

**Abschlussprüfung Sommer 2019**

Fachinformatiker Anwendungsentwicklung

Prozessorientierter Bericht

**Realisierung einer Anwendung zur automatischen Aktualisierung von Partnerlayouts**

Abgabetermin: 25.04.2019

**Prüfling:**Jerome Trope  
Untere Burg 41  
53881 Euskirchen

**Prüflingsnummer: …**

**Ausbildungsbetrieb:**TravelTrex GmbH  
Bonner Str. 484-486  
50968 Köln

# Vorwort

Im Vorliegenden Dokument wird die Durchführung der Realisierung des Projektes "automatische Aktualisierung von Partnerlayouts" im Rahmen des Abschlussprojektes prozessorientiert Beschrieben.

Sofern nicht anders gekennzeichnet, sind alle Anlagen/Dokumente und Code von mir angefertigt.

Inhalt

[Vorwort 2](#_Toc6918620)

[Änderungen zum Projektantrag 4](#_Toc6918621)

[Projektauftrag 4](#_Toc6918622)

[Projektumfeld 4](#_Toc6918623)

[Organisatorisches Umfeld 4](#_Toc6918624)

[Technisches Umfeld 4](#_Toc6918625)

[Ausgangssituation 5](#_Toc6918626)

[Projektziel 5](#_Toc6918627)

[Projektplanung 6](#_Toc6918628)

[Projektstruktur und Zeitplanung 6](#_Toc6918629)

[Qualitätsplanung 6](#_Toc6918630)

[Projektqualität 6](#_Toc6918631)

[Produktqualität 7](#_Toc6918632)

[Soll-Ist-Vergleich 7](#_Toc6918633)

[Projektdurchführung 7](#_Toc6918634)

[Analysephase 7](#_Toc6918635)

[Entwurfsphase 8](#_Toc6918636)

[Implementierungsphase 9](#_Toc6918637)

[Testen 11](#_Toc6918638)

[Soll-Ist-Vergleich 11](#_Toc6918639)

[Projektabschluss 11](#_Toc6918640)

[Abschließender Soll-Ist Vergleich 11](#_Toc6918641)

[Qualitätsbericht 11](#_Toc6918642)

[Abnahme durch den Auftraggeber 12](#_Toc6918643)

[Projektergebnis 12](#_Toc6918644)

[Reflektion 12](#_Toc6918645)

[Glossar 12](#_Toc6918646)

[Anlagenverzeichnis 12](#_Toc6918647)

[Abbildungsverzeichnis 12](#_Toc6918648)

Anlagenverzeichnis:

# Änderungen zum Projektantrag

Aufgrund des Fehlens der Planungsphase, wurde die Durchführungsphase um 5 Std gekürzt und dafür eine Planungsphase eingebracht.

# Projektauftrag

## Projektumfeld

### Organisatorisches Umfeld

Das Projekt wird als Abschlussprojekt der Ausbildung zum Fachinformatiker der Fachrichtung Anwendungsentwicklung in den Bürogebäuden der TravelTrex GmbH durchgeführt. Das Unternehmen beschäftigt zurzeit ca. 100 Mitarbeiter.

Projektbetreuer ist Herr Michael Kaesberg; Auftraggeber ist Herr Roman Konz, Teamleitung Website.

### Technisches Umfeld

Die Serverumgebung besteht im Wesentlichen aus Linux-Servern mit Ubuntu 18.04, PHP 7.2, ausgefuhrt auf dem Webserver Apache 2.2. Als Datenbank kommt die schemalose, dokumentenorientierte Datenbank MongoDB 2.6 zum Einsatz.

Für das Frontend wird Angular 6 in Kombination mit TypeScript und SCSS verwendet.

Zur Entwicklung steht jedem Entwickler JetBrains PhpStorm in der Version 2018.3 auf einem MacBook mit macOS Morawe zur Verfügung.

Der Export (bzw. Import in die Webseite) erfolgt über eine Schnittstelle, die in einem folgenden, Projekt erstellt wird.

## Ausgangssituation

Die TravelTrex GmbH ist ein Reiseveranstalter mit Sitz in Köln. Unter der Marke "SnowTrex" werden weltweit per Internet und Telefon Winterreisen vertrieben.

Zusätzlich zu der eigenen Internetseite werden die Reisen auch über Partneragenturen vermarktet. Dazu gibt es verschiedene Möglichkeiten:

IFrame Einbindung (im Projekt irrelevant) Subdomain Einbindung

Bei einer Subdomain Einbindung ist die SnowTrex Webseite im Farbton der Partnerwebseite gehalten, zusätzlich werden Seitenelemente vom Partner geladen. Die Elemente aus fremden Quellen sind in den Theme Einstellungen in der Datenbank gespeichert.

Aktuell werden die Aktualisierungen mit einem veraltetem Tool durchgeführt, welches nicht alle heutigen Anforderungen erfüllt. Daher müssen die Inhalte immer öfters händisch eingespielt werden. Die meisten Komponenten sind extrem veraltet, so dass es keine Möglichkeit zum Aktualisieren gibt.

Dies ist sehr zeitaufwendig, da das HTML geladen, manipuliert, und dann in die Datenbank der jeweiligen Umgebung eingespielt werden muss. Erschwerend kommt hinzu, dass die Partner unterschiedliche Dateiformate bereitstellen. Daher müssen die Daten erst in ein einheitliches Format gebracht werden. Danach muss der Partner umfangreich getestet werden, da durch die Aktualisierung nicht nur Darstellungsfehler in den aktualisierten Elementen entstehen können, sondern auch die anderen Inhalte von inkompatiblen Elementen gestört werden.

## Projektziel

Das Ziel des Projektes ist es, eine Software zu entwickeln, welche die alte Lösung ersetzt.

Es sollen Aktualisierungen geplant (terminiert) und manuell ("Knopfdruck"), über eine GUI eingespielt werden. Dieser Vorgang läuft vollautomatisiert ab.

Als erstes wird das HTML von den Links im Theme importiert. Im nächsten Schritt wird anhand selektierter/aktiver Regeln das HTML manipuliert. Während der Manipulation werden die verschiedenen Regeln angewendet um Elemente zu entfernen, selektieren oder ändern. Wenn dieser Schritt erfolgreich ist, wird eine neue Aktualisierung mit dem manipulierten HTML erstellt und in der Datenbank abgelegt und die Aktualisierung in die Webseite exportiert. Dazu wird eine Schnittstelle benutzt, die in einem Folgeprojekt angelegt wird und nicht Bestandteil dieses Projektes ist. Wenn im Theme Benachrichtigungen aktiviert sind, wird eine E-Mail mit den Details verschickt. Bei aktivierter Abnahme wird zuerst nur auf die Testumgebung exportiert, erst bei Bestätigung wird die Aktualisierung auf Live exportiert. Wenn ein Fehler auftritt, wird das Update abgebrochen. Falls der Export schon begonnen wurde, wird der vorherige Stand erneut auf Live exportiert. Der User wird per UI und per E-Mail Benachrichtigt.

In der Stammdatenverwaltung können Themes und Regeln bearbeitet/hinzugefügt/gelöscht werden. Jedes Theme hat mehrere Felder, welchem je eine URL und mehrere Regeln zugeordnet ist. Des Weiteren werden dort die Aktualisierungen chronologisch angezeigt. In den Theme Einstellungen kann ein Zeitpunkt/Intervall bestimmt werden, an welchem das Theme automatisch aktualisiert wird.

Das Projekt wird in zwei Teile gegliedert (Frontend/Backend). Dadurch ist die UI weitgehend losgelöst vom Backend.

# Projektplanung

## Projektstruktur und Zeitplanung

Nach dem offiziellen Start des Projektes wurden als erstes die Arbeitspakete (Abb. ) definiert und mithilfe von Draw.IO grafisch festgehalten. Darauf aufbauend wurde im nächsten Schritt eine umfangreiche Zeitplanung der Arbeitspakete mittels Toms Planner erstellt. (Abb. ) Das Gantt Diagramm deckt alle Phasen ab. Zur Gliederung wurden die einzelnen Projektphasen benutzt.

## Qualitätsplanung

Im Qualitätsplan wurde zwischen Produkt- und Prozessqualität unterschieden. (Abb. )

### Projektqualität

Zu Beginn der Projektplanungsphase wurde das Vorgehensmodell ausgewählt. Die Wahl fiel auf ein modifiziertes, erweitertes Wasserfallmodel nach Boehm. (Abb. ) Im Vergleich zum einfachen Wasserfallmodell bietet dieses die Möglichkeit, in vorherige Phasen zurückzuspringen. So können Fehler frühzeitig korrigiert werden oder Änderungen der Anforderungen umgesetzt werden. Dadurch wird außerdem die Kontrollierbarkeit des Fortschritts gewährleistet. Phasenrücksprünge, welche auf das Ändern von Anforderungen zurück zu führen waren, wurden nicht erwartet, da dem Projekt konkret definierte Anforderungen im Pflichtenheft/ RFC zugrunde liegen.

Ergänzt wurde das Modell durch Phasen des Projektmanagements und Qualitätssicherung.

Durch die umfangreiche Zeitplanung und den Meilensteinplan ist sichergestellt, dass auf Probleme frühzeitig reagiert werden kann.

### Produktqualität

Um eine hohe Produktqualität zu gewährleisten, wurden umfangreiche Unit-Tests während der Implementierung angelegt.

Zur Sicherung und Versionierung des Quellcodes wurde mit dem Versionierungssystem Git gearbeitet. Der Gesamte Quellcode wurde in einem Git-Repository gesichert und automatisch versioniert. So wurde ein Schutz vor Verlust und eine Historie realisiert. Der Code hält die Code-Guidelines des Unternehmens ein. Dies wird mit dem Einsatz von Lintern sichergestellt.

Die Testfälle wurden in einem Testfallkatalog zusammengeführt und bei Durchführung dokumentiert.

## Soll-Ist-Vergleich

Da im Pflichtenheft/RFC schon detaillierte Testfälle beschrieben waren, konnten bei der Erstellung des Testfallkatalogs Einsparungen erzielt werden.

|  |  |  |
| --- | --- | --- |
|  | Soll | Ist |
| Arbeitspakete definieren | 1 | 1 |
| Projektstrukturplan | 1 | 1 |
| Projektablaufplan | 1 | 1 |
| Qualitaetsplan | 1 | 0,5 |
| Meilensteinplan | 1 | 1 |

# Projektdurchführung

## Analysephase

Aus dem Pflichtenheft/RFC koennen die einzelnen Testfaelle und Szenarien entnommen werden. Diese werden im Use-Case Diagramm (Abb. ) grafisch dargestellt. Aus dem Use-Case Diagramm hinaus, wurde analysiert, welche Stammdaten (Tbl. ) benötigt werden und welche einzelnen Komponenten (Tbl. ) die Software beinhaltet.

## Entwurfsphase

Bei dem Entwurf wurde dem Top-Down Prinzip gefolgt, da bei der Analyse die einzelnen Komponenten definiert wurden. Aufbauend darauf, wurde im Design objektorientiert Vorgegangen. Diese Wahl wurde durch das Backend-Framework Symfony unterstützt. Symfony arbeitet zudem mit MVC-Pattern. Dadurch ließen sich die Daten zu einzelnen Objekten bzw. Klassen(Abb. ) unterteilen. Die Trennung von View und Logik erleichtert die Integration des Angular Frontends.

Nach der Komponentendefinierung wurden die benötigten Daten definiert und in Klassen zusammengetragen. Jede Komponente ist durch einen Controller abgebildet der eine eigene Schnittstelle bildet und das Single-Responsibility Prinzip einhält.

Das Frontend dient lediglich zur Anzeige der View Daten um die Geschaeftslogik an einer Stelle zu halten, um Zukunftigen Wartungsaufwand und Fehleranfaelligkeit zu verringern. Dynamische Frontends erhoehen die Benutzterfreundlichkeit, dafuer wird JavaScript Benutzt, da das Frontend nur zur Anzeige dient, wurde entschieden, einen Frontend Renderer einzusetzen. Angular bietet hier eine Moeglichkeit, die Konsistenz zwischen Backend und Frontend zu wahren. Durch das Single-Responsiblity Prinzip mithilfe des MVC Patterns kann dieselbe Struktur wie im Backend erzielt werden, dies sorgt fuer ein uebersichtlicheres Design des Gesamtsystems. Die Angular App kommuniziert mit dem Backend ueber die Datenstruktur JSON. Ziel dabei ist es auch, moeglichst wenig Logik im Frontend zu haben und nur die jeweilige Datenstruktur grafisch darzustellen und Benutzeraktionen zu ermöglichen.

Während der Schnittstellenplanung wurden zwei Typen definiert. Datenschnittstellen dienen zur Stammdatenverwaltung, diese besitzen jeweils eine new, edit, delete und show Methode. Die Kommunikation erfolgt mithilfe der Datenstruktur JSON auf Grundlage des jeweiligen Klassendiagramms. Dazu kommt die Aktion-Schnittstelle Update. Diese stellt den Kern Businesslogik da. Sie beinhaltet die Applikationslogik zum Updaten der Themes. Um den Ablauf darzustellen, wurde ein Aktivitaetsdiagramm (Abb. ) und ein Zustandsdiagramm (Abb. ) erstellt. Dazu kommen die Import und Export Schnittstellen. Um die Kommunikation mit Dritt-Systemen darzustellen wurde jeweils ein Sequenzdiagramm (Abb. Und Abb. ) angefertigt.

Um die Software in Zukunft erweiterbar für neue Anforderungen beim Import zu halten, wurde beim Update Prozess besonders auf leichte Integration von Erweiterungen geachtet. Um das zu gewährleisten wurde ein Adapterprinzip gewählt, dies wird im Komponentendiagramm (Abb. ) dargestellt.

Um die Komponenten zusammengeführt im Gesamtablauf darzustellen, wurde ein Aktivitaetsdiagramm (Abb. ) erstellt. Hier wird konkret gezeigt wie sich das System verhält, bei verschiedenen Benutzter eingaben und wie das System mithilfe von einzelnen Komponenten reagiert.

Ein Papierprototyp wurde nicht angefertigt, da sich das Design an den anderen TravelTrex Tools orientiert und das Standard Bootstrap Theme benutzt.

Zur Erstellung der Diagramme wurde PlanUML bzw. PlanText benutzt.

## Implementierungsphase

Die Implementierung des Entwurfs erfolgte in der Programmiersprache PHP 7.2 in der Entwicklungsumgebung PHPStorm.

Zuerst wurde über Composer ein neues Symfony Projekt erstellt. Dadurch wurden die Ordnerstruktur und das Basissystem eingerichtet. Danach wurden noch zusätzliche Pakete über Composer installiert. Dadurch wurde die Funktionalitaet des Frameworks um Formverarbeitung und Datenbankfunktionen mittels Doctrine erweitert.

Danach wurden zusätzliche Ordner für Importer, Manipulator, Events und Service angelegt.

Nun erfolgte die Konfiguration des Frameworks und der Pakete über die Konfigurationsdateien im Ordner config. Die Auslagerung der Konfiguration erlaubt eine leichte und schnelle Übertragung auf andere Umgebungen wie Live oder Prod zum Testen. Dies beinhaltet auch die Angabe vom Datenbank-Server.

Im nächsten Schritt wurden die Klassen aus den Diagrammen übertragen und zusätzliche Methoden wie getter und setter mithilfe von PHPStorm generiert. Nachdem die Klassen angelegt waren, wurde das Datenbankmapping mithilfe von PHP Annotations angelegt.

Daraufhin wurden einzelne Form Klassen angelegt, diese stellen da, wie die Daten in JSON definiert sein müssen, damit es valide Form Daten sind.

Als nächstes wurden die Controller im Ordner Controller angelegt. Die Methoden in den Controllern die per http erreichbar sein sollten, werden per PHP Annotation als Route gekennzeichnet. Symfony baut auf Dependency Injection auf. Dies ermöglicht eine leichte Auslagerung der Logik in Service und weitaus bessere Möglichkeiten zum Testen. Für die einzelnen Service wurde ein Test angelegt.

Die Stammdatencontroller nehmen den jeweiligen Request entgegen, validieren die Requestdaten gegen die Form, führen ggf. Datenbankoperationen aus und geben Rueckmeldung in Form von einem Response Objekt. Durch Symfony wird die Technische Logik in den Controllern minimiert. Die einzelnen Routenparameter werden entsprechend dem Datentypen automatisch gemappt. Dies erlaubt ein automatisches auflösen der Methodenparameter aus dem Request, dadurch wird der Controller schmaler übersichtlicher.

Der Manipulator Container ist ein Service, welcher als Container für die einzelnen Manipulatoren dient. Manipulatoren sind Service, welche das Manipulator Interface implementieren und per tag als Manipulator konfiguriert werden. Aufgrund des Typs entscheidet der Manipulator Container, welcher Manipulator benutzt werden muss und führt die Aktion zum Manipulieren des jeweiligen Manipulators aus.

Die einzelnen Importer und Exporter wurden als Service angelegt welche das Interface Importer oder Exporter implementieren. Zudem erweitern sie die jeweilige abstrakte Oberklasse (AbstractImpoter, AbstractExporter) wodurch ein automatisches registrieren als Event Subscriber erfolgt. Dadurch wird beim Import der entsprechende Importer anhand des Sourcetype ermitteln und bei Unterstützung aufgerufen. Zu Beginn des Projektes war noch nicht klar, ob es weitere Exporter benötigt, daher wurde die Möglichkeit gelassen, leicht weitere Exporter zu integrieren. Während der Entwicklungszeit wurde der alte Buchungsprozess abgestellt, daher wurde nur ein Standard Exporter implementiert und der Event für Benachrichtigungen benutzt.

Durch dieses Vorgehen wird die Erweiterbarkeit sichergestellt und eine Unabhängigkeit von der eingesetzten Technik der Partner gewährleistet.

Der Css Importer parst den Input Text in eine Tupel Liste, jeweils der Selektor und danach die Aktion. Als Basis dient das Symfony CSS Selektor Package. Es wurden die Grundlegendsten Methoden auf die des Packages gemappt und der Selektor übergeben. Ähnlich ist es auch bei dem PaulaScript Importer, allerdings wird hier der Input String anhand einer anderen Grammatik geparst, die Basis für die Grammatik bietet die JQuery Selektor Syntax.

Das Ergebnis von dem jeweiligen Controller wird per JSON Serializer zu JSON codiert und an den Browser ausgeliefert.

Für die Initialisierung der Angular App wurde das CLI benutzt. Nachdem die App Struktur generiert war, wurden die einzelnen Komponenten per CLI generiert.

Es wurde ein Service generiert, indem alle API Requests zum Backend als Methoden existieren, welche jeweils ein Objekt zurückgeben.

Danach wurden die HTML5 Formulare in den einzelnen Komponenten angelegt und mittels Angular Direktiven konnte dort View Logik verbaut werden, wodurch kaum Controller Logik zu implementieren war. Um die Entwicklungszeit zu verkürzen, wurde entschieden ein CSS Framework zu benutzten. Die Wahl fiel hierbei auf Bootstrap 4, da das Framework alle benötigten Komponenten besitzt und schon im Unternehmen eingesetzt wird.

## Testen

## Soll-Ist-Vergleich

Aufgrund Verzögerungen in der Implementierung hat sich die Implementierungsphase um 3 Stunden verlängert. Hier wurden 2 Stunden Puffer benötigt, da die Tests nur 0,5 Stunden dauerten und in der Planungsphase 0,5 Stunden gewonnen wurden.

|  |  |  |
| --- | --- | --- |
|  | Soll | Ist |
| Entwurfsphase | 4 | 4 |
| Implementierungsphase | 35 | 38 |
| Testen | 1 | 0,5 |

# Projektabschluss

## Abschließender Soll-Ist Vergleich

## Qualitätsbericht

Beim abschließenden Qualitätsbericht (Tbl. ) wurden die geplanten Qualitätsmaßnahmen mit den tatsächlich durchgeführten verglichen. Dafür wurde die im Qualitätsplan (Tbl. ) erstellte tabellarische Übersicht verwendet. Der Vergleich ergab, dass in diesem Projekt alle geplanten qualitätssichernden Maßnahmen vollständig umgesetzt wurden.

## Abnahme durch den Auftraggeber

Zum Abschluss des Projektes fand die Projektübergabe an den Auftraggeber statt. Die Ergebnisse des abschließenden Soll-Ist-Vergleiches und des abschließenden Qualitätsberichts wurden besprochen und von den beteiligten Personen zur Kenntnis genommen.

Die Anwenderdokumentation wurde erstellt und den beteiligten Personen präsentiert.

Nach Abnahme des Projekts wurde mit dem Verfassen des prozessorientierten Berichts begonnen. Dabei wurden die erstellten Dokumente einheitlich formatiert und mit Auszügen des Programmcodes als Anhang eingefügt. Der Bericht konnte innerhalb der geplanten Zeit fertiggestellt werden.

## Projektergebnis

Das Projekt wurde in der gegebenen Zeit erfolgreich umgesetzt. Das Wunschkriterium zeitgesteuerte Updates, wurde Vorbereitet und kann in einem Folgeprojekt leicht integriert werden.

## Reflektion

# Glossar

# Anlagenverzeichnis

# Abbildungsverzeichnis

[Abbildung 1 Vorgangsmodell 22](#_Toc6920318)

[Abbildung 2 Projektstrukturplan 23](#_Toc6920319)

[Abbildung 3 Zeitplan 23](#_Toc6920320)

[Abbildung 4 Preset Controller – create 24](#_Toc6920321)

[Abbildung 5 UpdateController - export 24](#_Toc6920322)

##### Projektantrag\\Mac\Home\Desktop\antrag_Trope_Jerome_5002317345_1-1 (verschoben).tiff\\Mac\Home\Desktop\antrag_Trope_Jerome_5002317345_1-2 (verschoben).tiff\\Mac\Home\Desktop\antrag_Trope_Jerome_5002317345_1-3 (verschoben).tiff\\Mac\Home\Desktop\antrag_Trope_Jerome_5002317345_1-4 (verschoben).tiff\\Mac\Home\Desktop\antrag_Trope_Jerome_5002317345_1-5 (verschoben).tiff\\Mac\Home\Desktop\antrag_Trope_Jerome_5002317345_1-6 (verschoben).tiff\\Mac\Home\Desktop\antrag_Trope_Jerome_5002317345_1-7 (verschoben).tiff

##### Projektgenehmigung

##### 

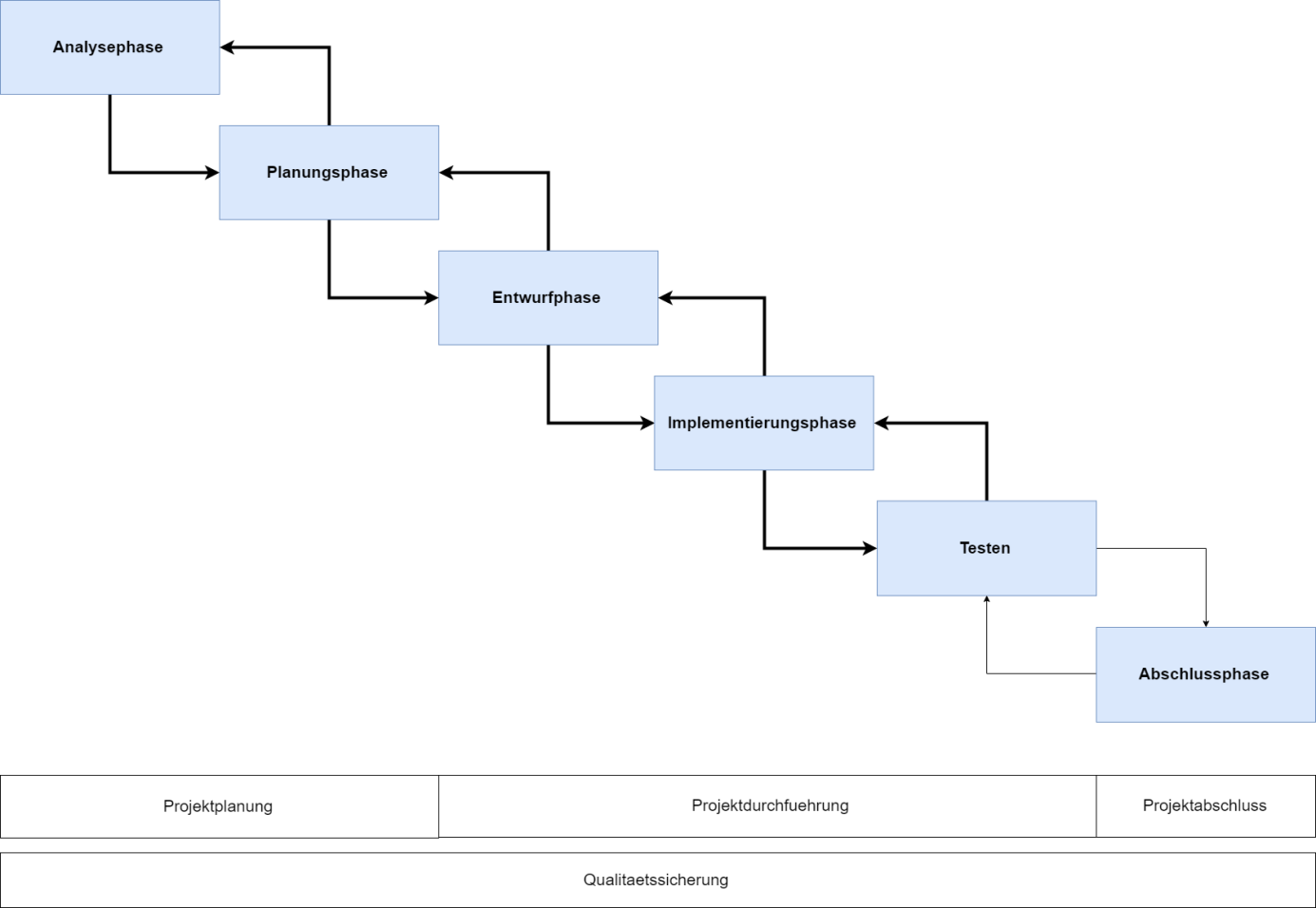


Abbildung Vorgangsmodell

|  |  |  |
| --- | --- | --- |
| Vorgangs-Nr. | Vorgangsname | Soll |
| 1 | Analysephase | 3 |
| 1.01 | Analyse Pflichtenheft / RFC | 3 |
| 2 | Planungsphase | 5 |
| 2.01 | Arbeitspakete definieren | 1 |
| 2.02 | Projektstrukturplan erstellen | 1 |
| 2.03 | Projektablaufplan erstellen | 1 |
| 2.04 | Meilensteinplan erstellen | 1 |
| 2.05 | Qualitätsplan erstellen | 1 |
| 3 | Durchführungsphase | 40 |
| 3.01 | Entwurfsphase | 4 |
| 3.01.01 | Komponenten definieren | 3 |
| 3.01.02 | Klassen definieren | 1 |
| 3.02 | Implementierungsphase | 35 |
| 3.02.01 | Implementierung | 34 |
| 3.02.02 | Review | 1 |
| 3.03.01 | Testen | 1 |
| 4 | Abschlussphase | 7 |
| 4.01 | Soll-Ist-Vergleich | 1 |
| 4.02 | Benutzerdokumentation erstellen | 5 |
| 4.03 | Abnahme | 1 |

Tabelle Vorgangsliste

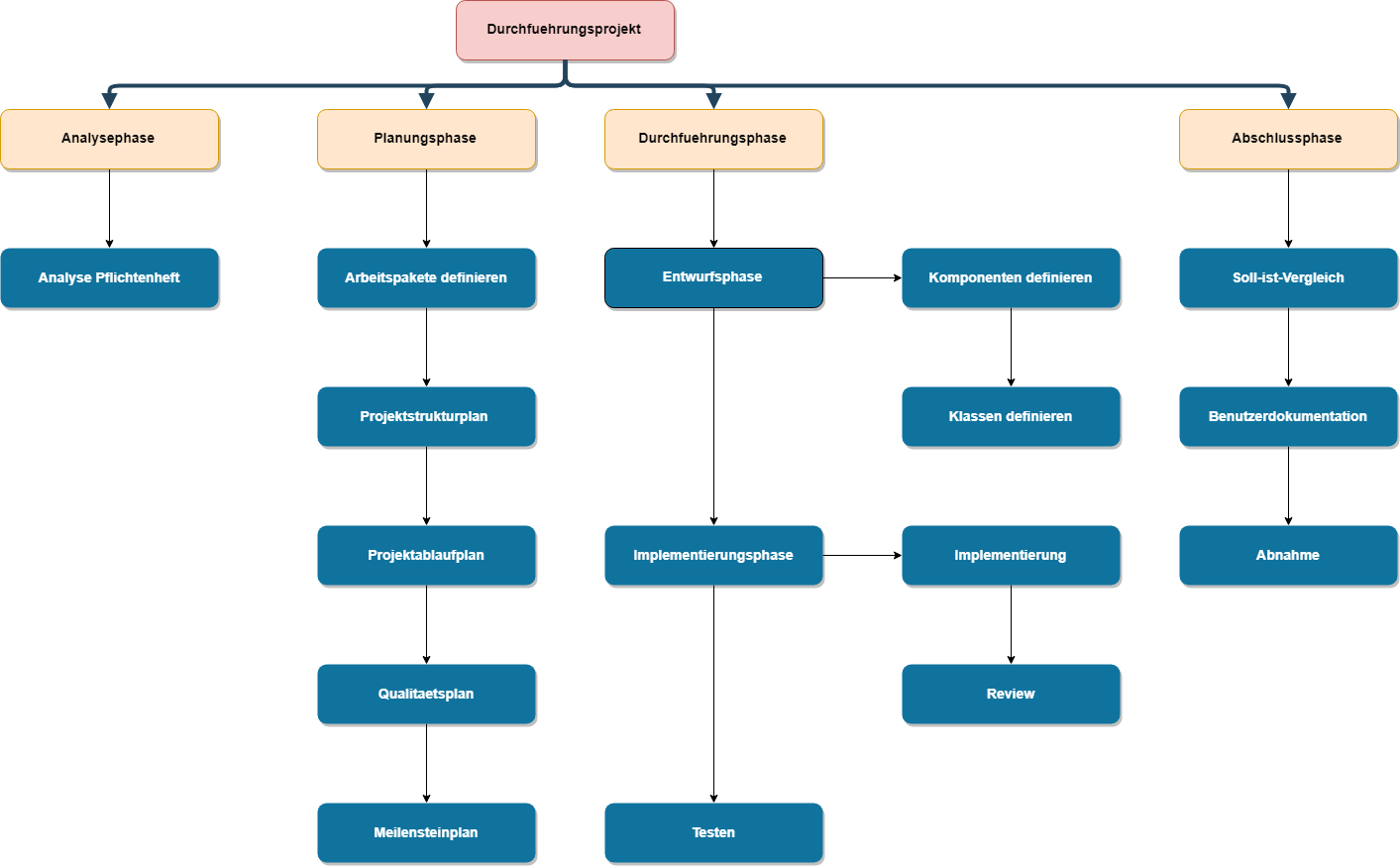


Abbildung Projektstrukturplan

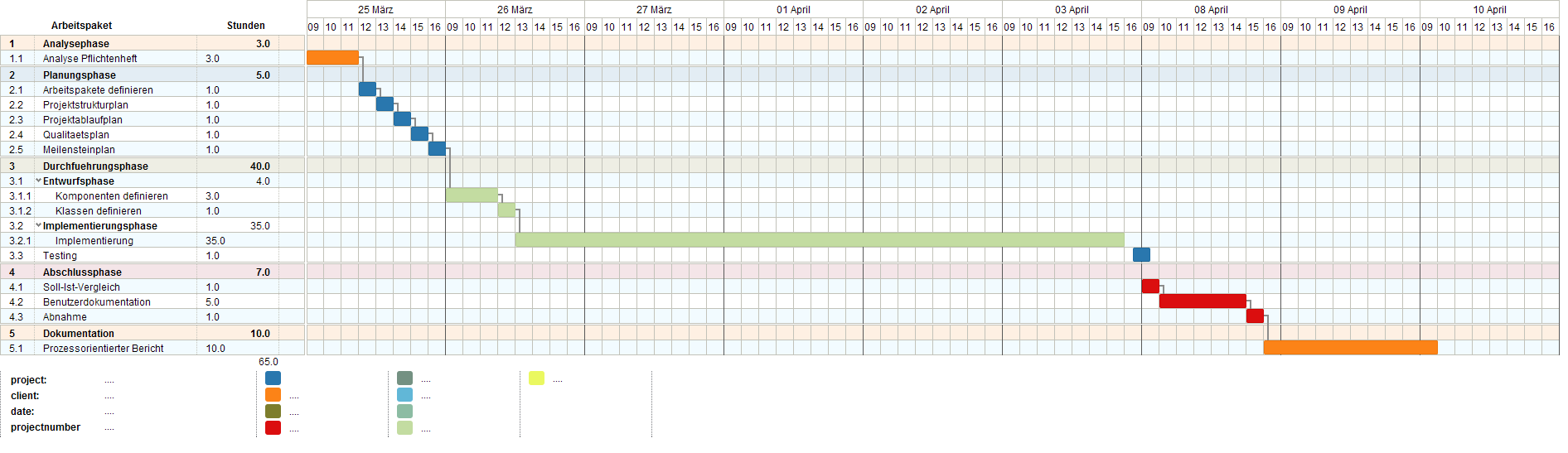


Abbildung Zeitplan

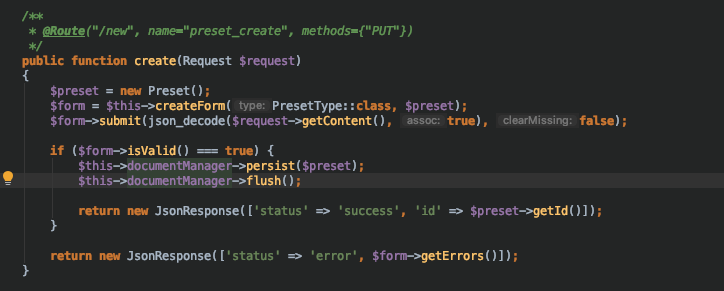


Abbildung Preset Controller – create



Abbildung UpdateController - export

##### Anwenderdokumentation