POLITECNICO
MILANO 1863

PowerEnJoy

Project Plan Document

Version 1.0

Authors:
Mosciatti Simone
Zanzottera Sara

Reference Professor:
Mottola Luca

January 22, 2017

Contents

1 Introduction

1.1 Purpose and Scope

The main purpose of this Project Plan Document is to assess the expected complexity of PowerEnJoy, , evaluate the risks related to the project, estimate costs and effort we expect will be required in order to develop our software successfully.

This document is also meant to provide some guidelines for the project manager to allocate budget and resources to the project, and to help defining a schedule for the development team.

We are using Function Points and COCOMO II analysis to estimate the size of the project, in terms of lines of code and costs and effort required to get the final product in time.

Then we provide a tentative schedule and a Gantt covering all the activities required to complete the project, from the very first operation to the actual deployement, that should help the managers defining a schedule. We also describe the expected load allocation for the team members.

Finally we provide a risk analysis for the most probable setbacks the project may face, and provide some preventive measures that should be performed in order to prevent a later, not recoverable failure in providing a software matching the expectations, in terms of feature, quality of code, costs and timing.

1.2 Definitions, Acronyms, Abbreviations

RASD Requirements and Specification Document.

DD Design Document.

User A customer of PowerEnJoy using the service.

Staff Operator (Operator) An employee of PowerEnJoy which takes care of the cars.

Ride The action of getting onboard of a PowerEnJoy car, start its engine, drive to destination and park.

Issue Any problem a car may incur in, or a user may face while using the service.

Nearby Cars Available cars located within a maximum distance to a specific position.

Available Cars Cars whose Availability Status is set to "Available".]

Booking (Reservation) The act to reserve a car for a limited amount of time for future use by a user.

Driver Whoever is driving a regularly booked PowerEnJoy car.

Driving License The state's issued driving license of the user.

Fine A fine issued by the local law enforcing officers to a user while driving a PowerEnJoy car.

Safe Area An parking area, predefined by the company, where is possible to safely park the cars of the PowerEnJoy fleet.

Car's Onboard System The controll system of the car that is able to exchange data with the central system and to relevate operation parameters.

Customer's App An implementation of the system frontend tailored to the need of the customers.

Staff's App An implementation of the system frontend tailored to the need of the staff.

Central System (Main Server) The central system for PowerEnJoy . All the command and all the data are streamed, analyzed and used here.

GPS : Global Positioning System is a global navigation satellite system (GNSS) that provides location and time information in all weather conditions, anywhere on or near the Earth where there is an unobstructed line of sight to four or more GPS satellites.

Location Pair of integer values as provided by GPS sensors.

Payment Method Set of data relative to a credit card.

Identity ID Personal code provided by local authorities to uniquely identify citizens.

Driving License ID The unique code reported on every legal driving license.

Scanned License An high quality image of the driving license acquired by the car's onboard system.

FP Function Points.

ILF Internal logic file

ELF External logic file.

DBMS Database Management System.

API Application Programming Interface.

UI User Interface.

UX User Experience.

MVP Minimum Viable Product.

M2M Communication Machine to machine communication.

SLOC Single Line of Code.

1.3 Reference Documents

- *Assignments AA 2016-2017.pdf* (Assignments document given by the teacher)
- *Reqiurements And Specification Document* (referring to this project)
- *Design Document* (referring to this project)
- Reference for Functional Point `https://web.archive.org/web/20160516160212/http://www.softwaremetrics.com/fpafund.htm`

2 Project size, cost and effort

2.1 Size estimation

Here we use the Function Point Approach to determine the size of the project.

Since we already have well defined components and the interaction of those components between external and internal actors, it seems natural to use the complexity of their interactions to determinate the Function Points for each component.

Provided that all the components have an intrisic, similar complexity, it is reasonable to assume that components that interact with few other components are quicker and simpler to develop. In our opinion the real complexity lies in the connection of components, not in the components themselves.

The complexity weights we use from now on are the following:

Function Type	LOW	AVERAGE	HIGH
Internal Logical Files	7	10	15
External Interface Files	5	7	10
External Input	3	4	6
External Inquiry	3	4	6
External Output	4	5	7

Table 1: Function Point's complexity weights

2.2 Internal Logical Files

All the internal files are stored in a database.

The model for our database was discussed in the Design Document, but we report again the diagram to simplify the discussion.

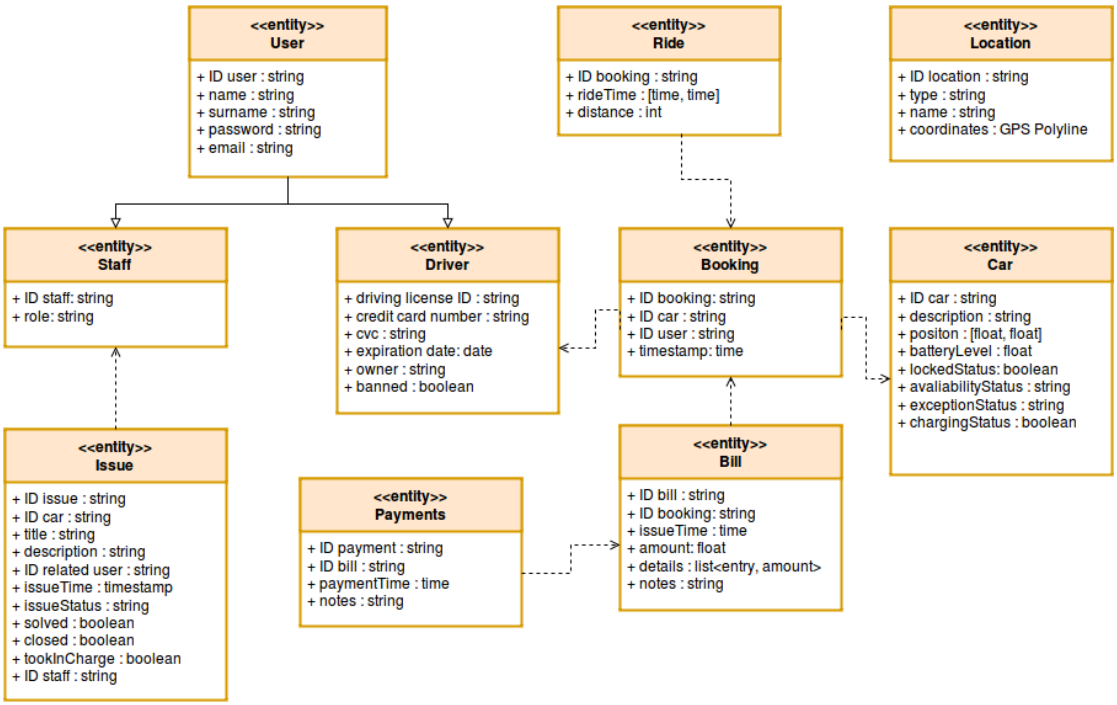

Figure 1: Class Diagram of the Model.

We are considering all the components of LOW complexity except for:

[noitemsep]BOOKING entity: HIGH complexity. it is the central entity in the whole application, with a lot of relationships with other components and very high insert rates. CAR entity: AVERAGE complexity. It has a relationship with BOOKING, but has a lot of very volatile fields and its update rates are very high.

Total for IFL

Total = 8 * 7 + 10 + 15 = 81

2.3 External Interface Files

The project involves 3 external data sources.

stripe We consider this service of AVERAGE complexity. They provide a very extensive set of endpoints, but the documentation is extremelly well done. In addition, we need to use only a small subset of the available endpoints and they provide several facilities to test the integration seamlessly.

linfo.io We consider this service of LOW complexity. Its only end point requires two inputs (center of the circle and radius) and return a set of objects inside that circle.

truelicense.com We consider this service of LOW complexity. Its APIs consist of only one single endpoint where to send an image, providing in output the number of the license of the input image, if valid.

Total for EIF

Total = 7 + 5 + 5 = 17

2.4 External Input

Following the list of functionalities provided in the Design Document, we identified the following External Input sources and assessed their complexity.

USER Component

USER/Register: **AVERAGE**
Some information need to be validated and it is necessary to be sure of the uniqueness of the users.

USER/{Login/Logout}: **LOW**
Only few check are necessary to LOG a user, such a validate the credentials.

USER/SetPaymentMethod: **AVERAGE**
It is necessary to coordinate the changes with the external system of paymet (stripe).

GEOLOCATION Component

GEOLOCATION/AvailableCar: **HIGH**
It handle geolocation queries. Its complexity should be AVERAGE, but we will put more effort in developing general routines in order to lower the complexity of all the other functionality that requires geolocation queries.

GEOLOCATION/Areas: **LOW**
We can reuse many routines from GEOLOCATION/AvailableCar to build this function with a minimum effort. On top of that, most of this work is delegate to the external service linf.io

GEOLOCATION/Issues: **LOW**
We can reuse many routines from GEOLOCATION/AvailableCar to build this function with a minimum effort.

GEOLOCATION/IsSafeArea: **LOW**
We can reuse many routines from GEOLOCATION/AvailableCar to build this function with a minimum effort.

POSITION Component

POSITION/Car: **AVERAGE**
This function requires the use of the car's onboard system and the communication system.

POSITION/User: **LOW**
This function uses the GPS inside the user device, however it is a pretty standard, well known function.

POSITION/Areas: **LOW**
Little more than a simple lookup in the database

BOOKING Component

BOOKING/Book: **LOW**
A simple write operation in the database. Most of the consistency checks are handled by the DBMS itself.

BOOKING/Unbook: **LOW**
A simple update in the database. Most of the consistency checks are handled by the DBMS itself.

BOOKING/Expire: **AVERAGE**
Other than modifying the database, this functionality needs to interact with the external payment system.

CAR Component

CAR/Unlock: **LOW**
Provided a reliable channel of communication with the car and a working on board system (assumption that I will keep for the rest of the documet), this functionality is a simple request to the car on board main system.

CAR/ValidateLicense: **LOW**
Most of the work is handled by the external component truelicense.com

CAR/Lock: **AVERAGE**
It is a simple command send to the car on board system. However, it has to trigger correctly the payment procedure starts.

CAR/TurnOff: **LOW**
It is a simple command send to the car on board system.

CAR/Telemetry: **HIGH**
By itself this component should have a LOW complexity. However, assigning a HIGH complexity we are considering the complexity of setting up a reliable communication channel between the main server and the cars.

CAR/SetStatus: **LOW**
It is a simple update in the database.

CAR/GetDetails: **LOW**
It is a simple read in the database. If data is too old, it also involves some communication from the server to the car, an operation still considered of LOW complexity.

RIDE Component

RIDE/Start: **LOW**
It simply performs an update in the database.

RIDE/End: **LOW**
It simply performs an update in the database.

ISSUE_MANAGER Component

ISSUE/New: **LOW**
It simply performs an update in the database.

ISSUE/TakeCharge: **LOW**
It simply performs an update in the database.

ISSUE/Solve: **LOW**
It simply performs an update in the database.

ISSUE/GiveUp: **LOW**
It simply performs an update in the database.

Total for External Input

Total = 18 * 3 + 6 * 6 + 2 * 6 = 102

2.5 External Inquiry

RIDE_MANAGER Component

RIDE/FindRides: **LOW**
It is a simple lookup in the database.

Total for External Inquiry

Total = 1 * 3 = 3

2.6 External Output

BILLING_MANAGER Component

BILL/CalculateRideFee: **AVERAGE**
It require to know the overall time of the ride and the bonus and malus applied.

BILL/CalculateExpireBookFee: **LOW**
It returns a constant value.

BILL/CalculateUnsafeParkingFine: **LOW**
It returns a constant value.

NOTIFIER

NOTIFY/Notify: **AVERAGE**
This functionality takes care of different types of notification. It is not "a size fits all".

Total for External Output

Total = 2 * 4 + 2 * 5 = 18

2.7 Overall count

The overal total is: 81 + 17 + 102 + 3 + 18 = 221
Lower bound of: 221 * 46 = 10166 lines of code
Upper bound of: 221 * 67 = 14807 lines of code.

3 Cost Estimation COCOMO II

In this section we are exploring the scale and the cost drivers for the COCOMO II model in order to find a reasonable timing for the project.

3.1 Scale Driver

Given the table listed in section 1.3: References Documents, we attribute the following values to the Scale Driver.

PREC (Precedentedness): 2.48
The team is Generally familiar with the problem space. It has never built the same product before, but it built several part of it a lot of times.

FLEX (Development Flexibility): 2.03
All the documents are written assuming that during the development phase a lot of unforeseeable issues would arise. We are expecting general conformity of the finished product to the designed one.

RESL (Architecture / Risk Resolution): 2.83
In the following part of the document we present a quite comprensive risk mitigation strategy.

TEAM (Team Cohesion): 1.10
The team is highly cooperative.

PMAT (Process Maturity): 3.12
The team already works in the industry, knows and applies best practise.

3.2 Cost Driver

RELY (Required Software Reliability): 1.00
In case of the interruption of the service, there won't be heavy financial losses but only easily recoverable ones.

DATA (Data Base Size): N/A
Our most populated table will have less than 100k rows on the foreseeable future. Given the lower bound of roughly 10k SLOC, our D/P ratio is of less than 10.

CPLX (Product Complexity): 1.00
Our code does not present any over complex structure, we will set to nominal the overall complexity.

RUSE (Developed for Reusability): 1.00
We are going to develop some functions for reusability across the project.

DOCU (Documentation Match to Life-Cycle Needs): 1.00
We are keeping the documentation in sync with our software's releases.

TIME (Execution Time Constraint): 1.00
The software is not particulary CPU bounded. We are expecting a fairly low use of CPU.

STOR (Main Storage Constraint): N/A
Storage is not an issue for our application.

PVOL (Platform Volatility): 1.00
We are going to base all our development stack on proven and widely used open source frameworks and components. We are conservatively setting this value to its nominal value.

ACAP (Analyst Capability): 0.85
We are confident that we conducted a fairly comprensive analysis of the overall project and so we set this value to a pretty high value.

PCAP (Programmer Capability): 0.88
We both have real word experience working in complex projects and in contributing to the open source.

PCON (Personnel Continuity): 0.81
There will be no turnover.

APEX (Applications Experience): 0.88
We chose technologies that we know and that we have already used before.

PLEX (Platform Experience): 0.91
We chose the platform basing on our previous experience.

LTEX (Language and Tool Experience): 0.91
We both have worked before with all the tools that we will need in the project as well as for the programming languange we selected.

TOOL (Use of Software Tools): 0.90
We are going to automate as much as possible the lifecycle of the product.

SITE (Multisite Development): 0.86
The team members are located in the same city and use all the technologies to communicate as much as possible.

SCED (Required Development Schedule): 1.00
We don't plan to inpose any schedule constraint on the project.

The overall result of the analysis of the cost driver is summed up in the following table.

RELY	Precedentedness	Nominal	1.00
DATA	Development Flexibility	Very Low	n/a
CPLX	Product Complexity	Nominal	1.00
RUSE	Developed for Reusability	Nominal	1.00
DOCU	Documentation Match to Life-Cycle Needs	Nominal	1.00
TIME	Execution Time Constraint	Nominal	1.00
STOR	Main Storage Constraint	Low	n/a
PVOL	Platform Volatility	Nominal	1.00
ACAP	Analyst Capability	High	0.85
PCAP	Programmer Capability	High	0.88
PCON	Personnel Continuity	Extra High	n/a
APEX	Applications Experience	High	0.88
PLEX	Platform Experience	High	0.91
LTEX	Language and Tool Experience	High	0.91
TOOL	Use of Software Tools	High	0.90
SITE	Multisite Development	Very High	0.86
SCED	Required Development Schedule	Nominal	1.00
Product			0,422

Table 2: Results of the analysis of cost drivers

4 Effort Estimation

Given the above points, we can estimate the required effort in Person-Month (PM).

We use the following formula:

$$\text{Effort} = A * EAF * KSLOC^{E}$$

Using the following substitutions:

$$A = 2.94$$

$$EAF = 0.422$$

$$E = B + 0.01 * sum(scaledriver)$$

we get E equals to:

$$E = 0.91 + 0.01 * 11.56 = 0.91 + 0.1156 = 1,0256$$

that leads us to the evaluations listed in table 3.

Now we calculate F as:

$$F = 0.28 + 0.2 * (E - B) = 0.28 + 0.2 * (1.0256 - 0.91) = 0.303$$

allowing us to determine also the duration's upper and lower bounds, also listed in Table 3.

Lower Bound	2.94 * 0.422 * 10.166 ** 1.0256	13.38
Upper Bound	2.94 * 0.422 * 14.807 ** 1.0256	19.68
Duration	3.67 * Effort ** F	?
Duration Lower Bound	3.67 * 13.38 ** 0.303	8.05
Duration Upper Bound	3.67 * 19.68 ** 0.303	9.05

Table 3: Effort Estimates

5 Schedule

Given that most of the analysis is already done, we are going to focus on the development schedule and the deployement of the software.

5.1 Timespan of the project development

Our risk mitigation strategy requires us to validate the product as early as possible. This request dictates slower development cycles, but ensures the quality of the final result we are getting, which is very relevant also from a business perspective.

We can accept a slower development schedule, since our time estimation are quite short. Nevertheless we account for an additional month of time, bringing the total time of development to roughly 10 person months.

We believe this is a reasonable tradeoff between the use of time and the risk management.

5.2 Deployement stages

The very first step consists in developing a very simple MVP that selected alpha users can use to rent cars once or twice a day for a small nominal fee.

A careful selection of the alpha testers should prevent the cars from being damaged, and prevent abuses on the alpha version of the system. At the same time, this process provides us extremelly valuable feedback on the system we are building and the interaction with the users.

During the MVP phase, techinical staff should be always present in order to manually fix any kind of problem that may arise.

After this phase, if the stackholders of the project agree, we proceed with the creation of the real platform and we decide what components of the MVP to keep (if any). The experience gained building the MVP will be extremelly valuable also after this phase, because will let us moving very fast and with confidence. During this phase we develop also the unit and the integration tests.

We aim to keep a clean building/deploy pipeline. In this way we can always deploy our platform. Keeping such pipeline is expensive and time consuming, but it provides a lot of benefits. With a clean pipeline we can:

> [noitemsep]Mirror the alpha traffic to the new service. Move alpha users to the new service and effectively make them beta users. Add more beta user to the service.

Using this approach the developing time will be a little longer, but we are keeping the risk as low as possible. We can afford this since our design is extremely decoupled and simple to build, so our developing time is short.

5.3 Detailed Schedule

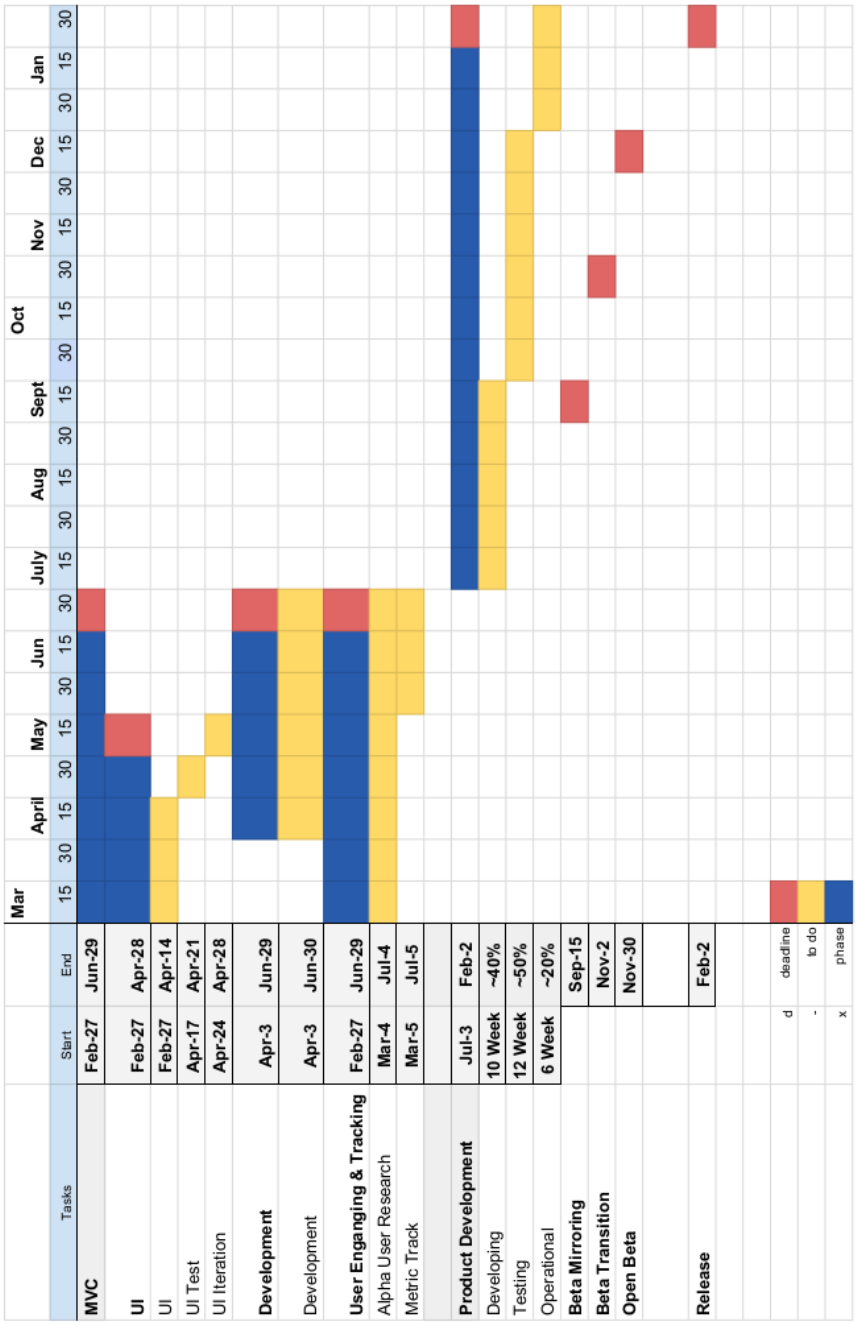

Figure 2: Gantt Chart for PowerEnJoy

In the above Gantt chart we did not actually specified what time slot is associated with which activity. We simply gave an estimate of the time percentage each activity should take on the whole time period.

This approach is required since all the activities on each module should procede in parallel with code and tests development, while the setting of the operational activity is set to start shortly thereafter.

These methodologies help us incorporating the feedback from both the user and the test environment as soon as possible during the development cycle.

For example, we may discover that the user is eager to pay something to keep the car booked for more time than a single hour: using this method we will be able to incorporate such feedback immediately in the system. Moreover, if we discover that some of our modules are not able to keep up with the expected load, we could immediately allocate more resources and time to fix that side of the project.

6 Resource Allocation

Our small team's members have different and complementary skill.

Sara is more experienced in UI / UX and in clearly communicating with the stakeholders; Simone is more experience in backend developing and M2M communication.

Both of us have a fairly good generic programming experience.

Since the components are already defined, as well as their interface, only a single developer will be responsible for every single component.

In order to fairly allocate the several components to each developer, we use the complexity factor calculated above.

Given the different skills and experience of the developing team members, the two developer are allocated to the different modules in the following way:

Simone

1. GEOLOCATION
2. POSITION
3. CAR
4. RIDE

Sara

1. USER
2. BOOKING
3. ISSUE
4. BILL
5. NOTIFY
6. User Interface

In this way both developers will handle closely related modules while working in the field they know best.

7 Risk Management

In the following section we list the possible risks that can determine the failure of our project, and we set a strategy to prevent each of them.

7.1 Risks and Selected Stretegies

1. **Risk: Users don't understand the User Interface**
   Selected strategy:
   [noitemsep, nolistsep]
   - Adopt standard symbols and icon
   - Test first mock up of the UI with target demography

2. **Risk: Software more complex than our estimation**
   Selected strategy:
   [noitemsep, nolistsep]Set several milestones along the development process, so to catch as early as possible if the project is going to run late.

3. **Risk: Users don't like the product**
   Selected strategy:
   [noitemsep, nolistsep]Provide a MVP. Get feedback from the market as soon as possible.
   (It is fine to loose money during this phase).

4. **Risk: Low Quality software**
   Selected strategy:
   [noitemsep, nolistsep]Automatic test regarding both performance and corecteness.

8 Conclusions

8.1 Tools used

During the development of this document we used the following tools:

- **Github** to version control the project
- **LaTeX** on TeXworks to redact this document
- **GoogleDocs** to redact the Gantt diagram

8.2 Hours of work

- SM: 5h on 16/01
- SZ: 5h on 16/01
- SM: 7h on 17/01
- SZ: 2h on 17/01
- SM: 3h on 18/01
- SZ: 2h on 18/01
- SZ: 1h on 22/01