1.3 Vision des Software Produkts

Jedes Parkdeck kennt die eigene Anzahl an Parkplätzen und weiß durch Sensoren im Boden, wann diese belegt sind und kann deshalb anzeigen, ob und welche Parkdecks noch freie Plätze besitzen.

Der Betreiber hat Zugriff auf die Finanzen und die Anzahl der Autos die in der letzten Zeit dort geparkt haben (vielleicht durchschnittliche Parkdauer, etc.?). Nach Bezahlen der Parkgebühr kann man das Parkhaus wieder verlassen.

kannst ja mal drüber schauen, ob dir noch was anderes einfällt, aber das ganze soll ja scheinbar abstrakt sein

1.4 User Stories

Als Autofahrer benötige ich einen Parkschein um in das Parkhaus hineinfahren zu können.

Als Betreiber des Parkhauses benötige ich Listen und Summen über die eingenommenen Gebühren und die Anzahl der Autos, um eine Übersicht zu bekommen.

Als Abonnent des Parkhauses benötige ich einen Ausweis, um ohne Parkschein in das Parkhaus hinein fahren zu können.

Als Ampel/Anzeigetafel muss ich die Anzahl der Parkplätze kennen, um zu wissen ob noch Platz ist.

Als Kassenautomat muss ich die Uhrzeit kennen und Zugriff auf die Preislisten haben, um den Preis berechen zu können.

Als Parkdeck muss ich die Anzahl der geparkten Autos zählen, um festzustellen ob es noch freie Plätze gibt.

Als Schranke muss ich Tickets entgegennehmen können, um die Schranke zu öffnen (wenn ein Auto das Parkhaus verlassen will).

3.2.1 Prakhaus-Simulation mit Sevlet

Eigene Zusatzfunktion

Die Zusatzfunktion besteht aus der Anzeige der kürzesten Parkdauer. Für den alljährlich, vom Commité de Parc (mit Sitz in Paris), ausgetragenen Kurzparkwettbewerb, ist diese Anzeige von höchster Wichtigkeit.

3.2.2 Demo Servlet mit Summe, Duchschnitt und eigener Zusatzfunktion

DemoServlet.java

3.2.3 Deplayment Descriptor web.xml

3.2.4 SimpleCORSFilter

4.3 UML-Diagramme für das Projekt (zu den User stories) Use Case Diagramm

Anforderungen an das Parkhaus System

1. Das Parkhaus besteht aus einer festen Anzahl an Parkdecks. Jedes Parkdeck hat eine feste Anzahl an Parkplätzen. Pro Parkplatz lässt sich ein Auto parken. Die Zahl der belegten Parkplätze muss nachgehalten werden. Sollte das Parkhaus voll sein, muss die Ampel auf rot schalten und die Schranke darf sich nicht mehr öffnen, um weitere Autos hineinzulassen.

2. Jeder Fahrer eines Autos, der in das Parkhaus einfährt, erhält, an der Schranke am Eingang, ein Parkticket. Dieses Ticket beinhaltet die Zeit, zu der der Fahrer das Parkhaus betreten hat. Dieses Ticket wird vom Ticketautomaten verwendet um abschließend die Parkdauer und die daraus resultierende Parkgebühr zu berechnen.

3. Kunden können zusätzlich ein Abonnement für das Parkhaus abschließen. Dadurch erhält der Kunde einen Ausweis, der den Parkschein ersetzt.

4. Für jedes ein- und ausgefahrene Auto wird die Parkdauer und die bezahlte Parkgebühr gespeichert. Zusätzlich werden die Abonnenten in der getrennten Liste erfasst.

5. An der Ausfahrt, muss der Automat überprüfen ob das Ticket bezahlt wurde und seit dem Bezahlvorgang keine zu große Zeit vergangen ist.

Klassendiagramm für das Parkhaus System

5.2 Robustheitsanalyse für Ihr Parkhaus System

Use Case Template für LeaveGarage

Use Case Template

Titel: LeaveGarage

Akteure: User, Schranke

Vorbedingung: User besitzt bezahltes Ticket

Standardschritte:

1. Der User führt das Ticket in die Schranke ein

2. Die Schranke kontrolliert, ob das Ticket entwertet wurde

3. Die Schranke öffnet sich, der User kann das Parkhaus verlassen

Ausnahmefälle: Das Ticket ist nicht bezahlt -> Die Schranke bleibt geschlossen

Nachbedingung: Der User verlässt das Parkhaus und die Schranke schließt sich nach einer Weile

Qualitätsanforderungen:

Kein User sollte in der Lage sein mit einem nicht entwerteten Ticket die Gerage zu verlassen. Gleichzeitig sollte es vermieden werden, dass sich die Schranke fälschlicherweise nicht öffnet.

Außerdem sollte die Schranke sich wieder schließen

Sequenzdiagramm für ein Use Case

( nicht gmeacht : 5.3: Parhaus-Servlet um Charts erweitern )

6.8 UML-Zustandsdiagramm Parkhaus Ticket

7.1 Aktivitätsdiagramm Parkhaus mit Schwimmbahnen

7.3 Filter-Map-Reduce

Implementation von ParkHaus, Auto und Kunde

8.1 Deplayment Diagram Parkhaus

8.4 CCD-Parkhaus Prinzipien denen wir folgen

1. Evolvierbarkeit: Unter anderem erfüllen wir diesen CCD-Wert, indem wir GitHub zur Versionsverwaltung verwenden, wodurch es uns möglich ist jederzeit zu einem früheren Zustand zurückzukehren, sollte unsere Entwicklung zu viele Fehler erzeugen oder nicht das gewünschte Ergebnis liefern. Außerdem halten wir uns an das Prinzip der "Separation of Concerns", sodass jede Klasse nur eine Aufgabe hat. 2. Korrektheit: Wir nutzen JUnitTests und Interfaces um sicherzustellen, dass der neue Code keinerlei alte Funktionaliät behindert und seine eigene Aufgabe erfüllt. 3. Produktionseffizienz: Wir versuchen unsere Software component-based zu planen, sodass ein späteres Austauschen oder entfernen von Komponenten einfach zu bewerkstelligen ist 4. Kontinuierliche Verbesserung: Wir tauschen uns während und nach unseren Sprints über das neu erlernte aus und versuchen dadurch ständig neues Wissen zu erlernen und anzuwenden.