### 1.1. Produkt-Vision

Vision Statement: Unsere Parkhaussimulation soll übersichtlich und benutzerfreundlich gestaltet sein. Die Simulation soll unterschiedliche Parkplätze, Preismodelle und Zähler für die Parkplätze bieten.

|  |  |  |
| --- | --- | --- |
| Zielgruppe | Parkhausbetreiber | Kunden |
| Bedarf | Hohe Automatisierung;  Übersichtliches Management (Wer war wann wo?);  Einnahmen (betriebswirtschaftlich relevante Statistiken) | Frauenparkplätze (breitere Parkplätze);  familiengerechter Parkplätze (breitere Parkplätze);  Behindertenparkplätze (nah am Ausgang);  Anzeige wv Parkplätze frei sind;  Kunde kann auch nachts parken;  Abo-Möglichkeit; |
| Produkt | Automatisches Abrechnungssystem,- Schranke, - Zähler/Sensor,  Übersichtliches Management: - Graphiken & Statistiken (Software wird Statistiken anzeigen durch Listen verschiedener Entitäten) | Kategorisierung von Parkplätzen (Frauen/Familien/Behinderten Parkplätze)  Parkplatzzähler  Verringerte Parkkosten nachts(automatisch)  Automatische Erkennung von Abo |
| Mehrwert | Deutlich geringere Personalkosten  Geringerer Arbeitsaufwand durch Digitalisierung  Höhere Kundenzufriedenheit - häufigere Nutzung und somit höhere Einnahmen |  |

### 1.2. User Stories

~~US01: Als Autofahrer kann ich kontaktlos bezahlen, um Zeit zu sparen.~~

US02: Als Betreiber kann ich einfach Tages-, Wochen- und Monatseinkommen angezeigt bekommen, um mir einen Buchhalter zu sparen (unnötiges Personal zu entlassen, um Geld zu sparen).

~~(US03: Als Abonnent des Parkhauses kann ich mein Monatsabo online buchen, um flexibel zu sein.)~~

~~US04: Als Autofahrer möchte ich über “die App” angezeigt bekommen, wo ich am günstigsten parken kann, um Kosten zu sparen.~~

~~US05: Als Autofahrer würde ich gerne einen Parkplatz im Voraus buchen können, um meinen Alltag besser planen zu können.~~

US06: Als Autofahrer würde ich gerne einen Kostenvoranschlag berechnet bekommen, um nicht von den Kosten überrascht zu werden.

~~US07: Als Autofahrer möchte ich automatisch bezahlen können ohne dabei an einen Automaten zu gehen, um Zeit zu sparen.~~

US08: Als Frau möchte ich Frauenparkplätze, um mich sicher zu fühlen.

US09: Als Familie möchte ich Familienparkplätze, um mit meinen Kindern besser parken zu können.

US10: Als Behindertet möchte ich einen Behindertenparkplatz, um besser parken zu können.

US11: Als Autofahrer möchte ich die Anzahl noch freier Parkplätze sehen könne, um zu entscheiden ob ich parke oder nicht.

US12: Als Autofahrer möchte ich auch nachts ins Parkhaus kommen, um parken zu können.

~~US13: Als Betreiber möchte ich vieles automatisieren, um Personalkosten zu sparen.~~

~~US15: Als Parkhausnutzer möchte ich eine Notification per SMS erhalten, die mich daran erinnert, dass meine (zuvor angegebene gewünschte) Parkdauer bald erreicht ist, um die Zeit nicht aus den Augen zu verlieren.~~

US16: Als E-Auto-Fahrer wünsche ich mir Ladestationen in Parkhäusern, um mein E-Auto laden zu können.

US17: Als Parkhausbetreiber möchte ich kleinere Parkplätze für Zweiräder, um Platz für mehr Parkplätze zu sparen.

US18: Als Autofahrer möchte ich wissen welcher Parkplatz frei ist, um nicht suchen zu müssen.

US19: Als Betreiber möchte ich sehen wann welches Fahrzeug wie lange in meinem Parkhaus parkt, um die Parksituation zu dokumentieren und zu analysieren.

~~US20: Als Autofahrer brauche ich eine Ansprechperson, um anfallenden Fragen geklärt zu bekommen.~~

### 1.3. Qualitätssicherung mit INVEST

Entfernt: US01 Schätzbarkeit nicht realisierbar, da zu aufwendig.

Geändert: US03 Als Abonnent des Parkhauses kann ich mein Monatsabo buchen, um flexibel zu sein (Aufwand für “online” buchen nicht abschätzbar)

Entfernt: US04 Weder klein, noch abschätzbar.

Entfernt: US05 Nicht abschätzbar, da man ein ganzes Buchungssystem obendrauf schreiben müsste.

Entfernt: US07 nicht abschätzbar und nicht klein genug.

Entfernt: US13 Nicht klein und abschätzbar.

Entfernt: US15 Nicht abschätzbar, da man ein ganzes Buchungssystem obendrauf schreiben müsste (Abhängig von US05).

Entfernt: US20 Keine Konsequenzen für Software.

### 1.4. Tasks für den ersten Scrum Sprint

Allgemein:

Git Repository einrichten

Recherche SEPP / Einzelne Komponenten

Server einrichten/Grundgerüst schreiben (das ganze Testen)

Parkhaus API recherchieren

Parkhaus API in Web Server einbinden

US02:

Aufbereitung von Parkdaten zur statistischen Analyse (Code)

Modul zur Generierung von Statistiken anhand von Aufbereiteten Daten (Code maybe Library)

US06

Modul zur Berechnung von Preisen anhand der Zeit und Klassifizierung (Client seitig)

Weboberfläche Gestalten die Alle Daten veranschaulicht

Verbindung Weboberfläche reale Daten

US08-US10

Implementierung verschiedener Parkpreise/ Unterscheidung von Parkplätzen

Datenstruktur zu Repräsentierung der Parkplätze

Datenstruktur zur Speicherung der Parkdaten

US11

Implementierung eines Zählers der die verfügbaren Parkplätzte für verschiedene Parkplätze festhält (Client seitig)

Abweisung von Autos die nicht mehr Parken können (Clientseitig)

US16 + US17

Änderung der Parkhaus API um E-Autos/Zweiräder zuzulassen (Vielleicht auch Eigene Typen unterstützt ,Recherche benötigt)

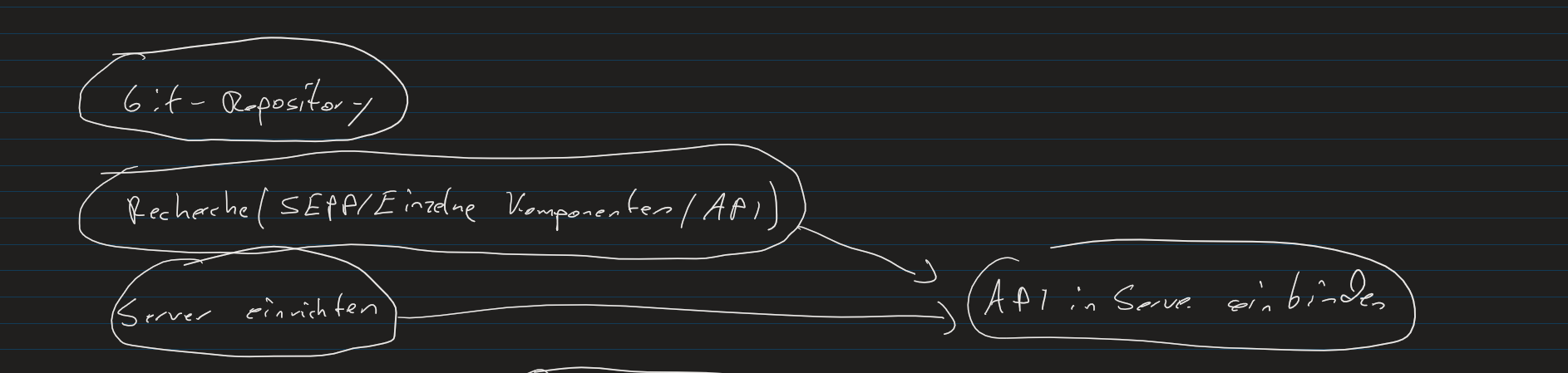
US18

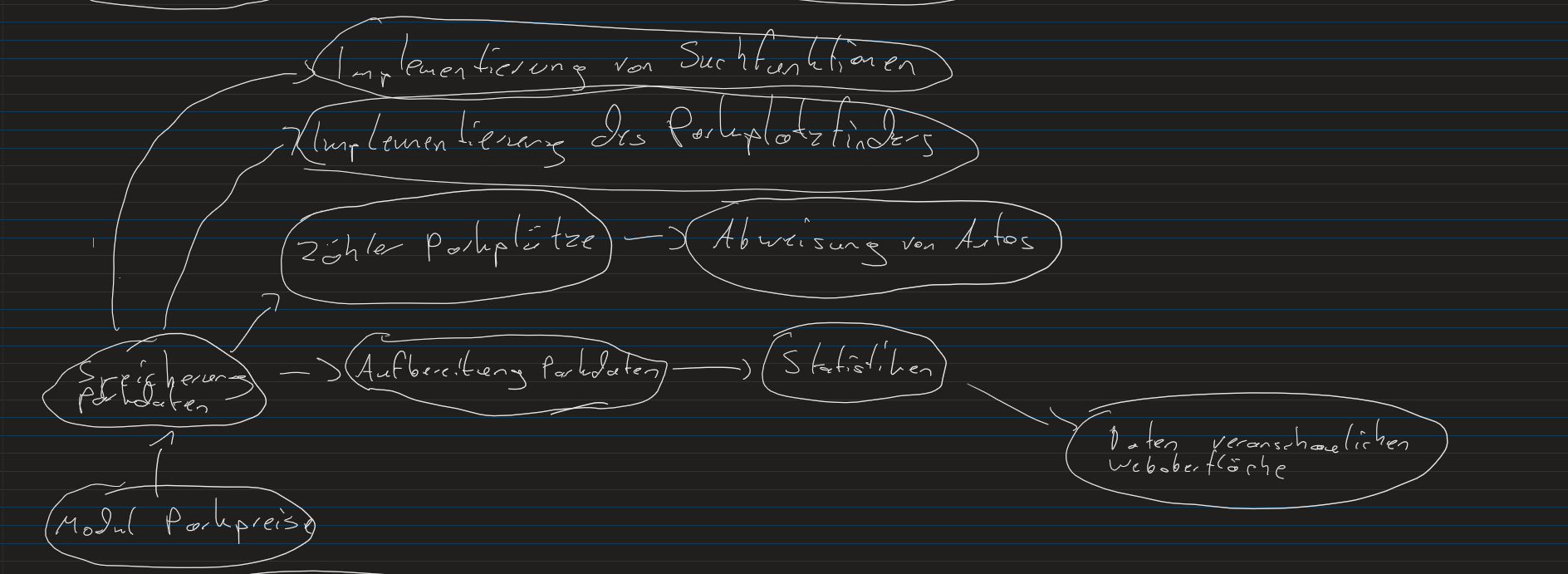
Implementierung eines Digitalen Zuweisers der Fahrzeug Parkplatz zurückgibt (Server seitig)

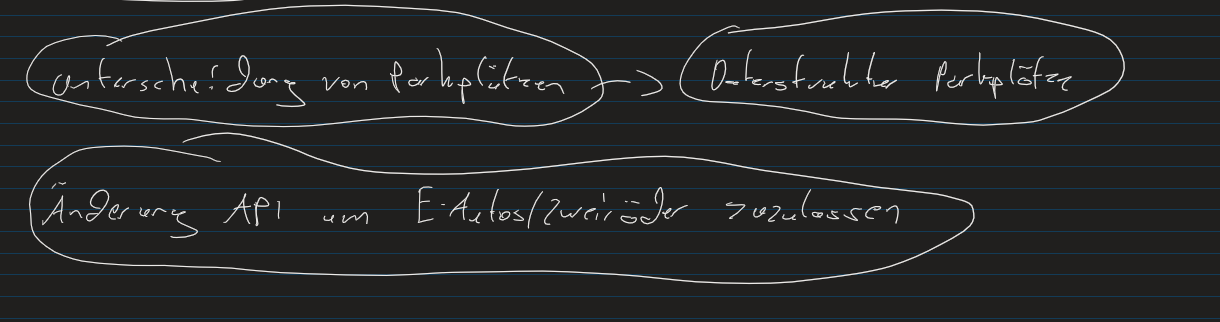
US19

Implementierung von Suchfunktionen auf den Parkdaten

### 1.5. Abhängigkeitsanalyse





1,2,3,5,8,13,21,34,55,89

1.6 Bewerten

|  |  |  |  |
| --- | --- | --- | --- |
| Story/Task | Nutzen | Aufwand | Nutzen/Aufwand |
| Git Repository einrichten | 89 | 1 | 89 |
| Server einrichten/Grundgerüst schreiben (das ganze Testen) | 89 | 2 | 44,5 |
| Änderung der Parkhaus API um E-Autos/Zweiräder zuzulassen (Vielleicht auch Eigene Typen unterstützt, Recherche benötigt | 8 | 5 | 1,6 |
| Recherche SEPP / Einzelne Komponenten | 89 | 3 | 26,66666 |
| Parkhaus API recherchieren | 89 | 3 | 26,66666 |
| Parkhaus API in Web Server einbinden | 89 | 1 | 89 |
| Aufbereitung von Parkdaten zur statistischen Analyse (Code) | 21 | 5 | 4,2 |
| Modul zur Generierung von Statistiken anhand von Aufbereiteten Daten (Code maybe Library) | 21 | 21 | 1 |
| Modul zur Berechnung von Preisen anhand der Zeit und Klassifizierung (Client seitig) | 34 | 3 | 11,333333 |
| Weboberfläche Gestalten die Alle Daten veranschaulicht | 34 | 8 | 4,25 |
| Verbindung Weboberfläche reale Daten | 34 | 8 | 4,25 |
| Implementierung verschiedener Parkpreise/ Unterscheidung von Parkplätzen | 13 | 2 | 6,5 |
| Datenstruktur zu Repräsentierung der Parkplätze | 34 | 2 | 17 |
| Datenstruktur zur Speicherung der Parkdaten | 34 | 2 | 17 |
| Implementierung eines Zählers der die verfügbaren Parkplätzte für verschiedene Parkplätze festhält (Client seitig) | 1 | 2 | 0,5 |
| Abweisung von Autos die nicht mehr Parken können (Clientseitig) | 1 | 3 | 0,333 |
| Implementierung eines Digitalen Zuweisers der Fahrzeug Parkplatz zurückgibt (Server seitig) | 21 | 3 | 7 |
| Implementierung von Suchfunktionen auf den Parkdaten | 8 | 5 | 1,6 |
| US02 | 21 | 21 | 1 |
| US06 | 34 | 89 | 0,382022 |
| US08 | 13 | 5 | 2,6 |
| US09 | 13 | 5 | 2,6 |
| US10 | 13 | 5 | 2,6 |
| US11 | 1 | 8 | 0,125 |
| US12 | 34 | 3 | 11,3333333 |
| US16 | 8 | 5 | 1,6 |
| US17 | 8 | 5 | 1,6 |
| US18 | 21 | 3 | 7 |
| US19 | 8 | 5 | 1,6 |

1.7 MVP

|  |  |  |  |
| --- | --- | --- | --- |
| Story/Task | Nutzen | Aufwand | Nutzen/Aufwand |
| Git Repository einrichten | 89 | 1 | 89 |
| Parkhaus API in Web Server einbinden | 89 | 1 | 89 |
| Server einrichten/Grundgerüst schreiben (das ganze Testen) | 89 | 2 | 44,5 |
| Recherche SEPP / Einzelne Komponenten | 89 | 3 | 26,66666 |
| Parkhaus API recherchieren | 89 | 3 | 26,66666 |
| Datenstruktur zu Repräsentierung der Parkplätze | 34 | 2 | 17 |
| Datenstruktur zur Speicherung der Parkdaten | 34 | 2 | 17 |
| Modul zur Berechnung von Preisen anhand der Zeit und Klassifizierung (Client seitig) | 34 | 3 | 11,333333 |
| US12 | 34 | 3 | 11,3333333 |
| Implementierung eines Digitalen Zuweisers der Fahrzeug Parkplatz zurückgibt (Server seitig) | 21 | 3 | 7 |
| US18 | 21 | 3 | 7 |
| Implementierung verschiedener Parkpreise/ Unterscheidung von Parkplätzen | 13 | 2 | 6,5 |
| Weboberfläche Gestalten die Alle Daten veranschaulicht | 34 | 8 | 4,25 |
| Verbindung Weboberfläche reale Daten | 34 | 8 | 4,25 |
|  |  |  |  |
| Aufbereitung von Parkdaten zur statistischen Analyse (Code) | 21 | 5 | 4,2 |
| US08 | 13 | 5 | 2,6 |
| US09 | 13 | 5 | 2,6 |
| US10 | 13 | 5 | 2,6 |
| Änderung der Parkhaus API um E-Autos/Zweiräder zuzulassen (Vielleicht auch Eigene Typen unterstützt, Recherche benötigt | 8 | 5 | 1,6 |
| US16 | 8 | 5 | 1,6 |
| US17 | 8 | 5 | 1,6 |
| Implementierung von Suchfunktionen auf den Parkdaten | 8 | 5 | 1,6 |
| US19 | 8 | 5 | 1,6 |
| Modul zur Generierung von Statistiken anhand von Aufbereiteten Daten (Code maybe Library) | 21 | 21 | 1 |
| US02 | 21 | 21 | 1 |
| Implementierung eines Zählers der die verfügbaren Parkplätzte für verschiedene Parkplätze festhält (Client seitig) | 1 | 2 | 0,5 |
| US06 | 34 | 89 | 0,382022 |
| Abweisung von Autos die nicht mehr Parken können (Clientseitig) | 1 | 3 | 0,333 |
| US11 | 1 | 8 | 0,125 |

### 1.8.1 Wissenschaftliche Recherche

#### *Nennen Sie die Besonderheiten von Software.*

* Software spielt in fast allen Bereichen des täglichen Lebens eine zentrale und tragende Rolle: Kommunikation, Verwaltung, Fertigung, Bank- und Finanzwesen, Bildung, Verkehr, Unterhaltung, Medizin, Landwirtschaft und Recht.
* Software hat die Eigenschaften der Abstraktheit, Komplexität, Unsichtbarkeit und Veränderbarkeit, die nicht leicht zu bewältigen sind.
* Software hat sowohl statische als auch dynamische Eigenschaften.
* Es gibt keine allgemeingültigen Qualitätsmaßstäbe für die Bewertung eines Softwareprodukts
* Software nutzt sich nicht ab
* Kommunikation, Koordination, Management und Design-Aktivitäten sind wichtig
* den Schwerpunkt auf die Nutzung unterschiedlicher Standpunkte zu legen, anstatt sie einzuschränken, ist wichtig
* Kommunikationsfähigkeit, Teamfähigkeit, Problemlösungsfähigkeit, Planungs- und Organisationsfähigkeit organisatorische Fähigkeiten, die Fähigkeit, Informationen zu beschaffen und zu verarbeiten sowie die Fähigkeit quantitative Daten zu analysieren, sind hilfreich bei der Software Entwicklung
* Entscheidungen auf der Grundlage quantitativer Daten sollten von SE getroffen werden, wenn dies angemessen und machbar ist
* Ethik: Verantwortungsbewusstes Handeln beim Entwickeln von Software ist Gebot

#### *Beschreiben Sie die Evolution von Software Engineering:*

* Das Wasserfallmodell war damals üblich (Struktur und Ordnung war sehr wichtig)
* Jede Phase musste abgeschlossen werden, bevor die nächste beginnen konnte (Anforderungsspezifikation, Entwurf, Implementierung, Prüfung und Dokumentation)
* Mit der sich verbreitenden Art flexibler Software Entwicklung zu organisieren, kam die Idee des Prototypings auf, bei der ein Prototyp dazu dienen kann, sowohl den Problem- als auch den Entwurfsraum zu erforschen. Ggf. könnte der Prototyp sich auch zum Endprodukt entwickeln.
* Um Kreislauf von Code & Fix zu umgehen, wurde das Spiralmodell entwickelt.
* Die Ausweitung des Spektrums von Softwareanwendungen, insbesondere in Wirtschaft und Handel, und das Aufkommen des Internets förderten in den 1990er Jahren die Einführung flexiblerer iterativer und inkrementeller Entwicklungsformen (Rapid Application Development)
* Es etablierte sich die Erkenntnis, dass der "Kunde" in die Entwicklung einbezogen werden muss und nicht nur ein Anbieter von Anforderungen ist, die zu Beginn eines Projekts festgelegt wurden.
* Das Tempo des Wandels in der Computer- und Softwaretechnologie steigert die Nachfrage nach neuen und weiterentwickelten Softwareprodukten.
* Diese Situation hat zu Kundenerwartungen und Wettbewerb geführt, welche die Fähigkeit zur Herstellung von Qualitätssoftware innerhalb akzeptabler Entwicklungszeiträume belastet.
* Das Angebot an qualifizierten Software-Ingenieuren hat mit der Nachfrage der Industrie nicht Schritt gehalten, so dass Systeme von Personen mit unzureichender Ausbildung oder Erfahrung entwickelt und gebaut werden.
* Agilität und Management nahm an Bedeutung zu.
* Während es früher auf Expertenerfahrungen und mathematische Modelle ankam, wurde im Laufe der Zeit durch praktische Erfahrungen und empirische Studien besser verstanden, wann und wie verschiedene Modelle funktionieren, und wie die Anpassung dieser Modelle an verschiedene Situationen und Bedingungen erfolgen muss.

### 1.8.2 Wissenschaftliche Recherche

#### *Was sind die grundlegenden Problemfelder des Software Engineerings?*

– Dekomposition

– Abstraktion

–Wiederverwendung

– Redundanz: vertikal/horizontal

– Abhängigkeiten: intern/extern, implizit/explizit

– Formalisierung

– Rahmenbedingungen: zeitlich, finanziell, rechtlich, technisch, personell

#### *Womit haben Software-Entwickler täglich zu kämpfen?*

Die Problemfelder decken das ab, womit Softwareentwickler täglich zu kämpfen haben:

* umfangreiche Aufgabenstellungen beherrschbar machen (Dekomposition, Abstraktion, Wiederverwendung)
* Lösungen dauerhaft konsistent und änderbar halten (Redundanz und Abhängigkeiten)
* Aufgaben für die automatisierte Verarbeitung erschließen (Formalisierung)
* Sachzwänge aus dem Projektumfeld einbeziehen (Rahmenbedingungen)

### 2.4. Semesterprojekt

US02: Als Betreiber kann ich einfach Tages-, Wochen- und Monatseinkommen angezeigt bekommen, um mir einen Buchhalter zu sparen (unnötiges Personal zu entlassen, um Geld zu sparen).

US03 Als Abonnent des Parkhauses kann ich mein Monatsabo buchen, um flexibel zu sein (Aufwand für “online” buchen nicht abschätzbar)

US06: Als Autofahrer würde ich gerne einen Kostenvoranschlag berechnet bekommen, um nicht von den Kosten überrascht zu werden.

US08: Als Frau möchte ich Frauenparkplätze, um mich sicher zu fühlen.

US09: Als Familie möchte ich Familienparkplätze, um mit meinen Kindern besser parken zu können.

US10: Als Behindertet möchte ich einen Behindertenparkplatz, um besser parken zu können.

US11: Als Autofahrer möchte ich die Anzahl noch freier Parkplätze sehen könne, um zu entscheiden ob ich parke oder nicht.

US12: Als Autofahrer möchte ich auch nachts ins Parkhaus kommen, um parken zu können.

US16: Als E-Auto-Fahrer wünsche ich mir Ladestationen in Parkhäusern, um mein E-Auto laden zu können.

US17: Als Parkhausbetreiber möchte ich kleinere Parkplätze für Zweiräder, um Platz für mehr Parkplätze zu sparen.

US18: Als Autofahrer möchte ich wissen welcher Parkplatz frei ist, um nicht suchen zu müssen.

US19: Als Betreiber möchte ich sehen wann welches Fahrzeug wie lange in meinem Parkhaus parkt, um die Parksituation zu dokumentieren und zu analysieren.

**Glossar**

* **Stats** Statistik zur Anzahl der verschiedenen Typen
* **Occupied** belegter Parkplatz
* **Leave** Fahrzeug verlässt einen Parkplatz
* **Enter** Fahrzeug “betritt”? Parkhaus
* **CarCounter** Fahrzeugzähler
* **Ticket** Parkschein
* Eindeutige Fahrzeugtypen?
* **Customer/ParkingGarageUser/ParkingGarageOwner** Kunde? Parkhausnutzer/Parkhausbesitzer
* **ParkingSpace** Parkplatz
* **Car** Fahrzeug
* …

**MVP überarbeitet:**

|  |  |  |  |
| --- | --- | --- | --- |
| Story/Task | Nutzen | Aufwand | Nutzen/Aufwand |
| Git Repository einrichten | 89 | 1 | 89 |
| Parkhaus API in Web Server einbinden | 89 | 1 | 89 |
| Server einrichten/Grundgerüst schreiben (das ganze Testen) | 89 | 2 | 44,5 |
| Recherche SEPP / Einzelne Komponenten | 89 | 3 | 26,66666 |
| Parkhaus API recherchieren | 89 | 3 | 26,66666 |
| Datenstruktur zu Repräsentierung der Parkplätze | 34 | 2 | 17 |
| Datenstruktur zur Speicherung der Parkdaten | 34 | 2 | 17 |
| Modul zur Berechnung von Preisen anhand der Zeit und Klassifizierung (Client seitig) | 34 | 3 | 11,333333 |
| US12 | 34 | 3 | 11,3333333 |
| Implementierung eines Digitalen Zuweisers der Fahrzeug Parkplatz zurückgibt (Server seitig) | 21 | 3 | 7 |
| US18 | 21 | 3 | 7 |
| Implementierung verschiedener Parkpreise/ Unterscheidung von Parkplätzen | 13 | 2 | 6,5 |
| Änderung der Parkhaus API um E-Autos/Zweiräder zuzulassen (Vielleicht auch Eigene Typen unterstützt, Recherche benötigt | 34 | 8 | 4,25 |
|  |  |  |  |
| Aufbereitung von Parkdaten zur statistischen Analyse (Code) | 21 | 5 | 4,2 |
| US08 | 13 | 5 | 2,6 |
| US09 | 13 | 5 | 2,6 |
| US10 | 13 | 5 | 2,6 |
| ~~Weboberfläche Gestalten die Alle Daten veranschaulicht~~ | 8 | 5 | 1,6 |
| US16 | 8 | 5 | 1,6 |
| US17 | 8 | 5 | 1,6 |
| Implementierung von Suchfunktionen auf den Parkdaten | 8 | 5 | 1,6 |
| US19 | 8 | 5 | 1,6 |
| Modul zur Generierung von Statistiken anhand von Aufbereiteten Daten (Code maybe Library) | 21 | 21 | 1 |
| US02 | 21 | 21 | 1 |
| Implementierung eines Zählers der die verfügbaren Parkplätzte für verschiedene Parkplätze festhält (Client seitig) | 1 | 2 | 0,5 |
| US06 | 34 | 89 | 0,382022 |
| Abweisung von Autos die nicht mehr Parken können (Clientseitig) | 1 | 3 | 0,333 |
| US11 | 1 | 8 | 0,125 |

Der Punkt 'Weboberfläche gestalten die Daten ' wurde gegen den Punkt 'Änderung der Parkhaus API um E-Autos/Zweiräder zuzulassen...' ausgetauscht. Punkt 1 fällt weg, da eine Weboberfläche schon existiert, Punkt 2 wurde ins MVP mit aufgenommen, da der Nutzen doch größer ist als Anfangs angenommen.

### **Aufgabe 4.2: UML-Diagramme für das Projekt**

UCD:

@startuml

left to right direction

skinparam packageStyle rectangle

actor CarParkUser

actor CarParkOwner

Package Server {

actor PriceCalcModule

actor ParkingGarageRelocator

actor StatisticsModule

}

rectangle CarPark {

CarParkUser "1" --"1" (checkin)

CarParkUser "1" -- "1" (checkout)

CarParkUser "1" -- "1" (payTicket)

CarParkUser "1" -- "0.\*" (getFreeSpaces)

CarParkUser "1" -- "0.\*" (getPricetable)

CarParkUser "1" -- "0.\*" (calculateCostEstimate)

CarParkOwner "1" -- "0.\*" (statistics)

(getPricetable) "0.\*" -- "1" PriceCalcModule

(calculateCostEstimate) "0.\*" -- "1" PriceCalcModule

(checkout) "0.\*" -- "1" ParkingGarageRelocator

(payTicket) "" -- "" ParkingGarageRelocator

(checkin) "" -- "" ParkingGarageRelocator

(getFreeSpaces) "" -- "" ParkingGarageRelocator

(statistics) "" -- "" StatisticsModule

(checkout) .> (checkin) : <<include>>

(checkout) .> (payTicket) : <<include>>

(checkin) .> (getFreeSpaces) : <<include>>

(getFreeSpaces) .> (statistics) : <<extends>>

(calculateCostEstimate) .> (getPricetable) : <<include>>

}

@enduml

CD:

@startuml

skinparam classAttributeIconSize 0

Class Car {

-params: String

+nr(): int

+begin(): long

+end(): long

+getDuration(): int

+getPrice(): long

+getVehicleTyp(): String

+equals(car: Car): boolean

}

Class Statistics {

- cars: ArrayList<Car>

+ addCar(car: Car)

+ getCarList(): ArrayList<Car>

+ getSum(): double

+ getAvg(): double

+ getTime(): double

+ getAvgTime(): int

}

Class PriceCalc {

- dayPrice: float

- nightPrice: float

+ calcDayNightPrice(priceInEuros: double, begin: long, duration: int): double

}

Class ParkingGarage{

-spaces: ParkingLot[]

+getGarage(): ParkingLot[]

+getGarageUnprotected(): ParkingLot[]

+parkCar(c: Car): int

+getFreeParkingSpaces(s: String[]): int[]

+getParkingSpaces(s: String[] ): int[]

+removeCar(c: Car) Car

}

Class ParkingLot{

-allowed: String[]

-vehicle: Car

+isOccupied(): boolean

+canPark(c: Car ): boolean

+removeVehicle(): Car

+parkVehicle(c: Car)

+getVehicle(): Car

+getAllowed(): String[]

+setAllowed(s: String[] )

+isAllowed(s: String): boolean

+carEquals(c: Car): boolean

}

Class CarParkServlet{

-NAME: String

-MAX: int

-config: String

+doGet(request: HttpServletRequest, response: HttpServletResponse)

+doPost(request: HttpServletRequest, response: HttpServletResponse)

+getContext(): ServletContext

+locator(car: Car): int

+cars(): ParkingGarage

+GetBody(request: HttpServletRequest): String

+destroy()

}

ParkingLot "0..1" \*-- "1" Car

ParkingGarage"1" \*-- "n" ParkingLot

CarParkServlet "1" o-- "1" ParkingGarage

CarParkServlet "1" o-- "1" Statistics

CarParkServlet "1" o-- "1" PriceCalc

Statistics "0..1" o-- "n" Car

@enduml

**Aufgabe 5 Robustheitsanalyse**

**Aufgabe 6** Zustandsmodellierung

Lebenszyklus eines Tickets:

1. Ticket wird gedruckt (Ticketnummer, Timestampbarcode)
2. Ticket wird eingesteckt
3. Ticket wird in bezahl Automaten gesteckt
4. Preis wird berechnet
5. Ticket wird bezahlt
6. Ticket wird herausgezogen
7. Ticket wird in schranke gesteckt
8. Ticket wird geschreddert

Zustände:

Startzustand : Frisch gedruckt:

Entry: Ticketnummer, Timestamp

--> event herausgezogen

Zustand Bezahlung

Entry: Preisberechnet

Geld eingeschmissen x := erhöht um geld

--> x=preis

Wechselgeld x > preis

Entry: rückgeld x-preis

--> endzustand

**Aufgabe 7 Funktionale Dekomposition und Programmierung**

**Aufgabe 8 Design Patterns**

**Aufgabe 9 Design PatternsTeil 2**

**Aufgabe 10 Muster (Patterns) Teil 3 (von 3)**

**Aufgabe 11 Testüberdeckung**