeln"* (May, et al., 2001): Nach dem Minimalprinzip soll *„ein gegebenes Ziel mit dem geringstmöglichen Mitteleinsatz"* (ebd.) erreicht werden. Das Maximalprinzip besagt, dass *„mit gegebenen Mitteln ein maximaler Zielerreichungsgrad angestrebt"* (ebd.) werden soll.

rung folgt bspw.: *„Ein Modell soll so lange verfeinert werden, bis die zusätzlichen Kosten der Verfeinerung gerade dem zusätzlichen Nutzen, der aus der Verfeinerung resultiert, gleich kommen.“* (ebd.).

GRUNDSATZ DER KLARHEIT

GoM-7: Der *Grundsatz der Klarheit* fordert *„die Verständlichkeit der Modelle […]. Hier stehen leichte Lesbarkeit, Anschaulichkeit und Verständlichkeit im Vordergrund. Insbesondere die adressatengerechte Hierarchisierung, Layoutgestaltung und Filterung sind Teilaspekte der Klarheit.“* (ebd.)

GRUNDSATZ DER VERGLEICHBARKEIT

GoM-8: Der *Grundsatz der Vergleichbarkeit* fordert:

1. *„Abläufe der Realwelt und der Vorstellungswelt, die gleich sind und innerhalb einer Modellierungssprache dokumentiert sind, sollten auch im Modell voll identisch sein, damit die tatsächliche Gleichheit (in der Realwelt) sich auch in einer Gleichheit im Modell widerspiegelt und damit die Modelle vergleichbar sind.“* (ebd.)
2. Um *„Modelle […], die in unterschiedlichen Modellierungssprachen dokumentiert sind [,] […] vergleichen zu können, müssen sie ineinander überführbar sein.“*[26] (ebd.)

GRUNDSATZ DES SYSTEMATISCHEN AUFBAUS

GoM-9: Der *Grundsatz des systematischen Aufbaus* fordert, dass wenn *„Sachverhalte der Realwelt oder der Darstellungswelt aus unterschiedlichen Sichten beschrieben [werden] […]. Wenn z. B. in einem Prozessmodell Organisationseinheiten notiert sind, dann sollten diese Organisationseinheiten denjenigen entsprechen, die im Organisationsmodell hinterlegt sind.“* (ebd.)

3.4.1.2 GRUNDSÄTZE ORDNUNGSGEMÄßER MODELLVISUALISIERUNG

Die Grundsätze ordnungsgemäßer Modellvisualisierung (GoMV) sind eine von Deelmann, et al. (2004) entwickelte Anwendung der GoM auf Modellvisualisierungen. Hier werden aus den GoM allgemeine, modellierungssprachenunabhängige Leitlinien zur Gestaltung grafischer Informationsobjekte abgeleitet (siehe Tab. 3-5). Die in den GoMV formulierten Regeln stützen *„sich auf verhaltenswissenschaftliche Hinweise für die Informationsvisualisierung sowie auf Grundlagen für die visuelle Gestaltung kommunikativer Situationen“* (Deelmann, et al., 2004).

GoMV-1: Der **GRUNDSATZ DER AUTHENTISCHEN DARSTELLUNG** fordert eine *„sachlogisch richtige[n] Gestaltung von Informationsobjekten.“* (ebd.)

GoMV-2: Der **GRUNDSATZ DES INHALTLICHEN MINIMALPRINZIPS** fordert *„bei der Erstellung von Informationsobjekten die Anzahl unterschiedlicher Objekte (i.S.v. Typen) [sic!] zu minimieren. Weiterhin sind diese lediglich mit Hilfe inhaltlich relevanter Elemente zu gestalten.“* (ebd.)

---

[26] *„Dies ist insbesondere dann möglich, wenn sie auf ähnlichen Konstrukten aufbauen und über eine Metamodelltransformation ineinander überführt werden können.“* (Becker, et al., 2012)

Original Grundsatz der GoM	Abgeleiteter Grundsatz in den GoMV
Grundsatz der Richtigkeit (semantisch)	*Grundsatz der authentischen Darstellung*
Grundsatz der Relevanz	*Grundsatz des inhaltlichen Minimalprinzips*
Grundsatz der Wirtschaftlichkeit	*Grundsatz des minimalen Visualisierungsgrades*
Grundsatz der Klarheit	*Grundsatz der Metaphernnutzung*
Grundsatz der Vergleichbarkeit	*Grundsatz der Wiederverwendung*
Grundsatz des systematischen Aufbaus	*Grundsatz der Konsistenz*

Tab. 3-5: Laut Deelmann, et al. (2004) aus den GoM abgeleitete GoMV.

GoMV-3: Der **GRUNDSATZ DES MINIMALEN VISUALISIERUNGSGRADES** fordert *„die Nutzung des kleinstmöglichen Visualisierungsgrades, also bspw. den Verzicht auf Farben oder räumliche Dimensionen, soweit diese keine zusätzlichen Informationsgehalt aufweisen."* (ebd.)

GoMV-4: Der **GRUNDSATZ DER METAPHERNNUTZUNG** besagt: *„Die Berücksichtigung von Metaphern und etablierte Symbolik bei der Gestaltung von Informationsobjekten erhöht die Nutzerfreundlichkeit von Modellen. Auch Aspekte wie Anschaulichkeit und Gefälligkeit von Modellelementen können zur Nutzungsfreundlichkeit der endgültigen Gesamtmodelle breitragen."* (ebd.)

GoMV-5: Der **GRUNDSATZ DER WIEDERVERWENDUNG** fordert die *„Wiederverwendung von Informationsobjekten für vergleichbare Tatbestände [und] […] modellübergreifende Konsistenz bei Erstellung und Einsatz von Informationsobjekten."* (ebd.)

GoMV-6: Der **GRUNDSATZ DER KONSISTENZ** fordert die *„Einhaltung des Konsistenzprinzips in der Informationsgestaltung. Formen, Linienart und -stärke, Schriftarten etc. sind für alle Objekte konsistent zu bestimmen."* (ebd.)

3.4.2 QUALITÄT EINZELNER SRS-MODELLBESTANDTEILE

Nachdem im vorherigen Abschnitt Qualitätsanforderung an das SRS-Modell und die SRS-Modellvisualisierungen im Allgemeinen aufgezeigt wurden, werden hier nun Qualitätskriterien für einzelne Teilmodelle und Modellelemente des *SRS-Modells* vorgestellt. Im Rahmen dieser Arbeit wird auf BPMN-Pools und -Startereignisse und Anforderungen eingegangen.

3.4.2.1 BPMN-PROZESS- UND -KOLLABORATIONS-DIAGRAMME

Im Bereich der BPMN wurden bereits einige Texte veröffentlicht, die sich mit dem Thema Qualitätssicherung von BPMN-Modellen oder allgemein Prozessmodellen auseinandersetzen (z.B. eCH, 2013; Overhage, et al., 2012). Als aktuelles Beispiel wird nachfolgend das 3QM-Framework vorgestellt, welches ein Qualitätssystem und Messverfahren für BPMN-Geschäftsprozesse vorschlägt. Die Schweizer Richtlinie *„eCH-0158 BPMN-Modellierungskonventionen für die öffentliche Verwaltung"* (eCH, 2013) wird als zweites Beispiel angeführt. eCH-0158 definiert Modellierungs- und Modellvisualisierungskonventionen für BPMN-Modelle und -Diagramme.

DAS 3QM-FRAMEWORK

Overhage, et al. (2012) entwerfen ein hierarchisches Qualitätssystem für die Bewertung der Güte von Geschäftsprozessmodellen als Kommunikationsmittel, das 3QM-Framework. Es besteht aus Qualitätsmerkmalen, -metriken und zugehörigen Messverfahren. Ziel der Autoren war es u.a. ein Qualitätssystem zu definieren, welches modellierungssprachenunabhängig ist. Handschriftlich verfasste Modelle der Sprachen UML, BPMN und EPK[27] sowie *„normsprachliche[n] Prozessbeschreibungen"* (Overhage, et al., 2012) wurden bereits mit Hilfe des Qualitätssystem von den Autoren im Rahmen von Fallstudien bewertet. Darüber hinaus wurde im Rahmen einer Expertenbefragung Bestätigung und Kritik bezüglich der Struktur des 3QM-Frameworks eingeholt.

Bei der Definition des Qualitätssystems orientieren Overhage, et al. sich an der Semiotik, *„d.h. der Theorie vom Wesen und Gebrauch sprachlicher Zeichen"* (ebd.). Als zentrale Qualitätsmerkmale definieren Overhage, et al. (2012) daher:

- *Syntaktik* (Untermerkmale: Wort-, Satz, und Textsyntax),
- *Semantik* (Untermerkmale: Vollständigkeit, Korrektheit, Flexibilität und Relevanz) sowie
- *Pragmatik* (Untermerkmale: Eindeutigkeit und Verständlichkeit).

QUALITÄTSMERKMALE

Die **SYNTAKTIK** soll bewerten, *„inwiefern die formalen Regeln der Modellierungssprachen in einem Geschäftsprozessmodell eingehalten werden."* (ebd.) Unter dem Teilaspekt *Wortsyntax* untersuchen sie, ob der im Geschäftsprozessmodell verwendete Zeichenvorrat für die Modellierungssprache gültig ist (d.h. in der BPMN: Werden ausschließlich Modellbestandteile der BPMN verwendet?), unter *Satzsyntax*, ob die Modellelemente bzw. Wörter der Modellierungssprache konform zur Syntax zu größeren Spracheinheiten verknüpft werden (d.h. in der BPMN dürfen z.B. Zwischenereignisse nur innerhalb eines Prozesses modelliert werden.) und unter *Textsyntax* die konforme Verwendung von Konnektoren zur Verknüpfung von „Sätzen" zu „Texten" (d.h. in der BPMN Syntaxregeln zu der Verwendung von Gateways).

Die **SEMANTIK** soll Aussagen darüber liefern, *„inwiefern der inhaltliche Bezug der Modellelemente angemessen ist, d.h. der zugrundeliegende Wirklichkeitsausschnitt im Modell in adäquater Weise repräsentiert wird"* (ebd.). Das untergeordnete Qualitätsmerkmal *Vollständigkeit* bewertet, inwiefern jene für das Verständnis des Modells erforderlichen Informationen im Modell dargestellt werden. Unter *Korrektheit* wird überprüft, inwiefern die Inhalte des Modells mit dem Sachverhalt im Original inhaltlich übereinstimmen. Ob im Original unabhängige Teilprozesse im Modell ebenfalls als parallel dargestellt und sequentialisiert werden, wird unter *Flexibilität* betrachtet. Unter Relevanz wird untersucht, ob jedes im Modell enthaltene Modellelement für die Kommunikation notwendig ist bzw. ob es überflüssige Elemente gibt.

[27] EPK ist die Abkürzung für Ereignisgesteuerte Prozessketten.

Die *„***PRAGMATIK** *gibt an, inwieweit die Darstellung des Geschäftsprozessmodells die Interpretation des zu kommunizierenden Inhalts unterstützt“* (ebd.). Unter dem Teilaspekt *Eindeutigkeit* wird betrachtet, ob der im Modell dargestellte Inhalt gut oder leicht interpretierbar ist. Ob der verwendete Sprachgebrauch eine Interpretation leicht macht, wird unter dem Aspekt *Verständlichkeit* untersucht, bspw. ob Wörter konsistent verwendet werden.

METRIKEN

Zusätzlich zum Qualitätssystem schlagen Overhage, et al. Metriken vor. Sie nutzen diese zur Kategorisierung von manuell identifizierten Fehlern in handschriftlich verfassten Modellen. Im zweiten Schritt leiten sie aus den berechneten Maßzahlen eine Bewertung der Modellqualität des Geschäftsprozessmodells ab. Das 3QM-Framework schlägt 35 Metriken zur Bestimmung der Qualitätsmerkmale vor. Für jede Metrik gibt es eine absolute (A) und eine relative Berechnungsvorschrift (R), z.B. für die Metrik „Original Verstoß“ des Qualitätsmerkmals *Wortsyntax*:

a) *„A = Anzahl erstmals nicht korrekt modellierter Sprachkonstrukte“* (ebd.)
b) *„R = 1 – A ÷ (Anzahl insgesamt modellierter Sprachkonstrukte)“* (ebd)

Für die Untermerkmale der Syntaktik werden je zwei Metriken vorgeschlagen: Zum einen die Anzahl der „Original“ Verstöße bezüglich des Merkmals und zum anderen die Anzahl der „Wiederholten“ Verstöße. Im Rahmen der *Semantik* werden für *Korrektheit* falsche, bei der *Relevanz* überflüssige und bei der *Vollständigkeit* fehlende Modellelemente gezählt mit je acht unterschiedlichen Metriken für die acht unterschiedlichen Arten von Modellelementen z.B. Aktionen, Ereignisse und Kontrollflüsse. Die *Flexibilität* unter *Semantik* hat abweichend dazu nur eine einzige Metrik „Einschränkung“: *„Werden die im Wirklichkeitsausschnitt unabhängigen Abläufe als parallele Abläufe modelliert?“* (Overhage, et al., 2012)). Dem Qualitätsmerkmal *Eindeutigkeit* und *Verständlichkeit* der *Pragmatik* wurden jeweils zwei Metriken zugeordnet:

- „Redundanz“ (Eindeutigkeit): *„A = Anzahl an Prozesselementen, die unnötigerweise wiederholt dargestellt werden“* (ebd.)
- „Widerspruch“ (Eindeutigkeit): *„A = Anzahl der Prozesselemente, die sich logisch widersprechen“* (ebd.)
- „Abweichende Bezeichnung“ (Verständlichkeit): *„A = Anzahl an Bezeichnungen, die im Modell variieren“* (ebd.)
- „Nicht normierte Bezeichnung“ (Verständlichkeit): *„A = Anzahl an Bezeichnungen, die gegen gängige Konventionen verstoßen“* (ebd.)

Das Vorgehen, wie Overhage, et al. aus den Maßzahlen der Metriken eine Gesamtbewertung der Modellqualität von Geschäftsprozessmodellen entwickelt haben, wird in Abschnitt 3.3.2 ausgeführt.

ECH-0158

Die *„BPMN-Modellierungskonventionen für die öffentliche Verwaltung“* (eCH, 2013) sind ein vom Verein eCH herausgegebener Standard. eCH ist ein Schweizer Verein zur Förderung von E-Government[28]-Standards. Der Standard eCH-0158

[28] < Def.: E-Government. >

wurde für *deskriptive* BPMN-Diagramme, die den normalen Prozessablauf auf mehreren Ebenen aus der Geschäftssicht beschreiben, entworfen. Laut eCH (2013) sollen mittels dieses Standards die Freiheitsgrade der Modellierungssprache eingeschränkt, vereinheitlicht und vereinfacht werden. Dies soll die Komplexität der Diagramme verringern, die Übersichtlichkeit und Verständlichkeit erhöhen sowie indirekte Kosten[29] verhindern.

Von eCH-0158 werden der Wortschatz und die Syntax der BPMN eingeschränkt sowie Konventionen für die Benennung und logische Verwendung von BPMN-Sprachelementen eingeführt (d.h. Modellierungskonventionen). Bspw. sind die BPMN-Sprachelemente Datenobjekte und auslösende Zwischenereignisse nach diesem Standard nicht zulässig. Weiterhin gibt es Konventionen, die Aussagen zu der Darstellung der BPMN-Diagramme und -elemente machen (d.h. Modellvisualisierungskonventionen) und es werden vom Standard zwei Prozessmuster vorgeschlagen. Die Prozessmuster sind als Vorlage oder Referenzdiagramme für Sachverhalte gedacht, die oft modelliert werden. Insgesamt enthält eCH-0158 65 teilweise nicht atomare Konventionen, von denen 18 festlegen, wie bestimmte BPMN-Modellbestandteile beschriftet werden sollten. (ebd.)

Beispielhaft schauen wir uns in der vorliegenden Arbeit die Konventionen für das Teilmodell Pool (in eCH-0158 „Pools“) und das Modellelement Startereignis an. In Tab. 3-6 (S. 43) sind die zugehörigen Konventionen aus der *eCH-0158* aufgeführt.

3.4.2.2 ANFORDERUNG

In unterschiedlichen Quellen werden Qualitätskriterien für einzelne Anforderungen vorgeschlagen (z.B. IEEE, 1998; Lamsweerde, 2009; Pohl, 2010; Rupp, et al., 2009). Die Mengen der in den Quellen jeweils aufgeführten Qualitätsattribute ähneln sich stark von der Anzahl und der Bezeichnung (siehe Tab. 3-7, S. 44). Welche Bedeutung jedoch unter dem jeweiligen Kriterium bzw. Bezeichnung verstanden wird, ist teilweise stark abweichend.

[29] Dem Standard zufolge können indirekte Kosten entstehen, zum einen durch unterschiedliche Interpretation der dokumentierten Prozesse oder zum anderen durch Neu-Dokumentation von Prozessen, um sie den organisationsspezifischen Modellierungsregeln anzupassen.

Kürzel[30]	Definition	Kategorie[31]
4.3	***Pool***	
eCH0158-Pool-01	*„Pools werden in der Regel mit Organisationseinheiten oder dem Namen anderer Prozessbeteiligter bezeichnet.“*	MVK
eCH0158-Pool-02	*„Vorlagen mit den am häufigsten verwendeten Pools erleichtern die Arbeit und erhöhen die Lesbarkeit.“*	MK
eCH0158-Pool-03	*„Reihenfolge oder Farbgebung kann der Identifizierung interner / externer Pools dienen.“*	MVK
eCH0158-Pool-04	*„In der Regel wird nur der eigene Pool aufgeklappt dargestellt.“*	MVK
eCH0158-Pool-05	*„In jedem aufgeklappten Pool wird genau ein vollständiger Prozess modelliert.“*	MK
eCH0158-Pool-06	*„Pools werden übereinander über die gesamte Diagrammbreite dargestellt.“*	MVK
eCH0158-Pool-07	*„Die Höhe des aufgeklappten Pools richtet sich nach dessen Inhalt.“*	MVK
eCH0158-Pool-08	*„Zugeklappte Pools enthalten mindestens einen eingehenden oder ausgehenden Nachrichtenfluss.“*	MK
4.5.1	***Startereignis***	
eCH0158-Startereignis-01	Unbestimmtes Startereignis: *„Wird nicht beschrieben, der Unterprozess wird durch den Aufruf auf der übergeordneten Prozessebene gestartet.“*	MK
eCH0158-Startereignis-02	Nachrichten-Startereignis: *„Wird nicht beschrieben, wenn die eingehende Nachricht auf dem (obligatorischen) Nachrichtenfluss ersichtlich ist.“*	MK
eCH0158-Startereignis-03	Bedingungs-Startereignis: *„Bedingung ist in der Bezeichnung des Elementes festzuhalten. Sollte ein Bedingungs-Startereignis einem Endereignis eines anderen Prozesses entsprechen, ist es identisch zu bezeichnen.“*	MK
eCH0158-Startereignis-04	Zeitgeber-Ereignis: *„Bezeichnung enthält den Zeitpunkt für den Start. Beispiele: „am 1. Je Monat“, „09:00 Uhr“.“*	MK
eCH0158-Startereignis-05	*„Unbestimmte Startereignisse werden nur verwendet, wenn der Prozess durch den Aufruf auf der übergeordneten Prozessebene gestartet wird.“*	MK
eCH0158-Startereignis-06	*„Ein Prozess hat mindestens ein Startereignis.“*	MK
eCH0158-Startereignis-07	*„Startereignisse werden innerhalb eines Pools dargestellt.“*	MK

Tab. 3-6: eCH-0158 zu Prozesse und Startereignisse.

[30] Das Kürzel für die Konventionen wurde für diese Arbeit entworfen.

[31] MK: Modellierungskonvention; MVK: Modellvisualisierungskonvention.

Quellen: / *Qualitätskriterien:*	**Balzert, 2009**	**IEEE, 1998**	**Lamsweerde, 2009**	**Pohl, 2010**	**Rupp, et al., 2009**
Abgestimmt				(X)[32]	X
Aktuell (Gültig und aktuell)				X	X
Atomar				X	
Bewertet	X	X		X	X
Eindeutig	X	X	X	X	X
Klassifizierbar					X
Konsistent	X			X	X
Korrekt	X	X	X	X	X
Nachvollziehbar	X		X	X	X
Notwendig				(X)[32]	X
Überprüfbar	X	X	X	X	X
Realisierbar			X		X
Sachdienlich			X		
Verständlich			X	X	X
Vollständig	X		X	X	X

Tab. 3-7: Qualitätskriterien für einzelne Anforderungen.

Pohl (2010) definiert in seinem Grundlagenbuch zu Requirements Engineering zehn Qualitätsanforderungen für dokumentierte Anforderungen: *Vollständig*, *nachvollziehbar*, *korrekt*, *eindeutig*, *verständlich*, *konsistent*, *prüfbar*, *bewertet*, *aktuell* und *atomar*. In dieser Arbeit werden seine Definitionen betrachtet, da er die Qualitätskriterien sehr klar und eindeutig formuliert und diese zusätzlich anhand von Beispielen erläutert: Er schreibt, welcher Teil bzw. Aspekt der Anforderung wie beschaffen sein muss, um dem Kriterium zu genügen[33]. Zusätzlich weist Pohl auf Schwierigkeiten hin. Bspw., dass unter dem Begriff *„korrekt“* in der Informatik das Prüfen gegen die Spezifikation verstanden wird und daher der Begriff in diesem Bereich irreführend sein kann. (Aus diesem Grund wird in dieser Arbeit nachfolgend das Kriterium „Richtigkeit“ bzw. „richtig“ nach Becker, et al. (2012) verwendet.)

[32] Pohl (2010) fasst die Attribute *notwendig* und *aktuell* unter dem Adjektiv *korrekt* zusammen.

[33] Zwei Definitionen des Qualitätskriteriums „vollständig“ im Vergleich (zum Thema Ungenauigkeit von Begriffsdefinitionen):

a) Rupp, et al. (2009): *„Jede einzelne Anforderung muss die geforderte und zu liefernde Funktionalität vollständig beschreiben.“*
b) Pohl (2010): *„Eine Anforderung ist vollständig, wenn sie die für die Anforderungsart spezifischen Regeln und Richtlinien erfüllt und keine Informationen auslässt, die für irgendeinen Stakeholder wichtig ist.“*

Pohl (2010) definiert seine Qualitätskriterien für Anforderungsartefakte[34] (engl. requirement artefact) wie folgt:

VOLLSTÄNDIG (REQ-1): *„A requirement artefact is complete if it adheres to the rules and guidelines defined for this type of requirements artefact and does not omit any piece of information that is relevant for some stakeholder (user, client, architect, tester, etc.)." (Pohl, 2010)*

NACHVOLLZIEHBAR (REQ-2): *„A requirement artefact is traceable if its source, its evolution as well as its impact and use in later development phases are traceable."* (ebd.)

KORREKT (REQ-3): *„A requirement artefact is correct, if the relevant stakeholders confirm its correctness and demand that the system must 45ealize the documented requirement completely."* (ebd.)

EINDEUTIG (REQ-4): *„A requirement artefact is unambiguous, if its documentation permits only one valid interpretation."* (ebd.)

VERSTÄNDLICH (REQ-5): *„A requirement is comprehensible if its content is easy to comprehend. The comprehensibility of a requirement artefact depends, among other things, on the documentation format chosen and the stakeholder(s) involved."* (ebd.)

KONSISTENT (REQ-6): *„A requirement artefact is consistent, if the statements within the artefact do not contradict each other."* (ebd.)

ÜBERPRÜFBAR (REQ-7): *„A requirement is verifiable, if the stakeholder can check whether the implemented system fulfils the documented requirement or not."* (ebd.)

BEWERTET (REQ-8): *"A requirement artefact is rated, if its relevance and/or its stability have been determined and documented."* (ebd.)

AKTUELL (REQ-9): *"A requirement artefact is up to date, if it reflects the current status of the system and system context, such as the current stakeholder wishes or current legal regulations."* (ebd.)

ATOMAR (REQ-10): *"A requirement artefact is atomic, if it describes a single, coherent fact."* (ebd.)

3.4.3 QUALITÄT DER SRS-DOKUMENTATION

Die vorliegende Arbeit fokussiert die Qualitätsbetrachtung von *SRS-Modellen* und *-Modellvisualisierungen*. Trotz dessen stellen wir in diesem Abschnitt aus den folgenden Gründen Qualitätskriterien für SRS-Dokumente vor:

- Zum einen, weil diese Kriterien oftmals die Gesamtqualität durch eine Qualitätsanforderung an alle Bestandteile des Dokuments definieren und somit Anforderungen an die Modellbestanteile darin zu finden sind,
- Zum anderen, da sich aus diesen Qualitätsanforderungen an das SRS-Dokument, durch unseren modellbasierten Ansatz der Dokumentationserstellung, Qualitätsanforderungen an das SRS-Modell, die Modellbestandteile und Diagramme ableiten lassen.

[34] Mit dem Begriff Anforderungsartefakt will Pohl (2010) explizit darauf hinweisen, dass es sich hierbei um dokumentierte Anforderungen handelt.

Für SRS-Dokumente gibt es eine Reihe von Qualitätskriterien (siehe Tab. 3-8) unterschiedlicher Autoren (z.B. Balzert 2009; IEEE, 1998; Lamsweerde, 2009; Pohl, 2010; Rupp, et al., 2009). Für diese Arbeit haben uns für die genauere Betrachtung der Definition des IEEE Std 830-1998 entschieden, da es sich hierbei um einen internationalen Standard handelt.

Quellen: / *Qualitätskriterien:*	**Balzert, 2009**	**IEEE, 1998**	**Lamsweerde, 2009**	**Pohl, 2010**	**Rupp, et al., 2009**
Abhängigkeiten angebbar	X				
Angemessener Umfang	X				X
Bewertet		X			
Eindeutig		X	X	X	X
Erweiterbar	X		X	X	X
Gemeinsam zugreifbar					X
Klare Struktur			X	X	X
Konsistent	X	X	X	X	X
Korrekt	X	X	X		
Modifizierbar	X	X	X		X
Nach verschiedenen Sichten auswertbar	X				
Nachvollziehbar		X	X	X	X
Optimiert bezüglich des gewählten Vorgehens					X
Qualitativ hochwertig					X
Realisierbar			X		
Sachdienlich			X		
Sortierbar					X
Überprüfbar		X	X		
Verständlich			X		
Vollständig	X	X	X	X	X

Tab. 3-8: Qualitätskriterien für einzelne SRS-Dokumente.

IEEE Recommended Practice for SRS

Der IEEE Std 830-1998 empfiehlt ein Vorgehen für das Erstellen von Anforderungsspezifikationen (SRS). Es werden die Inhalte einer SRS beschrieben, sowie Qualitätskriterien für SRS definiert. Teilweise werden zu den Qualitätskriterien Beispiele genannt oder Umsetzungsmöglichkeiten der Sicherstellung des jeweiligen Kriteriums angerissen. Im Folgenden werden die Definitionen dieser Kriterien aufgeführt.

KORREKT (SRS-1): *„An SRS is correct if, and only if, every requirement stated therin is one that the software shall meet." (IEEE, 1998)*

EINDEUTIG (SRS-2): *„An SRS is unambiguous if, and only if, every requirement stated therein has only one interpretation."* (ebd.)

VOLLSTÄNDIG (SRS-3): *„An SRS is complete if, and only if, it includes the following elements:*

a) All significant requirements, whether relating to functionality, performance, design constraints, attributes, or external interfaces. In particular any external requirements imposed by a system specification should be acknowledged and treated.

b) Definition of the responses of the software to all realizable classes of input data in all realizable classes of situations. Note that it is important to specify the responses to both valid and invalid input values.

c) *Full labels and references to all figures, tables, and diagrams in the SRS and definition of all terms and units of measure."* (ebd.)

KONSISTENT (SRS-4): *„Consistency refers to internal consistency. If an SRS does not agree with some higher-level document, such as a system requirements specification, then it is not correct (…)."*(ebd.)

BEWERTET (SRS-5): *„An SRS is ranked for importance and/or stability if each requirement in it has an identifier to indicate either the importance or stability of that particular requirement."* (ebd.)

ÜBERPRÜFBAR (SRS-6): *„An SRS is verifiable if, and only if, every requirement stated therein is verifiable. A requirement is verifiable if, and only if, there exists some finite cost-effective process with which a person or machine can check that the software product meets the requirement. In general any ambiguous requirement is not verifiable."* (ebd.)

MODIFIZIERBAR (SRS-7): *„An SRS is modifiable if, and only if, its structure and style are such that any changes to the requirements can be made easily, completely, and consistently while retaining the structure and style. Modifiability generally requires an SRS to*

a) Have a coherent and easy-to-use organization with a table of contents, an index, and explicit cross-referencing;

b) Not be redundant (i.e., the same requirement should not appear in more than one place in the SRS);

c) *Express each requirement separately, rather than intermixed with other requirements."* (ebd.)

NACHVOLLZIEHBAR (SRS-8): *„An SRS is traceable if the origin of each of its requirements is clear and if it facilitates the referencing of each requirement in future development or enhancement documentation. The following two types of traceability are recommended:*

a) Backward traceability (i.e., to previous stages of development). This depends upon each requirement explicitly referencing its source in earlier documents.

b) *Forward traceability (i.e., to all documents spawned by the SRS). This depends upon each requirement in the SRS having a unique name or reference number."* (ebd.)

3.4.4 Zusammenfassung

Für jeden im Rahmen dieser Arbeit betrachteten Aspekt wurden in diesem Abschnitt Qualitätseigenschaften und Definitionen aufgeführt: Die GoM machen allgemeine Aussagen und Qualitätsanforderungen zu Modellen (siehe Abschnitt 3.4.1.1), die GoMV zu Modellvisualisierungen (siehe 3.4.1.2). In Abschnitt 3.4.2 wurden Qualitätsanforderungen zu einzelnen Modellelementen vorgestellt:

- 3QM-Framework und eCH-0158 für BPMN-Prozesse und -Kollaborationen (siehe 3.4.2.1)
- Qualitätsanforderungen nach Pohl (2010) für Anforderungen (siehe 3.4.2.2)

Schließlich wurden bezüglich SRS die Qualitätsanforderungen nach IEEE Std 830-1998 vorgestellt (siehe 3.4.3).

Diese einzelnen Qualitätsdefinitionen werden im nachfolgenden Abschnitt 4.1 zu einem Qualitätssystem integriert, welches die Qualitätsfaktoren von *SRS-Modellen* und *-Modellvisualisierungen* definiert.

4 Entwurf der Qualitätssicherung von modellbasierten SRS

In diesem Abschnitt der Abschlussarbeit wird ein Entwurf für die Qualitätssicherung von *SRS-Modellen* und *-Modellvisualisierungen* skizziert. Ziel ist es, einen ersten Konzeptentwurf für eine Applikation zur automatischen Qualitätsprüfung, -messung und -bewertung für das Modellierungswerkzeug Innovator zu entwickeln. Hierzu wird ein Qualitätssystem für die Qualität *modellbasierter SRS* definiert und Vorschläge zur Qualitätsprüfung, -messung und -bewertung gegeben (Abschnitt 4.1). Anschließend wird der fachliche Konzeptentwurf für die Applikation vorgestellt (4.2).

Das im Folgenden vorgestellte Qualitätssystem ist ein Qualitätsmodell. Es definiert eine Ordnung aus Qualitätseigenschaften im Sinne einer Taxonomie: Die darin enthaltenen Qualitätseigenschaften sind durch Abhängigkeitsbeziehungen baumartig angeordnet, wobei die näher an der Wurzel liegenden Eigenschaften weniger spezifisch sind als die weiter entfernten. Mit Hilfe dieses Systems sollen die Qualitätsanforderungen für modellbasierte SRS differenziert und strukturiert werden. Im Zuge dessen entsteht eine Definition des Qualitätsbegriffs für modellbasierte SRS und im Speziellen für SRS-Modelle und -Modellvisualisierungen.

4.1 Entwurf des Qualitätssystems und Ansätze der Qualitätssicherung

Bisher gibt es in der Literatur noch keine uns bekannte Definition eines Qualitätssystems für die Qualitätsbewertung von *modellbasierten SRS* oder von *SRS-Modellen* zur Generierung modellbasierter SRS-Dokumente. In der Forschung wurden bisher allgemeine (z.B. GoM) oder ganz spezielle Ansätze für bspw. einzelne Modellierungssprachen (z.B. 3QM-Framework) entwickelt. In dieser Arbeit werden die besprochenen Ansätze aus Abschnitt 3.4 zu einem Qualitätssystem integriert. Das im Zuge dieser Bachelorarbeit entwickelte Qualitätssystem (siehe Abb. 4-1, S. 50) wird in diesem Abschnitt vorgestellt. Es beschreibt, von welchen Aspekten die Qualität einer modellbasierten Anforderungsspezifikation abhängt.

4.1.1 Herleitung des Qualitätssystems

Für die Definition des Qualitätssystems wurde eine Kombination aus GQM und FCM verwendet (siehe Abschnitt 3.3.1), um die Komplexität des Qualitätssystems besser handhabbar zu machen: Das System wird durch die Qualitätsfaktoren strukturiert und die Qualitätskriterien werden durch die Faktoren thematisch zusammengefasst. Dies ermöglicht klar abgegrenzte kleinere Teilthemen des Qualitätssystems zu betrachten, welche von der Anzahl der Bestandteile für den Betrachter überschaubar und begreifbar sind, welches wir unter einer besseren „Handhabbarkeit“ verstehen.

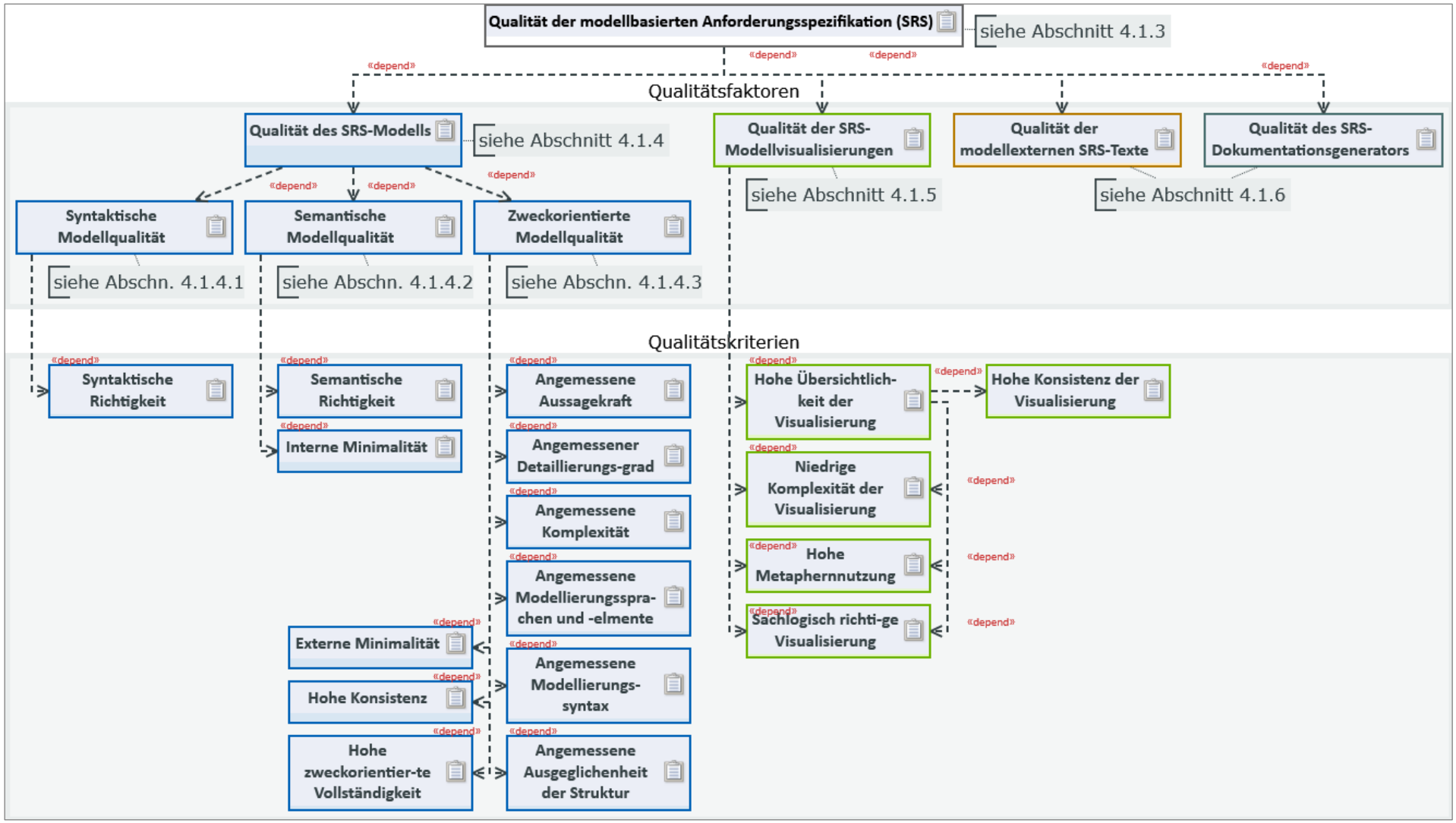


Abb. 4-1: Qualitätssystem für „Qualität modellbasierter Anforderungsspezifikationen“.

Nach dem FCM-Ansatz wird im ersten Schritt die Softwarequalität (d.h. hier: die *Qualität der modellbasierten SRS*) durch eine Menge von Qualitätsfaktoren definiert (z.B. *Qualität des SRS-Modells*, siehe Abb. 4-1, S. 50). In einem zweiten Schritt werden diese Qualitätsfaktoren teilweise iterativ erneut durch weitere Qualitätsfaktoren definiert (z.B. *Qualität des SRS-Modells* wird verfeinert durch *Syntaktische, Semantische* und *Zweckorientierte Modellqualität*). Die so entwickelten Qualitätsfaktoren (der jeweils untersten Ebene) werden für den GQM-Ansatz als die *„Goals“* eingesetzt. Dies sind die folgenden sechs Faktoren:

- *Syntaktische, Semantische* und *Zweckorientierte Modellqualität,*
- *Qualität der Modellvisualisierungen,*
- *Qualität der modellexternen SRS-Texte,*
- *Qualität des Dokumentationsgenerators*.

Die *„Questions“* des GQM-Ansatzes bilden im Qualitätssystem die Qualitätskriterien (z.B. *Syntaktische Richtigkeit*). Sie tragen zusätzlich zur Formulierung als Frage (z.B. *„Ist das Modell syntaktisch richtig?“*) einen Namen (z.B. *„Syntaktische Richtigkeit“*). Zu guter Letzt werden wie im GQM-Ansatz den Qualitätskriterien Metriken (*„Metrics“*) zugeordnet (z.B. *„#Verstöße gegen die Syntax“* für das Qualitätskriterium *Syntaktische Richtigkeit*). Diese Metriken stellen im Rahmen dieser Bachelorarbeit erste Vorschläge zur Messung des jeweiligen Kriteriums dar. (Das Qualitätssystem ist erweiterbar.)

Das von uns entwickelte Qualitätssystem besteht auf der obersten Ebene aus vier Qualitätsfaktoren:

- *Qualität des SRS-Modells*
- *Qualität der SRS-Modellvisualisierungen*
- *Qualität der modellexternen SRS-Texte*
- *Qualität des SRS-Dokumentationsgenerators*

Diese Aufteilung ist motiviert durch die einzelnen Komponenten aus denen die *modellbasierte SRS* besteht (das *SRS-Modell*, die *SRS-Modellvisualisierungen* und die *modellexternen SRS-Texte*) und die sie ausmachen (die Dokumentationsgenerierung bzw. der *Dokumentationsgenerator*; siehe Abschnitt 0). Da die Aufgabenstellung dieser Arbeit ist, die Modellqualität zu prüfen und zu bewerten, werden die Faktoren *Qualität des SRS-Modells* und *Qualität der Modellvisualisierungen* ausführlich besprochen. Die *Qualität der modellexternen Texte* als auch *des SRS-Dokumentationsgenerators* werden der Vollständigkeit halber benannt und definiert, jedoch im Rahmen dieser Arbeit nicht weiter betrachtet.

Für den Faktor *Qualität des SRS-Modells* haben wir zwölf Qualitätskriterien erarbeitet, für die *Qualität der SRS-Modellvisualisierungen* fünf (siehe Tab. 4-1), indem wir Definitionen aus Abschnitt 3.4 teilweise direkt übernommen, mehrere Definitionen unter einem Kriterium gebündelt oder Definitionen in mehrere Kriterien aufgegliedert haben (siehe Abschnitt 4.1.2).

Qualitätskriterium	Automatische konstruktive Sicherung	Automatische Prüfung	Manuelle Prüfung
***Qualität des SRS-Modells** • Syntaktische MQ*			
Syntaktische Richtigkeit	X[35]	X	
***Qualität des SRS-Modells** • Semantische MQ*			
Interne Minimalität			X
Semantische Richtigkeit		(X)[36]	X
***Qualität des SRS-Modells** • Zweckorient. MQ*			
Angemessene Ausgeglichenheit der Struktur		X	
Angemessene Aussagekraft	X[37]	X	
Angemessene Komplexität			X
Angemessene Modellierungssprachen und -elemente	X[38]	X	
Angemessene Modellierungssyntax	X[37]	X	
Angemessener Detaillierungsgrad		(X)[36]	X
Externe Minimalität		(X)[36]	X
Hohe Konsistenz		X	X
Hohe zweckorientierte Vollständigkeit		X	X
Qualität der SRS-Modellvisualisierungen			
Hohe Konsistenz der Visualisierung	X[39]	X	
Hohe Metaphernnutzung			X
Hohe sachlogisch richtige Visualisierung		X	X
Hohe Übersichtlichkeit der Visualisierung		X	
Niedrige Komplexität der Visualisierung	X[40]	X	

Tab. 4-1: Vorgeschlagene Umsetzung für die Qualitätssicherung der Qualitätskriterien.

[35] Über die Konfiguration sind Regeln für die erlaubten Modellierungselemente des Modells und Beziehungen zwischen den Elementen hinterlegt. So wird bereits ein Teil der möglichen syntaktischen Fehler durch das Modellierungswerkzeug verhindert, da sie im Werkzeug nicht modelliert werden können.

[36] (X): Es können Indizien für das Kriterium automatisch überprüft werden.

[37] Über die Konfiguration können für jedes Profil die erlaubten Beziehungen zwischen Modellelementtypen sowie die Anzahl der Beziehungen eingeschränkt werden.

[38] Über die Konfiguration kann für jedes Profil die mögliche Auswahl an Modellierungssprachen und -elementen eingeschränkt oder auch erweitert werden.

[39] Über die Konfiguration wird für einen Teil der Modellelemente eine einheitliche Visualisierung festgelegt. Des Weiteren können die Freiheitsgrade der unterschiedlichen Gestaltung zusätzlich eingeschränkt werden.

[40] Über die Konfiguration kann bspw. die maximale Anzahl an Beziehungen oder im Diagramm die maximale Anzahl von Instanzen eines Modellelementtyps festgelegt werden.

Ziel bei der Definition des Qualitätssystems war, die Qualitätskriterien so fein zu wählen, dass sie direkt im Modell nachweisbar und daher auch messbar und bewertbar sind.[41] Daher ist viel Arbeit in die Separierung von Qualitätseigenschaften in *Kriterien* und *Auswirkungen* eingegangen. *Kriterien* verstehen wir als direkt im Modell nachweisbare und überprüfbare Eigenschaften, z.B. *Angemessene Modellierungssprache und -wortschatz*. *Auswirkungen* sind Eigenschaften, die sich als Folge aus einem oder mehreren Kriterien ergeben und daher nicht direkt im Modell nachweisbar sind und somit auch nicht dort überprüft werden können, z.B. Verständlichkeit. Im Qualitätssystem sollen nur direkt im Modell nachweisbare und somit auch bewertbare *Kriterien* aufgeführt werden, keine *Auswirkungen*.

Für jedes Qualitätskriterium wird in Abschnitt 4.1.4 und 4.1.5 besprochen, ob die Qualitätssicherung nach unserem Kenntnisstand ausschließlich automatisch oder teilweise automatisch und teilweise manuell oder nur manuell umgesetzt werden kann. Zudem werden erste Ansätze für die automatische Umsetzung genauer erläutert. In Tab. 4-1 sind die vorgeschlagenen Umsetzungsarten für jedes Kriterium im Überblick zu sehen. Zusätzlich zur in den folgenden Abschnitten thematisierten manuellen und automatischen Qualitätssicherung haben wir die in Abschnitt 3.2.2 ebenfalls kurz vorgestellte Umsetzungsmöglichkeit einer automatischen konstruktiven Qualitätssicherung (Konfiguration) in der Tabelle aufgenommen.

In dem beschriebenen System aus Qualitätsfaktoren sind Ideen und Ansätze aus unterschiedlicher Literatur eingegangen (u.a. aus Abschnitt 3.4). Wir haben Standards und Qualitätsdefinitionen aus Wirtschaft (GoM, eCH-0158 und IEEE Std. 830-1998) sowie Forschung (GoMV, Pohl und 3QM-Framework) eingepflegt. Im Dialog mit den Betreuern der MID GmbH, sowie der universitären Betreuung aus dem Fachbereich Softwaretechnik, wurde das vorliegende Qualitätssystem von der Autorin iterativ erarbeitet.

Nachfolgend wird die Herleitung dieses Systems aus Qualitätsfaktoren, -kriterien und Metriken aus den benannten sieben Quellen beschrieben. (Die genaue Definition der Metriken wird in Abschnitt 4.2.3 aufgeführt.)

4.1.2 Detailbeschreibung des Eingehens der Definitionen aus 3.4 im Qualitätssystem

Nachdem wir Ansätze für Qualitätssicherung (siehe Abschnitt 3.2) betrachtet haben, wird in diesem Abschnitt detailliert beschrieben, ob und wo die Definitionen aus Abschnitt 3.4 in unser Qualitätssystem eingegangen sind.

4.1.2.1 GoM

Aus den von Becker, et al. (2012) vorgestellten GoM gehen in das von uns entwickelte Qualitätssystem alle ein (siehe Tab. 4-2).

[41] Sneed, et al., 2010 nennen den Aspekt, dass durch den GQM Ansatz messbare Ziele definiert werden können, als die ausschlaggebende Komponente für den Erfolg von GQM.

Kriterium	Def.	Vorkommen im Qualitätssystem
Grundsatz der Richtigkeit • Syntaktische Richtigkeit	GoM-1, S. 36	Qualität modellbasierter SRS • Qualität des SRS-Modells • Syntaktische Modellqualität • *Syntaktische Richtigkeit*
Grundsatz der Richtigkeit • Semantische Richtigkeit	GoM-2, S. 37	Qualität modellbasierter SRS • Qualität des SRS-Modells • Semantische Modellqualität • *Semantische Richtigkeit*
Grundsatz der Relevanz	GoM-3, S. 37	Qualität modellbasierter SRS • Qualität des SRS-Modells • Zweckorientierte Modellqualität • *Angemessener Detaillierungsgrad* **&** *Angemessene Modellierungssprachen und -elemente* **&** *Angemessene Aussagekraft* **&** *Hohe zweckorientierte Vollständigkeit*
Grundsatz der Relevanz • Externe Minimalität	GoM-4, S. 37	Qualität modellbasierter SRS • Qualität des SRS-Modells • Zweckorientierte Modellqualität • *Externe Minimalität*
Grundsatz der Relevanz • Interne Minimalität	GoM-5, S. 37	Qualität modellbasierter SRS • Qualität des SRS-Modells • Semantische Modellqualität • *Interne Minimalität*
Grundsatz der Wirtschaftlichkeit	GoM-6, S. 37	Qualität modellbasierter SRS • Qualität des SRS-Modells • Zweckorientierte Modellqualität • *Angemessener Detaillierungsgrad* **&** *Angemessene Ausgeglichenheit der Struktur*
Grundsatz der Klarheit	GoM-7, S. 38	Qualität modellbasierter SRS • Qualität des SRS-Modells • Zweckorientierte Modellqualität • *Angemessene Modellierungssprachen und -elemente* **&** *Angemessene Syntax* **&** *Angemessene Aussagekraft* Qualität modellbasierter SRS • Qualität der SRS-Modellvisualisierungen • *Hohe Übersichtlichkeit der Visualisierung*
Grundsatz der Vergleichbarkeit	GoM-8, S. 38	Qualität modellbasierter SRS • Qualität der SRS-Modellvisualisierungen • *Hohe Konsistenz der Visualisierung*
Grundsatz des systematischen Aufbaus	GoM-9, S. 38	Qualität modellbasierter SRS • Qualität des SRS-Modells • Zweckorientierte Modellqualität • *Hohe Konsistenz*

Tab. 4-2: Vorkommen der GoM im Qualitätssystem.

Dabei werden die folgenden Aspekte der GoM in unser Qualitätssystem identisch übernommen:

- *Syntaktische Richtigkeit* (Def. GoM-1) aus dem Grundsatz der Richtigkeit wird ins Qualitätssystem eingeordnet unter *Syntaktische Modellqualität* der Qualität des SRS-Modells
- *Semantische Richtigkeit* (Def. GoM-2) aus dem Grundsatz der Richtigkeit unter *Semantische Modellqualität* der Qualität des SRS-Modells
- *Externe Minimalität* (Def. GoM-4) aus dem Grundsatz der Relevanz unter *Zweckorientierte Modellqualität* der Qualität des SRS-Modells
- *Interne Minimalität* (Def. GoM-5) aus dem Grundsatz der Relevanz unter *Semantische Modellqualität* der Qualität des SRS-Modells

Bezüglich der Umsetzung des *Grundsatzes der Relevanz* (Def. GoM-3) ist in unserem Fall (der *modellbasierten SRS*) die allgemeine Zielsetzung des Modells bekannt. Ziel des *SRS-Modells* und der *–Modellvisualisierungen* ist die Kommunikation und Dokumentation der fachlichen Lösung des Problems, sowie den zugehörigen Anforderungen an das zu entwerfende Softwaresystem für alle Stakeholder. Den Grundsatz der Relevanz in diesen Kontext übersetzt ergibt die Forderung, dass im SRS-Modell nur solche Anforderungen enthalten sein sollen, die relevant sind, d.h. die das System erfüllen soll (vgl. *korrekt* in Abschnitt 3.4.3 und Qualitätskriterium *Hohe zweckorientierte Vollständigkeit*). Oder auch, dass nur solche Sprachen verwendet werden, die für den Modellierungszweck relevant sind (vgl. Qualitätskriterium *Angemessene Modellierungssprachen und -elemente*). Der Grundsatz der Relevanz geht somit an mehreren Stellen in Qualitätssystem unter dem Punkt *Zweckorientierte Modellqualität* der Qualität der SRS-Modellierung:

- *Angemessener Detaillierungsgrad,*
- *Angemessene Modellierungssprachen und -elemente,*
- *Angemessene Aussagekraft* und
- *Hohe zweckorientierte Vollständigkeit.*

Zu dem *Grundsatz der Wirtschaftlichkeit* (Def. GoM-6) schlagen Becker, et al. (2012) selbst Umsetzungsmöglichkeiten bzw. abgeleitete Regeln vor, die wir für das Qualitätssystem übernehmen. Unter dem Qualitätsfaktor *Zweckorientierte Modellqualität* siedeln wir das Qualitätskriterium *Angemessener Detaillierungsgrad* an, als auch das bereits erwähnte Kriterium *Angemessene Modellierungssprachen und -wortschatz*. Das Qualitätskriterium *Ausgeglichenheit der Struktur* lässt sich ebenfalls aus dem *Grundsatz der Wirtschaftlichkeit* ableiten: Es soll *„mit einem gegebenen Modellierungsaufwand ein Modell erreicht werden […], das dem Modellierungszweck am nächsten kommt“* (Becker, et al., 2012), somit kann ein Ziel sein, im Modell stets eine etwa gleichen Detaillierungsgrad über alle Teilmodelle zu halten. Daraus folgend werden die Detaillierungsebenen stufenweise für das gesamte Modell erarbeitet.

Den Grundsatz der Wirtschaftlichkeit (Def. GoM-6) ist zweifach im Qualitätssystem verortet:

- Unter *Zweckorientierte Modellqualität* der Qualität des SRS-Modells als *Angemessener Detaillierungsgrad*
- Unter *Zweckorientierte Modellqualität* der Qualität des SRS-Modells als *Angemessene Ausgeglichenheit der Struktur*

Der von Becker et al (2012) sehr allgemein gehaltene *Grundsatz der Klarheit* (Def. GoM-7) ist ebenfalls an mehreren Stellen unseres Qualitätssystems vertreten:

- Unter *Zweckorientierte Modellqualität* der Qualität des SRS-Modells
    - Als *Angemessene Modellierungssprachen und -elemente*
    - Als *Angemessene Syntax*
    - Als *Angemessene Aussagekraft*
- Unter *Qualität der SRS-Modellvisualisierungen*
    - Als *Niedrige Komplexität der Visualisierung*
    - Als *Sachlogisch richtige Visualisierung*
    - Als *Hohe Übersichtlichkeit der Visualisierung*

Die erste Forderung des *Grundsatzes der Vergleichbarkeit* (Def. GoM-8) kann unter dem Qualitätskriterium *Hohe Konsistenz der Visualisierung* im Qualitätssystem verortet werden (unter dem Faktor *Qualität des SRS-Modellvisualisierungen*), da im ersten Teil dieses Grundsatzes folgendes gefordert wird: Im Original übereinstimmende Abläufe, die in der gleichen Modellierungssprache im Modell beschrieben werden, sollen ebenfalls gleich dargestellt werden und somit die Gleichheit der Originale widerspiegeln. Dies kann durch die konsistente Darstellung ermöglicht werden. Der zweite Aspekt des *Grundsatzes der Vergleichbarkeit* beschäftigt sich mit der Überführbarkeit von Modellen unterschiedlicher Modellierungssprachen mit dem Ziel, diese Modelle miteinander vergleichen zu können. Da unser Ziel nicht der Vergleich mehrerer unterschiedlicher Modellierungen untereinander, sondern die Bewertung und Sicherstellungen eines definierten minimalen Qualitätsniveaus des SRS-Modells und seiner Modellvisualisierungen, ist dieser Aspekt für unsere Zwecke nicht relevant und wird somit ausgelassen.

Den Grundsatz der Vergleichbarkeit (Def. GoM-8) haben wir in unsere Qualitätsbetrachtung nicht mit einbezogen.

Den *Grundsatz des systematischen Aufbaus* (Def. GoM-9) ordnen wir als Teilaspekt in das Kriterium *Hohe Konsistenz* (unter *Zweckorientierte Modellqualität*) in unser Qualitätssystem ein.

In Tab. 4-2 ist nochmals zusammengefasst zu sehen, ob und an welcher Stelle der jeweilige Grundsatz der GoM in das Qualitätssystem in Abschnitt 5.2 eingeht.

4.1.2.2 GoMV

Die folgenden Grundsätze der GoMV wurden von uns unverändert in das Qualitätssystem unter *Qualität der SRS-Modellvisualisierungen* übernommen:

- *Grundsatz des authentischen Darstellung* (Def. GoMV-1) als *Sachlogisch richtige Visualisierung* unter Qualität der SRS-Modellvisualisierungen
- *Grundsatz der Metaphernnutzung* (Def. GoMV-4) als *Hohe Metaphernnutzung* unter Qualität der SRS-Modellvisualisierungen
- *Grundsatz der Konsistenz* (Def. GoMV-6) als *Hohe Konsistenz der Visualisierung* unter Qualität der SRS-Modellvisualisierungen

Diese Grundsätze übernehmen wir als Bestandteil eines anderen Kriteriums:

- *Grundsatz des inhaltlichen Minimalprinzips* (Def. GoMV-2) und
- *Grundsatz des minimalen Visualisierungsgrades* (Def. GoMV-3) unter *Niedrige Komplexität der Visualisierung* der Qualität der SRS-Modellvisualisierungen
- *Grundsatz der Wiederverwendung* (Def. GoMV-5) unter *Zweckorientierter Modellqualität* der Qualität des SRS-Modells

Ob und an welcher Stelle der jeweilige Grundsatz aus den GoMV in das Qualitätssystem in Abschnitt 5.2 eingeht, ist in Tab. 4-3 nochmals zusammengefasst zu sehen

Kriterium	Def.	Vorkommen im Qualitätssystem
Grundsatz des authentischen Darstellung	GoMV-1, S. 39	Qualität modellbasierter SRS • Qualität der SRS-Modellvisualisierungen • *Sachlogisch richtige Visualisierung*
Grundsatz des inhaltlichen Minimalprinzips	GoMV-2, S. 39	Qualität modellbasierter SRS • Qualität der SRS-Modellvisualisierungen • *Niedrige Komplexität der Visualisierung*
Grundsatz des minimalen Visualisierungsgrades	GoMV-3, S. 39	Qualität modellbasierter SRS • Qualität der SRS-Modellvisualisierungen • *Niedrige Komplexität der Visualisierung*
Grundsatz der Metaphernnutzung	GoMV-4, S. 38	Qualität modellbasierter SRS • Qualität der SRS-Modellvisualisierungen • *Hohe Metaphernnutzung*
Grundsatz der Wieder-verwendung	GoMV-5, S. 39	Qualität modellbasierter SRS • Qualität des SRS-Modells • Zweckorientierte Modellqualität • *Hohe Konsistenz*
Grundsatz der Konsistenz	GoMV-6, S. 39	Qualität modellbasierter SRS • Qualität der SRS-Modellvisualisierungen • *Hohe Konsistenz der Visualisierung*

Tab. 4-3: Vorkommen der GoMV im Qualitätssystem.

4.1.2.3 3QM-Framework

Von dem Qualitätssystem des 3QM-Frameworks haben wir die Strukturierung der Qualitätseigenschaften durch die Unterteilung in Syntax, Semantik und Pragmatik für unseren Qualitätsfaktor Modellqualität abgeschaut. Abweichend von der Unterteilung setzten wir an der Stelle der Pragmatik die Zweckorientierung ein.

Darüber hinaus übernehmen wir für die Syntax die Metriken bezüglich des *Wort-*, *Satz-* und *Textsyntax* geringfügig abgewandelt. Overhage, et al. (2010) untersuchen bspw. unter dem Begriff *Wortsyntax*, ob die verwendeten Zeichen konform zur Modellierungssprache sind. Da in unserem Kontext dies bereits durch das Modellierungswerkzeug konstruktiv sichergestellt wird, untersuchen wir unter dem Titel „Verstöße gegen Wortsyntax“, inwiefern die Wörter der Modellierungssprache richtig verwendet werden, d.h. z.B. BPMN-Aktivitäten beschreiben Tätigkeiten, BPMN-Ereignisse werden für die Beschreibung von Zuständen verwendet. Wie beim Qualitätssystem des 3QM-Frameworks wollen wir die Metriken für *Original* und *Wiederholte Verstöße* berechnen.

Die restlichen Metriken von Overhage, et al. sind für unsere Zwecke nicht brauchbar. Dies rührt daher, dass die Auswertung der Metriken von ihnen manuell vorgenommen wurde, sowie, dass die Modelle nicht in einem Modellierungswerkzeug modelliert wurden. Diese gegenüber der vorliegenden Arbeit abweichenden Voraussetzungen haben sich auf ihre Definition der Metriken als auch die Messvorschriften ausgewirkt, wie wir es bereits am Beispiel der Wortsyntax gezeigt haben.

4.1.2.4 eCH-0158

Aus den fünfzehn im Rahmen dieser Arbeit vollständig betrachteten Konventionen des eCH-0158 bezüglich Pools und Startereignisse sind elf ins Qualitätssystem eingegangen. Von diesen beziehen sich wiederum zwei Konventionen ausschließlich auf Gestaltungsaspekte, sind daher unter dem Kriterium Hohe Übersichtlich-

keit der Visualisierung der Qualität des SRS-Modellvisualisierungen eingeordnet (siehe Tab. 4-4). Die anderen neun sind unter Angemessene Aussagekraft der Zweckorientierten Modellqualität eingeordnet. Die vier nicht im Qualitätssystem berücksichtigten Konventionen werden bereits durch den Innovator gewährleistet (eCH0158-Pool-06 und eCH0158-Startereignis-07) oder sind konstruktiver Natur (eCH0158-Pool-01 und eCH0158-Pool-02).

4.1.2.5 ANFORDERUNGEN NACH POHL (2010)

Die Qualitätskriterien für Anforderungen nach Pohl (2010) werden an unterschiedlichen Stellen in das Qualitätssystem eingeordnet:

- *Vollständig* (Def. REQ-1), *nachvollziehbar* (Def. REQ-2), *korrekt* (Def. REQ-3) und *bewertet* (Def. REQ-8) unter dem Kriterium *Hohe zweckorientierte Vollständigkeit* der Zweckorientierten Modellqualität
- *Überprüfbar* (Def. REQ-7) und *atomar* (Def. REQ-10) unter dem Kriterium *Angemessene Aussagekraft* der Zweckorientierten Modellqualität

Das Kriterium *nachvollziehbar* nach Def. REQ-2 geht dabei nur teilweise in unser System ein. Für die betrachtete Phase der Anforderungsanalyse ist nur die Nachvollziehbarkeit der Anforderungsquelle von Bedeutung und zu diesem Zeitpunkt überprüfbar, da die Entwicklung und Auswirkungen der Anforderung erst zu späteren Phasen der Softwareentwicklung entstehen.

Die Kriterien *eindeutig* (Def. REQ-4) und *verständlich* (Def. REQ-5) sind mehrfach im Qualitätssystem verortet:

Eine Anforderung ist *eindeutig*, wenn zum einen sich eine Sprachgemeinschaft durch den Diskurs der Modellnutzer gebildet hat (vgl. Syntaktische Richtigkeit nach Def. GoM-1) und zum anderen das Modell konsistent ist und die in der Anforderung verwendeten Begriffe konsistent verwendet werden. Somit ordnen wir *eindeutig* (Def. REQ-4) als Bestandteil der folgenden Kriterien ein:

- *Semantische Richtigkeit* unter *Semantische Modellqualität* der Qualität des SRS-Modells und
- *Hohe Konsistenz* unter *Zweckorientierte Modellqualität* der Qualität des SRS-Modells

Wie verständlich eine Anforderung ist, wird von vielen Einflussgrößen bestimmt. Wie im vorherigen Abschnitt bereits als Beispiel erwähnt, ordnen wir Verständlichkeit der Kategorie der Auswirkungen zu. Wir wollen jedoch in unserem System Qualitätsaspekte betrachten, die Kriterien sind, d.h. direkt im Modell nachweisbar sind. Viele Qualitätskriterien des Qualitätssystems haben eine höhere Verständlichkeit des Modells zur Auswirkung wie z.B. alle Kriterien zur *Qualität der SRS-Modellvisualisierungen*.

Die Forderung nach Konsistenz innerhalb einer Anforderung, nach der Definition von REQ-6, ist in unserem Kontext eine Forderung nach Konsistenz für Modellelemente. Somit ist sie ein Teilaspekt des von uns aufgestellten Qualitätskriterium *Hohe Konsistenz* des *SRS-Modells*.

Kürzel	Definition	Vorkommen im Qualitätssystem
4.3	***Pool***	
eCH0158-Pool-01	*„Pools werden in der Regel mit Organisationseinheiten oder dem Namen anderer Prozessbeteiligter bezeichnet.“*	---
eCH0158-Pool-02	*„Vorlagen mit den am häufigsten verwendeten Pools erleichtern die Arbeit und erhöhen die Lesbarkeit.“*	---
eCH0158-Pool-03	*„Reihenfolge oder Farbgebung kann der Identifizierung interner / externer Pools dienen.“*	Qualität modellbasierter SRS • Qualität des SRS-Modellvisualisierungen • Hohe *Übersichtlichkeit der Visualisierung*
eCH0158-Pool-04	*„In der Regel wird nur der eigene Pool aufgeklappt dargestellt.“*	Qualität modellbasierter SRS • Qualität des SRS-Modells • Zweckorientierte Modellqualität • *Angemessene Aussagekraft*
eCH0158-Pool-05	*„In jedem aufgeklappten Pool wird genau ein vollständiger Prozess modelliert.“*	Qualität modellbasierter SRS • Qualität des SRS-Modells • Zweckorientierte Modellqualität • *Angemessene Aussagekraft*
eCH0158-Pool-06	*„Pools werden übereinander über die gesamte Diagrammbreite dargestellt.“*	---
eCH0158-Pool-07	*„Die Höhe des aufgeklappten Pools richtet sich nach dessen Inhalt.“*	Qualität modellbasierter SRS • Qualität des SRS-Modellvisualisierungen • Hohe *Übersichtlichkeit der Visualisierung*
eCH0158-Pool-08	*„Zugeklappte Pools enthalten mindestens einen eingehenden oder ausgehenden Nachrichtenfluss.“*	Qualität modellbasierter SRS • Qualität des SRS-Modells • Zweckorientierte Modellqualität • *Angemessene Aussagekraft*
4.5.1	***Startereignis***	
eCH0158-Startereignis-01	Unbestimmtes Startereignis: *„Wird nicht beschrieben, der Unterprozess wird durch den Aufruf auf der übergeordneten Prozessebene gestartet.“*	Qualität modellbasierter SRS • Qualität des SRS-Modells • Zweckorientierte Modellqualität • *Angemessene Aussagekraft*
eCH0158-Startereignis-02	Nachrichten-Startereignis: *„Wird nicht beschrieben, wenn die eingehende Nachricht auf dem (obligatorischen) Nachrichtenfluss ersichtlich ist.“*	Qualität modellbasierter SRS • Qualität des SRS-Modells • Zweckorientierte Modellqualität • *Angemessene Aussagekraft*
eCH0158-Startereignis-03	Bedingungs-Startereignis: *„Bedingung ist in der Bezeichnung des Elementes festzuhalten. Sollte ein Bedingungs-Startereignis einem Endereignis eines anderen Prozesses entsprechen, ist es identisch zu bezeichnen.“*	Qualität modellbasierter SRS • Qualität des SRS-Modells • Zweckorientierte Modellqualität • *Angemessene Aussagekraft*
eCH0158-Startereignis-04	Zeitgeber-Ereignis: *„Bezeichnung enthält den Zeitpunkt für den Start. Beispiele: „am 1. je Monat“, „09:00 Uhr“.“*	Qualität modellbasierter SRS • Qualität des SRS-Modells • Zweckorientierte Modellqualität • *Angemessene Aussagekraft*
eCH0158-Startereignis-05	*„Unbestimmte Startereignisse werden nur verwendet, wenn der Prozess durch den Aufruf auf der übergeordneten Prozessebene gestartet wird.“*	Qualität modellbasierter SRS • Qualität des SRS-Modells • Zweckorientierte Modellqualität • *Angemessene Aussagekraft*
eCH0158-Startereignis-06	*„Ein Prozess hat mindestens ein Startereignis.“*	Qualität modellbasierter SRS • Qualität des SRS-Modells • Zweckorientierte Modellqualität • *Angemessene Aussagekraft*
eCH0158-Startereignis-07	*„Startereignisse werden innerhalb eines Pools dargestellt.“*	---

Tab. 4-4: Vorkommen des eCH-0158 im Qualitätssystem.

Die geforderte Eigenschaft der *Aktualität* (Def. REQ-9) lässt sich schwer in unser Qualitätssystem einordnen, denn sie bezieht sich auf eine zeitliche Komponente, die in unserem Qualitätsmodell so nicht abgebildet wird. Eine Anforderung soll demnach den aktuellen Stand des Systems oder des Systemkontexts widerspiegeln, wie z.B. die aktuellen Wünsche der Stakeholder oder aktuelle gesetzliche Vorschriften. Es geht hier darum, ob das Original im Modellelement Anforderung so beschrieben wird, wie es zum aktuellen Zeitpunkt gewünscht wird. Dies impliziert semantische Aspekte wie folgende:

- Beschreibt das Modellelement das Original richtig? (vgl. *Semantische Richtigkeit*) und
- Beschreibt das Modellelement nur Komponenten, die auch im Original existieren? (vgl. *Interne Minimalität*)

Aber auch die Frage: Inwieweit sind die Aussagen, die das Modellelement über das Original angemessen? (vgl. *Angemessene Aussagekraft*) Angemessen bedeutet in diesem Fall aktuell. Da uns die Zuordnung zu keiner dieser Qualitätskriterien vollends passend erscheint, haben wir in unser Qualitätssystem die Forderung nach *aktuellen* Anforderungen vorerst nicht aufgenommen. Somit wird die Verständlichkeit in keinem der Kriterien direkt betrachtet, jedoch indirekt in fast jedem der Unterkriterien der vier Qualitätsfaktoren.

In Tab. 4-5 (S. 61) ist nochmals zu sehen, an welcher Stelle die Qualitätskriterien für Anforderungen von Pohl (2010) im Qualitätssystem eingegangen sind.

4.1.2.6 SRS-Dokumentation nach IEEE Std 830-1998

Die Qualitätsanforderung *korrekt* (Def. SRS-1), *eindeutig* (Def. SRS-2), *bewertet* (Def. SRS-5), *überprüfbar* (Def. SRS-6) und *nachvollziehbar* (Def. SRS-8) an SRS werden hier definiert durch eine Qualitätsanforderung an jede einzelne in der SRS enthaltene Anforderung. Unter *korrekt* wird bspw. gefordert, dass in der SRS nur Anforderungen enthalten sein dürfen, die von der zu entwickelnden Software erfüllt werden müssen. Diese Qualitätsanforderungen für Anforderung werden von Pohl (2010) ebenfalls definiert (siehe Abschnitt 3.4.2.2) und werden somit in Abschnitt 4.1.2.5 besprochen.

Das Qualitätsmerkmal *vollständig* (Def. SRS-3) für SRS hat nach der Definition von IEEE drei Aspekte, die wir wie folgt in unser Qualitätssystem einordnen:

a) Dass alle relevanten Anforderungen in der SRS enthalten sind, ist ein Teilaspekt unseres Kriteriums *Externe Minimalität* der Zweckorientierten Modellqualität (unter Qualität des SRS-Modells).
b) Dass alle Antworten des Systems zu allen möglichen Eingabedaten definiert im SRS definiert sein müssen, ist ein Teilaspekt der *Hohen zweckorientierten Vollständigkeit* ebenfalls der Zweckorientierten Modellqualität.
c) Dass alle Beschriftungen und Verweise bezüglich der Diagramme vollständig sind, sollte im Kontext der *Angemessenen Aussagekraft* untersucht werden. Die vollständige Definition aller Begriffe und Maßeinheiten ist ein Teilthema der *Hohen Zweckorientierten Modellqualität*. Dass alle Beschriftungen und Verweise bezüglich Abbildungen und Tabellen vollständig sind bezieht sich in unserem Fall auf die *Qualität der modellexternen Texte*.

Kriterium	Def.	Vorkommen im Qualitätssystem
Vollständig	REQ-1	Qualität modellbasierter SRS • Qualität des SRS-Modells • Zweckorientierte Modellqualität • *Hohe zweckorientierte Vollständigkeit*
Nachvollziehbar	REQ-2	Qualität modellbasierter SRS • Qualität des SRS-Modells • Zweckorientierte Modellqualität • *Hohe zweckorientierte Vollständigkeit*
Korrekt	REQ-3	Qualität modellbasierter SRS • Qualität des SRS-Modells • Zweckorientierte Modellqualität • *Hohe zweckorientierte Vollständigkeit*
Eindeutig	REQ-4	Qualität modellbasierter SRS • Qualität des SRS-Modells • Semantische Modellqualität • *Semantische Richtigkeit* Qualität modellbasierter SRS • Qualität des SRS-Modells • Zweckorientierte Modellqualität • *Hohe Konsistenz*
Verständlich	REQ-5	---
Konsistent	REQ-6	Qualität modellbasierter SRS • Qualität des SRS-Modells • Zweckorientierte Modellqualität • *Hohe Konsistenz*
Überprüfbar	REQ-7	Qualität modellbasierter SRS • Qualität des SRS-Modells • Zweckorientierte Modellqualität • *Angemessene Aussagekraft*
Bewertet	REQ-8	Qualität modellbasierter SRS • Qualität des SRS-Modells • Zweckorientierte Modellqualität • *Hohe zweckorientierte Vollständigkeit*
Aktuell	REQ-9	---
Atomar	REQ-10	Qualität modellbasierter SRS • Qualität des SRS-Modells • Zweckorientierte Modellqualität • *Angemessene Aussagekraft*

Tab. 4-5: Vorkommen Anforderungen nach Pohl (2010) im Qualitätssystem.

Unter *Konsistenz* (Def. SRS-4) nach IEEE wird nicht nur gefordert, dass das SRS-Dokument in sich widerspruchsfrei ist, d.h. in unserem Fall das SRS-Modell konsistent ist (vgl. *Hohe Konsistenz* unter Zweckorientierte Modellqualität). Sondern es wird auch verlangt, dass das Dokument auch zu übergeordneten Dokumenten konsistent ist. Dieser Aspekt wird in unserem Qualitätssystem nicht betrachtet, da wir ausschließlich die Qualität des Modells behandeln

Das Qualitätskriterium der *Modifizierbarkeit* (Def. SRS-7) für SRS-Dokumentationen erfüllt unsere modellbasierte SRS in einem gewissen Rahmen von vornherein dadurch, dass Änderungen im Modell vorgenommen werden und nicht im Dokument. Dadurch behält die Dokumentation bei Änderungen ihre Struktur und ihren Stil, wie es das Qualitätskriterium *modifizierbar* fordert (vgl. a der Def. SRS-7). Da es jedes Objekt im Modell nur einmal gibt, auch wenn es in der Dokumentation mehrfach auftaucht, ist die jeweilige Änderung vollständig und konsistent, ebenfalls wie gefordert (vgl. b) der Def. SRS-7). Die letzte Forderung der Def. SRS-7 (vgl. c), jede Anforderung soll einzeln und nicht mit anderen Anforderungen vermischt formuliert werden, fordert die Atomarität einzelner Anforderungen. Da sich dies ausschließlich auf das Modellelement Anforderung bezieht, wird dieses Kriterium nicht an dieser Stelle, sondern in Abschnitt 4.1.2.5 unter *atomar* besprochen.

An welcher Stelle die Qualitätskriterien für SRS-Dokumentationen des IEEE Std 830-1998 im Qualitätssystem eingegangen sind, ist zusätzlich in Tab. 4-6 aufgeführt.

Kriterium	Def.	Vorkommen im Qualitätssystem
Korrekt	SRS-1	Siehe *Anforderungen nach Pohl (2010)*, Abschnitt 4.1.2.5.
Eindeutig	SRS-2	Siehe *Anforderungen nach Pohl (2010)*, Abschnitt 4.1.2.5.
Vollständig	SRS-3	a) Qualität modellbasierter SRS • Qualität der SRS-Modells • Zweckorientierte Modellqualität • *Externe Minimalität* b) Qualität modellbasierter SRS • Qualität der SRS-Modells • Zweckorientierte Modellqualität • *Hohe zweckorientierte Vollständigkeit* c) Qualität modellbasierter SRS • Qualität der SRS-Modells • Zweckorientierte Modellqualität • *Hohe zweckorientierte Vollständigkeit* **&** *Angemessene Aussagekraft* **&** *Qualität der modellexternen Texte*
Konsistent	SRS-4	Qualität modellbasierter SRS • Qualität der SRS-Modells • Zweckorientierte Modellqualität • *Hohe Konsistenz*
Bewertet	SRS-5	Siehe *Anforderungen nach Pohl (2010)*, Abschnitt 4.1.2.5.
Überprüfbar	SRS-6	Siehe *Anforderungen nach Pohl (2010)*, Abschnitt 4.1.2.5.
Modifizierbar	SRS-7	---
Nachvollziehbar	SRS-8	Siehe *Anforderungen nach Pohl (2010)*, Abschnitt 4.1.2.5.

Tab. 4-6: Vorkommen SRS-Dokumentation nach IEEE Std 830-1998 im Qualitätssystem.

4.1.3 Qualität modellbasierter SRS

Mit Hilfe unseres hierarchischen Qualitätssystems wollen wir die Frage beantworten: *„In welchem Umfang erfüllt die modellbasierte Anforderungsspezifikation die Qualitätsanforderungen?“* Wie bereits in Abschnitt 0 erklärt, ist mit *modellbasierter Anforderungsspezifikation* hier das Dokument gemeint, welches mit Hilfe des Dokumentationsgenerators aus den im *SRS-Modell* hinterlegten Informationen, den Visualisierungen des Modells (*Modellvisualisierungen* bzw. Diagramme) und den hinterlegten *modellexternen Texten* erstellt wird. Durch diese Voraussetzungen ergibt sich die Qualität der modellbasierten SRS aus der Güte der folgenden vier Qualitätsfaktoren (siehe Abb. 4-2, S. 63):

- *Qualität des SRS-Modells*,
- *Qualität der SRS-Modellvisualisierungen*,
- *Qualität der modellexternen SRS-Texte* und
- *Qualität des SRS-Dokumentationsgenerators*.

Nachfolgend werden diese Qualitätsfaktoren definiert und die zugehörigen Qualitätskriterien des jeweiligen Faktors genauer betrachtet.

4.1.4 Qualität des SRS-Modells

Der Problemstellung dieser Arbeit folgend wird angenommen, dass eine modellbasierte SRS u.a. auf Grundlage eines SRS-Modells generiert wird. Dieses Modell muss eine gewisse Qualität aufweisen, denn alle im modellbasierten SRS-Dokument aufgeführten Informationen werden den im Modell hinterlegten Informationen entnommen.

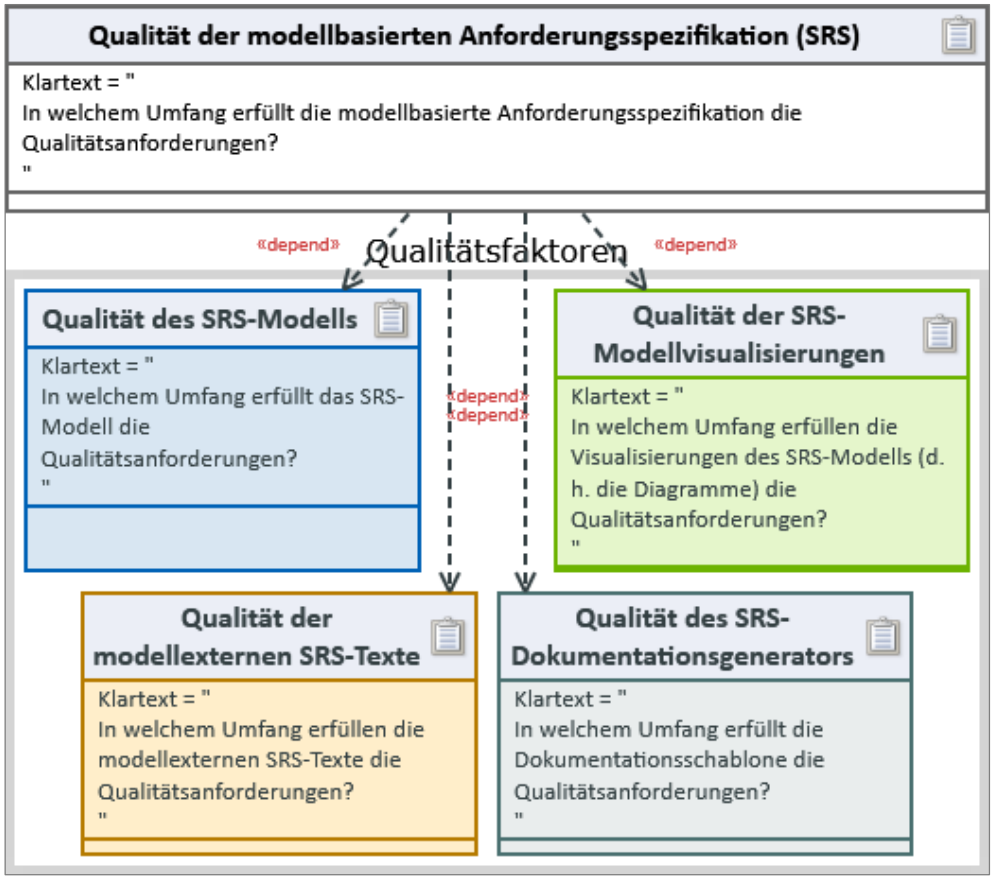


Abb. 4-2: Qualitätssystem Ebene 0–1 „Qualität der modellbasierten SRS“.

Bei der Betrachtung der *Qualität des SRS-Modells* soll die Frage beantwortet werden: *„In welchem Umfang erfüllt das SRS-Modell die Qualitätsanforderungen?“*. Ähnlich dem 3QM-Framework (3.4.2.1) unterscheiden wird hier die drei Teilbereiche *Syntaktische*, *Semantische* und *Zweckorientierte Modellqualität* (siehe Abb. 4-3). Diese Unterscheidung ist durch die in Abschnitt 2.1 eingeführte Eigenschaften von Modellen motiviert. Ein Modell hat demnach eine *Syntax* (d.h. Regeln für die jeweilige Modellierungssprache), eine *Semantik* (d.h. Bedeutung) und verfolgt mindestens einen *Zweck*.

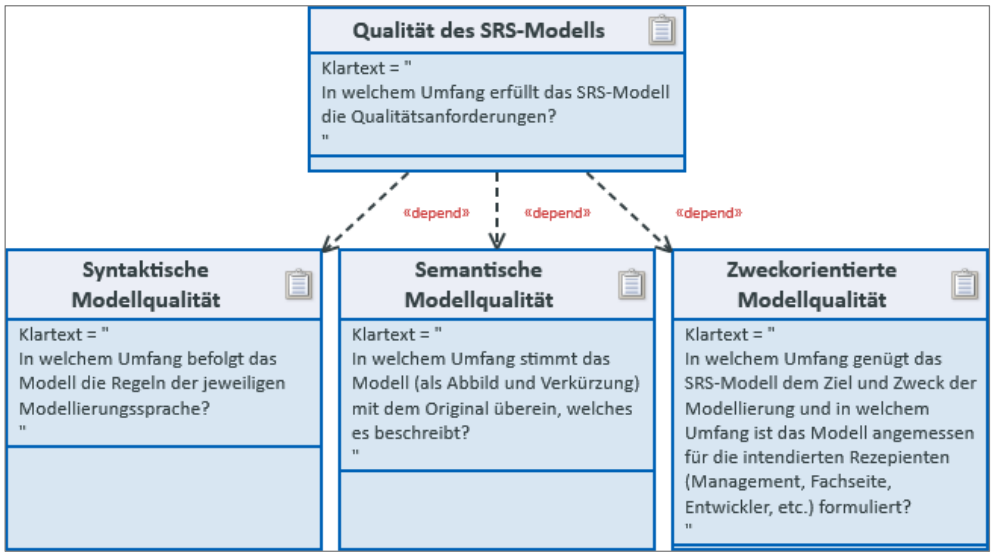


Abb. 4-3: Qualitätssystem Ebene 1–2 „Qualität des SRS-Modells“.

4.1.4.1 SYNTAKTISCHE MODELLQUALITÄT

Unter dem Qualitätsfaktor *Syntaktische Modellqualität* verstehen wir die Güte der Einhaltung der Modellierungssprachen-Syntax. Dieser Faktor hängt nach unserem Qualitätssystem ausschließlich von dem Qualitätskriterium *Syntaktische Richtigkeit* ab (siehe Abb. 4-4).

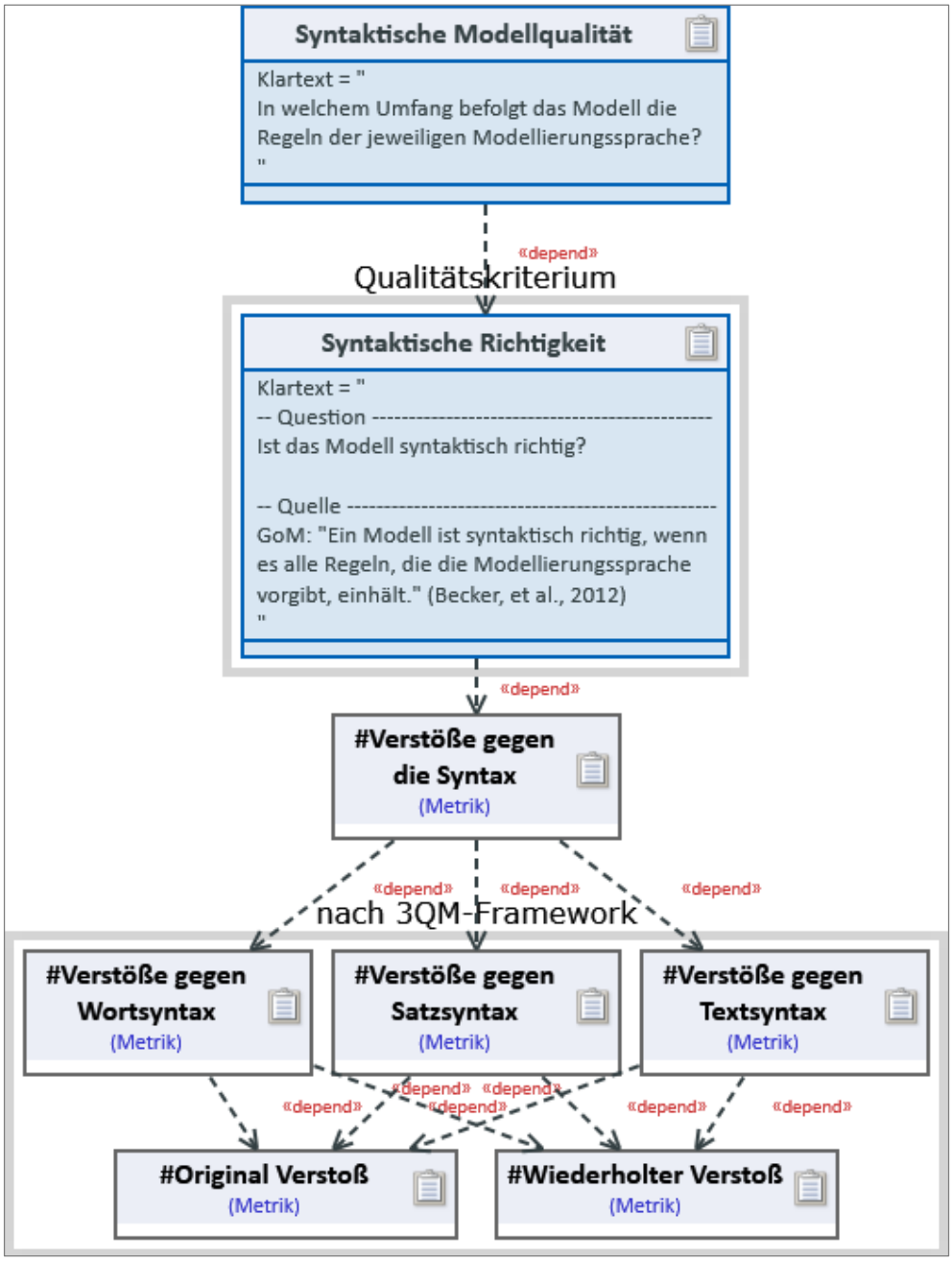


Abb. 4-4: Qualitätssystem Ebenen 2–n „Syntaktische Modellqualität“.

Bei der Modellbewertung trifft die *Syntaktische Richtigkeit* Aussagen darüber, inwiefern das Modell die Regeln der Modellierungssprache einhält. Da die Regeln der Syntax eindeutig formuliert und somit auch eindeutig auf Einhaltung überprüft werden können, kann *Syntaktische Richtigkeit* mittels **automatischer Prüfung** umgesetzt werden. Für die Messung und automatische Bewertung der syntaktischen Richtigkeit kann die Anzahl der Verstöße gegen die Syntax gezählt werden. Wir unterscheiden in unserem Qualitätssystem bei Syntaxverstößen zwischen einem Verstoß gegen die *Wort*-, *Satz*- oder *Text*syntax. Abweichend von Overhage,

et al. (2012) verstehen wir unter den Begriffen folgendes: Die *Text*syntax entspricht der Syntax des Modells von Becker, et al. (2012), es sollen hier modellübergreifende Beziehungen und Abhängigkeiten betrachtet werden wie z.B. Beziehungen zwischen Anforderungen und Anwendungsfällen. Die Modellteilbereiche wie bspw. ein einzelner BPMN-Prozess sind nach unserer Auffassung *Sätze* und die Einhaltung der Modellierungsregeln eines einzelnen Prozessschritts wird unter dem Punkt *Wort*syntax untersucht. Gegen die *Wort*syntax wird bspw. verstoßen, wenn ein BPMN-Ereignis mit einer Tätigkeit beschriftet wird. In diesem Fall wird das falsche *Wort* „Ereignis" verwendet, da eigentlich eine Tätigkeit modelliert werden soll. Das richtige *Wort* für den Ausdruck ist eine Aktivität also bspw. ein Task.

Darüber hinaus wird unterschieden zwischen einem *ersten* Verstoß gegen eine Regel der Modellierungssprache („Original Verstoß", d.h. die Anzahl der Verstoß-Arten) und *erneuten* Verstößen gegen die jeweilige Regel („Wiederholter Verstoß", d.h. die Anzahl des jeweiligen Verstoß-Vorkommens subtrahiert mit eins). Die erste Metrik liefert eine Zahl über die Anzahl der unterschiedlichen Regelverstöße und die zweite über Häufigkeit der Verstöße. Alle vorgestellten Metriken können automatisch ermittelt werden. Sind Schwellwerte für die jeweilige Metrik bezüglich Qualitätslevels definiert (siehe Abschnitt 4.2.4), kann basierend auf den Metriken das Qualitätskriterium *Syntaktische Richtigkeit* und somit auch die *Syntaktische Modellqualität* ebenfalls automatisch bewertet werden.

Zusammenfassend schlagen wir für die *Syntaktische Modellqualität* eine automatische Prüfung des Modells auf Verstöße gegen die Syntaxregeln der Modellierungssprache vor. Basierend auf den Prüfergebnissen können die vorgestellten sechs Metriken berechnet werden. Für die Bewertung des Qualitätsfaktors können Schwellwerte für jede Metrik festgelegt werden und so ermittelt werden, wie „gut" das Modell die Anforderungen an die geforderte Güte der *Syntaktischen Modellqualität* erfüllt.

4.1.4.2 SEMANTISCHE MODELLQUALITÄT

Das Modell, welches im Modellierungswerkzeug entworfen wird, bildet die Realität ab und stellt dabei an vielen Stellen eine Verkürzung dieser dar. Unter der *Semantischen Modellqualität* wird allgemein überprüft *In welchem Umfang das Modell als Abbild und Verkürzung mit dem Original übereinstimmt, welches es beschreibt*. Hierbei unterscheiden wir zwischen den Qualitätskriterien *Semantische Richtigkeit* und *Interne Minimalität* (siehe Abb. 4-5, S. 66) Die beiden Kriterien sind voneinander unabhängig und es sind Situationen vorstellbar in denen sie sich sogar widersprechen.

Unter *Semantischer Richtigkeit* wird untersucht, ob das Modell eine richtige Beschreibung des Originals ist. Wir übernehmen die Definition aus den GoM, dass ein Modell semantisch richtig ist, wenn sich die gutwilligen und sachkundigen Modellnutzer darauf geeinigt haben, das Original durch dieses Modell zu beschreiben (vgl. Def. GoM-2). Demnach muss die *Semantische Richtigkeit* hauptsächlich **manuell** geprüft werden (z.B. Walkthroughs), da das Modell hier gegen das Original „Realität" oder „zukünftige Realität" geprüft werden muss.

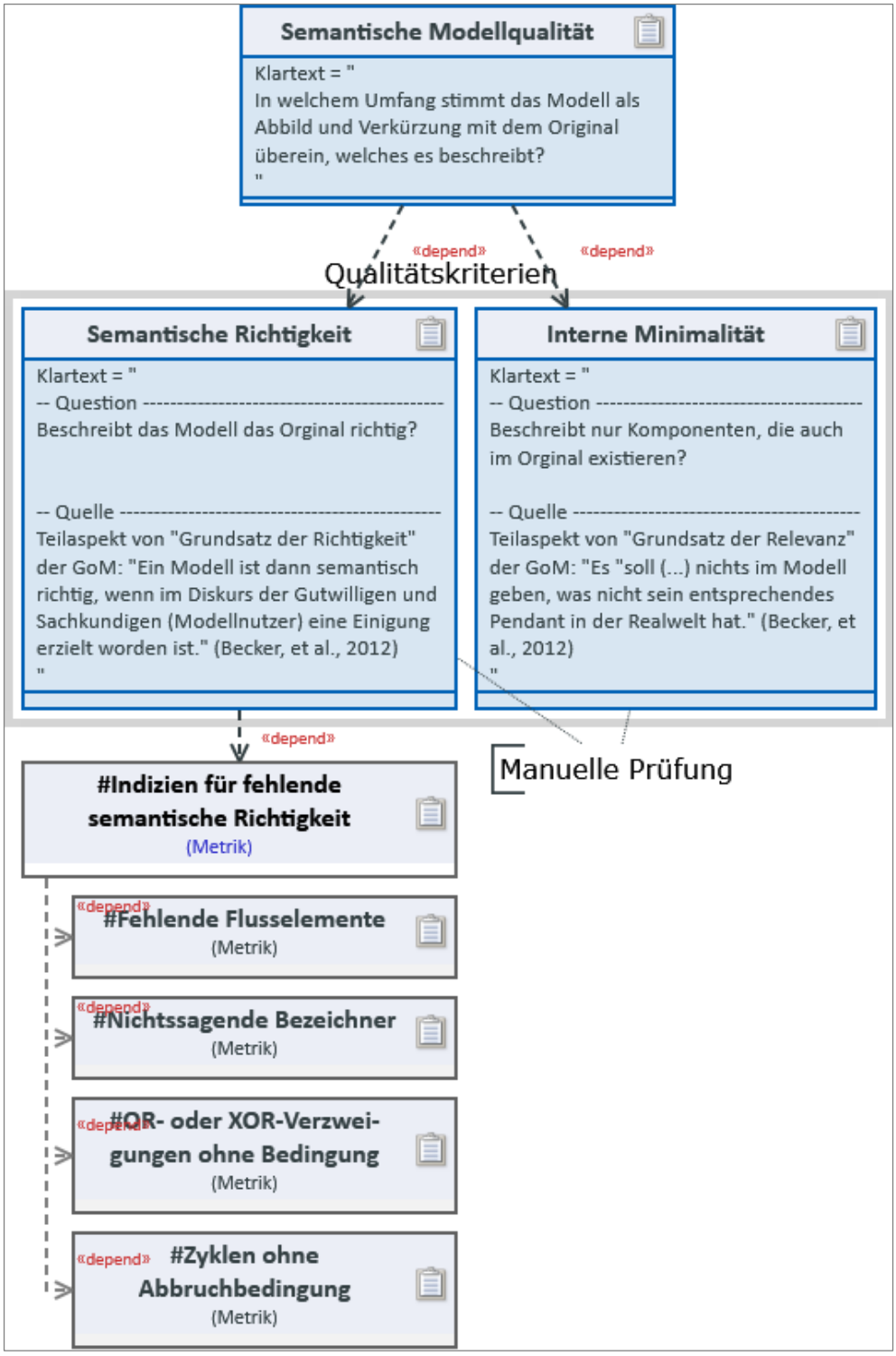


Abb. 4-5: Qualitätssystem Ebenen 2–n „Semantische Modellqualität“.

Unter der *Internen Minimalität* soll sichergestellt werden, dass die im Modell dargestellten Sachverhalte nicht über die der Realität hinausgehen, also dass das Dargestellte mindestens genauso oder komplexer bzw. konkreter ist. Bspw. muss der modellierte Prozess „Urlaubsanträge verwalten“ (siehe Abb. 1-5, S. 5, in Abschnitt 1.1) in der Realität mindestens die modellierte Abfolge von Prozessschritten beinhalten. In der Realität kann der Prozess weitere Schritte wie die mündliche Absprache der Urlaubsgenehmigung zwischen Vorgesetztem und Mitarbeiter beinhalten. Nach dem *Grundsatz der Relevanz* darf es jedoch keinen Prozessschritt im Modell geben, der in der Realität nicht existiert.

Semantische Richtigkeit und *Interne Minimalität* können nach unserem Kenntnisstand hauptsächlich **manuell** sichergestellt werden, da hier das Modell gegen die Realität geprüft werden muss. Jedoch können Indizien für fehlende Semantik formuliert werden, auf die das Modell **automatisch** überprüft werden kann. Als Beispiele für solche Prüfungen schlagen wir die folgenden Indizien vor:

- Zyklen ohne Abbruchbedingung,
- OR- oder XOR-Verzweigungen ohne Bedingung,
- Nichtssagende Bezeichner und
- Fehlende Ablauf- oder Flusselemente.

Die Definition dieser Indizien-Metriken ist in Abschnitt 4.2.3 in Tab. 4-14 bis Tab. 4-20 zu finden. In der BPMN ist ein „Zyklus ohne Abbruchbedingung“ z.B. ein Mehrfachtask ohne Abbruchbedingung. Ein Beispiel für nichtssagende Bezeichner im *SRS-Modell* sind die Default-Namen „Neue Anforderung“, „Neue Anforderung_185“, usw. für Anforderungen.

Auch bei der automatischen Prüfung sollte der Modellierer stets manuell überprüfen, ob an der identifizierten Stelle im Modell wirklich ein Fehler vorliegt. Zusätzlich zu der automatischen Berechnung, können Metriken auch manuell erhoben werden. Bspw. bei der manuellen Prüfung des Modells auf *Semantische Richtigkeit* und *Interne Minimalität* können die Verstöße gegen das jeweilige Kriterium erfasst und klassifiziert werden und aus den Daten Metriken berechnet werden. Über die so erhobenen Maßzahlen kann ermittelt werden, welche Fehler bezüglich der Semantik wie oft auftreten. Aus dieser Fehlerauswertung können entsprechende konstruktive Maßnahmen für die Fehlervermeidung entwickelt werden. Da in dieser Arbeit automatische Qualitätssicherung mittels Prüfungen und Metrikberechnung untersucht wird, wurde diesem Aspekt nicht weiter nachgegangen.

4.1.4.3 ZWECKORIENTIERTE MODELLQUALITÄT

In der Softwareentwicklung soll jedes Modell einen oder mehrere Zwecke erfüllen (siehe Abschnitt 2.1). Im Teilbereich *Zweckorientierte Modellqualität* unseres Qualitätssystems werden alle Qualitätskriterien bezüglich der Modellqualität aufgeführt, die durch den Zweck der Modellierung entstehen oder abhängig von diesem definiert werden. An diese Stelle werden somit alle projektspezifischen Qualitätskriterien oder jene, die durch bestimmte für das jeweilige Projekt vorgeschriebene Standards oder aus Vorgehensmodellen entstehen, verortet. Bspw. sind hier Teile der in Abschnitt 3.4.2.1 vorgestellte Konvention eCH-0158 unter den Kriterien *Angemessene Modellierungssprache und -wortschatz*, *Angemessene Modellierungssyntax* und *Angemessene Aussagekraft* wiederzufinden. Eine Übersicht der in diesem Rahmen formulierten neun Qualitätskriterien ist in Abb. 4-6 (S. 69) zu sehen. Die Qualitätskriterien der *Zweckorientierten Modellqualität* lassen sich in drei Hauptgruppen aufteilen: Qualitätskriterien, die Aussagen

- zu zweckorientierten Struktur und Aufbau,
- zur zweckorientierten Syntax oder
- zum zweckorientierten Inhalt bzw. zur Semantik

des Modells oder von Modellbestandteilen treffen.

QUALITÄTSKRITERIEN ZU ZWECKORIENTIERTER STRUKTUR UND AUFBAU

Angemessener Detaillierungsgrad, *Angemessene Komplexität* und *Angemessene Ausgeglichenheit der Struktur* sind die Kriterien, die bezogen auf den Zweck Aussagen über die Güte des Aufbaus und der Struktur des Modells treffen (siehe Abb. 4-7, S. 70).

Unter dem Qualitätskriterium *Angemessener Detaillierungsgrad* betrachten wir die Granularität (bzw. den Detaillierungsgrad) des Modelles. Damit ist der Unterschied zwischen bspw. der Modellierung von Prozessen auf nur der obersten Ebene (etwa Geschäftsprozesse) versus der detaillierten Betrachtung (z.B. jeder einzelnen Eingabe im Dialogfenster) gemeint. Ein Modell sollte nur soweit verfeinert werden, wie eine weitere Detaillierung auch einen Nutzen für den Zweck der SRS hat. Zudem sollten die Kosten für die Detaillierung und Pflege dieses Teilmodells nicht höher sein als der hinzugewonnene Nutzen. Alternativ kann für *SRS-Modelle* ein gewünschter Modellierungsgrad vorgegeben werden, d

er mit möglichst geringem Aufwand erreicht werden soll. Der gewünschte, optimale oder auch wirtschaftlich sinnvolle Detaillierungsgrad, bis zu dem modelliert werden soll, ist jeweils projektspezifisch zu definieren. Die Einhaltung dieser Detaillierung muss nach unserem Erkenntnisstand **manuell** überprüft werden. **Automatisch** kann das Modell nur auf Indizien für eine zu hohe oder geringe Detaillierung untersucht werden bspw. können hierfür Metriken des Qualitätskriteriums *Ausgeglichenheit der Struktur* herangezogen werden.

Als Unterpunkt der zweckorientierten Modellqualität, hat die *Angemessene Komplexität* zum Ziel, dass das Modell leicht änderbar, leicht überschaubar und für seine Adressaten verständlich, sowie der Aufwand für die Einarbeitung in einen Teilbereich des Modells relativ gering ist: Mit leichter Änderbarkeit ist gemeint, dass eine Änderung eines Modellbestandteils nicht Auswirkungen an vielen Stellen im Modell zur Folge hat (weil es viele Abhängigkeiten zwischen den Modellbestandteilen gibt, d.h. eine hohe Komplexität), sondern sich nur auf einen kleinen eingrenzbaren Bereich auswirkt. Die leichte Überschaubarkeit und Verständlichkeit ergibt sich daraus, dass das Modell nicht komplex ist. Da die Teilmodelle bzw. -bereiche des Modells bei einer geringen Komplexität voneinander relativ unabhängig sind, ist der Aufwand für die Einarbeitung in einen Teilbereich des Modells hier geringer als bei einem Modell mit einer hohen Komplexität, wo sich der Leser das gesamte Modell erschließen muss. Ein SRS-Modell mit einer geringen Komplexität hat zur Folge, dass bspw. ein Systemarchitekt sich nur in einen klar eingrenzbaren Teilbereich der SRS einarbeiten und sich nicht den Inhalt eines evtl. 500-seitigen SRS erarbeiten muss, weil alle Inhalte untrennbar miteinander verknüpft sind. Unter dem Begriff Komplexität soll im Zusammenhang mit dem SRS-Modell untersucht werden wie viele und wie stark die Beziehungen und Abhängigkeiten zwischen den Bestandteilen des Modells sind. Ziel sollte es sein die Komplexität **automatisch** zu bestimmen und im Zuge dessen dem Modellierer komplexitätsmindernde Maßnahmen für das Modell vorzuschlagen. Da die automatische Messung der Komplexität des Modells jedoch nicht trivial ist, muss das Verringern der Komplexität vorerst **manuell** durch konstruktive und analytische Maßnahmen vorgenommen werden.

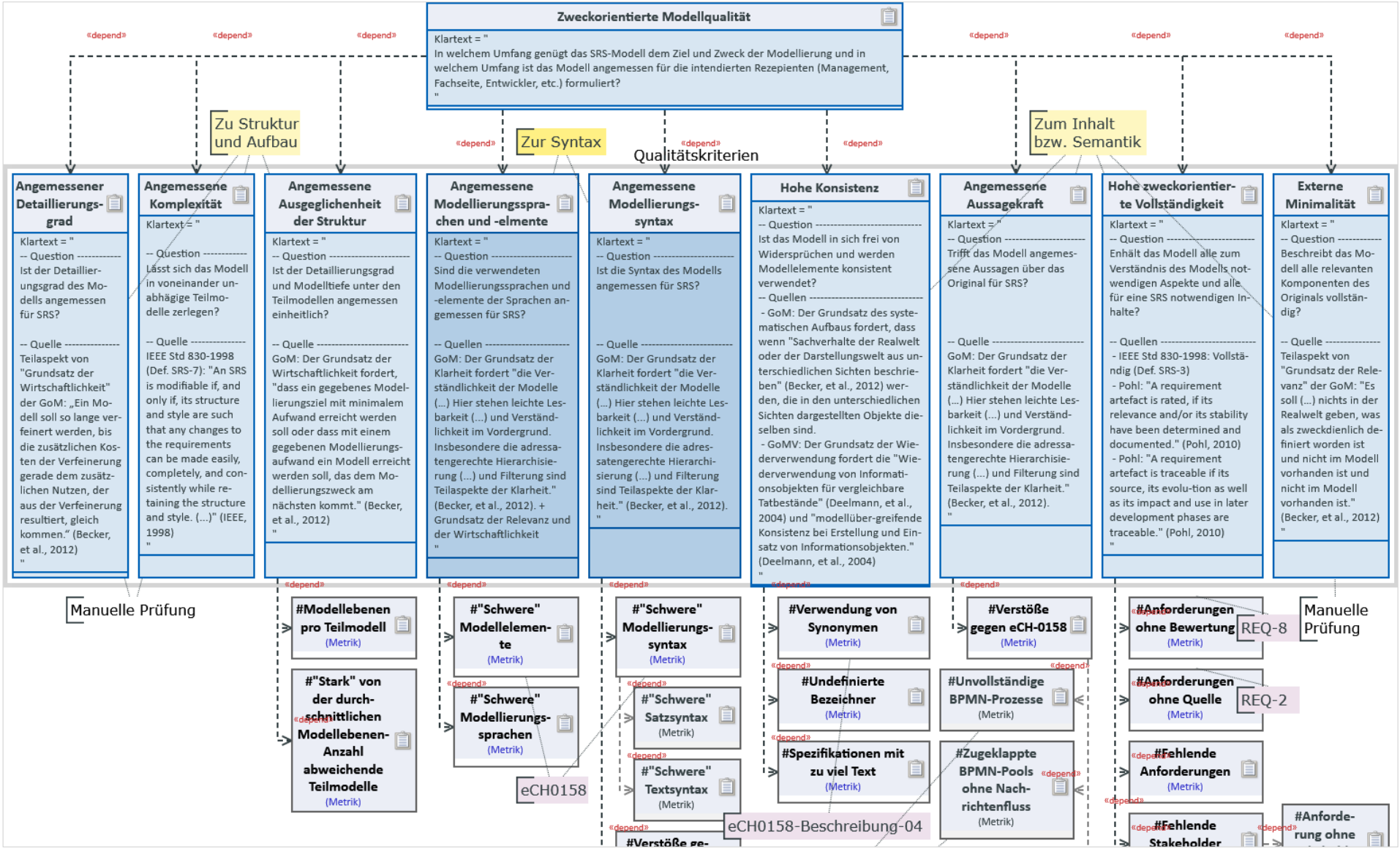


Abb. 4-6: Qualitätssystem Ebenen 2–n „Zweckorientierte Modellqualität“.

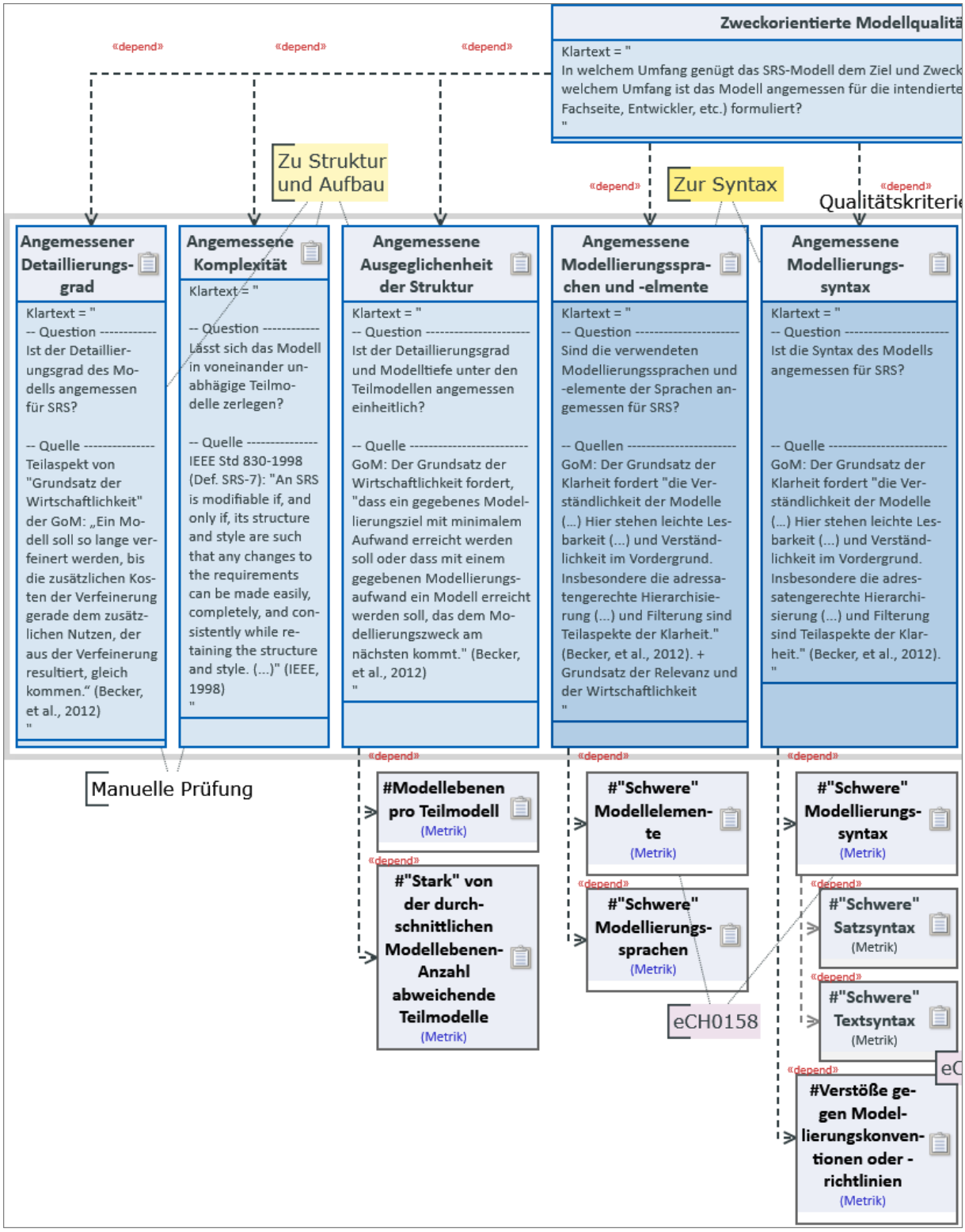


Abb. 4-7: Qualitätssystem Ebenen 2–n „Zweckorientierte Modellqualität“ (1/2).

Unter dem Punkt *Angemessene Ausgeglichenheit der Struktur* wird die Einheitlichkeit des Detaillierungsgrades der Teilmodelle betrachtet. Nach dem Grundsatz der Wirtschaftlichkeit soll mit einer vorgegebenen Zeit ein Modell erstellt werden, welches den Zweck am besten erfüllt. Somit kann das Ziel formuliert, werden im Modell stets eine konsistente Detaillierung und Strukturierung zu erreichen. Als **automatische** Prüfung können hier die Modellebenen (d.h. vertikale Abstraktionsebenen) gezählt („#Modellebenen“) und miteinander verglichen werden. Daraus kann ermittelt werden, wie viele Teilmodelle um bspw. ± 2 Ebenen von der durchschnittlichen Anzahl der Modellebenen abweichen („#“Stark“ von der durchschnittlichen Modellebenen-Anzahl abweichende Teilmodelle“). Das starke Abweichen der Anzahl der Modellebenen eines Teilmodells kann ein Indiz dafür sein, dass dieses Teilmodell unvollständig oder zu detailliert ist (vgl. *Angemessener Detaillierungsgrad*).

QUALITÄTSKRITERIEN ZU ZWECKORIENTIERTER SYNTAX

Zur zweckorientierten Syntax formulieren wir die Qualitätskriterien *Angemessene Modellierungssprachen und -elemente* sowie *Angemessene Modellierungssyntax* (siehe Abb. 4-7, S. 70). An dieser Stelle werden erneut Qualitätskriterien für die Syntax formuliert (vgl. *Syntaktische Modellqualität*, Abschnitt 4.1.4.1). Diese Kriterien unterscheiden sich zu den in Abschnitt 4.1.4.1 aufgestellten. Dort wird die Einhaltung der Syntaxregeln untersucht, welche durch Modellierungssprachen vorgegeben werden. Hier soll im Gegensatz dazu das Einhalten projekt- oder zweckspezifischer Syntaxregeln wie z.B. Modellierungskonventionen überprüft werden. Solche Regeln versuchen durch die zusätzliche Einschränkung der sprachspezifischen Modellierungsregeln bspw. die Verständlichkeit zu erhöhen oder die Komplexität zu verringern (vgl. eCH-0158 in Abschnitt 3.4.2.1). Modellierungskonventionen werden zweck- oder projektgebunden formuliert und daher diesem Abschnitt zugeordnet.

Bei dem Qualitätskriterium *Angemessene Modellierungssprachen und -elemente* geht es darum, inwiefern die verwendeten Modellierungssprachen und die verwendeten Elemente der jeweiligen Sprache für eine SRS angemessen sind. Eine SRS hat einen breiten Adressatenkreis (vom Informatiker bis zum Manager), wodurch die Schwierigkeit bei der Formulierung der SRS besteht, eine angemessene und für alle Parteien verständliche und eindeutige Sprache (sowohl Modellierungs- als auch textuelle Sprache) und einen eindeutigen Wortschatz zu wählen. Zu der Angemessenheit des Modellierungswortschatzes schränkt bspw. die eCH-0158 die BPMN-Modellelemente ein (z.B. sind hier Eskalations-, Fehler-, Abbruchereignisse etc. nicht zugelassen). Wir schlagen vor, die Verwendung von nicht zugelassenen oder der Erfahrung nach als schwer verständlich identifizierten Wörtern zu zählen („#“Schwere“ Modellelemente“) und daraus eine Art Komplexität der Verständlichkeit des Modells zu ermitteln, ähnlich wie ein Lesbarkeitsindex. Ebenso wie „schwere“ Wörter gezählt werden sollen, stellen wir zur Diskussion, Modellierungssprachen in „einfache“ und „schwere“ Sprachen aufzuteilen und die Anzahl der in einem Modell verwendeten „schweren“ Sprachen („#“Schwere“ Modellierungssprachen“) zu messen um bspw. die „Schwierigkeit“ oder die Höhe des Anspruch der SRS an den Leser abzuleiten. Denkbar ist, im Rahmen eines Softwareprojekts festzulegen, in welchen Teilbereichen des SRS-Modells und in welchem

Ausmaß (durch die Definition von Schwellwerten) „schwere“ Sprachen oder „schwerer“ Wortschatz verwendet werden darf. Sinnvoll kann es sein, dass in den Grundkapiteln wie dem ersten Kapitel „Ausgangssituation und Zielsetzung“ keine „schweren“ Wörter oder Sprachen erlaubt sind und erst auf tieferliegenden Ebenen oder bei bestimmten Kapiteln, die eine kleinere Gruppe adressieren, ein schwierigeres Sprachniveau erlaubt ist. Wie wir gezeigt haben, kann somit die *Angemessenheit der Modellierungssprachen und -element* **automatisch** gemessen und bewertet werden.

Das Kriterium *Angemessene Modellierungssyntax* ähnelt dem Qualitätskriterium *Angemessene Modellierungssprachen und -elemente* sehr. Hier wird anstelle der Sprache und des Wortschatzes untersucht, ob die verwendete Syntax des Modells für eine SRS angemessen ist. Bei vielen Sprachen (z.B. BPMN) können Inhalte auf unterschiedliche Arten modelliert werden. Dabei sind bestimmte syntaktische Konstruktionen besser oder einfacher verständlich als andere. Auch hierzu macht die Konvention eCH-0158 für die BPMN Einschränkungen (z.B. eCH0158-Aktivität-03[42]). Alternativ zum Zählen der Verstöße gegen die Konvention kann festgelegt werden, welche Sprachkonstruktionen als „schwer“ empfunden werden („#“Schwere“ Modellierungssyntax“). Wir haben hier die Metrik „#Verstöße gegen Modellierungskonventionen oder -richtlinien“ dem Qualitätskriterium *Angemessene Syntax* zugeordnet. Werden Modellierungskonventionen oder -richtlinien in einem Projekt vorgeschrieben, gehen wir davon aus, dass die von der Konvention etc. erlaubten Sprachkonstruktionen für alle Projektbeteiligten „einfacher“ verständlich und somit angemessener sind als jene die von dieser Norm abweichen. So kann auch die Angemessenheit der Modellierungssyntax **automatisch** überprüft, gemessen und anhand von Schwellwerten für Qualitätsniveaus bewertet werden.

QUALITÄTSKRITERIEN ZUR ZWECKORIENTIERTEN SEMANTIK

Im dritten Teilbereich der *Zweckorientierten Modellqualität*, der zweckorientierten Semantik, formulieren wir die folgenden vier Qualitätskriterien: *Hohe Konsistenz, Angemessene Aussagekraft, Hohe Zweckorientierte Vollständigkeit* und *Externe Minimalität* (siehe Abb. 4-8, S. 73). Die zweckorientierte Semantik unterscheidet sich von der in Abschnitt 4.1.4.2 bereits definierten *Semantischen Richtigkeit* dadurch, dass an dieser Stelle nicht überprüft wird, ob der Inhalt des Modells zum Original passt, sondern inwiefern der Inhalt, also die Bedeutung, den Zweck des SRS-Modells erfüllt.

Eine Qualitätsanforderung, die wir an unsere SRS stellen ist bspw. die konsistente Bezeichnung von Modellelementen oder Sachverhalten. Diesbezüglich wird unter *Hoher Konsistenz* in unserem Qualitätssystem die Frage gestellt: *„Ist das Modell in sich frei von Widersprüchen und werden Modellelemente konsistent verwendet?“* Eine Qualitätsbetrachtung ist hierbei, ob bspw. ein Element des Originals im Modell mehrfach als unterschiedliche Modellelemente auftaucht oder ob sich bestimmte Teilmodelle des Modells widersprechen. Die konsistente Verwendung von Wörtern im SRS-Modell kann **automatisch** überprüft werden. Hierzu müssen, im

[42] Def. eCH0158-Aktivität-03: *„Auf eine Aktivität folgt immer genau ein Sequenzfluss.“* (eCH, 2013)

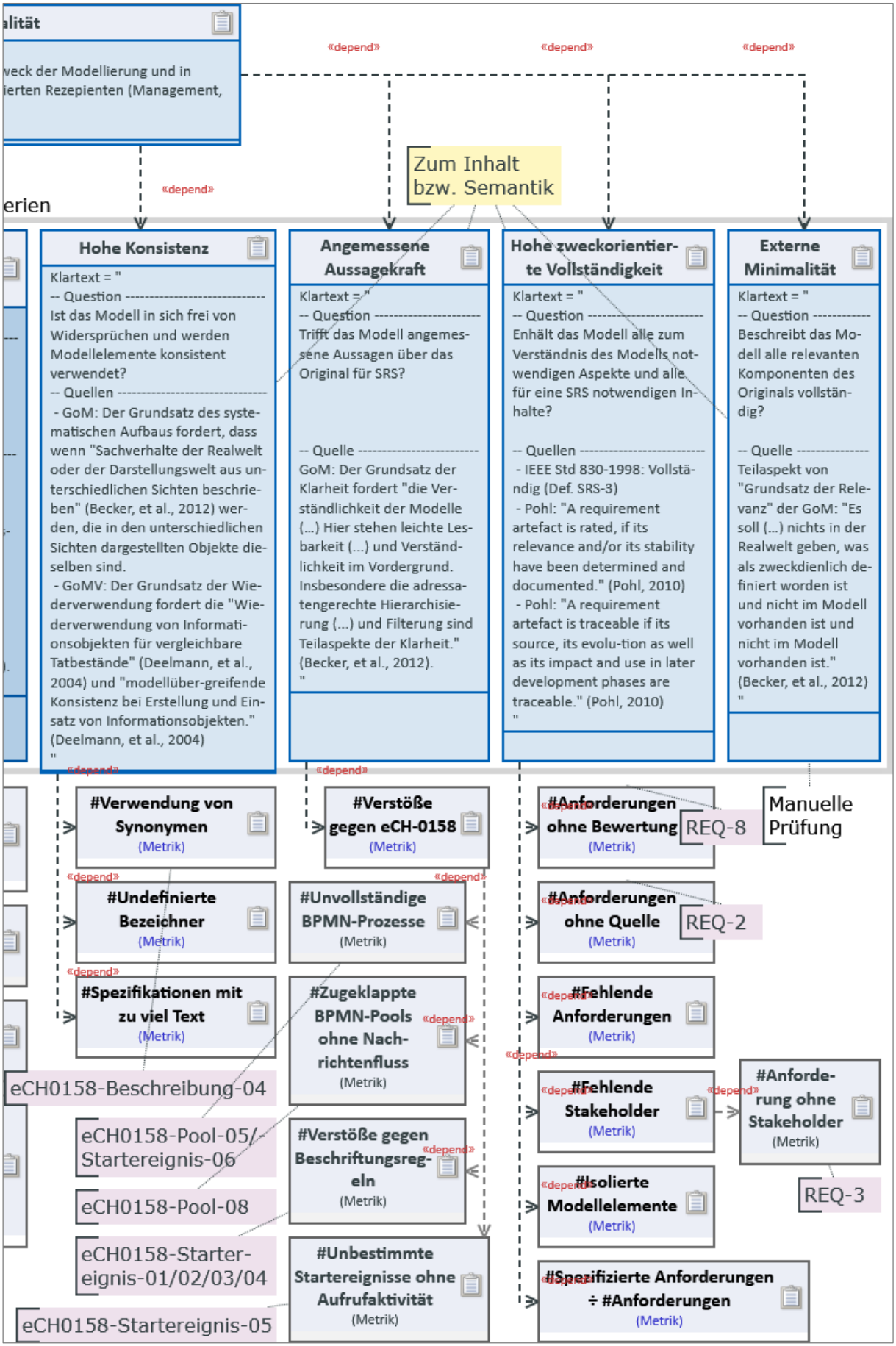


Abb. 4-8: Qualitätssystem Ebenen 2–n „Zweckorientierte Modellqualität“ (2/2).

Glossar des SRS-Modells zu den definierten Wörtern, die möglichen Synonyme des jeweiligen Wortes angegeben werden (vgl. Def. eCH0158-Beschreibung-04[43]). Synonyme dürfen im SRS-Modell nicht verwendet werden, denn es soll immer eindeutig sein um welches Modellelement es sich jeweils handelt, somit muss dieses konsistent und eindeutig benannt werden. Auf Grundlage der Synonymführung jedes Begriffs im Glossar kann das Modell automatisch auf die Verwendung dieser nicht erlaubten Wörter untersucht werden („#Verwendung von Synonymen"). Eine weitere Qualitätsanforderung, die an das Modell im Rahmen der *Hohen Konsistenz* gestellt werden kann, ist, dass jeder im Modell verwendete Bezeichner (z.B. Akteurnamen) im Glossar definiert sein muss („#Undefinierte Bezeichner"). Ein höheres, späteres Ziel für die Sicherstellung könnte eine Funktionalität sein, die die Definition von „erlaubtem" oder auch „verbotenem" Wortschatz bzw. Vokabular zur Verfügung stellt. Auf Grundlage dieses Wortschatzes könnte das Modell auf Einhaltung geprüft werden.

Unter dem Kriterium *Angemessene Aussagekraft* überprüfen wir, ob die im Modell getroffenen Aussagen und Beschreibungen über das Original angemessen für die SRS-Dokumentation und für die Phase der Anforderungsanalyse sind. Die SRS hat einen großen Adressatenkreis von der Fachseite übers Management bis zum Systementwickler. Da das SRS-Dokument zur Kommunikation der Ziele und Anforderungen der Parteien an das zu entwickelnde System dient, müssen die Inhalte entsprechend dokumentiert (d.h. modelliert) werden, sodass die Dokumentation für alle Parteien einigermaßen leicht lesbar und verständlich ist und die Inhalte dem Zweck der SRS entsprechen. Bspw. gehören technische Details wie z.B. die Behandlung von unzulässigen Eingaben des Mitarbeiters im Urlaubsantragsformular oder Informationen zur technischen Lösung nicht in das SRS-Modell. Auch das Auslassen von bestimmten Informationen oder unvollständige Modellbestandteile z.B. unvollständige Prozesse („#Unvollständige Prozesse") mindern die *Aussagekraft* des Modells. Gibt es für ein SRS-Modell definierte Regeln, die Aussagen dazu machen, wie etwas standardmäßig modelliert werden sollte, z.B. Regeln dafür, wie Anwendungsfälle oder BPMN-Startereignisse beschriftet werden sollen, so definieren diese die *Angemesse Aussagekraft*. Diese Regeln können manuell und teilweise **automatisch** überprüft werden (z.B. „#Verstöße gegen Beschriftungsregeln").

Bei der *Zweckorientierten Vollständigkeit* wird kontrolliert, *ob alle zum Verständnis des Modells notwendigen Aspekte und alle für eine SRS notwendigen Inhalte im Modell enthalten sind*. Hierfür soll bspw. das Fehlen von Objekten, Beziehungen oder Attributen aufgezeigt werden. Für Anforderungen besteht z.B. die Forderung, dass die Quelle zu jeder Anforderung angegeben sein muss (vgl. *nachvollziehbar*

[43] Def. eCH0158-Beschreibung-04: *„Auch wenn die Beschreibungskonventionen je BPMN-Element in Kapitel 4 befolgt werden, so wird trotz der Verwendung von Glossaren nicht verhindert, dass unterschiedliche Wörter mit derselben oder ähnlicher Bedeutung (Synonyme) verwendet werden (beispielsweise statt „prüfen" die Verben „begutachten", „kontrollieren", „mustern", etc.). Dies kann zu Fehlinterpretationen oder unterschiedlichen Benennungen derselben Elemente führen (beispielsweise „Rechnung prüfen", „Rechnung kontrollieren"). Aus diesen Überlegungen empfiehlt es sich, einen Begriff zu bevorzugen, ein projekt- oder unternehmensspezifisches Glossar zu führen und zu pflegen (...)."* (eCH, 2013)

nach Def. REQ-2) sowie die Relevanz und / oder Stabilität der Anforderung (vgl. *bewertet* nach Def. REQ-8). Bei einem SRS-Modell kann die Einhaltung dieser Qualitätsanforderungen an Anforderung **automatisch** überprüft werden. Dies ist möglich, da der jeweiligen Anforderung über Eigenschaftsfelder diese Informationen hinterlegt sind (siehe 5.1). Eine Prüfung kann diese Felder auf Belegung prüfen („#Anforderungen ohne Quelle“, „#Anforderungen ohne Bewertung“, „#Fehlende Stakeholder“). Ist für bestimmte Modellarten festgelegt, dass diese in der SRS nur auftauchen dürfen, wenn eine Anforderung ihre Existenz berechtigt, kann daraus gefolgert werden, wie viele Anforderungen im Modell fehlen („#Fehlende Anforderungen“). Mit der Metrik „#Spezifizierte Anforderungen ÷ #Dokumentierte Anforderungen“ soll eine Maßzahl für den Fertigstellungsgrad der Gesamtheit der Anforderungen berechnet werden und somit eine Art „Vollständigkeitsindex“ der Anforderungen. Spezifizierte Anforderungen sind „abgeschlossene“ Anforderungen, d.h. Anforderungen, die alle an sich gestellte Qualitätsanforderungen erfüllen und vom Bearbeitungsstatus auf „freigegeben“ gesetzt sind, d.h., die nach Def. REQ-3 korrekt sind. Dokumentierte Anforderungen sind solche, die bereits im Modell erstellt wurden, aber noch in Bearbeitung sind (d.h. mit Label „in Arbeit“, „Entwurf“ oder „in Überprüfung“) oder bei denen noch verpflichtende Angaben fehlen, d.h., die gegen Qualitätsanforderungen verstoßen. Neben der Betrachtung, ob alle für Anforderungen notwendigen Angaben gemacht wurden, sollte auch überprüft werden, ob alle für SRS notwendigen Inhalte modelliert sind. Ein anderer Aspekt der Vollständigkeit, der hier auch untersucht werden soll, ist, inwiefern alle zum Verständnis des Modells notwendigen Inhalte vorhanden sind. Nicht alle Aspekte der zweckorientierten Vollständigkeit können automatisch überprüft werden. Gerade die Kontrolle, ob für das Verständnis des Modells alle notwendigen Inhalte im Modell hinterlegt sind, muss **manuell** überprüft werden.

Externe Minimalität stellt die Frage, *ob das Modell alle relevanten Komponenten vollständig beschreibt*. Dies ist ebenfalls ein Aspekt der Vollständigkeit, jedoch aus einem anderen Blickwinkel. Hier soll von dem Original aus überprüft werden, ob jeder dort als relevant identifizierten Sachverhalte im Modell repräsentiert ist. Diese Vollständigkeit kann unserem Wissen nach ausschließlich **manuell** überprüft werden, da es hier das Modell auf Grundlage des Originals kontrolliert werden muss. Besser als manuelle, analytische Prüfung sind **konstruktive** Ansätze, die das nicht erfassen von relevanten Inhalten verringern, z.B. das Arbeiten mit Checklisten.

4.1.5 Qualität der SRS-Modellvisualisierungen

Diagramme (bzw. *Modellvisualisierungen*) zeigen bestimmte Ausschnitte eines Modells. Sie sind ein zentraler Bestandteil der modellbasierten SRS, da sie durch Anschaulichkeit zur Verständlichkeit des Modells, der Modellinhalte und somit der SRS beitragen sollen. Die Qualität dieser *Modellvisualisierungen* haben Auswirkungen auf bspw. die Verständlichkeit und die Schnelligkeit, mit der ein Leser das Dargestellte erfassen kann. Bei der Betrachtung der *Qualität der SRS-Modellvisualisierungen* soll die Frage beantwortet werden *„In welchem Umfang erfüllen die Visualisierungen des SRS-Modells (d.h. die Diagramme) die Qualitätsanforderungen?“*.

Die *Qualität der Visualisierung* wird in diesem Qualitätssystem durch mehrere Qualitätskriterien definiert:

- *Niedrige Komplexität der Visualisierung*,
- *Hohe Konsistenz der Visualisierung*,
- *Hohe Metaphernnutzung*,
- *Sachlogisch richtige Visualisierung* und
- *Hohe Übersichtlichkeit der Visualisierung*.

Wobei die *Übersichtlichkeit der Visualisierung* von sich selbst und den übrigen Qualitätskriterien abhängig ist (siehe Abb. 4-9, S. 77).

Ist das Modell bzw. das modellierte Softwaresystem komplex, so sollen trotz dessen die einzelnen Sichten auf das Modell so gewählt werden, dass sie jeweils eine geringe Komplexität aufweisen bzw. begreifbare und verständliche Zusammenhänge darstellen (*Niedrige Komplexität der Visualisierung*). Komplexität in Bezug auf Diagramme wird zum einen als die Größe des in der Darstellung verwendeten Modellierungswortschatzes („#Verwendete Elementtypen“) und zum anderen als der Grad der Verschachtelung der Diagramminhalte („#Schachtelungstiefe“) verstanden. Als Beispiel für die Größe des Modellierungswortschatzes und der damit verbundenen Komplexität des Diagramms, hat ein BPMN-Prozess eine höhere Komplexität, werden zusätzlich zu exklusiven Oder-Gateways inklusive Oder-Gateways und Parallele-Gateways im Diagramm verwendet. Die Komplexität der Modellvisualisierung kann mittels der vorgeschlagenen und weiteren Metriken **automatisch** bestimmt und folglich auch automatisch bewertet werden, wenn Schwellwerte vorliegen.

Unter dem Punkt *Hohe Konsistenz der Visualisierung* betrachten wir inwiefern die Gestaltung des Diagramms einheitlich ist. Teilaspekte sind bspw. gleiche Linienstärke, Schriftarten oder gleich große Modellelemente wie z.B. einheitliche Größe von Tasks in einem BPMN-Diagramm oder von Anwendungsfällen in einem Anwendungsfalldiagramm. Ein anderer Punkt ist die Verwendung von Farben (vgl. eCH0158-Darstellung-03[44]). Einige dieser Punkte können bei Modellierungswerkzeugen über die **Konfiguration** festgelegt werden, sodass eine stets einheitliche Gestaltung durch das Werkzeug vorgegeben wird. Andere Punkte wie die einheitliche Gestaltung der Größe von Anwendungsfällen können im Modell **automatisch** überprüft werden.

Die Nutzung von Metaphern bei der Visualisierung ist gewünscht (*Hohe Metaphernnutzung*). Die Vorgehensmodelle „Wasserfallmodell“ oder auch „V-Modell“ sind Beispiele für die Verwendung von Metaphern bei der Visualisierung. Hier wird ein Prozessablauf anhand einer visuellen Metapher (dem Wasserfall oder dem Buchstaben „V“) strukturiert, indem die Prozesselemente diese Form bilden. Dies strukturiert das Diagramm und schafft einen großen Wiedererkennungswert. Auch kann der Betrachter sich so eine größere Anzahl an Objekten merken oder auch

[44] Def. eCH0158-Darstellung-03: *„Grundsätzlich sind Farben zu vermeiden. (…) Ausnahme: Wenn es der Lesbarkeit dient, können den Elementtypen (z.B. alle Startereignisse Grün) oder auch einzelnen Elementen (z.B. IKS-Aktivitäten Rot) nach einem klaren Farbkonzept zurückhaltend entsprechende Farben zugewiesen werden.“* (eCH, 2013)

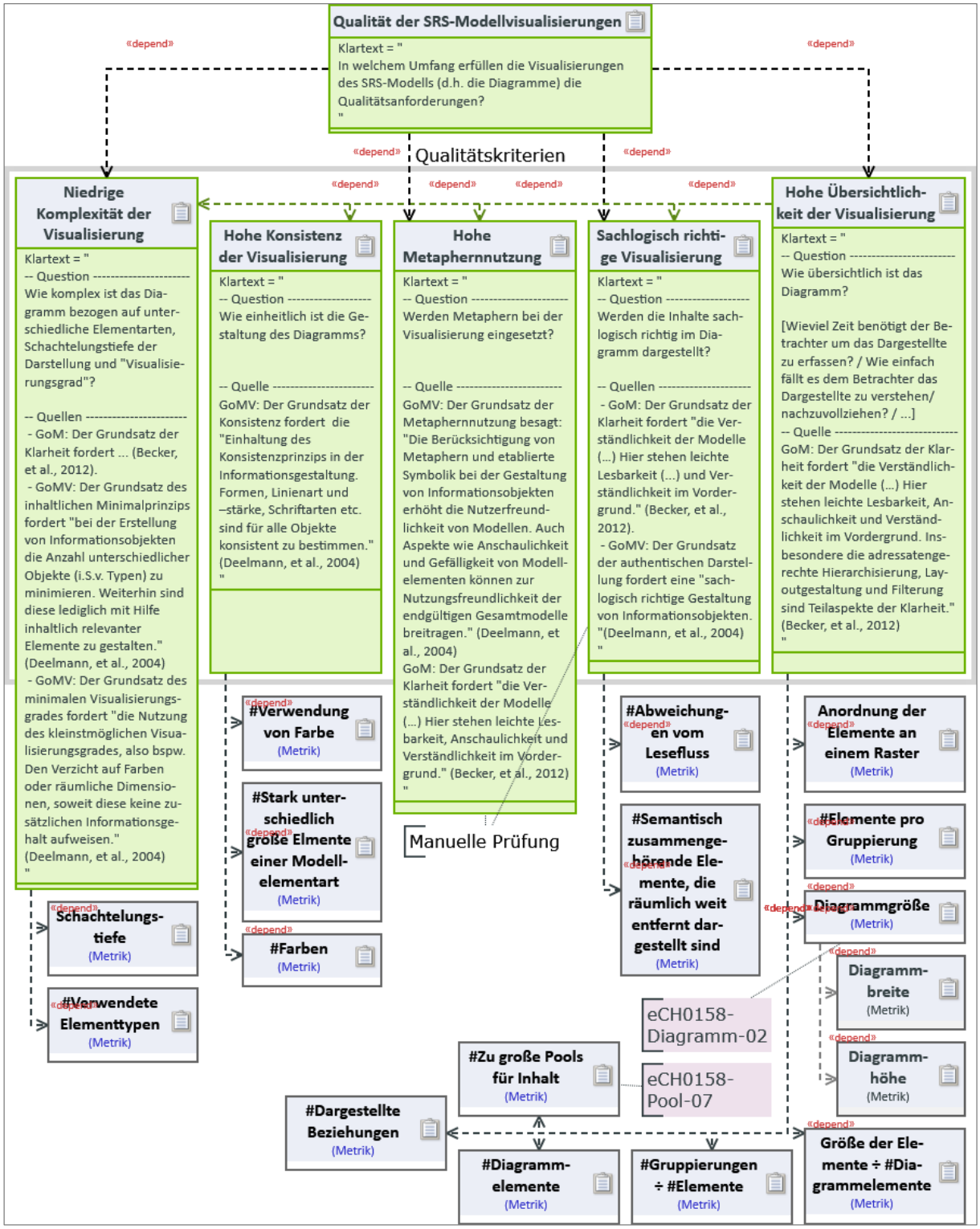


Abb. 4-9: Qualitätssystem Ebenen 1–n „*Qualität der SRS-Modellvisualisierung*“.

überschauen, da eine kleinere „Merk“-Einheiten so zu einer größeren Einheit zusammengefasst werden. Andere, simplerer Arten der Nutzung von Metaphern bei der Visualisierung sind folgende:

- Hierarchien oder Ableitungsbeziehungen von oben nach unten (z.B. unser Qualitätssystem, siehe z.B. Abb. 4-1, S. 50, in Abschnitt 4.1).
- Wichtigere Diagrammelemente weiter oben als unwichtige.
- Oben allgemeinerer Diagrammelemente, unten die speziellen (z.B. Klassendiagramm).
- Formen zur Darstellung nutzen: Baum, Kreis, Stern, Bus.
- Ablauf von links nach rechts (vgl. eCH0158-Darstellung-01[45]).

Der mögliche oder auch sinnvolle Einsatz von Metaphernnutzung kann nur **manuell** überprüft werden.

Das soeben vorgestellte Kriterium der Metaphernnutzung ist eng verknüpft mit dem nachfolgenden Qualitätskriterium der *Sachlogisch richtigen Visualisierung*. Das Kriterium der *Sachlogisch richtigen Visualisierung* zielt darauf ab, dass die Sachverhalte nach ihrer Logik aufgebaut dargestellt werden, d.h.: Werden in einem Prozess Prozessschritte zeitlich nacheinander ausgeführt, so sollen sie im Diagramm auch räumlich nacheinander dargestellt werden. Ein anderes Beispiel ist, inhaltlich zusammengehörende Elemente räumlich nah beieinander anzuordnen und umgekehrt inhaltlich unabhängige räumlich zu trennen oder in einzelnen Diagrammen darzustellen. Die sachlogisch richtige Visualisierung kann für bspw. die richtige Anordnung der Schritte eines Prozessdiagramms **automatisch** überprüft werden. Für jene „Logik“, die nicht durch Beziehungs- oder Ablaufelemente im Modell hinterlegt ist und somit nicht als automatisch überprüfbare Information vorliegt, muss **manuell** überprüft werden, ob die Visualisierungen auch diesbezüglich sachlogisch richtige Darstellungen sind.

Bei der Übersichtlichkeit (*Hohe Übersichtlichkeit der Visualisierung*) geht es darum, wie groß (d.h. räumliche „Größe“), umfangreich (d.h. inhaltliche „Größe“) und strukturiert das Diagramm ist. Dies hat Auswirkungen auf die Zeitspanne, die ein Betrachter benötigt um das Dargestellte zu erfassen und das Wahrgenommene im zweiten Schritt zu verstehen und nachzuvollziehen. Dabei werden die in Tab. 4-7 (S. 79) aufgeführten Eigenschaften betrachtet.

4.1.6 Weitere Qualitätsfaktoren

Nachfolgend werden die beiden Qualitätsfaktoren *Qualität der modellexternen SRS-Texte* und *des SRS-Dokumentationsgenerators* kurz vorgestellt. Sie beeinflussen ebenfalls die Qualität der modellbasierten SRS, stehen jedoch nicht im Fokus dieser Arbeit.

Qualität der modellexternen SRS-Texte

Die *modellexternen SRS-Texte* sind längere Fließtexte, die in einem anderen Werkzeug wie bspw. dem Texteditor Word verfasst werden. Sie beschreiben Hinter-

[45] Def. eCH0158-Darstellung-01: *„Entsprechend dem Sequenzfluss […] sind die Modelle von links nach rechts zu modellieren.“* (eCH, 2013)

grund- oder Zusatzinformationen zu dem zu entwickelnden System, welche nicht modelliert werden müssen oder können, wie z.B. die Vision des Softwaresystems oder der Lieferumfang. Für Texte gibt es Metriken die bspw. die „Komplexität" des Textes bewerten (z.B. fog-Index für englische Texte). Das Thema wird im Rahmen dieser Arbeit jedoch nicht näher betrachtet, da unser Fokus auf der Qualitätsbetrachtung des Modells und den Modellvisualisierungen liegt.

Eigenschaften	Fragestellungen	Metriken
Inhaltliche „Größe" (u.a. Detaillierung)	Wie viele Informationen werden dargestellt (z.B. Beziehungen im Anwendungsfalldiagramm)? Wie viele Informationen werden zu jedem Objekt dargestellt (z.B. Kommentare)?	▪ #Dargestellte Beziehungen
Räumliche „Größe"	Wie groß ist die Ausdehnung des Diagramms (z.B. größer als Din A4) und wie viele Modellbestandteile werden im Diagramm dargestellt?	▪ #Diagrammelemente ▪ Diagrammgröße ▪ Diagrammbreite ▪ Diagrammhöhe ▪ (Größe der Elemente ÷ #Diagrammelemente) ▪ #Zu große Pools für Inhalt
Strukturiertheit	Wie viele Strukturierungselemente (z.B. Gruppierung in BPMN) werden eingesetzt? Wie viele Elemente werden pro Strukturierungselement zusammengefasst?	▪ Anordnung der Elemente an einem Raster ▪ #Elemente pro Gruppierung ▪ Gruppierungen ÷ #Elemente

Tab. 4-7: Aspekte der „*Übersichtlichkeit der Visualisierung*"

QUALITÄT DES SRS-DOKUMENTATIONSGENERATORS

Die jeweilige Konfiguration (bezüglich der Eingabedaten und Ausgabeeinstellungen) des Dokumentationsgenerators ist verantwortlich dafür, welche Inhalte des Modells wie, wo und in welcher Reihenfolge im SRS-Dokument auftauchen. (Die Funktionsweise des Dokumentationsgenerators wird in Abschnitt 5.1.1 kurz erläutert.) Der Dokumentationsgenerator beeinflusst dabei die Qualität des SRS-Dokuments: Aspekte wie Leserlichkeit und Vollständigkeit des Dokuments werden durch ihn bestimmt.

Der Aspekt *Qualität des SRS-Dokumentationsgenerators* wird in dieser Arbeit nur angerissen, um ihn bei der Betrachtung der Qualität von modellbasierten SRS nicht zu übergehen. Die Dokumentationserstellung ist die Funktionalität in Innovator, die es dem Modellierer erlaubt „auf Knopfdruck" aus dem Modell und den Diagrammen eine Dokumentation zu erstellen (siehe Abschnitt 5.1.1). Diese „Dokumentationserstellung" ist nicht durch das Werkzeug vorgegeben, sondern meist kunden- und evtl. sogar projektspezifisch in der Ausgabeneinstellung der Dokumentationskonfiguration in der Konfiguration definiert. Im so generierten SRS-Dokument wird ein Status des SRS-Modells festgehalten. Dieses Dokument ist Bestandteil des Vertrages über die Erstellung der Softwarelösung zwischen Auftraggeber und -nehmer (siehe Abschnitt 2.3). Somit ist die Erstellung einer qualitativ hochwertigen Dokumentation aus dem hoffentlich hochwertigen Modell und Diagrammen durch den Dokumentationsgenerator von zentraler Bedeutung für das Softwareentwicklungsprojekt.

Dem Dokumentationsgenerator untergeordnet können Qualitätskriterien wie *Angemessenheit der Dokumentationsstruktur*, *Angemessenheit der Detaillierung der Dokumentation*, *Vollständigkeit der Modellabbildung* oder *Qualität der SRS-Dokumentationsinformationen* untersucht werden.

Mit der *Qualität des Dokumentationsgenerators* wurde der letzte Qualitätsfaktor unseres Qualitätssystems vorgestellt. Eine Übersicht des vollständigen Qualitätssystems für Anforderungsspezifikationen ist im Anhang zu finden (Abb. 4-1, S. 50, in Abschnitt 4.1).

4.1.7 Zusammenfassung

In diesem Teil der Arbeit haben wir ein Qualitätssystem für die Definition der *Qualität modellbasierte SRS* vorgestellt, bestehend aus vier Qualitätsfaktoren. Zwei dieser vier Faktoren wurden weiter heruntergebrochen auf siebzehn Qualitätskriterien, die direkt im Modell nachweisbar sind (vgl. Z1, Abschnitt 1.2). Diese Kriterien wurden anhand von Beispielen erläutert. Zudem wurden Vorschläge zur Umsetzung von Prüfungen des jeweiligen Kriteriums gemacht sowie herausgearbeitet, welche Kriterien ausschließlich manuell, teilweise manuell und teilweise automatisch oder komplett automatisch überprüft werden können (vgl. Z2). In diesem Rahmen sind einzelne Metriken für die metrikbasierte Bewertung der Qualität, welche in Abschnitt 4.2.4 erläutert wird, bereits vorgestellt worden.

4.2 Konzeptentwurf für die Applikation

Anschließend an die Definition des Qualitätssystems und den Vorschlägen zu Umsetzungsmöglichkeiten des jeweiligen Kriteriums im vorherigen Abschnitt der vorliegenden Arbeit, wird in diesem ein Entwurf für eine Applikation zur Qualitätsprüfung, -messung und -bewertung von *SRS-Modellen* und *-Modellvisualisierungen* präsentiert. Die Applikation soll mittels Qualitäts- und Quantitätsmetriken die Modellqualität (d.h. *Qualität des SRS-Modells* und *der SRS-Modellvisualisierungen*) bewerten. Die vorgeschlagenen Metriken basieren dabei oftmals auf Qualitätsprüfungen.

Die Vision und genaue Zielsetzung der Applikation wird in Abschnitt 4.2.1 erläutert. In 4.2.2 definieren wir das Bewertungsschema für die Prüfungen und spezifizieren automatische Prüfungen, welche die Einhaltung der eCH-0158-Konventionen für Pools und Startereignisse überprüfen. Die im Rahmen der Arbeit aufgestellten Metriken werden unter Punkt 4.2.3 definiert. Um eine Bewertung der Modellqualität vornehmen zu können, müssen für die Maßzahlen dieser Metriken Schwellwerte definiert werden. Diese ordnen der jeweiligen Maßzahl ein Qualitätsniveau bzw. -level zu. In Abschnitt 4.2.4 stellen wir eine Skala für die Bewertung vor und empfehlen ein Vorgehen zur Festlegung der erwähnten Schwellwerte sowie die Ableitung der Qualitätsbewertung aus den Einzelbewertungen.

4.2.1 Zielsetzung

Die Vision der zu entwerfenden Applikation ist, dass anhand definierter Qualitätslevel der Erreichungsgrad des Modells bezüglich des nächsten Levels ermittelt werden soll. Zudem soll eine Liste von Aufgaben ausgegeben werden, welche erledigt werden müssen, um das nächste Qualitätslevel zu erreichen. Die Level werden

vorab mittels Schwellwerten für die metrikbasierten Qualitätskriterien definiert. Es handelt sich bei der Applikation in erster Linie um einen analytischen, werkzeuggestützten Qualitätssicherungsansatz.

In den vorherigen Kapiteln wurde festgestellt, dass automatische Ansätze im Gegensatz zu manuellen Ansätzen weniger Zeit und Kosten verursachen. Zudem sind sie objektiv, d.h. das Ergebnis der Prüfung, Messung oder Bewertung ist unabhängig vom Durchführenden sowie auch der Erfolg der jeweiligen Maßnahme von ihm unabhängig ist (siehe Abschnitt 3.2.3). Aus diesen Gründen ist das Ziel der Applikation, Qualität automatisch zu prüfen, zu messen und zu bewerten.

Für die Modellqualitätsbewertung sollen die in Abschnitt 4.1 vorgestellten automatisch überprüfbaren Aspekte des *SRS-Modells* und der *SRS-Modellvisualisierungen* mittels Metriken gemessen und bewertet werden. Die Gegenstände der Messung sind im ersten Anlauf beschränkt auf die in Tab. 4-8 gelisteten und im Rahmen dieser Arbeit genauer behandelten Messobjekte, ihre Eigenschaften sowie ihre Beziehungen untereinander.

Messobjekte	**Modellierungssprachen**
Anforderungen	*Innovator spezifisch*
Anwendungsfälle	*UML*
Pools, Startereignisse	*BPMN*

Tab. 4-8: Messobjekte für die Qualitätsmessung und -bewertung.

4.2.2 PRÜFUNGEN

Um die Qualität der Einhaltung bestimmter Regeln oder Qualitätseigenschaften zu bewerten, muss zunächst über Prüfungen ermittelt werden, ob das Modell gegen diese verstößt. Dabei unterscheiden wir zwischen unterschiedlichen Arten bzw. unterschiedlicher Schwere eines Verstoßes durch Bewertungskategorien. Das aktuelle Bewertungsschema des Modellierungswerkzeugs Innovator sieht die drei Kategorien *Fehler*, *Warnung* und *Information* vor. Dieses Schema wird von uns übernommen. Über die Kategorien wird in diesem Rahmen der in Tab. 4-9 (S. 82) aufgeführte Informationsgehalt übermittelt.

Im Rahmen dieser Arbeit haben wir uns für die Konventionen zweier Teilbereiche der eCH-0158 (BPMN-Pools und -Startereignisse, siehe Abschnitt 3.4.2.1) überlegt, ob und wie diese durch automatische Prüfungen in Innovator umgesetzt werden können. Aus den fünfzehn Konventionen zu Pools und Startereignissen haben wir elf identifiziert, die teilweise oder vollständig automatisch überprüft werden können (siehe Tab. 8-1 und Tab. 8-2, in Abschnitt 8.5). Neun dieser Konventionen können direkt für die Standard-Konfiguration als automatische Prüfung umgesetzt werden. Aus diesen neun haben wir vierzehn separate Prüfungen abgeleitet. Auf den folgenden Seiten werden diese spezifiziert, beschrieben und nach der Schwierigkeit der Umsetzung bewertet (Tab. 4-10 bis Tab. 4-13).

Weitere zwei theoretisch automatisch überprüfbare Konventionen werden in den Tabellen Tab. 4-10 bis Tab. 4-13 nicht aufgeführt, da für diese Konventionen vorab Konfigurationen des Innovator-Profils vorgenommen werden müssen, bspw. für

eCH0158-Pool-03[46] muss ein Label für Pool (bzw. Prozess in Innovator) eingeführt werden, welches definiert, ob es sich bei dem Pool um einen internen oder externen Pool handelt. Erst aufbauend auf dieser Information kann automatisch ermittelt werden, ob alle internen und alle externen Pools jeweils bspw. direkt übereinander angeordnet sind oder die gleiche Farbgebung haben.

Kategorien	Bedeutung
Fehler	Der Prüfling ist falsch oder fehlerhaft (z.B. verstößt gegen eine Regel).
Warnung	Der Prüfling geht über einen Schwellwert hinaus (z.B. ist zu detailliert[47]), verstößt gegen die Konventionen bzw. Richtlinien oder liefert ein Indiz für einen *Fehler*.
Information	Der Prüfling liefert ein Indiz für eine *Warnung* oder hält eine Modellierungsempfehlung nicht ein (z.B. eCH0158-Beschreibung-06[48]).

Tab. 4-9: Kategorien für die Bewertung.

Die übrigen Konventionen des eCH-0158, welche nicht mit automatischen Prüfungen umgesetzt werden können, werden teilweise über die Konfiguration (siehe 5.1.2) umgesetzt und somit „automatisch" konstruktiv gesichert. Jene Richtlinien, die weder konfiguriert noch automatisch überprüft werden können, sollen dem Modellierer als Richtlinienkatalog zur manuellen Überprüfung und Sicherstellung an die Hand gegeben werden. Durch die Konfiguration und Modellprüfung kann die große Anzahl der anfangs 65 auf ca. 24 manuell zu überprüfende Konventionen reduziert werden.

[46] *„Reihenfolge oder Farbgebung kann der Identifizierung interner / externer Pools dienen."* (eCH, 2013)

[47] Frühzeitige, unnötige Einschränkungen können höhere Kosten verursachen, da durch sie unnötige Vorgaben oder Einschränkungen gemacht werden, die Entscheidungsalternativen in späteren Phasen der Entwicklung eingrenzen; Durch zu viel Modellierung, können sich die Aufwandskosten sowie die benötigte Zeit für die Anforderungsanalyse erhöhen.

[48] *„Beschriftungen und Bezeichnungen von Elementen sollen nach Möglichkeit keine Formatierungen (Kursiv, Fett, etc.) […] enthalten."* (eCH, 2013)

Konventions-Nr.	Konventionsbeschreibung	Name der Prüfung (Java-Klasse)	Fehler	Warnung	Information	Eingabe (API-Name)	Ausagabe	Beschreibung	SdU[49]	Status[50]
eCH0158-Pool-05	*„In jedem aufgeklappten Pool wird genau ein vollständiger Prozess modelliert.“*	**Startereignis vorhanden** (Check_BPMN_BPProcess_HasStartEventNode)		X		BPMN-Prozess (BPProcess)	faulty, wenn der Prozess kein Startereignis enthält, sonst ok.	Prüft ob mindestens ein Startereignis pro Prozess vorhanden ist.	3	2[51]
eCH0158-Pool-05	*„In jedem aufgeklappten Pool wird genau ein vollständiger Prozess modelliert.“*	**Endereignis vorhanden** (Check_BPMN_BPProcess_HasEndEventNode)		X		BPMN-Prozess (BPProcess)	faulty, wenn der Prozess kein Endereignis enthält, sonst ok.	Prüft ob mindestens ein Endereignis pro Prozess vorhanden ist.	3	1
eCH0158-Pool-05	*„In jedem aufgeklappten Pool wird genau ein vollständiger Prozess modelliert.“*	**Nur genau ein Prozess** (Check_BPMN_BPProcess_OnlyOneProcessPerPool)		X		BPMN-Prozess (BPProcess)	faulty, wenn mehr als ein selbständiger Prozess in einem Pool enthalten sind, sonst ok.	Prüft ob nur ein selbständiger Prozess im Pool enthalten ist.	5	1
eCH0158-Pool-05	*„In jedem aufgeklappten Pool wird genau ein vollständiger Prozess modelliert.“*	**Prozesselement ist über Sequenzfluss mit Startereignis verbunden** (Check_BPMN_BPFlowElement_IsConnectedWithStartEventNode)		X		BPMN-Flusselement (BPFlowElement)	faulty, wenn das Element nicht über Sequenzflüsse mit mindestens einem Startereignis verbunden ist, sonst ok.	Prüft ob das Element mit mindestens einem Startereignis über den Sequenzfluss verbunden ist.	4	1
eCH0158-Pool-05	*„In jedem aufgeklappten Pool wird genau ein vollständiger Prozess modelliert.“*	**Prozesselement ist über Sequenzfluss mit Endereignis verbunden** (Check_BPMN_BPFlowElement_IsConnectedWithEndEventNode)		X		BPMN-Flusselement (BPFlowElement)	faulty, wenn das Element nicht über Sequenzflüsse mit mindestens einem Endereignis verbunden ist, sonst ok.	Prüft ob das Element mit mindestens einem Endereignis über den Sequenzfluss verbunden ist.	4	1

Tab. 4-10: Spezifikationen für die Prüfungen der eCH-0158-Konventionen (1/4).

[49] SdU: Schwierigkeit der Umsetzung (1: Umsetzung bekannt, einfach; 2: Umsetzung bekannt, schwierig; 3: Umsetzung unbekannt, vermutlich einfach; 4: Umsetzung unbekannt, vermutlich schwierig; 5: Umsetzung unbekannt, vermutlich sehr schwierig; 0: Nicht einschätzbar.)

[50] 1: Vollständig spezifiziert; 2: Umgesetzt

[51] Siehe Abb. 8-32, S. 140, im Anhang (Implementierung: Siehe Abb. 8-1, S. 163, im Anhang).

Konventions-Nr.	Konventionsbeschreibung	Name der Prüfung (Java-Klasse)	Fehler	Warnung	Information	Eingabe (API-Name)	Ausgabe	Beschreibung	SdU[49]	Status[50]
eCH0158-Pool-05	*„In jedem aufgeklappten Pool wird genau ein vollständiger Prozess modelliert."*	**Endereignis ist über Sequenzfluss mit Startereignis verbunden** (Check_BPMN_BPEventNode_EndEventNodeIsConnectedWithStartEventNode)		X		BPMN-Ereignis (BPEventNode)	faulty, wenn Endereignis nicht mit mindestens einem Startereignis über Sequenzfluss verbunden, sonst ok.	Prüft ob Endereignis mit mindestens einem Startereignis verbunden ist.	4	2[52]
eCH0158-Pool-07	*„Die Höhe des aufgeklappten Pools richtet sich nach dessen Inhalt."*	**Poolgröße angemessen zu Inhalt** (Check_BPMN_BPDiaProcessViewNode_IsAdequateToTheContent)			X	BPMN-Prozess (BPDiaProcessViewNode)	faulty, wenn die Pool-Ausmaße größer als ± 2 cm der maximalen oder minimalen X- oder Y-Koordinate eines beinhalteten Elements sind, sonst ok.	Prüft ob der Pool entsprechend zum Inhalt nicht zu groß gewählt ist.	5	1
eCH0158-Pool-08	*„Zugeklappte Pools enthalten mindestens einen eingehenden oder ausgehenden Nachrichtenfluss."*	**Zugeklappter Beteiligter-Kollaboration hat mindestens einem Nachrichtenfluss** (Check_BPMN_BPParticipant_HasBPMessageFlow)		X		BPMN-Pool (BPParticipant)	faulty, wenn dem zugeklappten Kollaborationsbeteiligten kein ein- oder ausgehender Nachrichtenfluss zugeordnet ist, sonst ok.	Prüft ob der zugeklappte Kollaborationsbeteiligter mindestens ein Nachrichtenfluss hat.	3	1
eCH0158-Startereignis-01	Unbestimmtes Startereignis: *„Wird nicht beschrieben, der Unterprozess wird durch den Aufruf auf der übergeordneten Prozessebene gestartet."*	**Unbestimmtes Startereignis ohne Beschreibung und ohne Definition der Blankodefinition** (Check_BPMN_PBEventNode_UnspecifiedStartEventNodeHasNoName)		X		Ereignis (BPEventNode)	faulty, wenn unbestimmtes Startereignis eine Ereignisdefinition hat oder eine Bezeichnung angegeben ist, sonst ok.	Prüft ob für das unbestimmte Startereignis keine Beschriftung angegeben ist.	2	2[53]

Tab. 4-11: Spezifikationen für die Prüfungen der eCH-0158-Konventionen (2/4).

[52] Siehe Abb. 8-33, S. 184, im Anhang (Implementierung: Siehe Abb. 8-2 und Abb. 8-3, S. 160-161, im Anhang).

[53] Siehe Abb. 8-28, S. 138, im Anhang (Implementierung: Siehe Abb. 8-4, S. 159, im Anhang).

Konventions-Nr.	Konventionsbeschreibung	Name der Prüfung (Java-Klasse)	Fehler	Warnung	Information	Eingabe (API-Name)	Ausagabe	Beschreibung	SdU[49]	Status[50]
eCH0158-Startereignis-02	Nachrichten-Startereignis: *„Wird nicht beschrieben, wenn die eingehende Nachricht auf dem (obligatorischen) Nachrichtenfluss ersichtlich ist.“*	**Nachrichten-Startereignis ohne Beschreibung und ohne Definition der Nachrichtendefinition** (Check_BPMN_PBEventNode_MessageStartEventNodeHasNoNName)			X	Ereignis (BPEvent Node)	faulty, wenn Nachrichten-Startereignis eine Ereignisdefinition hat und für den zugehörigen Nachrichtenfluss eine Nachricht angegeben ist, sonst ok.	Prüft ob beim Nachrichten-Startereignis die Nichtangabe einer Nachrichtendefinition erlaubt ist.	4	1
eCH0158-Startereignis-03	Bedingungs-Startereignis: *„Bedingung ist in der Bezeichnung des Elementes festzuhalten. Sollte ein Bedingungs-Startereignis einem Endereignis eines anderen Prozesses entsprechen, ist es identisch zu bezeichnen.“*	**Bedingtes Startereignis mit Definition der Bedingungsdefinition** (Check_BPMN_PBEventNode_ConditionalStartEventNodeHasDefinition)		X		Ereignis (BPEvent Node)	faulty, wenn in der Bedingungs-Ereignisdefinition keine oder die Default-Definition angegeben ist, sonst ok.	Prüft ob für das bedingte Startereignis eine Definition in der Ereignisdefinition angegeben ist.	1	2[54]
eCH0158-Startereignis-04	Zeitgeber- Startereignis: *„Bezeichnung enthält den Zeitpunkt für den Start. Beispiele: „am 1. je Monat“, „09:00 Uhr“.“*	**Zeitliches Startereignis mit Definition der Zeitgeberdefinition** (Check_BPMN_PBEventNode_TimerStartEventNodeHasDefinition)		X		Ereignis (BPEvent Node)	faulty, wenn in der Zeitgeber-Ereignisdefinition keine oder die Default-Definition angegeben ist, sonst ok.	Prüft ob für das zeitliche Startereignis eine Definition in der Ereignisdefinition angegeben ist.	1	2[55]

Tab. 4-12: Spezifikationen für die Prüfungen der eCH-0158-Konventionen (3/4).

[54] Siehe Abb. 8-29, S. 139, im Anhang (Implementierung: Siehe Abb. 8-5, S. 164, im Anhang).

[55] Siehe Abb. 8-30, S. 140, im Anhang (Implementierung: Siehe Abb. 8-6, S. 165, im Anhang).

Konventions-Nr.	Konventionsbeschreibung	Name der Prüfung (Java-Klasse)	Information	Warnung	Fehler	Eingabe (API-Name)	Ausagabe	Beschreibung	SdU[49]	Status[50]
eCH0158-Startereignis-05	*„Unbestimmte Startereignisse werden nur verwendet, wenn der Prozess durch den Aufruf auf der übergeordneten Prozessebene gestartet wird.“*	**Unbestimmtes Startereignis nur, wenn Prozess eine Aufrufaktivität hat** (Check_BPMN_PBEventNode_UnsepecifiedStartEventNodeHasCallingActivity)		X		Ereignis (BPEven Node)	faulty, wenn der zugehörige Prozess nicht durch eine Aufrufaktivität aufgerufen wird, sonst ok.	Prüft ob der zugehörige Prozess eines unbestimmten Startereignisses durch eine Aufrufaktivität verwendet wird.	2	2[56]
eCH0158-Startereignis-06	*„Ein Prozess hat mindestens ein Startereignis.“*	**Startereignis vorhanden** (Check_BPMN_BPProcess_HasStartEventNode)		X		BPMN-Prozess (BPProcess)	faulty, wenn der Prozess kein Startereignis enthält, sonst ok.	Prüft ob mindestens ein Startereignis pro Prozess vorhanden ist.	3	2[57]

Tab. 4-13: Spezifikationen für die Prüfungen der eCH-0158-Konventionen (4/4).

[56] Siehe Abb. 8-28, S. 138, im Anhang (Implementierung: Siehe Abb. 8-7, S. 158, im Anhang).

[57] Siehe Abb. 8-32, S. 141, im Anhang (Implementierung: Siehe Abb. 8-1, S. 163, im Anhang).

4.2.3 MESSUNGEN

Die im Rahmen dieser Arbeit vorgeschlagenen Metriken (siehe in grau in Abb. 4-10, S. 88) für die Messung und anschließende Bewertung von *SRS-Modellen* und -*Modellvisualisierungen* werden in den Tabellen Tab. 4-14 bis Tab. 4-20 aufgelistet. Zu der jeweiligen Metrik werden in der Tabelle die Bewertungskategorie, die Berechnung, die Definition und die Art der Metrik angegeben. Zusätzlich ist zu jeder Metrik vermerkt, ob sie sprachenspezifisch oder allgemein ist. So haben wir insgesamt einen Katalog von ca. 50 Metriken entworfen.

Die als Qualitätsmetriken gekennzeichneten Metriken messen Qualitätsdefizite direkt, z.B. „#Verstöße gegen Wortsyntax“. Jeder weitere Verstoß mindert die Qualität des Modells zusätzlich. Die Quantitätsmetriken hingegen drücken die Größe bestimmter Eigenschaften des Modells in Zahlen aus. Daraus lässt sich die Qualität nicht direkt ablesen. In einem Zwischenschritt muss aus Erfahrungswerten oder anhand softwareergonomischer Grundsätze (wie z.B. 7 ± 2) ein Optimum für das jeweilige Größenmaß festgelegt werden. Ausgehend von dem Optimum kann dann der jeweiligen Maßzahl eine Qualitätsbewertung zugeordnet werden, z.B. weniger als neun Aktivitäten in einem BPMN-Diagramm ist das Optimum (vgl. eCH0158-Diagramm-06[58]).

[58] *„Maximal 9 – 15 Aktivitäten […] pro Diagramm.“* (eCH, 2013)

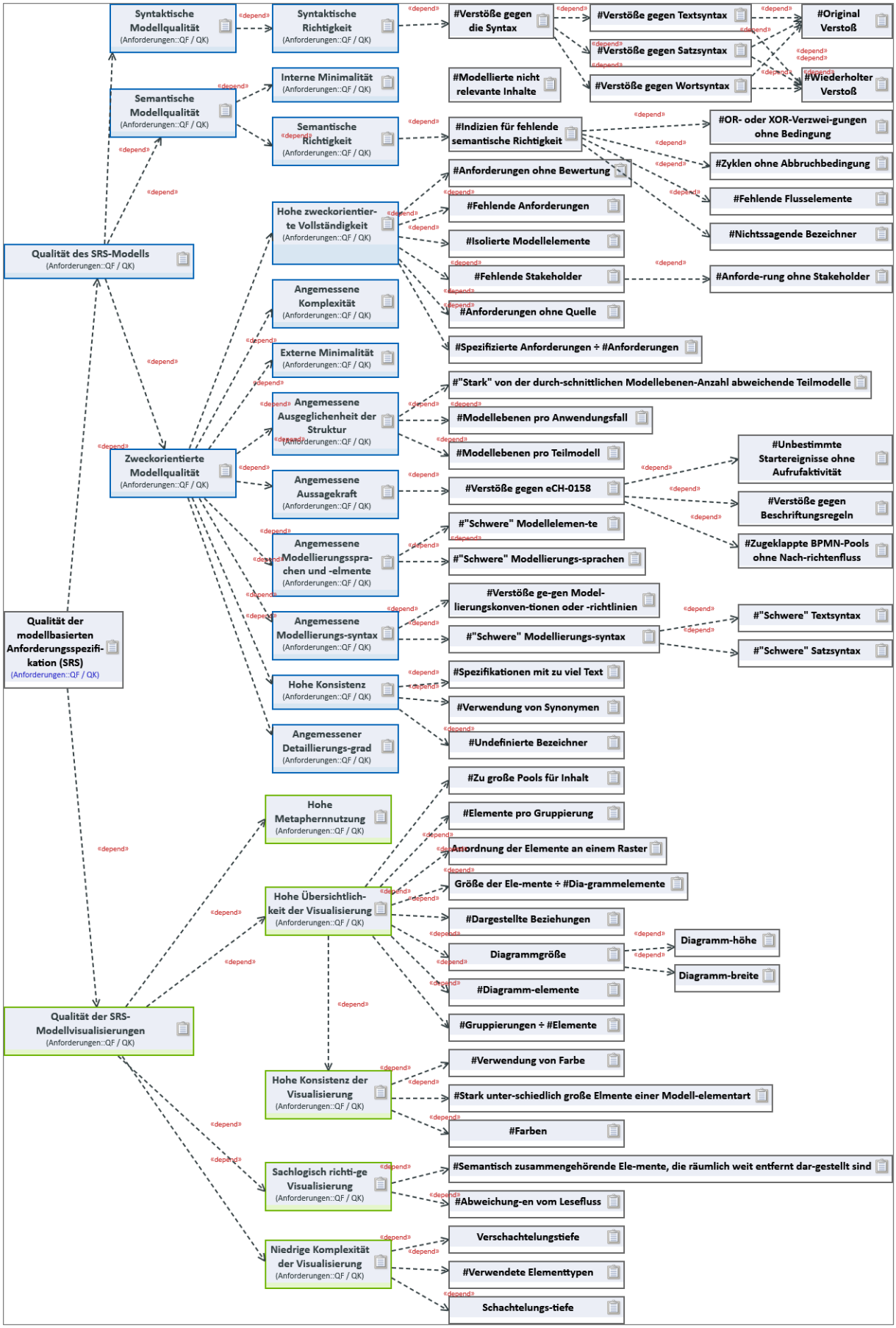


Abb. 4-10: Qualitätssystem mit allen Metriken (in grau) im Überblick.

Name	Alle	BPMN.	Anforderung	Berechnung	Definition	Art	Status[59]
Syntaktische Modellqualität							
#Original Verstoß (OV)	X			Prüf-IDs zählen.	Ein *„Original Verstoß“* liegt vor, wenn eine Syntaxregel einer Modellierungssprache zum ersten Mal nicht eingehalten wurde.	***Quali**tätsmetrik*	1
#Verstöße gegen Satzsyntax (VgSS)	X			= *VgSS.OV + VgSS.WV*	Ein *„Verstoß gegen Satzsyntax“* liegt vor, wenn eine „Satz“-Regel einer Modellierungssprache nicht eingehalten wurde.	***Quali**tätsmetrik*	1
#Verstöße gegen Syntax (VgS)	X			= *VgWS + VgSS + VgTS*	Ein *„Verstoß gegen Syntax“* liegt vor, wenn eine Syntaxregel einer Modellierungssprache nicht eingehalten wurde.	***Quali**tätsmetrik*	1
#Verstöße gegen Textsyntax (VgTS)	X			= *VgTS.OV + VgTS.WV*	Ein *„Verstoß gegen Textsyntax“* liegt vor, wenn eine „Text“-Regel einer Modellierungssprache nicht eingehalten wurde.	***Quali**tätsmetrik*	1
#Verstöße gegen Wortsyntax (VgWS)	X			= *VgWS.OV + VgWS.WV*	Ein *„Verstoß gegen Wortsyntax“* liegt vor, wenn eine „Wort“-Regel einer Modellierungssprache nicht eingehalten wurde.	***Quali**tätsmetrik*	1
#Wiederholter Verstoß (WV)	X			= $\sum_{n=OV}^{1-n}(PMfPID_n)$	Ein *„Wiederholter Verstoß“* liegt vor, wenn eine Syntaxregel einer Modellierungssprache erneut nicht eingehalten wurde.	***Quali**tätsmetrik*	1
#Prüfmeldungen für Prüf-ID (PMfPID)	X			Anzahl der Prüfmeldungen für die Prüf-ID zählen.	Eine *„Prüfmeldung für Prüf-ID“* liegt vor, wenn zu einem Modellbestandteil eine Prüfmeldung mir dieser Prüf-ID ergeben hat.	***Quali**tätsmetrik*	1
Semantische Modellqualität							
#Indizien für fehlende semantische Richtigkeit (IffsR)	X			= *ZoAB + VoB + NM + FFE*	Ein *„Indiz für fehlende semantische Richtigkeit“* ist eine Eigenschaft des Modells, die nahe lägt, dass das Original nicht richtig beschrieben wird an der Stelle wo die Eigenschaft gefunden wurde.	***Quali**tätsmetrik*	1
#Zyklen ohne Abbruchbedingung (ZoAB)		X		Anzahl der Prüfmeldungen für die Prüf-ID zählen.	Ein *„Zyklus ohne Abbruchbedingung“* ist eine Schleife, die beliebig oft (bzw. endlos) durchlaufen werden kann.	***Quali**tätsmetrik*	1

Tab. 4-14: Metriken (1/7).

[59] 0: Unvollständig spezifiziert; 1: Vollständig spezifiziert; 2: Umgesetzt

Name	Alle	BPMN.	Anforderung	Berechnung	Definition	Art	Status
Semantische Modellqualität							
#OR- oder XOR-Verzweigungen ohne Bedingung (VoB)		X		Anzahl der Prüfmeldungen für die Prüf-ID zählen.	Eine *„OR- oder XOR-Verzweigung“* ist eine Flussverzweigung ohne angegebene Bedingung.	***Quali**tätsmetrik*	1
#Nichtssagende Bezeichner (NB)	X			Anzahl der Prüfmeldungen für die Prüf-ID zählen.	Ein *„Nichtssagender Bezeichner“* ist bspw. ein Default-Name.	***Quali**tätsmetrik*	1
#Fehlende Flusselemente (FFE)		X		Anzahl der Prüfmeldungen für die Prüf-ID zählen.	Ein Flusselement fehlt, wenn bspw. ein Flusselement nicht in einen Fluss integriert ist (d.h. isoliert ist) oder der Fluss unvollständig ist.	***Quali**tätsmetrik*	1
Zweckorientierte Modellqualität							
#Modellebenen pro Teilmodell (MEpTM)	X			Anzahl der Modellebenen pro Teilmodell.		*Quantitätsmetrik*	0
#"Stark" von der durchschnittlichen Modellebenen-Anzahl abweichende Teilmodelle' (saMEA)	X			Anzahl der Teilmodelle, die um ± 2 von der durchschnittlichen Anzahl der Modellebenen abweicht.	Ein *„"Stark" von der durchschnittlichen Modellebenen-Anzahl abweichendes Teilmodell“* ist ein Teilmodell, das z.B. zwei Modellebenen mehr oder weniger als die durchschnittliche Anzahl von Modellebenen hat.	***Quali**tätsmetrik*	1
#"Schwere" Modellelemente (sME)	X			Anzahl der Prüfmeldungen für die Prüf-ID zählen.	Ein Modellelement ist *„schwer“*, wenn die Bedeutung des Elements nicht für alle Adressaten bekannt bzw. einfach (< 15 Minuten) herausgefunden und verstanden werden kann.	***Quali**tätsmetrik*	1
#"Schwere" Modellierungssprachen (sMS)	X			Anzahl der Prüfmeldungen für die Prüf-ID zählen.	Eine Modellierungssprache ist *„schwer“*, wenn sie die Adressaten nicht kennen oder sie für eine SRS nicht angemessen (z.B. zu detailliert oder Zweck der Modellierungssprache für eine SRS falsch) ist.	***Quali**tätsmetrik*	1
#"Schwere" Modellierungssyntax (sMS)	X			*= sSS + sTS*	Eine Modellierungssyntax ist *„schwer“*, wenn das „Sprachkonstrukt“ die Adressaten nicht kennen oder sie für eine SRS nicht angemessen ist.	***Quali**tätsmetrik*	1

Tab. 4-15: Metriken (2/7).

Name	Alle	BPMN.	Anforderung	Berechnung	Definition	Art	Status
Zweckorientierte Modellqualität							
#"Schwere" Satzsyntax (sSS)	X			Anzahl der Prüfmeldungen für die Prüf-ID zählen.	Eine Modellierungs-Satzsyntax ist *„schwer"*, wenn das „Sprachkonstrukt" die Adressaten nicht kennen oder sie für eine SRS nicht angemessen ist.	***Quali**tätsmetrik*	1
#"Schwere" Textsyntax (sTS)	X			Anzahl der Prüfmeldungen für die Prüf-ID zählen.	Eine Modellierungs-Textsyntax ist *„schwer"*, wenn das „Sprachkonstrukt" die Adressaten nicht kennen oder sie für eine SRS nicht angemessen ist.	***Quali**tätsmetrik*	1
#Verstöße gegen Modellierungskonventionen oder -richtlinien	X			Anzahl der Prüfmeldungen für die Prüf-ID zählen.		***Quali**tätsmetrik*	0
#Verwendung von Synonymen (VvS)	X			Anzahl der Prüfmeldungen für Synonym verwendet.	Ein Synonym ist im Glossar zu einem definierten Begriff als Synonym angegeben.	***Quali**tätsmetrik*	1
#Undefinierte Bezeichner (udB)	X			Anzahl der Prüfmeldungen für die Prüf-IDs zählen.	Ein Bezeichner ist *„undefiniert"*, wenn er leer ist oder ein Default-Name eingetragen ist.	***Quali**tätsmetrik*	1
# Spezifikationen mit zu viel Text (SmzvT)	X			Anzahl der Wörter.	Ein Spezifikationstext mit bspw. mehr als 20 Wörtern ist lang (daher Information), mehr als 50 Wörtern ist sehr lang (daher Warnung) und mehr als 100 Wörtern ist zu lang (d.h. hat *„zu viel Text"*, daher Fehler).	*Quantitätsmetrik*	1
#Verstöße gegen eCH-0158 (VgECH)		X				***Quali**tätsmetrik*	0
#Unvollständige BPMN-Prozesse (uvP)		X		Anzahl der Prozesse, die mindestens eine Prüfmeldung bzgl. *eCH0158-Pool-05* (siehe Tab. 4-10 und Tab. 4-11) haben.	Ein BPMN-Prozess ist unvollständig, wenn kein Startereignis, kein Endereignis, nur genau ein Prozess vorhanden, ein Prozesselemente nicht über den Sequenzfluss mit mindestens einem Startereignis und / oder ein Endereignis nicht mit mindestens einem Startereignis verbunden ist.	***Quali**tätsmetrik*	1
#Zugeklappte BPMN-Pools ohne Nachrichtenfluss		X				***Quali**tätsmetrik*	0

Tab. 4-16: Metriken (3/7).

Name	Alle	BPMN.	Anforderung	Berechnung	Definition	Art	Status
Zweckorientierte Modellqualität							
#Verstöße gegen Beschriftungsregeln (VgBR)	X					***Quali**tätsmetrik*	0
#Unbestimmte Startereignisse ohne Aufrufaktivität		X				***Quali**tätsmetrik*	0
#Spezifizierte Anforderungen ÷ #Anforderungen (sRE/RE)			X			*Projektmetrik*	0
#Isolierte Modellelemente (iME)	X			Anzahl der Prüfmeldungen zu isoliertes Element.	Ein Modellelement ist *„isoliert“*, wenn es keine Beziehung zu einem anderen Modellelement hat.	***Quali**tätsmetrik*	1
#Anforderungen ohne Bewertung (REoB)			X	Anzahl der Prüfmeldungen für die Prüf-ID zählen.	Eine Anforderung ist *„nicht bewertet“*, wenn weder das Eigenschaftsfeld „Relevanz“ noch „Stabilität“ belegt sind.	***Quali**tätsmetrik*	2
#Anforderungen ohne Quelle (REoQ)			X	Anzahl der Prüfmeldungen für die Prüf-ID zählen.	Eine Anforderung ist *„ohne Quelle“*, wenn das Eigenschaftsfeld „Quelle“ nicht ausgefüllt ist.	***Quali**tätsmetrik*	2
#Anforderung ohne Stakeholder (REoS)			X				2
#Fehlende Stakeholder (fS)	X			Anzahl der Prüfmeldungen für die Prüf-ID zählen.	Bestimmte Modellbestandteile dürfen im SRS-Modell nur vorkommen, wenn für sie ein Stakeholder angegeben ist z.B. jeder Anforderungen oder jedem Prozess muss mindestens ein Stakeholder hinterlegt sein.	***Quali**tätsmetrik*	1
#Fehlende Anforderungen (fRE)	X			Anzahl der Prüfmeldungen für die Prüf-ID zählen.	Bestimmte Modellbestandteile dürfen im SRS-Modell nur vorkommen, wenn sie auf einer Anforderung fußen z.B. jedem Anwendungsfall muss mindestens eine Anforderung hinterlegt sein.	***Quali**tätsmetrik*	1

Tab. 4-17: Metriken (4/7).

Name	Alle	BPMN.	Anforderung	Berechnung	Definition	Art	Status
Qualität der Modellvisualisierungen							
Schachtelungstiefe (ST)		X		Anzahl der Schachtelungen.	Die *Schachtelungstiefe* ist die Anzahl dafür, wie oft Diagrammelemente einer Elementart in einander enthalten sind z.B. Lanes oder BPMN-Teilprozesse.	*Quantitätsmetrik*	1
#Verwendete Elementtypen (VE)	X			Anzahl der im Diagramm dargestellten unterschiedlichen Modellelementtypen.		*Quantitätsmetrik*	0
#Verwendung von Farbe (VvF)	X			Anzahl der im Diagramm eingefärbten Elemente.	Die *„Anzahl der Verwendung von Farben“* ist wie oft ein Diagrammelement eingefärbt ist.	*Quantitätsmetrik*	1
#Farben	X			Anzahl im Diagramm verwendeten Farben.	Die *„Anzahl der Farben“*, ist die Zahl wie viele unterschiedliche Farben es im Diagramm gibt.	*Quantitätsmetrik*	1
#Stark unterschiedlich große Elemente einer Modellelementart	X						0
#Abweichungen vom Lesefluss (AvLF)		X				***Quali**tätsmetrik*	0
#Semantisch zusammengehörende Elemente, die räumlich weit entfernt dargestellt sind	X						0
Anordnung der Elemente an einem Raster	X					*Qualitätsmetrik*	0
#Dargestellte Beziehungen (DB)		X		Anzahl der im Diagramm dargestellten Beziehungen und Flüsse.	Eine *„Dargestellte Beziehung“* ist jede in der Modellvisualisierung dargestellte Beziehung oder auch Fluss zwischen Modellelementen.	*Quantitätsmetrik*	1
Diagrammgröße (DG)	X			*= DB * DH*	Die *„Diagrammgröße“* ist das Produkt aus Diagrammbreite und -höhe.	*Quantitätsmetrik*	1

Tab. 4-18: Metriken (5/7).

Name	Alle	BPMN.	Anforderung	Berechnung	Definition	Art	Status
Qualität der Modellvisualisierungen							
Diagrammbereite (DB)	X			Minimale X-Koordinate, die alle Modellelemente enthält.	Die *„Diagrammbreite“* ist die minimale X-Koordinate, die alle Diagramminhalte enthält.	*Quantitätsmetrik*	1
Diagrammhöhe (DH)	X			Minimale Y-Koordinate, die alle Modellelemente enthält.	Die *„Diagrammhöhe“* ist die minimale Y-Koordinate, die alle Diagramminhalte enthält.	*Quantitätsmetrik*	1
#Diagrammelemente (ME)	X			Anzahl der im Diagramm dargestellten Elemente.	Ein *„Diagrammelement“* ist jedes in der Modellvisualisierung dargestellte Modellelement.	*Quantitätsmetrik*	1
#Zu große Pools für Inhalt		X				***Quali**tätsmetrik*	0
#Elemente pro Gruppierung		X				*Quantitätsmetrik*	0
Größe der Elemente ÷ #Elemente	X						0
#Gruppierungen ÷ #Elemente		X					0

Tab. 4-19: Metriken (6/7).

Name	Alle	BPMN.	Anforderung	Berechnung	Definition	Art	Status
Metriken für UUCP							
Unadjusted Use Case Points (UUCP)				= *UCCW* + *UAW*	Nach Karner.	*Quantitätsmetrik*	2
Unadjusted Use Case Weight (UCCW)				= (eAWF * 5) + (mAWF * 10) + (kAWF * 15)	Nach Karner.	*Quantitätsmetrik*	2
Unadjusted Actor Weight (UAW)				= (kA * 3) + (mA * 2) + (eA * 1)	Nach Karner.	*Quantitätsmetrik*	2
#Einfache Anwendungsfälle (eAWF)				Einfache Anwendungsfälle zählen.	Ein Anwendungsfall ist *„einfach“*, wenn er weniger als 4 Prozessschritte hat.	*Quantitätsmetrik*	2
#Mittlere Anwendungsfälle (mAWF)				Mittlere Anwendungsfälle zählen.	Ein Anwendungsfall ist *„mittel“*, wenn er 4 bis 7 Prozessschritte hat.	*Quantitätsmetrik*	2
#Komplexe Anwendungsfälle (kAWF)				Komplexe Anwendungsfälle zählen.	Ein Anwendungsfall ist *„komplex“*, wenn er mehr als 7 Prozessschritte hat.	*Quantitätsmetrik*	2
#Einfache Akteure (eA)				Einfache Akteure zählen.		*Quantitätsmetrik*	0
#Mittlere Akteure (mA)				Mittlere Akteure zählen.		*Quantitätsmetrik*	0
#Komplexer Akteure (kA)				Komplexe Akteure zählen.		*Quantitätsmetrik*	0

Tab. 4-20: Metriken (7/7)

4.2.4 BEWERTUNG

In diesem Abschnitt stellen wir als erstes unsere Skala für die Qualitätsbewertung vor und erläutern anschließend wie Schwellwerte definiert und mittels dieser Werte den Maßzahlen Qualitätsbewertungen zugeordnet werden können. Abschließend beschreiben wir ein Ansatz zur Ableitung der Bewertung von Qualitätsfaktoren und -kriterien aus den untergeordneten Kriterien und Metriken.

BEWERTUNGSSKALA

Für die Bewertung der *Qualität modellbasierter SRS* wird in dieser Arbeit vorgeschlagen, zwischen den folgenden vier Qualitätslevel zu unterscheiden: *Hervorragend*, *gut*, *ausreichend* und *mangelhaft* (siehe Tab. 4-21). Diese Ordinalskala ist an die deutsche Schulnotenskala angelehnt („sehr gut“, „gut“, „befriedigend“, „ausreichend“, „mangelhaft“ und „ungenügend“). Unser *hervorragend* entspricht dem „sehr gut“. *Hervorragend* liefert uns die Information, dass der Prüfgegenstand unsere Qualitätsanforderungen komplett erfüllt, wir vielleicht sogar zu viel Arbeit in die Erstellung gesteckt haben (vgl. Def. GoM-6). Dass die Qualitätsanforderungen nicht erfüllt werden, jedoch noch in einem akzeptablen Maße, wird von dem Niveau *ausreichend* verkörpert. *Mangelhaft* ist im Gegensatz dazu das Qualitätsniveau, welches auf keinen Fall akzeptiert werden kann, da wir hier grob gegen unsere Zielwerte verstoßen.

Qualitätsniveau	Bedeutung
Hervorragend	Das Messobjekt ist *hervorragend*, wenn eine geringfügige Anzahl an Qualitätsdefiziten (i.S.v. Qualitätsmetriken) gefunden sowie eine geringfügige Anzahl an Quantitätsmetriken stark verfehlt wurde.
Gut	Das Messobjekt ist *gut*, wenn wenige Qualitätsdefizite (i.S.v. Qualitätsmetriken) gefunden sowie wenige Zielwerte der Quantitätsmetriken stark verfehlt wurden.
Ausreichend	Das Messobjekt ist *ausreichend*, wenn nicht zu viele Qualitätsdefizite (i.S.v. Qualitätsmetriken) gefunden sowie nicht zu viele Zielwerte der Quantitätsmetriken stark verfehlt wurden.
Mangelhaft	Das Messobjekt ist *mangelhaft*, wenn zu viele Qualitätsdefizite (i.S.v. Qualitätsmetriken) gefunden oder zu viele Zielwerte der Quantitätsmetriken stark verfehlt wurden.

Tab. 4-21: Definition der Qualitätsniveaus.

Die Noten „befriedigend“ und „ungenügend“ wurden ausgelassen, da sie keinen relevanten Informationsgehalt beinhalten, denn wir streben eine *gute* Qualität für die Qualitätsfaktoren an. Die zusätzliche Unterteilung der Niveaus *ausreichend* und *mangelhaft* in „befriedigend“ und „ungenügend“ könnte seine Berechtigung haben, wenn das Ziel der Skala ist, nicht nur eine Feststellung der Erreichung des angestrebten Qualitätsniveaus, sondern als Ansporn für die Modellierer und Bewertung des Fortschritts der Qualitätsverbesserung dienen soll. Dies trifft auf unseren angestrebten Einsatzzweck nicht zu. Zusätzlich ist darauf hinzuweisen, dass durch die Hinzunahme der Level „befriedigend“ und „ungenügend“ sich der Aufwand für die Definition der Qualitätslevel erhöht. Denn in Folge müssten anstelle von drei für jede Metrik fünf Schwellwerte definiert und voneinander abgegrenzt werden.

SCHWELLWERTE DEFINIEREN UND QUALITÄTSLEVEL ZUORDNEN

Wie können nun auf Grundlage der Bewertung der Maßzahlen die Qualitätskriterien der untersten Ebene unseres Qualitätssystems bewertet werden? Die erste Möglichkeit ist, dass für jede Metrik drei Schwellwerte definiert werden, welche die Ergebnisse der jeweiligen Metrik in den Bereich der *hervorragenden*, *guten*, *ausreichenden* und *mangelhaften* Maßzahlen aufteilt. Die Schwellwerte können entweder absolut oder relativ zur Größe des Modells oder relativ zur Anzahl der Elemente, auf die die jeweilige Metrik ausgeführt wird, definiert werden. Um auf die unterschiedlichen Größen von *SRS-Modellen* bei der Bewertung einzugehen, schlagen wir eine Definition der Qualitätsniveaus relativ zur Größe des *SRS-Modells* vor. Für die Größe der SRS werden bspw. in der Literatur unterschiedliche Metriken vorgeschlagen, z.B. Use Case Points (siehe Abschnitt 4.2.3) oder #Anforderungen (Sneed, et al., 2010). Für unsere Qualitätsbewertung haben wir uns entschieden die Schwellwerte relativ zu den Use Case Points zu definieren.

Die jeweiligen Schwellwerte der Metriken, für die Zuordnung des Qualitätslevel und somit die Bewertung, sollte projektspezifisch festgelegt werden. Als Richtwert für die Definition der Qualitätslevels kann, wie schon für die Bewertungsskala, die Schulskalendefinition dienen (siehe Tab. 4-22).

Noten	*Sehr gut*	*Gut*	*Befriedigend*	*Ausreichend*	*Mangelhaft*	*Ungenügend*
Punkte	100 – 92	91 – 81	80 – 67	66 – 50	49 – 30	29 – 0

Tab. 4-22: Definition der Notenskala bei 100 Gesamtpunkten (IHK, 2006).

Analog zu der Schulskala kann für das Qualitätslevel *gut* definiert werden: Die Maßzahl einer ***Qualitäts**metrik* ist als *gut* zu bewerten, wenn folgendes gilt:

$$92\% > \left(1 - \frac{Maßzahl}{UCP}\right) > 80\%$$

Somit ist für das folgende Beispiel die Maßzahl als gut zu bewerten:

$$\#Verstöße\ gegen\ Syntax\ (VgS)_{einfach} = 12\ ;\ UCP = 100$$

$$1 - (12\ \div 100) = 88\% \implies gut$$

Um auf die unterschiedliche Schwere der Qualitätsdefizite (d.h. Fehler, Warnungen und Informationen) einzugehen, können die Defizite vorab untereinander entsprechend gewichtet werden, analog zu den UUCW der UCP (siehe Abschnitt 3.3) schlagen wir vor:

Beispiel einer Maßzahlberechnung:

$$VgS_{Fehler} = 5\ ;\ VgS_{Warnung} = 2\ ;\ VgS_{Information} = 1\ ;\ UCP = 100$$

$$\#Verstöße\ gegen\ Syntax\ (VgS)_{gewichtet}$$
$$= (VgS_{Fehler} * 3) + (VgS_{Warnung} * 2) + (VgS_{Information} * 1)$$
$$= (5 * 3) + (2 * 2) + (1 * 1) = 20$$

Beispiel einer Qualitätslevel-Zuordnung:

$$1 - (20\ \div 100) = 80\% \implies befriedigend$$

In diesem Fall verschiebt sich durch die Gewichtung die Maßzahlenbewertung. Um dieser Verschiebung entgegenzuwirken, kann entweder die berechnete Maßzahl

zum Schluss durch den Mittelwert der Gewichte geteilt, oder die Schwellwerte der Bewertungsskala werden entsprechend angepasst.

Die Bewertungszuordnung für eine ***Quantitäts**metriken* kann wie beim folgenden Beispiel $\#\ Spezfikationen\ mit\ zu\ viel\ Text\ (SmzvT)$ vorgenommen werden:

> Bezeichnet $\mathcal{s}$ eine Spezifikation aus der Menge der betrachteten Spezifikationen $\mathcal{S}$ und $\mathcal{W}(\mathcal{s})$ eine Funktion, die eine Spezifikation auf die Anzahl der darin enthaltenen Wörter abbildet, dann kann die Maßzahl der Quantitätsmetrik $SmzvT$ wie folgt berechnet und ihr ein Qualitätslevel zugeordnet werden:
>
> **Definition der Schwellwerte:**
>
> $SmzvT_{Fehler} = |\{\mathcal{s} \in \mathcal{S} : \mathcal{W}(\mathcal{s}) \geq 100\}|$
>
> $SmzvT_{Warnung} = |\{\mathcal{s} \in \mathcal{S} : 50 \leq \mathcal{W}(\mathcal{s}) < 100\}|$
>
> $SmzvT_{Information} = |\{\mathcal{s} \in \mathcal{S} : 20 \leq \mathcal{W}(\mathcal{s}) < 50\}|$
>
> Wobei $|\mathcal{A}|$ für die Anzahl der Elemente der Menge $\mathcal{A}$ steht.
>
> **Beispiel einer Maßzahlberechnung:**
>
> $SmzvT_{Fehler} = 3\ ;\ SmzvT_{Warnung} = 2\ ;\ SmzvT_{Information} = 1\ ;$
> $UCP = 100$
>
> $SmzvT_{einfach} = SmzvT_{Fehler} + SmzvT_{Warnung} + SmzvT_{Information}$
> $= 2 + 0 + 3 = 6$
>
> $SmzvT_{gewichtet}$
> $= (SmzvT_{Fehler} * 3) + (SmzvT_{Warnung} * 2) + (SmzvT_{Information} * 1)$
> $= (2 * 3) + (0 * 2) + (3 * 1) = 10$
>
> **Beispiel einer Qualitätslevel-Zuordnung:**
>
> $1 - (10\ \div 100) = 90\% \implies gut$

QUALITÄTSBEWERTUNG ABLEITEN

Für die Ableitung der Bewertung eines Qualitätsfaktors oder -kriteriums aus der Bewertung der untergeordneten Kriterien oder Metriken gibt es unterschiedliche Möglichkeiten (siehe Abschnitt 3.3.2). Folgende Variante schlagen wir für die Bewertung der *SRS-Modell-* und *Modellvisualisierungsqualität* vor, welcher unserer Meinung nach der simpelste ist:

Wurden für die Metriken die Schwellwerte für jedes Qualitätslevel definiert, so ist der einfachste Ansatz zur Bewertung des übergeordneten Kriteriums bzw. Knoten im Baum des Qualitätssystems wie folgt:

> *Der übergeordnete Knoten kann maximal so gut bewertet werden, wie die schlechteste Bewertung der untergeordneten Knoten. Folglich schlägt die schlechteste Bewertung im Bewertungsbaum bis zur Wurzel durch.*

Unser Ziel ist es nicht unterschiedliche Modelle auf Basis der Metriken miteinander zu vergleichen, sondern eine angemessen Qualität des Prüfgegenstandes zu sichern. Daher schlagen wir diesen Bewertungsansatz vor, der die Erfüllung aller Teilaspekte der Qualität fordert.

4.2.5 ZUSAMMENFASSUNG

In diesem Abschnitt haben wir gezeigt, wie die automatische Qualitätsbewertung theoretisch für ein Modellierungswerkzeug umgesetzt werden kann.

Hierzu haben wir zunächst das Ziel formuliert, mittels prüfungsbasierten Metriken das Qualitätslevel des *SRS-Modells* und der *SRS-Modellvisualisierungen* automatisch zu bestimmen, im ersten Schritt beschränkt auf die Modellbestandteile Anforderungen, Anwendungsfälle sowie BPMN-Startereignisse und Pools. Beispielhaft wurden für Richtlinien aus eCH-0158 (für Pools und Startereignisse) Prüfungen spezifiziert. Anschließend haben wir die Metriken des Qualitätssystems definiert. Abschließend wurde eine Skala für die Qualitätsbewertung vorgeschlagen sowie ein Verfahren für die Bewertung von Maßzahlen relativ zur Größe des Modells und für die Ableitung der Bewertung der übergeordneten Qualitätskriterien und -faktoren aus den untergeordneten Kriterien oder Metriken.

5 Umsetzung in Innovator

In diesem Kapitel wird das Modellierungswerkzeug Innovator und speziell die Dokumentationsgenerierung sowie Möglichkeiten der Qualitätssicherung in Innovator vorgestellt (Abschnitt 5.1). Zudem wird die Umsetzung von Modellprüfungen (5.2) sowie Modellvermessung (5.3) in Innovator erläutert.

5.1 Modellierungswerkzeug Innovator

Nachdem in den vorherigen Abschnitten der Entwurf für die Qualitätsbewertung von *SRS-Modellen* und *-Modellvisualisierungen* entwickelt wurde, wird hier das für die Umsetzung relevante Modellierungswerkzeug Innovator beschrieben. Dieses Werkzeug wird von der in Nürnberg ansässigen Firma MID GmbH entwickelt. Die neuste Generation des Tools ist „Innovator X“, Version 11.5 von November 2012 (siehe Abb. 5-1). Mit dem ersten Release 1986 ist Innovator ein über die letzten Jahrzehnte „gewachsenes“ Werkzeug (Wikipedia, 2013a). Heute umfasst das Werkzeug je nach Auslieferungs- bzw. Lizenzumfang die grafischen Modellierungssprachen UML 2.0, BPMN 2.0, ER- und Datenbank-Diagramme, sowie toolspezifische Notationsarten (siehe Tab. 5-1). In Innovator können nicht nur Diagramme erstellt werden, sondern es kann ein Modell im Sinne eines Systems erstellt werden (d.h. mit diagrammübergreifenden Beziehungen und Abhängigkeiten zwischen den Modellbestandteilen). Dokumentationen sowie Code können aus einem solchen Modell generiert werden. Laut MID haben über 600 Unternehmen und Organisationen den Innovator im Einsatz, u.a. die BARMER GEK, T-Systems und die Bundesagentur für Arbeit (MID GmbH, 2013a). Laut dem Hersteller kann Innovator für Geschäftsprozessmodellierung, objektorientierte Softwareanalyse, objektorientiertes Design und relationale Datenmodellierung eingesetzt werden (MID GmbH, 2013i).

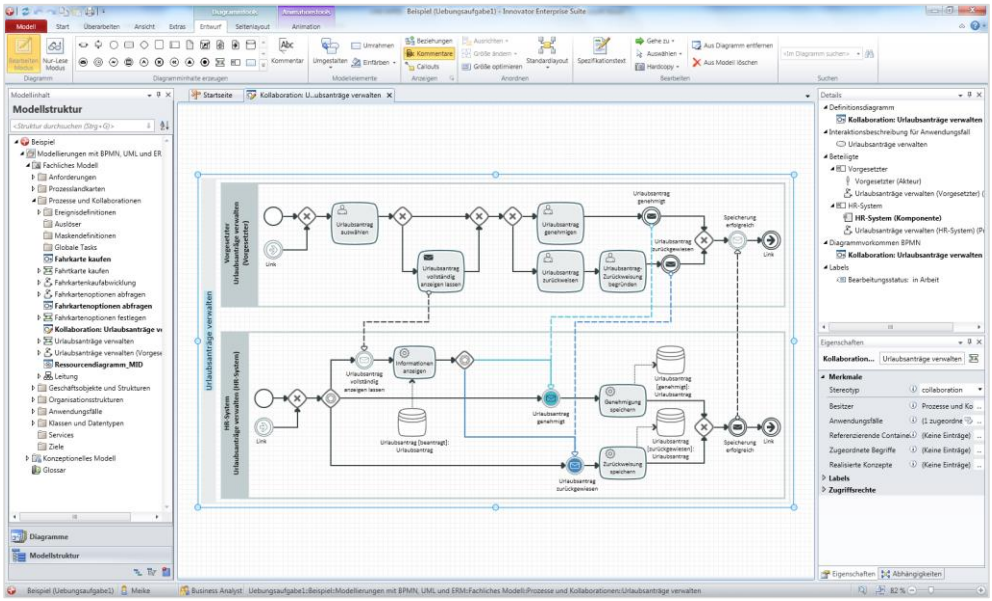


Abb. 5-1: Screenshot der Innovator X-Modellierungsoberfläche.

Notationen	Beschreibungsgegenstand
BPMN	
Prozessdiagramm	Beschreibt Prozesse und Kollaborationen[60] (siehe Abb. 1-5, S.5, in Abschnitt 1.1).
Innovator-Spezifische	
Diagramm für Geschäftsobjektzustände[61]	Beschreibt Verhalten und Zustände eines Geschäftsobjekts (siehe Abb. 8-22, S. 162, in Abschnitt 8.3).
Geschäftsressourcendia-gramm	Beschreibt Beziehungen unter Geschäftsressourcen und Organisationsstrukturen.
Organigramm	Beschreibt Organisationsstrukturen.
Prozesslandkarte	Beschreibt Struktur und Zusammenwirken von Geschäftsprozessen[62] (siehe Abb. 8-25, S. 163, in Abschnitt 8.3).
Strukturdiagramm (für Geschäftsobjekte)	Beschreibt Struktur „vereinfacht“[63] (siehe Abb. 8-23, S. 162, in Abschnitt 8.3).
Whiteboard-Diagramm (bzw. Übersichtsdiagramm)	Beschreibt Beziehungen zwischen Diagrammelementen unterschiedlicher Dia-grammtypen (siehe Abb. 8-27, S. 165, in Abschnitt 8.3).
Weitere:	*Maskenflussdiagramm*, *Paketdiagramm*.
UML	
Aktivitätsdiagramm	Beschreibt Prozesse (siehe Abb. 8-26, S. 164, in Abschnitt 8.3).
Klassendiagramm	Beschreibt Struktur durch Klassen, Klasseneigenschaften sowie Beziehungen zwischen den Klassen (siehe Abb. 8-24, S. 163, in Abschnitt 8.3).
Anwendungsfalldiagramm (bzw. Use Case Diagramm)	Beschreibt zu Systemen Systemgrenzen, -akteure und Anwendungsfälle (siehe Abb. 8-21, S. 161, im Anhang).
Objektdiagramm	Beschreibt Instanzen von Klassen (siehe rechte Abbildung in Abb. 3-3, S. 35, in Abschnitt 3.4).
Zustandsdiagramm (bzw. -automaten)	Beschreibt Zustände und Zustandsübergänge von z.B. Datenobjekten.
Weitere:	*Komponentendiagramm* , *Kompositionsstrukturdiagramm*, *Sequenzdiagramm*, *Verteilungsdiagramm*.
Diverse	
Entity-Relationship-Diagramme (in UML-, IDEF1X, Chen, DSA, SERM und James-Martin-Notation), Daten-bank-Diagramme (IDEF1X-Notation).	

Tab. 5-1: Von Innovator 11.5 unterstützte Notationen (MID GmbH, 2013g).

[60] Kollaborationen beschreiben die Interaktion mehrerer Prozesse untereinander.

[61] Die Notation ist angelehnt an UML-Zustandsdiagramme.

[62] Ein Geschäftsprozess ist *„Eine Folge von logisch zusammenhängenden Aktivitäten, die für das Unternehmen einen Beitrag zur Wertschöpfung leisten, einen definierten Anfang und ein definiertes Ende haben, wiederholt durchgeführt werden und sich in der Regel am Kunden orientieren.“* (Laudon, et al., 2006)

[63] Diese Diagrammnotationsart soll durch eine einfache Modellierungsnotation ermöglichen, dass die Fachabteilung ihre Objektstrukturen selbst beschreiben kann, ohne schwierige Notationen wie Klassendiagramme oder ER-Diagramme erlernen zu müssen.

Die Software (Innovator) basiert seit 1992 auf einer Client-Server-Architektur (Wikipedia, 2013a). Dies ermöglicht die zeitgleiche Arbeit mehrerer Benutzer an einem Modell. Über eine rollenbasierte Rechteverwaltung können für Modellinhalte „Lese- und Schreibrechte“[64] verschiedener angelegter Rollen verwaltet werden. Einem Anwender können eine oder mehrere dieser Rollen zugewiesen werden (MID GmbH, 2013h).

Aus den Modelldaten können mittels der Funktion des Dokumentationsgenerators Dokumentationen (siehe 5.1.1) und Code generiert werden (z.B. C#, Java, DB-Schemata). Des Weiteren bietet eine Office Integration die Möglichkeit Anwendungsfallbeschreibungen oder textuelle Anforderungen mit Excel- oder Word-Dokumentinhalten zu synchronisieren. Speziell für diese Arbeit interessant ist die Möglichkeit, dass mittels vordefinierter oder selbst entwickelter Prüfroutinen Innovator-Modelle automatisiert geprüft werden können (MID GmbH, 2013c). Die Funktionsweise dieser Prüfungen mittels des „Assistenten zur Modellprüfung“ wird ausführlich in Abschnitt 5.1.2 erläutert.

Bei dem Modellierungswerkzeug Innovator wird unterschieden zwischen dem Modell einerseits und Diagrammen, d.h. grafischen Darstellungen von Teilen des Modells, andererseits. Ein Modellelement kann folglich in mehreren Diagrammen dargestellt werden. Bspw. kann ein BPMN-Prozess, welcher in einem Diagramm definiert ist, in demselben oder weiteren Diagrammen als derselbe Prozess dargestellt werden. Bei BPMN-Prozessen besteht die Besonderheit, dass genau eine Darstellung die definierende ist, wenn ein Prozess mehrfach dargestellt wird. Bei der Wiederverwendung kann die verknüpfte Darstellung des Prozesses (d.h. nicht die definierende Darstellung des Prozesses) in einer abgespeckten Form angezeigt werden (d.h. Bestandteile des Prozesses können ausgeblendet werden), jedoch kann der Prozess nur in der definierenden Darstellung erweitert werden, nicht in der verknüpften.

Die Beziehungen und Abhängigkeiten zwischen Bestandteilen des Modells können über Eigenschaftsfelder (siehe Abb. 5-2, S. 103), Beziehungs- (siehe Abb. 5-3, S. 103) und Abhängigkeitskanten (siehe Abb. 5-4, S. 103) modelliert werden. Bspw. werden Anforderungen über Abhängigkeitskanten den Objekten wie z.B. Anwendungsfällen im Modell zugeordnet (siehe Abb. 5-4, S. 103). Welche und wie Beziehungen zwischen den Modellbestandteilen gezogen werden dürfen, wird in der Profil-Konfiguration festgelegt. Dort wird auch definiert, welche Modellbestandteile in welchem Diagramm, Modell oder überhaupt zur Verfügung stehen (siehe Abschnitt 5.1.2).

[64] In Innovator wird zwischen Zugriffs-, Ausführungs-, Verfahrens- und Administratorrechten unterschieden (MID GmbH, 2013h).

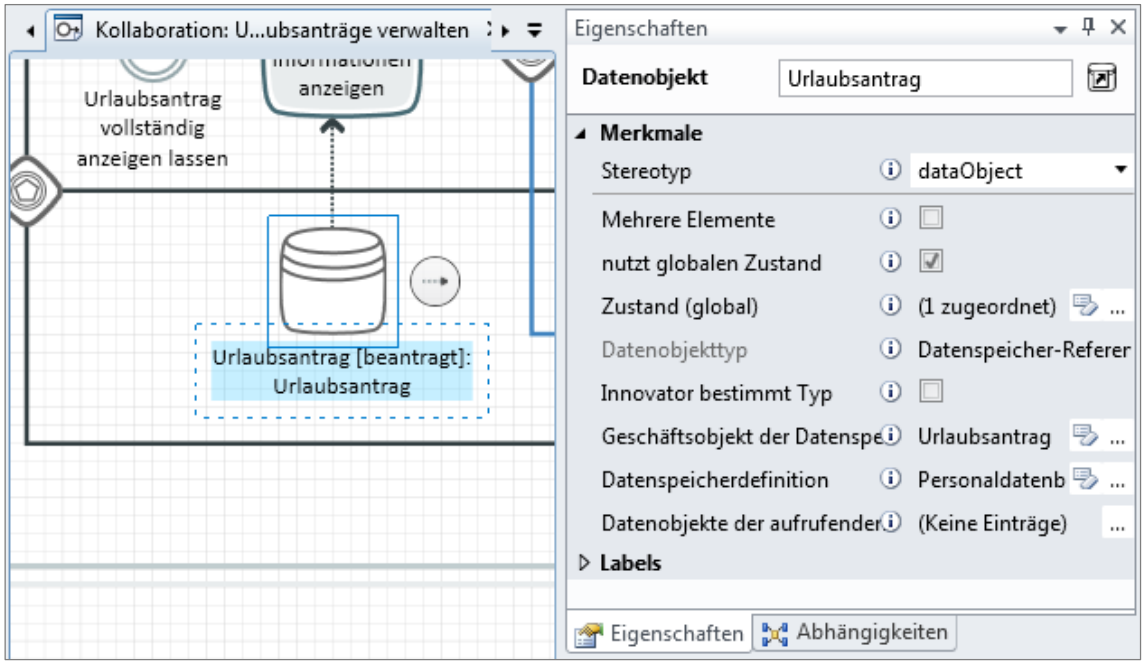


Abb. 5-2: Eigenschaftsinformationen (Bsp.: Zu einem BPMN-Datenobjekt).

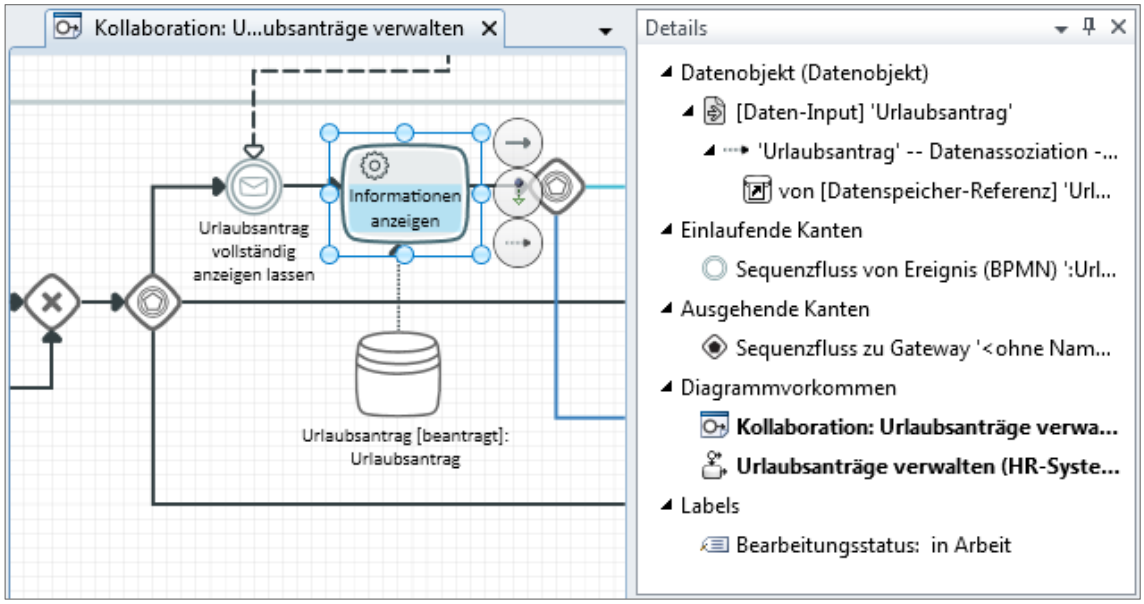


Abb. 5-3: Informationen zu u.a. Beziehungskanten (Bsp.: Zu einem BPMN-Task).

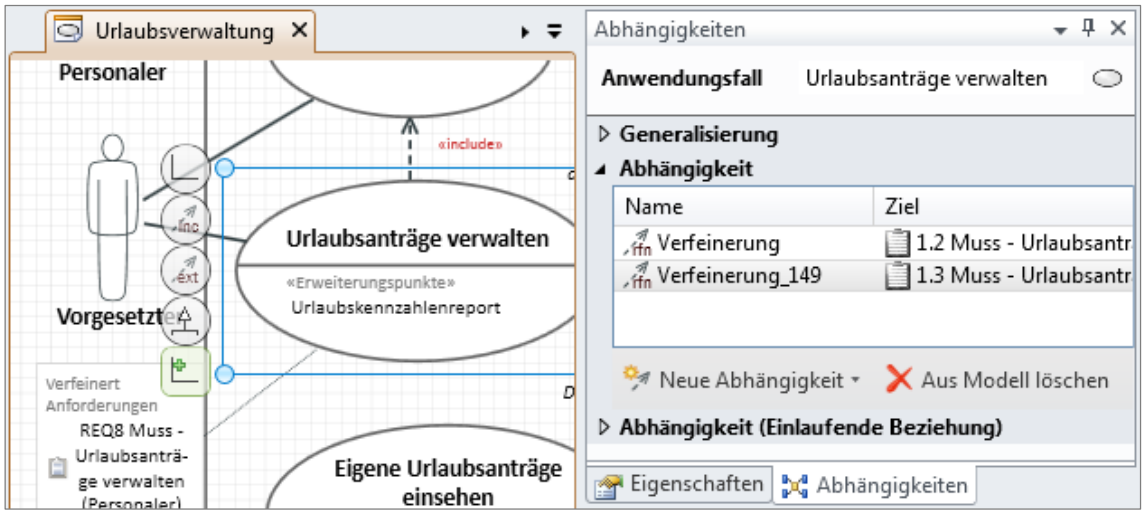


Abb. 5-4: Informationen zu Abhängigkeitskanten (Bsp.: Zu einem Anwendungsfall).

5.1.1 Dokumentationsgenerierung

Innovator bietet die Möglichkeit aus dem erstellten Modell eine Dokumentation als Word- oder HTML-Dokument (bzw. XML-Dokument) „auf Knopfdruck“ erstellen zu lassen. Bei der Ausführung der Generierung kann die Menge der Eingabedaten definiert werden (siehe Abb. 5-5).

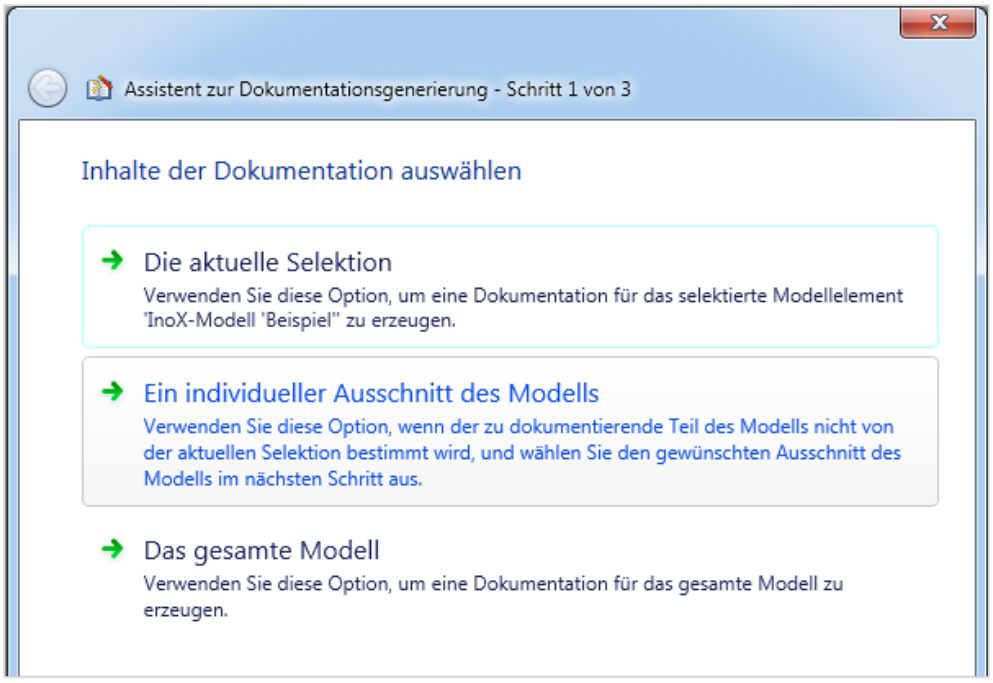


Abb. 5-5: Eingabemenge für die Doku.-Generierung definieren.

Die Ausgabeeinstellungen werden in der Konfiguration festgelegt: Dort wird vorgegeben, welche Elementtypen des Modells und zugehörigen Informationen wie und wo in der Dokumentation erscheinen sollen. Je nach Konfiguration des Profils📖 ist es auch möglich bei der Generierung zwischen einer Auswahl an unterschiedlichen Ausgabeformaten zu entscheiden (siehe Abb. 5-6, S. 105).

Einsatz von Innovator für die Generierung eines modellbasierten Lastenhefts nach V-Modell XT

In Abb. 5-7 (S. 105) ist ein sehr simples Beispiel für die Struktur eines generierten Dokuments (in blau) mit der Zuordnung der Modellbestandteile, die für die Generierung verwendet werden (in orange). In blau sind die Teilkapitel des Lastenhefts nach dem V-Modell XT aufgeführt. Denen sind in orange Innovator-Modellbestandteiltypen (Ausnahme: „Textdokument“ ist modellextern, siehe Abschnitt 1.1) gegenübergestellt. Aus ihren Instanzen können die Informationen für das jeweilige Kapitel des modellbasierten Lastenhefts bei der Dokumentationsgenerierung gespeist werden. Wie bereits erwähnt, handelt es sich hierbei um ein simples Beispiel, welches die Funktionalität der Dokumentationsgenerierung erklären soll. Die Modellierung von u.a. Daten durch bspw. Klassen- oder ER-Diagramme wird hier daher nicht dargestellt, zumal diese Modellbestandteile im Rahmen dieser Arbeit nicht behandelt werden.

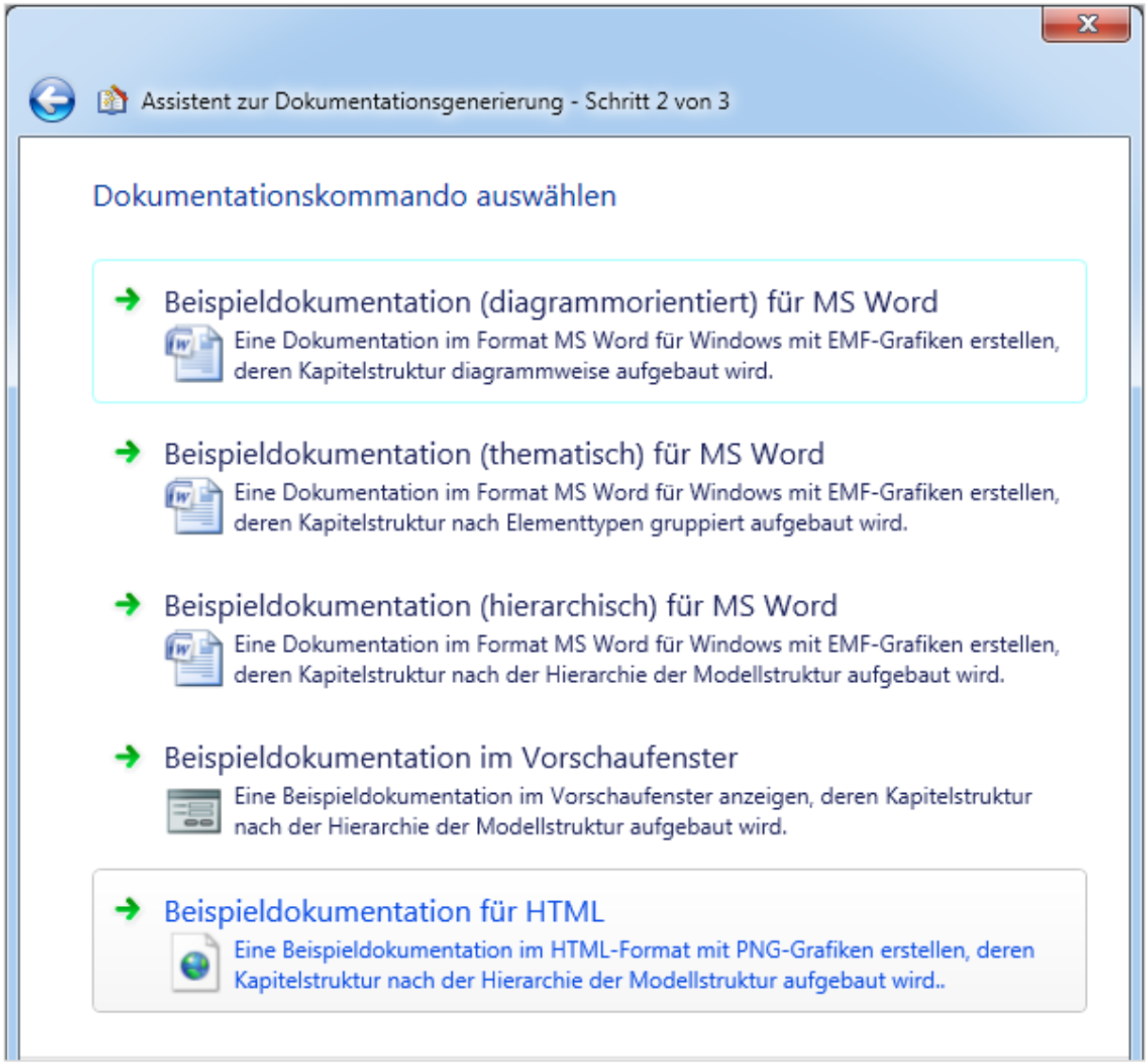


Abb. 5-6: Ausgabeformate und -arten für die Dokumentationsgenerierung.

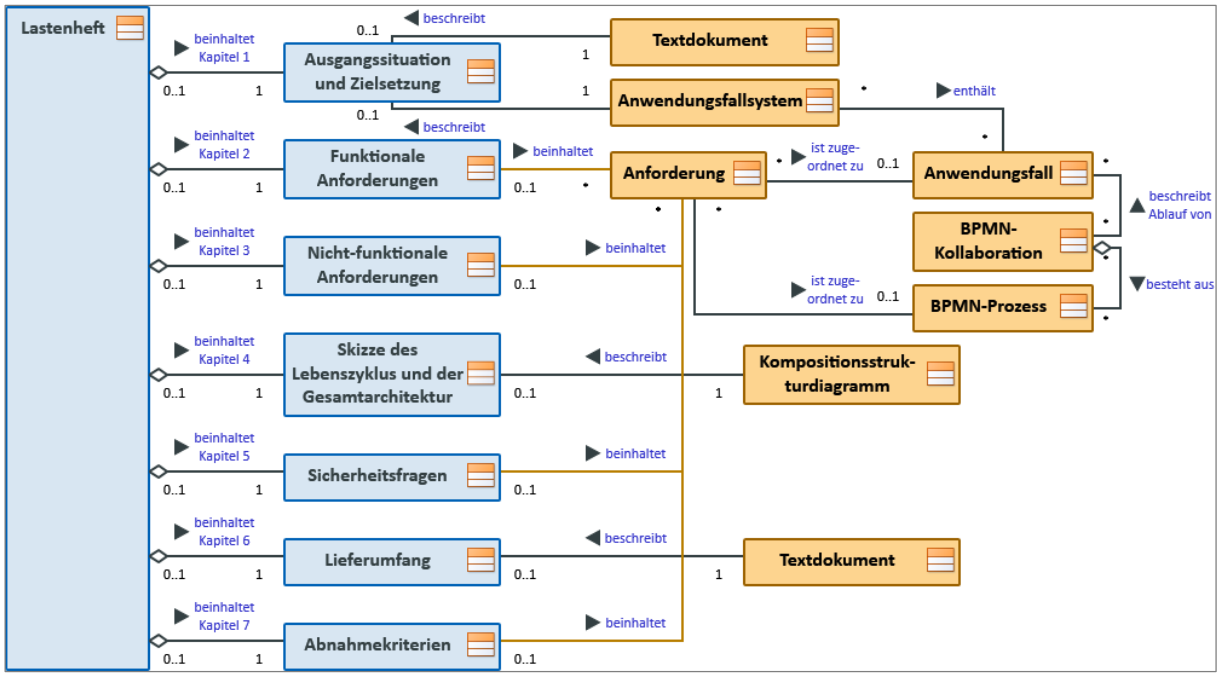


Abb. 5-7: Beispiel für eine Umsetzung des V-Modell XT-Lastenhefts in Innovator.

Nach dem V-Modell wird im ersten Kapitel des Lastenhefts die „Ausgangssituation und Zielsetzung“ des Softwareentwicklungsprojekts vorgestellt. Dieses Kapitel kann mit Hilfe von Innovator inhaltlich und grafisch mit bspw. Informationen des zentralen Anwendungsfalldiagramms des SRS-Modells gefüttert werden. Zusätzlich geht an dieser Stelle ein modellexterner Text in die SRS-Dokumentation ein. Im darauf folgenden Lastenheftkapitel werden die funktionalen Anforderungen an das System aufgeführt. Hier werden im modellbasierten Lastenheft alle Modellelemente des Typs „funktionale Anforderung“ aufgeführt. Die jeweilige Anforderung kann sich auf einen Anwendungsfall oder einen Prozess (oder auch andere Modellelemente, die in Abb. 5-7 (S. 105) nicht aufgeführt sind, z.B. Prozesseschritt, Datenobjekt, etc.) beziehen. Wird die Anforderung einem Anwendungsfall zugeordnet, so kann der Ablauf des Anwendungsfalls wiederum grafisch mit Hilfe eines Aktivitätsdiagramms oder eine BPMN-Kollaboration beschrieben werden. Die BPMN-Kollaboration besteht wiederum aus einem oder mehreren BPMN-Prozessen. Die Inhalte der Kapitel 3, 5 und 7 des Lastenhefts können ebenfalls als Anforderungen modelliert werden. In diesem Fall Anforderungen zu nicht-funktionalen Aspekten, Sicherheitsfragen und Abnahmekriterien. So können zu jedem Kapitel des Lastenhefts entsprechende Informationen aus dem Modell ausgelesen und mittels der definierten Dokumentationsgenerierung in Form eines Word- oder HTML-Dokuments in ein SRS überführt werden. Die Ausnahme ist das sechste Kapitel „Lieferumfang“. Für den Inhalt dieses Kapitels kann in der Konfiguration vorgesehen werden, dass ein entsprechendes Textdokument hinterlegt wird. Bei der Generierung des Dokuments wird der Inhalt für dieses Kapitel aus dem Textdokument wie bspw. MS[65]-Word-Dokument ausgelesen und für das entsprechende Kapitel des Lastenhefts übernommen.

Auszüge einer Beispieldokumentation sind in Abschnitt 1.1 zu sehen (siehe Abb. 1-7, S. 7).

5.1.2 Möglichkeiten der „automatischen“ Qualitätssicherung

In Innovator haben wir drei mögliche Herangehensweisen für die werkzeuggestützte Qualitätssicherung identifiziert. Die erste Möglichkeit ist konstruktiv. Über die Profil-Konfiguration können die Freiheitsgrade der Modellierung in Innovator eingeschränkt werden, bspw. kann die Struktur des Modells vorgegeben und eingeschränkt werden. Somit ist es dem Modellierer versagt bestimmte Modellkonstruktionen in Innovator-Modell zu erstellen. Eine analytische Möglichkeit des Qualitätssicherung ist der Modellprüfassistenten, durch welchen das Modell überprüft werden kann und somit automatisch Regel- oder Richtlinienverstöße aufgezeigt werden können. Um nicht nur einzelne Prüfergebnisse durch den Modellprüfungsassistenten anzeigen zu lassen, sondern zusätzlich Größen- und Qualitätsmaße auszugeben, können Metriken- als auch die in Abschnitt 4.2.4 vorgeschlagene Qualitätsniveauberechnung über eine Engineering-Aktion in Innovator umgesetzt werden.

[65] MS: Microsoft.

Somit gibt es für die Umsetzung der Qualitätssicherung die folgenden Alternativen in Innovator:

- *Modellierungseinschränkung* über die Konfiguration des „Profils“
- *Automatische Prüfung* über den Modellprüfungsassistenten
- *Metrik-Berechnung* über die Engineering-Aktionen

Im Folgenden werden diese Möglichkeiten jeweils ausführlich vorgestellt.

5.1.2.1 MODELLIERUNGSEINSCHRÄNKUNG (PROFIL-KONFIGURATION)

In Innovator kann für ein Profil[66] bspw. die Menge der für die Modellierung allgemein oder für einen bestimmten Diagrammtyp (siehe Abb. 5-8, S. 108) möglichen Bestandteile beschränkt oder auch erweitert werden. Darüber kann einem Modellierer, der z.B. ein BPMN-Diagramm erstellt, versagt werden Fehlerereignisse oder für eine BPMN-Aktivität mehr als einen ausgehenden Sequenzfluss zu modellieren (vgl. eCH, 2013).

In Abb. 5-9 (S. 108) ist zu sehen, dass einem Task nach der Standard BPMN 2.0 Konfiguration beliebig viele Sequenzflüsse zugeordnet werden. Um bspw. das Modellieren nach dem Standard eCH-0158 in Innovator über die Konfiguration umzusetzen, können wir in einem neuen Profil eine neue Ausprägung der Stereotype BPMN-Diagramm erzeugen, nämlich das eCH-0158-konforme Diagramm. Für dieses Diagramm legen wird fest, dass in einem solchen Diagramm für jeden Task maximal ein zugeordneter Sequenzfluss dargestellt werden darf.

[66] *„Ein Profil stellt in Innovator eine für sich abgeschlossene Einheit dar, die bestimmte Aspekte der Konfiguration komplett enthält oder aufbauend auf einem importierten Profil erweitert. Das vorrangige Konstrukt für eine solche Erweiterung ist das Stereotyp, welches als Teil eines Profils definiert und auf existierende Metaklassen angewendet wird. Bestimmte Profile beinhalten die Strukturen und Stereotype der UML 2, der BPMN 2 bzw. des IMM/CWM der OMG. Außer dem Basisprofil können alle Profile andere Profile importieren. Ein Profil kann als Einheit (Profil-Vorlage) gespeichert und geladen werden.*“ (MID GmbH, 2013b)

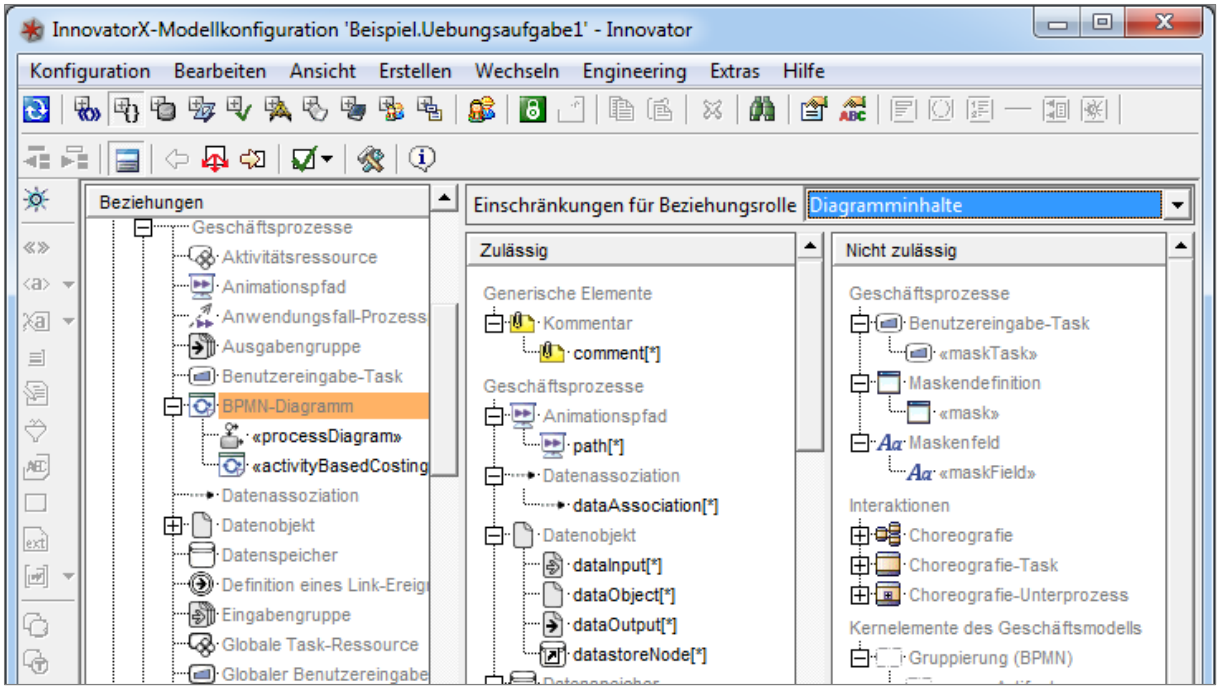


Abb. 5-8: Konfiguration der Beziehung „Diagramminhalte“ von BPMN-Diagrammen.

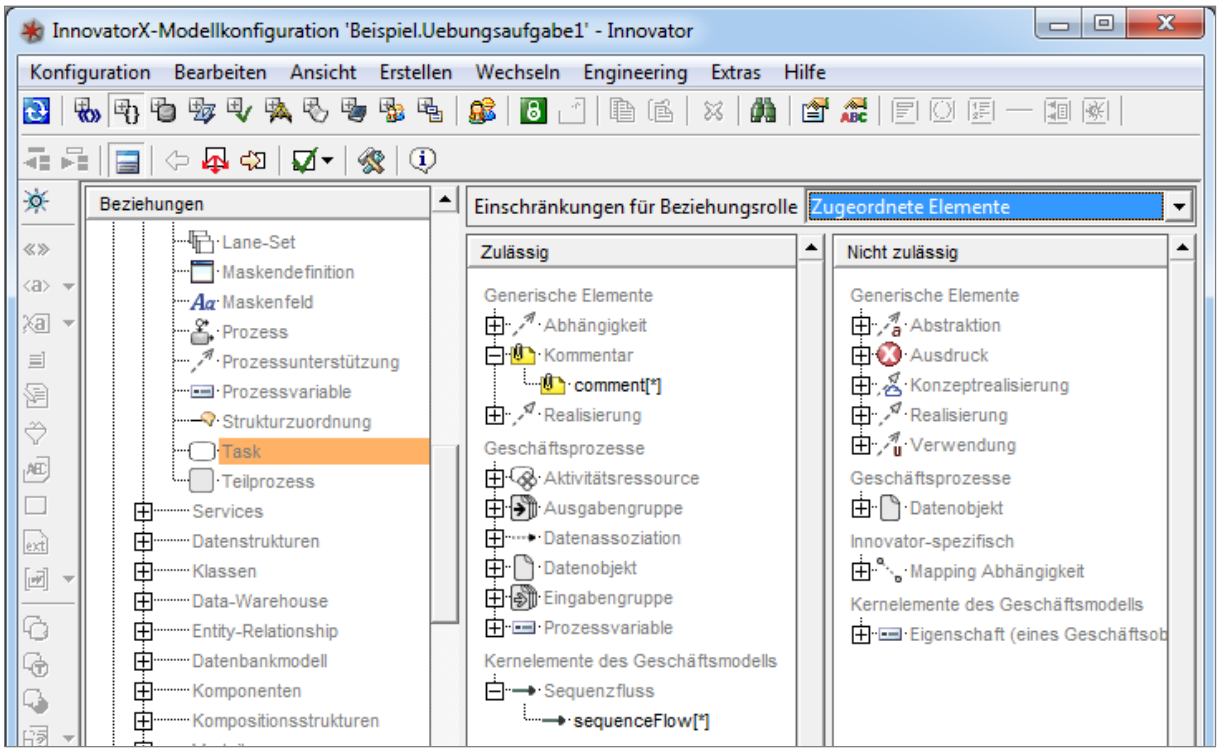


Abb. 5-9: Konfiguration der Beziehungen „Zugeordnete Elemente“ von Tasks.

5.1.2.2 AUTOMATISCHE PRÜFUNG (MODELLPRÜFUNGSASSISTENT)

Das Modellierungswerkzeug Innovator bietet die Möglichkeit, im Werkzeug modellierte Modelle zu prüfen (mittels Modellprüfungsassistenten). Dabei können unterschiedliche Aspekte betrachtet werden. Standardmäßig wird das Modellierungswerkzeug mit den in Abb. 5-10 aufgeführten drei Prüfroutinen ausgeliefert.

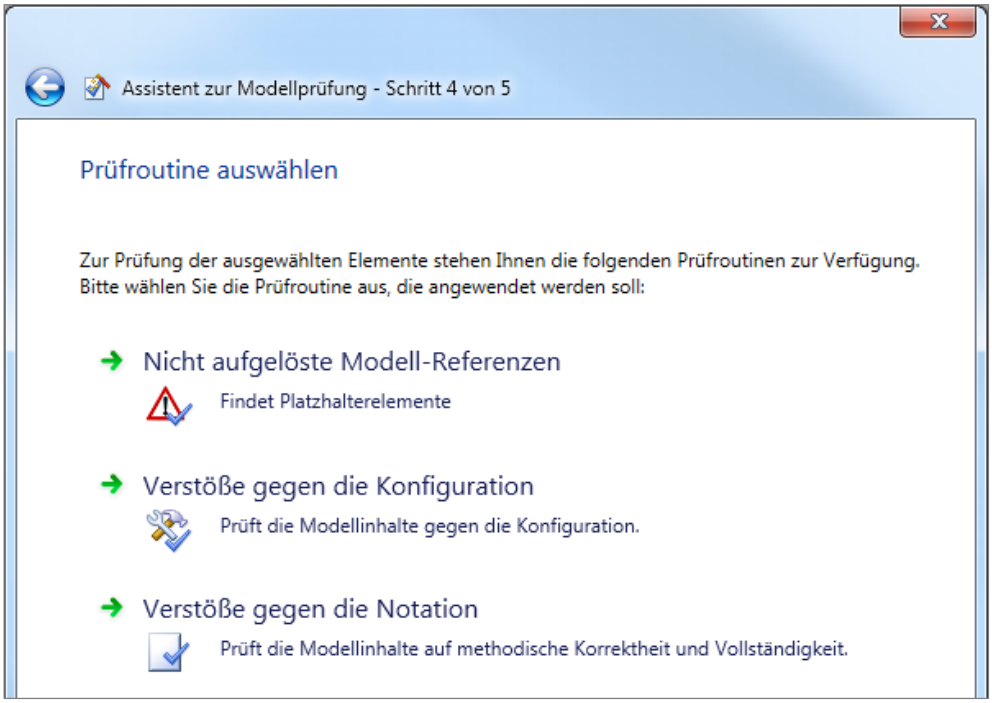


Abb. 5-10: Mögliche Prüfroutinen des Modellprüfungsassistenten.

Eine Prüfroutine ist eine Bündelung von Einzelprüfungen. Eine Prüfung eines Modells mittels einer solchen Prüfroutine, wie „Verstöße gegen die Notation“ (siehe Abb. 5-10, S. 109), kann für ein ganzes Modell oder unterschiedliche Teilbereiche eines Modells aufgerufen werden (siehe Tab. 5-2, S. 110). Nach Ausführung werden die Ergebnisse der Prüfung im Fenster Prüfergebnisse aufgelistet (siehe Abb. 5-11, S. 111). Zusätzlich können am rechten Rand in Diagrammen die Prüfergebnisse für die im jeweiligen Diagramm abgebildeten Modellelemente aufgeführt werden (siehe Abb. 5-12, S. 111).

Die Liste der Prüfergebnisse bietet standardmäßig Auskunft über die folgenden Punkte:

- Zu welchen Prüfgegenständen bzw. Modellbestandteilen hat die Prüfung eine Meldungen ergeben (identifiziert durch den Objektnamen, z.B. „Startoberfläche anzeigen“),
- Von welchem Modellbestandteiltyp ist der jeweilige Prüfgegenstand,
- Wie viele Meldungen vom Meldungstyp Information, Warnung oder Fehler hat die Prüfung zu dem jeweiligen Prüfgegenstand ergeben und
- Unter welchem Pfad in der Modellstruktur befindet sich der Prüfgegenstand.

Modellbereich	Erläuterung
Das aktive Diagramm	Geprüft wird das *„aktive Dokument (…), d.h. das im Dokumentenbereich der Anwendung aktuell sichtbare Diagramm bzw. die sichtbare Tabelle.“* (MID GmbH, 2014a)
Die aktuelle Selektion	*„Geprüft werden die Modellelemente, die beim Aufruf des Prüfassistenten im aktiven Fenster der Anwendung selektiert waren.“* (MID GmbH, 2014a)
Elemente relativ zur aktuellen Selektion	*„Basierend auf der aktuellen Selektion wird eine Suche durchgeführt. Alle Modellelemente, die bei dieser Suche gefunden werden, werden geprüft. Die Auswahl der Suche erfolgt im nächsten Schritt.“* (MID GmbH, 2014a)
Ein individueller Ausschnitt des Modells	Geprüft werden die Modellelemente, die vom Benutzer im anschließenden Dialogfenster ausgewählt werden. (MID GmbH, 2014a)
Das gesamte Modell	Geprüft werden *„Alle Modellelemente des gesamten Modells“* (MID GmbH, 2014a).

Tab. 5-2: Auswahlmöglichkeiten für die Prüfmenge (MID GmbH, 2014a).

Eine Meldung ist eine Auskunft darüber, dass ein Prüfobjekt eine Prüfung nicht erfüllt. Zu jeder Meldung wird die Identifikationsnummer der Prüfung angegeben („Prüfungsnummer“), welche die Meldung verursacht hat, sowie die Meldung selbst. In der Regel benennt die Meldung meist nochmals das Objekt selbst (durch die Typbezeichnung und den Namen des Objekts) und beschreibt anschließend den „Mangel“.

Zusätzlich zu den vordefinierten Prüfroutinen bietet Innovator die Möglichkeit eigene Prüfroutinen individuell aus Einzelprüfungen zusammenzustellen. Hierzu können in der Konfiguration ein eigenes Profil📖 erstellt und in diesem Profil jedem Innovator-Modellelementtyp eine Auswahl oder alle für dieses Modellelement möglichen der bereits definierten Einzelprüfungen zugeordnet werden.

Eine weitere Möglichkeit ist, Einzelprüfungen für Modelle in Java oder C# individuell zu programmieren und diese anschließend über eine Konfigurationsdatei im Werkzeug einzubinden. Diese Prüfungen werden im Modellprüfassistenten als zusätzliche Prüfmethoden mit dem jeweiligen Namen und der Beschreibung aufgeführt (siehe Abb. 5-15 „Einhaltung des eCH-0158 Standards prüfen“). In Abschnitt 5.2 wird anhand eines Beispiels die Java-Variante für die Programmierung und das Einbinden einer Einzelprüfung in den Innovator-Modellprüfungsassistenten erläutert.

5.1.2.3 METRIK-BERECHNUNG (ENGINEERING-AKTIONEN)

In dem Modellierungswerkzeug Innovator kann Java- oder C#-Code als sogenannte „Engineering-Aktion“ aufgerufen werden. Mit Hilfe von Funktionen kann so lesend und schreibend auf das Modell, Modellbestandteile und die Visualisierungen des Modells zugegriffen werden. Die jeweilige Klasse der Implementation wird in der Konfiguration des jeweiligen Profils📖 hinterlegt (siehe Abb. 5-16. S. 113) und kann dann im Werkzeug unter dem Reiter „Extras“ als Kommando aufgerufen werden (siehe Abb. 5-13, S. 113). Eine Engineering-Aktion (z.B. „Use Case Points

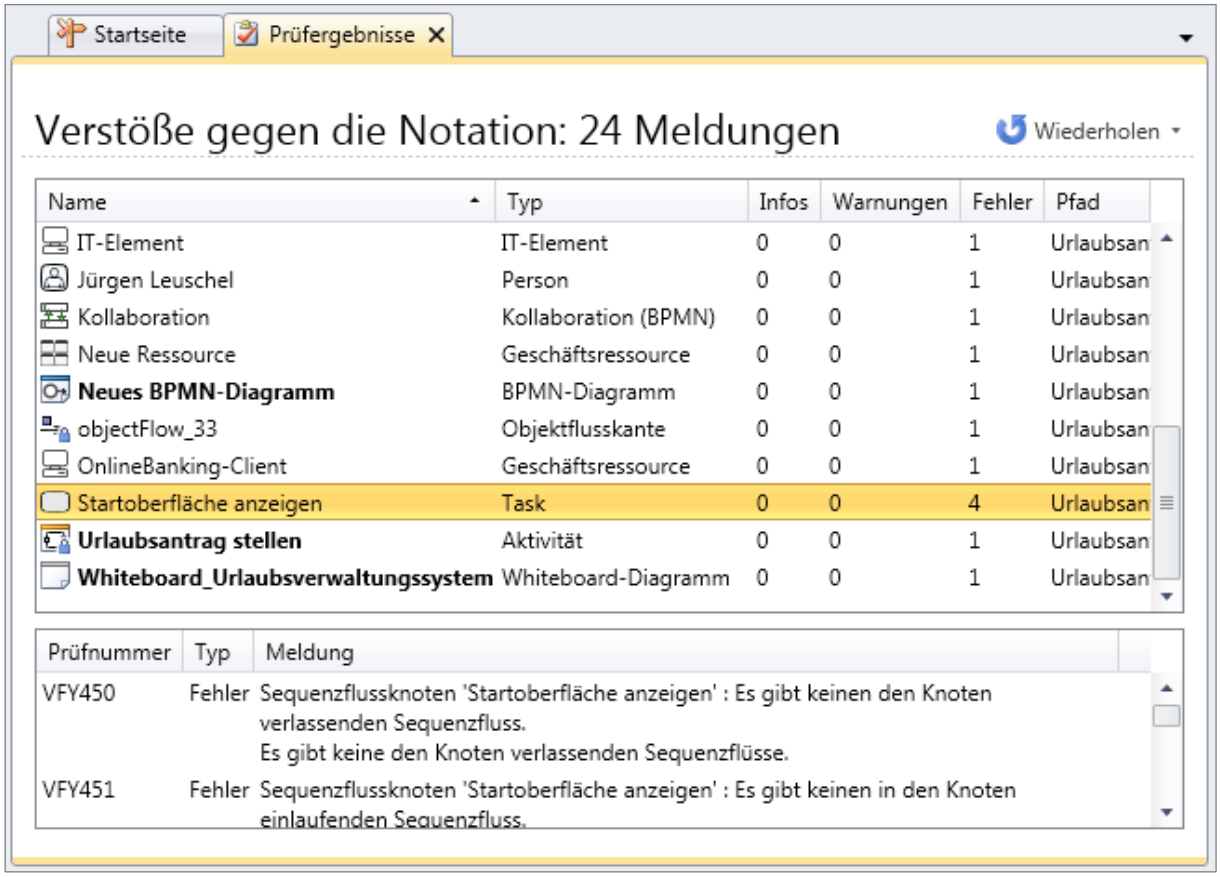


Abb. 5-11: Prüfergebnisse einer Prüfung (Bsp.: Prüfung „Verstöße gegen die Notation“).

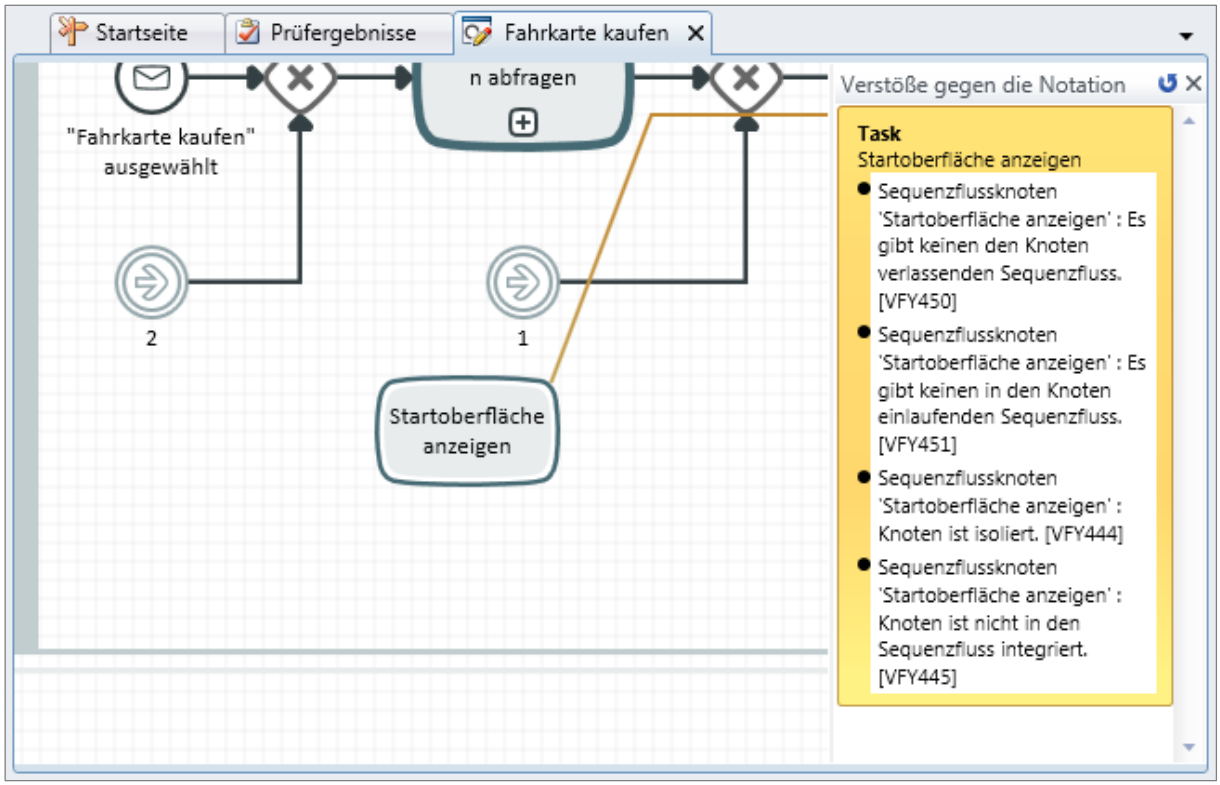


Abb. 5-12: Anzeige von vier Prüfergebnis-Meldungen in einem BPMN-Diagramm.

(UCP) berechnen“) kann aus mehreren Aktionen d.h. mehreren Programmen bestehen, die nacheinander ausgeführt werden. Ein Beispiel für eine solche Aktion sind die in Abb. 5-16 (S. 113) aufgeführten „Unadjusted Use Case Points (UUCP)“.

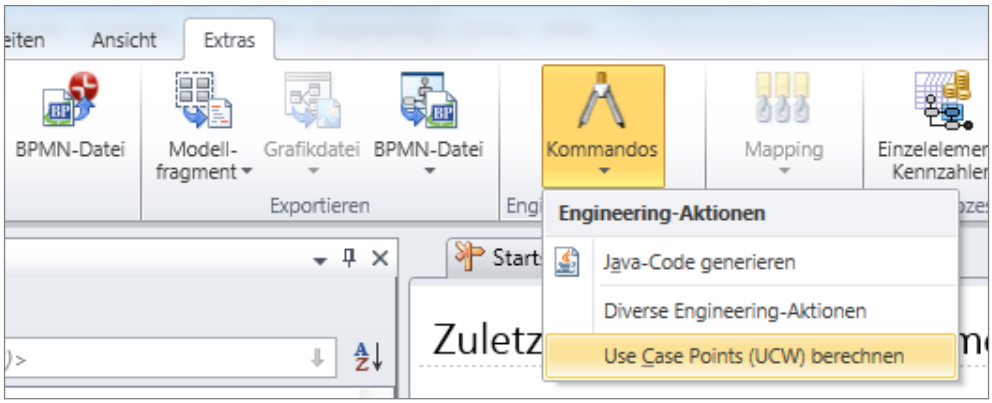


Abb. 5-13: Screenshot des Menüs „Engineering-Aktionen“.

5.2 Umsetzung der Qualitätsprüfung

In diesem Abschnitt wird beispielhaft gezeigt, wie in Innovator Prüfungen (d.h. Prüfaktionen) erstellt und in den Innovator-Modellprüfungsassistenten eingebunden werden können.

In Innovator setzt sich eine Prüfaktion aus Prüfgruppen zusammen. Prüfgruppen bestehen wiederum aus Einzelprüfungen (siehe Abb. 5-17, S. 116). Eigene Einzelprüfungen für den Innovator-Modellprüfassistenten können in Java oder C# erstellt werden. Für die im Rahmen dieser Arbeit erstellten Prüfungen wurde Java verwendet.[67] Im Folgenden wird anhand eines Beispiels das Erstellen einer solchen Einzelprüfung gezeigt. Anschließend wird beschrieben, wie die Einzelprüfung in eine Innovator-Prüfaktion eingebunden werden kann.

Eigene Einzelprüfungen erstellen

Als Beispiel implementieren wir die Konvention eCH0158-Diagramm-06 (siehe Abb. 5-14). Diese legt fest, dass in einem BPMN-Diagramm *„Maximal 9 – 15 Aktivitäten (…) pro Diagramm“* dargestellt werden sollen (eCH, 2013).

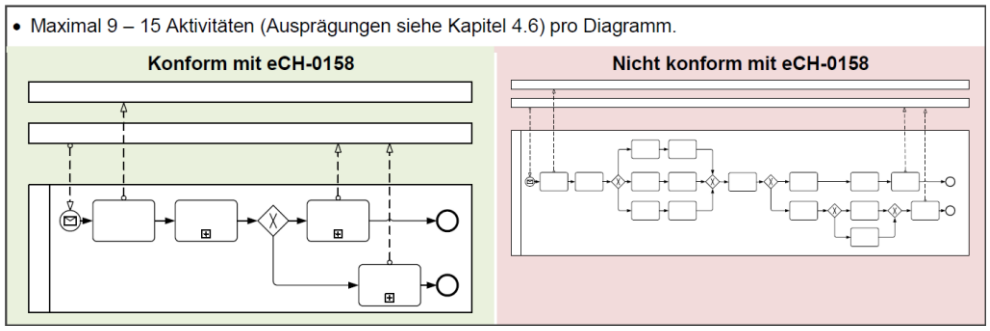


Abb. 5-14: Konvention „eCH0158-Diagramm-06“ (eCH, 2013).

[67] Die Dokumentation der Innovator Java API ist unter http://help.innovator.de/11.5/en_us/java/InnovatorAPI abrufbar.

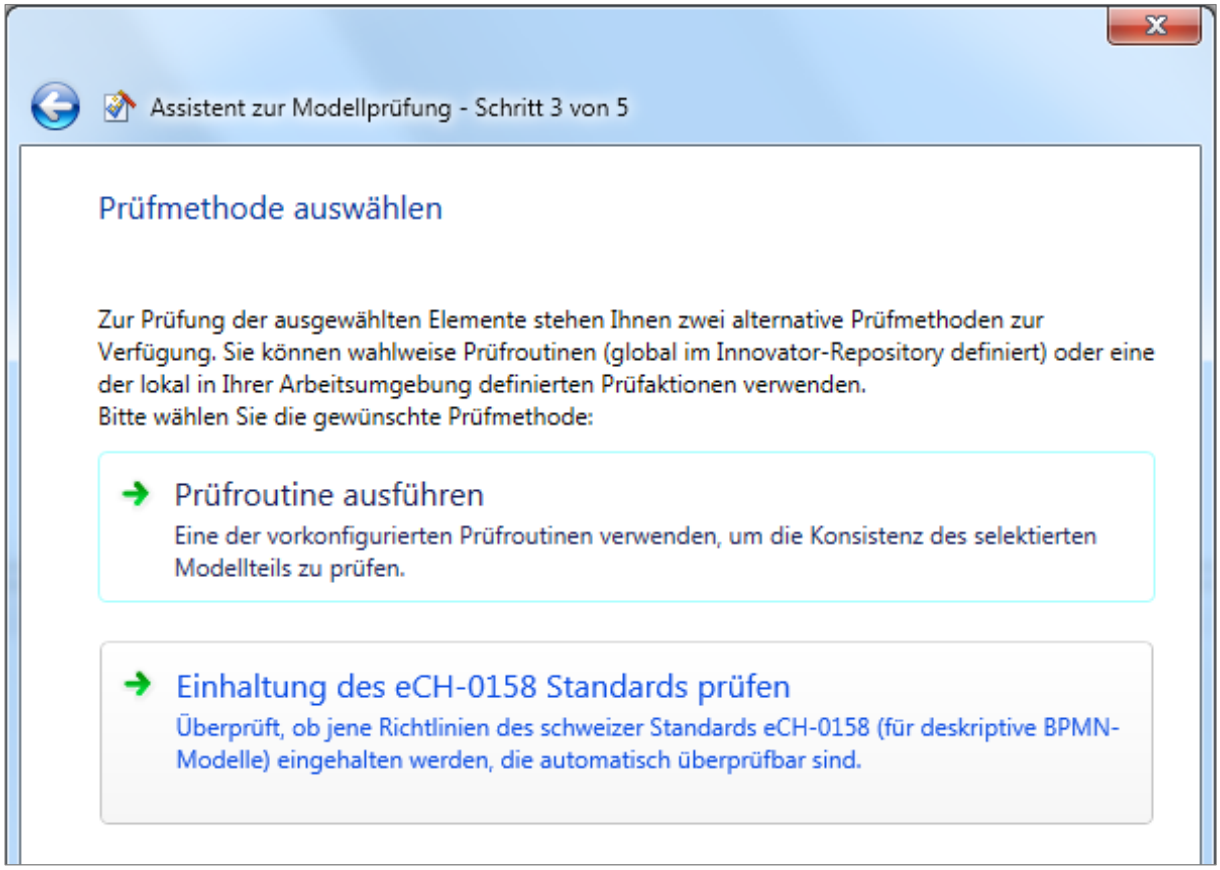


Abb. 5-15: Beispiel möglicher Prüfmethoden des Modellprüfungsassistent.

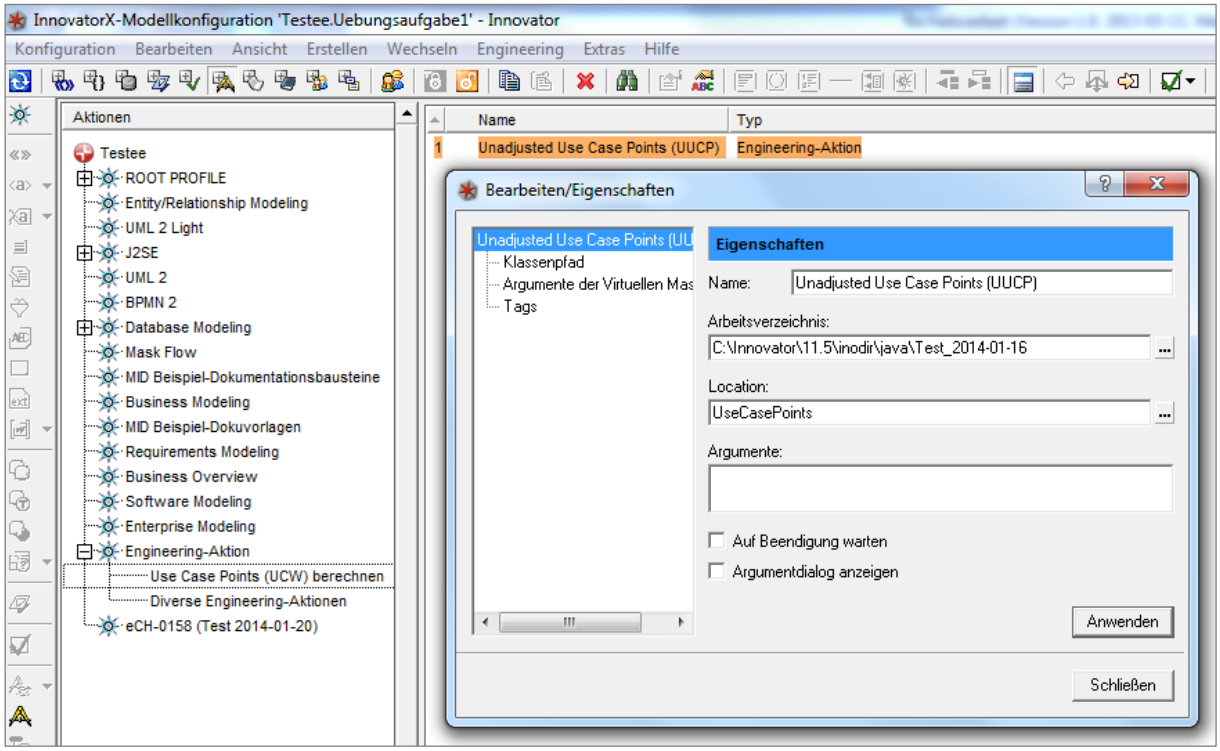


Abb. 5-16: Konfiguration einer Engineering-Aktion.

Unter dem Begriff „Aktivität“ wird nach eCH-0158 die Menge der Tasks und Aufrufaktivitäten verstanden. Aus der Forderung lassen sich die folgenden zwei Prüfungen ableiten, da sie zu unterschiedlichen Prüfungsmeldungen führen sollen, wie nachfolgend erklärt wird:

a) Werden in dem Diagramm größer gleich 15 Aktivitäten dargestellt?
b) Werden in dem Diagramm größer gleich 9 Aktivitäten dargestellt?

Diese Prüfungen führen zu unterschiedlichen Meldungen insofern, als dass wir im Fall von a) eine Prüfergebnis-Meldung vom Typ „Warnung“ zurückgeben wollen und im zweiten Fall eine „Information“.

Da die Prüfschnittstelle es erlaubt Parameter zu übergeben, können wir die beiden Prüfungen zu einer einzelnen Prüfung zusammenfassen. Wir übergeben der Java-Klasse als Parameter einen Zahlenwert, der das Maximum festlegt („numberOfActivities“ in Z. 27, Abb. 5-18, S. 117). Zusätzlich wird hier noch die die Information, ob das Maximum oder Minimum überprüft werden soll („minimumOrMaximum“ in Z. 26, Abb. 5-18, S. 117), als Parameter übergeben. Dies erlaubt eine Mindestanzahl an Aktivitäten für ein BPMN-Diagramm zu überprüfen, was jedoch in der eCH-0158 nicht gefordert wird. Des Weiteren wird der Methode das Objekt „testee“ zur Laufzeit übergeben (siehe Z. 34, Abb. 5-18, S. 117), auf dem die Prüfung aufgerufen wird. In diesem Fall gehen wir davon aus, dass die Prüfung auf einem BPMN-Diagramm (API-Name: „BPDia“) aufgerufen wurde, welches wir als erstes überprüfen (siehe Z. 42, in Abb. 5-18, S. 117). Die Funktion „isFaulty“, die wir für die Prüfung implementieren, hat als Rückgabewert „CheckResultState“, welcher die in Tab. 5-3 aufgeführten „Zustände“ einnehmen kann. „CheckResultState.*notApplicable*“ geben wir zurück, wenn das Objekt, auf dem die Prüfung aufgerufen wurde, nicht vom Typ BPMN-Diagramm ist oder wir einen der Parameter nicht korrekt auslesen können. Für den Fall, dass die Anzahl der im Diagramm dargestellten Aktivitäten innerhalb des Maximums bzw. Minimums liegt, geben wir „CheckResultState.*ok*“ zurück und „CheckResultState.*ok*“, wenn dies nicht der Fall ist (siehe Abb. 5-19, S. 118).

CheckResultState	**Erläuterung**
CheckResultState.*faulty*	Die Prüfung hat angeschlagen.
CheckResultState.*ok*	Die Prüfung hat nichts gefunden.
CheckResultState.*notApplicable*	Die Prüfung kann nicht angewendet werden.

Tab. 5-3: Mögliche „Zustände“ von „CheckResultState“.

Um herauszubekommen, wie viele Aktivitäten in dem Diagramm dargestellt werden, brauchen wir zunächst eine Liste aller im Diagramm dargestellten Prozesse (siehe Z. 45f, in Abb. 5-18, S. 117), denn wir wissen, dass Aktivitäten nach der Standard-Innovator-Konfiguration nur in Prozessen dargestellt werden dürfen. Anschließend sammeln wir in einer Liste alle Aktivitäten all dieser Prozesse (siehe Z. 47ff, in Abb. 5-18, S. 117). Die Länge der Liste „activities“ ist die im Diagramm dargestellte Anzahl an Aktivitäten. Zum Schluss muss nur noch diese Länge mit dem Maximum vergleichen werden und dementsprechend ein „faulty“ oder „ok“ zurückgegeben werden (siehe Z. 53-58, in Abb. 5-19, S. 118).

Um die Einzelprüfung nachfolgend in Innovator-Modellprüfassistenten einbinden zu können, muss die Einzelprüfung in einem entsprechenden Ordner kopiert werden. Hierfür kompilieren und kopieren wir an letzter Stelle die Prüfung über eine „Ant Build"-Aktion in das Innovator-Benutzerverzeichnis „C:\Innovator\11.5\ inodir\java\Verification\bin".

EIGENE EINZELPRÜFUNGEN EINBINDEN

Nachdem wir eine Java-Einzelprüfung für den Innovator-Modellprüfungsassistenten erstellt haben, wollen wir diese in den Assistenten einbinden, so dass wir die Prüfung im Modellierungswerkzeug ausführen können.

Eigene Prüfaktionen, wie unsere Prüfung zu eCH0158-Diagramm-06 aus dem vorherigen Abschnitt, können über das XML-Dokument „inoverify.xml" in Innovator eingebunden werden. Die jeweilige Einzelprüfung verweist dabei über den Klassennamen auf den Code (XML-Element: „Check"), welcher die Prüfung implementiert. Des Weiteren werden für unsere generische Einzelprüfung die in Tab. 5 4 aufgeführten Informationen im XML-Dokument hinterlegt.

In Abb. 8-16 (S. 156, in Abschnitt 8.2) ist die XML-Definition der Prüfung des BPMN-Diagramms auf maximal 15 enthaltene BPMN-Tasks als Ausschnitt der „inverify.xml"-Datei zu sehen. Über die Datei können mehrere generische Einzelprüfungen in das Modellierungswerkzeug eingebunden werden. Um weitere Prüfungen, die den eCH-0158-Standard betreffen zu bündeln, haben wir die Prüfaktion „Einhaltung des eCH-0158 Standards prüfen" definiert (XML-Element: „VerificationAction"). Unter dieser Aktion fassen wir mehrere Einzelprüfungen wiederum zu „fachlichen" Prüfgruppen zusammen, z.B. „Angemessene Modellierungssyntax überprüfen" (XML-Element: „VerificationGroup"). Für die fachliche Gruppierung verwenden wir die Qualitätskriterien unseres Qualitätssystems (siehe Abschnitt 4.1.1). Unterhalb der jeweiligen Prüfgruppe werden die zugehörigen Prüfungen aufgeführt (XML-Element: „Singles"). Für eine generische Einzelprüfung werden die in Tab. 5-4 (S. 118) aufgeführten Informationen im XML-Dokument hinterlegt.

Führt der Benutzer später die Prüfaktion „Einhaltung des eCH-0158 Standards prüfen" aus, so werden alle im XML-Dokument zu der Aktion hinterlegten Einzelprüfungen nach den Prüfungsgruppen gruppiert im Modellprüfungsassistenten angezeigt (siehe Abb. 5-20, S. 119), wie in der XML-Datei definiert. Im Diagramm „eCH0158-Diagramm-06" haben wir das Beispiel aus dem eCH-Standard (siehe Abb. 5-14, S. 112) nachmodelliert und unsere Prüfungen „Maximal 15 Aktivitäten pro BPMN-Diagramm." und „Maximal 9 Aktivitäten pro BPMN-Diagramm." für dieses Diagramm auswerten lassen (siehe Abb. 5-21, S. 119). Es ist zu sehen, dass sowohl die Prüfmeldung, dass mehr als 15 Aktivitäten in Diagramm dargestellt sind als Warnung und mehr als 9 als Fehler ausgegeben werden. Dass an dieser Stelle „Fehler" als Kategorie anstelle einer „Information" steht, ist ein momentaner Bug des Innovator-Modelprüfassistenten.

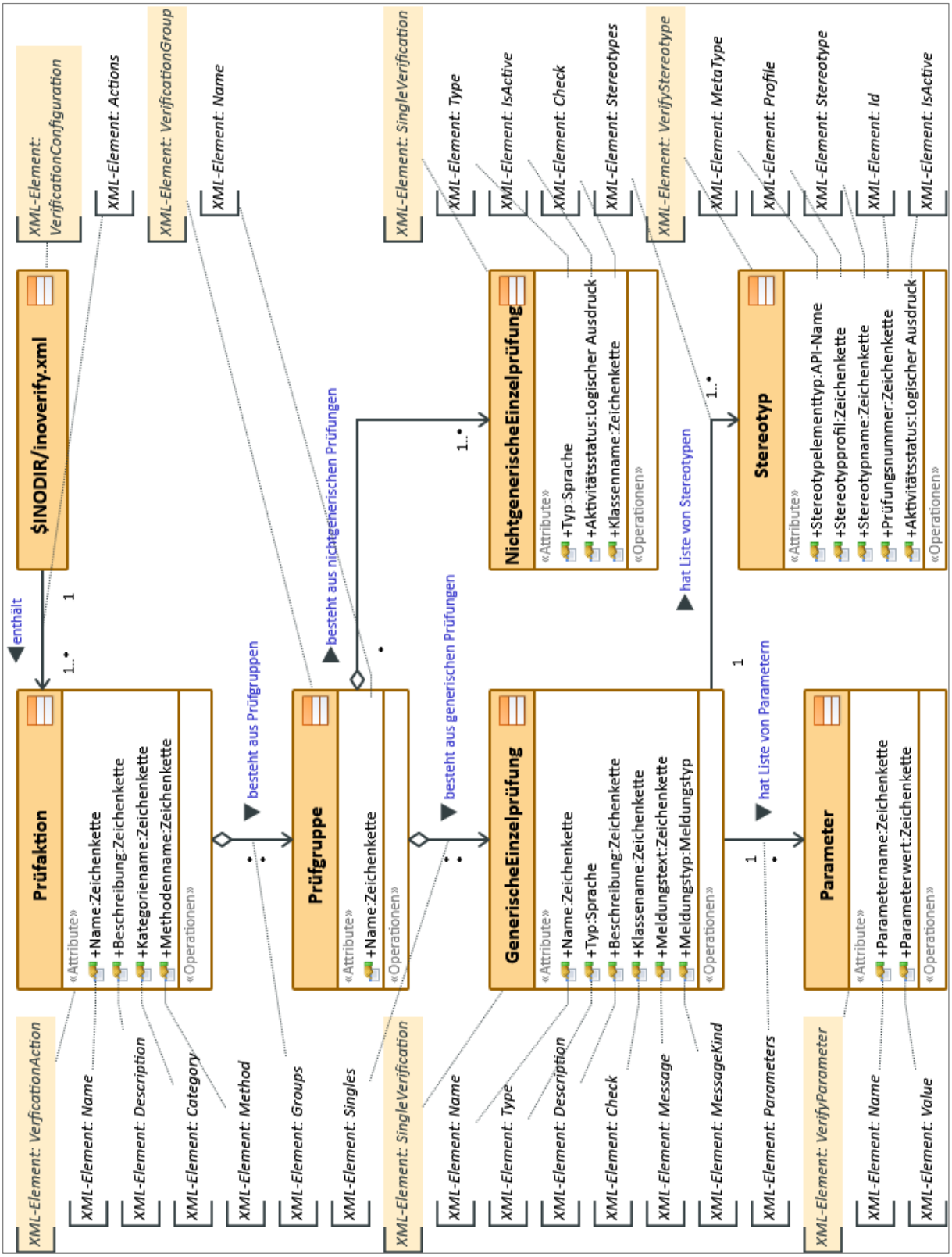


Abb. 5-17: Klassendiagramm zum Aufbau von Prüfaktionen und ihrer XML-Bezeichnung.

```java
package de.mid.engineering.innovator.verify;

import java.util.LinkedList;
import java.util.List;

import de.mid.engineering.innovator.pruefmanager.IFConfigProvider;
import de.mid.engineering.innovator.pruefmanager.check.GenericCheckBase;
import de.mid.engineering.innovator.pruefmanager.view.IFModelCheckTestee;
import de.mid.innovator.srv2api.icw2bp.BPActivityNode;
import de.mid.innovator.srv2api.icw2bp.BPProcess;
import de.mid.innovator.srv2api.icw2bpdia.BPDia;
import de.mid.innovator.srv2api.icw2elem.ELNamedElement;
import de.mid.innovator.srv2api.icw2meta.ADModel;

/**
 * This function checks whether the number of activities shown in the given
 * diagram complies with the given maximum (or minimum, optional depending on
 * the value of minimumOrMaximum) for the number of activities
 * (numberOfActivities).
 *
 * @author Meike Weber (meweber)
 *
 */
public class Check_BPMN_BPDia_NumberOfActivities extends GenericCheckBase {

    private static final String PARAM_MINIMUM_OR_MAXIMUM = "minimumOrMaximum";
    private static final String PARAM_NUMBER_OF_ACTIVITIES = "numberOfActivities";

    public Check_BPMN_BPDia_NumberOfActivities(ADModel model,
            IFConfigProvider cfgProvider) throws Exception {
        super(model, cfgProvider);
    }


    protected CheckResultState isFaulty(IFModelCheckTestee testee)
            throws Exception {

        ELNamedElement element = testee.getNamedElement();
        CheckResultState crs = CheckResultState.notApplicable;
        String type = getParameterValue(PARAM_MINIMUM_OR_MAXIMUM);
        int numberOfActivities = getParameterIntegerValue(PARAM_NUMBER_OF_ACTIVITIES);

        if (element instanceof BPDia) {
            BPDia diagram = (BPDia) element;
            crs = CheckResultState.faulty;
            List<BPProcess> processes = diagram
                    .getModelElementOfTransitiveOwnedPresentationElement(BPProcess.class);
            List<BPActivityNode> activities = new LinkedList<BPActivityNode>();

            for (BPProcess process : processes) {
                activities.addAll(process
                        .getFlowElementTransitiveDown(BPActivityNode.class));
            }
```

Abb. 5-18: Java-Prüfung „Anzahl Aktivitäten pro BPMN-Diagramm“ (1/2).

```
53              if (type.equals("maximum")) {
54                  if (activities.size() <= numberOfActivities) {
55                      return crs = CheckResultState.ok;
56                  } else {
57                      return crs = CheckResultState.faulty;
58                  }
59
60              } else {
61                  if (type.equals("minimum")) {
62                      if (activities.size() >= numberOfActivities) {
63                          return crs = CheckResultState.ok;
64                      } else {
65                          return crs = CheckResultState.faulty;
66                      }
67                  } else {
68                      return crs = CheckResultState.notApplicable;
69                  }
70              }
71          }
72          return crs;
73      }
74
75  }
```

Abb. 5-19: Java-Prüfung „Anzahl Aktivitäten pro BPMN-Diagramm“ (2/2).

XML-Element	Erläuterung
Name	Unter diesem Namen wird die Einzelprüfung im Modellprüfassistenten aufgeführt.
Type	Gibt an, ob es sich um eine Java oder C#-Prüfung handelt.
Description	Erscheint als Erläuterung für die Einzelprüfung im Prüfungsassistenten.
Check	Gibt den qualifizierten Name der implementierenden Prüfungsklasse an.
Message	Erscheint als Meldungstext für die Einzelprüfung bei dem Prüfergebnis.
MessageKind	Hier wird der Meldungstyp (Fehler, Warnung oder Information) angegeben, der im Falle eines „faulty“-Rückgabewerts für die Einzelprüfung im Prüfergebnis angegeben wird.
Parameters	Gibt Parameter (XML-Element: „Parameter“), identifiziert durch ihren Namen (XML-Element: „Name“) mit ihrem zugehörigen Wert (XML-Element: „Value“) an.
Stereotypes	Gibt an, auf welche Stereotypen (d.h. Modellelementtypen) die Einzelprüfung aufgerufen werden soll: Hierzu wird für jeden Stereotyp (XML-Element: „VerifyStereotype“) der API-Name des Stereotyps (XML-Element: „Metatype“), das Konfigurations-Profil, in dem der Stereotyp definiert ist (XML-Element: „Profile“) und der Namen des Stereotyps (XML-Element: „Stereotype“) angeben. Ebenfalls kann eine Prüfungsidentifikationsnummer (XML-Element: „Id“) frei vergeben werden und es wird angegeben, „Ob die Einzelprüfung für diesen Stereotyp aktiv ist, d.h. ausgeführt werden soll.“ (MID GmbH, 2013b) (XML-Element: „IsActive“).

Tab. 5-4: XML-Elemente für die Definition Generischer Einzelprüfungen (MID GmbH, 2013b).

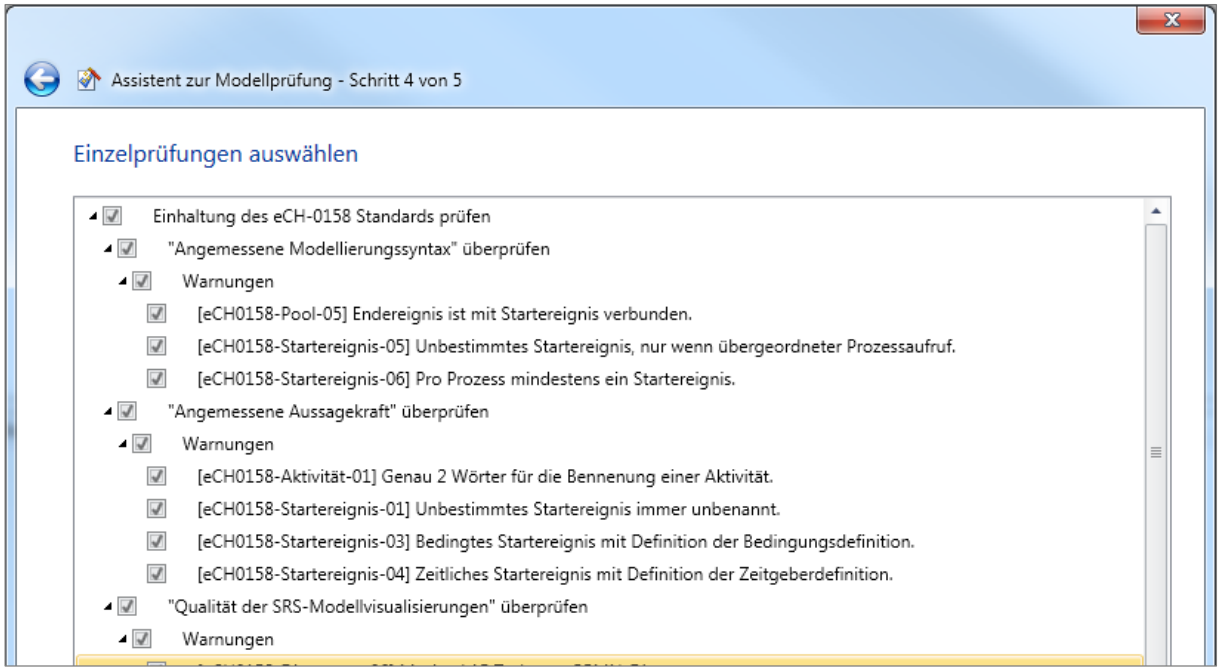


Abb. 5-20: Selbsterstellte Einzelprüfungen im Modellprüfungsassistenten.

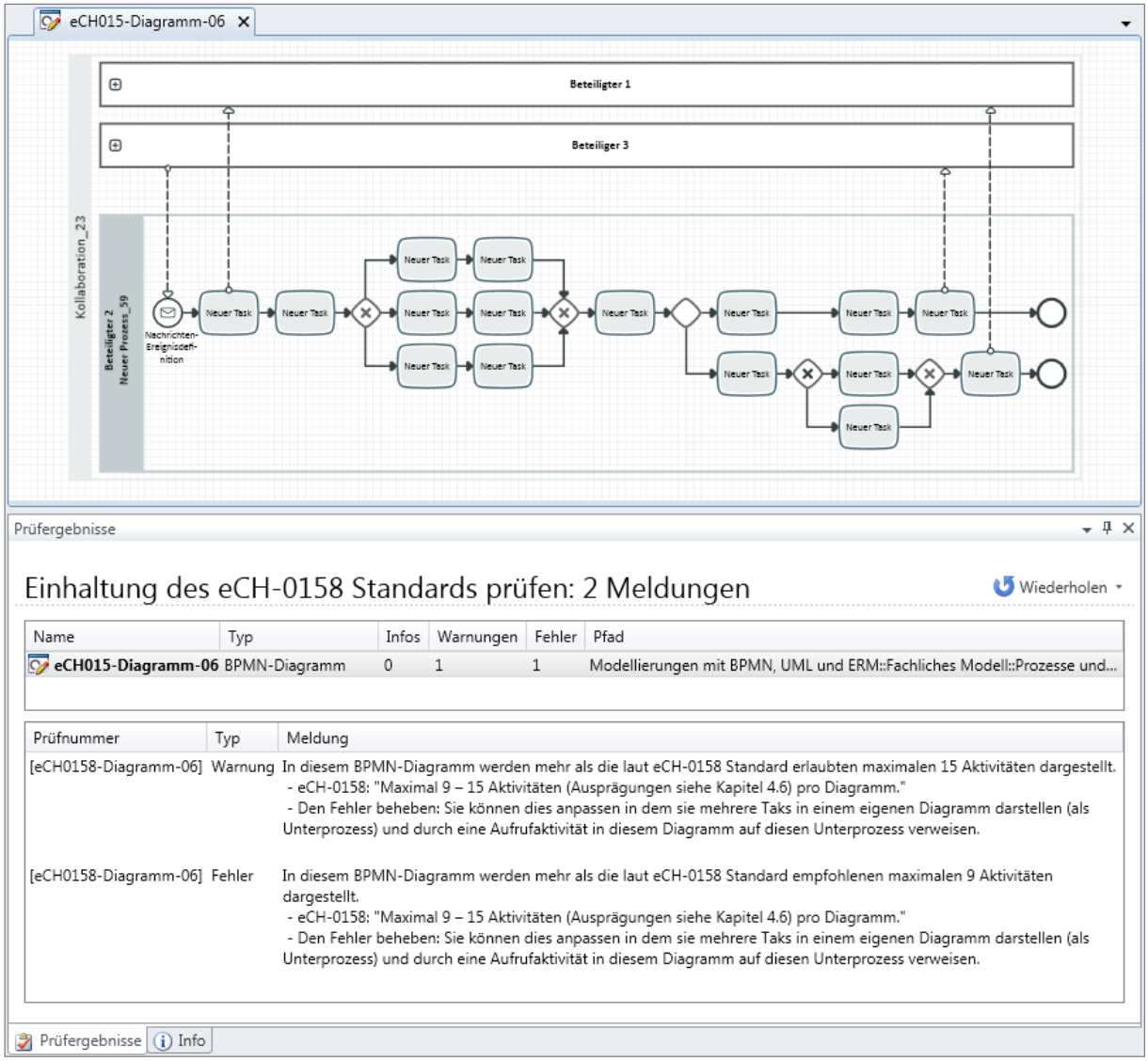


Abb. 5-21: Prüfung „eCH0158-Diagramm-06“.

5.3 Umsetzung der Qualitätsmessung

Für die beispielhafte Umsetzung einer Metrik in Innovator nehmen wir die Use Case Points (UCP), konkret haben wir die Berechnung der Unadjusted Use Case Points (UCCW) implementiert (siehe Abb. 5-22, S. 123). Des Weiteren wurden im Rahmen der Bachelorarbeit die Berechnung der folgenden Metriken programmiert (siehe Abb. 5-23, S. 123; Implementierung: Siehe Abb. 8-19 und Abb. 8-20, S. 159f., in Abschnitt 8.2):

- #Anforderungen ohne Quelle,
- #Anforderungen ohne Stakeholder,
- #Anforderungen ohne Priorität und
- #Spezifizierte Anforderungen ÷ #Anforderungen.

Die Vermessung des SRS-Modells kann über eine Innovator-Erweiterung einer sogenannten „Engineering-Aktion" realisiert werden (siehe Abschnitt 5.1.2). Unsere Berechnung der UUCW haben wir in Java implementiert. Abb. 5-24 (S. 124) bis Abb. 5-29 (S. 128) zeigt den Code, welchen wir nachfolgend erläutern.

Implementierung der Berechnung von UUCP[68]

Die Berechnung und Ausgabe der UCCW geschieht in der „run"-Methode (siehe Abb. 5-25, S. 125). Die Funktion erhält eine Liste von Objekten („sel"), die der Modellierer beim Aufruf der Engineering-Aktion selektiert hatte.[69] Für alle in der Liste enthaltenen Modelle („MEModel") wollen wir die UUCP berechnen. Um die UUCP zu berechnen, müssen wir die Summanden UUCW und UAW ermitteln.

Um die UUCW berechnen zu können, benötigen wir zu jedem im Modell enthaltenen Anwendungsfall (engl.: Use Case) die Anzahl seiner Transaktionen (bzw. „Schritte" nach Frohnhoff 2009). Das Verhalten und somit die „Schritte" eines Anwendungsfalls werden bei modellbasierten SRS mit Aktivitäts- oder BPMN-Diagrammen beschrieben. Da in dieser Arbeit BPMN zur Verhaltensmodellierung untersucht wird, haben wir uns entschieden, die Transaktionen-Anzahl eines Anwendungsfalls aus den hinterlegten BPMN-Kollaborationen herzuleiten. Wir definieren, dass die Anzahl der Schritte äquivalent zu der Anzahl der in den Prozessen der Kollaborationsbeteiligten enthaltenen Anzahl an Aktivitäten ist. Um diese Anzahl für jeden Anwendungsfall zu bestimmen, lassen wir uns im ersten Schritt eine Liste aller im Modell enthaltenen Anwendungsfälle ausgeben („useCases"). Von jedem Anwendungsfall ermitteln wir im zweiten Schritt die Anzahl der Schritte, d.h. die Anzahl der Aktivitäten der in den zugehörigen Kollaborationen enthaltenen Prozesse („getListOfNumberOfTransactions(useCases)", siehe Abb. 5-26, S. 126). Die Liste „numberUCsPerCategory" enthält an erster Stelle die Anzahl der Anwendungsfälle mit weniger vier Schritten, an zweiter Stelle jene mit vier bis sieben und an dritter mit mehr als 7 Transaktionen (siehe Abb. 5-26, S. 126). In der Methode „calculateUucw" (siehe Abb. 5-29, S. 128) wird die

[68] Siehe Abschnitt 3.3.

[69] Um die UUCP berechnen zu können, muss er den Modellknotenpunkt des fachlochen Modells oder einen oberhalb liegenden ausgewählt haben.

UCCW berechnet, durch die Gewichtung der Anwendungsfall-Anzahl der jeweiligen Kategorie.

Für die UAW wird in der „run“-Methode (siehe Abb. 5-25, S. 125) die Anzahl der im Modell enthaltenen Akteure abgefragt („numberOfActors“). Da in der Innovator-Standardkonfiguration nicht zwischen den drei Kategorien für Akteure nach Karner unterschieden wird, gehen wir bei der Gewichtung davon aus, dass jeder Akteur ein Mensch ist. Somit multiplizieren wir die Anzahl der enthaltenen Akteure mit drei und erhalten so unsere UAW.

Die UUCP setzt sich zusammen aus der UUCW und UAW, welche miteinander addiert werden. Das Ergebnis sowie Zusatzinformationen, wie die Anzahl der im Model enthaltenen Anwendungsfälle, lassen wir über die „Konsolenausgabe“ in Innovator ausgeben (siehe Abb. 5-25, S. 125; Ausgabe: Siehe Abb. 5-22, 123).

AUSBLICK DER BERECHNUNG VON UCP

Bei der bisherigen Implementierung haben wir Anwendungsfälle mit keiner zugeordneten Kollaboration als einfachen Anwendungsfall gewertet (d.h. mit 5 Punkten). Eine andere Möglichkeit ist diese Anwendungsfälle aus der Zählung rauszulassen und eine Warnung oder Information auszugeben, welche Anwendungsfälle nicht in die Berechnung mit einbezogen werden konnten und warum, nämlich weil ihr Verhalten nicht durch mindestens eine BPMN-Kollaboration beschrieben wird und daher nicht die zugehörigen Transaktionsschritte für die Gewichtung des Anwendungsfalls berechnet werden können. Das Gleiche gilt für Anwendungsfälle, denen zwar eine Kollaboration zugeordnet ist, deren Prozesse jedoch keine Prozessschritte enthalten.

Sollte Wert darauf gelegt werden, im Modell unterscheiden bzw. festlegen zu können, welche BPMN-Aktivitäten als Transaktionen gezählt werden sollen und welche nicht, kann für Aktivitäten ein Eigenschaftsfeld eingeführt werden, wo eben dieses festlegt werden kann.

Des Weiteren wird in der momentanen Implementierung jeder Akteur als Mensch gewichtet (d.h. mit der Komplexität „komplex“). Die Einteilung der Akteure in „einfach“, „mittel“ und „komplex“ kann in Innovator über ein Label realisiert werden. Ist dies der Fall, kann der Wert des Eigenschaftsfelds in der Engineering-Aktion abgerufen werden und genauso wie die UUCP jeder Akteur nach seiner Ausprägung gewichtet und die UAW berechnet werden.

5.4 ZUSAMMENFASSUNG

In diesem Abschnitt haben wir das Modellierungswerkzeug Innovator und drei Ansätze der Qualitätssicherung in Innovator vorgestellt. In dem Werkzeug können konstruktiv durch die Profil-Konfiguration die Qualität des Modells und der Modellvisualisierungen gesichert werden, analytisch durch Prüfungen mittels des Prüfungsassistenten sowie über Metriken, die Aussagen über die Größe und die Qualität treffen können. An dem Beispiel der Konvention „eCH0158-Diagramm-06“ wurde gezeigt, wie Prüfungen für den Modellprüfungsassistenten des Innovators implementiert und eingebunden werden können. Im letzten Abschnitt haben wir demonstriert, wie Funktionen zur Metrikberechnung über die sogenannte „Engineering-Aktionen“ in Innovator eingebunden werden können.

Somit wurde aufgezeigt, wie die vorgeschlagenen Prüfungen und Metriken aus Abschnitt 4 umgesetzt werden können. Vorausgesetzt, es wurden im Softwareentwicklungsprojekt Qualitätsniveaus definiert sowie Schwellwerte für jede Metrik (siehe Abschnitt 4.2.4), so können die einzelnen Maßzahlen in einem zweiten Schritt bewertet werden und abgeleitet aus der Bewertung der Maßzahlen auch die Kriterien und anschließend die Faktoren und somit die *Qualität des Modells* und *der Modellvisualisierungen* einer modellbasierten SRS. Eine solche automatische Bewertung wurde für das Modellierungswerkzeug bisher noch nicht umgesetzt.

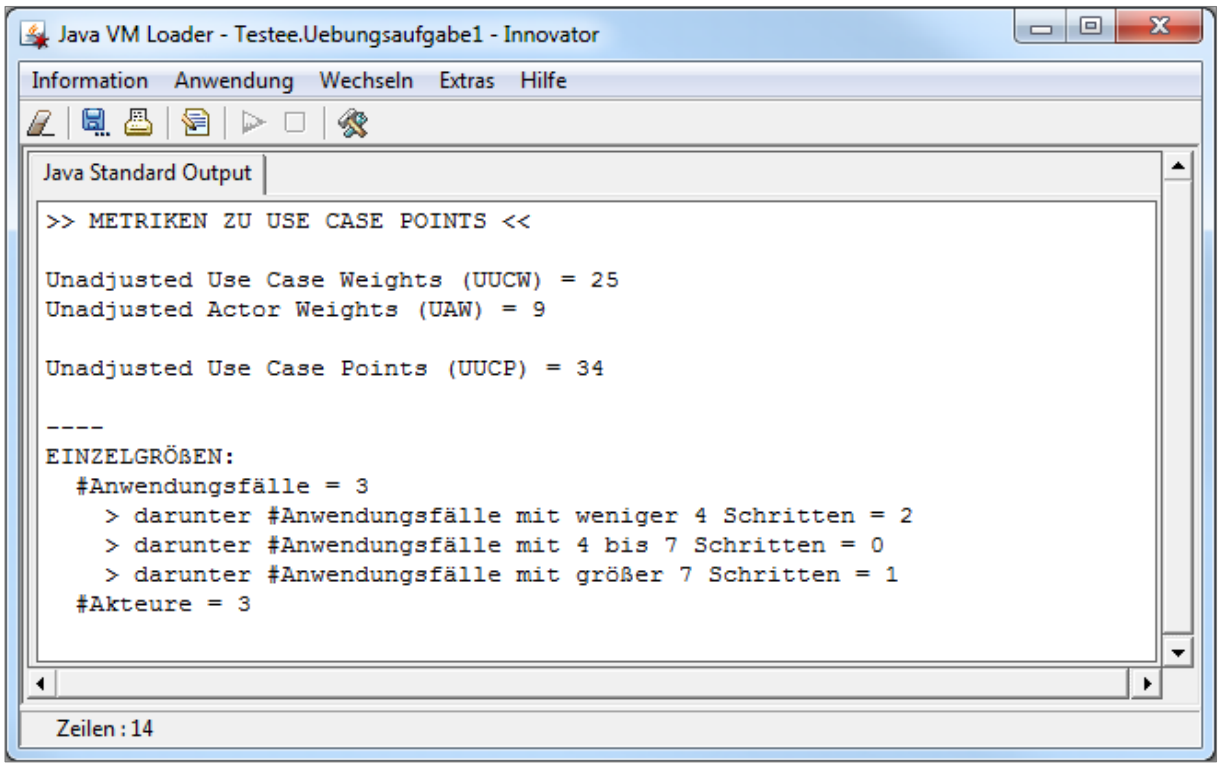


Abb. 5-22: Beispiel für die Auswertung von UUCP.

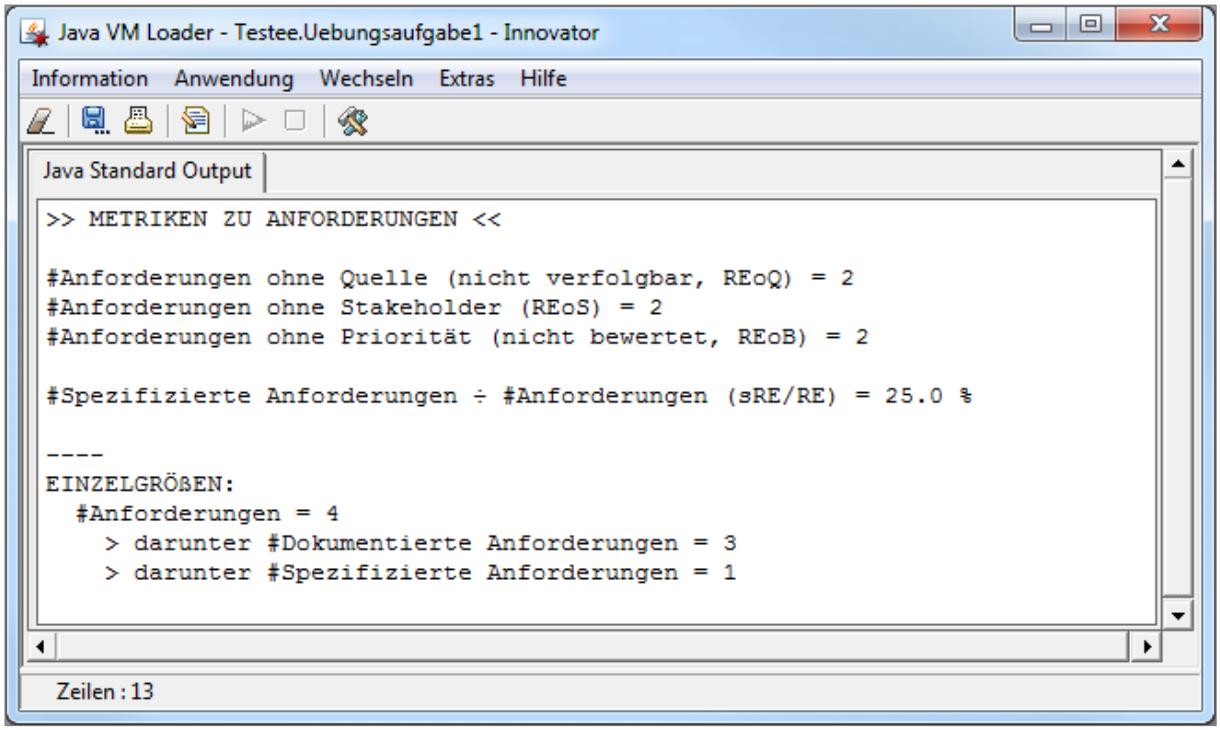


Abb. 5-23: Beispiel für die Auswertung von Metriken bzgl. Anforderungen.

```
import java.util.LinkedList;
import java.util.List;

import de.mid.innovator.client.InoClientContext;
import de.mid.innovator.net.InoNetException;
import de.mid.innovator.srv.SrvErrorException;
import de.mid.innovator.srv2api.icw2bp.BPActivityNode;
import de.mid.innovator.srv2api.icw2bp.BPCollaboration;
import de.mid.innovator.srv2api.icw2bp.BPParticipant;
import de.mid.innovator.srv2api.icw2bp.BPProcess;
import de.mid.innovator.srv2api.icw2bp.BPUseCaseInteractionRel;
import de.mid.innovator.srv2api.icw2br.BRResourceCL;
import de.mid.innovator.srv2api.icw2class.CLUseCase;
import de.mid.innovator.srv2api.icw2elem.ELContainerAble;
import de.mid.innovator.srv2api.icw2elem.ELElement;
import de.mid.innovator.srv2api.icw2model.MEModel;
import de.mid.innovator.ui.InnovatorApplication;
import de.mid.innovator.ui.InnovatorApplicationDefault;
import de.mid.innovator.ui.Property;
import de.mid.innovator.ui.PropertyCfg;
import de.mid.innovator.util.InoNlsException;

public class UseCasePoints extends InnovatorApplicationDefault implements
        InnovatorApplication {

    public String Usage() {
        return null;
    }

    private final static String PARAMETER_1 = "parameter_1";

    private static Property[] argumentsCfg = { new PropertyCfg<String>(
            PARAMETER_1, "Der erste Parameter.") };

    public static void main(String[] args) throws Exception {
        InnovatorApplicationDefault.create(UseCasePoints.class, argumentsCfg,
                args);
    }
```

Abb. 5-24: Java-Code zur Ermittlung der UUCP (1/6).

```
/**
 * This function gets a list of objects, that have been selected by the user
 * in Innovator. If one of the objects is an MEModel, the value of
 * Unadjusted Use Case Points (UUCP) will be calculated for the model. The
 * value will be shown in the info-window of Innovator.
 */
public void run() throws InoNlsException {

    int uucp = 0;
    int uucw = 0;
    int uaw = 0;

    int numberOfUseCases = 0;
    int numberOfActors = 0;

    List<Integer> numberUCsPerCategory = new LinkedList<Integer>();
    List<ELElement> sel = InoClientContext.getInstance().getSelection();

    for (ELElement element : sel) {

        if (element instanceof MEModel) {

            MEModel model = (MEModel) element;
            List<CLUseCase> useCases = model
                    .getOwnedElementTransitiveDown(CLUseCase.class);

            if (useCases != null) {
                numberUCsPerCategory = getListOfNumberOfTransactions(useCases);
                uucw = calculateUucw(numberUCsPerCategory);
                numberOfUseCases = useCases.size();
            }

            List<BRResourceCL> actors = model
                    .getOwnedElementTransitiveDown(BRResourceCL.class);
            numberOfActors = actors.size();
            uaw = (numberOfActors * 3);
            uucp = uucw + uaw;

            // AUSGABE
            System.out.println(">> METRIKEN ZU USE CASE POINTS <<");
            System.out.println();
            System.out.println("Unadjusted Use Case Weights (UUCW) = "
                    + uucw);
            System.out.println("Unadjusted Actor Weights (UAW) = " + uaw);
            System.out.println();
            System.out.println("Unadjusted Use Case Points (UUCP) = "
                    + uucp);
            System.out.println();
            System.out.println("----");
            System.out.println("EINZELGRÖßEN: ");
            System.out.println("  #Anwendungsfälle = " + numberOfUseCases);
            System.out
                    .println("    > darunter #Anwendungsfälle mit weniger 4"
                            + " Schritten = " + numberUCsPerCategory.get(0));
            System.out
                    .println("    > darunter #Anwendungsfälle mit 4 bis 7"
                            + " Schritten = " + numberUCsPerCategory.get(1));
            System.out
                    .println("    > darunter #Anwendungsfälle mit größer 7"
                            + " Schritten = " + numberUCsPerCategory.get(2));
            System.out.println("  #Akteure = " + numberOfActors);

        } else {
            System.out
                    .println("Fehler: Das selektierte Element ist kein Modell.");
        }
    }
}
```

Abb. 5-25: Java-Code zur Ermittlung der UUCP (2/6).

```java
/**
 * Gets the number of transaction for each use case contained in the list
 * and returns a list: @0: the number of use cases with less than 4
 * transactions @1: the number of use cases with between 4 and 7
 * transactions and @2: the number of use cases with more than 7
 * transactions.
 *
 * @param useCases
 *            A list of use cases.
 * @return A list of how many use cases have less than 4, between 4 and 7 or
 *         more than 7 transactions.
 * @throws InoNetException
 * @throws SrvErrorException
 */
private List<Integer> getListOfNumberOfTransactions(List<CLUseCase> useCases)
        throws InoNetException, SrvErrorException {

    List<Integer> numberOfTransactions = new LinkedList<Integer>();
    int lessThan4 = 0;
    int between4and7 = 0;
    int moreThan7 = 0;
    for (CLUseCase useCase : useCases) {
        List<BPProcess> processes = getProcessOfUseCase(useCase);

        if (processes != null) {
            int number = getNumberOfTransactions(processes);

            if (number < 4) {
                lessThan4++;
            } else if (number > 7) {
                moreThan7++;
            } else {
                between4and7++;
            }
        }
    }
    numberOfTransactions.add(lessThan4);
    numberOfTransactions.add(between4and7);
    numberOfTransactions.add(moreThan7);

    return numberOfTransactions;
}
```

Abb. 5-26: Java-Code zur Ermittlung der UUCP (3/6).

```
    /**
     * Gets all BPMN-processes that are included in a BPMN-collaboration, that
     * describe the behaviour of the given use case.
     *
     * @param useCase
     *            A use case.
     * @return A list of processes that describe the use case.
     * @throws InoNetException
     * @throws SrvErrorException
     */
    private List<BPProcess> getProcessOfUseCase(CLUseCase useCase)
            throws InoNetException, SrvErrorException {

        List<BPProcess> bpps = new LinkedList<BPProcess>();
        List<BPUseCaseInteractionRel> bpucirs = useCase
                .getIncomingInteractionUseCaseRel(null);

        if (bpucirs != null) {
            for (BPUseCaseInteractionRel bpucir : bpucirs) {
                ELContainerAble source = bpucir.getSource();

                if (source instanceof BPCollaboration) {
                    BPCollaboration bpc = (BPCollaboration) source;
                    List<BPParticipant> participants = bpc.getParticipant(null);

                    if (participants != null) {
                        for (BPParticipant participant : participants) {
                            BPProcess process = participant.getProcess();
                            if (process != null) {
                                bpps.add(process);
                            }
                        }
                    }
                }
            }
        }
        return bpps;
    }
```

Abb. 5-27: Java-Code zur Ermittlung der UUCP (4/6).

```
    /**
     * Calculates the number of transaction of the processes (i.e. the number of
     * BPMN-activities owned by the processes).
     *
     * @param processes
     *            A list of processes.
     * @return The number of transactions (or rather activities) of all
     *         processes.
     * @throws InoNetException
     * @throws SrvErrorException
     */
    private Integer getNumberOfTransactions(List<BPProcess> processes)
            throws InoNetException, SrvErrorException {
        int numberOfTransactions = 0;
        for (BPProcess process : processes) {
            List<BPActivityNode> actitvities = process
                    .getOwnedElementTransitiveDown(BPActivityNode.class);

            if (actitvities != null) {
                numberOfTransactions += actitvities.size();
            }
        }
        return numberOfTransactions;
    }
```

Abb. 5-28: Java-Code zur Ermittlung der UUCP (5/6).

```
    /**
     * Calculates the Unadjusted Use Case Weights (UUCW) by the given values of
     * the list.
     *
     * @param numberOfTransactions
     *            A List of Integers: @0: number of use cases with less then 4
     *            transactions, @1: number of use cases with between 4 and 7
     *            transaction, @2: number of use cases with more than 7
     *            transactions.
     * @return The calculated value of Unadjusted Use Case Weights (UUCW).
     */
    private int calculateUucw(List<Integer> numberOfTransactions) {

        int uucw = 0;

        if (numberOfTransactions != null) {
            int lessThan4 = numberOfTransactions.get(0);
            int between4and7 = numberOfTransactions.get(1);
            int moreThan7 = numberOfTransactions.get(2);

            uucw = ((lessThan4 * 5) + (between4and7 * 10) + (moreThan7 * 15));
        }
        return uucw;
    }

}
```

Abb. 5-29: Java-Code zur Ermittlung der UUCP (6/6).

6 Fazit und Ausblick

In der vorliegenden Arbeit wurde ein Ansatz zur Definition eines Qualitätssystems für die Qualitätssicherung modellbasierter SRS entworfen. Darüber hinaus haben wir ein Vorgehen für die Entwicklung einer Applikation zur automatischen Bestimmung von Qualitätslevels anhand von Beispielen vorgestellt.

Fazit

Speziell die Qualitätsfaktoren *Qualität des SRS-Modells* und *Qualität der Modellvisualisierungen* des erarbeiteten Qualitätssystems wurden detailliert betrachtet und mittels Qualitätskriterien definiert. Diese Kriterien sind im Prüfgegenstand (d.h. dem SRS-Modell und den SRS-Modellvisualisierungen) direkt nachweisbar (vgl. Z1, Abschnitt 1.2). Zu jedem Qualitätskriterium gaben wir an, ob es automatisch prüfbar ist, durch eine Kombination aus manueller und automatischer Prüfungen oder ausschließlich manuell sichergestellt werden kann (vgl. Z2).

Der zweite Teil der Arbeit bestand darin, ein Vorgehen vorzustellen, welches die automatische Prüfung, Messung und Bewertung von Qualitätskriterien und -faktoren ermöglicht. Qualitätsdefizite werden durch die Überprüfung der Modellbestandteile gefunden (vgl. Z3). Die Überprüfung findet auf Grundlage von definierten und automatisch überprüfbaren Regeln statt. Wie eine Überprüfung einer solchen Regel implementiert und eingebunden werden kann, zeigten wir exemplarisch für den Modellprüfassistenten des Modellierungswerkzeugs Innovator. Anschließend stellten wir Metriken auf, welche überwiegend die Anzahl der Qualitätsdefizite in Zahlen ausdrücken (vgl. Z4). Auch wie Metrikberechnung für den Innovator programmiert und im Werkzeug integriert werden kann, wurde anhand der Metrik Unadjusted Use Case Points mit einer sogenannten Engineering-Aktion gezeigt. Abschließend erläuterten wir ein Vorgehen zur Ableitung einer Qualitätsbewertung aus den Maßzahlen (vgl. Z5). Wie die Bewertung für eine einzelne Maßzahl vorgenommen werden kann, veranschaulichten wir anhand eines Beispiels sowohl für Qualitäts- als auch Quantitätsmetriken.

Somit haben wir in dieser Arbeit einen Einblick in die Vielfalt und Bandbreite der Möglichkeiten der Qualitätssicherung geben können, zum einen im Rahmen der RE-Phase, zum anderen im Zusammenhang mit dem Einsatz von Modellierungswerkzeugen.

Ausblick

Nun wäre es interessant, im ersten Schritt einzelne Abschnitte des Qualitätssystems an einem SRS-Modell eines realen Softwareentwicklungsprojekts, bei dem Innovator eingesetzt wurde, zu testen. Hierbei sollte untersucht werden, inwiefern die formulierten Metriken und Kriterien Qualitätsdefizite wirklich aufdecken können sowie ob das System um weitere Merkmale erweitert werden sollte oder bereits alle automatisch erfassbaren Qualitätsdefizite im System behandelt werden. Ebenfalls gilt es zu prüfen, ob ähnliche Qualitätskriterien zu einem einzigen Kriterium zusammengefasst werden sollten, da bspw. die Unterscheidung zwischen den Kriterien keinen erkennbaren Mehrwert ausmacht und durch das Zusammenfassen die Komplexität des Qualitätssystems verringert wird. Des Weiteren wäre in diesem

Zuge eine erstmalige projektspezifische Definition der Schwellwerte für die verwendeten Metriken spannend, um mittels des entwickelten Qualitätssystems ein erstes SRS-Modell oder Teile dieses Modells (d.h. einzelne Qualitätskriterien und -faktoren) bewerten zu können.

Im zweiten Schritt ist besonders die vollständige Umsetzung des entwickelten Qualitätssystems interessant sowie die Betrachtung einer größeren Menge von SRS-Modellen aus realen Softwareentwicklungsprojekten. Dadurch können Vergleichsgrößen für übliche Maßzahlen ermittelt werden bspw. die übliche Anzahl an Syntaxfehlern oder die Anzahl dargestellter Modellelemente pro Diagramm. Sinnvoll ist hier sicherlich eine Betrachtung der Mittelwerte, Varianz und Ausreißer unter den Vergleichsmessungen (vgl. Lanza, et al., 2006; Overhage, et al., 2012).

Zusammenfassend sollten einzelne Aspekte in der Praxis überprüft werden und das Qualitätssystem entsprechend weiterentwickelt und projektspezifisch erweitert werden.

Das Ziel dieser Arbeit ist die metrikbasierte Qualitätsbewertung, jedoch kann analytische Qualitätssicherung auch andere Gesichter haben. Neben dem momentanen Einsatz, der Prüfroutinen zur Überprüfung des Modells im Anschluss an die Modellierung, wäre auch das permanente Überprüfen denkbar, ähnlich der Rechtschreibkorrektur von Texteditoren wie z.B. Word. Diese Funktionalität würde dem Modellierer während der Modellierung bei der Sicherstellung eines qualitativ hochwertigen Modells unterstützen, durch eine permanente Überprüfung der neu modellierten Modellbestandteile. Die defizitären Modellbestandteile könnten eingefärbt oder farbig hinterlegt werden, um den Modellierer auf die kritischen Stellen aufmerksam zu machen. Wird eines der markierten Objekte vom Modelliere ausgewählt, so könnten ihm die Prüfungsmeldung mit Anweisungen zur Lösung des Problems angezeigt werden. Eventuell kann auch der Modellierer dem Modellierungswerkzeug die Behebung überlassen. Letzteres würde bedeuten, dass im Modellierungswerkzeug zu der jeweiligen Prüfung konkrete Maßnahmen zur Problembehebung hinterlegt sind und das Werkzeug in der Lage ist, diese Korrekturmaßnahmen selbstständig durchzuführen.

Ein anderer interessanter Qualitätssicherungsansatz könnte die Funktionalität sein, dass das Modellierungswerkzeug selbst erkennt, wenn bestimmte Inhalte oder Konstruktionen bereits ähnlich im Modell vorhanden sind. In diesem Fall sollte es den Modellierer auf die Gleichartigkeit aufmerksam machen. Dadurch können eventuelle Redundanzen aufgezeigt oder es kann auf ähnliche Inhalte hingewiesen werden. Der Modellierer kann entscheiden gleichartige Inhalte zu abstrahieren, welches zur Folge hat, dass die Sachverhalte einfach anstelle von mehrfach im Modell beschrieben werden. Dies verringert bspw. Aufwände bei Modelländerungen. Da das Werkzeug selbst nicht entscheiden kann, ob bspw. der gleiche Sachverhalt an mehreren Stellen im Modell beschrieben wird oder es sich um ähnliche, jedoch unterschiedliche Wirklichkeitsausschnitte handelt, weißt das Werkzeug den Modellierer nur auf die Ähnlichkeiten hin. Die inhaltliche Überprüfung muss durch den Modellierer vorgenommen werden.

Autolayouter sind eine weitere interessante Funktionalität, welche den Modellierungsprozess deutlich effizienter gestalten würde. Diese Layouter optimieren automatisch die Darstellung des Modells d.h. die Gestaltung des Diagramms. Für

Organigramme und Geschäftsressourcendiagramme bietet der Innovator bereits diese Möglichkeit. Wünschenswert wäre eine solche Funktionalität auch für andere Diagrammarten.

In dieser Arbeit wurde die analytische Qualitätssicherung fokussiert. Neben dieser kann Qualität in Modellierungswerkzeugen auch **konstruktiv** umgesetzt werden. Die folgenden Ansätze erachten wir in diesem Rahmen als besonders interessant:

- Modellvorlagen, die für bestimmte Modellierungssachverhalte bereits eine Grundstruktur des Modells erstellen, welches der Modellierer nur noch auf seinen spezielleren Sachverhalt anpassen muss (siehe Abschnitt 3.2.1).
- Modellschablonen, welche die Grundbestandteile eines Teilmodells erzeugen, z.B. einen BPMN-Prozess mit Start- und Endereignis, welche über einen Sequenzfluss miteinander verbunden sind. Diese Schablonen vereinfachen den Modellierungsstart für den ungeübten Modellierer oder für unbekannte Modellierungssprachen, da sie eine Idee vermitteln, wie und was mit der jeweiligen Modellierungssprache modelliert werden kann. Des Weiteren spart auch der geübte Modellierer Zeit, da er die Grundelemente des Teilmodells nicht manuell anlegen muss, sondern sie für ihn automatisch angelegt werden.
- Modellierungswizard bzw. -assistent, der durch die Modellierung einzelner Diagramme führt. Schrittweise werden die für die Generierung des Modells benötigten Informationen beim Modellierer abgefragt und auf Grundlage dessen das Modell erstellt. Unerfahrene Modellierer müssen somit nicht die Modellierungssprache beherrschen, sondern nur den zu modellierenden Prozess, z.B. Wie viele unabhängige Prozesse gibt es? Wie viele unterschiedliche Rollen sind an dem jeweiligen Prozess beteiligt? Wie lauten die Rollen? Durch was wird der Prozess angestoßen? Etc.

Wir nehmen an, dass ein möglicher Nebeneffekt der analytischen Qualitätssicherungsmaßnahmen sein könnte, dass durch sie ein gemeinsames Qualitätsverständnis in Softwareentwicklungsprojekten gebildet und geprägt wird, sich somit ein Standard für die Modellierung durchsetzt. Die Modellierer werden bspw. durch die Behebung der Qualitätsdefizite automatisch darin geschult, welche Konstruktionen die Qualität mindern oder unerwünscht sind. Dadurch lernen sie für ihren Modellierungsstil dazu und modellieren zukünftig den Sachverhalt von vornherein konform zum vorgegebenen Standard. Somit wird simultan konstruktive Qualitätssicherung betrieben.

Weiterhin und auch für die Zukunft ist und bleibt es ein erstrebenswertes Ziel, dass das Modellierungswerkzeug möglichst viele Qualitätssicherungsaufgaben und -maßnahmen übernimmt und so bspw. der Anforderungsanalytiker so wenig Zeit wie möglich für Tätigkeiten wie die Prüfung oder Umgestaltung der Diagramme und Modellinhalte investieren muss. Er kann sich infolgedessen auf seine zentralen Aufgaben konzentrieren: Die Erfassung, logische und inhaltliche Prüfung sowie Konsensfindung der Anforderungen an das zu entwickelnde Softwaresystem.

7 LITERATURVERZEICHNIS

Auer, B. und Schmidt, P. 2013. *Grundkurs Buchführung.* Wiesbaden : Springer Gabler, 2013. S. 14-16.

Balzert, H. 2000. Lastenheft und Glossar. *Lehrbuch der Software-Technik.* s.l. : Spektrum, 2000, S. 53–72.

Barnert, S., et al. 2003. *Der Brockhaus Computer und Informationstechnologie.* Mannheim : Brockhaus GmbH, 2003. S. 588.

Bartelt, C., Ternité, T. und Zieger, M. 2005. Modellbasierte Entwicklung mit dem V-Modell XT. *SIGS Satacom.* 2005, S. 53-64.

Basili, V., Caldiera, G. und Rombach, H. 1994. The Goal Question Metric Approach. *Encyclopedia of software engineering 2.* 1994.

Becker, J. 2012. Grundsätze ordnungsmäßiger Modellierung. *Enzyklopädie der Wirtschaftsinformatik.* [Online] 31. Oktober 2012. [Zitat vom: 28. Januar 2014.] http://www.enzyklopaedie-der-wirtschaftsinformatik.de/wi-enzyklopaedie/lexikon/is-management/Systementwicklung/Hauptaktivitaten-der-Systementwicklung/Problemanalyse-/Grundsatze-ordnungsgemaser-Modellierung.

Becker, J., Probandt, W. und Vering, O. 2012. *Grundsätze ordnungsgemäßer Modellierung.* Heidelberg : Springer Gabler, 2012. S. 31-36.

Bibliographisches Institut GmbH. 2014. Norm, die. *Duden.* [Online] 4. Februar 2014. [Zitat vom: 29. April 2014.] http://www.duden.de/node/651184/revisions/1315107/view.

Brügge, B. und Dutoit, A. 2004. *Objektorientierte Softwaretechnik.* München : Pearson Studium, 2004. S. 36, 59-62.

Deelmann, T. und Loos, P. 2004. Grundsätze ordnungsgemäßer Modellvisualisierung. *Proceedings of Modelling.* 2004.

Ebert, J. 2013. *Softwaremessung.* https://svn.uni-koblenz.de/ebert/ws1314/131118VST04metrics1.pdf : s.n., 2013.

eCH. 2013. eCH-0158: BPMN-Modellierungskonventionen für die öffentliche Verwaltung. *eCH - E-Government-Standards.* [Online] 26. Juni 2013. [Zitat vom: 5. November 2013.] http://www.ech.ch/vechweb/page?p=dossier&documentNumber=eCH-0158.

Fieber, F., Huhn, M. und Rumpe, B. 2008. Modellqualität als Indikator für Softwarequalität: eine Taxonomie. *Informatik-Spektrum, 31(5).* 2008, S. 408–424.

France, R. und Rumpe, B. 2007. Model-driven Development of Complex Software: A Research Roadmap. *2007 Future of Software Engineering.* s.l. : Washington, DC, USA: IEEE Computer Society, 2007. S. 37–54.

Freund, J. und Rücker, B. 2012. *Praxishandbuch BPMN 2.0.* Wien : Hanser, 2012. S. 1-89.

Frohnhoff, S. 2009. *Use Case Points 3.0.* Paderborn : s.n., Februar 2009.

Goll, Joachim. 2012. *Methoden des Software Engineering.* Wiesbaden : Springer Vieweg, 2012. S. 32-45, 55, 59-62, 69-76.

Grechenig, T., et al. 2010. *Softwaretechnik.* München : Pearson Studium, 2010. S. 181-184.

IEEE. 1998. *IEEE Recommended Practice for Software Requirements Specifications.* s.l. : IEEE, 1998.

—. 2012. SEVOCAB: Software and Systems Engineering Vocabulary. [Online] 2012. [Zitat vom: 28. Oktober 2013.] (S.247-248). http://pascal.computer.org/sev_display/printCatalog.action.

IHK. 2006. IHK-Notenschlüssel (dezimal). *Industrie- und Handelskammer für München und Oberbayern.* [Online] 24. Januar 2006. [Zitat vom: 15. April 2014.] https://www.muenchen.ihk.de/de/bildung/Anhaenge/Notenschluessel-dezimal.pdf.

IT-Beauftragte der Bundesregierung, [Hrsg.]. 2006. *V-Modell XT Bund.* 2006. http://gsb.download.bva.bund.de/BIT/V-Modell_XT_Bund/V-Modell-XT-Bund-Gesamt.pdf.

IT-Stab des Bundesministerium des Innern. 2014. Häufig gestellte Fragen zum V-Modell XT. *Die Beauftragte der Bundesregierung für Informationstechnik.* [Online] 14. April 2014. [Zitat vom: 14. April 2014.] http://www.cio.bund.de/Web/DE/Architekturen-und-Standards/V-Modell-XT/Haeufig-gestellte-Fragen/haeufig_gestellte_fragen_node.html#doc4623024bodyText6.

Lackes, R. und Siepermann, M. 2014. Softwarehaus. *Gabler Wirtschaftslexikon.* [Online] Springer Gabler Verlag, 29. April 2014. [Zitat vom: 29. April 2014.] http://wirtschaftslexikon.gabler.de/Archiv/76206/softwarehaus-v8.html.

Lamsweerde, A. 2009. Requirements Specification and Documentation. *Requirements Engineering.* s.l. : Wiley, 2009, S. 119–183.

Lanza, M. und Marinescu, R. 2006. *Object-Oriented Metrics in Practice.* Heidelberg : Springer, 2006, S. 1-21.

Laudon, K., Laudon, J. und Schoder, D. 2006. *Wirtschaftsinformatik.* München : Pearson Studium, 2006. S. 30.

Liggesmeyer, P. 2002. *Software-Qualität.* Heidelberg : Spektrum, 2002. S. 209-248.

May, H. und May, U. 2001. *Lexikon der ökonomischen Bildung.* München : Oldenbourg, 2001. S. 391-393.

McCall, J., Richards, P. und Walters, G. 1977. Task Objectives. *Factors in Software Quality.* New York : NTIS, 1977, S. 1-1–1-4.

MID GmbH. 2013a. Anwenderberichte. [Online] 2013a. [Zitat vom: 12. Januar 2014.] http://www.mid.de/unternehmen/anwenderberichte0.html.

—. 2014a. Assistent zur Modellprüfung. *Innovator 11 R5 Hilfe.* [Online] 7. Januar 2014a. [Zitat vom: 24. Februar 2014.] http://help.innovator.de/11.5/de_de/InnovatorX/Content/Ref.XShell/DialogVerifyWizard.htm.

—. 2013b. Eigene Prüfaktionen konfigurieren. *Innovator 11 R5 Hilfe.* [Online] 2013b. [Zitat vom: 16. 01 2014.] http://help.innovator.de/11.5/de_de/InnovatorX/Content/Customizing/Configure_Your_Own_Verification.htm.

—. 2013c. Features. [Online] 2013c. [Zitat vom: 10. Januar 2014.] http://www.mid.de/produkte/innovator-enterprise-modeling/features2.html.

—. 2013g. Notationen (Referenz). *Innovator 11 R5 Hilfe.* [Online] 2013g. [Zitat vom: 10. Januar 2014.] http://help.innovator.de/11.5/de_de/InnovatorX/Content/Reference/Notations_(Reference).htm.

—. 2013h. Rollen und Berechtigungen. *Innovator 11 R5 Hilfe.* [Online] 2013h. [Zitat vom: 12. Januar 2014.] http://help.innovator.de/11.5/de_de/InnovatorX/Content/Basics.Rights/Roles_and_Authorizations.htm.

—. 2013i. Überblick und Einführung in die Arbeit mit Innovator X Generation. *Innovator 11 R5 Hilfe.* [Online] 2013i. [Zitat vom: 10. Januar 2014.] http://help.innovator.de/11.5/de_de/InnovatorX/Content/Basics.Overview/Overview_and_Introduction.htm.

OMG. 2011. Business Process Model and Notation (BPMN). *Business Process Model and Notation.* [Online] Januar 2011. [Zitat vom: 14. April 2014.] http://www.omg.org/spec/BPMN/2.0/PDF.

Overhage, S., Birkmeier, D. und Schlauderer, S. 2012. Qualitätsmerkmale, -metriken und -messverfahren für Geschäftsprozessmodelle. *Wirtschaftsinformatik.* Mai 2012, Bd. 54, 5, S. 217-235.

Pfuhl, M. 2014. Taxonomien. *Enzyklopädie der Wirtschaftsinformatik.* [Online] 29. April 2014. [Zitat vom: 29. April 2014.] http://www.enzyklopaedie-der-wirtschaftsinformatik.de/wi-enzyklopaedie/lexikon/daten-wissen/Wissensmanagement/Wissensmodellierung/Wissensreprasentation/Semantisches-Netz/Taxonomien.

Pohl, K. 2010. *Requirements Engineering: Fundamentals, Principles, and Techniques.* s.l. : Springer, 2010.

Rausch, A. und Niebuhr, D. 2005. Erfolgreiche IT-Projekte mit dem V-Modell XT. *OBJEKTspektrum.* 2005, 03, S. 42-49.

Rupp, C. und SOPHISTEN. 2009. *Requirements-Engineering und -Management.* Wien : Hanser, 2009. S. 12-17, 23-28, 293-307.

Rupp, C., Queins, S. und Zengler, B. 2007. *UML 2 glasklar.* Wien : Hanser, 2007. S. 11-18.

Scheible, J. 2010. Ein Framework zur automatischen Ermittlung der Modellqualität bei eingebetteten Systemen. *MBEES.* 2010.

Sneed, H., Seidl, R. und Baugartner, M. 2010. *Software in Zahlen.* s.l. : Carl Hanser Verlag, 2010. S. 1-19; 91-114.

Statistisches Bundesamt. Kleine und mittlere Unternehmen (KMU). *Statistisches Bundesamt.* [Online] [Zitat vom: 12. 12 2013.] https://www.destatis.de/DE/ZahlenFakten/GesamtwirtschaftUmwelt/UnternehmenHandwerk/KleineMittlereUnternehmenMittelstand/KMUBegriffserlaeuterung.html.

Staud, J. 2014. BPMN - Einführung und Einschätzung ENTWURF / DRAFT : 1 Einleitung. *staud.* [Online] 20. Feburar 2014. [Zitat vom: 29. April 2014.] http://www.staud.info/bpmn/bp_f_1.htm.

Strahringer, S. 2014. Modell. *Enzyklopädie der Wirtschaftsinformatik.* [Online] 29. April 2014. [Zitat vom: 29. April 2014.] http://www.enzyklopaedie-der-wirtschaftsinformatik.de/wi-enzyklopaedie/lexikon/is-management/Systementwicklung/Hauptaktivitaten-der-Systementwicklung/Problemanalyse-/konzeptuelle-modellierung-von-is/modell.

Thomas, O. 2005. *Das Modellverständnis in der Wirtschaftsinformatik: Historie, Literaturanalyse und Begriffsexplikation.* s.l. : Inst. für Wirtschaftsinformatik, 2005.

WEIT e.V. 2006. V-Modell XT. *Die Beauftragte der Bundesregierung für Informationstechnik.* [Online] 2006. [Zitat vom: 10. Dezember 2013.] http://ftp.tu-clausthal.de/pub/institute/informatik/v-modell-xt/Releases/1.4/V-Modell-XT-Gesamt.pdf.

White, S. 2004a. *Business Process Modeling Notation.* 3. Mai 2004a.

—. 2004b. Introduction to BPMN. *Business Process Model and Notation.* [Online] Mai 2004b. [Zitat vom: 14. April 2014.] http://www.omg.org/bpmn/Documents/Introduction_to_BPMN.pdf.

Wikipedia. 2013a. Innovator (Software). *Wikipedia.* [Online] 4. Dezember 2013a. [Zitat vom: 10. Januar 2014.] http://de.wikipedia.org/wiki/Innovator_(Software).

—. 2013b. Softwaremetrik. *Wikipedia.* [Online] 6. April 2013b. [Zitat vom: 03. Dezember 2013.] http://de.wikipedia.org/wiki/Softwaremetrik.

—. 2014. Unified Modeling Language. *Wikipedia.* [Online] 28. April 2014. [Zitat vom: 18. April 2014.] http://en.wikipedia.org/wiki/Unified_Modeling_Language.

8 ANHANG

In den folgenden Abschnitten werden folgende Inhalte angeführt

Abschnitt 8.1 Das *Glossar* mit den zentralen Begriffen der Arbeit.

Abschnitt 8.2 *Code* zu weiteren Prüfungen und Metriken, welche im Rahmen der Arbeit implementiert wurden sowie die Konfigurationsdatei für den Modellprüfassistenten.

Abschnitt 8.3 Beispiel-*Diagramme* der einzelnen Diagrammarten aus Innovator.

Abschnitt 8.4 *Screenshots* beispielhafter Überprüfungen der Inhalte von BPMN-Diagrammen mittels der im Rahmen dieser Arbeit umgesetzten eCH-0158-Prüfungen.

Abschnitt 8.5 *Tabellen* der Umsetzungsart der eCH-0158-Konventionen zu BPMN-Pools und -Startereignissen.

8.1 GLOSSAR 🕮

Im Folgenden werden alphabetisch sortiert zentrale und möglicherweise semantisch unterschiedlich geprägte Begriffe der vorliegenden Arbeit kurz definiert.

ANFORDERUNG (nach ISO/IEC/IEEE 24765:2010)
„Eine Bedingung oder Fähigkeit, welche das System, die Systemkomponente, das Produkt oder der Dienst erfüllen oder besitzen muss, um einen Vertrag, einen Standard, eine Spezifikation oder anderen vorgeschriebenen Dokumente zu genügen.“[70] (IEEE, 2012)

ANFORDERUNGSANALYSE
Die Phase des Software-Lebenslaufs, in der die Anforderungen an das Softwaresystem definiert und dokumentiert werden (IEEE, 2012).

ANFORDERUNGSSPEZIFIKATION
Siehe SRS🕮.

ARTEFAKT *„Ein greifbares Stück Information, das durch Mitarbeiter erzeugt, geändert und benutzt wird, wenn sie Aktivitäten ausführen.“* (Balzert, 2000)

DIAGRAMM Grafische Beschreibung eines Modellausschnitts.

FACHSEITE Die Gruppe der Nutzer und Anwender des zu erstellenden Softwaresystems.

LASTENHEFT Durch den Auftraggeber erstellte SRS🕮 (siehe Abschnitt 2.3.1).

[70] Übersetzung von Meike Weber; Original: *“condition or capability that must be met or possessed by a system, system component, product, or service to satisfy an agreement, standard, specification, or other formally imposed documents”* (IEEE, 2012)

MAßZAHL Der Zahlenwert, den die Metrik ausgibt (Wikipedia, 2013b).

METRIK Die Abbildung einer bestimmten Eigenschaft einer ausgemessenen Einheit auf einen numerischen Wert (Lanza, et al., 2006).

MODELL Abbild eines Originals, die gegenüber dem Original abstrahiert und nur die für den Zweck des Modells relevanten Sachverhalte des Originals darstellt (siehe Abschnitt 2.1).

MODELLBESTANDTEIL
Eine Modellelement🕮 oder Teilmodell🕮.

MODELLELEMENT
Kleinste Einheit des Modells.

MODELLVISUALISIERUNG
Siehe Diagramm🕮.

NORM *"Vorschrift, Regel, Richtlinie o.Ä."* (Bibliographisches Institut GmbH, 2014).

PFLICHTENHEFT
Durch den Auftragnehmer erstellte SRS🕮 (siehe Abschnitt 2.3.2).

PROFIL (in Innovator)
„Ein Profil stellt in Innovator eine für sich abgeschlossene Einheit dar, die bestimmte Aspekte der Konfiguration komplett enthält oder aufbauend auf einem importierten Profil erweitert." (MID GmbH, 2013)

QUANTITÄTSMETRIK (nach Sneed, et al., 2010)
Softwaremetrik🕮, die die Größe einer definierten Menge misst. Trifft eine Aussage über den Umfang der Software, z.B. Anzahl der funktionalen Anforderungen.

QUALITÄT *„Die Fähigkeit eines Produkts, Service, Systems, einer Komponente oder eines Prozesses Wünsche, Erwartungen oder Anforderungen der Kunden oder Benutzer zu erfüllen."*[71] (IEEE, 2012)

QUALITÄTSMETRIK (nach Sneed, et al., 2010)
Softwaremetrik🕮, die die Abweichung von einer Norm🕮 misst, z.B. Anzahl der Verstöße gegen eine Namenskonvention für Aktivitäten in BPMN-Diagrammen.

QUALITÄTSSYSTEM (in dieser Arbeit)
Ein Qualitätsmodell, welches eine Ordnung aus Qualitätseigenschaften im Sinne einer Taxonomie definiert: Die darin enthaltenen Qualitätseigenschaften sind durch Abhängigkeitsbeziehungen baumartig angeordnet, wobei die näher an der Wurzel

[71] Übersetzung von Meike Weber, Original: *„ability of a product, service, system, component, or process to meet customer or user needs, expectations, or requirements"* (IEEE, 2012).

liegenden Eigenschaften weniger spezifisch sind als die weiter entfernten. Mit Hilfe dieses Systems werden die Qualitätsanforderungen für modellbasierte SRS differenziert und strukturiert.

SICHT Betrachtung eines Teilbereichs eines Modells. (Brügge, et al., 2004)

SOFTWARE Alle Artefakte📖, die während der Softwareentwicklung entstehen, vom Programm-Code bis hin zur Anforderungsdokumentation (z.B. Pflichtenheft) (McCall, et al., 1977).

SOFTWARE-LEBENSZYKLUS
Summe aller Phasen einer Software von der Planung über bspw. Anforderungserfassung, Implementation, etc. bis hin zur Ausmusterung der Software, dem „Lebensende".

SRS (nach ISO/IEC/IEEE 29148:2011)
„eine strukturierte Kollektion der Anforderungen an die Software und ihre externen Schnittstellen"[72].

SRS-MODELL Ein Modell📖, dessen Inhalte den Zweck erfüllen, auf Grundlage dieses Modells eine modellbasierte SRS-Dokumentation generieren zu können.

STAKEHOLDER Eine Person oder Gruppe, die von dem zu erstellenden Softwaresystem tangiert sind oder auf dieses einwirken können, dazu gehören insbesondere: Auftraggeber, Benutzer und Entwickler der Software.

SYSTEM Ein System setzt sich aus Systemkomponenten zusammen, z.B. Prozesse, Akteure, Daten, Software und Hardware etc. Die Systemkomponenten sind miteinander verbunden (Brügge, et al., 2004), es gibt bspw. Beziehungen und Abhängigkeiten zwischen ihnen.

SOFTWAREHAUS
„Unternehmen, das vorrangig Softwareprodukte für externe Auftraggeber (Individualsoftware) oder Standardsoftware herstellt" (Lackes, et al., 2014)

TEILMODELL Ein Teil eines Modells, innerhalb welcher die gleiche Modellierungssprache verwendet wird.

[72] Übersetzung der Autorin, Original: *"structured collection of the requirements […] of the software and its external interfaces"* (IEEE, 2012)

8.2 CODE

In diesem Abschnitt wird Java-Code weiterer, im Rahmen dieser Abschlussarbeit erstellten Prüfungen aufgeführt (siehe Abb. 8-1, S. 141 bis Abb. 8-7, S. 147). Diese Prüfungen sind für die Einbindung in den Innovator-Modellprüfassistenten gedacht und überprüfen Regeln der eCH-0158-Konventionen bezüglich BPMN-Pools und BPMN-Startereignissen. In Abb. 8-8 (S. 148) bis Abb. 8-18 (S. 158) ist die Konfigurationsdatei „inoverify.xml“ zu sehen, welche diese Java-Funktionen in den Modellprüfassistenten einbindet. Durch diese Konfigurationsdatei werden die Prüfungen aus Abb. 8-1 (S. 141) bis Abb. 8-7 (S. 147) im Modellprüfassistenten angezeigt und können von dort aus ausgeführt werden. Zuletzt die Implementation einiger Metriken bezüglich Anforderungen gezeigt (siehe Abb. 8-19, S. 159 und Abb. 8-20, S. 160).

```java
package de.mid.engineering.innovator.verify;

import java.util.List;

import de.mid.engineering.innovator.pruefmanager.IFConfigProvider;
import de.mid.engineering.innovator.pruefmanager.check.GenericCheckBase;
import de.mid.engineering.innovator.pruefmanager.view.IFModelCheckTestee;
import de.mid.innovator.srv2api.icw2bp.BPEventNode;
import de.mid.innovator.srv2api.icw2bp.BPProcess;
import de.mid.innovator.srv2api.icw2elem.ELNamedElement;
import de.mid.innovator.srv2api.icw2elem.K_BP_EVENT_TYPE;
import de.mid.innovator.srv2api.icw2meta.ADModel;

/**
 * This function checks whether the BPMN-process has at least one
 * start-event-node.
 *
 * @author meweber
 *
 */
public class Check_BPMN_BPProcess_HasStartEventNode extends GenericCheckBase {

    public Check_BPMN_BPProcess_HasStartEventNode(ADModel model,
            IFConfigProvider cfgProvider) throws Exception {
        super(model, cfgProvider);
    }

    protected CheckResultState isFaulty(IFModelCheckTestee testee)
            throws Exception {

        ELNamedElement element = testee.getNamedElement();
        CheckResultState crs = CheckResultState.notApplicable;

        if (element instanceof BPProcess) {

            BPProcess process = (BPProcess) element;
            List<BPEventNode> eventNodes = process
                    .getOwnedElementTransitiveDown(BPEventNode.class, null);

            for (BPEventNode eventNode : eventNodes) {
                if (eventNode.getEventType().equals(K_BP_EVENT_TYPE.Start))
                    return crs = CheckResultState.ok;
            }

            return crs = CheckResultState.faulty;

        } else
            return crs;
    }
}
```

Abb. 8-1: Code der Prüfung „Startereignis vorhanden“.

```
package de.mid.engineering.innovator.verify;

import java.util.List;

import de.mid.engineering.innovator.pruefmanager.IFConfigProvider;
import de.mid.engineering.innovator.pruefmanager.check.GenericCheckBase;
import de.mid.engineering.innovator.pruefmanager.view.IFModelCheckTestee;
import de.mid.innovator.net.InoNetException;
import de.mid.innovator.srv.SrvErrorException;
import de.mid.innovator.srv2api.icw2bp.BPEventNode;
import de.mid.innovator.srv2api.icw2bp.BPSequenceFlow;
import de.mid.innovator.srv2api.icw2bp.BPSequenceFlowNode;
import de.mid.innovator.srv2api.icw2elem.ELNamedElement;
import de.mid.innovator.srv2api.icw2elem.K_BP_EVENT_TYPE;
import de.mid.innovator.srv2api.icw2meta.ADModel;

/**
 * This function checks whether the end-event-node is connected with at least
 * one start-event-node by sequence flow.
 *
 * @author meweber
 *
 */
public class Check_BPMN_BPEventNode_EndEventNodeIsConnectedWithStartEventNode
        extends GenericCheckBase {

    public Check_BPMN_BPEventNode_EndEventNodeIsConnectedWithStartEventNode(
            ADModel model, IFConfigProvider cfgProvider) throws Exception {
        super(model, cfgProvider);
    }

    protected CheckResultState isFaulty(IFModelCheckTestee testee)
            throws Exception {

        ELNamedElement element = testee.getNamedElement();
        CheckResultState crs = CheckResultState.notApplicable;
           if (element instanceof BPEventNode) {
               BPEventNode eventNode = (BPEventNode) element;
               crs = CheckResultState.ok;

               if (eventNode.getEventType().equals(K_BP_EVENT_TYPE.End)) {
                   List<BPSequenceFlow> incomings = eventNode
                           .getIncomingSequenceFlow(null);
                   for (BPSequenceFlow incoming : incomings) {
                       if (hasAnyStartEventNode(incoming)) {
                           return crs = CheckResultState.ok;
                       } else {
                           return crs = CheckResultState.faulty;
                       }
                   }

                   return crs = CheckResultState.faulty;

               } else
                   return crs;
           } else
               return crs;
       }
```

Abb. 8-2: Code der Prüfung „Endereignis ist über Sequenzfluss mit Startereignis verbunden“ (1/2).

```
    /**
     * This Function searches recursively for a start-event-node.
     *
     * @param incoming
     *            A BPSequenceFlow.
     * @return 'true', if a source of the sequence-flow is a start-event-node,
     *         else 'false'.
     * @throws SrvErrorException
     * @throws InoNetException
     */
    private boolean hasAnyStartEventNode(BPSequenceFlow incoming)
            throws InoNetException, SrvErrorException {
        BPSequenceFlowNode sourceNode = incoming.getSourceNode();
        if (sourceNode instanceof BPEventNode) {
            BPEventNode eventNode = (BPEventNode) sourceNode;
            if (eventNode.getEventType().equals(K_BP_EVENT_TYPE.Start)) {
                return true;
            }
        }
        List<BPSequenceFlow> incomings = sourceNode
                .getIncomingSequenceFlow(null);
        for (BPSequenceFlow newIncoming : incomings) {
            if (hasAnyStartEventNode(newIncoming)) {
                return true;
            }
        }
        return false;
    }
}
```

Abb. 8-3: Code der Prüfung „Endereignis ist über Sequenzfluss mit Startereignis verbunden“ (2/2).

```
package de.mid.engineering.innovator.verify;

import de.mid.engineering.innovator.pruefmanager.IFConfigProvider;
import de.mid.engineering.innovator.pruefmanager.check.GenericCheckBase;
import de.mid.engineering.innovator.pruefmanager.view.IFModelCheckTestee;
import de.mid.innovator.srv2api.icw2bp.BPEventNode;
import de.mid.innovator.srv2api.icw2elem.ELNamedElement;
import de.mid.innovator.srv2api.icw2elem.K_BP_EVENTDEFINITION_TYPE;
import de.mid.innovator.srv2api.icw2elem.K_BP_EVENT_TYPE;
import de.mid.innovator.srv2api.icw2meta.ADModel;

/**
 * This function checks whether the unspecified start-event-node has no name.
 *
 * @author meweber
 *
 */
public class Check_BPMN_BPEventNode_UnspecifiedStartEventNodeHasNoName extends
        GenericCheckBase {

    public Check_BPMN_BPEventNode_UnspecifiedStartEventNodeHasNoName(
            ADModel model, IFConfigProvider cfgProvider) throws Exception {
        super(model, cfgProvider);
    }

    protected CheckResultState isFaulty(IFModelCheckTestee testee)
            throws Exception {

        ELNamedElement element = testee.getNamedElement();
        CheckResultState crs = CheckResultState.notApplicable;

        if (element instanceof BPEventNode) {
            BPEventNode eventNode = (BPEventNode) element;
            crs = CheckResultState.ok;

            if (eventNode.getEventType().equals(K_BP_EVENT_TYPE.Start)) {

                if (eventNode.getEventDefinitionType().equals(
                        K_BP_EVENTDEFINITION_TYPE.Unspecified)) {
                    crs = CheckResultState.faulty;

                    if (eventNode.getName().isEmpty()) {
                        return crs = CheckResultState.ok;
                    } else {
                        return crs;
                    }

                } else
                    return crs;
            } else
                return crs;
        } else
            return crs;
    }
}
```

Abb. 8-4: Code der Prüfung „Unbestimmtes Startereignis ohne Beschreibung und ohne Definition der Blankodefinition“.

```
package de.mid.engineering.innovator.verify;

import java.util.List;

import de.mid.engineering.innovator.pruefmanager.IFConfigProvider;
import de.mid.engineering.innovator.pruefmanager.check.GenericCheckBase;
import de.mid.engineering.innovator.pruefmanager.view.IFModelCheckTestee;
import de.mid.innovator.srv2api.icw2bp.BPEventDefinition;
import de.mid.innovator.srv2api.icw2bp.BPEventNode;
import de.mid.innovator.srv2api.icw2elem.ELNamedElement;
import de.mid.innovator.srv2api.icw2elem.K_BP_EVENTDEFINITION_TYPE;
import de.mid.innovator.srv2api.icw2elem.K_BP_EVENT_TYPE;
import de.mid.innovator.srv2api.icw2meta.ADModel;

/**
 * This function checks whether the conditional-start-event-node has a defined
 * conditional-definition.
 *
 * @author meweber
 *
 */
public class Check_BPMN_PBEventNode_ConditionalStartEventNodeHasDefinition
        extends GenericCheckBase {

    public Check_BPMN_PBEventNode_ConditionalStartEventNodeHasDefinition(
            ADModel model, IFConfigProvider cfgProvider) throws Exception {
        super(model, cfgProvider);
    }

    protected CheckResultState isFaulty(IFModelCheckTestee testee)
            throws Exception {

        ELNamedElement element = testee.getNamedElement();
        CheckResultState crs = CheckResultState.notApplicable;

        if (element instanceof BPEventNode) {
            BPEventNode eventNode = (BPEventNode) element;
            crs = CheckResultState.ok;

            if (eventNode.getEventType().equals(K_BP_EVENT_TYPE.Start)) {
                List<BPEventDefinition> eventDefinitions = eventNode
                        .getEventDefinition(BPEventDefinition.class);

                for (BPEventDefinition eventDefinition : eventDefinitions) {
                    if (eventDefinition.getEventDefinitionType().equals(
                            K_BP_EVENTDEFINITION_TYPE.Conditional)) {
                        if (eventDefinition.getDisplayName().isEmpty())
                            return crs = CheckResultState.faulty;
                        if (eventDefinition.getDisplayName().startsWith(
                                "bedingte"))
                            return crs = CheckResultState.faulty;
                    }
                }
            } else
                return crs;
        }
        return crs;
    }
}
```

Abb. 8-5: Code der Prüfung „Bedingtes Startereignis mit Definition des Bedingungsdefinition“.

```
package de.mid.engineering.innovator.verify;

import java.util.List;

import de.mid.engineering.innovator.pruefmanager.IFConfigProvider;
import de.mid.engineering.innovator.pruefmanager.check.GenericCheckBase;
import de.mid.engineering.innovator.pruefmanager.view.IFModelCheckTestee;
import de.mid.innovator.srv2api.icw2bp.BPEventDefinition;
import de.mid.innovator.srv2api.icw2bp.BPEventNode;
import de.mid.innovator.srv2api.icw2elem.ELNamedElement;
import de.mid.innovator.srv2api.icw2elem.K_BP_EVENTDEFINITION_TYPE;
import de.mid.innovator.srv2api.icw2elem.K_BP_EVENT_TYPE;
import de.mid.innovator.srv2api.icw2meta.ADModel;

/**
 * This function checks whether the timer-start-event-node has a defined
 * timer-definition.
 *
 * @author meweber
 *
 */
public class Check_BPMN_PBEventNode_TimerStartEventNodeHasDefinition extends
        GenericCheckBase {

    public Check_BPMN_PBEventNode_TimerStartEventNodeHasDefinition(
            ADModel model, IFConfigProvider cfgProvider) throws Exception {
        super(model, cfgProvider);
    }

    protected CheckResultState isFaulty(IFModelCheckTestee testee)
            throws Exception {

        ELNamedElement element = testee.getNamedElement();
        CheckResultState crs = CheckResultState.notApplicable;

        if (element instanceof BPEventNode) {
            BPEventNode eventNode = (BPEventNode) element;
            crs = CheckResultState.ok;

            if (eventNode.getEventType().equals(K_BP_EVENT_TYPE.Start)) {
                List<BPEventDefinition> eventDefinitions = eventNode
                        .getEventDefinition(BPEventDefinition.class);
                for (BPEventDefinition eventDefinition : eventDefinitions) {

                    if (eventDefinition.getEventDefinitionType().equals(
                            K_BP_EVENTDEFINITION_TYPE.Timer)) {
                        if (eventDefinition.getDisplayName().isEmpty()) {
                            return crs = CheckResultState.faulty;
                        }
                        if (eventDefinition.getDisplayName().startsWith(
                                "zeitliche")) {
                            return crs = CheckResultState.faulty;
                        }
                    }
                }
            } else
                return crs;
        }
        return crs;
    }
}
```

Abb. 8-6: Code der Prüfung „Zeitliches Startereignis mit Definition des Zeitgeberdefinition“.

```java
package de.mid.engineering.innovator.verify;

import de.mid.engineering.innovator.pruefmanager.IFConfigProvider;
import de.mid.engineering.innovator.pruefmanager.check.GenericCheckBase;
import de.mid.engineering.innovator.pruefmanager.view.IFModelCheckTestee;
import de.mid.innovator.srv2api.icw2bp.BPEventNode;
import de.mid.innovator.srv2api.icw2bp.BPProcess;
import de.mid.innovator.srv2api.icw2elem.ELContainerAble;
import de.mid.innovator.srv2api.icw2elem.ELNamedElement;
import de.mid.innovator.srv2api.icw2elem.K_BP_EVENTDEFINITION_TYPE;
import de.mid.innovator.srv2api.icw2elem.K_BP_EVENT_TYPE;
import de.mid.innovator.srv2api.icw2meta.ADModel;

/**
 * This function checks whether the BPMN-process, which contains an unspecified
 * start-event-node, has at least one calling activity.
 *
 * @author meweber
 *
 */
public class Check_BPMN_BPEventNode_UnspecifiedStartEventNodeHasCallingActivity
        extends GenericCheckBase {

    public Check_BPMN_BPEventNode_UnspecifiedStartEventNodeHasCallingActivity(
            ADModel model, IFConfigProvider cfgProvider) throws Exception {
        super(model, cfgProvider);
    }

    protected CheckResultState isFaulty(IFModelCheckTestee testee)
            throws Exception {

        ELNamedElement element = testee.getNamedElement();
        CheckResultState crs = CheckResultState.notApplicable;

        if (element instanceof BPEventNode) {
            BPEventNode eventNode = (BPEventNode) element;
            crs = CheckResultState.ok;

              if (eventNode.getEventType().equals(K_BP_EVENT_TYPE.Start)) {

                  if (eventNode.getEventDefinitionType().equals(
                          K_BP_EVENTDEFINITION_TYPE.Unspecified)) {
                      ELContainerAble owner = eventNode
                              .getFirstOwnerWhichIsConformTo(BPProcess.class);
                      crs = CheckResultState.faulty;

                      if (owner instanceof BPProcess) {
                          BPProcess ownerProcess = (BPProcess) owner;
                          if (ownerProcess.getProcessCalling(null).isEmpty()) {
                              return crs = CheckResultState.faulty;
                          } else {
                              return crs = CheckResultState.ok;
                          }

                      } else
                          return crs;
                  } else
                      return crs;
              } else
                  return crs;
          } else
              return crs;
      }
  }
```

Abb. 8-7: Code der Prüfung „Unbestimmtes Startereignis nur, wenn Prozess eine Aufrufaktivität hat“.

```
<?xml version="1.0" encoding="utf-8"?>
<VerificationConfiguration xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.mid.de/inoverify inoverify.xsd" xmlns="http://www.mid.de/inoverify">
    <Actions>
        <VerificationAction>
            <Name>
                <German>Einhaltung des eCH-0158 Standards prüfen</German>
                <English>Check adherence of eCH-0158 standard</English>
            </Name>
            <Description>
                <German>Überprüft, ob jene Richtlinien des schweizer Standards eCH-0158 (für deskriptive BPMN-Modelle) eingehalten werden, die automatisch überprüfbar
                sind.</German>
                <English>Checks whether the model and the diagrams meet those guidelines of the Swiss standard eCH-0158, that can be checked automatically.</English>
            </Description>
            <Groups>

                <VerificationGroup>
                    <Name>
                        <German>"Angemessene Modellierungssyntax" überprüfen</German>
                        <English>Check suitable syntax of the model</English>
                    </Name>
                    <Singles>
```

Abb. 8-8: Dokument „inoverify.xml“ (1/11).

```
                        <!-- Endereignis ist mit mindestens einem Starteregnis verbunden. -->
                        <SingleVerification xsi:type="JavaGenericVerification">
                            <Name>
                                <German>Endereignis ist mit Startereignis verbunden.</German>
                                <English>Endevent is connected with startevent.</English>
                            </Name>
                            <Type>JavaGeneric</Type>
                            <Description>
                                <German>Prüft ob Endereignis mit mindestens einem Startereignis verbunden ist.</German>
                                <English>Checks whether the Endevent ist connected with a startevent.</English>
                            </Description>
                            <Check>de.mid.engineering.innovator.verify.Check_BPMN_BPEventNode_EndEventNodeIsConnectedWithStartEventNode</Check>
                            <Message>
                                <German>Das Endereignis ist nicht mit mindestens einem Startereignis verbunden, somit ist der modellierte Prozess nicht vollständig.
- eCH-0158: "In jedem aufgeklappten Pool wird genau ein vollständiger Prozess modelliert."
- Den Fehler beheben: Erstellen Sie ein Startereignis im Pool des Endereignisses. Verbinden Sie das Startereignis per Sequenzfluss mit derjenigen Aktivität (innerhalb
des Pools), mit welcher der im Pool modellierte Prozess beginnen soll.
                                </German>
                                <English />
                            </Message>
                            <MessageKind>Warning</MessageKind>
                            <Parameters />
                            <Stereotypes>
                                <VerifyStereotype>
                                    <Metatype>BPEventNode</Metatype>
                                    <Profile>ROOT PROFILE</Profile>
                                    <Stereotype>event</Stereotype>
                                    <Id>[eCH0158-Pool-05]</Id>
                                    <IsActive>true</IsActive>
                                </VerifyStereotype>
                            </Stereotypes>
                        </SingleVerification>
```

Abb. 8-9: Dokument „inoverify.xml“ (2/11).

```
                        <!-- Unbestimmtes Startereignis, nur wenn übergeordneter Prozessaufruf. -->
                        <SingleVerification xsi:type="JavaGenericVerification">
                            <Name>
                                <German>Unbestimmtes Startereignis, nur wenn übergeordneter Prozessaufruf.</German>
                                <English>Undefined Startevent only if atleast one activity is calling the process.</English>
                            </Name>
                            <Type>JavaGeneric</Type>
                            <Description>
                                <German>Prüfung ob ein unbestimmte Startereignisse von mindestens einem übergeordneten Prozess aufgerufen werden.
(Hintergrund: Ist dies nicht der Fall, muss das Startereignis bestimmt werden).</German>
                                <English>Checks whether undefined startevents are called by atleast one Activity.</English>
                            </Description>
                            <Check>de.mid.engineering.innovator.verify.Check_BPMN_BPEventNode_UnspecifiedStartEventNodeHasCallingActivity</Check>
                            <Message>
                                <German>Das Startereignis ist unbestimmt und wird von keinem übergeordneten Prozess aufgerufen.
- eCH-0158: "Unbestimmte Startereignisse werden nur verwendet, wenn der Prozess durch den Aufruf auf der übergeordneten Prozessebene gestartet wird."
- Den Fehler beheben: a) Bestimmen sie das Startereignis in dem sie dem Ereignis eine Ereignisdefinition zuordnen. b) Rufen Sie den Prozess durch den ihm übergeordneten
Prozess auf (über die Eigenschaft 'Aufgerufenes Element').
                                </German>
                                <English />
                            </Message>
                            <MessageKind>Warning</MessageKind>
                            <Parameters />
                            <Stereotypes>
                                <VerifyStereotype>
                                    <Metatype>BPEventNode</Metatype>
                                    <Profile>ROOT PROFILE</Profile>
                                    <Stereotype>event</Stereotype>
                                    <Id>[eCH0158-Startereignis-05]</Id>
                                    <IsActive>true</IsActive>
                                </VerifyStereotype>
                            </Stereotypes>
                        </SingleVerification>
```

Abb. 8-10: Dokument „inoverify.xml“ (3/11).

```
                    <!-- Pro Prozess mindestens ein Startereignis. -->
                    <SingleVerification xsi:type="JavaGenericVerification">
                        <Name>
                            <German>Pro Prozess mindestens ein Startereignis.</German>
                            <English>Atleast one Startevent per process.</English>
                        </Name>
                        <Type>JavaGeneric</Type>
                        <Description>
                            <German>Prüfung zum Vorhandensein mindestens eines Startereignisse pro BPMN-Prozess.
(Hintergrund: Ein Prozess ist unvollständig ohne Startereignis).</German>
                            <English>Checks whether the process has atleast one startevent.</English>
                        </Description>
                        <Check>de.mid.engineering.innovator.verify.Check_BPMN_BPProcess_HasStartEventNode</Check>
                        <Message>
                            <German>Der Prozess beinhaltet kein Startereignis, ist somit unvollständig.
- eCH-0158: "Ein Prozess hat mindestens ein Startereignis."
- Den Fehler beheben: Um den Fehler zu beheben fügen sie ein Ereignis als Startereignis zu dem Prozess hinzu und verbinden es über einen Sequenzfluss mit dem Objekt,
mit dem der Prozess beginnen soll.
                            </German>
                            <English />
                        </Message>
                        <MessageKind>Warning</MessageKind>
                        <Parameters />
                        <Stereotypes>
                            <VerifyStereotype>
                                <Metatype>BPProcess</Metatype>
                                <Profile>ROOT PROFILE</Profile>
                                <Stereotype>process</Stereotype>
                                <Id>[eCH0158-Startereignis-06]</Id>
                                <IsActive>true</IsActive>
                            </VerifyStereotype>
                        </Stereotypes>
                    </SingleVerification>

                </Singles>
            </VerificationGroup>
```

Abb. 8-11: Dokument „inoverify.xml“ (4/11).

```
            <VerificationGroup>
                <Name>
                    <German>"Angemessene Aussagekraft" überprüfen</German>
                    <English>Check suitable Meaning</English>
                </Name>
                <Singles>
                    <!-- Genau 2 Wörter für die Bennenung einer Aktivität. -->
                    <SingleVerification xsi:type="JavaGenericVerification">
                        <Name>
                            <German>Genau 2 Wörter für die Bennenung einer Aktivität.</German>
                            <English>Two words for the name of an Activity.</English>
                        </Name>
                        <Type>JavaGeneric</Type>
                        <Description>
                            <German>Prüfung zur Anzahl der Wörter im Namen einer Aktivität.</German>
                            <English>Checks whether an activity is named with two words.</English>
                        </Description>
                        <Check>de.mid.engineering.innovator.verify.Check_ANY_ELNamedElement_NumberOfWordsInDisplayName</Check>
                        <Message>
                            <German>Der Name der Aktivität enthält zu viele oder zu wenige Wörter. Er sollte nach eCH-0158 aus 2 Wörtern bestehen.
- eCH-0158: "Aktivitäten werden immer mit einem vorangestellten Substantiv und einem Verb in der Grundform (Infinitiv) benannt. Beispiel: „Korrekturen begründen"."
- Den Fehler beheben: Formulieren Sie die Bezeichnung um, sodass sie aus zwei Wörtern besteht.
                            </German>
                            <English />
                        </Message>
                        <MessageKind>Warning</MessageKind>
                        <Parameters>
                            <VerifyParameter>
                                <Name>numberOfWords</Name>
                                <Value>2</Value>
                            </VerifyParameter>
                        </Parameters>
                        <Stereotypes>
                            <VerifyStereotype>
                                <Metatype>BPTaskNode</Metatype>
                                <Profile>ROOT PROFILE</Profile>
                                <Stereotype>task</Stereotype>
                                <Id>[eCH0158-Aktivität-01]</Id>
                                <IsActive>true</IsActive>
                            </VerifyStereotype>
                        </Stereotypes>
                    </SingleVerification>
```

Abb. 8-12: Dokument „inoverify.xml“ (5/11).

```
                        <!-- Unbestimmtes Startereignis immer unbenannt. -->
                        <SingleVerification xsi:type="JavaGenericVerification">
                            <Name>
                                <German>Unbestimmtes Startereignis immer unbenannt.</German>
                                <English>Undefined startevent always unnamed.</English>
                            </Name>
                            <Type>JavaGeneric</Type>
                            <Description>
                                <German>Prüfung zur Bennenung eines unbestimmten Startereignis.</German>
                                <English>Checks whether the undefined startevent has no name.</English>
                            </Description>
                            <Check>de.mid.engineering.innovator.verify.Check_BPMN_BPEventNode_UnspecifiedStartEventNodeHasNoName</Check>
                            <Message>
                                <German>Das unbestimmte Startereignis trägt einen Namen. Nach eCH-0158 soll diese Ereignis unbennant sein.
- eCH-0158: "Unbestimmtes Startereignis: Wird nicht beschrieben, der Unterprozess wird durch den Aufruf auf der übergeordneten Prozessebene gestartet."
- Den Fehler beheben: Löschen Sie den eingetragenen Namen in den Eigenschaften des Objekts.
                                </German>
                                <English />
                            </Message>
                            <MessageKind>Warning</MessageKind>
                            <Parameters />
                            <Stereotypes>
                                <VerifyStereotype>
                                    <Metatype>BPEventNode</Metatype>
                                    <Profile>ROOT PROFILE</Profile>
                                    <Stereotype>event</Stereotype>
                                    <Id>[eCH0158-Startereignis-01]</Id>
                                    <IsActive>true</IsActive>
                                </VerifyStereotype>
                            </Stereotypes>
                        </SingleVerification>
```

Abb. 8-13: Dokument „inoverify.xml“ (6/11).

```
                    <!-- Bedingungs Startereignis ist definiert. -->
                    <SingleVerification xsi:type="JavaGenericVerification">
                        <Name>
                            <German>Bedingtes Startereignis mit Definition der Bedingungsdefinition.</German>
                            <English>Conditional startevent with definition of conditionaldefinition.</English>
                        </Name>
                        <Type>JavaGeneric</Type>
                        <Description>
                            <German>Prüft ob für das bedingte Startereignis eine Definition in der Ereignisdefinition angegeben ist.</German>
                            <English>Checks whether the conditional startevent has definition for the conditionaldefinition.</English>
                        </Description>
                        <Check>de.mid.engineering.innovator.verify.Check_BPMN_PBEventNode_ConditionalStartEventNodeHasDefinition</Check>
                        <Message>
                            <German>Das bedingte Startereignis ist nicht definiert.
- eCH-0158: Bedingungs-Startereignis: "Bedingung ist in der Bezeichnung des Elementes festzuhalten. Sollte ein Bedingungs-Startereignis einem Endereignis eines anderen
Prozesses entsprechen, ist es identisch zu bezeichnen."
- Den Fehler beheben: Geben Sie eine Definition (in diesem Fall: Bedingung) in der Ereignisdefinition des bedingten Startereignisses an.
                            </German>
                            <English />
                        </Message>
                        <MessageKind>Warning</MessageKind>
                        <Parameters />
                        <Stereotypes>
                            <VerifyStereotype>
                                <Metatype>BPEventNode</Metatype>
                                <Profile>ROOT PROFILE</Profile>
                                <Stereotype>event</Stereotype>
                                <Id>[eCH0158-Startereignis-03]</Id>
                                <IsActive>true</IsActive>
                            </VerifyStereotype>
                        </Stereotypes>
                    </SingleVerification>
```

Abb. 8-14: Dokument „inoverify.xml“ (7/11).

```
                    <!-- Zeitliches Startereignis ist definiert. -->
                    <SingleVerification xsi:type="JavaGenericVerification">
                        <Name>
                            <German>Zeitliches Startereignis mit Definition der Zeitgeberdefinition.</German>
                            <English>Timer-Startevent with Definition Timerdefinition.</English>
                        </Name>
                        <Type>JavaGeneric</Type>
                        <Description>
                            <German>Prüft ob für das zeitliche Startereignis eine Definition in der Ereignisdefinition angegeben ist.</German>
                            <English>Checks whether the timer startevent has definition for the timerdefinition.</English>
                        </Description>
                        <Check>de.mid.engineering.innovator.verify.Check_BPMN_PBEventNode_TimerStartEventNodeHasDefinition</Check>
                        <Message>
                            <German>Das zeitliche Startereignis ist nicht definiert.
- eCH-0158: Zeitgeber-Startereignis: "Bezeichnung enthält den Zeitpunkt für den Start. Beispiele: „am 1. je Monat", „09:00 Uhr"."
- Den Fehler beheben: Geben Sie eine Definition (in diesem Fall: Zeitpunkt) in der Ereignisdefinition des zeitlichen Startereignisses an.
                            </German>
                            <English />
                        </Message>
                        <MessageKind>Warning</MessageKind>
                        <Parameters />
                        <Stereotypes>
                            <VerifyStereotype>
                                <Metatype>BPEventNode</Metatype>
                                <Profile>ROOT PROFILE</Profile>
                                <Stereotype>event</Stereotype>
                                <Id>[eCH0158-Startereignis-04]</Id>
                                <IsActive>true</IsActive>
                            </VerifyStereotype>
                        </Stereotypes>
                    </SingleVerification>

                </Singles>
            </VerificationGroup>
            <VerificationGroup>
                <Name>
                    <German>"Qualität der SRS-Modellvisualisierungen" überprüfen</German>
                    <English>Check Quality of the Diagram</English>
                </Name>
                <Singles>
```

Abb. 8-15: Dokument „inoverify.xml“ (8/11).

```
                        <!-- Maximal 15 Aktivitäten pro BPMN-Diagramm. -->
                        <SingleVerification xsi:type="JavaGenericVerification">
                            <Name>
                                <German>Maximal 15 Aktivitäten pro BPMN-Diagramm.</German>
                                <English>No more than 15 activities per BPMN-diagram.</English>
                            </Name>
                            <Type>JavaGeneric</Type>
                            <Description>
                                <German>Prüfung zur Anzahl der Aktivitätsknoten in einem BPMN-Diagramm.
(Hintergrund: Bei einer großen Anzahl von Aktivitätsknoten wird das Diagramm sehr groß und komplex, daher unübersichtlich).</German>
                                <English>Checks the number of activties in a BPMN-diagram.</English>
                            </Description>
                            <Check>de.mid.engineering.innovator.verify.Check_BPMN_BPDia_NumberOfActivities</Check>
                            <Message>
                                <German>In diesem BPMN-Diagramm werden mehr als die laut eCH-0158 Standard erlaubten maximalen 15 Aktivitäten dargestellt.
- eCH-0158: "Maximal 9 - 15 Aktivitäten (Ausprägungen siehe Kapitel 4.6) pro Diagramm."
- Den Fehler beheben: Sie können dies anpassen in dem sie mehrere Taks in einem eigenen Diagramm darstellen (als Unterprozess) und durch eine Aufrufaktivität in diesem
Diagramm auf diesen Unterprozess verweisen.
                                </German>
                                <English />
                            </Message>
                            <MessageKind>Warning</MessageKind>
                            <Parameters>
                                <VerifyParameter>
                                    <Name>minimumOrMaximum</Name>
                                    <Value>maximum</Value>
                                </VerifyParameter>
                                <VerifyParameter>
                                    <Name>numberOfActivities</Name>
                                    <Value>15</Value>
                                </VerifyParameter>
                            </Parameters>
                            <Stereotypes>
                                <VerifyStereotype>
                                    <Metatype>BPDia</Metatype>
                                    <Profile>ROOT PROFILE</Profile>
                                    <Stereotype>bpmnDia</Stereotype>
                                    <Id>[eCH0158-Diagramm-06]</Id>
                                    <IsActive>true</IsActive>
                                </VerifyStereotype>
                            </Stereotypes>
                        </SingleVerification>
```

Abb. 8-16: Dokument „inoverify.xml“ (9/11).

```
                    <!-- Maximal 9 Aktivitäten pro BPMN-Diagramm. -->
                    <SingleVerification xsi:type="JavaGenericVerification">
                        <Name>
                            <German>Maximal 9 Aktivitäten pro BPMN-Diagramm.</German>
                            <English>No more than 9 activities per BPMN-diagram.</English>
                        </Name>
                        <Type>JavaGeneric</Type>
                        <Description>
                            <German>Prüfung zur Anzahl der Aktivitätsknoten in einem BPMN-Diagramm.
(Hintergrund: Bei einer großen Anzahl von Aktivitätsknoten wird das Diagramm sehr groß und komplex, daher unübersichtlich).</German>
                            <English>Checks the number of activties in a BPMN-diagram.</English>
                        </Description>
                        <Check>de.mid.engineering.innovator.verify.Check_BPMN_BPDia_NumberOfActivities</Check>
                        <Message>
                            <German>In diesem BPMN-Diagramm werden mehr als die laut eCH-0158 Standard empfohlenen maximalen 9 Aktivitäten dargestellt.
- eCH-0158: "Maximal 9 - 15 Aktivitäten (Ausprägungen siehe Kapitel 4.6) pro Diagramm."
- Den Fehler beheben: Sie können dies anpassen in dem sie mehrere Taks in einem eigenen Diagramm darstellen (als Unterprozess) und durch eine Aufrufaktivität in diesem
Diagramm auf diesen Unterprozess verweisen.
                            </German>
                            <English />
                        </Message>
                        <MessageKind>Info</MessageKind>
                        <Parameters>
                            <VerifyParameter>
                                <Name>minimumOrMaximum</Name>
                                <Value>maximum</Value>
                            </VerifyParameter>
                            <VerifyParameter>
                                <Name>numberOfActivities</Name>
                                <Value>9</Value>
                            </VerifyParameter>
                        </Parameters>
                        <Stereotypes>
                            <VerifyStereotype>
                                <Metatype>BPDia</Metatype>
                                <Profile>ROOT PROFILE</Profile>
                                <Stereotype>bpmnDia</Stereotype>
                                <Id>[eCH0158-Diagramm-06]</Id>
                                <IsActive>true</IsActive>
                            </VerifyStereotype>
                        </Stereotypes>
                    </SingleVerification>
```

Abb. 8-17: Dokument „inoverify.xml“ (10/11).

```
                </Singles>
            </VerificationGroup>

            <VerificationGroup>
                <Name>
                    <German>Java Sample Verifications</German>
                    <English>Java Sample Verifications</English>
                </Name>
                <Singles>
                    <!-- Hierbei handelt es sich um ein Workaround. Die Java Runtime wird
                        (bis einschließlich 11R5 SR2 C = 11.5.3.31206) nicht geladen, wenn in inoverify.xml
                        nur Generische JavaChecks drin sind. Daher ist hier auch noch ein nicht generischer
                        Check enhalten. Dieser kann dann in einer der nächsten Versionen entnommen
                        werden. -->
                    <SingleVerification xsi:type="JavaVerification">
                        <Name>
                            <German>UseCase Namenslänge</German>
                            <English></English>
                        </Name>
                        <Type>Java</Type>
                        <Description>
                            <German></German>
                            <English></English>
                        </Description>
                        <IsActive>true</IsActive>
                        <Check>de.mid.engineering.innovator.pruefmanager.check.custom.demo.UseCaseNamenslaenge</Check>
                    </SingleVerification>
                </Singles>
            </VerificationGroup>

        </Groups>
    </VerificationAction>
</Actions>
</VerificationConfiguration>
```

Abb. 8-18: Dokument „inoverify.xml“ (11/11).

```
import java.util.LinkedList;
import java.util.List;

import de.mid.innovator.client.InoClientContext;
import de.mid.innovator.net.InoNetException;
import de.mid.innovator.srv.SrvErrorException;
import de.mid.innovator.srv2api.icw2elem.ELElement;
import de.mid.innovator.srv2api.icw2model.MEModel;
import de.mid.innovator.srv2api.icw2model.MERequirement;
import de.mid.innovator.srv2api.icw2model.MEStereotypeAble;
import de.mid.innovator.ui.InnovatorApplication;
import de.mid.innovator.ui.InnovatorApplicationDefault;
import de.mid.innovator.ui.Property;
import de.mid.innovator.ui.PropertyCfg;
import de.mid.innovator.util.InoNlsException;

public class MetricsAboutRequirements extends InnovatorApplicationDefault
        implements InnovatorApplication {

    public String Usage() {
        return null;
    }

    private final static String PARAMETER_1 = "parameter_1";

    private static Property[] argumentsCfg = { new PropertyCfg<String>(
            PARAMETER_1, "Der erste Parameter.") };

    public static void main(String[] args) throws Exception {
        InnovatorApplicationDefault.create(MetricsAboutRequirements.class,
                argumentsCfg, args);
    }
    public void run() throws InoNlsException {
        List<ELElement> sel = InoClientContext.getInstance().getSelection();

        for (ELElement element : sel) {
            if (element instanceof MEModel) {

                MEModel model = (MEModel) element;
                List<MERequirement> requirements = model
                        .getOwnedElementTransitiveDown(MERequirement.class);
                List<MERequirement> spezifiedRequirements = new LinkedList<MERequirement>();
                List<MERequirement> documentedRequirements = new LinkedList<MERequirement>();
                int requirementsWithoutSource = 0;
                int requirementsWithoutStakeholder = 0;
                int requirementsWithoutPriority = 0;
                double numberOfrequirements = requirements.size();

                for (MERequirement requirement : requirements) {
                    boolean hasSource = Check_Inno_MERequirement_HasSource(requirement);
                    boolean hasStakeholder = Check_Inno_MERequirement_HasStakeholder(requirement);
                    boolean hasPriority = Check_Inno_MERequirement_HasPriority(requirement);

                    if (!hasStakeholder)
                        requirementsWithoutStakeholder++;
                    if (!hasPriority)
                        requirementsWithoutPriority++;
                    if (!hasSource)
                        requirementsWithoutSource++;

                    if (hasStakeholder && hasPriority && hasSource) {
                        spezifiedRequirements.add(requirement);
                    } else {
                        documentedRequirements.add(requirement);
                    }
                }
```

Abb. 8-19: Code der Metrik-Berechnung bzgl. Anforderungen (1/2).

```
                // AUSGABE
                System.out.println(">> METRIKEN ZU ANFORDERUNGEN <<");
                System.out.println();
                System.out
                        .println("#Anforderungen ohne Quelle (nicht verfolgbar, REoQ) = "
                                + requirementsWithoutSource);
                System.out.println("#Anforderungen ohne Stakeholder (REoS) = "
                        + requirementsWithoutStakeholder);
                System.out
                        .println("#Anforderungen ohne Priorität (nicht bewertet, REoB) = "
                                + requirementsWithoutPriority);
                System.out.println();
                System.out
                        .println("#Spezifizierte Anforderungen ÷ #Anforderungen (sRE/RE) = "
                                + (spezifiedRequirements.size()
                                        / numberOfrequirements * 100) + " %");
                System.out.println();
                System.out.println("----");
                System.out.println("EINZELGRÖßEN: ");
                System.out.println("  #Anforderungen = "
                        + (int) numberOfrequirements);
                System.out
                        .println("    > darunter #Dokumentierte Anforderungen = "
                                + documentedRequirements.size());
                System.out
                        .println("    > darunter #Spezifizierte Anforderungen = "
                                + spezifiedRequirements.size());
            } else {
                System.out
                        .println("Fehler: Das selektierte Element ist kein Modell.");
            }
        }

    }
    private boolean Check_Inno_MERequirement_HasSource(MERequirement requirement)
            throws InoNetException, SrvErrorException {
        if (requirement.getReqSource().isEmpty()) {
            return false;
        } else {
            return true;
        }
    }

    private boolean Check_Inno_MERequirement_HasStakeholder(
            MERequirement requirement) throws InoNetException,
            SrvErrorException {
        if (requirement.getStakeholderOfRequirement(MEStereotypeAble.class)
                .isEmpty()) {
            return false;
        } else {
            return true;
        }
    }

    private boolean Check_Inno_MERequirement_HasPriority(
            MERequirement requirement) throws InoNetException,
            SrvErrorException {
        if (requirement.getPropStringValue(null, "Priorität").contentEquals(
                "Noch festzulegen")) {
            return false;
        } else {
            return true;
        }
    }
}
```

Abb. 8-20: Code der Metrik-Berechnung bzgl. Anforderungen (2/2).

8.3 DIAGRAMME

Die folgenden Abbildungen (Abb. 8-21, S. 161 bis Abb. 8-27, S. 165) sind Beispiel-Diagramme für unterschiedliche Diagrammarten gezeigt. Sie wurden in Innovator erstellt und sollen einen Eindruck der im Modellierungswerkzeug angebotenen Diagrammarten vermitteln.

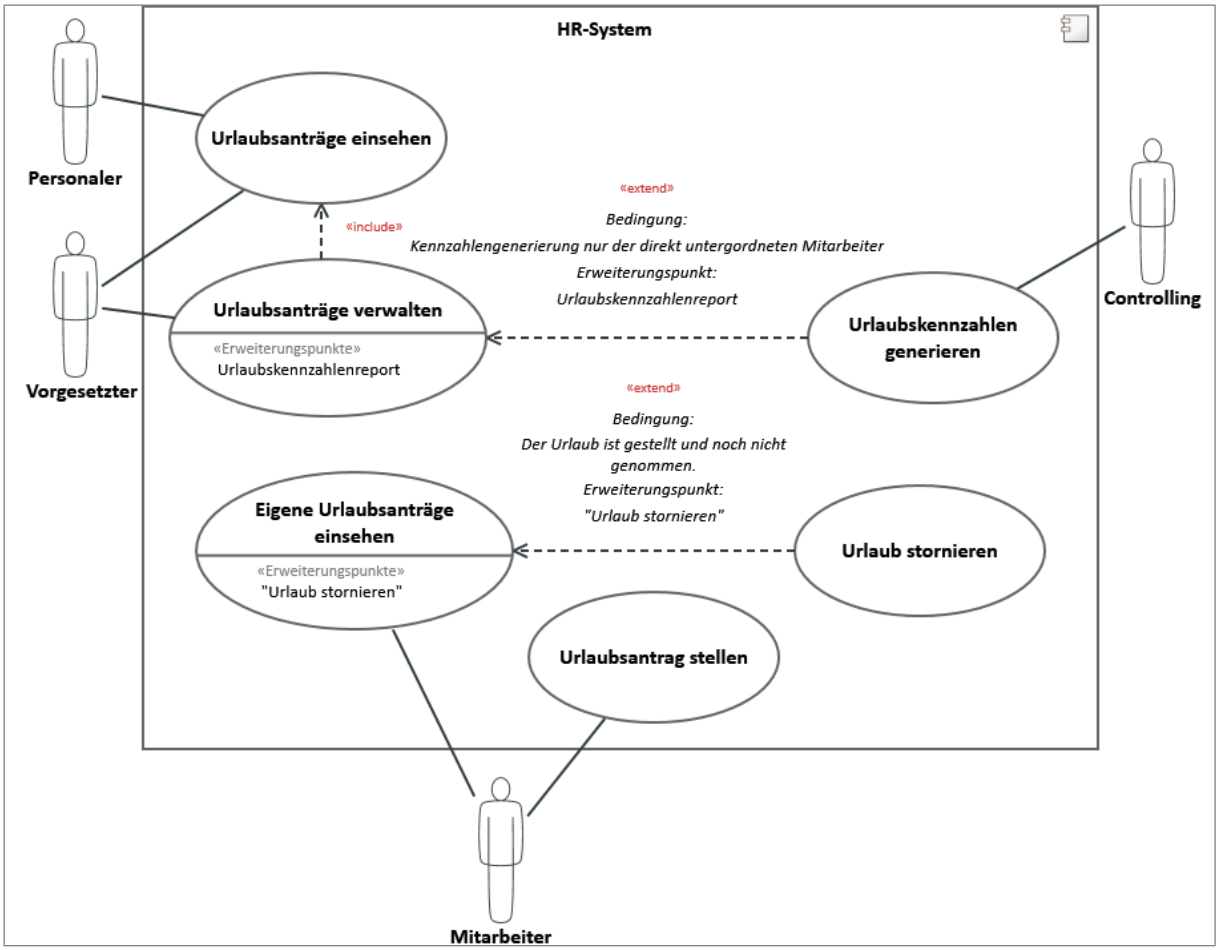


Abb. 8-21: Bsp. *Anwendungsfalldiagramm* „Urlaubsverwaltung“.

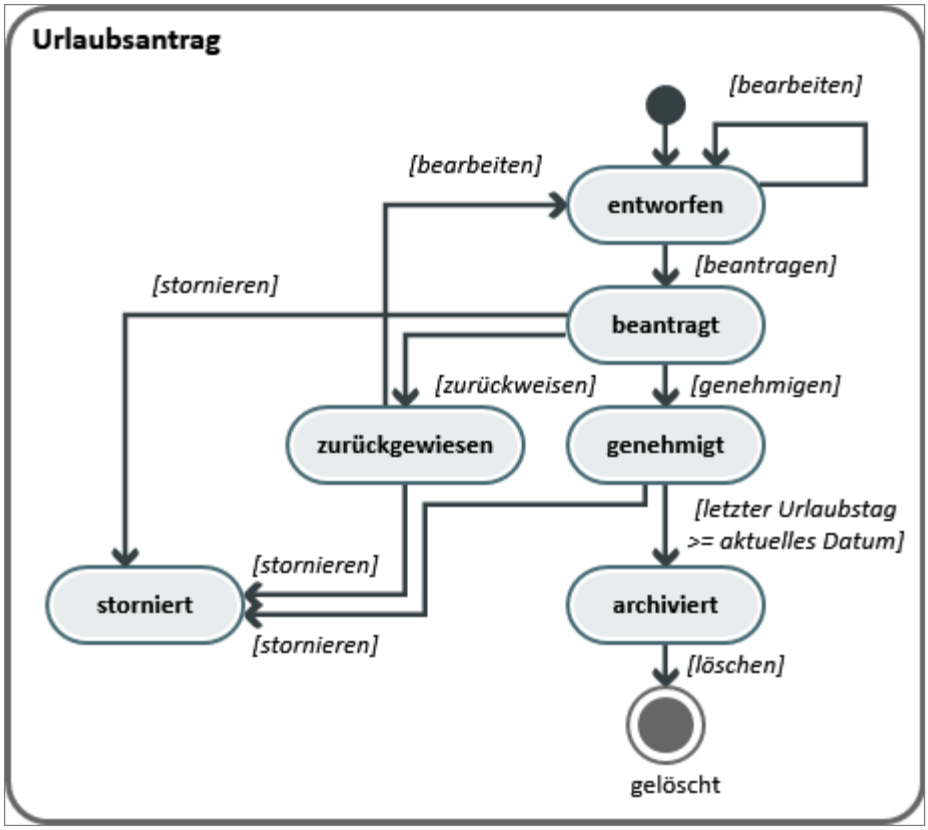


Abb. 8-22: Bsp. *Diagramm für Geschäftsobjektzustände* „Urlaubsantrag“.

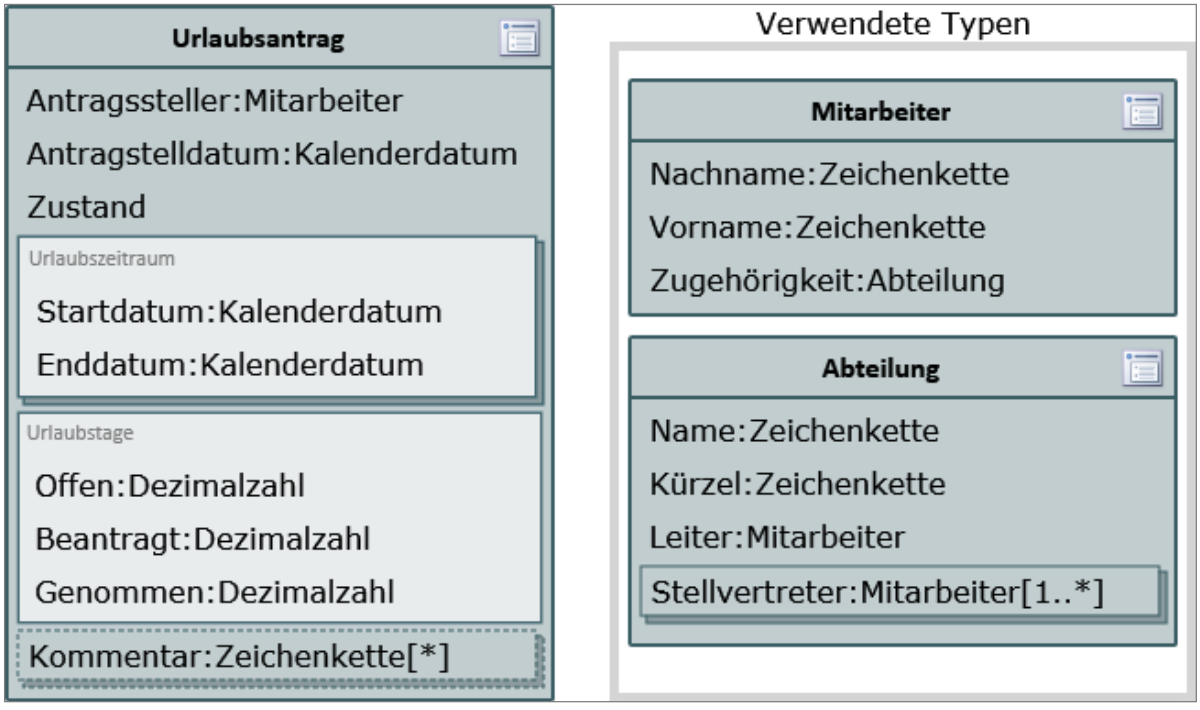


Abb. 8-23: Bsp. *Strukturdiagramm* „Urlaubsantrag“.

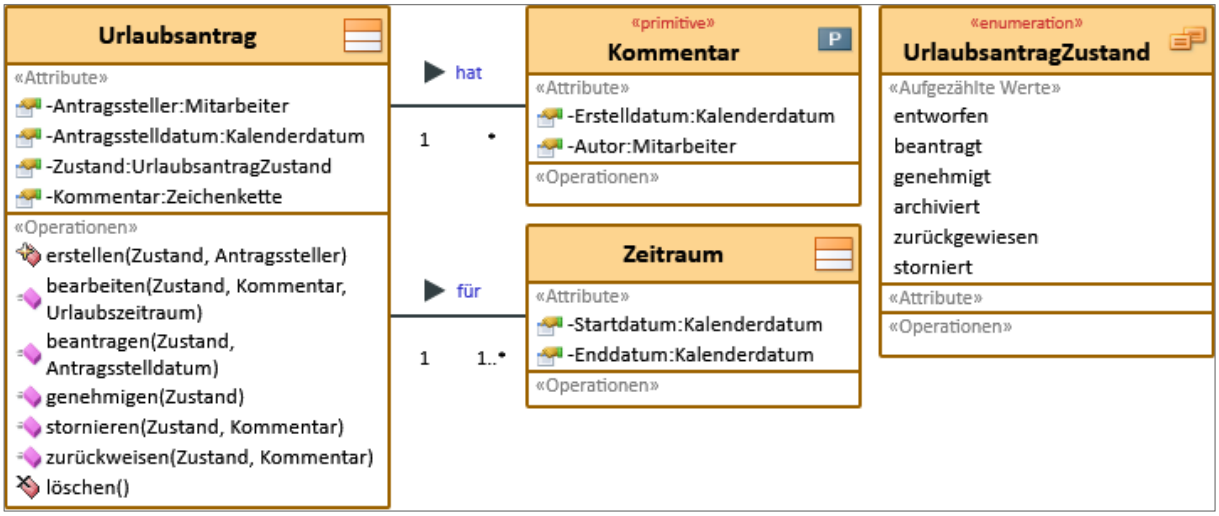


Abb. 8-24: Bsp. *Klassendiagramm* „Urlaubsantrag“.

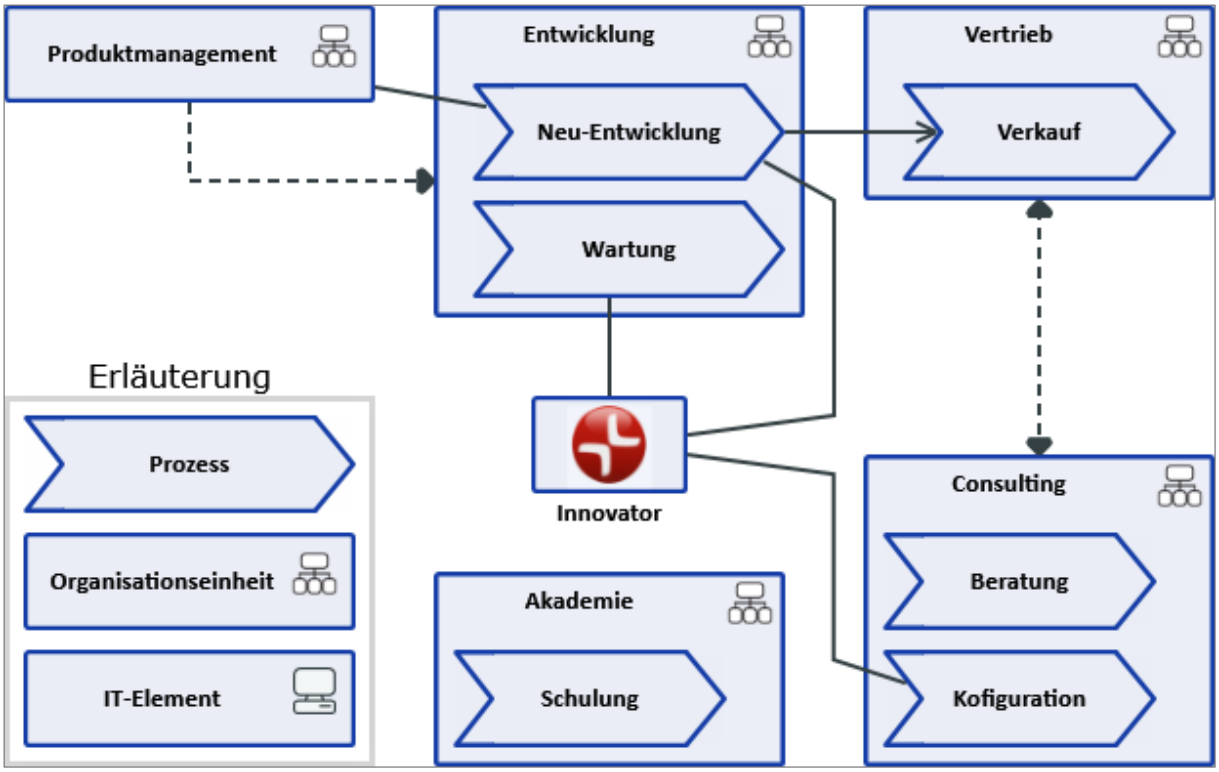


Abb. 8-25: Bsp. *Prozesslandkarte* „Geschäftsprozesse“.[73]

[73] *Erläuterung*: Gestrichelte Linie bedeutet Informationsfluss; Einfache Linie mit Pfeil bedeutet Prozesssequenz; Einfache Linie bedeutet Ressourcenzuordnung.

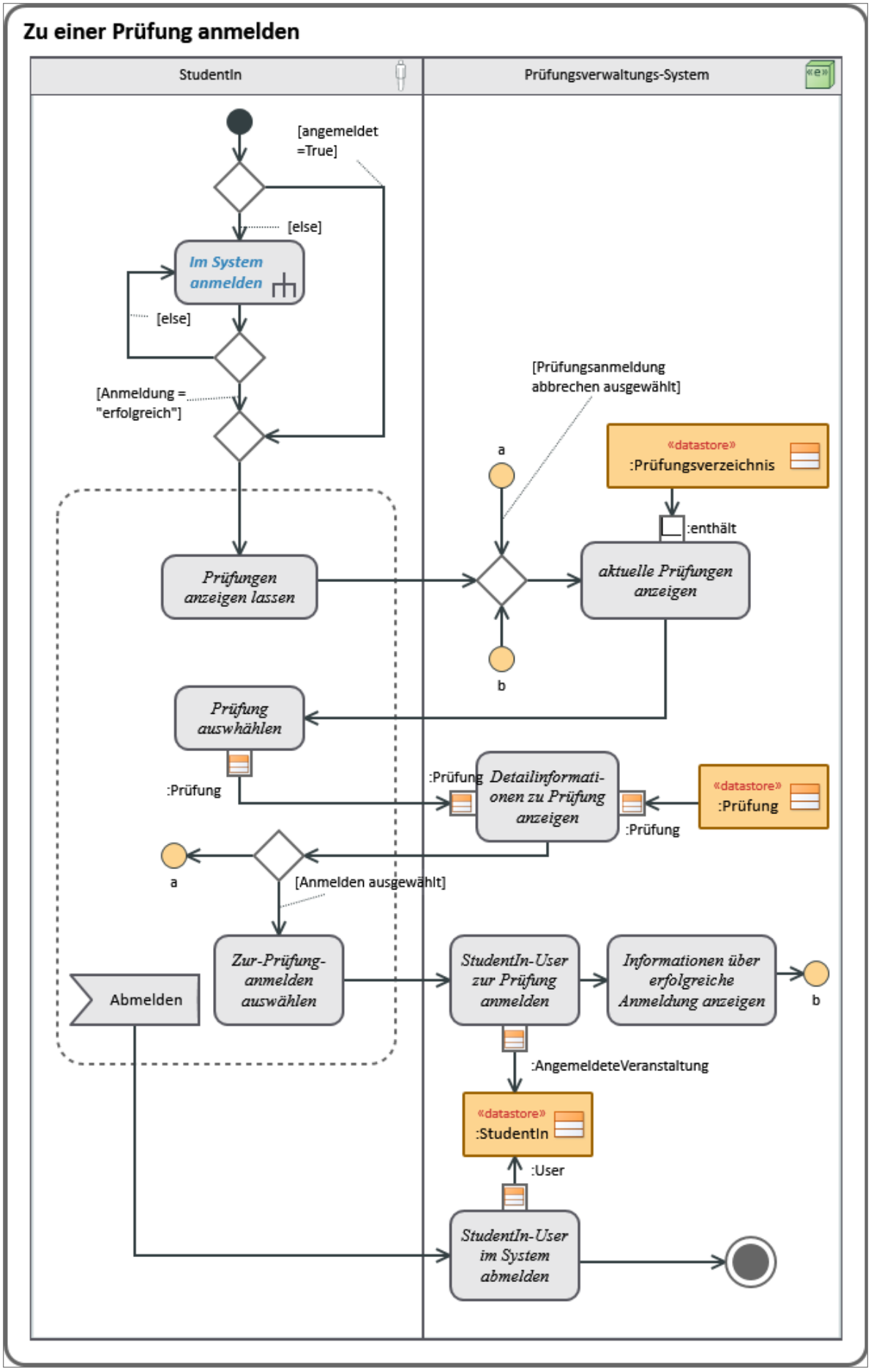


Abb. 8-26: Bsp. *Aktivitätsdiagramm* „Zu einer Prüfung anmelden“.

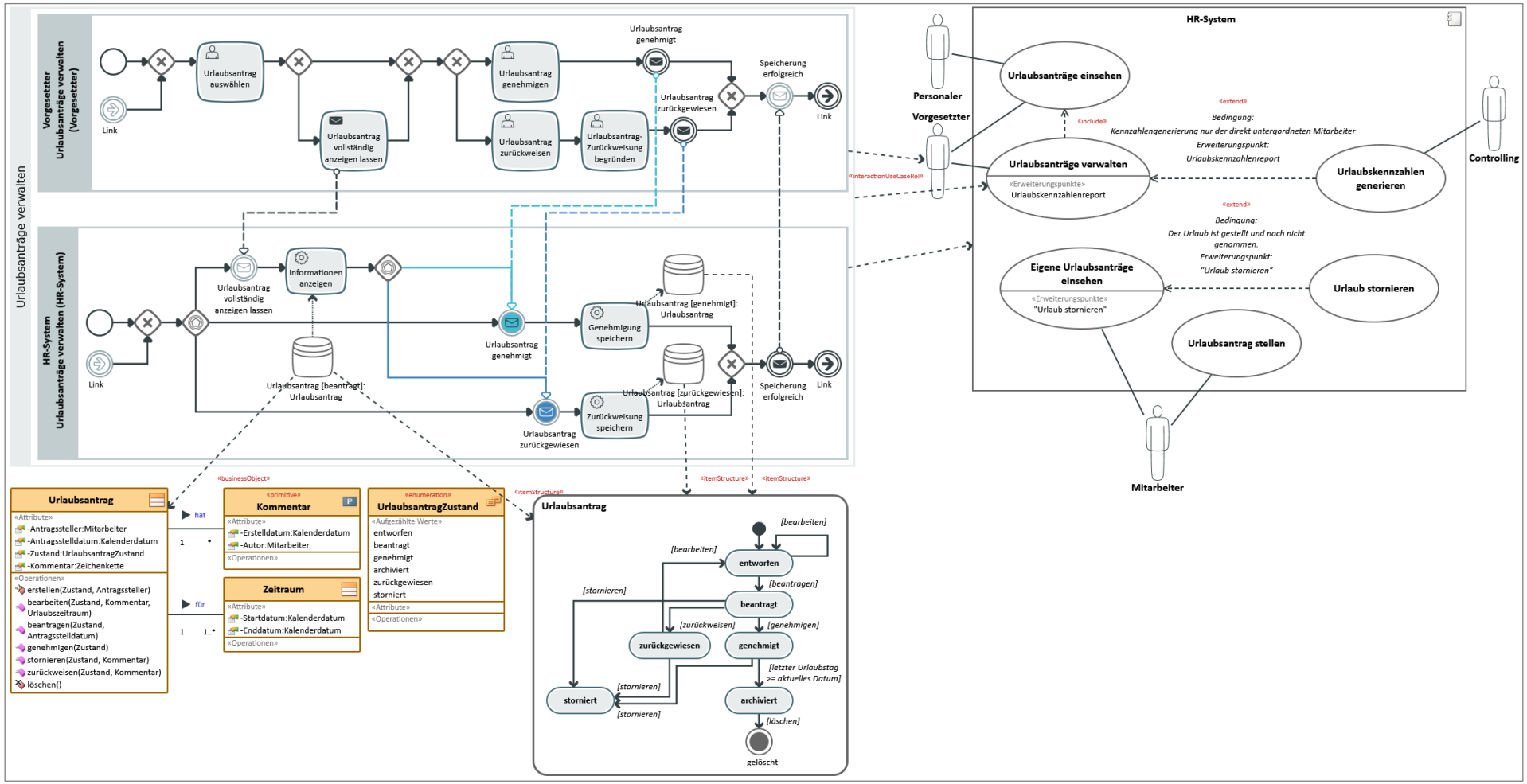


Abb. 8-27: Bsp. *Whiteboard-Diagramm* „Urlaubsverwaltung“.

8.4 Screenshots

Die beispielhafte Überprüfung von BPMN-Diagramminhalten auf Einhaltung der eCH-Modellierungskonventionen wird in den folgenden Screenshots gezeigt (siehe Abb. 8-28, S. 166 bis Abb. 8-34, S. 171). Die Überprüfung wurde anhand der entwickelten Prüfungen vorgenommen. Im oberen Bildausschnitt jeder Abbildung ist der Diagrammabschnitt zu sehen, welcher gegen eCH-0158-Konventionen verstößt. In der unteren Bildhälfte sind die Prüfergebnisse der eCH-0158-Prüfung der Diagramminhalte aufgeführt sowie die Prüfmeldung mit der Beschreibung des Qualitätsdefizits, der zugehörigen Konvention und einer Anleitung zur Behebung des Fehlers.

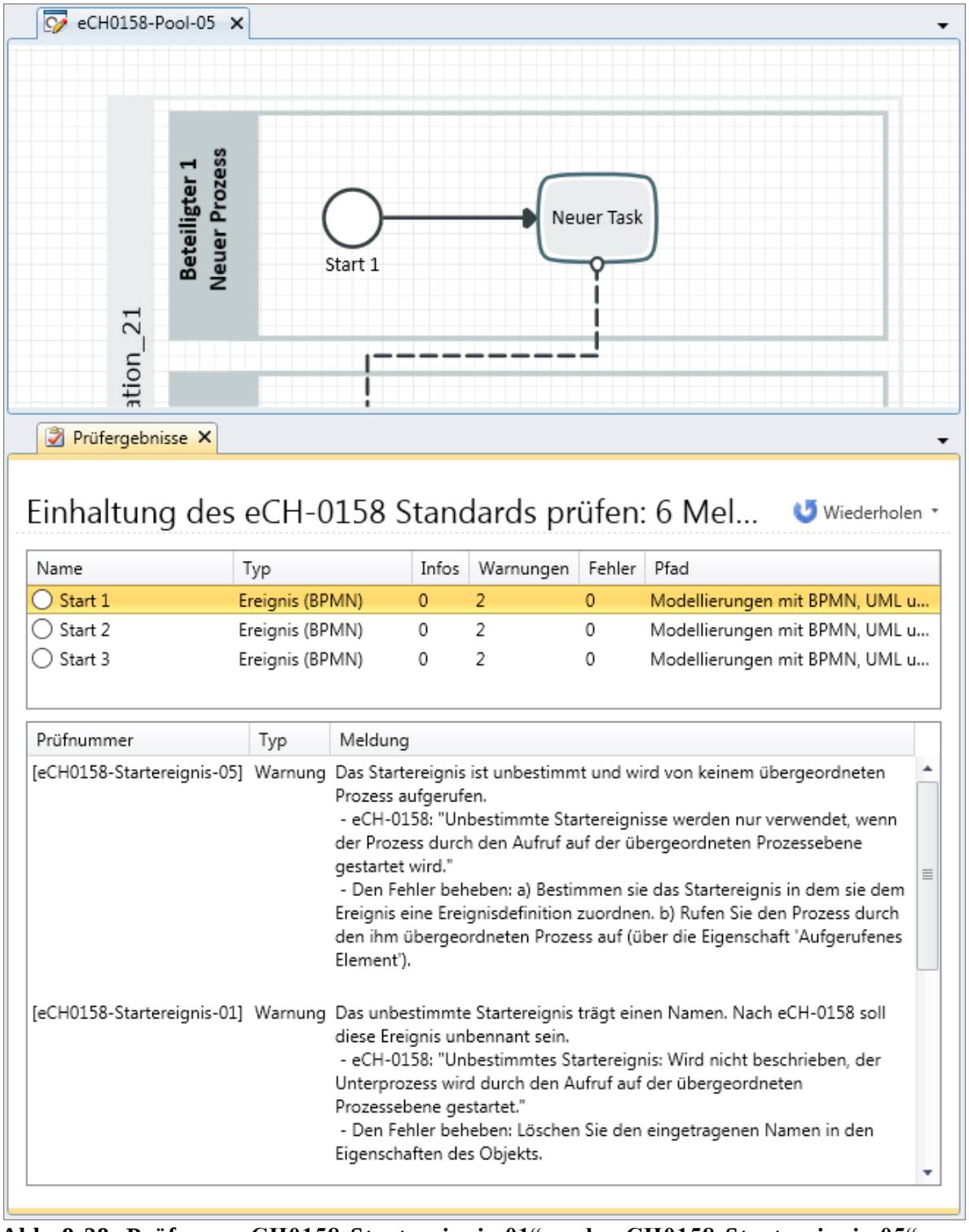


Abb. 8-28: Prüfung „eCH0158-Startereignis-01“ und „eCH0158-Startereignis-05“.

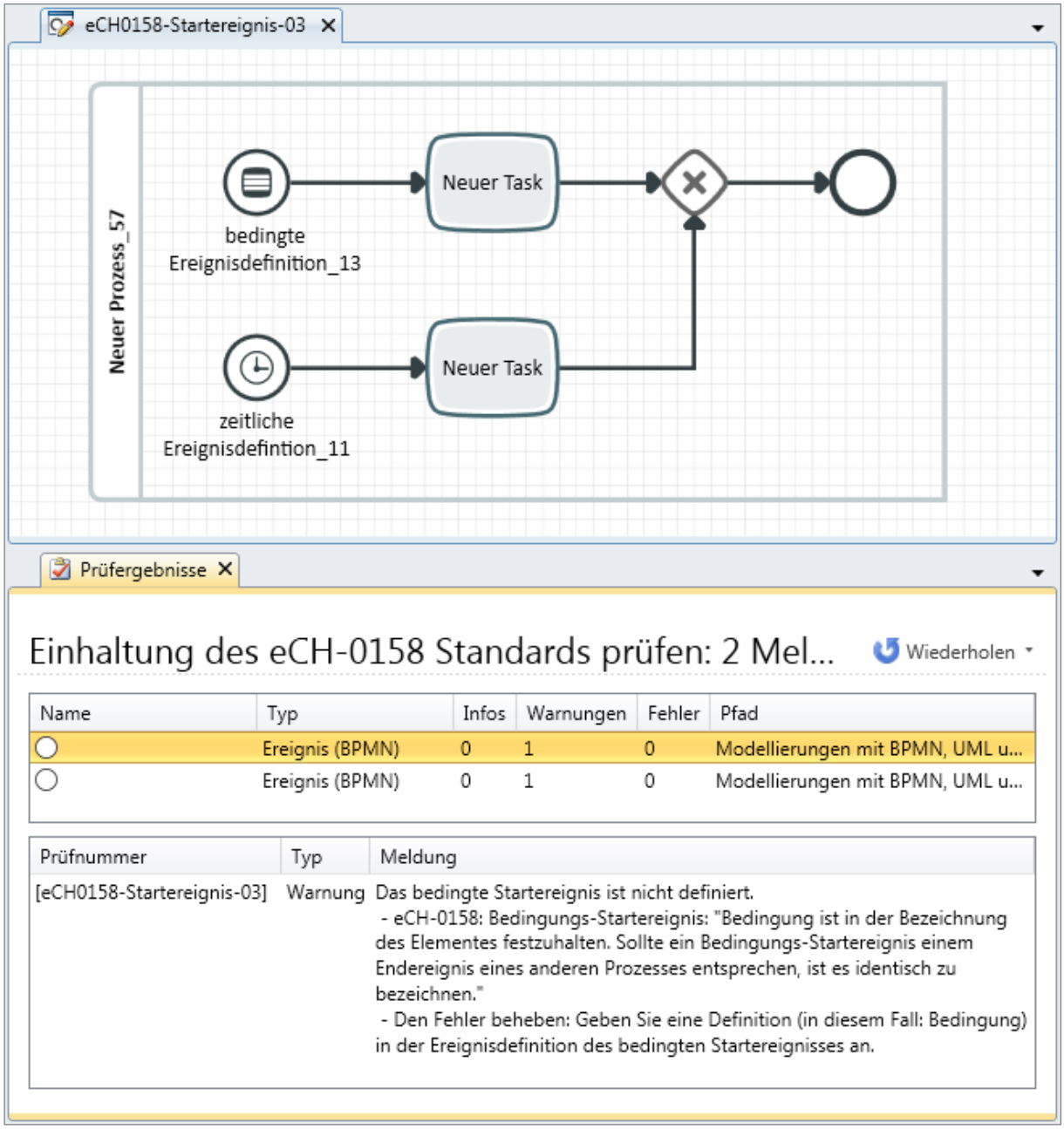


Abb. 8-29: Prüfung „eCH0158-Startereignis-03“

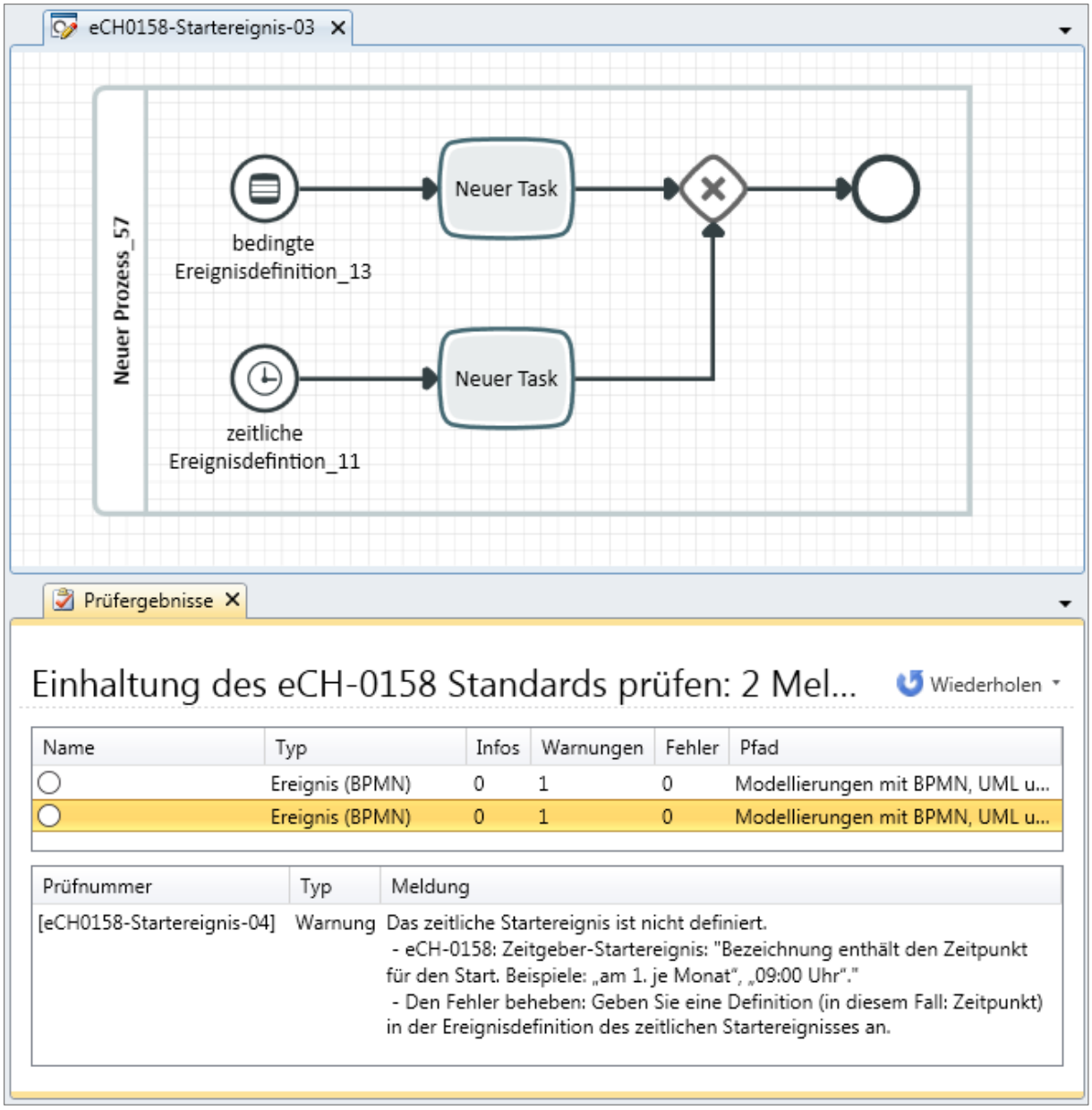


Abb. 8-30: Prüfung „eCH0158-Startereignis-04".

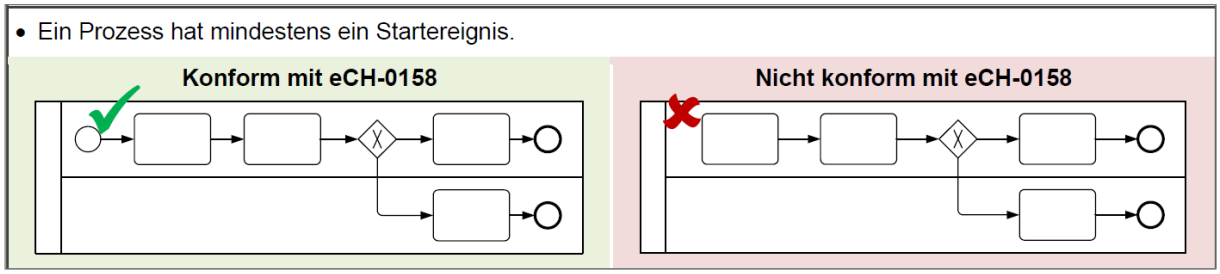


Abb. 8-31: Konvention „eCH0158-Startereignis-06“ (eCH, 2013).

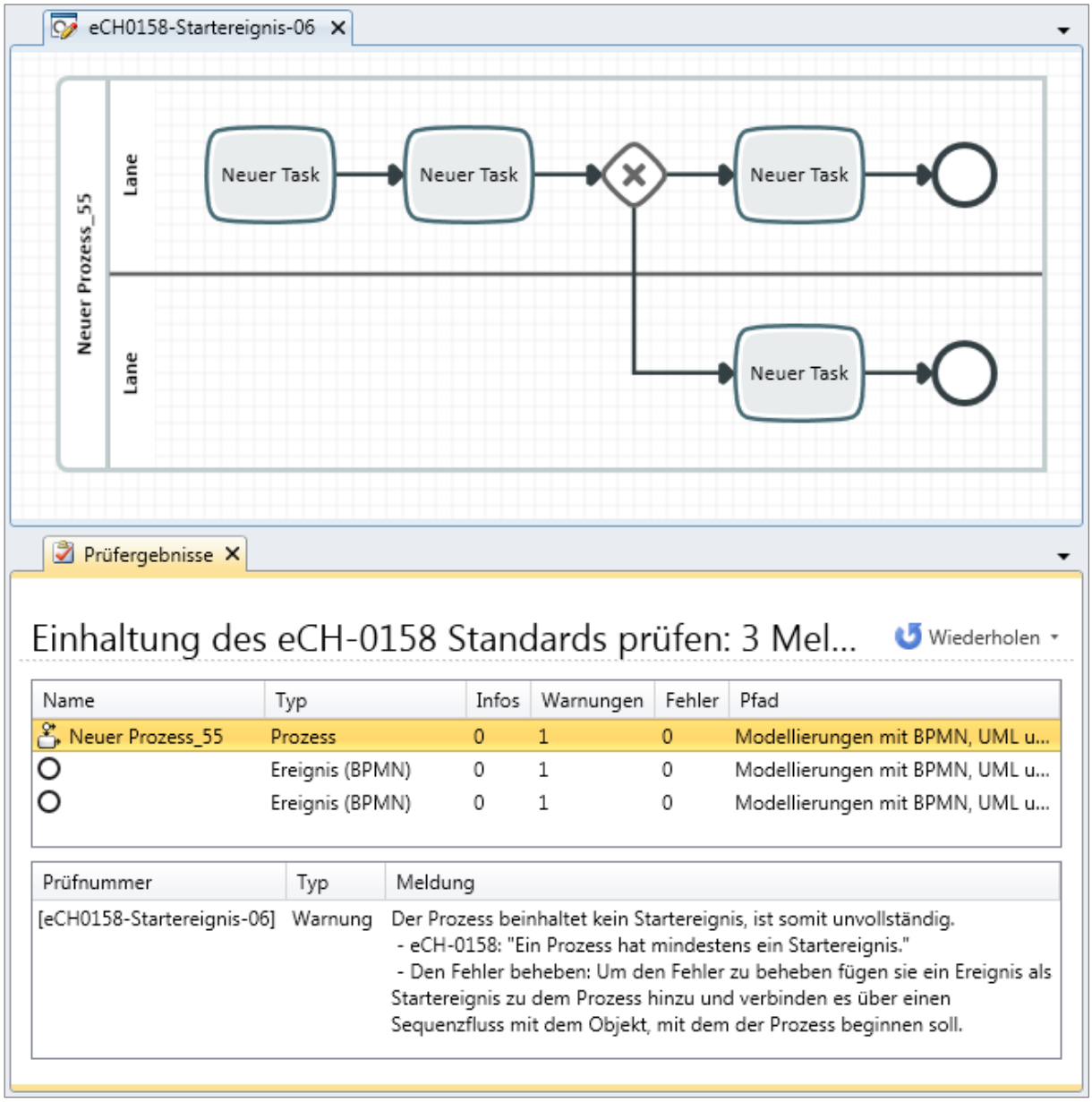


Abb. 8-32: Prüfung „eCH0158-Startereignis-06“.

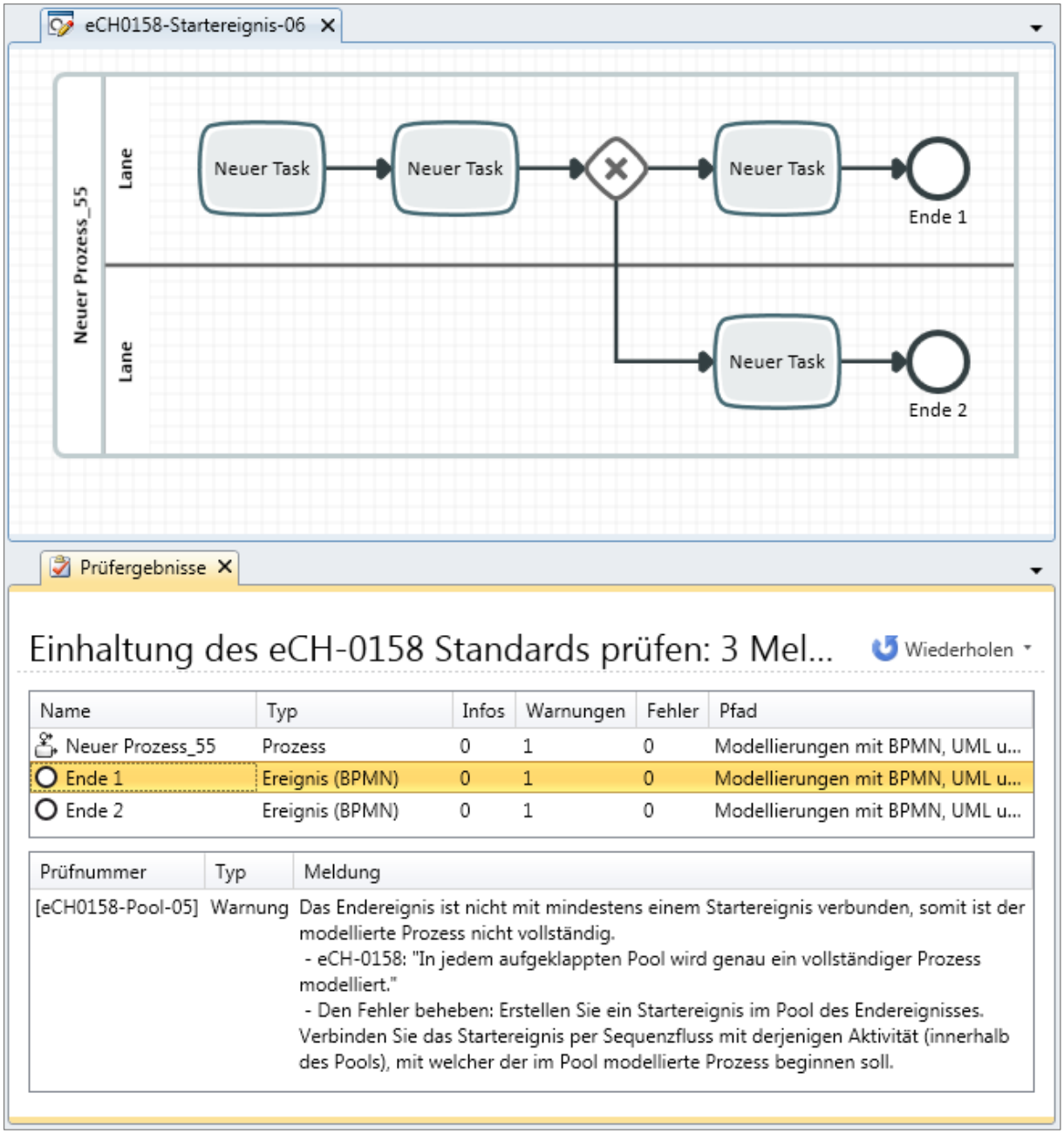


Abb. 8-33: Prüfung „eCH0158-Pool-05“.

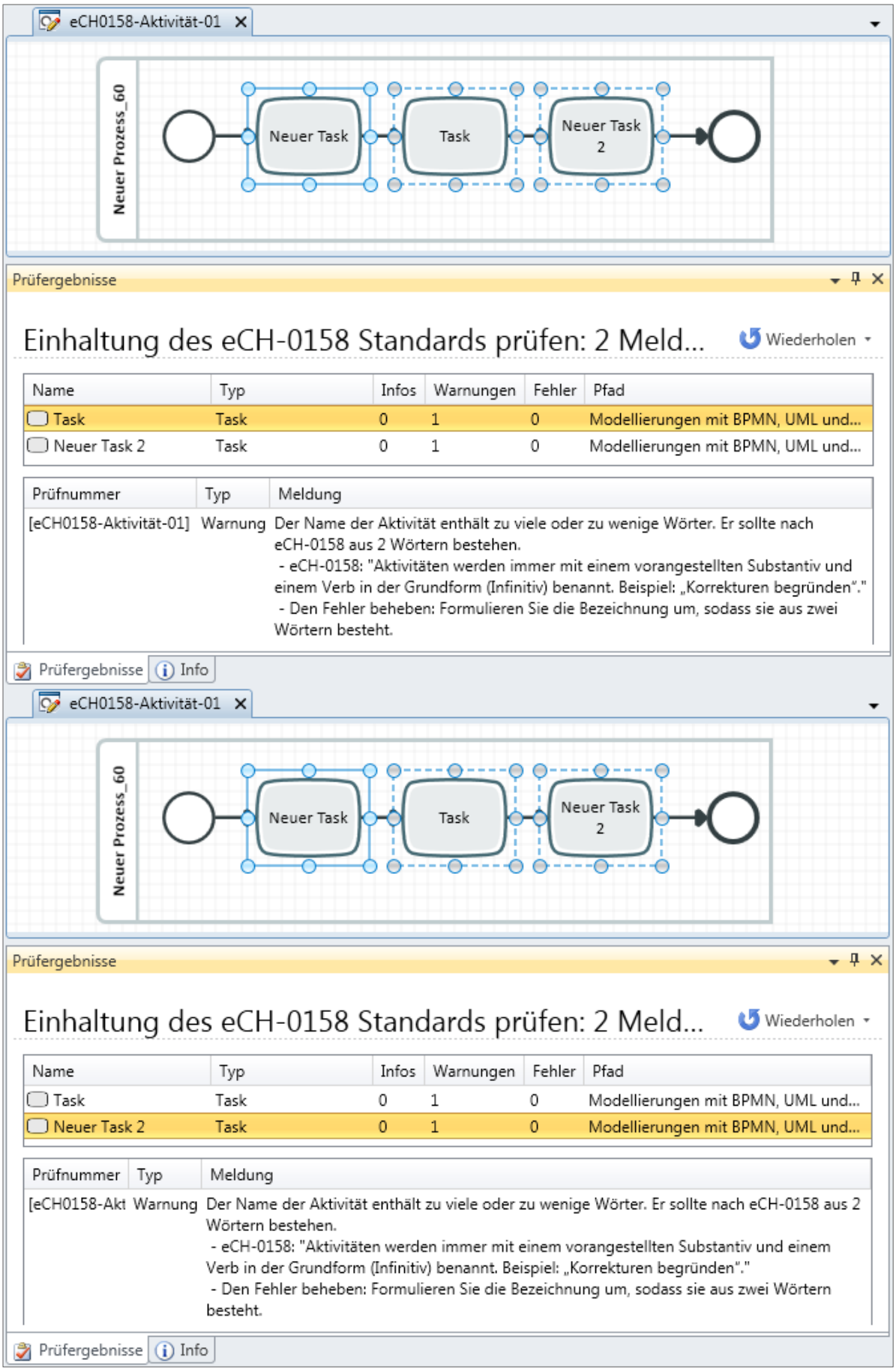


Abb. 8-34: Prüfung „eCH0158-Aktivität-01“.

8.5 TABELLEN

In den folgenden Tabellen sind für die eCH-0158-Konventionen zu BPMN-Pools (siehe Tab. 8-1) und zu BPMN-Startereignissen (siehe Tab. 8-2, S. 173) die von uns vorgeschlagene Umsetzungsart der jeweiligen Konvention sowie ob und wo diese Konvention im Qualitätssystem verordnet ist benannt.

Kürzel	Definition	Konfiguration	Automatische QS	Manuelle QS	Vorkommen im Qualitätssystem
4.3	***Pool***				
eCH0158-Pool-01	*„Pools werden in der Regel mit Organisationseinheiten oder dem Namen anderer Prozessbeteiligter bezeichnet.“*			X	---
eCH0158-Pool-02	*„Vorlagen mit den am häufigsten verwendeten Pools erleichtern die Arbeit und erhöhen die Lesbarkeit.“*				---
eCH0158-Pool-03	*„Reihenfolge oder Farbgebung kann der Identifizierung interner / externer Pools dienen.“*		(X)[74]	X	Qualität modellbasierter SRS • Qualität des SRS-Modellvisualisierungen • Hohe *Übersichtlichkeit der Visualisierung*
eCH0158-Pool-04	*„In der Regel wird nur der eigene Pool aufgeklappt dargestellt.“*		(X)[74]	X	Qualität modellbasierter SRS • Qualität des SRS-Modells • Zweckorientierte Modellqualität • *Angemessene Aussagekraft*
eCH0158-Pool-05	*„In jedem aufgeklappten Pool wird genau ein vollständiger Prozess modelliert.“*		X		Qualität modellbasierter SRS • Qualität des SRS-Modells • Zweckorientierte Modellqualität • *Angemessene Aussagekraft*
eCH0158-Pool-06	*„Pools werden übereinander über die gesamte Diagrammbreite dargestellt.“*	X			---
eCH0158-Pool-07	*„Die Höhe des aufgeklappten Pools richtet sich nach dessen Inhalt.“*		X		Qualität modellbasierter SRS • Qualität des SRS-Modellvisualisierungen • Hohe *Übersichtlichkeit der Visualisierung*
eCH0158-Pool-08	*„Zugeklappte Pools enthalten mindestens einen eingehenden oder ausgehenden Nachrichtenfluss.“*		X		Qualität modellbasierter SRS • Qualität des SRS-Modells • Zweckorientierte Modellqualität • *Angemessene Aussagekraft*

Tab. 8-1: Umsetzung der eCH-0158-Konventionen zu BPMN-Pool.

[74] Für die automatische Prüfung dieser Konvention sind vorab Anpassungen in der Konfiguration notwendig.

Kürzel	Definition	Konfiguration	Automatische QS	Manuelle QS	Vorkommen im Qualitätssystem
4.5.1	***Startereignis***				
eCH0158-Startereignis-01	Unbestimmtes Startereignis: *„Wird nicht beschrieben, der Unterprozess wird durch den Aufruf auf der übergeordneten Prozessebene gestartet.“*		X		Qualität modellbasierter SRS • Qualität des SRS-Modells • Zweckorientierte Modellqualität • *Angemessene Aussagekraft*
eCH0158-Startereignis-02	Nachrichten-Startereignis: *„Wird nicht beschrieben, wenn die eingehende Nachricht auf dem (obligatorischen) Nachrichtenfluss ersichtlich ist.“*		X		Qualität modellbasierter SRS • Qualität des SRS-Modells • Zweckorientierte Modellqualität • *Angemessene Aussagekraft*
eCH0158-Startereignis-03	Bedingungs-Startereignis: *„Bedingung ist in der Bezeichnung des Elementes festzuhalten. Sollte ein Bedingungs-Startereignis einem Ereignis eines anderen Prozesses entsprechen, ist es identisch zu bezeichnen.“*		X		Qualität modellbasierter SRS • Qualität des SRS-Modells • Zweckorientierte Modellqualität • *Angemessene Aussagekraft*
eCH0158-Startereignis-04	Zeitgeber-Ereignis: *„Bezeichnung enthält den Zeitpunkt für den Start. Beispiele: „am 1. je Monat“, „09:00 Uhr“.“*		X	X	Qualität modellbasierter SRS • Qualität des SRS-Modells • Zweckorientierte Modellqualität • *Angemessene Aussagekraft*
eCH0158-Startereignis-05	*„Unbestimmte Startereignisse werden nur verwendet, wenn der Prozess durch den Aufruf auf der übergeordneten Prozessebene gestartet wird.“*		X		Qualität modellbasierter SRS • Qualität des SRS-Modells • Zweckorientierte Modellqualität • *Angemessene Aussagekraft*
eCH0158-Startereignis-06	*„Ein Prozess hat mindestens ein Startereignis.“*		X		Qualität modellbasierter SRS • Qualität des SRS-Modells • Zweckorientierte Modellqualität • *Angemessene Aussagekraft*
eCH0158-Startereignis-07	*„Startereignisse werden innerhalb eines Pools dargestellt.“*	X			---

Tab. 8-2: Umsetzung der eCH-0158-Konventionen zu BPMN- Startereignis.