



Abschlussprüfung Winter 2017/18

Fachinformatiker für Anwendungsentwicklung
Dokumentation zur betrieblichen Projektarbeit

AO-Beitragsportal

Webbasiertes Tool zur Ermittlung und Anzeige von Beitragsdaten

Abgabetermin: Vechta, den 06.12.2017

Prüfungsbewerber:

Jonas Hellmann
Kirchwiesen 7
49377 Vechta-Langförden



Ausbildungsbetrieb:

ALTE OLDENBURGER Krankenversicherung AG
Theodor-Heuss-Str. 96
49377 Vechta

Inhaltsverzeichnis

Abbildungsverzeichnis	III
Tabellenverzeichnis	III
Listings	III
Abkürzungsverzeichnis	IV
1 Einleitung	1
1.1 Projektbeschreibung	1
1.2 Projektziel	1
1.3 Projektbegründung	2
1.4 Projektschnittstellen	2
1.5 Projektabgrenzung	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	5
3.2.1 „Make or Buy“-Entscheidung	5
3.2.2 Projektkosten	5
3.2.3 Amortisationsdauer	6
3.3 Anwendungsfälle	7
3.4 Lastenheft	7
4 Entwurfsphase	7
4.1 Zielplattform	7
4.2 Architekturdesign	8
4.3 Entwurf der Benutzeroberfläche	8
4.4 Datenmodell	9
4.5 Geschäftslogik	9
4.6 Maßnahmen zur Qualitätssicherung	10
4.7 Deployment	10
4.8 Pflichtenheft	11
5 Implementierungsphase	11
5.1 Implementierung der Datenstrukturen	11
5.2 Implementierung der Geschäftslogik	12

5.3	Implementierung der Benutzeroberfläche	12
5.4	Testen der Anwendung	13
6	Abnahme- und Deploymentphase	13
6.1	Code-Review	13
6.2	Deployment und Einführung	14
7	Dokumentation	14
8	Fazit	14
8.1	Soll-/Ist-Vergleich	14
8.2	Lessons Learned	15
8.3	Ausblick	15
	Literaturverzeichnis	16
A	Anhang	i
A.1	Detaillierte Zeitplanung	i
A.2	Verwendete Ressourcen	ii
A.3	Aktivitätsdiagramme	iii
A.4	Amortisation	iv
A.5	Use-Case-Diagramm	v
A.6	Lastenheft	vi
A.7	Komponentendiagramm	vii
A.8	Mockups	viii
A.9	Entity-Relationship-Modell	ix
A.10	Klassendiagramm (Auszug)	x
A.11	Statische Codeanalyse	xi
A.12	Verteilungsdiagramm	xii
A.13	Gradle-Build-Datei (Auszug)	xiii
A.14	JBoss-Konfiguration	xiv
A.15	Jenkins-Konfiguration (Auszug)	xv
A.16	Screenshot der Build-Pipeline	xvi
A.17	Pflichtenheft (Auszug)	xvii
A.18	Tabellenmodell	xviii
A.19	Listing von Java-Code	xix
A.20	JSF-Vorlage	xxi
A.21	Screenshots	xxii
A.22	Test mit JUnit	xxiii
A.23	Benutzerdokumentation (Auszug)	xxiv
A.24	Entwicklerdokumentation	xxv

Abbildungsverzeichnis

1	Aktivitätsdiagramm zur Ermittlung der Kosten	iii
2	Aktivitätsdiagramm zur Ermittlung der Beitragsverläufe	iii
3	Grafische Darstellung der Amortisation	iv
4	Use-Case-Diagramm	v
5	Komponentendiagramm	vii
6	Mockup der Startseite	viii
7	Mockup der Beitrag-Seite	viii
8	Entity-Relationship-Modell	ix
9	Auszug des Klassendiagramms zur Veranschaulichung der Architektur	x
10	Übersicht der statische Codeanalyse mit JaCoCo, PMD, FindBugs und Checkstyle . .	xi
11	Ergebnisse von PMD und CheckStyle	xi
12	Verteilungsdiagramm der Anwendung	xii
13	Screenshot der Build-Pipeline von der Jenkins-Weboberfläche	xvi
14	Relationales Tabellenmodell	xviii
15	Screenshot der Anzeige von Beiträgen	xxii
16	Screenshot der Anzeige der überprüften Beitragsverläufe	xxii
17	Screenshot der Entwicklerdokumentation	xxv

Tabellenverzeichnis

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

Listings

1	Gradle-Build-Datei	xiii
2	JBoss-Konfigurationsdatei	xiv
3	Jenkins-Konfiguration mit der Scripted Pipeline	xv
4	Klasse BeitragsverlaufController	xix
5	Klasse BeitragsverlaufService	xix
6	Klasse CsvVerarbeiter	xix
7	Klasse BeitragsverlaufVerarbeiter	xx
8	JSF-Vorlage zur Erstellung der Oberflächen	xxi
9	Integrationstest der Klasse TarifJpaRepository mit JUnit	xxiii

Abkürzungsverzeichnis

AO	ALTE OLDENBURGER Krankenversicherung AG
API	Application Programming Interface
BAP	Beitragsanpassung
CDI	Contexts and Dependency Injection
CI	Continuous Integration
CSS	Cascading Style Sheets
CSV	Comma-separated values
EAP	Enterprise Application Platform
ERM	Entity-Relationship-Modell
HTML	Hypertext Markup Language
Java EE 7	Java Enterprise Edition 7
JPA	Java Persistence API
JSF	JavaServer Faces
MoSCoW	Must, Should, Could, Would
MVC	Model View Controller
PK	Provinzial Krankenversicherung
POJO	Plain Old Java Object
PT	Personentag
PUMA	Prozessunterstützung Mathematik
SCM	Source Code Managment
SVN	Subversion
TDD	Test Driven Development
VERSIS	Versicherungsinformationssystem (Bestandsführungssystem der AO)
VGH	Versicherungsgruppe Hannover
VVG	Versicherungsvertragsgesetz
XML	Extensible Markup Language

1 Einleitung

In der folgenden Projektdokumentation wird der Ablauf des Abschlussprojektes, das durch den Autor im Rahmen seiner Abschlussprüfung zum Fachinformatiker mit der Fachrichtung Anwendungsentwicklung durchgeführt wurde, erläutert. Das Projekt wird in der ALTE OLDENBURGER Krankenversicherung AG (AO) durchgeführt, welche der Ausbildungsbetrieb des Autors ist. Die AO ist eine private Krankenversicherung mit Sitz in Vechta, die zur Versicherungsgruppe Hannover (VGH) gehört. Bei der AO sind zurzeit 240 Mitarbeiter beschäftigt.¹ Zu den Produkten der AO zählen neben privaten Krankheitskostenvollversicherungen und Pflegeversicherungen auch Zusatzversicherungen, die die gesetzliche Krankenversicherung um zusätzliche Leistungen ergänzen. Ziel dieser Dokumentation ist es, die durchzuführenden Schritte des Projektes von der Planung bis zum Deployment zu erläutern und dies mit geeigneten Diagrammen und Dokumenten zu unterstützen.

1.1 Projektbeschreibung

Im Rahmen des Projektes zur Automatisierung des Prozesses der Beitragsanpassung (BAP) und zur Entlastung der Mathematik-Abteilung, zusammengefasst im Projekt Prozessunterstützung Mathematik (PUMA), soll eine Webanwendung mit Datenbankzugriff erstellt werden, die automatisch verschiedene Daten in Bezug auf die Beiträge und Kosten eines Versicherungstarifes ermittelt, verarbeitet und darstellt.

Der Fachbereich benötigt vor allem bei der Erstellung von Versicherungsangeboten eine Übersicht, die die aktuell geltenden Beiträge, die eine Person abhängig von dem gewählten Tarif und ihrem Alter bei Vertragsabschluss zahlen muss, darstellt. Außerdem ist die AO durch das Versicherungsvertragsgesetz (VVG) dazu verpflichtet, eine Übersicht darüber zu führen, wie hoch die einmaligen und laufenden Kosten bei einem Vertragsabschluss eines Versicherten in einem Tarif sind. Ebenfalls wird durch das VVG vorgeschrieben, dass eine Übersicht über die Beitragsverläufe der letzten zehn Jahre einer 35-jährigen Person für alle Tarife erstellt werden muss.

Diese Schritte sind jeweils zu einer BAP notwendig, die jährlich zum 01. Januar und eventuell zum 01. Juli durchgeführt wird. Dabei kann der Beitrag, der in einem Tarif bezahlt werden muss, z. B. aufgrund von höheren Kosten im Gesundheitsbereich angehoben oder gesenkt werden. Laut VVG ist es notwendig, eine Übersicht zu pflegen, in der ersichtlich wird, wann der Beitrag eines bestimmten Tarifes zum letzten Mal durch eine BAP verändert wurde. Außerdem möchte die Antragsabteilung eine Übersicht über die historischen BAPs eines Tarifs haben.

1.2 Projektziel

Ziel dieses Projektes ist die erfolgreiche Umsetzung einer neuen Webanwendung, mit der automatisiert verschiedene Daten für den Nutzer aufbereitet dargestellt und als Excel-Datei zum Download bereitgestellt werden. Dadurch sollen die Mathematik-Abteilung entlastet und momentan auf einem Netzlaufwerk abgelegte Excel-Dateien durch die Anwendung ersetzt werden.

¹Kennzahlen zum Stichtag 02.10.2017, vgl. ALTE OLDENBURGER KRANKENVERSICHERUNG AG [2017, S. 4].

1.3 Projektbegründung

Die Erstellung der vom Fachbereich benötigten und durch das [VVG](#) geforderten Dokumente erfolgt derzeit manuell durch die Mathematik-Abteilung, wobei es in der Vergangenheit mehrfach zu Fehlern beim manuellen Übertragen der Werte aus der Datenbank in eine Excel-Datei gekommen ist. Dies kann im Zweifel zu rechtlichen Problemen führen, da die Dateien neben der internen Nutzung auch an externe Partner weitergeleitet werden müssen. Eine Korrektur von Fehlern erfordert ein hohes Maß an Aufwand. Hinzu kommt bei der Erstellung der Dokumente auch der hohe zeitliche Aufwand. Obwohl alle Daten schon in einer Datenbank vorhanden sind, findet eine manuelle Übertragung statt. Ein weiteres Problem ist das Fehlen einer fachlichen Prüfung im aktuellen Prozess, sodass neben Übertragungsfehlern auch die möglichen fachlichen Fehler in den in der Datenbank hinterlegten Daten nicht erkannt werden können. Aufgrund dieser Probleme und der dadurch entstehenden hohen Fehleranfälligkeit hat sich die [AO](#) dazu entschieden, diesen Prozess zu automatisieren.

1.4 Projektschnittstellen

Um an die Daten zu gelangen, muss die Anwendung mit einer Oracle-Datenbank kommunizieren und Werte abfragen können. Diese befindet sich im internen Firmennetz und kann direkt angesprochen werden. Zusätzlich ist für die Erstellung der Beitragsverläufe indirekt eine Kommunikation mit dem Versicherungsinformationssystem ([VERSIS](#)) notwendig, da die Beitragsverläufe dort ermittelt werden und es die Möglichkeit geben muss, die daraus resultierenden CSV-Dateien in die Anwendung einzuspielen.

Zwar ist dieses Abschlussprojekt dem internen Projekt [PUMA](#) zugeordnet, lässt sich davon aber weitestgehend abgrenzen, da es keine direkten Schnittstellen gibt. Jedoch ist eine Abstimmung mit den Projektbeteiligten notwendig. Auch der Fachbereich soll insb. bei der Oberflächengestaltung mit in den Entwicklungsprozess einbezogen werden.

1.5 Projektabgrenzung

Da der Projektumfang begrenzt ist, ist die Erstellung der Beitragsverläufe und die Aufbereitung als CSV-Datei in [VERSIS](#) kein Bestandteil des Projektes.

2 Projektplanung

2.1 Projektphasen

Für die Umsetzung des Projektes standen 70 Stunden zur Verfügung, wie es die IHK Oldenburg vorschreibt.² Bevor mit dem Projekt gestartet wurde, fand eine Aufteilung auf verschiedene Phasen statt, die den kompletten Prozess der Softwareentwicklung abdecken. Eine grobe Zeitplanung mit den Hauptphasen lässt sich aus Tabelle 1 entnehmen. Eine detailliertere Zeitplanung mit den einzelnen Schritten der unterschiedlichen Phasen findet sich im Anhang [A.1: Detaillierte Zeitplanung](#) auf Seite i.

²Vgl. [IHK OLDENBURG \[2013, S. 2\]](#).

Projektphase	Geplante Zeit
Analysephase	8 h
Entwurfsphase	9 h
Implementierungsphase	42 h
Abnahme- und Deploymentphase	7 h
Dokumentationsphase	4 h
Gesamt	70 h

Tabelle 1: Grobe Zeitplanung

2.2 Ressourcenplanung

Anschließend wurden alle Ressourcen im Anhang [A.2: Verwendete Ressourcen](#) auf Seite [ii](#) aufgelistet. Die Planung umfasst dabei neben allen Hard- und Softwareressourcen, die im Rahmen des Projektes verwendet wurden, auch das Personal. Im Hinblick auf anfallende Kosten wurde bei der Wahl der verwendeten Software darauf geachtet, dass keine Lizenzkosten anfallen, benötigtes Know-How vorhanden ist und nicht gegen die Architekturrichtlinien der AO verstoßen wird. Diese legen zur Entwicklung unter anderem die Nutzung von Java Enterprise Edition 7 ([Java EE 7](#)) und den Jenkins-Server als Tool für Continuous Integration ([CI](#)) fest.

2.3 Entwicklungsprozess

Als letzter Schritt vor dem tatsächlichen Beginn des Projektes musste durch den Autor ein geeigneter Entwicklungsprozess gewählt werden. Durch diesen wird die Vorgehensweise bei der Umsetzung des Projektes definiert.

Grundsätzlich soll sich die Umsetzung des Projektes an den Phasen des Wasserfall-Modells orientieren, da die Anforderungen durch das VVG überwiegend eindeutig definiert sind. Beim Wasserfall-Modell werden die einzelnen Phasen der Software-Entwicklung nacheinander durchlaufen, wobei eine Rückkehr in eine vorherige Phase jederzeit möglich ist.³ Im Bereich der Oberflächengestaltung soll es jedoch durch regelmäßige Rücksprache mit dem Fachbereich einen agilen Prozess geben. Ebenso wird in Bezug auf die fachliche Logik und Prüfung der verarbeiteten Daten durch Plausibilitäten in iterativen Zyklen ein Anwendertest durch den Fachbereich schon in die Implementierungsphase integriert, so dass eine eigenständige Testphase entfallen kann. Durch diese Einbindung des Fachbereichs in den Entwicklungsprozess kann die Abnahmephase kürzer ausfallen.

Der gewählte Entwicklungsprozess soll dabei um die Praktik des Test Driven Development ([TDD](#)) erweitert werden, bei dem vor der Implementierung der fachlichen Logik jeweils ein automatisierter Test geschrieben wird. Diese Tests sollen sicherstellen, dass die Anwendung die erwartete Funktionalität tatsächlich umsetzt und nachträgliche Änderungen am Code keine unerwarteten Effekte nach sich ziehen, die das existierende Verhalten beeinträchtigen. Dadurch wird unter anderem auch die Motivation zum Refactoring, also dem Ändern des Codes ohne eine Veränderung des Verhaltens, erhöht.⁴ Damit

³Vgl. [SOMMERVILLE \[2007, S.97\]](#).

⁴Vgl. [LANGR U. A. \[2015, S. 3f., S.95, S. 153f\]](#).

das Prinzip des [TDD](#) möglichst optimal durchgeführt werden kann, wurde in der zeitlichen Planung unter Punkt [2.1 \(Projektphasen\)](#) viel Zeit für die Implementierung eingerechnet.

3 Analysephase

3.1 Ist-Analyse

Wie bereits unter [1.1 \(Projektbeschreibung\)](#) beschrieben setzt sich das Projekt AO-Beitragsportal aus vier verschiedenen Teilbereichen zusammen. In der Ist-Analyse soll jeweils herausgearbeitet werden, wie die Daten aktuell ermittelt und verarbeitet werden.

Die zum Eintritt in einen Tarif zu zahlenden Beiträge sind in einer Oracle-Datenbank abgelegt. Aktuell werden von der Mathematik-Abteilung manuell Excel-Tabellen mit Werten aus dieser Datenbank gefüllt, was zeitaufwendig ist und oftmals zu Fehlern führt, die korrigiert werden müssen. Diese Dateien werden dann an die EDV-Abteilung gegeben, damit sie mit Subversion ([SVN](#)) versioniert auf einem Netzlaufwerk abgelegt und vom Fachbereich eingesehen werden können.

Auch die einmaligen und laufenden Kosten, die für die durch das [VVG](#) geforderten Dateien benötigt werden, sind in dieser Datenbank vorhanden. Wieder werden diese Daten durch die Mathematik-Abteilung händisch in Excel-Dateien übertragen und an die EDV-Abteilung gegeben. Da diese Daten an externe Softwarehäuser geliefert werden müssen, die verschiedene Dateiformate vorschreiben, muss eine manuelle Umformatierung und Konvertierung vorgenommen werden, bevor diese Dateien ebenfalls versioniert auf einem Netzlaufwerk abgelegt werden. Im Anhang [A.3: Aktivitätsdiagramme](#) auf Seite [iii](#) ist ein Aktivitätsdiagramm abgebildet, in dem dieser Prozess dargestellt wird.

Für die Beitragsverläufe wird die Entwicklung des Beitrages für eine Person mit einem Alter von 35 Jahren und verschiedenen Tarif-Kombinationen in [VERSIS](#) 10 Jahre rückwirkend simuliert und in eine [CSV](#)-Datei exportiert. Anschließend wird im aktuellen Bestand nach einer tatsächlich versicherten Person mit vergleichbarem Alter und vergleichbaren Tarifkombinationen gesucht und manuell auf Auffälligkeiten verglichen. Nach eventuellen Korrekturen wird die CSV-Datei wieder in [VERSIS](#) eingespielt. Zur Veranschaulichung befindet sich im Anhang [A.3: Aktivitätsdiagramme](#) auf Seite [iii](#) ein Aktivitätsdiagramm.

Um das Datum zu finden, an dem ein Tarif zuletzt durch eine [BAP](#) angepasst wurde, werden die erstellten Beitragsverläufe jeweils manuell durchgeschaut, bis eine Veränderung gefunden wird. Eine historische Übersicht wird aktuell in einer Excel-Datei von der Antragsabteilung gepflegt.

In allen Bereichen fällt sehr viel überflüssige manuelle Arbeit an. Außerdem ist an keiner Stelle eine automatisierte fachliche Prüfung anhand von fest definierten Plausibilitäten vorgesehen, sodass Fehler nicht direkt erkannt werden können.

3.2 Wirtschaftlichkeitsanalyse

3.2.1 „Make or Buy“-Entscheidung

Die in dem Projekt verarbeiteten Daten betreffen das Kerngeschäft der Krankenversicherung der AO. Insbesondere die Kosten, die sich aus der Tarifikalkulation ergeben, sollen nicht an eine externe Anwendung gegeben werden, sondern in der eigenen Hand bleiben. Zudem muss die Anwendung mit dem von der AO entwickelten Bestandsführungssystem **VERSIS** zusammenarbeiten und kommunizieren, welches eine proprietäre Schnittstelle bereitstellt. Als strategische Entscheidung, das Kerngeschäft in **VERSIS** umzusetzen und sich nicht von einem Dienstleister abhängig zu machen und betriebsinterne Daten in externe Anwendungen zu exportieren, wurde sich bei diesem Projekt für eine Eigenentwicklung entschieden.

3.2.2 Projektkosten

Im Folgenden sollen die Kosten, die im Laufe des Projektes anfallen, kalkuliert werden. Neben den anfallenden Personalkosten des Entwicklers und weiterer Projektbeteiligter müssen auch die Aufwendungen für die verwendeten Ressourcen, die unter 2.2 (**Ressourcenplanung**) aufgeführt sind, eingeplant werden.

Da die tatsächlichen Personalkosten nicht herausgegeben werden dürfen, wird die Kalkulation anhand von Stundensätzen durchgeführt, die durch die Personalabteilung festgelegt wurden. Die aufgeführten Stundensätze beinhalten zum Großteil das Bruttogehalt sowie die Sozialaufwendungen des Arbeitgebers. Hinzu kommen die oben erwähnten Kosten, die für die Nutzung der Ressourcen anfallen.

Für einen Mitarbeiter wird ein Stundensatz von 25,00 € eingeplant. Der Stundensatz für einen Auszubildenden ist auf 10,00 € festgelegt. Als Satz für die Ressourcennutzung werden pauschal 15,00 € angenommen.

Die Durchführungszeit des Projektes beträgt 70 Stunden. In Tabelle 2: **Kostenaufstellung** sind die Kosten unterteilt nach den einzelnen Projektvorgängen aufgelistet, sowie summiert dargestellt, um die Gesamtkosten, die während des Projektes anfallen, zu erhalten. Diese belaufen sich auf 1990,00 €.

Vorgang	Zeit	Kosten pro Stunde	Kosten
Entwicklungskosten	70 h	10,00 € + 15,00 € = 25,00 €	1750,00 €
Fachgespräch	3 h	25,00 € + 15,00 € = 40,00 €	120,00 €
Abnahmetest	1 h	25,00 € + 15,00 € = 40,00 €	40,00 €
Code-Review	2 h	25,00 € + 15,00 € = 40,00 €	80,00 €
			1990,00 €

Tabelle 2: Kostenaufstellung

3.2.3 Amortisationsdauer

Im Folgenden soll die Amortisationsdauer betrachtet werden, also ab welchem Zeitpunkt sich die Entwicklung des Projektes amortisiert. Die Anschaffungskosten wurden bereits in Tabelle 2 ([Kostenaufstellung](#)) bestimmt. Durch die Automatisierung der Erstellung der Excel-Dateien lässt sich die Zeit dieser manuellen Arbeit einsparen. Außerdem entfällt die Versionierung und Nachbearbeitung sowie Kontrolle der Dateien wie unter 3.1 ([Ist-Analyse](#)) beschrieben. Die daraus resultierende Zeitersparnis ist in Tabelle 3: [Zeiteinsparung pro Jahr](#) detaillierter aufgeführt. Anhand von Erfahrungen aus den letzten Durchläufen der Prozesse wurden von den jeweils zuständigen Abteilungen die Vorgangszeiten ermittelt.

In der neuen Anwendung muss durch einen Mitarbeiter die Erstellung der neuen Dateien einmalig nach einer [BAP](#) angestoßen werden, wenn sichergestellt wurde, dass alle Daten in der Datenbank hinterlegt worden sind. Für diesen Arbeitsschritt werden 10 Minuten kalkuliert.

Grundsätzlich wird wie unter 3.2.2 ([Projektkosten](#)) ein Personentag (PT) mit 7,6 Stunden angenommen, was 456 Minuten entspricht. Es wird angenommen, dass die genannten Schritte im Durchschnitt 1,5 mal pro Jahr ausgeführt werden, da wie unter 1.1 ([Projektbeschreibung](#)) erwähnt eine [BAP](#) immer zum 01. Januar, aber nicht zwangsweise zum 01. Juli durchgeführt wird.

Für den Betrieb der Anwendung wurde von der Administration eine Pauschale von einem PT pro Jahr veranschlagt. Dies umfasst u. a. das Einspielen von Betriebssystem- oder Application-Server-Updates und die Wartung bei Ausfällen.

Vorgang	Anzahl pro Jahr	Zeit alt pro Vorgang	Zeit neu pro Vorgang	Einsparung pro Jahr
Erstellung der Excel-Dateien	1,5	2 PT $\hat{=}$ 912 min	10 min	1353 min
Versionierung der Dateien	1,5	0,5 PT $\hat{=}$ 228 min	-	342 min
Aufbereitung der Dateien	1,5	4 PT $\hat{=}$ 1824 min	-	2736 min
Kontrolle der Daten	1,5	1 PT $\hat{=}$ 456 min	-	684 min
Betrieb der Anwendung	1	-	1 PT $\hat{=}$ 456 min	-456 min
				4659 min

Tabelle 3: Zeiteinsparung pro Jahr

Dadurch ergibt sich eine jährliche Einsparung von $\frac{4659 \text{ min}}{60 \text{ min/h}} \cdot (25 + 15) \text{ €/h} = 3106,00 \text{ €}$.

Die Amortisationszeit beträgt also $\frac{1990,00 \text{ €}}{3106,00 \text{ €/Jahr}} \approx 0,64 \text{ Jahre} \approx 7,5 \text{ Monate}$. Eine grafische Darstellung befindet sich im Anhang [A.4: Amortisation](#) auf Seite [iv](#). Wie schon vorher beschrieben, findet dieser Prozess nur ein- bis zweimal jährlich statt. Aus der ermittelten Zeit lässt sich also ableiten, dass sich aufgrund der großen Einsparung an manueller Arbeit das Projekt schon mit der zweiten Ausführung amortisiert hat, da der Prozess durchschnittlich alle 7,5 Monate stattfindet. Daher lässt sich das Projekt als wirtschaftlich einordnen und soll umgesetzt werden.

3.3 Anwendungsfälle

Um eine grobe Übersicht zu erhalten, wie verschiedene Abteilungen mit der Anwendung arbeiten können sollen und welche Anwendungsfälle aus Endanwendersicht abgebildet werden müssen, wurde im Laufe der Analyse-Phase ein Use-Case-Diagramm erstellt. Dieses befindet sich im Anhang [A.5: Use-Case-Diagramm](#) auf Seite [v](#).

Die Akteure, die später mit der Anwendung arbeiten werden, lassen sich in einen normalen Mitarbeiter des Fachbereichs und Mitarbeiter der Abteilungen Vertrieb, Mathematik und EDV aufteilen. Im Diagramm ist eine Vererbungshierarchie erkennbar, da jeder Akteur mit zusätzlichen Anwendungsfällen auch jeweils die des vorhergehenden Akteurs mit einbezieht. Daraus lassen sich die verschiedenen Rollen und Berechtigungen für die Anwendung ableiten.

3.4 Lastenheft

Am Ende der Analysephase wurde gemeinsam mit den Abteilungen Vertrieb, Mathematik und Antrag ein Lastenheft erstellt. In dem Lastenheft sind alle zu berücksichtigenden Anforderungen des Auftraggebers an die zu implementierende Anwendung sortiert nach der Wichtigkeit der Umsetzung festgehalten. Die Formulierung wurde mit Hilfe der Must, Should, Could, Would (MoSCoW)-Methode umgesetzt. Das heißt, dass in jeder Anforderung die Wichtigkeit aufgeschlüsselt nach „muss“, „soll“, „kann“ oder „würde gerne“ festgehalten wird.⁵ Anhand dieser Methodik wurde auch die Priorisierung vorgenommen. Das vollständige Lastenheft befindet sich im Anhang [A.6: Lastenheft](#) auf Seite [vi](#).

4 Entwurfsphase

4.1 Zielplattform

Wie bereits unter [1.1 \(Projektbeschreibung\)](#) erwähnt, soll das Abschlussprojekt als eigenständige Webanwendung entwickelt werden. Dies macht das Deployment und die Aktualisierung auf eine neue Version einfacher. Die Daten, auf die zugegriffen werden soll, sind in einer bereits bestehenden Oracle-Datenbank abgelegt. Auf die Installation eines weiteren Datenbanksystems kann daher verzichtet werden.

Als Programmiersprache soll Java eingesetzt werden. Die Auswahl dieser Sprache ergibt sich aus den Architekturrichtlinien, die durch die [AO](#) vorgegeben sind. Als Spezifikation zur Entwicklung der Webanwendung wird [Java EE 7](#) benutzt, da diese auch schon in anderen Projekten in der [AO](#) zum Einsatz gekommen ist. Laufen wird die Anwendung auf einem JBoss Enterprise Application Platform ([EAP](#)) 7, da die [AO](#) bereits Lizenzen für den Betrieb und Support dieses Application Servers besitzt. Zudem ist durch den Einsatz in anderen Projekten bereits Erfahrung mit dem JBoss [EAP](#) vorhanden, wobei sich dieser als geeignet erwiesen hat.

⁵Vgl. [HAUGHEY, DUNCAN \[2014\]](#).

4.2 Architekturdesign

Als Basis für das architekturelle Design soll das Model View Controller (MVC)-Architekturmuster dienen. Dieses soll aber durch Komponenten aus dem [Java EE 7](#)-Standard erweitert werden. Die Architektur ist im Anhang [A.7: Komponentendiagramm](#) auf Seite [vii](#) dargestellt.

Grundsätzlich besteht die Anwendung aus drei Schichten. Die Model-Schicht enthält die Domäne, das Repository, die Services und das View-Model. In der Domäne sind neben den Entitäten auch andere Klassen vorhanden, in denen die fachliche Logik abgebildet wird. In den Services werden die einzelnen Domänen-Klassen so miteinander verbunden, dass die Use-Cases umgesetzt werden. Hier wird die Schnittstelle zur Oberfläche bereitgestellt. Das Repository beinhaltet lediglich Interfaces, um eine Schnittstelle für eine Anbindung an die Persistenzschicht vorzugeben. Ein Plain Old Java Object (POJO) beschreibt eine Klasse, die keine Logik enthält, sondern nur Daten hält und Getter und Setter besitzt. Das View-Model besteht ausschließlich aus diesen POJOs und nimmt die in der Web-Oberfläche eingegebenen Daten entgegen. Dieses wird später gegebenenfalls auf ein Domänen-Objekt gemappt.

Die Aufgabe der Persistenz-Schicht ist, die Funktionalität bereitzustellen, um Daten in einer Datenbank persistieren zu können. In dieser Anwendung wird dafür die Java Persistence API (JPA) verwendet, die Teil des [Java EE 7](#)-Standards ist. In der View-Schicht wird mit JavaServer Faces (JSF) ein weiterer Bestandteil des erwähnten Standards verwendet. Die Controller-Komponente dient abschließend als eine Verbindung zwischen den im Service bereitgestellten Funktionalitäten und der sichtbaren Web-Oberfläche.

Wichtig zu beachten sind dabei auch die Abhängigkeiten der Schichten und Komponenten. Es wird darauf geachtet, dass die reine fachliche Logik keine Abhängigkeiten auf andere Komponenten hat. Gleiches gilt für die Model-Schicht als Ganzes. Durch die bereitgestellten Schnittstellen über die Service- und Repository-Komponenten ist ein Austausch der Datenhaltung und Darstellung jederzeit sehr einfach möglich.

4.3 Entwurf der Benutzeroberfläche

Wie unter [2.3 \(Entwicklungsprozess\)](#) beschrieben, soll es im Bezug auf die Oberfläche eine Rücksprache mit dem Fachbereich geben, da dieser vorrangig damit arbeiten wird. Dazu wurden zuerst Entwürfe angefertigt. Diese Mockups befinden sich im Anhang [A.8: Mockups](#) auf Seite [viii](#). Die Anwendung wird im Intranet laufen und alle Mitarbeiter der Fachabteilung arbeiten an Desktop-Computern. Aus diesem Grund besteht keine konkrete Anforderung nach einer Responsivität.

Die Webseite soll einen einheitlichen Grundaufbau haben, zu dem der Header, der aus dem Logo der AO und einer Navigation besteht, sowie der Footer gehören. Die Inhalte werden dann in einem in weiß abgesetzten Bereich angezeigt, der von einem hellblauen Rahmen umschlossen wird. Einige dargestellte Funktionalitäten sollen dem Benutzer nur dann angezeigt werden, wenn die entsprechende Berechtigung gesetzt ist.

Die Absprache mit dem Fachbereich hat ergeben, dass die Entwürfe so umgesetzt werden können, die Navigation aber möglichst einfach gehalten werden soll und nicht zu viele Seitenwechsel erforderlich sein dürfen, um eine bestimmte Unterseite zu erreichen.

4.4 Datenmodell

Das Datenmodell besteht aus den Entitäten **Tarif**, **Beitragsanpassung**, **Beitrag**, **Beitragsverlauf**, **Kosten** und **Mandant**. Der **Tarif** speichert dabei den Namen eines Tarifes. Zur Ermittlung der Historie der **BAP**-Daten besteht eine Beziehung zur **Beitragsanpassung**, wo jeweils eine **BAP** mit ihrem Start- und Endedatum abgebildet ist. An der Beziehung hängt das Attribut **Beobachtungseinheit**, da eine **BAP** z. B. abhängig vom Geschlecht durchgeführt werden kann. Dabei handelt es sich um eine m:n-Beziehung.

Ein **Tarif** ist eindeutig einem **Mandanten** zuzuordnen. Als **Mandant** werden hier die beiden Gesellschaften **AO** und Provinzial Krankenversicherung (**PK**) bezeichnet. Zu einem **Mandanten** können aber mehrere **Tarife** gehören, weswegen hier eine 1:n-Beziehung besteht.

Zwischen **Tarif** und **Beitrag** besteht eine 1:n-Beziehung, da ein **Beitrag** mit dem vom Alter und Geschlecht abhängig zu zahlenden Betrag genau einem **Tarif** zugeordnet ist. Eine weitere Beziehung unterhält der **Beitrag** zu den **Beitragsanpassungen**. Da ein **Beitrag** nur ab einer **BAP** gültig ist, besteht auch hier wieder eine 1:n-Beziehung.

Zwischen den beiden Entitäten **Beitrag** und **Kosten** besteht eine 1:1-Beziehung, wobei es sein kann, dass es zu einem **Beitrag** auch gar keine **Kosten** gibt, da bei einigen Tarifen keine Kosten berechnet werden dürfen. Die **Kosten** bilden ab, wie hoch die laufenden Abschluss- und Verwaltungs- sowie die einmaligen Verwaltungskosten sind.

Die Entität **Beitragsverlauf** speichert Verläufe mit jeweils 10 Werten, die in einem bestimmten Jahr für ein Geschlecht gültig sind. Diese sind eindeutig einem **Tarif** zugeordnet, wobei ein **Tarif** in verschiedenen Jahren zu mehr als einem **Beitragsverlauf** gehören kann. Es besteht also eine 1:n-Beziehung.

Diese Beziehungen sind im Anhang [A.9: Entity-Relationship-Modell](#) auf Seite [ix](#) als Entity-Relationship-Modell (**ERM**) dargestellt. Anhand dieses Modells sollen später u. a. die Klassen der Domäne implementiert werden.

4.5 Geschäftslogik

Aufgrund der testgetriebenen Entwicklung ist es nicht möglich, vor Beginn der Implementierung der Anwendung ein Klassendiagramm mit den Klassen, mit denen die Geschäftslogik umgesetzt werden soll, zu entwerfen. Diese werden sich auf Grundlage der Tests mit anschließendem Refactoring erst ergeben. Der grundsätzliche Aufbau der Anwendung mit beteiligten Komponenten und deren Abhängigkeiten wurde bereits unter [4.2 \(Architekturdesign\)](#) näher erläutert. Für eine nachträgliche Dokumentation der Geschäftslogik wurde ein Klassendiagramm aus dem Programmcode heraus generiert, das sich als Ausschnitt im Anhang [A.10: Klassendiagramm \(Auszug\)](#) auf Seite [x](#) befindet. Dort lässt

sich die schon unter 4.2 ([Architekturdesign](#)) beschriebene Trennung der einzelnen Komponenten und deren Abhängigkeiten untereinander erkennen. Das Model hat keine externen Abhängigkeiten und bietet in diesem Ausschnitt Schnittstellen über die Services `BeitragsverlaufService` und `TarifService` und das Interface `TarifRepository` an.

4.6 Maßnahmen zur Qualitätssicherung

Während der Durchführung des Projektes wurden Maßnahmen zur Qualitätssicherung ergriffen. Das Projekt wurde wie unter 2.3 ([Entwicklungsprozess](#)) beschrieben testgetrieben entwickelt, um fachliche Fehler möglichst von Beginn an zu vermeiden und sicherzustellen, dass auch bei nachträglichen Änderungen die Qualität der Anwendung nicht abnimmt. Zusätzlich wurde ein Code-Review mit dem Ausbilder durchgeführt, wobei neben der fachlichen Richtigkeit auch die technische Umsetzung überprüft werden sollte. Dieses Vorgehen wird unter 6.1 ([Code-Review](#)) näher beschrieben. Schon zu Beginn der Implementierungsphase wurde ein Jenkins-Build eingerichtet, um [CI](#) sicherzustellen. Bei diesem Build werden automatisiert verschiedene Tools zur statischen Code-Analyse ausgeführt (siehe 4.7). Ein Screenshot dieser Analysen befindet sich im Anhang A.11: [Statische Codeanalyse](#) auf Seite xi.

4.7 Deployment

Bei dem Projekt handelt es sich um eine Web-Anwendung, weswegen ein Deployment auf einen Web-Server notwendig ist. Bevor dies jedoch geschah, wurden zuerst noch einige andere Schritte ausgeführt. Diese sind im Anhang A.12: [Verteilungsdiagramm](#) auf Seite xii visualisiert.

Als Entwicklungsumgebung wird *Eclipse Oxygen for Java EE Developers* eingesetzt. Das Projekt wird mit *Gradle* gebaut. Dabei handelt es sich um ein Build-Management-Tool, mit dem Anwendungen durch einen automatisierten Prozess erzeugt werden können. Die für dieses Projekt erstellte Build-Datei, in der alle Schritte zum vollständigen Bauen der Anwendung inklusive der benötigten Abhängigkeiten angegeben sind, befindet sich im Anhang A.13: [Gradle-Build-Datei \(Auszug\)](#) auf Seite xiii.

Auf dem Build-Server wird eine Instanz eines *Tomcat*-Servers betrieben. In dieser Umgebung laufen ein *Jenkins*-Server, ein *Source Code Management (SCM)*-Server sowie *Artifactory*, ein Programm zur Verwaltung von Artefakten. *Jenkins* ist ein webbasiertes Tool zur [CI](#).

Die Anwendung wird von dem lokalen Computer in das [SCM](#) deployt und anschließend vom *Jenkins* gebaut. Dazu muss im *Jenkins* ein Job für diese Anwendung angelegt werden. *Gradle* lädt beim Build die Abhängigkeiten aus *Artifactory* herunter. War das Bauen erfolgreich, wird die Anwendung auf den Web-Server deployt, auf dem ein *JBoss*-Application-Server betrieben wird. Der Application-Server nutzt eine *Oracle*-Datenbank. Diese Verbindung wird automatisch hergestellt, sobald man eine sogenannte Datasource in der JBoss-Konfiguration eingerichtet hat. Das Skript, mit dem dies eingestellt wird, befindet sich im Anhang A.14: [JBoss-Konfiguration](#) auf Seite xiv. Der Client kann im Anschluss mit *Firefox* den Webserver kontaktieren und nach der Anwendung fragen.

Dieses Deployment soll vollständig automatisiert ablaufen. Der Jenkins-Server wurde dazu so konfiguriert, dass eine Veränderung im Repository des [SCM](#) den Start eines Builds bewirkt. Die Konfiguration

des Jenkins-Jobs befindet sich in einer Datei mit dem Namen *Jenkinsfile*, die sich im Anhang [A.15: Jenkins-Konfiguration \(Auszug\)](#) auf Seite [xv](#) befindet. Es wurde bei der Erstellung der Datei die sogenannte *Scripted Pipeline* verwendet. Im Anhang [A.16: Screenshot der Build-Pipeline](#) auf Seite [xvi](#) befindet sich ein Screenshot vom Jenkins, der diese Pipeline grafisch darstellt. Als letzter Schritt, welcher durch das *Jenkinsfile* festgelegt wird, findet ein Deployment auf einen Test-Server der AO statt.

4.8 Pflichtenheft

Auf Grundlage der ausgearbeiteten Entwürfe wurde am Ende der Entwurfsphase ein Pflichtenheft erstellt. Dieses baut auf dem Lastenheft auf und erfasst die konkrete Umsetzung zu den in [3.4 \(Lastenheft\)](#) ermittelten Anforderungen. Es dient dazu, am Ende des Projektes überprüfen zu können, ob alle Anforderungen umgesetzt wurden und kann auch schon während der Entwicklung als Leitfaden dienen. Abgebildet ist das Pflichtenheft im Anhang [A.17: Pflichtenheft \(Auszug\)](#) auf Seite [xvii](#).

5 Implementierungsphase

Zu Beginn der Implementierungsphase musste zuerst einmal das Java-Projekt angelegt werden. Um den üblichen Konventionen in Java zu entsprechen, lässt sich dies mit einem Gradle-Befehl `gradle init -type java-library` umsetzen. Ist in der `gradle.build`-Datei (siehe [4.7](#)) das Eclipse-Plugin aktiviert, lässt sich mit `gradle eclipse` ein importierbares Eclipse-Projekt generieren. Dies kann dann als Grundlage benutzt werden.

5.1 Implementierung der Datenstrukturen

Bei der Implementierung der Domänen-Klassen, die die Daten aus der Persistenz-Schicht halten, wurde sich an dem [ERM](#) (siehe [4.4](#)) orientiert. Jedoch mussten kleine Anpassungen gemacht werden, da die Anwendung an eine bereits bestehende Datenbank angeschlossen werden musste. Das entsprechende relationale Tabellenmodell befindet sich im Anhang [A.18: Tabellenmodell](#) auf Seite [xviii](#). **Tarif** und **Mandant** entsprechen dabei den Tabellen `DM_TARIF` und `S_MDT`. Die Tabelle `KV_RG` lässt sich der Entität **Beitrag** zuordnen und `KV_KOSTEN` stellt die **Kosten** dar. Die im [ERM](#) modellierte Entität **Beitragsverlauf** konnte ohne Änderungen als neue Tabelle in der Datenbank angelegt werden.

Daraus hat sich ergeben, dass es eine Beitragsanpassung nicht als eigenständige Entität gibt, sondern lediglich Datumsangaben an den Entitäten **Beitrag** und **Kosten** gespeichert werden. Zudem ist die Abbildung einer Beziehung zwischen **Beitrag** und **Kosten** nicht umsetzbar, da die gegebene Datenbankstruktur dies nicht hergibt. `KV_RG` bezieht sich immer auf ein bestimmtes Geschlecht und Alter, während in `KV_KOSTEN` ein Zeitraum und verschiedene Geschlechter angegeben werden können. Um zu einem Beitrag die Kosten zu erhalten, muss dies über einen Service stattfinden, der sich die Daten aus der Datenbank holt. Ein Zuordnung erfolgt dann über den Tarif, das Geschlecht und das Alter sowie Beginn- und Ablaufdatum.

Die Verknüpfung zwischen den Klassen und den Tabellen wurde mit [JPA](#)-Annotationen umgesetzt. Jede Klasse muss als `@Entity` gekennzeichnet werden und kann mit `@Table(name="")` einer Tabelle zugeordnet werden. Um die einzelnen Attribute mit den passenden Spalten zu verbinden, wurde die `@Column(name="")`-Annotation verwendet. Hibernate wurde nun so konfiguriert, dass die Datenbankstruktur nach einem Neustart des Application Servers gegen die in der Anwendung genutzten Entitäten validiert wird. Dies ist durch folgenden Eintrag in der Datei `persistence.xml`, also der Konfiguration der Persistenz-Schicht möglich.

```
<property name="hibernate.hbm2ddl.auto" value="validate" />
```

5.2 Implementierung der Geschäftslogik

Die Geschäftslogik umfasst allgemein die Abbildung der fachlichen Anforderungen und Verarbeitung der Daten. Im ersten Schritt wurden alle unter [3.3 \(Anwendungsfälle\)](#) ermittelten Anwendungsfälle in der Service-Schicht dargestellt. Gleichzeitig wurden auch die entsprechenden Test-Klassen angelegt, um der testgetriebenen Entwicklung gerecht zu werden. Um die in den Services vorhandenen Methoden zu implementieren, war es notwendig, den Zugriff auf die Daten umzusetzen. Dafür wurden für den Zugriff auf die verschiedenen Entitäten jeweils zuerst die Interfaces aus dem *Repository*-Package (siehe [4.2 \(Architekturdesign\)](#)) angelegt. Diese Interfaces wurden dann im *JPA*-Package implementiert. Um die Datenbankschicht möglichst einfach austauschen zu können, wurden in den Services anschließend die Interfaces als Instanzvariablen festgelegt und mit `Inject` annotiert. Durch diese Annotation wird das Objekt automatisch durch Contexts and Dependency Injection ([CDI](#)) instanziiert und da nur eine einzige Klasse das jeweilige Interface implementiert, wird diese für das Erstellen des Objektes verwendet. Mit dem Datenbank-Zugriff war es möglich, die Geschäftslogik testgetrieben umzusetzen.

Ein Teil der Logik bestand darin, eine hochgeladene Datei, die Beitragsverläufe im CSV-Format beinhaltet, in Objekte der Klasse `Beitragsverlauf` zu konvertieren. Diese mussten anschließend auf die Entwicklung der Beiträge überprüft werden, da eine zu große Veränderung auf Fehler hindeutet. Dazu nutzt der `BeitragsverlaufController` entsprechende Schnittstellen des `BeitragsverlaufServices`, der den Aufruf an die zuständige Domain-Klasse weiterleitet. Im Anhang [A.19: Listing von Java-Code](#) auf Seite [xix](#) ist der entsprechende Code zu sehen.

5.3 Implementierung der Benutzeroberfläche

Auf Grundlage der unter [4.3 \(Entwurf der Benutzeroberfläche\)](#) beschriebenen Mockups konnte die Benutzeroberfläche implementiert werden. Umgesetzt wurde die Oberfläche mit der unter [4.2 \(Architekturdesign\)](#) genannten *JSF*-Technologie aus dem *Java EE 7*-Standard. *JSF* beinhaltet grundsätzlich selbst keinen Java-Code, sondern ist eine Extensible Markup Language (*XML*)-Datei, die nach Hypertext Markup Language (*HTML*) gerendert wird, damit die Anwendung im Browser angezeigt werden kann. Über Ausdrücke, die von `#{}-Blöcken` umgeben werden, lässt sich auf Methoden der Controller zugreifen, um so eine Verknüpfung zum Code herzustellen. Mit folgendem Tag lässt sich beispielsweise über eine Liste aller Tarife iterieren:

```
<ui:repeat var="tarif" value="#{tarifController.tarife}"></ui:repeat>
```

Um die Einheitlichkeit der einzelnen Seiten zu gewährleisten, ist es in [JSF](#) möglich, Vorlagen anzulegen und diese dann zu verwenden. Diese Vorgehensweise wurde bei allen implementierten Oberflächen genutzt. Diese Vorlage ist im Anhang [A.20: JSF-Vorlage](#) auf Seite [xxi](#) zu finden.

Die Gestaltung der Oberfläche wurde dann mit eingebundenen Cascading Style Sheets ([CSS](#))-Dateien umgesetzt. Unter anderem wurde dabei das [CSS-Framework Bootstrap](#)⁶ eingesetzt. Im Anhang [A.21: Screenshots](#) auf Seite [xxii](#) befinden sich Screenshots der Anwendung, die nach der Implementierungsphase entstanden sind.

Bei der Gestaltung der Oberfläche wie auch während der gesamten Entwicklung wurden die Softwareergonomie-Richtlinien beachtet. Dazu zählt vor allem die Aufgabenangemessenheit durch eine Minimierung der Interaktionen und geeigneter Funktionalität und die Fehlertoleranz. So werden z. B. bei Fehlern während der Validierung der CSV-Datei dem Benutzer sprechende Fehlermeldungen angezeigt. Durch die Umsetzung von Shortcuts zur Navigation ist auch eine erweiterte Steuerung des Programmes möglich. Der Shortcut **Shift+B** öffnet beispielsweise die Seite für die Beiträge.

5.4 Testen der Anwendung

Wie schon unter [2.3 \(Entwicklungsprozess\)](#) erwähnt, wurde die Anwendung testgetrieben entwickelt. So wurden während der Implementierungsphase stetig Tests geschrieben. Diese Tests lassen sich in drei verschiedene Kategorien einteilen. Die Unit-Tests überprüfen die Funktionalität einer einzelnen Klasse. In den Integrationstests wird die Kommunikation zwischen den verschiedenen Komponenten kontrolliert. In den Oberflächen-Tests wird abschließend die Interaktion des Benutzers mit der Anwendung simuliert und auf mögliche Fehler getestet. Im Anhang [A.22: Test mit JUnit](#) auf Seite [xxiii](#) ist ein Test mit *JUnit* abgebildet. Dabei handelt es sich um einen Integrationstest, der die Kommunikation mit der Persistenzschicht überprüft. Dabei wird anhand der eingespielten Testdaten überprüft, ob es möglich ist, nach allen Tarifen oder nach einem bestimmten Tarif, der durch seine ID identifiziert wird, zu suchen. Um nicht die Daten der Produktions-Datenbank zu manipulieren, wird für diesen Test eine Test-Datenbank mit einer identischen Struktur eingesetzt. Dafür wird die Datenbank „Apache Derby“ verwendet, da diese sehr leichtgewichtig und ohne große Installation nutzbar ist.

6 Abnahme- und Deploymentphase

6.1 Code-Review

Bevor die Anwendung abschließend zum Test dem Fachbereich vorgeführt wurde, fand ein Code-Review durch den Ausbilder statt. Dabei wurde neben fachlichen und technischen Fehlern vor allem auch auf die Verständlichkeit des Codes in Bezug auf die Benennung von Variablen und Komplexität der Methoden geachtet. Da der Ausbilder mit dem Thema des Projektes vertraut war, war kein Vorgespräch zur Erläuterung der Anwendung notwendig. Die Anmerkungen wurden als TODO-Kommentare direkt in den Code geschrieben. Nach dem Review konnten in einer Nachbesprechung

⁶Vgl. <http://getbootstrap.com/>, Zugriff am 21.11.2017.

offene Fragen bezüglich der Anmerkungen geklärt werden. Ziel des Reviews war es, die Qualität des Codes zu steigern und diesen für andere Entwickler verständlicher zu machen.

6.2 Deployment und Einführung

Das Verfahren des automatisierten Deployments wurde unter [4.7 \(Deployment\)](#) schon ausführlich beschrieben. Für die abschließende Einführung musste im Jenkinsfile das Ziel des Deployments geändert werden. Dort war bisher ein JBoss-Server eingetragen, der als Entwicklungsumgebung gedient hat. Anschließend war die Anwendung unter `http://ao-intranet/ao-beitragsportal` erreichbar. Damit die Endanwender schneller auf die Anwendung zugreifen können, wurde auf dem Desktop eine Verknüpfung angelegt. Da der Fachbereich schon während der Entwicklung durch Rücksprachen die Anwendung kennengelernt hat, konnte auf eine Benutzerschulung verzichtet werden.

7 Dokumentation

Die Dokumentation der Anwendung setzt sich neben der Projektdokumentation, in der die während der Umsetzung des Projektes durchlaufenden Phasen beschrieben werden, auch aus der Benutzer- und Entwicklerdokumentation zusammen.

Das Benutzerhandbuch soll dem Endanwender helfen, sich schnell in dem Programm zurechtzufinden und ihn bei Problemen unterstützen. Es enthält Informationen über die Funktionsweise und erklärt die Navigation in der Anwendung. Dabei wird ausführlich die Benutzung des Programmes dokumentiert und anhand von Screenshots erklärt. Ein Ausschnitt befindet sich im Anhang [A.23: Benutzerdokumentation \(Auszug\)](#) auf Seite [xxiv](#).

Die Entwicklerdokumentation beinhaltet eine detaillierte Beschreibung aller implementierten Klassen. Dazu gehören auch deren Attribute und Methoden sowie untereinander bestehende Abhängigkeiten. Diese Dokumentation soll bei der Weiterentwicklung der Anwendung oder einer Anpassung durch einen anderen Entwickler als Übersicht und Nachschlagewerk dienen. Die Dokumentation wurde direkt im Code in Form von Javadoc verfasst. Daraus lässt sich mit Hilfe des Gradle-Befehls `gradle javadoc` automatisiert HTML generieren. Ein Screenshot der generierten Entwicklerdokumentation lässt sich im Anhang [A.24: Entwicklerdokumentation](#) auf Seite [xxv](#) finden.

Zusätzlich wurde für Dokumentationszwecke ein Klassendiagramm generiert, um einem anderen Entwickler die Möglichkeit zu geben, sich schnell einen Überblick über die Anwendung zu schaffen. Ein Ausschnitt von diesem Diagramm befindet sich im Anhang [A.10: Klassendiagramm \(Auszug\)](#) auf Seite [x](#).

8 Fazit

8.1 Soll-/Ist-Vergleich

Abschließend soll die Planung des Projektes rückblickend überprüft werden. Dazu wird die benötigte mit der im Vorfeld kalkulierten Zeit verglichen. Wie in [Tabelle 4](#) zu erkennen ist, konnte die Zeitplanung

8 Fazit

bis auf wenige Ausnahmen eingehalten werden. Das gesamte Projekt konnte in der von der IHK Oldenburg vorgegebenen Zeit von 70 Stunden umgesetzt werden, da die geringfügigen Abweichungen von der Planung stets kompensiert werden konnten.

Die Abnahmephase fiel um eine halbe Stunde kürzer aus, da das für zwei Stunden eingeplante Code-Review durch den Ausbilder schneller durchgeführt und besprochen werden konnte. Zudem konnte auch bei dem Deployment Zeit eingespart werden, da die Zielumgebung schon existierte und ein automatisches Deployment der Anwendung eingerichtet werden konnte. Zudem wurde für die Dokumentation eine halbe Stunde weniger Zeit benötigt, da die Entwicklerdokumentation schon während der Implementierung in Form von JavaDoc gepflegt wurde. Für die Implementierung hingegen war die eingeplante Zeit nicht ausreichend und wurde um 1,5 Stunden überschritten. Grund dafür war die erhöhte Komplexität bei der Implementierung der Datenstrukturen wegen des Mappings auf die vom Datenmodell abweichende Persistenzschicht.

Phase	Geplant	Tatsächlich	Differenz
Analysephase	8 h	8 h	0 h
Entwurfsphase	9 h	9 h	0 h
Implementierungsphase	42 h	43,5 h	+1,5 h
Abnahme- und Deploymentphase	7 h	6 h	-1 h
Dokumentationsphase	4 h	3,5 h	-0,5 h
Gesamt	70 h	70 h	0h

Tabelle 4: Soll-/Ist-Vergleich

8.2 Lessons Learned

Anhand der Umsetzung dieses Abschlussprojektes konnte der Autor wertvolle Erfahrungen über die Arbeit an einem Projekt mit allen dazugehörigen Arbeitsschritten sammeln. Vor allem in der Planungs- und Entwurfsphase sowie in der Absprache mit dem Fachbereich konnten neue Erkenntnisse gewonnen werden. Auch in Bezug auf die Programmierung konnten mit dem Konzept des [TDD](#) für die tägliche Arbeit wichtige Grundlagen vermittelt werden. Spannend war vor allem der Einsatz von verschiedenen Frameworks, um alle Komponenten der [Java EE 7](#)-Anwendung testen zu können. Zudem konnte der Autor auch in Bezug auf die Enterprise Edition von Java viel Neues lernen. Dazu gehört unter anderem [CDI](#), die Anbindung an Datenquellen mit [JPA](#) oder die Gestaltung der Oberflächen mit [JSF](#).

8.3 Ausblick

Auch wenn die an dieses Projekt gestellten Anforderungen umgesetzt wurden, ist nicht auszuschließen, dass zukünftig neue Themenfelder, die dem AO-Beitragsportal zugeordnet werden können, entstehen. Diese könnten dann sehr einfach mit in diese Anwendung integriert werden, da der Aufbau modular ist. Denkbar ist auch, dass nach erfolgreicher Einführung die [PK](#), ein Schwesterunternehmen der [AO](#), an einer solchen Webanwendung interessiert ist.

Literaturverzeichnis

Alte Oldenburger Krankenversicherung AG 2017

ALTE OLDENBURGER KRANKENVERSICHERUNG AG: *Geschäftsbericht über das Jahr 2016*. https://www.alte-oldenburger.de/files/medien/Ueber_Uns/Geschaeftsberichte/Geschaeftsbericht2016ALTE.OLDENBURGER.Krankenversicherung.AG.pdf. Version: 2017. – Zugriff am: 02.10.2017

Haughey, Duncan 2014

HAUGHEY, DUNCAN: *MoSCoW Method*. <https://www.projectsmart.co.uk/moscow-method.php>. Version: 2014. – Zugriff am: 16.10.2017

IHK Oldenburg 2013

IHK OLDENBURG: *Merkblatt zur Abschlussprüfung der IT-Berufe*. <https://ihk-oldenburg.de/blob/olihk24/geschaeftsfelder/AusbildungWeiterbildung/Ausbildung/downloads/3344312/164b0bb27936272ffdd85555eb73b79d/Merkblatt-zur-Abschlusspruefung-der-IT-Berufe-data.pdf>. Version: 2013. – Zugriff am: 02.10.2017

Langr u. a. 2015

LANGR, Jeff ; THOMAS, Dave ; HUNT, Andy: *Pragmatic Unit Testing: in Java 8 with JUnit*. USA : The Pragmatic Programmers, 2015. – ISBN 9781941222591

Sommerville 2007

SOMMERVILLE, Ian: *Software Engineering*. Pearson Studium, 2007. – ISBN 9783868940992

A Anhang

A.1 Detaillierte Zeitplanung

Analysephase	8 h
1. Durchführen der Ist-Analyse	2 h
2. Durchführen der Wirtschaftlichkeitsanalyse	2 h
3. Ermitteln von Use-Cases	2 h
4. Unterstützen des Fachbereiches bei der Erstellung des Lastenheftes	2 h
Entwurfsphase	9 h
1. Entwerfen der Benutzeroberfläche	2 h
2. Entwerfen der Datenbankstruktur (ERM)	2 h
3. Ableiten des Tabellen- und Domänenmodells aus dem ERM	1 h
4. Planen der Architektur	1 h
5. Erstellen des Pflichtenheftes	3 h
Implementierungsphase	42 h
1. Anlegen des Java-Projekts	1 h
2. Einrichten des Gradle-Builds und der statischen Code-Analyse	2 h
3. Implementieren der Speicherung der Beitragsverläufe in einer Datenbank in Natural	4 h
4. Implementieren der Domänenklassen inkl. Tests	5 h
5. Herstellen der Datenbank-Verbindung in Java	2 h
6. Implementieren der Datenverarbeitung in Java	21 h
6.1. Implementieren der Verarbeitung der Erstbeiträge inkl. Tests	6 h
6.2. Implementieren der Verarbeitung der Kosten inkl. Tests	5 h
6.3. Implementieren der Verarbeitung der Beitragsverläufe inkl. Tests	6 h
6.4. Implementieren der Ermittlung der BAP-Daten inkl. Tests	4 h
7. Implementieren der Oberfläche inkl. Tests	7 h
Abnahme und Deployment	7 h
1. Code-Review mit dem Ausbilder	2 h
2. Abnahme durch die Fachabteilung	1 h
3. Continuous Integration mit Jenkins	2 h
4. Deployment der Anwendung	2 h
Erstellen der Dokumentation	4 h
1. Erstellen der Entwicklerdokumentation mit JavaDoc	1 h
2. Erstellen des Benutzerhandbuchs	3 h
Gesamt	70 h

A.2 Verwendete Ressourcen

Hardware

- Büroarbeitsplatz mit Thin-Client

Software

- Balsamiq – Programm zur Erstellung von Mockups
- Eclipse Oxygen for Java EE Developers – Entwicklungsumgebung [Java EE 7](#)
- Eclipse Oxygen mit TeXlipse – Entwicklungsumgebung $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$
- Enterprise Architect – Programm zum Erstellen verschiedener Modelle und Diagramme
- Git – Verteilte Versionsverwaltung
- Gradle – Build-Tool
- Java Code Coverage, PMD, Checkstyle, FindBugs – statische Codeanalyse
- JBoss [EAP 7](#) – Application Server als Container für die Webanwendung
- Jenkins – Buildserver
- JUnit – Framework zur Durchführung von Unit-Tests
- MiKTeX – Distribution des Textsatzsystems $\text{T}_{\text{E}}\text{X}$
- Mockito – Mocking-Framework zur Erstellung von Pseudoklassen
- Oracle – Datenbanksystem
- Oracle SQL Developer – Verwaltungswerkzeug für Oracle-Datenbanken
- Redmine – Projektmanagment
- Windows 7 Enterprise mit Service Pack 1 – Betriebssystem

Personal

- Anwendungsentwickler – Review des Codes
- Auszubildender – Umsetzung des Projektes
- Mitarbeiter der Mathematikabteilung / der Antragsabteilung / des Vertriebs – Festlegung der Anforderungen, Abstimmung der Oberfläche und Abnahme des Projektes

A.3 Aktivitätsdiagramme

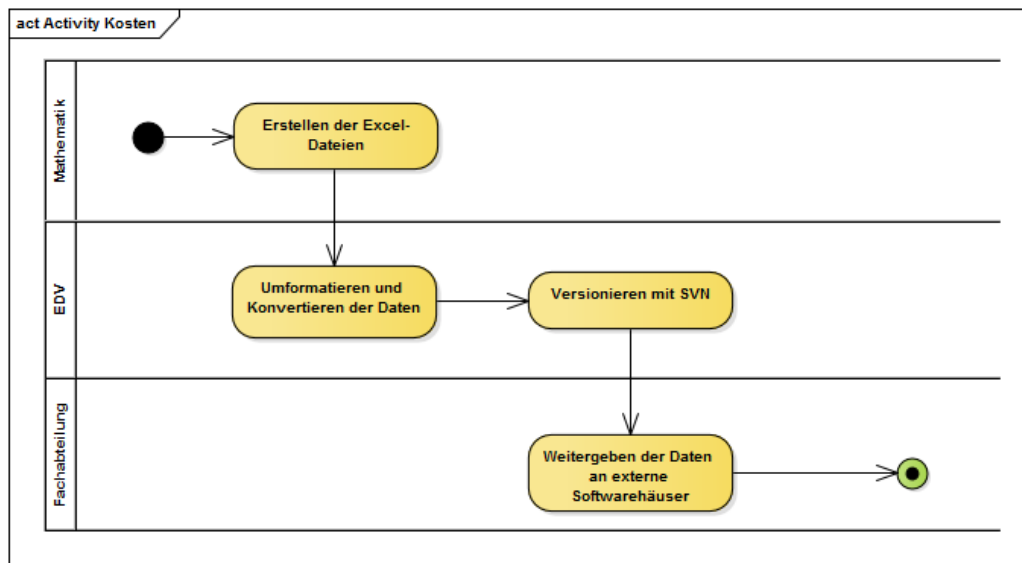


Abbildung 1: Aktivitätsdiagramm zur Ermittlung der Kosten

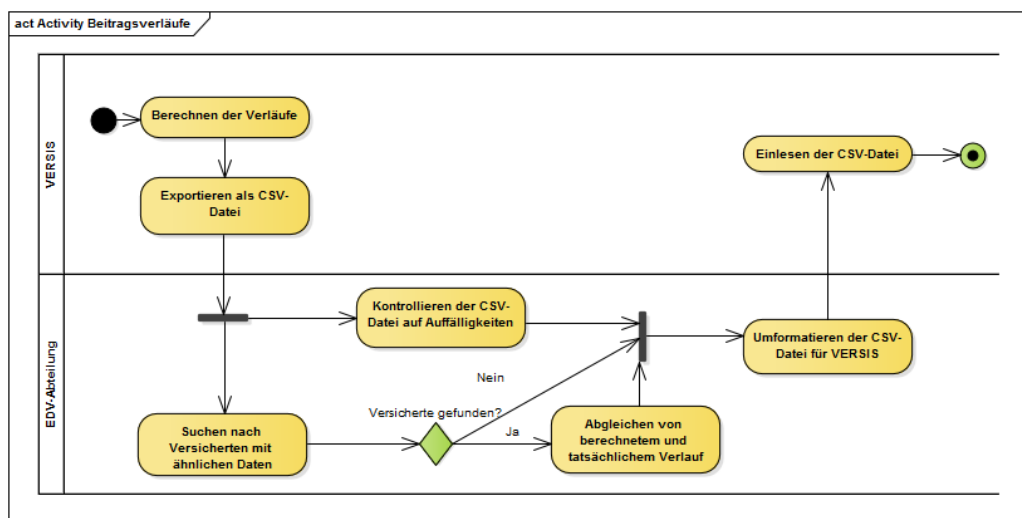


Abbildung 2: Aktivitätsdiagramm zur Ermittlung der Beitragsverläufe

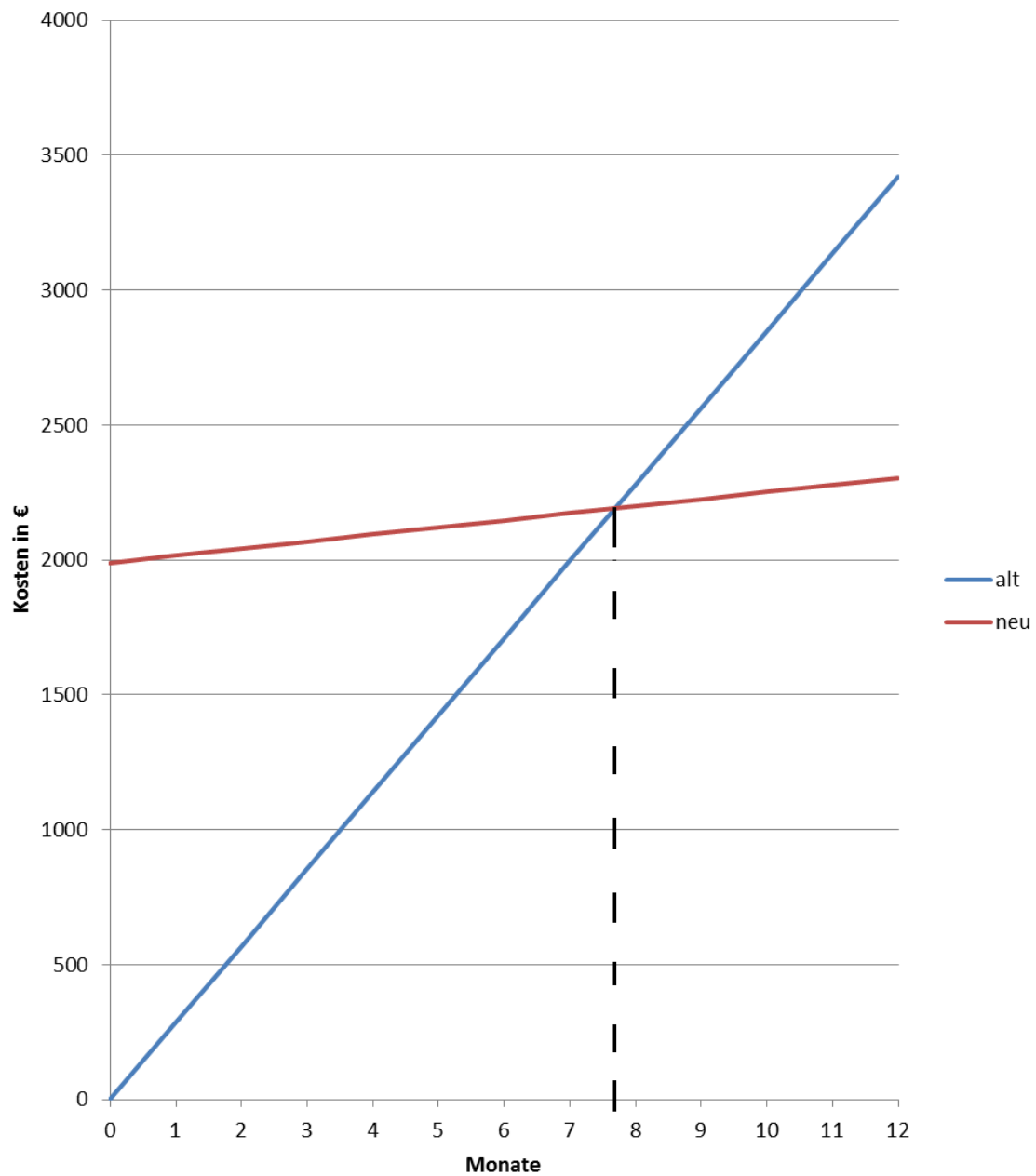
A.4 Amortisation

Abbildung 3: Grafische Darstellung der Amortisation

A.5 Use-Case-Diagramm

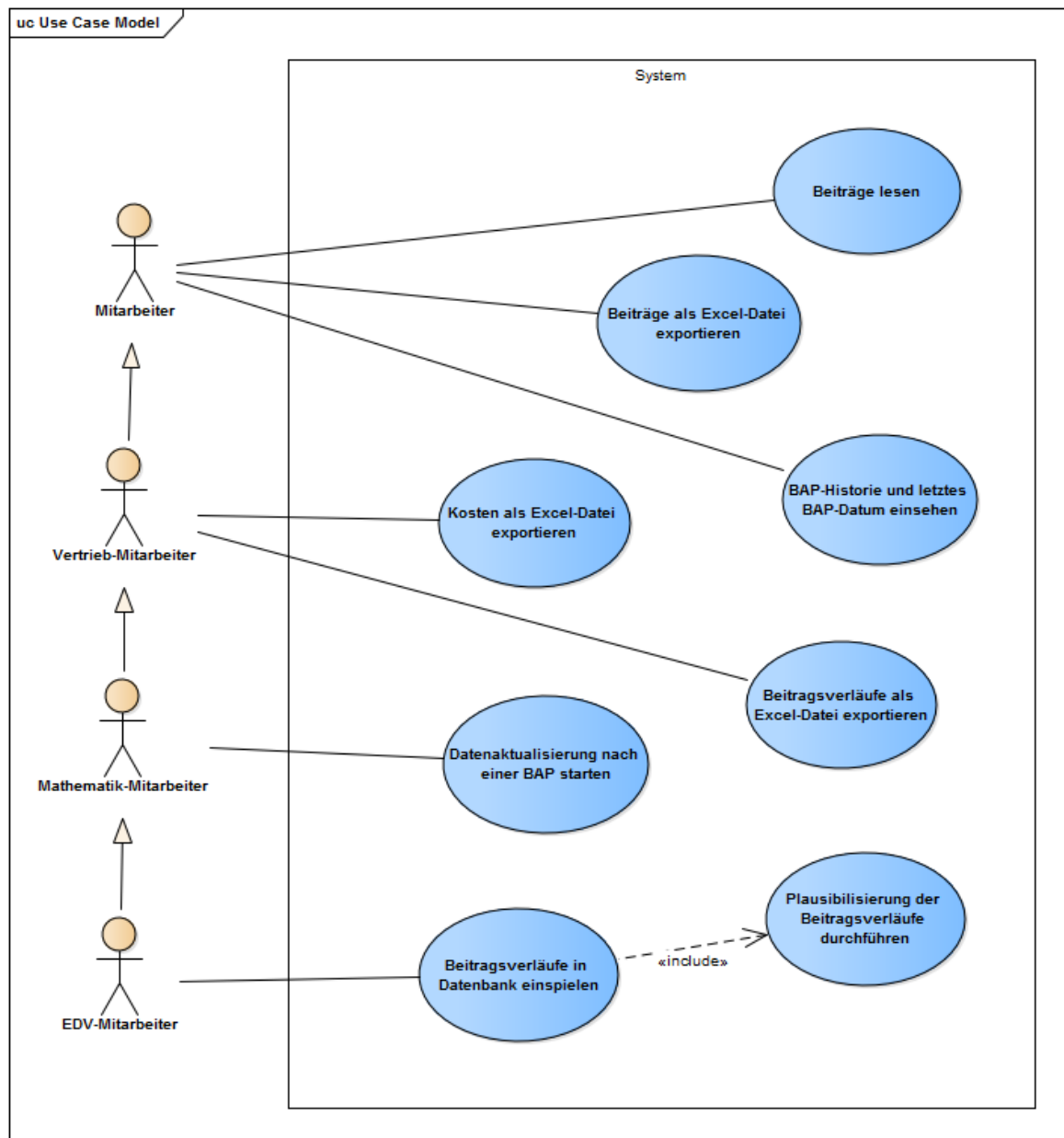


Abbildung 4: Use-Case-Diagramm

A.6 Lastenheft

Die Anwendung muss folgende Anforderungen erfüllen:

1. Verarbeitung der Daten

- 1.1. Die Anwendung muss Daten bezüglich der Beiträge, Beitragsverläufe und Kosten aus einer Datenbank auslesen und verarbeiten.
- 1.2. Die Beiträge, Beitragsverläufe und Kosten müssen ermittelt und verarbeitet werden.
- 1.3. Die aus dem Bestandsführungssystem ermittelten Beitragsverläufe sollen in einer Datenbank anstatt in einer CSV-Datei gespeichert werden.
- 1.4. Bei der Verarbeitung der Beiträge und Kosten sollen Plausibilitätsprüfungen vorgenommen werden.
- 1.5. Die ermittelten Beitragsverläufe sollen mit Echtdaten aus [VERSIS](#) verglichen werden.
- 1.6. Das letzte Datum der Anpassung eines Tarifes durch eine [BAP](#) soll bestimmt werden.
- 1.7. Eine Historie der Änderungen an einem Tarif durch eine [BAP](#) soll ermittelt werden.

2. Darstellung der Daten

- 2.1. Die Anwendung muss die Beiträge, die aktuell in einem Tarif zu zahlen sind, nach Alter gestaffelt präsentieren.
- 2.2. Die Anwendung muss die Kosten zu einem Tarif ermitteln und in zwei fest definierten Excel-Datei-Formaten exportieren.
- 2.3. Die Anwendung muss die Beitragsverläufe zu einem Tarif ermitteln und in zwei fest definierten Excel-Datei-Formaten exportieren.
- 2.4. Die Anwendung muss das letzte Datum, zu dem der Beitrag eines Tarif durch eine [BAP](#) verändert wurde, ausgeben.
- 2.5. Die Anwendung soll sich an das Corporate Design der AO anpassen.
- 2.6. Bei der Oberflächen-Gestaltung sollen Ergonomie-Richtlinien eingehalten werden.
- 2.7. Die Darstellung der Beiträge eines Tarifes kann so umgesetzt werden, dass ein einfaches Kopieren in eine Excel-Datei möglich ist.
- 2.8. Die Anwendung kann eine Möglichkeit zum Drucken bereitstellen.

3. Sonstige Anforderungen

- 3.1. Die Anwendung muss im Intranet der [AO](#) erreichbar sein.
- 3.2. Die Anwendung muss im Webbrowser Mozilla Firefox (Version 38.2) lauffähig sein.
- 3.3. Die im Unternehmen geltenden Coding-Richtlinien müssen eingehalten werden.
- 3.4. Da sich der Fachbereich und externe Softwarehäuser auf diese Daten verlassen, muss die Anwendung korrekte Werte liefern.
- 3.5. Ein Single-Sign-On sollte an der Anwendung möglich sein.

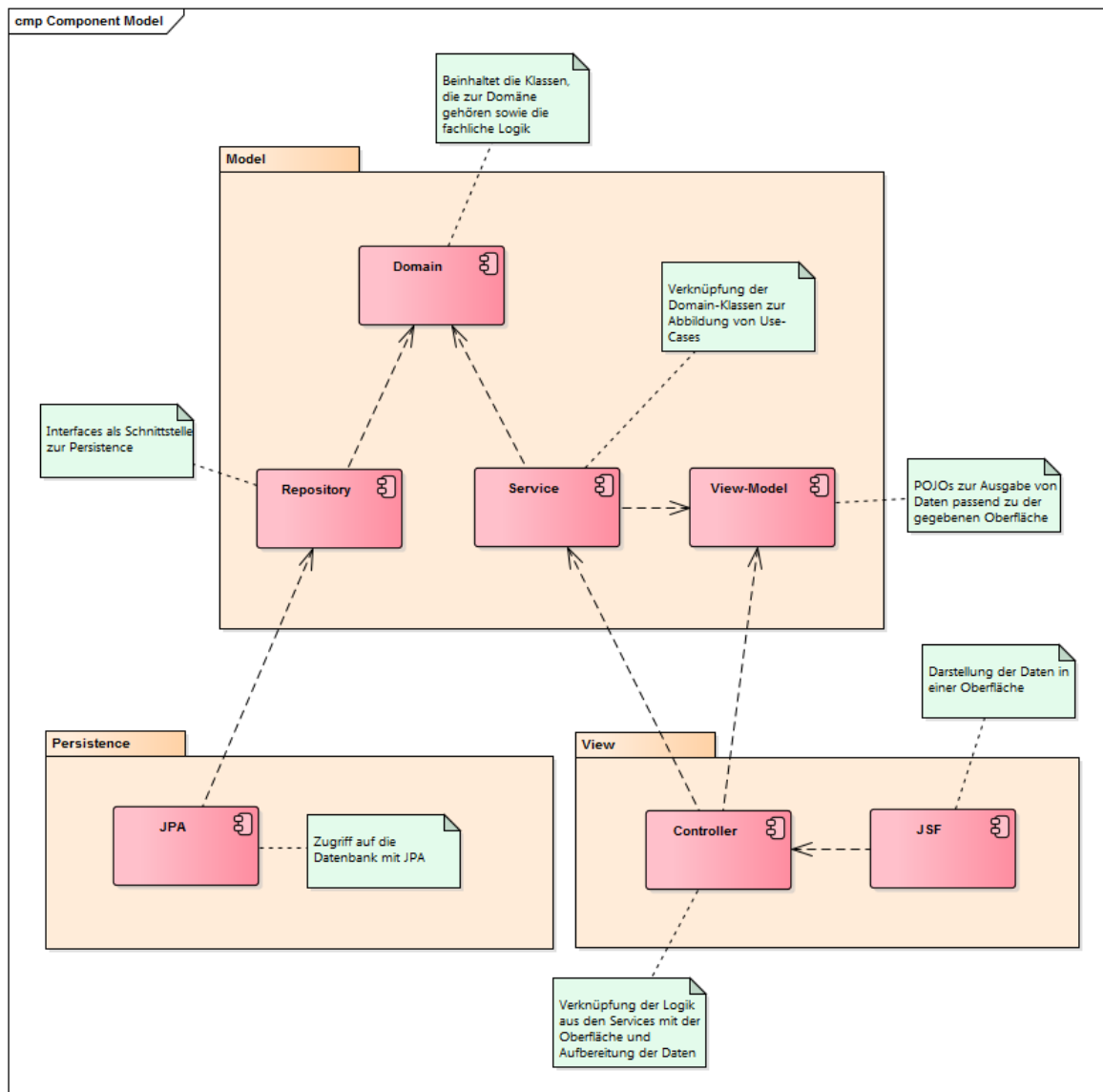
A.7 Komponentendiagramm

Abbildung 5: Komponentendiagramm

A.8 Mockups

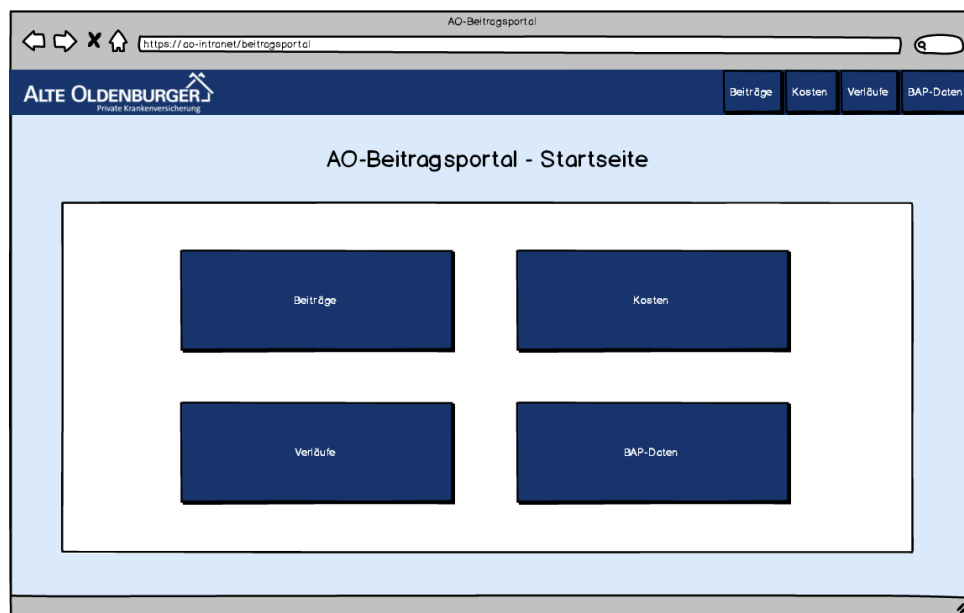


Abbildung 6: Mockup der Startseite

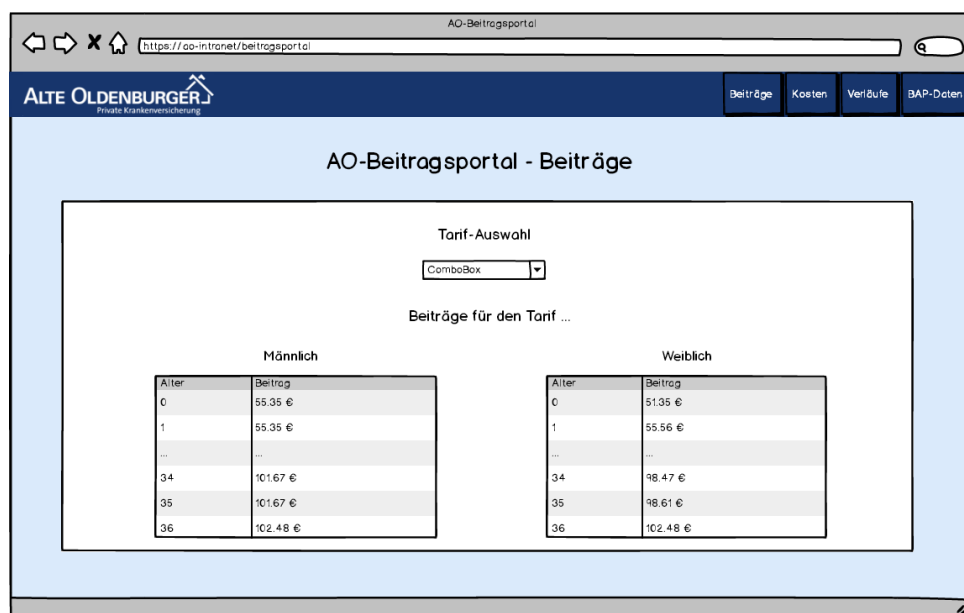


Abbildung 7: Mockup der Beitrag-Seite

A.9 Entity-Relationship-Modell

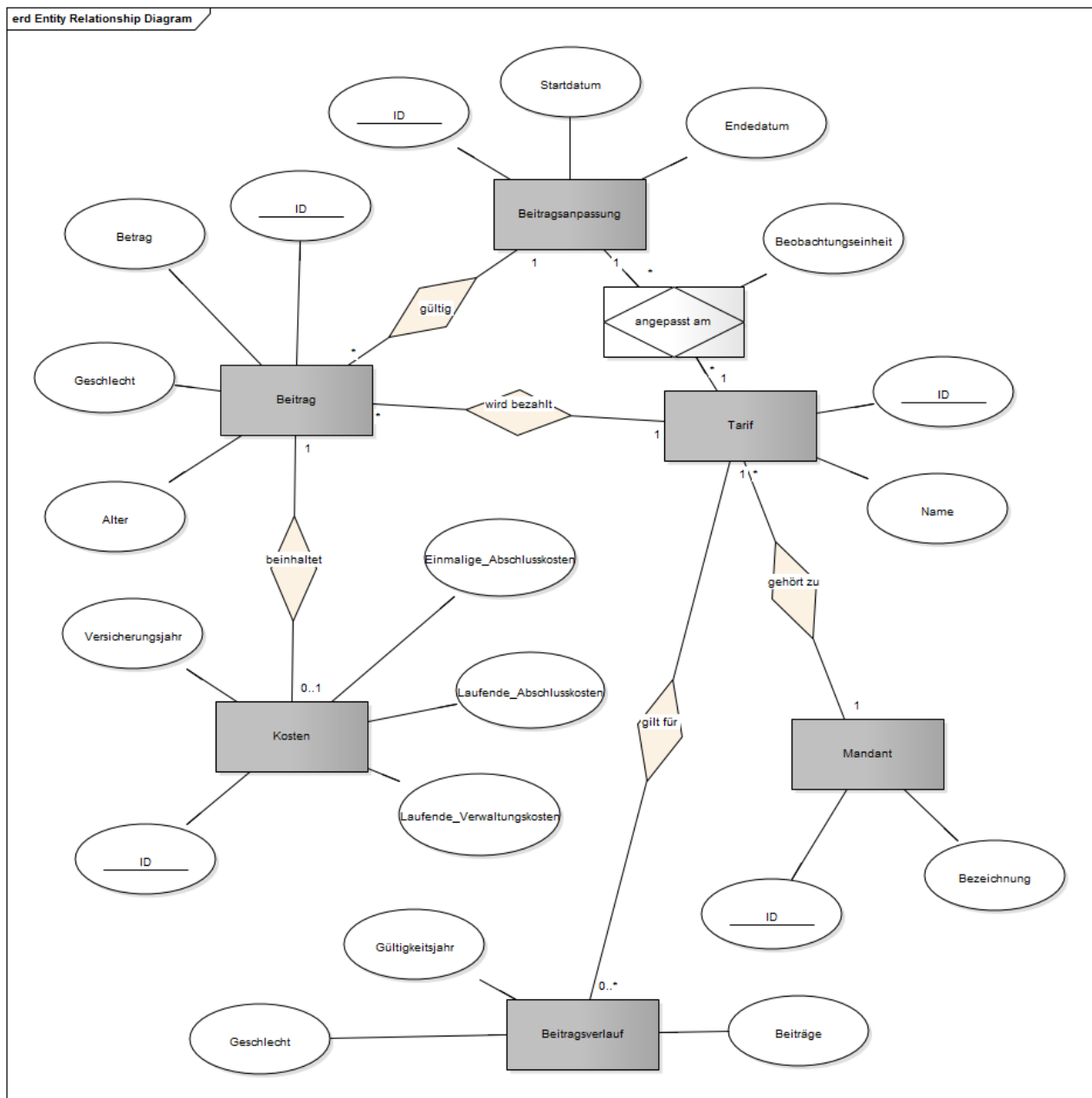


Abbildung 8: Entity-Relationship-Modell

A.10 Klassendiagramm (Auszug)

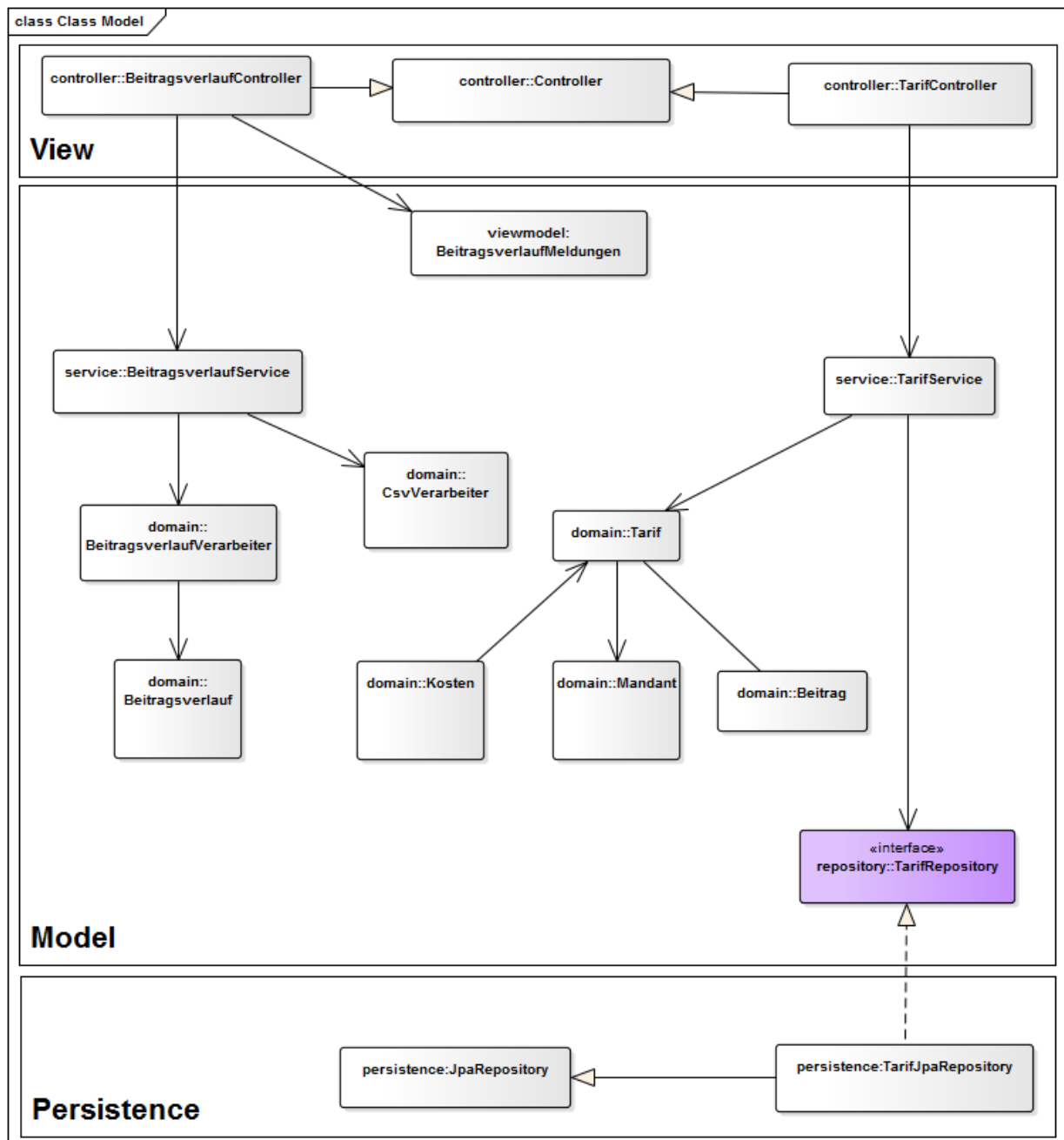


Abbildung 9: Auszug des Klassendiagramms zur Veranschaulichung der Architektur

A.11 Statische Codeanalyse

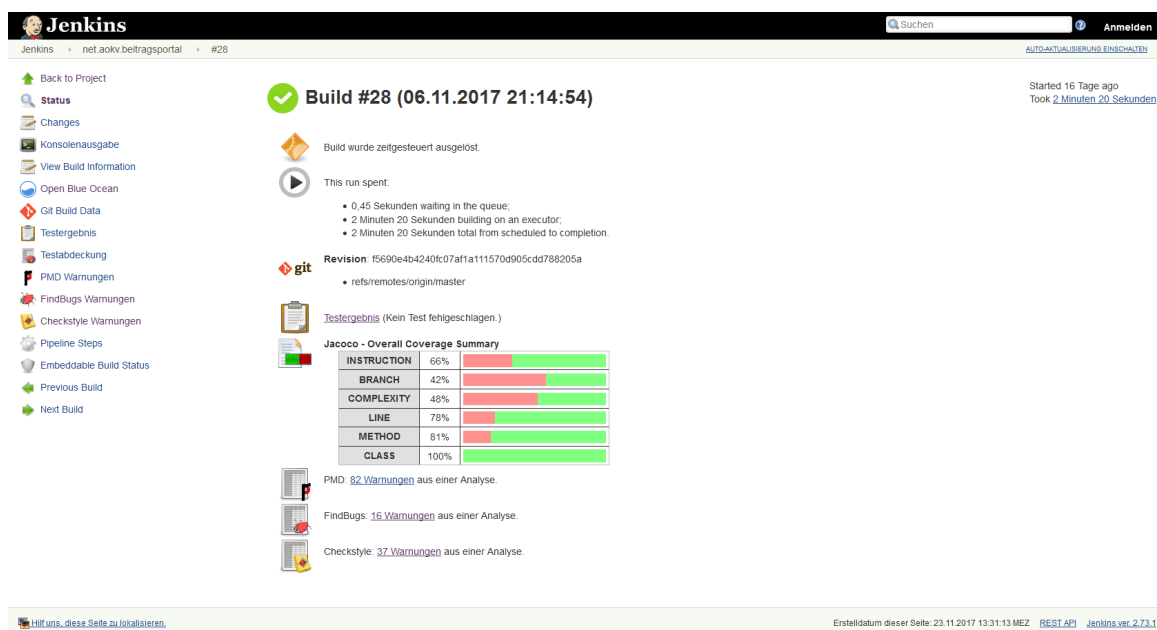


Abbildung 10: Übersicht der statische Codeanalyse mit JaCoCo, PMD, FindBugs und Checkstyle

PMD Ergebnis

Vergleich mit letzter Analyse

Alle Warnungen	Neue Warnungen	Behobene Warnungen
82	0	0

Zusammenfassung

Gesamt	Hohe Priorität	Normale Priorität	Niedrige Priorität
82	0	82	0

Details

Packages	Dateien	Kategorien	Typen	Warnungen	Ursprung	Details
Typ		Gesamt		Verteilung		
AbstractClassWithoutAbstractMethod		1				
AvoidFieldNameMatchingTypeName		1				
BeanMembersShouldSerialize		55				
DefaultPackage		1				
JUnitTestContainsTooManyAsserts		4				
LongVariable		13				
ShortVariable		4				
UnnecessaryFinalModifier		1				
UseUtilityClass		2				
Gesamt		82				

CheckStyle Ergebnis

Vergleich mit letzter Analyse

Alle Warnungen	Neue Warnungen	Behobene Warnungen
37	0	0

Zusammenfassung

Gesamt	Hohe Priorität	Normale Priorität	Niedrige Priorität
37	0	37	0

Details

Packages	Dateien	Kategorien	Typen	Warnungen	Ursprung	Details
Typ		Gesamt		Verteilung		
JavadocMethodCheck		13				
LineLengthCheck		1				
OperatorWrapCheck		9				
SummaryJavadocCheck		14				
Gesamt		37				

Abbildung 11: Ergebnisse von PMD und CheckStyle

A.13 Gradle-Build-Datei (Auszug)

```

1  buildscript {
2      repositories {
3          maven { url = "${project.artifactsURL}${project.artifactsRepoKey}".toURL() }
4          ivy { url = "${project.artifactsURL}${project.artifactsRepoKey}".toURL() }
5      }
6  }
7
8  apply plugin: 'java'
9  apply plugin: 'eclipse'
10 apply plugin: 'eclipse-wtp'
11 apply plugin: 'war'
12
13 dependencies { // Alle benötigten Abhängigkeiten werden hier eingetragen, Versionsnummern werden in 'gradle.
    properties' festgelegt
14     compile "javax:javaee-api:${project.javaEeVersion}",
15             "org.apache.poi:poi-ooxml:${project.apachePoiVersion}", // Apache POI zum Erstellen von Excel-Dateien
16     testCompile "org.mockito:mockito-core:${mockitoVersion}", // Mockito zum Mocken von Objekten in Tests
17             "org.hamcrest:hamcrest-library:${hamcrestVersion}", // Hamcrest stellt Assertions bereit
18             "org.junit.jupiter:junit-jupiter-api:${junitJupiterVersion}", // Abhängigkeiten fuer JUnit 5
19             "org.junit.jupiter:junit-jupiter-params:${junitJupiterVersion}",
20             "com.google.code.bean-matchers:bean-matchers:${project.beanMatcherVersion}", // Testen von POJOs
21             "com.pholser:junit-quickcheck-core:${project.junitQuickcheckVersion}", // Property Based Testing mit JUnit
                Quickcheck
22             "com.pholser:junit-quickcheck-generators:${project.junitQuickcheckVersion}"
23     testRuntime "org.junit.jupiter:junit-jupiter-engine:${junitJupiterVersion}",
24             "org.junit.platform:junit-platform-launcher:${junitPlatformVersion}"
25     integrationTestCompile "org.eclipse.persistence:eclipselink:${project.eclipseLinkVersion}" // JPA-
        Implementierung
26     integrationTestRuntime "org.apache.derby:derby:${project.derbyVersion}" // Apache Derby als Test-DB
27     // ...
28 }
29
30 tasks.withType(JavaCompile) { // Encoding zum Kompilieren wird festgelegt
31     options.encoding = project.encoding
32 }
33
34 task compileAllJava(){ // Eigener Gradle-Task für den Build wird erstellt
35     description 'Compiles Main, Test and IntegrationTest'
36     dependsOn 'compileJava'
37     dependsOn 'compileTestJava'
38     dependsOn 'compileIntegrationTestJava'
39 }
40 // ...

```

Listing 1: Gradle-Build-Datei

A.14 JBoss-Konfiguration

```
1 batch
2
3 # Treiber fuer Oracle hinzufuegen
4 module add
5     --name=com.oracle.jdbc7
6     --resources=S:\\\\Entwicklung\\\\JBoss\\\\jdbc7.jar
7     --dependencies=javax.api,javax.transaction.api
8 /subsystem=datasources/jdbc-driver=oracle:add
9     (driver-name=oracle,
10     driver-module-name=com.oracle.jdbc7,
11     driver-class-name=oracle.jdbc.driver.OracleDriver)
12
13 # DataSource anlegen
14 data-source add
15     --name=BeitragsportalDS
16     --driver-name=oracle
17     --driver-class=oracle.jdbc.driver.OracleDriver
18     --connection-url="jdbc:oracle:thin:@ao-sl11-oracdb:1521:cdb"
19     --jndi-name=java:jboss/jdbc/BeitragsportalDS
20     --user-name="jhellman"
21     --password="passwort"
22
23 run-batch
```

Listing 2: JBoss-Konfigurationsdatei

A.15 Jenkins-Konfiguration (Auszug)

```

1 // ... Einstellung und Initialisieren von Variablen
2 node {
3     try {
4         stage('Checkout') {
5             git branch: branch, credentialsId: 'a8107e09-72f3-4660-9851-6ae4603ef9cd', url: scmUrl
6         }
7         stage('Compile') {
8             powershell('gradle compileAllJava')
9         }
10        stage('Test') {
11            parallel (
12                Tests: { powershell('gradle test') }, Integrationstests: { powershell('gradle integrationTest') }
13            )
14        }
15        stage('Codeanalyse') { // Durchführen der Codeanalyse
16            parallel (
17                Checkstyle: { powershell('gradle checkstyle') }, FindBugs: { powershell('gradle findbugs') },
18                JDepend: { powershell('gradle jdepend') }, PMD: { powershell('gradle pmd') }, JaCoCo: {
19                    powershell('gradle jacoco') }
20            )
21        }
22        stage('Deployment') {
23            powershell("plink -load jenkins-prod sudo JBOSS_HOME/bin/jboss-cli.sh --connect --controller='
24                HOST:PORT' --command='deploy ${env.WORKSPACE}/build/libs/${env.JOB_NAME}-1.0.0-
25                SNAPSHOT.war'")
26        }
27        stage('Post-Build') { // Veröffentlichen der Analyse-Ergebnisse
28            junit allowEmptyResults: true, testResults: 'build/test-results/*.xml'
29            jacoco classPattern: '**/build/classes/main/**', execPattern: '**/build/jacoco/**/*.exec', sourcePattern:
30                '**/src/main/java/**'
31            pmd canComputeNew: false, defaultEncoding: '', healthy: '', pattern: '**/build/reports/pmd/*.xml',
32                unhealthy: ''
33            findbugs canComputeNew: false, defaultEncoding: '', excludePattern: '', healthy: '', includePattern: '',
34                pattern: '**/build/reports/findbugs/*.xml', unhealthy: ''
35            checkstyle canComputeNew: false, defaultEncoding: '', healthy: '', pattern: '**/build/reports/checkstyle
36               /*.xml', unhealthy: ''
37        }
38    }
39    catch (Exception e) {
40        mail bcc: '', body: "Build fehlgeschlagen. Bitte besuche ${env.BUILD_URL} für weitere Informationen", cc
41            : '', from: 'intranet', mimeType: 'text/plain', replyTo: '', subject: "Jenkins-Build für ao-beitragsportal
42                fehlgeschlagen. #${env.BUILD_NUMBER}", to: 'jonas.hellmann@alte-oldenburger.de'
43        throw e
44    }
45 }

```

Listing 3: Jenkins-Konfiguration mit der Scripted Pipeline

A.16 Screenshot der Build-Pipeline

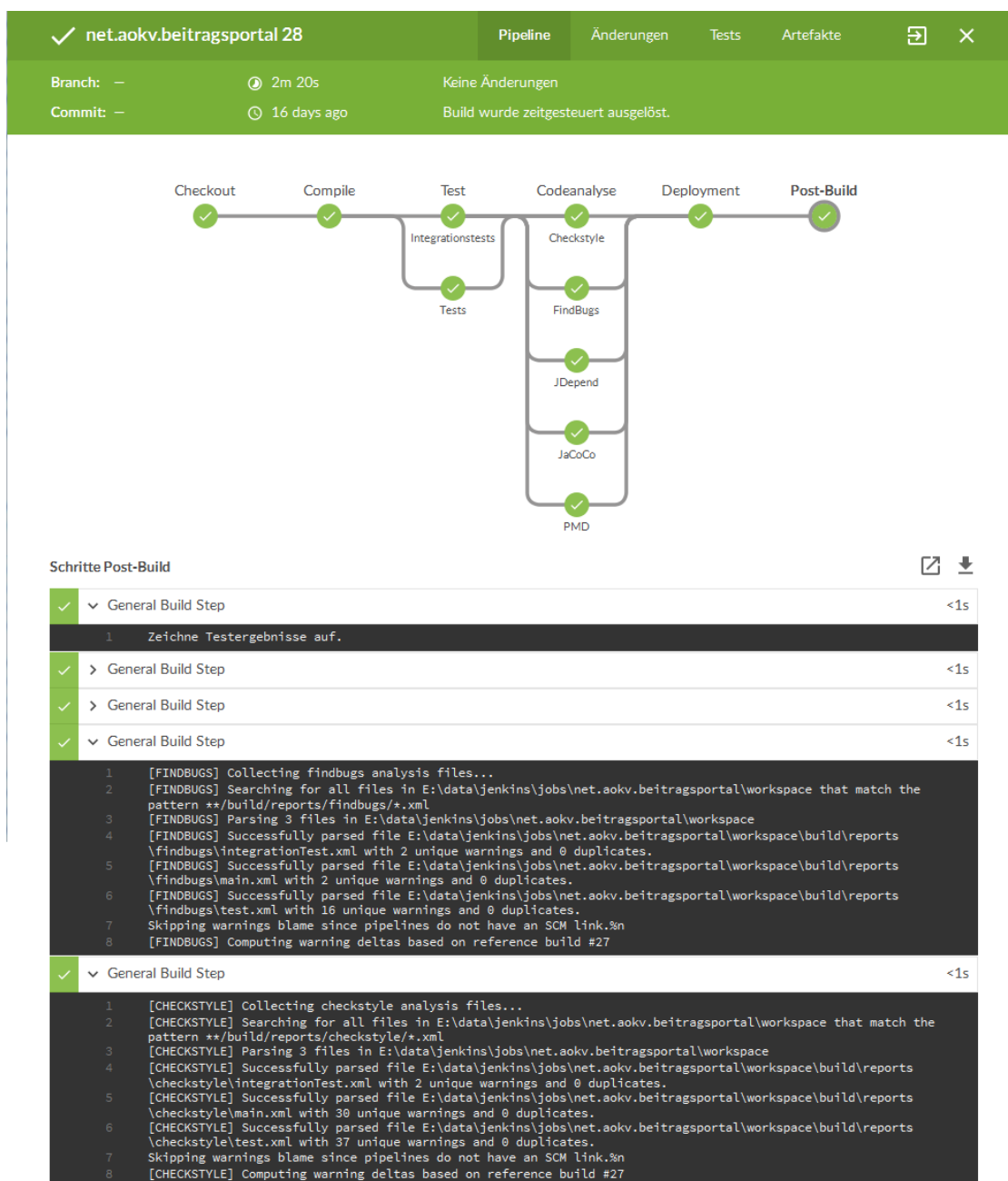


Abbildung 13: Screenshot der Build-Pipeline von der Jenkins-Weboberfläche

A.17 Pflichtenheft (Auszug)

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

1. Plattform

- 1.1. Zur Entwicklung der Anwendung wird Eclipse Oxygen for Java EE Developers eingesetzt.
- 1.2. Die Anwendung wird mit der Programmiersprache Java in der Version 8 programmiert.
- 1.3. Weitergehend wird die Spezifikation [Java EE 7](#) eingesetzt.
- 1.4. Die Anwendung muss aus dem Netzwerk der [AO](#) unter der Adresse <http://ao-intranet/ao-beitragsportal> erreichbar sein.
- 1.5. Als Zielplattform wird ein Server mit dem Betriebssystem *openSUSE* eingesetzt. Die Anwendung wird dort in einem Application Server ausgeführt. Dafür wird eine JBoss-[EAP-7](#)-Instanz verwendet.
- 1.6. Damit das Projekt automatisch gebaut werden kann, wird als Build-Management-Automatisierungs-Tool Gradle eingesetzt.
- 1.7. Um Continuous Integration sicherzustellen, kommt ein Jenkins-Server zum Einsatz.
- 1.8. Der Build-Prozess wird ein automatisches Deployment beinhalten.
- 1.9. Zur Dokumentation des Build-Prozesses wird dieser in einem Jenkinsfile in Form der Scripted Pipeline vom Jenkins notiert und diese zum Bauen eingesetzt.

2. Datenbank

- 2.1. Die Anbindung der Datenbank erfolgt mit JPA.
- 2.2. Dabei wird mit der Implementierung *Hibernate* gearbeitet.
- 2.3. Die Daten der Webanwendung müssen in einer Oracle-Datenbank abgelegt sein, die sich auf einem Oracle-Server der ALTE OLDENBURGER Krankenversicherung befindet.

3. Oberfläche

- 3.1. Die Oberfläche muss mit HTML5 und CSS3 umgesetzt werden.
- 3.2. Um dies zu generieren, wird die [JSF](#)-Technologie genutzt.

4. Geschäftslogik

- 4.1. Zur Erstellung von Objekten wird [CDI](#) verwendet.
- 4.2. Die Tests werden mit JUnit in der Version 5 geschrieben.
- 4.3. Die Excel-Dateien werden mit der Java-Bibliothek *Apache POI* erstellt.

A.18 Tabellenmodell

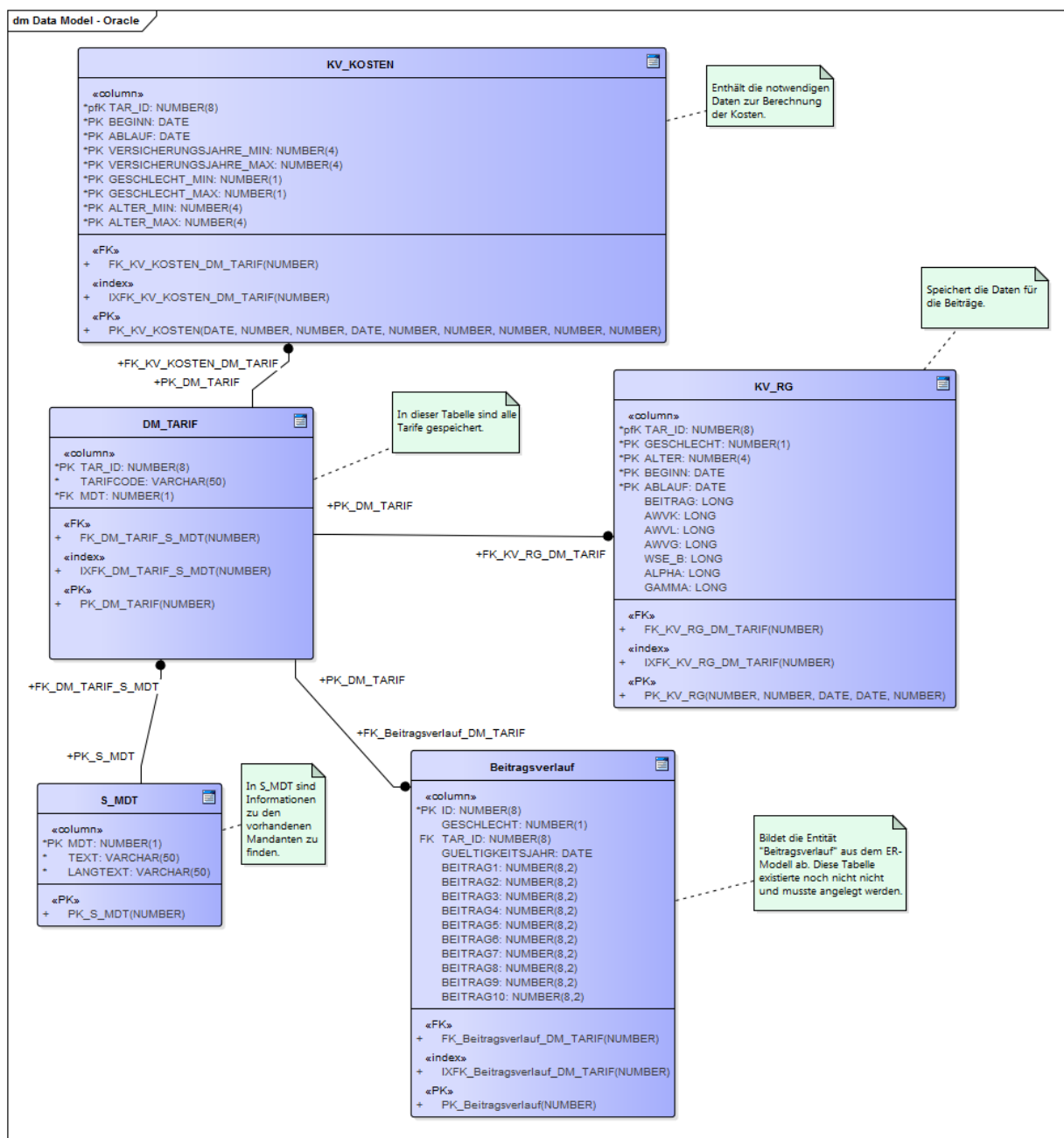


Abbildung 14: Relationales Tabellenmodell

A.19 Listing von Java-Code

```

1 @Named
2 @RequestScoped
3 public class BeitragsverlaufController {
4     @Inject private BeitragsverlaufService service;
5     private Part datei;
6     private List<Beitragsverlauf> verlaeufe;
7     private List<BeitragsverlaufMeldungen> meldungen;
8
9     // Wird durch einen Button der Oberfläche aufgerufen
10    public void liesDatei () {
11        try {
12            // Einlesen der Datei in eine Liste von Strings
13            List<String> zeilen = IOUtils.readLines(datei.getInputStream(), StandardCharsets.UTF_8);
14            verlaeufe = service.konvertiereCsvZuBeitragsverlaeufe(zeilen);
15            validiereBeitragsverlaeufe ();
16        } catch (IOException e) {
17            // Erzeugt eine Fehlermeldung, die dem Benutzer angezeigt wird
18            Controller.erzeugeNachricht(FEHLER_DATEI_LESEN);
19        }
20    }
21
22    // BeitragsverlaufMeldungen beinhaltet zur Ausgabe zu dem Verlauf eine Liste von Meldungen als Strings
23    private void validiereBeitragsverlaeufe () {
24        meldungen = verlaeufe.stream()
25            .map(b -> new BeitragsverlaufMeldungen(b, service.validiereBeitragsverlauf(b)))
26            .collect ( Collectors.toList () );
27    }
28 }

```

Listing 4: Klasse BeitragsverlaufController

```

1 public class BeitragsverlaufService {
2     @Inject CsvVerarbeiter csvVerarbeiter;
3     @Inject BeitragsverlaufVerarbeiter beitragsverlaufVerarbeiter ;
4
5     public List<Beitragsverlauf> konvertiereCsvZuBeitragsverlaeufe(List<String> csvZeilen) {
6         return csvVerarbeiter.konvertiereCsvZuBeitragsverlaeufe(csvZeilen);
7     }
8
9     public List<String> validiereBeitragsverlauf(Beitragsverlauf beitragsverlauf) {
10        return beitragsverlaufVerarbeiter . validiereBeitragsverlauf ( beitragsverlauf );
11    }
12 }

```

Listing 5: Klasse BeitragsverlaufService

```

1 public class CsvVerarbeiter {
2
3     private static final int CSV_SPALTEN = 12;

```


A Anhang

```

4 private static final String CSV_DELIMITER = ',';
5
6 public List<Beitragsverlauf> konvertiereCsvZuBeitragsverlaeufe(List<String> csvZeilen) {
7     return csvZeilen.stream()
8         .map(zeile -> konvertiereZuBeitragsverlauf(zeile))
9         .collect ( Collectors.toList () );
10 }
11
12 private Beitragsverlauf konvertiereZuBeitragsverlauf(String string) {
13     String[] spalten = string.replace( ',', ' ' ).split ( CSV_DELIMITER );
14     List<BigDecimal> beitraege =
15         IntStream.range(2, CSV_SPALTEN)
16             .mapToObj(i -> new BigDecimal(spalten[i]))
17             .collect ( Collectors.toList () );
18     return new Beitragsverlauf(
19         new Tarif(spalten[0]) ,
20         Geschlecht.fromString(spalten[1]) ,
21         beitraege);
22 }
23 }

```

Listing 6: Klasse CsvVerarbeiter

```

1 public class BeitragsverlaufVerarbeiter {
2
3     public static final BigDecimal MAXIMALE_ABWEICHUNG_ZUM_VORJAHR = new BigDecimal("0.25");
4
5     public List<String> validiereBeitragsverlauf(Beitragsverlauf beitragsverlauf) {
6         List<BigDecimal> beitraege = beitragsverlauf.getBeitraege();
7         return IntStream.range(0, beitraege.size() - 1)
8             .filter ( i -> isVeraenderungZuHoch(
9                 berechneProzentualeVeraenderung(beitraege.get(i), beitraege.get(i + 1)))
10             // i + 1 und i + 2, da Stream und Liste Null-basiert sind
11             .mapToObj(i ->
12                 String.format("Die Veränderung zwischen Jahr %d und %d ist zu groß", i + 1, i + 2))
13             .collect ( Collectors.toList () );
14     }
15
16     private BigDecimal berechneProzentualeVeraenderung(BigDecimal startwert, BigDecimal endwert) {
17         return endwert.subtract(startwert).divide(startwert, 2, BigDecimal.ROUND_HALF_EVEN);
18     }
19
20     private boolean isVeraenderungZuHoch(BigDecimal veraenderung) {
21         return MAXIMALE_ABWEICHUNG_ZUM_VORJAHR.compareTo(veraenderung) <= 0
22             || MAXIMALE_ABWEICHUNG_ZUM_VORJAHR.multiply(new BigDecimal("-1"))
23                 .compareTo(veraenderung) >= 0;
24     }
25
26 }

```

Listing 7: Klasse BeitragsverlaufVerarbeiter

A.20 JSF-Vorlage

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4 <html xmlns="http://www.w3.org/1999/xhtml"
5     xmlns:f="http://java.sun.com/jsf/core"
6     xmlns:h="http://java.sun.com/jsf/html"
7     xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
8     xmlns:c="http://java.sun.com/jsp/jstl/core">
9 <h:head>
10     <title>AO-Beitragsportal – ALTE OLDENBURGER</title>
11     <ui:include src="/WEB-INF/vorlagen/head.xhtml" />
12 </h:head>
13 <h:body>
14     <header>
15     <nav>
16         <div id="nav" class="navbar navbar-default">
17             <div id="navbarCollapse" class="collapse navbar-collapse">
18                 
19                 <ul class="nav navbar-nav">
20                     <li><a href="#{navigationController.beitraege()}">Beiträge</a></li>
21                     <li><a href="#{navigationController.verlaeufo()}">Verläufe</a></li>
22                     <li><a href="#{navigationController.kosten()}">Kosten</a></li>
23                     <li><a href="#{navigationController.bapdaten()}">BAP Daten</a></li>
24                 </ul>
25             </div>
26         </div>
27     </nav>
28 </header>
29     <div id="wrapper">
30         <h2><ui:insert name="titel" /></h2>
31         <div id="nachrichten"><h:messages /></div>
32         <div id="inhalt"><ui:insert name="inhalt" /></div>
33     </div>
34     <footer>
35     <div id="footer-nav"><a href="#{navigationController.start()}">Startseite</a></div>
36     <div id="copyright">
37         <a href="https://alte-oldenburger.de"><strong>ALTE OLDENBURGER Krankenversicherung AG</strong>
38         </a>
39     </div>
40 </h:body>
41 </html>

```

Listing 8: JSF-Vorlage zur Erstellung der Oberflächen

A.21 Screenshots



ALTE OLDENBURGER
Private Krankenversicherung

Beiträge Verläufe Kosten BAP Daten

AO-Beitragsportal - Beiträge

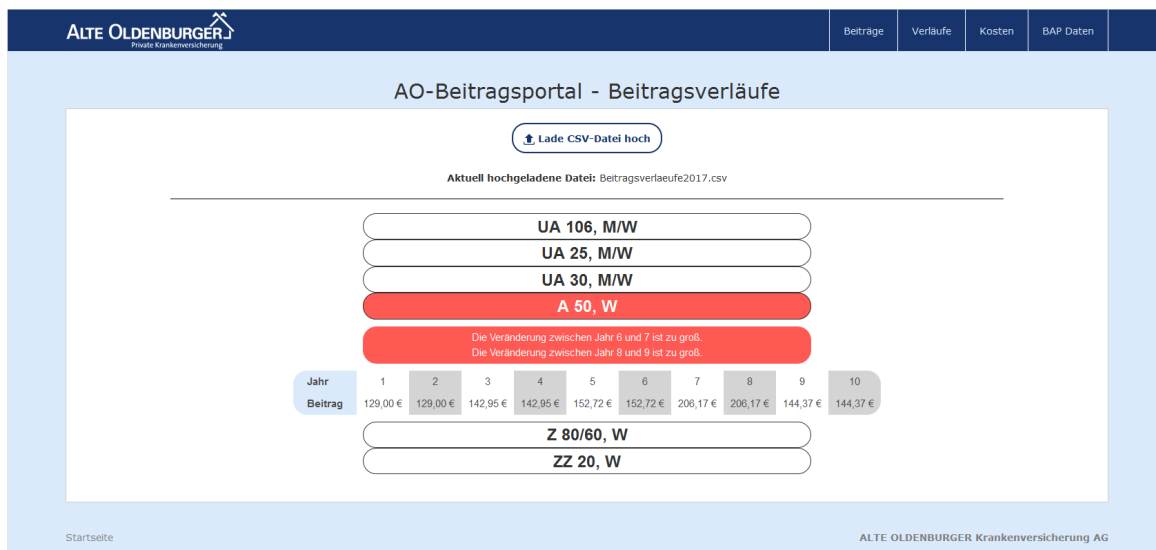
A 106 20.11.2017

Zeige Beiträge an

Beiträge des Tarifs A 106 zum 20.11.2017

Männlich		Weiblich	
Eintrittsalter	Beitrag	Eintrittsalter	Beitrag
0	62,45 €	0	62,45 €
1	62,45 €	1	62,45 €
2	62,45 €	2	62,45 €
3	62,45 €	3	62,45 €
4	62,45 €	4	62,45 €
5	62,45 €	5	62,45 €
6	62,45 €	6	62,45 €
7	62,45 €	7	62,45 €
8	62,45 €	8	62,45 €
9	62,45 €	9	62,45 €
10	62,45 €	10	62,45 €
11	62,45 €	11	62,45 €
12	62,45 €	12	62,45 €
13	62,45 €	13	62,45 €
14	62,45 €	14	62,45 €
15	60,98 €	15	73,92 €
16	60,98 €	16	73,92 €
17	60,98 €	17	73,92 €
18	60,98 €	18	73,92 €
19	60,98 €	19	73,92 €

Abbildung 15: Screenshot der Anzeige von Beiträgen



ALTE OLDENBURGER
Private Krankenversicherung

Beiträge Verläufe Kosten BAP Daten

AO-Beitragsportal - Beitragsverläufe

Lade CSV-Datei hoch

Aktuell hochgeladene Datei: Beitragsverläufe2017.csv

UA 106, M/W
UA 25, M/W
UA 30, M/W
A 50, W
Die Veränderung zwischen Jahr 6 und 7 ist zu groß.
Die Veränderung zwischen Jahr 8 und 9 ist zu groß.

Jahr	1	2	3	4	5	6	7	8	9	10
Beitrag	129,00 €	129,00 €	142,95 €	142,95 €	152,72 €	152,72 €	206,17 €	206,17 €	144,37 €	144,37 €

Z 80/60, W
ZZ 20, W

Startseite ALTE OLDENBURGER Krankenversicherung AG

Abbildung 16: Screenshot der Anzeige der überprüften Beitragsverläufe

A.22 Test mit JUnit

```

1  @DisplayName("Integrations-Test: JPA-Repository fuer Tarife sollte")
2  public class TarifJpaRepositoryIntegrationTest {
3      private TarifJpaRepository sut; // sut = system under test
4
5      @BeforeAll // wird einmalig vor allen Tests ausgeführt
6      public static void setUpClass() {
7          final EntityManager entityManager = RepositoryIntegrationTest.getEntityManager();
8          final EntityTransaction transaction = entityManager.getTransaction();
9          transaction.begin();
10         new TestDatenErzeuger(entityManager).erzeugeTestDaten(); // Befüllen der Datenbank mit Test-Daten
11         transaction.commit();
12     }
13
14     @BeforeEach // wird vor jedem Test erneut ausgeführt
15     public void setUp() {
16         // für jeden Test wird eine neue Instanz erzeugt,
17         // um Abhängigkeiten zwischen den Tests zu vermeiden
18         sut = new TarifJpaRepository(RepositoryIntegrationTest.getEntityManager());
19     }
20
21     @Test
22     @DisplayName("Tarife finden")
23     public void tarifeFinden() {
24         final List<Tarif> ergebnis = sut.findeTarife();
25
26         assertThat(ergebnis, hasItems(TestDaten.valideTarife().toArray(new Tarif[TestDaten.valideTarife().size()])));
27     }
28
29     @Test
30     @DisplayName("einen bestimmten Tarif finden, wenn dieser vorhanden ist")
31     public void einenBestimmtenTarifFindenWennVorhanden() {
32         final Optional<Tarif> ergebnis = sut.findeTarif(TestDaten.validerTarif().getId());
33
34         assertThat(ergebnis.isPresent(), is(true));
35         assertThat(ergebnis.get(), is(TestDaten.validerTarif()));
36     }
37
38     @Test
39     @DisplayName("keinen Tarif finden, wenn der gesuchte nicht vorhanden ist")
40     public void keinenTarifFindenWennNichtVorhanden() {
41         int ungeltigeTarifId = 1000;
42         final Optional<Tarif> ergebnis = sut.findeTarif(ungeltigeTarifId);
43
44         assertThat(ergebnis.isPresent(), is(false));
45     }
46 }

```

Listing 9: Integrationstest der Klasse TarifJpaRepository mit JUnit

A.23 Benutzerdokumentation (Auszug)



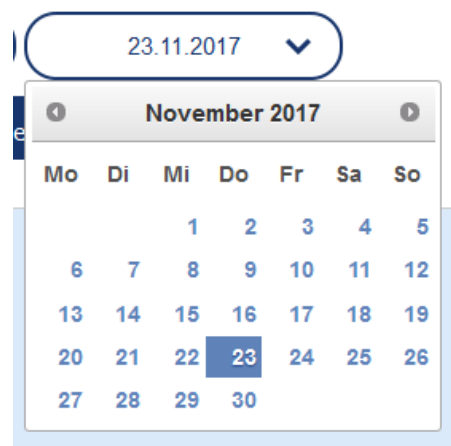
Um auf die Seite zu gelangen, auf der die Beiträge dargestellt werden, klicken Sie in der Liste am oberen Bildschirmrand auf „Beiträge“ (1) oder wählen Sie auf der Startseite den Button „Beiträge“ (2). Alternativ können Sie den Shortcut **Shift + B** benutzen.



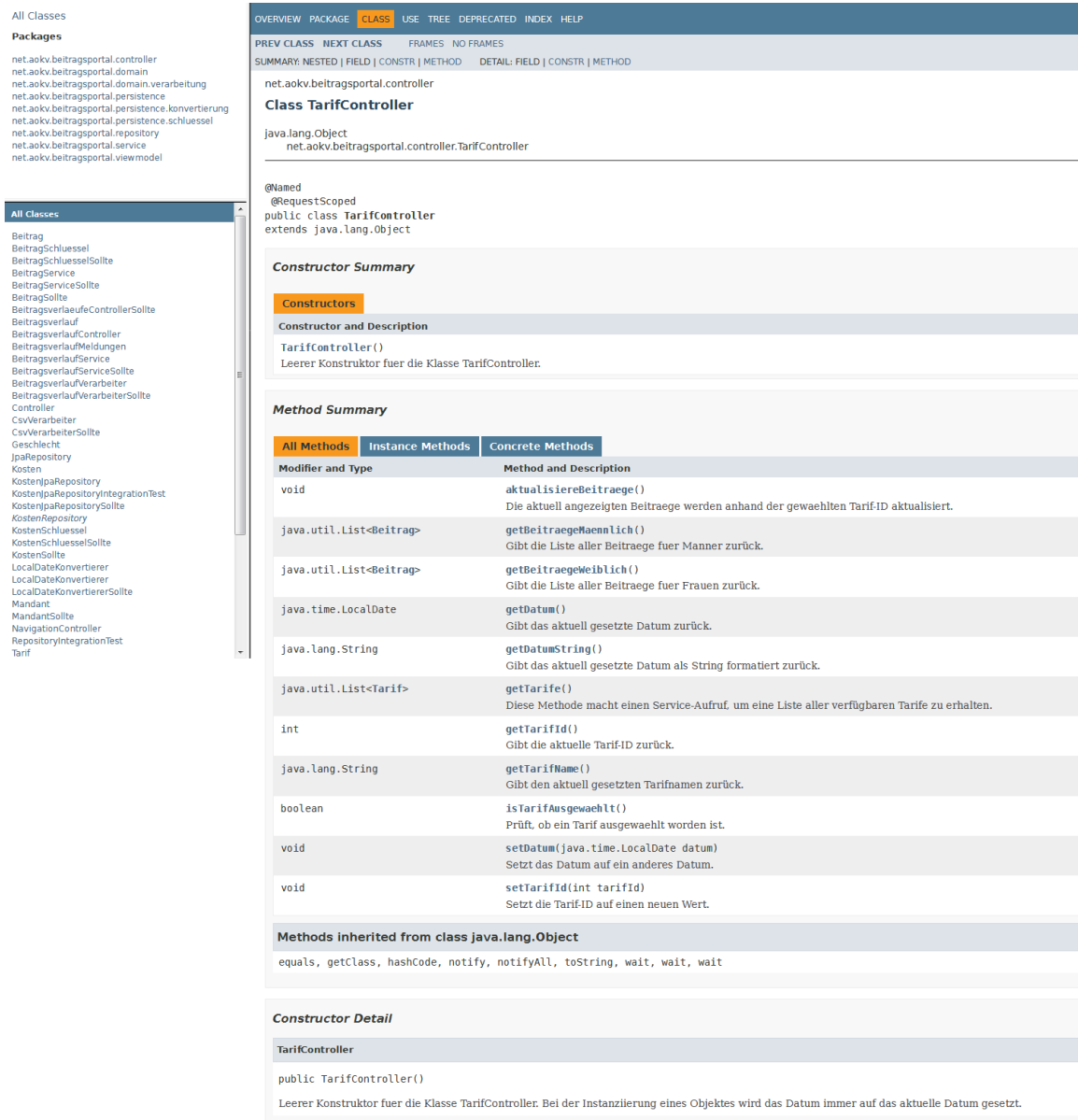
Klicken Sie nun zuerst auf die Tarif-Auswahl (3). Es öffnet sich ein Menü mit allen verfügbaren Tarifen. Der gewünschte Tarif kann mit einem Mausklick ausgewählt werden.

Sollte der gewünschte Stichtag für die Beiträge vom heutigen Datum abweichen, so können Sie diesen verändern (4). Es sollte sich ein Kalender (siehe Abb. rechts) öffnen. Mit den Pfeilen in der rechten und linken oberen Ecke lässt sich zum vorherigen und nächsten Monat wechseln. Der Tag lässt sich durch einen Mausklick auswählen.

Nach einem Klick auf den Button „Zeige Beiträge an“ (5) werden die entsprechenden Beiträge geladen und nach Geschlechtern getrennt angezeigt. Bei Unisex-Tarifen findet diese Trennung nicht statt.



A.24 Entwicklerdokumentation



All Classes

Packages

- net.aokv.beitragsportal.controller
- net.aokv.beitragsportal.domain
- net.aokv.beitragsportal.domain.verarbeitung
- net.aokv.beitragsportal.persistence
- net.aokv.beitragsportal.persistence.konvertierung
- net.aokv.beitragsportal.persistence.schlüssel
- net.aokv.beitragsportal.repository
- net.aokv.beitragsportal.service
- net.aokv.beitragsportal.viewmodel

All Classes

- Beitrag
- BeitragSchlüssel
- BeitragSchlüsselSolite
- BeitragService
- BeitragServiceSolite
- BeitragSolite
- BeitragsverlaufeControllerSolite
- Beitragsverlauf
- BeitragsverlaufController
- BeitragsverlaufMeldungen
- BeitragsverlaufService
- BeitragsverlaufServiceSolite
- BeitragsverlaufVerarbeiter
- BeitragsverlaufVerarbeiterSolite
- Controller
- CsvVerarbeiter
- CsvVerarbeiterSolite
- Geschlecht
- JpaRepository
- Kosten
- KostenJpaRepository
- KostenJpaRepositoryIntegrationTest
- KostenJpaRepositorySolite
- KostenRepository
- KostenSchlüssel
- KostenSchlüsselSolite
- KostenSolite
- LocalDateKonvertierer
- LocalDateKonvertierer
- LocalDateKonvertiererSolite
- Mandant
- MandantSolite
- NavigationController
- RepositoryIntegrationTest
- Tarif

OVERVIEW PACKAGE **CLASS** USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

net.aokv.beitragsportal.controller

Class TarifController

java.lang.Object
net.aokv.beitragsportal.controller.TarifController

@Named
@RequestScoped
public class TarifController
extends java.lang.Object

Constructor Summary

Constructors

Constructor and Description
TarifController() Leerer Konstruktor fuer die Klasse TarifController.

Method Summary

All Methods	Instance Methods	Concrete Methods
Modifier and Type	Method and Description	
void	aktualisiereBeitraege()	Die aktuell angezeigten Beitrage werden anhand der gewählten Tarif-ID aktualisiert.
java.util.List<Beitrag>	getBeitraegeMaennlich()	Gibt die Liste aller Beitrage fuer Manner zurück.
java.util.List<Beitrag>	getBeitraegeWeiblich()	Gibt die Liste aller Beitrage fuer Frauen zurück.
java.time.LocalDate	getDatum()	Gibt das aktuell gesetzte Datum zurück.
java.lang.String	getDatumString()	Gibt das aktuell gesetzte Datum als String formatiert zurück.
java.util.List<Tarif>	getTarife()	Diese Methode macht einen Service-Aufruf, um eine Liste aller verfügbaren Tarife zu erhalten.
int	getTarifId()	Gibt die aktuelle Tarif-ID zurück.
java.lang.String	getTarifName()	Gibt den aktuell gesetzten Tarifnamen zurück.
boolean	isTarifAusgewaehlt()	Prüft, ob ein Tarif ausgewaehlt worden ist.
void	setDatum(java.time.LocalDate datum)	Setzt das Datum auf ein anderes Datum.
void	setTarifId(int tarifId)	Setzt die Tarif-ID auf einen neuen Wert.

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

TarifController

public TarifController()
Leerer Konstruktor fuer die Klasse TarifController. Bei der Instanziierung eines Objektes wird das Datum immer auf das aktuelle Datum gesetzt.

Abbildung 17: Screenshot der Entwicklerdokumentation