Software Engineering 2: PowerEnJoy

Project Planning Document

Nardo Loris, Osio Alberto

Politecnico di Milano

1 Introduction

1.1 Purpose and Scope

This is the Project Planning Document for the PowerEnJoy car sharing system. This document has to be read after the Requirements Analysis and Specification Document and the Design Document. It's aim is to provide a (retrospective) analysis of the project complexity, size and effort, and an overview of the steps leading to the realization of the system. This document is intended to be read by the stakeholders, that can find here information about complexity and time required by the project and about risks, and by core development and design team, that can find informations about the activity assigned to them and their schedule.

1.2 Definitions, acronyms and abbreviations

- **System**: the system to be developed for PowerEnJoy
- **User**: a generic person interacting with the system
- **Employee**: a person who works for PowerEnJoy
- **Actor**: can refer to both users and employees
- **Car**: an electric vehicle owned by the company
- **Bill**: an amount of money a user has to pay. It is related to only a single ride
- **Pending bill**: a bill that the user has not paid yet
- **Paid bill**: a bill that the user has already paid for
- **Area**: a space delimited by a polygonal line whose vertices are a set of geographical coordinates
- **Geographical coordinates**: a tuple of latitute and longitude describing a location on Earth
- **Geographical region**: an area where a user can reserve at most one car. They do not overlap
- **Safe area**: an area where it is possible to park a car and optionally to recharge it, in order to make it available for another user
- **Safe parking area**: a special safe area where it is not possible to recharge the car
- **Recharging station area**: a special safe area where it is possible to plug the car for recharging its battery
- **Registered user**: a user who has completed the sign up process
- **Logged user**: a user who has completed the log in process and has not yet started the log out process
- **Banned user**: a registered user who cannot reserve a car until all his pending bills are estinguished
- **Reservor user**: the user who has made a reservation for the specific car. A user is considered the reservor user of a car until the reservation expires or the user is charged with the bill
- **Available car**: a locked car for which no reservation exists
- **Reserved car**: a locked car for which it exists a user who has reserved it
- **Becoming available car**: an unlocked car is said to be "becoming available" as soon as all the passengers and the driver of this car exits the car, the doors of the car are closed and it is parked in a safe area
- **In maintenance car**: a locked car is said to be "in maintenance" as soon as its battery level is below 20%, the car cannot be reserved
- **In use car**: an unlocked car which is not becoming available
- **GPS**: A system capable of providing the location of a receiver device with a good precision (5 meters)
- **Overlapping areas**: Two areas are said to be overlapping if there exists at least one geographical coordinates which is contained inside the two areas
- **Expiration of a car reservation**: when a reservation expires, the car becomes available again, the reservor user loses his reservation and he is charged a fee of 1€
- **Percentage delta**: a discount or a raise based on percentage
- **Applying a raise** or **a discount**: The operation of increasing or reducing the amount of a bill for a specific reason. The amount is computed just before the system charges the user of a bill, and then all those amounts (each one related to a specific reason) are algebraically added to the same bill.
- **RASD**: Requirements Analysis and Specification Document
- **DD**: Design Document
- **ITPD**: Integration Test Plan Document
- **DBMS**: DataBase Management System
- **COTS**: Commercial Off The Shelf
- **OWASP**: Open Web Application Security Project https://www.owasp.org/
- **XSS**: Cross-site Scripting
- **CSRF**: Cross-site Request Forgery
- **FPs**: Function Points

1.3 Reference documents

- Project rules of the Software Engineering 2 project
- Template for the Design Document
- Requirement Analysis and Specification Document (previous deliverable)
- Design Document (previous deliverable)
- Function Point definition and calculation http://www.functionpointmodeler.com/
- Function Point constants for estimating SLOC http://www.qsm.com/resources/function-point-languages-table
- COCOMO II specification
  http://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_modelman2000.0.pdf

1.4 Document structure

In the first section of this document we are going to provide first an estimation of the size of the project using Function Points, and right after an estimation of the required effort calculated applying COCOMO II.
In the second and third sections we are going to provided a detailed schedule of the project and to allocate all the defined activities to out team members.
In the last section we are goign to carry out a risk analysis for the project and to elaborate a mitigation strategy for each of the risks presented.

2 Project size, cost and effort estimation

2.1 Size estimation: Function Points

Function Point is an algoritmical methodology aimed at estimating the size of a project, basing on a combination of program characteristics, categorized in:

- **Internal Logic Files (ILFs)**, which looks at how the program stores the data
- **External Interface Files (EIFs)**, which looks at the data used by the system but managed by thirdy parts
- **External Inputs (EIs)**, which looks at the elementary operations involved in processing data coming from the users of the system
- **External Outputs (EOs)**, which looks at data generated by the system and destinated to the external environment
- **External Inquiries (EQs)**, which look at elementary input / output operations

In the following paragraphs we are going to provide both the measuring criteria for each of the cathegories and to apply those criteria to the proposed system for PowerEnJoy car sharing. The measuring criteria come from statistical analysis of real projects, which have been normalized and condensed into the given tables.

2.1.1 Internal Logic Files

2.1.1.1 Measuring criteria

	Data Elements		
Record elements	**1 - 19**	**20 - 50**	**51+**
1	Low	Low	Average
2 - 5	Low	Average	High
6+	Average	High	High

Complexity	FPs
Low	7
Average	10
Height	15

2.1.1.2 System analysis

In order to measure the FPs for a specific Internal Logic File, we need to analyze the structure of the data contained in it.

- **Account data**
  This file contains informations about accounts of people interacting with the system, i.e. users and employees. This accounts for 2 record elements. For each account we have a username and a password, then users have a much more detailed representation than employees, and so they are also characterized by name, date of birth, driving licence number and credit card data, which contain owner, number, expiration date and CVV. Overall, this accounts for 9 Data Elements, so this ILF is considered of low complexity.
- **Reservation data**
  This file contains informations about reservations. For each reservation, we store the reservor user, the reserved car and the creation time. This accounts for 1 Record Element and 3 Data Elements, hence this ILF il considered of low complexity.
- **Ride data**
  This file contains informations about rides. For each ride, we store the car involved in the ride, its reservor user, the time at which the ride started and the number of passengers that were in the car at the begin of the ride. This accounts for 1 Record Element and 4 Data Elements, hence this ILF is considered of low complexity
- **Car data**
  This file contains informations about cars. For each car, we store the plate, the battery charge level, the position

expressed as a tuple of latitute and longitude, the safe area where the car is parked in and the geographical area where the car is parked in. This accounts for 1 Record Element and 6 Data Elements, hence this ILF is considered of low complexity.
- **Safe area data**
  This file contains informations about safe areas. There are two types of safe areas: safe parking areas and recharging stations, hence this accounts for 2 Record Types. Each safe area is described by an identifier and the set of polygons which delimit it. Furthermore, recharging stations also need to have the number of plugs specified. This accounts for 3 Data Elements, so this ILF should be considered of low complexity. However, due to the intrinsic complexity of the *multipolygon* data type, which is used to store the bounding polygon, we decided to consider this as of average complexity (also because data elements refer to elementary data, not to structured data, hence multipolygon violates this hypothesis and we must correct the estimation).
- **Geographical area data**
  This file contains informations about safe areas. Just like safe areas, each geographical area is described by an identifier and a set of polygons which delimit it. For the very same reasoning done for safe areas, this ILF is of average complexity

Here is a wrap up table

ILF	Complexity	FPs
Account data	Low	7
Reservation data	Low	7
Ride data	Low	7
Bill data	Low	7
Car data	Low	7
Safe area data	Average	10
Geographical area data	Average	10

Overall sum of calculated FPs gives **55 FPs**

2.1.2 External Interface Files

2.1.2.1 Measuring criteria

	Data Elements		
Record elements	**1 - 19**	**20 - 50**	**51+**
1	Low	Low	Average
2 - 5	Low	Average	High
6+	Average	High	High

Complexity	FPs
Low	5
Average	7
Height	10

2.1.2.2 System analysis

External Interface Files strongly depends on the system which PowerEnJoy interfaces with. These systems are:

- The external system for address translations, which carries out the geocoding of given addresses
- The external system for map rendering, which displays maps in the user application
- The payment processing system, which is responsible both for credit card data validation and payment processing
- The driving licence validation system, which is responsible for driving licence validation

For each of these systems, we need to analyze the amount of data that pass through the interface

- **Geocoding system**
  This interaction is fairly simple, as we only send a string (the address to geocode) and get back a tuple of two coordinates (latitute and longitude). This accounts for 2 Record Elements and a overall count of 3 Data Elements, hence this EIF is considered of low complexity.
- **Map retrival system**
  This interaction is again fairly simple, as we only send a tuple of latitute and longitude on which the map has to be centered and a set of coordinates to highlight on the map, and get back an image. This accounts for 2 Record Elements, and a overall count of 3 Data Elements (even if one of them, we can consider it simple because it's just a list of pairs of floating point numbers). Hence, this EIF is considered of low complexity.
- **Payment processing system**
  This interaction is not just simple like the two before, as it has to arry out two tasks. For credit card validation, we need to send the credit card parameters (owner, number, CVV and expiration date), and get back a boolean answer. For payment processing, we again need to send credit card parameters plus the amount of the bill to pay, and then get back another boolean value indicating whether the payment was successful or not. Overall, this accounts for 3 Record Elements (credit card validation request, payment request, operation response) and 6 Data Elements. Hence, this EIF can be considered of low complexity again.
- **Driving licence validation system**
  This interaction is the most simple in this section. We send the driving licence number and the driving licence owner and get back the validity of the licence. Hence, we have 2 Record Elements and an overall count of 3 Data Elements(driving licence owner, number and validity). Hence, this EIF is considered of low complexity.

Here we have a wrap up table

EIF	Complexity	FPs
Geocoding	Low	5
Map retrival	Low	5
Payment processing	Low	5
DL validation	Low	5

So far, EIF analysis yields a overall sum of **20 FPs**

2.1.3 External Inputs

2.1.3.1 Measuring criteria

	Data Elements		
File Types	**1 - 4**	**5 - 15**	**16+**
0 - 1	Low	Low	Average
2 - 3	Low	Average	High
4+	Average	High	High

Complexity	FPs
Low	3
Average	4
Height	6

2.1.3.2 System analysis

In order to compute FPs for External Inputs, we must consider for each input source (e.g. each API exposed to users or external systems) the complexity of the interaction, measured in terms of both the exchanged data and the ILF / EIF it interacts with. The follwing list carries out the analysis for each external input of the system. The count of the exchanged data is done basing upon the API description carried out in the Design Document.

- **GET /users/{id}/login**
  This is a fairly simple functionality, as it evolves only the exchange of two data and interacts with only one ILF (the accounts one). Hence we can consider it of low complexity.
- **POST /users/{id}/register**

With respect to the previous one, this is a fairly much complext functionality, as it involves the exchange of 9 Data Elements, but interacts with 1 ILF (the accounts one) and 2 EIF (the payment processing system and the driving licence validation system), hence it has to be considered of average complexity.

- **DELETE /users/me/bills**
  As far as data exchange is concerned, this is a very simple input, as it has no attached data elements (except for the implicit session token). Instead, it must interact with 1 ILF (the bill one) and 1 EIF (the payment processing system). Hence, we can consider it of low complexity
- **POST /users/{id}/reservations**
  This input requires 2 Data Elements, and interacts with 2 ILF (the user and the car ones, as specified in the DD) and 1 EIF (for address geocoding), so it can be considered of low complexity.
- **PATCH /cars/{plate}/unlock**
  This input requires 3 Data Elements to be specified, and interacts with only 3 ILF (the user one, in order to ensure that the user is not banned, the reservation one to ensure the user has a reservation for the car, and the ride one, in order to create the ride). So far, we can consider it of low complexity.
- **PATCH /area/geographicals/{id}/split**
  This input requires 2 Data Elements to be specified, and interacts with only one ILF (the geographical regions one). But if we consider well the required elements, we notice that we have a fairly complex, structured field required. Than, despite the numbers want this input to be classified as of low complexity, we have to classify it as of average complexity, for reasons which have already been explained for ILF related to geographical regions.
- **PATCH /area/geographicals/{id}/merge**
  This input requires two Data Elements to be specified, and interacts with only 1 ILF (the one concerned in geographical regions storage), hence we can consider it of low complexity. Despite the similarity, the fields of this input are ways simpler than those of previous one.
- **DELETE /area/safes/{id}**
  This input only require 1 Data Element to be specified, and only interacts with 1 ILF (the one concerned in safe areas storage), hence we can consider it of low complexity
- **POST /area/safes**
  The analysis of this input is pretty much similar to that of geographical areas split. In fact, we have 2 Data Elements, of which one fairly complex for the very same reasons as before, and we interact only with 1 ILF (again, the one concerned with safe area storage). Hence, for the same rationale, we consider it of average complexity.
- **PATCH /area/safes/{id}**
  The analysis of this input is identical to that of the previous one, so we consider it of average complexity for the same reasons as before.
- **Car status (message)**
  This is the only input which uses the messaging API. It is composed of 4 data elements, and only interacts with 1 ILF (the one concerned with car information storage). Hence, we can consider it of low complexity.
- **Payment completion notification**
  This input comes from the payment processing system when it processes a payment, and 2 data elements: the former tells if the payment was successful or not, and the latter is the identifier of the bill that was requested to be paid. This input involves an interaction with the ILF concerning bills in order to update the status of the bill. Hence, it can be considered of low complexity.

Moreover, we have to consider that each input (except for those belonging to the /users API) requires user credencials validation and user authentication, hence we must consider it in the overall count. We can model it just as a new input which involves the interaction with one ILF (the one concerned with user information storage) and 1 data element (the session token). Hence we can consider it of low complexity.

Here we have a wrap up table

EI	Complexity	FPs
GET /users/{id}/login	low	3
POST /users/{id}/register	average	4
DELETE /users/me/bills	low	3
POST /users/{id}/reservations	low	3
PATCH /cars/{plate}/unlock	low	3
PATCH /area/geographicals/{id}/split	average	4
PATCH /area/geographicals/{id}/merge	low	3
DELETE /area/safes/{id}	low	3
POST /area/safes	average	4
PATCH /area/safes/{id}	average	4
Car status message	low	3
Payment completion notification	low	3
User authentication	low	3

From the table above, EIs analysis yields a result of **43 FPs**.

2.1.4 External Outputs

2.1.4.1 Measuring criteria

	Data Elements		
File Types	**1 - 5**	**6 - 19**	**20+**
0 - 1	Low	Low	Average
2 - 3	Low	Average	High
4+	Average	High	High

Complexity	FPs
Low	4
Average	5
Height	7

2.1.4.2 System analysis

The system sometimes need to communicate with the external environment outside the context of an inquiry, and this is where External Output comes in. To evaluate the complexity of each external output, we must consider how many ILF or EIF the output is produced from, and the data elements which the output carries in. Here is a list of all the external outputs.

- **Ride duration**
  This output is produced on the display on board of the cars in order to inform the driver about how is the ride lasting. It only carries one data element (actually, the difference between two timestamps, the actual one and that one the ride begun at), and has to interact with one ILF (the one that stores data about rides). Hence, it can be considered of low complexity (4 FPs).
- **Lock (message)**
  This output is an async message sent over the message-driven architecture. It has no payload, but needs to interact with 1 ILF (the one concerning data about cars, in order to know where car has stopped). Hence it can be considered of low complexity (4 FPs).
- **Unlock (message)**
  Just like the previous one, this output is a message sent over the messaging API. It still have no payload, and need to interact with only one ILF (the one concerning reservations), hence we can consider it of low complexity (4 FPs).
- **Payment processing request**
  This output sends to the payment processing system a request to carry out a payment (and a payment is

described by the amount to pay and the credit card data to get this amount from, so other 4 data elements, which are the credit card owner, number, CVV and expiration date). It must deal with 2 ILF (the one concerning bills for the bill to pay and the one concerning users for the credit card data) and 1 ELF (the one concerning payment processing). Hence we can consider it of low complexity (even if it is quite a borderline classification)

The analysis of External Outputs yields an overall result of **16 FPs**.

2.1.5 External Inquiries

2.1.5.1 Measuring criteria

	Data Elements		
File Types	**1 - 5**	**6 - 19**	**20+**
0 - 1	Low	Low	Average
2 - 3	Low	Average	High
4+	Average	High	High

Complexity	FPs
Low	3
Average	4
Height	6

2.1.5.2 System analysis

As the system is built mainly with a client-server architectural style, there are many external inputs that require a response by the system which sents him some requested data. The difference with external input is mainly that in the context of an inquiry the server replies with an answer to question of the user without updating his internal status, rather than executing a user command in order to alter its internal status, like in external inputs.

In order to evaluate the complexity of an inquiry, we must understand which files it references and how many data elements are exchanged during both the request and the response phases.

Here is a list of all the inquiries, including those that involve thirdy part systems.

- **GET /users/{id}/bills**
  This is a fairly simple inquiry, it involves only 1 data element in the input side and an array of tuples of 2 data element in the output side. As of files, it only interacts with 1 ILF (the one dealing with bills). Hence, we can consider it of low complexity.
- **GET /users/{id}/reservations**
  This is a fairly complex inquiry, it involves 4 data element in the input side and an array of tuples of 4 data elements in the output side. Moreover, it has to interact with 1 ILF in order to carry out the task. However, it can still be considered of low complexity
- **GET /users/{id}/reservations/{plate}**
  This inquiry requires 2 data elements in the input side and 4 data elements in the output side, and still has to interact with only 1 ILF (the one concerning reservations), hence it can be considered of low complexity.
- **GET /cars**
  This inquiry is fairly complex for the amout of data elements involved, as it involves 5 data elements in the input side and an array of tuples of 7 data elements in the output side. However, it only interacts with 1 ILF (the one concerning cars), and thus it can be considered of low complexity.
- **GET /cars/{plate}**
  This inquiry is similar to the previous one, except for that the input side only requires 1 data element instead of 5. Still, we can consider it of low complexity.
- **GET /area/geographicals**
  The complex part of this inquiry is the output side. In fact, the input side requires no parameter, and as far as files are concerned, it only interacts with one file, the one concerning the storage of geographical regions. The output side is instead quite complex, as it is constituited by a list of areas, which are consitituted by sets of paths delimiting them, plus a few other data elements. For these reasons we can consider it of average complexity, just like we did in the whole analysis of FPs when we have to deal with data involving paths and polygons.
- **GET /area/geographicals/{id}**

Very similar to the previous one, this inquiry requires an input parameter instead of 0 and outputs only one region instead of a list. However, we still have to deal with paths and polygons like before, hence the complexity of this inquiry is still average.

- **GET /area/safes**
  This inquiry is again similar to the *GET /area/geographicals* one, except that it deals with safe areas instead of geographical areas. Still, it has no data element in input, a quite complex output, as it is differs only for a few data elements from the previous one, and involves only one ILF (the one that deals with safe areas). Again, this is a functionality of average complexity.
- **GET /area/safes/{id}**
  This inquiry, like the previous one, can be analyzed in the very same way of *GET /area/geographicals/{id}*. Hence we can consider it again of average complexity.
- **Driving licence validation**
  This inquiry requires as input the driving licence owner and number, and yields as output the validity of the given licence. It deals with only one EIF (the one concerning driving licence validation). Hence we can consider it of low complexity.
- **Address geocoding**
  This inquiry requires as input the only address to geocode, and yields as output the latitute and longitude of the given address. So far, it only deals with 3 data elements and 1 EIF (the one concerning geocoding), hence we can consider it of low complexity.
- **Map retrival**
  This inquiry requires only a tuple of latitute and longitude, plus a number of tuples of latitude and longitude to highlight on the map. As these points are usually position of cars, we can consider that it interacts with 1 ILF (the one concerning cars), 1 EIF (the one concerning map retrival) and 3 data elements, of which one is a structured one. Hence, we can consider it of average complexity.
- **Credit card validation**
  This inquiry requires in the input side 4 data elements (credit card owner, number, CVV and expiration date), and yields as output only 1 data element indicating whether the credit card is valid or not. It deals with 1 EIF (the one concerning payments processing). Hence, we can consider it of low complexity.

Just like for external inputs, we should consider the user authentication, but we can just take advantage of the functionality presented in the external inputs section and avoid counting it another time in this context.

Here is a summarizing table

EQ	Complexity	FPs
GET /users/{id}/bills	low	3
GET /users/{id}/reservations	low	3
GET /users/{id}/reservations/{plate}	low	3
GET /cars	low	3
GET /cars/{plate}	low	3
GET /area/geographicals	average	4
GET /area/geographicals/{id}	average	4
GET /area/safes	average	4
GET /area/safes/{id}	average	4
Driving licence validation	low	3
Address geocoding	low	3
Map retrival	average	4
Credit card validation	low	3

Overall sum of calculated FPs gives **44 FPs**.

2.1.6 Overall analysis

The following table summarizes the FPs analysis.

Function type	FPs
Internal logic files	55
External logic files	20
External inputs	43
External outputs	16
External inquiries	44
TOTAL	178

Considering Java Enterprise Edition as the deployment environment and ignoring the HTML, JavaScript involved in user and employee applications (which is just a very small part if compared to the server logic and the data structure development, and can be thought as presentation logic only, with no business logic in it), we can get an approximation about the number of line of code that this project will require.

Average approximation:

$$SLOC = 178 \times 46 = 8188$$

Conservative approximation:

$$SLOC = 178 \times 67 = 11926$$

The conversion rates are statistically derived from real projects.

2.2 Effort estimation: COCOMO II

COCOMO II is an algorithmical methodology aimed at estimating the effort (in person-hours) requried by a project, given its size (expressed in kilo lines of code). It is based on the evaluation of scale factors, which account for how much complexity scales for a unary change in size, and of cost drivers, which account for all the factors related both to project and process that can increase or decrease time required by the development.

In the following paragraphs, we are going to evaluate both scale factors and cost drivers. At the end of the evaluation, the last paragraph will contain the effort estimation.

2.2.1 Scale factors

In order to evaluate scale factors, we refer to the official COCOMO II table

For each scale factor, here is a concise explanation of the reasons of the evaluation:

2.2.1.1 Precedentedness (PREC)

Precedentedness accounts for the familiarity of our development team with projects that are similar to this one. In order to evaluate precedentedness, we need to evaluate several factors:

- **Organizational understanding of product objectives**
  Since we had a lot of feedback by the stakeholders, and the analysis of functional and non functional requirements was carried out very attentively, we decided to evaluate this parameter as **considerable**.
- **Experience in working with related software systems**
  As our team has never worked for sharing companies, and it is not very familiar in dealing with this domain, we evaluate this parameter as **moderate**.
- **Concurrent development of associated new hardware and operational procedures**
  This project does not require any specific hardware to be developed, as the only hardware needed is the sensor system of the car, which is already present on board of the very same cars. Hence, we decided to evaluate this parameter with the best evaluation possible: **some** (according to the official table).
- **Need for innovative data processing architectures, algorithms**
  This project does not requires innovative algorithms to be designed in order to handle the tasks he has to carry out. The only tricky part, that was specified in the DD, does not require any innovative algotithm, as all the algorithms used in it are quite known. Hence, we evaluate this parameter as **minimal**.

Averaging these parameters we obtain a rating for the PREC scale factor of **high**.

2.2.1.2 Development flexibility (FLEX)

Development flexibility accounts for the flexibility allowed in the development project with respect to requirements and external interface conformance. In order to evaluate development flexibility, we need ot evaluate several factors:

- **Need for software conformance with preestablished requirements**
  Since we have almost no legal constraint, but quite precise and strict functional requirements that derive from an accurate requirements analysis, we evaluate this parameter as **considerable**
- **Need for software conformance with external interface specifications**
  In this project, there are interactions with external systems, but as we discussed in the FP analysis, all systems have very simple interfaces (at least for the services that we are using), then we decide to evaluate this parameter as **basic**
- **Combination of inflexibilities above with premium on early completion**
  Even if inflexibilities at the previous points are not that serious, we cannot forget them as we still need to stick to them precisely, and this is always a time consuming activity, hence we decided to evaluate this parameter as **medium**.

Averaging these parameters we obtain a rating for the FLEX scale factor of **high**.

2.2.1.3 Architecture / Risk Resolution (RESL)

This scale factor accounts for awarness and reactiveness to risks, and it is strictly connected with the risk plan discussed later in this document. In order to evaluate this scale factor, we need to evaluate several parameters.

- **Risk Management Plan identifies all critical risk items**
  Even if the risk analysis was carried out very carefully, we cannot grant it is complete, hence we decided to evaluate this parameter as **generally**.
- **Schedule is compatible with Risk Management Plan**
  Since we have a quite tight schedule, the is not much window to handle risks that are time consuming or that are not included in the risk plan, hence we decided to evaluate this parameter as **little**
- **Percent of development schedule devoted to establishig architecture, given general product objectives**
  As it is clear from the Gantt Chart included in this document, the percentage of development schedule devoted to the architectural design is of **18.75%** (about 3 weeks over the overall duration of 16 weeks).
- **Percent of required top software architects available to the project**
  Since we have no other project running in parallel, we can devote the **100%** of our workforce to this project.
- **Tool support available for resolving risk items, developing and verifying architectural specs**
  As we don't have plenty of tools to verify architectural specs or resolving risk items (in fact, the main tool is *Alloy* specification language), we evaluate this parameter as **little**
- **Number and criticality of risk items**
  This can be read directly from the risk analysis, since we have only very few critical risks, we can evaluate this parameter assigning it a **nominal** level.

Averaging these parameters we obtain an overall rating for the RESL scale factor of **nominal**.

2.2.1.4 Team cohesion (TEAM)

The Team Cohesion scale factor accounts for the sources of project turbulence and entropy because of difficulties in synchronizing the project's stakeholders (from COCOMO specification). In order to evaluate the TEAM scale factor, we need to provide an evaluation of different parameters:

- **Consistency of stakeholder objectives and cultures**
  As almost all the stakeholders come from a quite similar cultural background (in fact, the project has quite limited geographical scope), we evaluate this parameter as *strong*
- **Ability, willingness of stakeholders to accommodate other stakeholders' objectives**
  Basing on our knowledge of the stakeholders involved in the project, we evaluate this parameter as **little** (in fact, PowerEnJoy manager only aims at earning as much as possible)
- **Experience of stakeholders in operating as a team**
  PowerEnJoy is a relatively new company, the feeling among stakeholders might not been so run-in, even if we don't are likely to find serious problems, hence we evaluate this parameter as **basic**
- **Stakeholder teambuilding to achieve shared vision and commitments)**
  For similar reasons as the second point in this list, we evaluate this parameter as **little**

Averaging this parameters we obtain an overall rating for the TEAM scale factor of **nominal**

2.2.1.5 Process Maturity (PMAT)

We are a very young software company, and we only reached CMM level 2 (process is characterized for projects but it is often reactive and heaviliy relies on the people making it work). So far, the PMAT parameter can be attributed a level of **nominal**.

2.2.1.6 Scale factor review

The following tables summarizes the evaluation of scale factors and maps the rating levels to the corresponding coefficients, according to the official COCOMO specification.

Scale Factor	Rating	Coefficient
PREC	High	2.48
FLEX	High	2.03
RESL	Nominal	4.24
TEAM	Nominal	3.29
PMAT	Nominal	4.68

Overall, the sum of the coefficients yields a value of S = 16.72.
From here, we can calculate the E parameter

```
E = B + 0.01 × S = 1.0772
```

2.2.2 Cost drivers

For cost driver estimation we use the *Post-Architecture* model, as we have already a well detailed design of the system architecture. This allows us to evaluate much more parameters than the *Early-design* model, thus we'll obtain a more precise estimation of th effort needed for the project.

For each cost driver, we provide the evaluation and a brief explanation of the rational undelying it.

2.2.2.1 Required Software Reliability (RELY)

This driver accounts for the increasing complexity that corresponds to increasing reliability requirements. According to the COCOMO specification, we can classify the effects of a software failures as *moderate, easy recoverable losses*, as we are not dealing with life-critical or economic-critical applications, so higher rating levels are not appropriate in this context.
Hence, RELY is rated at **nominal** level (coefficient: 1.00)

2.2.2.2 Data Base Size (DATA)

This driver accounts for the effort needed to develop a larger test database. Hence we neither need tons of test data nor deal with complex or unstructurable data or media files, we can estimate the database size in a range of 10 to 100 bytes per line of code (in a very pessimistic estimation, the overall database size would not be greater than 1MB).
Thus, DATA is rated at **nominal** level (coefficient: 1.00)

2.2.2.3 Product Complexity (CPLX)

This driver meaning is really implicit in its name. For the evaluation, we need to average the evaluation of 5 different areas

- **Control Operations**
  Code has in general a not so complex control flow graph, except for the algorithms that manipulate safe areas and geographical areas, which use a very nested, recursive control flow graph with lots of predicates, hence according to the COCOMO specification, we can rate it at a **high** level

- **Computational Operations**
  Even if the complexity of several algorithms is not trivial for the reasons pointed out just before, the underlying operations are kind of simple, there is no differential calculus, numerical analysis or heavy matrix computations, hence we evaluate this parameter as **low**
- **Device-dependent operations**
  The system under consideration does not require specific or specialized hardware, and more it carries out input/output operations using a textual protocol (HTTPS), hence we evaluate this parameter as **low**
- **Data management operations**
  The system requires the processing of moderately complex queries, but the database structure is fixed from the design phase, so we can evaluate this parameter as **low**
- **User Interface Management Operations**
  The user interface designed for the system is quite simple as we can see from the mockups, and can easily be implemented as a set of widgets, so this parameter can be rated **nominal**

Averaging these parameters we obtain a overall rating for the CPLX cost driver of a *nominal* level (coefficient: 1.00)

2.2.2.4 Developed For Reusability (RUSE)

This cost driver accounts for the additional effort to be put in developing and testing reusable components instead of domain specific components. As our company, as an internal policy, always plans to build as much reusable components as possible, in order to advantage in facing future problems, we must evaluate this cost driver as **high** (coefficient: 1.07), as we will put some effort in trying to achieve reusability.

2.2.2.5 Documentation Match to Life-Cycle Needs (DOCU)

This cost driver accounts for the effort that need to be put in writing documentation that fits well product lifecycle needs (and in particular maintenance). As the schedule is quite tight, but documentation is very important too, we try to write a quite extensive documentation that well fits product lifecycle and helps a lot in component reusability. Hence, this cost driver is assigned a rating level of **nominal** (coefficient: 1.00)

2.2.2.6 Execution Time Constraint (TIME)

This cost driver accounts for the relative time consumption of the system in performing the tasks it is intended to with respect to the total completion time. The system we have designed has not to carry out very time expansive operations, but it is planned to be used by many users at a time, thus requiring about the **70%** of the available computational time (coefficient: 1.11)

2.2.2.7 Main Storage Constraint (STORE)

This cost driver accounts for the degree of main storage required by this system to store its data. Hence it is not the case that the data manipulated by the system grow exponentially in time, and since there are no multimedia data, but mainly integers and strings, we can rate this cost driver at a level of **nominal**, which corresponds to a relative use of main storage with respect to available storage of less than 50% (coefficient: 1.00)

2.2.2.8 Platform Volatility (PVOL)

The main platform we work with are J2EE and browsers. As far as J2EE is concerned, platform is almost stable with a major release every about 2 years. The Java language has more frequent major releases, but it aims to be retrocompatible, so it does not present many problems of compatibility. Instead, browsers have more frequent releases (a release every 2 months approximatively), but the HTML5 specification (and the ECMAscript and CSS3 related specifications) are almost stable. Hence, we can rate this cost driver at **low** level (coefficient: 0.87)

2.2.2.9 Analyst Capability (ACAP)

This effort multiplier accounts for the aggregate ability of the analyst team to communicate, cooperate and design, it also account for the efficiency and thoroughness of the team. Since we have dedicated much time in the effort of analysing the problem requirements we can rate this effort multiplier at a level of **high** (coefficient: 0.85)

2.2.2.10 Programmer Capability (PCAP)

This effort multiplier accounts for the aggregate ability of the programmer team to communicate and cooperate, it also account for the efficiency and thoroughness of the team. Our cooperation started some years ago and since then we have already developed different software product and thus our ability to cooperate is rated **high** (coefficient: 0.88)

2.2.2.11 Personnel Continuity (PCON)

This effort multiplier accounts for the project's annual personnel turnover. Since our development team is stable, no personnel turnover is expected, and so we rate this as **very high** (coefficient: 0.81)

2.2.2.12 Applications Experience (APEX)

This effort multiplier accounts for the application experience of the team in developing the software system. Since this application field (car sharing) is new to our developing team we set this effort multiplier to **very low** (coefficient: 1.22)

2.2.2.13 Platform Experience (PLEX)

This effort multiplier accounts for the knowledge in many platforms such as graphic user interface, database, networking… Since this knowledge is around 1 year, we can rate this effort multiplier at a level of **nominal** (coefficient: 1.00)

2.2.2.14 Language and Tool Experience (LTEX)

This effort multiplier accounts for the level of the programming language and the software tool experience of the project team. Software development also includes program styles and tools for the requirements and design reprenentation and analysis. We have mixed experience in language and tools in fact we have a solid experience of more than 3 years in Javascript, HTML and CSS; an experience of about 1 year in Java, and some experience in modelling language such as Alloy. Considering all these factor we opted for a level of **nominal** (coefficient: 1.00)

2.2.2.15 Use of Software Tools (TOOL)

This effort multiplier accounts for the use of tools helping the team at different project stages. The tools can be from a simple debugger to a more integrated system and since we plan to use tools like Eclipse as IDE, Maven as dependency manager, GIT as versioning tool, and some tool for testing we can rate this effort multiplier at a level of **high** (coefficient: 0.90)

2.2.2.16 Multisite Development (SITE)

This effort multiplier takes into account two factors: the site collocation and the communication support. Even if we work from different city (**nominal** rate for site collocation) we plan to usually have meetings in Milan and in addition we plan to use chats and email so we can rate the communication support at a level of **very high**. Combining the two factors yields a global rate of level **high** (coefficient: 0.93)

2.2.2.17 Required Development Schedule (SCED)

This effort multiplier measures the schedule constraint imposed on the project team. While accelerated schedules tend to produce more effort in the earlier phases to reduce risks, we are fine with a normal schedule with no stretches, we rate this factor at a level of **nominal** (coefficient: 1.00)

2.2.3 Effort estimation

From scale factor evaluation we get a coefficient

```
E = 1.0772
```

From cost driver evaluation we get a overall multiplier of

```
M = 0.6393
```

Hence an overall pessimistic extimation yields a value of

$PM = A \times (size^{E}) \times M \approx 27.5 \text{ Person / Months}$
where A = 2.94, size = 11.926 [KSLOC]

2.2.4 Duration

COCOMO provides a simple schedule estimation capability. This estimation is based on the following equation

$$TDEV = [C \times PM^{D+0.2\times(E-B)}] \times \frac{SCED\%}{100} \approx 10.5 \text{ Months}$$

where C = 3.67, D = 0.28, B = 0.91, PM and E are the values calculated in the previous paragraph

This estimation is anyway too pessimistic and provides an upper bound of the time needed, it uses the pessimistic code size and it does not take into consideration that some part of the code needed can be found in already available libraries and some part of the development is carried out in parallel so our final estimation is 6 months.

3 Project schedule

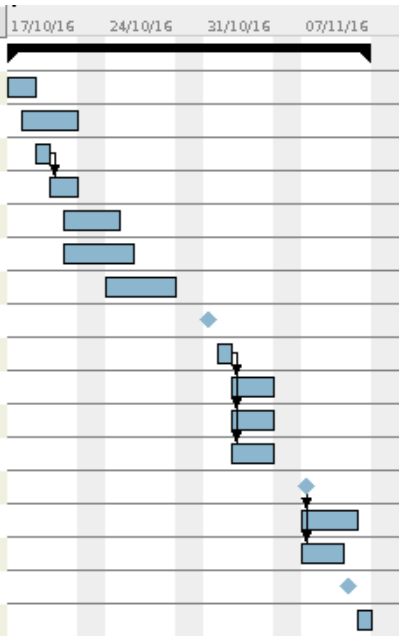
In this section we are going to provide a high-level schedule of the tasks involved in the projects.
Even if we did not go through implementation, testing and system deployment, we thought it was meaningful to include them as a part of the schedule in order to have a more complete overview of the overall plan devised for the project.

In order to maximize readability, we will present four separate schedules, one per each development phase. They are to be thought one after the other, in the order we present them. However, there are dates on top of each schedule which define the right period in which each activity is planned.

At the end of each development phase, a corresponding document is produced.

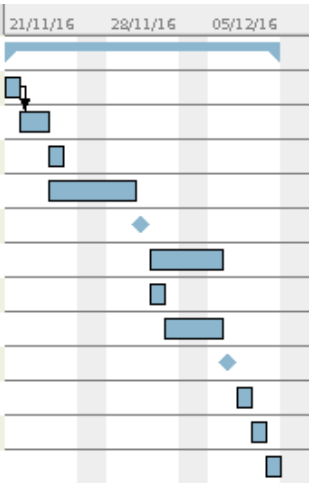
3.1 Requirements analysis and specification

Nome	Data d'inizio	Data di fine
Requirements Analysis and Specification	17/10/16	11/11/16
Talk to customers	17/10/16	18/10/16
Domain definition	18/10/16	21/10/16
Stakeholder identification	19/10/16	19/10/16
Scenario collection	20/10/16	21/10/16
Use case definition	21/10/16	24/10/16
Goal identification	21/10/16	25/10/16
Requirements identification	24/10/16	28/10/16
Internal checkup meeting	31/10/16	31/10/16
Customer review	01/11/16	01/11/16
Domain definition refinement	02/11/16	04/11/16
Goal refinement	02/11/16	04/11/16
Requirements refinement	02/11/16	04/11/16
Internal checkup meeting	07/11/16	07/11/16
Model verification	07/11/16	10/11/16
UI mockups	07/11/16	09/11/16
Quality assurance review	10/11/16	10/11/16
Final refinements	11/11/16	11/11/16

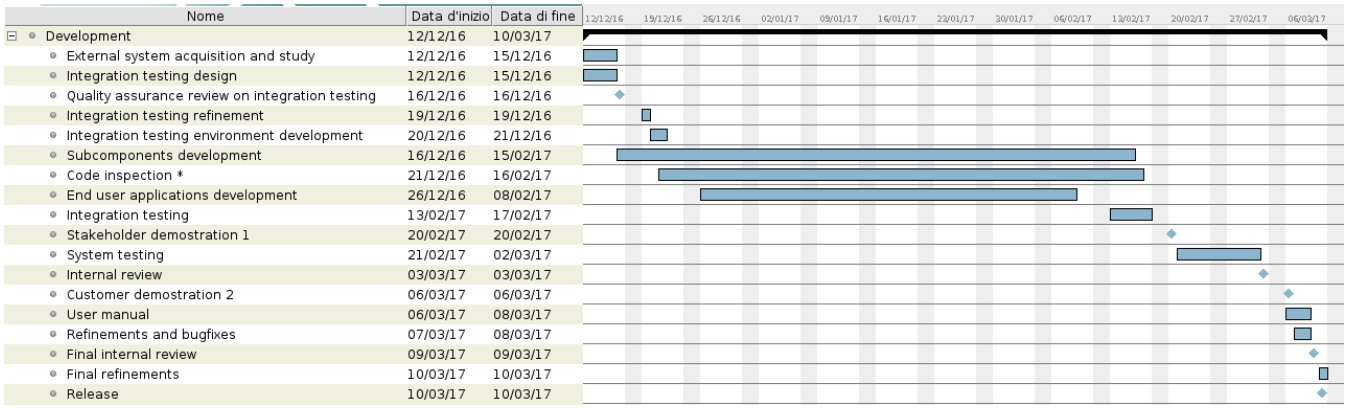


3.2 Design

Nome	Data d'inizio	Data di fine
Design	21/11/16	09/12/16
High level architecture draft	21/11/16	21/11/16
Main components identification	22/11/16	23/11/16
Data schema design	24/11/16	24/11/16
Dynamic system modeling	24/11/16	29/11/16
Internal review	30/11/16	30/11/16
Algorithm design	01/12/16	05/12/16
Deployment design	01/12/16	01/12/16
User interface design	02/12/16	05/12/16
Internal review	06/12/16	06/12/16
Meeting with stakeholders	07/12/16	07/12/16
Quality assurance review	08/12/16	08/12/16
Final refinements	09/12/16	09/12/16



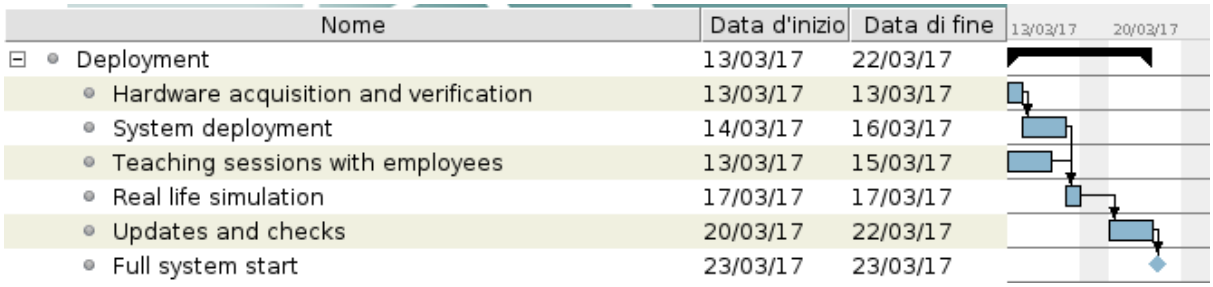
3.3 Implementation



Nome	Data d'inizio	Data di fine
Development	12/12/16	10/03/17
External system acquisition and study	12/12/16	15/12/16
Integration testing design	12/12/16	15/12/16
Quality assurance review on integration testing	16/12/16	16/12/16
Integration testing refinement	19/12/16	19/12/16
Integration testing environment development	20/12/16	21/12/16
Subcomponents development	16/12/16	15/02/17
Code inspection *	21/12/16	16/02/17
End user applications development	26/12/16	08/02/17
Integration testing	13/02/17	17/02/17
Stakeholder demostration 1	20/02/17	20/02/17
System testing	21/02/17	02/03/17
Internal review	03/03/17	03/03/17
Customer demostration 2	06/03/17	06/03/17
User manual	06/03/17	08/03/17
Refinements and bugfixes	07/03/17	08/03/17
Final internal review	09/03/17	09/03/17
Final refinements	10/03/17	10/03/17
Release	10/03/17	10/03/17

Important note: Even if code ispection is indicated in the activities with its own performance period, it has not to be intended as a continuous activity. In the diagram, we have indicated the interval over time in which code inspections sessions will be held. Code inspection schedule is to be defined later in the development workflow, according to the availability of the external company performing the inspections.

3.4 Deployment



Nome	Data d'inizio	Data di fine
Deployment	13/03/17	22/03/17
Hardware acquisition and verification	13/03/17	13/03/17
System deployment	14/03/17	16/03/17
Teaching sessions with employees	13/03/17	15/03/17
Real life simulation	17/03/17	17/03/17
Updates and checks	20/03/17	22/03/17
Full system start	23/03/17	23/03/17

4 Resource allocation

In order to better highlight parallel work on same tasks or on different tasks, resource allocation is presented in a tabular format, with each activity represented by a rectangle and identified by a label. Moreover, in order to increase readability, we have split the allocation table in four tables, one for each development phase. The numbering of the weeks reflects that of the Gantt, with week1 starting at Oct. 17th.

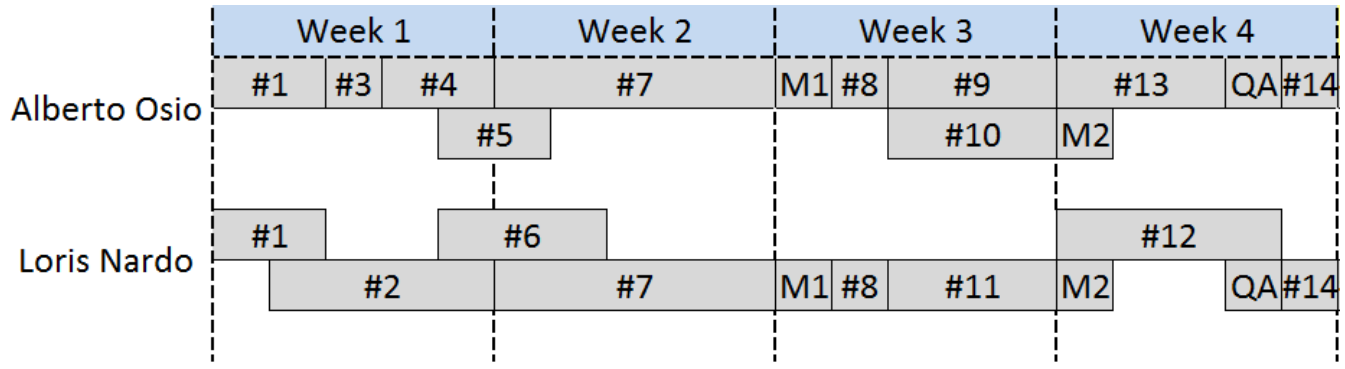
Not all the tasks described in the Gantt diagram are reflected in the resource allocation map. Here is a list of the exceptions:

- Code inspection
  As described in the previous chapter, code inspection is not really a task we have to carry out, as there will be held several code inspections sessions during the development period. So far, we have not included those session in the chart because we considered it as a part of the development tasks.
- System testing
  System testing is not included in the chart because it is carried out by an external company, and thus it not requires one of ou developers to be allocated to it.

For each development phase, we provide a map between activities and their identifier and the resource allocation table

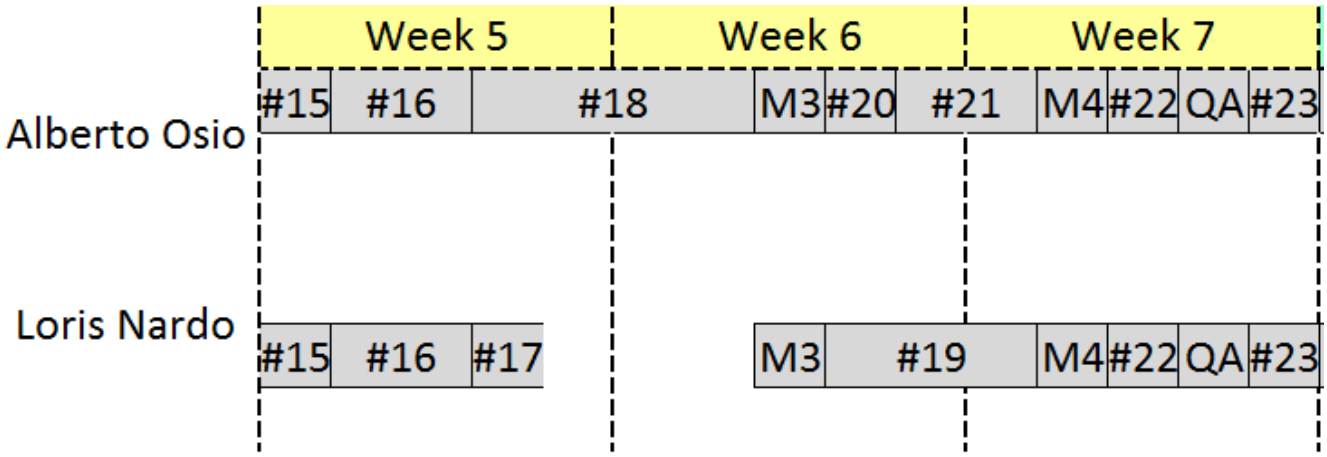
4.1 Requirements analysis and specification

Identifier	Activity
#1	Talk to customers
#2	Domain definition
#3	Stakeholder identification
#4	Scenario collection
#5	Use case definition
#6	Goal identification
#7	Requirements identification
M1	Internal checkup meeting (31/10/2016)
#8	Customer review
#9	Domain definition refinement
#10	Goal refinement
#11	Requirements refinement
M2	Internal checkup meeting (07/11/2016)
#12	Model verification
#13	UI mockups
QA	Quality Assurance Review (10/11/2016)
#14	Final refinements



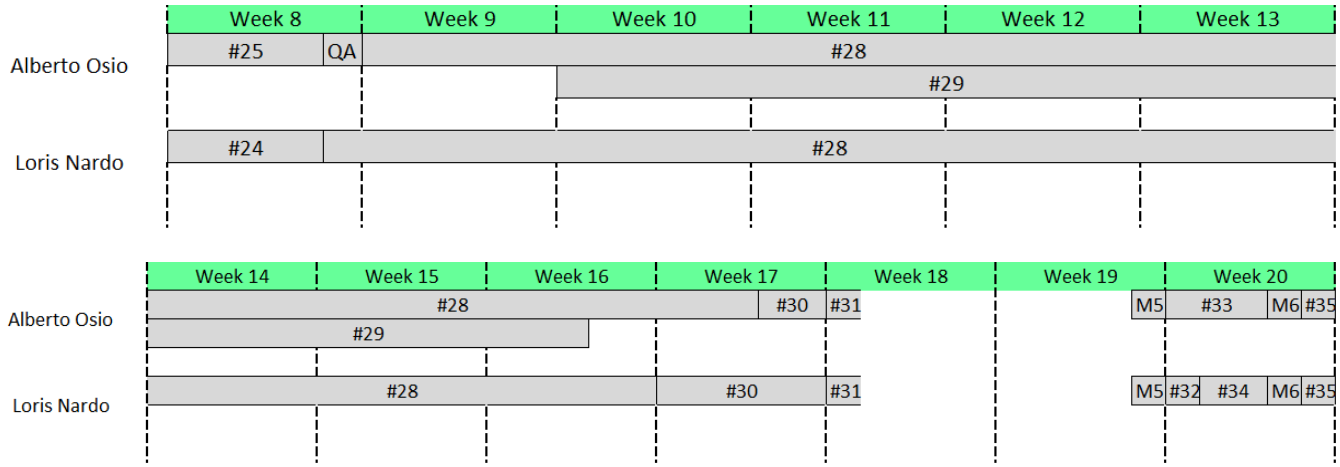
4.2 Design

Identifier	Activity
#15	High level architecture draft
#16	Main components identification
#17	Data schema design
#18	Dynamic system modeling
M3	Internal review (30/11/2016)
#19	Algorithm design
#20	Deployment design
#21	User interface design
M4	Internal review (06/12/2016)
#22	Meeting with stakeholders (07/12/2016)
QA	Quality Assurance Review (08/12/2016)
#23	Final refinements



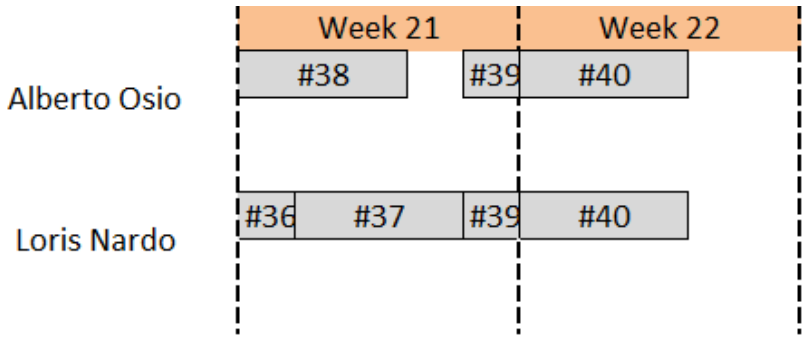
4.3 Development

Identifier	Activity
#24	External system acquisition and study
#25	Integration testing design
QA	Quality assurance review (16/12/2016)
#26	Integration testing refinement
#27	Integration testing environment development
#28	Subcomponents development
#29	End user applications development
#30	Integration testing
#31	Stakeholder demostration 1
M5	Internal review (20/01/2017)
#32	Customer demostration 2
#33	User manual
#34	Refinements and bugfixes
M6	Final internal review
#35	Final refinements



4.4 Deployment

Identifier	Activity
#36	Hardware acquisition and verification
#37	System deployment
#38	Teaching sessions with employees
#39	Real life simulation
#40	Updates and checks



5 Risk management

Risks must be taken into consideration in a project planning.
Even if it is not possible to foresee any possible risk, a good risk strategy for the most probable risks must be taken into consideration. Three main risk categories are described in the following part of this section of the document.

- **Project risks**: these risks involve the project plan itself, leading to raise in cost or to deadline missing.
- **Technical risks**: these risks involve the implementation of the project and they affect the overall quality of the software we have to develop.
- **Business risks**: these risks involve the company developing the software itself.

5.1 Project risks

Risk	Level	Description
Too optimistic schedules	Critical	The project may require additional time. If this is the case, we can release an incomplete but working version (e.g. without the ability for employee to define the geographical areas, or with a simplier user interface). However, in order to mitigate this risk, some slack time is inserted in the schedule. This is considered critical as our schedule has not mush slack
Too pessimistic schedules	Less critical	The project may require much less time. If this is the case, we can propose our client some additional services for free (e.g. extended support). For this reason, we do not consider this as a critical risk.
Requirements change	Moderately critical	This risk can be mitigated by designing the system (from requirements analysis up to actual implementation) in a conscious way, so that it is easy to extend or change it. When implementation is in progress, basic principles of OOP (coding to an interface, polymorphism, ...) must be carefully and consistently applied in the whole project and verified with code inspections on the most critical pars. This risk is considered moderately critical because, even if we apply the best design techniques, we cannot predict all changes that may come up
Team collaboration issues	Less critical	Misunderstanding or things left as intended may lead to an incorrect repartition of tasks. This can be mitigated with meetings and a complete and precise task assignment. We consider this less critical because we inserted in the schedule frequent meetings
Turnover in development team	Moderately critical	Since the IT job market is quite flexible, this is always risk to be considered, that can be mitigated by splitting responisibilities among people of the company. We evaluate this risk factor as moderately critical, because given our staff it is extremely unlikely that key members quit the company during the project development, but we cannot avoid it. If happens, we plan to pay extra hours to other members in order to reschedule tasks assigned to those people
Illness of key members	Moderately critical	The classification of this factor is really similar to that of the previous one
Gold plating	Less critical	A software product is never perfect, but this does not mean it cannot be released. Successive releases may refine some aspects. This risk is avoided if deadlines for task assignment are defined and enforced, and for this reasons we classify it as less critical
Misunderstanding in requirements	Moderately critical	This is always a very serious problem, but can be mitigated by deeply involving stakeholders in requirements analysis, asking for frequent feedbacks and organizing frequent meetings. We consider this a moderately critical risk because we put in the schedule frequent meetings

5.2 Technical risks

Risk	Level	Description
Lack of experience in programming with Java EE	Moderately critical	This must be taken into consideration in the early stages of planning (e.g. adding a timespan devoted to learning of this technology or expanding the time related to the actual coding of the software project).
Server downtime	Critical	Downtimes are often indicators that the software is not scalable or that the hardware running the server components is not enough powerful. To prevent this, one must design the software with scalability in mind from the early stages of the DD, and choose the appropriate hardware. This is considered critical because can only be detected during system testing, which is performed at the very end of the development of the product.
Security flaws	Critical	The application may be susceptible to security issues and to data leaks if not well designed. All the raccomandation from OWASP must be followed, thus the user input must be correctly handled; tests must be done to ensure that most common flaws are handled (e.g. XSS, CSRF, SQL Injection, …). This is considered critical because it is not easy to discover even with a careful system testing.
Data loss	Moderately ritical	Data loss can occure because of hardware fault or software errors. This problem can be mitigated enforcing periodic backups. Our company performs daily backups archivied both locally and on a cloud service, so it is extremely unikely that this happens
Unstructured code	Moderately critical	With the growth of the project and the approaching deadline the code may become badly structured, with repetitions. This problem can be mitigated performing periodical code inspections. We have allocated a quite long time period to perfomr code inspections, so we can consider this moderately critical
Component integration failure	Less critical	After having implemented some components, a test may fail and this can require to rewrite large portions of some component. This risk is mitigated by specifying the contracts and interfaces of each component and by starting integration tests earlier so that there is still time to fix problems. As we start integration testing as early as possible, this is considered less critical

5.3 Business risks

Risk	Level	Description
Competitors	Critical	Other companies might develop and release a similar software product before us. A possible solution for this problem is to continuosly enhance the software product, to highlight peculiarities, or to provide better non functional requirements or designing a most attractive user interface. We consider this a critical risk because we cannot anticipate it in any way.
Bankruptcy	Less critical	While the income for this project is known, its cost is not, a good feasibility study helps to avoid this situation. We consider this less critical because we have carried out a very accurate feasibility study.
PowerEnJoy may violete some future laws	Less critical	Local and State regulators can change some rules and this can lead to unpredictable impact. This risk cannot be avoided, but the problematic portion of the product can be disabled temporarily and the team must work to adapt to the new regulations as soon as possible. This is a future risk, and so it is unlikely that it causes schedule shifts

6 Appendix

6.1 Effort spent

- Nardo Loris: 11 hours of work
- Osio Alberto: 13 hours of work