



Abschlussprüfung Sommer 2015

Fachinformatiker für Anwendungsentwicklung
Dokumentation zur betrieblichen Projektarbeit

RoutingDSL

Entwicklung einer DSL zur Beschreibung von Routingregeln

Abgabetermin: 06.05.2015

Prüfungsbewerber:

Markus Amshove

Straße

Wohnort



Ausbildungsbetrieb:

ALTE OLDENBURGER Krankenversicherung AG

Theodor-Heuss-Straße 96

49377 Vechta

Inhaltsverzeichnis

Abbildungsverzeichnis	IV
Tabellenverzeichnis	V
Listings	VI
1 Einleitung	1
1.1 Projektbeschreibung	1
1.2 Projektziel	1
1.3 Projektumfeld	1
1.4 Projektbegründung	2
1.5 Projektschnittstellen	2
2 Projektplanung	2
2.1 Projektphasen	2
2.2 Ressourcenplanung	3
2.3 Entwicklungsprozess	3
3 Analysephase	4
3.1 Ist-Analyse	4
3.2 Wirtschaftlichkeitsanalyse	4
3.2.1 „Make or Buy“-Entscheidung	5
3.2.2 Projektkosten	5
3.2.3 Amortisationsdauer	5
3.3 Anwendungsfälle	6
3.4 Lastenheft	6
4 Entwurfsphase	7
4.1 Auswahl des DSL -Frameworks	7
4.2 Ermittlung der Aktivitäten	7
4.3 Deployment	8
4.4 Entwurf der DSL	8
4.5 Schnittstelle zu BREPL	9
4.6 Pflichtenheft	9
5 Implementierungsphase	10
5.1 Iterationsplanung	10
5.2 Implementierung der DSL	10
5.3 Darstellung der DSL	11
5.4 Implementierung der Codegenerierung	11
5.5 Implementierung der Schnittstelle	12

6	Abnahme- und Deploymentphase	13
6.1	Abnahme durch den BREPL -Entwickler	13
6.2	Deployment und Einführung	14
7	Dokumentation	14
7.1	Benutzerhandbuch	14
7.2	Entwicklerdokumentation	14
8	Fazit	15
8.1	Soll-/Ist-Vergleich	15
8.2	Lessons Learned	16
8.3	Ausblick	16
	Literaturverzeichnis	17
A	Anhang	i
A.1	Detaillierte Zeitplanung	i
A.2	Verwendete Ressourcen	ii
A.3	Klassendiagramm der derzeit genutzten Eigenschaften	iii
A.4	Beispiel von Regeln mit C# -Code	iv
A.5	Ausschnitt aus dem Regelwerk mit möglicher Umsetzung	v
A.6	Amortisation	v
A.7	Use-Case-Diagramm	vi
A.8	Lastenheft (Auszug)	vii
A.9	Nutzwertanalyse zur Auswahl des DSL -Frameworks	viii
A.10	Aktivitätsdiagramm zum Anlegen einer Regel	ix
A.11	Deploymentdiagramm	x
A.12	Entwurf der DSL	x
A.13	Entwurf der DSL mit Klammern	xi
A.14	Entwurf der Schnittstelle zu BREPL	xii
A.15	Pflichtenheft (Auszug)	xiii
A.16	Iterationsplan	xiv
A.17	Grammatik der DSL in Xtext	xv
A.18	Beispiel der Regeln mit RoutingDSL -Code	xvi
A.19	Screenshots der Entwicklungsumgebung	xvii
A.20	Ausschnitt aus der Codeformatierung in Xtend	xviii
A.21	Xtend -Methoden zur Ermittlung des zugehörigen C# -Operators und Beispiel einer generierten C# -Regel	xix
A.22	Generierter C# -Unittest	xxi
A.23	Ermittlung aller Regeln durch .NET Reflection-API	xxi
A.24	Instanziierung aller ermittelten Regelklassen	xxi
A.25	Ausschnitt aus dem Benutzerhandbuch zur Verdeutlichung des Workflows	xxii

A.26 Ausschnitt aus dem CI-Prozess	xxiii
A.27 Ausschnitt aus der Entwicklerdokumentation	xxiv
A.28 Erstellte Codemetriken	xxv

Abbildungsverzeichnis

1	Generierte Unittests der Regeln	13
2	Klassendiagramm	iii
3	Ausschnitt der Liste der darzustellenden Operationen	v
4	Graphische Darstellung der Amortisation	v
5	Use-Case-Diagramm	vi
6	Aktivitätsdiagramm	ix
7	Deploymentdiagramm	x
8	Entwurf der Schnittstelle zu BREPL	xii
9	Screenshot der Entwicklungsumgebung mit Outline, Validierung, Fehlern und Warnungen	xvii
10	Screenshot der Definition von Code-Templates	xvii
11	Darstellung des Workflows	xxii
12	Fehlgeschlagener Build durch fehlgeschlagenen Regeltest	xxiii
13	Erfolgreicher Build nach Korrektur der Regelreihenfolge	xxiii
14	Auszug aus der Entwicklerdokumentation für Xtend	xxiv
15	Auszug aus der Entwicklerdokumentation für die .NET-Schnittstelle	xxiv
16	Metriken des alten BREPL -Routings	xxv
17	Metriken des neuen BREPL -Routings	xxv

Tabellenverzeichnis

1	Grobe Zeitplanung	3
2	Kostenaufstellung	5
3	Zeiteinsparung	6
4	Soll-/Ist-Vergleich	15

Listings

1	Regeln in Form von C#	iv
2	Entwurf der DSL	x
3	Entwurf der DSL mit Klammern	xi
4	Definition der DSL in Xtext	xv
5	Regeln in Form von RoutingDSL	xvi
6	Codeformatierung in Xtend	xviii
7	Tests der Codeformatierung in Xtend	xviii
8	Ausschnitt aus dem Bedingungsgenerator	xix
9	Beispiel einer generierten Regel	xx
10	Generierte Testmethode zum Test einer Regel	xxi
11	Ermittlung aller Klassen zu einem Attribut	xxi
12	Instanziierung aller ermittelten Regelklassen und aufsteigend nach Reihenfolge sortiert.	xxi

AO	ALTE OLDENBURGER Krankenversicherung AG
AST	Abstract Syntax Tree
BREPL	Beschlagworten Routen Erkennen Prüfen Leisten (Eigenentwicklung)
CI	Continuous Integration
DMS	Dokumentenmanagementsystem
DSL	Domain Specific Language
JVM	Java Virtual Machine
PAM	Professional Archive Manager (DMS der AO)
PK	Provinzial Krankenversicherung Hannover AG

1 Einleitung

Die folgende Projektdokumentation schildert den Ablauf des IHK-Abschlussprojektes, welches der Autor im Rahmen seiner Ausbildung zum Fachinformatiker mit Fachrichtung Anwendungsentwicklung durchgeführt hat. Ausbildungsbetrieb ist die ALTE OLDENBURGER Krankenversicherung AG (AO), eine private Krankenversicherung mit Standort in Vechta. Zur Zeit beschäftigt die AO 228 Mitarbeiter.¹ Zu den Produkten des Unternehmens zählen nicht nur private Krankenvoll- und Pflegeversicherungen, sondern auch Zusatzversicherungen für privat und gesetzlich Versicherte.

1.1 Projektbeschreibung

Die AO erhält täglich Post von Versicherungsmaklern und Kunden, welche je nach Art des Schreibens an Sachbearbeiter oder Abteilungen weitergeleitet werden muss. So werden Neuanträge für Versicherungen an die jeweilige Antragsgruppe und Leistungsabrechnungen, wie z. B. Arztrechnungen, an die Leistungsabteilung weitergeleitet. Dies geschieht durch das System Beschlagworten Routen Erkennen Prüfen Leisten (BREPL), welches anhand von vordefinierten Regeln einen Vorgang im Dokumentenmanagementsystem (DMS) Professional Archive Manager (PAM) anlegt, die Dokumente anhängt und dem zuständigen Sachbearbeiter zuteilt. Die vorhandenen Routingregeln liegen derzeit nur in Form von C#-Programmcode vor. Bei Nachfragen über das Routing von Vorgängen, z.B. warum Vorgang X an Mitarbeiter Y zugewiesen wurde, beim Hinzufügen von Regeln und bei Änderungen an vorhandenen Regeln muss sich derzeit ein Entwickler durch den Programmcode arbeiten. Dieses Vorgehen nimmt viel Zeit in Anspruch, da die Komplexität des Programmcodes durch viele Verzweigungen sehr hoch ist. Aus diesen Gründen soll eine vereinfachte Möglichkeit zur Bearbeitung von Routingregeln im Laufe dieses Projektes erstellt werden.

1.2 Projektziel

Ziel des Projektes ist die Erstellung einer Domain Specific Language (DSL) zur Beschreibung von Routingregeln im innerbetrieblichen Postverkehr. Eine DSL ist eine Programmier- oder auch Spezifikationssprache, welche dafür ausgelegt ist, Probleme einer spezifischen Domäne, und nur dieser, zu lösen.² Hierbei soll es möglich sein, die Regeln in einem Editor textuell zu beschreiben und daraus gültigen C#-Programmcode zu generieren. Der generierte Programmcode muss dann in einer Schnittstelle zu BREPL implementiert werden, damit BREPL das neue Regelwerk konsumieren kann. Außerdem muss künftig für jeden Vorgang protokolliert sein, durch welche Regel dieser geroutet wurde. Durch die Beschreibung der Regeln in Form einer DSL soll die Wartbarkeit des Routings durch das BREPL-System erheblich erhöht und die Fehleranfälligkeit verringert werden.

1.3 Projektumfeld

Auftraggeber des Projektes ist die AO und deren Inputmanagement. Auslöser des Projektes ist der BREPL-Entwickler.

¹Kennzahl zum Stichtag 20.11.2013, vgl. ALTE OLDENBURGER [2013, S. 4].

²Vgl. STEFAN TILKOV [2012].

In jeder Abteilung der **AO** gibt es für das Inputmanagement zuständige Personen, welche die Regeln für das Routing von Vorgängen in ihrer Abteilung vorgeben. Diese Regeln bestimmen, welche Bedingungen erfüllt sein müssen, um Vorgänge in **PAM** an bestimmte Personen oder Gruppen der jeweiligen Abteilung zu routen. Dies ändert sich z. B. bei neuen Aktionen der **AO**, neuen Tarifen oder auch bei Personaländerungen.

Der **BREPL**-Entwickler ist neben der Weiterentwicklung von **BREPL** damit beschäftigt, Routingregeln in das Programm einzuführen und geroutete Vorgänge für den Fachbereich nachzuvollziehen.

1.4 Projektbegründung

Die Hauptschwachstelle des momentanen Regelwerks ist das hohe Maß an manueller Arbeit, welche beim Ändern, Einsehen, Löschen oder Hinzufügen neuer Regeln anfällt. Hierzu muss ein Entwickler den bestehenden C#-Programmcode lesen und die richtige Stelle finden, an der die Regel hineinpasst. Dies ist ein sehr zeitaufwändiges Unterfangen, da eine neue Regel an der falschen Stelle zu einem ungewollten Seiteneffekt, wie z. B. dem Routen aller Vorgänge in ein falsches Postfach, führen kann. Ebenso sind Auswertungen über das Routing und die vorhandenen Regeln derzeit zeitaufwändige Aufgaben, da die Regeln nicht klar voneinander getrennt und identifizierbar sind. Ein weiteres Problem ist, dass es durch die derzeitige Komplexität des Programmcodes und Intransparenz der vorhandenen Regeln nicht möglich ist, eine Aussage darüber zu treffen, ob überhaupt alle vorhandenen Regeln in Kraft treten können. Aufgrund dieser Probleme und manuellen Arbeiten hat sich der **BREPL**-Entwickler dazu entschlossen, die Entwicklung einer Alternativmöglichkeit in Auftrag zu geben.

1.5 Projektschnittstellen

Damit die generierten C#-Regeln genutzt werden können, muss im Laufe des Projektes eine Schnittstelle zu **BREPL** in C# programmiert werden. Im gesamten Inputmanagement der **AO** ist **BREPL** das Kernsystem, welches alle Aufgaben der Aufbereitung von Vorgängen durchführt. Das Routing spielt hierbei eine essenzielle Rolle, da die Mitarbeiter der **AO** sonst keine Vorgänge in **PAM** bekommen und nicht weiter arbeiten können.

2 Projektplanung

In der Projektplanung soll die notwendige Zeit und die benötigten Ressourcen sowie ein Ablauf der Durchführung des Projektes geplant werden.

2.1 Projektphasen

Für die Umsetzung des Projektes standen dem Autor 70 Stunden zur Verfügung. Diese wurden vor Projektbeginn auf verschiedene Phasen verteilt, die während der Softwareentwicklung durchlaufen werden. Eine grobe Zeitplanung sowie die Hauptphasen lassen sich der Tabelle 1: **Grobe Zeitplanung** entnehmen. Außerdem können die einzelnen Hauptphasen noch in kleinere Unterphasen zerlegt werden. Eine detaillierte Übersicht dieser Phasen befindet sich im Anhang **A.1: Detaillierte Zeitplanung** auf S. i.

Projektphase	Geplante Zeit
Analyse	6 h
Entwurf	11 h
Implementierung	39 h
Abnahme und Deployment	5 h
Dokumentation	9 h
Gesamt	70 h

Tabelle 1: Grobe Zeitplanung

2.2 Ressourcenplanung

In der Übersicht, welche sich im Anhang [A.2: Verwendete Ressourcen](#) auf S. ii befindet, sind alle Ressourcen aufgelistet, die für das Projekt eingesetzt wurden. Damit sind sowohl Hard- und Softwareressourcen, als auch das Personal gemeint. Bei der Auswahl der verwendeten Software wurde darauf geachtet, dass diese kostenfrei (z. B. als Open Source) zur Verfügung steht oder die AO bereits Lizenzen für diese besitzt. Dadurch wurden anfallende Projektkosten möglichst gering gehalten.

2.3 Entwicklungsprozess

Bevor mit der Realisierung des Projektes begonnen werden konnte, musste sich der Autor für einen geeigneten Entwicklungsprozess entscheiden. Dieser definiert die Vorgehensweise, nach der die Umsetzung erfolgen soll.

Im Zuge des Projektes entschied sich der Autor für einen agilen Entwicklungsprozess. Bei der agilen Softwareentwicklung geht es darum, möglichst schnell auf sich ändernde Anforderungen reagieren zu können.³ Dies unterscheidet sich insofern von der klassischen Vorgehensweise, da das zu entwickelnde System nicht im Voraus in allen Einzelheiten genau geplant und dann in einem einzelnen langen Durchgang entwickelt wird. Bei der agilen Softwareentwicklung steht ein iterativer Entwicklungsprozess im Mittelpunkt. Die Entwicklung geschieht in kurzen Abschnitten, nach denen jeweils ein Artefakt entsteht, welches dem Kunden gezeigt werden kann. Sollte der Kunde einen Anpassungswunsch haben, kann auf diesen während der nächsten Iteration schnell reagiert werden.⁴ Im Zuge der Erstellung einer DSL kann dieser Entwicklungsprozess gleichzeitig als Schulung im Umgang mit der DSL dienen.

Die gesamte Implementierungsphase wird durch Continuous Integration (CI) in Form eines Buildservers begleitet. Der Buildserver unterstützt hierbei mehrere Aufgaben der Entwicklung. Einerseits wird sichergestellt, dass die Anwendung keine Abhängigkeiten auf andere Anwendungen oder Bibliotheken hat, welche nur auf der Entwicklermaschine vorhanden sind. Dies wird durch das Kompilieren der Anwendung auf dem Buildserver sichergestellt. Andererseits führt der Buildserver auch die automatisierten Tests der Anwendung aus, um auch die Lauffähigkeit auf einem anderen System als dem des Entwicklers zu garantieren. Als letzter Schritt wird auch das Deployment durch den Buildserver sichergestellt. Ist ein Build erfolgreich, werden die resultierenden Artefakte aufbewahrt und für andere

³Vgl. [TURK U. A. \[2014, S.1\]](#).

⁴Vgl. [TURK U. A. \[2014, S.1\]](#).

Programme bereitgestellt. Hierdurch wird das Deployment in [BREPL](#) realisiert, da [BREPL](#) bei jedem neuen Kompilieren das neueste Regelwerk konsumieren kann.

3 Analysephase

Nach der Projektplanung kann die Analyse durchgeführt werden. Diese dient der Ermittlung des Ist-Zustandes. Hierbei wird vor allem auch der wirtschaftliche Aspekt des Projektes betrachtet.

3.1 Ist-Analyse

Wie bereits im Abschnitt [1.1 \(Projektbeschreibung\)](#) erwähnt wurde, erhält die [AO](#) täglich Post von Geschäftspartnern. Für diese Post wird von [BREPL](#) ein neuer Vorgang in [PAM](#) angelegt und entweder einem Mitarbeiter aus dem Fachbereich oder einer ganzen Abteilung zugewiesen. Diese Zuweisung geschieht anhand von Regeln, die auf bestimmte Eigenschaften der eingescannten Dokumente reagieren. Diese Regeln werden vom Fachbereich der [AO](#) definiert und dem [BREPL](#)-Entwickler mitgeteilt, woraufhin dieser den bestehenden [C#](#)-Programmcode in [BREPL](#) anpassen muss. Im Laufe der Analysephase wurde ein Klassendiagramm über die derzeit von Regeln genutzten Eigenschaften zur Ermittlung der Zuweisung erstellt. Das Klassendiagramm ist im Anhang [A.3: Klassendiagramm der derzeit genutzten Eigenschaften](#) auf S. [iii](#) zu finden.

Weiterhin ist ein Auszug aus den derzeit in [C#](#) programmierten Regeln im Anhang [1: Regeln in Form von C#](#) auf S. [iv](#) zu finden. Dieser kleine Ausschnitt des Programmcodes in [C#](#) umfasst insgesamt sechs mögliche Wege, das Dokument zu routen. Zwei Wege sind zwei verschiedene „MUSTERGRUPPE“-Gruppen. Die dritte Möglichkeit ist die Zuweisung an den Benutzer „MUSTERMANN“, falls das [Basiskennzeichen](#) des Vorganges dem des Basistarifes entspricht. Der vierte und fünfte Weg würden je nach Name der versicherten Person den Vorgang an den Benutzer „MUSTERFRAU“ oder „MUSTERMANN“ routen. Die sechste Möglichkeit tritt ein, wenn keine der vorherigen Regeln zutreffen würde.

Während der Ist-Analyse wurde eine Excelliste mit möglichen Regeln angefertigt. Diese gibt an, welche Regeln durch eine [DSL](#) abgebildet werden müssen, und wie man diese umsetzen könnte. Hierbei wurde vor allem ein Augenmerk auf die abzudeckenden Prüfungen gelegt. Im obigen Beispiel muss also geprüft werden, ob der Name mit einem Buchstaben zwischen *A* und *M* oder einem Buchstaben zwischen *N* und *Z* beginnt. Ein Ausschnitt dieser Liste ist im Anhang [A.5: Ausschnitt aus dem Regelwerk mit möglicher Umsetzung](#) auf S. [v](#) zu finden.

3.2 Wirtschaftlichkeitsanalyse

Aufgrund der Probleme des momentanen Prozesses beim Ändern, Löschen und Hinzufügen von Regeln, die in Abschnitt [1.4 \(Projektbegründung\)](#) erläutert wurden, ist die Umsetzung der [DSL](#) erforderlich. Die wirtschaftliche Betrachtung und die Entscheidung, ob die Realisierung des Projektes gerechtfertigt ist, wird in den folgenden Abschnitten getroffen.

3.2.1 „Make or Buy“-Entscheidung

Da es sich bei den zu beschreibenden Regeln und eingehenden Dokumenten um unternehmensspezifische Vorgänge der AO handelt, ist eine Lösung durch ein gekauftes Produkt nicht möglich. Deshalb muss eine Lösung durch die AO entwickelt werden.

3.2.2 Projektkosten

Die Kosten, die während der Entwicklung des Projektes anfallen, werden im Folgenden kalkuliert. Die für die Realisierung des Projektes benötigten Personal- und Ressourcenkosten sind von der Personalabteilung der AO festgelegte Pauschalsätze⁵, da die genauen Stundensätze nicht herausgegeben werden dürfen. Der Stundensatz eines Auszubildenden beträgt demzufolge 10 €, der eines Mitarbeiters 25 €. Die Ressourcennutzung umfasst einen Büroarbeitsplatz, die Hardware- und Softwarenutzung sowie Gemeinkosten für bspw. Strom. Hierfür wurde von der Personalabteilung ein pauschaler Stundensatz von 15 € vorgegeben. Die Kosten, die für die einzelnen Vorgänge des Projektes anfallen, sowie die gesamten Projektkosten lassen sich der Tabelle 2: [Kostenaufstellung](#) entnehmen.

Vorgang	Mitarbeiter	Zeit	Personal ⁶	Ressourcen ⁷	Gesamt
Entwicklungskosten	1 x Auszubildender	70 h	700,00 €	1.050,00 €	1.750,00 €
Fachgespräch	2 x Mitarbeiter, 1x Azubi	3 h	180,00 €	135,00 €	315,00 €
Code-Review	1 x Mitarbeiter	4 h	100,00 €	60,00 €	160,00 €
Abnahme	2 x Mitarbeiter	1 h	50,00 €	30,00 €	80,00 €
Projektkosten gesamt					2.305,00 €

Tabelle 2: Kostenaufstellung

3.2.3 Amortisationsdauer

Nachfolgend wird ermittelt, ab welchem Zeitpunkt sich die Entwicklung der DSL amortisiert hat. Durch die Beschreibung der Regeln in Form einer DSL würde sich das Hinzufügen, Ändern, Löschen und Lesen von Regeln erheblich vereinfachen. Auf Grund der Projektstruktur der DSL ist es möglich, Regeln schnell zu identifizieren und zu bearbeiten. Durch die Versionierung der Regeln ließe sich auch besser nachvollziehen, ab wann welche Regeln in BREPL vorhanden sind und genutzt werden. Außerdem eröffnet die DSL zusätzliche Möglichkeiten zur Bestimmung von Bedingungen, so ist es bspw. möglich, Regeln erst ab einem bestimmten Zeitpunkt oder bis zu einem bestimmten Zeitpunkt gültig zu machen, z. B. wenn Regeln erst ab dem Jahreswechsel gültig sein sollen. Da das Generieren des C#-Regelcodes, das Testen der Nutzbarkeit aller Regeln und die Integration in BREPL durch CI automatisiert ablaufen, fallen die manuellen Arbeitsschritte zum Testen des Routings durch eine DSL weg. Nachfolgend soll nun die Zeiteinsparung tabellarisch ermittelt werden. Die Anzahl an Vorgängen pro Quartal und die Zeit pro Vorgang wurden vom BREPL-Entwickler ermittelt.

⁵Die aufgeführten Stundensätze setzen sich insbesondere aus dem Gehalt und den Sozialaufwendungen des Arbeitgebers zusammen.

⁶Personalkosten pro Vorgang = Anzahl Mitarbeiter · Zeit · Stundensatz.

⁷Ressourcenbeitrag pro Vorgang = Anzahl Mitarbeiter · Zeit · 15 € (Ressourcenbeitrag pro Stunde).

⁸Einsparung pro Quartal = Anzahl pro Quartal · (Zeit (alt) pro Vorgang - Zeit (neu) pro Vorgang).

Vorgang	Anzahl pro Quartal	Zeit (alt) pro Vorgang	Zeit(neu) pro Vorgang	Einsparung pro Quartal ⁸
Routing anpassen	8	40 min	5 min	280 min
Routing nachvollziehen	4	25 min	2 min	92 min
Auswertung erstellen	4	nicht möglich	3 min	-12 min
Zeiteinsparung gesamt pro Quartal				360 min

Tabelle 3: Zeiteinsparung

Berechnung der Amortisationsdauer:

$$\text{StundensatzMitarbeiter} = 25 \text{ €} + 15 \text{ €} = 40 \text{ €} \quad (1)$$

$$360 \frac{\text{min}}{\text{Quartal}} \times 4 \frac{\text{Quartale}}{\text{Jahr}} = 1.440 \frac{\text{min}}{\text{Jahr}} = 24 \frac{\text{h}}{\text{Jahr}} \quad (2)$$

$$24 \frac{\text{h}}{\text{Jahr}} \times \text{StundensatzMitarbeiter} = 960,00 \frac{\text{€}}{\text{Jahr}} \quad (3)$$

$$\frac{2.305,00 \text{ €}}{960,00 \frac{\text{€}}{\text{Jahr}}} = 2,40 \text{ Jahre} \approx 2 \text{ Jahre } 5 \text{ Monate} \quad (4)$$

Zusätzlich wurde die Amortisationsdauer graphisch dargestellt: Anhang [A.6: Amortisation](#) auf S. v.

Anhand der Amortisationsrechnung ergibt sich für das Projekt eine Amortisationsdauer von 2 Jahren und 5 Monaten. Da es sich bei [BREPL](#) um ein Kernsystem der [AO](#) handelt, eine externe Lösung nicht möglich ist, der Vorgang zur Erstellung von Auswertungen derzeit nicht möglich ist und die Amortisationsdauer im Vergleich zum langfristigen Einsatz von [BREPL](#) in der [AO](#) gering ausfällt, entschied die [AO](#) sich dafür, das Projekt durchzuführen.

3.3 Anwendungsfälle

Um eine grobe Übersicht über die abzudeckenden Anwendungsfälle zu erhalten, wurde im Zuge der Analysephase ein Use-Case-Diagramm erstellt. Hierbei wurden die betroffenen Akteure identifiziert und deren Anforderungen an das Projekt ermittelt. Das Use-Case-Diagramm ist im Anhang [A.7: Use-Case-Diagramm](#) auf S. vi zu finden.

3.4 Lastenheft

Am Ende der Analysephase wurde zusammen mit dem [BREPL](#)-Entwickler und dem Fachbereich das Lastenheft erstellt. Das Lastenheft ist anhand der MoSCoW⁹-Priorisierung und aus Sicht der Akteure aus dem Use-Case Diagramm der Analysephase formuliert. Ein Auszug des Lastenheftes ist im Anhang [A.8: Lastenheft \(Auszug\)](#) auf S. vii zu finden.

⁹Vgl. [HAUGHEY \[2014\]](#).

4 Entwurfsphase

Als Folge der Analysephase wurde vor der eigentlichen Implementierung des Projektes eine Entwurfsphase durchgeführt. Hierbei wird entworfen, wie das System später aussehen soll und wie dies technisch umzusetzen ist. Am Ende der Entwurfsphase entsteht das Pflichtenheft, welches den Auftraggebern des Projektes vorgelegt wird.

4.1 Auswahl des DSL-Frameworks

Die DSL soll, wie bereits in Abschnitt 1.2 (Projektziel) und im Anhang A.8: Lastenheft (Auszug) auf S. vii erwähnt wurde, in einem Editor textuell beschrieben werden können. Zur Bestimmung und Programmierung einer DSL kommen verschiedene Frameworks in Frage. Durch eine Internetrecherche kam der Autor auf insgesamt drei in Frage kommende Frameworks. Einerseits kam das Xtext¹⁰-Framework in Frage. Dieses Framework wird bereits für ein anderes Projekt in der AO genutzt. Außerdem standen noch ANTLR.NET¹¹ und MPS¹² zur Auswahl. ANTLR.NET bietet sich auf Grund der direkten Schnittstelle zu .NET und somit C# an, da die Schnittstelle zu BREPL und BREPL selbst in C# entwickelt wurden. MPS wurde auf Grund des dahinter stehenden Unternehmens JetBrains¹³ in die Auswahl mit aufgenommen, da dieses Unternehmen im .NET-Umfeld bekannt ist. Die Auswahl des Frameworks, mit dem die DSL realisiert werden soll, wurde anhand einer Nutzwertanalyse durchgeführt. Im Anhang A.9: Nutzwertanalyse zur Auswahl des DSL-Frameworks auf S. viii sind die einzelnen Kriterien mit ihren jeweiligen Gewichtungen und den Bewertungen für die verschiedenen Frameworks aufgelistet. Für die Bewertung wurden Werte zwischen 0 und 2 verwendet.¹⁴ Die Gewichtung der Nutzwertanalyse wurde vom Autor anhand von Erfahrungen in der Nutzung von betriebsfremden Frameworks in anderen Projekten bestimmt. Nach der Durchführung der Nutzwertanalyse ergab sich für das Xtext-Framework ein Nutzwert von 1,9 Punkten. Dieser ist im Vergleich zu ANTLR.NET (0,9 Punkte) und MPS (1,1 Punkte) am größten. Auf Grund dieser Resultate hat sich der Autor zur Durchführung des Projektes für Xtext entschieden.

4.2 Ermittlung der Aktivitäten

Im Zuge der Entwurfsphase verschaffte sich der Autor einen Überblick über die Aktivitäten, welche sich durch die Analyse der Anwendungsfälle ergaben. Am Beispiel der Aktivität des Erstellens einer Regel wurde im Hinblick auf die spätere Entwicklung der DSL unter Berücksichtigung von CI und Versionierung ein Aktivitätsdiagramm erstellt. Dieses Diagramm ist im Anhang A.10: Aktivitätsdiagramm zum Anlegen einer Regel auf S. ix zu finden und beschreibt den Ablauf, den der Regelentwickler durchlaufen muss, wenn er eine Regel erstellt. Auf Seiten des Entwicklers muss die Regel in Form der RoutingDSL verfasst und dann in Git versioniert werden. Anschließend wird der Buildserver diese Regel kompilieren und daraus C#-Programmcode generieren. Dieser Programmcode wird in das Schnittstellenprojekt kopiert und dort mit BREPL kompiliert.

¹⁰Xtext: Language Development Made Easy - <http://www.eclipse.org/Xtext/>

¹¹ANTLR.NET: ANOther Language Recognition - <http://www.antlr.org>

¹²MPS: Meta Programming System - <https://www.jetbrains.com/mps/>

¹³JetBrains: <https://www.jetbrains.com>

¹⁴Gewichtung: 0 = nicht vorhanden/nicht möglich, 1 = negativ, 2 = positiv.

4.3 Deployment

Beim Deployment der **DSL** soll möglichst viel automatisiert ablaufen. Wie bereits in Abschnitt 4.2 (**Ermittlung der Aktivitäten**) erläutert, spielt der Buildserver hierbei eine zentrale Rolle. Das Kompilieren und Testen von Projekten ist hierbei die Aufgabe des Jenkins¹⁵, welcher als Webapplikation auf dem Buildserver läuft. Jedes Projekt ist hierzu in einen *Job* unterteilt, welcher das Kompilieren und das Testen übernimmt. Wenn ein Job ohne Fehler durchläuft – das heißt, das Projekt kann kompiliert werden und die Tests schlagen nicht fehl – wird das Kompilat in Artifactory¹⁶ für andere Projekte bereitgestellt. Artifactory ist eine Webapplikation zur Verwaltung von Repositories, welche zur Bereitstellung von Kompilaten aus der Java- und .NET-Entwicklung dienen. Im Zuge dieses Projektes wird Artifactory genutzt, um das RoutingDSL-Plugin für eine Eclipse-Instanz bereitzustellen. Dieses Verfahren der Verteilung wird im Anhang A.11: **Deploymentdiagramm** auf S. x verdeutlicht.

4.4 Entwurf der **DSL**

Anhand des in Abschnitt 3.1 (**Ist-Analyse**) angefertigten Klassendiagramms konnte der Autor die extrahierten Regeln in Form einer **DSL** verfassen, um somit schon ein Gefühl für die benötigten Funktionen und den Aufbau der **DSL** zu bekommen. Im Folgenden wird der Entwurf der **DSL** erklärt, der im Anhang 2: **Entwurf der DSL** auf S. x zu finden ist. Großgeschriebene Wörter, welche mit Unterstrichen umschlossen sind, werden als Definition angesehen und sind in den folgenden Auflistungen wiederzufinden. Wörter in blauer Schrift sind Schlüsselwörter der **DSL**. Hierbei hat sich der Autor für **Regelwerk** als oberstes Element entschieden. Dieses soll einem **namespace** entsprechen. Ein **namespace** ist ein Schlüsselwort aus der C#-Programmierung, welches verwendet wird, um einen Bereich zu deklarieren, welcher einen Satz aus verknüpften .NET-Elementen enthält.¹⁷ Dies wird hier benutzt, um verschiedene **Regeln** zu organisieren und in einem **Regelwerk** zu kombinieren.

Ein **Regelwerk** muss durch folgende Eigenschaften identifiziert werden:

1. **Name**: Name des **Regelwerkes**, z. B. AoVertrag.
2. **Unternehmen**: Zielunternehmen dieses **Regelwerkes**, z. B. **AO** oder **PK**.
3. **Stapelkategorie**: Art des Vorganges, z. B. Vertrag oder Leistung.
4. **Clearingzuweisung**: Das Postfach, in welches Vorgänge geroutet werden, die keinen **Regeln** entsprechen. Dies könnten z. B. Anträge zu Versicherungen sein, welche nichts mit der Krankenversicherung zu tun haben.

Ein **Regelwerk** kann ein oder mehrere **Regeln** enthalten. Diese **Regeln** müssen durch einen Text beschrieben werden, um das Nachvollziehen der gerouteten Vorgänge anhand des Loggings möglichst einfach zu gestalten. Außerdem müssen **Regeln** eine Zuweisung an eine Gruppe, Abteilung oder einen Benutzer der **AO** enthalten. Einige der extrahierten **Regeln** machten es auch notwendig, beim Routen der Vorgänge bestimmte Eigenschaften zu befüllen, um z. B. den Langtext des **IERmittelterVorgang** zu setzen. Durch diese Erkenntnisse ergab sich folgender Aufbau einer **Regel**:

¹⁵Jenkins: Open Source Continuous Integration Server - <http://jenkins-ci.org>

¹⁶Artifactory: Binary Repository Manager - <http://jfrog.com/artifactory/>

¹⁷Vgl. MICROSOFT [2013].

1. **Beschreibung:** Eine möglichst genaue Beschreibung der **Regel**, um diese durch das Logging identifizieren zu können, z. B. „Sachbearbeiter im Betreff angegeben“.
2. **Zuweisung:** Routing des Vorganges, wenn diese Regel in Kraft tritt, z. B. an die Gruppe Leistungsabteilung.
3. **Bedingungen:** Bedingungen, die erfüllt sein müssen, z. B. die Postleitzahl auf einen Wert prüfen.
4. **Wertzuweisungen:** Zu setzende Eigenschaften am Vorgang, z. B. einen Langtext zur Beschreibung der weiteren Verarbeitung.

Beim Beschreiben der ersten extrahierten **Regeln** in Form der entworfenen **DSL** wurde schnell klar, dass viele **Regeln** redundante Bedingungen besitzen. So wurde bspw. an fünf **Regeln** dieselbe Bedingung gestellt. Um diese Redundanzen zu vermeiden, wurde die **Klammer** eingeführt, welche Bedingungen an die sich in ihr befindenen **Regeln** vererbt. Der Entwurf mit der **Klammer** befindet sich im Anhang 3: [Entwurf der DSL mit Klammern](#) auf S. xi. Die **Klammern** sind folgendermaßen aufgebaut:

1. **Klammername:** Der Name der **Klammer**, damit dieser im Logging festgehalten werden kann, z. B. „Basistarife Auskuenfte“.
2. **Bedingungen:** Die Bedingungen, welche an die untergeordneten **Regeln** weitervererbt werden sollen, z. B. **DokumentenTyp** „BTAUSKUENFTE“.

4.5 Schnittstelle zu **BREPL**

BREPL erwartet einen **Router** mit einem **IErmittelterVorgang** als Parameter aufrufen zu können und als Rückgabetypen einen **IGerouteterVorgang** zu erhalten. Aus diesem Grund entwarf der Autor eine Klasse namens **Router**, welche dieser Schnittstelle durch die Methode **SetzeZuweisung()** gerecht wird. Innerhalb der Schnittstelle sollen die Regeln durch eine abstrakte Klasse **Regel** abgebildet werden. Diese Klasse definiert zwei abstrakte Eigenschaften namens **Reihenfolge**, welche die Sortierung der Priorität ermöglichen soll, und die **RegelBeschreibung**, welche der Beschreibung der **Regel** in **DSL**-Form entspricht. Zusätzlich definiert **Regel** noch eine Eigenschaft **Log**, welche nicht überschrieben werden kann und das benötigte Logging ermöglichen soll. Zur Ermittlung der möglichen **Regeln**, welche für einen Vorgang zutreffen können, wurde noch ein Interface namens **IRegelProvider** entworfen. Der Entwurf der Schnittstelle ist im Anhang [A.14: Entwurf der Schnittstelle zu BREPL](#) auf S. xii als Klassendiagramm zu finden.

4.6 Pflichtenheft

Anhand der Entwürfe wurde am Ende der Entwurfsphase ein Pflichtenheft erstellt. Hierbei wird die konkrete Umsetzung der im Abschnitt [3.4 \(Lastenheft\)](#) ermittelten Anforderungen erfasst. Hiermit kann am Ende des Projektes überprüft werden, ob alle Anforderungen an die Anwendung abgedeckt und ob diese auch wie vereinbart umgesetzt wurden. Ein Auszug aus dem Pflichtenheft ist im Anhang [A.15: Pflichtenheft \(Auszug\)](#) auf S. xiii zu finden.

5 Implementierungsphase

Anhand des erstellten Pflichtenheftes konnte der Autor mit der Implementierung des Projektes beginnen.

5.1 Iterationsplanung

Zu Anfang der Implementierungsphase wurde ein Iterationsplan erstellt. Der Iterationsplan soll einen gegliederten Ablauf der einzelnen zu erfüllenden Aufgaben während der Implementierungsphase darstellen. Dieser Plan dient dem Entwickler als Orientierung während der Entwicklung. So kann man nachvollziehen, wo man sich im Projekt befindet, welcher Teil des Workflows schon abgeschlossen ist oder noch vor einem liegt. Der Iterationsplan ist im Anhang [A.16: Iterationsplan](#) auf S. [xiv](#) zu finden.

5.2 Implementierung der DSL

Zur Implementierung der DSL wurde, wie im Abschnitt [4.1 \(Auswahl des DSL-Frameworks\)](#) ermittelt, Xtext verwendet, welches neben der Definition einer DSL auch die benötigten Klassen zum Darstellen dieser generiert. Die Grammatik ist eine Zusammenstellung aus Regeln, welche die Form der gültigen Elemente gemäß der Sprachsyntax beschreibt.¹⁸ Nach der Definition der Grammatik generiert Xtext einen *Lexer* und einen *Parser* in Java.

Eine Programmiersprache, und somit auch eine DSL, wird in sogenannte Tokens heruntergebrochen. Diese Tokens können entweder ein *Schlüsselwort* (z. B. `class` in Java), ein *Bezeichner* (z. B. der Klassenname) oder sogenannte *Symbolnamen* (z. B. Variablennamen) sein. Der Prozess der Konvertierung einer geschriebenen Sprache in eine Sequenz aus Tokens wird *lexikalische Analyse* genannt und durch den Lexer durchgeführt.¹⁹ Anhand der Sequenz an Tokens kann nun durch den Parser die *syntaktische Analyse* durchgeführt werden. Hierbei wird sichergestellt, dass die Sequenzen aus Tokens gültige Anweisungen der DSL ergeben. Nach der *syntaktischen Analyse* wird die *semantische Analyse* durch den Parser durchgeführt. Semantische Kontrollen beinhalten z. B. das Sicherstellen von Typsicherheit, sofern diese in der DSL gewünscht ist. Auch das Verwalten einer *Symboltabelle* ist die Aufgabe eines Parsers. Die Symboltabelle verwaltet bspw. die Sichtbarkeit von Variablen in bestimmten Teilen eines Programmes. Zur Unterstützung des Parsers wird beim Parsen der Sprache der *Abstract Syntax Tree (AST)* gebaut. Der AST ist die Darstellung der abstrakten Syntaxstrukturen des Programmes in Form eines Baums. In diesem Baum stellt jedes Element ein Konstrukt des Programmes dar. Der AST wird im Speicher gehalten und bewirkt, dass der Parser die Tokens nicht ständig neu parsen muss, um neue semantische Kontrollen durchzuführen.²⁰ Xtext generiert neben dem Lexer und Parser für jedes Grammatikelement der DSL eine Darstellung in Form einer Java-Klasse. Nach dem Parsen der DSL mit Xtext werden alle Elemente des AST in Form von Objekten in Java instanziiert.

Zur Definition einer DSL in Xtext wird die Grammatik textuell erfasst und beginnt immer mit einem Oberelement. Als Oberelement in der RoutingDSL wurde die `RoutingDatei` gewählt, welche entweder ein `Regelwerk` oder aber die `Konfiguration` für den Router enthält. Ein `Regelwerk`

¹⁸Vgl. BETTINI [2013, S. 11].

¹⁹Vgl. BETTINI [2013, S. 10].

²⁰Vgl. BETTINI [2013, S. 12].

muss ein Unternehmen (AO oder PK), eine Stapelkategorie (Vertrag oder Leistung), sowie das Clearingpostfach enthalten. Das Regelwerk bietet nach der Deklaration dieser Pflichtangaben die Möglichkeit, RegelwerkElemente zu definieren. Ein RegelwerkElement ist entweder eine Regel oder eine Klammer. Durch eine Klammer hat der Regelentwickler die Möglichkeit, Bedingungen zu definieren, welche auf Regeln innerhalb der Klammer vererbt werden. Hierdurch werden Redundanzen in der Beschreibung der Regeln verhindert und die Lesbarkeit der DSL erhalten. Die Konfiguration wird benötigt, um eine zentrale Stelle für Konfigurationen im Routing zu schaffen. Hier wird z. B. definiert, welche Postleitzahlen zu einem bestimmten Unternehmen gehören. Sollte sich dies ändern, muss auch die Codegenerierung der DSL die Änderungen mitbekommen. Ein Ausschnitt der in Xtext beschriebenen Grammatik ist im Anhang 4: Definition der DSL in Xtext auf S. xv zu finden.

Am Ende der Implementierung der DSL wurden alle 146 Regeln in die Form der RoutingDSL übertragen. Stellvertretend ist das im Abschnitt 3.1 (Ist-Analyse) genannte Beispiel von Regeln in C# in Form der RoutingDSL im Anhang A.18: Beispiel der Regeln mit RoutingDSL-Code auf S. xvi zu finden.

5.3 Darstellung der DSL

Zur Darstellung und Entwicklung der DSL bietet sich Eclipse an, da Xtext die Möglichkeit bietet, ein Plugin für Eclipse zu bauen. Durch das Plugin werden z. B. Codevervollständigung und -unterstützung, Syntaxhighlighting und automatisiertes Kompilieren der DSL angeboten. Mit Hilfe der geparsten Elemente ist es möglich, die Unterstützung in Eclipse auszubauen und zu konfigurieren. Im Laufe der Implementierungsphase sorgte der Autor dafür, den Regelentwicklern ein möglichst gutes Erlebnis und viel Unterstützung bei der Entwicklung zu bieten. Hierzu wurde bspw. die Outline von Eclipse durch eine sprechendere Darstellung mit aussagekräftigen Symbolen erweitert. Auch die Eingabevalidierung wird durch sprechende Fehler und Compilerwarnungen ausgedrückt. Weiterhin ist es möglich, den von Eclipse bereitgestellten Code-Formatter zu nutzen, um den DSL-Code nach selbstdefinierten Regeln formatieren zu lassen. Auch Code-Templates werden unterstützt und können im Editor genutzt werden. Diese Beispiele sind im Anhang A.19: Screenshots der Entwicklungsumgebung auf S. xvii demonstriert. Außerdem ist ein Ausschnitt der Codeformatierung mit zugehörigen Tests im Anhang A.20: Ausschnitt aus der Codeformatierung in Xtend auf S. xviii zu finden.

5.4 Implementierung der Codegenerierung

Die generierten Parser und Lexer des Xtext-Frameworks konsumieren die geschriebene DSL und bauen daraus JVM-Artefakte in Form von Java-Klassen. Anhand der geparsten Artefakte kann nun mit Hilfe von Xtend Code generiert werden. Xtend bietet die Möglichkeit, innerhalb von Stringliterals die üblichen Kontrollstrukturen von Programmiersprachen zu nutzen. Diese Literale sind in Xtend gesondert durch drei Apostrophe zu umschließen. Kontrollstrukturen sind mit « und » zu umschließen. Xtend bietet auch die Möglichkeit, sogenannte Erweiterungsmethoden zu definieren. Erweiterungsmethoden können vorhandenen Klassen weitere Methoden hinzufügen, ohne von diesen Klassen ableiten zu müssen. Erweiterungsmethoden sind nötig, da die Klassen, welche die DSL abbilden, generiert sind und somit jegliche Änderungen an diesen beim nächsten Kompilieren überschrieben würden.

Die Schwierigkeit bei der Codegenerierung liegt in den vielen verschiedenen Möglichkeiten, die [DSL](#) zu schreiben. Die RoutingDSL bietet an bestimmten Positionen bis zu 11 verschiedene Möglichkeiten an Elementen, welche eingegeben werden können. All diese Elemente müssen bei der Codegenerierung beachtet und abgedeckt werden. Für die Codegenerierung wurden für möglichst kleine Aufgaben jeweils eigene Klassen definiert, um diese gesondert testen zu können. Die Codegenerierung ist Teil des RoutingDSL-Eclipse. Hier wird beim Speichern der [DSL](#) automatisch der C#-Code generiert. Für [CI](#) wurde ein gesondertes Konsolenprogramm zur Codegenerierung entwickelt.

Die Codegenerierung unterteilt sich in 4 verschiedene Teilgeneratoren:

1. **UnternehmensAttributGenerator**: Generiert das Klassenattribut aus **Unternehmen** und **StapelKategorie** zu einem **Regelwerk**.
2. **RegelGenerator**: Generiert alle **Regeln** innerhalb eines **Regelwerkes**.
3. **UnittestGenerator**: Generiert Unittests zu allen **Regeln** innerhalb eines **Regelwerkes**.
4. **RoutingKonfigurationsGenerator**: Generiert eine Konfigurationsklasse zur zentralen **Konfiguration** des Routers.

Jedes Regelwerk durchläuft die ersten drei Generatoren und generiert dabei für jede Regel eine C#-Klasse, eine C#-Testklasse mit Testfällen zu allen Regeln und eine Attributsklasse, um die Regeln in C# diesem Attribut zuzuordnen. Der **RoutingKonfigurationsGenerator** wird gesondert für das **Konfigurationselement** aufgerufen. Der **RegelGenerator** ermittelt alle Bedingungen und Aktionen zu einer **Regel** und übersetzt diese in C#. Hierbei ist zu beachten, dass **Regeln** auch innerhalb von **Klammern** definiert sein können, wodurch eine Vererbungshierarchie entsteht, welche nach Bedingungen und Aktionen ausgewertet und der eigentlichen **Regel** hinzugefügt werden muss. Ein Ausschnitt aus der Codegenerierung anhand des **RegelGenerators** in Xtend und eine damit generierte **Regel** in C# ist im Anhang [A.21: Xtend-Methoden zur Ermittlung des zugehörigen C#-Operators und Beispiel einer generierten C#-Regel](#) auf S. [xix](#) zu finden. Der daraus in C# generierte Unittest ist im Anhang [10: Generierte Testmethode zum Test einer Regel](#) auf S. [xxi](#) zu finden.

5.5 Implementierung der Schnittstelle

Die Schnittstelle zu [BREPL](#) wurde in C# entwickelt. Hierbei musste beachtet werden, dass die generierten Regelklassen zum Zeitpunkt des Builds erkannt und mitkompiliert werden müssen. Außerdem müssen die Klassen zur Laufzeit instanziiert und verwendet werden können. Um die Regelklassen zum Zeitpunkt des Builds zu generieren, wurde der RoutingDSL-Compiler in Form eines Kommandozeilentools in den [BREPL](#)-Buildprozess integriert. Die hierbei generierten Klassen werden in die entsprechenden Ordner im [BREPL](#)-Projekt kopiert und während des Builds kompiliert. Zur Instanziierung der Klassen wurde die Reflection-API des .NET-Frameworks genutzt, um die kompilierte DLL der Schnittstelle auf Klassen, welche von der abstrakten Klasse **Regel** erben, zu durchsuchen und diese zu instanziierten. Zur Veranschaulichung wurde diese Methode im Anhang [A.23: Ermittlung aller Regeln durch .NET Reflection-API](#) auf S. [xxi](#) beigelegt. Hierbei werden die Klassen anhand der Zugehörigkeit zum Unternehmen und der Stapelkategorie gefiltert und der Priorität nach sortiert. Die

Zugehörigkeit wird während der Codegenerierung durch Attribute an den Regelklassen definiert. Das Filtern und Instanzieren der Klassen geschieht bei der Instanziierung der `Router`-Klasse, welche die Schnittstelle zwischen dem `BREPL`-Projekt und dem `BREPL`-Routing-Projekt bildet. Diese Klasse wird beim Starten von `BREPL` instanziiert, das heißt die Regelobjekte werden beim Start instanziiert und haben dieselbe Lebenszeit wie `BREPL` selbst. Die Methode zum Instanzieren und Sortieren der Regelklassen ist im Anhang A.24: [Instanziierung aller ermittelten Regelklassen](#) auf S. xxi zu finden.

Auf Seite des Entwicklers konnten nach Implementierung des Routing-Projektes die generierten Unittests und die Integrationstests von `BREPL` ausgeführt und als erste Rückmeldung auf die Funktionalität gewertet werden. Zu der Zeit gab es insgesamt 146 Regeln, welche durch das Projekt als gültige Routingregeln erkannt werden mussten. Durch die generierten Unittests ist aufgefallen, dass eine Regel nicht routbar war. Dies ist auf Abbildung 1 zu sehen. Der generierte Unittest ist im Anhang 10: [Generierte Testmethode zum Test einer Regel](#) auf S. xxi zu finden.

```
Tests run: 437, Errors: 0, Failures: 1, Inconclusive: 0, Time: 4,0624062 seconds
Not run: 0, Invalid: 0, Ignored: 0, Skipped: 0

Errors and Failures:
1) Test Failure : BREPL.Routing.Tests.Regelwerke.PkVertragSollte.Regel47RoutenKoennen
    Expected string length 51 but was 44. Strings differ at index 10.
    Expected: "PkVertrag.PT-Dynamisierung Emden im Bereich A bis M"
    But was:  "PkVertrag.Emden Regionaldirektion 598 TEAM 4"
    -----^
```

Abbildung 1: Generierte Unittests der Regeln

Durch das Ändern der Reihenfolge in der Definition der Regeln konnte dieser Fehler schnell gefunden und behoben werden, wodurch alle Unittests erfolgreich ausgeführt wurden. Auch die Integrationstests von `BREPL` waren erfolgreich.

Durch die Implementierung der `DSL` hat sich ein neuer Workflow in der Entwicklung von Routingregeln ergeben. Der Workflow wurde visualisiert und der Benutzerdokumentation, siehe Kapitel 7.1, hinzugefügt und ist im Anhang A.25: [Ausschnitt aus dem Benutzerhandbuch zur Verdeutlichung des Workflows](#) auf S. xxii zu finden.

6 Abnahme- und Deploymentphase

Während der Abnahme- und Deploymentphase wurde das Projekt für die Abnahme in die `BREPL`-Testumgebung eingeführt und später nach erfolgreicher Testphase in die Produktion aufgenommen.

6.1 Abnahme durch den `BREPL`-Entwickler

Wie bereits erwähnt war die erste Phase die Abnahme. Hierzu wurde das Projekt auf die `BREPL`-Testumgebung deployt und mit echten Daten aus ausgewählten Tagen der produktiven Umgebung getestet. Das Hinzufügen des zusätzlichen Datenbankfeldes für das Routinglog wurde durch den Datenbankadministrator durchgeführt. Bei der Abnahme wurde das Augenmerk auf die Praktikabilität der `DSL` gegenüber der vorherigen Lösung gelegt. Hierbei wurde beispielsweise geprüft, wie lange es dauert, ein Routing nachvollziehen zu können. Diese Faktoren wurden von den betroffenen Personen

sehr positiv bewertet, da das zusätzliche Datenbankfeld für das Log und die eindeutige Beschreibung der genutzten Regel schnell auf die Bedingungen dieser Regeln schließen lassen. Auch das Anlegen, Ändern und Löschen von Regeln wurde sehr positiv empfunden, da durch die generierten Unittests eine Prüfung aller Regeln vorhanden ist und bei Fehlern, z. B. einer falschen Reihenfolge, diese sehr schnell behoben werden können. Auch die Integration in den Buildprozess von **BREPL** wurde vom Entwickler gut bewertet, da dies vollständig automatisiert abläuft und keine manuelle Arbeit mehr erforderlich ist.

6.2 Deployment und Einführung

Das Deployment und die Einführung auf der Produktion lief auf Grund der Änderung im Buildprozess sehr einfach. Hierzu musste in Git der Entwicklungsbranch in den Produktivbranch überführt und durch den Buildserver einmal gebaut und getestet werden. Hierbei werden alle Unittests und Integrationstests von **BREPL** einmal ausgeführt, um die Funktionalität gewährleisten zu können. Schlagen die Unittests auf dem Buildserver fehl, kann **BREPL** nicht in der jeweiligen Umgebung eingeführt werden. Nachdem dieser Build und die zugehörigen Tests erfolgreich verliefen, wurde der **BREPL**-Service angehalten und aktualisiert. Der tägliche Postverkehr wird nun durch das Routing aus der RoutingDSL gesteuert und der Workflow für das Verwalten von Routing-Regeln hat sich vereinfacht. Screenshots über den **CI**-Prozess befinden sich im Anhang [A.26: Ausschnitt aus dem CI-Prozess](#) auf S. xxiii.

7 Dokumentation

Für den Einsatz und den Abschluss des Projektes wurde jeweils eine Dokumentation für den Endanwender (Benutzerhandbuch) und eine Dokumentation für Entwickler geschrieben. Diese Dokumentationen werden im Folgenden erläutert.

7.1 Benutzerhandbuch

Für den Regelentwickler wurde eine Dokumentation geschrieben, um diesem einen Einstieg in die Entwicklung von Routingregeln zu gewährleisten. Angefangen wurde mit einer Beschreibung, wie die Arbeitsstation aussehen muss. Dies beschreibt dem Anwender, welche Programme benötigt werden und wie diese konfiguriert werden müssen. Außerdem liegt eine visualisierte Beschreibung zum Workflow der Entwicklung bei (siehe Anhang [A.25: Ausschnitt aus dem Benutzerhandbuch zur Verdeutlichung des Workflows](#) auf S. xxii). Die Dokumentation beinhaltet außerdem den Einstieg in die Entwicklung, also wie man konkret ein **Regelwerk** mit **Regeln** definiert. Die einzelnen Elemente und Syntaxregeln sind in einem Handbuch verfasst und dienen als Nachschlagewerk bei der Entwicklung von Routingregeln.

7.2 Entwicklerdokumentation

Für die Entwicklerdokumentation wurden der Programmcode der **DSL**, der Programmcode der Codegenerierung und der Programmcode der **C#**-Schnittstelle dokumentiert. Die Dokumentation erfolgt direkt am Programmcode mit Hilfe von Javadoc²¹ auf Seiten der Codegenerierung und **DSL**. Auf Seiten der **C#**-Schnittstelle wurde mit Hilfe von XMLDOC²² der Programmcode direkt dokumentiert.

²¹Javadoc: <http://www.oracle.com/technetwork/articles/java/index-jsp-135444.html>

²²XMLDOC: XML Documentation Comments - <https://msdn.microsoft.com/en-us/library/b2s063f7.aspx>

Diese Verfahren wurden gewählt, um dem Entwickler ein Nachschlagewerk zu bieten und mit Hilfe der Entwicklungsumgebungen die Dokumentationen direkt einzusehen. Ein Ausschnitt aus der Entwicklerdokumentation ist im Anhang [A.27: Ausschnitt aus der Entwicklerdokumentation](#) auf S. xxiv zu finden.

8 Fazit

Zum Abschluss des Projektes zieht der Autor ein Fazit über das Gelernte und gibt einen Ausblick auf die Zukunft des Projektes und die Auswirkung in der [AO](#).

8.1 Soll-/Ist-Vergleich

Bei der rückblickenden Betrachtung des Projektes kann festgestellt werden, dass alle vorher definierten Anforderungen gemäß Pflichtenheft erfüllt wurden. Der zu Beginn des Projektes im Abschnitt [2.1 \(Projektphasen\)](#) erstellte Projektplan konnte eingehalten werden. In [Tabelle 4: Soll-/Ist-Vergleich](#) wird die tatsächlich benötigte Zeit mit der vorher eingeplanten Zeit verglichen. Durch die direkte Integration in [BREPL](#) und die Umstellung im [CI-Prozess](#) konnte Zeit beim Deployment gespart werden. Diese gesparte Zeit musste dafür bei der Implementierung der Schnittstelle aufgewendet werden.

Projektphase	Soll	Ist	Differenz
Analyse	6 h	6 h	0 h
Entwurf	11 h	11 h	0 h
Implementierung	39 h	41 h	+ 2 h
Abnahme und Deployment	5 h	3 h	-2 h
Erstellen der Dokumentation	9 h	9 h	0 h
Gesamt	70 h	70 h	0 h

Tabelle 4: Soll-/Ist-Vergleich

Außerdem wurde ein Vorher-Nachher-Vergleich auf Code-Ebene erstellt. Hierzu wurde mit SourceMonitor²³ eine statische Codeanalyse des vorherigen und des neuen Routingprojekts durchgeführt. Hierbei liegt das Augenmerk auf der durchschnittlichen sowie maximalen Komplexität und Tiefe des Programmcodes. Die Komplexität beschreibt die Anzahl der Verzweigungen innerhalb der Methoden, kontrolliert z. B. durch `if/else` und `switch` Anweisungen. Die Tiefe beschreibt die Verschachtelung innerhalb der Klassen. Hierbei fällt auf, dass die durchschnittliche Komplexität innerhalb des Routings mit der [DSL](#) insgesamt 1.0 beträgt, also eine geradlinige Ausführung beschreibt. Insgesamt fällt also die Komplexität des generierten Codes durch den Verzicht auf Verzweigungen im Vergleich zum vorherigen Projekt sehr gering aus. Beispielsweise wurde die durchschnittliche Komplexität auf fast ein Fünftel des vorherigen Wertes verringert. Somit wurde außer den eigentlichen Anforderungen auch das ursprüngliche Problem von komplexem Programmcode behoben. Die Metriken sind im Anhang [A.28: Erstellte Codemetriken](#) auf S. xxv zu finden.

²³SourceMonitor: <http://www.campwoodsw.com/sourcemonitor.html>

8.2 Lessons Learned

Im Zuge des Projektes konnte der Autor viele Erfahrungen über die Arbeit an einem vollständigen Projekt sammeln. Hierbei wurde deutlich, dass großer Wert auf die Analyse und den Entwurf eines Projektes gelegt werden muss, die Kommunikation mit dem Endbenutzer dann jedoch nicht abbrechen darf. Es ist von großem Vorteil, stetiges Feedback zu bekommen und sich ändernde Anforderungen schnell zu identifizieren und umzusetzen. Durch das Projekt wurden auch fachliche Kompetenzen erworben. So konnte der Autor viele Erkenntnisse zur Erstellung von Programmiersprachen und Compilern erwerben. Die Einführung eines Projektes in ein bestehendes System erwies sich als spannendes und lehrreiches Unterfangen, da das Einhalten der Schnittstellen hierbei oberste Priorität hat.

8.3 Ausblick

Für den Autor, aber auch die ALTE OLDENBURGER, ist es eine große Erkenntnis, welchen Nutzen man aus einer [DSL](#) ziehen kann. Daher ist es denkbar, dass in Zukunft weitere Projekte, welche eine [DSL](#) umfassen, entstehen, um den Nutzen dieser in andere Bereiche der Entwicklung einfließen zu lassen.

Literaturverzeichnis

ALTE OLDENBURGER 2013

ALTE OLDENBURGER: *Geschäftsbericht 2013*. http://www.alte-oldenburger.de/web/export/sites/aob/_resources/download_galerien/downloads_pdf/daten_und_fakten/Geschaeftsbericht_2013_ALTE_OLDENBURGER_Krankenversicherung_AG.pdf. Version: 2013. – Datum des Abrufs: 17. März 2015

Bettini 2013

BETTINI, Lorenzo: *Implementing Domain-Specific Languages with Xtext and Xtend*. Packt Publishing, 2013. – ISBN 978-1-78216-030-4

Haughey 2014

HAUGHEY, Duncan: *MoSCoW Method*. <http://www.projectsmart.co.uk/moscow-method.php>. Version: 2014. – Datum des Abrufs: 16. März 2015

Microsoft 2013

MICROSOFT: *Namespace (C#-Referenz)*. <https://msdn.microsoft.com/de-de/library/z2kcy19k.aspx>. Version: 2013. – Datum des Abrufs: 30. März 2015

Stefan Tilkov 2012

STEFAN TILKOV: *Domain Specific Languages*. <https://www.innoq.com/de/articles/2012/07/domain-specific-languages/>. Version: 2012. – Datum des Abrufs: 16. März 2015

Turk u. a. 2014

TURK, Dan ; FRANCE, Robert ; RUMPE, Bernhard: Assumptions underlying agile software development processes. In: *arXiv preprint arXiv:1409.6610* (2014)

A Anhang

A.1 Detaillierte Zeitplanung

Analyse	6 h
1. Ist-Analyse durchführen (extrahieren vorhandener Regeln aus BREPL -Code)	2 h
2. Wirtschaftlichkeitsprüfung und Amortisationsrechnung des Projektes durchführen	1 h
3. Unterstützung des Fachbereiches bei der Erstellung des Lastenheftes	3 h
Entwurf	11 h
1. Recherche von DSL -Frameworks	2 h
2. Nutzwertanalyse zur Auswahl des DSL -Frameworks erstellen	1 h
3. Aktivitätsdiagramm zum Erstellen einer Regel erstellen	1 h
4. Schnittstelle zu BREPL entwerfen	3 h
5. Deploymentdiagramm erstellen	1 h
6. Pflichtenheft erstellen	3 h
Implementierung	39 h
1. Implementierung der BREPL -Schnittstelle mit Tests	10 h
1.1. Implementierung von Schnittstellen und Basisklassen	3 h
1.2. Implementierung der Routingklassen	7 h
2. Implementierung der DSL mit Tests	18 h
2.1. Implementierung der Syntax	5 h
2.2. Implementierung der Grammatik	5 h
2.3. Implementierung der Eingabevalidierung	3 h
2.4. Umsetzung der DSL	5 h
3. Implementierung der Codegenerierung mit Tests	10 h
3.1. Implementierung der Codegenerierung für Regelklassen	4 h
3.2. Implementierung der Codegenerierung für Tests der Regelklassen	4 h
3.3. Implementierung eines Konsolenprogramms zum Aufruf der Codegenerierung durch den Buildserver	2 h
4. Integration der Schnittstelle in BREPL	1 h
Deployment	5 h
1. Bereitstellung und Konfiguration von Eclipse	1 h
2. Continuous Integration (Einrichten von automatischen Builds)	2 h
3. Deployment des Eclipse-Plugins	1 h
4. Deployment der C#-Schnittstelle	1 h
Dokumentation	9 h
1. Erstellen der Projektdokumentation	7 h
2. Erstellen der Entwicklerdokumentation	1 h
3. Erstellen der Benutzerdokumentation	1 h
Gesamt	70 h

A.2 Verwendete Ressourcen

Hardware

- Büroarbeitsplatz mit Thin-Client

Software

- Windows 7 Enterprise mit Service Pack 1 – Betriebssystem
- Eclipse Luna DSL Tools 4.4 – Entwicklungsumgebung Xtend und Xtext
- Visual Studio Professional 2013 – Entwicklungsumgebung C#
- Enterprise Architect – Programm zum Erstellen verschiedener Modelle und Diagramme
- SourceMonitor – Programm zur statischen Codeanalyse
- Git – Verteilte Versionsverwaltung
- NUnit – Framework zur Durchführung von Unit-Tests
- Moq – Mocking-Framework zur Erstellung von Pseudoklassen
- JUnit – Framework zur Durchführung von Unit-Tests auf der [JVM](#)
- Eclipse Luna mit TeXlipse – Entwicklungsumgebung L^AT_EX
- MiKTeX – Distribution des Textsatzsystems T_EX

Personal

- [BREPL](#)-Entwickler – Festlegung der Anforderungen und Abnahme des Projektes
- Fachbereich – Festlegung der Anforderungen und Abnahme des Projektes
- Entwickler – Umsetzung des Projektes
- Anwendungsentwickler – Review des Codes

A.3 Klassendiagramm der derzeit genutzten Eigenschaften

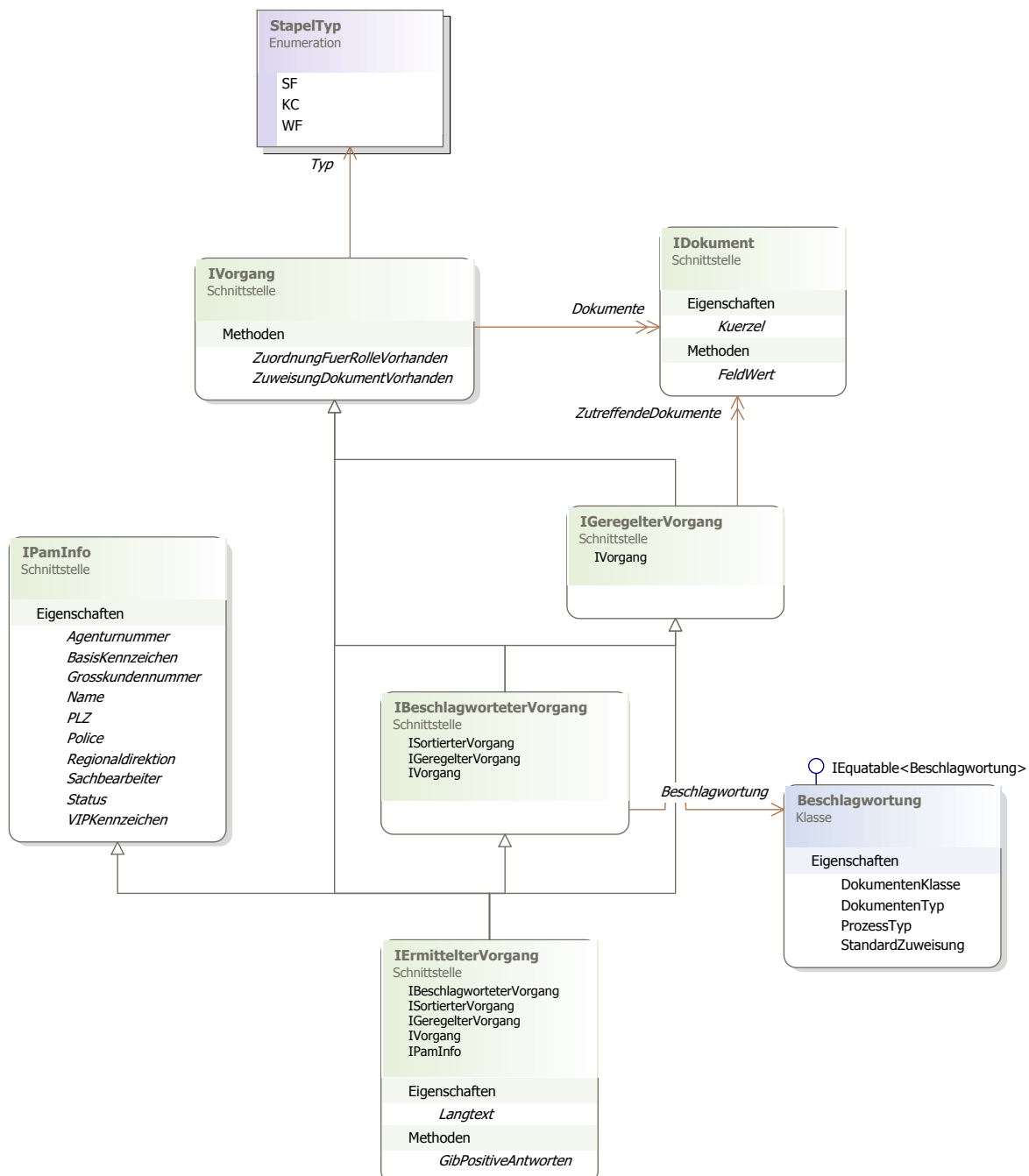


Abbildung 2: Klassendiagramm

A.4 Beispiel von Regeln mit C#-Code

```

1 [...]
2 if (beschlagwortung.DokumentenKlasse.Equals("ABRECH") &&
    beschlagwortung.DokumentenTyp.Equals("RECHNUNG"))
3 {
4     foreach (var gruppe in this.gruppen)
5     {
6         if (HatPamSperreBeiGruppe(vorgang.Status, vorgang.ZuweisungPolice, gruppe))
7         {
8             zuweisung = new ZuweisungGruppe(String.Format("MUSTERGRUPPE_{0}",
9                 gruppe));
10            return true;
11        }
12    }
13 if (vorgang.BasisKennzeichen.Equals(Versis.KENNZEICHEN_BASISTARIF))
14 {
15     zuweisung = new ZuweisungBenutzer("MUSTERMANN");
16     return true;
17 }
18
19 if (vorgang.BasisKennzeichen.Equals(Versis.KENNZEICHEN_STANDARDTARIF))
20 {
21     var name = vorgang.Name;
22     if (NameIstImBereich(name, 'A', 'M'))
23     {
24         zuweisung = new ZuweisungBenutzer("MUSTERFRAU");
25         return true;
26     }
27     if (NameIstImBereich(name, 'N', 'Z'))
28     {
29         zuweisung = new ZuweisungBenutzer("MUSTERMANN");
30         return true;
31     }
32 }
33
34 zuweisung = null;
35 return false;
36 [...]

```

Listing 1: Regeln in Form von C#

A.5 Ausschnitt aus dem Regelwerk mit möglicher Umsetzung

Eigenschaft	Operant	Parameter	Beispiel	Mögliche Umsetzung
Eigenschaft	ist	String1	vorgang.Typ.Equals(StapelTyp.KC)	Eigenschaft = "String1"
Vorgang	ZuordnungFürRolle	Rolle	vorgang.ZuordnungFuerRolleVorh	Regel... => Zuordnung MailBetreff
Dokument	HatZuweisung	ZuweisungsName	vorgang.ZuweisungDokumentVorh	Regel... => Zuordnung MasterDokument
Vorgang(Gruppe)	HatPamSperr	Gruppenname	HatPamSperrBeiGruppe(vorgang	HatPamSperrBeiGruppe "Gruppenname"
Eigenschaft	zwischen	String1 String2	NameIstImBereich(name, 'A', 'M')	Name zwischen "A" und "M"
Eigenschaft	IstAokAgentur	Agenturnummer	IstAokAgentur(agenturnummer)	IstAokAgentur
Dokumente	alle Kuerzel sind	Kuerzel	vorgang.Dokumente.All(...))	DokumentenKuerzel sind "Kuerzel"
Dokumente	irgendein Kuerzel ist	Kuerzel	vorgang.Dokumente.Any(...))	DokumentenKuerzel enthalten "Kuerzel"

Abbildung 3: Ausschnitt der Liste der darzustellenden Operationen

A.6 Amortisation

Der Schnittpunkt der beiden Geraden gibt den Zeitpunkt der Amortisation an.

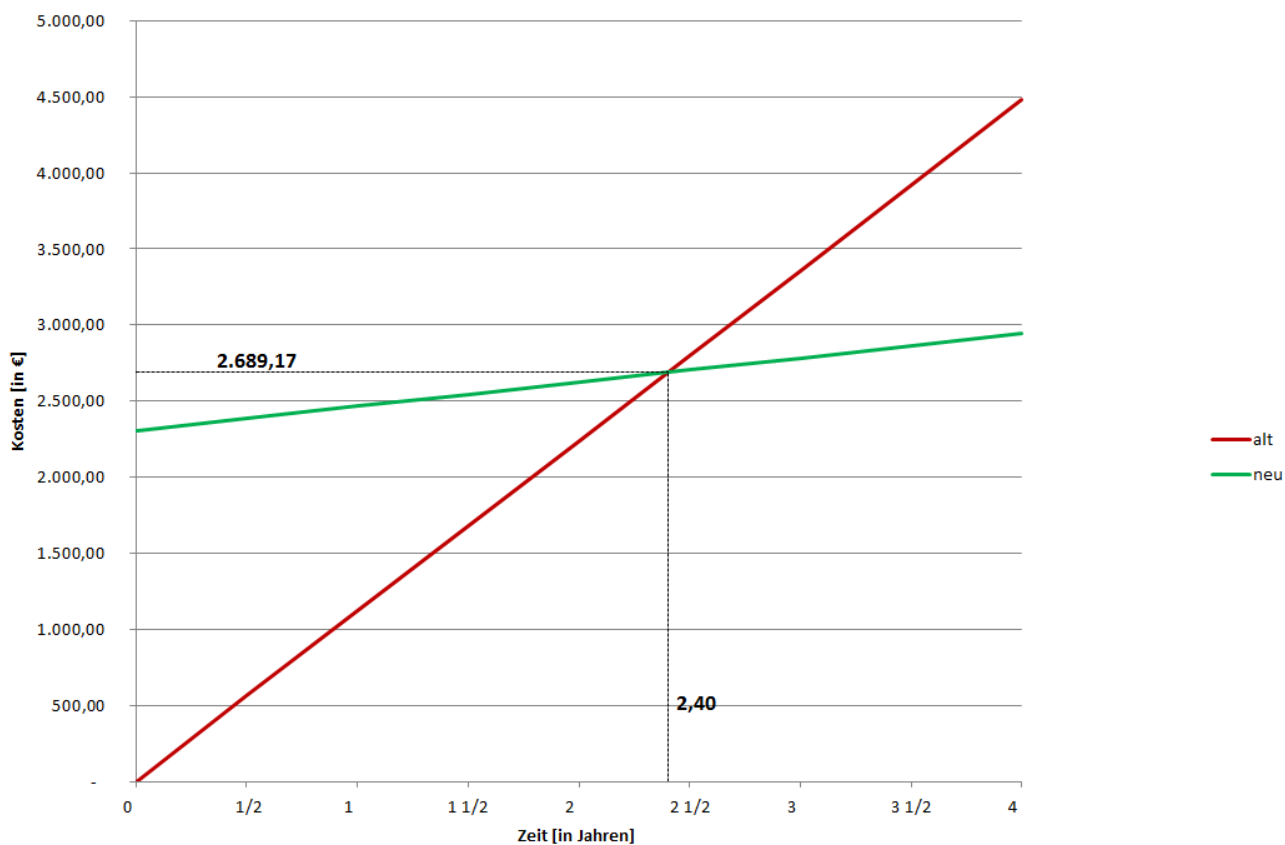


Abbildung 4: Graphische Darstellung der Amortisation

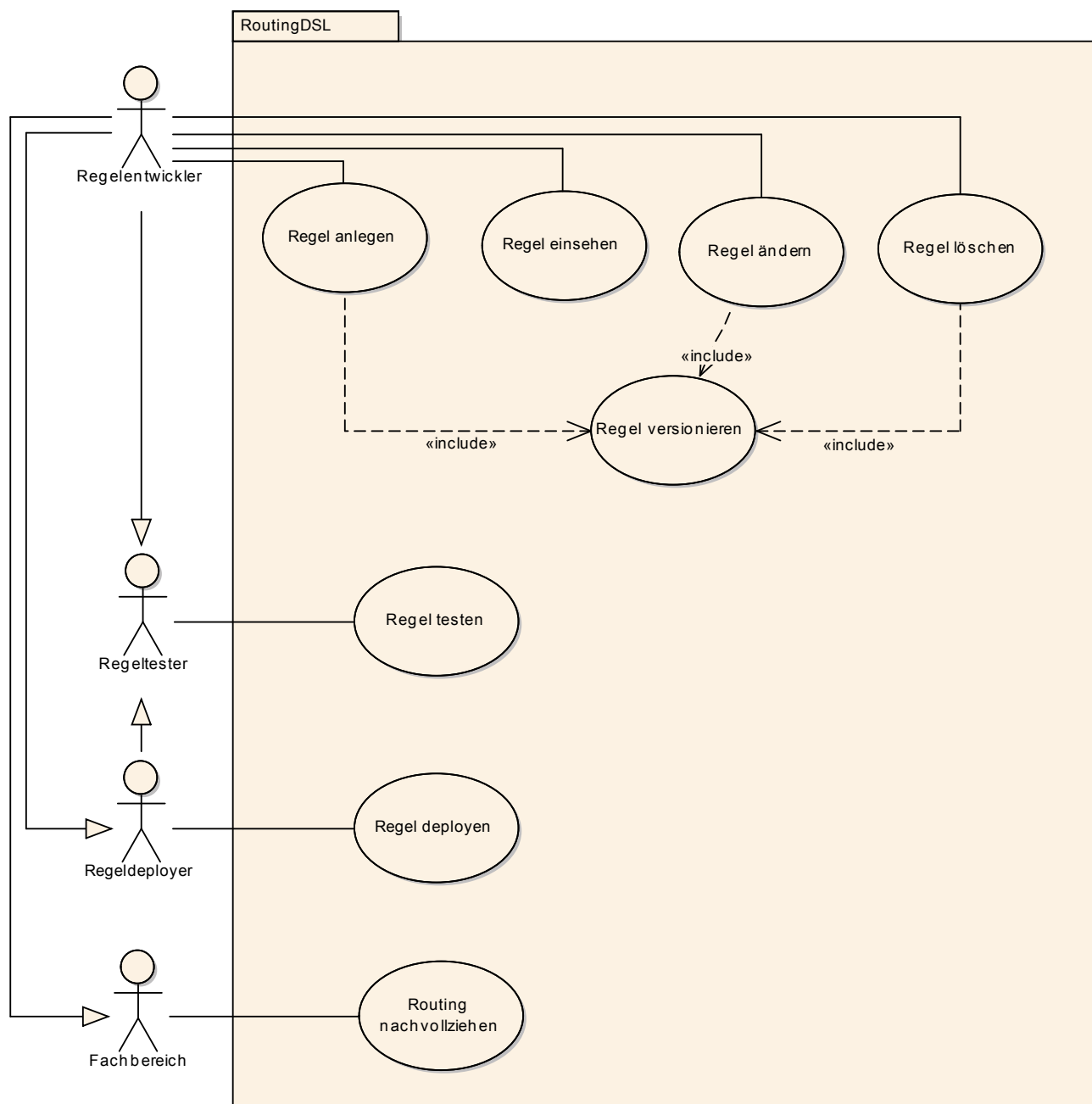
A.7 Use-Case-Diagramm

Abbildung 5: Use-Case-Diagramm

A.8 Lastenheft (Auszug)

Im folgenden Auszug aus dem Lastenheft werden die Anforderungen definiert, die die zu entwickelnde Anwendung erfüllen muss. Betrachtet wird die Anwendung aus Sicht der im Anhang [A.7: Use-Case-Diagramm](#) auf S. [vi](#) ermittelten Akteure.

Anforderungen

Es werden folgende Anforderungen an die Anwendung gestellt:

- Als Regelentwickler muss ich Regeln anlegen können, weil neue Regeln vom Fachbereich zeitnah ins System eingepflegt werden müssen.
- Als Regelentwickler muss ich Regeln einsehen können, um mir Bedingungen von vorhandenen Regeln klar machen zu können.
- Als Regelentwickler muss ich Regeln ändern können, da eine Änderung der Zuweisung möglichst schnell umgesetzt werden muss.
- Als Regelentwickler muss ich Regeln löschen können, da ein Wegfall von Regeln vorkommen kann.
- Als Regelentwickler muss ich Regeln versionieren können, da ich Änderungen am Regelwerk nachvollziehen und dokumentieren muss.
- Als Regelentwickler sollte ich Regeln testen können, damit ich mir sicher bin, dass alle Regeln nutzbar sind.
- Als Regelentwickler möchte ich textuell entwickeln können, da ich mit der Tastatur schneller arbeite als mit der Maus.
- Als Regeltester muss ich Regeln testen können, damit sichergestellt ist, dass alle Regeln nutzbar sind.
- Als Regeltester möchte ich, dass automatisiert getestet wird, damit die Tests zuverlässig regelmäßig durchgeführt werden.
- Als Regeldeployer muss ich Regeln deployen können, damit diese von [BREPL](#) genutzt werden können.
- Als Regeldeployer möchte ich Regeln auf Knopfdruck deployen können, damit ich beim Deployen nichts vergessen kann.
- Als Mitarbeiter möchte ich das Routing nachvollziehen können, damit ich weiß, warum ein Vorgang bei mir landet.

[...]

A.9 Nutzwertanalyse zur Auswahl des DSL-Frameworks

Eigenschaft	Gewichtung	Xtext (bew.)	ANTLR.NET (bew.)	MPS (bew.)	Xtext (gew.)	ANTLR.NET (gew.)	MPS (gew.)
Umfang der Dokumentation	10	2	1	2	20	10	20
Community	10	2	0	1	20	0	10
Erfahrung in der AO	30	2	0	0	60	0	0
Textuelle Entwicklung	20	2	1	2	40	20	40
Versionierbarkeit	10	2	2	2	20	20	20
Lizenzkosten	10	2	2	2	20	20	20
Continuous Integration	10	1	2	0	10	20	0
Gesamt:	100				190	90	110
Nutzwert:					1,9	0,9	1,1

Eigenschaft	0 Punkte	1 Punkt	2 Punkte
Umfang der Dokumentation	keine Dokumentation vorhanden	wenig Dokumentation vorhanden	ausführliche Dokumentation vorhanden
Community	keine Community vorhanden	es gibt ein Forum der Entwickler	es gibt aktive Foren und Open-Source-Projekte
Erfahrung in der AO	nicht vorhanden	schon eingesetzt	wird bereits eingesetzt
Textuelle Entwicklung	nicht möglich	ist möglich ohne Toolunterstützung	es gibt Toolunterstützung
Versionierbarkeit	nicht möglich, binär	nur durch Zusatztools möglich	ist möglich
Lizenzkosten	hohe Kosten	geringe Kosten	keine Kosten
Continuous Integration	nicht möglich	durch Eigenarbeit möglich	es gibt vorhandene Lösungen

A.10 Aktivitätsdiagramm zum Anlegen einer Regel

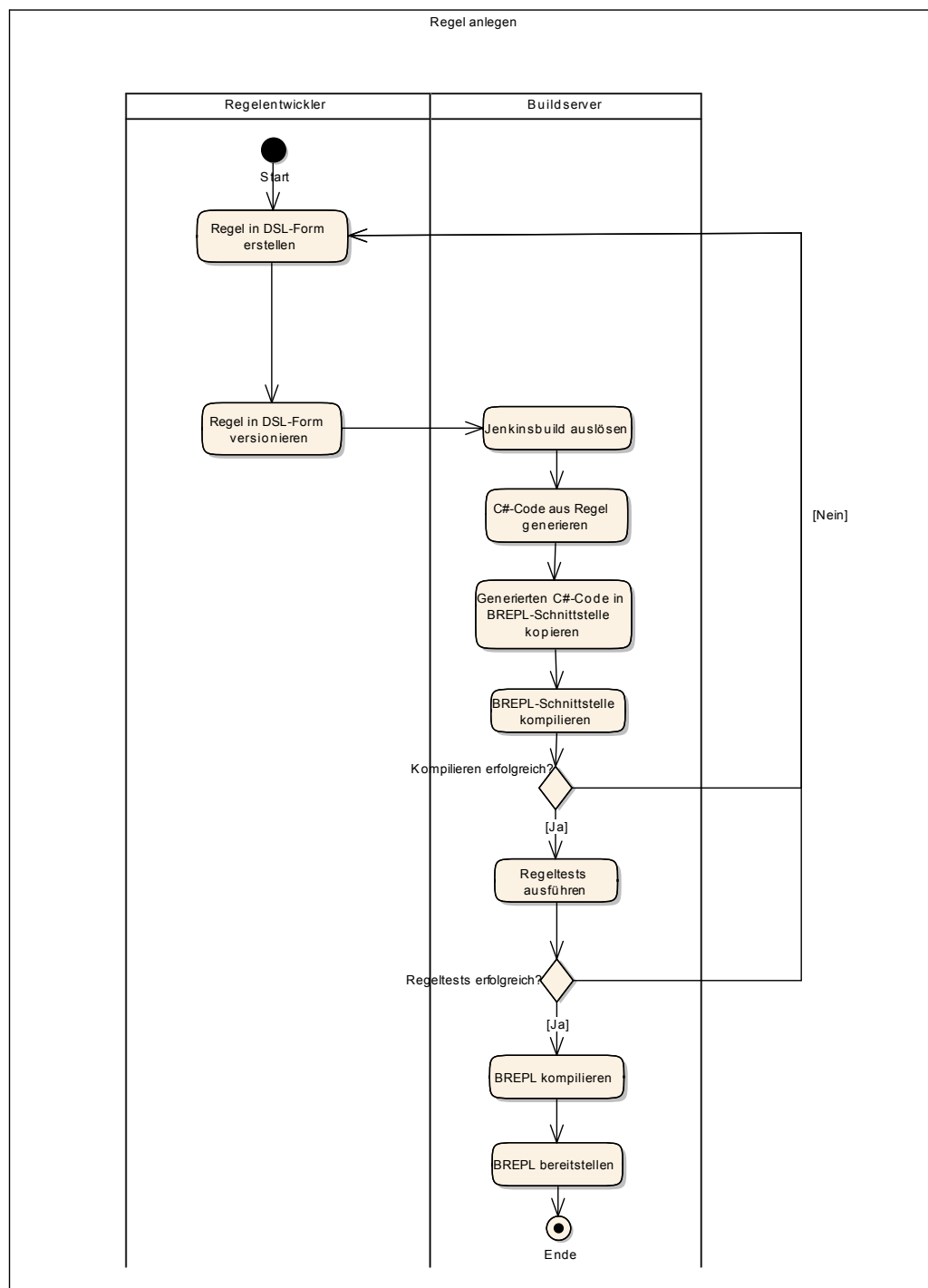


Abbildung 6: Aktivitätsdiagramm

A.11 Deploymentdiagramm

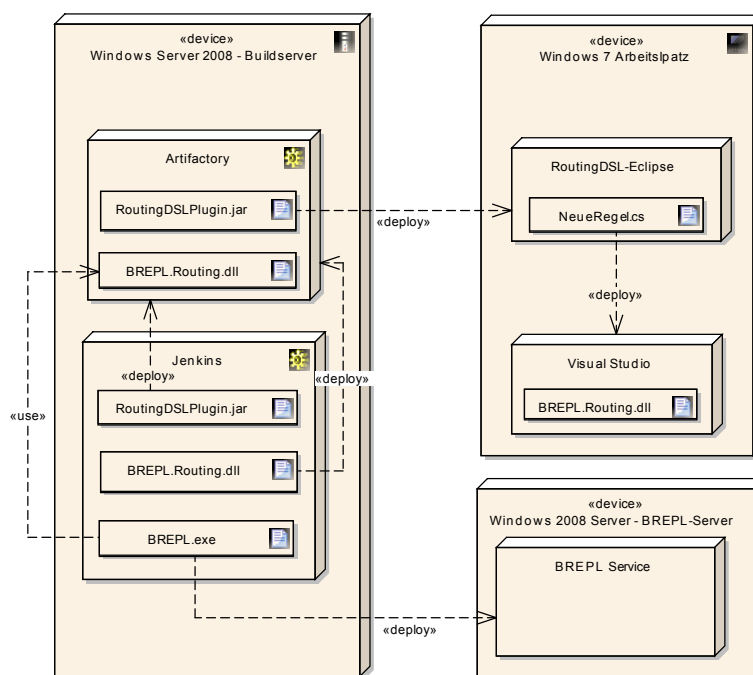


Abbildung 7: Deploymentdiagramm

A.12 Entwurf der DSL

```

1 Regelwerk _NAME_
2 {
3   Unternehmen: _UNTERNEHMEN_
4   Stapelkategorie: _STAPELKATEGORIE_
5   Clearing: _CLEARINGZUWEISUNG_
6
7   Regel _BESCHREIBUNG_ => _ZUWEISUNG_
8   {
9     _BEDINGUNGEN_
10
11     _WERTÄNDERUNG_
12   }
13 }

```

Listing 2: Entwurf der DSL

A.13 Entwurf der DSL mit Klammern

```
1 Regelwerk _NAME_  
2 {  
3   Unternehmen: _UNTERNEHMEN_  
4   Stapelkategorie: _STAPELKATEGORIE_  
5   Clearing: _CLEARINGZUWEISUNG_  
6  
7   Klammer _KLAMMERNAME_  
8   {  
9     _BEDINGUNGEN_  
10  
11    Regel _BESCHREIBUNG_ => _ZUWEISUNG_  
12    {  
13      _BEDINGUNGEN_  
14  
15      _WERTÄNDERUNG_  
16    }  
17  
18    Regel _BESCHREIBUNG_ => _ZUWEISUNG_  
19    {  
20      _BEDINGUNGEN_  
21  
22      _WERTÄNDERUNG_  
23    }  
24  }  
25 }
```

Listing 3: Entwurf der DSL mit Klammern

A.14 Entwurf der Schnittstelle zu BREPL

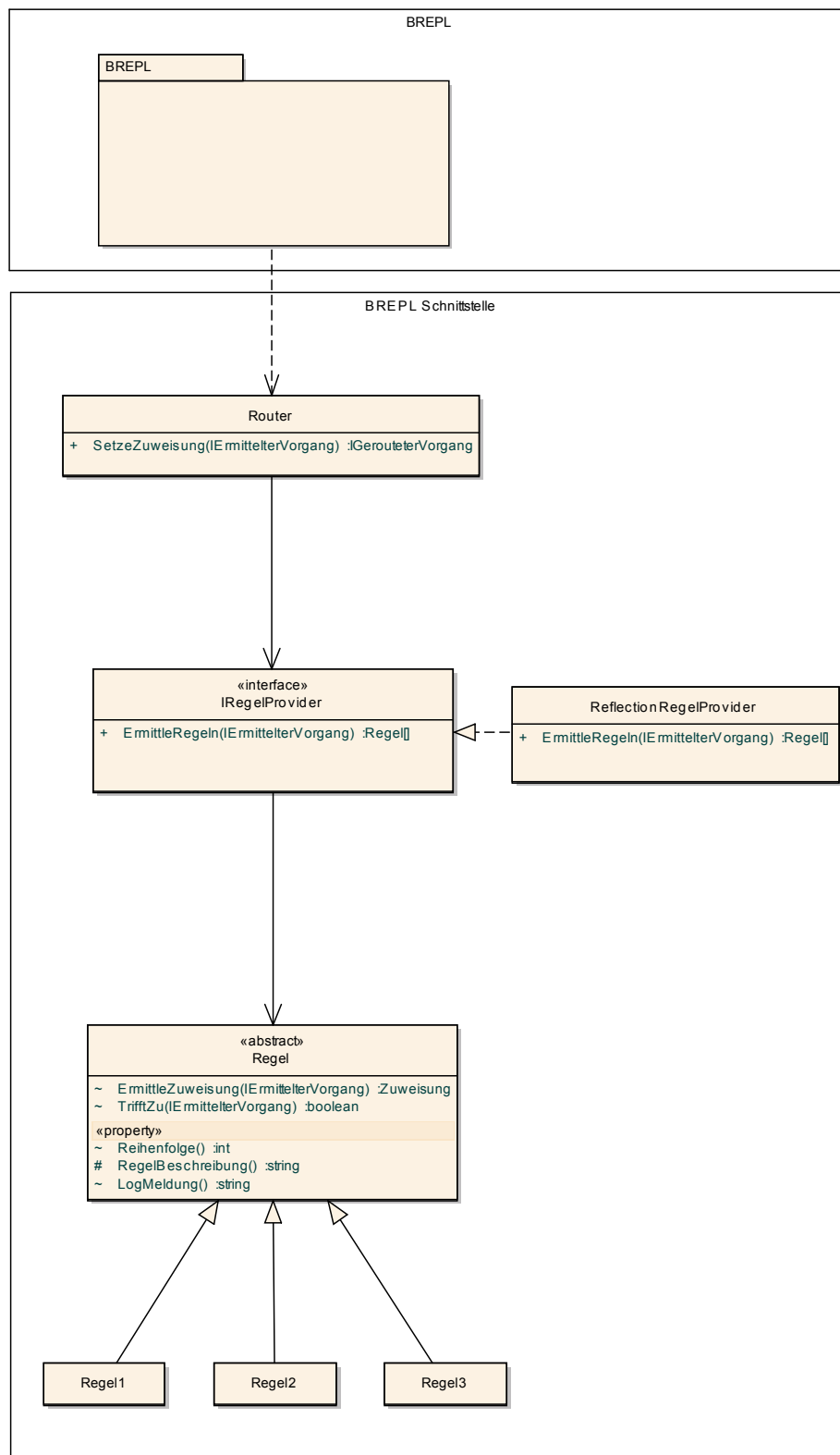


Abbildung 8: Entwurf der Schnittstelle zu BREPL

A.15 Pflichtenheft (Auszug)

In folgendem Auszug aus dem Pflichtenheft wird die geplante Umsetzung der im Lastenheft definierten Anforderungen beschrieben:

- Um als Regelentwickler Regeln zeitnah in das System einzupflegen, muss das Anlegen von Regeln in einer Eclipse-Instanz erfolgen.
- Um mir als Regelentwickler Bedingungen von vorhanden Regeln klarzumachen, muss das Einsehen von Regeln in einer Eclipse-Instanz ermöglicht werden.
- Um als Regelentwickler eine Änderung an Regeln schnell umzusetzen, muss das Ändern von Regeln in einer Eclipse-Instanz möglich sein.
- Um als Regelentwickler auf einen Wegfall von Regeln reagieren zu können, muss das Löschen von Regeln in einer Eclipse-Instanz möglich sein.
- Um als Regelentwickler Änderungen an Regelwerken nachzuvollziehen, müssen Regeln mit Hilfe von Git im [BREPL](#)-Repository versionierbar sein können.
 - Repository-URL: `http://url-zu-git/git/.NET/BREPL`
 - Konvention: Commit startet mit „Ticket #TICKETNUMMER“
- Um als Regelentwickler effizienter arbeiten zu können, könnte mir durch das Anpassen des Eclipses ein gutes Erlebnis geboten werden.
 - Sprechende Eclipse-Outline durch Symbole und Beschriftung.
 - Aussagekräftige Compilerwarnungen und -fehler.
 - Eclipse-Quickfixes für bestimmte Eingabetypen (Datum).
 - Eclipse-Templates für Regeln und Klammern.
- Um mir als Regeltester sicher zu sein, dass alle Regeln nutzbar sind, muss ich generierte Unittests zu jeder Regel lokal ausführen können.
- Um als Regeldeployer keine manuelle Arbeit beim Deployment zu haben, könnten die Regeln während des Buildprozess mit deployt werden.
 - In den MSBuild-Targets zum Kopieren der generierten Regeln ein „BeforeBuild-Target“ nutzen.
- Um als Fachbereich das Routing nachvollziehen zu können, wird das Routing komplett geloggt.
 - Typ: Oracle Datenbank
 - Datenbank: DBNAME
 - Benutzer: DBUSER

A.16 Iterationsplan

1. Erstellung des DSL-Projektes

- Ziel: Ein versionierbares Eclipse Projekt für die Entwicklung von Xtext und Xtend.
- Repository-URL: <http://url-zu-git/git/Xtext/RoutingDsl>
- Classpath-Konvention: „net.aokv.“

2. Erstellung eines Jobs auf dem Buildserver für das DSL-Projekt

- Ziel: Ein erfolgreicher Build auf dem Buildserver, welcher die [DSL](#) baut und testet.
- Ziel-URL: <http://url-zu-jenkins/job/net.aokv.brepl.dsl.routingdsl/>

3. Implementierung der Grammatik

- Ziel: Generierung von Java-Klassen durch Xtext für alle abzudeckenden [DSL](#)-Sprachkonstrukte.

4. Implementierung der Codegenerierung

- Generierung von C#-Regelklassen
- Generierung von C#-Unittests
- Generierung der Routingkonfiguration in C#

5. Erstellung des Schnittstellen-Projektes

- Ziel: Ein .NET-Projekt innerhalb von [BREPL](#), welches automatisch beim Kompilieren von [BREPL](#) mitkompiliert wird.
- Repository-URL: <http://url-zu-git/git/.NET/BREPL>
- Namespace-Konvention: „BREPL.Projektname“
- Projekt-Namespace: „BREPL.Routing“
- Testprojekt-Namespace: „BREPL.Routing.Tests“

6. Implementierung des Schnittstellen-Projektes

- Ziel: Die generierten Regeln müssen zur Kompilierzeit von [BREPL](#) analysiert und mit in die Assembly kompiliert werden, um zur Laufzeit nutzbar zu sein.

A.17 Grammatik der DSL in Xtext

```

1 RoutingDatei:
2   Regelwerk | Konfiguration;
3
4 Regelwerk:
5   'Regelwerk' name=ID
6   '{'
7   'Unternehmen:' unternehmen=Unternehmen
8   'Stapelkategorie:' stapelkategorie=Stapelkategorie
9   'Clearing:' clearingzuweisung=STRING
10  regelwerkElemente+=RegelwerkElemente*
11  '}' ;
12
13 Konfiguration:
14  {Konfiguration} 'Konfiguration' '{'
15  routingKonfigurationen+=RoutingKonfiguration*
16  '}' ;
17
18 RegelwerkElemente:
19  Regel | Klammer;
20
21 Regel:
22  'Regel' name=STRING '=>' zuweisung=Zuweisung
23  '{'
24  bedingungen+=Bedingung*
25  aktionen+=Aktion*
26  '}' ;
27
28 Klammer:
29  'Klammer' name=STRING
30  '{'
31  bedingungen+=Bedingung*
32  elemente+=RegelwerkElemente*
33  '}' ;
34
35 RoutingKonfiguration:
36  Einzeltyp | Array;
37
38 [...]

```

Listing 4: Definition der DSL in Xtext

A.18 Beispiel der Regeln mit RoutingDSL-Code

```
1 [...]
2 Regel "Pam-Sperre Gruppe TA" => Gruppe "Mustergruppe"
3 {
4     DokumentenKlasse = "ABRECH"
5     DokumentenTyp = "RECHNUNG"
6     HatPamSperreBeiGruppe "TA"
7 }
8
9 Regel "Pam-Sperre Gruppe Pflege" => Gruppe "Andere Gruppe"
10 {
11     DokumentenKlasse = "ABRECH"
12     DokumentenTyp = "RECHNUNG"
13     HatPamSperreBeiGruppe "Pflege"
14 }
15
16 Regel "Kennzeichen Basistarif" => Benutzer "Mustermann"
17 {
18     BasisKennzeichen = "B"
19 }
20
21 Klammer "Kennzeichen Standardtarif"
22 {
23     BasisKennzeichen = "S"
24
25     Regel "Name A bis M" => Benutzer "Musterfrau"
26     {
27         Name zwischen "A" und "M"
28     }
29
30     Regel "Name N bis Z" => Benutzer "Mustermann"
31     {
32         Name zwischen "N" und "Z"
33     }
34 }
35 [...]
```

Listing 5: Regeln in Form von RoutingDSL

A.19 Screenshots der Entwicklungsumgebung

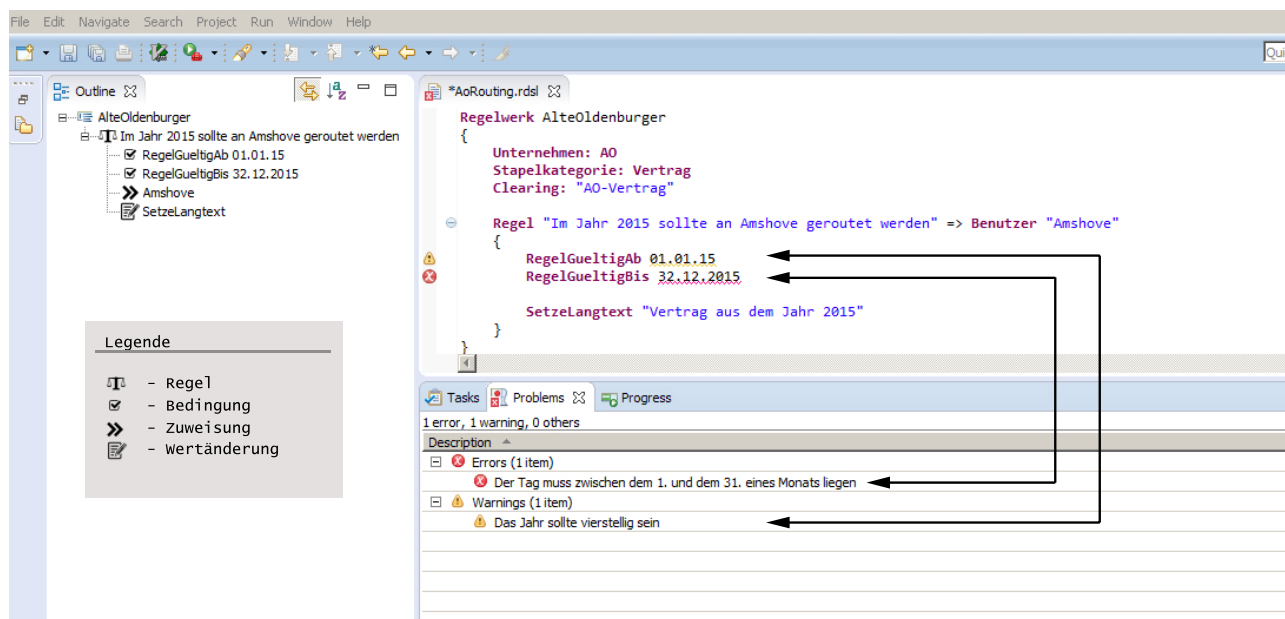


Abbildung 9: Screenshot der Entwicklungsumgebung mit Outline, Validierung, Fehlern und Warnungen

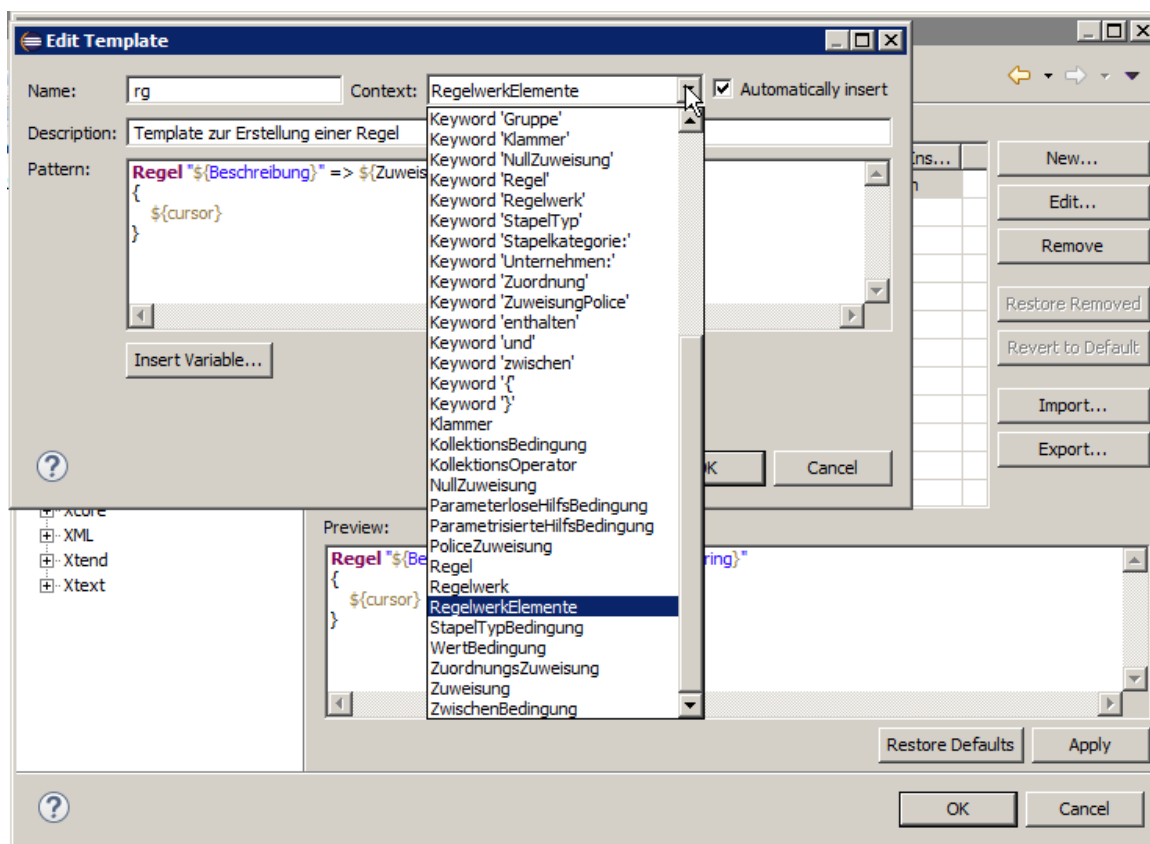


Abbildung 10: Screenshot der Definition von Code-Templates

A.20 Ausschnitt aus der Codeformatierung in Xtend

```

1 private def regelFormatierung(FormattingConfig konfiguration)
2 {
3     val entities = access.regelAccess
4     konfiguration.setIndentation(entities.leftCurlyBracketKeyword_4,
5         entities.rightCurlyBracketKeyword_7)
6     konfiguration.setLinewrap(2).before(entities.regelKeyword_0)
7     konfiguration.setLinewrap.around(entities.leftCurlyBracketKeyword_4)
8     konfiguration.setLinewrap.around(entities.rightCurlyBracketKeyword_7)
9     konfiguration.setLinewrap.around(entities.bedingungenAssignment_5)
10    konfiguration.setLinewrap(2).before(entities.aktionenAktionParserRuleCall_6_0)
11    konfiguration.setLinewrap.around(entities.aktionenAssignment_6)
12 }

```

Listing 6: Codeformatierung in Xtend

```

1 @Test
2 def void zweiRegelnKorrektTrennen()
3 {
4     '''
5     Regelwerk MeinRegelwerk { Unternehmen: A0 Stapelkategorie: Leistung Clearing:
6         "Asd"
7     Regel "Eins" => Clearing { } Regel "Zwei" => Clearing { } }
8     '''
9     .assertFormattedAs(
10         '''
11         Regelwerk MeinRegelwerk
12         {
13             Unternehmen: A0
14             Stapelkategorie: Leistung
15             Clearing: "Asd"
16
17             Regel "Eins" => Clearing
18             {
19             }
20
21             Regel "Zwei" => Clearing
22             {
23             }
24         }'''
25     )
26 }

```

Listing 7: Tests der Codeformatierung in Xtend

A.21 Xtend-Methoden zur Ermittlung des zugehörigen C#-Operators und Beispiel einer generierten C#-Regel

```

1 private def String operator(Bedingung bedingung)
2 {
3     switch bedingung
4     {
5         EinzelwertBedingung:
6             ermittleCsharpOperator(bedingung.operator)
7         StapelTypBedingung:
8             ermittleCsharpOperator(bedingung.operator)
9         KollektionsBedingung:
10            switch (bedingung.operator)
11            {
12                case OPERANT_KOLLEKTIONSBEDINGUNG_ENTHAELT: '=='
13                case OPERANT_KOLLEKTIONSBEDINGUNG_ENTHAELT_NICHT: '!='
14                case OPERANT_KOLLEKTIONSBEDINGUNG_SIND: '=='
15                case OPERANT_KOLLEKTIONSBEDINGUNG_SIND_NICHT: '!='
16                default: '''Kein Operator für Kollektionsoperator <<bedingung.operator>>
                        gefunden'''
17            }
18        DatumsBedingung:
19            switch (bedingung.eigenschaft)
20            {
21                case DatumsEigenschaft.REGEL_GUELTIG_AB: '>='
22                case DatumsEigenschaft.REGEL_GUELTIG_BIS: '<='
23            }
24    }
25 }
26
27 private def String ermittleCsharpOperator(String operator)
28 {
29     switch (operator)
30     {
31         case OPERANT_EINZELWERTBEDINGUNG_IST: '=='
32         case OPERANT_EINZELWERTBEDINGUNG_IST_NICHT: '!='
33         default: '''Kein Operator für <<operator>> gefunden'''
34     }
35 }

```

Listing 8: Ausschnitt aus dem Bedingungsgenerator

```
1 [...]
2 [Attribute.PkVertrag]
3 internal class Regel47 : Regel
4 {
5     private Zuweisung ClearingZuweisung
6     {
7         get { return new ZuweisungGruppe("Clearing"); }
8     }
9
10    internal override int Reihenfolge
11    {
12        get { return 47; }
13    }
14
15    internal override IGerouteterVorgang ErmittleZuweisung(IGermittelterVorgang
16        vorgang)
17    {
18        var zuweisung = new ZuweisungGruppe("Mustergruppe");
19        zuweisung.RoutingLog = Log;
20        return vorgang.Geroutet(zuweisung);
21    }
22
23    internal override Boolean TrifftZu(IGermittelterVorgang vorgang)
24    {
25        return vorgang.Regionaldirektion == "598"
26            && vorgang.Dokumente.Any(dokument => dokument.Kuerzel == "5230")
27            && vorgang.Name.IstZwischen("A", "M");
28    }
29
30    protected override string RegelBeschreibung
31    {
32        get { return "PkVertrag.PT-Dynamisierung Emden im Bereich A bis M"; }
33    }
34 }
```

Listing 9: Beispiel einer generierten Regel

A.22 Generierter C#-Unittest

```

1 [Test]
2 public void Regel47RoutenKoennen()
3 {
4     var vorgang = ErstelleTestvorgang();
5     var router = new Router();
6     vorgang.Regionaldirektion = "598";
7     vorgang.DokumenteFeld.Add(new TestDokument("5230"));
8     vorgang.Name = "A";
9
10    var erwartet = "PkVertrag.PT-Dynamisierung Emden im Bereich A bis M";
11    var ist = router.SetzeZuweisung(vorgang).Zuweisung.RoutingLog;
12    Assert.That(ist, Is.EqualTo(erwartet));
13 }

```

Listing 10: Generierte Testmethode zum Test einer Regel

A.23 Ermittlung aller Regeln durch .NET Reflection-API

```

1 private IEnumerable<Type> ErmittleKlassen(Type attribut)
2 {
3     var alleKlassen = typeof(ReflectionRegelProvider).Assembly.GetTypes();
4     return from klasse in alleKlassen
5         where klasse.IsSubclassOf(typeof(Regel))
6             && !klasse.IsAbstract
7             && klasse.GetCustomAttributes(attribut, true).Length > 0
8         select klasse;
9 }

```

Listing 11: Ermittlung aller Klassen zu einem Attribut

A.24 Instanziierung aller ermittelten Regelklassen

```

1 private Regel[] InstanziiereKlassen(Type attribut)
2 {
3     return (from klasse in ErmittleKlassen(attribut)
4         select klasse.GetConstructor(Type.EmptyTypes)
5             into konstruktor
6             where konstruktor != null
7             select (Regel)konstruktor.Invoke(new object[] { }))
8         .OrderBy(regel => regel.Reihenfolge).ToArray();
9 }

```

Listing 12: Instanziierung aller ermittelten Regelklassen und aufsteigend nach Reihenfolge sortiert.

A.25 Ausschnitt aus dem Benutzerhandbuch zur Verdeutlichung des Workflows

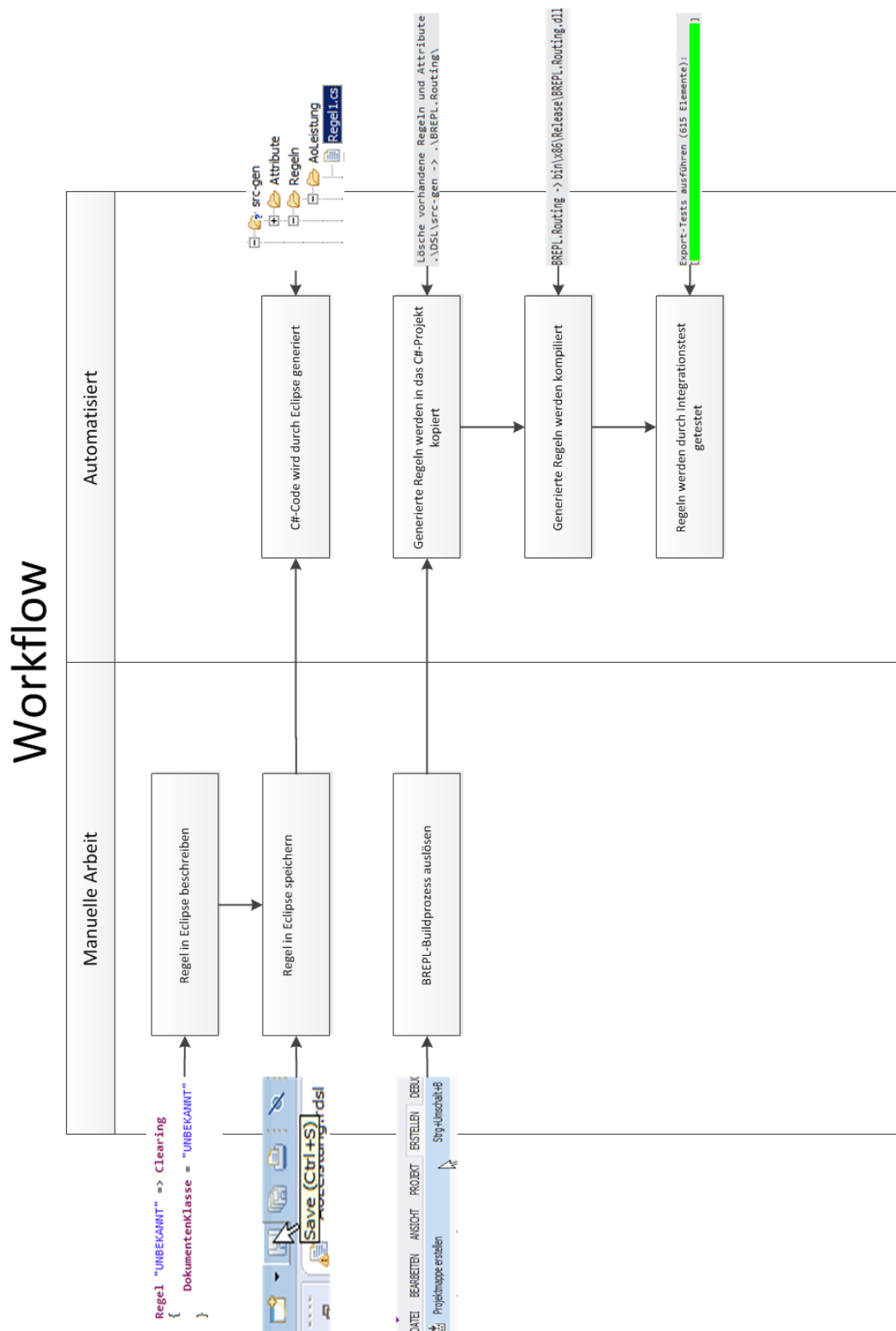


Abbildung 11: Darstellung des Workflows

A.26 Ausschnitt aus dem CI-Prozess

Testergebnisse

Fehlschläge (+1)

Tests (+1)

Dauer: 5.3 Sekunden

 Beschreibung hinzufügen

Alle fehlgeschlagenen Tests

Testname	Dauer	Alter
+ BREPL.Routing.Tests.Regelwerke.PkVertragSollte.Regel47RoutenKoennen	31 ms	1


Alle Tests

Package	Dauer	Fehlgeschlagen	(Diff.)	Übersprungen	(Diff.)	Pass	(Diff.)	Summe	(Diff.)
BREPL.Beschlagwortung.Tests	0,98 Sekunden	0		0		45		45	
BREPL.DatenbankSchicht.Tests	0,62 Sekunden	0		0		44		44	
BREPL.Domaenenmodell.Tests	0,3 Sekunden	0		0		77		77	
BREPL.Routing.Tests.Regelwerke	0,63 Sekunden	1	+1	0		146		147	+1
BREPL.Sortierung.Tests	79 ms	0		0		13		13	
BREPL.StapelProduktion.Tests	2,4 Sekunden	0		0		95		95	
BREPL.Stapelverarbeitung.Tests	0,29 Sekunden	0		0		16		16	

Abbildung 12: Fehlgeschlagener Build durch fehlgeschlagenen Regeltest

**Build #748 (13.04.2015 10:52:02)**

Diesen Build unbefristet aufbewahren

 Beschreibung hinzufügen

Vor 3 Minuten 47 Sekunden gestartet

Dauer: 1 Minute 40 Sekunden



Changes

1. Ticket #9073: Refactoring des Regelproviders ([detail](#))
2. Ticket #9073: Reihenfolge in PKVertrag korrigiert ([detail](#))

[Build wurde durch eine SCM-Änderung ausgelöst.](#)

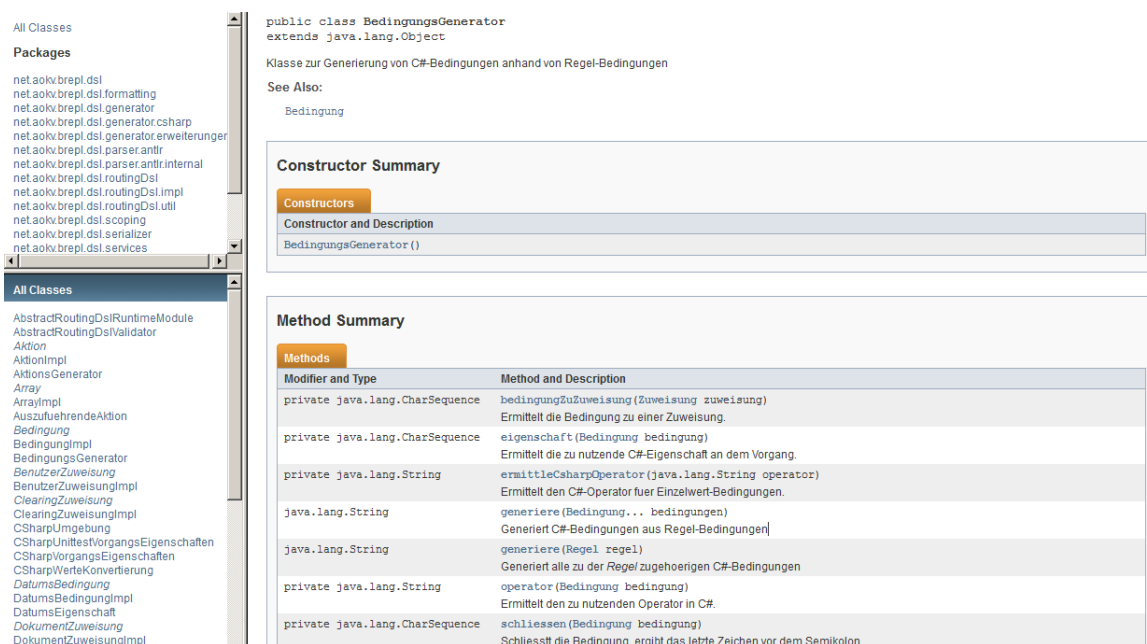
Revision: 4b0b30105f8790d4500a60497da82dd9172a54c0

- origin/hauptversion

[Testergebnis](#) (Kein Test fehlgeschlagen.)

Abbildung 13: Erfolgreicher Build nach Korrektur der Regelreihenfolge

A.27 Ausschnitt aus der Entwicklerdokumentation



The screenshot shows the Xtend developer documentation for the `BedingungsGenerator` class. On the left, a sidebar lists 'All Classes' and 'Packages'. The main content area displays the class signature, a brief description, and a 'See Also' link to `Bedingung`. Below this, there are two summary sections: 'Constructor Summary' and 'Method Summary'.

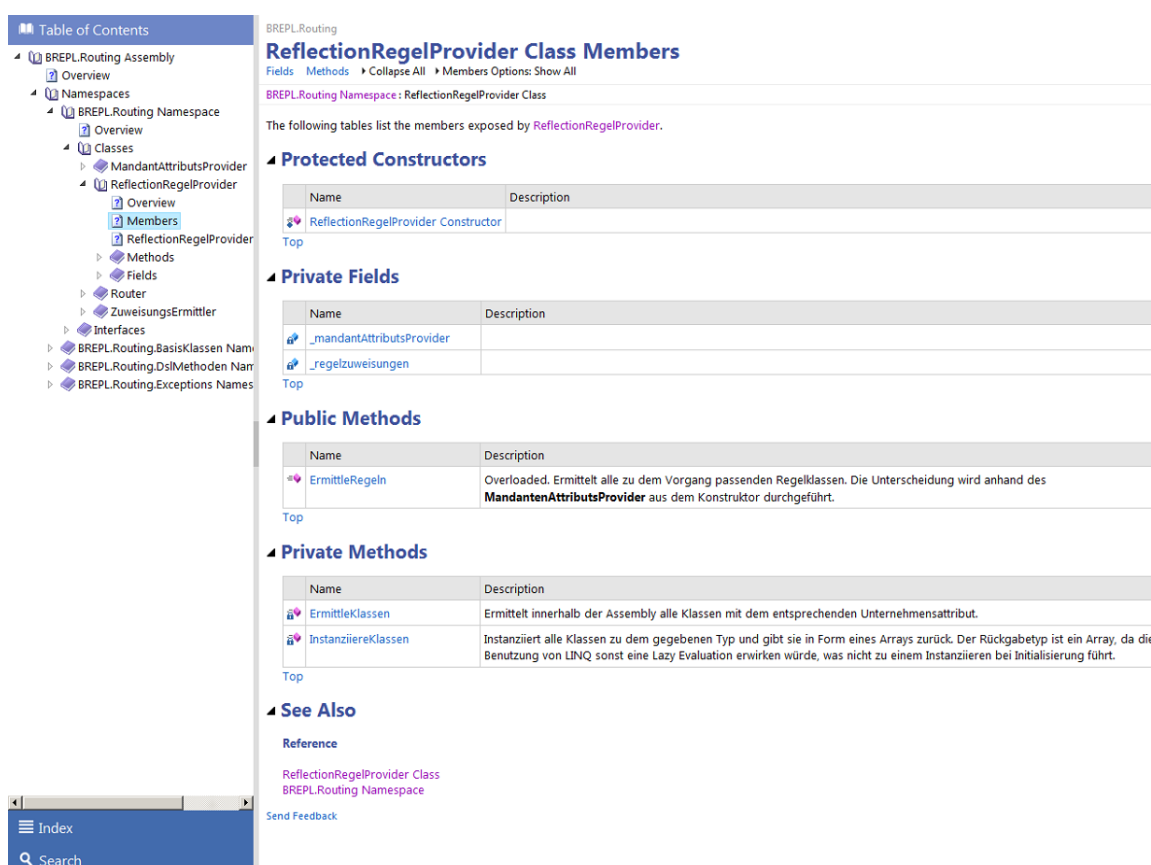
Constructor Summary

Constructor and Description
<code>BedingungsGenerator ()</code>

Method Summary

Modifier and Type	Method and Description
<code>private java.lang.CharSequence</code>	<code>bedingungZuZuweisung (Zuweisung zuweisung)</code> Ermittelt die Bedingung zu einer Zuweisung.
<code>private java.lang.CharSequence</code>	<code>eigenschaft (Bedingung bedingung)</code> Ermittelt die zu nutzende C#-Eigenschaft an dem Vorgang.
<code>private java.lang.String</code>	<code>ermittleCSharpOperator (java.lang.String operator)</code> Ermittelt den C#-Operator fuer Einzelwert-Bedingungen.
<code>java.lang.String</code>	<code>generiere (Bedingung... bedingungen)</code> Generiert C#-Bedingungen aus Regel-Bedingungen.
<code>java.lang.String</code>	<code>generiere (Regel regel)</code> Generiert alle zu der Regel zugehoerigen C#-Bedingungen.
<code>private java.lang.String</code>	<code>operator (Bedingung bedingung)</code> Ermittelt den zu nutzenden Operator in C#.
<code>private java.lang.CharSequence</code>	<code>schliessen (Bedingung bedingung)</code> Schliesst die Bedingung, ergibt das letzte Zeichen vor dem Semikolon.

Abbildung 14: Auszug aus der Entwicklerdokumentation für Xtend



The screenshot shows the .NET developer documentation for the `ReflectionRegelProvider` class. On the left, a 'Table of Contents' sidebar lists the navigation structure. The main content area displays the class signature, a brief description, and a 'See Also' link to `ReflectionRegelProvider Class`.

ReflectionRegelProvider Class Members

The following tables list the members exposed by `ReflectionRegelProvider`.

Protected Constructors

Name	Description
<code>ReflectionRegelProvider Constructor</code>	

Private Fields

Name	Description
<code>_mandantAttributsProvider</code>	
<code>_regelzuweisungen</code>	

Public Methods

Name	Description
<code>ErmittleRegeln</code>	Overloaded. Ermittelt alle zu dem Vorgang passenden Regelklassen. Die Unterscheidung wird anhand des <code>MandantenAttributsProvider</code> aus dem Konstruktor durchgeführt.

Private Methods

Name	Description
<code>ErmittleKlassen</code>	Ermittelt innerhalb der Assembly alle Klassen mit dem entsprechenden Unternehmensattribut.
<code>InstanziiereKlassen</code>	Instanziert alle Klassen zu dem gegebenen Typ und gibt sie in Form eines Arrays zurück. Der Rückgabety ist ein Array, da die Benutzung von LINQ sonst eine Lazy Evaluation erwirken würde, was nicht zu einem Instanzieren bei Initialisierung führt.

See Also

Reference

`ReflectionRegelProvider Class`
`ReflectionRegelProvider Namespace`

Send Feedback

Abbildung 15: Auszug aus der Entwicklerdokumentation für die .NET-Schnittstelle

A.28 Erstellte Codemetriken

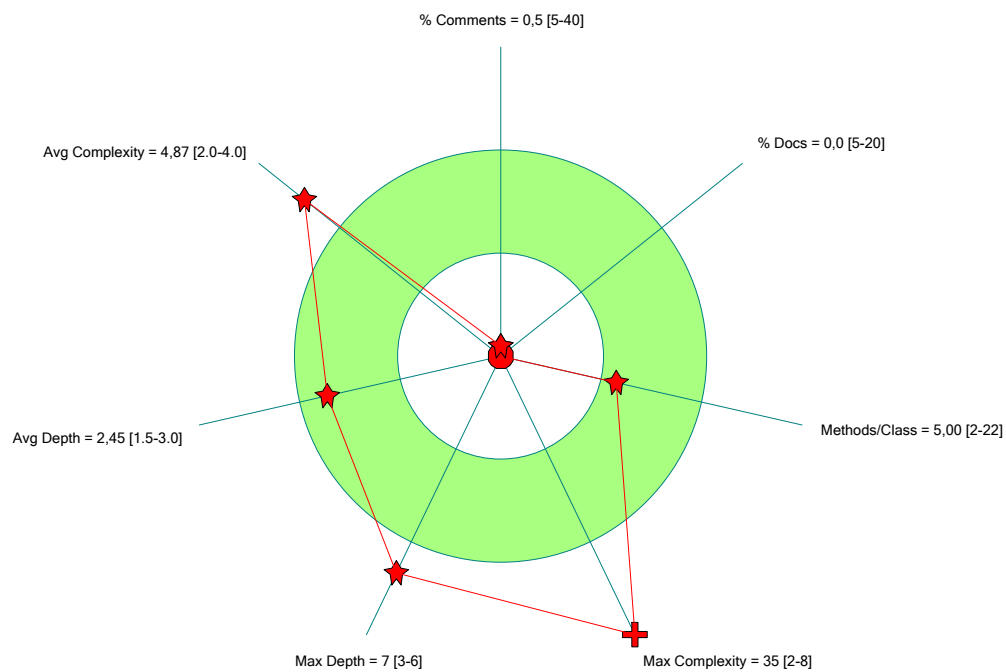


Abbildung 16: Metriken des alten BREPL-Routings

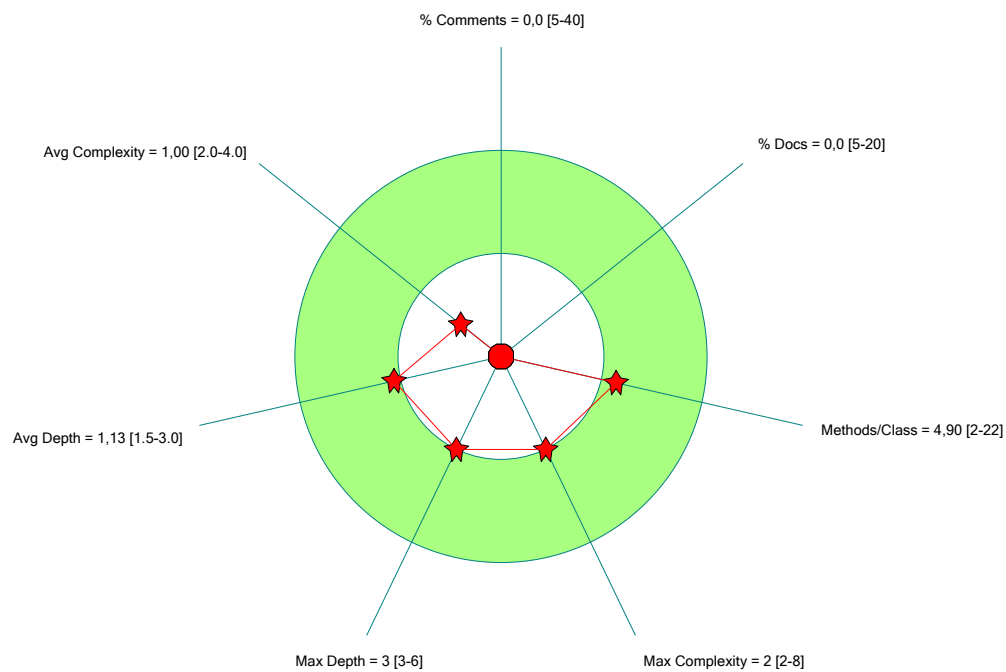


Abbildung 17: Metriken des neuen BREPL-Routings