Softwareprojekt

Fachhochschule Bielefeld
Campus Minden
Bachelor of Computer Science
Prof. Dr. Jörg Brunsmann
Sommersemester 2024

Dieses Dokument beschreibt den Inhalt und Ablauf sowie Aufgabenstellungen und Formales der Lehrveranstaltung.

1 Lernziele der Lehrveranstaltung

- Kollaborative Softwareentwicklung (mit Github) im Team
- Softwareprojekt planen und durchführen durch Spezifikation, Implementierung und Demonstration eines Softwareprodukts
- Anwendung von Techniken der agilen Vorgehensmodelle (Scrum, Kanban-Board)
- Erweiterung der Kompetenzen durch Einarbeitung in Technologien, die noch nicht beherrscht werden

- Selbstständigkeit im Lernen, Verstehen, Planen, Organisieren, Kommunizieren, Präsentieren, Zusammenarbeit im Team

2 Organisation und Ablauf der Lehrveranstaltung

- Um die Credit Points für die Lehrveranstaltung zu erlangen, müssen im Team folgende Leistungen / Inkremente im Semesterverlauf bearbeitet werden:

- **Kick-Off**
  - Teamfindung
  - Ideenfindung für Softwareprodukt
- **Softwarespezifkation**
  - Anforderungsanalyse
  - Technologieauswahl
- **Proof of Concept / MVP**
  - Implementierung
  - Präsentation / Demonstration des Software-Prototypen
- **Abgabe und Projektpräsentation**
  - Abgabe der Ideen, Spezifikation, Vortragsfolien, Source-Code
  - Präsentation / Demonstration des vollständigen Softwareprodukts
  - Seminarvortrag
    - Technologien
    - Lessons Learned

- Die Modulprüfung ist eine Projektarbeit, die mit der Übergabe von Software und weiteren Artefakten abgelegt wird.
- Die Lehrveranstaltung findet sowohl in Präsenz als auch Online statt.
- Die Durchführungsform (digital, präsenz), die Termine und die Aufgabenbeschreibungen zu den einzelnen „Inkrementen“ (Ideen, Spezifikation, etc.) werden jeweils bei Abschluss eines vorhergehenden Inkrements bekannt gegeben.
- Weitere Ankündigungen geschehen im Semesterverlauf auch über das Etherpad „Informationen, Termine, Abläufe“ im ILIAS.
- Sehen Sie dort unaufgefordert und in regelmäßigen Abständen nach, um auf dem aktuellen Stand der Organisation zur Lehrveranstaltung zu bleiben.

2.1 Formales

Leistungen und Bewertungkritierien (Gewichtung in %)

- **Softwareproduktidee** (15%)
  - Innovation, Kreativität, Komplexität und Umfang der Funktionalität
- **Softwarespezifkation** (15%)
  - Vollständigkeit, Korrektheit, Umfang, Rechtschreibung und Sorgfalt der Spezifikation
- **Proof of Concept / MVP** (10%)
  - Umfang und Abgabetermin (je früher desto besser)
- **Projektpräsentation / Seminarvortrag** (25%)
  - Qualität der Software-Demo und Verständlichkeit der Projektpräsentation
  - Sprachliches und schriftliches Ausdrucksvermögen
- **Softwareprodukt** (35%)
  - Komplexität und Qualität der Software (z.B. automatische Tests)
  - Vollständigkeit der Funktionalität gemäß Spezifikation

Übergreifende Bewertungkritierien:

- **Individuell**
  - Eigeninitiative, Beitrag, Einsatz und Übernahme von Verantwortung
  - Selbstständigkeit und Problemlösungskompetenz
  - Individuelles Zeitmanagement
  - Neugier und Offenheit
  - Selbstlernkompetenz: eigenständig Informationen aneignen, analysieren und teilen
  - Teilnahme an den Lehrveranstaltungen
- **Team**
  - Teamarbeit, Zusammenarbeit, Kollaboration
  - Projektbezogenes Zeitmanagement
  - Arbeiten mit und nach Projektplan
  - Kontinuierlicher Fortschritt und strukturierte Vorgehensweise
  - Gleichverteilte Aufgabenverteilung und gegenseitiges Mentoring
  - Pflege der Projektdokumentation

Verhaltensmuster im Team

- Das Praktikum findet in Teamarbeit mit verteilten Verantwortlichkeiten statt
- Es werden individuelle Noten vergeben, das ganze Praktikumsteam kann jedoch auch dieselbe Note erhalten.
- Sollten Teammitglieder nicht mitarbeiten, so dass es zum Beispiel zu Verzögerungen oder Verärgerungen kommt, so sprechen Sie frühzeitig zunächst mit den jeweiligen Teammitgliedern offen über die Situation und suchen nach deren Lösung.
- Sollte sich danach keine Besserung einstellen, so suchen Sie das Gespräch mit mir.

2.2 Weiterer Ablauf / Nächste Schritte / TODOs

- Bilden Sie ein Team von x bis y Studenten (x, y siehe ILIAS-Etherpad)
- Github-Organisation/Repository (public- oder private) erstellen und github-Account 'brunsmann' einladen.
- Entwickeln Sie gemeinsam im Team eine Softwareproduktidee und erstellen Sie dafür Dateien im Repository
- Präsentieren Sie die Idee in einer Woche
- Falls Sie Fragen zu Ihrer anvisierten Produktidee haben oder falls Sie unsicher sind, ob Ihre Produktidee geeignet ist, senden Sie mir bitte eine E-Mail.

3 Aufgabe 1

Findung einer Softwareproduktidee und deren Präsentation

Finden Sie im Team eine Idee für ein marktfähiges Softwareprodukt und stellen Sie diese Idee so kurz wie möglich z.B. als Präsentation vor. Berücksichtigen und beschreiben Sie, wie Sie das Softwareprodukt monetarisieren können. Berücksichtigen Sie ebenfalls die Größe Ihres Teams, so dass das anvisierte Softwareprodukt nicht zu viel oder zu wenig Komplexität besitzt. Fokussieren Sie in dieser Aufgabe zunächst nicht auf die Technik, sondern nur auf die Funktionalität. Es bietet sich an, zu überlegen, welche Alltags-Probleme Sie selbst haben, um daraus eine Idee abzuleiten. Ihre Geschäftsidee kann ebenfalls „physikalische“ Elemente enthalten, die Sie im Rahmen dieser Lehrveranstaltung nicht realisieren können. So kann zum Beispiel zur Vernetzung und Mobilität von Senioren eine Kleinbusflotte enthalten, die für die zu implementierende Smartphone-App simuliert wird.

Das Thema des Softwareprodukts ist frei. Die Idee soll jedoch innovativ und kreativ sein sowie eine umfangreiche Funktionalität besitzen. Die Innovation und Kreativität bezieht sich auf die Idee und nicht auf die zu verwendete Technik. Prinzipiell abgelehnt werden Softwareideen für „Messaging“, „E-Mail“, „Chat“, „Koch/Backrezept-Verwaltung“, „Fitness-Manager“, „Tauschbörsen“, „Warenbestellung/lieferung“, „Einkaufslisten-Verwaltung“ und „Geld/Haushaltsplaner“, „Schul-/Hausarbeiten-Verwaltung“, „E-Learning-Plattform“, „Terminverwaltung“, „Pflanzen-Monitoring“. Beachten Sie, dass z.B. eine „Chat“-Funktionalität ein Bestandteil einer größeren Idee sein kann.

Technologisch kann das Softwareprodukt eine Webanwendung, mobile Anwendung, eingebettete Software oder Desktop-Software sein und in einer oder mehreren Programmiersprachen implementiert werden. Das Softwareprodukt muss eine graphische Benutzerschnittstelle besitzen sowie eine Datenbank verwenden. Mehrere „Frontends“ (web-basiert, mobile Endgeräte und/oder Sprachassistenten) für das Softwareprodukt sind möglich und wünschenswert, insesondere dann, wenn die Anzahl der Teammitglieder größer als 2 ist.

Sollten Sie für dieses Modul Ideen verwenden, die Sie bereits in einem anderen Modul oder in früheren Semestern erstellt haben, so wird die Projektarbeit mit „mangelhaft“ bewertet.

Als Hilfestellung für die Präsentation der Ideen folgen einige Fragen, die Sie bei der Präsentation Ihrer Idee thematisieren können/sollen. Beantworten Sie diese Fragen nicht einzeln, sondern erstellen Sie aus den Antworten einen Fließtext. Es ist möglich, dass für Ihre Idee nicht alle Fragen sinnvoll zu beantworten sind.

- **Allgemein**
  - Projektname: Wie heißt das Produkt?
  - Darstellung der Produktvision in Prosa (5-10 Sätze)
- **Ziele**
  - Was sind Ziele des Produkts/des Services?
  - Warum ist das Produkt sinnvoll?
  - Was macht dieses Produkt/der Service anders?
  - Was sind die Anwendungsbereiche?
  - Abgrenzung: Was ist das Softwareprodukt *nicht*?
  - Welche Alleinstellungsmerkmale hat das Produkt und was macht dieses Produkt anders?
  - Zu welchen anderen Produkten steht das Produkt in Konkurrenz?
  - Welche alternativen Lösungsideen existieren für den identifizierten Bedarf?
  - Anwendungsbereiche, Motivation, Umfang
  - Welche Marktanforderungen werden bedient?
  - ggfs. SWOT-Analyse
  - Monetarisierung: Wie kann mit dem Produkt Geld verdient werden?
- **Stakeholderanalyse**
  - Für wen ist das Produkt/der Service?
  - Welche Stakeholder sind betroffen und wie stehen diese zu der Projektidee?
  - Was sind die Bedürfnisse, die das Produkt befriedigt?
  - Was ist die Zielbenutzergruppe und deren Merkmale (Bildung, Erfahrung, Sachkenntnis)?
  - Warum sollte der Kunde dieses Produkt/den Service „kaufen“ oder nutzen?
- **Risikoanalyse**
  - Lohnt sich das Projekt?
  - Wie hoch sind Aufwand und erwarteter Nutzen und stehen diese in einem sinnvollen Verhältnis?
  - Verfügen Sie über die notwendigen Kompetenzen? (Umsetzbarkeit)
  - Welche Risiken und negativen Nebeneffekte sind zu erwarten?

4 Aufgabe 2

Spezifikation des Softwareprodukts

Nachdem die Idee für ein Softwareprodukt im ersten Schritt gefunden (und im „Lastenheft“ dargelegt) wurde, sollen Sie im weiteren Verlauf die ausgewählte Idee im Team organisatorisch und technisch (u.a. mit UML-Diagrammen) spezifizieren. Die Spezifikation ist keine Arbeitsbeschaffungsmaßnahme, sondern dient dazu, die anstehenden Arbeiten zu koordinieren und Sachverhalte so früh wie möglich vor der Implementierung abzuklären. Denn je später ein Fehler im Projektverlauf entdeckt wird, desto aufwändiger ist es, diesen Fehler zu beheben. In dem zu erstellenden „Pflichtenheft“ sollten mindestens folgende Aspekte des Softwareprodukts / Softwareprojekts beschrieben werden:

- **Softwareprodukt**
  - Funktionalität
    - User Stories / Use Cases
    - GUI-Mockups
  - Qualitätsanforderungen
    - Nicht-funktionale Anforderungen
- **Softwaresystem**
  - Architektur
    - Software- und Systemarchitektur
  - Schnittstellen
  - Abläufe, Zustände und Datenmodelle
- **Softwareprojekt**
  - Projekt-Organisation
    - Meilensteinplan / Projektplan
    - Verteilung der Verantwortlichkeiten auf die Teammitglieder

Erstellen Sie die technische Spezifikation der ausgewählten Produktidee im Detail in der Art und Weise, wie es die unten angefügte Schablone vorgibt. Es ist nicht ungewöhnlich, wenn Sie für Ihr konkretes Softwareprodukt nicht sämtliche Kapitel der Schablone ausfüllen. Die nicht ausgefüllten Bestandteile sind aus dem Dokument zu löschen. Des Weiteren kann es möglich sein, dass Sie es für die Beschreibung sinnvoll erachten, weitere Unterkapitel für Ihr Softwareprodukt hinzuzufügen.

Modellieren Sie Ihr Softwareprodukt so umfangreich, sorgfältig und detailliert wie möglich und markieren Sie Funktionalitäten, die Sie nicht implementieren werden, als „optional“. Zum Beispiel können die Abläufe zur Bezahlung (über Drittanbieter) spezifiziert werden, ohne dass diese im Detail tatsächlich implementiert werden.

Es ist nicht davon auszugehen, dass Sie Ihr Softwareprodukt zu Beginn des Projekts zu 100% komplett korrekt und vollständig spezifizieren. Die Spezifikation wird endgültig am Ende des Semesters abgegeben, so dass neue Erkenntnisse (z.B. kleine Änderungen am Datenbankmodell), die während der Implementierung auftauchen, in die Spezifikation einfließen können.

Überlegen Sie, wie Sie die Teile der Spezifikation zur Bearbeitung sinnvoll auf die Teammitglieder aufteilen, so dass die Arbeit so gleichmäßg wie möglich aufgeteilt wird. So kann beispielsweise der Verantwortliche für das mobile Frontend die entsprechenden GUI-Mockups erstellen und der Verantwortliche für das Web-Frontend erstellt die entsprechenden GUI-Mockups. Des Weiteren kann das Datenbankmodell vom Backend-Verantwortlichen und die API-Definition von den Frontend-Verantwortlichen erstellt werden.

Die folgende Schablone zur Spezifikation ist in Markdown dargestellt und kann als Text kopiert und dann ausgefüllt werden.

```
# Anforderungs- und Entwurfsspezifikation ("Pflichtenheft")

# 0 Titelseite

* Projektname
* Autoren (Vor- und Nachnamen der beteiligten Studierenden)
* Link zum Source Code Repository

* Inhaltsverzeichnis

# 1 Einführung

## 1.1 Beschreibung

* Text der Produktidee einfügen

# 2 Anforderungen

## 2.1 Stakeholder

| Funktion / Relevanz | Name | Kontakt / Verfügbarkeit | Wissen  | Interessen / Ziele  |
|---|---|---|---|---|
|  |  |  |  |  |

### Beispiel

| Funktion / Relevanz | Name | Kontakt / Verfügbarkeit | Wissen  | Interessen / Ziele  |
|---|---|---|---|---|
| Leiter der Bibliothek, Fachlicher Entscheider  |  Herr Bauer | Tel. 409000, Von 9-19 Uhr telefonisch erreichbar, Mitarbeit zu 30% möglich, Nürnberg  | Kennt das Altsystem aus
```

der Anwendersicht, soll mit dem System arbeiten | Vereinfachung der Ausleihprozesse |
| Administrator, Informationslieferant bzgl. Wartungsanforderungen | Herr Heiner |
Heiner@gmx.net, Per E-Mail, immer erreichbar, Verfügbarkeit 50%, Nürnberg | Vertraut mit
vergleichbarer Verwaltungssoftware | Stabiles System, geringer Wartungsaufwand |
| Product-Owner, Entscheider - als Koordinator der Stakeholderanforderungen | Paul
Ottmer | po@ottmer.de, Per E-Mail und tel. tagsüber, Verfügbarkeit 100%, Nürnberg |
Koordinator für die Inputs der Stakeholder | ROI des Systems sicherstellen |

2.2 Funktionale Anforderungen

* ggfs. Use-Case Diagramme
* Strukturierung der Diagramme in funktionale Gruppen
* Definition der Akteure
* Akteure sowie andere Begriffe der implementierten Fachdomäne definieren
* Begriffe konsistent in der Spezifikation verwenden
* Begriffe im Glossar am Ende des Dokuments darstellen

2.3 Nicht-funktionale Anforderungen

2.3.1 Rahmenbedingungen

* Normen, Standards, Protokolle, Hardware, externe Vorgaben

2.3.2 Betriebsbedingungen

* Vorgaben des Kunden (z.B. Web Browser / Betriebssystem Versionen, Programmiersprache)

2.3.3 Qualitätsmerkmale

* Externe Qualitätsanforderungen (z.B. Performance, Sicherheit, Zuverlässigkeit,
Benutzerfreundlichkeit)

Qualitätsmerkmal	sehr gut	gut	normal	nicht relevant
Zuverlässigkeit				
Fehlertoleranz |X|-|-|-|
Wiederherstellbarkeit |X|-|-|-|
Ordnungsmäßigkeit |X|-|-|-|
Richtigkeit |X|-|-|-|
Konformität |-|X|-|-|
Benutzerfreundlichkeit				
Installierbarkeit |-|-|X|-|
Verständlichkeit |X|-|-|-|
Erlernbarkeit |-|X|-|-|
Bedienbarkeit |-|X|-|-|
Performance				
Zeitverhalten |-|-|X|-|
Effizienz|-|-|-|X|

Sicherheit | | | | |
Analysierbarkeit |X|-|-|-|
Modifizierbarkeit |-|-|-|X|
Stabilität |X|-|-|-|
Prüfbarkeit |X|-|-|-|

2.4 Graphische Benutzerschnittstelle

* GUI-Mockups passend zu User Stories
* Screens mit Überschrift kennzeichnen, die im Inhaltsverzeichnis zu sehen ist
* Unter den Screens darstellen (bzw. verlinken), welche User Stories mit dem Screen abgehandelt werden
* Modellierung der Navigation zwischen den Screens der GUI-Mockups als Zustandsdiagramm
* Mockups für unterschiedliche Akteure
* Mockups für unterschiedliche Frontends (Mobil, Web, Desktop)

2.5 Anforderungen im Detail

* User Stories mit Akzeptanzkritierien
* Optional: Name (oder ID) und Priorität ("Must", "Should", "Could", "Won't")
* Strukturierung der User Stories in funktionale Gruppen
* Sicherheit: Misuse-Stories formulieren

Schablone für User Stories

Als	**möchte ich**	**so dass**	**Akzeptanz**
Wer	Was	Warum	Wann akzeptiert

Beispiel 1

Als	**möchte ich**	**so dass**	**Akzeptanz**
Benutzer	bei Fehleingabe die Lösung angezeigt bekommen	ich lernen kann	Lösung wird angezeigt

Beispiel 2

Name	**In meiner Rolle als**...	...**möchte ich**...	..., **so dass**...	**Erfüllt, wenn**...	**Priorität**
Lernen	Benutzer	bei Fehleingabe die Lösung angezeigt bekommen	ich lernen kann	Lösung wird angezeigt	Muss

3 Technische Beschreibung

3.1 Systemübersicht

* Systemarchitekturdiagramm ("Box-And-Arrow" Diagramm)
* Kommunikationsprotokolle, Datenformate

Das Diagramm in Kapitel "Systemübersicht" ist statisch und nicht dynamisch und stellt daher keine Abläufe dar. Abläufe werden im Kapitel "Abläufe" dargestellt. Im Kapitel "Systemübersicht" soll genau ein Diagramm dargstellt werden. Das "Box-and-Arrow"-Diagramm soll als Systemarchitekturdiagramm eine abstrakte Übersicht über das Softwaresystem geben. Dazu stellt es die Rechnerknoten und deren Kommunikationsbeziehungen (Protokoll (z.B. HTTP), Datenformat (z.B. JSON)) dar. Also Rechtecke und gerichtete Pfeile. Ähnlich einem UML-Deployment-Diagramm, aber noch abstrakter, denn es zeigt nicht die Verteilung der Softwarebausteine auf die Rechnerknoten. So erlangt der Leser einen schnellen und guten Überblick über das Softwaresystem.

3.2 Softwarearchitektur

* Darstellung von Softwarebausteinen (Module, Schichten, Komponenten)

Hier stellen Sie die Verteilung der Softwarebausteine auf die Rechnerknoten dar. Das ist die Softwarearchitektur. Zum Beispiel Javascript-Software auf dem Client und Java-Software auf dem Server. In der Regel wird die Software dabei sowohl auf dem Client als auch auf dem Server in Schichten dargestellt.

* Server
  * Web-Schicht
  * Logik-Schicht
  * Persistenz-Schicht

* Client
  * View-Schicht
  * Logik-Schicht
  * Kommunikation-Schicht

Die Abhängigkeit ist bei diesen Schichten immer unidirektional von "oben" nach "unten". Die Softwarearchitektur aus Kapitel "Softwarearchitektur" ist demnach detaillierter als die Systemübersicht aus dem Kapitel "Systemübersicht". Die Schichten können entweder als Ganzes als ein Softwarebaustein angesehen werden. In der Regel werden die Schichten aber noch weiter detailliert und in Softwarebausteine aufgeteilt.

3.2.1 Technologieauswahl

Beschreiben Sie hier, welche Frameworks / Technologien / Bibliotheken / Datenformate / Protokolle benutzt werden.

3.3 Schnittstellen

* Schnittstellenbeschreibung (API)
* Auflistung der nach außen sichtbaren Schnittstelle der Softwarebausteine

Hier sollen sämtliche Schnittstellen definiert werden:

* die externen Schnittstellen nach außen. Über welche Schnittstelle kann z.B. der Client den Server erreichen?
* die internen Schnittstellen der unter 3.2 definierten Softwarebausteine

Es ist sinnvoll, wenn die API von denjenigen definiert werden, die die Anforderungen an die API kennen: in einem Client-Server-System haben die Client-Entwickler Anforderungen an die Backend-Entwickler, so dass in diesem Fall die Client-Entwickler die API definieren sollten, die dann vom Backend-Entwickler implementiert werden.

3.3.1 Ereignisse

* In Event-gesteuerten Systemen: Definition der Ereignisse und deren Attribute

3.4 Datenmodell

* Konzeptionelles Analyseklassendiagramm (logische Darstellung der Konzepte der Anwendungsdomäne)
* Modellierung des physikalischen Datenmodells
  * RDBMS: ER-Diagramm bzw. Dokumentenorientiert: JSON-Schema

3.5 Abläufe

* Aktivitätsdiagramme für relevante Use Cases
* Aktivitätsdiagramm für den Ablauf sämtlicher Use Cases
* Aktivitätsdiagramm mit Swimlanes sind in der Regel hilfreich
  für die Darstellung der Interaktion von Akteuren der Use Cases / User Stories
* Abläufe der Kommunikation von Rechnerknoten (z.B. Client/Server)
  in einem Sequenz- oder Aktivitätsdiagramm darstellen
* Modellieren Sie des weiteren die Diagramme, die für das (eigene) Verständnis des
  Softwaresystems hilfreich sind.

3.6 Entwurf

* Detaillierte UML-Diagramme für relevante Softwarebausteine

3.7 Fehlerbehandlung

* Mögliche Fehler / Exceptions auflisten
* Fehlercodes / IDs sind hilfreich
* Nicht nur Fehler technischer Art ("Datenbankserver nicht erreichbar") definieren, sondern auch im Hinblick auf
  Kapitel 3.8 sind fachliche Fehler wie "Kunde nicht gefunden". "Nachricht wurde bereits gelöscht" o.ä.

3.8 Validierung

* Relevante (Integrations)-Testfälle, die aus den Use Cases abgeleitet werden können
* Testfälle für
  * Datenmodell
  * API
  * User Interface
* Fokussieren Sie mehr auf Integrationstestfälle als auf Unittests
* Es bietet sich an, die IDs der Use Cases / User Stories mit den Testfällen zu verbinden, so dass erkennbar ist, ob Sie alle Use Cases getestet haben.

4 Projektorganisation

4.1 Annahmen

* Nicht durch den Kunden definierte spezifische Annahmen, Anforderungen und Abhängigkeiten
* Verwendete Technologien (Programmiersprache, Frameworks, etc.)
* Aufteilung in Repositories gemäß Software- und Systemarchitektur und Softwarebausteinen
* Einschränkungen, Betriebsbedingungen und Faktoren, die die Entwicklung beeinflussen (Betriebssysteme, Entwicklungsumgebung)
* Interne Qualitätsanforderungen (z.B. Softwarequalitätsmerkmale wie z.B. Erweiterbarkeit)

4.2 Verantwortlichkeiten

* Zuordnung von Personen zu Softwarebausteinen aus Kapitel "Systemübersicht" und "Softwarearchitektur"
* Rollendefinition und Zuordnung

Softwarebaustein	Person(en)
Komponente A	Thomas Mustermann

Rollen

Überlegen Sie, ob es sinnvoll ist, wenn Sie die Rollen für Product-Owner und Scrum-Master vergeben, wobei Sie bedenken sollten, ob diese Rollen über den gesamten Projektzeitraum aktiv sein werden. Neben diesen Rollen können folgende Rollen sinnvoll sein:

Softwarearchitekt

Entwirft den Aufbau von Softwaresystemen und trifft Entscheidungen über das Zusammenspiel der Softwarebausteine.

Frontend-Entwickler

Entwickelt graphische oder andere Benutzerschnittstellen, insbesondere das Layout einer

Anwendung.

Backend-Entwickler

Implementiert die funktionale Logik der Anwendung. Hierbei werden zudem diverse Datenquellen und externe Dienste integriert und für die Anwendung bereitgestellt.

DevOps-Engineer

Ist für die Repositories und das Deployment verantwortlich.

Rollenzuordnung

Name	Rolle
Thomas Mustermann	Frontend-Entwickler

4.3 Grober Projektplan

- Meilensteine

Meilensteine
* KW 43 (21.10)
  * Abgabe Pflichtenheft
* KW 44 (28.10) / Projekt aufsetzen
  * Repository Struktur
* KW 45 (4.11) / Implementierung
  * Implementierung #3 (Final)
* KW 48 (18.12) / Abnahmetests
  * manuelle Abnahmetests
  * Präsentation / Software-Demo

5 Anhänge

5.1 Glossar

* Definitionen, Abkürzungen, Begriffe

5.2 Referenzen

* Handbücher, Gesetze
* z.B. Datenschutzgrundverordnung

5.3 Index