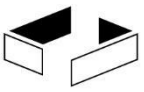


HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Fakultät

Ingenieurwissenschaften und Informatik

HAUSARBEIT IM FACH

OBJEKTORIENTIERTE ANALYSE

UND DESIGN

Akku Vita

Autoren: Steffen Herweg

873475

Benno Steinkamp

855624

Joana Wegener

855518

Abgabedatum: 18.2.2020

Inhaltsverzeichnis

1 Einleitung

Im Rahmen der Projektarbeit für das Modul "Objektorientierte Analyse und Design Wintersemester 20/21" der Hochschule Osnabrück wurde überlegt, wie ein Thema gewählt werden kann welches sowohl den zugrunde liegenden Anforderungen (vgl. [@Afo]) gerecht wird, als auch allgemein interessant ist.

Gleichzeitig wurde das Projektmitglied Joana Wegener in ihrer beruflichen Umgebung “Enviado” auf ein Problem aufmerksam gemacht, welches einer Lösung durch eine Software benötigte. Da dieses Projekt dieselben Anforderungen stellte, wie das Hochschulprojekt und durch den realen Bezug sehr interessant ist fiel die Themenwahl auf das angesprochene Projekt. Durch die Kooperation erhält die Hausarbeit unter anderem reale Stakeholder und klar definierte Ziele.

Enviado ist ein auf Lieferdienste fokussiertes Unternehmen, welches auf Service spezialisierte Produkte entwickelt, wie zum Beispiel Liefer-E-Bikes, Thermoboxen und Rucksäcke.
Die Firma vermietet unter anderem die für die E-Bikes benötigten Akkus mit denen wir uns im Rahmen der Projektarbeit beschäftigen.

Das Hauptziel dieser Projektarbeit ist die Erstellung eines Lebenslaufes für die Akkus, sodass etwaige Reparaturen oder Vermietungen nachvollziehbar sind.

Das Projekt wird den Namen "Akku Vita" tragen.

1.1 Aufgabenstellung

Bei enviado werden die Akkus für die E-Bikes vermietet. Diese können bei Defekten kostenfrei zurückgeschickt werden. Die defekten Akkus werden bei enviado geprüft und bei Bedarf von einem Reparateur repariert.

Derzeit gibt es keine Übersicht über die Zustände der Akkus. Dies ist problematisch, denn es ist nicht bekannt wie oft ein Akku repariert wurde ,wie lange er sich bei der Reparatur befand oder welcher Kunde den Akku benutzt hat.

Allgemein hat man keinen Zugriff auf den Verlauf eines Akkus und kann nur abschätzen wie gut dieser noch brauchbar ist.

Daraus resultiert Erstens, dass es nicht möglich ist eine finanzielle Übersicht zu erstellen.

Zweitens kann die Qualität der Akkus nicht bestimmt werden, was dazu führen kann das man sie mehrmals zur Reparatur schickt ohne, dass der Fehler behoben beziehungsweise erkannt wird.

Hinzu kommt, dass die Firma keine Übersicht darüber hat ob Kunden auffällig oft den selben Defekt für Akkus melden.

Das Ziel ist es also eine Übersicht über die Akkus zu schaffen, welche den Lebenslauf mit den dazugehörigen Events für jeden Akku darstellt.

Dies soll über eine Software ablaufen welche die einzelnen Abschnitte wie “Akku hinzufügen”, “Akku Event hinzufügen” und “Akku Löschen" in einer Datenbank dokumentiert und darüber hinaus auch einsehbar macht.. Das bedeutet konkret, dass auf die Datenbank sowohl geschrieben als auch Ausgelesen werden soll.

Zu dokumentierende Daten sind unter anderem die Akku-ID, die Kundennummer und die Häufigkeit der Reparaturen.

Das Projekt soll als WebApp entwickelt werden.

1.2 Aufbau der Hausarbeit

Bei dem Aufbau der Hausarbeit wurde das Projekt in verschiedene Phasen aufgeteilt. Begonnen wurde mit der Anforderungsanalyse woraus das Grobdesign resultierte.

Durch eine erneute Analyse und Absprache entstand daraufhin die Modellierung mit der zugehörigen Implementierung.

Nach der Implementierung wurde das Projekt validiert.

Die Phasen des Projekts orientieren sich an den Phasen, welche in dem Buch “Grundkurs Software-Engineering” (vgl. [Kle18]) behandelt werden. Die Dokumentation orientiert sich ebenfalls an den einzelnen Phase

2 Grundlagen

Für die Verbindung zwischen Web-Oberfläche und Geschäftslogik wurde die Requestfactory des Google Web Toolkits verwendet. Das selbe Toolkit wurde auch für den Aufbau Web-Oberfläche verwendet.

Als Datenbank wurde sich für HSQLDB entschieden, sowie Hibernate als dazugehörigen Objektrelationalen Mapper.

3 Anforderungsanalyse

Die erste Phase eines erfolgreichen Software-Entwicklungsprozess ist die Anforderungsanalyse, die in diesem Abschnitt der Dokumentation beschrieben wird.
Dazu wurde eng mit enviado zusammengearbeitet und in einem Vorgespräch zunächst mit einem Servicemitarbeiter, einem Lagermitarbeiter und der Geschäftsführung die Probleme des aktuellen Prozesses, sowie die Anforderungen aus Firmensicht besprochen. Mit Hilfe dieser Informationen wurden die Schritte der Anforderungsanalyse durchlaufen.

Diese Schritte werden in den zu diesem Kapitel gehörenden Abschnitten erörtert.

3.1 Prozessdiagramm vorher und nachher

In dem Vorgespräch mit enviado ergab sich der aktuelle Stand, welcher danach in einem Prozessdiagramm dargestellt wurde (siehe Anhang 1.1).

Wenn momentan ein neuer oder ein reparierter Akku geliefert wird, wird dieser eingelagert. Dabei erfolgt durch das Warenwirtschaftssystem eine Einbuchung, wodurch der Bestand angepasst wird. Beim versenden des Akkus an einen Kunden wird erneut der Bestand angepasst und zusätzlich die Seriennummer des Akkus erfasst.
Für die Rückgabe eines Akkus kann der Kunde zwei Gründe haben. Erstens, der Kunde hat sein Abonnement gekündigt und gibt den Mietakku zurück oder Zweitens, der Akku ist defekt und wird deshalb ersetzt. Dies ist in dem aktuellen Prozess noch nicht relevant, wird aber Teil des angepassten Prozesses sein.
Hier werden nach der Rückgabe alle Akkus von einem Lager- oder Servicemitarbeiter auf Defekte geprüft. Ein nicht defekter Akku wird erneut eingelagert. Ein defekter Akku hingegen wird zur Reparatur geschickt und kommt nach einer unbestimmten Zeit repariert zurück.

Aus diesem Prozess kann man leicht die Probleme ablesen, denn der Prozess kann in der Theorie nicht so funktionieren, denn die Akkus verlassen den Kreislauf des versendens und reparierens offiziell nie. Das erklärt den fehlenden Endpunkt in dem Prozessdiagramm.

Konkret ist dieses natürlich nicht umsetzbar, da es ab einem gewissen Punkt nicht mehr wirtschaftlich ist die Akkus zu reparieren. Deshalb wird momentan von einem Lager- oder Servicemitarbeiter ohne eine datenbasierte Grundlage oder einen geregelten Ablauf entschieden, wann ein Akku aussortiert wird. Die in dieser Dokumentation beschriebene WebApp soll dieses Problem lösen.

Zur Verdeutlichung der Verbesserung durch die WebApp und zur Findung der Anforderungen wurde nach dem Vorgespräch ein soll-Prozess erstellt, welcher den verbesserten, Ablauf darstellt.

In Anhang 1.2 kann man den durch die Software verbesserten Prozess sehen. Der Prozess beginnt genauso, wie der Aktuelle. Eine Änderung wird erst ersichtlich, wenn der Zustand des Akkus geprüft wird. Dort ist nun auch festgelegt, dass ein Lagermitarbeiter das Prüfen übernimmt. Im Zuge dessen kann er die Akku Vita ergänzen und so die Prüfung, sowie den Kunden dokumentieren.

Nun gibt es zwei verschiedene Wege, die der Akku gehen kann. Ist der Akku nicht defekt, wird er wieder eingelagert.

Wenn die Prüfung allerdings ergeben hat, dass der Akku defekt ist, übernimmt nun ein Servicemitarbeiter und überprüft die Akku Vita. Das genaue Entscheidungsverfahren wird intern von enviado ermittelt und nimmt keinen Einfluss auf unseren Prozess, allerdings werden dafür folgende Daten benötigt: Anzahl der Reparaturen, Alter des Akkus, Art der vorherigen Defekte. In diesem Schritt kann der Servicemitarbeiter auch den Kunden betrachten und überblicken, ob sich bestimmte Defekte häufen.

Nun entscheidet der Servicemitarbeiter, ob der Akku reparabel ist oder nicht. Ein irreparabler Akku wird aussortiert. Auch dort wird die Akku Vita ergänzt und der Prozess für den Akku beendet. Ein reparabler Akku wird zur Reparatur geschickt. Bei dem erneuten einliefern des Akkus wird die Akku Vita ergänzt und der Akku wird eingelagert.

In einem Detailgespräch wurde der Prozess abgesegnet.

3.2 Stakeholder

Nach dem Vorgespräch wurden die Stakeholder anhand der Checkliste zum Finden von Stakeholdern (vgl. [Kle18]: 56-60) herausgearbeitet, sodass sich folgende Tabelle ergibt:

Stakeholder	konkrete Personen
Endanwender	Geschäftsführung enviado, Lagermitarbeiter, Servicemitarbeiter

Management des Auftragnehmers	Steffen Herweg, Benno Steinkamp, Joana Wegener
Käufer des Systems	Geschäftsführung enviado, Hs Osnabrück
Prüfer	IT, -Serviceabteilung enviado, Prof. Kleuker
Entwickler	Steffen Herweg, Benno Steinkamp, Joana Wegener
Wartung- und Servicepersonal	IT-Abteilung enviado
Schulungs- und Trainingspersonal	IT-Abteilung enviado
Standard und Gesetze	DSGVO

3.3 Zielbeschreibung

Für die Beschreibung der Ziele wurde die “Schablone zur Dokumentation von Projektzielen” verwendet (vgl. [Kle18]: 62-64). Daraus ergeben sich folgende Tabellen:

Zielbeschreibung:	1. Die Software muss die Vita eines Akkus erstellen / ergänzen können
Stakeholder:	Lagermitarbeiter, IT-Abteilung enviado
Auswirkungen auf Stakeholder:	**Lagermitarbeiter:** Änderung des Prozesses zur Einbuchung von Retoure-Akkus **Lagermitarbeiter:** Änderung des Prozesses zur Einbuchung von reparierten Akkus **IT-Mitarbeiter:** muss Lagermitarbeiter schulen und auf Fragen eingehen können
Randbedingungen:	- Der Akku wird über die Seriennummer identifiziert
Abhängigkeiten:	-

Sonstiges:	- Akku Vita wird vor und nach der Reparatur gepflegt

Zielbeschreibung:	2. Die Akku Vita muss ausgelesen werden können
Stakeholder:	Servicemitarbeiter, IT-Abteilung enviado, Geschäftsführung enviado
Auswirkungen auf Stakeholder:	**Servicemitarbeiter:** kann begründet entscheiden, ob Akkus aussortiert werden sollen **Servicemitarbeiter:** kann Kunden auf großen Akkuverschleiß aufmerksam machen **Geschäftsführung enviado:** kann sich einen allgemeinen Überblick der Kosten-Nutzen Verteilung einzelner Akkus anschauen **IT-Mitarbeiter**: muss Lagermitarbeiter schulen und auf Fragen eingehen können **Lagermitarbeiter:** Änderung des Prozesses zur Einbuchung von Retouren-Akkus, da die Aussortierung durch einen Servicemitarbeiter erfolgt
Randbedingungen:	- Der Akku wird über die Seriennummer identifiziert
Abhängigkeiten:	- abhängig von Ziel 1, da die Akku-Vita zunächst erstellt werden muss und gepflegt werden muss um sinnvolle Informationen auslesen zu können
Sonstiges:	- Die gespeicherten Informationen sollen nach verschiedenen Faktoren gefiltert werden (Seriennummer, Kunden ID, Art des Defektes, Datum)

3.4 Use Cases

Aufgrund der Informationen aus dem Vorgespräch wurden schrittweise die Use-Cases (siehe Abb. 1) erstellt.
Dabei wurden zunächst die Basisinformationen herausgearbeitet. Diese sind hier:

1. Daten zur Akku Vita hinzufügen
2. Daten aus Akku Vita herauslesen

Danach wurden die Verantwortlichen Stakeholder hinzugefügt. In diesem Fall sollen sich die Lagermitarbeiter komplett um die Pflege der Akku Vita Daten kümmern. Die Aufgabe der Servicemitarbeiter ist es die Daten auszulesen und entsprechend zu handeln. Dies wird in dem bereits vorgestellten Prozessdiagramm, welches den Prozess mit Hilfe der Webapp darstellt (siehe Anhang 1.2) sehr deutlich.

Nachdem die Stakeholder bestimmt wurden, wurden die Basisinformationen in Anforderungen umgewandelt.

1. Akku Vita erstellen
2. Akku Vita erweitern
3. Akku Vita auslesen
4. Akku aussortieren

Use Cases zur reinen Pflege sind in diesem Projekt nicht benötigt und so konnte das Use-Case Diagramm erstellt werden.

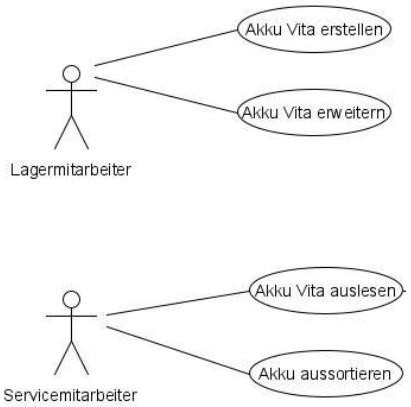


Abb. 1 - Use Case

Infolgedessen wurden mit Hilfe der “Schablone zur Dokumentation von Use Cases” (vgl. [Kle18]: 67) eine Dokumentation der Use Cases erstellt. Diese ist im Anhang 2 zu finden.

3.5 Formulierung von funktionalen Anforderungen

Die funktionalen Anforderungen wurden mit Hilfe der “Anforderungsschablone nach Rupp” (vgl. [Kle18]: 77) ausformuliert.

1. Wenn ein Akku das erste mal an einen Kunden versendet wird, muss das System dem Lagermitarbeiter die Möglichkeit bieten die Akku Vita für den Akku zu erstellen.
2. Wenn ein Akku von einem Kunden zurückgeschickt wird und im Lager bearbeitet wird, muss das System dem Lagermitarbeiter die Möglichkeit bieten die Akku Vita des Akkus zu ergänzen.
3. Wenn ein Akku von der Reparatur zurückgeschickt wird und im Lager bearbeitet wird, muss das System dem Lagermitarbeiter die Möglichkeit bieten die Akku Vita des Akkus zu ergänzen.
4. Wenn der Akku nach dem prüfen von einem Lagermitarbeiter als defekt eingeordnet wird, muss das System dem Servicemitarbeiter die Möglichkeit bieten die Akku Vita auszulesen.

5. Wenn der Servicemitarbeiter die Akku Vita ausgelesen hat, muss das System dem Servicemitarbeiter die Möglichkeit geben die Akku Vita zu ergänzen und den Akku als irreparabel zu markieren.

4 Modellierung

In der Phase der Modellierung wurde inkrementell ein UML-Klassendiagramm entwickelt, welche die Packages “domain.entity” und “shared” abbildet. In “shared” befinden sich zudem die Packages “proxy” und “service”.

In “domain-entity” befinden sich die Klassen, die nur von dem Server benutzt werden können und in “shared” befinden sich die Klassen, welche sowohl von dem Server als auch dem Client benutzt werden können.

Das Klassendiagramm worauf sich in diesem Kapitel bezogen wird, ist im Anhang 3.

4.1 Package “domain.entity”

In diesem Package befinden sich die Entitäten, auf die nur der Server Zugriff hat. Die drei Hauptentitäten sind “Akku”, “Kunde” und “AkkuEvent”. Zusätzlich gibt es die AkkuEvent-Arten “AkkuPruefungEvent”, “AkkuReparaturEingangsEvent”, “AkkuReparaturAusgangsEvent” und “AkkuAusmusterungsEvent”, welche jeweils von “AkkuEvent” erben. Somit besitzt jede Eventklasse die Objektvariablen “akku” und “datum”.

Zwischen “AkkuEvent” und “Akku” besteht eine bidirektionale Beziehung. Diese ist notwendig, da die Akkus ihre zugehörigen Events speichern müssen, damit für einen Akku alle Events ausgelesen werden können. Zusätzlich muss ein Event auch den zugehörigen Akku speichern, denn wenn das Event ein “AkkuPruefungsEvent” ist wird es im Kunden gespeichert werden. Wenn nun für einen Kunden alle Prüfungsevents ausgegeben werden sollen, ist die Speicherung der Akkus in dem Event notwendig, um dort die Verknüpfung zu dem Akku herzustellen.

Auch zwischen “Kunde” und “AkkuPruefungsEvent” besteht eine bidirektionale Beziehung aus dem gleichen Grund. Wenn für einen Kunden ein Prüfungsevent angeschaut wird, muss dort der zugehörige Akku angegeben werden und wenn für einen Akku ein Prüfungsevent angeschaut wird, muss dort der zugehörige Kunde angezeigt werden.

4.2 Package “shared”

Das Package “shared” beinhaltet die Packages “proxy” und “service”, sowie die Enumeration “AkkuDefekt”, die alle bekannten Defekte beinhaltet.

4.2.1 Package “proxy”

Das Package “proxy” beinhaltet die Stellvertreterklassen zu dem Klassen aus “domain.entity” als Interfaces.

4.2.2 Package “service”

Das Package “service” setzt das Request Factory Pattern um, auf welches weiter in der Implementierung eingegangen wird.

Außerdem werden in den Interfaces dieses Package Methoden zur persistenten Speicherung und suche bereitgestellt.

5 Implementierung

Die Implementierung des UML-Klassendiagramms sorgt für die Schnittstelle zwischen Front- und Backend.

5.1 Request Factory

Die Verbindung zwischen der Geschäftslogik und der Datenbank wird mithilfe der im Google Web Toolkit enthaltenen RequestFactory gelöst.

Diese ermöglicht es auf Entitäten auf der Serverseite über Stellvertreterobjekte, genannt "Proxy", zuzugreifen. Zu jeder Entität wird deshalb eine entsprechendes Proxy-Schnittstelle erzeugt. Diese müssen von der Klasse EntityProxy abgeleitet werden und werden mit der @ServiceFor(...)-Annotation gekennzeichnet:

```
@ProxyFor(Akku.class)
public interface AkkuProxy extends EntityProxy { … }
```

Alle Methoden, die nun der Client, verwenden soll, werden hier mit der gleichen Signatur definiert. Der Unterschied ist, dass alle Entitäten durch ihre entsprechenden Proxy-Schnittstellen ersetzt werden. So wird aus

```
public Set<AkkuEvent> getEvents() { … }
```

aus der Klasse Akku

```
Set<AkkuEventProxy> getEvents();
```

im Interface AkkuProxy. Hierbei ist AkkuEventProxy die Proxy-Schnittstelle zu AkkuEvent.

Außerdem muss zu jeder Klasse ein entsprechender Service zur Verfügung gestellt werden. Hierbei reicht es, wenn ein Service für eine Superklasse existiert. Dafür muss der entsprechende Proxytyp jedoch mithilfe der @ExtraTypes(...)-Annotation bekannt gemacht werden. Diese Service-Interfaces werden dann in einem RequestFactory Interface gebündelt der Anwendung zur Verfügung gestellt. Die Instanziierung der RequestFactory findet über die GWT.create(...) Methode statt.

```
public interface AkkuVitaRequestFactory extends RequestFactory {
```

```
AkkuRequest akkuRequest();
AkkuEventRequest akkuEventRequest();
KundeRequest kundeRequest();
}
```

5.2 Datenbank

Als Datenbank wird HSQLDB genutzt. Sie befindet sich im Verzeichnis projekt/db. In der Datei server.properties wird die Datenbank konfiguriert. Sie kann über die Datei StartDatenbank.bat ausgeführt werden.

5.3 Hibernate

Wie im Kapitel Grundlagen erwähnt, wird für das Mapping der Entitäten in die Datenbank der Objektrelationale Mapper Hibernate verwendet.

Damit die Hibernate Bibliothek die verwendeten Entitäten in der Datenbank persistieren kann, muss das Mapping zum Compilezeitpunkt feststehen. Dies kann entweder durch entsprechende .hbm.xml-Dateien oder durch Java-Annotationen und den entsprechenden Annotationsprozessor geschehen. In diesem Fall wurde sich für die zweite Möglichkeit entschieden.. Im folgenden Code-Snippet wird beispielhaft einen Teil der Implementation der Entität “Akku” in Akku.java gezeigt.

```
package de.enviado.akkuvita.domain.entity;

@Entity
@Table(name = "AKKU")
public class Akku implements Serializable {
  @Id
  @NotNull
  @Column(name = "SERIAL", unique = true)
  private String seriennummer;

  @NotNull
```

```
    @DecimalMin("0")
    @Column(name = "VERSION")
    private Integer version = 0;

    @Column(name = "PRODUCTION_DATE")
    private Date produktionsdatum;

    ...
}
```

Des Weiteren muss eine globale Konfigurationsdatei “hibernate.cfg.xml” erstellt werden. Diese teilt Hibernate mit, wie die Verbindung mit der Datenbank aufgenommen werden soll, sowie welche Klassen persistiert werden sollen. Zunächst die Konfiguration der Verbindungsdaten, die Auswahl des JDBC-Treibers und des Entsprechenden SQL-Dialekts:

```
!-- Database connection settings -->
<property name="connection.driver_class">org.hsqldb.jdbcDriver</property>
<property name="connection.url">jdbc:hsqldb:hsql://localhost/akkuvitadb</property>
<property name="connection.username">sa</property>
<property name="connection.password"/>
<!-- SQL dialect -->
<property name="dialect">org.hibernate.dialect.HSQLDialect</property>
```

Dann die Konfiguration der zu mappenden Klassen:

```
<mapping class="de.enviado.akkuvita.domain.entity.Akku"/>
<mapping class="de.enviado.akkuvita.domain.entity.AkkuEvent"/>
<mapping class="de.enviado.akkuvita.domain.entity.AkkuPruefungsEvent"/>
<mapping class="de.enviado.akkuvita.domain.entity.AkkuReparaturEingangsEvent"/>
<mapping class="de.enviado.akkuvita.domain.entity.AkkuReparaturAusgangsEvent"/>
<mapping class="de.enviado.akkuvita.domain.entity.Kunde"/>
```

Diese Datei muss sich zum Zeitpunkt der Ausführung im Klassenpfad der Anwendung befinden. Hier liegt sie im Wurzelverzeichnis der Java Quelltextdateien dieses Projektes (project/src/main/java).

Außerdem wurde eine serverseitige Hilfsklasse HibernateUtil zur leichteren Verwendung der konfigurierten Verbindung genutzt.

Die tatsächliche Persistenz der Entitäten wird in den einzelnen Klassen in der Methode persist() implementiert. Hier wird zunächst eine neue Sitzung und danach eine neue Transaktion erzeugt. Anschließend wird das Objekt mit der Methode saveOrUpdate(...) in die Datenbank gespeichert. Zum Schluss wird die Transaktion mit commit() beendet. Das Schließen der Sitzung geschieht automatisch am Ende des try-with-resources-Blocks, da die Session die Closable-Schnittstelle implementiert.

6 Validierung

Für eine erfolgreiche Validierung wurde jede der fünf funktionalen Anforderungen unter Realbedingungen getestet.

Die erste Anforderung wurde erfüllt. Dazu wurden 10 Akkus erstellt. Akkus ohne Seriennummer konnten nicht erstellt werden.

Auch die zweite und dritte funktionale Anforderung wurde erfüllt. Dazu wurde jedem der zuvor hinzugefügten Akkus ein Prüfungs-, Reparatureingangs-, oder Reparaturausgangsevent hinzugefügt. Auch das hinzufügen mehrerer Events zu einem Akku hat funktioniert.

Die vierte Anforderung wurde erfüllt. Dazu wurden die AkkuVita der bereits hinzugefügten Akkus ausgelese. Gesucht wurde der Akku mit seiner Seriennummer.

Die fünfte und letzte Anforderung ist die einzige, die nicht komplett erfüllt wurde. Es kann ein Ausmusterungsevent zu einem Akku hinzugefügt werden. Allerdings können dem Akku trotzdem noch weitere Events hinzugefügt werden. Es ist zwar gewünscht, dass ein Akku nach der Ausmusterung weiterhin in der Datenbank enthalten und sichtbar bleibt, aber es sollte nicht möglich sein weitere Events hinzuzufügen.

7 Abschlussbetrachtungen

Abschließend wurden die Anforderungen, die in der Anforderungsanalyse aufgestellt wurden mit der erreichten Implementation verglichen.

7.1 Zusammenfassung

Das Ziel des Projektes “Akku Vita”, welches im Rahmen einer Hausarbeit der Fachhochschule Osnabrück für das Modul “objektorientierte Analyse und Design” bearbeitet wurde, war es die Historie der Mietakkus mit Hilfe einer Software nachvollziehbar zu gestalten.

Ein Mietakku wird zunächst neu geliefert und kann danach an einen Kunden ausgeliehen werden. Falls der Akku defekt ist, kann der Kunde ihn kostenlos zurückschicken und bekommt im Austausch einen neuen Akku. Der defekte Akku wird nun überprüft und zur Reparatur gesendet. Sobald der Akku repariert zurückgeschickt wurde, kann er erneut ausgeliehen werden und der Prozess beginnt von vorne.

Ohne die Informationen über den Verlauf eines Akkus ist es unmöglich nachzuvollziehen, ob ein Akku wiederholt denselben Defekt aufweist. Daraus resultiert, dass eine datenbasierte Ausmusterung konstant defekter oder sehr alter Akkus nicht möglich ist und eine teure Reparatur öfter als nötig bezahlt wird.

Außerdem kann man nicht sehen, ob ein Kunde stets Akkus mit demselben Defekt zurückgibt.

Zur Implementierung einer Software, welche dieses Problem löst wurden die Schritte der Anforderungsanalyse durchlaufen, sodass nach der Erstellung eines effizienteren Arbeitsprozesses und der Bestimmung von Stakeholdern und Use Cases folgende, grundlegende funktionale Anforderungen herausgearbeitet wurden:

1. Das System muss dem Lagermitarbeiter die Möglichkeit bieten die Akku Vita zu ergänzen.
2. Das System muss dem Servicemitarbeiter die Möglichkeit bieten die Akku Vita auszulesen.

Es wurde sich für eine WebApp entschieden, die diese Funktionen implementiert.

Abschließend kann man sagen, dass die Implementation der Basisfunktionen gut gelungen ist, es allerdings weiterhin Bedarf zur Verbesserung der Software gibt.

7.2 Ausblick

Die implementierte Software stellt aktuell nur die Basisfunktionen zur Verfügung. Denkbar und praktisch wären allerdings noch weitere.

7.2.1 Anbindung des Warenwirtschaftssystem “plentymarkets”

Enviado benutzt momentan das Warenwirtschaftssystem “plentymarkets” für eine Vielzahl von Funktionen. Unter anderem wird bei dem Warenausgang der Bestand gepflegt. Hier wäre es denkbar in diesem Prozess für Mietakkus auch die Seriennummer zu erfassen, sodass die Akku Vita auch beim Ausgang an einen Kunden ergänzt wird. So kann man aus den Daten ablesen, wie lange ein Akku bei einem Kunden in Betrieb war.
Eine Ergänzung ohne die Anbindung des Warenwirtschaftssystems ist ineffizient, da das Ereignis zweimal an verschiedenen Stellen - einmal in der Akku Vita und einmal in dem Warenwirtschaftssystem - eingetragen werden müsste.

Wenn ein Akku von einem Kunden reklamiert wird, erstellt ein Servicemitarbeiter in dem Warenwirtschaftssystem ein Ticket, welches dort dem Akku zugeordnet ist und den Defekt anhand der Aussage des Kundens beschreibt. Die Zuordnung des Tickets zu der dazugehörigen Akku Vita wäre sinnvoll, damit ein Lagermitarbeiter zunächst den vermuteten Defekt überprüfen kann.

7.2.2 Aufbereitung der Daten

Ein denkbares Feature wäre die Aufbereitung aller Daten, sodass z.B. das Kosten-Nutzen Verhältnis von Akkus berechnet und bereitgestellt wird.

7.2.3 Erinnerungsfunktionen

Sinnvoll wäre es zusätzlich eine Erinnerungsfunktion einzufügen, sodass ein Mitarbeiter der Serviceabteilung eine Benachrichtigung bekommt, wenn ein Akku zum x-ten Mal defekt oder bereits x Jahre alt ist.

A Verantwortlichkeiten

Artefakt	verantwortlich	mitwirkend	
Kapitel 1, 6	Steffen Herweg	Joana Wegener	
Kapitel 2, 5	Benno Steinkamp	Joana Wegener	Steffen Herweg
Kapitel 3, 7, 4	Joana Wegener	Steffen Herweg	
Anforderungsanalyse	Joana Wegener	Steffen Herweg	
Modellierung	Benno Steinkamp	Steffen Herweg	Joana Wegener
Implementierung Backend	Benno Steinkamp	Joana Wegener	Steffen Herweg
Implementierung Frontend	Steffen herweg	Joana Wegener	Benno Steinkamp
Validierung	Steffen Herweg		
Organisation	Joana Wegener	Steffen Herweg	Benno Steinkamp

B Abgabe

Verzeichnis	Inhalt
/Hausarbeit	Dokumentation als .docx und .pdf
/Quellcode	Projekt “Akku Vita”
/Literatur	benutzte Webseiten
/Dokumente	UML-Diagramm

C Stundenzettel

<Die Abgabe ist verpflichtend, geht aber nicht in die Note ein. Die Werte werden u. a. für statistische Zwecke genutzt, um den realen Aufwand von Studierenden zu ermitteln. Die hier gezeigte Tabelle ist recht grob und könnte thematisch weiter untergliedert werden. Man beachte, dass echte Arbeitsstunden anzugeben sind, dazu gehören z. B. Zeiten zur Anreise und Abreise nicht.>

Name	Stunden
Steffen Herweg	38h
Benno Steinkamp	42 h
Joana Wegener	41 h

D Erklärung

Hiermit versichern wir, dass wir diese Hausarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben.

Datum: 17.02.21 

Benno Steinkamp

Datum: 17.02.21 

Steffen Herweg

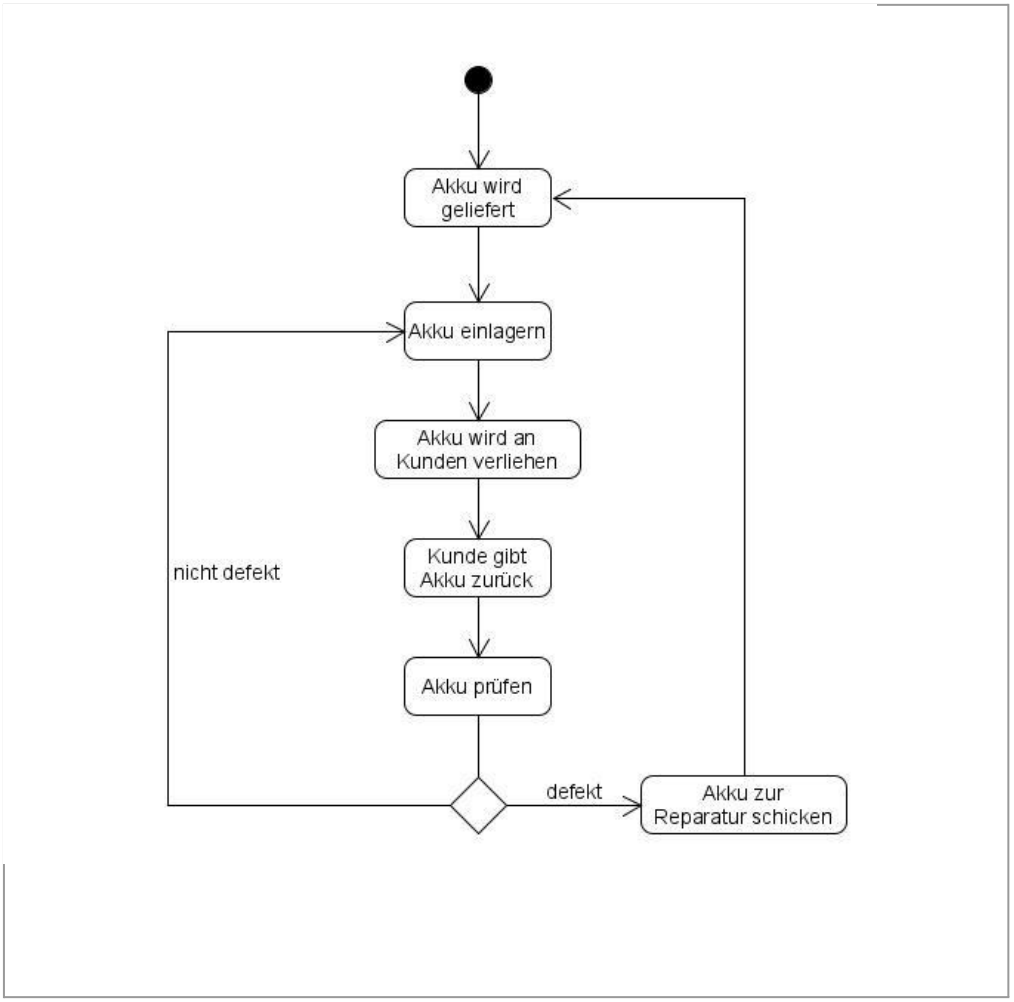
Datum: 17.02.21 Joana Wegener

E Literaturverzeichnis

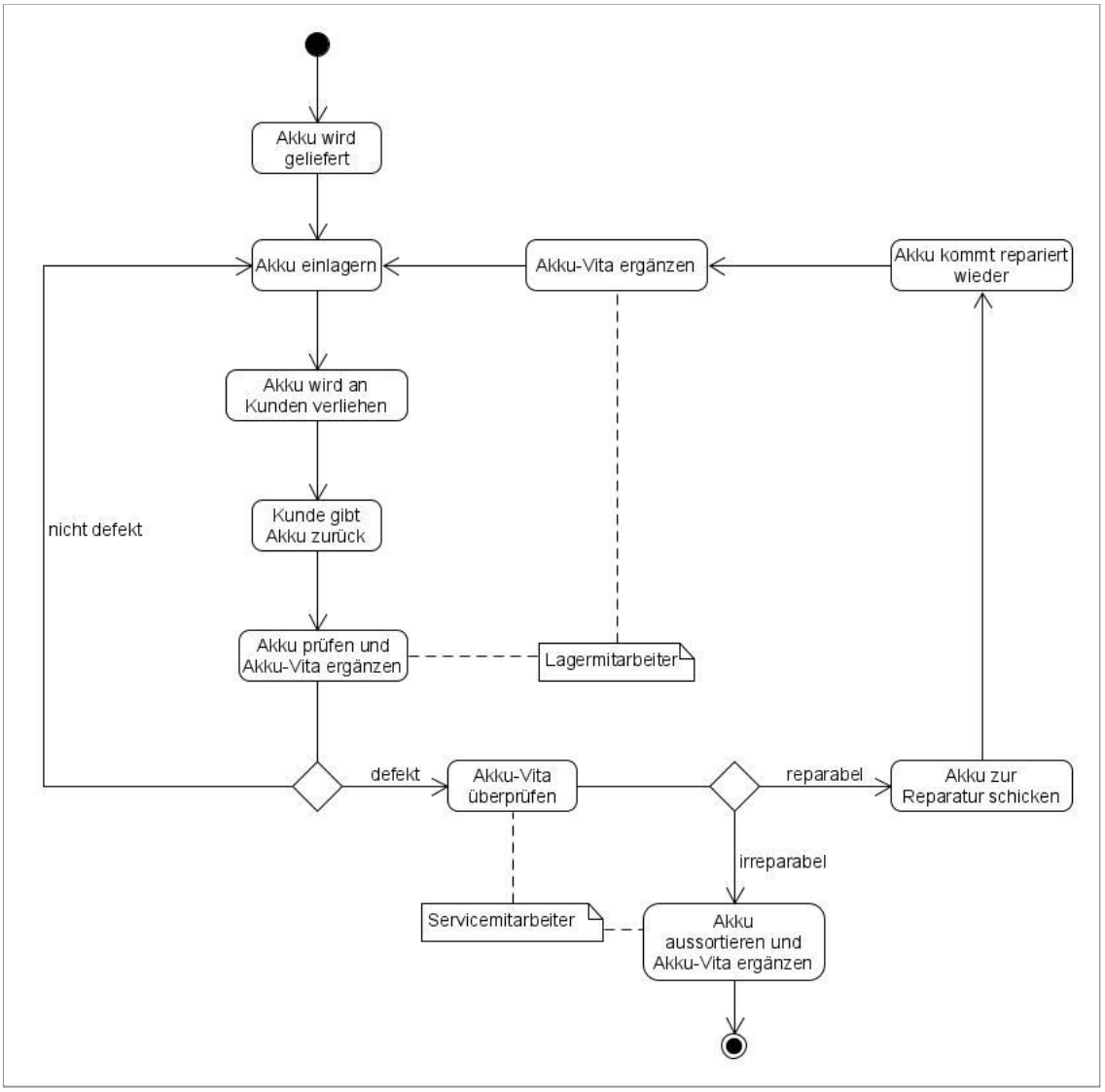
Weblinks zuletzt abgerufen am 17.02.2021.

[@Afo]	S. Kleuker, Anforderungen an Abschlussaufgaben, http://home.edvsz.hs-osnabrueck.de/skleuker/querschnittlich/AnforderungenAbschlussarbeit.pdf
[Kle18]	S. Kleuker, Grundkurs Software-Engineering mit UML, 4. aktualisierte Auflage, Springer Vieweg, Wiesbaden, 2018
[@gwt]	GWT Dokumentation, http://www.gwtproject.org/doc/latest/DevGuide.html
[@jgwt]	GWT Java Dokumentation, http://www.gwtproject.org/javadoc/latest/
[@mgwt]	Maven Plugin für GWT Dokumentation, https://tbroyer.github.io/gwt-maven-plugin/
[@db]	HSQLDB http://hsqldb.org/
[@rm]	Hibernate relational mapper https://hibernate.org/orm/

F Anhang



Anhang 1.1, Prozess vorher



Anhang 1.2, Prozess nacher

Anhang 2: Dokumentationen von Use-Cases

Name des Use Case	Akku Vita erstellen
Nummer	U1
Autor	Steffen Herweg
Paket	-
Kurzbeschreibung	Mitarbeiter des Lagers haben die Möglichkeit neue Akku Vita einträge anzulegen.
beteiligte Aktoren	Lagerarbeiter
Fachverantwortliche	Leitung der Lagerabteilung
Referenzen	Akkudaten
Vorbedingungen	Die Software ist vollständig installiert und wurde gestartet.
Nachbedingungen	Neue Akkus und Akkus welche noch nicht eingetragen wurden.
typischer Ablauf	1. Nutzer wählt Funktionalität zur erstellung von Akku Vita aus 2. Nutzer legt neue Akku Vita an 3. Nutzer fügt die Daten des Akkus ein 4. Nutzer verlässt die Funktionalität
alternative Abläufe	Nutzer kann existierendes Akkus auswählen. Nutzer kann Daten eines existierenden Akkus ändern.

Kritikalität	sehr hoch, System macht ohne Funktionalität keinen Sinn
Verknüpfungen	Nötig für die “Akku Vita auslesen” und “Akku aussortieren” Usecases
funktionale Anforderungen	Funktionale Anforderung 1,2,3

Name des Use Case	Akku Vita auslesen
Nummer	U2
Autor	Steffen Herweg
Paket	-
Kurzbeschreibung	Mitarbeiter des Service haben die Möglichkeit Akku Vita auszulesen
beteiligte Aktoren	Servicemitarbeiter
Fachverantwortliche	Leitung der Servicemitarbeiter
Referenzen	-
Vorbedingungen	Die Software ist vollständig installiert und wurde gestartet.
Nachbedingungen	Akku ID des auszulesenden Akkus
typischer Ablauf	1. Nutzer wählt Funktion zur auslesung von Akku Vita aus 2. Nutzer Sucht über die ID des Akkus die zugehörigen Daten 3. Butzer liest die Daten aus 4. Nutzer verlässt die Funktion
alternative Abläufe	-
Kritikalität	Hoch, für die Firma ist das einsehen der Daten sehr relevant
Verknüpfungen	Ist abhängig von dem “Akku Vita erstellen” Usecase
funktionale Anforderungen	Funktionale Anforderung 4

Name des Use Case	Akku Vita aussortieren
Nummer	U3
Autor	Steffen Herweg
Paket	-
Kurzbeschreibung	Mitarbeiter des Service haben die Möglichkeit Akkus zu entfernen
beteiligte Aktoren	Servicemitarbeiter
Fachverantwortliche	Leitung der Servicemitarbeiter
Referenzen	-
Vorbedingungen	Die Software ist vollständig installiert und wurde gestartet.
Nachbedingungen	Akku ID des zu entfernenden Akkus
typischer Ablauf	1. Nutzer wählt die Funktion zur aussortierung von Akku aus 2. Nutzer sucht über ID den Akku der aussortiert werden soll 3. Nutzer sortiert den Akku aus 4. Nutzer verlässt die Funktion
alternative Abläufe	-
Kritikalität	Niedrig, System funktioniert auch ohne die Funktionalität
Verknüpfungen	Ist abhängig von dem “Akku Vita erstellen” Usecase

funktionale Anforderungen	Funktionale Anforderung 5

Anhang 3: UML-Klassendiagramm