Final Presentation

Software Engineering 2 Project

M. Redaelli F. Zanoli

Politecnico di Milano

February 27, 2017

Outline

Outline - Introduction

Introduction

The project we have been assigned is called *PowerEnJoy* and it is a complex software system that should implement a car sharing service. In order to rationalize, clarify, and put in structured and standardized documents all the relevant concepts and informations, we designed and delivered several documents such as the **RASD**, the **DD**, the **ITPD**, and the **PPD**. These slides will only present an overview of the concepts thoroughly described in the above mentioned documents.

Tools

We composed the documents we had to using some tools such as:

- **TexStudio:** to compile LaTeX document.
- **StarUML:** to draw UML diagrams.
- **Alloy Analizer 4.2:** to checking model consistency.
- **Draw.io:** to build mockups.
- **GitHub:** for storing the project.
- **Skype:** for team collaboration

Outline - Requirement Analysis and Specification

Outline

Overview

The aim of the software is to provide a new digital management system for car-sharing service that exclusively uses electrical cars. It can be applied to different small and big city and even in a large urban area.

Goals

This new service pretends to achieve various goals, such as:

- **G4:** Allow user logged in to book a car in a certain location.
- **G6:** A user who request a rent can abort the process when ever he/she wants.
- **G9:** A non registered users can only register once to the service.
- **G10:** A user can get discount or overtaxes from his/her last rent.

Actors - Users, Cars and Customer Service

Below are listed the four main actors that will interact with the application once deployed:

- **Registered User:** A person who subscribed to the system and can access to all functionalities of the application.
- **Visitors:** A person that needs to move from a position to another one among the city and wants to use *PowerEnJoy* in order to do so, but has not registered yet to the service.
- **Car:** The car reserved / rented / Parked that communicates with the system.
- **External Customer Service:** A team that provide technical support in case of problems related to the car.

Product Perspective

Our *PowerEnJoy* is a **completely new product**, not based on previous ones.
It relies on **location data** received via **Internet** from each user application and car system: all the involved smartphones already have a **GPS antenna** installed inside, that communicates their position to the service.
Being a partially **distributed application**, *PowerEnJoy* requires a fully operative **Internet** connection in order to work properly, both on server and client side: **no service is intended to be provided offline**.

More on Product Perspective

All the data generated by this software are stored in a database, accordingly to current normative and laws about privacy and personal data management.
In addition, several **API**s are provided in order to allow further improvements and expansions of the software: for instance, the payment will be managed from an external entity (we supposed **PayPal**) and the map will be provided by **Google**.

Outline

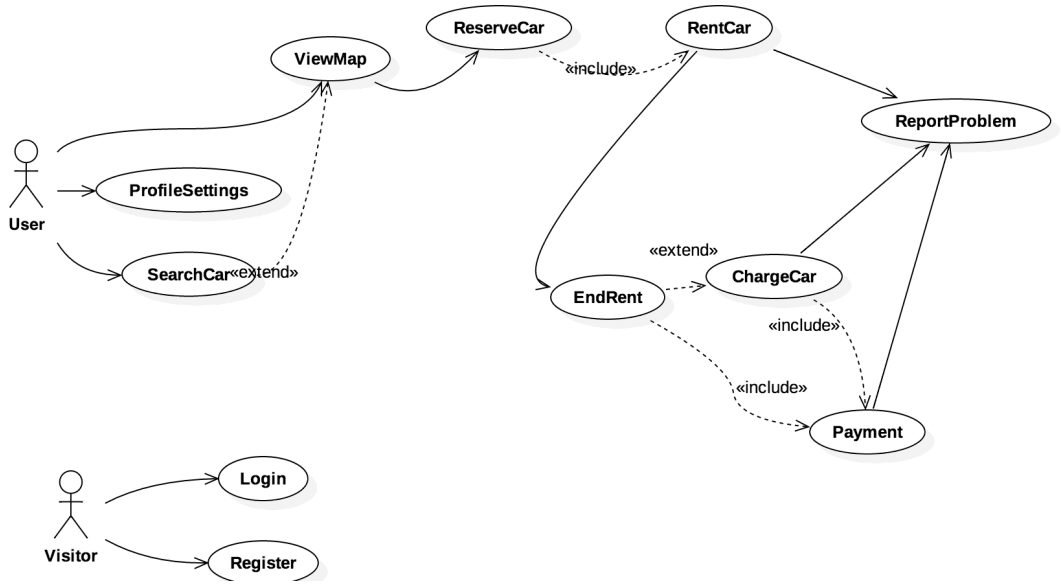
UML Diagrams

We provided a variety of UML diagrams, each type having a different purpose.

- **UML Use Case**
- **UML Sequence Diagram**
- **UML Class diagram**

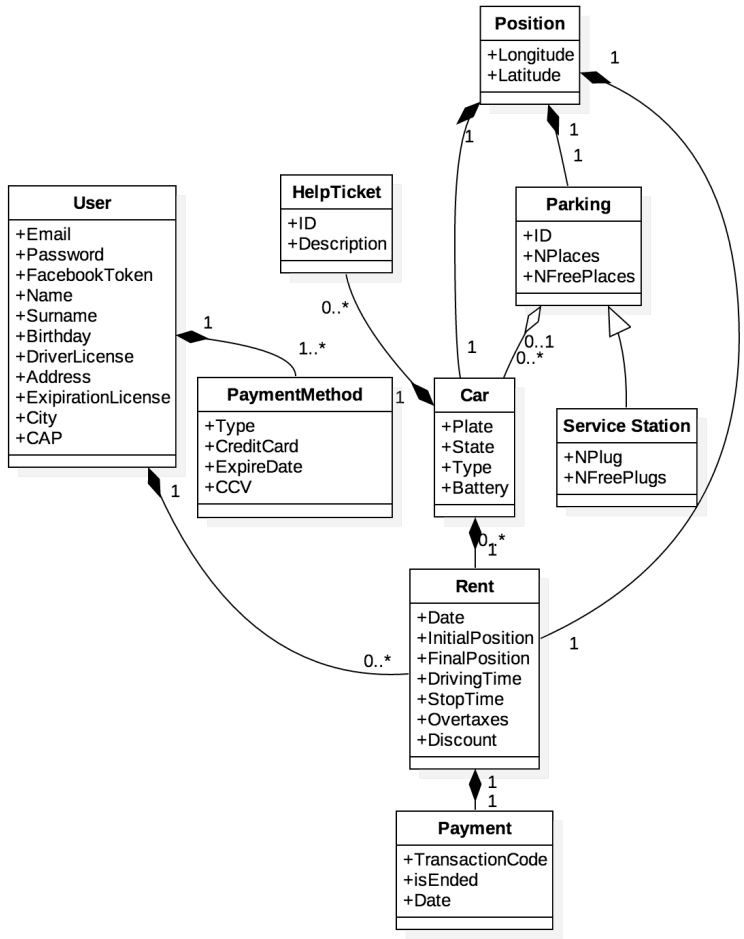
UML Use Case Diagram

This is perhaps the most useful diagram that can be designed in the early phase of the development of a software project.



UML Class Diagram

Furthermore we designed a class diagram for an early evaluation of the basic software components that consists in a sort of **Model** for *PowerEnJoy*.



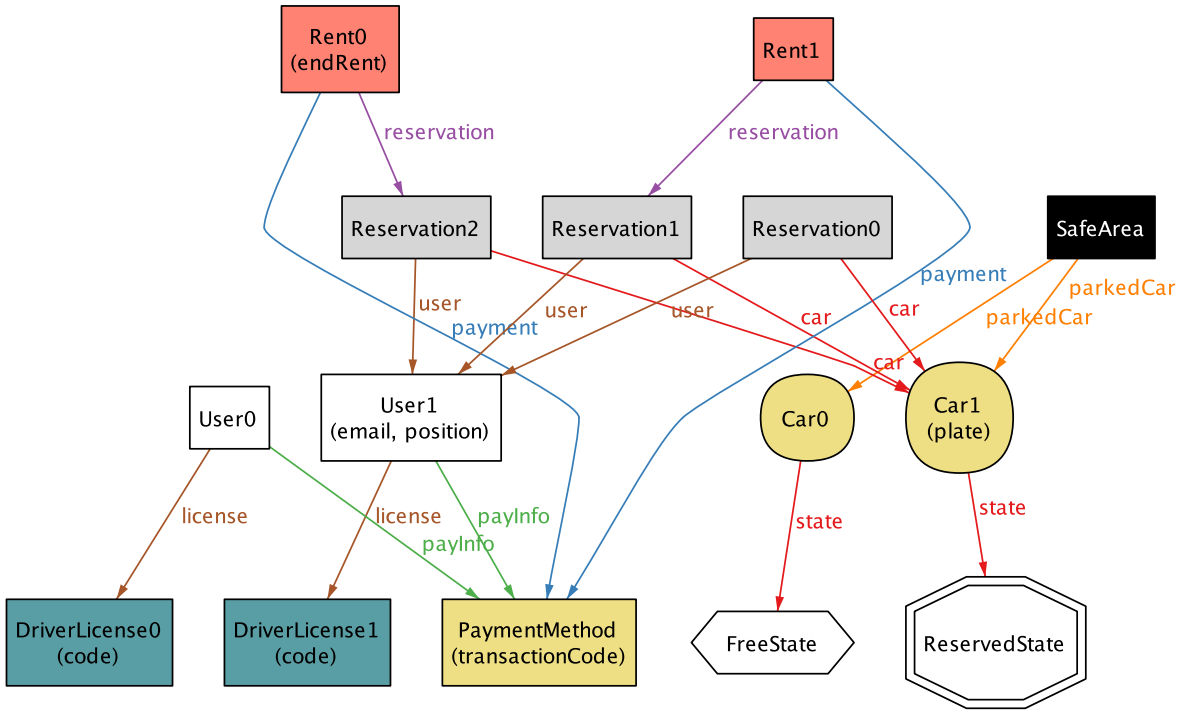
Outline

Alloy

Alongside the **UML Class Diagram** we built **Alloy Models** using the **Alloy** modeling language with the help of **Alloy Analyzer 4.2**.
The tool didn't find a proof of the inconsistency of our **Alloy Models**, and that along with the **Automatic Generation** (and **Manual Verification**) of interesting worlds, made us aware of the **Consistency** of those **Models** within a reasonable level of confidence.

Alloy Simple World

Here is an example of one among the **simplest world** we generated and double checked using both **Alloy Analyzer 4.2** and **manual checking**.



Outline - Design

Outline

Tiers

The selected software architecture follows the principles of the **3-Tiers** architecture. Therefore our structure expands this concept by adding 1 additional tiers to interact with the external word
In particular the tiers are:

- The **Client**, the mobile application designed and projected
- The **Application Server**, the main core of the system that include **Controller** and **Model** into it
- The **DB Server**, the data storage of the *PowerEnJoy*' data.
- The **Car System**, that is the interface between the physical system and the software, furnished by sensor and display in order to comunicate with the user.

High level components and their interaction

The system is composed of many **distributed** components: those will communicate with a **Client-Server** style and through **Point to Point** messaging system.

- The **Client-Server** style is used to give the many Clients connected to the Server the opportunity of sending different requests (e.g. a **Car Reservation** or **Open Car Request**).
- The **Point to Point** bidirectional communication channel is made necessary to enable the Server the delivery of various messages and requests to the Clients and the Car:
  - Generic notifications
  - Service messages
  - Internal message to the car as Open or Check the status

Layer

The selected software architecture follows the principles of the **Model View Controller** architectural pattern, therefore three main software components have been identified and those are:

- The **View**, the user application View that is designed to perform comfortable user expirience

Layer

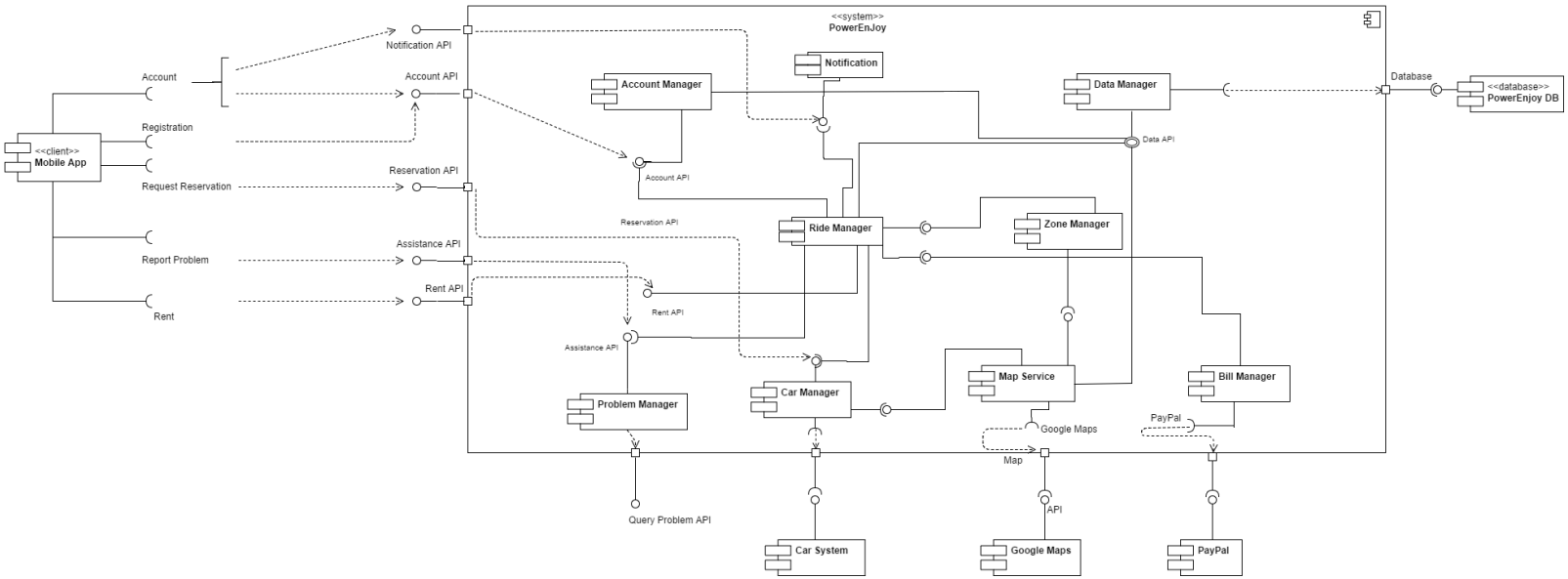
- The **Controller**, in charge of leading the communication between views and process either **synchronous responses** or **asynchronous events**.
- The **Model** that guarantees a high level interface to store and manage all the *PowerEnJoy* relevant data and an abstraction of **Relational Database** in a software component that is in direct connection with the **Controller**

Component View

Several components has been designed to provide all the functionalities needed for *PowerEnJoy* to work. Many subsystems have been identified:

- Ride Manager,Bill Manager,Zone Manager...
- Database ...
- External API ...

Component View - UML Component Diagram

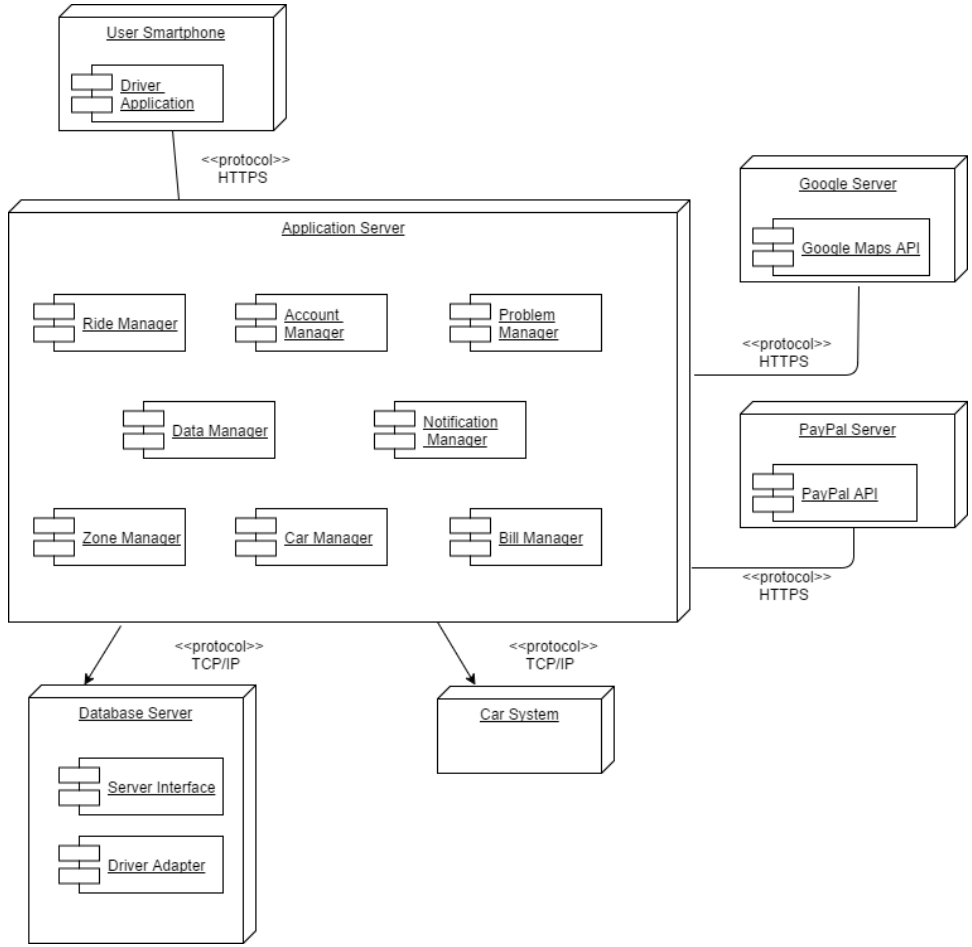


Deployment View

The best way found to deploy the software components identified, is to consider 4 different nodes (7 if considering the Google Server and the PayPal server) that correspond with the **Tiers** with in addiction the **External Extentions**

Deployment View - UML Deployment Diagram

The following diagram shows how **software components** are mapped into the **physical system**.



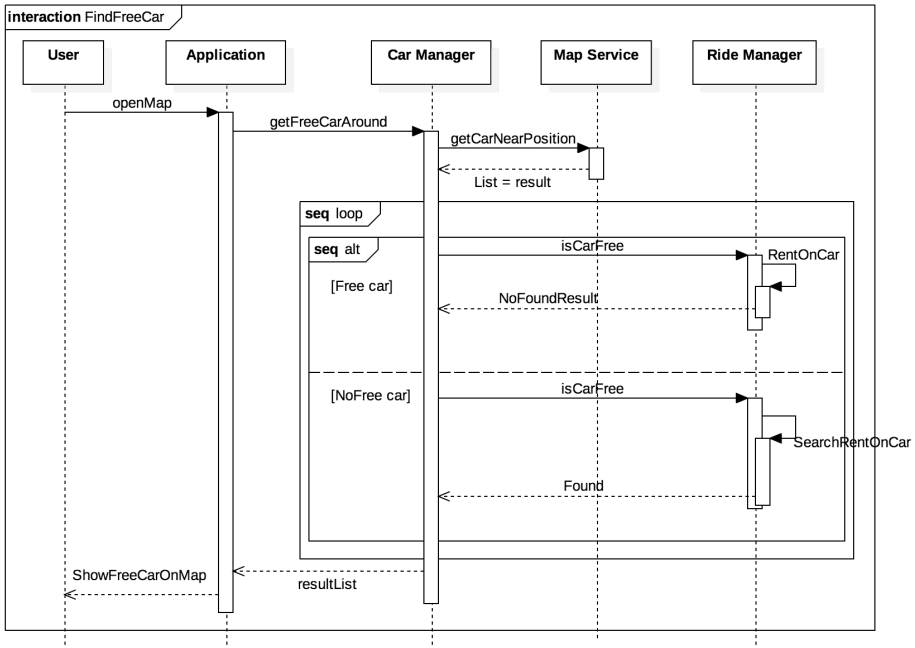
Runtime View

Here are proposed some of the most meaningful **UML Sequence Diagrams** with respect to show how software components interacts. The chosen functionalities are:

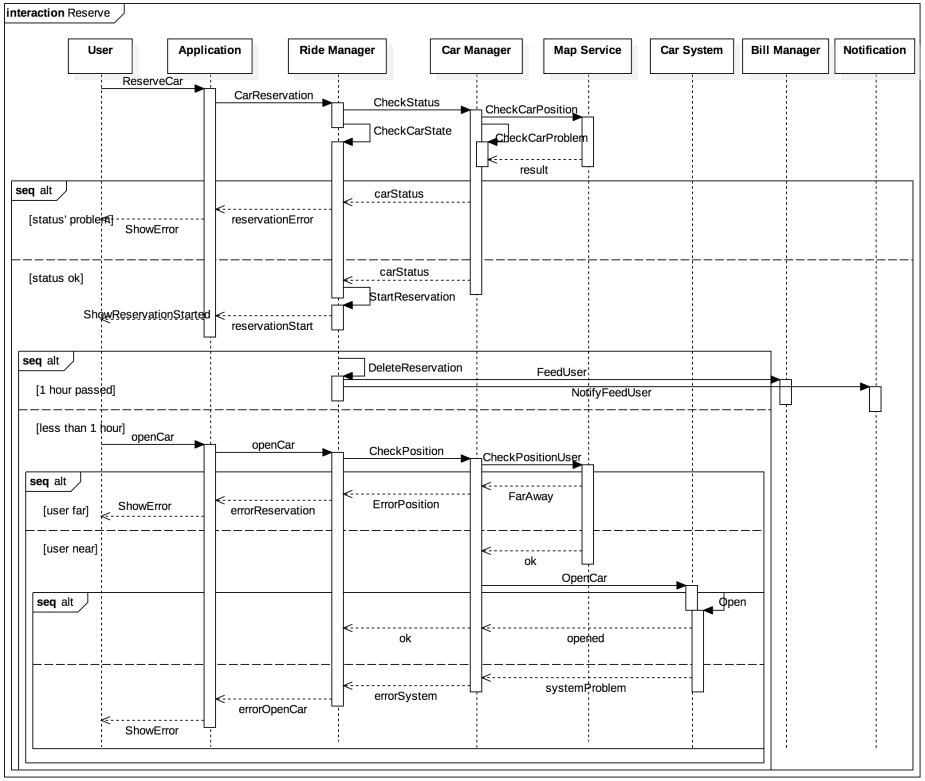
- Find a FreeCar
- Make a Reservation

There are other functionalities whose **UML Sequence Diagram** is not reported here for space and time constraints:

Find a FreeCar



Make a Reservation



Other design decisions - Car System

We assumed to a Car system that can completly control any important aspect of the car as:

- **Open the car:**
- **Count the Passenger:**
- **Display Message thought a navigator' LCD:**
- **Check car' position:**
- ....

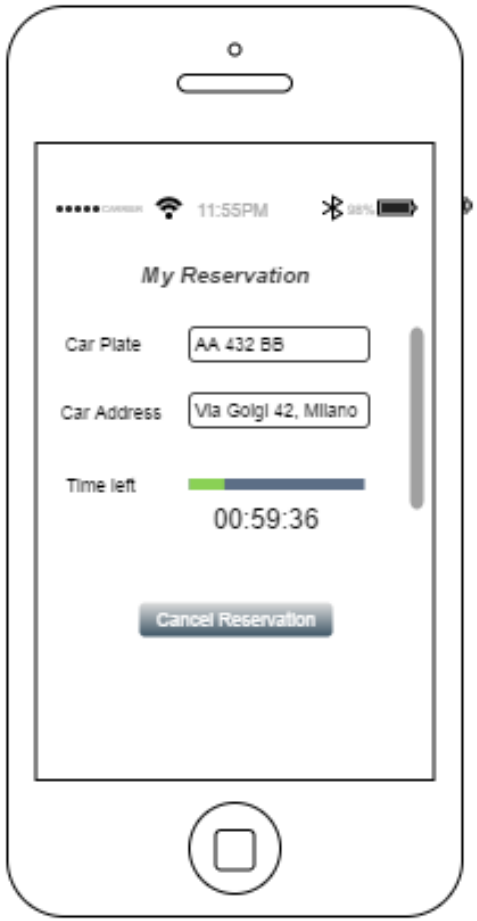
Outline

GUI Design

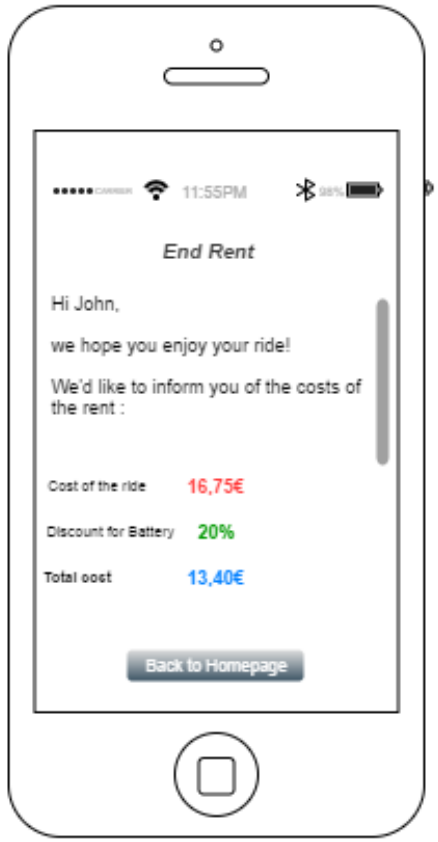
In this section we provide the **most important and meaningful mockups** for every class of screens we have designed.

Car Info Mockups

Reservation Mockups



End Rent Mockups



Outline - Integration Test Plan

Outline

Integration Testing Strategy

This project phase is highly based on the **Design** one.

The **bottom-up integration testing approach** has been chosen, because for a medium sized project like *PowerEnJoy*, it is best to proceed step by step in a careful yet coherent integration strategy.

We have considered all the **Subsystems**. and we had divided into 3 different categories, in particular we divided the system in part based on the drivers' number. Trying to minimize the number of drivers to use in a **bottom-up** strategy.

Entry Criteria

Before starting the integration testing of any software component that has been designed for *PowerEnJoy* system, few points have to be underlined:

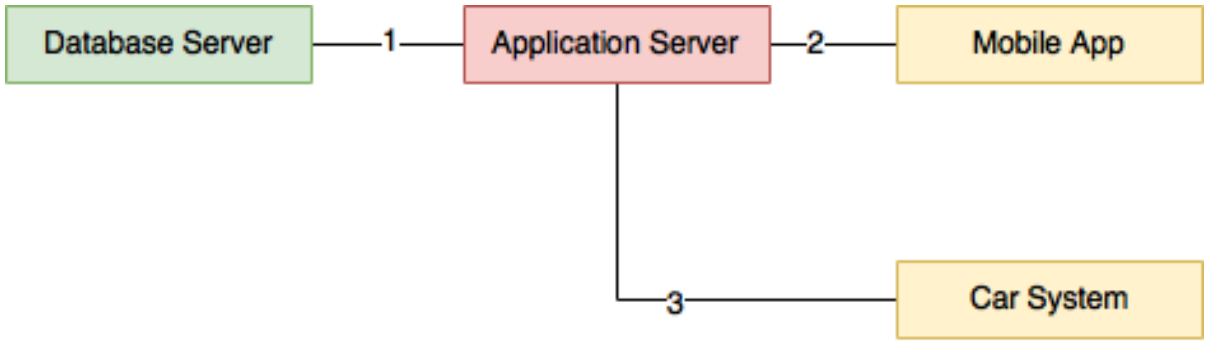
- The **internal functions** of the considered component must be **unit tested** using an appropriate framework.
- We suppose that **Google Maps API and PayPal API** are well tested by **Google and PayPal** and thus we can use them without testing any further.

Outline

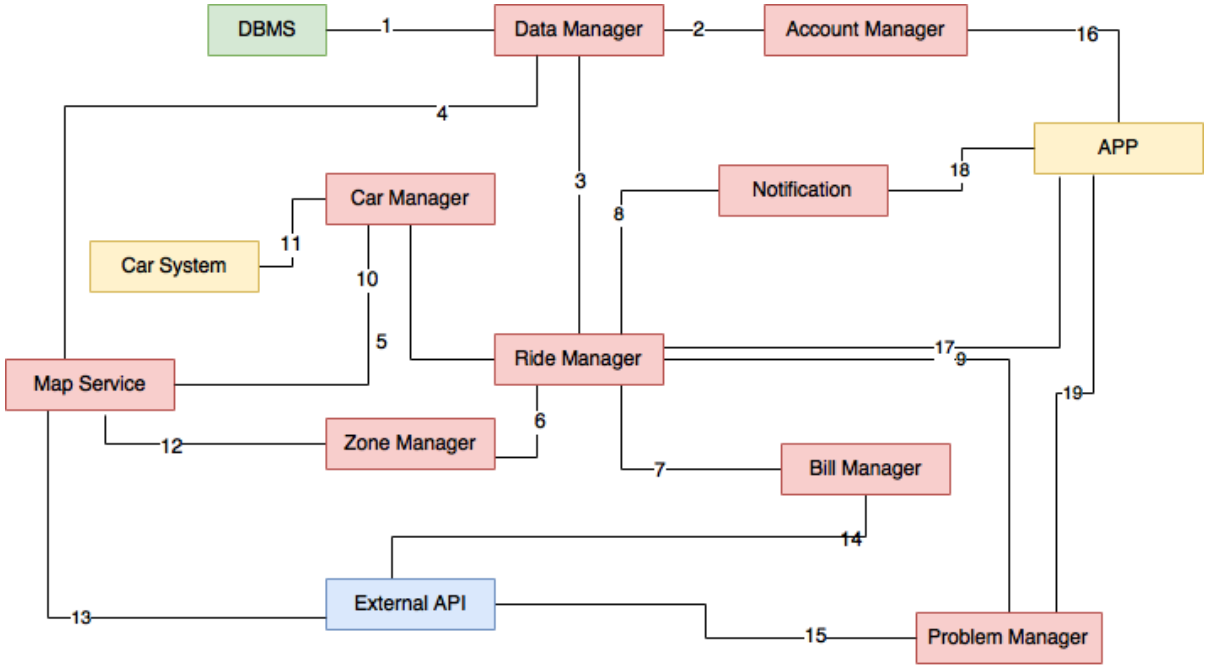
Convention adopted - Blocks

- **Green:** This block is not dependent on any lower level component in *PowerEnJoy* and therefore it is integrated as a starting point in the current diagram.
- **Red:** This block is going to use some Drivers in order to perform a complete test on all its functionalities
- **Yellow:** This block is going to be tested without Drivers because it's at the end of the process.
- **Blue:** This block will be not tested.

System Integration Sequence



Subsytem Integration Sequence



Outline - Project Plan

Outline

Plan Contents

The **Project Plan** consists in tables, **Gantt diagrams**, charts and natural language descriptions of the planning, scheduling and management of *PowerEnJoy* development.

In order to estimate the project effort, we followed the assumption that the **dimension of the software** can be characterized by correlating **the kind of functionalities offered** with **the source lines of code (SLOC) of the software itself**

Outline

Function Points Approach

The **Function Points approach**, defined in 1975 by Allan Albrecht:

- Consists in a technique to assess the effort needed to design and develop custom software applications.
- Correlates the kind of functionalities offered with the source lines of code of the software itself.

Function Points Approach

This technique consists in combining the following program characteristics to obtain a final result:

- **Internal Logic Files**
- **External Logic Files**
- **External Input**
- **External Output**
- **External Inquiry**

Function Points Summary

All the calculated FP_i sums up to FP, which is the total Function Points value:

$$\begin{aligned} FP &= FP_{ILF} + FP_{ELF} + FP_{EI} + FP_{EO} + FP_{EIQ} \\ &= 80 + 10 + 35 + 16 + 8 \\ &= 149 \end{aligned}$$

COCOMO II - Parameters

COCOMO II - Constructive Cost Model

Software Size Sizing Method: Function Points

Unadjusted Function Points: 149 Language: Java

Software Scale Drivers

Precedentedness	Nominal	Architecture / Risk Resolution	Nominal	Process Maturity	High
Development Flexibility	High	Team Cohesion	Very High		

Software Cost Drivers

Product		Personnel		Platform	
Required Software Reliability	Nominal	Analyst Capability	Nominal	Time Constraint	Nominal
Data Base Size	Low	Programmer Capability	High	Storage Constraint	Nominal
Product Complexity	High	Personnel Continuity	High	Platform Volatility	Low
Developed for Reusability	High	Application Experience	Nominal	**Project**	
Documentation Match to Lifecycle Needs	Nominal	Platform Experience	Nominal	Use of Software Tools	High
		Language and Toolset Experience	High	Multisite Development	Nominal
				Required Development Schedule	Nominal

Maintenance Off

Software Labor Rates

Cost per Person-Month (Dollars) 1500

Calculate

COCOMO II - Results

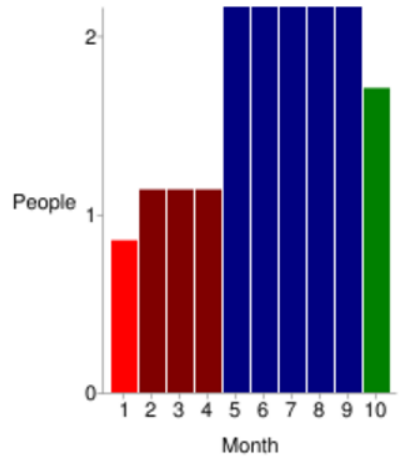
Results

Software Development (Elaboration and Construction)

Effort = 16.4 Person-months
Schedule = 9.2 Months
Cost = $24660

Total Equivalent Size = 7897 SLOC

Staffing Profile



Acquisition Phase Distribution

Phase	Effort (Person-months)	Schedule (Months)	Average Staff	Cost (Dollars)
Inception	1.0	1.2	0.9	$1480
Elaboration	3.9	3.5	1.1	$5918
Construction	12.5	5.8	2.2	$18742
Transition	2.0	1.2	1.7	$2959

Software Effort Distribution for RUP/MBASE (Person-Months)

Phase/Activity	Inception	Elaboration	Construction	Transition
Management	0.1	0.5	1.2	0.3
Environment/CM	0.1	0.3	0.6	0.1
Requirements	0.4	0.7	1.0	0.1
Design	0.2	1.4	2.0	0.1
Implementation	0.1	0.5	4.2	0.4
Assessment	0.1	0.4	3.0	0.5
Deployment	0.0	0.1	0.4	0.6

Your output file is http://csse.usc.edu/tools/data/COCOMO_January_20_2017_16_05_48_617569.txt

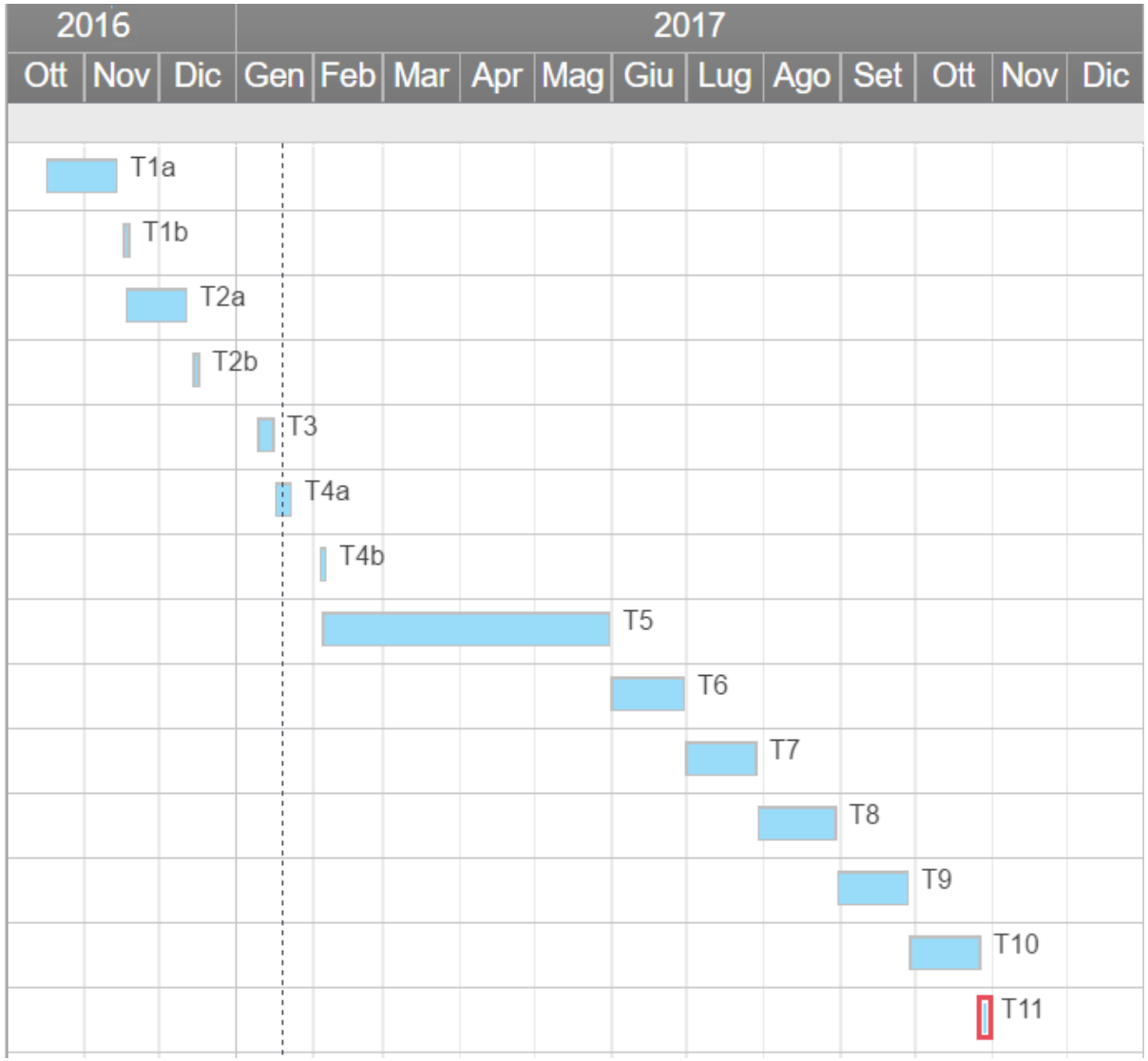
Created by Ray Madachy at the Naval Postgraduate School. For more information contact him at rjmadach@nps.edu

Outline

Tasks

Task	Description	Completed?
T1a	RASD - Writing	Yes
T1b	RASD - Presentation	Yes
T2a	DD - Writing	Yes
T2b	DD - Presentation	Yes
T3a	ITPD - Writing	Yes
T3b	ITPD - Presentation	Yes
T4a	PPD - Writing	Yes
T4b	Final Presentation	Yes
T5	Implementation	No
T6	Unit Testing	No
T7	Integration Testing	No
T8	System Testing	No
T9	User Acceptance - Alpha Testing	No
T10	User Acceptance - Beta Testing	No

Gantt Diagram



End

Thank you for your attention.

Appendix- Next version

We could also keep updating the system. For example we could add an **Admin workstation** in order to allow the owner to have a control on the data and to maintain the system by itself.

Appendix- Algorithm

Event	***Consequences***
D exits C	$S.startChecking()$

Check the distance between the SA and the current position	**if** $sA.nearest() - D.currPos() \geq 3$ **then**
	  $D.applyTax()$
	else
	  if $i + k \leq maxval$ **then**
	  $D.applyDiscount()$

Check the number of passengers	**if** $LoP.size() \geq 2$ **then** $D.applyDiscount()$

Check the battery state	**if** $B.getState() \leq 20$ **then**
	  $D.applyTax()$
	if $B.getState() \geq 50$ **then**
	  $D.applyDiscount()$

D ends the rent	$C.status \leftarrow Ready$
D has 5 minutes to charge the car and take a discount	$oldState \leftarrow B.getState()$ $wait(5)$ **if** $B.getState() \geq oldState$ **then** $D.applyDiscount()$