HOCHSCHULE LANDSHUT

FAKULTÄT INFORMATIK

Entwicklung eines webbasierten Demonstrators für das Tailoring hybrider Vorgehensmodelle im Projektmanagement.

B a c h e l o r a r b e i t

vorgelegt von
Lidia Linkewitsch
aus Landshut
Eingereicht: 12.06.2020

Prüfer: Prof. Dr.-Ing. H. Timinger
Zweitprüfer: Prof. Dr. A. Khelil

ERKLÄRUNG ZUR BACHELORARBEIT

(gemäß §11, Abs. (4) 3 APO)

Name, Vorname der

Studierenden: ..... **Lidia Linkewitsch**

Hochschule Landshut

Fakultät Informatik

Hiermit erkläre ich, dass ich die Arbeit selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

....

(Datum) (Unterschrift der/des Studierenden)

Zusammenfassung

Heute zeichnet sich ein erfolgreich durchgeführtes Projekt durch ein richtig ausgewähltes Vorgehensmodell aus. Am meisten profitieren Unternehmen, wenn dieses an die eigenen Rahmenbedingungen angepasst wird (vgl. [1]). Ein solch maßgeschneidertes und individuell erstelltes Vorgehensmodell zu konstruieren, ist eine komplexe und kostenintensive Aufgabe. Daher ist es sinnvoll, die Erstellung individueller Vorgehensmodelle zu automatisieren. Ziel dieser Arbeit ist die Entwicklung und Implementierung eines webbasierten Demonstrators für das Tailoring hybrider Vorgehensmodelle. Der Demonstrator beinhaltet alle agile, traditionelle und hybride Methoden des Projektmanagements. Auf Basis projektspezifischer Parameter wird ein individuelles agiles, traditionelles oder hybrides Vorgehensmodell abgeleitet. Im Laufe der Arbeit wird sowohl die technische Seite des Demonstrators als auch das Tailoring hybrider Vorgehensmodelle diskutiert.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation und Aufgabenstellung

Modernes Projektmanagement kennt traditionelle, agile und hybride Vorgehensmodelle, die in der Literatur gut dokumentiert sind (vgl. [2]). Traditionelle Vorgehensmodelle stammen aus der Mitte des 20. Jahrhunderts, als die Sicherheit von Projekten die zentrale Rolle im Projektmanagement einnahm. Sie sind bekannt als plangetriebene Vorgehensmodelle. Gleich am Anfang eines Projektes werden alle Projektphasen durchgeplant sowie Kosten und Risiken eingeschätzt. Solche Modelle spielen heute noch eine große Rolle und werden breit angewendet (vgl. [2], S.29–159). Ende des 20. Jahrhunderts kamen die agilen Ansätze ins Spiel. Diese Vorgehensmodelle fördern einen anderen Denkansatz als die traditionellen Varianten. Sie stellen den Kunden ins Zentrum, setzen auf qualifizierte und selbstorganisierte Teams und propagieren inkrementelle Projektphasen (vgl. [2], S. 161–239). Beide Welten der Vorgehensmodelle enthalten viele Methoden und Werkzeuge zum Planen, Koordinieren und Steuern eines Projektes. Um ein Projekt erfolgreich durchzuführen, sollte ein passendes Vorgehensmodell ausgewählt und an die Rahmenbedingungen des jeweiligen Unternehmens angepasst werden (vgl. [1]). Laut der Studie „The Benefits of Tailoring" (vgl. [1]) wurden 82 % aller erfolgreichen Projekte in Unternehmen durchgeführt, die ein an die eigenen Rahmenbedingungen adaptiertes Vorgehensmodell verwendeten. Mittlerweile gewinnen zunehmend hybride Modelle an Bedeutung (vgl. [3]), die eine weitere Herausforderung mit sich bringen. Sie vereinen das Beste aus traditionellen und agilen Vorgehensmodellen. Von beiden werden die Methoden und Werkzeuge ausgewählt, die zum unternehmens- und projektspezifischen Vorhaben am besten passen. Die Kombination der beiden Welten kann vielfältig sein: „Die Menge an möglichen Kombinationen von Vorgehensbestandteilen übersteigt das, was realistischer Wei-

se ohne Softwareunterstützung analysiert und optimiert werden kann", schreiben Seel und Timinger dazu in ihrem Werk (vgl. [4]). So entwickelt auch das Institut für Projektmanagement und Informationsmodellierung (vgl. [5]) der Hochschule Landshut Adaptionsmechanismen für hybrides und adaptives Projektmanagement, die die Auswahl automatisieren sollen. Deshalb soll im Rahmen dieser Arbeit ein webbasierter Demonstrator für das Tailoring hybrider Vorgehensmodelle entwickelt und implementiert werden. Er soll alle möglichen Methoden und Vorgehensmodelle beinhalten und auf Basis der konfigurierten Adaptionsparameter ein individuelles Vorgehensmodell ableiten. Die Adaptionsparameter (näher erläutert in Kapitel 2) repräsentieren unternehmens- und projektspezifische Daten, die vom Anwender konfiguriert werden. Sie dienen auch als Eingangsparameter für den Algorithmus, der das Tailoring hybrider Vorgehensmodelle vornimmt. In dieser Arbeit wird der Algorithmus zunächst zwischen agilen und traditionellen Vorgehensmodellen unterscheiden. Für die Darstellung des individuellen Vorgehensmodells wird der Ordnungsrahmen für adaptives hybrides Projektmanagement, HyProMM (vgl. [6]), verwendet. Weiterhin soll der Demonstrator responsiv sein, das heißt, er soll in der Lage sein, die Daten sowohl auf einem Desktop-Computer als auch auf mobilen Geräten darzustellen. Zudem soll der Demonstrator modular aufgebaut werden, damit er leicht verändert oder weiterentwickelt werden kann.

1.2 Forschungsmethodik

Die Bearbeitung dieser Arbeit folgt der Problemlösung nach Haberfellner (vgl. [7]). Zunächst wird in diesem Kapitel der Problemlösungsmechanismus dargestellt und es werden die Zusammenhänge zur vorliegenden Arbeit hergestellt. Als Erstes lässt sich das Prinzip ‚Vom Groben zum Detail' anwenden. Hier wird von der Idee des Projektes schrittweise ins Detail gegangen. Somit unterteilt sich die Entwicklung des Demonstrators in mehrere Studien. Die Vorstudie leitet sich aus der in Kapitel 1.1 dargestellten Motivation und Aufgabenstellung ab. Die Hauptstudie befasst sich mit der Architektur des webbasierten Demonstrators. Die Detailstudien beschäftigen sich dann mit den unternehmens- und projektspezifischen Daten und deren Repräsentation sowie mit dem Algorithmus zur Auswahl des Vorgehensmodells. Jede Studie wird nach dem Phasenmodell, abgeleitet vom Problemlösungszyklus, detailliert abgearbeitet. Das Phasenmodell besteht aus fünf Schritten:

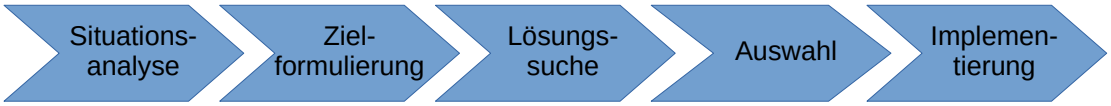

Abbildung 1.1: Phasen des Problemlösungszyklus (in Anlehnung an [2], S. 248)

Die Haupt- und Detailstudien analysieren zunächst die Situation bezüglich der Herausforderungen und Probleme. Weiterhin kristallisieren sich mit der Zielformulierung die Anforderungen an den Demonstrator heraus. Mit der Lösungssuche werden mögliche Lösungen und Alternativen diskutiert und anschließend sinnvolle Varianten ausgewählt. Abschließend wird die Implementierung der ausgesuchten Lösungen dargestellt.

1.3 Aufbau der Arbeit

Die Arbeit ist in sechs Kapitel unterteilt. Im ersten Kapitel werden durch Motivation und Aufgabenstellung erste Einblicke in die Arbeit gegeben. Zudem wird hier der Problemlösungsmechanismus dargestellt, um aufzuzeigen, wie in der Arbeit geforscht wird. Das zweite Kapitel erläutert die wichtigsten Grundlagen über Adaptionsparameter und Ordnungsrahmen für das hybride Projektmanagement. In Kapitel drei wird in Abhängigkeit von Anforderungen an den Demonstrator die Wahl der Programmiersprache und die Wahl des Frameworks diskutiert sowie die Architektur des Demonstrators dargestellt. Im Fokus des vierten Kapitels stehen die Details des Demonstrators. Schritt für Schritt werden in dem Kapitel die Repräsentation der Daten für den Benutzer diskutiert und die Entwicklungsdetails dargestellt. Zudem werden hier die Lösungsansätze zur Entwicklung des Algorithmus für das Tailoring agiler und traditioneller Vorgehensmodelle herausgearbeitet. Anschließend wird der Algorithmus präsentiert und erklärt. In Kapitel fünf wird die Arbeit zusammengefasst und es werden die Ergebnisse diskutiert. Ein Ausblick auf die weitere Entwicklung des Demonstrators beschließt die Arbeit.

2 Grundlagen

In diesem Kapitel werden die Grundlagen vorgestellt. Das Prinzip des Demonstrators besteht darin, auf Basis von Adaptionsparametern ein individuelles, adaptiertes Vorgehensmodell abzuleiten. Dazu werden Eingabegrößen benötigt. Dies sind die Adaptionsparameter, die in diesem Kapitel vorgestellt werden. Zudem wird ein Ordnungsrahmen für adaptives hybrides Projektmanagement vorgestellt, aus dem agile, traditionelle oder hybride Vorgehensmodelle abgeleitet werden können.

2.1 Adaptionsparameter

Um ein adaptiertes Vorgehensmodell abzuleiten, werden viele Kriterien untersucht. Jedes Unternehmen hat individuelle Vorgaben, um ein Projekt erfolgreich durchzuführen. Damit die Auswahl des adaptierten Vorgehensmodells automatisiert werden kann, müssen diese Kriterien herauskristallisiert und abstrahiert werden. Sie werden so ausgewählt, dass sie sowohl agilen als auch traditionellen Vorgehensmodellen genügen. Schon Boehm und Turner (vgl. [8]) zeigten, dass in Abhängigkeit von bestimmten Kriterien deutlich wird, ob eher agile oder traditionelle Vorgehensmodelle verwendet werden sollten. Ihr Modell berücksichtigt fünf Kriterien:

- Menschen,
- Stabilität der Anforderungen,
- Unternehmenskultur,
- Projektteamgröße und
- Gefährdungspotenzial.

Špundak (vgl. [9]) baut das Modell von Boehm und Turner weiter aus und fügt noch folgende Kriterien dazu:

- Komplexität des Projektgegenstandes,
- Teamverteilung,
- Ausrichtung der Organisation bei Projekten,
- Einbeziehung des Kunden und
- Dokumentationsanforderungen.

Basierend auf den oben vorgestellten Kriterien entwickelten Paukner, Seel und Timinger ein Modell (vgl. [10]), das aus acht Parametern besteht. Das Modell wird auch in dieser Arbeit verwendet und in Abbildung 2.1 dargestellt.

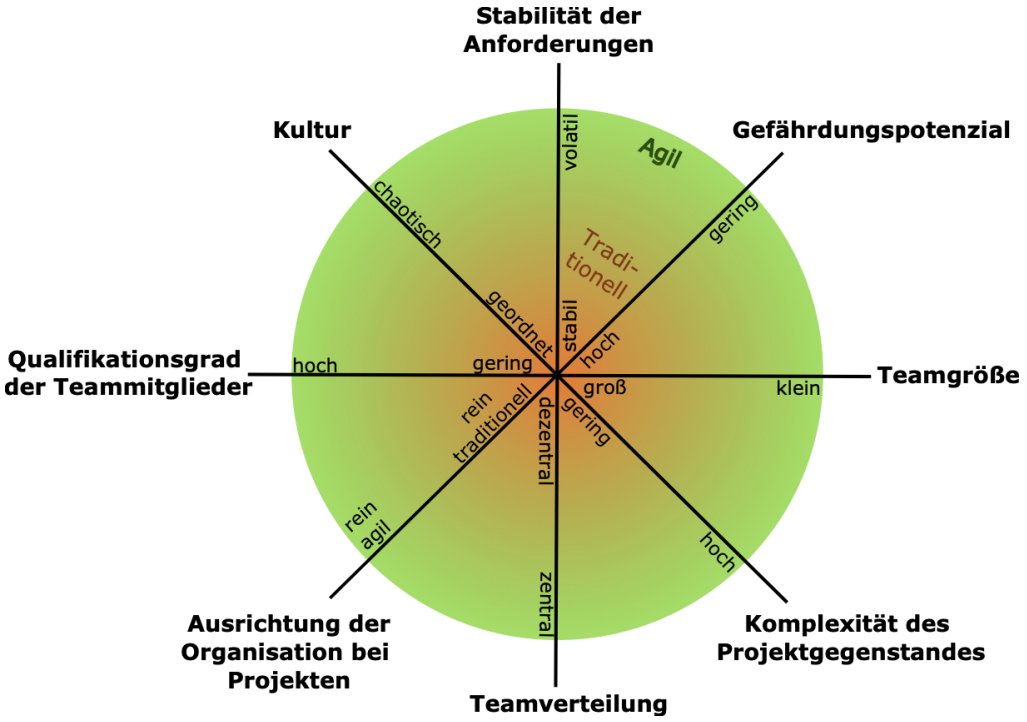

Abbildung 2.1: Adaptionsparameter (in Anlehnung an [10], S. 168)

Die acht vorgestellten Parameter werden bewertet und in Abhängigkeit von der Bewertung wird ein adaptiertes agiles, traditionelles oder hybrides Vorgehensmodell abgeleitet. Dafür erhält jeder Parameter eine agile oder traditionelle Ausprägung. Für die Wahl traditioneller Vorgehensmodelle sprechen die hohe Stabilität der Anforderungen, ein großes und verteiltes Team sowie ein geringer Qualifikationsgrad der Teammitglieder. Weiterhin werden für eine geordnete Unternehmenskultur mit

klaren Regeln eher traditionelle Vorgehensmodelle ausgewählt. Auch rein traditionell ausgerichtete Organisationen verwenden eher traditionelle Vorgehensmodelle. Dagegen sprechen für die Wahl der agilen Vorgehensmodelle volatile Anforderungen, ein kleines, zentrales und hoch qualifiziertes Team. Für eine chaotische Unternehmenskultur sind ebenfalls eher agile Vorgehensmodelle geeignet. (vgl. [10])

2.2 Ordnungsrahmen für adaptives hybrides Projektmanagement: HyProMM

Vorgehensmodelle beinhalten passende Methoden und Werkzeuge. Traditionelle Vorgehensmodelle wie das V- oder das Wasserfallmodell sowie agile Vorgehensmodelle wie Scrum oder Kanban werden in der Literatur entsprechend visualisiert (vgl. [2]). Allerdings schließt die Form der Darstellung eines Modells ein weiteres Modell aus. So entwickelten Timinger und Seel einen Ordnungsrahmen für adaptives hybrides Projektmanagement (vgl. [6]), siehe Abbildung 2.2. Mit Hilfe dieses Ordnungsrahmens können traditionelle, agile oder hybride Vorgehensmodelle abgeleitet werden, die zudem noch an die individuellen Rahmenbedingungen angepasst werden können.

Der Ordnungsrahmen wird in drei Bereiche unterteilt:

- Führung: Dieser Bereich wird als Dach in der Abbildung dargestellt.
- Projektlebenszyklus, der in der Mitte der Abbildung als Looping zu erkennen ist.
- Kontinuierliche Abläufe des Projektmanagements. Dieser Bereich ist unten in der Abbildung dargestellt.

Die Bereiche bestehen wiederum aus drei Sichten für

- Prozesse,
- Methoden und Werkzeuge sowie
- Rollen.

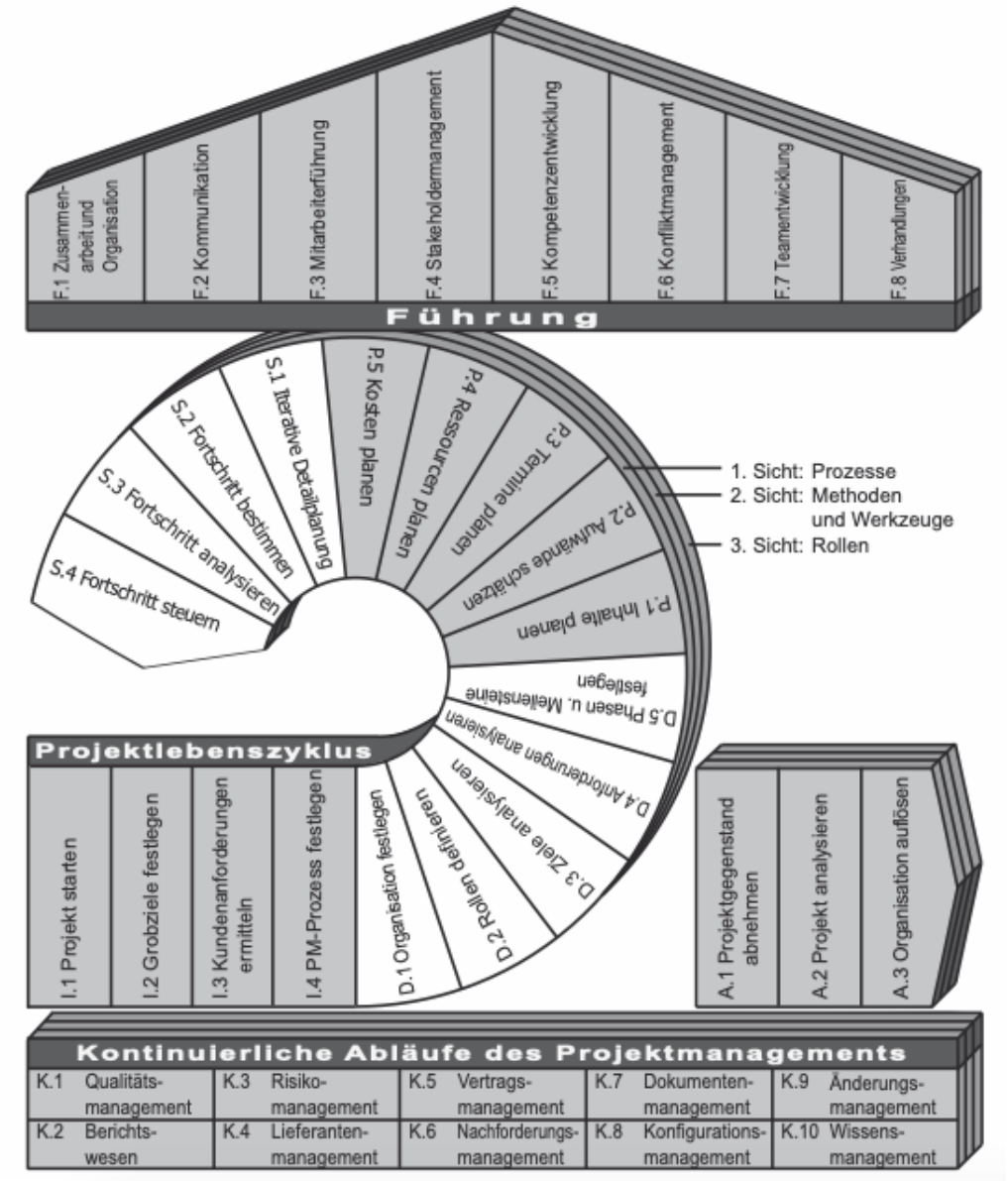

Abbildung 2.2: Ordnungsrahmen für adaptives hybrides Projektmanagement (vgl. [6])

Der Bereich Projektlebenszyklus wird in dieser speziellen Form dargestellt, um zu zeigen, dass dieser Bereich mehrmals durchlaufen werden kann, was wiederum für die agilen Vorgehensmodelle entscheidend ist. Der Projektlebenszyklus besteht aus fünf Phasen:

- I: Initialisierung, mit den Prozessen: I.1, I.2, I.3, I.4
- D: Definition, mit den Prozessen: D.1, D.2, D.3, D.4, D.5

- P: Planung, mit den Prozessen: P.1, P.2, P.3, P.4, P.5
- S: Steuerung, mit den Prozessen: S.1, S.2, S.3, S.4
- A: Abschluss, mit den Prozessen: A.1, A.2, A.3

Jedes Element beinhaltet agile und traditionelle Prozesse, Methoden und Werkzeuge sowie Rollen. Bei der Ableitung eines individuellen Vorgehensmodells filtern sich für das Unternehmen relevante Prozesse heraus. Damit die Prozesse realisiert werden können, werden entsprechende Methoden und Werkzeuge ausgewählt. Somit entsteht ein adaptiertes Vorgehensmodell (vgl. [6]).

In dieser Arbeit wird ein reduzierter Ordnungsrahmen verwendet, der aus dem mittleren Bereich des Projektlebenszyklus besteht.

3 Vorbereitung und Entwurf des Demonstrators

Webanwendungen unterteilen sich hauptsächlich in klassische serverseitige und clientseitige Anwendungen – letztere werden auch Single Page Applications (SPA) genannt. Die Anforderungen an den Demonstrator sprechen überwiegend für eine SPA, die mit modernen Frameworks wie Angular realisiert werden kann. Dieses Kapitel stellt gängige Technologien zur Entwicklung von clientseitigen Anwendungen vor. Es beschreibt weiterhin detaillierter das Angular Framework und dessen Werkzeuge. Zum Schluss wird ein Angular-Template erstellt sowie die Architektur des Demonstrators für die weitere Implementierung vorbereitet.

3.1 Anforderungen an die Webanwendung

Der Demonstrator soll als eine Webanwendung entwickelt werden und nach Möglichkeit in allen gängigen Webbrowsern wie Safari und Google Chrome laufen. Zudem soll der Demonstrator responsiv sein. Er muss seinen Inhalt und seine Funktionalität sowohl am Desktop-Computer als auch auf herkömmlichen mobilen Geräten vollständig abbilden können. Zu den relevanten Aspekten zählt auch die Modularisierung. Sie soll die Weiterentwicklung und Veränderung des Demonstrators in der Zukunft gewährleisten. Tabelle 3.1 fasst alle Anforderungen zusammen.

Kennziffer der Anforderung	Beschreibung der Anforderung
A1	Die Anwendung soll plattformunabhängig in jedem aktuellen Browser funktionieren.
A2	Die Anwendung soll ein Responsive Webdesign darstellen.
A3	Die Anwendung soll modular aufgebaut werden.
A4	Die Anwendung soll intuitiv zu bedienen sein.
A5	Der Nutzer der Webanwendung muss die Adaptionsparameter konfigurieren können.
A6	Der Nutzer muss das vorgeschlagene individuelle Vorgehensmodell inklusive Methoden und Werkzeuge ablesen können.

Tabelle 3.1: Anforderungen an den Demonstrator

3.2 Überblick über Webanwendungen

3.2.1 Klassische Webanwendungen

Eine klassische Webanwendung stellt ein Client-Server-Architektur dar. Dabei übernimmt der Browser die Rolle eines Clients. Der Server spielt hier eine zentrale Rolle. Er ist für die statischen Inhalte einer Seite verantwortlich. Im klassischen Sinne besteht eine Webanwendung aus verlinkten Dokumenten oder Seiten, die der Server eigenständig generiert. Jede Aktion des Benutzers sorgt für das Laden einer weiteren Seite vom Server. Bei interaktiven Anwendungen ist die Kommunikation zwischen Client und Server sehr intensiv, was wiederum zu einer Darstellungsverzögerung einzelner Seiten führen kann. Abbildung 3.1 demonstriert die Kommunikation zwischen Client und Server.

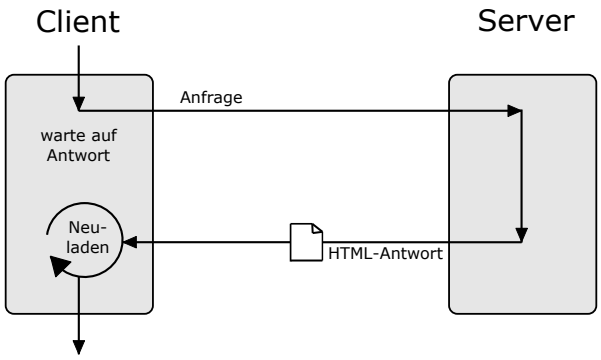

Abbildung 3.1: Kommunikation zwischen Client und Server bei klassischen Webanwendungen

3.2.2 Single Page Application

Bei der Single Page Application ist die Kopplung zwischen Client und Server nicht mehr so stark. Solche Anwendungen streben die clientseitige Verarbeitung der Daten an. Oft bestehen sie aus einem einzigen HTML-Dokument, dessen Inhalte dynamisch zur Laufzeit nachgeladen werden. Somit entfällt das ständige Laden neuer Seiten. Dies wirkt sich positiv auf Benutzerinteraktionen aus und lässt die Anwendung flüssiger erscheinen. Auch der Zustand der Anwendung bleibt im Client erhalten und entlastet somit den Server. Der Server hält lediglich die Nutzdaten bereit und kann sie dem Client durch REST[1] bereitstellen. Die SPAs sind für Desktop-Computer und auch für mobile Geräte gut geeignet. Als Nachteil ist zu erwähnen, dass solche Anwendungen schwer mit den Suchmaschinen durchsucht werden können, da Inhalte über JavaScript nachgeladen werden. Abbildung 3.2 stellt die Kommunikation zwischen Client und Server dar.

[1]REST: Representational State Transfer ist ein Architekturstil für Web Services, der auf Ressourcen per HTTP-Request-Methoden zugreift.

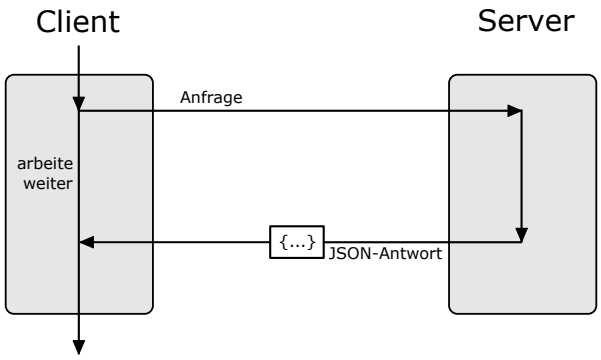

Abbildung 3.2: Kommunikation zwischen Client und Server bei Single Page Applications

3.2.3 Demonstrator als Single Page Application

Angesichts der Anforderungen eignet sich eine clientseitige Anwendung ideal für die Entwicklung des Demonstrator. Die SPA bietet einen leichten Einstieg und lässt den Softwareentwickler in wenigen Schritten schnell zur lauffähigen Anwendung kommen. Außerdem verarbeitet der Demonstrator wenige bis keine Daten vom Server-Backend. Das Gleiche gilt auch für Benutzeraktionen innerhalb der Anwendung. Sie lassen sich hauptsächlich lokal im Browser und ohne Backend-Unterstützung verarbeiten. Eine Implementierung der serverseitigen Anwendung würde in dem Fall nicht nur unnötigen Aufwand, sondern womöglich auch Einbußen bei der User Experience mit sich bringen.

3.3 Frameworks für Single Page Applications

Single Page Applications werden meistens mit einem JavaScript-Framework oder einer JavaScript-Bibliothek umgesetzt. Laut Stack Overflow (vgl. [11]) sind die populärsten Frameworks/Bibliotheken: React, Angular und Vue.

React React wurde von Facebook entwickelt und im Jahr 2013 als Open-Source-Bibliothek veröffentlicht. Es erlaubt die Wiederverwendbarkeit der Komponenten durch modulare Grundlagen. Die Bibliothek ist auf JavaScript aufgebaut und lässt sich als Erweiterung in die bestehende Anwendung integrieren oder als autonome Webanwendung entwickeln. Die Entwickler von React empfehlen bei der Erstellung einer Webanwendung eine JSX-Programmiersprache zu verwenden, die aber

optional ist. JSX ist die syntaktische Erweiterung zu JavaScript und enthält JavaScript und XML. Nach dem Kompilieren werden JSX-Ausdrücke in normales JavaScript umgewandelt. (vgl. [12])

Angular Angular ist ein vollständiges JavaScript-Framework, hinter dem das Team von Google steht. Es ist ebenfalls als Open-Source-Software publiziert. Angular wurde im Jahr 2016 veröffentlicht und unterscheidet sich stark vom Vorgänger AngularJS. Das Framework ist komponentenorientiert aufgebaut, was die modulare Entwicklung ermöglicht. Es verwendet in der aktuellen Version die TypeScript-Programmiersprache, die eine objektorientierte Programmierung und auch statische Typisierung erlaubt. Der TypeScript Compiler transformiert den Quellcode in ein JavaScript. (vgl. [13])

Vue Vue ist ein JavaScript-Framework, das im Jahr 2014 von Evan You publiziert wurde. Das Framework basiert auf JavaScript und ist für die schrittweise Umsetzung geeignet. Das bedeutet, dass Vue entweder in die bestehende Anwendung integriert werden kann oder zusammen mit entsprechenden Tools und weiteren Bibliotheken das Grundgerüst komplexer Webanwendungen bilden kann. (vgl. [14])

Alle drei JavaScript-Frameworks sind für die Entwicklung der Anwendung vollständig geeignet. Dieser Demonstrator wird mit Angular entwickelt. Das Framework kommt der Struktur und Syntax einer objektorientierten Java-Programmiersprache sehr nahe, über die die Autorin Kenntnisse und Erfahrungen verfügt.

3.4 Erstellung eines Anwendungstemplates

3.4.1 Installation der benötigten Software

Eine komfortable Entwicklung der Webanwendung mit dem Angular-Framework erfordert die Installation von zusätzlicher Software.

Node.js und Node Package Manager Node.js besteht hauptsächlich aus drei Komponenten. Die Node.js-Engine stellt eine plattformübergreifende JavaScript

Runtime Enviroment dar (vgl. [15]). Der Node Package Manager (NPM) ist ein Manager-Programm. Es verwaltet eine der größten Open-Source-Bibliotheken und deren Abhängigkeiten. Das Tool ermöglicht es, beliebige Bibliotheken, Designkomponenten und funktionale Module auf einfache Art und Weise an die eigene Anwendung anzubinden (vgl. [16]). Die letzte Komponente in dem Tool-Set ist der Server. Er läuft auf dem Entwickler-Rechner, verarbeitet die Requests und liefert den Inhalt für einen Browser. Standardmäßig kommen der NPM, der Server und das Node.js zusammen als ein Installationspaket. Die Versionen lassen sich in einem Terminal wie folgt abfragen:

```
node -v
12.13.0
npm -v
6.12.0
```

Code Editor Im Vergleich zum einfachen Texteditor bieten IDEs[2], auch integrierte Entwicklungsumgebungen genannt, viele Vorteile an. Sie nehmen dem Entwickler die Routinearbeit ab und stellen hilfreiche Werkzeuge wie Syntaxhervorhebungen oder Autovervollständigung bereit. Für die Angular-Entwicklung sind mehrere IDEs geeignet. Die bekanntesten sind Visual Studio Code, Visual Studio, Webstorm und Angular IDE als Plugin für Eclipse. Für diese Arbeit wird Visual Studio Code verwendet. Die kostenlose IDE wurde von Microsoft entwickelt und unterstützt TypeScript sehr gut.

Angular CLI Das Angular Command Line Interface (CLI) ist ein Kommandozeilen-Tool. Es ermöglicht in nur wenigen Schritten die Erstellung eines voll funktionalen Anwendungstemplates. Außerdem unterstützt das CLI die Verwaltung und Weiterentwicklung eines Projektes und kümmert sich um eine einheitliche Programmstruktur. Die Installation des Angular-CLI erfolgt über die Kommandozeile

```
npm install -g @angular/cli
```

Mit der Ausführung des Befehls wird das CLI global auf dem verwendeten System installiert.

[2]IDE: integrated development environment

3.4.2 Erstellung eines Templates

Die Angular-CLI erzeugt ein Template, das eine simple, aber ausführbare Anwendung darstellt. Die Anwendung enthält keine Geschäftslogik, lediglich ein minimales HTML-Dokument mit vorkonfigurierten Projekteinstellungen. Dennoch ist das Template ein guter Startpunkt, um eine eigene Applikation zu entwickeln. Um ein Template mit dem Angular-CLI zu erzeugen, wird in der Kommandozeile folgender Befehl ausgeführt:

```
ng new hybrides-pm
```

Das Angular-CLI startet einen Wizard und fragt für das Projekt nach Grundeinstellungen. Danach erstellt das Angular-CLI in dem Ordner „hybrides-pm" das Angular-Template und installiert die benötigten Angular-Softwarepakete. Im Anschluss lässt sich die generierte Webanwendung starten, da das Angular-CLI hierfür über einen lokalen Entwicklungsserver verfügt. Mit dem Befehl

```
ng serve
```

kompiliert das CLI die Anwendung und führt sie auf dem lokalen Server aus. Somit ist die Anwendung unter der Adresse http://localhost:4200 mit einem Internetbrowser erreichbar. Abbildung 3.4 stellt hierfür das Angular-Template dar.

Abbildung 3.3 links zeigt die Projektstruktur im Visual Studio Code. Neben den vielen vorinstallierten Daten sind vor allem folgende Dateien erwähnenswert:

- ***app:*** Das Root-Verzeichnis hält alle Internetseiten als Komponenten in Form einer Baumstruktur zusammen.
- ***main.ts:*** Main Entry Point der Webanwendung.
- ***angular.json:*** Die zentrale CLI-Konfigurationsdatei. Sie verwaltet alle in der Anwendung angebundenen Bibliotheken.
- ***index.html:*** Das einzige HTML-Dokument, aus dem die Single Page Application besteht.

Diese und viele weiteren Dateien sind unter angular.io (vgl. [17]) gut dokumentiert.

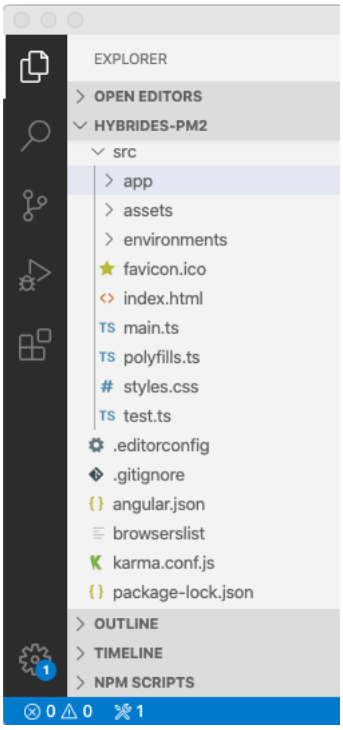

Abbildung 3.3: Projektstruktur

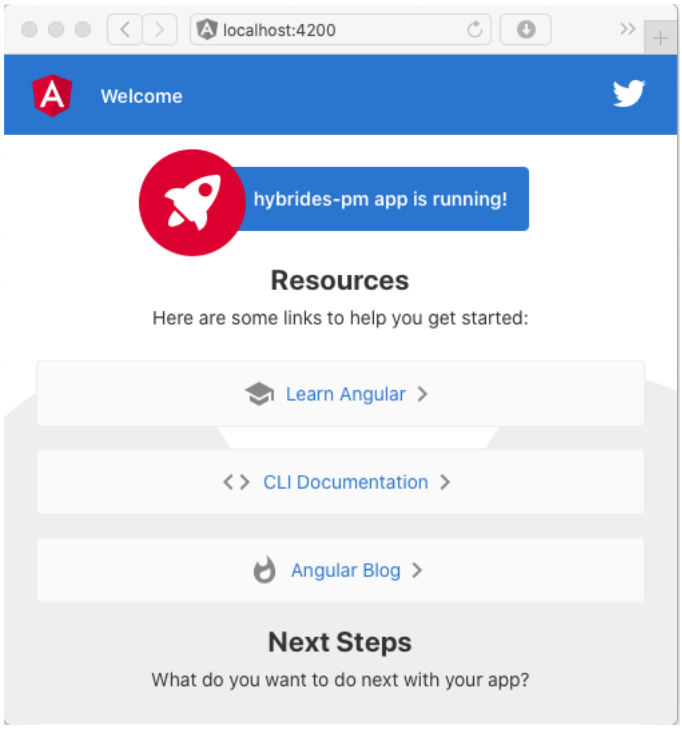

Abbildung 3.4: Hello-World-Projekt des Angular-CLI

3.5 Aufbau der Webanwendung

Dieser Abschnitt modularisiert die Webanwendung und baut die Architektur auf. Der Demonstrator folgt einer Struktur, die im Groben aus drei Teilen besteht. Als Erstes sollen die Adaptionsparameter interaktiv gestaltet werden, so dass der Anwender sie über eine Schnittstelle konfigurieren kann. Somit entstehen Daten, die als Eingabeparameter für das Tailoring hybrider Vorgehensmodelle verwendet werden. Der Algorithmus verarbeitet die Daten und stellt die Ausgabedaten in Form von Funktionen der agilen oder traditionellen Vorgehensmodelle bereit. Zum Schluss sollen dem Anwender ausgewählte Methoden in Form des reduzierten Ordnungsrahmens (vgl. Kapitel 2.2) präsentiert werden.

3.5.1 Die Angular-Komponente

Angular unterstützt Modularisierung durch Module oder Komponenten. Der Demonstrator besitzt eine überschaubare Architektur, deswegen werden hier Komponenten zur Modularisierung verwendet. Ein Modul ist ein eigenständiger, in sich abgekapselter Programm-Baustein, der aus vielen Komponenten bestehen kann. Er deklariert und administriert für sich alle seine Angular-Komponenten in seiner Root-Komponente. Die Deklaration aller Komponenten dieses Moduls findet in der Datei app.module.ts im Array declarations[ ] statt. Außerdem wird hier bei der Ausführung der Anwendung dem Angular-Framework der Einstiegspunkt über die bootstrap-Eigenschaft mitgeteilt.

Bei der Generierung des Anwendungstemplates wurde die erste Komponente, die AppComponent, bereits im Kapitel 3.4.2 generiert. Sie ist eine Root-Komponente und enthält somit die app.module.ts-Datei, deren Code in Listing 3.1 dargestellt wird.

```
...
@NgModule({
  declarations: [
        AppComponent
  ],
  ...
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Listing 3.1: app.module.ts

Alle Komponenten folgen einem einheitlichen Prinzip – siehe dazu die Abbildung 3.5. Dieses kapselt die drei Dateien *.html, *.css und *.ts zu einer Komponente. Eine Komponente repräsentiert meistens einen Teilinhalt einer Webseite im Internetbrowser. Sie stellt Funktionalitäten bereit, mit deren Hilfe der Anwender mit der Komponente interagieren kann. Jede Datei übernimmt dabei eine Aufgabe. So beschreibt die *.html-Datei die Struktur und den Inhalt der HTML-Seite. Die Datei *.css ist hingegen eine Stylesheet-Datei und dient zur Gestaltung und Formatierung der HTML-Seite. Die Logik der Komponente beschreibt die Datei *.ts.

Die Datei *.spec.ts hat bei der Abbildung der Komponente im Browser keinen direkten Einfluss, denn sie ist dafür bestimmt, die Funktionalität der Komponente im Entwicklungsprozess zu testen.

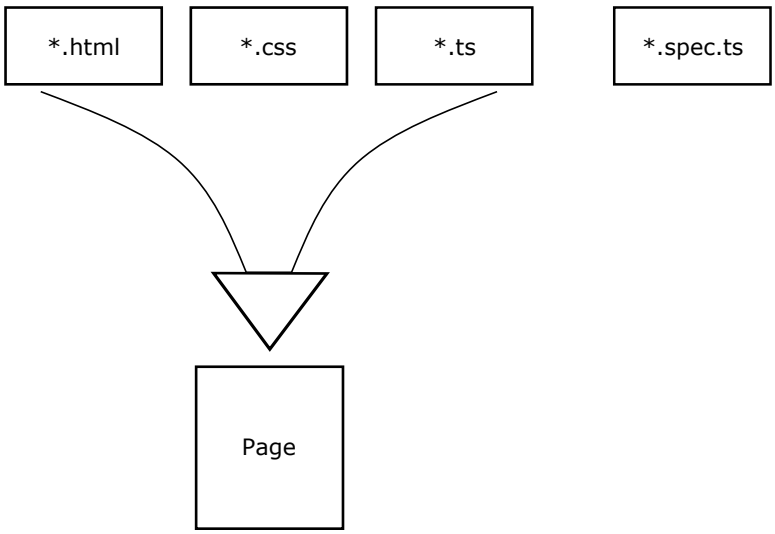

Abbildung 3.5: Bestandteile einer Komponente

Eine neue Komponente lässt sich folgendermaßen generieren:

```
ng g c componentName
```

Das Angular-CLI erstellt nicht nur vier Dateien, sondern kümmert sich auch um die Aufnahme einer neuen Komponenten in das declarations-Array in der Datei app.module.ts. Außerdem hängt es an den Dateinamen das Suffix ‚Component‘, was zu den Best Practices von Angular gehört.

3.5.2 Entwurf der Komponenten für den Demonstrator

Das bereits generierte Anwendungstemplate wird nun um zwei weitere Komponenten erweitert. Die erste Komponente ist die ParameterComponent. Ihre Hauptaufgabe besteht darin, die Eingaben eines Benutzers zu verarbeiten. Gleichzeitig spiegelt sie die vorgenommenen Konfigurationen auf der grafischen Benutzeroberfläche wider. Die ModellComponent ist die zweite Komponente. Sie stellt das adaptive

Modell in Form eines reduzierten Ordnungsrahmens dar. Die AppComponent bildet zusammen mit den beiden Komponenten ein Komponentenbaum der Webanwendung (vgl. die Abbildung 3.6).

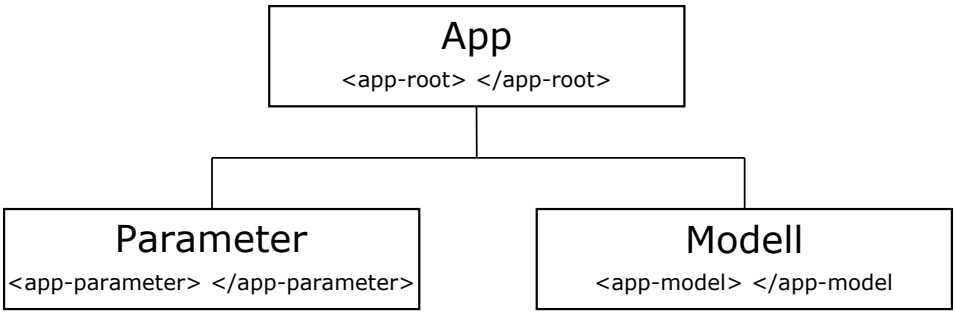

Abbildung 3.6: Komponentenbaum des Demonstrators

Die beiden Komponenten werden mit den folgenden Befehlen generiert.

```
ng g c parameter
ng g c modell
```

Die generierten TypeScript-Dateien sehen folgendermaßen aus:

```
@Component({
selector: 'app-parameter',
templateUrl: './parameter.component.html',
styleUrls: ['./parameter.component.css']
})
export class ParameterComponent implements OnInit { }
```

Listing 3.2: parameter.component.ts

```
@Component({
selector: 'app-modell',
templateUrl: './modell.component.html',
styleUrls: ['./modell.component.css']
})
export class ModellComponent implements OnInit { }
```

Listing 3.3: modell.component.ts

Die Eigenschaften templateUrl und styleUrls referenzieren entsprechende Style- und Content-Dateien. Der Selector repräsentiert die Komponente als ein HTML-Tag. Findet Angular diesen Tag in einem DOM[3]-Baum, so wird der Body des Tags durch die entsprechende Komponente ersetzt. So können die beiden Komponenten über den HTML-Tag in der Root-Komponente in der Datei app.component.html deklariert werden (siehe Listing 3.4). Somit haben die Komponenten eine geringe Kopplung und können unabhängig voneinander und von den restlichen Komponenten weiterentwickelt oder verändert werden.

```
...
<app-parameter> </app-parameter>
<app-modell> </app-modell>
...
```

Listing 3.4: app.component.html

3.5.3 Entwurf eines Service für den Demonstrator

Das Tailoring hybrider Vorgehensmodelle ist ein Rechenmodell. Es bekommt die Adaptionsparameter als Input und gibt als Output ein Vorgehensmodell aus. Die Vorgehensweise lässt sich wie folgt definieren:

1. Speicherung von Methoden der agilen und traditionellen Vorgehensmodelle.

[3]DOM: Document Object Model: standardisierte Spezifikation einer Schnittstelle, die Abfrage und Manipulation der Elemente eines HTML-Dokuments erlaubt.

2. Kapselung von Anwendungslogik, die das Tailoring hybrider Vorgehensmodelle enthält.

Das Rechenmodell arbeitet im Hintergrund und trägt in sich keine grafischen Elemente. Aufgrund dieser Erkenntnis muss das Programm als ein eigenständiger Service hervortreten. Dafür stellt das Angular das Dependency Injection Framework bereit und verfolgt das Ziel einer losen Kopplung zwischen Services und Komponenten. Die lose Kopplung wird so realisiert, dass die Instanziierung des Service nicht direkt in einer Komponente stattfindet, sondern eine Instanz des Frameworks den Service erzeugt und ihn an die Komponente übergibt, die den Service anfragt. Dafür bietet das Framework die Klasse Injektor, die die Services als sogenannte Provider verwaltet. Abbildung 3.7 zeigt exemplarisch die Abhängigkeiten zwischen Komponenten und Services.

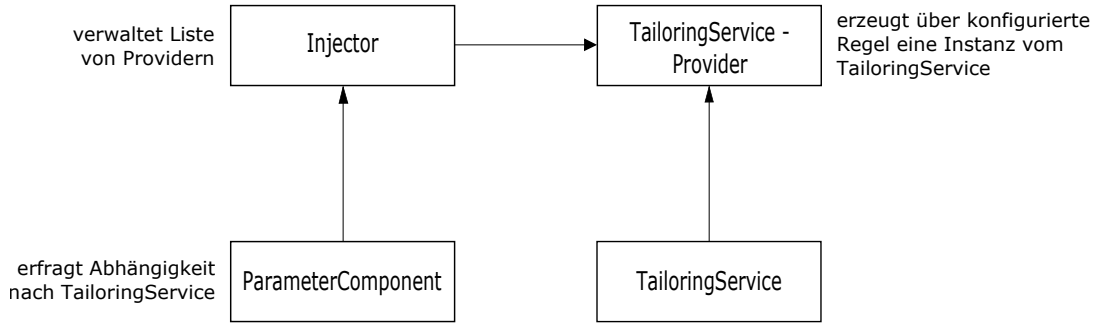

Abbildung 3.7: Angular Dependency Injection (in Anlehnung an [18], S. 278)

Die ParameterComponent fragt nach dem TailoringService. Der Injector durchsucht seine lokale Liste von Providern und gibt die Service-Instanz zurück. Sollte der Injector keine Instanz finden, so wendet er sich an den TailoringService-Provider. Letzterer erzeugt seinerseits eine Instanz des TailoringService. Danach speichert der Injector die Instanz in der Provider-Liste und behält diese über die gesamte Programmlaufzeit. Dieser Ansatz unterstützt nicht nur die unabhängige Realisierung untereinander, sondern lässt eine Cross-Kommunikation zwischen den Services und Komponenten zu.

Das Angular-CLI bietet den Befehl „service" an, um den TailoringService zu generieren.

```
ng generate service services/tailoring
```

Das CLI legt bei der Generierung des Service automatisch einen Ordner an. Das Verzeichnis enthält die Datei „tailoring.service.ts". Listing 3.5 zeigt einen Code-Ausschnitt des TailoringService.

```
@Injectable({
providedIn: 'root'
})
export class TailoringService { }
```

Listing 3.5: tailoring.component.ts

Die Eigenschaft „provideIn" bestimmt die Lebensdauer des Service. Mit der Zuweisung 'root' an die Eigenschaft „provideIn" entspricht die Lebensdauer des Service der der Anwendung.

Der TailoringService wird sowohl von der ParameterComponent als auch von der ModellComponent verwendet. Die ParameterComponent stellt die konfigurierten Adaptionsparameter bereit, die für den Algorithmus als Eingabeparameter dienen. Die ModellComponent verwendet wiederum die Ausgabe des Algorithmus und repräsentiert die ausgewählten Methoden im Ordnungsrahmen, die im Service gespeichert sind. Somit wird eine einzige Instanz des Service erzeugt und von beiden Komponenten verwendet, was wiederum bedeutet, dass die Änderungen an dem Service in den beiden Komponenten sichtbar werden.

```
@Component({
selector: 'app-parameter',
templateUrl: './parameter.component.html',
styleUrls: ['./parameter.component.css']
})
export class ParameterComponent implements OnInit{
constructor(private tailoringService: TailoringService) { }
}
```

Listing 3.6: parameter.component.ts mit injiziertem TailoringService

Der Code-Ausschnitt oben zeigt die ParameterComponent mit dem TailoringService. Die Injektion des Service geschieht innerhalb des Konstruktors als Parameter

und bekommt den Sichtbarkeitsmodifikator „private". In der spezifischen Kurzschreibweise laufen gleich mehrere Aktionen ab:

- Der TailoringService wird in die Klasse ParameterComponent injiziert.
- Eine private Membervariable *tailoringService* wird für den Service in der Klasse ParameterComponent angelegt.
- Der Konstruktorparameter *tailoringService* wird an die private Membervariable *tailoringService* zugewiesen.

Zum Abschluss dieser Phase steht eine ausführbare Webanwendung. Sie lässt sich bereits vom Server ausführen und im Browser darstellen. Der Inhalt der Webseite ist leer. Der Demonstrator in sich ist jedoch klar strukturiert. Die Parameter-Comonent übernimmt die Schnittstelle zum Benutzer. Sie verarbeitet den Input und reicht ihn an den TailoringService weiter. Der Service rechnet das Vorgehensmodell aus. Die ModelComponent nimmt die Daten vom Service auf und zeigt sie dem Endbenutzer. Auch wenn die Programm-Bausteine an der Stelle minimal implementiert sind, so ist der Datenfluss jedoch jetzt schon sichergestellt. Die Entwurfsphase ist somit beendet. Es steht die Ausprogrammierung an.

4 Programmierung der Webanwendung

Das im vorherigen Kapitel entworfene Template wird im Folgenden weiterentwickelt. Im Fokus dieses Kapitels steht die Programmierung der beiden Komponenten und des Service. Zunächst aber soll die Visualisierung der beiden Komponenten diskutiert und die Ansätze zur Modellierung des Tailorings erklärt werden.

4.1 Entwicklung der Komponente: ParameterComponent

4.1.1 Repräsentation der Daten

Die Adaptionsparameter können verbal oder als Grafik dargestellt werden. Die verbale Darstellung der Daten neigt dazu sehr schnell unübersichtlich zu werden. Grafiken dagegen halten Informationen in einem Bereich und werden von Menschen besser wahrgenommen. Da Adaptionsparameter konfiguriert sein sollen, ist es naheliegend, sie als Slider darzustellen. Acht Slider können in unterschiedlicher Weise repräsentiert werden. Eine Möglichkeit ist in Abbildung 4.1 dargestellt.

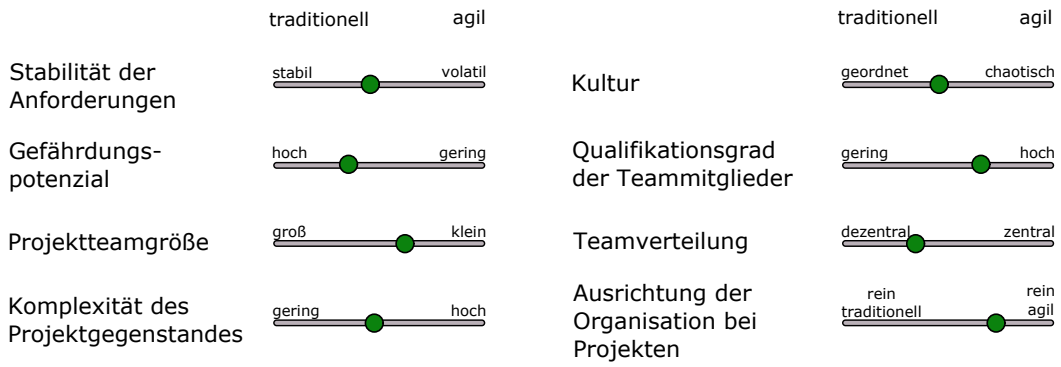

Abbildung 4.1: Skizze einer möglichen Darstellung der Parameter

Hier werden Adaptionsparameter als Slider nacheinander abgebildet. Die Abbildung genügt den Anforderungen des Demonstrators, da die Parameter vom Benutzer eingestellt werden können. Jedoch unterliegt die Abbildung einer Präsentationsschwäche, die in der Darstellung des Sliders liegt. Bei der Präsentation des Sliders in einem Kreis, wie in Abbildung 4.2 dargestellt, entstehen durch die konfigurierten Parameter Gebilde, die daraufhin deuten, ob eher ein agiles oder ein traditionelles Vorgehensmodell gewählt werden soll.

Abbildung 4.2: Adaptionsparameter (in Anlehnung an [10], S. 168)

Wenn sich das Gebilde eher in der Umgebung des Mittelpunktes konzentriert,

spricht das für ein agiles Vorgehensmodell. Auf ein traditionelles Vorgehensmodell deutet es dagegen hin, wenn sich das Gebilde eher am Rand des Kreises befindet. Ein weiterer Nachteil der Darstellung in Skizze 4.1 ist, dass bei einem mobilen Gerät mit einem schmalen Display alle Slider nacheinander aufgelistet werden, was wiederum die Übersichtlichkeit erschwert. Ein weiterer entscheidender Faktor sind Farben, die bei der Visualisierung helfen, Informationen noch deutlicher zu unterscheiden. Daher wird entschieden, die Repräsentation der Parameter auf der Grundlage der Abbildung 4.2 darzustellen.

4.1.2 Programmierung der ParameterComponent

Die Darstellung der einzelnen Elemente aus Abbildung 2.1 ist spezifisch. Jedes Element besitzt eine bestimmte Position und kann deswegen nicht wie in einem gewöhnlichen HTML-Dokument fließend nach den anderen Elementen dargestellt werden. Die Positionen der einzelnen Elemente sollen schon vor deren Darstellung ausgerechnet werden.

Kreis Im ersten Schritt wird der Kreis dargestellt. Dazu soll ein div-Tag mit Hilfe der css-Property *border-radius* zu einem Kreis geformt werden. Die css-Funktion *radial-gradient()* bestimmt hier den radialen Farbverlauf des Kreises. Die Funktion *containerCalc(w:number)* rechnet den Durchmesser des Kreises anhand der Breite des Browserfensters w aus. Die ausgerechnete Größe wird dynamisch in der HTML-Datei als Style des Kreises gesetzt. Angular stellt hierfür eine Sonderlösung bereit, die die Verbindung zwischen Anwendungslogik der Komponente und deren HTML-Template ermöglicht. Ändern sich die Werte in der Logik-Datei, aktualisiert Angular die HTML-Datei automatisch. Die Initialisierung des Kreises geschieht in der Methode *ngOnInit()*.

Slider Im nächsten Schritt werden Slider implementiert, die zu konfigurierende Adaptionsparameter abbilden. Der Benutzer konfiguriert die Adaptionsparameter durch einen Klick oder durch das Ziehen eines Schiebereglers. Somit lässt sich die Anforderung A6 abdecken. Die Slider stammen aus der Bibliothek Angular

Material[1] und lassen sich sowohl mit der Maus als auch mit dem Touchscreen bedienen.

Im Laufe der Entwicklung des Demonstrators wurden Änderungen vorgenommen. Es sollen nicht acht, sondern zehn Parameter erfasst werden. Somit enthält die ParameterComponent zehn Slider. Die Anzahl der Slider bestimmt die Konstante *slidAnz*. Dadurch wird sichergestellt, dass Änderungen bezüglich der Slider-Anzahl unkompliziert vorgenommen werden können. Jeder Slider beginnt im Mittelpunkt des Kreises und seine Länge entspricht dem Radius des Kreises. Außerdem ordnen sich die Slider in einem bestimmten Winkel zueinander an. Die Winkel, in denen die Slider sich drehen, werden im Konstruktor anhand der Konstante *slidAnz* ausgerechnet und im Array *slidWinkel[ ]* gespeichert. Zusätzlich speichert jeder Slider für sich folgende Informationen ab:

- id: ID des Sliders
- textSlider: Bezeichnung des Adaptionsparameters
- disabled: boolesche Variable, die bestimmt, ob ein Slider aktiv oder inaktiv ist
- value: Wert des Sliders, der vom Benutzer eingestellt wird
- invert: boolesche Variable, die angibt, ob ein Slider invertiert wird oder nicht

Das Array *sliders[ ]* hält alle Slider zusammen.

In der Datei parameter.component.html werden Slider mit Hilfe der NgFor-Direktive[2] deklariert (siehe Listing 4.1). Die Direktive iteriert über das Array *sliders[ ]* und erzeugt für jeden Slider aus dem Array eine Slider-Instanz. *Let s* bildet das entsprechende Slider-Objekt ab, auf dessen Variablen durch die Punkt-Notation zugegriffen werden kann.

Propertys *min*, *max* und *step* bestimmen die Menge der Werte, die ein Slider annehmen kann. Somit ist diese Menge durch die Menge [0, 1, 2, 3, 4, 5, 6, 7, 8] begrenzt.

[1] Angular Material ist eine Open-Source-Bibliothek. Sie wurde speziell für Angular entwickelt und enthält auf Basis von HTML, CSS und JavaScript viele Oberflächengestaltungselemente. (vgl. [19])

[2] Die NgFor-Direktive ist eine strukturelle Direktive, die neue Elemente in den DOM-Baum dynamisch einfügt.

Die ngStyle-Direktive beschreibt das Style-Attribut, das bestimmt, um wie viel Grad der jeweilige Slider gedreht werden soll.

```
/* Eine strukturelle Direktive, die einer for-Schleife
   entspricht */
<mat-slider *ngFor="let s of sliders; index as i;"

/* Standardeinstellungen für einen Slider */
max="8" min="0" step = "1"

/* Drehung des Sliders um den Winkeln, die das Array
   sliderWinkel[] enthält */
[ngStyle]="{'transform': 'rotate('+sliderWinkel[i]+'deg'}"

/* Zuweisung den Werten des Sliders an die entsprechenden
   Propertys */
[id]="s.id" [disabled]="s.disabled" [invert]="s.invert"

/* Two-Way Data Binding des Angular*/
[(ngModel)]="s.value"

/* Bei der Änderung des Wertes, wird die Methode onUpdate()
   aufgerufen. */
(change)="onUpdate()"

/* Darstellung des Sliders bezüglich der z-Koordinate */
[class.inputDisable]="s.disabled" [class.inputEnable]="!s.
   disabled">

</mat-slider>
```

Listing 4.1: Definition der Slider in der Datei parameter.component.html

Die spezielle Syntax {{s.id}} stellt die Interpolation dar, mit deren Hilfe die Daten aus der Komponente im HTML-View als String ausgegeben werden. Auf diese Weise werden Werte der Slider den entsprechenden Propertys zugewiesen.

Mit [(ngModel)] realisiert Angular das Two-Way-Binding, das ermöglicht, die Än-

derungen beidseitig zu erfassen. Das bedeutet einerseits, dass die Änderungen des Wertes eines Sliders in der Logikdatei der Komponente automatisch im View aktualisiert werden. Dies erfolgt zum Beispiel beim Starten der Anwendung, wenn die Slider-Values initialisiert werden. Andererseits lässt sich der modifizierte Wert eines Sliders im View, was beispielsweise bei der Einstellung des Sliders erfolgt, automatisch an die Variable *value* des Slider-Objekts zuweisen. Auf diese Weise realisiert Angular das Event-Binding.

Das change-Event bindet die Methode onUpdate an, damit der modifizierte Wert des Sliders nicht nur für den Algorithmus, sondern auch für die Aktualisierung des Polygons erfasst wird, das nachfolgend erklärt wird. Die Methode ermittelt die ID des Sliders, der das Event ausgelöst hat und aktualisiert die Werte des Polygons.

Die Property *disabled* spiegelt den Zustand des Sliders wider, der entweder aktiviert oder deaktiviert sein kann. Dies ergibt sich daraus, dass die Positionierung der Slider einen Konflikt untereinander auslöst. Alle zehn Slider werden mit der Position absolut an einer einzigen Stelle im Browser abgebildet. Danach lässt sich jeder Slider um den angegebenen Winkel drehen. Das mat-slider-Feld selbst ist aber breiter als der abgebildete Slider. Somit entsteht ein Konflikt zwischen den Slidern im Mittelpunkt des Kreises. Abbildung 4.3 demonstriert diesen Konflikt. Das Problem zeigt sich darin, dass sich mit einem Klick auf Sleider 1 im Nullpunkt Slider 10 einstellt. Somit gelingt es nie, Slider 1 mit einem Klick auf den Nullpunkt einzustellen. Ein identisches Problem tritt auch bei den mobilen Geräten mit schmalen Displays auf. Hier bedient der Benutzer die Anwendung mit Hilfe des Fingers oder des Zeigestiftes, der sehr präzise den zu konfigurierenden Slider berühren soll. Bei einer solchen Darstellung kann es aber passieren, dass der Benutzer den nächstgelegenen oder sogar schon eingestellten Slider trifft. Eine mögliche Lösung besteht darin, dass zu einem bestimmten Zeitpunkt nur ein einziger Slider konfiguriert werden kann. Icons in Form von Dreiecken dienen dazu, den Zustand des Sliders zu verändern. Der Klick auf das Icon aktiviert den entsprechenden Slider und deaktiviert alle anderen. Die boolesche Slider-Variable *disabled* bestimmt dynamisch, welche css-Klasse für den Slider angewendet wird. Die beiden css-Klassen inputDisable und inputEnable bestimmen wiederum die Position des Sliders bezüglich der z-Koordinate, die angibt, ob ein Slider vor den anderen abgebildet wird oder nicht.

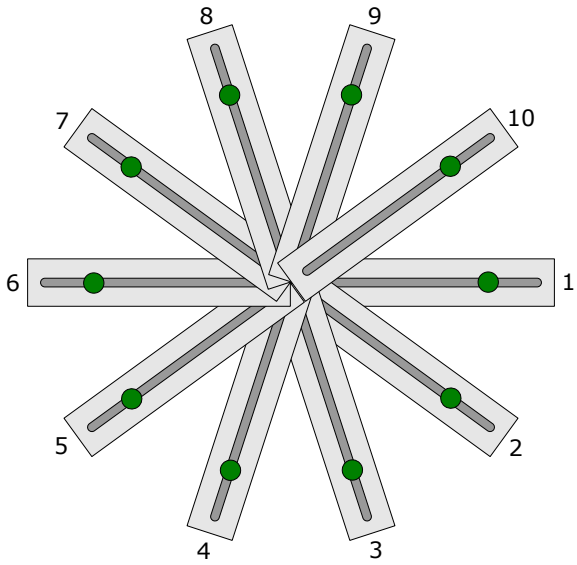

Abbildung 4.3: Konflikt zwischen den mat-slider-Feldern

Icons Mit dem Klick auf ein Icon lässt sich der entsprechende Slider aktivieren. Icons werden aus der Bibliothek Icon Angular Material verwendet. Bevor das Icon verwendet wird, soll es im Konstruktor der verwendeten Komponente registriert und der Pfad zu dem Icon angegeben werden. Jedes Icon besitzt folgende Struktur:

- iconId: ID des Icons
- iconCoord: Koordinaten des Icons

Das Array *iconSliders[ ]* hält Informationen über alle Icons zusammen. Die Methode *iconSlidsCalc()* errechnet die Positionen des Icons mit Hilfe vom Radius des Kreises und den Winkeln aus dem Array *slidWinkel[ ]*. Mit der scc-Property *transform* werden Icons entsprechend gedreht und positioniert. Das click-Event bindet an die Methode *iconClick($event)*, die den entsprechenden Slider aktiviert und alle anderen deaktiviert.

Polygon Polygon stellt das Gebilde zwischen den Knöpfen des Sliders dar und besteht aus geraden Linien. Es wird mit Hilfe des svg[3] realisiert und durch einen String von Punkten definiert. Die Methode *polygonInit()* berechnet die Punkte anhand der Länge, die vom Mittelpunkt bis zum Knopf des Sliders entsteht, sowie anhand des Winkels aus dem Array *slidWinkel[ ]* und speichert sie im Array

[3]Scalable Vector Graphics

polygonPoints[ ] ab. Die Methode *getPolygonPoints()* transformiert das Array *polygonPoints[ ]* in den String. Die Methode *polygonUpdate(slidId:number, currentValue:number)* aktualisiert die Punkte, wenn ein Slider konfiguriert wird.

Button Der Anwender betätigt den Button „Übernehmen" durch einen Klick oder Touch und löst somit folgende Aktivitäten aus:

- Die Werte der konfigurierten Adaptionsparameter werden an den Algorithmus übergeben.
- Der Algorithmus rechnet das individuelle Vorgehensmodell aus.
- Der Algorithmus übergibt das ausgerechnete Modell an die Modell-Komponente.

Technisch realisiert der Button das click-Event, indem er das Event an die Methode *buttonClick()* bindet. Die Methode iteriert über das Array *sliders[ ]* und übernimmt in das Array *slidValues[ ]* nur die konfigurierten Werte der Slider. Danach ruft sie die Methode *update(this.slidValues)* des Service auf und übergibt als Parameter das Array *slidValues[ ].*

Responsivität Die Responsivität wird mit Hilfe des resize-Events umgesetzt, das an die Methode *onResize()* bindet. Die Methode *onResize()* fragt die Breite des Browserfensters ab und aktualisiert anhand dieser Information den Durchmesser des Kreises, die Koordinaten des Icons und die Punkte des Polygons. Für die Breite des Browserfensters bis 590 px wird eine angepasste Darstellung realisiert. Die Bezeichnungen der Slider um den Kreis werden mit Hilfe des css Media Queries *@media screen and (max-width: 590px)* ausgeblendet. Ein Rahmen oberhalb des Kreises lässt sich darstellen, in dem sich die Bezeichnung des aktivierten Sliders befindet. Somit aktiviert sich beim Klick auf das Icon der entsprechende Slider und die Bezeichnung des aktivierten Sliders im Rahmen aktualisiert sich. Die Methode *textChange()* bestimmt, welche Bezeichnung eingeblendet wird.

Die Abbildungen 4.4 und 4.5 zeigen den ausprogrammierten ParameterComponent im Webbrowser Safari.

Abbildung 4.4: Abbildung der Komponente auf einem Desktop-PC

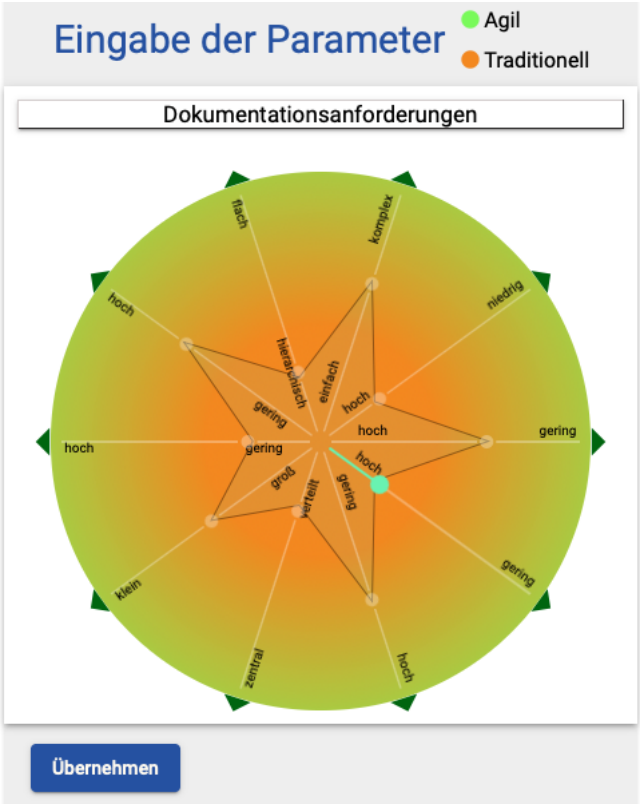

Abbildung 4.5: Abbildung der Komponente auf einem Smartphone

4.2 Entwicklung der ModellComponent

4.2.1 Grafische Darstellung des adaptiven Vorgehensmodells

Die ausgewählten Methoden und Werkzeuge des adaptiven Vorgehensmodells lassen sich in unterschiedlicher Weise darstellen. Eine Möglichkeit wäre, sie als Liste darzustellen. Diese Darstellungsweise hat einen Nachteil, denn es gibt keinerlei Anzeichen, dass die Methoden agil durchgeführt werden können, der iterative Ansatz fehlt in diesem Fall. Eine andere Möglichkeit stellt die Visualisierung jedes Vorgehensmodells für sich selbst dar. Dieser Ansatz schließt allerdings die Darstellung der hybriden Vorgehensmodelle aus und nimmt viel Zeit bei der Programmierung jedes einzelnen Vorgehensmodells in Anspruch. Ein in sich schlüssiges Modell stellt der Ordnungsrahmen dar, der im Kapitel 2.2 erläutert wurde. Der Ordnungsrahmen erlaubt die einheitliche Darstellung des agilen, traditionellen oder hybriden

Vorgehensmodells. Somit wird in der Arbeit für die Visualisierung des individuellen Vorgehensmodells der Ordnungsrahmen verwendet.

4.2.2 Programmierung der ModellComponent

Der Ordnungsrahmen lässt sich mit Hilfe von svg realisieren. Diese projiziert ihre Abbildungen in eine View-Box, für die die Breite und Höhe angegeben werden sollen. Die Responsivität wird hier gewährleistet, indem die View-Box anhand der Breite des Browserfensters skaliert wird. Die Methode *modelCalc(w:number)* mit dem Parameter *w*, der der Breite des Browserfensters entspricht, stellt die entsprechende Funktionalität bereit und speichert die Größe der View-Box in der Membervariable *modelWidth*. Das Modell selbst besteht aus mehren Teilen, die in Abbildung 4.6 abgebildet werden.

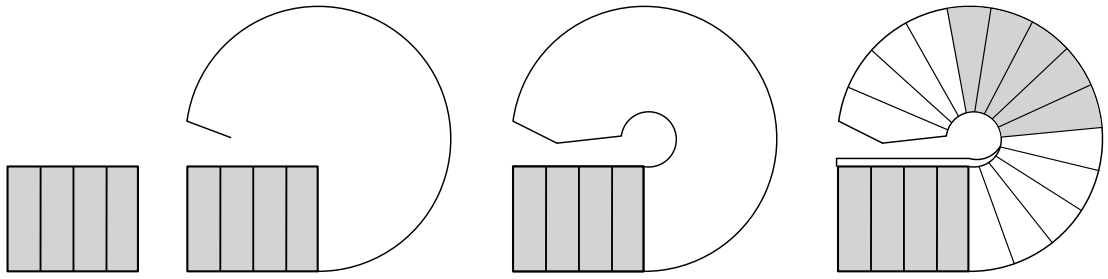

Abbildung 4.6: Bestandteile des Ordnungsrahmens

Die Idee, wie sich das Modell abbilden lässt, besteht darin, die View-Box als ein Quadrat zu definieren, das sich in vier Quadranten unterteilen lässt. Somit entsteht ein gedankliches Koordinatensystem, das bei der weiteren Implementierung Hilfe leistet. Auf dieser Stelle ist zu erwähnen, dass die Koordinaten der View-Box in der oberen linken Ecke bei (0,0) beginnen. Somit wächt die x-Koordinate von links nach rechts und die y-Koordinate hingegen von oben nach unten. Dabei liegt der Koordinatenursprung genau in der Mitte der View-Box, was für den Ursprung bedeutet, dass seine Koordinaten den x=y=*modelWidth*/2 entsprechen.

Als Erstes lässt sich ein Viereck im dritten Quadranten der View-Box darstellen. Das Listing 4.2 zeigt den Quellcode des Rechtecks in der HTML-Datei.

```
...
<rect
/* (x,y)-Koordinaten der oberen linken Ecke des Rechtecks */
[attr.x]="2" [attr.y]="modelWidth - rectHeight"

/* Breite und Höhe des Rechtecks */
[attr.width]="rectWidth" [attr.height]="rectHeight"

/* Style des Rechtecks */
class="rectStyle">
</rect>

<line class="lineRect"

/* Startpunkt der Strecke */
[attr.x1]="methodRect[0]" [attr.y1]="modelWidth - rectHeight"

/* Endpunkt der Strecke */
[attr.x2]="methodRect[0]" [attr.y2]="odelWidth">
</line>
...
```

Listing 4.2: Ausschnitt aus der Datei modell.component.html stellt die svg-Grundform <rect> und <line> dar

Das Viereck lässt sich mit der svg-Grundform *<rect>* realisieren. Die Zeichnung des Vierecks beginnt bei der oberen linken Ecke mit den angegebenen (x,y)-Koordinaten. Die Breite und die Höhe des Rechtecks werden mit der Methode *coordRect(mWidth: number)* ausgerechnet und entsprechen jeweils 50 % und 40 % von der Größe der View-Box. Außerdem lässt sich das Viereck in vier Segmente unterteilen, die mit der svg-Grundform *<line>* dargestellt werden. Die svg-Grundform *<line>* stellt eine Strecke dar, mit der Angabe des Start- und Endpunktes. Die x-Koordinaten der Start- und Endpunkten einer Strecke sind in diesem Fall gleich, da die Strecken parallel zur y-Achse verlaufen. Die Methode *coordRect(mWidth:number)* berechnet die x-Koordinaten und speichert sie im Array *methodRect[ ]*. Die y-Koordinaten

entsprechen denen des Rechtecks.

Im nächsten Schritt lässt sich das Gebilde im zweiten Teil der Abbildung 4.6 mit Hilfe des Grundelementes *<path>* darstellen. Im Vergleich dazu der Quellcode im Listing 4.3.

```
...
<path
/*Style des Pfades */
class="pathStyle"

/* Anfangskoordinaten des Pfades */
[attr.d]="'M'+ rectWidth +','+ modelWidth +" +

/* Ein großer Kreisbogen mit Angaben des Radius und Richtung */
"' A'+ rectWidth +','+ rectWidth + ' 0 1 0 '" +

/* Endkoordinaten des Kreisbogens */
"+ lastPoint.p2.x + ',' + lastPoint.p2.y" +

/* Linie zu angegebenen Koordinaten */
"+' L'+ endX +','+ endY"

/>
...
```

Listing 4.3: Quellcode zum zweiten Gebilde aus Abbildung 4.6

Zuerst wird der Style des Pfades angegeben, in dem die Farbe definiert wird. Der Pfad selbst fängt bei der unteren rechten Ecke des Rechtecks mit den angegebenen Koordinaten an. Der weitere Befehl des Pfades erzeugt einen Bogen mit dem Radius, der der Breite des Rechtecks entspricht. Die Endkoordinaten des Bogens lassen sich in der Methode *lastPointCalc()* ausrechnen. Der letzte Befehl lässt eine gerade Linie zwischen den Endkoordinaten des Bogens und den angegebenen Koordinaten *endX* und *endY* zeichnen, die anhand der Breite des Rechteckes und des Modells ausgerechnet werden

Darauffolgend lässt sich äquivalent der kleine Bogen darstellen. Die Anfangskoordinaten entsprechen der oberen rechten Ecke des Rechtecks. Der Radius des Bogens wird in der Methode *coordRect(mWidth: number)* bestimmt und beträgt 10 % der View-Box-Größe. Die Endkoordinaten des kleinen Bogens lassen sich in der Methode *lastPointCalc()* anhand von Radius und Winkel ausrechnen. Wie beim großen Bogen auch verbindet eine gerade Linie den Endpunkt des kleinen Bogens mit den Koordinaten ($endX$, $endY$).

Anschließend lassen sich Strecken zwischen den beiden Bogen darstellen, indem sie mit der svg-Grundform *<line>* abgebildet werden. Die Anfangskoordinaten liegen auf dem kleinen Bogen jeweils um 19° verschoben, die Endkoordinaten sind entsprechend auf dem großen Bogen um 19° voneinander entfernt. Die Punkte lassen sich mit Hilfe der entsprechenden Radien und des 19°-Winkels berechnen. Die Methode *pointsCalc()* berechnet die Punkte und speichert sie in das Array *linePoints[ ]*.

Die Methoden und Werkzeuge eines individuellen, adaptiven Vorgehensmodells werden im Service in einem zweidimensionalen Array gespeichert. Ein zweidimensionales Array wird aus dem Grund verwendet, weil für jedes Element des Ordnungsrahmens gleich mehrere Methoden hinterlegt werden können. Die Positionen der Bezeichnungen berechnet die Methode *pointsCalc()* und speichert sie in das Array *textPoints[ ]*. Zudem berechnet die Methode auch die Winkel, um die die Bezeichnungen gedreht werden sollen und speichert sie in das Array *textAngle[ ]*. Mit Hilfe der ausgerechneten Daten werden Bezeichnungen positioniert und gedreht. Wenn ein Element im Ordnungsrahmen mehrere Methoden enthält, wird für das Element dynamisch eine ausklappbare Liste erstellt. Dafür enthält das Array *Collaps[ ]* für jedes Element des Ordnungsrahmens eine ID des Elements und eine boolesche Variable *toogle,* die bestimmt, ob die Liste ein- bzw. ausgeklappt wird. Das Ein- Ausklappen lässt sich mit Hilfe des click-Events realisieren. Bei einem Klick wird der Variable *toogle* das Gegenteil *!toogle* zugewiesen. Ein Beispiel für eine ausgeklappte Liste ist in Abbildung 4.7 dargestellt.

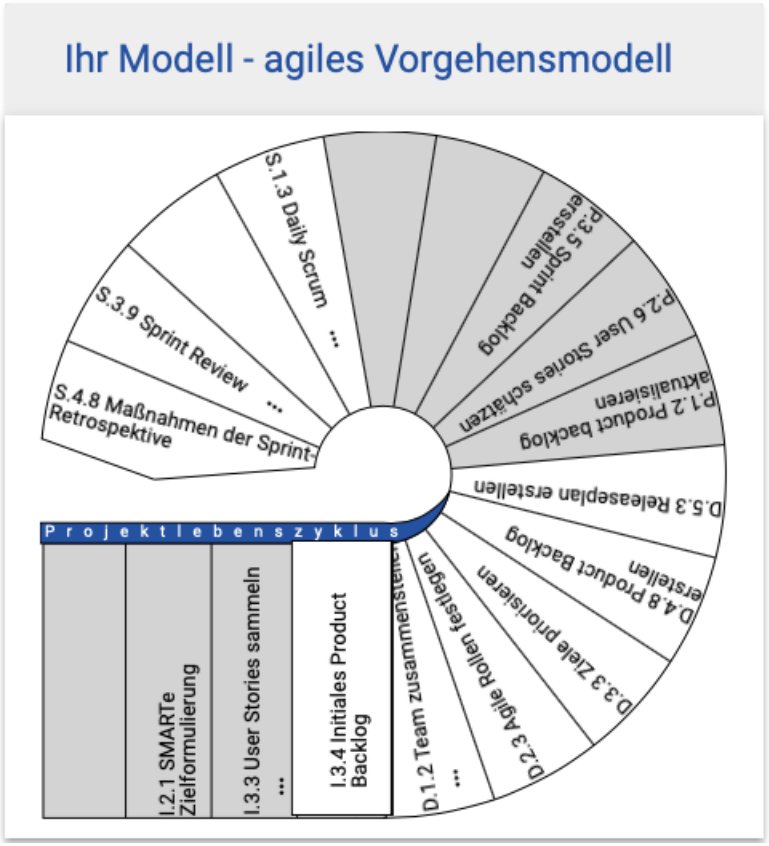

Abbildung 4.7: Ordnungsrahmen mit einer ausgeklappten Liste

4.3 Modellierung des Service

Das Tailoring eines adaptiven Vorgehensmodells lässt sich als Service realisieren. Ein Ansatz zur Auswertung der Adaptionsparameter wurde von Paukner, Seel und Timinger in ihrem Werk (vgl. [10]) vorgestellt. Dort wird das Tailoring hybrider Vorgehensmodelle anhand des mehrstufigen Adaptionsmechanismus in einem Top-Down-Verfahren definiert. Da in dieser Arbeit der Algorithmus zuerst nur zwischen agilen und traditionellen Vorgehensmodellen unterscheiden soll und die Arbeit zeitlich begrenzt ist, wird hier ein Algorithmus entwickelt, der im größten Teil auf dem arithmetischen Mittel basiert. Der Algorithmus wird zum Schluss des Kapitels ausführlich erklärt.

Die Klasse TailoringService realisiert den Service und definiert das Tailoring hybrider Vorgehensmodelle. Der Service enthält eine Liste *listeOfMethods* von allen

möglichen Methoden und Werkzeugen (vgl. [2], S. 377–500) der agilen und traditionellen Vorgehensmodelle und wird als *Map<string, string>* definiert. Jeder Eintrag der Liste besteht aus einem Schlüssel-Wert-Paar, indem der Schlüssel die aus dem Ordnungsrahmen abgeleitete Identifizierungsnummer darstellt und der Wert die Bezeichnung der jeweiligen Methode enthält. Das dazu passende Listing 4.4 sieht folgendermaßen aus.

```
listeOfMethods: Map<string, string> = new Map([
["I.1.1", "Projektsteckbrief erstellen"],
["I.1.4", "Teambuilding fördern"],
["I.2.1", "SMARTe Zielformulierung"],
["I.3.2","Lastenheft erstellen"],
["I.3.3","User Stories sammeln"],
...
]);
```

Listing 4.4: Der Ausschnitt aus der Klasse TailoringService stellt die Liste aller möglichen Methoden dar

Die Klasse definiert außerdem Methodensätze jeweils für agile und traditionelle Vorgehensmodelle, die als Funktionen *methodenSatzAgil()* und *methodenSatzTrad()* realisiert werden. Die beiden Funktionen fügen entsprechende Methoden in das zweidimensionale Array *methodsList[ ][ ]* ein, das nach dem Tailoring das adaptive Vorgehensmodell enthält. Ein zweidimensionales Array wird verwendet, weil jedes Element des Ordnungsrahmens mehrere Methoden enthalten kann.

Das Tailoring selbst lässt sich mit der Methode *update(values:number[ ])* realisieren, die als Eingabeparameter die vom Benutzer konfigurierten Adaptionsparameter erhält. Die Methode löscht alle Methoden und Werkzeuge, die das Array *methodsList[ ][ ]* enthält und startet den Algorithmus.

Der Algorithmus beruht im Allgemeinem auf dem arithmetischen Mittel, wohingegen die rein agilen oder rein traditionellen Ausprägungen der Parameter mehr berücksichtigt werden. Darüber hinaus wird angenommen, dass jeder Parameter das gleiche Gewicht besitzt. Der Algorithmus lässt sich in Abbildung 4.8 in Form eines Aktivitätsdiagramms darstellen.

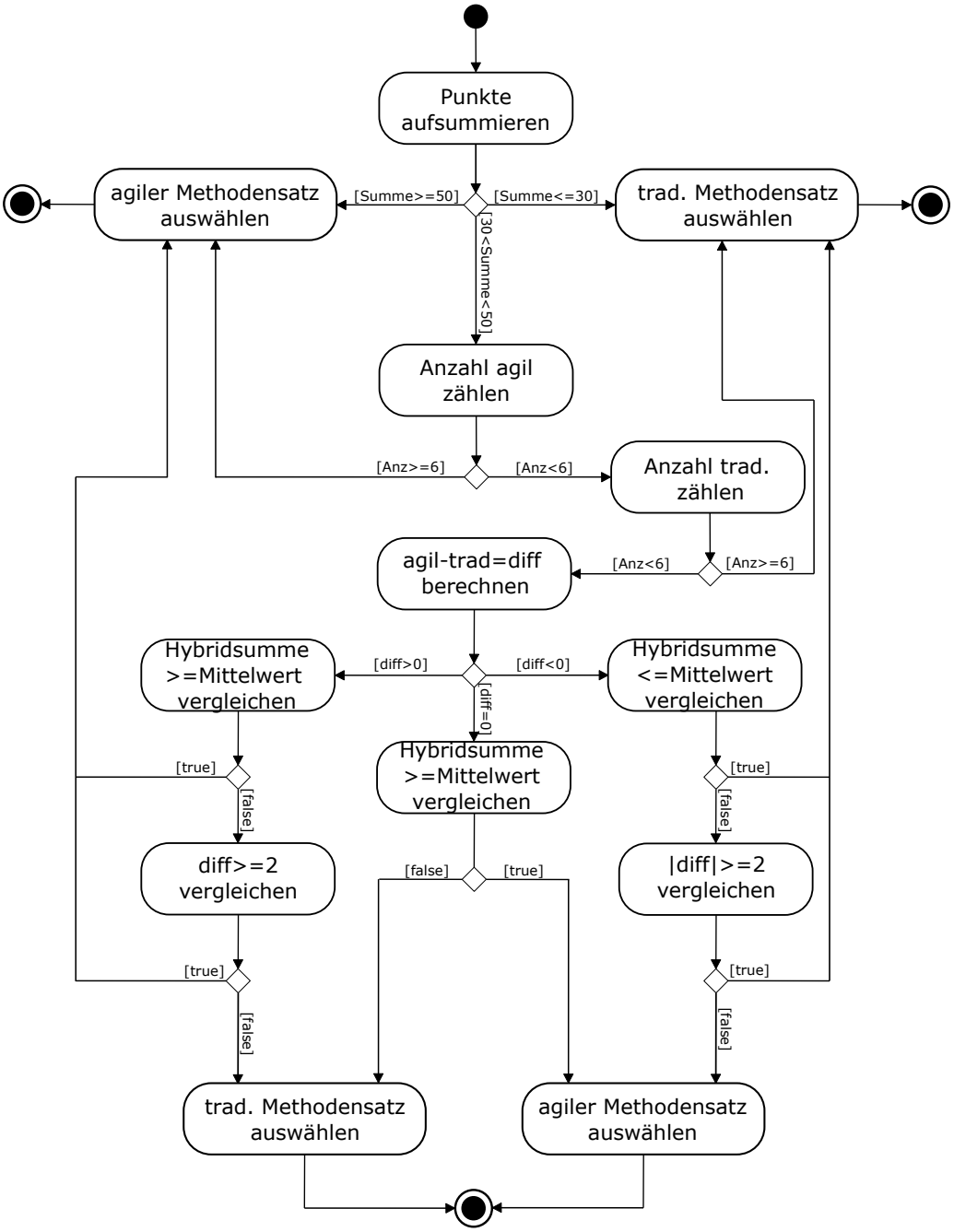

Abbildung 4.8: Algorithmus für das Tailoring hybrider Vorgehensmodelle

Die Adaptionsparameter können neun unterschiedliche Werte aus der Menge [0, 1, 2, 3, 4, 5, 6, 7, 8] enthalten. Die Menge lässt sich in drei fest definierte Teilmengen aufteilen. Die erste Teilmenge [0, 1, 2] entspricht den traditionellen Werten. Die zweite Teilmenge [3, 4, 5] berücksichtigt die hybriden Ansätze. Die

agilen Werte lassen sich durch die dritte Teilmenge [6, 7, 8] beschreiben. Als Erstes werden in dem Algorithmus alle Punkte der konfigurierten Adaptionsparameter zusammengezählt und daraus die Summe gebildet. Bei zehn Adaptionsparametern kann die Summe maximal 80 Punkte erreichen. Liegt die Summe im Bereich von 0 bis 30, empfiehlt sich ein traditionelles Vorgehensmodell. Ein agiles Vorgehensmodell wird hingegen dann bevorzugt, wenn die Summe über 50 Punkte erreicht. Bewegt sich die Summe zwischen den beiden Bereichen, wird weiter analysiert.

Als Nächstes wird untersucht, ob die Anzahl der Parameter, die eine agile Ausprägung besitzen, mehr als die Hälfte der konfigurierten Parameter beträgt. Wenn das der Fall ist, empfiehlt sich ein agiles Vorgehensmodell. Im anderen Fall lässt sich die Anzahl der Parameter auswerten, die eine traditionelle Ausprägung enthalten. Beträgt diese Anzahl mehr als die Hälfte aller konfigurierten Parameter, empfiehlt sich ein traditionelles Vorgehensmodell.

Ansonsten wird die Differenz zwischen der Anzahl der agil ausgeprägten Parameter und der Anzahl der traditionell ausgeprägten Parameter bestimmt. Die positive Differenz bedeutet, dass die Anzahl agil ausgeprägter Parameter höher als die Anzahl traditionell ausgeprägten Parameter ist. Außerdem lässt sich die hybride Summe aus den Parametern bilden, die eine hybride Ausprägung besitzen und mit dem Mittelwert vergleichen, der aus der Anzahl dieser Parameter mal vier gebildet wird. Somit wird entschieden, ob diese Parameter mehr für agile oder traditionelle Formen beitragen. Anschließend werden die Differenz und die hybride Summe genauer analysiert. Ein agiles Vorgehensmodell empfiehlt sich, wenn sowohl die Anzahl der agil ausgeprägten Parameter höher als die Anzahl der traditionell ausgeprägten Parameter ist als auch die hybride Summe mehr die agilen Formen unterstützt. Trägt die hybride Summe zu den traditionellen Formen bei, wohingegen aber die Differenz zwei oder mehr ist, wird ebenfalls ein agiles Vorgehensmodell bestimmt. Ein traditionelles Vorgehensmodell wird in dem Fall empfohlen, wenn die Anzahl der agil ausgeprägten Parameter nur um einen höher als die Anzahl der traditionell ausgeprägten Parameter ist und die hybride Summe mehr die traditionellen Formen unterstützt. Andersherum wird vorgegangen, wenn die Differenz negativ ist. Ist die Anzahl der traditionell ausgeprägten Parameter nur um einen höher als die Anzahl der agil ausgeprägten Parameter und die hybride Summe spricht für die agile Formen, empfiehlt sich ein agiles Modell. In allen anderen Fällen bei negativer Differenz wird ein traditionelles Vorgehensmodell bevorzugt.

Ist die Anzahl der agil ausgeprägten Parameter gleich der Anzahl der traditionell ausgeprägten Parameter, wird anhand der hybriden Summe entschieden.

4.4 Konfigurationsbeispiele des Demonstrators

Nachfolgend werden einige Beispiele der Anwendung gezeigt. Abbildungen 4.9 und 4.10 stellen die Webanwendung nach dem Programmstart dar.

Abbildung 4.9: ParameterComponent nach dem Starten der Anwendung

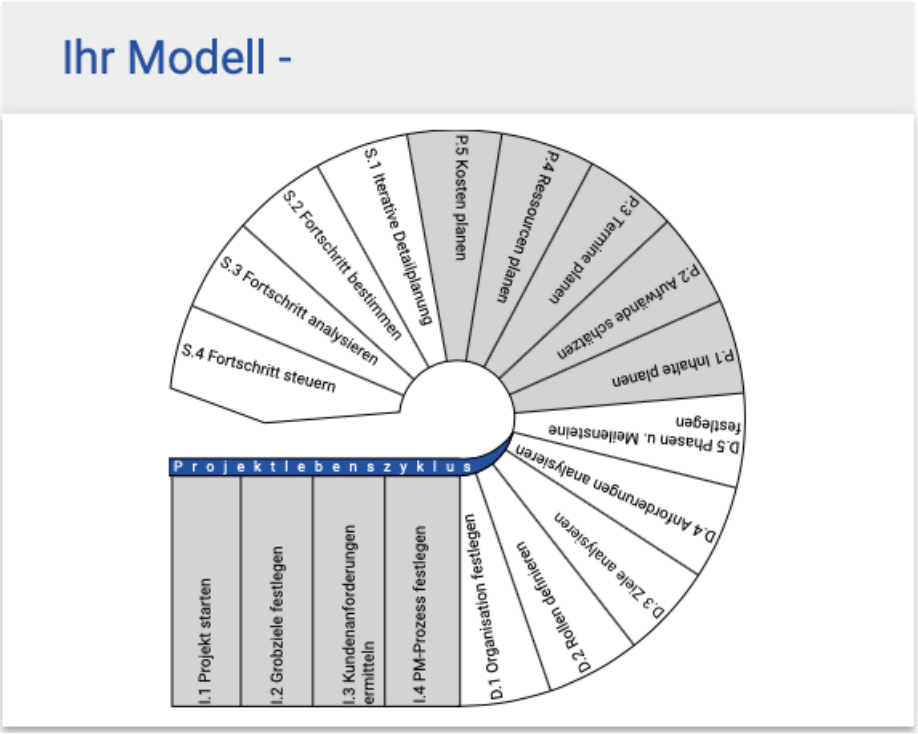

Abbildung 4.10: ModellComponent nach dem Starten der Anwendung

Abbildung 4.11 illustriert eine mögliche Konfiguration der Parameter durch den Benutzer und stellt anhand von konfigurierten Parameter das adaptive Vorgehensmodell dar.

Abbildung 4.12 veranschaulicht eine weitere Konfiguration der Parameter und das entsprechende adaptive Vorgehensmodell.

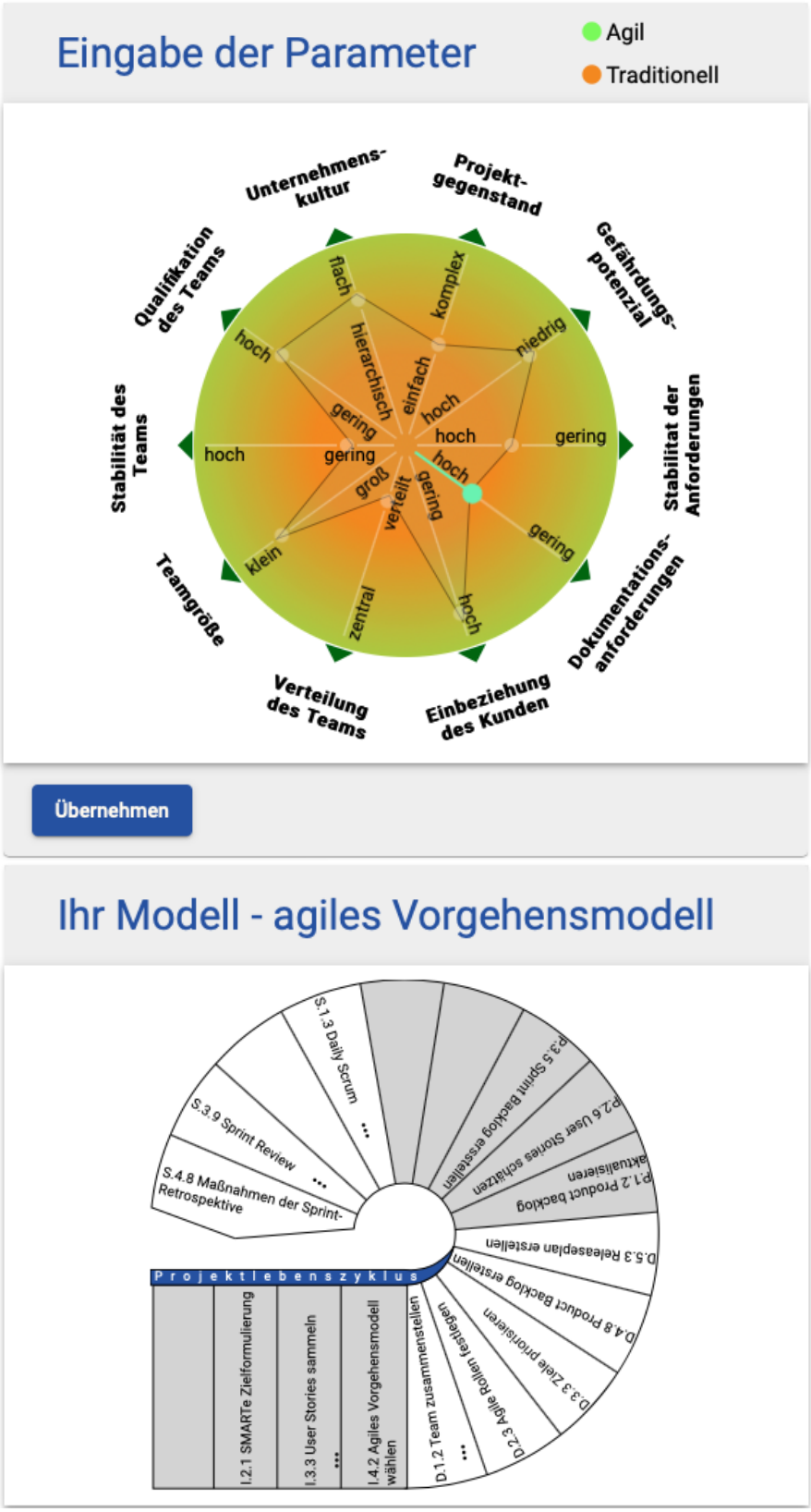

Abbildung 4.11: Eine mögliche Konfiguration der Parameter und das entsprechende adaptive Vorgehensmodell

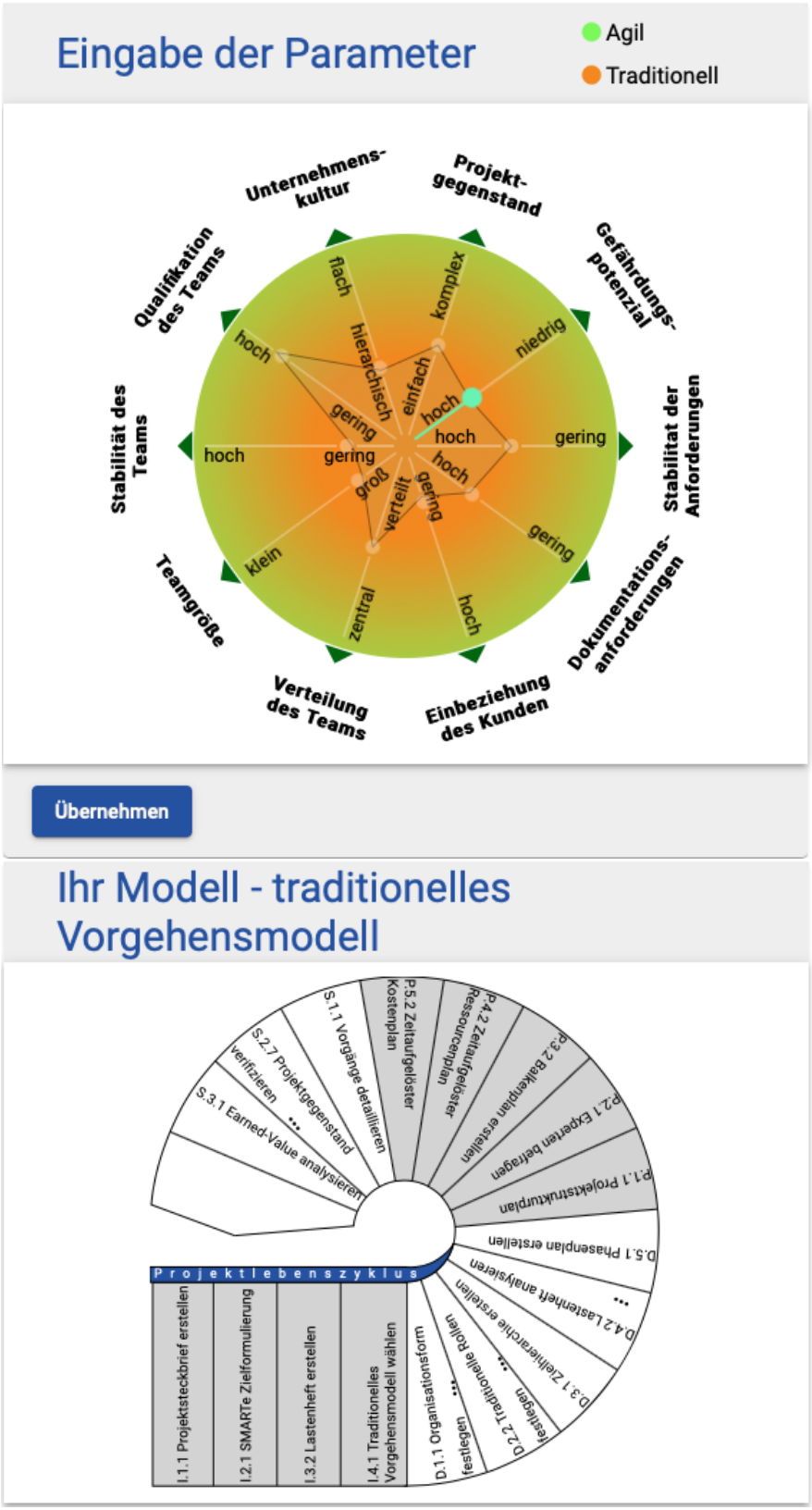

Abbildung 4.12: Eine weitere Konfiguration der Parameter und das entsprechende adaptive Vorgehensmodell

5 Zusammenfassung und Ausblick

Das abschließende Kapitel fasst die Arbeit kurz zusammen und gibt einen Überblick über die behandelten Themen. Darüber hinaus gibt es einen Ausblick über bevorstehende Tätigkeiten, die die Schwachstellen der Software beheben und Erweiterungen an ihr vornehmen sollen.

5.1 Zusammenfassung

Ziel dieser Arbeit war es, einen Demonstrator zu entwickeln, der anhand von konfigurierten Adaptionsparametern die Auswahl eines individuellen, adaptiven Vorgehensmodells automatisieren soll. Ein solches Modell zu konstruieren, aus einer Vielzahl von Methoden und Werkzeugen die richtigen auszuwählen mit zugleich wachsenden Kombinationen an hybriden Vorgehensmodellen, ist schwierig. Um die Auswahl zu automatisieren, müssen Kriterien gefunden werden, die sowohl agilen als auch traditionellen Vorgehensmodellen genügen. Diese Kriterien – die Adaptionsparameter – wurden als Grundlage in der Arbeit beschrieben, ebenso wie der Ordnungsrahmen für adaptives hybrides Projektmanagement, der eine einheitliche Darstellung agiler, traditioneller und hybrider Vorgehensmodelle erlaubt.

Schritt für Schritt wurde erläutert, wie sich die Anwendung zusammenbauen lässt. Als Erstes wurden die aktuellen Technologien vorgestellt, mit denen sich eine Webanwendung effizient realisieren lässt und die Wahl des Frameworks Angular begründet. Des Weiteren wurden Angular und dessen Werkzeuge näher erläutert und aufbauend auf diesen Grundlagen die Architektur des Demonstrators mit Berücksichtigung der Anforderungen aufgebaut. Somit entstand das Grundgerüst für den

Demonstrator mit zwei voneinander unabhängigen Komponenten und einem Service. Zum Schluss wurde die Visualisierung der Daten ausführlich besprochen und die Implementierung der beiden Komponenten und des Service erklärt.

Resümierend lässt sich sagen, dass viele Anforderungen realisiert wurden. Die Anforderungen A1 und A3 wurden durch die Wahl des Frameworks Angular abgedeckt, das sowohl die plattformübergreifende Portierbarkeit als auch die komponentenbezogene Zusammensetzung ermöglicht, mit deren Hilfe sich die Architektur des Demonstrators modular aufbauen ließ. Die Anforderung A4 wurde durch die aussagekräftige Repräsentation der Daten und durch intuitive Handhabung der zu konfigurierenden Adaptionsparameter in Form von Slidern realisiert. Die Anforderungen A5 und A6 lassen sich ebenfalls abdecken, indem die Konfiguration der Adaptionsparameter durch Slider und das individuelle, adaptive Vorgehensmodell durch Ordnungsrahmen für hybrides Projektmanagement realisiert wurden. Die Anforderung A2 ließ sich abdecken, indem einerseits die Repräsentation der Daten an die Breite des Browserfensters skaliert wird, andererseits die Anwendung die unterschiedlichen Eingabemechanismen unterstützt.

5.2 Ausblick

Die Anwendung wurde zur Entwicklungszeit auf einem lokalen Server ausgeführt. Um die Anwendung einer breiten Öffentlichkeit zur Verfügung zu stellen, soll sie auf einen externen Webserver deployed werden. Das Angular-CLI besitzt den Befehl,

```
ng build --prod
```

mit dem ein für den Produktionsmodus optimiertes Build erstellt wird. Die Ausgabe des Befehls befindet sich im Ordner dist. Auf welchen Webservern und wie das Angular-CLI das Deployment unterstützt, ist unter angular.io (vgl. [20]) gut dokumentiert.

Aktuell basiert die Anwendung komplett auf JavaScript, das heißt, wenn bei dem Anwender JavaScript im Browser deaktiviert ist, wird eine leere Seite dargestellt. Dieses Problem lässt sich beheben, indem die Anwendung auf dem Server gerendert wird, wobei das Framework Angular das Vorgehen unterstützen kann.

Konfiguration der Adaptionsparameter Die Adaptionsparameter lassen sich durch einen Klick oder durch das Ziehen des Schiebereglers einstellen. Bei der Konfiguration der Parameter durch Klick oder Touch lässt sich das Polygon einwandfrei formen. Wenn die Parameter jedoch durch Ziehen eingestellt werden, passt sich das Polygon erst an, wenn der Benutzer die Aktion beendet hat. Benutzerfreundlicher wäre es, wenn sich die entsprechende Ecke des Polygons mit dem Knopf des konfigurierten Sliders bewegt.

Ordnungsrahmen Zurzeit lässt sich der Ordnungsrahmen samt Methoden einwandfrei darstellen. Es kann aber nicht gewährleistet werden, dass die Bezeichnungen der Methoden nicht aus den Elementen des Ordnungsrahmens herausragen. In der Arbeit wurde eine überschaubare Anzahl von Methoden verwendet, was dazu führt, dass jedes Element angepasst werden konnte. Daher sollte hier ein Konzept erarbeitet werden, wie eine Beschriftung durch ein jeweiliges Element begrenzt wird.

Das Tailoring hybrider Vorgehensmodelle Der Algorithmus, der in der Arbeit erarbeitet wurde, basiert auf der Grundlage der arithmetischen Mittel. Er dient größtenteils dazu, den Datenfluss der Anwendung sicherzustellen. Es soll auf jeden Fall ein Algorithmus implementiert werden, der die Adaptionsmechanismen verwendet, die am IPIM zurzeit entwickelt werden.

Listingverzeichnis

Abbildungs- und Tabellenverzeichnis

Literaturverzeichnis

[1] S. Whitaker. The benefits of tailoring: Making a project management methodology fit. PMI White Paper., 2014. [Online]. Available: https://www.pmi.org/learning/library/tailoring-benefits-project-management-methodology-11133 [Zugriff am 11.03.2020].

[2] H. Timinger. *Modernes Projektmanagement : Mit traditionellem, agilem und hybridem Vorgehen zum Erfolg.* Wiley-VCH Verlag GmbH & Co. KGaA, 2017. 1. Auflage.

[3] A. Komus und M. Kuberg. Status Quo Agile. GPM, 2017. [Online]. Available: https://www.gpm-ipma.de/fileadmin/user_upload/GPM/Know-How/Studie_Status_Quo_Agile_2017.pdf [Zugriff am 30.03.2020].

[4] C. Seel und H. Timinger. *Ein adaptives Vorgehensmodell für hybrides Projektmanagement. In: T. Barton et al.: Prozesse, Technologie, Anwendungen, Systeme und Management 2017: Angewandte Forschung in der Wirtschaftsinformatik : Tagungsband zur 30. AKWI-Jahrestagung vom 17.09.2017 bis 20.09.2017 an der Hochschule Aschaffenburg.* Hochschule Aschaffenburg; mana-Buch, Aschaffenburg, Heide, 2017. S.20-21.

[5] H. Timinger. Projektmanagement und Informationsmodelierung (IPIM). [Online]. Available: https://www.haw-landshut.de/kooperationen/institute/ipim.html. [Zugriff am 02.03.2020].

[6] H. Timinger und C. Seel. Ein Ordnungsrahmen für adaptives hybrides Projektmanagement. GPM, 2016. [Online]. Available: https://www.gpm-ipma.de/fileadmin/user_upload/Know-How/pmaktuell/2016_04/PMa_4_16_S55.pdf [Zugriff am 11.03.2020].

[7] R. Haberfelner et al. *Systems Engineering Grundlagen und Anwendung.* Orell Füssli Verlag AG, 2015. 13. aktualisierte Auflage.

[8] B. Boehm und R. Turner. *Balancing agility and discipline. A guide for the perplexed*. Boston: Addison-Wesley, 2008.

[9] M. Špundak. Mixed agile/traditional project management methodology-reality or illusion, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S187704281402196X [Zugriff am 24 .03.2020].

[10] M. Paukner et al. *Projektparameter für das Tailoring hybrider Projektmanagementvorgehensmodelle In: T. Barton et al: Angewandte Forschung in der Wirtschaftsinformatik 2018: Tagungsband zur 31. AKWI-Jahrestagung vom 09.09.2018 bis 12.09.2018 an der Hochschule für Angewandte Wissenschaften Hamburg*. mana-Buch, Heide, 2018. S. 166-176.

[11] Medi Madelen Gwosdz. Die top-10-frameworks und was tech-recruiter darüber wissen müssen, 2020. [Online]. Available: https://stackoverflow.blog/2020/02/18/die-top-10-frameworks-und-was-tech-recruiter-daruber-wissen-mussen/ [Zugriff am 05.04.2020].

[12] reactjs.org. [Online]. Available: https://reactjs.org/docs/getting-started.html [Zugriff am 06.04.2020].

[13] angular.io. [Online]. Available: https://angular.io/guide/architecture [Zugriff am 06.04.2020].

[14] vuejs.org. [Online]. Available: https://vuejs.org/v2/guide/index.html [Zugriff am 07.04.2020].

[15] nodejs.org. [Online]. Available: https://nodejs.org/en/about/ [Zugriff am 08.04.2020].

[16] npmjs.com. [Online]. Available: https://docs.npmjs.com/about-npm/ [Zugriff am 08.04.2020].

[17] angular.io. [Online]. Available: https://angular.io/guide/file-structure [Zugriff am 10.04.2020].

[18] Christoph Höller. *Angular: Das umfassende Handbuch*. Bonn : Rheinwerk Verlag, 2019. 2., aktualisierte und erweiterte Auflage. S.27.

[19] material.angular.io. [Online]. Available: https://material.angular.io/guide/getting-started [Zugriff am 10.05.2020].

[20] angular.io. [Online]. Available: https://angular.io/guide/deployment [Zugriff am 10.05.2020].