



Abschlussprüfung Winter 2022

Fachinformatiker/Fachinformatikerin (VO 2020) Fachrichtung:
Anwendungsentwicklung, Leipzig FIAE 4 (AP T2V1)
Dokumentation zur betrieblichen Projektarbeit

ReklaTool

**Entwicklung einer Webanwendung zur Abfrage und Anzeige
von Kalkulationsvorgängen aus dem Kfz-Bereich
nach dem Client-Server-Prinzip**

Abgabetermin: Leipzig, den 23.11.2022

Prüfungsbewerber:

Ingolf Schieck
Identnummer: 661537
Hähnelstraße 21
04177 Leipzig



Ausbildungsbetrieb:

IcamSystems GmbH
Käthe-Kollwitz-Straße 84
04109 Leipzig

Dieses Werk einschließlich seiner Teile ist **urheberrechtlich geschützt**. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Autors unzulässig und strafbar. Das gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen sowie die Einspeicherung und Verarbeitung in elektronischen Systemen.

Inhaltsverzeichnis

Abbildungsverzeichnis	III
Tabellenverzeichnis	IV
Listings	V
Abkürzungsverzeichnis	VI
1 Einleitung	1
1.1 Projektumfeld	1
1.2 Projektziel	1
1.3 Projektbegründung	1
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 Amortisation	6
3.3 Nicht-monetärer Nutzen	7
3.4 Anwendungsfälle	7
3.5 Qualitätsanforderungen	7
3.6 Lastenheft/Fachkonzept	7
4 Entwurfsphase	7
4.1 Zielplattform	8
4.2 Architekturdesign	8
4.3 Entwurf der Benutzeroberfläche	9
4.4 Datenmodell	9
4.5 Geschäftslogik	9
4.6 Maßnahmen zur Qualitätssicherung	10
4.7 Pflichtenheft/Datenverarbeitungskonzept	10
5 Implementierungsphase	10

5.1	Konfiguration	10
5.2	Implementierung der Datenstrukturen	10
5.3	Implementierung der Benutzeroberfläche	11
5.4	Implementierung der Geschäftslogik	11
6	Abnahmephase	11
7	Einführungsphase	12
8	Dokumentation	12
9	Fazit	12
9.1	Soll-/Ist-Vergleich	12
9.2	Lessons Learned	13
9.3	Ausblick	13
	Literaturverzeichnis	14
	Eidesstattliche Erklärung	15
A	Anhang	i
A.1	Detaillierte Zeitplanung	i
A.2	Ressourcen	ii
A.3	Break-Even-Point	iii
A.4	Lastenheft (Auszug)	iii
A.5	Aktivitätsdiagramm	v
A.6	Use-Case-Diagramm	vi
A.7	Sequenzdiagramm	vi
A.8	Pflichtenheft (Auszug)	vii
A.9	Datenmodell	ix
A.10	Oberflächenentwürfe	x
A.11	Screenshots der Anwendung	xii
A.12	Entwicklerdokumentation	xiv
A.13	Testfall und sein Aufruf auf der Konsole	xvi
A.14	Klasse: ComparedNaturalModuleInformation	xvii
A.15	Klassendiagramm	xx
A.16	Benutzerdokumentation	xxi

Abbildungsverzeichnis

1	Break-Even-Point	iii
2	Aktivitätsdiagramm	v
3	Use-Case-Diagramm	vi
4	Sequenzdiagramm	vi
5	XML-Modell für Antwort	ix
6	XML-Modell für Anfrage	ix
7	Liste der Module mit Filtermöglichkeiten	x
8	Anzeige der Übersichtsseite einzelner Module	xi
9	Anzeige und Filterung der Module nach Tags	xi
10	Anzeige und Filterung der Module nach Tags	xii
11	Liste der Module mit Filtermöglichkeiten	xiii
12	Aufruf des Testfalls auf der Konsole	xvii
13	Klassendiagramm	xx

Tabellenverzeichnis

1	Zeitplanung	3
2	Kostenaufstellung	5
3	Zeitersparnis	6
4	Soll-/Ist-Vergleich	13

Listings

1	Testfall in PHP	xvi
2	Klasse: ComparedNaturalModuleInformation	xvii

Abkürzungsverzeichnis

BPMS	Business Process Management Software
bzw.	beziehungsweise
ca.	circa
CG	ClaimsGuard
DI	Dependency Injection
DBMS	Datenbankmanagementsystem
DTO	Data Transfer Object
ggf.	gegebenenfalls
HTTP	Hypertext Transfer Protocol
Icam	IcamSystems GmbH
IDE	Integrated Development Environment
JS	JavaScript
MVC	Model View Controller
PDF	Portable Document Format
SME	Subject Matter Expert
SPA	Single-Page Application
u.a.	unter anderem
VID	Vehicle Information Database
XML	Extensible Markup Language
z.B.	zum Beispiel

1 Einleitung

1.1 Projektumfeld

Die IcamSystems GmbH (Icam) bietet Softwarelösungen im Bereich der Kfz-Schadensregulierung an. Sie ist Teil eines Firmenverbundes mit ca. 160 Mitarbeitern. Zu den Kunden zählen unter anderem Versicherungsunternehmen, Sachverständigenorganisationen, Prüfdienstleister und Schadensteuerungsgesellschaften.

Zum Leistungsspektrum gehört die automatisierte Prüfung von Gutachten und Kostenvoranschlägen, welche durch Gutachter oder Werkstätten erstellt wurden.

Das Projekt wurde intern durch das Team Reklamationsbearbeitung aus der Abteilung Prozessautomatisierung in Auftrag gegeben. Die Fachabteilung setzt unternehmensspezifische Prozesse u.a. mittels Business Process Management Software (BPMS) um.

1.2 Projektziel

Ziel ist eine Webanwendung mit einer funktionalen Benutzeroberfläche, welche Daten von einem Backend-Service abfragt, um diese strukturiert und übersichtlich darzustellen. Der Prozess der Reklamationsbearbeitung soll so schneller bearbeitet werden können und die händische Suche nach allen Teilinformationen überflüssig machen.

1.3 Projektbegründung

Die Vorgangsprüfung erfolgt über das automatisierte Prüfgewerk ClaimsGuard (CG). Darin prüft ein individuell erstelltes Regelwerk die, z.B. von Werkstätten, erstellten Kostenvoranschläge und Gutachten. Für die individuellen Parameter jedes Fahrzeugtyps greift der CG auf die recherchierten Fahrzeugdaten der Vehicle Information Database (VID) zurück. Die VID bietet für jedes Kfz-Bauteil Informationen zur Beschaffenheit, Verarbeitung und Lage im Fahrzeug. Dort sind auch die Daten zu den Fahrzeugmodellen insgesamt abgelegt.

Bei der Plausibilitätsprüfung durch den CG entsteht ein detaillierter Prüfbericht, in dem Diskrepanzen aufgeführt werden.

Sollte der Kunde das Prüfergebnis beanstanden, so hat er die Möglichkeit eine Reklamation an die Icam zu senden.

Aktuell können nur die Projektverantwortlichen selbst, mittels des jeweiligen Aktenzeichens der Reklamation, eine händische Suche in mehreren Datenbanken und dem Regelwerk des CG durchführen, um den Vorgang nachzuvollziehen. Es soll für die Bewertung des Vorgangs kompakt ersichtlich sein, welche Regeln ausgelöst wurden und welche Daten der VID in die Kalkulation einbezogen wurden.

Dieser Prozess ist aufgrund des hohen Anteils manueller Arbeit zeitaufwändig und fehleranfällig.

Weiterhin kann dieser, wegen der Komplexität der Datenbanksysteme, nur von Experten durchgeführt werden. Durch das ReklaTool sollen die Suchanfragen vereinfacht und standardisiert werden, was wiederum eine Erweiterung des Nutzerkreises möglich macht.

1.4 Projektschnittstellen

Technische Schnittstellen

In der Abteilung Prozessautomatisierung wurde mittels BPMS eine Datenbank-API erstellt. Diese trägt Daten aus Produktiv- und Archivdatenbanken zusammen. Die Daten werden mit den fahrzeugspezifischen Daten aus der VID ergänzt. Hinzu kommen die im Vorgang ausgelösten Regeln des CG. Dieses Datenpaket wird der Webanwendung in strukturierter Form zur Verfügung gestellt.

Verantwortlichkeit

Das Projekt wird durch die Abteilung Projektmanagement begleitet.

Benutzer der Anwendung

Anwender sind zum jetzigen Stand die Projektverantwortlichen des CG. Aufgrund der bisherigen Komplexität der Datenbankabfragen waren bisher nur Subject Matter Expert (SME) für die Abfragen zuständig. Durch die Vereinfachung dieses Vorgangs, ist eine Erweiterung des Nutzerkreises denkbar.

Endabnahme

Das Ergebnis des Projekts wird in der IT-Abteilung und von den SME getestet und abgenommen. Die Dokumentation wird der Projektmanagementabteilung übergeben.

1.5 Projektabgrenzung

Der Webservice für den Datenbankzugriff wurde via BPMS in der Abteilung Prozessautomatisierung realisiert und bereitgestellt. Er existiert unabhängig vom ReklaTool.

2 Projektplanung

2.1 Projektphasen

Für die Bearbeitung des Projekts standen dem Autor im Projektzeitraum täglich etwa 5 Stunden zur Verfügung. Insgesamt wurde das Projekt in 80 Stunden umgesetzt. Diese Zeit wurde in Phasen aufgeteilt, welche den Projektablauf widerspiegeln. Aus Tabelle 1 kann die grobe Zeitplanung entnommen werden.

Eine detailliertere Zeitplanung befindet sich im Anhang A.1: Detaillierte Zeitplanung auf Seite i.

Projektphase	Geplante Zeit
Analyse	8 h
Entwurf	16 h
Implementierung und Tests	40 h
Abnahme	3 h
Dokumentation	13 h
Gesamt	80 h

Tabelle 1: Zeitplanung

2.2 Ressourcenplanung

Im Anhang A.2: Ressourcen auf Seite ii befindet sich eine Auflistung der verwendeten Ressourcen. Dort werden sowohl Hardware- und Softwareressourcen, als auch das beteiligte Personal aufgeführt. Es wurde nur auf Software zurückgegriffen, für die bereits Lizenzen im Unternehmen vorhanden war, bzw. die kostenfrei genutzt werden konnte. Dieser Umstand wirkte sich positiv auf die Projektkosten aus.

2.3 Entwicklungsprozess

Das Projekt wird vom Autor als einzelner Entwickler in einem überschaubaren Zeitraum von drei Wochen umgesetzt. Aus diesen Gründen und da die Anforderungen zu Beginn schon klar definiert wurden, wurde das Projekt anhand der Phasen des Wasserfallmodells umgesetzt. Eine Eigenschaft des Wasserfallmodells ist die lineare Abfolge der einzelnen Projektphasen. Bei der Implementierung wurde bewusst ein inkrementeller Ansatz gewählt, um die Produktqualität zu gewährleisten.

Besonders bei Analyse und Entwurf ist die Rücksprache mit dem Fachbereich vorgesehen.

Zur Sicherstellung der späteren Wartbarkeit wurde die Webanwendung nach den fünf SOLID-Prinzipien für objektorientierte Programmierung entworfen.

Um die einzelnen Schritte während der Implementierung nachvollziehbar zu halten, wurde das Projekt in die Versionsverwaltung für Quellcode via *Git* integriert. Dazu wurden Änderungen im lokalen Projektverzeichnis mit dem Clientprogramm SmartGit überwacht. Bei Fertigstellung einer Implementierungseinheit wurde das lokale Projekt in das Remote-Repository im Firmennetz gepusht. Die Versionsverwaltung in der Firma erfolgt über eine lokal gehostete Instanz von GitLab. Zum Zweck des Codereview wurden Mitarbeiter der IT-Abteilung zum GitLab-Projekt hinzugefügt.

Die Qualitätssicherung während der Entwicklung der Webanwendung wurde durch Unit-Tests umgesetzt. Die Tests sorgen dafür, dass das erwartete Verhalten einzelner Komponenten sichergestellt wird. Weiterhin ist dadurch sichergestellt, dass zukünftige Änderungen an der Anwendung deren Lauffähigkeit nicht beeinträchtigen.

Die Integration der einzelnen Module der Anwendung wurde während der Entwicklung immer wieder durch Whitebox-Tests sichergestellt.

3 Analysephase

Um die Anforderungen an das ReklaTool zu bestimmen, wurde der aktuelle Prozess zur Reklamationsbearbeitung zusammen mit der Fachabteilung erfasst. Aus der Analyse des Vorgangs ergaben sich einzelne Anwendungsfälle (Use-Cases) für die zu erstellende Webanwendung. Basierend auf dem Umfang der Anwendung wurde eine Wirtschaftlichkeitsanalyse durchgeführt und schließlich das Lastenheft erstellt.

3.1 Ist-Analyse

Um den im Folgenden beschriebenen Prozess zu verbildlichen, wurde das im Anhang A.5: Aktivitätsdiagramm auf Seite v befindliche Schaubild erstellt.

Zum Leistungsspektrum der Icam gehört die automatisierte Prüfung von Gutachten und Kostenvoranschlägen, welche durch Gutachter oder Partnerwerkstätten erstellt wurden. Die Prüfung wird durch die Versicherungen oder Sachverständigenorganisationen (Kunden) veranlasst und wird durch den CG durchgeführt. Als Ergebnis bekommt der Kunde zusätzlich zum Prüfbericht eine Rückmeldung über die ausgelösten Regeln des CG.

Im Fall, dass die Regelauslösungen für den Kunden nicht nachvollziehbar sind, hat er die Möglichkeit eine Reklamation an die Icam zu senden. Diese wird firmenintern geprüft und an die Fachabteilung Prozessautomatisierung weitergeleitet.

Ein/-e Mitarbeiter/-in des Teams Reklamationsbearbeitung recherchiert anschließend mittels des Aktenzeichens des Vorgangs die dazugehörigen Daten. Je nach Zeitpunkt des Vorgangs können die Daten in Produktiv- oder Archivdatenbanken abgelegt sein. Der/die Mitarbeiter/-in sucht mittels Datenbankmanagementsystem (DBMS) und selbst erstellten SQL-Skripten nach allen Daten zum Vorgang.

Mit diesen Daten kann dann die beanstandete Regelauslösung nachvollzogen werden. Dazu bietet der CG die Möglichkeit Vorgänge Schritt für Schritt durchzugehen. Fällt dabei eine fehlerhafte Prüfregel auf, so wird diese angepasst.

Liegt die Ursache der fehlerhaften Regelauslösung nicht in der Regelimplementierung, wird die Reklamation an die Rechercheabteilung gegeben. Diese prüft die Fahrzeugdaten des Vorgangs auf Vollständigkeit und Korrektheit. Auch hier werden bei Auffälligkeiten Daten korrigiert oder nachrecherchiert und ergänzt.

Der Kunde bekommt Rückmeldung über die angepassten Daten und Prüfregeln. Sind die Daten jedoch nach Prüfung initial richtig gewesen, so bekommt der Kunde auch darüber eine Rückmeldung, inklusive von Quellen (z.B. Herstellerdaten) als Nachweis. Danach ist der Prozess abgeschlossen.

3.2 Wirtschaftlichkeitsanalyse

Wie bereits im Abschnitt Ist-Analyse zu erkennen ist, war der vorherige Reklamationsprozess mit viel manueller Recherchearbeit verbunden. Im Folgenden wird analysiert, ob das Projekt durch die Zeitersparnis, welche mit dem ReklaTool einhergeht, wirtschaftlich sinnvoll ist.

3.2.1 „Make or Buy“-Entscheidung

Das ReklaTool greift mittels des Webservice auf sensible Firmendaten zu, mit denen auch Verpflichtungen gegenüber Kunden einhergehen. Weiterhin soll die Webanwendung unternehmensspezifischen Anforderungen genügen. So liegen Daten zwar im XML-Format vor, sind aber je nach Anforderung stark firmenspezifisch strukturiert. Aus diesen Gründen ist von dem Beziehen von Software von Dritten abzusehen und die Eigenentwicklung vorzuziehen.

3.2.2 Projektkosten

Dieser Abschnitt betrachtet die Kosten, welche bei der Umsetzung des Projekts entstehen.

Diese setzen aus Personalkosten und aus Kosten für die verwendeten Ressourcen (siehe Kapitel 2.2: Ressourcenplanung) zusammen und können der Tabelle 2 entnommen werden.

Die Personalkosten (brutto) je Projektmitarbeiter wurden durch die Personalabteilung vorgegeben. Für einen Mitarbeiter der IT-Abteilung wurde ein Stundensatz in Höhe von 30,00 € angenommen. Mitarbeiter der Fachabteilung gehen mit 35,00 € pro Stunde in die Berechnung ein. Da der Autor aufgrund seiner Umschulung nicht direkt von der Praktikumsfirma bezahlt wird, wird für diesen das Azubigehalt des dritten Lehrjahrs, mit 7,50 € pro Stunde angenommen.

Die Betriebskosten für die Webanwendung werden mit jährlich 110,00 € beziffert und für die Nutzung der Ressourcen gilt ein Pauschalbetrag von 20,00 € pro Stunde.

Die Gesamtkosten des Projekts betragen somit 2575,00 €.

Vorgang	Beteiligung	Zeit	Kosten pro Stunde	Kosten
Entwicklung	Autor	80 h	7,50 € + 20,00 € = 27,50 €	2200,00 €
Code-Review	IT-Abteilung	1 h	30,00 € + 20,00 € = 50,00 €	50,00 €
Abnahme technisch	IT-Abteilung	1 h	30,00 € + 20,00 € = 50,00 €	50,00 €
Analyse/Lastenheft/Mock-Ups	Fachabteilung	4 h	35,00 € + 20,00 € = 55,00 €	220,00 €
Abnahme fachlich	Fachabteilung	1 h	35,00 € + 20,00 € = 55,00 €	55,00 €
				2575,00 €

Tabelle 2: Kostenaufstellung

3.2.3 Amortisation

Der Wegfall der in Abschnitt 3.1 Ist-Analyse beschriebenen Recherchearbeit in mehreren Datenbanken schlägt sich in einer Zeitersparnis nieder, welche sich u. a. über die Lohnkosten als finanzieller Vorteil beziffern lässt.

Vorgang	Anzahl pro Jahr	Zeit (alt) pro Vorgang	Zeit (neu) pro Vorgang	Ersparnis pro Jahr
Entwicklung	110	15 min	1 min	1540 min
				1540 min

Tabelle 3: Zeitersparnis

Berechnung der Amortisationsdauer

Die Ermittlung der Amortisationsdauer soll dabei helfen, die Wirtschaftlichkeit des Projekts zu beurteilen. Für die Berechnung wird die vorher im Abschnitt 3.2.2 Projektkosten kalkulierte Gesamtsumme mit den Einsparungen verglichen. Zur Ermittlung der finanziellen Ersparnis wird die jedes Jahr gewonnene Zeit mit dem Stundensatz, plus der Ressourcenpauschale, multipliziert. Es ergeben sich somit:

$$\frac{1540 \text{ min/Jahr}}{60} \cdot (35 + 20) \text{ €/h} \approx 1411,67 \text{ €/Jahr.} \quad (1)$$

Um nun die Amortisationsdauer zu berechnen, werden die Projektkosten mit den jährlichen Ersparnissen ins Verhältnis gesetzt. Dabei werden auch die Betriebskosten des ReklaTool von 110,00 € pro Jahr mit beachtet.

$$\frac{2575,00 \text{ €}}{(1411,67 - 110,00) \text{ €/Jahr}} \approx 1,98 \text{ Jahre.} \quad (2)$$

In der Abbildung Anhang A.3: Break-Even-Point auf Seite iii ist die Amortisierung grafisch dargestellt. An der Stelle, wo sich Projektkosten und Einsparung nach einer bestimmten Zeit treffen, lässt sich der *Break-Even-Point* ablesen. Durch diesen hat der Autor eine Aussage darüber, ab welchem Zeitpunkt sich das Projekt amortisiert hat.

Da sich der Arbeitsablauf der Fachabteilung auf absehbare Zeit wenig verändern wird, ist die Amortisierung nach knapp zwei Jahren als wirtschaftlich zu betrachten. Hinzu kommen die im folgenden Kapitel beschriebenen Aspekte.

3.3 Nicht-monetärer Nutzen

Der alte Reklamationsprozess (Vgl. Kapitel 3.1 Ist-Analyse) wurde aufgrund seiner Komplexität meist von SME durchgeführt. Durch die hohe Priorität der Reklamationen wurde der Arbeitsablauf dieser Mitarbeiter für einen längeren Zeitraum unterbrochen. Diese Unterbrechung wird durch den automatisierten Zugriff auf alle relevanten Quellen mit dem Reklatool verkürzt. Durch die einfachere Handhabung ist nun auch die Einbeziehung weiterer Mitarbeiter in das Reklamationsmanagement möglich.

Weiterhin wird durch die Vereinheitlichung des Prozesses und die Reduzierung manueller Teilschritte, die Fehlerquote minimiert.

3.4 Anwendungsfälle

Zusammen mit der Fachabteilung wurden während der Analyse Anforderungen definiert und daraus Anwendungsfälle abgeleitet. Im Anhang A.6: Use-Case-Diagramm auf Seite vi befindet sich das dabei entstandene Use-Case-Diagramm.

Die Ausdifferenzierung der einzelnen Fälle diente dann später im Entwurf als Richtlinie für einzelne Features.

3.5 Qualitätsanforderungen

Die Software soll eine funktionale und übersichtliche Benutzeroberfläche bieten. Diese soll ohne Installation, in Form einer Webanwendung im Browser erreichbar sein. Nach einer Nutzeranfrage an die Anwendung soll das Ergebnis in einer festgelegten Zeit erscheinen, um den Arbeitsablauf nicht zu sehr zu stören. Die Funktionalität und Richtigkeit der Software soll mittels Tests sichergestellt werden.

3.6 Lastenheft/Fachkonzept

Aus den zusammengetragenen Anforderungen erstellte der Autor mit der Fachabteilung das Lastenheft. Darin sind alle gewünschten Funktionen und Eigenschaften der zu erstellenden Software festgeschrieben.

Ein Auszug befindet sich im Anhang A.4: Lastenheft (Auszug) auf Seite iii.

4 Entwurfsphase

Dieser Abschnitt behandelt alle wichtigen Entscheidungen bezüglich der Grundprinzipien, Techniken und Technologien zum Erstellen der Webanwendung.

4.1 Zielplattform

Wie unter 1.2 Projektziel erwähnt, soll eine Webanwendung erstellt werden. Aus der IT-Abteilung kamen dazu Informationen zu bereits in anderen Projekten eingesetzten Technologien.

Neue Software wird dort überwiegend in der Programmiersprache C# umgesetzt. Dadurch existiert für diese bereits umfangreiches Wissen bei den Mitarbeitern. Bereits vorhandene Ressourcen und Lizenzen konnten für das Projekt genutzt werden. Um das Endprodukt besser im Team wartbar und erweiterbar zu halten, hat sich der Autor ebenfalls für C# in der .NET-Umgebung von Microsoft entschieden.

Unter weiterer Rücksprache mit der Softwareabteilung wurden verschiedene Ansätze zur Umsetzung einer Webanwendung durchgesprochen. Besonders wurde hier zwischen client- und serverbasierten Anwendungen unterschieden. Da die Benutzerinteraktion eher gering ausfallen würde, wurde sich für die Servervariante entschieden. Diese ist gegenüber einer Single-Page Application (SPA) auf einem Client weniger agil und die Reaktionszeit ist eingeschränkt.

Die Wahl fiel hier auf das Webframework *ASP.Net Core MVC*, zu dem es schon gute Erfahrungen im Team gab. Es bietet bereits Projektvorlagen mit der grundlegenden Programmstruktur und Möglichkeiten zur Konfiguration an.

4.2 Architekturdesign

Wie bereits an der Wahl des Webframeworks erkennbar ist, basiert die Anwendung auf dem MVC-Entwurfsmuster. Dieses beinhaltet die Teile **Model**, **View** und **Controller** – also Datenabbildung, Benutzeroberfläche und Vermittlung. Durch die klare Trennung in diese drei Bereiche vermindern sich die Abhängigkeiten untereinander. Implementierungen dieser Teile können leichter verändert oder sogar ausgetauscht werden. Ein weiterer Vorteil ist der gut nachvollziehbare Datenfluss innerhalb der Anwendung.

Dieser Kern des Programms wird durch Services erweitert, welche jeweils eine spezielle Aufgabe haben. Dazu bietet das Framework einen Injektor, mit dem das Prinzip der *Inversion of Control* via *Dependency Injection* umgesetzt wird. Die daraus folgende lose Verbindung der einzelnen Module trägt weiter zur besseren Wartbarkeit bei. Dabei müssen sich die Teile des Programms nicht um die Erstellung von Instanzen ihrer Abhängigkeiten kümmern, sondern Bekommen diese vom Injektor übergeben. Dieser übernimmt das komplette Management der erstellten Objekte und kann diese, je nach Lebensdauer, ggf. entfernen, sollten sie nicht mehr gebraucht werden.

Zur Steuerung der Kommunikation mit der Datenbank-API ist ein eigener Service nötig. Um die Performance von mehrfach aufgerufenen Vorgängen zu erhöhen, wird dieser um einen *Caching*-Service erweitert. Beim *Caching* werden die erhaltenen Daten in Dateien gespeichert, welche bestimmten Abfragen zugeordnet und später wieder eingelesen werden können. Das Model wird mittels verhaltensloser Datenklassen erstellt.

4.3 Entwurf der Benutzeroberfläche

Um den einfachen Zugriff von beliebigen Rechnern im Firmenintranet zu gewährleisten, fiel die Entscheidung auf eine Umsetzung mit Weboberfläche. Diese sollte in den gängigsten Browsern angezeigt werden können. Da die Anwendung lediglich auf Desktopcomputern zum Einsatz kommen wird, wurde bewusst auf Anpassungen für mobile Geräte verzichtet und der Fokus auf eine funktionale Gestaltung gelegt.

Über mehrere Korrekturschleifen wurde zusammen mit der Fachabteilung der Entwurf für eine Benutzeroberfläche fertiggestellt. Mithilfe von *Mockups* wurden die Bedürfnisse der Endbenutzer so gut wie möglich nachempfunden. Diese befinden sich im Anhang A.10: Oberflächenentwürfe auf Seite x.

Die Webseite soll in der Hauptsache aus einer Suchleiste für Nutzeranfragen und einem Bereich zur Darstellung der Vorgangsdaten bestehen. Eine Checkbox soll die Möglichkeit geben, den Umfang der Suche einzuschränken.

Die Vorgangsdaten werden nach erfolgreicher Suche in einem Bereich angezeigt, der die einzelnen Daten mittels Registerkarten in Kategorien unterteilt. Zur strukturierten Darstellung werden filterbare Tabellen genutzt. Diese werden durch das Framework *TelerikUI* eines Drittanbieters bereitgestellt und bieten vielfältige Einstellungen zum Filtern, Sortieren und Gruppieren an. Eine Lizenz ist hierfür bereits im Unternehmen vorhanden. Weiterhin soll es möglich sein, eine PDF- oder XML-Datei mit den Vorgangsdaten anzuzeigen bzw. diese zum Download anzubieten (siehe 3.4 Anwendungsfälle).

4.4 Datenmodell

Die Datenmodelle für sowohl Anfragen an die Datenbank-API, als auch deren Antwort basieren auf den versendeten XML-Dateien. Diese wurden vom Fachbereich bereitgestellt und durch Deserialisierung in DTO-Klassen überführt.

Im Anhang A.9: Datenmodell auf Seite ix befindet sich dazu eine Übersicht.

4.5 Geschäftslogik

Unter den Anwendungsfällen ist das Abfragen der Daten vom Datenbankwebservice der wichtigste. Anhand diesem wird ein Großteil des Programmablaufs nachvollziehbar und kann im Sequenzdiagramm im Anhang A.7: Sequenzdiagramm auf Seite vi betrachtet werden.

Im gezeigten Fall ruft die Mitarbeiterin der Fachabteilung über ihren Browser das *ReklaTool* auf. Dort gibt sie über das Suchfeld das Aktenzeichen zum gesuchten Vorgang ein. Da die Aktenzeichen kein festes Format besitzen muss zusätzlich deren Art (Dateiname, Vorgangsnummer, Schadennummer oder Auftragsnummer) ausgewählt werden. Um die Suche einzuschränken kann ein Haken gesetzt werden, ob Dokumente zum Vorgang mitgeliefert werden sollen. Nach

Betätigung der Suchen-Taste wird eine *GET*-Anfrage mit Aktenzeichen und Filter an den *ReklaTool*-Server gestellt. Dort bietet der *VorgangController* Endpunktmethoden für http-Anfragen an. Über den *UiEndpointService* werden die Anfragen an den *HttpMsgService* durchgereicht. Dieser erstellt mittels des *RequestBuilder* ein Anfrage-Objekt.

Die erzeugte Anfrage wird zunächst durch den *ResponseCacheService* mit den bereits zwischengespeicherten Vorgängen verglichen. Ist eine Datei zum Aktenzeichen vorhanden, so wird diese an den *UiEndpointService* zurückgegeben. Andernfalls wird nun eine Anfrage an den *Datenbank-Webservice* gestellt. Das zurückgelieferte XML-Dokument wird deserialisiert und zusammen mit den Anfrageinformationen in die *Cache* abgelegt. Danach geht auch dieses zurück an den *UiEndpointService*.

Hier werden die Daten, je nach Client-Anfrage, vom kompletten Vorgangs-Model auf einzelne *View-Models* gemappt. Zurück im *VorgangController* wird nun die Server-Response zurück an den Client gesendet und die Daten an die entsprechenden Komponenten gebunden, wo sie für den Nutzer sichtbar werden.

4.6 Maßnahmen zur Qualitätssicherung

4.7 Pflichtenheft/Datenverarbeitungskonzept

5 Implementierungsphase

Bei der Implementierung wurden die in 4 Entwurfsphase beschriebenen Architekturvorgaben und Programmteile umgesetzt. Begonnen wurde mit dem Erstellen des Projekts nach einer vom *Telerik-Framework* bereitgestellten Vorlage für eine *ASP.Net Core MVC*-Anwendung. Diese liefert bereits *Boilerplate-Code*, der z. B. die Konfiguration vereinfachen soll.

5.1 Konfiguration

dhfg

5.2 Implementierung der Datenstrukturen

Zur Erstellung der Einzelnen DTOs wurden XML-Dateien aus dem Fachbereich deserialisiert. Diese wurden zur Kommunikation mit dem Datenbank-Webservice vorgegeben. Dabei handelt es sich um eine Anfrage- und ein Antwortformat.

Das Anfrage-Model beinhaltet das jeweilig gewünschte Aktenzeichen und eine Information über die Anwendung eines Suchfilters, welcher die zurückgelieferten Daten einschränkt.

Das Antwort-Model wurde bei der automatischen Deserialisierung bereits mit den notwendigen

Annotationen versehen, um einen reibungslosen Austausch mit dem XML-Format zu garantieren. Zur Datenbindung an die einzelnen Komponenten der Benutzeroberfläche wurden weitere *Viewmodels* erstellt, welche das Mapping auf das Antwort-Model ermöglichen.

Für das *Caching* wurde ein eigenes DTO umgesetzt, welches das Anfrage- und Antwort-Model enthält. Dieses wird beim Lesen und Schreiben der *Cache*-Dateien serialisiert bzw. deserialisiert. Über das *Hashing* des gesamten Cache-Objekts sind die Dateien eindeutig identifizierbar.

5.3 Implementierung der Benutzeroberfläche

Bereits in 4.3 Entwurf der Benutzeroberfläche wurde festgelegt, dass die Benutzeroberfläche als Webseite in einem Browser ausgeführt werden soll.

Die von *TelerikUI* bereitgestellte Vorlage für eine Model View Controller (MVC)-Anwendung verlangt die Implementierung der *View* mittels spezieller *.cshtml*-Dateien. Diese erlauben, mittels definierter Schlüsselwörter, das Verwenden von *C#*-Befehlen in einem *html*-Dokument. Die Oberflächenkomponenten, wie z. B. Buttons, Suchfelder oder Tabellen, werden daher auch in *C#* bereitgestellt und darüber konfiguriert. Siehe Listing.

Um die ereignisbasierte Interaktion mit dem Nutzer zu gestalten, wurde, wie bei den meisten Webseiten, auf die Verwendung der Programmiersprache JS zurückgegriffen.

5.4 Implementierung der Geschäftslogik

Laut 3.4 Anwendungsfälle ist der Hauptanwendungsfall das Abfragen der Daten vom Datenbankservice. Anhand dessen wurde in der Entwurfsphase ein Sequenz-Diagramm erstellt (Vgl. Anhang A.7: Sequenzdiagramm auf Seite vi).

Nachdem das Projekt erstellt wurde, wurden nun die aus den Diagrammen ersichtlichen Serviceklassen und die dazugehörigen Interfaces erstellt. Danach konnten diese in der Datei *Program.cs* über *builder.Services* dem Programm-Container hinzugefügt werden und standen somit dem Injektor des Frameworks zur Verfügung. Nach diesem Schritt wurden nun die Services wie im Klassendiagramm Anhang A.15: Klassendiagramm auf Seite xx miteinander verknüpft. Dies geschieht über DI, speziell Konstruktorinjektion.¹

6 Abnahmephase

Vor der Auslieferung des Programms wurde ein Code-Review durch IT-Abteilung durchgeführt. Nachdem der Code freigegeben wurde, konnte eine Mitarbeiterin der Fachabteilung die einzelnen

¹Dependency Injection in der Microsoft-Dokumentation:

<https://learn.microsoft.com/de-de/dotnet/core/extensions/dependency-injection>

Funktionen des ReklaTool testen. Nach der Abnahme der Software wurde diese an die IT-Abteilung übergeben.

7 Einführungsphase

Die Auslieferung des fertigen Programms wird durch einen Mitarbeiter aus der IT-Abteilung realisiert. Das ReklaTool soll innerhalb eines mit der Containerisierungs-Software Docker erstellten Containers laufen. Dort wird noch eingestellt, wie der Server über das Netzwerk erreichbar sein soll.

8 Dokumentation

Um den Einstieg in das Programm zu vereinfachen und dessen Aufbau zu beschreiben, wurde eine Dokumentation angefertigt. Diese besteht aus dem Benutzerhandbuch für die Endanwender und der Entwicklerdokumentation. Im Handbuch werden die einzelnen Funktionen der Webanwendung und die Gliederung der Benutzeroberfläche mit Texten und Bildern dargestellt. Dadurch wird die Anlernphase verkürzt.

Ein Auszug aus dem Benutzerhandbuch befindet sich im Anhang A.16: Benutzerdokumentation auf Seite xxi.

Die Entwicklerdokumentation wurde über Screenshots aus der IDE erstellt. Diese wurden nummeriert und in Textform beschrieben. Ausschnitte aus diesem Dokument sind im Anhang A.12: Entwicklerdokumentation auf Seite xiv zu finden.

9 Fazit

Durch die Einführung des ReklaTool konnten die Mitarbeiter spürbar entlastet werden. Die Verkürzung der Abfragedauer sorgt dafür, dass die täglichen Arbeitsabläufe weniger stark unterbrochen werden. In der gewonnenen Zeit können sich die Nutzer, zu denen auch Teamleiter gehören, anderen Aufgaben zuwenden. Sobald weitere Nutzer durch den vereinfachten Prozess hinzugefügt werden, wird sich dieser Effekt noch mehr verstärken. Nicht zu klein zu bewerten ist der finanzielle Aspekt dieser Optimierung, welche dadurch als Erfolg gewertet werden darf.

9.1 Soll-/Ist-Vergleich

In Tabelle 4 ist zu erkennen, dass die Zeitplanung überwiegend eingehalten werden konnte. Über das gesamte Projekt hinweg kam es in zwei Phasen zu Abweichungen von je einer Stunde.

Phase	Geplant	Tatsächlich	Differenz
Analyse	8 h	8 h	
Entwurf	16 h	17 h	+1 h
Implementierung	40 h	39 h	-1 h
Abnahme	3 h	3 h	
Dokumentation	13 h	13 h	
Gesamt	80 h	80 h	

Tabelle 4: Soll-/Ist-Vergleich

Durch die gute Kommunikation mit dem Fachbereich bei der Analyse, Planung und Durchführung, konnten die angesetzten Zeiträume überwiegend eingehalten werden. Lediglich in der Entwurfsphase wurde eine Stunde mehr aufgewendet. Dieses kam der Implementierungsphase zugute, welche um eine Stunde verkürzt werden konnte.

9.2 Lessons Learned

Die Umsetzung des Projekts brachte dem Autor viele Erfahrungen im Bereich der Projektplanung und -durchführung. Dabei nahm die Kommunikation mit den Kollegen der Fachabteilung und der IT-Abteilung eine besondere Stelle ein. Die Planung und Implementierung der Software waren dazu geeignet, sich weiter in die einzelnen Programmarchitekturen zu vertiefen. Dabei konnten wichtige Erkenntnisse sowohl über die verwendeten Frameworks, als auch Programmiersprachen gesammelt werden. Hervorzuheben sind hier die Wirkmechanismen der Kommunikation über HTTP.

9.3 Ausblick

Mit dem Projektabschluss wurden die Anforderungen der Fachabteilung erfüllt. Durch Veränderung der Prozesse innerhalb des Unternehmens entsteht jedoch durchaus eine Perspektive auf zukünftige Anpassungen des ReklaTool, um diesem Umstand bestmöglich Rechnung zu tragen. Optimierungen im Hinblick auf die Erweiterung des Nutzerkreises sind genauso denkbar, wie die Erweiterung der Darstellung der vom Datenbankservice bereitgestellten Daten.

Literaturverzeichnis

Eidesstattliche Erklärung

Ich, Ingolf Schieck, versichere hiermit, dass ich meine **Dokumentation zur betrieblichen Projektarbeit** mit dem Thema

*ReklaTool – Entwicklung einer Webanwendung zur Abfrage und Anzeige
von Kalkulationsvorgängen aus dem Kfz-Bereich
nach dem Client-Server-Prinzip*

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, wobei ich alle wörtlichen und sinngemäßen Zitate als solche gekennzeichnet habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Leipzig, den 23.11.2022

INGOLF SCHIECK

A Anhang

A.1 Detaillierte Zeitplanung

Analysephase	8 h
1. Analyse des Ist-Zustands	1 h
2. Make-Or-Buy-Analyse	1 h
3. Ermittlung von Use-Cases zusammen mit dem Fachbereich und Erstellung eines Use-Case-Diagramms	1 h
4. Soll-Konzept	2 h
5. Unterstützung der Fachabteilung bei der Erstellung des Lastenhefts	1 h
6. Wirtschaftlichkeitsbetrachtung mit Amortisationsrechnung	1 h
7. Zeit- und Ablaufplanung	1 h
Entwurfsphase	16 h
1. Planung der Projektstruktur (Versionskontrolle, Ordnerstruktur, etc.)	3 h
2. Erstellung von Mock-Ups der Weboberfläche	4 h
3. Entwurf der Systemabgrenzung, Erstellung eines Systemkontextdiagramms	2 h
4. Entwurf des Programmflusses, Erstellung eines Sequenzdiagramms	4 h
5. Entwurf der Klassen, Erstellung eines Klassendiagramms	3 h
Implementierung und Tests	40 h
1. Erstellung und Konfigurierung des C#-Projekts	3 h
2. Erstellen der Modell-Klassen	3 h
3. Erstellen des http-Service, inkl. Tests	8 h
4. Erstellen des Cache-Service, inkl. Tests	6 h
5. Erstellung und Ausführen von Integrationstests	6 h
6. Implementierung der User-Authentifizierung	6 h
7. Erstellung der Weboberfläche, inkl. Datenbindung und Interaktionen	8 h
Abnahme	3 h
1. Codereview und technische Abnahme	2 h
2. Abnahme durch den Fachbereich	1 h
Dokumentation	13 h
1. Erstellen der Nutzerdokumentation	4 h
2. Erstellen der Entwicklerdokumentation	1 h
3. Projektbewertung	1 h
4. Erstellen der Projektdokumentation	7 h
Gesamt	80 h

A.2 Ressourcen

Hardware

- Büroarbeitsplatz mit MS Windows Desktop-PC

Software

- Microsoft Windows 10 Pro – Betriebssystem
- Microsoft Visual Studio Enterprise 2022 (64-bit) – v17.3.6 – Entwicklungsumgebung
- Microsoft MS Test – Framework für Unit-Tests in Visual Studio
- JetBrains ReSharper Tools v2022.1 – Erweiterung für Visual Studio für u. a. Refactoring
- Microsoft Visual Studio Code v1.72.1 – Texteditor mit Highlightfunktion und vielen Erweiterungen
- draw.io – Programm zum Erstellen von UML-Diagrammen
- syntevo SmartGit v21.1.1 – Clientprogramm zur Versionsverwaltung mit Git
- GitLab – intern gehostete Versionsverwaltung
- Progress Telerik UI – Framework für Benutzeroberflächen
- Google Chrome (64-bit) v105.0.5195.102 – Webbrowser
- MiKTeX – Distribution des LaTeX-Textsatzsystems
- LaTeX Workshop – Erweiterung für Visual Studio Code zum Editieren von LaTeX-Dokumenten
- Atlassian Confluence – Wiki-Software zur Projektdokumentation
- Atlassian Jira – Ticketsystem zur Projektverwaltung
- Postman
- Screenpresso

Personal

- Umschüler Fachinformatiker für Anwendungsentwicklung – Umsetzung des Projekts
- Entwickler der Softwareabteilung – Codereview, technische Abnahme
- Entwicklerin der Fachabteilung – Anforderungen des Reklatool, Unterstützung beim Entwurf der Benutzeroberfläche, Endabnahme der Anwendung

A.3 Break-Even-Point

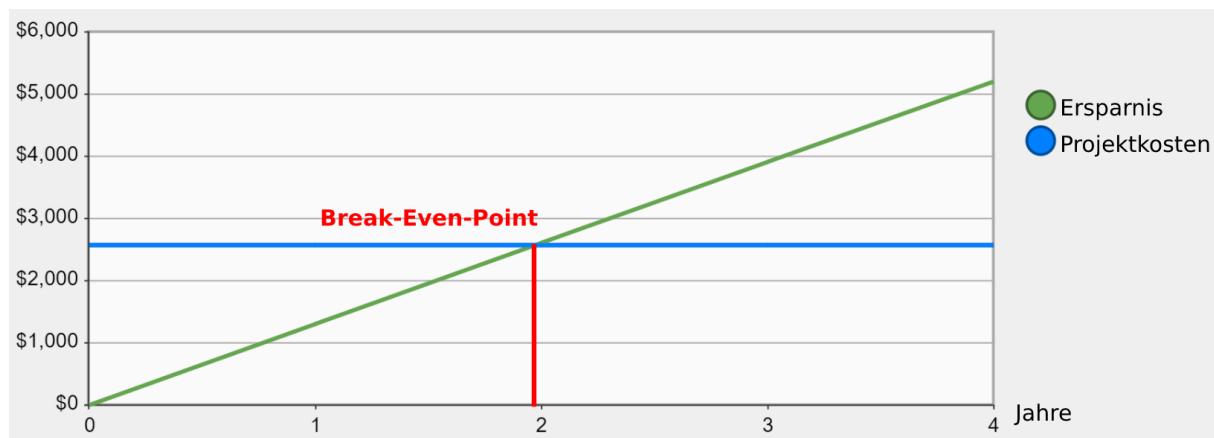


Abbildung 1: Break-Even-Point

A.4 Lastenheft (Auszug)

Es folgt ein Auszug aus dem Lastenheft mit Fokus auf die Anforderungen:

Die Anwendung muss folgende Anforderungen erfüllen:

1. Verarbeitung der Moduldaten
 - 1.1. Die Anwendung muss die von Subversion und einem externen Programm bereitgestellten Informationen (z.B. Source-Benutzer, -Datum, Hash) verarbeiten.
 - 1.2. Auslesen der Beschreibung und der Stichwörter aus dem Sourcecode.
2. Darstellung der Daten
 - 2.1. Die Anwendung muss eine Liste aller Module erzeugen inkl. Source-Benutzer und -Datum, letztem Commit-Benutzer und -Datum für alle drei Umgebungen.
 - 2.2. Verknüpfen der Module mit externen Tools wie z.B. Wiki-Einträgen zu den Modulen oder dem Sourcecode in Subversion.
 - 2.3. Die Sourcen der Umgebungen müssen verglichen und eine schnelle Übersicht zur Einhaltung des allgemeinen Entwicklungsprozesses gegeben werden.
 - 2.4. Dieser Vergleich muss auf die von einem bestimmten Benutzer bearbeiteten Module eingeschränkt werden können.
 - 2.5. Die Anwendung muss in dieser Liste auch Module anzeigen, die nach einer Bearbeitung durch den gesuchten Benutzer durch jemand anderen bearbeitet wurden.
 - 2.6. Abweichungen sollen kenntlich gemacht werden.
 - 2.7. Anzeigen einer Übersichtsseite für ein Modul mit allen relevanten Informationen zu diesem.

3. Sonstige Anforderungen

- 3.1. Die Anwendung muss ohne das Installieren einer zusätzlichen Software über einen Webbrowser im Intranet erreichbar sein.
- 3.2. Die Daten der Anwendung müssen jede Nacht bzw. nach jedem **SVN!**-Commit automatisch aktualisiert werden.
- 3.3. Es muss ermittelt werden, ob Änderungen auf der Produktionsumgebung vorgenommen wurden, die nicht von einer anderen Umgebung kopiert wurden. Diese Modulliste soll als Mahnung per E-Mail an alle Entwickler geschickt werden (Peer Pressure).
- 3.4. Die Anwendung soll jederzeit erreichbar sein.
- 3.5. Da sich die Entwickler auf die Anwendung verlassen, muss diese korrekte Daten liefern und darf keinen Interpretationsspielraum lassen.
- 3.6. Die Anwendung muss so flexibel sein, dass sie bei Änderungen im Entwicklungsprozess einfach angepasst werden kann.

A.5 Aktivitätsdiagramm

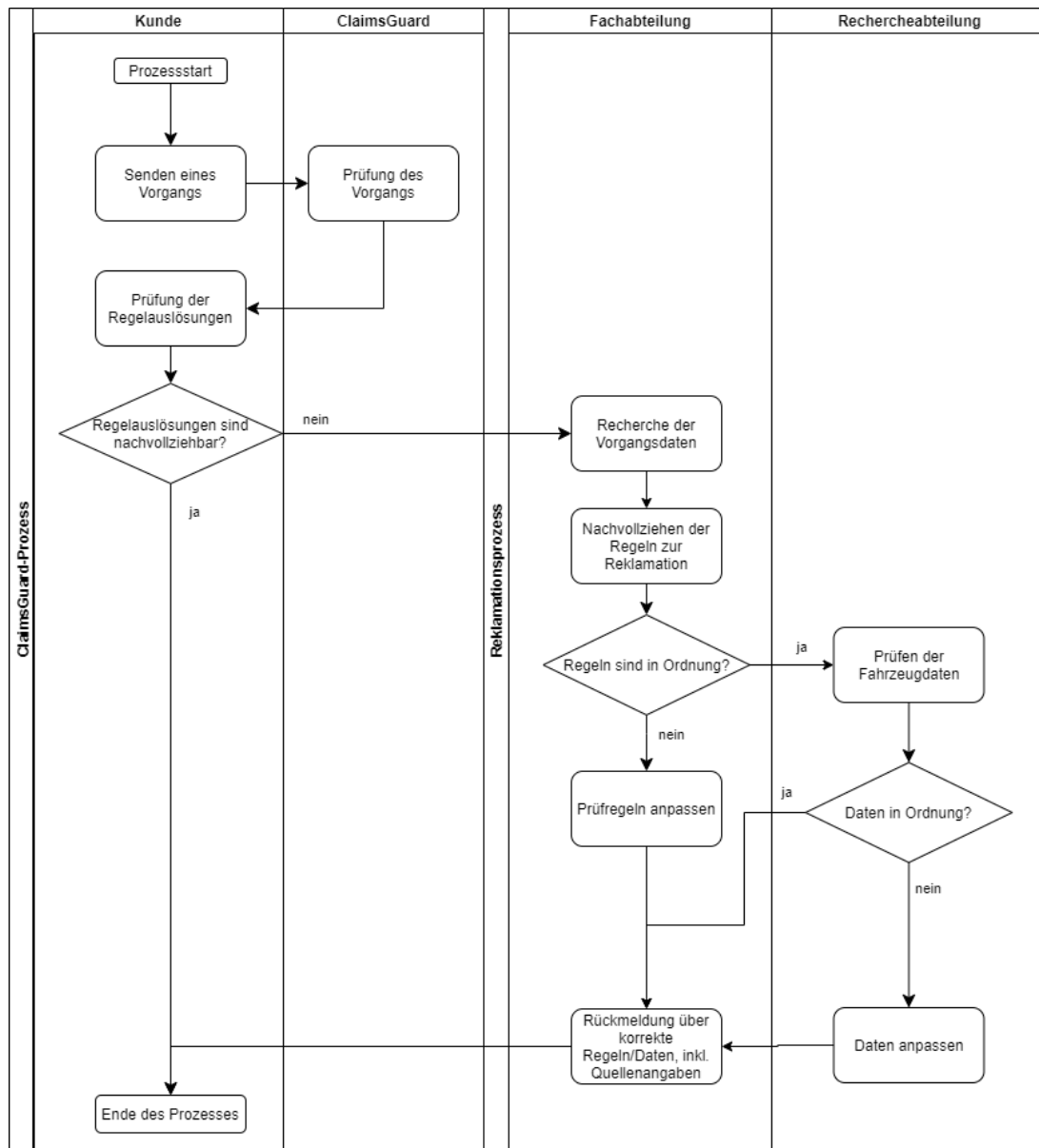


Abbildung 2: Aktivitätsdiagramm

A.6 Use-Case-Diagramm

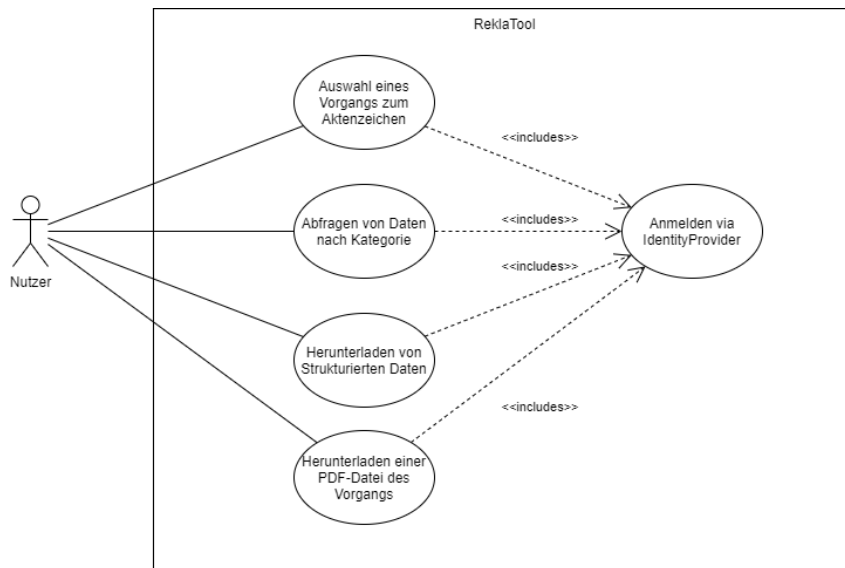


Abbildung 3: Use-Case-Diagramm

A.7 Sequenzdiagramm

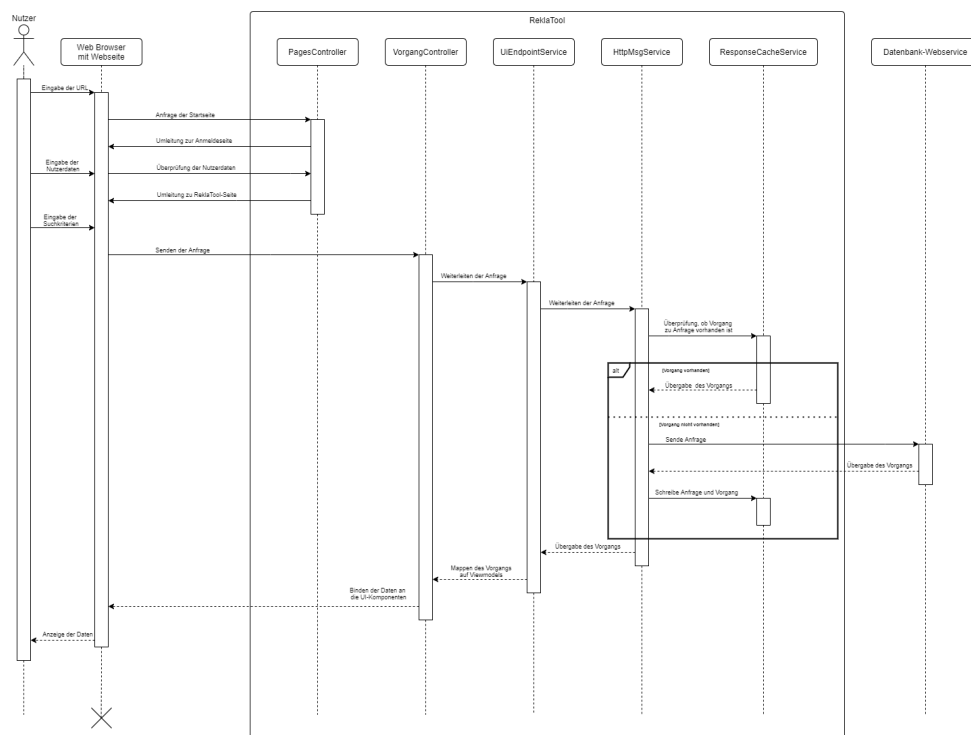


Abbildung 4: Sequenzdiagramm

A.8 Pflichtenheft (Auszug)

Zielbestimmung

1. Musskriterien

1.1. Modul-Liste: Zeigt eine filterbare Liste der Module mit den dazugehörigen Kerninformationen sowie Symbolen zur Einhaltung des Entwicklungsprozesses an

- In der Liste wird der Name, die Bibliothek und Daten zum Source und Kompilat eines Moduls angezeigt.
- Ebenfalls wird der Status des Moduls hinsichtlich Source und Kompilat angezeigt. Dazu gibt es unterschiedliche Status-Zeichen, welche symbolisieren in wie weit der Entwicklungsprozess eingehalten wurde bzw. welche Schritte als nächstes getan werden müssen. So gibt es z. B. Zeichen für das Einhalten oder Verletzen des Prozesses oder den Hinweis auf den nächsten zu tätigenden Schritt.
- Weiterhin werden die Benutzer und Zeitpunkte der aktuellen Version der Sourcen und Kompilate angezeigt. Dazu kann vorher ausgewählt werden, von welcher Umgebung diese Daten gelesen werden sollen.
- Es kann eine Filterung nach allen angezeigten Daten vorgenommen werden. Die Daten zu den Sourcen sind historisiert. Durch die Filterung ist es möglich, auch Module zu finden, die in der Zwischenzeit schon von einem anderen Benutzer editiert wurden.

1.2. Tag-Liste: Bietet die Möglichkeit die Module anhand von Tags zu filtern.

- Es sollen die Tags angezeigt werden, nach denen bereits gefiltert wird und die, die noch der Filterung hinzugefügt werden könnten, ohne dass die Ergebnisliste leer wird.
- Zusätzlich sollen die Module angezeigt werden, die den Filterkriterien entsprechen. Sollten die Filterkriterien leer sein, werden nur die Module angezeigt, welche mit einem Tag versehen sind.

1.3. Import der Moduldaten aus einer bereitgestellten **CSV!**-Datei

- Es wird täglich eine Datei mit den Daten der aktuellen Module erstellt. Diese Datei wird (durch einen Cronjob) automatisch nachts importiert.
- Dabei wird für jedes importierte Modul ein Zeitstempel aktualisiert, damit festgestellt werden kann, wenn ein Modul gelöscht wurde.
- Die Datei enthält die Namen der Umgebung, der Bibliothek und des Moduls, den Programmtyp, den Benutzer und Zeitpunkt des Sourcecodes sowie des Kompilats und den Hash des Sourcecodes.
- Sollte sich ein Modul verändert haben, werden die entsprechenden Daten in der Datenbank aktualisiert. Die Veränderungen am Source werden dabei aber nicht ersetzt, sondern historisiert.

1.4. Import der Informationen aus **SVN!** (**SVN!**). Durch einen „post-commit-hook“ wird nach jedem Einchecken eines Moduls ein **PHP!**-Script auf der Konsole aufgerufen, welches die Informationen, die vom **SVN!**-Kommandozeilentool geliefert werden, an **NatInfo!** übergibt.

1.5. Parsen der Sourcen

- Die Sourcen der Entwicklungsumgebung werden nach Tags, Links zu Artikeln im Wiki und Programmbeschreibungen durchsucht.
- Diese Daten werden dann entsprechend angelegt, aktualisiert oder nicht mehr gesetzte Tags/Wikiartikel entfernt.

1.6. Sonstiges

- Das Programm läuft als Webanwendung im Intranet.
- Die Anwendung soll möglichst leicht erweiterbar sein und auch von anderen Entwicklungsprozessen ausgehen können.
- Eine Konfiguration soll möglichst in zentralen Konfigurationsdateien erfolgen.

Produkteinsatz

1. Anwendungsbereiche

Die Webanwendung dient als Anlaufstelle für die Entwicklung. Dort sind alle Informationen für die Module an einer Stelle gesammelt. Vorher getrennte Anwendungen werden ersetzt bzw. verlinkt.

2. Zielgruppen

NatInfo wird lediglich von den **Natural!** (**Natural!**)-Entwicklern in der EDV-Abteilung genutzt.

3. Betriebsbedingungen

Die nötigen Betriebsbedingungen, also der Webserver, die Datenbank, die Versionsverwaltung, das Wiki und der nächtliche Export sind bereits vorhanden und konfiguriert. Durch einen täglichen Cronjob werden entsprechende Daten aktualisiert, die Webanwendung ist jederzeit aus dem Intranet heraus erreichbar.

A.9 Datenmodell

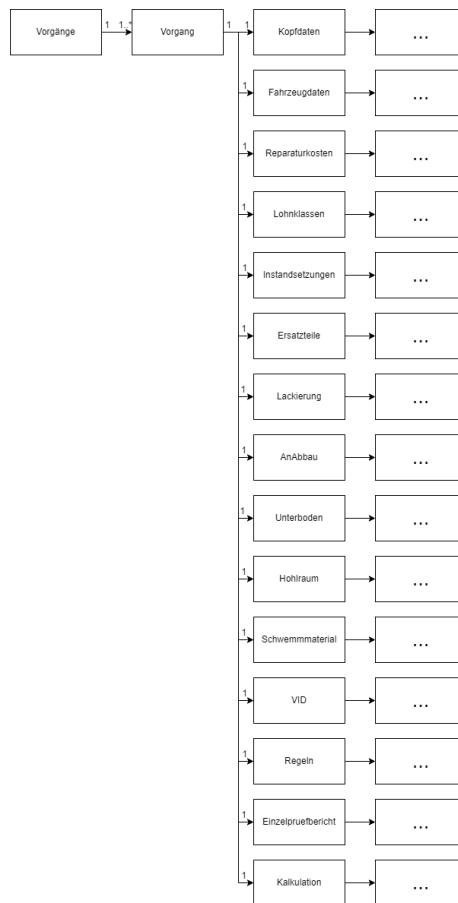


Abbildung 5: XML-Modell für Antwort

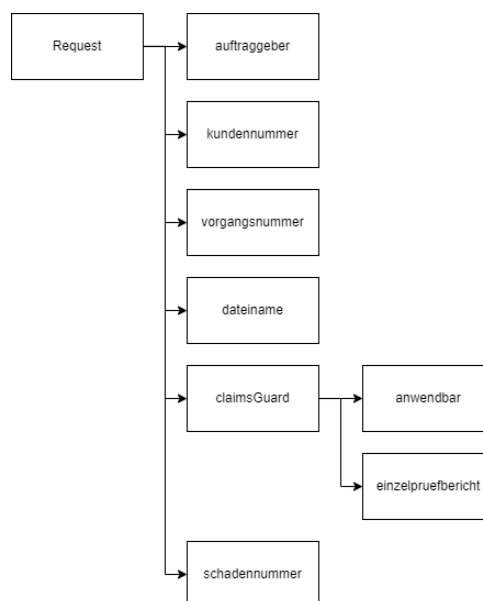
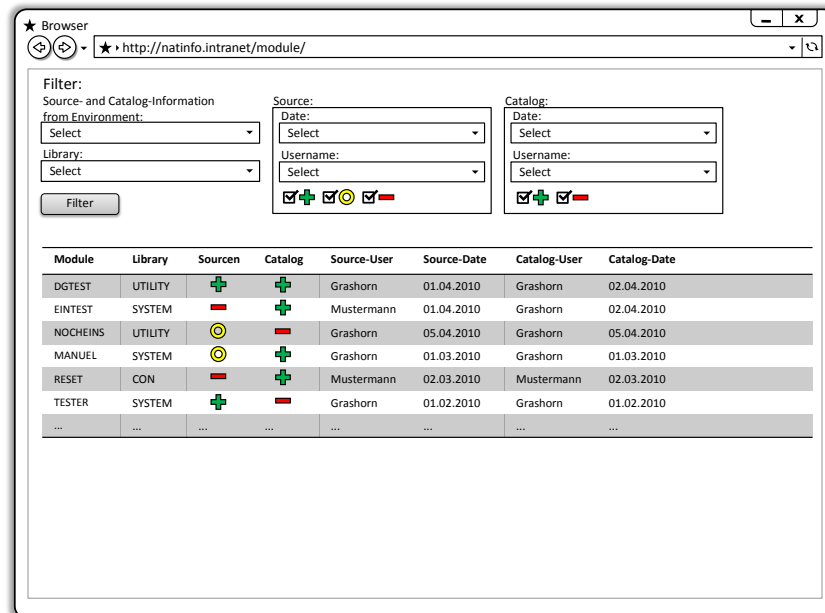


Abbildung 6: XML-Modell für Anfrage

A.10 Oberflächenentwürfe



The screenshot shows a web browser window with the address bar displaying `http://natinfo.intranet/module/`. The page contains a filter section at the top and a table of modules below.

Filter:
Source- and Catalog-Information from Environment:
Library:

Source:
Date:
Username:
☒ ☒ ☒ ☒
Catalog:
Date:
Username:
☒ ☒ ☒ ☒

Module	Library	Sourcen	Catalog	Source-User	Source-Date	Catalog-User	Catalog-Date
DGTEST	UTILITY	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Grashorn	01.04.2010	Grashorn	02.04.2010
EINTEST	SYSTEM	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Mustermann	01.04.2010	Grashorn	02.04.2010
NOCHAINS	UTILITY	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Grashorn	05.04.2010	Grashorn	05.04.2010
MANUEL	SYSTEM	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Grashorn	01.03.2010	Grashorn	01.03.2010
RESET	CON	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Mustermann	02.03.2010	Mustermann	02.03.2010
TESTER	SYSTEM	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Grashorn	01.02.2010	Grashorn	01.02.2010
...

Abbildung 7: Liste der Module mit Filtermöglichkeiten

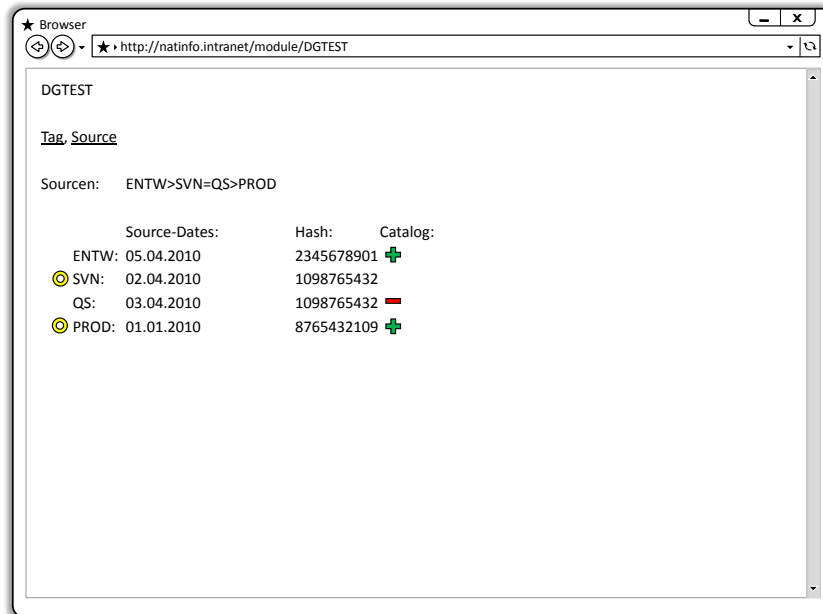


Abbildung 8: Anzeige der Übersichtsseite einzelner Module

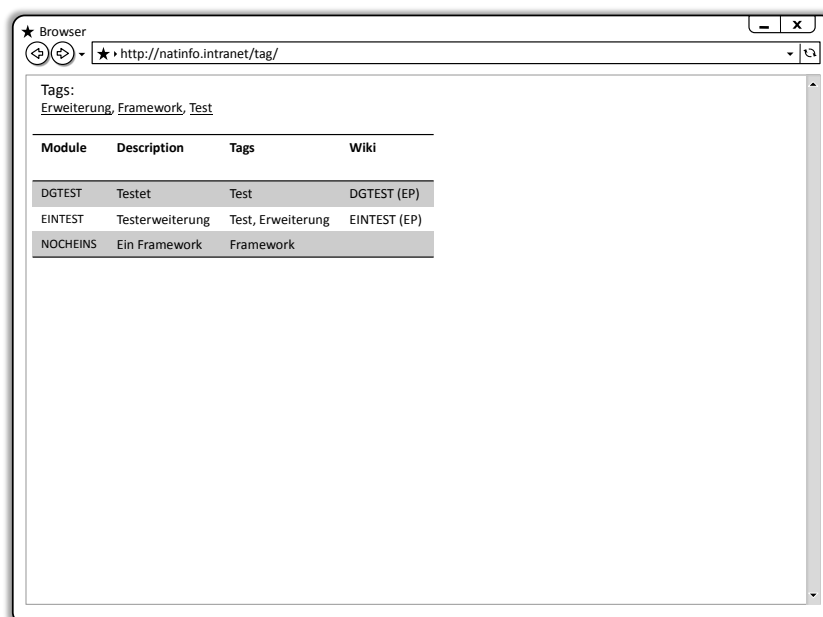


Abbildung 9: Anzeige und Filterung der Module nach Tags

A.11 Screenshots der Anwendung

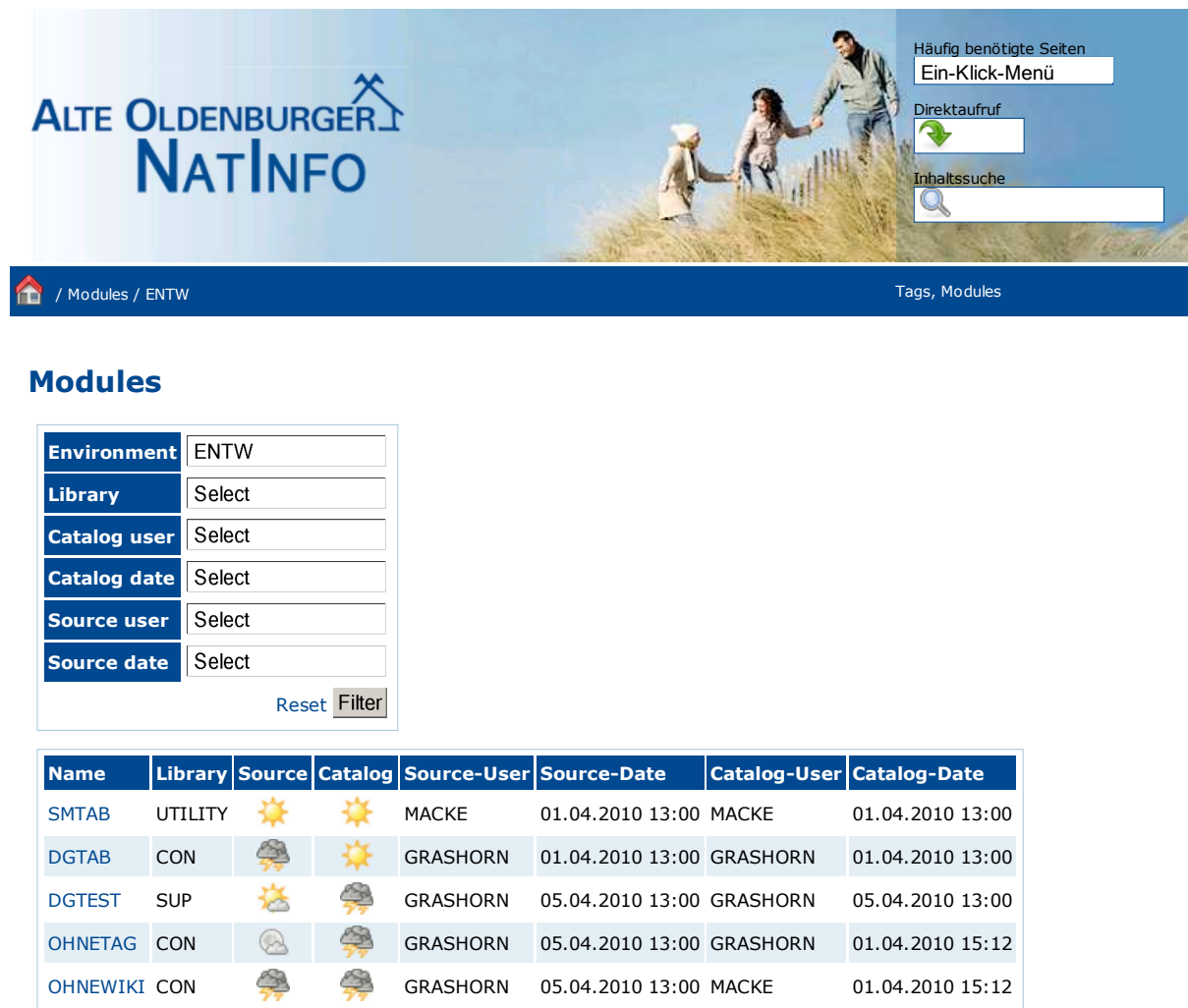


Tags

Project, Test

Modulename	Description	Tags	Wiki
DGTEST	Macht einen ganz tollen Tab.	HGP	SMTAB_(EP), b
MALWAS		HGP, Test	
HDRGE		HGP, Project	
WURAM		HGP, Test	
PAMIU		HGP	

Abbildung 10: Anzeige und Filterung der Module nach Tags



Modules

Environment: ENTW
 Library: Select
 Catalog user: Select
 Catalog date: Select
 Source user: Select
 Source date: Select
 Reset Filter

Name	Library	Source	Catalog	Source-User	Source-Date	Catalog-User	Catalog-Date
SMTAB	UTILITY			MACKE	01.04.2010 13:00	MACKE	01.04.2010 13:00
DGTAB	CON			GRASHORN	01.04.2010 13:00	GRASHORN	01.04.2010 13:00
DGTEST	SUP			GRASHORN	05.04.2010 13:00	GRASHORN	05.04.2010 13:00
OHNETAG	CON			GRASHORN	05.04.2010 13:00	GRASHORN	01.04.2010 15:12
OHNEWIKI	CON			GRASHORN	05.04.2010 13:00	MACKE	01.04.2010 15:12

Abbildung 11: Liste der Module mit Filtermöglichkeiten

A.12 Entwicklerdokumentation

lib-model

[class tree: lib-model] [index: lib-model] [all elements]

Packages:
[lib-model](#)

Files:
[Naturalmodulename.php](#)

Classes:
[Naturalmodulename](#)

Class: Naturalmodulename

Source Location: /Naturalmodulename.php

Class Overview

```
BaseNaturalmodulename
|
--Naturalmodulename
```

Subclass for representing a row from the 'NaturalModulename' table.

Methods

- [__construct](#)
- [getNaturalTags](#)
- [getNaturalWikis](#)
- [loadNaturalModuleInformation](#)
- [__toString](#)

Class Details

[line 10]
Subclass for representing a row from the 'NaturalModulename' table.

Adds some business logic to the base.

[\[Top \]](#)

Class Methods

constructor `__construct` [line 56]

```
Naturalmodulename __construct ( )
```

Initializes internal state of Naturalmodulename object.

Tags:

see: parent::__construct()
access: public

[\[Top \]](#)

method `getNaturalTags` [line 68]

```
array getNaturalTags ( )
```

Returns an Array of NaturalTags connected with this Modulename.

Tags:

return: Array of NaturalTags
access: public

[\[Top \]](#)

method getNaturalWikis [line 83]

```
array getNaturalWikis( )
```

Returns an Array of NaturalWikis connected with this Modulename.

Tags:

return: Array of NaturalWikis
access: public

[\[Top \]](#)

method loadNaturalModuleInformation [line 17]

```
ComparedNaturalModuleInformation  
loadNaturalModuleInformation( )
```

Gets the ComparedNaturalModuleInformation for this NaturalModulename.

Tags:

access: public

[\[Top \]](#)

method __toString [line 47]

```
string __toString( )
```

Returns the name of this NaturalModulename.

Tags:

access: public

[\[Top \]](#)

Documentation generated on Thu, 22 Apr 2010 08:14:01 +0200 by [phpDocumentor 1.4.2](#)

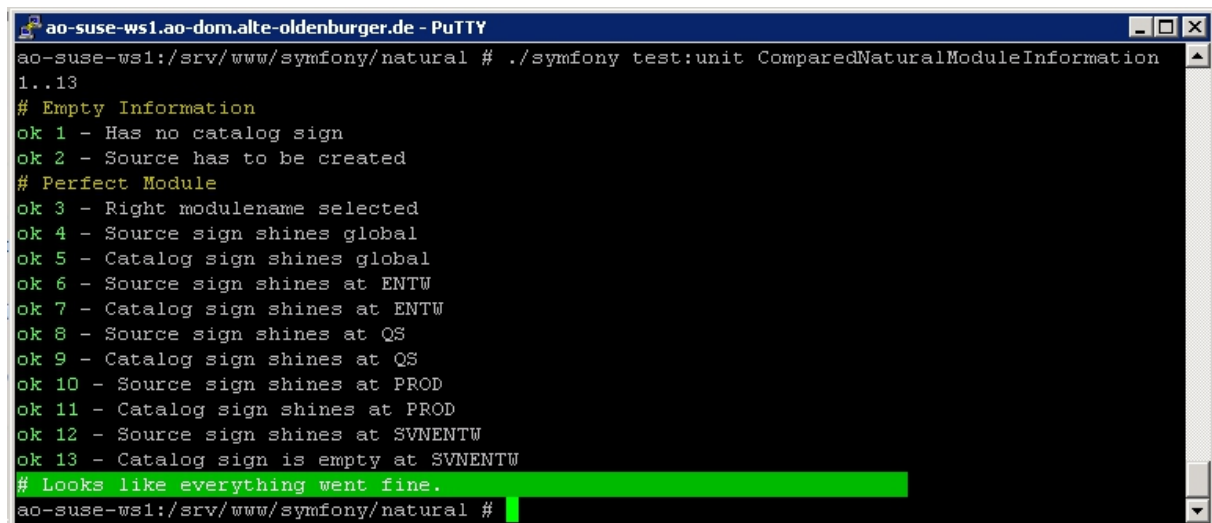
A.13 Testfall und sein Aufruf auf der Konsole

```

1 <?php
2 include(dirname(__FILE__).'../bootstrap/Propel.php');
3
4 $t = new lime_test(13);
5
6 $t->comment('Empty Information');
7 $emptyComparedInformation = new ComparedNaturalModuleInformation(array());
8 $t->is($emptyComparedInformation->getCatalogSign(), ComparedNaturalModuleInformation::
    EMPTY_SIGN, 'Has no catalog sign');
9 $t->is($emptyComparedInformation->getSourceSign(), ComparedNaturalModuleInformation::
    SIGN_CREATE, 'Source has to be created');
10
11 $t->comment('Perfect Module');
12 $criteria = new Criteria();
13 $criteria->add(NaturalmodulePeer::NAME, 'SMTAB');
14 $moduleName = NaturalmodulePeer::doSelectOne($criteria);
15 $t->is($moduleName->getName(), 'SMTAB', 'Right module name selected');
16 $comparedInformation = $moduleName->loadNaturalModuleInformation();
17 $t->is($comparedInformation->getSourceSign(), ComparedNaturalModuleInformation::SIGN_OK, 'Source
    sign shines global');
18 $t->is($comparedInformation->getCatalogSign(), ComparedNaturalModuleInformation::SIGN_OK, 'Catalog
    sign shines global');
19 $infos = $comparedInformation->getNaturalModuleInformations();
20 foreach($infos as $info)
21 {
22     $env = $info->getEnvironmentName();
23     $t->is($info->getSourceSign(), ComparedNaturalModuleInformation::SIGN_OK, 'Source sign shines at ' .
        $env);
24     if($env != 'SVNENTW')
25     {
26         $t->is($info->getCatalogSign(), ComparedNaturalModuleInformation::SIGN_OK, 'Catalog sign shines at
            ' . $info->getEnvironmentName());
27     }
28     else
29     {
30         $t->is($info->getCatalogSign(), ComparedNaturalModuleInformation::EMPTY_SIGN, 'Catalog sign is
            empty at ' . $info->getEnvironmentName());
31     }
32 }
33 ?>

```

Listing 1: Testfall in PHP



```
ao-suse-ws1.ao-dom.alte-oldenburger.de - PuTTY
ao-suse-ws1:/srv/www/symfony/natural # ./symfony test:unit ComparedNaturalModuleInformation
1..13
# Empty Information
ok 1 - Has no catalog sign
ok 2 - Source has to be created
# Perfect Module
ok 3 - Right modulename selected
ok 4 - Source sign shines global
ok 5 - Catalog sign shines global
ok 6 - Source sign shines at ENTW
ok 7 - Catalog sign shines at ENTW
ok 8 - Source sign shines at QS
ok 9 - Catalog sign shines at QS
ok 10 - Source sign shines at PROD
ok 11 - Catalog sign shines at PROD
ok 12 - Source sign shines at SVNENTW
ok 13 - Catalog sign is empty at SVNENTW
# Looks like everything went fine.
ao-suse-ws1:/srv/www/symfony/natural #
```

Abbildung 12: Aufruf des Testfalls auf der Konsole

A.14 Klasse: ComparedNaturalModuleInformation

Kommentare und simple Getter/Setter werden nicht angezeigt.

```
1 <?php
2 class ComparedNaturalModuleInformation
3 {
4     const EMPTY_SIGN = 0;
5     const SIGN_OK = 1;
6     const SIGN_NEXT_STEP = 2;
7     const SIGN_CREATE = 3;
8     const SIGN_CREATE_AND_NEXT_STEP = 4;
9     const SIGN_ERROR = 5;
10
11     private $naturalModuleInformations = array();
12
13     public static function environments()
14     {
15         return array("ENTW", "SVNENTW", "QS", "PROD");
16     }
17
18     public static function signOrder()
19     {
20         return array(self::SIGN_ERROR, self::SIGN_NEXT_STEP, self::
21             SIGN_CREATE_AND_NEXT_STEP, self::SIGN_CREATE, self::SIGN_OK);
22     }
23
24     public function __construct(array $naturalInformations)
25     {
26         $this->allocateModulesToEnvironments($naturalInformations);
27         $this->allocateEmptyModulesToMissingEnvironments();
28     }
29 }
```



```

27     $this->determineSourceSignsForAllEnvironments();
28 }
29
30 private function allocateModulesToEnvironments(array $naturalInformations)
31 {
32     foreach ($naturalInformations as $naturalInformation)
33     {
34         $env = $naturalInformation->getEnvironmentName();
35         if (in_array($env, self::environments()))
36         {
37             $this->naturalModuleInformations[array_search($env, self::environments())] = $naturalInformation;
38         }
39     }
40 }
41
42 private function allocateEmptyModulesToMissingEnvironments()
43 {
44     if (array_key_exists(0, $this->naturalModuleInformations))
45     {
46         $this->naturalModuleInformations[0]->setSourceSign(self::SIGN_OK);
47     }
48
49     for ($i = 0; $i < count(self::environments()); $i++)
50     {
51         if (!array_key_exists($i, $this->naturalModuleInformations))
52         {
53             $environments = self::environments();
54             $this->naturalModuleInformations[$i] = new EmptyNaturalModuleInformation($environments[$i]);
55             $this->naturalModuleInformations[$i]->setSourceSign(self::SIGN_CREATE);
56         }
57     }
58 }
59
60 public function determineSourceSignsForAllEnvironments()
61 {
62     for ($i = 1; $i < count(self::environments()); $i++)
63     {
64         $currentInformation = $this->naturalModuleInformations[$i];
65         $previousInformation = $this->naturalModuleInformations[$i - 1];
66         if ($currentInformation->getSourceSign() <> self::SIGN_CREATE)
67         {
68             if ($previousInformation->getSourceSign() <> self::SIGN_CREATE)
69             {
70                 if ($currentInformation->getHash() <> $previousInformation->getHash())
71                 {
72                     if ($currentInformation->getSourceDate('YmdHis') > $previousInformation->getSourceDate('
                        YmdHis'))
73                     {
74                         $currentInformation->setSourceSign(self::SIGN_ERROR);
75                     }

```

```

76         else
77         {
78             $currentInformation->setSourceSign(self::SIGN_NEXT_STEP);
79         }
80     }
81     else
82     {
83         $currentInformation->setSourceSign(self::SIGN_OK);
84     }
85 }
86 else
87 {
88     $currentInformation->setSourceSign(self::SIGN_ERROR);
89 }
90 }
91 elseif ($previousInformation->getSourceSign() <> self::SIGN_CREATE && $previousInformation->
    getSourceSign() <> self::SIGN_CREATE_AND_NEXT_STEP)
92 {
93     $currentInformation->setSourceSign(self::SIGN_CREATE_AND_NEXT_STEP);
94 }
95 }
96 }
97
98 private function containsSourceSign($sign)
99 {
100     foreach($this->naturalModuleInformations as $information)
101     {
102         if ($information->getSourceSign() == $sign)
103         {
104             return true;
105         }
106     }
107     return false ;
108 }
109
110 private function containsCatalogSign($sign)
111 {
112     foreach($this->naturalModuleInformations as $information)
113     {
114         if ($information->getCatalogSign() == $sign)
115         {
116             return true;
117         }
118     }
119     return false ;
120 }
121 }
122 ?>

```

Listing 2: Klasse: ComparedNaturalModuleInformation

A.15 Klassendiagramm

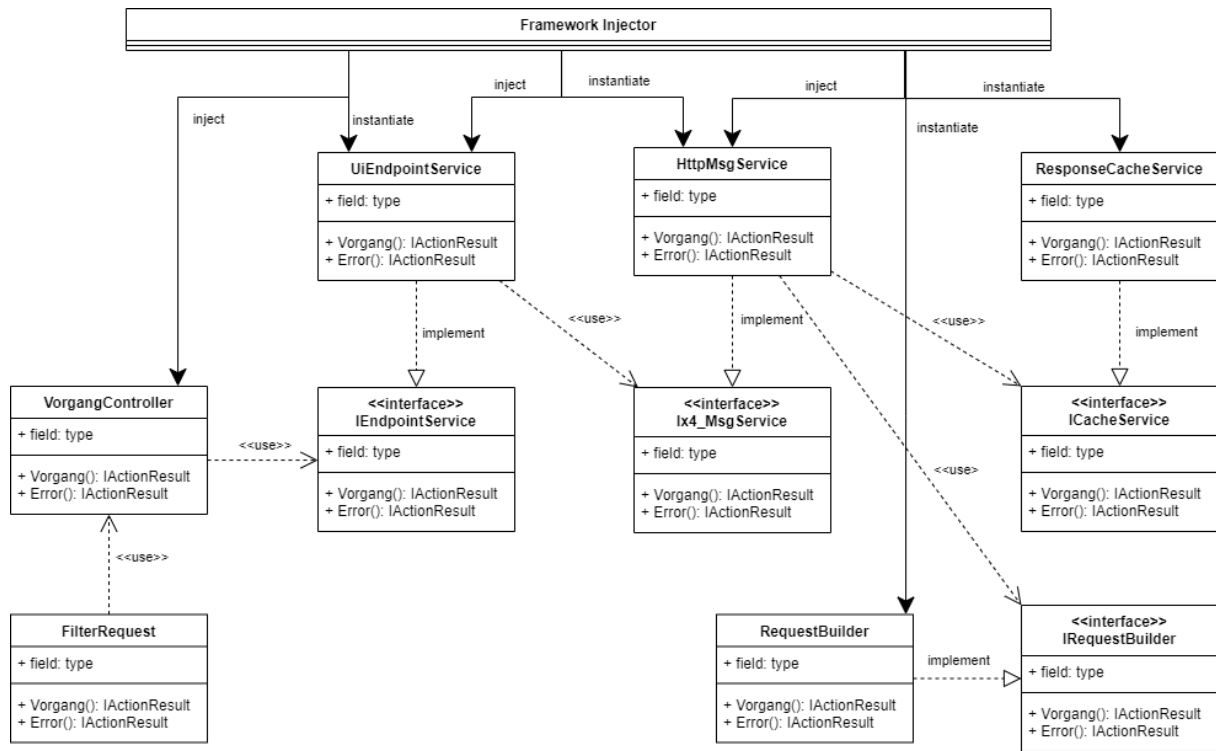







Abbildung 13: Klassendiagramm

A.16 Benutzerdokumentation

Ausschnitt aus der Benutzerdokumentation:

Symbol	Bedeutung global	Bedeutung einzeln
	Alle Module weisen den gleichen Stand auf.	Das Modul ist auf dem gleichen Stand wie das Modul auf der vorherigen Umgebung.
	Es existieren keine Module (fachlich nicht möglich).	Weder auf der aktuellen noch auf der vorherigen Umgebung sind Module angelegt. Es kann also auch nichts übertragen werden.
	Ein Modul muss durch das Übertragen von der vorherigen Umgebung erstellt werden.	Das Modul der vorherigen Umgebung kann übertragen werden, auf dieser Umgebung ist noch kein Modul vorhanden.
	Auf einer vorherigen Umgebung gibt es ein Modul, welches übertragen werden kann, um das nächste zu aktualisieren.	Das Modul der vorherigen Umgebung kann übertragen werden um dieses zu aktualisieren.
	Ein Modul auf einer Umgebung wurde entgegen des Entwicklungsprozesses gespeichert.	Das aktuelle Modul ist neuer als das Modul auf der vorherigen Umgebung oder die vorherige Umgebung wurde übersprungen.