Loyola Marymount University

Integration of Hybrid HW&SW System
Uber Platform Analysis

Eve Huang

September 7, 2016

Contents

Part 1. Overview

A. Synopsis: about the system

System boundary: Uber is a ride share technology platform that using smartphone apps to connect private drivers and riders. In this case, the system boundary is mobile clients and focus on Uber App and smartPhone users. The website uber.com and PC clients, mobile vehicles are not belong to the system but external system.

Uber App Platform: Uber's app is backed by Google, integrated into Google Maps, and available on the major smartphone platforms. It has driver side app and customer side app. Customer side app allows customers to order cars through their smartphone based on their location. They can see who their driver will be, the rates of the trip, and track the arrival of their car. At the end of a ride, the complete fare is automatically billed to the customer's credit card and an e-receipt will send to customer's phone. Also he customer could give reviews and feedbacks about the trip at the e-receipt. Driver side app allows drivers response customers ride request and track the location of the customer. At the end of the ride, the earns will go to driver's account automatically. Also, it allows the driver rate the riders, too.

B.Users

Users include customer side and driver side. For customer side, users are anyone has smartphones that installed Uber app, and need a ride. For driver side, users are anyone that has car and want to use the car to make some extra incomes.

C. Functions

Uber platform major functions: 1) Google Map and GPS track drivers and riders on real time. Customers can check nearby Uber cars and set up pick up location; drivers can find the location of customers. 2)Ride Rates Inquiry and Surge Pricing. Customer could inquiry ride rates - after the customer input the destination address and Uber car level, Uber platform will give a general cost for this ride. For drivers, a heat map visualization, which shows where demand and fares will temporarily rise 3) Ride Request/ Response. The customer send out the ride request to nearby drivers, the available drivers could response the ride request. 4) Payment. Uber platform charge the ride fee though customer account directly and pay to driver's account. 5) Performance Rating. After every ride, passengers are prompted to rate drivers on a 1-to 5-star scale. Also, drivers could rate their customers, too.

D. Hardware components:

Hardware for Uber apps: SmartPhones, such as iPhone, Android phones, windows phones, that could download and install Uber apps.

Uber servers: web servers to receive requests, application servers to process the data, and a database server to store the data.

Other hardware components: internet, connect all the mobile users and Uber servers.

E. Software components:

Uber Driver App, Uber Ride App, Uber server management system.

1) Dispatch System: it is written by node.js and has thousands of nodes. It is a real-time platform that matches drivers with riders using mobile phones and match a dynamic demand with a dynamic supply in realtime. Dispatch system has following subsystem: a) Google's S2 Geometry Library for Geospatial Index, which can track user ID and update clients' real time location. b) Google Maps and ETA (estimated time of arrival), set up location and calculate maps and routing information, in which street maps and historical travel times are used to estimate current travel times. c) Response/Dispatch, sorted by ETA, the riders can get nearby candidate that meet requirements. After matched the drivers and riders, the available driver response the rider's request. 2) Surge Pricing System. 3) Post trip pipeline system: a) performance ratings system that collecting customer reviews. b) Payment System that manage the payment between drivers and riders. 4) Database Management System, store all the data information.

F. Hardware and Software Integrated

Drivers and riders use their smartPhones to download Uber Apps from App store/ Android Store. Then install drive app and ride app to their smartPhones. Clients through the apps installed at smartphones connect to the Uber server through internet. The Geospatial Index system at server, receive and update the smartPhones or the users' real-time location. Dispatch system handles and process the data information, such as the real time request, finds the nearby candidates, and matches drivers and riders. The database server store the the data information of clients. In general, through smartPhone platform, Uber apps installed to client's mobile phones; through internet, Uber achieved information exchange between drivers and riders.

Part 2. Hybrid System UML Diagrams and Analysis

A. Use Case Diagram

In Uber System, the Use Case includes Rider, Driver, and Uber Platform

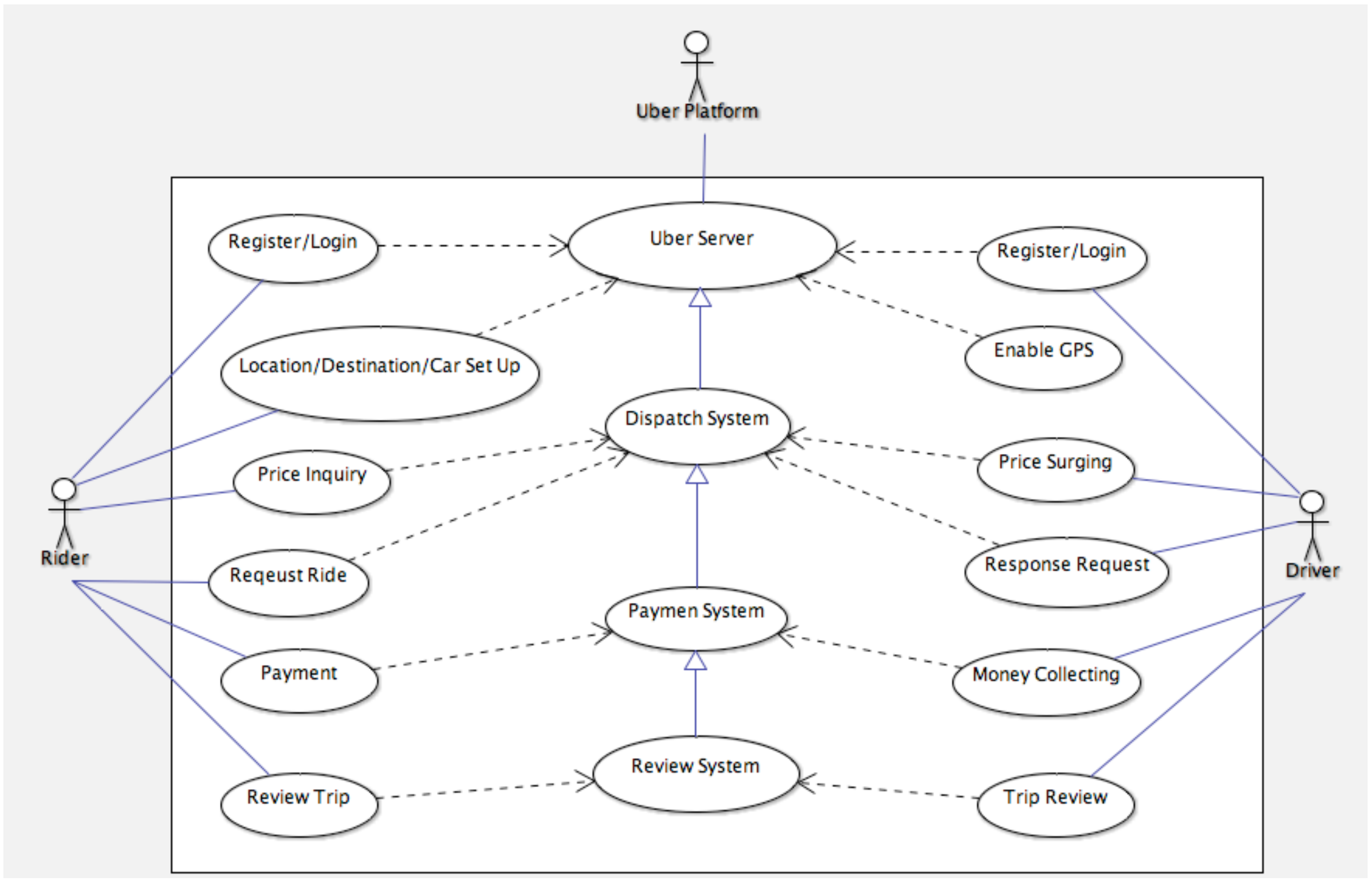


Diagram-a User Case Diagram

User Case Rider has the function of register and login system, set up the location, destination, and car level, inquiry the price of the trip, request the ride, pay the trip, and review the trip.

User Case Driver has the function of register and login the system, enable the GPS to set up the location, find the heat map area, response the riders, collect the payment, and review the customers.

User Case Uber Platform including Server that authorize the users account, set up the account details. The server also including the dispatch system that match riders and drivers, the payment system and review system.

B. Deployment Diagram

Deployment Diagram described the hardware and software integrated in Uber system.

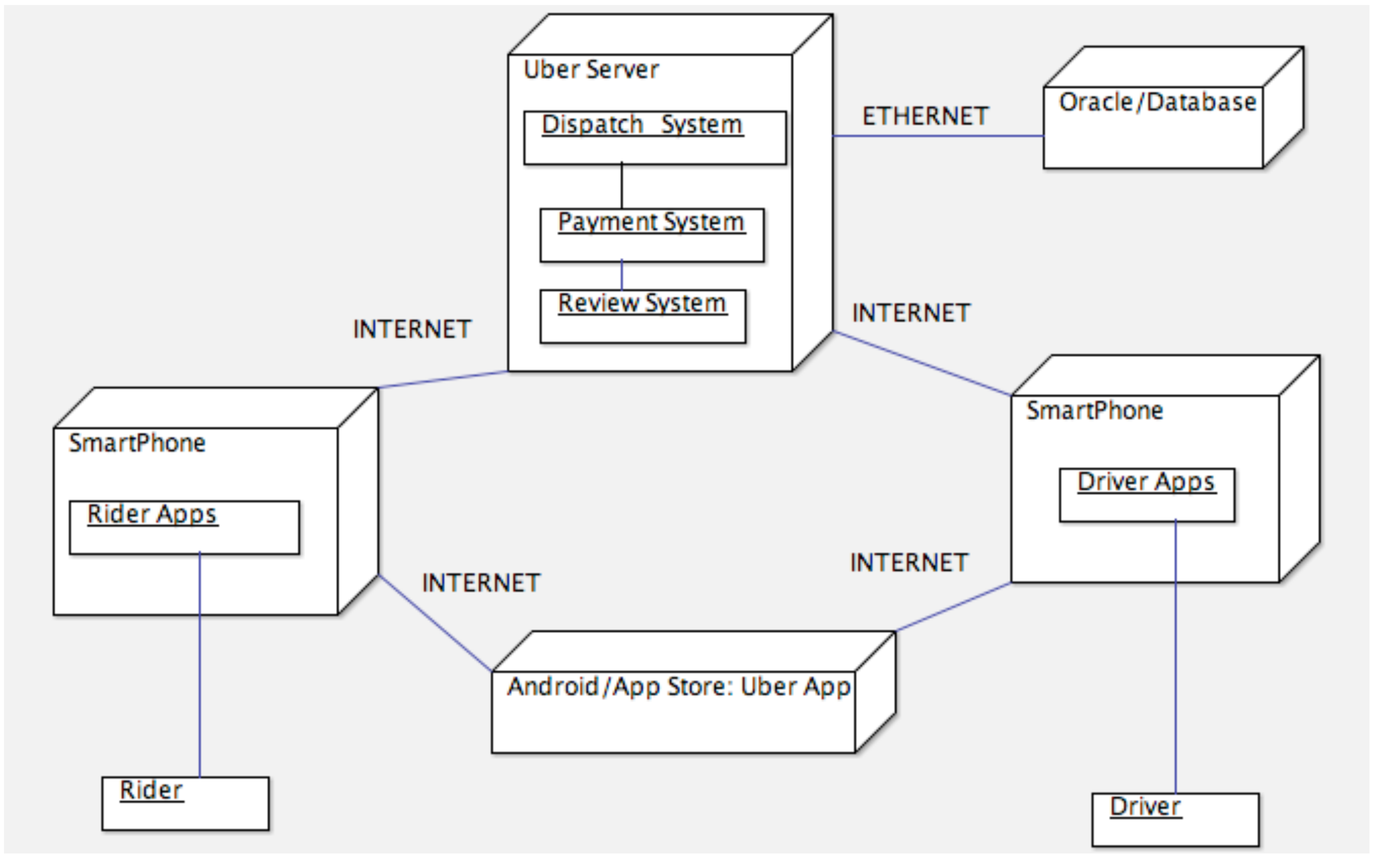


Diagram-b Deployment Diagram

Hardware for Uber apps: SmartPhones, such as iPhone, Android SmartPhones, Windows Phones, that could download and install Uber apps.

Uber servers: web servers to receive requests, application servers to process the data, and a database server to store the data.

1) Drivers and riders use their smartPhones to download Uber Apps from App store/ Android Store through internet. 2) Then install drive app and ride app to their smartPhones. 3) The Uber apps installed at smartphones connected to the Uber server through internet. 4)The Geospatial Index system at server, receive and update the smartPhones or the users' real-time location. Dispatch system handles and process the data information, such as the real time request, finds the

nearby candidates, and matches drivers and riders. 5) The database server store the the data information of clients. Uber server connects to Database through ethernet.

C. An Use Case Outcomes (Select an main use case from Diagram-1 Use Case Diagram)

From Diagram-1 Use Case Diagram, select the Rider as the example. All the outcomes for riders as List-1 OutComes for Riders :

Use Case - Riders Outcomes:

1) **Register**

Successful/ Failure: 1) personal information error 2) invalid credit card 3) invalid address 4) invalid email

2) Login

Successful/ Failure: 1) wrong user name 2) wrong password 3) phone ISO update 4) Uber version update

3) Set Up Location/ Destination/ Car Level

Successful/ Failure: 1) GPS not work 2) map error 3) invalid address 4) internet error

4) Price Inquiry

Successful/ Failure: 1) internet error 2) price error 3) map track error 4) system calculate error

5) Ride Request and Confirm

Successful/ Failure: 1) no available cars 2) no response from drivers 3) long waiting

6) Trip Completed

Successful/ Failure: 1) driver unable to find the pick up location 2) driver lost on the way 3) bad traffic 4) car accident

7) Payment

Successful/ Failure: 1) invalid credit card 2) invalid billing address 3) invalid papal/ apple pay account

8) **Review/ Feedback**

Successful/ Failure: 1) no receipt 2) invalid input comments 3) unable to submit the feedback

List-c OutComes for Riders

D. Sequence Diagram

Select two successful outcomes from List-1 OutComes for Riders as sequence diagram

1) The rider inquiry the trip price successful

2) The rider completed the whole trip and review the trip successful

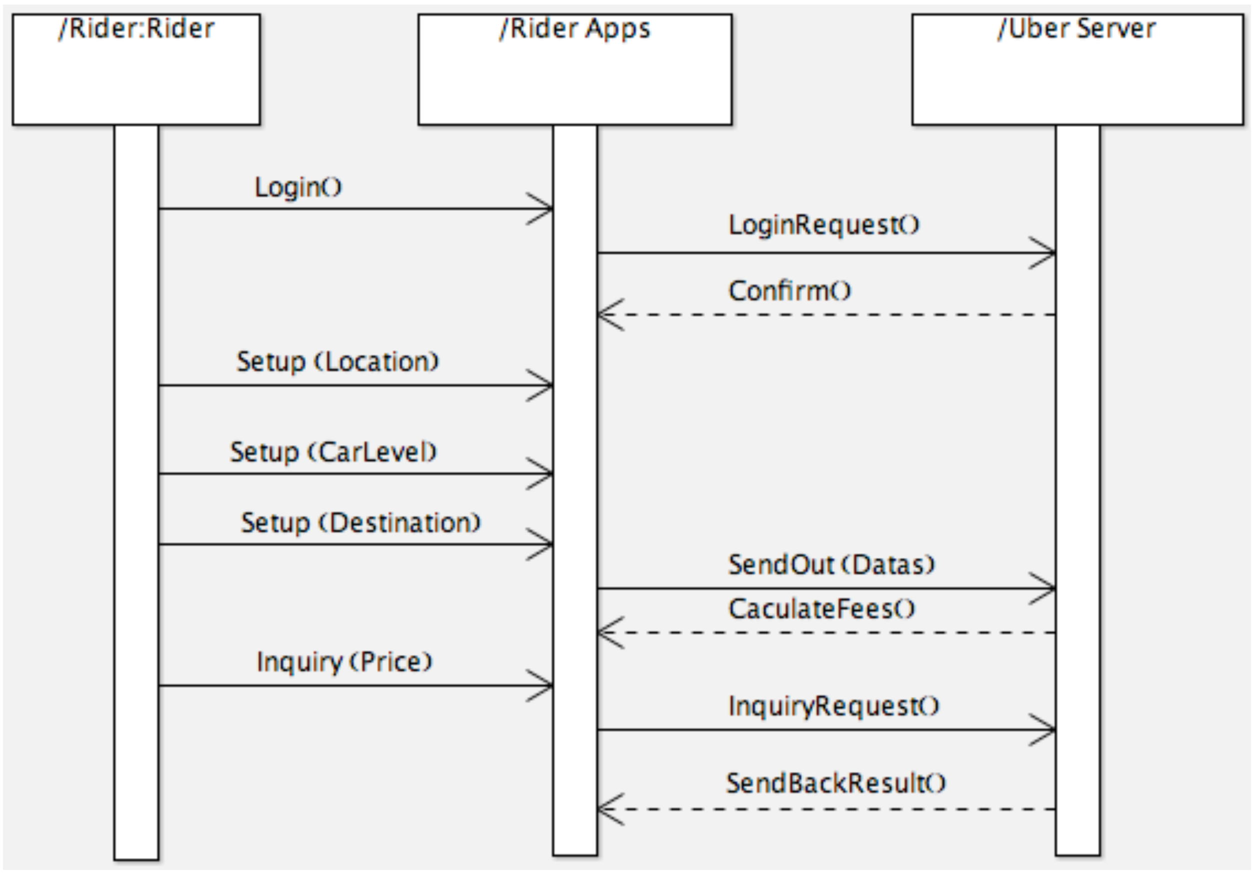


Diagram-d_1 Sequence Diagram 1_Inquiry Price

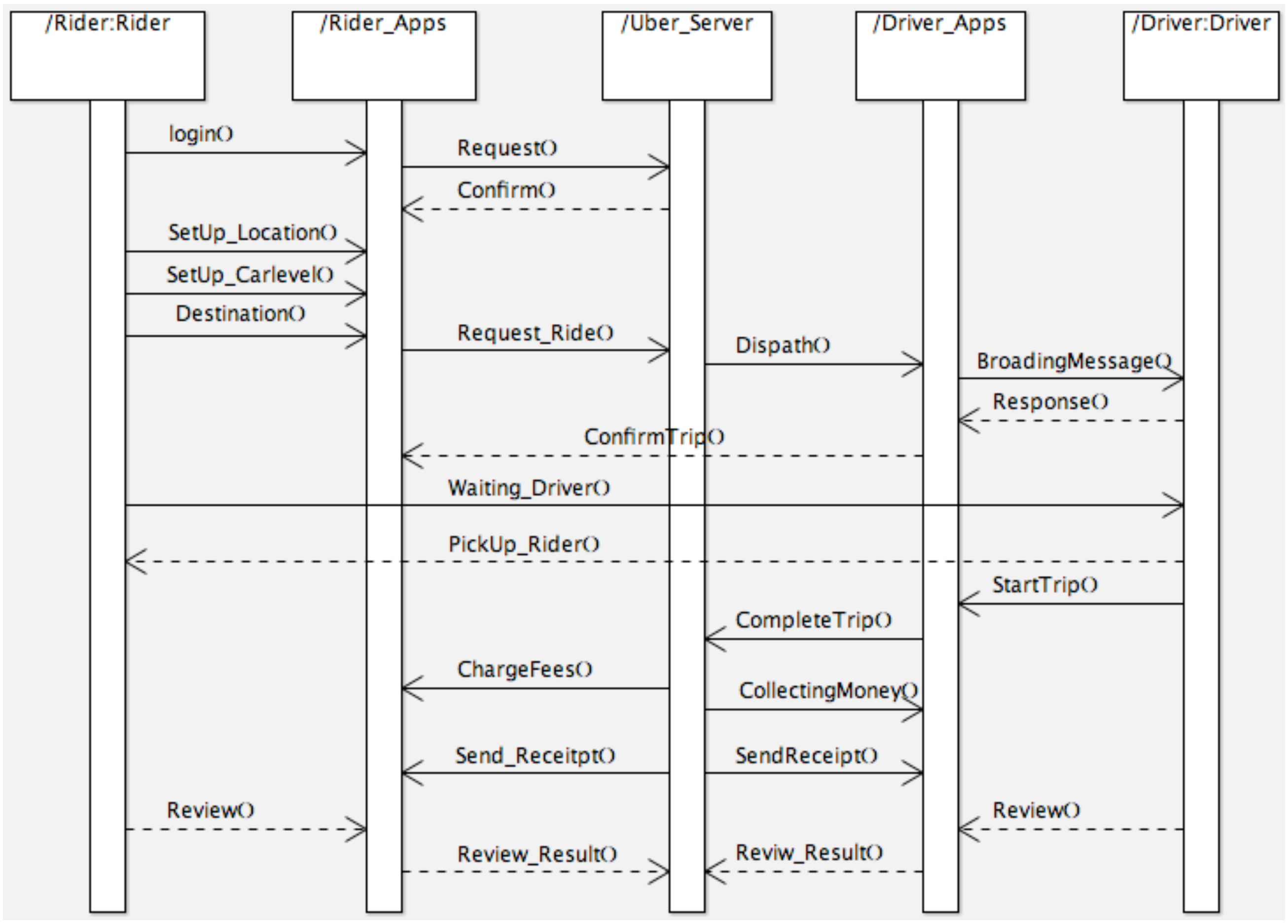
1. Riders inquiry the trip fees successful:

1) The rider login to the account through Rider Apps.

2) Uber Apps send the login request and server authorize the login.

3) Rider set up the pick up location, choose the car level, set up the destination

4) Uber Apps send the datas to Uber server to calculate the estimating fees for the trip.



5) Rider inquiry the price, Uber server send back the estimating price for the trip.

Diagram-d_2 Sequence Diagram 2_Trip Successful

2. Riders complete the trip successful and give back the review

1) The rider login to the account through Rider Apps. 2) Uber Apps send the login request and server authorize the login. 3) Rider set up the pick up location, choose the car level, set up the destination. 4) Riders send out the ride request, and the Uber Apps send the request to Uber dispatch system. 5) Uber dispatch system match the nearby drivers, the Driver Apps broadcasting the request message to matched drivers. 6) The driver response the ride request and confirm the request to riders. And pick up riders at pick up location. 7) After picked up riders successful, drivers start the trip and arrived at destination address. Drivers confirm the trip completed through Driver Apps. 8) Uber payment system charge the fees through rider account and send the money to driver through drive's account. 9) Uber server send the receipt to drivers and riders. 10) Riders and drivers write the comments and give the feedback for this trip.

E. Class Diagram

In Uber System, the class could have users, apps, and vehicles

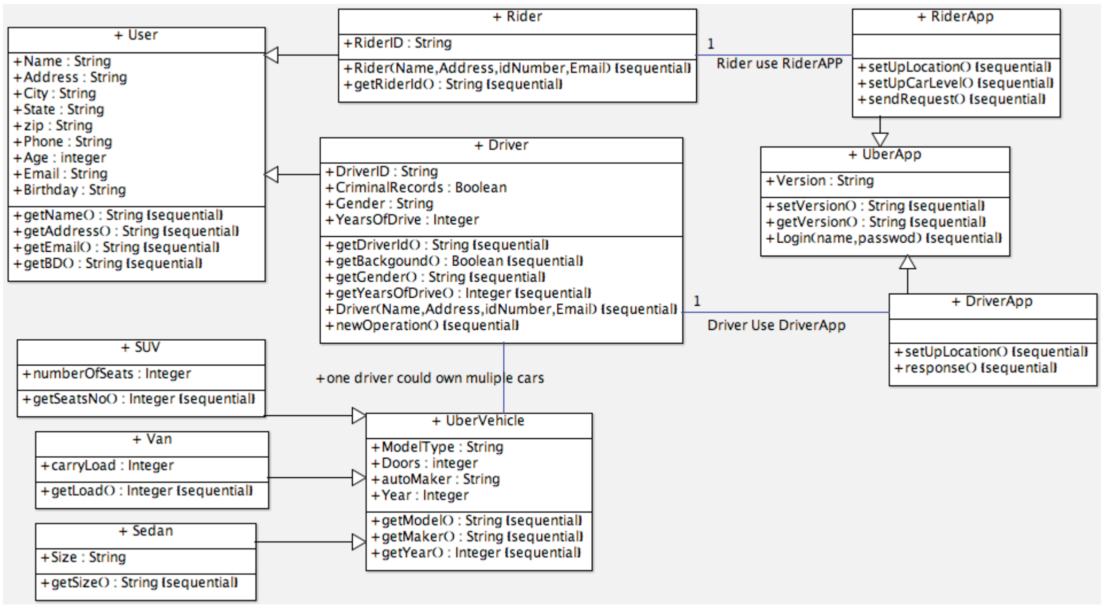

+ User
+Name : String
+Address : String
+City : String
+State : String
+zip : String
+Phone : String
+Age : integer
+Email : String
+Birthday : String
+getName() : String {sequential}
+getAddress() : String {sequential}
+getEmail() : String {sequential}
+getBD() : String {sequential}
+ Rider
+RiderID : String
+Rider(Name,Address,idNumber,Email) {sequential}
+getRiderId() : String {sequential}
1
Rider use RiderAPP
+ RiderApp
+setUpLocation() {sequential}
+setUpCarLevel() {sequential}
+sendRequest() {sequential}
+ Driver
+DriverID : String
+CriminalRecords : Boolean
+Gender : String
+YearsOfDrive : Integer
+getDriverId() : String {sequential}
+getBackgound() : Boolean {sequential}
+getGender() : String {sequential}
+getYearsOfDrive() : Integer {sequential}
+Driver(Name,Address,idNumber,Email) {sequential}
+newOperation() {sequential}
+ UberApp
+Version : String
+setVersion() : String {sequential}
+getVersion() : String {sequential}
+Login(name,passwod) {sequential}
1
Driver Use DriverApp
+ DriverApp
+setUpLocation() {sequential}
+response() {sequential}
+ SUV
+numberOfSeats : Integer
+getSeatsNo() : Integer {sequential}
+one driver could own muliple cars
+ Van
+carryLoad : Integer
+getLoad() : Integer {sequential}
+ Sedan
+Size : String
+getSize() : String {sequential}
+ UberVehicle
+ModelType : String
+Doors : integer
+autoMaker : String
+Year : Integer
+getModel() : String {sequential}
+getMaker() : String {sequential}
+getYear() : Integer {sequential}

Diagram-e Class Diagram

Class Generalization:

1) Users class. It has two subClasses: one is Uber Riders, the other one is Uber Drivers.

2) Uber Vehicle class. It has several subclass, such as SUV, VAN, Sedan car, etc

3) Uber Apps class. It has two subClass: UberDriver Apps, the other one is UberRider Apps.

Class Associations:

1) UberDrivers with UberVehicles: one Uber driver could have multiple uber cars.

2) Drivers associated with UberDriver Apps; Riders associated with UberRider Apps.

F. StateChart Diagram

StateChart Diagram for the riders of the Uber system.

Riders initial the status, then 1) Set up location and car before the rides. 2) After send out the ride request, if the request is successful, riders will waiting at home for Uber driver arriving; or will wait the available drivers and start the request again. 3) Waiting at pick up location. 4) Riders in the car after been picked up. 5) Riders at road/trip during the driving. 6) If the trip has no accident, the riders will be at the destination; or the riders will on the way somewhere, will start a new trip again.

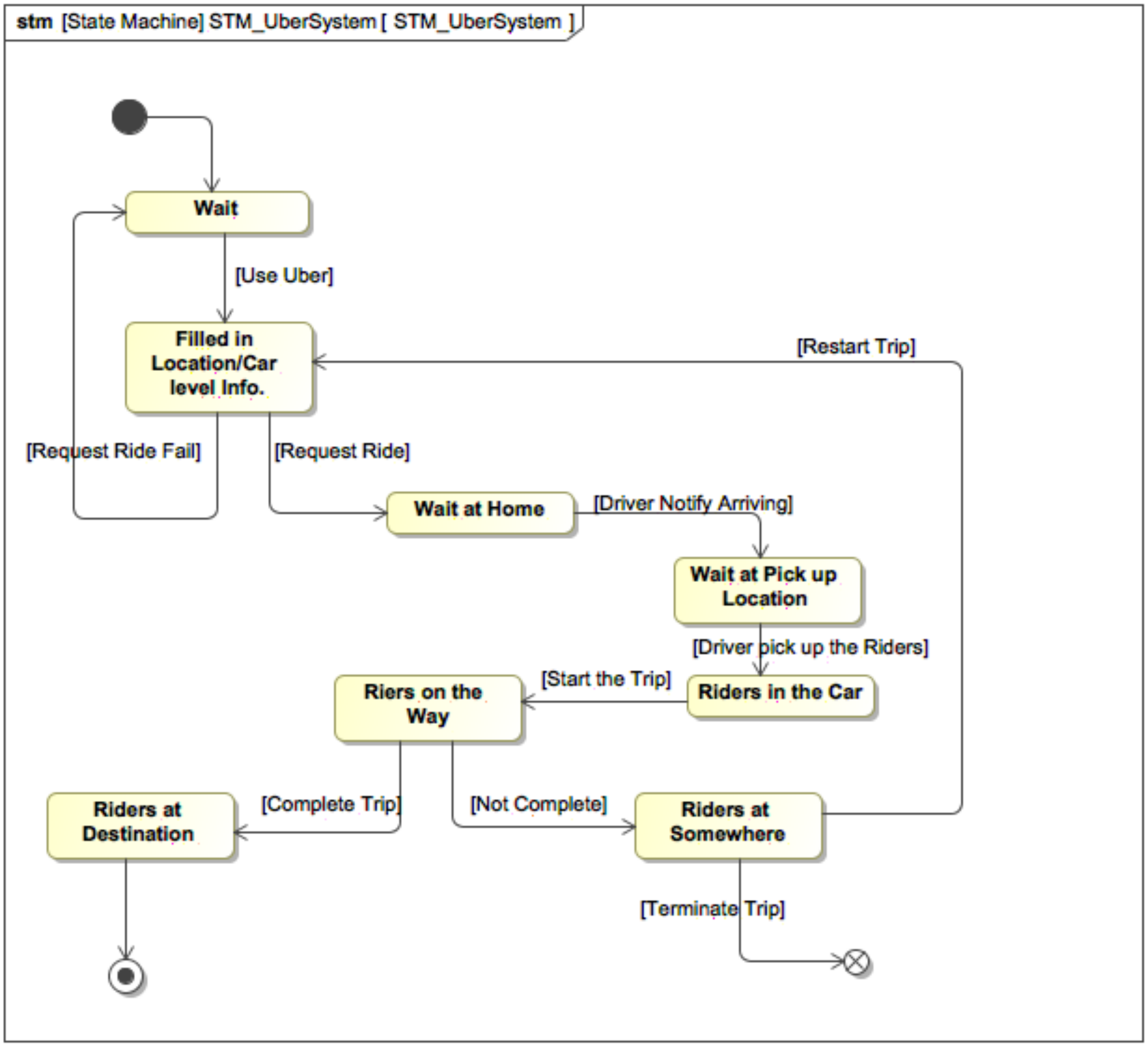
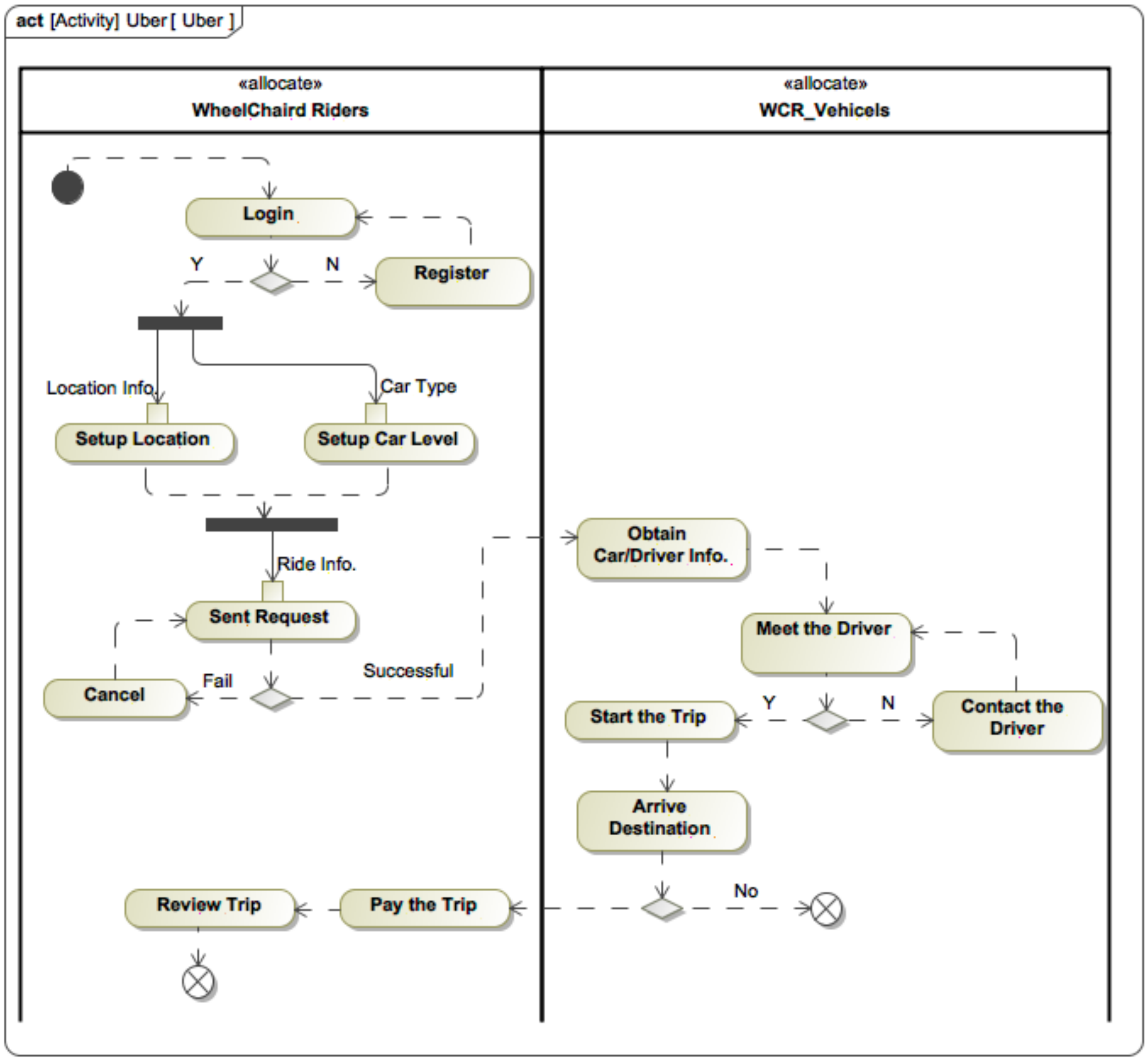


Diagram-f StateChart Diagram

G. Activity Diagram

1. Riders complete the trip successful and give back the review :

1) The rider login to the account through rider apps, if successful then continue step 2, if not then go back to register and login again. 2) Rider set up the pick up location, choose the car level, set up the destination. 3) Riders send out the ride request and waiting for response, if the response failed then cancel the request, resent the request again; if the request successful, move to next step. 4) Riders head to pick up location and waiting: if driver not show up, then contact driver; if meet the driver successful, then move to next step. 5) Riders start the trip. 6) Riders completed the trip: if yes, then pay the trip and review; if fail then start the new trip again.



Diagram_e1 Activity Diagram 1 _ Riders

2. Drivers pick up riders successful and complete the trip successful and give back the review :

1)The driver login to the account through driver apps, if successful then continue step 2, if not then go back to register and login again. 2) Driver set up the location, and set up the status. 3) Driver waiting for the rider request. 4) The driver response the ride request and confirm the request to riders. And pick up riders at pick up location. 5) If the riders didn't show up, then contact the riders; or pick up the riders successful. 6) After picked up riders successful, drivers start the trip and arrived at destination address. 7) Driver confirm the trip completed through Driver Apps. 8) Driver receive the payment and review the trip.

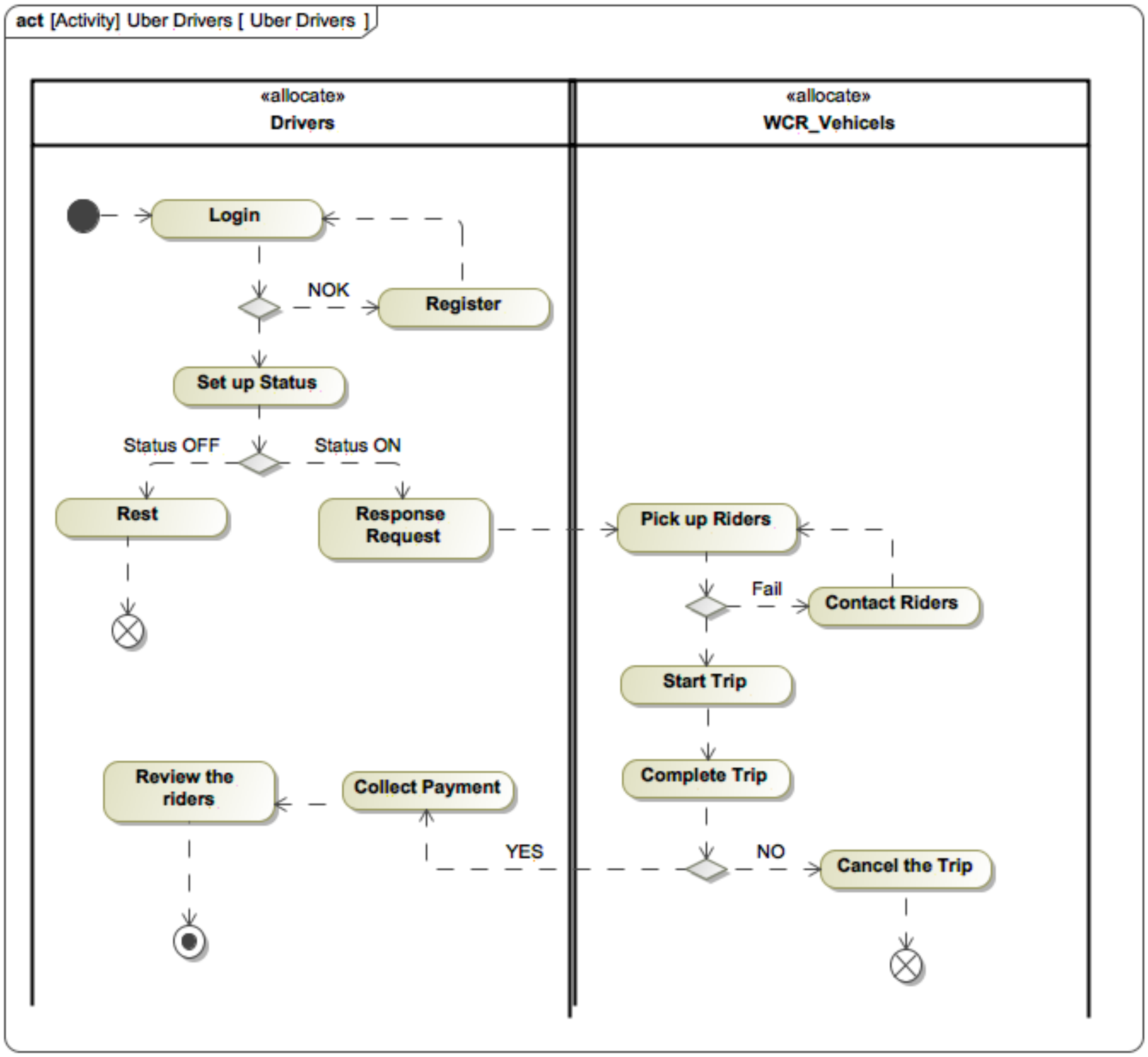


Diagram e2 Activity Diagram 2 _ Drivers

Part 3. Hybrid System Test

1. Non-functional Requirements

Category of Requirement	Description	Verification Metric/Method
a. Ease of Use(User Interface/User Experience)	Uber Apps UI shall be well organized, simple and easy to operate.	The user could register, login, and process request ride without help process.
b. Reliability& Availability	Uber Apps shall be stable and not be crashed.	The tolerable crash time should be under two times one month.
	The riders shall be able to request ride any time and the Uber shall cover as much area as it can.	The available time for Uber riders should be 24/7 hrs, and the coverage area of Uber cars should at least 65%.
c. Performance	The Uber Apps and Drivers shall response the ride request as soon as possible.	The response time of ride request should be less than 3 minutes.
	The Uber drivers shall pick up the riders as soon as possible.	The riders' waiting time for Uber drivers should be less than 10 minutes.
d. Security	The riders/ drivers account shall be safe and find the account back if been the user forget the password.	The riders/ drivers account should recovered by email, safe questions, etc.
e. Maintainability	Uber server shall backup all the customers data and easy to recover from crash.	The data backup procedures should be once a day. And the recover process should less than 2 hrs.
f. Portability	The customers shall install the Uber Apps at all different kind of cell phones at different version OS.	Uber Apps should be compatible to IOS, Android, and Windows phones. Also the old and newest version of Apps could to installed the different OS phone Versions.
g. Constraints	The cost of riding Uber shall be affordable than traditional taxi.	The cost of each trip should save at least 20% than taxi.
h. Interface to other systems	Uber platform shall be easily update and match to the newest technology.	Uber system could implement newest google api or google map, etc.
i. Error Handling	Uber shall stop the mistake request and be able to cancel the ride request.	Uber Apps should have CANCEL function/button to cancel the orders or terminate the trip by mistakes.

2. Test Strategy for Uber Platform

My test strategy for Uber platform including following tests:

1) **White Box Unit Testing:** a) statement coverage: this test will check and execute each statement of the Uber Apps. Take Uber rider app as an example, test the login/ register account, set up location, sent out the request, pay the trip, review the trip, etc. b) branch coverage: make sure all operation branches executed. For example, test ride request process and then cancel the process. c) path coverage: the test cover all possible paths from the beginning of the code to various end points in the code.
2) **Black Box Unit Testing:** test all inputs and outputs of the functional aspects. For Uber rider app, make sure it has the account function such as login, register, payment, account set up, etc. It could input the location information, set up route for destination, and inquire the cost of the trip, etc. It could input the payment information and review of the trip, etc.
3) **Interface Testing:** test each class like the interface between rider side to Uber server side, the interface between Uber dispatch system with driver apps, etc.
4) **Integration Testing:** extending unit testing to the subsystem. For example, when test Uber Server, testing the subsystem like Uber dispatch system, Google API implementation, database server, payment system, price surge system, etc.
5) **System Testing on Non-Functional Requirements:** using the metrics provided to test those non-functional test. Checking the apps crash time for reliability test; checking the time for Uber rider apps every 5 mins to make sure it is available 24/7 hrs, and checking in all different location to make sure the coverage area of Uber cars at least 65%; checking the ride response time is less than 3 minutes and waiting for pick up time less than 10 minutes; testing Uber Apps is compatible to IOS, Android, and Windows phones, and also testing the old and newest the version of Apps to match different OS phone Versions, etc.
6) **The last installation and customer acceptance testing:** after system testing, sending the samples out for customer acceptance testing. Collecting the feedback from customers, improve the products. If the customer accepts the product, then testing is completed.

Define a regression testing method:

The whole Uber system including rider side, driver side, and uber serve side. It is a complex system. To do regression testing, first we need divide the system to modules. For Uber Rider Apps, it could be divided to account subsystem, operation subsystem, payment subsystem, interface subsystem, etc.

If any updates or changes on one subsystem, just need test related subsystem or modules rather than test whole system.

3. Analysis of Adoption Problems

There are three categories problems should be considered during deploying and adopting the Uber system:

1) **Machine/ Machine,** consider all the technical issues of replacing legacy software with new solutions.
   a) Deploy legacy system problem. Uber Apps need deploy to all different type of cell phones. As the development of mobile phone technology growing, some operate system are out of date. Uber Apps need develop specular interface for those old system.
   b) Third party software adoption problem. In the Uber system, it implemented third party software , such as Google API, Google Maps, GPS, etc. Uber need update the newest interface to match those third party software if they development new technology or new version. Also, sometimes could happened the third party company terminated the software and no longer to provide the support for the software due to the markets or some other reasons, Uber need to find a replacement software or need deploy new interface to the third party software each time if new technology come out to the markets.

2) **Man/ Machine,** consider all the issues caused by business process changes caused by new solutions.
   Programming language adoption problems. Currently, Uber system use language such as javascript, nodes, Node.js, Python, and Redis, etc. The database languages are Postgres and MySQL. Most of those language are very popular those days. However, as time passed some of those language will be out of date or replaced by other new programming language. Human beings are habit-driven. It is easier for the programmers to use the old products. Lots of programmers prefer to using the old language and it is hard for them to learn new language.

3) **Man/ Man,** consider the pact of the organizational, social, and political context.
   a) **Organizational adoption problems.** The organizations is lack of leadership to support for innovations, or not be able to understand the changes.
   b) **Social adoption problems.** If the user accept the new technology or new version of the system.
   c) **Political adoption problems.** When introducing the Uber system to new area or new country, need consider the local laws and regulations. For example, first time Uber trying to enter China, it encountered huge resistance from local taxi company and local governments. Any new innovation or solutions possible trigger local laws and regulations. Hence, Uber need consider those political issues during deploy its system.

Part 4. Executive Summary

Technology Frame Summary

1. The paper contains the general technology frame of Uber platform.

1) The use case diagram described the users of Uber, which including riders and drivers. It also displayed how the users interaction with each other through Uber platform.
2) Deployment diagram shows the external architecture of Uber system. How the software and hardware connected by networks, cables, etc.
3) Use case outcome lists, which list all the possible results of the framework outputs.
4) Sequence diagram examples displayed the whole process and steps about how the riders start the car service and completed the whole trip.

Some comments about Uber Apps:

Uber app is sensational, and fully committed to a great customer experience, exhibit aggressive and resilient growth. The biggest success factor of Uber is that the app is extremely simple to use, and provides everything that the customer would want. The customer can see the estimated cost of their trip, the estimated time of pickup, and a brief bio about the driver. There are also a variety of options ranging from simple cars and cheap rides to fancy SUV's at a slightly higher price. Compare to traditional taxi service, it is more affordable, flexible, and providing optional and diversified matching service。

Overall Uber Apps is a marvel in and of itself, but no company can survive on their product alone. Uber has shown a commitment to customer service that has built brand loyalty with their customers. Both the drivers and Uber's corporate staff have great customer service qualities, whether it be a joyful riding experience, or a quick response to a complaint email. This is a great characteristic of this growing organization.

The final key to Uber's rise has been their rapid and relentless expansion. Uber enters new cities at a blistering pace, making their presence felt regardless of how they're received. They have faced legal issues along the way, but they continue to operate and overwhelm the local governments to change the rules in their ways.

ULM Diagrams Summary

The paper contains several UML diagram of Uber platform.

1) Class Diagram
2) StateChart Diagram
3) Activity Diagram

Some challenges during drawing the diagrams:

1. Class diagram: this paper displayed Uber users, Uber Apps, and Uber vehicle classes. Uber users has two subclass: Uber driver and Uber Rider; Uber Apps has two subclass: driver apps and riders apps. Uber driver is associated to Uber driver apps; Uber rider is associated to Uber rider apps.

Challenges: during we design the class, we need consider a lot of details and the actual situation in real life, such as how deep the class shall be created? In this case, what's the subclass of Uber rider/driver? For example, uber riders could have subclass disabled people, the disable riders has subclass like blinded people and deaf people. Another type of riders is pets like dogs. What's the relationships between those type of riders and the Uber users?

2. StateChart Diagram: StateChart diagrams are used to model dynamic aspect of a system. In this paper, it displayed the status of riders that completed a ride events.

Challenges: to understand the difference between a State Diagram and a Flowchart. A flowchart illustrates processes that are executed in the system that change the state of objects. A state diagram shows the actual changes in state, not the processes or commands that created those changes.

3. Activity Diagram: the activity diagram is suitable for modeling the activity flow of the system. This paper listed two activity diagram. One is driver's activity diagram, the other is rider's activity diagram.

Challenges: during the time I draw the activity diagram, the very beginning, I planned to draw both driver's and rider's activity in one diagram, which make the diagram looks very complicated. An application can have multiple systems. Activity diagram captures these systems and describes flow from one system to another. Before drawing an activity diagram we must have a clear understanding about the elements used in activity diagram. The main element of an activity diagram is the activity itself. An activity is a function performed by the system. After identifying the activities we need to understand how they are associated with constraints and conditions.