Institut für Softwaretechnologie
Abteilung Software Engineering
Universität Stuttgart
Universitätsstraße 38
70569 Stuttgart

Diplomarbeit Nr. 2447

Konzeption und Spezifikation einer Methodik-Unterstützung für JUST

Marcus Kaschowitz

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. rer. nat. Jochen Ludewig
Betreuer:	Dipl.-Ing. Rainer Schmidberger
begonnen am:	24. Januar 2006
beendet am:	24. Juli 2006
CR-Klassifikation	D.2, D.2.1, D.2.2, D.2.3, H.4, H.4.1, I.6, K.6, K.6.1, K.6.3

Kurzfassung

Zahlreiche Werkzeuge zur Verwaltung und Erhebung von Anforderungen stehen zur Zeit zur Verfügung. Unter ihnen befinden sich auch einige, die die Benutzer bei einer methodischen Vorgehensweise bei der Erfassung, Bearbeitung und Änderung von Anforderungen unterstützen. Jedoch gibt es nicht „die eine Methode“, sondern viele Herangehensweisen, die je nach Art des Projektes unterschiedlich ausfallen können. Im Rahmen dieser Diplomarbeit ist nun eine Spezifikation einer Erweiterung des Systems JUST entstanden, die genau dieses Problem angeht: „Wie unterstütze ich den Anwender bei seiner projektspezifischen Anforderungserhebung?“. Teil dieser Lösung ist das Prozessbeschreibungswerkzeug KoPra. Seine modellierten Prozesse dienen der Erweiterung von JUST als Grundlage für eine Wizard geführte Anforderungserhebung.
Die Forderungen an die Notationselemente zur Prozessmodellierung durch KoPra ergaben sich aus der Sichtung einschlägiger Literatur zum Thema Anforderungsanalyse und aus dem Kontakt zu einem Industriepartner.

Danksagung

Hiermit bedanke ich mich bei allen, die mir diese Diplomarbeit ermöglicht und mir bei dessen Umsetzung und Verwirklichung geholfen haben. Dies sind insbesondere

- mein Betreuer Rainer Schmidberger, der durch seine hilfreichen Ratschläge und einfallsreichen Ideen diese Diplomarbeit stark bereicherte und
- die Mitarbeiter der Abteilung Software Engineering des Instituts für Softwaretechnologie.

Inhaltsverzeichnis

Abkürzungsverzeichnis viii

Abbildungsverzeichnis ix

Tabellenverzeichnis xi

1 Einleitung 1
- 1.1 Motivation . . . 1
- 1.2 Hintergrund . . . 1
- 1.3 Aufgabenstellung . . . 2
- 1.4 Gliederung dieser Diplomarbeit . . . 3
- 1.5 Vorgehensweise . . . 3
  - 1.5.1 Sichtung der Literatur . . . 4
  - 1.5.2 Aufbereitung der Ergebnisse der Literaturrecherche . . . 4
  - 1.5.3 Erstellen des Metamodells der Analysemethoden . . . 5
  - 1.5.4 Sammeln von Ideen für die Erweiterung von JUST . . . 5
  - 1.5.5 Spezifikation der Erweiterung von JUST . . . 5
  - 1.5.6 Erweiterung von KoPra . . . 5

2 Grundlagen 7
- 2.1 Grundlegende Begriffe . . . 7
  - 2.1.1 Die Anforderung . . . 7
  - 2.1.2 Anforderungsanalyse . . . 10
  - 2.1.3 Methode versus Methodologie . . . 11
- 2.2 JUST . . . 12
  - 2.2.1 JUST im Überblick . . . 13
  - 2.2.2 Architektur von JUST . . . 15
- 2.3 KoPra . . . 16
  - 2.3.1 KoPra im Überblick . . . 17
  - 2.3.2 Architektur von KoPra . . . 20

3 Analysemethoden 22
- 3.1 Rückblick . . . 22
- 3.2 Die Analyse . . . 23
- 3.3 Übersicht über die objektorientierten Methodologien . . . 25
- 3.4 Structured Analysis . . . 25
  - 3.4.1 Methodisches Vorgehen . . . 26

3.4.2 Gewonnene Erkenntnisse . . . 33
3.5 Object Modeling Technique (OMT) . . . 34
3.5.1 Methodisches Vorgehen . . . 35
3.5.2 Gewonnene Erkenntnisse . . . 36
3.6 Rational Unified Process (RUP) . . . 37
3.6.1 Methodisches Vorgehen . . . 38
3.6.2 Gewonnene Erkenntnisse . . . 41

4 Metamodell der Analysemethoden 42
4.1 Vorgehensweise zur Erstellung . . . 42
4.2 Berücksichtigte Elemente . . . 42
4.3 Das Metamodell . . . 46
4.4 Erläuterungen zum Metamodell . . . 47

5 Methodik-Unterstützung für JUST 48
5.1 Vorüberlegungen . . . 48
5.2 Überblick über die Erweiterung . . . 50
5.2.1 Prozessimport . . . 50
5.2.2 Prozessvisualisierung . . . 50
5.2.3 Regulierung des Kontrollflusses . . . 52
5.2.4 Der Wizard . . . 57
5.2.5 Die TODO-Liste . . . 64

6 Die Erweiterung von KoPra 66
6.1 Verwendete Technologien . . . 66
6.1.1 JAVA . . . 66
6.1.2 JAVA - Swing . . . 67
6.1.3 JGraph . . . 67
6.1.4 JPOM . . . 68
6.1.5 Apache Tomcat . . . 68
6.1.6 Rational Rose . . . 68
6.2 Anforderungserhebung und Spezifikation . . . 69
6.3 Entwurf . . . 76
6.4 Implementierung . . . 80
6.5 Test . . . 81
6.5.1 Durchführung . . . 81
6.5.2 Ergebnisse . . . 81

7 Fazit und Ausblick 82
7.1 Rückblick . . . 82
7.2 Fazit . . . 84
7.3 Ausblick . . . 84
7.4 Schlussbemerkung . . . 85

A Structured Analysis **86**
A.1 Datenflussdiagramm (DFD) 86
A.2 Kontextdiagramm 88
A.3 Datenlexikon 88

B Die Prozesselemente und ihre Bedeutung **90**

C Begriffslexikon **93**

D Systemtest **97**
D.1 Zweck des Tests 97
D.2 Testumfang 97
D.3 Testumgebung 97
D.4 Testplan 98

Literaturverzeichnis **104**

Abkürzungsverzeichnis

API	Application Programming Interface
CVS	Concurrent Versioning System
FURPS	Functionality Usability Reliability Performance Supportability
HTML	Hypertext Markup Language
HTTP	Hypertext Transport Protocol
JDBC	Java Database Connectivity
JPOM	Java Persistence Object Manager
JUST MoRe	JUST Modification Request
KoPra	Konfigurierbarer Prozess- und Projektassistent
OMG	Object Management Group
OMT	Object Modeling Technique
RUP	Rational Unified Process
SPEM	Software Process Engineering Metamodel
SSADM	Structured Systems Analysis and Design Methodology
UML	Unified Modeling Language
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language - Transformation

Abbildungsverzeichnis

2.1 Taxonomie der Anforderung nach dem IEEE Std. 610 8
2.2 Taxonomie der Anforderung nach „FURPS+“ 9
2.3 Schematischer Aufbau von JUST [JUST-Team (2004a)] 13
2.4 Architektur von JUST [JUST-Team (2004b)] 15
2.5 Überblick über das KoPra-System [KoPra-Team (2005b)] 17
2.6 Schematischer Aufbau von KoPra [KoPra-Team (2005b)] 18
2.7 Architektur von KoPra 21

3.1 Historische Entwicklung der objektorientierten Methodologien [Oestereich (2004)] 25
3.2 Übersicht über die „Structured Analysis“-Phasen 26
3.3 Teilprozess: Problemevaluation 27
3.4 Teilprozess: Ist-Analyse durchführen 28
3.5 Teilprozess: Lösungsvorschläge konzipieren 29
3.6 Teilprozess: Lösungsvorschlag konkretisieren 30
3.7 Teilprozess: System spezifizieren 31
3.8 Teilprozess: Entwurf erstellen 32
3.9 Übersicht über die „Object Modeling Technique“-Aktivitäten 35
3.10 Geschäftsprozessmodellierung nach dem RUP [Cooperation (2003)] 38
3.11 Anforderungsanalyse nach dem RUP [Cooperation (2003)] 40

4.1 Metamodell der Analysemethoden 46

5.1 Prozessvisualisierung in JUST 51
5.2 Spaltung mit zwei ausgehenden Pfaden 53
5.3 Abfrage bei einer Spaltung mit zwei Ausgängen 53
5.4 Spaltung mit einem ausgehenden Pfad 54
5.5 Abfrage bei einer Spaltung mit einem ausgehenden Pfad 54
5.6 Abfrage bei einer Spaltung mit mehreren ausgehenden Pfaden 55
5.7 Der Sprung zu einer bereits abgeschlossenen Aktivität 56
5.8 Synchronisation durch den Synchronisationsbalken 56
5.9 Prozessbeispiel für die Wizardführung 59
5.10 Aktivierung der Aktivitäten durch den Anwender 60
5.11 Aktivierung der Aktivitäten anhand der Vorbedingungen 61
5.12 Wann sind die Aktivitäten „offen“? 61
5.13 Der Wizard 64
5.14 Die TODO-Liste 65

6.1 Beispielmethode 69
6.2 Domänenmodell der KoPra-Erweiterung 76
6.3 KoPra-Pakete 77

A.1 Ein typisches DFD 87
A.2 Verknüpfung von Datenflüssen 88

Tabellenverzeichnis

1.1 Literaturliste der untersuchten Analysemethoden 4

2.1 Prozesselemente eines KoPra-Prozesses 20

3.1 Beispiel für eine Aktivität 33

4.1 Allgemeingültige Eigenschaften aller Elemente 43
4.2 Elemente des Metamodells, die ein Analyseverfahren beschreiben 43
4.3 Eigenschaften der „direkten“ Elemente 44
4.4 Elemente, die den Kontrollfluss beeinflussen 45

5.1 Erläuterung der Tabelle aus Abbildung 5.1 52
5.2 Erläuterung der Tabelle aus Abbildung 5.14 65

6.1 Ergebnis des Vergleichs zwischen KoPra und der Beispielmethode 70
6.2 Mögliche Relationen zwischen den Prozesselementen 74
6.3 Export von Prozesselementen 75
6.4 Entwicklungsumgebung 80

7.1 Aufgaben und Ergebnisse dieser Diplomarbeit 82
7.2 Wichtige Meilensteine 83

A.1 Erläuterung der Backus-Naur-Form 89

1 Einleitung

1.1 Motivation

Es ist allgemein bekannt, dass der Einsatz von Werkzeugen zur Anforderungsverwaltung eine Verbesserung der Softwarequalität und eine Verringerung der Softwareentwicklungskosten bewirken kann.
JUST ist ein Werkzeug zur Anforderungsverwaltung, was sich von anderen Werkzeugen dieser Art durch eine Mehrbenutzerfähigkeit, einer Versionierung von einzelnen Projektinformationen, der Möglichkeit, einzelne Projektinformationen miteinander in Beziehung zu setzen und der Besonderheit, verschiedene Projektinformationen verwalten zu können, abhebt. Jedoch unterstützt JUST den Anwender nicht bei der Art und Weise, wie er diese Projektinformationen erheben soll. Ein Anwendungsspezialist oder aber auch ein Systemanalytiker, der auf Grundlage eines projektspezifischen Prozesses eine Anforderungsanalyse betreibt, sammelt Anforderungen nach einer gewissen Methodik. Diese Methodik-Unterstützung soll nun in JUST durch einen Wizard umgesetzt werden. Dadurch soll der Anwender in seiner Arbeit unterstützt werden. So können Checklisten, Leitlinien oder gar Tipps für ein mögliches Vorgehen vom Wizard für den Anwender bereitgestellt werden, sodass dieser bei Unerfahrenheit oder Unwissenheit auf diese Hilfe zurückgreifen kann. Darüber hinaus soll die Wizardführung z.B. den Anwendungsspezialist möglichst „an der Hand führend“ Schritt für Schritt durch die Anforderungsanalyse begleiten. Dadurch wird ihm ein methodisches Vorgehen „aufgezwungen“, was aber zu einer klareren Arbeitsweise führt.
Da viele Unternehmen und Organisationen keinen einzelnen, allgemein anwendbaren Prozess für alle ihre Projekte haben, sondern typischerweise viele Prozesse, die auf die Eigenheiten der Projekte zugeschnitten sind, besitzen, soll die zukünftige Erweiterung von JUST eine möglichst generische Methodik-Unterstützung anbieten. So muss das neue JUST flexibel genug sein, um unterschiedliche Prozesse (die die Methodiken repräsentieren) einlesen zu können und anhand dieser Prozesse die Wizard-gestützte Anforderungserhebung durchzuführen.

1.2 Hintergrund

JUST ist ein System zur Erfassung und Verwaltung von Anforderungen und unterstützt speziell die Projektphasen der Analyse und Spezifikation. Die ursprüngliche Entwicklung von JUST fand in der Abteilung Software Engineering im Rahmen eines Studienprojekts an der Universität Stuttgart statt. Es gab auch Weiterentwicklungen, so u.a. JUST MoRe.

Allerdings fehlt JUST ebenso wie den konventionellen Spezifikationsdokumenten eine Methodik, die den Benutzer zu systematischer Anforderungserhebung anleitet. Es wäre aber nicht sinnvoll „die“ Methode zur Anforderungserhebung festzulegen, denn je nach Art eines Projekts können völlig verschiedene Methoden zur Anforderungserhebung erforderlich oder sinnvoll sein. So findet man in der Praxis und in der Literatur viele Methoden zur Anforderungserhebung, die sich z.B. bei den beteiligten Rollen, den Aktivitäten, deren Abfolge oder der Qualitätssicherung teilweise deutlich unterscheiden.

Eine methodische Unterstützung in JUST sollte somit konfigurierbar sein. Die Konfiguration und Erstellung sollte graphisch als Prozessmodell ermöglicht werden.

Ein solches Werkzeug zur graphischen Prozessmodellierung ist KoPra, das, ebenso wie JUST, in der Abteilung Software Engineering im Rahmen eines Studienprojekts entwickelt wurde.

1.3 Aufgabenstellung

In dieser Diplomarbeit sollen aus der Literatur Methoden zur Analyse und Spezifikation entnommen und zusammengestellt werden. Hierzu ist eine ausführliche Literaturrecherche erforderlich. Die gefundenen Methoden sollen zu einem gemeinsamen Methoden-Metamodell abstrahiert werden. Ausgangspunkt dieses Metamodells sollte das Domänenmodell der Anwendung KoPra sowie das Software Process Engineering Metamodel (SPEM) der OMG [spe (2006)] sein.

Für JUST soll nun die Methodik-Unterstützung spezifiziert werden. Da hierzu das Methoden-Metamodell die Grundlage bildet, wird für die jeweiligen Modellelemente die Funktion von JUST spezifiziert. Es soll aber auch an einer beispielhaften Analysemethode ein durchgängiger Ablauf in JUST beschrieben werden. Besonderer Wert wird auf eine einfache und schlüssige Bedienung von JUST gelegt.

JUST soll das Methoden-Modell als Konfigurationsdatei einlesen. Das Dateiformat hierzu soll ebenfalls spezifiziert werden.

In einem nächsten Schritt soll das Prozessmodellierungswerkzeug KoPra mit der Darstellung der Analysemethoden erweitert werden. Insbesondere soll der Workflow-Aspekt, also die bedingte Abfolge von Aktivitäten, sowie eventuell noch fehlende Modellelemente des Methoden-Metamodells in KoPra integriert werden. Analyse-Modelle, die in KoPra erstellt werden, sollen in Form einer Konfigurationsdatei für JUST abgespeichert werden können.

Die Implementierung der spezifizierten Erweiterung für JUST ist nicht Teil dieser Arbeit.

Folgende Teilaufgaben sind während der Diplomarbeit durchzuführen:

- Erstellen eines Projektplans mit expliziter Angabe von Meilensteinen
- Literaturrecherche zum genannten Themengebiet
- Erstellen eines Metamodells der Analysemethoden
- Spezifikation der JUST-Erweiterung

- Spezifikation der KoPra-Erweiterung
- Spezifikation des Formats der JUST-Konfigurationsdatei
- Implementierung der Erweiterungen von KoPra
- Präsentation der Ergebnisse der Diplomarbeit in einem Zwischen- und einem Endvortrag
- Berichterstellung

1.4 Gliederung dieser Diplomarbeit

Die vorliegende Diplomarbeit gliedert sich in sieben Kapitel.

Kapitel 1 gibt einen Einstieg in die Thematik der vorliegenden Diplomarbeit. Hier wird zunächst die Aufgabe, die dieser Diplomarbeit zugrunde liegt, vorgestellt und dann die einzelnen Arbeitsschritte, die für die Bearbeitung des Themas nötig waren, präsentiert.

Kapitel 2 beschreibt die Grundlagen, die für das Verständnis dieser Diplomarbeit erforderlich sind. Insbesondere werden dabei die Definitionen von verwendeten Begriffen, das JUST- und das KoPra-System vorgestellt.

Kapitel 3 beschreibt die aus der Literaturrecherche gewonnenen Analysemethoden.

Kapitel 4 stellt auf Basis der gewonnenen Erkenntnisse aus der Untersuchung der Analysemethoden ein Metamodell der Analysemethoden vor.

Kapitel 5 stellt die Konzeption der Erweiterung von JUST vor. Wichtige Funktionen wie der Wizard und die Visualisierung von KoPra-Prozessen sowie die Beziehung zwischen Wizard und dem zugrunde liegenden Prozess werden näher beschrieben.

Kapitel 6 beschreibt die Realisierung der KoPra-Erweiterung. Es werden die Spezifikation, der Entwurf, die Implementierung und der Test vorgestellt.

Kapitel 7 fasst die Ergebnisse dieser Diplomarbeit zusammen und gibt einen Ausblick auf mögliche Erweiterungen und Ergänzungen.

Im Anhang befinden sich eine Beschreibung der Symbole zur Prozess-Modellierung, Erläuterungen zu der untersuchten Analysemethode Structured Analysis, ein Testplan für die KoPra-Erweiterung und ein Begriffslexikon.

1.5 Vorgehensweise

Dieses Kapitel beschreibt die einzelnen Tätigkeiten meiner Diplomarbeit. Die Arbeitsschritte waren stets auf die im Projektplan [Kaschowitz (2006b)] festgesetzten Termine und Tätigkeiten abgestimmt und wurden strikt befolgt.

1.5.1 Sichtung der Literatur

Zu Beginn meiner Diplomarbeit habe ich viele Standardwerke rund um das Themengebiet der Anforderungsanalyse gesichtet. Mein Anliegen war es, mir einen Überblick über die gängigsten Methoden und Methodologien der Anforderungsanalyse zu verschaffen, deren Unterschiede und Gemeinsamkeiten herauszufinden und ihre Bedeutung für die spätere Einbindung in JUST zu ermitteln. Zu einem sehr frühen Zeitpunkt meiner Diplomarbeit konzentrierte ich mich dabei auf:

- eine gezielte Literaturrecherche,
- eine gezielte Internetrecherche und
- das Sichten von Standards und Normen.

Die Bücher und Dokumente, die mir von Bedeutung erschienen, wurden in einem eigenen Dokument verwaltet, um sie zu einem späteren Zeitpunkt schnell wieder finden zu können.

1.5.2 Aufbereitung der Ergebnisse der Literaturrecherche

Im nächsten Schritt habe ich die gewonnenen Erkenntnisse dokumentiert. Zunächst sammelte ich alle recherchierten Methodologien zur Anforderungsanalyse und begann sie zu kategorisieren. Ich stellte schnell fest, dass es sehr viele von ihnen gibt, gerade zur Analyse und zum Entwurf von Softwaresystemen. Daher musste ich mich bei der Wahl der zu untersuchenden Methodologien einschränken. Ich wählte einige Vertreter aus den Themengebieten der objektorientierten- und strukturierten Analyse aus, die ich später eingehender untersuchen wollte. Die Tabelle 1.1 zeigt welche Methoden und Methodologien ich untersucht habe.

Methodologie	Literaturverweis
Booch-Methode	[Booch (1994, 1987)]
Extreme Programming	[Beck (1999, 2003)]
Jackson Structured Programming	[Kilberth (1989)]
Object-modeling technique (OMT)	[Rumbaugh (1993)]
Object-oriented Analysis (OOA)	[Booch (1994); Larman (1998)]
Rational Unified Process (RUP)	[Jacobson Ivar (2001); Cooperation (2003)]
Structured Analysis	[Edward (1992); Weinberg (1980); DeMarco (1979)]

Tabelle 1.1: Literaturliste der untersuchten Analysemethoden

Von ihnen werden in dieser Diplomarbeit allerdings nur der RUP, die Structured Analysis und die OMT vorgestellt.
Abschließend erstellte ich anhand der untersuchten Methodologien ein Dokument, in dem ich zwei konkrete Vorgehensweisen der Anforderungsanalyse genau beschrieb [Kaschowitz (2006c)].

1.5.3 Erstellen des Metamodells der Analysemethoden

Im weiteren Verlauf meiner Diplomarbeit erstellte ich ein Metamodell der Analysemethoden. Die vorhergehende Analyse der Methodologien und Methoden der Anforderungsanalyse und das Software Engineering Process Metamodel (SPEM) [spe (2006)] dienten als Grundlage für dieses Modell (siehe Kapitel 4).

1.5.4 Sammeln von Ideen für die Erweiterung von JUST

Vor der eigentlichen Spezifikation des Systems überlegte ich mir, wie man mit Hilfe von JUST den Anwender bei seiner Anforderungserhebung unterstützen könnte. Sehr schnell bekam ich die Idee mit Hilfe eines Wizards das Problem zu lösen. Dieser sollte dem Anwender bei der Erfassung seiner Projektinformationen helfen und ihn durch die Anforderungsanalyse Schritt für Schritt führen. Dadurch sollte der Anwender zu einem methodischen Vorgehen angehalten werden. Ich zeichnete ein paar Skizzen eines möglichen Wizards, überlegte mir dann, wie ich die wesentlichen Elemente des Metamodells der Analysemethoden in diesen Wizard einbinden und die Prozesse von KoPra elegant in JUST integrieren könnte. Es entstand eine Sammlung von Skizzen und erläuternden Texten, die mir später als Grundlage für Gespräche mit meinem Betreuer und für die Ausarbeitung der eigentlichen Spezifikation hilfreich waren (Genaueres ist im Kapitel 5 zu finden).

1.5.5 Spezifikation der Erweiterung von JUST

Der nächste Schritt stand an - die Spezifikation schreiben. Unter Verwendung meiner Skizzen, Ideen und der Informationen aus den Gesprächen mit meinem Betreuer fing ich an, die Spezifikation für die Erweiterung von JUST zu schreiben. Zunächst beschrieb ich die nicht-funktionalen Anforderungen, dann in Form von Use-Case-Diagrammen und Use-Case-Beschreibungen die funktionalen Anforderungen und schließlich das Verhalten von JUST im Hinblick auf die einzelnen Prozesselemente eines KoPra-Prozesses.

1.5.6 Erweiterung von KoPra

Prozesse, die mit KoPra modelliert werden, sollen später in die Erweiterung von JUST importiert werden können und dann die eigentliche Wizard geführte Anforderungserhebung ermöglichen. Der Aufbau eines solchen Prozesses, die beteiligten Prozesselemente und deren Eigenschaften legen letztendlich die Benutzerführung durch den Wizard fest. Daher ist es wichtig, dass ein solcher Prozess über die nötige Ausdruckskraft verfügt, um Analysemethoden modellieren zu können. Anhand des Metamodells der Analysemethoden und der Spezifikation der JUST-Erweiterung haben sich über die bereits bestehenden Funktionalitäten von KoPra hinaus neue Anforderungen an das System ergeben. Resultat der letzten Aufgabe meiner Diplomarbeit war die Erweiterung des Systems KoPra um folgende Funktionen:

- Ein XML-Export, der den Export von modellierten Prozessen gewährleisten sollte.
- Auf Basis des Metamodells der Analysemethoden mussten
  - fehlende Prozesselemente sowie
  - fehlende Kontrollflusselemente ergänzt und
  - spezielle Eigenschaften den einzelnen Prozesselementen noch hinterlegt werden.

Die Realisierung dieser Erweiterung umfasste die Spezifikation, den Entwurf, die Implementierung und den abschließenden Test. Sowohl die Spezifikation als auch der Entwurf dieser Erweiterungen wurden durch vorhandene Dokumente (KoPra-Spezifikation [KoPra-Team (2005b)] und Domänenmodell [KoPra-Team (2005a)]) erleichtert. Die Implementierung teilte sich in kleinere Arbeitspakete auf. Der erfolgreiche Abschluss eines Arbeitspaketes war mit einer Präsentation beim Betreuer verbunden.

2 Grundlagen

2.1 Grundlegende Begriffe

Sucht man in der Literatur nach den Begriffen Anforderungsanalyse oder Methode fällt es auf, dass es viele Auslegungen und Definitionen allein für die zwei Begriffe gibt. Daher ist es für das Verständnis dieser Diplomarbeit notwendig, die zentralen Begriffe eindeutig zu definieren und voneinander abzugrenzen. Da ich mich im Rahmen dieser Diplomarbeit mit der Untersuchung von Methoden zur Anforderungsanalyse beschäftigen musste, die die Grundlage für das spätere Metamodell der Analysemethoden bilden, ist es unumgänglich, die Begriffe Anforderung, Anforderungsanalyse, Methode und Methodologie einzuführen und zu erläutern.

2.1.1 Die Anforderung

Nach dem IEEE-Standard 610 [IEEE Std. 610.12 (1990)] ist eine Anforderung

„(1)...ein Zustand oder eine Fähigkeit, die von einem Benutzer gefordert wird, um ein Problem zu lösen oder ein Ziel zu erreichen.
(2)...ein Zustand oder eine Fähigkeit, welche durch ein System oder eine Systemkomponente erzielt werden muss, um einen Vertrag, einen Standard oder eine Spezifikation einzuhalten.
(3)...eine dokumentierte Darstellung eines Zustandes oder einer Fähigkeit, wie sie in (1) oder (2) beschrieben ist.“

Kategorisierung der Anforderungen nach dem IEEE

Es gibt unterschiedliche Ausprägungen von Anforderungen. Je nach Auswirkung und Bedeutung einer Anforderung wird sie anders kategorisiert.

Die IEEE unterscheidet sechs Typen von Anforderungen: die *Implementierungsanforderungen*, *Entwurfsanforderungen*, *Schnittstellenanforderungen*, *Performanzanforderungen*, *physikalische Anforderungen* und *funktionale Anforderungen*. Diese sind in dem IEEE Standard 610 [IEEE Std. 610.12 (1990)] wie folgt definiert:

Implementierungsanforderung:

„Eine Anforderung, die die Implementierung eines Systems oder einer Systemkomponente einschränkt oder spezifiziert, bezeichnet man als Implementierungsanforderung.“

Entwurfsanforderung:

„Eine Anforderung, die den Entwurf eines Systems oder einer Systemkomponente einschränkt oder spezifiziert, bezeichnet man als Entwurfsanforderung."

Schnittstellenanforderung:

„Anforderungen, die eine externe Quelle spezifizieren, mit der das System interagieren muss oder jegliche Restriktionen, wie Formate oder Forderungen an die Ausführungszeit, die die Kommunikation zwischen dem System und der externen Quelle beeinflussen, bezeichnet man als Schnittstellenanforderungen."

Performanzanforderung:

„Eine Anforderung, die Bedingungen einer funktionalen Anforderung auferlegt, bezeichnet man als Performanzanforderung. Dies kann z.B. eine Anforderung sein, die die Ausführungsgeschwindigkeit, die Genauigkeit oder den Speicherverbrauch einer Funktion festlegt."

Physikalische Anforderung:

„Eine Anforderung, die physikalische Eigenschaften des Systems spezifiziert, so z.B. das Material, die Größe oder die Form oder das Gewicht eines Systems, bezeichnet man als physikalische Anforderung."

Funktionale Anforderung:

„Eine Anforderung, die eine Funktion beschreibt, die ein System oder eine Systemkomponente ausführen soll, bezeichnet man als funktionale Anforderung."

Die Abbildung 2.1 zeigt die Taxonomie der in dem IEEE-Standard eingeführten Typen von Anforderungen.

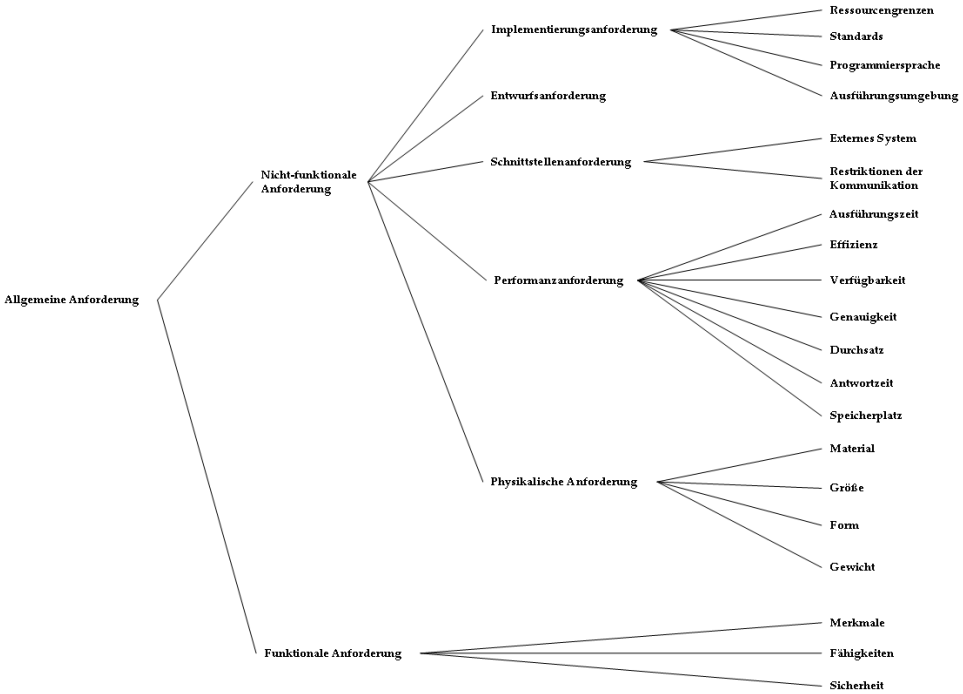


Abbildung 2.1: Taxonomie der Anforderung nach dem IEEE Std. 610

Dabei versteht man unter nicht-funktionalen Anforderungen solche Anforderungen, die die Funktionen eines Systems einschätzen und beurteilen. Typische nicht-funktionale Anforderungen sind z.B. Zuverlässigkeit, Skalierbarkeit oder die Kosten der Umsetzung einer Funktionalität. Nicht-funktionale Anforderungen werden mitunter auch als Qualitätseigenschaft einer Anforderung angesehen. Im Gegensatz dazu spezifizieren funktionale Anforderungen konkrete Verhaltensweisen von Funktionen. Sie definieren den internen Ablauf einer Software in Form von notwendigen Berechnungen, technischen Details oder Datenmanipulationen.

Auf der von dem IEEE eingeführten Kategorisierung von Anforderungen setzt das „FURPS+“-Modell auf [Grady (1992)]. Dieses beschreibt eine weiterführende Kategorisierung der Anforderungen. Das Akronym FURPS steht dabei für die fünf Kategorien „Functionality“, „Usability“, „Reliability“, „Performance“ und „Supportability“. Das „+“ soll signalisieren, dass die Kategorien aus dem Standard der IEEE ebenfalls dazugehören. Die Abbildung 2.2 veranschaulicht die Typen von Anforderungen nach dem „FURPS+“-Modell.

Abbildung 2.2: Taxonomie der Anforderung nach „FURPS+“

2.1.2 Anforderungsanalyse

Die Anforderungsanalyse wird mitunter auch als

- Requirement Engineering oder
- Requirement Analysis oder
- Systemanalyse

bezeichnet.

Das Hauptkriterium für den Erfolg einer Software ist, ob sie das erfüllt, wofür sie eigentlich angedacht war. Deutlicher gesagt, eine Software ist nur dann gut und erfolgreich, wenn sie den Vorstellungen und Wünschen des Kunden und der Benutzer und den Forderungen und Ansprüchen aus Vereinbarungen entspricht. Die Anforderungsanalyse ist ein Prozess, der genau diesen Vorsatz verfolgt. Dadurch, dass der Systemanalyst die Interessengruppen identifiziert, ihre Wünsche und Vorstellungen an das neue System herausfindet, kann der Anspruch, eine erfolgreiche Software zu entwickeln, erst verwirklicht werden.
Doch die Anforderungsanalyse ist weit mehr als nur die Erhebung von Wünschen und Ideen und selbst diese einfachen Ziele bergen schon einige Probleme. So können die Ziele zwischen den Interessengruppen unterschiedlich sein, ja sogar im Konflikt zueinander stehen, da Personen unterschiedliche Interessen verfolgen können. Aber auch unklare, schwammige Aussagen, unpräzise und missverstandene Anforderungen können zu Problemen führen.

Nachdem nun deutlich gemacht wurde, welche Rolle die Anforderungsanalyse in der Softwareentwicklung einnimmt, folgen nun ein paar Definitionen.

Für Barry Boehm [Boehm (1976)] ist die Anforderungsanalyse

„...a discipline for developing a complete, consistent, unambiguous specification, which can serve as a basis for common agreement among all parties concerned and describing what the software product will do (but not how it will do it).“

Barry Boehm sieht die Anforderungsanalyse lediglich als eine Tätigkeit zur Erstellung einer Spezifikation an. Diese soll vollständig, konsistent und eindeutig sein, um dadurch mögliche Probleme zu vermeiden. Doch diese Definition ist mir etwas zu einseitig, da hier nur auf das Resultat wertgelegt wird. Der eigentliche Zweck und die Absicht der Anforderungsanalyse bleiben meiner Meinung nach völlig unberücksichtigt - die Verwirklichung von Kundenwünschen und -vorstellungen.

In dem IEEE-Standard [IEEE Std. 610.12 (1990)] wird die Anforderungsanalyse folgendermaßen definiert:

*„(1) Die Anforderungsanalyse ist ein Prozess, der die Bedürfnisse und Wünsche der Benutzer untersucht und in einer Definition einer Software-, Hardware- oder Systemanforderung mündet.
(2) Die Anforderungsanalyse ist ein Prozess, der sich mit der Untersuchung und Verfeinerung von Software-, Hardware- oder Systemanforderungen beschäftigt. “*

Diese Definition ist meiner Meinung nach besser als die vorherige. Sie beschreibt eine der wichtigsten Aufgaben der Anforderungsanalyse: das Bestimmen und Beschreiben von Kundenwünschen in Form von Anforderungen. Auch die explizite Erwähnung der ständigen

Verbesserung von Anforderungen erscheint mir angebracht, da sie wesentlich für die Anforderungsanalyse ist.

Aber ich bevorzuge die Definition von Zave [Zave (1997)], die in dieser Diplomarbeit angewandt wird. Die Anforderungsanalyse definiert er als

„...the branch of software engineering concerned with the real-world goals for, functions of, and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behaviour, and to their evolution over time and across software families."

Ich ziehe diese Definition aus verschiedenen Gründen den anderen vor. Zum einen wird in der Definition von Zave die Bedeutung von „wirklichkeitsnahen Zielen" (im engl. real-world goals) hervorgehoben. Diese begründen ja erst eine Entwicklung von Software. Zum anderen betont er, dass Anforderungen präzise und genau dokumentiert werden müssen („precise specifications"). Dies sind meiner Meinung nach die zentralen Aufgaben der Anforderungsanalyse. Darüber hinaus deutet er darauf hin, dass Anforderungen über längere Zeit hinweg entwickelt und verfeinert werden müssen („evolution over time") und dass die Wiederverwendung von bestehenden Spezifikationen und Anforderungen nützlich sein kann („across software families").

2.1.3 Methode versus Methodologie

Die beiden Begriffe Methodologie und Methode werden oft als Synonyme verwendet, was häufig zu Missverständnissen führt. Die Wissenschaft ist daher bemüht, die zwei Begriffe klar voneinander abzugrenzen. Insbesondere die Softwaretechnik hat sich dieser Thematik gestellt. Durch Diskussionen und verschiedene Beiträge rund um dieses Themengebiet hat sie viel zur Klärung und Definition der beiden Begriffe beigetragen.

Man könnte argumentieren, dass eine Methode eine Folge von Schritten ist, um eine Software zu entwickeln, wohingegen eine Methodologie ein ganzes Bündel an empfohlenen Praktiken und Disziplinen darstellt, aus denen eine für ein Projekt geeignete gewählt werden kann. Wenn man es so betrachtet, ist eine Methode ein Teil von einer Methodologie, was auch der Definition der IEEE entspricht.

Der ANSI/IEEE Standard 1002 [ANSI/IEEE Std. 1002 (1987)] definiert eine Methode als

„...einen Prozess (oder ein Verfahren), der in der Entwicklung von Produkten oder in der Durchführung von Diensten angewendet wird."

Eine Methodologie wird definiert als [IEEE Std. 1074 (1991)]

„eine Vereinigung von Methoden, Regeln und Vorraussetzungen die für eine Aufgabe gebraucht werden."

Wendet man nun die beiden Definitionen an, stellt man fest, dass die Softwaretechnik reich an Methoden und arm an Methodologien ist.

So gibt es nur wenig wirkliche Prozessmodelle die als Methodologien in betracht kommen. Beispiele für solche sind

- strukturiertes Programmieren,

- strukturierte Systemanalyse und Entwurfsmethodologien (SSADM),
- top-down Programmieren,
- objektorientiertes Programmieren,
- Rational Unified Process,
- Agile Unified Process und
- Extreme Programming.

Der Einsatz von geeigneten Methoden ist in der Anforderungsanalyse ein großer Vorteil. Der Anwendungsspezialist und der Systemanalytiker werden mit der Durchführung einer bereits erprobten und bewährten Vorgehensweise von einer Menge unnötiger Entscheidungen befreit. Diese werden nämlich von der Methode vorweggenommen. Dadurch spart er kostbare Zeit ein, vermeidet unnötige Fehler und verringert seine Anstrengungen bei der Erfüllung seiner Aufgabe.

2.2 JUST

Im Verlauf dieser Diplomarbeit entstand eine Spezifikation [Kaschowitz (2006a)] einer Erweiterung von JUST. In ihr wird die Realisierung eines Wizards beschrieben, der den Anwender durch die Anforderungserhebung führen soll. Das System JUST, das als Grundlage für diese Erweiterung dient, wurde im Rahmen eines einjährigen Studienprojektes an der Universität Stuttgart am Institut für Softwaretechnologie entwickelt. An ihrer Entwicklung waren zehn Studenten beteiligt. Es ist ein webbasiertes System zur Erfassung, Versionierung und Verwaltung von Projektinformationen. Dadurch, dass JUST auf einer Plugin-Architektur konzipiert wurde, ist es möglich, zusätzliche Typen von Projektinformationen zu integrieren, so unter anderem Anforderungen, Use-Cases, Testfälle, Problemberichte oder Gesprächsnotizen.

Es wurden bereits zwei Plugins realisiert.

Plugin zur Anforderungsverwaltung: Das Basissystem JUST liefert standardmäßig ein Plugin zur Erfassung und Verwaltung von Anforderungen mit.

Plugin zur Behandlung von Software Problem Reports (SPR) und Maintenance Reports (MR): Als Ergebnis einer früheren Diplomarbeit [Schönleber (2005)] ist eine Erweiterung mit dem Namen „JUST MoRe“ entstanden. Mit ihr ist es möglich, Probleme vom Auftreten bis zu deren Abschluss zu behandeln. Es unterstützt die beiden Bereiche Support (mit Hilfe von SPRs) und Software-Wartung (mit Hilfe von MRs).

2.2.1 JUST im Überblick

Generelles

JUST-Objekte sind Instanzen einer Projektinformation.

Jedes *JUST-Objekt* hat einen Typ, der abhängig vom jeweiligen Plugin ist. So kann ein *JUST-Objekt* z.B. eine Anforderung, ein Use-Case oder eine Gesprächsnotiz repräsentieren.

JUST-Objekte werden in Ordnerstrukturen hierarchisiert und geordnet.

Jedes *JUST-Objekt* und jeder Ordner gehören zu einem Projekt.

JUST-Objekte, Ordner und Projekte werden mit dem Begriff *Baumelement* zusammengefasst.

Jedes *Baumelement* kann mit Zugriffsrechten versehen werden.

Die Oberfläche von JUST

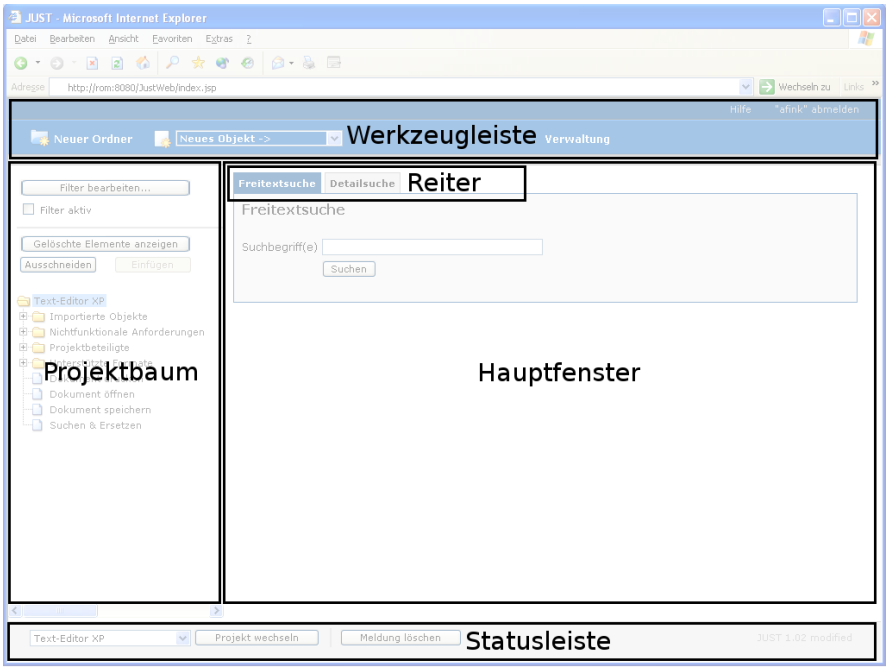


Abbildung 2.3: Schematischer Aufbau von JUST [JUST-Team (2004a)]

Dieser Abschnitt beschreibt die Oberfläche von JUST. Die Abbildung 2.3 stellt den schematischen Aufbau von JUST dar.

Projektbaum: Über dem Projektbaum kann man durch die Baumelemente navigieren. Wird ein Baumelement aktiviert, so wird dessen Detail-Reiter angezeigt. Außerdem können mit Hilfe des Filters benutzerdefinierte Baumelemente im Projektbaum angezeigt werden.

Hauptfenster: Hier werden die Detailinformationen zu dem im Projektbaum markierten Baumelement angezeigt. Über das Hauptfenster wird ebenfalls die Erstellung von Reports, die Suche und die Administration ausgeführt.

Reiter: Über dem Reiter lassen sich bestimmte Ansichten eines JUST-Objekts auswählen. Mit Hilfe der Reiter ist es möglich, bestimmte Eigenschaften des JUST-Objekts festzulegen.

Werkzeugleiste: Mit Hilfe der Werkzeugleiste können neue Baumelemente einem Projekt hinzugefügt, das komplette System administriert, Reports erstellt und die Suche aufgerufen werden.

Statusleiste: In der Statusleiste werden vom System generierte Erfolgs- oder Fehlermeldungen angezeigt. Außerdem ist es möglich, das aktuelle Projekt zu wechseln.

Unterstützte Funktionen

JUST unterstützt den Anwender bei der Verwaltung und Erfassung von Projektinformationen durch verschiedene Funktionen. Diese sind im Einzelnen [JUST-Team (2004a)]:

JUST-Objekte verwalten

JUST-Objekte werden mit Hilfe von Reitern angesprochen. Jedes JUST-Objekt kann über die vier Reiter *Details*, *Beziehungen*, *Historie* und *Zugriffsrechte* bearbeitet werden. Optional kann es auch einen Reiter *Kommentar* geben. Durch Plugins ist es möglich, weitere Reiter hinzuzufügen.

Details: Im Reiter Detail können der Titel, die Beschreibung, die Priorität und weitere pluginspezifische Eigenschaften eines JUST-Objektes festgelegt werden. Zusätzlich können noch Dateien angehängt werden.

Beziehungen: Mit Hilfe des Reiters Beziehungen lassen sich JUST-Objekte untereinander verknüpfen.

Historie: Der Reiter Historie zeigt die Entwicklungsgeschichte eines JUST-Objektes an. Mit dessen Hilfe lassen sich alte, bereits versionierte JUST-Objekte wiederherstellen.

Zugriffsrechte: Mit Hilfe des Reiters Zugriffsrechte ist es möglich, Zugriffsrechte für ein Baumelement festzulegen.

Kommentar: Der Reiter Kommentar beinhaltet die Kommentarfunktion. Durch ihn werden bereits erstellte Kommentare angezeigt und neue Kommentare erstellt.

Suche

In JUST wird sowohl eine Freitextsuche als auch eine Detail-Suche angeboten.

Freitextsuche: Mit Hilfe der Freitextsuche kann nach JUST-Objekten gesucht werden, deren Titel oder Beschreibung einem Suchtext entsprechen.

Detailsuche: Mit Hilfe der Detailsuche kann nach spezifischen Eigenschaften eines JUST-Objektes gesucht werden. So lässt diese Suche es unter anderem zu, nach JUST-Objekten zu suchen, die ein bestimmter Benutzer erstellt hat, oder die in einem gewissen Zeitraum entstanden sind.

Reports erstellen

Mit Hilfe der Reportfunktion können JUST-Objekte (wie Anforderungen oder Modification Requests) in einer druckbaren Darstellung ausgegeben werden.

Vergabe von Zugriffsrechten

JUST verfügt über ein simples Rechtekonzept für JUST-Objekte. Die Zugriffsrechte werden entweder für ein JUST-Objekt oder rekursiv für einen ganzen Teilbaum vergeben. Bei der Vergabe der Zugriffsrechte gelten folgende Regeln

- *Der Anwender hat keine Rechte:* Hat der Anwender keine Rechte auf ein JUST-Objekt, so ist es für ihn nicht sichtbar.
- *Der Anwender hat Leserechte:* Hat der Anwender Leserechte auf ein JUST-Objekt, so kann er sich dessen Eigenschaften anzeigen lassen, diese aber nicht bearbeiten.
- *Der Anwender hat Lese- und Schreibrechte:* Hat der Anwender Lese- und Schreibrechte auf ein JUST-Objekt, so kann er dieses auch bearbeiten.

Administration [JUST-Team (2004c)]

Die Administration von JUST lässt sich in drei Komponenten aufgliedern:

Benutzerverwaltung: Mit Hilfe der Benutzerverwaltung können neue Benutzer angelegt, bereits erstellte Benutzer geändert oder erstellte Benutzer gelöscht werden.

Gruppenverwaltung: Mit Hilfe der Gruppenverwaltung können vorhandene Benutzer zu organisatorischen Einheiten gruppiert werden. Mit ihr können Gruppen angelegt, bearbeitet und gelöscht werden.

Projektverwaltung: Mit Hilfe der Projektverwaltung können Projekte in JUST angelegt, bearbeitet, gelöscht oder bereinigt werden. Die Bereinigung eines Projektes löscht alle Versionen (bis auf die aktuelle Version) eines JUST-Objektes.

2.2.2 Architektur von JUST

Dieses Kapitel beschreibt die Architektur, die dem System JUST zugrunde liegt [JUST-Team (2004b)]. Ein Überblick über die Architektur gibt die Abbildung 2.4.

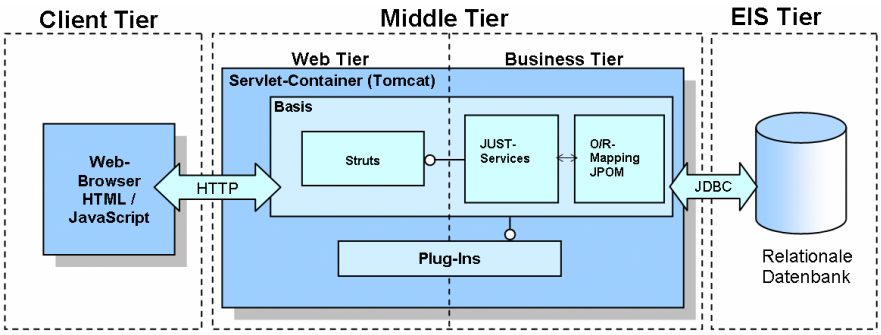


Abbildung 2.4: Architektur von JUST [JUST-Team (2004b)]

Bei der dargestellten Architektur handelt es sich um eine Multi-Tier-Architektur. Dies bedeutet, dass das System in mehrere Schichten (Tiers) unterteilt ist. Das JUST-System ist in drei Schichten unterteilt:

- *Client Tier:* Die Client-Schicht repräsentiert den Anwender, der mit Unterstützung eines Web-Browsers mit dem Servlet-Container (im Falle von JUST ist es ein beliebiger Web-Container) interagiert. Als Kommunikationsprotokoll wird das Standardprotokoll HTTP verwendet.
- *Middle Tier:* Die mittlere Schicht entspricht dem Servlet-Container, der auf einem Server läuft. Die Anfragen, die vom Client kommen, werden in dieser Schicht bearbeitet und entsprechende Antworten an den Client zurückgesendet. Diese Schicht wird ihrerseits wieder in zwei Schichten unterteilt:
  - *Web Tier:* Die Web-Schicht beinhaltet die Präsentationslogik. Diese wird mit Unterstützung des Framework Struts realisiert.
  - *Business Tier:* Die Geschäftsschicht beinhaltet die gesamte Geschäftslogik, die durch die JUST-Services[1] und die eigentlichen fachlichen Klassen repräsentiert wird.
- *EIS Tier:* Das Enterprise Information System (EIS) Tier entspricht einer relationalen Datenbank, die durch die JUST-Services und mit Hilfe des O/R-Mapping-Framework JPOM aus der mittleren Schicht angesprochen wird.

Die Architektur von JUST sieht ein Plugin-Konzept vor. Dadurch kann das System zu einem späteren Zeitpunkt durch zusätzliche Plugins erweitert werden. Die einfache Schnittstelle zwischen dem JUST-System und den Plugins sowie die gut dokumentierten, einfach gehaltenen Dienste machen dies möglich.

2.3 KoPra

Der Anwender soll zukünftig bei der Erfassung und Verwaltung von Projektinformationen mit Hilfe von JUST zu einem methodischen und systematischen Vorgehen angehalten werden. Er soll einer konkreten Methodik folgen, die ihm helfen soll, seine Aufgaben zu erkennen, anzugehen und zu lösen. Diese Unterstützung wird durch einen Wizard erzielt. Das heißt aber nicht, dass die Führung des Benutzers durch einen Wizard starr und für alle Projekte gleich ablaufen soll, sondern individuell auf die Ziele und Eigenheiten eines Projektes ausgerichtet werden können muss. Um das zu erreichen, wird das Werkzeug KoPra benötigt. Die Prozesse aus KoPra dienen nämlich als Grundlage für den in JUST eingesetzten Wizard.
KoPra steht für „konfigurierbarer Prozess- und Projektassistent“ und wurde ebenso wie JUST im Rahmen eines einjährigen Studienprojekts an der Universität Stuttgart am Institut für Softwaretechnologie entwickelt. Das Projektteam bestand aus sechs Studierenden. KoPra ist ein webbasiertes Prozessmodellierungswerkzeug mit dessen Hilfe Prozesse, Projekte

[1] JUST-Services sind JAVA-Klassen, in denen die Geschäftlogik von JUST zusammengefasst ist. Alle Service-Klassen erben von der Basisklasse *JustServices*, in der wichtige Methoden definiert sind, die alle Service-Klassen benötigen.

und konkrete Projektabläufe erstellt werden können. Zur Modellierung von Prozessen gibt es eine bewusst klein gehaltene Anzahl an Prozesselementen. Dies soll nicht nur die Bedienbarkeit, sondern auch die Lesbarkeit und Einfachheit der erzeugten Prozesse erhöhen.

2.3.1 KoPra im Überblick

Generelles

Die nachfolgende Abbildung 2.5 [KoPra-Team (2005b)] gibt einen groben Überblick über das System.

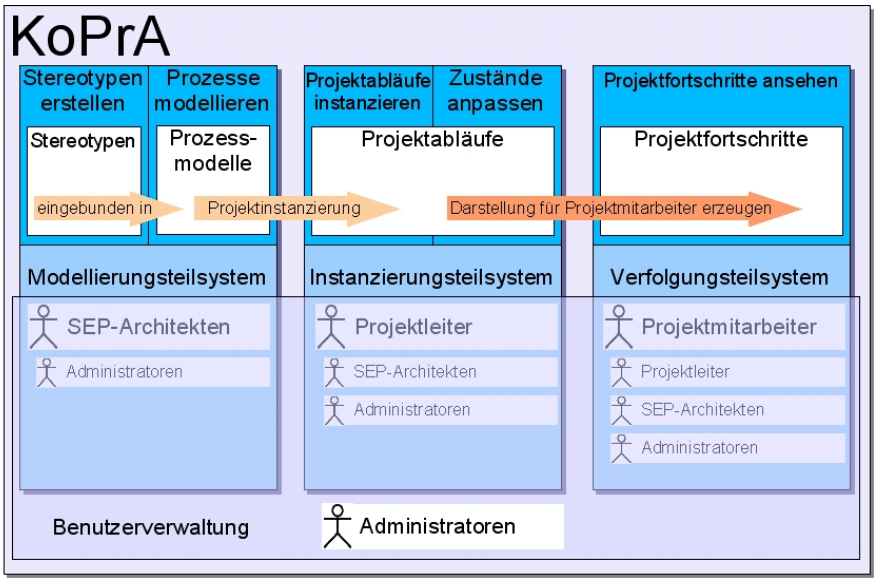


Abbildung 2.5: Überblick über das KoPra-System [KoPra-Team (2005b)]

Prozesse werden in KoPra als *Prozessmodelle* bezeichnet.

Ein *Prozesselement* ist ein Bestandteil eines Prozessmodells.

Ein *Stereotyp* ist ein Schema für ein Prozesselement. Es gibt zum Beispiel für das Prozesselement Artefakt den Stereotyp „Spezifikation“.

Ein *Zustand* ist ein spezielles Attribut eines Prozesselementes. Es kennzeichnet, welchen Status ein Prozesselement im Projektfortschritt und Projektablauf annimmt. So kann z.B. das Artefakt „Spezifikation“ die Zustände „unbearbeitet“, „in Bearbeitung“ und „abgeschlossen“ annehmen.

KoPra gliedert sich in die drei Teilsysteme *Modellierung*, *Instanzierung* und *Verfolgung* auf.

Das *Modellierungsteilsystem* bildet den Kern von KoPra. Über dieses Teilsystem lassen sich Prozesse modellieren.

Über das *Instanzierungsteilsystem* lässt sich auf Grundlage eines Prozessmodells ein Projektablauf erstellen. Dieser kann dann anschließend an die Vorgaben eines realen Projektes angepasst werden.

Über das *Verfolgungsteilsystem* kann der aktuelle Stand eines Projektes eingesehen werden.

KoPra besitzt die vier Benutzergruppen *Administrator*, *SEP-Architekt*, *Projektleiter* und *Projektmitarbeiter*. Jeder Anwender wird einer dieser vier Benutzergruppen zugeordnet. Dadurch wird festgelegt, welche Teile des Systems der Anwender nutzen darf und welche nicht.

Die Oberfläche von KoPra

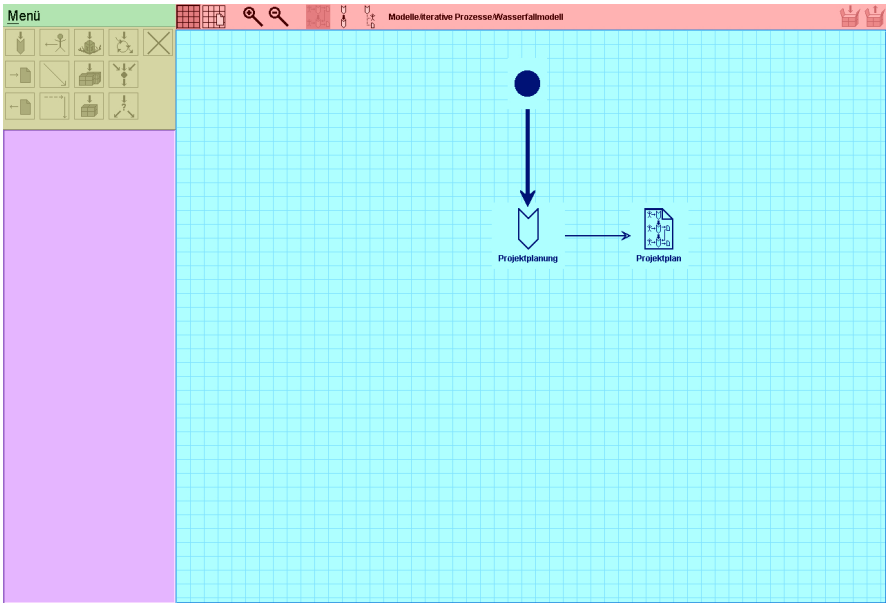


Abbildung 2.6: Schematischer Aufbau von KoPra [KoPra-Team (2005b)]

In diesem Abschnitt wird die Oberfläche des Systems KoPra [KoPra-Team (2005b)] beschrieben. Die Abbildung 2.6 stellt den schematischen Aufbau dar, wobei die einzelnen Bereiche des Fensters farblich hervorgehoben sind.

Hauptfenster (hellblau): Das Hauptfenster ist für die Präsentation des Inhalts verantwortlich. Die anderen Fensterbereiche sind Auswahl- und Entscheidungshilfe für die Ausgabe im Hauptfenster. In ihm werden vor allem die Prozesse erstellt.

Menü (grün): Über das dynamische Menü werden die verschiedenen Teilsysteme des Systems erreicht. Es wird in Abhängigkeit der Zugriffsrechte des Anwenders gefüllt.

Navigationsleiste (rot): Über die Navigationsleite können zusätzliche Funktionen zur Prozessmodellierung angestoßen werden. So kann unter anderem die Ansicht des Modells vergrößert oder verkleinert, Prozesselemente automatisch am Raster ausgerichtet oder das Raster ein- und ausgeblendet werden.

Eigenschaftsfenster (lila): Über das Eigenschaftsfenster können die Eigenschaften eines im Hauptfenster markierten Prozesselementes geändert werden. Es ist in Reiter, die in Abhängigkeit des Elementtyps eingeblendet werden, aufgeteilt.

Werkzeugleiste (gelb): Mit Hilfe der Werkzeugleiste lassen sich neue Prozesselemente in ein Prozess oder in einen Projektablauf einfügen und bestehende Prozesselemente entfernen.

Unterstützte Funktionen

Zur Modellierung von Prozessen und zum Erstellen von Projekten bietet KoPra eine Reihe von Funktionen an, die im nachfolgenden Abschnitt beschrieben werden.

Prozessmodelle verwalten

In KoPra werden Prozessmodelle in einer Verzeichnisstruktur unter einem eindeutigen Namen gespeichert. Diese können zu einem späteren Zeitpunkt wieder geladen, entfernt, umbenannt oder in ein anderes Verzeichnis verschoben werden.

Prozessmodelle bearbeiten

Das Erstellen von Prozessmodellen ist die zentrale Funktion von KoPra. Ein Prozessmodell kann aus verschiedenen Prozesselementen aufgebaut werden. Über die Werkzeugleiste werden Prozesselemente, die in Tabelle 2.1 beschrieben sind, dem aktuellen Prozessmodell hinzugefügt.

Diagrammansichten wechseln

Wird ein Prozess im Hauptfenster dargestellt, so kann dies auf drei unterschiedliche Arten erfolgen:

- *Standardansicht:* In der Standardansicht werden alle Prozesselemente des Prozesses angezeigt.
- *Prozessansicht:* In dieser Ansicht werden alle Artefakte und Rollen ausgeblendet. Dadurch stehen die Aktivitäten und die Prozesselemente, die den Kontrollfluss steuern, im Vordergrund.
- *Baumansicht:* Die Baumansicht ist unterteilt in verschiedene Hierarchieebenen, in denen die Prozesselemente alphabetisch angeordnet werden. So werden auf der ersten Ebene alle Aktivitäten, auf der zweiten Ebene deren verbundene Kontrollflusselemente und schließlich auf der letzten Ebene die Artefakte und Rollen dargestellt.

Stereotypen verwalten

Mit Hilfe von KoPra lassen sich unterschiedliche Stereotypen für alle Prozesselemente erstellen und verwalten. Dies umfasst das Erstellen, Entfernen, Umbenennen und Bearbeiten von Stereotypen.

Zustände verwalten

Für jeden Stereotyp lassen sich beliebig viele Zustände erstellen. Zustände sind wichtig, um Projektabläufe darstellen zu können.

Element	Bedeutung
Aktivität	Eine Aktivität ist eine Tätigkeit während eines Prozesses, die zu einem Ergebnis führen soll.
Artefakt	Ein Artefakt ist ein Dokument. Es kann von einer Aktivität konsumiert (verwendet) oder produziert (erzeugt) werden.
Direkte Relation	Eine direkte Relation verbindet zwei Prozesselemente des Prozesses miteinander.
Indirekte Relation	Indirekte Relationen verbinden Aktivitäten mit Artefakten oder Rollen.
Iterationen	Mit Hilfe von Iterationsstartknoten und Iterationsendknoten lassen sich Iterationen modellieren. Die Prozesselemente zwischen dem Start- und Endsymbol werden beliebig oft wiederholt.
Paket	Das Paket repräsentiert einen Teilprozess, der dem aktuellen untergeordnet ist.
Rolle	Eine Rolle symbolisiert eine Person oder Personengruppe, die in einer bestimmten Beziehung zu einem Artefakt oder einer Aktivität steht.
Startknoten	Der Startknoten symbolisiert den Beginn eines Prozesses oder eines Projektes.
Verbindungsknoten	An einem Verbindungsknoten werden mehrere eingehende Pfade zu einem ausgehenden Pfad zusammengeführt.
Verzweigungsknoten	An einem Verzweigungsknoten spaltet sich der eingehende Pfad in mehrere ausgehende Pfade.

Tabelle 2.1: Prozesselemente eines KoPra-Prozesses

Projektablauf instanzieren

Aus einem beliebigen Prozessmodell lassen sich Projekte erstellen. Ein solches Projekt kann dann vom Projektleiter an die Eigenschaften eines realen Softwareprojekts angepasst werden. Projektmitarbeiter können den Projektfortschritt aus einem Projektablauf mitverfolgen.

Administration

KoPra verfügt über eine Benutzerverwaltung, mit dessen Hilfe Benutzer und verschiedene Zugriffsrechte vergeben werden können.

2.3.2 Architektur von KoPra

Dieses Kapitel beschreibt die Architektur, die dem System KoPra zugrunde liegt [KoPra-Team (2005a)]. Die Abbildung 2.7 veranschaulicht grob den Architekturaufbau des Systems. KoPra ist ebenso wie JUST ein webbasiertes System, das in drei Schichten (Tiers) aufgeteilt ist.

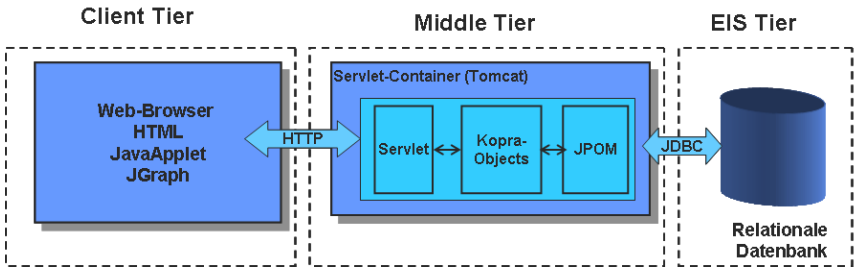


Abbildung 2.7: Architektur von KoPra

KoPra ist ebenso wie JUST eine Server-Client-Anwendung mit den drei Schichten *Client Tier*, *Middle Tier* und *EIS Tier*. Im Detail zeigen sich jedoch große Unterschiede zwischen den beiden Systemen:

- *Client Tier:* Der Anwender interagiert über einen Browser mit dem System. Ein JavaApplet auf dem Client ist zuständig für die Repräsentationslogik. Mit Unterstützung von Java Swing (6.1.2) und der Java Bibliothek JGraph (6.1.3) wird die GUI des Applets dargestellt.
- *Middle Tier:* Die mittlere Schicht ist sehr einfach konzipiert. Auf einem Server liegen die fachlichen Klassen von KoPra. Über diese wird mittels des O/R Mapping Frameworks JPOM (6.1.4) auf eine relationale Datenbank zugegriffen. Des Weiteren befindet sich dort eine zentrale Klasse, die die Anfragen der Clients entgegennimmt, bearbeitet und die entsprechenden Informationen an den Client zurücksendet. Sie ist die Schnittstelle zwischen Client und Server.
- *EIS Tier:* Das Enterprise Information System (EIS) Tier entspricht einer relationalen Datenbank, die durch die fachlichen Klassen und mit Hilfe des O/R-Mapping-Frameworks JPOM aus der mittleren Schicht angesprochen wird.

3 Analysemethoden

In diesem Kapitel stelle ich die Ergebnisse meiner Untersuchungen über die Analysemethoden des Requirement Engineerings vor. Die Ergebnisse bilden die Grundlage für das in Kapitel 4 beschriebene Metamodell der Analysemethoden.

3.1 Rückblick

In den frühen Tagen der Computer- und Softwareentwicklung schrieb keine Einzelperson einfach mal so ein großes Programm. Ihm fehlten die nötigen Hardwareressourcen (Maschinen hatten einen sehr kleinen Arbeitsspeicher, Programme mussten mit sehr langsamen magnetischen Trommeln als Festplattenspeicher arbeiten und Prozessoren hatten eine Durchlaufzeit von mehreren 100 Mikrosekunden), sodass sie auf kleine Programme beschränkt waren. In den 60ern und 70ern des vergangenen Jahrhunderts wandelte sich aber der Computermarkt. Hardware wurde zunehmend kostengünstiger und erschwinglicher trotz der Tatsache, dass das Leistungsvermögen stark zunahm. Ein Ergebnis dieses plötzlichen Wandels war die zunehmende Nachfrage nach komplexeren und größeren Anwendungen. Hochsprachen wie Fortran und Cobol traten in die Computerszene ein. Sie wurden schnell zu wichtigen Werkzeugen, um Softwaresysteme zu erstellen. Solche Sprachen sorgten für eine erhöhte Produktivität des Entwicklers und des gesamten Entwicklungsteams - ironischerweise stieg dadurch das Verlangen, Softwaresysteme mit immer größer werdender Komplexität umsetzen zu wollen.

Viele Methoden wurden daher in den 60ern und 70ern, zu der Zeit des großen Umbruches, zwischen Hardware und Software entwickelt und richteten sich an die komplexer werdende Software. Die einflussreichste unter ihnen war die *top-down structured design* Methode, auch bekannt unter dem Namen *composite design*. Diese Methode wurde direkt beeinflusst durch die zwei neuen Programmiersprachen Fortran und Cobol. Diese Sprachen haben die fundamentale Eigenschaft, in kleinere Unterprogramme aufgeteilt zu werden. Das endgültige Programm hat die Aufrufstruktur eines Baumes, in welchem die einzelnen Unterprogramme ihre Arbeit dadurch erfüllen, indem sie andere Unterprogramme aufrufen. Und genau das ist der Denkansatz, der hinter der *top-down structured design* Methode steckt: Das große Problem wird in mehrere kleinere Probleme (Unterprogramme) zerstückelt (*algorithmic decomposition*).

Das Leistungsvermögen heutiger Rechner ist mit den von 1960 und 1970 nicht mehr zu vergleichen. *Structured design* und *structured analysis* haben keineswegs an Wert verloren, jedoch wie Jacob Stein treffend sagte [Booch (1994)]:

„Structured programming appears to fall apart when applications exceed 100.000 lines or so of code“. Es wurden viele Analyse- und Entwurfsmethoden entwickelt. Viele von ihnen sind sehr an der *structured analysis and design* orientiert, aber auch neue sind in den Jahren hinzugekommen, nicht zuletzt die objektorientierten Methoden, die den Fokus auf Objekte und Klassen legen.

Sommerville [Booch (1994)] hat einmal angeregt, die Methoden, in die drei Kategorien:

- Top-down structured design,
- Data-driven design und
- Object-oriented design

aufzuteilen.

3.2 Die Analyse

Ein wichtiger Wegbereiter der Analyse war Tom DeMarco [Weinberg (1980); DeMarco (1979)], der die „Structured Analysis“ als erster erfolgreich vertrat und der den Begriff der Analyse wie folgt definierte:

„Analysis is the study of a problem, prior to taking some action. In the specific domain of computer systems development, analysis refers to the study of some business area or application, usually leading to the specification of a new system.“

Alternativ kann man sie auch folgendermaßen definieren:

Die Analyse ist ein Prozess, bei der wir uns von der realen Welt ein idealisiertes Modell erstellen und hierbei diverse Zwänge ignorieren.

Bestrebung in der Analysephase ist es, das Problem genau zu beleuchten, alle notwendigen Erkenntnisse über das Problemfeld zu gewinnen und sie in ein Analysemodell oder in eine Problembeschreibung zu gießen. Ist das Projekt an einen Kunden gebunden, so müssen die gesammelten Probleme durch die Wünsche des Kunden ergänzt werden und diese abschließend in ein Lastenheft geschrieben werden. Das Analysemodell stellt eine Abstraktion des Problems dar. Das Modell wiederum dient als Grundlage für die Spezifikation und darf keinen Implementierungsbedingungen unterliegen.

Um die Aufgabe der Analyse zu meistern, muss der Systemanalytiker nach DeMarco [DeMarco (1979)] folgende Ergebnisse anstreben:

- das bestmögliche Ziel identifizieren,
- eine detaillierte Spezifikation des Zieles erstellen und
- alle wesentlichen Größen, die mit dem Ziel verbunden sind, abschätzen (Kosten, Meilensteine, Risikoabschätzung, Qualitätsansprüche).

Kommunikation:

Um eine Software erfolgreich entwickeln zu können, müssen alle Beteiligten eine ihnen verständliche Sprache sprechen. Jedoch verwenden die Benutzer des zu erstellenden Systems, die Entwickler oder auch der Kunde ihre eigene Fachsprache, welche für ihren Fachbereich gängig ist. Es können somit leicht Missverständnisse entstehen, die aufgrund von falschen Annahmen und Schlussfolgerungen eines Wortes herrühren. Gleiche Worte können nämlich von verschiedenen Personengruppen unterschiedlich aufgefasst werden. Die Folge sind fehlerhafte Ideen und Konzepte. Die Aufgabe des Systemanalytikers ist es, die Anforderungen der Benutzer und Kunden in fachlich-technische Spezifikationen für die Programmierer zu übersetzen.

Wandelnde Anforderungen:

Im Laufe der Analysephase muss sich nun der Systemanalytiker in die Welt des Benutzers und Kunden einfinden und dessen spezifische Fachsprache erlernen. Anforderungen und Wünsche werden sich während der Projektzeit ändern. Ziele und Anforderungen können sich wandeln. Das muss in der Analysephase ebenfalls berücksichtigt werden. Es müssen Möglichkeiten zur Anpassung und Aufnahme von neuen oder sich ändernden Anforderungen in die Analyse einbezogen werden.

3.3 Übersicht über die objektorientierten Methodologien

Das untere Schaubild (Abbildung 3.1) zeigt die historische Entwicklung der objektorientierten Methodologien ab den 90er Jahren. Es ist deutlich zu erkennen, dass es Ende der 80er, Anfang der 90er Jahre eine Zeit der Methodenentwicklung gab, die in enger Verbindung mit der aufkommenden objektorientierten Denkweise stand. Viele Methoden sind entstanden und in den darauf folgenden Jahren war man bemüht, diese möglichst zu standardisieren.

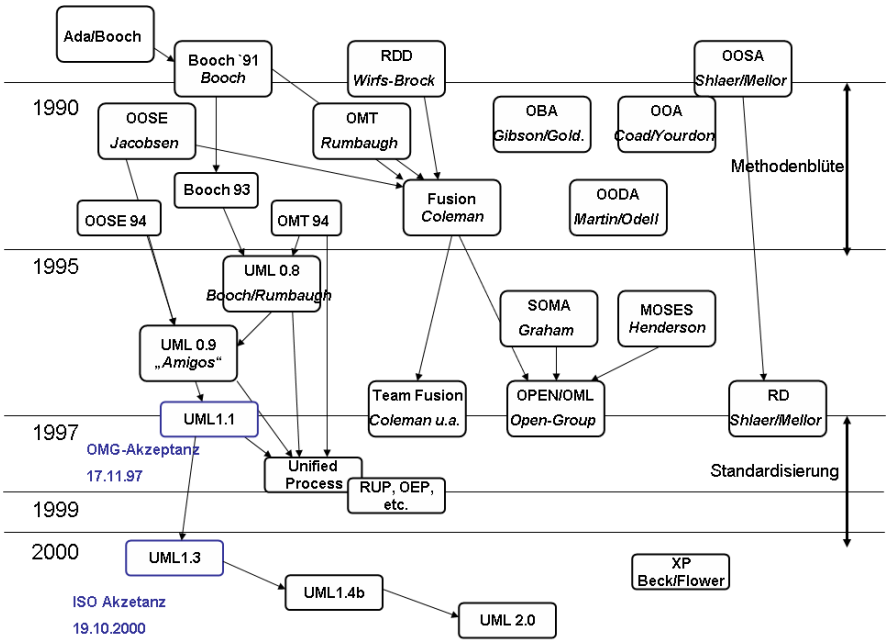


Abbildung 3.1: Historische Entwicklung der objektorientierten Methodologien [Oesterreich (2004)]

Die hier aufgeführten Methoden und Methodologien werden nicht alle vorgestellt werden können. Der Fokus wird auf folgende objektorientierte Methodologien liegen:

- *Object Modeling Technique (OMT)*
- *Rational Unified Process (RUP)*

Als ein Vertreter einer nicht objektorientierten Methodologie wird die *Structured Analysis* vorgestellt.

3.4 Structured Analysis

Tom DeMarco ist der Wegbereiter der strukturierten Analyse. Mit seiner Veröffentlichung aus dem Jahre 1979 „Structured Analysis and Systems Specification“ war er der Erste, der erfolgreich für diese Methode eintrat.

Die strukturierte Analyse unterstützt den Systemanalytiker bei seinen Aufgaben. Durch einfache, unkomplizierte Modelle wird das Verständnis des Problems erleichtert und Kommunikationsprobleme zwischen Auftraggebern und Projektbeteiligten verringert.

Mit Hilfe von Datenflussdiagrammen (DFD), Datenstrukturdiagrammen (DSD) und dem Datenverzeichnis werden die Ziele und Anforderungen des neuen Systems und der momentane Ist-Zustand des aktuellen Systems modelliert. (Nähere Informationen hierzu sind im Anhang A zu finden.)

3.4.1 Methodisches Vorgehen

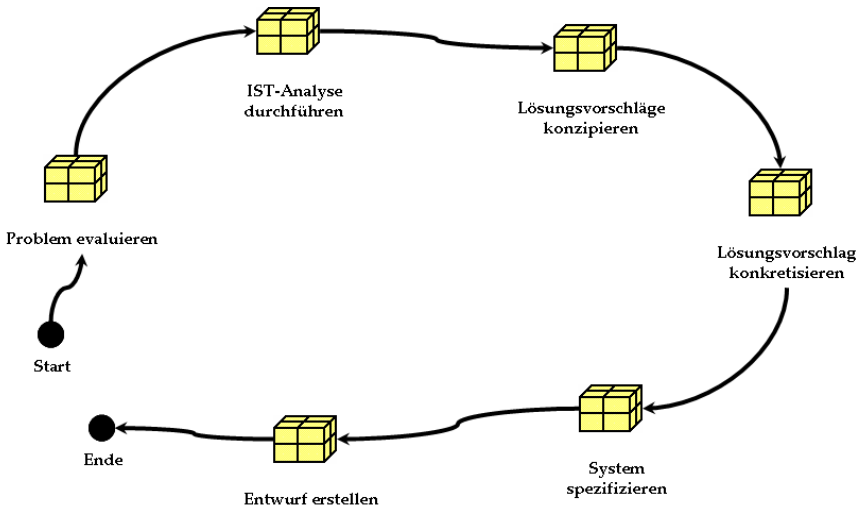


Abbildung 3.2: Übersicht über die „Structured Analysis“-Phasen

Eine mögliche Vorgehensweise der „Structured Analysis“ ist in der Abbildung 3.2 dargestellt. Die einzelnen Teilprozesse (oder auch Phasen genannt) werden sequentiell von der Problemevaluation bis hin zur Erstellung des Entwurfs durchlaufen. Eine Übersicht und Definition der Notationselemente, die in dieser Abbildung verwendet wurden, befindet sich im Anhang B. Nachfolgend werden die Teilprozesse einzeln aufgeführt und die wesentlichen Aktivitäten vorgestellt.

Teilprozess 1: Problemevaluation / Angebotsüberprüfung

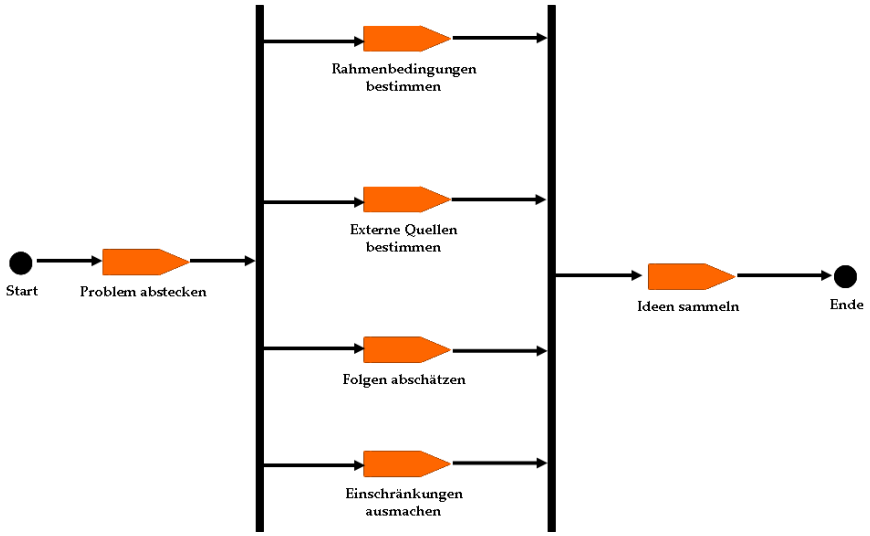


Abbildung 3.3: Teilprozess: Problemevaluation

Die Aufgabe in diesem Schritt ist es, das Problem des Kunden kennen zu lernen. Das Ergebnis dieses Teilprozesses ist eine einleitende Problembeschreibung. Das Dokument sollte nach Fertigstellung von einer entsprechenden Kontrollstelle, wie z.B. der Qualitätssicherung, geprüft werden.

Das Problem abstecken
Zu Beginn der Analyse müssen Informationen zu einem Angebot gesammelt werden. Hier können Ausschreibungen oder Anfragen als Quelle dienen. Etwaige Hintergrundinformationen, die Aufschluss über die Ursachen oder die Auswirkungen des in dem Angebot beschriebenen Problems geben, können ebenso nutzbringend sein. Auch der Anlass für ein ausgeschriebenes Angebot kann Erkenntnisse bringen.

Rahmenbedingungen bestimmen
Nachdem die ersten Informationen über ein Angebot und dessen Problemstellung gesammelt wurden, müssen die Rahmenbedingungen geklärt werden. Dazu sollten Informationen über das Auftrag gebende Unternehmen eingeholt und erste Kontakte zu dem Kunden aufgebaut werden. Es ist ebenfalls wichtig, dass Konkurrenten, die um das Angebot „mitstreiten“, genauer untersucht werden. So können deren Stärken und Schwächen, Einfluss auf die Aufstellung und Präsentation des eigenen Unternehmens haben.

Externe Quellen bestimmen
Zur Lösung eines Problems, bietet es sich an, verwandte, ähnliche Problemlösungen zu studieren, um deren Erkenntnisse und Ergebnisse für das eigene Projekt zu verwenden. Hierzu müssen solche externen Quellen erstmal ausfindig gemacht werden.

Folgen abschätzen
Eine wesentliche Aufgabe der Problemanalyse ist es, die Folgen eines Problems abzuschätzen. Hierzu sind Zahlen, wie der Ausfall an Geschäftseinnahmen oder vermeidbare Kos-

ten, sehr gut geeignet. Aber auch ineffiziente Geschäftsoperationen müssen aufgedeckt werden.

Einschränkungen ausmachen
Im Zusammenhang mit der Aufgabe müssen auch mögliche Einschränkungen für das Projekt untersucht werden, wie z.B. Vorschriften, Grundsätze, Unternehmensgesetze, einzusetzende Methoden, Standards und Richtlinien sowie Stichtage (Meilensteine).

Ideen sammeln
Die letzte Tätigkeit, die es gilt in diesem Teilprozess durchzuführen, besteht darin, Ideen und Lösungsvorschläge für die kommende Umsetzung zu sammeln.

Teilprozess 2: Ist-Analyse durchführen

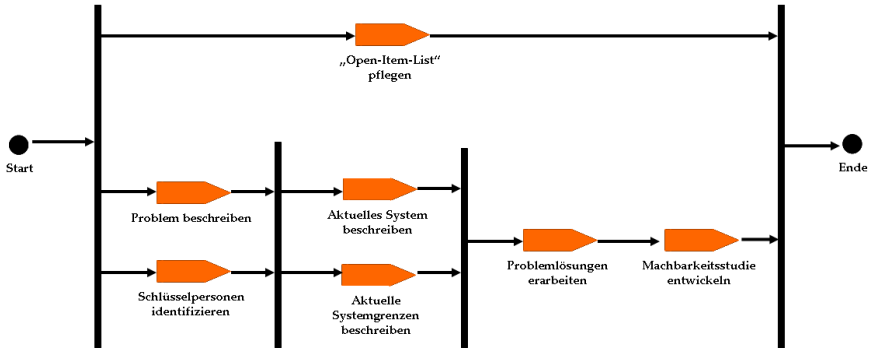


Abbildung 3.4: Teilprozess: Ist-Analyse durchführen

Dieser Teilprozess beschreibt die Ist-Analyse eines bereits bestehenden Softwaresystems. Dabei sollen die Fragen

- „Welche Systeme existieren bereits?“,
- „Wie wird aktuell trotz des Problems im Unternehmen gearbeitet?“ und
- „Was sind die Auswirkungen des Problems für das Unternehmen?“

beantwortet werden. Zusätzlich sollen konkrete Lösungsansätze erarbeitet und eine Machbarkeitsstudie erstellt werden.

Das Problem beschreiben
In dieser Aktivität wird versucht die Frage „Was ist eigentlich das Problem?“ zu beantworten. Der einfachste Weg ist es, den Kunden zu befragen. Dabei sollte auf unkomplizierte, gewöhnliche Fragen Wert gelegt werden, die dann in einem Fragenkatalog oder Fragebogen zusammengetragen werden sollten.

Schlüsselpersonen identifizieren
Informationen über Auftraggeber, Benutzer, Interessengruppen oder Kunden, die in engem Kontakt mit dem zukünftigen oder aktuellen System stehen, sind wesentlich für die weitere Analyse. Durch Beantwortung einfacher Fragen, wie z.B. „Wer sind die Benutzer des aktuellen Systems?“, können potenzielle Schlüsselpersonen ausgemacht werden.

Aktuelles System beschreiben
Wesentlicher Bestandteil der Ist-Analyse ist die Beschreibung des aktuellen Systems. Mit Hilfe der „Structured Analysis“ eigenen Diagrammen, wie z.B. den Datenfluss- und den Datenstrukturdiagrammen, sollen die Funktionen des Systems modelliert werden. Neben den Einsatz von Diagrammen ist auch der Einsatz von Notizen und Beschreibungen in einfachen Worten hilfreich. Ziel ist es, eine verständliche, vollständige, konsistente Programmbeschreibung zu erstellen, aus der auch erkenntlich wird, mit welchen Methoden und Techniken das System benutzt wird.

Aktuelle Systemgrenzen beschreiben
Das aktuell eingesetzte System unterliegt Grenzen und Auflagen. Diese können eine der Ursachen für das Problem sein. Daher ist es wichtig, dass die Grenzen des aktuellen Systems ergründet und beschrieben werden. Zu ihnen können unter anderem die Ausführungszeit, der benötigte Speicherplatz, die Fehlerrate, die Kosten, die für den Betrieb oder die Wartung anfallen, getroffene Vereinbarungen oder auch die zur Verfügung stehenden Funktionen zählen. Aber auch durchgeführte Änderungen im Geschäftsbereich, wie z.B. die Änderung eines bestehenden Prozesses oder eine Werkzeugeinführung, müssen auf ihren Einfluss auf das System untersucht werden.

Problemlösungen erarbeiten
Nachdem nun das aktuelle System analysiert und das Problem eingegrenzt wurde, müssen Lösungsansätze und Ideen entwickelt werden, die helfen sollen, das Problem zu unterbinden. Für jede viel versprechende Lösungsalternative sind dazu die zu erwartenden Kosten und der Aufwand für die Umsetzung zu ermitteln. Ebenso ist auch wichtig, dass der Nutzen und der Mehrgewinn, den die Einführung der Lösungsalternative mit sich bringt, bestimmt werden.

Machbarkeitsstudie entwickeln
Zum Abschluss der Ist-Analyse werden die Ergebnisse der Untersuchung in einer Machbarkeitsstudie zusammengetragen. Mit ihr wird Bilanz gezogen, ob es für das eigene Softwarehaus möglich ist, dass Projekt ökonomisch erfolgreich durchzuführen.

„Open-Item-List“ pflegen
Parallel zur Analyse des aktuellen Systems wird ein Dokument erstellt, in dem die offenen Punkte und Probleme fortlaufend protokolliert werden.

Teilprozess 3: Lösungsvorschläge konzipieren

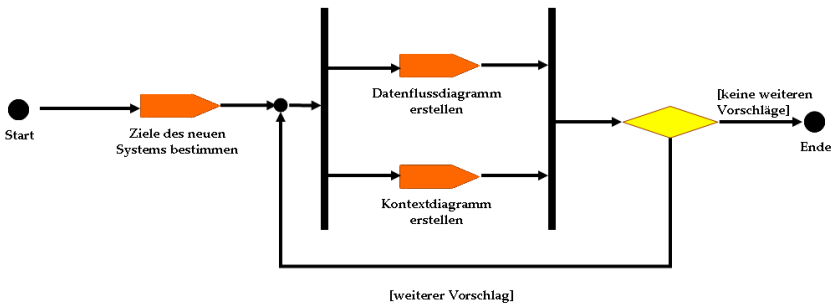


Abbildung 3.5: Teilprozess: Lösungsvorschläge konzipieren

In dieser Phase sollen die Systemanalytiker das Wissen von dem aktuellen System und die Informationen aus der Problembeschreibung zur Erstellung von Lösungsvorschlägen verwenden. Die Ergebnisse sollen skizzenhafte Entwürfe des neuen Systems sein. Es handelt sich nur um gedankliche, vorläufige Vorschläge, von denen einer später ausgewählt und konkretisiert wird.

Ziele des neuen Systems bestimmen
Entwickelt man ein neues System, so ist darauf zu achten, dass zuvor die Ziele und die Absichten, die mit dem neuen System verfolgt werden sollen, benannt werden. Des Weiteren ist es wichtig, die Grenzen des aktuellen Systems aufzuzeigen, denn dadurch wird erst deutlich, worin der Nutzen einer Neuentwicklung liegt. Bei der Angabe des Zieles ist auf präzise, nachvollziehbare Aussagen zu achten. Ebenso müssen die gesammelten Anforderungen und Wünsche des Kunden gruppiert und abschließend gewichtet werden. Diese sind die Basis für die kommenden Phasen.

Datenflussdiagramm erstellen
Nachdem die Ziele des neuen Systems bestimmt wurden, werden mit Hilfe von Datenflussdiagrammen die wichtigsten Anforderungen des Kunden beschrieben.

Kontextdiagramm erstellen
Mit Hilfe von Kontextdiagrammen wird der Anwendungsbereich des zu modellierenden Systems beschrieben. Hierzu werden die Schnittstellen mit der Außenwelt modelliert, wobei das eigentliche System als ein großer Prozess dargestellt wird.

Die Raute repräsentiert eine bedingte Verzweigung des Pfades. Dadurch, dass sie den Kontrollfluss am Ende spaltet, können entweder weitere Lösungsvorschläge erstellt oder der Teilprozess beendet werden.

Teilprozess 4: Lösungsvorschlag konkretisieren

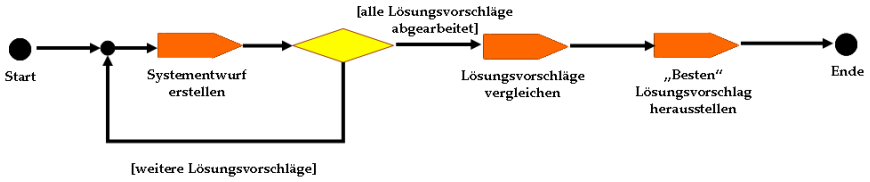


Abbildung 3.6: Teilprozess: Lösungsvorschlag konkretisieren

Ziel dieses Teilprozesses ist die Konkretisierung der Lösungsalternativen, um möglichst die „Beste“ unter ihnen finden zu können. Das Ergebnis schließt mit einem Vergleich der Alternativen ab und favorisiert einen Vorschlag, der den Kundenwünschen und Anforderungen am meisten entgegenkommt.

Systementwurf erstellen
Die Methodik der „Structured Analysis“ sieht vor, dass zu jedem Lösungsvorschlag ein System in groben Zügen entworfen wird. Diese Systeme werden dann untereinander verglichen, um zu ermitteln, welches System das „Beste“ ist. Neben den Nutzen und der Wirkung für den Benutzer können Faktoren wie Wartbarkeit, benötigte Entwicklungskosten und der Zeitaufwand für die Entwicklung des Systems so besser abgeschätzt werden.

Lösungsvorschläge vergleichen
Nachdem die rudimentären Systeme für die Lösungsalternativen entwickelt wurden, werden sie alle untereinander verglichen und die als geeignet erscheinende ausgewählt. Als Kriterien können die Benutzerzufriedenheit, der geschätzte Entwicklungsaufwand, die geschätzten Entwicklungskosten, das Erreichen der Kundenziele oder andere nicht-funktionale oder funktionale Kriterien dienen.

„Besten" Lösungsvorschlag herausstellen
Nun muss der favorisierte Lösungsvorschlag beschrieben werden. Außerdem muss die Wahl für diese Alternative begründet werden.

Teilprozess 5: System spezifizieren

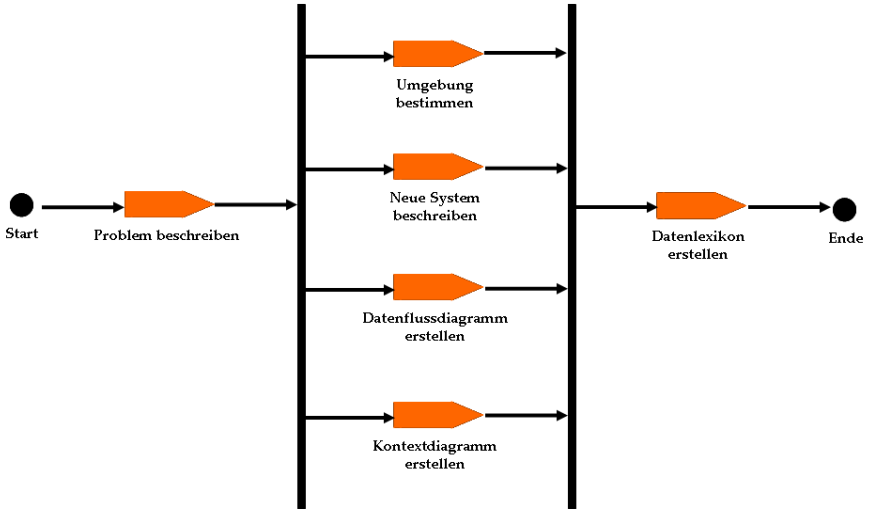


Abbildung 3.7: Teilprozess: System spezifizieren

Das Ziel dieses Teilprozesses ist es, die gesammelten Anforderungen und Wünsche des Kunden, die Ziele des umzusetzenden Systems und die Erkenntnisse aus der Problembeschreibung zu einer Spezifikation auszuarbeiten. Diese sollte sowohl die nicht-funktionalen als auch die funktionalen Anforderungen an das neue System abdecken.

Problem beschreiben
Zu Beginn soll ein weiteres Mal das Problem beschrieben werden. Dabei soll noch mal die Problematik und die aktuelle Situation des Auftraggebers geschildert werden.

Neues System beschreiben
Um das neue System zu beschreiben, müssen präzise, nachvollziehbare, schlüssige Aussagen verwendet werden. So sollten vor allem die allgemeinen Fähigkeiten und die erwarteten Verbesserungen, die das neue System bringen soll, in der Spezifikation beschrieben werden. Aber auch die Grenzen und der Zeit- und Kostenaufwand für die Entwicklung sollten benannt werden. Die Spezifikation ist letztendlich ein juristisches Dokument und alle getroffenen Aussagen sind für beide Seiten (Auftraggeber und Auftragnehmer) verbindlich.

Datenflussdiagramm erstellen
Die Datenflussdiagramme der Structured Analysis können zur Modellierung von Funktionen eingesetzt werden.

Kontextdiagramm erstellen
Durch den Einsatz von Kontextdiagrammen ist es möglich, das neue System in seinem Anwendungsbereich zu modellieren. Dabei wird das ganze System als ein Prozess dargestellt, der mit seiner Umgebung interagiert.

Umgebung bestimmen
Ebenso wie die funktionalen Anforderungen müssen auch die nicht-funktionalen Anforderungen gesammelt und dokumentiert werden. Teil dieser Tätigkeit ist es, die Einflüsse der Umgebung auf das neue System zu bestimmen. So können Eigenschaften wie Bedienbarkeit, Sicherheit, Antwortzeiten des Systems oder anfallende Datenlast für das zu entwickelnde System signifikant sein und müssen demnach aufgeführt werden.

Datenlexikon erstellen
Als ein möglicher letzter Schritt soll das Datenlexikon (siehe Anhang A) erstellt und angepasst werden. Das in den Datenfluss- und Kontextdiagrammen verwendete Vokabular wird in einem solchen Dokument definiert. Dies sind vor allem die Prozesse und Datenflüsse der DFDs.

Teilprozess 6: Entwurf erstellen

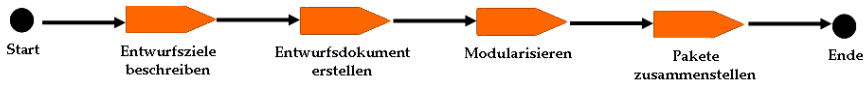


Abbildung 3.8: Teilprozess: Entwurf erstellen

Ziel dieses Teilprozesses ist es, eine Architektur der Anwendung zu entwerfen und die zugehörigen Datenstrukturen zu beschreiben. Das Ergebnis sollte so präzise und ausgereift sein, dass es alle Informationen zur Verfügung stellt, um das System umsetzen zu können.

Entwurfsziele beschreiben
Bevor der Entwurf in der Structured Analysis erstellt wird, müssen die Entwurfsziele ausgemacht werden. So stellen zum Beispiel Antwortzeiten, Ausführungszeiten, zu gewährleistender Funktionsumfang, beanspruchte Ressourcen oder auch die veranschlagten Wartungskosten Ziele dar, die es im Entwurf zu berücksichtigen gilt.

Entwurfsdokument erstellen
Die zentrale Tätigkeit in diesem Teilprozess ist es, den Entwurf zu erstellen. Im Entwurf werden unter anderem der Zugriff auf externe Ressourcen, Sicherheitskonzepte und wichtige Kernfunktionen des neuen Systems und die Architektur beschrieben.

Modularisieren
Die einzelnen Module des Entwurfs müssen in einer Hierarchie angeordnet werden. Dadurch kann man eine Schichtenstruktur erzielen, mit deren Hilfe Modulabhängigkeiten eingeschränkt werden können. Außerdem sollen die Module gruppiert werden.

Pakete zusammenstellen
Wenn das zu entwickelnde Projekt zu groß oder komplex ist, bietet es sich an, dieses in mehrere kleinere Teilprojekte aufzuteilen. Eine ähnliche Aufteilung ist auch mit Anwendungen möglich. Durch Einführung von Paketen können einzelne Arbeitsteams an unterschiedlichen Aufgaben arbeiten.

3.4.2 Gewonnene Erkenntnisse

Die Structured Analysis war die erste Methodologie die ich tiefer gehender untersuchte und sie zählt meiner Meinung nach zu den wenigen ausgearbeiteten und durchdachten Analysemethoden. Sie hat den Vorzug gegenüber anderen Anforderungsanalysemethoden, wie der OMT, OOSE oder Booch-Methode, dass sie schon früh in der Analyse, bei der Angebotserstellung, beginnt und sich konsequent bis hin zur Implementierung um eine schlüssige Beschreibung eines methodischen Vorgehens bemüht. Es werden viele wichtige Elemente der Anforderungsanalyse von ihr berücksichtigt, wie die Ist-Analyse eines bereits bestehenden Systems, eine ausführliche Ursachenergründung des zu lösenden Problems oder das Zusammenstellen von Wünschen und Forderungen des Auftraggebers.
Ich habe viele Erkenntnisse gerade aus dieser Methodologie für meine weiteren Arbeiten gewinnen können. Ich stelle sie jetzt der Reihe nach vor.

Teilprozess: Die Structured Analysis ist in Phasen unterteilt (siehe Abbildung 3.2). Eine Phase (ab hier von mir als Teilprozess bezeichnet) ist im Grunde nichts weiter als ein Behälter für eine Reihe von Tätigkeiten. Genauer gesagt stellt ein Teilprozess ein Paket von Aufgaben dar, die alle helfen sollen ein Ziel zu erreichen.

Aktivität: Eine Aktivität beschreibt eine Tätigkeit oder eine Aufgabe die es zu lösen gilt. So ist zum Beispiel „Aktuelles System beschreiben“ eine Aktivität. Jede Aktivität ist in einen Prozess eingegliedert, was bedeutet, dass sie einem Ablauf unterliegt. So kann die Aktivität „Aktuelles System beschreiben“ erst ausgeführt werden, wenn die beiden Aktivitäten „Schlüsselpersonen identifizieren“ und „Problem beschreiben“ abgeschlossen sind. Eine Aktivität besteht immer aus einer Beschreibung der durchzuführenden Tätigkeit oder der zu lösenden Aufgabe und einem Vorgehen, das die Frage „Wie ist diese Tätigkeit durchzuführen?“ oder die Frage „Wie ist die Aufgabe zu lösen?“ zu beantworten versucht. Es folgt ein Beispiel in dem diese Eigenschaften einer Aktivität vorgestellt werden.

Aktivität: „Schlüsselpersonen identifizieren“
Beschreibung: Schlüsselpersonen, die in dem Unternehmen arbeiten, sollen identifiziert werden. Sie verfügen über ein weitreichendes Wissen über das Unternehmen, kennen die Sorgen und Probleme der Mitarbeiter und sind mitverantwortlich für die Budgetierung. Eine Schlüsselperson begleitet das Projekt über die gesamte Projektdauer hinweg, wirkt unterstützend bei der Analyse von Problemen und notwendigen Änderungen und stellt sich für ein Pilotprojekt zur Verfügung.
Vorgehen: Man sollte versuchen die folgenden Fragen zu beantworten: • „Wer sind die Auftraggeber?“ • „Was für Leute arbeiten in diesem Unternehmen?“ • „Was für Teams und Abteilungen gibt es?“ Anschließend sollte man sich mit Mitarbeitern unterhalten, um gemeinsam zu ergründen: • „Was für welche Änderungen schweben ihnen vor?“ und • „Sind sie skeptisch, optimistisch oder gleichgültig gegenüber den zu erwartenden Neuerungen?“

Tabelle 3.1: Beispiel für eine Aktivität

Artefakt: Ein Artefakt in der Structured Analysis ist ein Dokument, das von einer Aktivität entweder verwendet (konsumiert) oder erzeugt (produziert) wird. Ein Beispiel für produzierte Artefakte sind die Datenflussdiagramme, die während der Aktivität „Datenflussdiagramm erstellen“ entstehen. Ein weiteres Beispiel, diesmal aber für ein Artefakt, das konsumiert wird, ist das Entwurfsdokument. Dieses wird von der Aktivität „Modularisieren“ benötigt.

Parallel und sequentiell ablaufende Aktivitäten: In der Structured Analysis können Aktivitäten sowohl sequentiell als auch parallel ablaufen. Die Darstellung von sequentiell ablaufenden Aktivitäten wird durch einen gerichteten Pfeil erzielt, der zwei Aktivitäten miteinander in Beziehung setzt. Sobald die Ursprungsaktivität (die Aktivität, aus der der Pfeil geht) abgeschlossen ist, kann die nächste Aktivität (die Aktivität, zu der der Pfeil zeigt) ausgeführt werden. Das gleichzeitige Ausführen von Aktivitäten, wird mit Hilfe von Synchronisationsbalken erreicht. Ein Synchronisationsbalken kann beliebig viele eingehende Pfade und ausgehende Pfade besitzen. Alle eingehenden Pfade werden zuerst synchronisiert* ehe die ausgehenden Pfade anschließend alle gleichzeitig weiterverfolgt werden.

Bedingte Verzweigung (Spaltung): Es kommt in der Structured Analysis auch vor, dass das weitere Vorgehen von externen Ereignissen und Zuständen abhängt. So wird der Teilprozess drei „Lösungsvorschläge konzipieren“ erst verlassen, wenn es keine weiteren Lösungsvorschläge mehr gibt. Diese Information muss aber von „außen“ kommen und kann nicht vom Prozess selber vorgegeben werden. Für solche äußeren Einflusse, die den Verlauf des Prozesses beeinflussen, habe ich eine Raute eingeführt. Mit deren Hilfe ist es möglich den Kontrollfluss zu steuern.

Zusammenführung von Pfaden (Verschmelzung): Neben dem Synchronisationsbalken, der es ermöglicht mehrere Pfade zusammenzuführen, habe ich noch ein weiteres Element (einen schwarzen Kreis) eingeführt, mit dessen Hilfe sich Pfade zusammenführen lassen. Der schwarze Kreis, verfügt über mehrere Eingänge und einem Ausgang. Im Unterschied zum Synchronisationsbalken, reicht es aus wenn ein eingehender Pfad verfolgt wird, um den ausgehenden Pfad weiterzuverfolgen. Es wird keine Synchronisation über alle eingehende Pfade vorgenommen.

3.5 Object Modeling Technique (OMT)

Ein weiteres Analyseverfahren ist die OMT, die von J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy und W. Lorensen ausgearbeitet und veröffentlicht wurde. Die OMT wendet die objektorientierte Denkweise konsequent auf alle Phasen der Softwareentwicklung von der Analyse über den Entwurf bis zur Implementierung an. Durch die Konzentration auf Objekte und Beziehungen zwischen Objekten wurde damals eine „moderne“ Sicht auf die Entwicklung von Systemen eingenommen. Diese, so behaupten die Begründer der OMT [Rumbaugh (1993)], kommt den Anforderungen an Verständlichkeit, Ausdrucksstärke, Wartungsfreundlichkeit und Wiederverwendung realer Systeme wesentlich näher als die vorwiegend funktionalen Ansätze. Ähnlich wie bei anderen objektorientierten Methodologien stehen die Modelle im Mittelpunkt der Problemlösung.

3.5.1 Methodisches Vorgehen

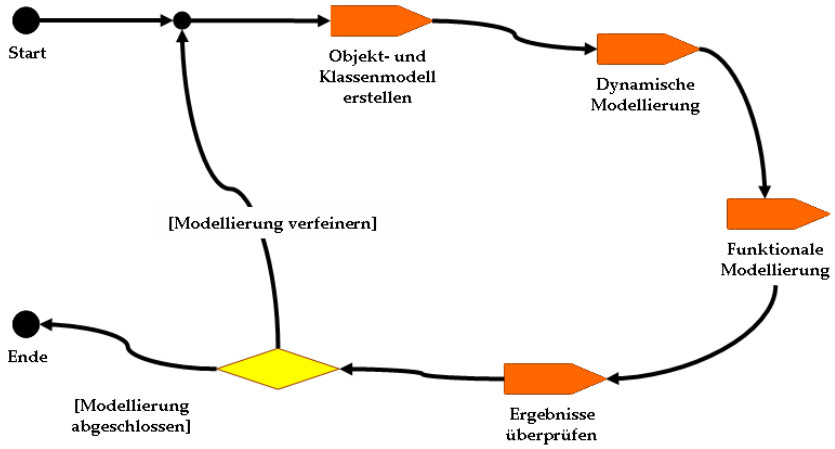


Abbildung 3.9: Übersicht über die „Object Modeling Technique"-Aktivitäten

In der Object Modeling Technique wird ein stark objektorientiertes Vorgehen zur Analyse und zum Entwurf eines Systems vorgeschlagen. Bereits zu Beginn wird dem Systemanalyst nahe gelegt, die Anforderungen und Wünsche in ein objektorientiertes Modell zu gießen. Es ist mir aufgefallen, dass diese Methode den Einsatz von Modellierungstechniken fokussiert und weniger auf die Tätigkeiten und Aufgaben des Systemanalysten eingeht. Dies ist in meinen Augen eine große Schwäche, die sich auch in anderen objektorientierten Methoden widerspiegelt. Der zentrale Aspekt einer Methode ist meiner Meinung nach die Beschreibung eines Vorgehens zur Erlangung eines Zieles und nicht der richtige Einsatz von Modellierungstechniken. Das hier vorgestellte methodische Vorgehen ist eine mögliche Variante der OMT, eine Anforderungsanalyse zu betreiben.

Objekt- und Klassenmodell erstellen
Zu Beginn der Aktivität soll eine Problembeschreibung erstellt oder beschafft werden. Aus dieser Beschreibung, sollen zum Beispiel durch das „Informal English Description"-Verfahren[1], Objekte und Klassen (die den statischen Aufbau des System veranschaulichen) abstrahiert werden. Anschließend sollen die Beziehungen, die Attribute und die Hierarchie (diese wird mittels Vererbung abgebildet) dem Klassen- und Objektmodell hinzugefügt werden. Das Ergebnis bildet die Grundlage für die nächste Aktivität.

Dynamische Modellierung
Mittels der dynamischen Modellierung wird das Verhalten des Systems modelliert. Dafür stellt die OMT zwei Diagrammtypen zur Verfügung: das *Zustandsdiagramm* und das *Ereignisflussdiagramm* (auch *Sequenzdiagramm* genannt) [Kendall (1998)].

Funktionale Modellierung
Die funktionale Modellierung wird mit Hilfe von Datenflussdiagrammen (siehe Structured Analysis) durchgeführt. Ziel ist es, die Funktionen und die Abhängigkeiten zwischen den

[1] Abbott führte zur Bestimmung von Klassen und Objekten das „Informal English Description"-Verfahren ein. Dabei werden alle Substantive und Verben einer Problembeschreibung unterstrichen und als potenzielle Kandidaten für Objekte und Klassen (die Substantive) und Methoden (die Verben) genauer untersucht.

Daten zu beschreiben. Die Konstruktion eines DFD kann aus der Structured Analysis entnommen werden.

Ergebnisse überprüfen
In der letzten Aktivität werden die Ergebnisse auf Vollständigkeit, Konsistenz und Eindeutigkeit geprüft. Stellt sich heraus, dass Funktionen noch nicht beschrieben wurden oder einzelne Szenarien zur Verhaltensbeschreibung des Systems fehlen, so werden die oben angeführten Aktivitäten ein weiteres Mal durchlaufen. Das zyklische Vorgehen der OMT zielt auf eine ständige Verbesserung und Verfeinerung der Modelle ab. Erst wenn die Ergebnisse den gewünschten Ansprüchen genügen, gilt die Analyse als abgeschlossen.

3.5.2 Gewonnene Erkenntnisse

Die OMT stellvertretend für eine Reihe von objektorientiert geführten Analysemethoden beschreibt meiner Meinung nach keine Anforderungsanalyse sondern lediglich Möglichkeiten der Modellierung, mit denen man eine Analyse betreiben kann. Wichtige Aspekte bleiben völlig unberücksichtigt, so mitunter die Ist-Analyse eines existierenden Systems, die Ursachenergründung von Problemen und die Anforderungserhebung. Es stehen hier Techniken und Notationen im Vordergrund aber die Aufgaben, Tätigkeiten und Personen werden vollkommen vernachlässigt. Ich habe keine neuen, zusätzlichen Erkenntnisse aus dieser Analysemethode gewinnen können. Auch aus der Booch-Methode, der OOSE, der JSP und der OOA, die ähnlich unergiebig blieben, konnte ich nichts für die Diplomarbeit relevantes herausziehen.

3.6 Rational Unified Process (RUP)

Der Rational Unified Process ist ein konkretes Vorgehensmodell des Unified Process (UP). Er wurde von der Firma Rational entwickelt und veröffentlicht. Der UP ist ein Metamodell von Methoden und Techniken zur Softwareentwicklung, der von den drei aus der objektorientierten Programmierung bekannten Personen Grady Booch, Ivar Jacobson und James Rumbaugh [Jacobson Ivar (2001)] entwickelt wurde. Sowohl der UP als auch der RUP verwenden als Notationssprache die UML. Sie wurde parallel zur Entwicklung des UPs ebenfalls von den Begründern des UP entworfen und ist eine Notationssprache, mit deren Hilfe Vorzugsweise objektorientierte Systeme modelliert werden können.

Der RUP ist in neun Disziplinen untergliedert:

Kerndisziplinen:

- Geschäftsprozessmodellierung (engl. Business Modeling)
- Anforderungsanalyse (engl. Requirement Engineering)
- Analyse und Design (engl. Analysis and Design)
- Implementierung (engl. Implementation)
- Test
- Auslieferung (engl. Deployment)

Unterstützende Diziplinen:

- Konfigurations- und Änderungsmanagement (engl. Configuration and Change Management)
- Projektmanagement (engl. Project Management)
- Infrastruktur (engl. Environment)

Orthogonal zu diesen Disziplinen verlaufen vier Phasen des RUPs, in welcher jede Disziplin mehr oder weniger intensiv durchgeführt wird. Diese Phasen sind:

- Konzeptionsphase (engl. Inception)
- Entwurfsphase (engl. Elaboration)
- Konstruktionsphase (engl. Construction)
- Übergangsphase (engl. Transition)

Diese Phasen basieren auf Iterationen. Somit ist der RUP ein Prozessmodell.

In meiner Untersuchung des RUP habe ich die Schwerpunkte auf die beiden Disziplinen *Geschäftsprozessmodellierung* und *Anforderungsanalyse* gelegt. Beide Disziplinen sind sehr detailliert beschrieben und bieten eine Fülle an Methoden und Techniken an, mit deren Hilfe Anforderungen erhoben, verwaltet, weiterentwickelt und verbessert werden können. Diese hier alle vorzustellen ist nicht Ziel dieser Diplomarbeit und wäre auch nicht sehr ergiebig. Daher verweise ich hier auf eine andere Ausarbeitung [Kaschowitz (2006c)], in der eine ausführliche Beschreibung des RUP zu finden ist. Ich stelle in dieser Diplomarbeit kurz die

beiden Workflows[2], der beiden Disziplinen *Geschäftsprozessmodellierung* und *Anforderungsanalyse* vor und erläutere dann, welche Erkenntnisse ich aus ihnen gewinnen konnte.

3.6.1 Methodisches Vorgehen

Geschäftsprozessmodellierung

Das Ziel der Geschäftsprozessmodellierung (Business Modeling) ist es, dass Unternehmen besser zu verstehen und den Kontakt zwischen Auftraggeber und Auftragnehmer herzustellen. Das Unternehmen kennen und verstehen lernen, heißt, dass der Anwendungsspezialist die Prozesse und Arbeitsabläufe, die Organisation und den Aufbau eines Unternehmens ergründen muss. Des Weiteren muss er aktuelle Probleme im Unternehmen und mögliche Verbesserungen in den Geschäftsprozessen aufdecken. Der entscheidende Beitrag der Geschäftsmodellierung zur Softwareentwicklung ist, dass sowohl der Endbenutzer und Kunde als auf der Gegenseite der Entwickler und Architekt beide ein gemeinsames, möglichst identisches Verständnis der Problematik haben. Die Abbildung 3.10 zeigt wie der RUP die Geschäftsprozessmodellierung umsetzt.

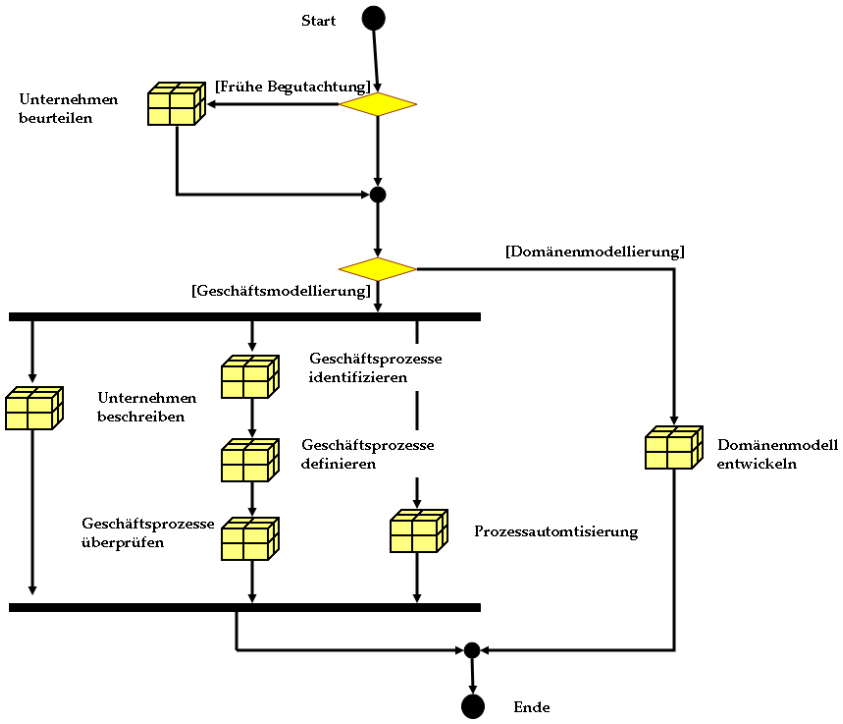


Abbildung 3.10: Geschäftsprozessmodellierung nach dem RUP [Cooperation (2003)]

[2]Ein Workflow beschreibt einen Arbeitsablauf, den es gilt erfolgreich abzuschließen.

Unternehmen beurteilen
In den ersten frühen Arbeitsschritten soll der aktuelle Zustand des Unternehmens untersucht und beurteilt werden. Zu begutachten sind die Geschäftsprozesse, Kunden, verwendete Werkzeuge, aber auch Einstellungen von Mitarbeitern, technologische Trends, verbesserungswürdige Geschäftsbereiche und mitunter auch Konkurrenten des Unternehmens.

Unternehmen beschreiben
Das Ziel in diesem Arbeitsschritt ist es, den momentanen Zustand des Unternehmens zu beschreiben und zu dokumentieren. So können Erkenntnisse aus dem früheren Teilprozess „Unternehmen beurteilen“ zweckdienlich für diese Aufgabe sein.

Geschäftsprozesse identifizieren
Bevor die Geschäftsprozesse eingehender untersucht werden können, müssen sie erst identifiziert werden. Dies ist Aufgabe dieses Teilprozesses.

Geschäftsprozesse definieren
Die Geschäftsprozesse (die in dem vorherigen Teilprozess identifiziert wurden) sollen nun ausführlich beschrieben werden. Außerdem soll ermittelt werden, welche Unternehmensziele ein Geschäftsprozess abdeckt.

Geschäftsprozesse überprüfen
Aufgabe dieses Teilprozesses ist es, die erarbeiteten Ergebnisse über die einzelnen Geschäftsprozesse zu prüfen und entsprechende Fehler zu korrigieren.

Prozessautomatisierung
Aufgabe dieses Teilprozesses ist es, zu ermitteln, welche Geschäftsprozesse automatisiert werden können und welche es unbedingt werden sollten.

Domänenmodell entwickeln
Dieser Teilprozess stellt eine Alternative zu der ausführlichen Beschreibung des Unternehmens dar. Aufgabe ist es ein Domänenmodell für das zu untersuchende Unternehmen zu entwickeln. In diesem Modell sollen die signifikanten Geschäftsprozesse, Konzepte, Leistungen und Produkte des Unternehmens beschrieben werden. Im Gegensatz zur Geschäftsmodellierung stellt es nur einen groben Überblick über das Unternehmen dar.

Anforderungsanalyse

In der Anforderungsanalyse die der RUP vorschreibt, steht der Endbenutzer und der Kunde im Fokus der Betrachtung. Die Wünsche und Forderungen die sie stellen, müssen zuerst bestimmt und anschließend dokumentiert werden. Dadurch wird das neue System hohe Chancen haben den Erwartungen des Kunden zu genügen. Außerdem müssen Änderungen der Kundenwünsche rechtzeitig erkannt und beschrieben werden. Der Nutzen einer solchen Vorgehensweise (siehe Abbildung 3.11) ist laut [Jacobson Ivar (2001)] der, dass:

- die Kosten reduziert werden, da Änderungen schon so früh als möglich berücksichtigt werden und
- das der Kunde zufrieden gestellt wird. Dadurch, dass versucht wird „korrekte“ Anforderungen zu sammeln, kann auch nur ein den Kunden zufrieden stellendes Produkt entstehen.

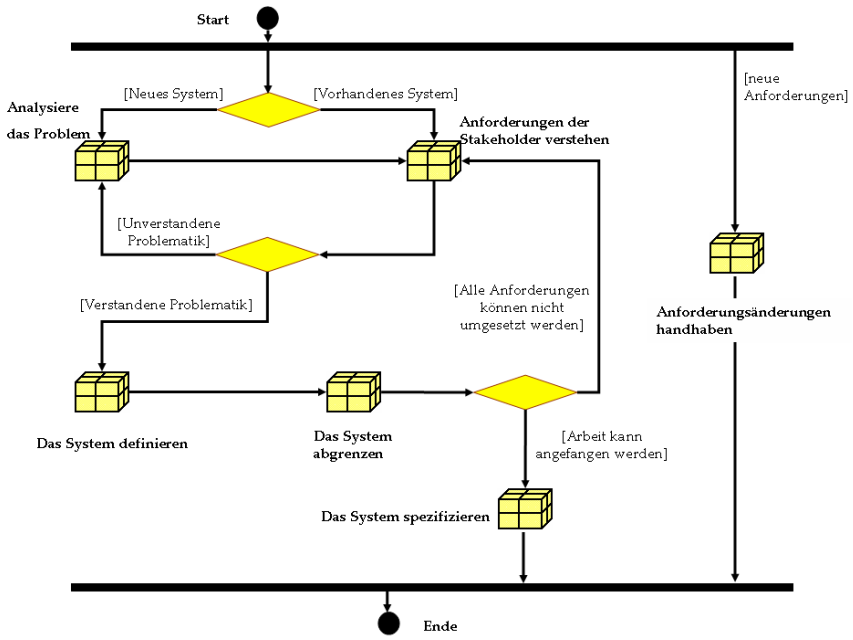


Abbildung 3.11: Anforderungsanalyse nach dem RUP [Cooperation (2003)]

Analysiere das Problem
Ziel dieses Teilprozesses ist es ein Einverständnis zwischen Kunden und Auftragnehmer über das zu lösende Problem zu erlangen. Außerdem sollen Messgrößen ermittelt werden, die die Auswirkungen des Problems für das Unternehmen unterstreichen sollen.

Anforderungen der Stakeholder[2] verstehen
Die Aufgabe dieses Teilprozesses ist es die Bedürfnisse und Wünsche der Stakeholder herauszufinden.

Das System definieren
Das System definieren bedeutet, dass die Wünsche und Forderungen an das neue System beschrieben werden sollen. Der RUP sieht hier die Verwendung von Use-Cases vor, die einen guten abstrakten Überblick geben können.

Das System abgrenzen
Das System abgrenzen bedeutet, dass festgelegt wird welche Funktionen vom System umgesetzt und an den Kunden ausgeliefert werden. Dazu werden die einzelnen gewünschten Anforderungen des zukünftigen Systems nach „Bedeutung“ gewichtet und geordnet. Diese Gewichtung ist ein Kriterium für die Umsetzung von Anforderungen.

Das System spezifizieren
Eine Spezifikation dient als ein Vertrag zwischen dem Kunden und dem Anbieter und ist die Grundlage für kommende Aktivitäten wie der Entwurf oder der Test. Sie entsteht als ein Resultat dieses Teilprozesses. Dazu werden die nicht-funktionalen Anforderungen und

[2]Der Stakeholder ist eine Rolle, die verschiedene Interessengruppen repräsentiert. So kann ein Stekeholder ein Benutzer des neuen Systems, ein Auftraggeber, ein Kunde oder gar ein Abnahmetester sein.

funktionalen Anforderungen (die als Use-Cases beschrieben werden) des zukünftigen Systems ausgearbeitet und in Zusammenarbeit mit den Stakeholdern festgelegt.

Anforderungsänderungen handhaben
Das Handhaben von Änderungen an Anforderungen beschreibt, wie damit umgegangen wird, wenn sich eine Anforderung nach Projektbeginn ändert.

3.6.2 Gewonnene Erkenntnisse

Der Rational Unified Process vereint viele Methoden, Konzepte, Techniken und Hilfestellungen für eine Reihe von Abschnitten der Softwareentwicklung. Bei der Analyse des RUP wurde ich zunächst von der gewaltigen Menge an Information „überwältigt“. Es schien mir, als seien dort „alle“ Methoden und Konzepte die es zur Softwareentwicklung gibt, aufgeführt. Ich untersuchte die oben vorgestellten Disziplinen der *Geschäftsprozessmodellierung* und der *Anforderungsanalyse*, ob ich dort neue Ideen und Erkenntnisse für das Metamodell der Analysemethoden finden würde. Drei neue Elemente, die für die Darstellung einer Methode förderlich sind, führe ich jetzt hier auf.

Hilfestellung: Der RUP stellt für viele Aktivitäten eine Reihe von Hilfestellungen zur Verfügung. So können Artikel und Aufsätze zu bestimmten Themen, aber auch Checklisten, Leitlinien, Beispiele oder Vorlagen, wie die zu einer Spezifikation, gelesen werden. Diese zusätzlichen Informationen zu einer Aktivität helfen dem Analysten dabei, seine Tätigkeit korrekt durchzuführen. Daher wird dieses Element auch in das Metamodell aufgenommen.

Vorbedingung und Ziel einer Aktivität: Eine Aktivität ist im RUP immer mit einem Resultat verbunden. Mal müssen Anforderungen prioritisiert werden, ein anderes Mal müssen Stakeholder bestimmt werden. Alle Aktivitäten verfolgen demnach ein Ziel. Aber nicht nur das. Einige Aktivitäten besitzen auch Vorbedingungen die nicht aus der Abfolge eines Prozesses herrührt. So kann eine Aktivität die dafür zuständig ist Use-Cases ausführlich zu beschreiben, z.B. erst dann ausgeführt werden, wenn der entsprechende Use-Case priorisiert wurde. Alle diese Betrachtungen haben mich dazu bewogen, ein Element hinzuzufügen, welches es erlaubt Aktivitäten mit Bedingungen zu versehen.

Rolle: Der RUP sieht nicht nur eine Rolle zur Anforderungsanalyse vor, sondern unterschiedlich viele, je nach Art und Umfang der eingesetzten Methode. So gibt es eine Rolle die dafür zuständig ist die Anforderungen beim Kunden zu erheben, eine andere Rolle hat die Funktion die Spezifikation zu schreiben. So gibt es verschiedene Rollen, die für bestimmte Aufgaben spezialisiert sind. Auch diese Eigenschaft ist für mich bei dem Entwurf des Metamodells der Analysemethoden von Bedeutung.

4 Metamodell der Analysemethoden

4.1 Vorgehensweise zur Erstellung

Ich bin nun so vorgegangen, dass ich versucht habe die Fragen „Was macht ein Metamodell der Analysemethoden eigentlich aus?“ und „Mit welchen Mitteln lässt sich eine Analyse gut beschreiben?“ zu beantworten. Dafür habe ich die in Kapitel 4 vorgestellten Methodologien der Anforderungsanalyse untersucht und die für mich essentiellen Elemente, die eine Analysemethode beschreiben können, herausgefunden. Dieses waren zum einen Elemente die die Methode „direkt“ beschreiben, wie z.B. das Artefakt oder die Aktivität und auf der anderen Seite, Elemente, die zur Verknüpfung und Regulierung dieser „direkten“ Elemente dienen, wie z.B. der Synchronisationsbalken, oder der Kontrollfluss im Allgemeinen. Nachdem ich nun alle Elemente, die für mich wichtig schienen, erfasst hatte, definierte ich für jedes einzelne Element die entsprechenden charakteristischen Eigenschaften (z.B. beinhaltet eine Aktivität immer eine Vorgehensweise, die beschreibt, welche Aufgaben, wie durchzuführen sind.). Anschließend untersuchte ich das SPEM, um Ideen für die Umsetzung der Elemente in ein Metamodell zu bekommen. Außerdem habe ich das Domänenmodell von KoPra berücksichtigt, da ich ja zu einem späteren Zeitpunkt der Diplomarbeit, fehlende Elemente dort nachträglich ergänzen musste. So entwickelte ich Schritt für Schritt das Metamodell, wobei ich darauf Wert legte, dass es einfach, leicht verständlich und universell einsetzbar ist. Daher enthält es möglichst wenige Restriktionen. Im nachfolgenden Kapitel stelle ich alle Elemente des Metamodells und deren Eigenschaften vor, danach das eigentliche Metamodell als UML-Klassen-Diagramm und abschließend erläutere ich die Besonderheiten und Feinheiten des Modells.

4.2 Berücksichtigte Elemente

Nach meiner Untersuchung der Analysemethoden habe ich eine Reihe von Elementen identifiziert, die zur Modellierung von Methoden hilfreich sein können. Diese Elemente lassen sich grob in zwei Kategorien einteilen. Die Elemente der einen Kategorie, ich nenne sie die „direkten“ Elemente, legen fest, wer, was, wie konkret durchzuführen hat. Das heißt, diese Elemente beschreiben die Aufgaben, Resultate und Personen die zu einer Methode gehören. Die Elemente, welche zu der anderen Kategorie zählen, ich nenne sie die „regulierenden“ Elemente, bestimmen den zeitlichen Verlauf und den Ablauf einer solchen Methode. Außerdem kann durch diese Elemente der Verlauf einer Methode gesteuert und verändert werden. Daneben gibt es noch Eigenschaften die für beide Gruppen gleichermaßen gelten.

Eigenschaften, die für alle Elemente gelten:

Eigenschaft	Bedeutung
Beschreibung	Mit Hilfe der Beschreibung lässt sich jedes Element näher beschreiben.
Beziehung	Jedes Element kann mit einem anderen Element verknüpft werden. Beziehungen zwischen zwei Elementen sind fundamental für das Metamodell. So können z.B. Aktivitäten untereinander verknüpft werden oder ein Artefakt und eine Aktivität.
Namen	Alle Elemente besitzen einen Namen.

Tabelle 4.1: Allgemeingültige Eigenschaften aller Elemente

„Direkte Elemente“:

Element	Bedeutung
Aktivität	Eine Aktivität beschreibt eine durchzuführende Tätigkeit oder eine zu lösende Aufgabe.
Artefakt	Ein Artefakt ist ein Dokument.
Bedingung	Eine Bedingung gibt an ob eine Aktivität oder ein Teilprozess ausgeführt oder abgeschlossen werden können.
Hilfestellung	Eine Hilfestellung kann ein Hinweis oder ein Ratschlag zum lösen einer Aufgabe, aber auch eine Hintergrundinformationen über ein Element sein. So kann es eine Checkliste, eine Vorlage oder auch nur einen themenrelevanten Artikel darstellen.
Rolle	Eine Rolle stellt eine Personengruppe dar, die zur Durchführung gewisser Aufgaben spezialisiert ist.
Teilprozess	Ein Teilprozess ist ein Behälter, der mehrere Aktivitäten, die ein und dasselbe Ziel verfolgen, beinhaltet.

Tabelle 4.2: Elemente des Metamodells, die ein Analyseverfahren beschreiben

„Direkte Elemente“ und ihre Eigenschaften:

Element	Attribut	Bedeutung
Aktivität	Aufgabe / Tätigkeit	Dieses Attribut beschreibt die Aufgabe der Aktivität die gelöst werden soll. Es kann aber auch einfach die Tätigkeit beschreiben, die eine Person durchzuführen hat.
	Empfohlene Vorgehen	Dieses Attribut beschreibt das methodische Vorgehen um die Aufgabe oder Tätigkeit zu erfüllen.
	Vorbedingung	Dieses Attribut legt die Vorbedingung einer Aktivität fest.
	Ziel	Dieses Attribut legt das Ziel, was durch eine Aktivität erreicht werden soll, fest.
Artefakt	Werkzeug	Dieses Attribut beschreibt ein Werkzeug, mit dessen Hilfe das Dokument erstellt werden kann.
	Wird produziert?	Dieses Attribut legt fest, ob ein Artefakt von einem anderen Element (vorzugsweise einer Aktivität) konsumiert oder produziert wird.
Hilfestellung	Art	Dieses Attribut legt die Art der Hilfestellung fest. Eine Hilfestellung kann eine Checkliste, Vorlage, Referenz oder ein Dokument sein.

Tabelle 4.3: Eigenschaften der „direkten“ Elemente

„Regulierende“ Elemente:

Element	Bedeutung
Bedingte Verzweigung (Synonym: Spaltung)	Durch eine bedingte Verzweigung wird der Kontrollfluss in mehrere, an eine Bedingung gebundene, ausgehende Pfade gespalten. Es wird der ausgehende Pfad weiter verfolgt, dessen Bedingung zutrifft.
Endknoten	Der Endknoten repräsentiert das Ende einer Methode oder eines Prozesses.
Startknoten	Der Startknoten repräsentiert den Beginn einer Methode oder eines Prozesses.
Synchronisationsbalken	Mit Hilfe des Synchronisationsbalkens lassen sich parallel ablaufende Pfade synchronisieren. Erst wenn alle eingehenden Pfade verfolgt wurden, werden die ausgehenden Pfade des Synchronisationsbalkens weiterverfolgt. Er stellt somit eine UND-Verknüpfung der eingehenden Pfade dar.
Zusammenführung von Pfaden (Synonym: Verschmelzung)	Eine Alternative zum Synchronisationsbalken ist die Verschmelzung, mit deren Hilfe sich ebenfalls mehrere eingehende Pfade zusammenführen lassen. Im Gegensatz zur Synchronisation, reicht es bei diesem Element aber aus, wenn ein eingehender Pfad verfolgt wurde, um alle ausgehenden Pfade zu verfolgen. Er entspricht somit einer ODER-Verknüpfung der eingehenden Pfade.

Tabelle 4.4: Elemente, die den Kontrollfluss beeinflussen

Die Elemente, die den Kontrollfluss beeinflussen können, besitzen neben den universell gültigen Eigenschaften (Name, Beschreibung, Verknüpfungsfähigkeit) keine weiteren.

4.3 Das Metamodell

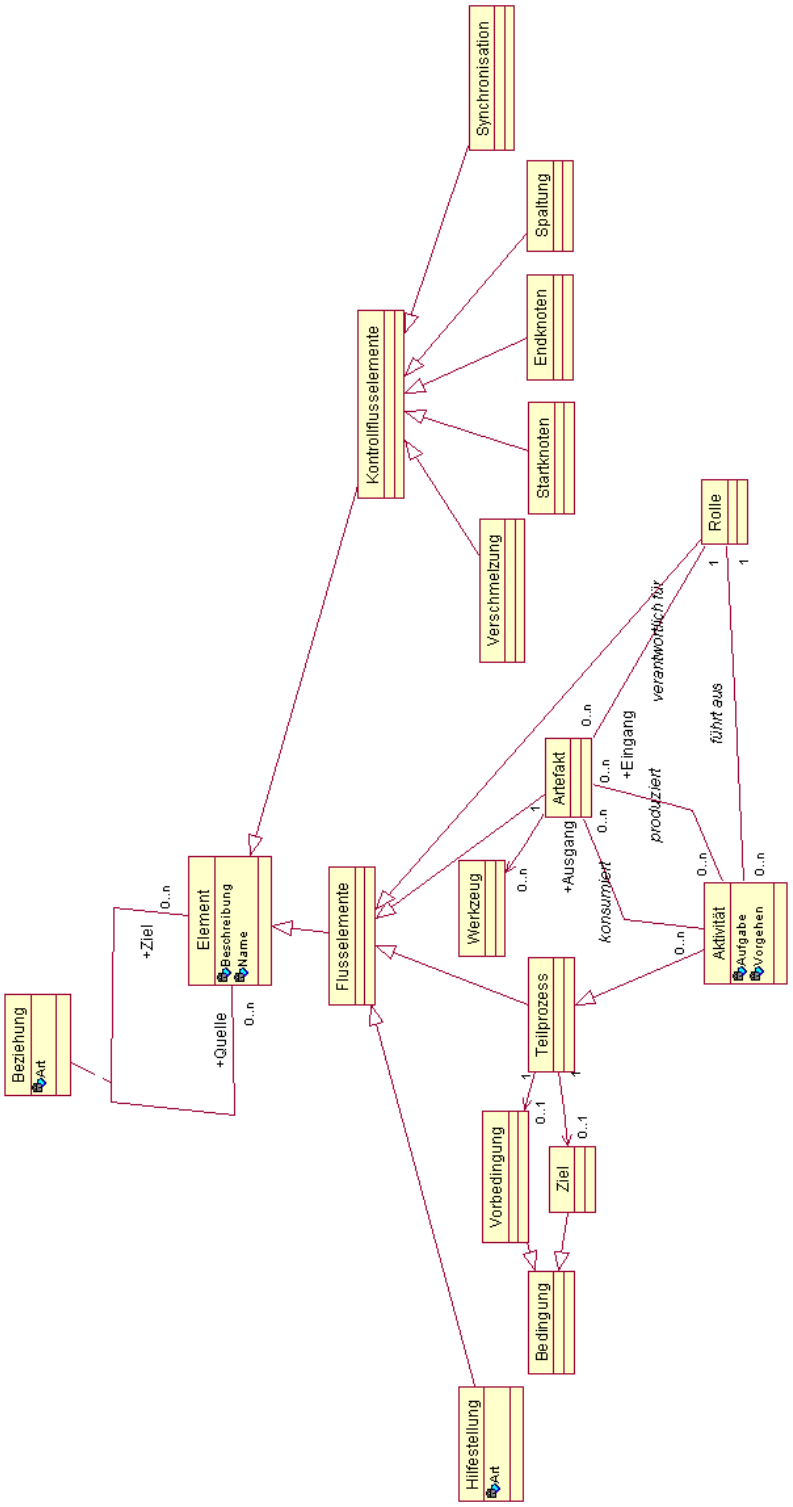


Abbildung 4.1: Metamodell der Analysemethoden

4.4 Erläuterungen zum Metamodell

Die Abbildung 4.1 zeigt das Metamodell der Analysemethoden. Mit diesem Modell lassen sich Analyseverfahren abbilden. Die zentrale Klasse ist das Element, welche alle an einem Analyseverfahren beteiligten Elemente repräsentiert. Alle verfügbaren Elemente, wie z.B. das Artefakt, die Rolle oder der Synchronisationsbalken, erben die allgemein gültigen Eigenschaften dieses Elements, die *Beschreibung*, den *Namen* und die Fähigkeit mit anderen Elementen eine *Beziehung* einzugehen. Dabei besteht die Beziehung aus zwei Elementen, von denen das eine als Quelle und das andere als Ziel ausgewiesen ist. Ich habe hier bewusst auf Restriktionen verzichtet (außer zwischen den drei Elementen Rolle, Artefakt und Aktivität). Sollen Beziehungen zwischen Elementen ausgeschlossen werden, so muss dieses später in der Implementierung geschehen. Die „regulierenden“ Elemente, die den Ablauf der Methode beeinflussen, werden unter der Klasse Kontrollflusselemente zusammengefasst. Die „direkten“ Elemente, die bestimmen, wer was wie während der Methode zu tun hat, werden unter der Klasse Flusselemente zusammengefasst. Ein Teilprozess kann über eine Vorbedingung und ein Ziel verfügen. Beides sind Bedingungen, die ausdrücken, ob ein Teilprozess überhaupt ausgeführt (bei einer Vorbedingung) oder abgeschlossen (bei einem Ziel) werden kann. Da eine Aktivität die Eigenschaften des Teilprozesses erbt, kann sie ebenso mit Bedingungen versehen werden. Wird eine Aktivität „Spezifikation erstellen“ ausgeführt, so werden verschiedene Dokumente als Grundlage benötigt, wie ein Domänenmodell oder Use-Case-Diagramme. Außerdem können unterschiedliche Dokumente neben der Spezifikation entstehen, wie z.B. eine Offene-Punkte-Liste oder Änderungswünsche des Kunden. Daher ist es nötig, dass eine Aktivität sowohl mehrere Artefakte konsumieren als auch produzieren kann. Eine Rolle soll in der Lage sein unterschiedliche Aktivitäten durchzuführen. Dadurch, dass eine Rolle verschiedene Aktivitäten durchführen kann, ändert sie zwangsweise eine Reihe aktivitätsgebundener Artefakte. Für die Erstellung und Verwaltung solcher Artefakte sind sie dann mitverantwortlich.

Ich habe bewusst ein möglichst restriktionsfreies Modell entworfen, da gewünschte Einschränkungen immer noch später in einer konkreten Umsetzung eingeführt werden können. In der Umsetzung der Erweiterung von KoPra ist eine Lösung entstanden, die auf diesem Metamodell beruht. Dabei habe ich einige Restriktionen gerade bei den Start- und Endknoten während der Implementierung umgesetzt.

5 Methodik-Unterstützung für JUST

Auf Grundlage der Erkenntnisse aus den untersuchten Analysemethoden entstand eine Spezifikation [Kaschowitz (2006a)] einer zukünftigen Erweiterung von JUST. In ihr wird eine Methodik-Unterstützung zur Anforderungserhebung spezifiziert. In diesem Kapitel werden die Ergebnisse der Spezifikation präsentiert und das Zusammenspiel zwischen Prozess und Wizard untersucht.

5.1 Vorüberlegungen

Problematik:

Ein Anwendungsspezialist möchte Anforderungen erheben und diese in einem geeigneten Werkzeug verwalten. Er stellt sich die Fragen:

- Wie gehe ich vor?,
- Womit fange ich eigentlich an?,
- Gibt es einen Prozess, an den ich mich halten kann? und
- Gibt es Dinge, die ich unbedingt beachten muss, die ich nicht vergessen darf?.

Die Konsequenz ist, dass er Hilfe braucht. Er benötigt Hilfe und geeignete Ratschläge, wie er vorzugehen hat, um Anforderungen zu erheben. Nun ist jedes Projekt anders, das eine groß und komplex z.B. für ein Flugzeugunternehmen, das andere eine Software für ein eingebettetes System z.B. für Handys. Die Richtlinien, der Prozess sieht je nach Projekt und Aufgabe, je nach den Unternehmen, in denen entwickelt wird, anders aus.

Das Ziel:

Das Ziel ist es nun eine Lösung zu schaffen, die beides miteinander vereint:

- einerseits die einfache, unkomplizierte Hilfe für den Anwendungsspezialist und
- andererseits die Austauschbarkeit von methodischen Prozessen.

Unter Verwendung der beiden Werkzeuge JUST [2.2] und KoPra [2.3] kann dieses Ziel erreicht werden.

Die Lösung:

„Nun wie sieht die Lösung aus?“ Eine Teillösung dieses Problems sieht folgendermaßen aus:

Die beiden Werkzeuge JUST und KoPra müssen enger zusammenarbeiten, die Prozesse, die in KoPra erstellt und definiert werden, sollen von JUST eingelesen werden können und den Anwendungsspezialist bei seinem Vorgehen lenken und leiten. Dazu werden beide um eine XML-Schnittstelle erweitert, mit deren Hilfe aus KoPra Prozesse exportiert und in JUST eingelesen werden können. In JUST werden die Prozesse gesammelt, in einer Liste graphisch aufbereitet und später kann dann ein beliebiger Prozess jeweils einem Softwareprojekt zugeordnet werden. Dieser kann vom Projektleiter oder Administrator in JUST aktiviert oder deaktiviert werden. Somit wäre die Integration der Prozesse von KoPra in JUST umgesetzt.

„Ja gut, aber wie erreicht man, dass der Anwendungsspezialist sich an die Vorgaben des Prozesses hält und wie erhält er die nötigen Hilfestellungen?“

Nun das ist schwieriger umzusetzen. Die folgende Idee ist mir zur Lösung dieses Problems gekommen:

Der gesamte Prozess, der das methodische Vorgehen beschreibt, wird in drei unterschiedlichen Darstellungsformen angezeigt. Diese sollen den Anwender zu einer methodischen, prozessgestützten Anforderungsanalyse “anleiten“. Dies sind:

- *Die Visualisierung des Prozesses:* In einem neuen JUST-Reiter soll der gesamte Prozess visualisiert werden. Dadurch soll sich der Anwender einen Überblick über den zugrunde liegenden Prozess machen können. Ferner soll er auch in der Lage sein, durch den Prozess zu navigieren, um gezielt einzelne Aktivitäten auszuführen.
- *Die TODO-Liste:* Diese Liste stellt alle, den Anwender betreffenden, Aufgaben dar. Dadurch wird zum einen sichergestellt, dass nur Aktivitäten von den dafür vorgesehenen Personen durchgeführt werden können, zum anderen erhält der Anwender stets einen Überblick über seine offenen Aufgaben. Über die TODO-Liste soll die eigentliche Arbeit durchgeführt werden. Sie wird automatisch auf Basis des zugrunde liegenden Prozesses aktualisiert. Dessen Aufbau und Ablauf bestimmen, welche Aktivitäten wann in die TODO-Liste aufgenommen werden.
- *Der Wizard:* Der Wizard ersetzt die herkömmliche Anforderungserfassung von JUST, (die den Fokus auf einzelne Anforderungen legte) durch eine auf Aktivitäten beruhende Erfassung von ganzen Artefakten. Durch ihn soll gewährleistet werden, dass der Benutzer, in unserem Fall der Anwendungsspezialist, immer weiß, was er zu tun und was er nicht zu tun hat.

5.2 Überblick über die Erweiterung

In diesem Abschnitt wird ein Überblick über die Erweiterung von JUST gegeben. Tiefer gehend werden die zentralen Funktionen und Konzepte wie:

- der Prozessimport (Import von KoPra-Prozessen in JUST),
- die Prozessvisualisierung (graphische Visualisierung der importierten Prozesse),
- die Prozesselemente und ihre Wechselwirkung mit JUST,
- der Wizard (dessen Darstellung und Funktionsweise) und
- die TODO-Liste („Offene-Aufgaben-Liste“)

betrachtet.

5.2.1 Prozessimport

Prozesse, die mit KoPra modelliert und in eine XML-Datei exportiert wurden, sollen später in JUST importiert werden können. Jedes Projekt in JUST soll einem Prozess zugeordnet werden können, was zur Folge hat, dass in diesem Projekt die Anforderungserhebung mit Hilfe eines Wizards durchgeführt wird.
Nur die Administratoren sind dazu in der Lage, nur sie können festlegen:

- welche Projekte welchen Prozessen folgen müssen,
- welche Benutzer welchen „Prozessrollen“ entsprechen.

Die Zuordnung von Artefakten (Dokumente, die eine Aktivität konsumiert oder produziert) zu WorkProductions[1] werden in KoPra durchgeführt und sind nicht Teil der JUST-Erweiterung.

5.2.2 Prozessvisualisierung

Die Prozesse sollen in JUST in einem separaten Reiter entweder als statisches Bild oder als Applet angezeigt werden. Die zugehörigen Aktivitäten und Teilprozesse werden in einer Tabelle aufgelistet. Die Visualisierung sollte dann wie in Abbildung 5.1 aussehen.

[1] Container in JUST, der für das verwalten von Artefakten ausgelegt ist.

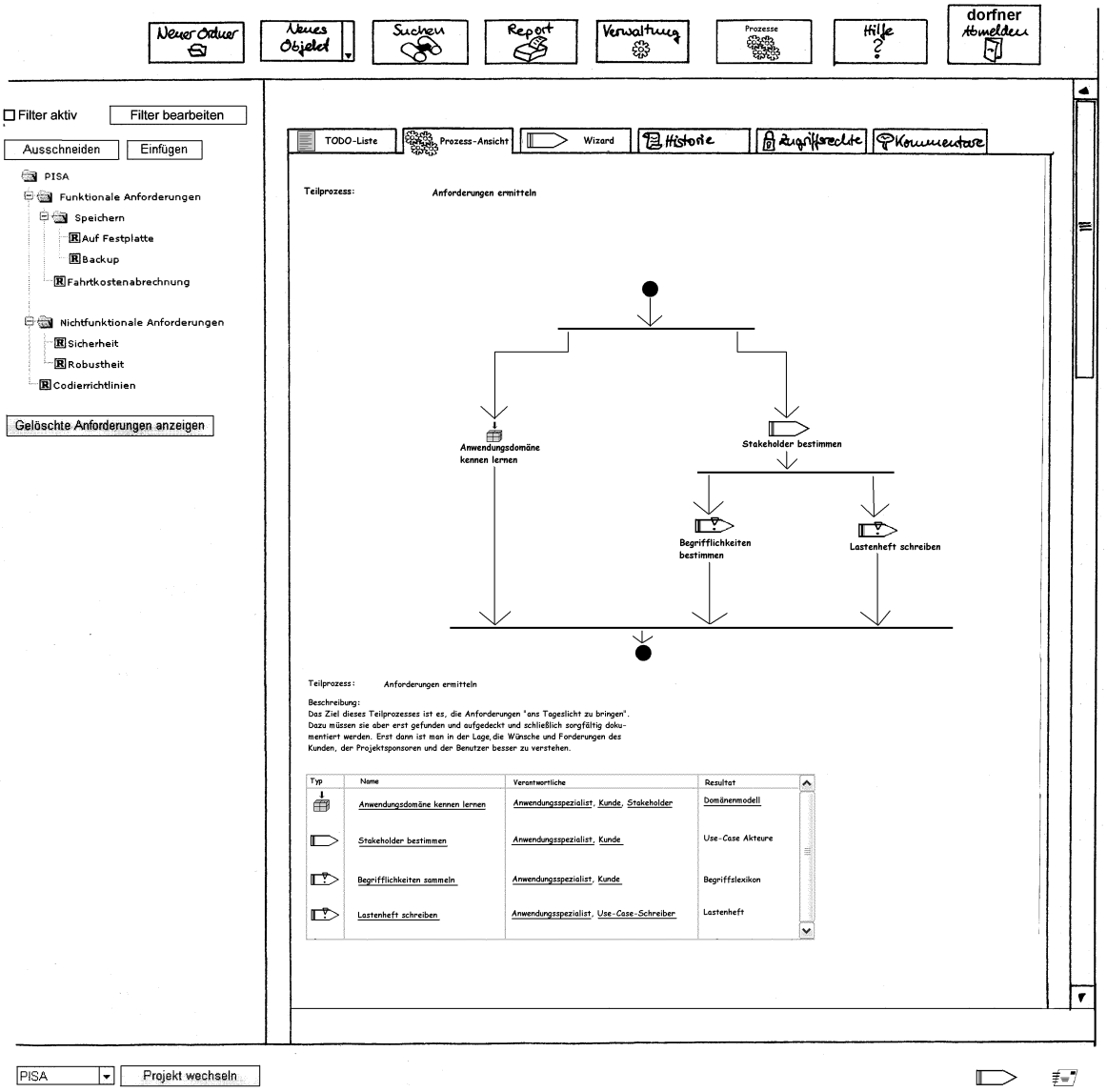


Abbildung 5.1: Prozessvisualisierung in JUST

Oben im Reiter „Prozess-Ansicht“ ist das Abbild des Teilprozesses „Anforderungen ermitteln“ zu sehen. Darunter befinden sich eine kurze Beschreibung des Teilprozesses und eine Tabelle, in der alle am Teilprozess „Anforderungen ermitteln“ beteiligten Aktivitäten und Teilprozesse aufgelistet sind.

Die Tabelle besteht aus den vier Spalten die in der Tabelle 5.1 beschrieben werden.

Klickt ein Anwender in der Tabelle auf einen Teilprozess, so wird dessen Prozess angezeigt und der Reiter entsprechend aktualisiert. Klickt er dagegen auf eine Aktivität, so wird der Wizard zu der gewählten Aktivität geöffnet und der Benutzer kann je nach Zugriffsrechten entweder neue Artefakte erstellen und bereits bestehende Artefakte bearbeiten oder aber nur lesend auf die Artefakte zugreifen.

Spaltenname	Bedeutung
Typ	Dieses Attribut bestimmt den Typ und den Zustand eines Prozesselements und kann eine der folgenden Werte einnehmen: = offene Aktivität = abgeschlossene Aktivität = noch anstehende Aktivität = Teilprozess
Name	Dieses Attribut gibt den Namen des Prozesselements an.
Verantwortliche	Dieses Attribut gibt die für eine Aktivität verantwortlichen Rollen an. Ist das Prozesselement ein Teilprozess, so bleibt diese Zelle leer.
Resultat	Dieses Attribut gibt die produzierten Artefakte einer Aktivität an. Handelt es sich um einen Teilprozess, so bleibt diese Zelle leer.

Tabelle 5.1: Erläuterung der Tabelle aus Abbildung 5.1

5.2.3 Regulierung des Kontrollflusses

In diesem Abschnitt wird die Wirkung der einzelnen Kontrollflusselemente (Spaltung, Verschmelzung und Synchronisation) auf den Kontrollfluss beschrieben. Darüber hinaus wird erklärt, welchen Einfluss der Anwender bei der Steuerung des Prozesses hat.

Der Kontrollfluss

Der Kontrollfluss wird durch gerichtete, schwarze Pfeile dargestellt, die die Elemente eines Prozesses miteinander verbinden. Ein Element kann dabei (siehe Metamodell 4, Anhang B) sowohl mehrere ausgehende Pfade als auch mehrere eingehende Pfade besitzen.

Die Spaltung

Die Spaltung mit zwei ausgehenden Pfaden

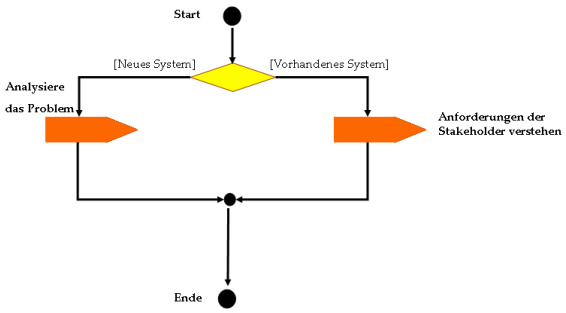


Abbildung 5.2: Spaltung mit zwei ausgehenden Pfaden

Spaltungen werden immer mit Rauten-Symbolen dargestellt. Eine Spaltung hat genau einen eingehenden Pfad (dies ist eine bewusste Restriktion) und kann einen, zwei oder mehr ausgehende Pfade besitzen. JUST behandelt eine Spaltung, die zwei ausgehende Pfade besitzt, einfach über eine Dialogbox-Abfrage an den Anwender. Die Frage und die Antwortmöglichkeiten sind an der Raute hinterlegt. Eine beispielhafte Frage, die nur zwei Antworten und somit zwei alternative Pfade zulässt, wäre:

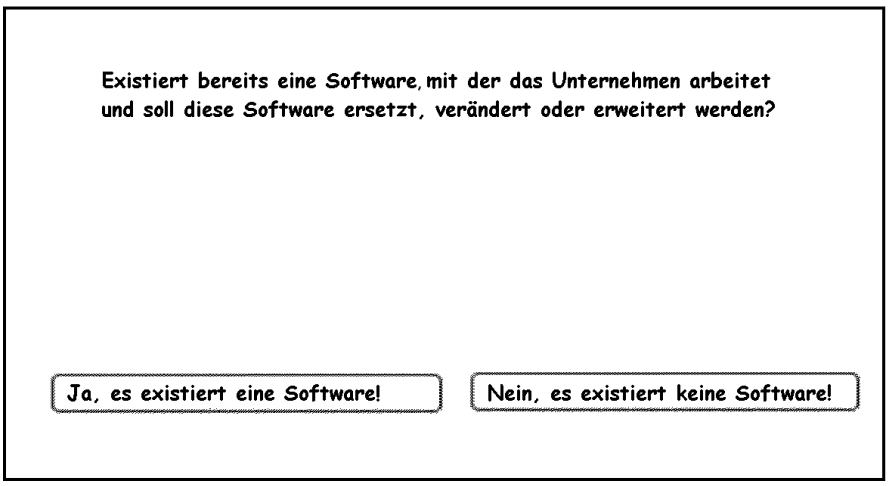


Abbildung 5.3: Abfrage bei einer Spaltung mit zwei Ausgängen

Nachdem der Anwender seine Wahl getroffen hat, wird der Kontrollfluss des ausgewählten Pfades weiterverfolgt. Der andere Pfad bleibt inaktiv und alle folgenden Elemente des inaktiven Pfades bleiben „unberührt“.

Die Spaltung mit einem ausgehenden Pfad

Neben der Bedingung mit zwei ausgehenden Pfaden können auch solche mit einem ausgehenden Pfad modelliert und in JUST umgesetzt werden. Die Abbildung 5.4 zeigt ein solches Beispiel, bei der die Spaltung nur eine Aktivität aktiviert, falls sie aktiviert wird. Wird sie nicht aktiviert, so verläuft der Pfad in eine Sackgasse und endet dort. Er hat dann keinerlei Auswirkungen mehr.

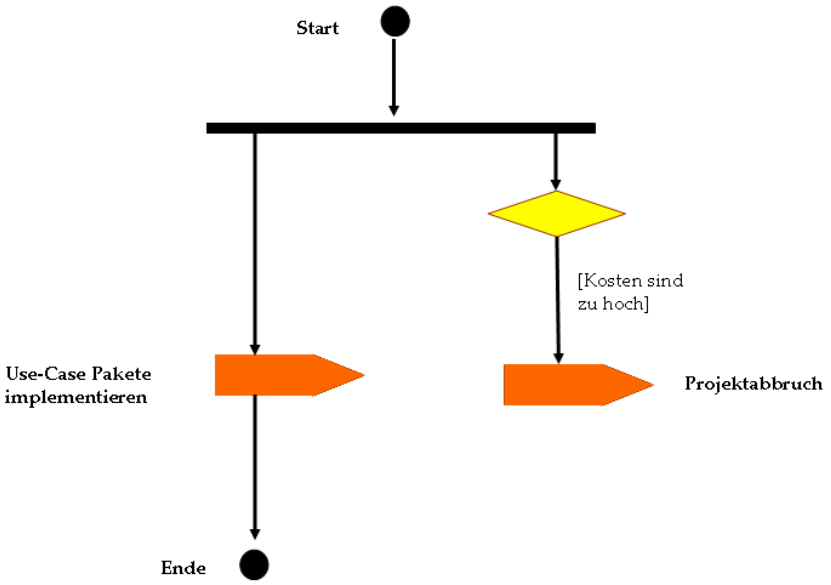


Abbildung 5.4: Spaltung mit einem ausgehenden Pfad

Auch bei der Spaltung mit nur einem ausgehenden Pfad wird der Anwender über ein Dialogfenster befragt, welches Vorgehen er beabsichtigt. Die Frage ist an der Raute hinterlegt (siehe Abbildung 5.5).

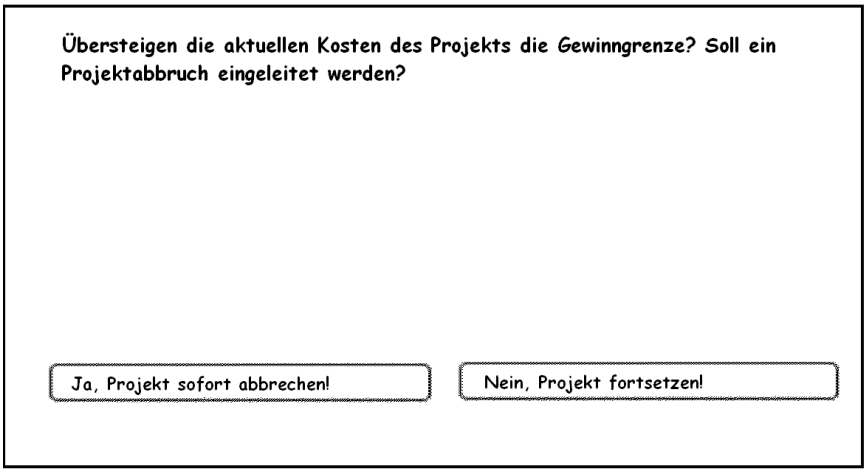


Abbildung 5.5: Abfrage bei einer Spaltung mit einem ausgehenden Pfad

Die Spaltung mit mehreren ausgehenden Pfaden (> 2 Pfade)

Abschließend gilt es noch das Verhalten von JUST zu beschreiben, sobald das System auf eine Spaltung mit mehr als zwei Ausgängen trifft. In diesem letzten Fall einer Spaltung wird ebenfalls ein Dialogfenster angezeigt, mit dessen Unterstützung der Anwender das weitere Vorgehen festlegt.

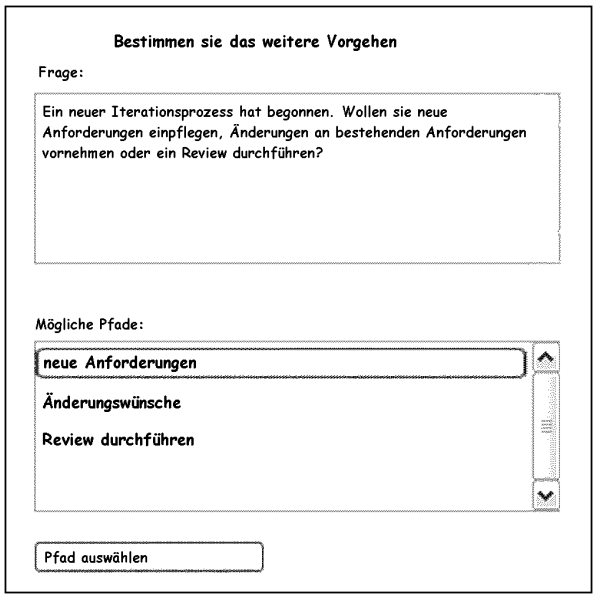


Abbildung 5.6: Abfrage bei einer Spaltung mit mehreren ausgehenden Pfaden

Er wählt aus einer Liste von möglichen Alternativen einen Pfad aus und aktiviert ihn. Die anderen Pfade bleiben inaktiv.

Die Spaltung als Sprung einsetzen

Eine weitere Ausprägungsart der Spaltung dient dazu Sprünge zu modellieren. Da die ausgehenden Pfade einer Spaltung mit beliebigen Aktivitäten und Kontrollflusselementen verknüpft werden können, ist es möglich auf im Kontrollfluss vorhergehende oder noch bevorstehende Aktivitäten oder aber Kontrollflusselemente zu verweisen. Somit ist es möglich, gezielt Aktivitäten, die schon bearbeitet wurden, noch einmal bearbeiten zu lassen (Zyklen schaffen) oder Aktivitäten im Kontrollfluss zu überspringen. Ein Sprung bedarf keiner Synchronisation und wird durch eine gelbe Raute dargestellt (siehe Abbildung 5.7). Der Benutzer wird ebenfalls über ein Dialogboxfenster befragt, wie er fortzufahren gedenkt.

Verschmelzung (Pfade zusammenführen)

Pfade können durch einen schwarzen Kreis zu einem Pfad verschmolzen werden (siehe Abbildung 5.7). Ein solches Prozesselement besitzt also beliebig viele eingehende Pfade und mindestens einen ausgehenden Pfad. Die ausgehenden Pfade werden genau dann verfolgt, wenn der Kontrollfluss durch einen der eingehenden Pfade auf dieses Prozesselement trifft. Somit beschreibt dieses Prozesselement eine Oder-Verknüpfung der eingehenden Pfade.

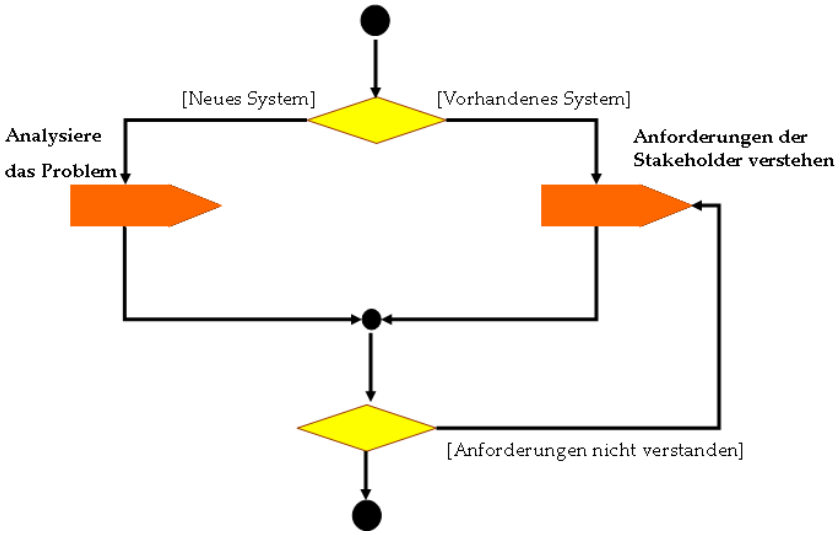


Abbildung 5.7: Der Sprung zu einer bereits abgeschlossenen Aktivität

Durch dieses Prozesselement können durch Spaltungen aufgespaltete Pfade zusammengeführt werden oder aber getrennte, parallel ablaufende Pfade zu einem Pfad verschmolzen werden.

Die Synchronisation

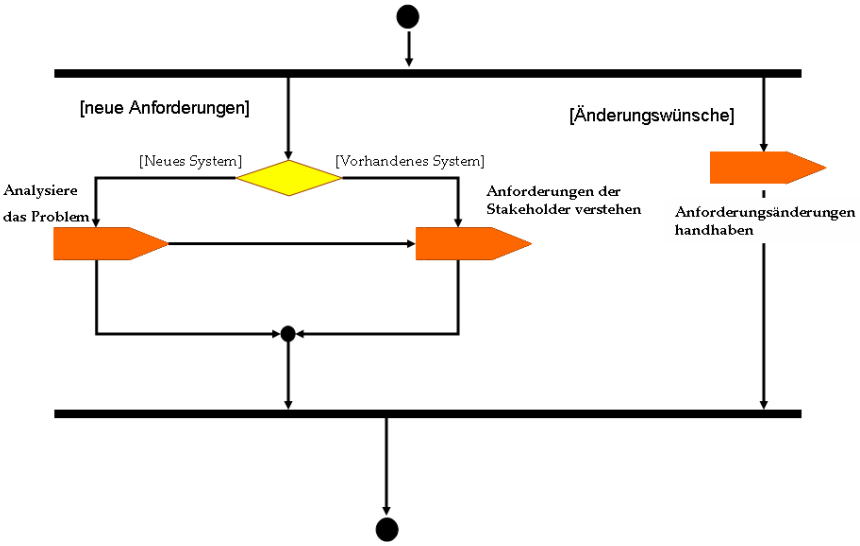


Abbildung 5.8: Synchronisation durch den Synchronisationsbalken

Die Synchronisation von Pfaden wird über einen Synchronisationsbalken verwirklicht. Der Synchronisationsbalken kann mehrere eingehende und ausgehende Pfade besitzen. Die ausgehenden Pfade eines Synchronisationsbalkens werden solange nicht weiterverfolgt, **bis alle** eingehenden Pfade *aktiv* (siehe „Was bedeutet es wenn eine Aktivität *aktiv*, *geschlossen* oder

offen ist?“) sind. Erst dann werden alle ausgehenden Pfade verfolgt und gleichzeitig aktiv geschaltet. Im Gegensatz zur Verschmelzung müssen bei der parallelen Aufspaltung alle eingehenden Pfade aktiv geschaltet sein. In dem Beispiel aus Abbildung 5.8 wären es zwei Aktivitäten die abgeschlossen sein müssten, um den Endknoten zu erreichen, die Aktivität „Anforderungsänderungen handhaben“ und je nach Antwort auf die Bedingung einer der beiden Aktivitäten “Analysiere das Problem“ oder „Anforderungen der Stakeholder verstehen“.

5.2.4 Der Wizard

Das Verhalten von JUST für die Kontrollflusselemente: Verschmelzung, Synchronisation und Spaltung wurde oben ausführlich beschrieben, aber die Frage, „Wie verhält sich JUST bei Aktivitäten und Teilprozessen?“ ist noch unbeantwortet. Diese Frage wird in diesem Abschnitt im Zusammenhang mit dem Wizard geklärt.

Wizardführung

Die einzelnen Aktivitäten eines Prozesse sollen mit Hilfe eines Wizards, der den Anwender Schritt für Schritt „an der Hand“ führt, durchgeführt werden. Dabei unterstützt der Wizard den Anwender in folgenden Punkten:

- Erstellen von Artefakten (Spezifikation, Lastenheft, Use-Case-Modell, etc.)
- Übersicht über den Inhalt und den Aufbau eines Artefakts
- Erstellen von JUST-Objekten wie Anforderungen oder Use-Cases zu einem Artefakt
- Automatisiertes Erkennen und Erzeugen von Abhängigkeiten zwischen Artefakten
- Automatisiertes Erkennen und Erzeugen von Abhängigkeiten zwischen JUST-Objekten (oder Testfällen, Handbüchereinträgen, etc)
- Auflisten der Resultate
- Beschreibung von häufig begangenen Fehlern und bekannten Tücken
- Vorschlagen von bewährten Praktiken und Methoden
- Anzeigen von möglichen Bedingungen (Vorbedingung und Ziel)

Der Wizard ist mit einem Rechtekonzept versehen, das wie folgt zu interpretieren ist:

- **Der Anwender hat Schreib- und Leserechte:** Der Anwender kann JUST-Objekte eines Artefakts bearbeiten und neue hinzufügen. Er kann Änderungen einer Aktivität abspeichern. Weiterhin kann er Aktivitäten *abschließen*.
- **Der Anwender hat nur Leserechte:** Der Anwender darf sich nur Informationen anzeigen lassen, seien es Artefakte oder deren JUST-Objekte, die Beziehungen zwischen den JUST-Objekten, Vorlagen, Hilfsmittel oder Techniken. Alle angezeigten JUST-Objekte, seien es Anforderungen, Test-Cases, Beziehungen usw. werden ebenso nur mit Lesezugriff geöffnet. Dies betrifft insbesondere die Reiter *„Wizard"*, die Plugin eigenen Reiter und den Reiter *„Beziehungen"*.

In den anschließenden Textpassagen werden ein paar grundlegende Fragen zum Verhalten des Wizards beantwortet:

- Was bedeutet es, wenn eine Aktivität *aktiv*, *geschlossen* oder *offen* ist?
- Wie verhält sich JUST bei *aktiven*, *geschlossenen* oder *offenen* Aktivitäten?
- Wie verhält sich JUST bei *aktiven*, *geschlossenen* oder *offenen* Teilprozessen?
- Was passiert, wenn eine Aktivität geschlossen wird?

Dazu ist es wichtig zu wissen, dass eine Aktivität an Bedingungen (Vorbedingung und Ziel) gebunden sein kann und dass sie über mehrere eingehende und ausgehende Pfade verfügt.

1. Frage: Was bedeutet es, wenn eine Aktivität *aktiv*, *geschlossen* oder *offen* ist?

(1) Ein Pfad ist *aktiv*, sobald:
er verfolgt wird.

(2) Ein für eine Aktivität verantwortlicher Anwender ist:
ein JUST-Benutzer, der die Rolle, die im Prozess der Aktivität zugedacht ist, zugewiesen bekommen hat.

(3) Eine Aktivität ist *offen*, sobald:
einer der eingehenden Pfade aktiv ist **und** alle Vorbedingungen der Aktivität erfüllt sind **oder** ein verantwortlicher Anwender sie explizit auf *offen* setzt.

(4) Eine Aktivität ist *geschlossen*, sobald:
alle Ziele einer Aktivität erfüllt sind **und** ein verantwortlicher Anwender sie *schließt*.

(5) Eine Aktivität ist *aktiv*, sobald:
einer der eingehenden Pfade aktiv ist **(exklusiv) oder** alle Vorbedingungen der Aktivität erfüllt sind.

Die Vereinbarungen (1), (3), (4) und (5) treffen auch auf die Teilprozesse zu.

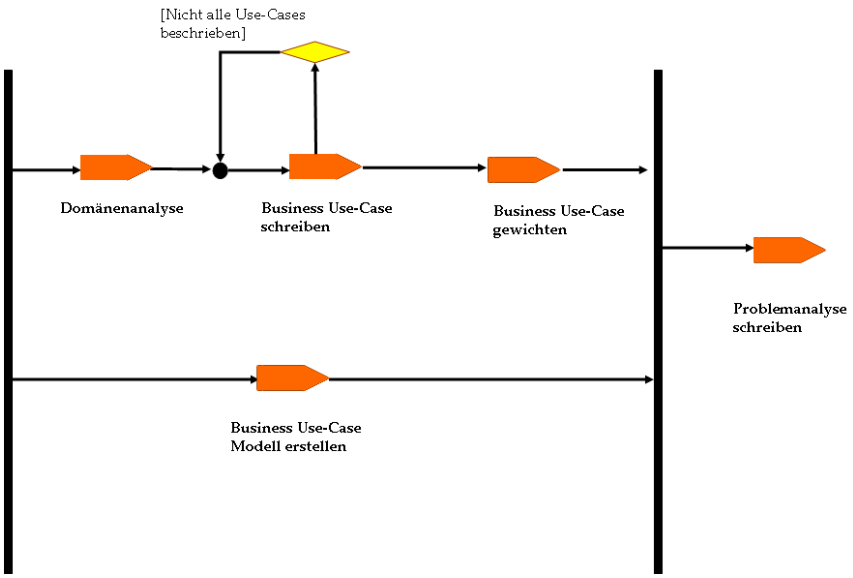


Abbildung 5.9: Prozessbeispiel für die Wizardführung

Zur Verdeutlichung dieser Vereinbarungen wird der Prozess aus Abbildung 5.9 nun eingehender untersucht.

Folgende Bemerkungen:

Aktivität:	Domänenanalyse (A1)
Vorbedingungen:	keine (v1)
Ziel:	Domänenanalyse-Dokument fertig gestellt (n1)
Wiederholung:	inaktiv
Verantwortlicher:	Anwendungsspezialist, Business-Analyst

Aktivität:	Business-Use-Case schreiben (A2)
Vorbedingungen:	Domänenanalyse-Dokument fertig gestellt (v2)
Ziel:	1 Business-Use-Case fertig (n2)
Wiederholung:	Alle Business-Use-Cases erstellt? (Referenz: selbst) (w1)
Verantwortlicher:	Business-Analyst

Aktivität:	Business-Use-Case gewichten (A3)
Vorbedingungen:	1 Business-Use-Case fertig und noch nicht gewichtet (v3)
Ziel:	1 Business-Use-Case gewichtet (n3)

Wiederholung:	inaktiv
Verantwortlicher:	Business-Analyst
Aktivität:	Business-Use-Case-Modell erstellen (A4)
Vorbedingungen:	Alle Business-Use-Cases fertig (v4)
Ziel:	Business-Use-Case-Modell fertig (n4)
Wiederholung:	inaktiv
Verantwortlicher:	Business-Analyst
Aktivität:	Problemanalyse schreiben (A5)
Vorbedingungen:	Business-Use-Case-Modell fertig (v5)
Ziel:	Lastenheft fertig
Wiederholung:	inaktiv
Verantwortlicher:	Anwendungsspezialist, Business-Analyst

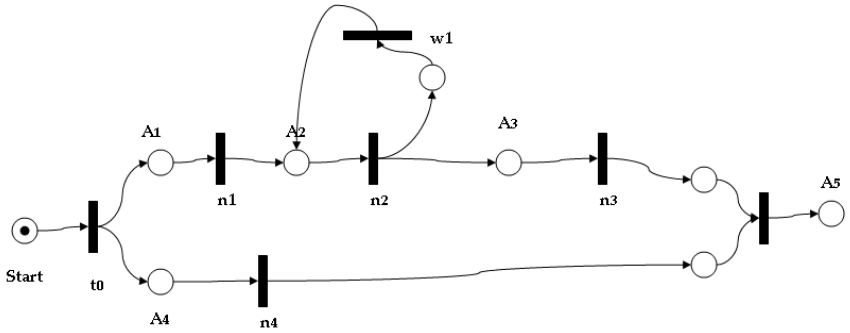


Abbildung 5.10: Aktivierung der Aktivitäten durch den Anwender

Das Petrinetz der Abbildung 5.10 soll die Aktivierung der Aktivitäten durch den Anwender veranschaulichen. Eine Aktivität wird aktiviert, wenn ein verantwortlicher Anwender die Vorgängeraktivität abgeschlossen hat. (Eine Aktivität gilt als abgeschlossen, wenn der Akteur diese mit Hilfe des Wizards abschließt, wobei die Ziele dieser Aktivität erfüllt sein müssen.)
Die Ziele entsprechen den Transitionen n1-n4, wobei t0 bedeutet, dass der Akteur am Projekt angemeldet ist. Die Stellen A1-A5 entsprechen den Aktivitäten. Die drei leeren Stellen werden gebraucht, um das richtige Verhalten der Aktivierung durch ein Petrinetz darstellen zu können.

Das Petrinetz der Abbildung 5.11 soll die Aktivierung von Aktivitäten durch die Vorbedingungen veranschaulichen. Jede Aktivität hat ein Bündel an Vorbedingungen, die ebenfalls zur Aktivierung einer Aktivität beitragen. Erst wenn alle Vorbedingungen einer Aktivität erfüllt sind, wird sie *aktiv* geschaltet. Die Transitionen entsprechen den Vorbedingungen der jeweiligen Aktivität, die Stellen entsprechen den Aktivitäten. Sind in beiden Petrinetzen die Stellen mit identischer Nummer mit einer Markierung belegt, so ist ihr Status *offen*. Ist dagegen die Vorbedingung einer Aktivität erfüllt, d.h. die Markierung gesetzt, aber der Akteur hat sie noch nicht aktiv geschaltet, so ist sie lediglich *aktiv* und diese Aktivität kann von dem Verantwortlichen nur mit Lesezugriff geöffnet werden.

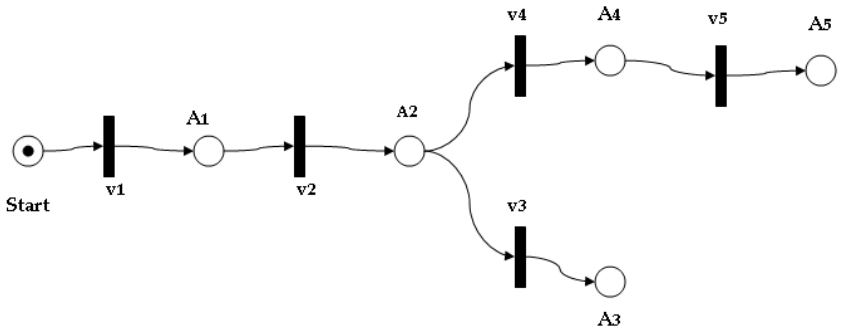


Abbildung 5.11: Aktivierung der Aktivitäten anhand der Vorbedingungen

Erst wenn sowohl ein Akteur eine Aktivität durch Abschließen einer Vorgängeraktivität *aktiv* schaltet als auch die Aktivität alle nötigen Vorbedingungen erfüllt, gilt sie für den Verantwortlichen als *offen*.

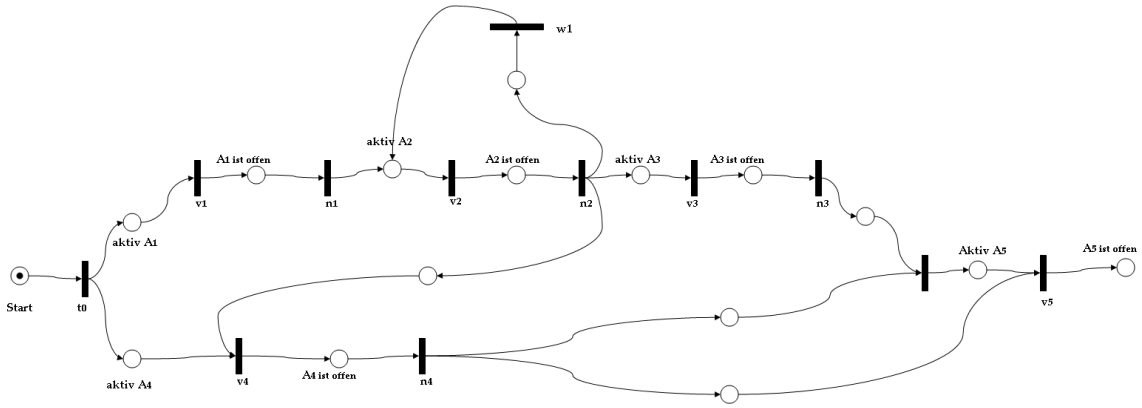


Abbildung 5.12: Wann sind die Aktivitäten „offen“?

Das Petrinetz der Abbildung 5.12 vereint beide Petrinetze und zeigt, wie für das gegebene Beispiel die einzelnen Zustände der Aktivitäten erreicht werden können. Die Transitionen v1-v5 stehen für die einzelnen Vorbedingungen, die Transitionen n1-n5 für die Ziele der Aktivitäten A1-A5. Die Transition t0 ist geschaltet sobald der Prozess betreten wird. Die Aktivitäten werden in zwei Zuständen dargestellt. Der aktive Zustand bedeutet, dass die Aktivität von einer Vorgängeraktivität *aktiv* geschaltet wurde (der Benutzer schließt die Vorgängeraktivität ab). Sobald eine Aktivität *aktiv* ist, kann sie durch das Eintreffen ihrer Vorbedingungen auf den Status *offen* gesetzt werden. (Die Darstellung eines Petrinetzes, in dem eine Aktivität erst durch die Vorbedingung *aktiv* geschaltet und anschließend durch den Anwender auf *offen* gesetzt wird, lasse ich hier weg, da es für das Verhalten des Wizards keine neuen Erkenntnisse einbringt.)

Neben dieser vom System gesteuerten Bestimmung des Status einer Aktivität erhält der Anwender, nach Bedingung (3), seinerseits auch die Möglichkeit Aktivitäten auf *offen* zu setzen. Er kann entweder eine beliebige nicht offene Aktivität über die Prozess-Ansicht öffnen und sie so explizit auf *offen* setzen oder aber er setzt über einen Button im Wizard die nächsten Aktivitäten auf *offen*, ohne dass seine Aktivität bereits abgeschlossen sein muss. Somit erhält

der Anwender die Möglichkeit den Prozess zu beeinflussen. Der Wizard dient lediglich der Unterstützung, er soll den Akteur nicht „gängeln“ und in seiner Arbeit einschränken.

2. Frage: Wie verhält sich JUST bei *aktiven*, *geschlossenen* oder *offenen* Aktivitäten?

Aktive Aktivitäten werden nicht in der TODO-Liste (siehe Kapitel 5.2.5) aufgeführt. Sie beeinflussen den Wizard nicht. Der Wizard öffnet sich mit Leserechten.

Geschlossene Aktivitäten werden in der TODO-Liste (siehe Kapitel 5.2.5) durch einen besonders gekennzeichneten Eintrag aufgeführt. Sie gelten als beendet, können aber erneut den Status *aktiv* oder *offen* erhalten. Auch bereits geschlossene Aktivitäten können durch einen zugewiesenen Anwender zu jedem Zeitpunkt wieder auf *offen* gesetzt werden. Der Wizard öffnet sich mit Leserechten.

Offene Aktivitäten werden in der TODO-Liste (siehe Kapitel 5.2.5) durch einen besonders gekennzeichneten Eintrag aufgeführt. Nur offene Aktivitäten können von einem verantwortlichen Akteur bearbeitet werden. Der Wizard öffnet sich mit Schreib- und Leserechten.

3. Frage: Wie verhält sich JUST bei *aktiven*, *geschlossenen* oder *offenen* Teilprozessen?

Aktive Teilprozesse werden dem Anwender in keiner Weise erkenntlich gemacht und haben auch sonst keine Auswirkungen für ihn.

Ein Teilprozess nimmt den Zustand *geschlossen* an, sobald sein zugehöriger Endknoten erreicht wurde. Sobald ein Teilprozess geschlossen ist, werden alle seine ausgehenden Pfade auf *aktiv* gesetzt.

Offene Teilprozesse werden geöffnet und der zugehörige Startknoten aktiviert. Alle von diesem Prozesselement ausgehenden Pfade werden anschließend auf *aktiv* gesetzt.

4. Frage: Was passiert, wenn eine Aktivität geschlossen wird?

Eine Aktivität verfügt über mehrere ausgehende Pfade. Jeder einzelne Pfad wird der Reihe nach auf *aktiv* gesetzt. Wird ein Prozesselement (Aktivität, Teilprozess oder Kontrollflusselement) erreicht, werden die folgenden Fälle überprüft und gegebenenfalls ausgeführt.

Fall I: Die abgeschlossene Aktivität verfügt über keinen ausgehenden Pfad:
Falls die Aktivität über keinen ausgehenden Pfad verfügt, „verpufft“ hier der Kontrollfluss und der Pfad gilt als abgeschlossen.

Fall II: Eine Aktivität folgt der abgeschlossenen Aktivität:
Die anschließende Aktivität wird *aktiv* geschaltet. Sie wird auf ihre Vorbedingungen geprüft. Sollten alle Vorbedingungen erfüllt sein, so ist die Aktivität *offen* und erscheint in der TODO-Liste, andernfalls verbleibt sie im aktiven Zustand, bis ihre Vorbedingungen erfüllt sind.

Fall III: Eine Spaltung folgt der abgeschlossenen Aktivität:
Der Akteur entscheidet über ein Dialogfenster, wie er fortzufahren gedenkt. Ist die Entscheidung gefallen wird der entsprechende Pfad *aktiv* gesetzt und diese Überprüfung beginnt von vorne.

Fall IV: Eine Verschmelzung folgt der abgeschlossenen Aktivität:
Sobald die Verschmelzung erreicht wird, werden alle ausgehenden Pfade *aktiv* gesetzt und diese Überprüfung beginnt von vorne.

Fall V: Ein Synchronisationsbalken folgt der abgeschlossenen Aktivität:
Es werden alle eingehenden Pfade in den Synchronisationsbalken geprüft. Sind alle eingehenden Pfade *aktiv*, so werden alle ausgehenden Pfade aus dem Synchronisationsbalken *aktiv* gesetzt. Ist jedoch mindestens ein Pfad nicht aktiv, so verharrt der Kontrollfluss an dieser Stelle.

Fall VI: Der Endknoten folgt der abgeschlossenen Aktivität:
Der Teilprozess gilt als *geschlossen*. (Das weitere Verhalten kann in der dritten Frage nachgelesen werden.)

Fall VII: Ein Teilprozess folgt der abgeschlossenen Aktivität:
Der Teilprozess wird auf *aktiv* gesetzt. Anschließend werden alle seine Vorbedingungen geprüft. Sind alle erfüllt wird der Teilprozess auf *offen* gesetzt. (Das weitere Verhalten kann der dritten Frage nachgelesen werden.)

Wizarddarstellung

Die Wizarddarstellung ist zweigeteilt (siehe Abbildung 5.13). Auf der linken Seite des Fensters befindet sich die Ansicht der Artefakte, die von der Aktivität gebraucht werden. Jedes Artefakt besteht aus einer Reihe von Just-Objekten (als Baum im Fenster erkennbar). Wird ein JUST-Objekt aus einem Artefakt ausgewählt, so wird der Reiter „Wizard“ mit den entsprechenden Angaben des Just-Objekts befüllt. Des Weiteren werden je nach Typ des Just-Objekts die Felder:

- Beschreibung,
- empfohlenes Vorgehen,
- einzusetzende Techniken und
- die jeweiligen Plugin-spezifischen Reiter

geladen.

Die zurzeit geöffnete Aktivität enthält zusätzliche Angaben bezüglich ihrer Vorbedingungen und Ziele. Es wird angezeigt, ob diese erfüllt oder nicht erfüllt sind.

Je nachdem, über welche Zugriffsrechte der Anwender verfügt, kann er Änderungen an den Artefakten und JUST-Objekten vornehmen oder auch nicht. Ein Anwender erhält nur dann schreibenden Zugriff auf eine Aktivität, falls er die dafür notwendige verantwortliche Rolle innehat.

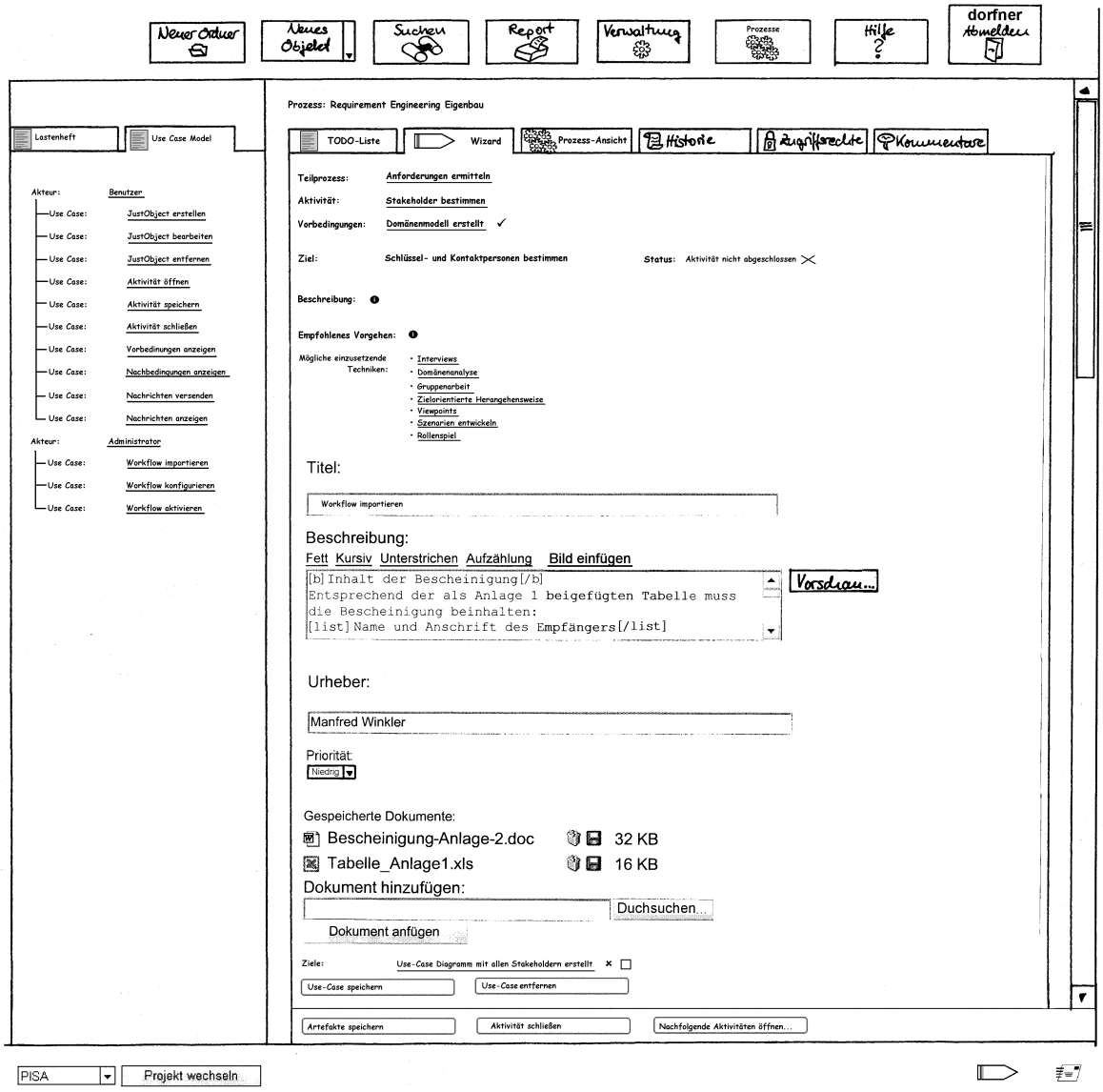


Abbildung 5.13: Der Wizard

5.2.5 Die TODO-Liste

Die TODO-Liste (siehe Abbildung 5.14) zeigt alle Aktivitäten, für die der Anwender verantwortlich ist, an. Selektiert ein Anwender eine offene Aktivität, so wird der Wizard mit Schreib- und Leserechten geöffnet, wählt er dagegen eine bereits abgeschlossene oder noch anstehende Aktivität aus der Tabelle aus, so wird über ein Dialogbox-Fenster erfragt, ob die gewählte Aktivität den Status offen erhalten soll. Bejaht er dies, so wird die Aktivität auf offen gesetzt und der Anwender kann schreibend und lesend auf den Wizard zugreifen. Verneint er die Frage, so bleibt die Aktivität in ihren alten Zustand und der Wizard wird mit Lesezugriff geöffnet. Die Tabelle 5.2 hat folgenden Aufbau:

Spaltenname	Bedeutung
Status	Dieses Attribut beschreibt den Status einer Aktivität und kann eine der folgenden Werte einnehmen: = offene Aktivität = abgeschlossene Aktivität = noch anstehende Aktivität
Teilprozess	Dieses Attribut beschreibt, zu welchem Teilprozess eine Aktivität gehört. Gehört sie zu keinem Teilprozess, bleibt diese Zelle leer.
Aktivität	Dieses Attribut gibt den Namen einer Aktivität an.
Resultat	Dieses Attribut gibt die produzierten Artefakte einer Aktivität an.

Tabelle 5.2: Erläuterung der Tabelle aus Abbildung 5.14

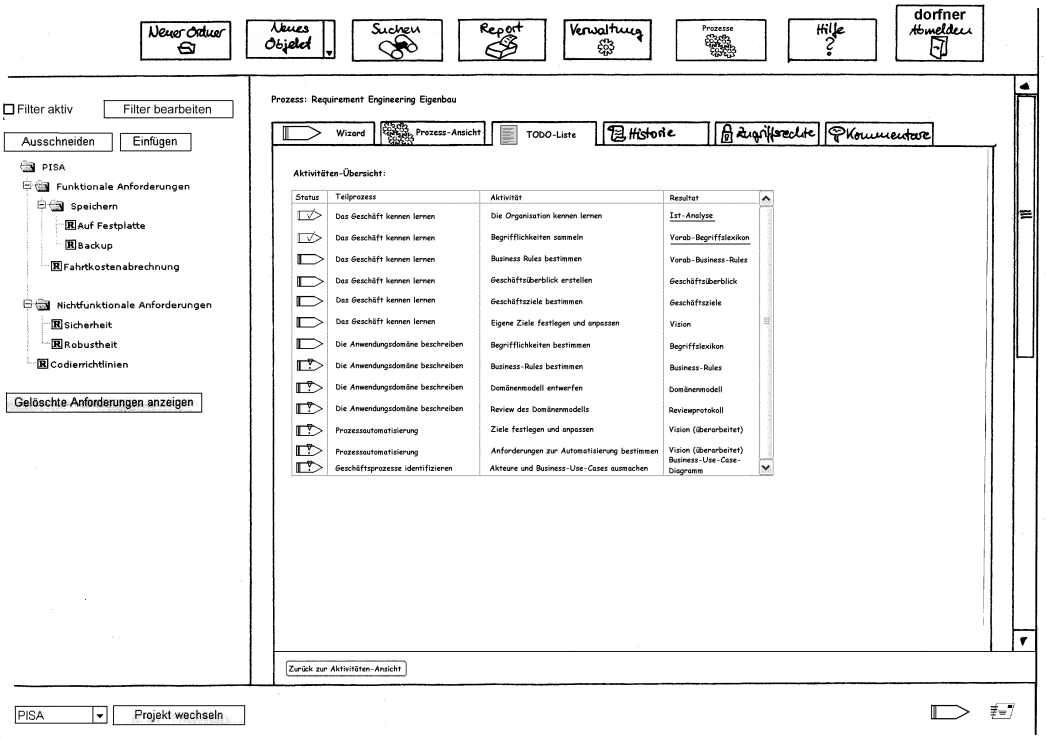


Abbildung 5.14: Die TODO-Liste

6 Die Erweiterung von KoPra

In diesem Kapitel wird die Umsetzung einer Erweiterung von KoPra beschrieben. Dabei wird die Entwicklung eines XML-Exports, der den Export von Prozessmodellen ermöglichen soll und das Hinzufügen von zusätzlichen Prozesselementen für die Prozessmodellierung, vorgestellt.

6.1 Verwendete Technologien

6.1.1 JAVA

Java [JAVA] ist eine objektorientierte Programmiersprache und hat den Vorzug gegenüber anderen Programmiersprachen, plattformunabhängig zu sein. Das heißt, sobald die Anwendung entwickelt wurde, läuft sie auf allen Plattformen, die eine Java-Laufzeitumgebung (Java Runtime Enviroment, kurz JRE) besitzen. Die Idee, die dahinter steckt, ist recht einfach. Java-Quellcode wird nicht in reinen Maschinencode, sondern zuvor in eine Zwischensprache, einen Byte-Code übersetzt, der dann von einem Interpreter, der JRE, in ausführbaren Maschinencode übersetzt wird. Dieser Interpreter ist jedoch plattformabhängig und besteht aus reinem Maschinencode. In der Praxis ist diese Plattformunabhängigkeit jedoch nicht immer gegeben. Die Java-Laufzeitumgebung, die für die Ausführung von Java-Byte-Code notwendig ist, existiert für weit verbreitete Betriebssysteme wie Microsoft Windows, Solaris, Linux, Mac OS, AIX und andere. Des Weiteren gibt es JRE nicht nur für Desktop- und Serversysteme, sondern auch für Embedded Systeme wie PDAs, Smartcards, Mobiltelefone und viele andere.

Die wesentlichen Merkmale der Sprache sind [Wikipedia JAVA]:

- Der Zugriff auf Objekte ist über Zeiger implementiert, die jedoch aus Sicherheitsgründen nicht manipuliert werden können.
- Zusammengehörige Klassen werden in Pakete zusammengefasst, die eine Einschränkung der Sichtbarkeit auf Klassen, Strukturierung von größeren Projekten und Trennung von Namensräumen ermöglichen.
- Es werden Threads (parallel ablaufende Programmteile) und Exceptions (Ausnahmenbehandlung) unterstützt.
- Die Speicherbereinigung wird automatisch von der JRE übernommen.

- Die Sprache unterscheidet zwischen Schnittstellen und Klassen. Eine Klasse kann dabei viele beliebige Schnittstellen implementieren, aber nur eine Basisklasse als Vaterklasse besitzen. Mehrfachvererbung ist somit ausgeschlossen.

Die Urversion von Java ist 1992 entstanden. Hauptentwickler und Mitbegründer der Sprache Java ist James Gosling, der heute als Urvater dieser Sprache gilt. Der Name „Java“ ist auf eine spezielle Java-Kaffeebohne zurückzuführen, die für eine starke Kaffeesorte verwendet wird. Diese haben nämlich die Entwickler sehr gerne getrunken.

6.1.2 JAVA - Swing

Swing [JAVA Swing] ist eine API, die zum Entwickeln von graphischen Oberflächen verwendet wird. Sun Microsystems hat sie für die Programmiersprache Java entwickelt und sie zählt neben Java2D und AWT (Abstract Window Toolkit) zu den Java Foundation Classes (JFC), einer Bibliothek zur Entwicklung von graphischen Oberflächen.

Wesentliche Merkmale

„Look and Feel“: Das Erscheinungsbild (Look) und das Verhalten (Feel) von einzelnen Swing-Komponenten kann angepasst werden. Für die meisten gängigen Plattformen existieren sowohl plattformneutrale als auch plattformabhängige „Look and Feel“-Vorlagen. So kann z.B. ein und dieselbe GUI-Anwendung auf zwei verschiedenen Plattformen zwei vollkommen verschiedene Erscheinungsbilder besitzen, die aber dem zugrunde liegenden Betriebssystem entsprechen.

Leichtgewichtige Komponenten: Die Swing-Komponenten werden nicht vom Betriebssystem gerendert, sondern direkt von Java. Damit sind sie unabhängig vom Betriebssystem und „funktionieren“ auf allen identisch. Der Nachteil ist, dass eine entwickelte Anwendung erstmal nicht wie eine für das Betriebssystem geschriebene Anwendung aussieht. Dies kann aber durch das Merkmal des „Look and Feel“ kompensiert werden.

6.1.3 JGraph

JGraph [Benson (2006)] ist eine Open-Source-JAVA-Bibliothek zur Visualisierung von Graphen sowohl in Client- als auch in Serveranwendungen. Die Visualisierung eines Graphen beruht dabei auf mathematischen Theorien über Netzwerke und der Graphentheorie. (Es wird hier angenommen, dass ein Graph aus Knoten und Kanten besteht. Dabei stellt eine Kante eine Verbindungslinie zwischen zwei Knoten dar.)
JGraph ist voll kompatibel mit der GUI-Bibliothek Swing, die ebenfalls von KoPra verwendet wird. Neben der Visualisierung von Graphen werden auch

- die Interaktion mit Graphen,
- das Festlegen von individuellen Layouts für Knoten und Kanten eines Graphen und
- das Durchführen von Analysen über einen Graphen

von der Bibliothek unterstützt.

Für diese Diplomarbeit wurde die Version 1.2 von JGraph verwendet.

6.1.4 JPOM

JPOM (Java Persistence Object Manager) [JPOM] ist ein Open-Source-Persistenzframework zur Abbildung von objektorientierten Strukturen auf relationale Datenbanken. Es werden die Basisfunktionen zur Gewährleistung der Persistenz von Objekten bereitgestellt. Das sind unter anderem die Vergabe von eindeutigen Objektidentifikationsnummern, die Unterstützung von Transaktionen, Objektanfragen und die Berücksichtigung von Vererbungshierarchien.

Im Verlaufe der Implementierung der Erweiterung von KoPra wurde die Version 1.6 von JPOM verwendet.

6.1.5 Apache Tomcat

Der Apache Tomcat [Apache Software Foundation] ist ein Open-Source-Projekt der Apache Software Foundation. Er ist ein Servlet-Container, mit dessen Unterstützung Java-Code auf Web-Servern ausgeführt werden kann. Mit Hilfe eines integrierten JSP-Compilers können auch JavaServer Pages (JSP) in Servlets übersetzt werden. Zurzeit wird der Tomcat im Rahmen des Jakarta-Projektes der Apache Software Foundation weiterentwickelt.

Für die Implementierung der KoPra-Erweiterung wurde die Version 5.5 von Tomcat eingesetzt.

6.1.6 Rational Rose

Rational Rose [Rational Rose] ist ein Designwerkzeug zur Beschreibung von Software. Es verwendet dazu die Unified Modeling Language (UML) als Modellierungssprache. Zusätzlich können mit Hilfe von Rational Rose Programmrümpfe aus UML-Diagrammen generiert und aus Programmfragmenten UML-Diagramme erzeugt werden. Ursprünglich wurde Rational Rose von der Firma Rational Software entwickelt. IBM übernahm jedoch Rational Software im Jahre 2002 und ist nun für die Weiterführung von Rational Rose zuständig.

6.2 Anforderungserhebung und Spezifikation

Auf Grundlage des Metamodells der Analysemethoden (siehe Kapitel 4) und der Methodik-Unterstützung für JUST (siehe Kapitel 5) haben sich neue Anforderungen an das System KoPra ergeben. Für mich stellte sich die Frage: „Welche Eigenschaften benötigt KoPra zusätzlich, um Prozesse zu modellieren, die es erlauben, den Wizard von JUST für unterschiedliche Analyseverfahren einzusetzen?“. Um eine Lösung für diese Frage zu finden, habe ich zunächst das aktuelle KoPra-System auf bereits existierende Funktionalitäten hin untersucht. Anschließend erstellte ich eine Beispielmethode (auf Basis des Metamodells der Analysemethoden), anhand der ich die momentane Ausdruckskraft der Prozessmodellierung von KoPra eingeschätzt habe. Nach dieser Betrachtung konnte ich die Lücken und Unzulänglichkeiten - die Elemente die im Metamodell aufgeführt sind, die KoPra aber nicht unterstützt - leicht identifizieren.

Die Beispielanalysemethode

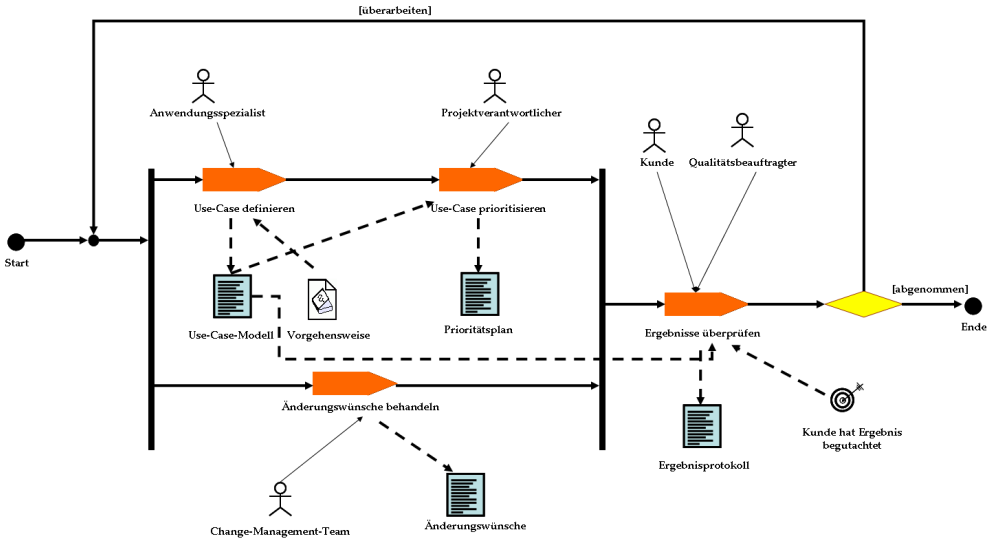


Abbildung 6.1: Beispielmethode

Die Abbildung 6.1 zeigt die Beispielmethode, mit deren Unterstützung ich ermittelt habe, welche Prozesselemente in KoPra bereits umgesetzt wurden und welche nicht. Die Methode beginnt mit der Definition von *Use-Cases*. Diese Aktivität ist dem *Anwendungsspezialist* zugewiesen, der auf Grundlage der beschriebenen Vorgehensweise die einzelnen *Use-Cases* definiert und sie in dem Artefakt *Use-Case-Modell* zusammenfasst. Dieses Artefakt wird bereits in der nächsten Aktivität benötigt, wo der *Projektverantwortliche* die einzelnen *Use-Cases* des *Use-Case-Modells* gewichten soll. Das Ergebnis ist ein *Prioritätsplan*, worin die einzelnen *Use-Cases* nach Bedeutung und „Wichtigkeit“ aufgeführt werden. Parallel zu diesen beiden Aktivitäten werden Änderungswünsche des Kunden von dem Change-Management-Team gesammelt und in dem Artefakt *Änderungswünsche* dokumentiert. Sobald der *Prioritätsplan* erstellt wurde und keine neuen Änderungswünsche mehr vom Kunden eingehen, wird das *Use-Case-Modell* von der *Qualitätssicherung* im Beisammensein mit dem *Kunden* geprüft. Wird es abgenommen, so gilt der Prozess als abgeschlossen. Werden allerdings Mängel oder Feh-

ler gefunden, so wird der gesamte Prozess erneut angestoßen. Die Aktivität „Ergebnis überprüfen“ kann erst abgeschlossen werden, wenn der Kunde das *Use-Case-Modell* ebenfalls geprüft hat.

Was ist neu, was existiert bereits und was muss geändert werden?

Neue Prozesselemente	Vorhandene Prozesselemente	Zu ändernde Prozesselemente
Bedingung Endknoten Hilfestellung Synchronisationsbalken Rolle Spaltung Startknoten Teilprozess Verschmelzung	Aktivität Artefakt Datenfluss Kontrollfluss	Aktivität (neue Eigenschaften) Datenfluss (neue Relationen) Kontrollfluss (neue Relationen) Spaltung (ausgehende Pfade)

Tabelle 6.1: Ergebnis des Vergleichs zwischen KoPra und der Beispielmethode

Aktivität
Die Aktivität, das zentrale Prozesselement der Prozessmodellierung, existiert bereits in KoPra. Jedoch muss dieses Prozesselement um zwei Eigenschaften ergänzt werden. Das ist zum einen die Eigenschaft *Aufgabe*, welche die Aktivität beschreibt und zum anderen die Eigenschaft *Vorgehen*, mit deren Hilfe das empfohlene Vorgehen zur Durchführung einer Aktivität beschrieben wird.

Bedingung
Das neue Prozesselement, die Bedingung, bestimmt, ob eine Aktivität oder ein Teilprozess durchgeführt (Vorbedingung) oder abgeschlossen (Ziel) werden kann. Die Bedingung existiert in zwei Ausprägungsarten: als Vorbedingung und als Ziel. Beide sollen als Stereotypen (siehe Kapitel 2.3) umgesetzt werden. Es darf jeweils nur eine Vorbedingung und ein Ziel mit einer Aktivität und einem Teilprozess verbunden werden.

Sie besitzt zusätzlich zwei Eigenschaften.

Die erste Eigenschaft ist die *informale Bedingung*, die durch einen informalen Text dargestellt wird, der die Bedingung natürlichsprachig festlegt. In JUST wird ein solcher Text später einfach als entsprechende Vorbedingung oder Ziel angezeigt, wobei der Anwender durch Aktivieren oder Deaktivieren einer Checkbox entscheidet, ob diese Bedingung erfüllt oder nicht erfüllt ist. Die informale Bedingung wird also später in JUST von dem Anwender geprüft und gesetzt.

Im Gegensatz dazu soll eine zweite Eigenschaft, die *formale Bedingung*, eingeführt werden. Mit ihr soll es möglich sein, Bedingungen durch eine einfache Prädikatenlogik auszudrücken, die später in JUST automatisch geprüft wird. Die formale Bedingung kann nur für Aktivitäten, nicht für Teilprozesse festgelegt werden.

Eine formale Bedingung unterliegt dem folgenden regulären Ausdruck:

Formale Bedingung = (Artefakt Quantor JUST-Objekt Eigenschaft Operator Wert)

Artefakt = ((a | ... | z | A | ... | Z | -)*)

Quantor = (für alle | für genau einen | für keinen)

JUST-Objekt = ((a | ... | z | A | ... | Z | -)*)

Eigenschaft = Eigenschaft des JUST-Objekts

Operator = (== | !=)

Wert = ((a | ... | z | A | ... | Z | - | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9)*)

Formale Bedingungen einer Aktivität können dann so aussehen:

Beispiel 1:
Formale Bedingung1 = Use-Case-Modell für alle Use-Cases Status == erstellt

Beispiel 2:
Formale Bedingung2 = Use-Case-Modell für genau einen Use-Case Priorität == gesetzt

Beispiel 3:
Formale Bedingung3 = Spezifikation für keinen Anforderung dokumentiert == nein

Die einzelnen formalen Bedingungen können auch miteinander verknüpft werden. Dieser Verknüpfung liegt folgende kontextfreie Grammatik[1] zugrunde:

G = ({&&, | |, !, (,), bedingung}, {S, A}, P, S)

P = {S $\rightarrow \epsilon$,
S $\rightarrow$ A,
A $\rightarrow$ bedingung,
A $\rightarrow$ bedingung | | A,
A $\rightarrow$ bedingung && A,
A $\rightarrow$!(A),
A $\rightarrow$ (A)}

[1] Ohne die paarweise auftretenden Klammern würde eine rechtslineare Grammatik genügen

Dabei sind die Terminalsymbole wie folgt zu interpretieren:

&&	Dieses Symbol entspricht einer Und-Verknüpfung zweier formaler Bedingungen.
||	Dieses Symbol entspricht einer Oder-Verknüpfung zweier formaler Bedingungen.
!	Dieses Symbol entspricht einer Negation des gesamten Ausdrucks.
(...)	Diese beiden Symbole klammern einen Ausdruck.
bedingung	Dieses Symbol entspricht dem regulären Ausdruck einer formalen Bedingung.

Folgende Konventionen gelten für die Negation des regulären Ausdrucks *formale Bedingung*:

(1) ! und == wird zu !=.
(2) ! und != wird zu ==.
(3) ! und **für alle** wird zu für **für keinen**.
(4) ! und **für keinen** bleibt bei **für keinen** (keine Änderung!).
(5) ! und für **genau einen** wird zu **für alle**.

So können formale Bedingungen mit Verknüpfung wie folgt aussehen:

Beispiel 1:
(Formale Bedingung2 && (Formale Bedingung1))

Beispiel 2:
!(Formale Bedingung3 || Formale Bedingung2 || Formale Bedingung1)

Beispiel 3:
!(!(Formale Bedingung2 && (Formale Bedingung1 || !(Formale Bedingung3))))

Eine Bedingung wird mit Hilfe des Datenflusses mit den in der Tabelle 6.2 spezifizierten Prozesselementen verbunden.

Endknoten
Neben dem Startknoten, der den Beginn eines Prozesses repräsentiert, soll nun auch der Endknoten eingefügt werden. Dieser signalisiert das Ende eines Prozesses oder Teilprozesses und darf genau nur einmal eingefügt werden. Vom Startknoten ausgehend muss mindestens ein Pfad den Endknoten erreichen. Der Endknoten besitzt keine zusätzlichen Eigenschaften.

Ein Endknoten wird mit Hilfe des Kontrollflusses mit den in der Tabelle 6.2 spezifizierten Prozesselementen verbunden.

Hilfestellung
Die Hilfestellung verfügt über drei Eigenschaften:

- einen *URL-Link*, der auf eine externe Hilfe verweist,
- einer *Beschreibung*, die einen Hilfetext beinhaltet und
- einer *Datei*, in der die Dokumente (Checklisten, Vorlagen oder Artikel zur Hilfestellung) hinterlegt werden können.

Eine Hilfestellung kann mit den in der Tabelle 6.2 beschriebenen Prozesselementen über einen Datenfluss verknüpft werden. Bei der Anbindung von Dateien, ist darauf zu achten, dass es möglich ist, einmal an eine Hilfestellung angehängte Dateien wieder zu entfernen oder zu ändern.

Kontrollfluss und Datenfluss
Für die neuen Prozesselemente müssen die entsprechenden Relationen spezifiziert werden. Es muss festgelegt werden, mit welchen Prozesselementen ein Prozesselement verknüpft werden kann und mit welchen nicht. Die Tabelle 6.2 stellt die möglichen Verknüpfungen dar. Unter dem Begriff Kontrollflusselement verstehe ich hier (anders als im Anhang B aufgeführt) nur den Synchronisationsbalken, die Spaltung und die Verschmelzung, und nicht den Start- und Endknoten. Das „K“ (für KoPra) bedeutet, dass diese Verknüpfungsmöglichkeit bereits existiert, dagegen bedeutet das „N“ (für neu), dass diese Verknüpfung neu hinzugekommen ist. In der äußeren linken Spalte stehen zeilenweise die Prozesselemente als Quelle der Verknüpfung, in dem Tabellenkopf hingegen sind die Prozesselemente spaltenweise als Ziel einer Verknüpfung zu deuten.

	Ak	Ar	Be	En	Hi	Ko	Ro	St	Te
Ak	K	K	N	N	N	K	K		K
Ar	K				N		K		
Be	N								N
En									
Hi	N	N	N			N	N		N
Ko	K			N	N				K
Ro	K			N	N				
St	K			N		K			K
Te	K		N	N	N	K			K

Tabelle 6.2: Mögliche Relationen zwischen den Prozesselementen

Legende:
Ak = Aktivität
Ar = Artefakt
Be = Bedingung
En = Endknoten
Hi = Hilfestellung
Ko = Kontrollflusselement
Ro = Rolle
St = Startknoten
Te = Teilprozess

Spaltung
Eine Spaltung in KoPra soll die zusätzliche Eigenschaft der *Frage* erhalten, in der die Frage hinterlegt wird, mit deren Hilfe entschieden wird, welcher aus der Spaltung herausführende Pfad verfolgt werden soll. Zusätzlich sollen noch die Antwortmöglichkeiten als Eigenschaft *Antwort* an den ausgehenden Pfaden einer Spaltung hinterlegt werden. Dadurch wird festgelegt, welcher Pfad welcher Antwort entspricht und der Anwender in JUST entscheidet dann durch das Treffen einer Antwortmöglichkeit, welcher Pfad verfolgt werden soll (siehe Kapitel 5.2.3).

Eine Spaltung wird mit Hilfe des Kontrollflusses mit den in der Tabelle 6.2 spezifizierten Prozesselementen verbunden.

Synchronisationsbalken
Ein Synchronisationsbalken synchronisiert alle eingehenden Pfade und aktiviert, sobald alle eingehenden Pfade aktiv sind (siehe Kapitel 5.2.3), alle seine ausgehenden Pfade. Er soll als größenveränderbarer schwarzer Balken dargestellt werden.

Eine Synchronisationsbalken wird mit Hilfe des Kontrollflusses mit den in der Tabelle 6.2 spezifizierten Prozesselementen verbunden.

Prozessexport
Prozesse, die in KoPra modelliert wurden, sollen als XML-Datei exportiert werden. Dazu soll ein XML-Schema erstellt werden, das beschreibt, welche Prozesselemente überhaupt

exportiert werden und wie diese exportierten Prozesselemente in der XML-Datei angeordnet sind. Die exportierte XML-Datei ist die Grundlage für den Wizard in JUST.

Der Export eines Prozessmodells soll dann über das Menü erreichbar sein, wenn ein Modell erfolgreich geladen wurde.

In der Tabelle 6.3 werden die Prozesselemente dargestellt, die exportiert werden müssen und diejenigen, die unberücksichtigt bleiben können. Da die Methodik-Unterstützung von JUST nur die Prozesselemente des Metamodells der Analysemethoden unterstützt, können einige der Prozesselemente wegfallen.

Prozesselemente	Vom XML-Export berücksichtigte Elemente	Vom XML-Export unberücksichtigte Elemente
Aktivität	X	
Artefakt	X	
Bedingung	X	
Direkte Relation	X	
Endknoten	X	
Hilefstellung	X	
Indirekte Relation		X
Iteration		X
Rolle	X	
Startknoten	X	
Spaltung	X	
Teilprozess	X	
Synchronisation	X	
Verschmelzung	X	

Tabelle 6.3: Export von Prozesselementen

6.3 Entwurf

Dieser Abschnitt beschreibt den Entwurf der Erweiterung von KoPra. Dies betrifft zum einen das Domänenmodell, das um die neuen Prozesselemente ergänzt wird, und die Java-Pakete, die um die neuen Funktionen ausgebaut werden.

Das angepasste Domänenmodell

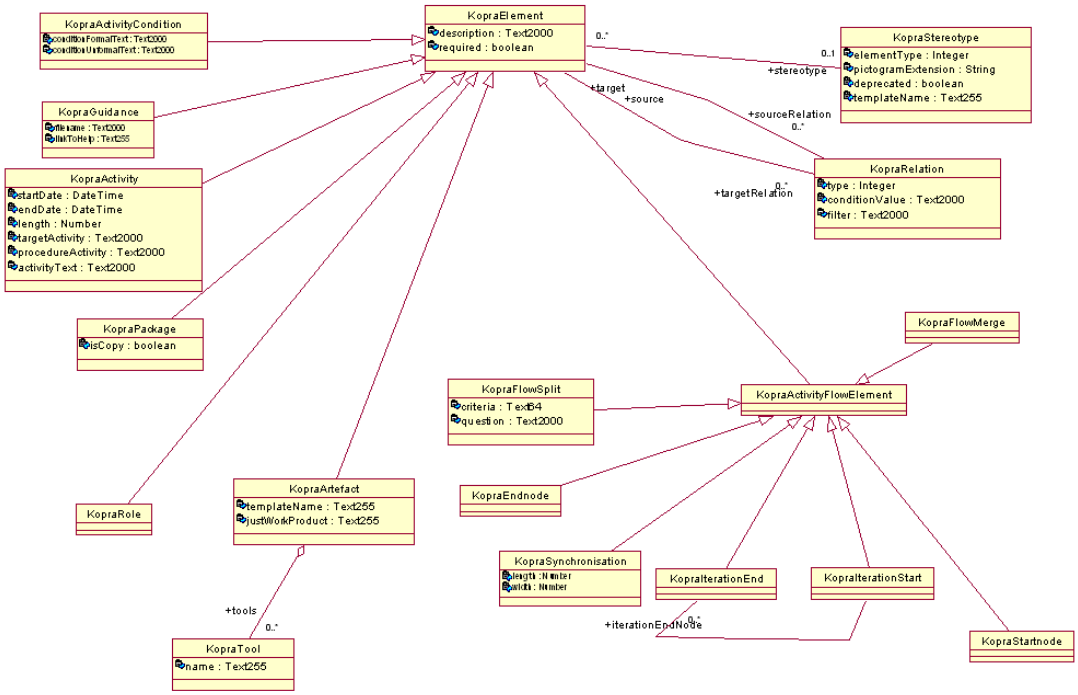


Abbildung 6.2: Domänenmodell der KoPra-Erweiterung

In Abbildung 6.2 ist das Domänenmodell der Erweiterung von KoPra mit allen Neuerungen abgebildet. Die Klasse KopraActivityFlowElement entspricht den Kontrollflusselementen des Metamodells und KopraPackage ist identisch mit der Klasse Teilprozess. Wenn man das Domänenmodell genauer untersucht, wird man feststellen, dass es keine Domänenklasse für die „direkten“ Elemente des Metamodells gibt. Die „direkten“ Elemente erben direkt von der Klasse KopraElement und werden nicht, wie im Metamodell, unter einer weiteren Klasse (Flusselement) zusammengefasst.

Die Pakethierarchie

Da KoPra eine webbasierte Anwendung ist, die auf eine zentrale relationale Datenbank zugreift, ist es unausweichlich, die Pakete in die zwei Gruppen „Server“ und „Client“ (siehe Abbildung 6.3) aufzuteilen. Die Pakete, die in dem gestrichelten Rechteck Server liegen, werden später auf dem Server ausgeführt. Die Client-Pakete dagegen werden von jedem einzelnen Client ausgeführt. So haben wir eine klassische Zweiteilung des Quellcodes, was vor allem später beim Test zu berücksichtigen ist.

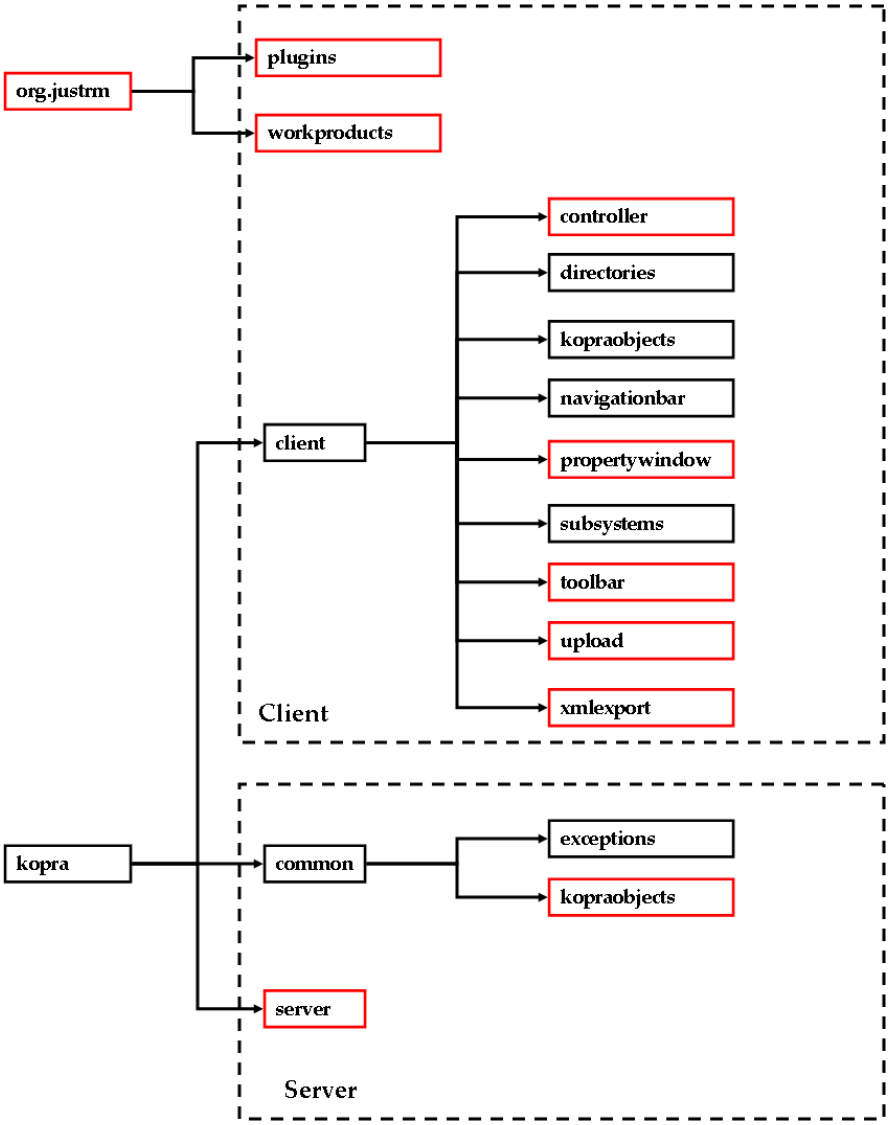


Abbildung 6.3: KoPra-Pakete

Die rot umrandeten Pakete enthalten Quellcode der geändert werden muss. Dagegen bleiben die schwarz umrandeten Pakete unverändert. Nachfolgend werden die einzelnen Pakete und ihre jeweilige Bedeutung für das System erläutert. Handelt es sich dabei um ein rot umrandetes Paket werden zusätzlich auch die beabsichtigten Änderungen beschrieben.

org.justrm:
Dieses Paket wird KoPra neu hinzugefügt. Es beinhaltet die Klassen *BasePlugin* und *BaseWorkProduct*. Die Klasse BaseWorkProduct beschreibt den Aufbau eines Artefakts in JUST und kann sich aus unterschiedlichen Just-Objekten zusammensetzen. Ein *Workproduct* (eine Kindklasse von BaseWorkProduct) muss also jedem KoPra-Artefakt eindeutig zugewiesen werden, ansonsten gäbe es keine Bindung zwischen Artefakten aus KoPra und Just-Objekten in JUST. Beide Klassen werden für das dynamische Laden von Workproducts gebraucht. Zur Ausführungszeit werden dabei die auf dem Server befindlichen Workproducts und JUST-Objekte geladen, um bei der Modellierung von KoPra-Artefakten zur Verfügung zu stehen.

kopra.client.controller:
Dieses Paket enthält die Klasse *ClientController*, die das Applet darstellt und die Kommunikation zwischen den Bestandteilen der Benutzungsoberfläche steuert. Außerdem befinden sich im Paket *controller* Klassen, die diese Bestandteile im Applet darstellen.

Änderungen:
Da Änderungen in der Werkzeugleiste und im Menü vorgenommen werden, müssen auch kleinere Anpassungen beim Clientcontroller vorgenommen werden. Dies ist unumgänglich, da es sich um ein zentrales Paket zur graphischen Oberflächengestaltung handelt.

kopra.client.directories:
Dieses Paket besitzt Klassen, um die Verzeichnisstruktur darstellen, kontrollieren und verwalten zu können.

kopra.client.kopraobjects:
Dieses Paket ist für die Darstellung der Prozesselemente im Graphen zuständig.

kopra.client.navigationbar:
Dieses Paket ist für die Darstellung und Kontrolle der Navigationsleiste zuständig.

kopra.client.propertywindow:
Dieses Paket ist für die Kontrolle und Darstellung des Eigenschaftsfensters mit seinen verschiedenen Reitern zuständig.

Änderungen:
Da viele neue Eigenschaften, wie die Frage und Antwortmöglichkeiten einer Spaltung, hinzugekommen sind, muss dieses Paket angepasst werden.

kopra.client.subsystems:
Dieses Paket ist verantwortlich für die Kontrolle und Darstellung des Inhalts im Hauptfenster. Es kümmert sich dabei insbesondere um die Benutzerverwaltung, das Anzeigen von Fehlerseiten, dem System-Login und der Verwaltung von Modellen und Stereotypen.

kopra.client.toolbar:
Dieses Paket übernimmt die Aufgabe, die Werkzeugleiste darzustellen und ihre Funktionen zu überwachen.

Änderungen:
Drei neue Buttons, der Synchronisationsbalken, der Endknoten und die Hilfestellung, müssen nachträglich in dieses Paket eingefügt werden.

kopra.client.upload:
Dieses Paket beschäftigt sich mit dem Upload von Dateien vom Client zum Server.

Änderungen:
Der Upload im aktuellen Stand unterstützt kein Hochladen von beliebigen Datei-Formaten wie PDF- oder Word-Dateien, sondern lediglich Bildformate. Das Upload-Paket muss daher um diese zusätzliche Funktion ergänzt werden, da eine Hilfestellung mit Dateien beliebigen Formats versehen werden kann.

kopra.client.xmlexport:
Dieses neue Paket beinhaltet Klassen, die für den XML-Export von Prozessmodellen zuständig sind.

kopra.common.exceptions:
Dieses Paket beinhaltet die Ausnahmen (im engl. Exception), die innerhalb des Systems auftreten können.

kopra.common.kopraobjects:
Dieses zentrale Paket beinhaltet alle Klassen des Domänenmodells. Diese Klassen besitzen Methoden mit denen die Daten aus der Datenbank geladen, gespeichert und bearbeitet werden können. Zusätzlich zu den Daten aus der Datenbank enthalten die Klassen lokale Eigenschaften der Prozesselemente, die nicht in der Datenbank gespeichert werden.

Änderungen:
Die zuvor spezifizierten neuen Prozesselemente Synchronisation, Endknoten und Hilfestellung sowie einige zusätzliche Attribute (siehe Domänenmodell) müssen ergänzt werden.

kopra.server: Das Paket *kopra.server* besitzt Klassen, die für die Auswertung von Clientanfragen und für das Empfangen und Speichern von hochgeladenen Dateien notwendig sind. Weiterhin besitzt es eine Klasse, die die Serverkonfigurationsdateien ausliest.

Änderungen: Dieses Paket muss zum einen so erweitert werden, dass es Dateien mit beliebigen Format empfangen und (für den Export) versenden kann, zum anderen müssen die Server-Klassen um eine Funktion ergänzt werden, die es ermöglicht, Workproducts, die auf dem Server abgelegt sind, zu laden.

6.4 Implementierung

Dieser Abschnitt beschäftigt sich mit der Implementierung der in Kapitel 7.1 spezifizierten und in Kapitel 7.2 entworfenen Erweiterung von KoPra.

Als Entwicklungsumgebung habe ich die in der Tabelle 6.4 aufgeführten Hardware- und Softwarekomponenten verwendet.

Eingesetzte Hardware und Software	
Prozessor	1,6 GHz Intel Pentium
Hauptspeicher	512 MB
Betriebssystem	Windows XP Professional SP2
Entwicklungsumgebung	Eclipse SDK 3.1.2
Browser	Mozilla Firefox 1.5.0.4
Relationale Datenbank	MySQL Server 5.0
Web-Server	Apache Tomcat 5.5.16
Java	Java 2 SDK 1.4.2_11
Konfigurationsverwaltung	CVSNT 2.5.03

Tabelle 6.4: Entwicklungsumgebung

Da ich ein bereits bestehendes System erweitern musste, bestand meine erste Aufgabe darin, mich mit dem Programm und dem zugrunde liegenden Quellcode vertraut zu machen. Dies vollzog sich schnell, da der Quellcode ausführlich dokumentiert war. Auch die zusätzliche Dokumentation wie der Entwurf und die Spezifikation waren sehr hilfreich, um den Aufbau und das Verhalten von KoPra nachzuvollziehen.

Die Implementierung war erfolgreich. Ich war in der Lage, alle spezifizierten Anforderungen an die Erweiterung von KoPra umzusetzen. Der konsequente Einsatz eines Funktionstest nach Umsetzung einer Anforderung hat dazu beigetragen, dass die Implementierung schnell und ohne größere Komplikationen verlief. Blicke ich auf diese Arbeit zurück, erinnere ich mich nur noch an zwei kleinere Schwierigkeiten bei der Umsetzung der Erweiterungen von KoPra. Der erste war die Einarbeitung in die API von JGraph, die zur Modellierung von Prozessmodellen in KoPra eingesetzt wird. Die zweite weitaus größere Herausforderung bestand darin, das XML-Schema für den Export auszuarbeiten. Da ich am Anfang mit dem Nicht-Determinismus der Grammatik für die formalen Bedingungen kämpfen musste (XML-Schemas unterstützen aus verarbeitungstechnischen Gründen keinen Nicht-Determinismus) und ich nicht in der Lage war, ungeordnete Sequenzen von Prozesselementen abzubilden, kostete es mich dann doch mehr Überlegungen und Anstrengungen dieses Schema zu entwickeln. Doch blieben diese beiden kleineren Probleme die einzigen, mit denen ich während der Implementierung zu kämpfen hatte, sodass ich schnell mit dem Systemtest fortfahren konnte.

6.5 Test

Zwei Arten von Tests wurden durchgeführt: der Funktionstest, der von mir direkt nach der Implementierung einer Funktion ausgeführt wurde und der Systemtest, der anhand eines Testplans (siehe Anhang D) am Ende der Implementierung durchgeführt wurde. Der Testplan für den Systemtest besteht aus systematisch aufgebauten Testszenarien, die das spezifizierte Verhalten der Erweiterung möglichst komplett abdecken sollen.

6.5.1 Durchführung

Nach Abschluss der Implementierung erstellte ich anhand des spezifizierten Verhaltens der Erweiterung von KoPra einen Testplan (siehe Anhang D). Ich führte alle Testfälle des Testplans nacheinander aus, notierte zu jedem Testszenario das Testergebnis und machte, wenn es nötig war, noch einige Bemerkungen zu den Resultaten, auf die ich später bei der Korrektur Bezug nahm. Schlug ein Testszenario fehl, bedeutete dies kein Testabbruch, sondern es wurden immer alle Testszenarien durchgespielt und anschließend die gesammelten Testergebnisse zur Korrektur verwendet. Der Test galt als abgeschlossen, sobald alle Testszenarien erfolglos durchgeführt wurden.

6.5.2 Ergebnisse

Der Systemtest war ergiebiger als zunächst vermutet. Durch das Nachbauen einzelner Analysemethoden, wie die der Structured Analysis (siehe Abbildung 3.2) oder der Beispielmethode (siehe Abbildung 6.1), wurden einige Mängel und Fehler aufgedeckt. So ergab der Systemtest, dass das Verknüpfen von Bedingungen und Teilprozessen nicht funktionierte. Auch war es möglich, mehr als einen Endknoten in einen Prozess einzubinden, was laut Spezifikation auch verboten ist. Ebenso stellte sich heraus, dass die Hilfestellung nicht mit Bedingungen und Rollen verknüpft werden konnte. Alle diese kleineren Fehler wurden durch das ausführliche Testen aufgedeckt.

7 Fazit und Ausblick

In diesem Kapitel wird zunächst ein Überblick über den Verlauf dieser Diplomarbeit gegeben. Dabei werde ich auch auf Probleme eingehen, die während meiner Arbeit auftraten und über meine Erfahrungen, die ich im Rahmen dieser Diplomarbeit gewonnen habe, berichten. Abschließend hebe ich den Nutzen und die Möglichkeiten einer Methodik-Unterstützung für JUST hervor und gebe einen kurzen Ausblick auf mögliche, zukünftige Erweiterungen.

7.1 Rückblick

Die Aufgaben, die es auf dem Weg zu diesem Abschlussbericht zu bewältigen galt, habe ich dann trotz anfänglicher Schwierigkeiten alle lösen können (siehe Tabelle 7.1).

Teilaufgabe	Ergebnis
Einen Projektplan mit expliziter Angabe der Meilensteine erstellen.	[Kaschowitz (2006b)]
Eine Literaturrecherche zum Thema Anforderungsanalyse durchführen.	[Kaschowitz (2006c)]
Ein Metamodell der Analysemethoden erstellen.	[siehe Kapitel 4]
Eine Methodik-Unterstützung für JUST konzipieren und spezifizieren.	[Kaschowitz (2006a)]
Erweiterungen von KoPra spezifizieren.	[siehe Kapitel 6]
Die KoPra-Erweiterung implementieren.	Die Implementierung der Erweiterung von KoPra liegt als WAR-Datei vor.
Einen Abschlussbericht mit allen Ergebnissen schreiben.	Das Ergebnis ist diese Diplomarbeit.

Tabelle 7.1: Aufgaben und Ergebnisse dieser Diplomarbeit

Der Verlauf dieser Diplomarbeit richtete sich stark nach den Aufgaben. Dies belegen auch die im Projektplan festgesetzten Phasen und Meilensteine, die ich stets bemüht war strikt einzuhalten. Meiner Diplomarbeit lag ein Wasserfallmodell zugrunde, das sich in sieben Phasen unterteilte:

- Literaturrecherche
- Analysemethoden

- Metamodell
- Konzeption und Spezifikation einer Methodik-Unterstützung für JUST
- Spezifikation der Erweiterung von KoPra
- Umsetzung der Erweiterung von KoPra
- Abschlussbericht

Alle sieben Phasen wurden rein sequentiell durchlaufen. Erst nach erfolgreichem Abschluss eines Arbeitsschrittes habe ich mit dem nächsten Arbeitsschritt begonnen. Dieses Vorgehen war für mich sehr dienlich und unausweichlich, da jeder Arbeitsschritt auf den vorherigen Arbeitsschritten und deren Resultate aufbaute. Paralleles Arbeiten an mehreren Tätigkeiten war gar nicht möglich. Die Phasen wurden im Projektplan mit Meilensteinen versehen, die ich alle einhalten konnte (siehe Tabelle 7.2).

Meilenstein	Soll-Termin	Ist-Termin
Projektplan	27.01.2006	27.01.2006
Literaturrecherche	10.02.2006	10.02.2006
Analysemethoden	06.03.2006	03.03.2006
Metamodell	24.03.2006	20.03.2006
Spezifikation der JUST-Erweiterung	21.04.2006	17.04.2006
Spezifikation der KoPra-Erweiterung	03.05.2006	25.04.2006
Umsetzung der Erweiterung von KoPra	29.05.2006	22.05.2006
Abschlussbericht	24.07.2006	24.07.2006

Tabelle 7.2: Wichtige Meilensteine

Aufgetretene Probleme

Wie bereits weiter oben schon angedeutet, war gerade der Beginn der Diplomarbeit mit vielen Problemen verbunden. Es fiel mir unerwartet schwer, für mein Thema zweckdienliche Literatur zu finden, was nicht zuletzt an einer mangelnden Erfahrung, wie man eine „richtige“ Literaturrecherche betreibt, lag. So war gerade am Anfang der Einstieg in das Themengebiet, das Auseinandersetzen mit dem neuen Stoff, kritisch. Mit erhöhtem Fleiß und nicht zuletzt guten Ratschlägen seitens des Betreuers gelang es mir dann doch, zugegeben langsam, mich in das Stoffgebiet der Anforderungsanalyse und seiner Methoden und Methodologien zu vertiefen. Das Filtern von gewinnbringenden Informationen, das richtige Zuordnen und Einordnen von Methoden und Methodologien erschwerte die Arbeit ungemein. Ich glaube gerade diese beiden Punkte, einerseits eine richtige Literaturrecherche durchzuführen, andererseits das geschickte, effiziente Aufbereiten von Informationen waren die größten Probleme, mit denen ich im Verlaufe meiner Diplomarbeit zu kämpfen hatte.

Ansonsten traten nur die gewohnten, bereits bekannten Probleme bei der Implementierung und Spezifikation auf, wie zum Beispiel der korrekte Aufbau einer Spezifikation, das Einrichten der Entwicklungsumgebung, das Fehlverhalten von entwickelten Funktionen, das Dokumentieren und Benennen von Anforderungen an ein System und die üblichen Dinge, die während der Implementierung noch auftreten können.

7.2 Fazit

Ziel dieser Diplomarbeit war es, ein System zu konzipieren, das den Anwender zu einer methodischen Anforderungserhebung anleitet. Diese Führung des Anwenders durch das System sollte aber möglichst dem Benutzer verborgen bleiben, ihn nicht mit unnötigen Aufgaben „gängeln“ und in seiner Arbeit einschränken, sondern ihm helfend und unterstützend bei „Seite stehen“. Und das alles vereint die neu konzipierte Methodik-Unterstützung für JUST. Mit Hilfe eines Wizards, der von KoPra-Prozessen „gespeist“ wird, wird der Anwender „an der Hand“ durch seine Aufgaben geführt. Betrachte ich diese Diplomarbeit und die einzelnen Teilaufgaben, so gelange ich zum Ergebnis, dass diese Diplomarbeit für mich ein voller Erfolg war. Nicht nur das eigentliche Endresultat, die Methodik-Unterstützung für JUST, lässt mich zu dieser Annahme kommen, sondern vielmehr die kleinen Arbeitsschritte, die zu dieser Gesamtlösung führten, waren lehrreich für mich. Da wäre zuerst die Literaturrecherche zu nennen. Es war das erste Mal, dass ich eine solche ausführliche und weit reichende Literatursichtung durchgeführt habe. Aber auch die Erkenntnisse, die ich aus den Methoden der Analyse, den vielen verschiedenen Methodologien gewonnen habe, stufe ich für mich höher ein als die spätere Konzeption und Spezifikation der Erweiterungen. So habe ich sehr viel über die Structured Analysis und den Rational Unified Process in Erfahrung bringen können. Ich kann nun durchaus kritischer und besser die Analysemethoden einschätzen, kenne jetzt viele ihrer Stärken und auch einige ihrer Schwächen und bin in der Lage, die Bedeutung der verschiedenen Analyseverfahren einzuschätzen.

7.3 Ausblick

Wizardführung automatisiert vs. benutzerbeeinflusst?

Die Wizardführung, wie sie jetzt existiert, verläuft halbautomatisch. Durch die formalen Bedingungen einer Aktivität lässt sich der methodische Ablauf automatisieren. Durch das nachträgliche Öffnen einer Aktivität seitens des Benutzers wird dieser Automatismus zum Teil allerdings wieder außer Kraft gesetzt. Dies führt zu unklaren und zweideutigen Systemverhalten, was zur Verwirrung bei dem Anwender führen kann. Ein möglicher Ansatz wäre es, den Wizard zu konfigurieren. So könnte man projektweise festlegen, ob die Wizardführung vollautomatisch (also nur über das System), halbautomatisch (so wie in dieser Diplomarbeit beschrieben) oder gar nur benutzergesteuert ausgeführt werden soll.

Ansicht des Projektstandes?

Die in dieser Diplomarbeit vorgestellte Methodik-Unterstützung für JUST bietet keine Ansicht, die Aufschluss über den aktuellen Stand des Projektes oder das Vorankommen einer Methode gibt. Hier könnte eine geeignete Ansicht für den Projektverantwortlichen entworfen werden, wo neben den graphisch aufbereiteten aktuellen Stand auch Metriken und Auswertungen angezeigt werden, die Aufschluss über den Fortschritt eines Projekts geben. Zusätzlich könnte man sich überlegen, die gemessenen Größen anderer Projekte als Vergleichsdaten mit heranzuziehen. Hier gibt es ein breites, offenes Gebiet mit vielen neuen Ideen.

7.4 Schlussbemerkung

Ich hoffe, dass diese Diplomarbeit in der nahen Zukunft Grundlage für eine Umsetzung in JUST sein wird und dass die hier vorgeschlagenen Ideen und Konzepte dort eingehen werden.

Anhang A

Structured Analysis

A.1 Datenflussdiagramm (DFD)

Mit dem DFD kann der Systemanalytiker das System als Netzwerk von Prozessen darstellen. Diese Prozesse sind durch „Pipelines“ miteinander verbunden, durch die ein Datenstrom fließen kann. In der Fachliteratur kursieren etliche Synonyme für Datenflussdiagramme herum:

- Bubble chart,
- DFD (diese Abkürzung werde ich verwenden),
- Bubblediagram,
- Prozessmodell,
- Arbeitsflussdiagramm und
- Funktionsmodell.

Das DFD wird insbesondere für aufgabenorientierte Systeme verwendet, bei denen die Funktionen des Systems viel wichtiger und komplizierter sind als die Daten, die im System verwaltet werden. (Im Gegensatz dazu gibt es objektorientierte Systeme, bei denen Objekte, also Daten und Funktionen, gemeinsam im Vordergrund stehen.) DFD wurden in der Softwaretechnik zuerst verwendet, um Planungskriterien zu studieren. Die Symbole wurden aus den frühen Werken der Graphentheorie entlehnt. Wenn ein System entwickelt werden soll, in dem die Datenbeziehungen im Vordergrund stehen und nicht die Funktionen, wird den DFD keine so große Bedeutung beigemessen, sondern dann eher den Entity-Relationship-Diagrammen (ERD).

Die Komponenten des DFD

Von einem Beispiel ausgehend werden nun die wichtigsten Komponenten eines Datenflussdiagramms erläutert.

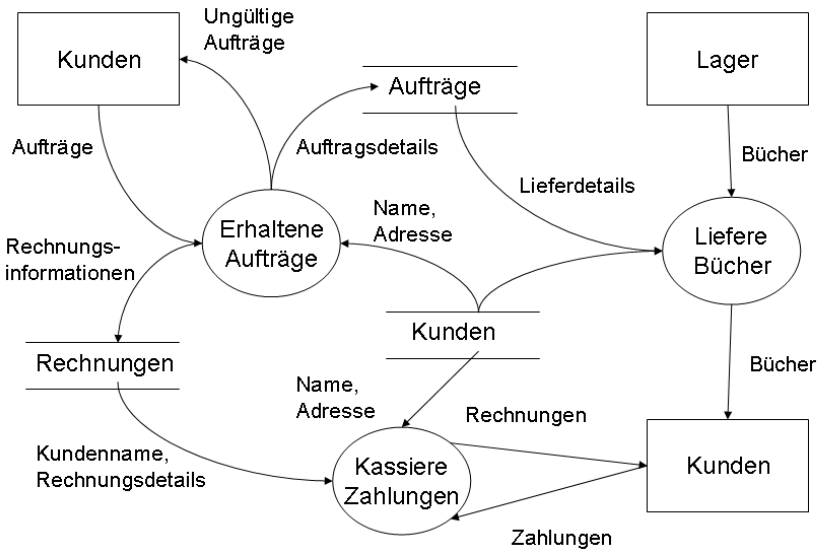


Abbildung A.1: Ein typisches DFD

Das Diagramm muss eigentlich kaum erklärt werden. Es modelliert ein System eines Bücherversandhauses. Das System soll Kundenaufträge verwalten, Bonitätsprüfungen zulassen und Lieferungen verschicken können. Wenn man es betrachtet, fällt die einfach gehaltene Notation auf. Dieser Punkt ist sehr wichtig, denn nicht der Systemanalytiker soll das Modell zu lesen verstehen, sondern der Kunde oder der Benutzer, ohne dass er hierzu Fachliteratur wälzen muss.

Diese Komponente wird Prozess genannt. Der Prozess stellt den Teil des Systems dar, der Eingaben in Ausgaben umwandelt bzw. transformiert. Er wird als Kreis dargestellt.

Der Fluss wird graphisch durch einen Pfeil dargestellt, der in einen Prozess hinein- oder aus diesem herauszeigt. Mit dem Fluss werden Informationsbewegungen von einem Teil des Systems zu einem anderen beschrieben. Flüsse repräsentieren also Daten in Bewegung. Eingabe- und Ausgabeflüsse können logische Beziehungen aufweisen. Es gibt davon drei, die da wären:

*	eine logische UND-Verknüpfung (AND),
⊕	eine exklusive ODER-Verknüpfung (XOR) und
0	eine ODER-Verknüpfung (OR).

Das folgende Beispiel soll eine solche Verknüpfung illustrieren (siehe Abbildung A.2):

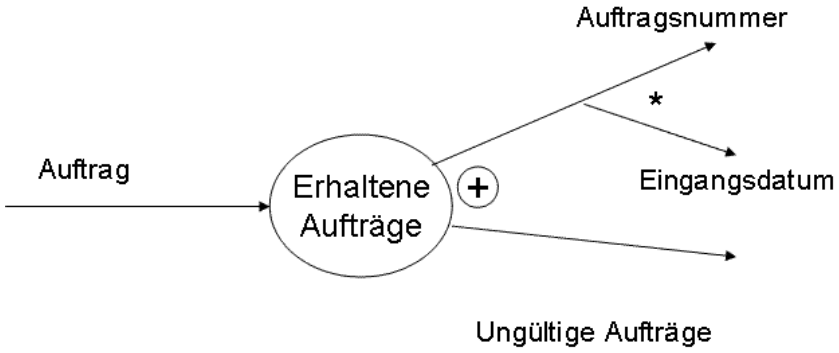


Abbildung A.2: Verknüpfung von Datenflüssen

Der Speicher stellt einen Datenbestand im „Ruhezustand“ (also Dateien, Datenbankeinträge, etc.) dar. Normalerweise wird als Speichername der Plural des Flussnamen gewählt, der in den Speicher hinein- oder herausführt. Der Speicher kann mehr als nur Dateien oder Datenbanken bezeichnen.

Eine weitere Komponente im DFD ist der Terminator. Er repräsentiert ein externes Objekt, mit dem das System kommuniziert. Normalerweise ist der Terminator eine Person oder eine Gruppe, manchmal kann er aber auch ein anderes System darstellen. Terminatoren befinden sich außerhalb des Systems, für das ein Modell erstellt wird. Flüsse, die die Terminatoren mit den Prozessen verbinden, sind die Schnittstellen des Systems zur Außenwelt.

A.2 Kontextdiagramm

Das Kontextdiagramm beschreibt die „höchste“ Ebene in der Datenflusshierarchie. Es veranschaulicht das zu modellierende System als einen einzigen abstrakten Prozess, der mit seiner Außenwelt durch Schnittstellen kommuniziert. Durch ein Kontextdiagramm wird also das Gesamtsystem veranschaulicht und die Interaktion mit der Außenwelt verdeutlicht. Es dürfen keine Speicher in einem Kontextdiagramm vorkommen.

A.3 Datenlexikon

Das Datenlexikon (im engl. data dictionary) beschreibt mit Hilfe der Backus-Naur-Notation (siehe Tabelle A.1) die verwendeten Datenflüsse und Speicher aus den DFDen. Dadurch wird die Semantik dieser beiden Elemente, also die inhaltliche Bedeutung, definiert.

Backus-Naur-Ausdruck	Bedeutung
=	Dieses Element steht für die Äquivalenz.
+	Dieses Element stellt eine Verknüpfung von Elementen zu einer Sequenz dar.
[a | b | ...]	Die Auswahl zwischen mehreren Elementen wird über die eckigen Klammern erzielt.
(...)	Mit Hilfe von Klammern wird ein optionaler Ausdruck dargestellt, der entweder kein- oder genau einmal vorkommen kann.
(n)...(m)	Mit Hilfe der geschweiften Klammern wird eine Wiederholung des Ausdrucks von n mal bis m mal ausgedrückt.

Tabelle A.1: Erläuterung der Backus-Naur-Form

Das Datenlexikon erlaubt eine Überprüfung der Daten auf Redundanz, Konsistenz und Eindeutigkeit.

Anhang B

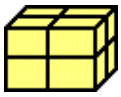
Die Prozesselemente und ihre Bedeutung

Die in dieser Diplomarbeit verwendeten Bilder zur Darstellung von Prozessen basieren auf eigenen Prozesselementen. Zur Interpretation und zum Verständnis dieser Bilder werden nachfolgend die verwendeten Symbole definiert.

Graphische Symbole und ihre Bedeutung

	Aktivität	Eine Aktivität beschreibt eine durchzuführende Tätigkeit oder eine zu lösende Aufgabe. Sie kann sowohl beliebig viele eingehende als auch ausgehende Pfade besitzen.
	Artefakt	Ein Artefakt ist ein Dokument, welches von einer Aktivität entweder verwendet oder produziert wird.
	Bedingung	Eine Bedingung gibt an, ob eine Aktivität oder ein Teilprozess ausgeführt (Vorbedingung) oder abgeschlossen (Ziel) werden können.
	Datenfluss	Durch einen gestrichelten, gerichteten Pfeil werden Datenflüsse modelliert. So werden Relationen zwischen Prozesselementen dargestellt, bei denen mindestens ein Prozesselement: • ein Artefakt, • eine Hilfestellung oder • eine Bedingung ist.
	Endknoten	Dieses Symbol repräsentiert das Ende eines Teilprozesses oder des gesamten Prozesses. Es kann mehrere eingehende Pfade besitzen, muss aber mindestens einen besitzen. Es besitzt keine ausgehenden Pfade.

	Hilfestellung	Eine Hilfestellung kann eine Checkliste, eine Vorlage oder eine Anleitung sein, die den Anwender mit zusätzlichen Informationen bei der Arbeit unterstützen soll.
	Kontrollfluss	Mit Hilfe eines durchgehenden, gerichteten Pfeils werden die Prozesselemente: • Aktivitäten, • Teilprozesse und • Kontrollflusselemente miteinander verbunden.
	Rolle	Eine Rolle repräsentiert eine Personengruppe oder eine organisatorische Gruppe, die für Aktivitäten verantwortlich sind.
	Spaltung	Die Spaltung trennt den Kontrollfluss in mehrere Pfade auf. Jede Spaltung beinhaltet eine Frage, die darüber entscheidet, welcher ausgehende Pfad verfolgt wird. Die Antwortmöglichkeiten werden durch die ausgehenden Pfade repräsentiert. Es wird immer genau ein Pfad verfolgt.
	Startknoten	Dieses Symbol repräsentiert den Anfang eines Teilprozesses oder des gesamten Prozesses. Es kann mehrere ausgehende Pfade besitzen, muss aber mindestens einen haben. Es besitzt keine eingehenden Pfade.
	Synchronisation	Durch einen Synchronisationsbalken wird die Synchronisation von mehreren Pfaden erreicht. Ein Synchronisationsbalken verfügt über mehrere eingehende und ausgehende Pfade, wobei die ausgehenden Pfade erst dann alle gleichzeitig verfolgt werden, wenn alle eingehenden Pfade aktiv sind.

	Teilprozess	Ein Teilprozess ist ein Behälter, der mehrere Aktivitäten, die ein und dasselbe Ziel verfolgen, beinhaltet. Er kann sowohl beliebig viele eingehende als auch ausgehende Pfade besitzen.
	Verantwortlich für	Mit Hilfe dieses dünneren Pfeils werden Rollen mit Aktivitäten verknüpft.
	Verschmelzung	Dieser schwarze, kleine Kreis dient zur Zusammenführung von mehreren Pfaden. Er hat beliebig viele eingehende und ausgehende Pfade. Sobald einer der eingehenden Pfade aktiv ist, werden alle ausgehenden Pfade der Verschmelzung aktiviert.

Anhang C

Begriffslexikon

Im Zusammenhang mit dieser Diplomarbeit wurden Begriffe verwendet, deren Bedeutung mitunter unklar oder mehrdeutig ist, da sie in anderen Kontexten anders verwendet werden oder undeutlich beschrieben sind. Daher werden in diesem Begriffslexikon die Begrifflichkeiten, die dieser Arbeit zugrunde liegen, eindeutig beschrieben.

Aktivität

(siehe Appendix B)

Anwender

Ein Anwender ist eine Person, die eine Software verwendet.

Applet

Unter einem Applet versteht man soviel wie eine „kleine Anwendung“. In dieser Diplomarbeit wird mit Applet ein Java-Applet gemeint. Dies ist ein kleines Computerprogramm, das in einem Web-Browser läuft und in der Programmiersprache Java geschrieben ist.

Artefakt

(siehe Appendix B)

Bedingung

(siehe Appendix B)

Datenfluss

(siehe Appendix B)

Direkte Elemente

Direkte Elemente beschreiben Elemente, die direkten Einfluss auf eine Methode haben. Dies sind im Einzelnen: die Aktivität, das Artefakt, die Hilfestellung, die Rolle und der Teilprozess. Im Metamodell der Analysemethoden entsprechen direkte Elemente den Flusselementen.

Eigenschaft eines Prozesselements

Eine Eigenschaft eines Prozesselements ist ein Attribut, das ein Prozesselement näher beschreibt.

Framework

Das Framework gibt die Architektur einer Software vor. Es handelt sich dabei um eine Ansammlung von bereits implementierten Klassen und Bibliotheken, die nur an bestimmten Stellen mit anwendungsspezifischen Quellcode ergänzt werden. Ein Framework ist auf bestimmte Anwendungsbereiche ausgerichtet.

Hilfestellung

(siehe Appendix B)

JUST

JUST ist ein webbasiertes Anforderungsverwaltungswerkzeug, das in Rahmen eines Studienprojekts entstanden ist. Eine Spezifikation für eine Methodik-Unterstützung für JUST ist im Rahmen dieser Diplomarbeit entstanden.

JUST MoRe

JUST MoRe ist eine Erweiterung von JUST, um Probleme vom Auftreten bis zu deren Abschluss zu behandeln. Es unterstützt die beiden Bereiche Support (mit Hilfe von Software Problem Reports) und Software-Wartung (mit Hilfe von Modification Requests).

JUST-Objekt

Das JUST-Objekt ist ein elementares Datenobjekt des JUST-Systems. Es repräsentiert eine Instanz einer Projektinformation.

Kontrollfluss

(siehe Appendix B)

Kontrollflusselement

Ein Kontrollflusselement beeinflusst den Kontrollfluss. Es ist entweder ein Startknoten, Endknoten, Synchronisationsbalken, Spaltung oder eine Verschmelzung.

KoPra

KoPra ist ein webbasiertes Prozessbeschreibungswerkzeug, das im Rahmen eines Studienprojekts entstanden ist. Mit KoPra ist es möglich, Prozesse zu modellieren. Außerdem können Projektabläufe aus modellierten Prozessen generiert werden.

Petrinetz

Ein Petrinetz besteht aus zwei Arten von Knoten: Knoten, die durch Kreise dargestellt sind, die *Stellen* oder *Bedingungen* genannt werden sowie Knoten, die durch Balken dargestellt sind, die *Transitionen* oder *Ereignisse* genannt werden. Solche Knoten sind durch Pfeile miteinander verbunden. Die Pfeile gehen nur von Stellen zu Transitionen oder von Transitionen zu Stellen. Ein Petrinetz ist also ein bipartiter Graph. (In dieser Diplomarbeit werden nur unmarkierte Petrinetze verwendet.)

Plugin

Ein Plugin ist eine Erweiterung einer bestehenden Software um zusätzliche Funktionen.

Prozess

Der Prozess ist die abstrakte Form der Durchführung eines Projekts. Er wird von Organisationen eingesetzt, um die Arbeit der Individuen zu koordinieren.

Prozesselement

Ein Prozesselement kann eine Aktivität, ein Artefakt, eine Hilfestellung, ein Kontrollflusselement, eine Rolle oder ein Teilprozess sein.

Prozessmodell

Ein Prozessmodell ist die graphische Darstellung eines Prozesses in KoPra. Es besteht aus verschiedenen Prozesselementen wie zum Beispiel Artefakten, Rollen, Aktivitäten und deren Relationen untereinander.

Prozessvisualisierung

Die Prozessvisualisierung ist die graphische Darstellung eines Prozesses in JUST. Sie wird in einem separaten Reiter angezeigt und besteht aus verschiedenen Prozesselementen und deren Relationen untereinander.

Regulierende Elemente

Regulierende Elemente regulieren den Ablauf einer Methode. Sie bestimmen wann, wer eine Aktivität durchzuführen hat. Dies sind im Einzelnen: der Endknoten, die Spaltung, der Startknoten, die Synchronisation und die Verschmelzung. Im Metamodell der Analysemethoden entsprechen regulierende Elemente den Kontrollflusselementen.

Relation

Eine Relation stellt, in dieser Diplomarbeit, eine Beziehung zwischen zwei Prozesselementen dar.

Rolle

(siehe Appendix B)

Spaltung

(siehe Appendix B)

Synchronisation

(siehe Appendix B)

Synchronisationsbalken

Mit Hilfe des Synchronisationsbalkens werden parallel ablaufende Pfade synchronisiert.

Teilprozess

(siehe Appendix B)

TODO-Liste

Eine TODO-Liste ist eine Liste mit offenen Aufgaben.

Verantwortlicher für eine Aktivität

Ein Verantwortlicher für eine Aktivität ist eine Person, dessen Rolle der Aktivität zugewiesen wurde.

Verschmelzung

(siehe Appendix B)

Vorbedingung

Eine Vorbedingung ist eine Ausprägung der Bedingung. Ist sie erfüllt, kann die zugewiesene Aktivität oder der zugewiesene Teilprozess ausgeführt werden, andernfalls nicht.

Wizard

Ein Wizard bezeichnet eine graphische Oberfläche, mittels derer ein Anwender durch mehrere Dialoge geführt wird. Oft wird er auch Assistent genannt.

Workflow

Ein Workflow (engl. Arbeitsablauf) beschreibt einen Vorgang, den es gilt erfolgreich abzuschließen, um diesen Vorgang abzuschließen werden möglichst genaue Arbeitsschritte beschrieben.

Ziel

Ein Ziel ist eine Ausprägung der Bedingung. Ist es erfüllt, kann die zugewiesene Aktivität oder der zugewiesene Teilprozess abgeschlossen werden, andernfalls nicht.

Weitere Begriffe zu JUST oder KoPra sind in (JUST-Team (2004d), KoPra-Team (2005b)] zu finden.

Anhang D

Systemtest

D.1 Zweck des Tests

Dieser Systemtest dient als Grundlage, um die spezifizierten Erweiterungen von KoPra zu prüfen. Dies umfasst den Export von Prozessmodellen und die neuen Prozesselemente Synchronisation, Endknoten, Bedingung und Hilfestellung.

D.2 Testumfang

Getestet werden alle in Kapitel 7.1 spezifizierten Funktionen. Die im Testplan aufgeführten Testszenarien müssen nacheinander durchgeführt werden. Dadurch wird das spezifizierte Systemverhalten der Erweiterung von KoPra geprüft.

D.3 Testumgebung

Server:
- Prozessor: AMD Athlon XP 2000+ (1,66 GHz)
- RAM: 512 MB
- MySQL Server 5.0
- Apache Tomcat 5.5.16

Client:
- Prozessor:1,6 GHz Intel Pentium
- RAM: 512 MB

Betriebssystem:
- Windows XP Professional SP2
- Suse Linux 8.2

Browser:
- Mozilla Firefox 1.5.0.4
- Internet Explorer 6.0

D.4 Testplan

Vorbereitungsarbeiten für den Test
Zunächst muss das KoPra-Installationsarchiv in das Verzeichnis entpackt werden, in dem sich die Web-Anwendungen befinden, die innerhalb des Apache Tomcat-Containers ausgeführt werden. Anschließend muss die Konfigurationsdatei von KoPra angepasst werden. Dies betrifft insbesondere den Datenbankverweis auf die MySQL-DB. Schließlich müssen die benötigten Datenbank-Tabellen in der Datenbank angelegt werden. Hierzu steht ein SQL-Dump als ASCII-Text-Datei zur Verfügung. Abschließend muss der Apache Tomcat-Dienst neu gestartet und die KoPra-Start-URL aufgerufen werden.

Aufräumarbeiten nach dem Test
Als Erstes müssen die MySQL-Datenbank-Tabellen gelöscht und der MySQL-Dienst beendet werden. Anschließend wird der Apache Tomcat beendet und das KoPra-Verzeichnis aus dem Verzeichnis für Web-Anwendungen gelöscht.

Testfall-ID:	01
Vorbedingung:	Der Benutzer ist als Administrator am System angemeldet.
Anweisungen:	1. Der Benutzer erstellt ein neues Prozessmodell.
Soll-Ausgabe:	Das Hauptfenster wird aktualisiert und eine weiße Oberfläche mit einem Startknoten ist zu sehen.
Ist-Ausgabe:	Sie ist identisch zur Soll-Ausgabe.
Testfall-ID:	02
Vorbedingung:	Der Benutzer ist als Administrator am System angemeldet. Der Benutzer hat ein neues Prozessmodell erstellt.
Anweisungen:	1. Der Reihe nach werden folgende Prozesselemente dem Prozessmodell hinzugefügt: - Aktivität - Artefakt - Rolle - eine weitere Aktivität - ein Synchronisationsbalken - eine Verschmelzung - der Endknoten - ein Teilprozess - eine Hilfestellung zu einer Aktivität 2. KoPra wird verlassen. 3. KoPra wird neu gestartet und das vorherige Prozess-

	modell geladen.
Soll-Ausgabe:	Das Hauptfenster wird aktualisiert und das Prozessmodell wird mit den zuvor erstellten Prozesselementen angezeigt.
Ist-Ausgabe:	Sie ist identisch zur Soll-Ausgabe.
Testfall-ID:	03
Vorbedingung:	Der Benutzer ist als Administrator am System angemeldet. Der Benutzer hat ein neues Prozessmodell erstellt.
Anweisungen:	1. Eine Aktivität wird dem Prozessmodell hinzugefügt. 2. Die Eigenschaften: Name, Beschreibung, empfohlene Vorgehen und Aufgabe werden geändert. 3. Ein anderes Prozesselement wird selektiert und anschließend wird die vorher geänderte Aktivität wieder selektiert.
Soll-Ausgabe:	Die Eigenschaften des Prozesselementes entsprechen den Eingaben des Benutzers.
Ist-Ausgabe:	Sie ist identisch zur Soll-Ausgabe.
Testfall-ID:	04
Vorbedingung:	Der Benutzer ist als Administrator am System angemeldet. Der Benutzer hat ein neues Prozessmodell erstellt.
Anweisungen:	1. Eine Aktivität wird dem Prozessmodell hinzugefügt. 2. Ein konsumiertes und produziertes Artefakt werden mit der Aktivität verbunden. 3. Eine Bedingung wird der Aktivität angehängt. 4. Eine weitere Bedingung wird der Aktivität angehängt. 5. Der Benutzer versucht eine weitere Bedingung an die Aktivität anzuhängen.
Soll-Ausgabe:	1. Ein Warnhinweis wird ausgegeben, dass nur eine Vorbedingung und ein Ziel mit einer Aktivität verknüpft werden kann. 2. Mit der Aktivität sind genau eine Vorbedingung und genau ein Ziel verknüpft.
Ist-Ausgabe:	Sie ist identisch zur Soll-Ausgabe.
Testfall-ID:	05
Vorbedingung:	Der Benutzer ist als Administrator am System angemeldet. Der Benutzer hat ein neues Prozessmodell erstellt.

Anweisungen:	1. Eine Aktivität wird dem Prozessmodell hinzugefügt. 2. Eine Hilfestellung wird mit der Aktivität verknüpft. 3. Die Eigenschaften: URL und Beschreibung werden mit Text gefüllt. 4. Eine Datei wird der Hilfestellung angehangen.
Soll-Ausgabe:	1. Die Datei liegt auf dem Server unter der entsprechenden JPOM-ID. 2. Die URL und die Beschreibung der Hilfestellung wurden in der Datenbank hinterlegt.
Ist-Ausgabe:	Sie ist identisch Identisch zur Soll-Ausgabe.
Testfall-ID:	06
Vorbedingung:	Der Benutzer ist als Administrator am System angemeldet. Der Benutzer hat ein neues Prozessmodell erstellt. Der Benutzer hat eine Bedingung mit einer Aktivität verknüpft, die ein Artefakt produziert und konsumiert.
Anweisungen:	1. Der Benutzer gibt eine informale Bedingung zu dem Prozesselement Bedingung an. 2. Der Benutzer gibt eine formale Bedingung zu dem Prozesselement Bedingung an.
Soll-Ausgabe:	Die beiden Eigenschaften formale- und informale Bedingung sind gespeichert. Die formale Bedingung entspricht dem spezifizierten regulären Ausdruck für formale Bedingungen und verletzt nicht die spezifizierte Grammatik.
Ist-Ausgabe:	Sie ist identisch zur Soll-Ausgabe.
Testfall-ID:	07
Vorbedingung:	Der Benutzer ist als Administrator am System angemeldet. Der Benutzer hat ein neues Prozessmodell erstellt. Der Benutzer hat eine Bedingung mit einem Teilprozess verknüpft.
Anweisungen:	1. Der Benutzer gibt eine informale Bedingung zu dem Prozesselement Bedingung an. 2. Der Benutzer gibt eine formale Bedingung zu dem Prozesselement Bedingung an.
Soll-Ausgabe:	1. Die beiden Eigenschaften formale- und informale Bedingung sind gespeichert.

	2. Die formale Bedingung entspricht dem spezifizierten-regulären Ausdruck für formale Bedingungen und verletzt nicht die spezifizierte Grammatik.
Ist-Ausgabe:	Sie ist identisch zur Soll-Ausgabe.
Testfall-ID:	08
Vorbedingung:	Der Benutzer ist als Administrator am System angemeldet. Der Benutzer hat ein neues Prozessmodell erstellt.
Anweisungen:	1. Der Benutzer erstellt zwei Aktivitäten, einen Teilprozess und einen Synchronisationsbalken. 2. Beide Aktivitäten und der Teilprozess führen in den Synchronisationsbalken ein. 3. Zwei weitere Aktivitäten und zwei Teilprozess werden erstellt und mit dem Synchronisationsbalken über einen ausgehenden Pfad verbunden.
Soll-Ausgabe:	Die Prozesselemente lassen sich ohne Fehlermeldungen und sonstigen Komplikationen miteinander verknüpfen.
Ist-Ausgabe:	Sie ist identisch zur Soll-Ausgabe.
Testfall-ID:	09
Vorbedingung:	Der Benutzer ist als Administrator am System angemeldet. Der Benutzer hat ein neues Prozessmodell erstellt.
Anweisungen:	1. Der Benutzer erstellt ein Prozessmodell anhand des Beispiels aus der Structured Analysis (siehe Abbildung 3.2).
Soll-Ausgabe:	Das Prozessmodell entspricht der Abbildung 3.2.
Ist-Ausgabe:	Sie ist identisch zur Soll-Ausgabe.
Testfall-ID:	10
Vorbedingung:	Der Benutzer ist als Administrator am System angemeldet. Der Benutzer hat ein neues Prozessmodell erstellt.
Anweisungen:	1. Der Benutzer baut das Geschäftsprozessmodell des RUPs (siehe Abbildung 3.10) nach.
Soll-Ausgabe:	Das Prozessmodell entspricht der Abbildung 3.10.
Ist-Ausgabe:	Sie ist identisch zur Soll-Ausgabe.

Testfall-ID:	11
Vorbedingung:	Der Benutzer ist als Administrator am System angemeldet. Der Benutzer hat ein neues Prozessmodell erstellt.
Anweisungen:	1. Der Benutzer baut die Beispielmethode aus der JUST-Spezifikation (siehe Abbildung 6.1) nach.
Soll-Ausgabe:	Das Prozessmodell entspricht der Abbildung 6.1.
Ist-Ausgabe:	Sie ist identisch zur Soll-Ausgabe.
Testfall-ID:	12
Vorbedingung:	Der Benutzer ist als Administrator am System angemeldet. Der Benutzer lädt das Prozessmodell aus Testfall 1.
Anweisungen:	1. Der Benutzer versucht das Prozessmodell zu exportieren
Soll-Ausgabe:	Der Export wird mit einer Fehlermeldung, dass ein Endknoten im Prozessmodell fehlt, abgebrochen.
Ist-Ausgabe:	Sie ist identisch zur Soll-Ausgabe.
Testfall-ID:	13
Vorbedingung:	Der Benutzer ist als Administrator am System angemeldet. Der Benutzer lädt das Prozessmodell aus Testfall 2.
Anweisungen:	1. Der Benutzer exportiert das Prozessmodell.
Soll-Ausgabe:	Der Export wird erfolgreich durchgeführt.
Ist-Ausgabe:	Sie ist identisch zur Soll-Ausgabe.
Testfall-ID:	14
Vorbedingung:	Der Benutzer ist als Administrator am System angemeldet. Der Benutzer lädt das Prozessmodell aus Testfall 5.
Anweisungen:	1. Der Benutzer exportiert das Prozessmodell.
Soll-Ausgabe:	Der Export wird erfolgreich durchgeführt. Alle an der Hilfestellung angehängten Dateien wurden mit exportiert.
Ist-Ausgabe:	Sie ist identisch zur Soll-Ausgabe.

Testfall-ID:	15
Vorbedingung:	Der Benutzer ist als Administrator am System angemeldet. Der Benutzer lädt das Prozessmodell aus Testfall 11.
Anweisungen:	1. Der Benutzer exportiert das Prozessmodell.
Soll-Ausgabe:	Der Export wird erfolgreich durchgeführt. Alle an der Hilfestellung angehängten Dateien wurden mit exportiert.
Ist-Ausgabe:	Sie ist identisch zur Soll-Ausgabe.

Literaturverzeichnis

spe 2006
Software Process Engineering Metamodel Specification, Version 1.1, 2006

ANSI/IEEE Std. 1002 1987
IEEE Standard Taxonomy for Software Engineering Standards. (1987)

Apache Software Foundation
http://jakarta.apache.org/. – Online Ressource, Abruf: 08. 06. 2006

Beck 1999
BECK, Kent: *Extreme programming explained*. Addison Wesley, 1999

Beck 2003
BECK, Kent: *Extreme Programming : die revolutionäre Methode für Softwareentwicklung in kleinen Teams*. Addison Wesley, 2003

Benson 2006
BENSON, David: *JGraph and JGraph Layout Pro - User Manual*, 2006

Boehm 1976
BOEHM, Barry: *Software engineering*. IEEE Transactions on computers, C-25, 1976

Booch 1987
BOOCH, Grady: *Software Engineering with Ada*. 2. The Benjamin / Cummings Publishing Company, Inc, 1987

Booch 1994
BOOCH, Grady: *Object-oriented analysis and design with applications*. 2. The Benjamin / Cummings Publishing Company, Inc, 1994

Cooperation 2003
COOPERATION, Rational S.: *Rational Unified Process Help*. 2003

DeMarco 1979
DEMARCO, Tom: *Structured Analysis and System Specification*. Prentice Hall, 1979

Edward 1992
EDWARD, Yourdon: *Moderne Strukturierte Analyse*. Prentice Hall, 1992

Grady 1992
GRADY, Robert: *Practical Software Metrics for Project Management and Process Improvement*. Prentice-Hall, 1992

IEEE Std. 1074 1991
IEEE Standard for Developing Software Life Cycle Processes. (1991)

IEEE Std. 610.12 1990
IEEE Standard Glossary of Software Engineering Terminology. (1990)

Jacobson Ivar 2001
JACOBSON IVAR, Rumbaugh J.: *The unified software development process*. 3. Addison Wesley, 2001

JAVA
http://java.sun.com. – Online Ressource, Abruf: 12. 06. 2006

JAVA Swing
http://java.sun.com/products/jfc/index.jsp. – Online Ressource, Abruf: 12. 06. 2006

JPOM
http://www.jpom.de. – Online Ressource, Abruf: 16. 06. 2006

JUST-Team 2004a
JUST-TEAM: *JUST - Benutzerhandbuch*, 2004

JUST-Team 2004b
JUST-TEAM: *JUST - Entwurf, Version 3.0*, 2004

JUST-Team 2004c
JUST-TEAM: *JUST - Installations- und Administrationshandbuch*, 2004

JUST-Team 2004d
JUST-TEAM: *JUST - Spezifikation, Version 2.26*, 2004

Kaschowitz 2006a
KASCHOWITZ, Marcus: *Erweiterung von JUST: Methodik-Unterstützung zur Anforderungserhebung, Version 0.19*, 2006

Kaschowitz 2006b
KASCHOWITZ, Marcus: *Projektplan Methoden des Requirement Engineerings, Version 1.1*, 2006

Kaschowitz 2006c
KASCHOWITZ, Marcus: *Requirement-Engineering Methodik*. 2006

Kendall 1998
KENDALL, Scott: *UML konzentriert*. Addison-Wesley-Longman, 1998

Kilberth 1989
KILBERTH, Klaus: *Einführung in die Methode des Jackson structured programming (JSP)*. Vieweg, 1989

KoPra-Team 2005a
KOPRA-TEAM: *Studienprojekt KoPra: Entwurf, Version 0.45*, 2005

KoPra-Team 2005b
KOPRA-TEAM: *Studienprojekt KoPra: Spezifikation, Version 3.2*, 2005

Larman 1998
LARMAN, Craig: *Applying UML and patterns : an introduction to object-oriented analysis and design*. Prentice Hall, 1998

Oestereich 2004
OESTEREICH, Bernd: *Analyse und Design mit UML 2: Objektorientierte Softwareentwicklung*. Oldenbourg Wissenschaftsverlag, 2004

Rational Rose
http://www.ibm.com/software/rational. – Online Ressource, Abruf: 16. 06. 2006

Rumbaugh 1993
RUMBAUGH, Premerlani Eddy L.: *Objektorientiertes Modellieren und Entwerfen*. Prentice Hall, 1993

Schönleber 2005
SCHÖNLEBER, Tim: *Entwicklung eines webbasierten Systems zur Erfassung und Bearbeitung von Modification Requests*, 2005

Weinberg 1980
WEINBERG, Victor: *Structured Analysis*. Prentice Hall, 1980

Wikipedia JAVA
http://de.wikipedia.org/wiki/Java. – Online Ressource, Abruf: 12. 06. 2006

Zave 1997
ZAVE, P: *Classification of Research Efforts in Requirements Engineering*. ACM Computing Surveys, 1997

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe.
Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken nach bestem Wissen und Gewissen als solche kenntlich gemacht habe.

Stuttgart, den 20. Juli 2006 _____

(Marcus Kaschowitz)