Project Plan

Document Version 1.2

Matteo Frosi (mat. 875393)
Luca Costa (mat. 808109)

Contents

1 Introduction

1.1 Purpose and scope

This document represents the Project Plan Document for Power Enjoy service. Planning is a crucial part of the project because a bad estimation can lead to a total failure. A Project plan gives information about the size of the project, in terms of line of code and cost/effort actually required on the base of different approaches (Function Points and COCOMO). Moreover it studies how to map the different tasks (different parts to be developed) on the actual person working at the project. In the end a risk analysis is done to face before every possible problem that can be discovered during the development.

1.2 Definitions and abbreviations

- Function Points estimation: is an estimation of the size of a specific software in terms of line of codes.
- SLOC (Source Lines Of Code): it measures the lines of code of a project (empty and commented lines excluded).
- FP (function point): is a unit of measurement to express the amount of business functionality an information system provides to a user.
- Gannt diagram: a type of bar chart illustrating a project schedule.
- 4GL: is a computer programming language envisioned as a refinement of the style of languages classified as third-generation programming language (3GL). Languages claimed to be 4GL may include support for database management, report generation, mathematical optimization, GUI development, or web development.
- Gearing factor: it is the average number of new plus modified (Effective) Source Lines of Code per function point in the completed project. Gearing factors are calculated by dividing the effective code count for a completed project by the final function point count.
- Internal logic file (ILF): homogeneous set of data used and managed by the application.
- External interface file (EIF): homogeneous set of data used by the application but generated and maintained by other applications.
- External input (EI): elementary operation to elaborate data coming from the external environment.
- External output (EO): elementary operation that generates data for the external environment.
- External inquiry (EQ): elementary operation that involves input and output, without significant elaboration of data from logic files.

2 Estimated measurements

Various measurement must be considered to provide a clear and detailed view of the project. Among them in this section we are going to consider size, effort, cost and human (and not) resources.

2.1 Function points

Widely accepted as an industry standard for functional sizing, FPs measure systems from a functional perspective they are independent from technology. Regardless of language, development method, or hardware platform used, the number of function points for a system will remain constant. The only variable is the amount of effort needed to deliver a given set of function points.
The function point counts at the end of requirements and/or designs can be compared to function points actually delivered. If the project has grown, there has been scope creep. The amount of growth is an indication of how well requirements were gathered by and/or communicated to the project team. If the amount of growth of projects declines over time it is a natural assumption that communication with the user has improved.
FPs can be used to estimate LOC depending on the average number of LOC per FP for a given language, considering the following formula:

$$LOC = FP * AVC$$

Where LOC indicates the lines of code, FP the number of function points and AVC a language-dependent factor varying from 200-300 for assembly language to 2-40 for a 4GL.
Function points are classified under different types. For each type, a different weight is also assigned for each function point according if the concerned function is estimated as being simple, complex, or average. For the estimation of the size of PowerEnJoy the standard weights are indicated in the following table:

Function types	**Weight**		
	Simple	Medium	Complex
N. Inputs	3	4	6
N. Outputs	4	5	7
N. Inquiry	3	4	6
N. ILF	7	10	15
N. EIF	5	7	10

Function point analysis is based on 5 user identifiable logical ”functions” which are divided into 2 data function types and 3 transactional function types (table above). For a given software application, each of these elements is quantified and weighted, counting its characteristic elements, like file references or logical fields.

2.1.1 External Inputs (EIs)

The application interacts both with users and cars, that are the main actors of the system. We may also consider as a secondary actor the operators called when an accident happens, even though it is not direct part of the system, with the exception of an interface. Each of them has its own set of possible interactions. Considering the user, the system has to grant him:

- Login / Logout: operation that involves the LoginController component of the system.
- Research of nearby cars: this operation takes place when the user is going to reserve or monitor a car. Before it, the system checks his position, or the suggested one by the user, to show him the cars available within. This operation has medium complexity.
- Reservation of a car: after having researched a car and selected it, the user is invited to make a reservation, if it is available, or a monitoring. In the first case, the request is sent and the system has to check for its correctness. After that, the user has to be notified that the car that has been reserved, or that the request has been rejected. This operation , due to use of external elements, as the ones to send or receive messages, is complex.
- Monitoring of a car: after having researched a car and selected it, the user is invited to make a reservation, if it is available, or a monitoring. In the latter case, the request is sent and the system has check for its correctness. After that, the user has to be notified that the request on the car has been successful, or has been rejected. With respect to the reservation process, it's a slightly more complex procedure, because it also involves the continuous check of the car status.
- Deletion of a reservation: deleting has an average complexity because all the monitoring users on the interested car have to be notified.
- Deletion of a monitoring: deleting a monitoring has a low complexity, because it only involves the deletion of the user in the list of monitoring user for a specific car.
- Request of the access code: when the user has made a reservation and it's in time to pick up the car, he/she has to inform the system that he/she needs the code to access the car and benefit from the service. The complexity is low because the usual checks are made and an unique code is generated and stored for that user, until expiration.
- Registration of a new account: registering implies an average complexity because it has to write on the main database new information.
- Emergency button press: the system has to receive the signal and simply contact an external operator, redirecting him/her to the user. The complexity is very low.

Considering the car, the system receive the data of the ride and compute them, sending back as result the cost of the ride. This operation is complex. Moreover, there is the retrieval of the car data using the VehicleInterface component. It is a complex operation because it involves the conversion of data format and small elaboration.
Lastly the system includes the action to add, delete or update a safe zone, made by the system itself or an operator with particular privileges. Each of this action has an high complexity, because it involves the use of the database and the update of all the car status (just think about the deletion of a safe area in which some cars are parked...).

EI	Complexity	Function Points
Login	Low	3
Logout	Low	3
Reservation of a car	High	6
Reservation of a car	High	6
Monitoring of a car	High	6
Deletion of a reservation	Medium	4
Deletion of a monitoring	Low	3
Request of the access code	Medium	4
Registration of a new account	Medium	4
Emergency button press	Low	3
Data elaboration from the cars	High	6
Addition, deletion and update of safe zones	Medium	4x3
Car data retrieval	High	6
TOTAL		66

2.1.2 External Outputs (EOs)

Our application must send information to the mobile phone of the user, to the car and to the external operator, outside the inquiries world.

- Map construction: it refers to the information that the system sends to the user mobile phone through the connection when the user searches for the available cars within a certain range (including position, real availability, battery, and so on). It is a simple transmission of data, hence the complexity is low.
- Notification of successful/unsuccessful reservation.
- Notification of successful/unsuccessful monitoring.
- Notification of ended monitoring (car taken)/available monitored car (car reservation taken back or expired): it refers to the notification sent when the car monitored becomes available again, using the push gateway, or when the car has been picked up, and all the users monitoring it are deleted from the list and informed. It has medium complexity, due to the logic involved in the elimination and coherence with the database.
- Notification of the cost of the ride with the applied bonuses/overcharges.
- Car updated when certain event occurs, such as end of the ride, code reset, automatic lock (imposed by the server).
- Notification of the operator: it refers to the connection made possible by the system between the user and the operator, with a low complexity.

EO	Complexity	Function Points
Map construction	Low	4
Successful/unsuccessful reservation notification	Low	4
Successful/unsuccessful monitoring notification	Low	4
Ended monitoring (car taken)/available monitored car notification	Medium	5
Total cost of the ride	Low	4
Car update	Medium	5
TOTAL		26

2.1.3 External Inquiries (EQs)

An inquiry is essentially a data retrieval request performed by an actor. The external inquiries includes:

- Request from the user of the positions of the nearby cars, from his/her location or from the specified point. This is done checking every car positions in the user zone and the user positions and considering only the cars included in a certain radius. This procedure is very complex, because the distance cannot easily calculated with an Euclidean method, too simple, but considering the structure of the roads.

- Request from the operator of the ride data. Even though it is not included in the software, the system shall provide, along with the communication interface with the operator, a module that allows the retrieval of specific data, useful to understand better the situation of an accident. It is quite complex because it should provide only little access, for the operator, to the content of the server knowledge.

EI	Complexity	Function Points
Position of the cars	High	6
Accident data (from operator)	High	6
TOTAL		12

2.1.4 Internal Logic Files (ILFs)

The class diagram belonging to the RASD document can be used to display the different (but not all) the ILFs that we are going to consider. After the diagram, there is a table in which every component is associated with it's complexity and function points score.

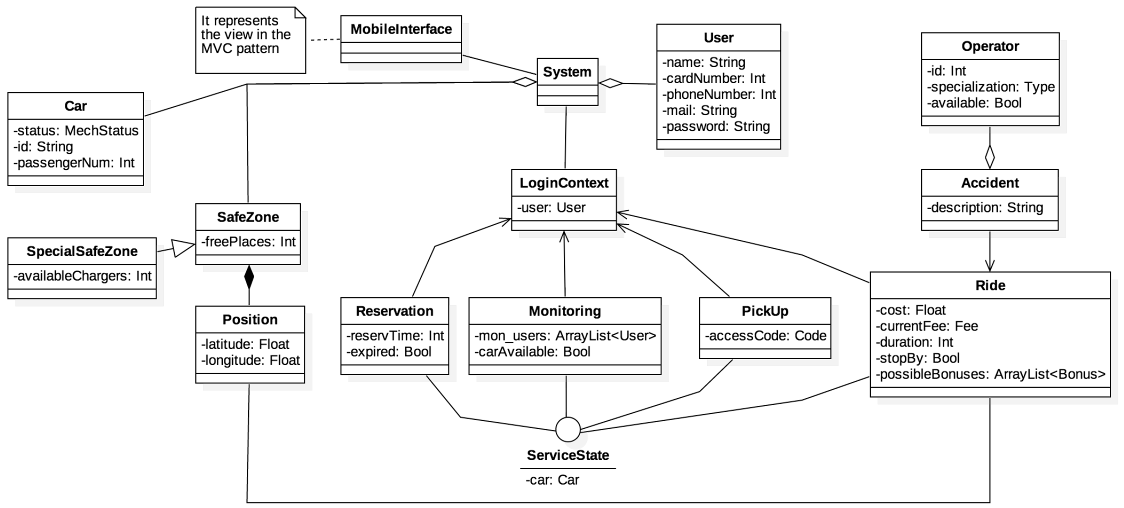


Figure 1: class diagram used in the RASD

ILF	Complexity	Function Points
System	High	15
Reservation	Medium	10
Monitoring	Medium	10
PickUp	Low	7
Ride	High	15
Position	Low	7
Accident	Low	7
SafeZone	Low	7
Car	Medium	10
User	Medium	10
TOTAL		98

The system class was an extremely simplified representation of all the other components that interacts with the model (that is what is represented in the diagram of figure 1. We have also to include every components and its modules. This, however, is a difficult task, because we don't exactly know the number of modules of each component. For such reason we set the complexity of the system class as maximum, so 15.

2.1.5 External Interface Files (EIFs)

The system doesn't rely on any external service. In fact, as far as the car nearby check, it is used a specific algorithm created by us, already comprehended in the software.

2.1.6 Accumulated Function Points and code size estimation

We can now collect all the data in a single table:

Function Type	**Function Points**
External Input	66
External output	26
External Inquiry	12
Internal Logic File	98
External Interface	0
TOTAL	202

From the table we can see a huge gap between internal logic and all the other parts. An explanation could be the fact that the client app and the cars mainly give data to the system, which is the one that computes them and gives back new information about ride. The other important part is the external input, because on that relies all the computation of the system. However it seems overestimated the difference between them. To conclude, the aim of the project is concentrated on the client and the reliability of the system, so it is coherent considering more importantly these two parts in the function points calculation.

2.1.7 Lines of Code

On the base of the Function Points result, we can now estimate the number of lines of code.
In a perfectly symmetrical distribution of gearing factors, the average and the median will be identical or very close. The average is obtained by summing the gearing factors and then dividing by the number of gearing factors included in that sum. Although its purpose is to measure ”central tendency”, the average can be pulled up or down by extreme data values (or outliers). The median, on the other hand, is simply the data point that lies in the center of an ordered list of gearing factors. One half of the data points will lie above (and one half below) the median.
To conclude, the median may be a more accurate indicator of the central tendency. Using J2EE to develop our software, we refer to particular values of gearing factors.

Average bound:	*SLOC = 202 * 46 = 9292*
Medium bound:	*SLOC = 202 * 49 = 9898*
Lower bound:	*SLOC = 202 * 15 = 3030*
Upper bound:	*SLOC = 202 * 67 = 13534*

2.2 COCOMO

2.2.1 COCOMO II

In COCOMO II effort is expressed as Person-Months (PM). A person month is the amount of time one person spends working on the software development project for one month. This number excludes time typically devoted to holidays, vacations, and weekend time off.

$$EFFORT = 2.94 * EAF * (KSLOC)^{E}$$

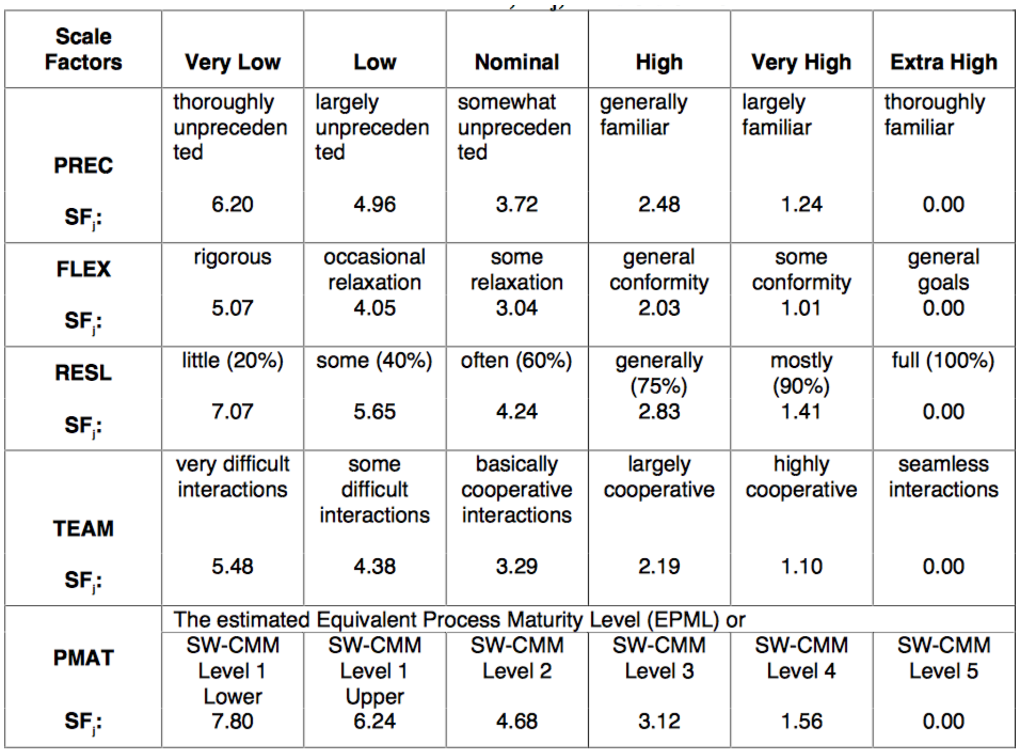
Where :

- EAF = product of the cost drivers
- KSLOC = number of thousands estimated by Function Points
- E = exponent derived from scale drivers.

Follows, in detail, the explanation of every component of the expression.

2.2.2 Scale Factors

To estimate correctly values, we refer to the COCOMO official tables :



Scale Factors	Very Low	Low	Nominal	High	Very High	Extra High
PREC	thoroughly unprecedented	largely unprecedented	somewhat unprecedented	generally familiar	largely familiar	thoroughly familiar
SF_j:	6.20	4.96	3.72	2.48	1.24	0.00
FLEX	rigorous	occasional relaxation	some relaxation	general conformity	some conformity	general goals
SF_j:	5.07	4.05	3.04	2.03	1.01	0.00
RESL	little (20%)	some (40%)	often (60%)	generally (75%)	mostly (90%)	full (100%)
SF_j:	7.07	5.65	4.24	2.83	1.41	0.00
TEAM	very difficult interactions	some difficult interactions	basically cooperative interactions	largely cooperative	highly cooperative	seamless interactions
SF_j:	5.48	4.38	3.29	2.19	1.10	0.00
PMAT	The estimated Equivalent Process Maturity Level (EPML) or					
	SW-CMM Level 1 Lower	SW-CMM Level 1 Upper	SW-CMM Level 2	SW-CMM Level 3	SW-CMM Level 4	SW-CMM Level 5
SF_j:	7.80	6.24	4.68	3.12	1.56	0.00

Figure 2: COCOMO scale factors table

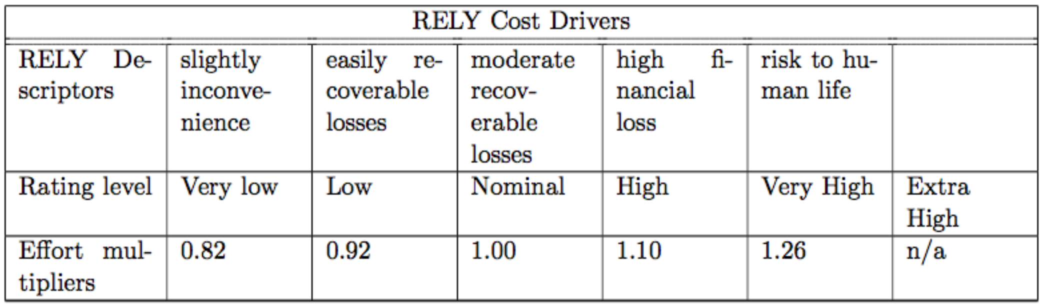
- Precedentedness: (PREC) : it refers to the experience of the team and on how much this project is similar the others did before. Since this is our second project, this parameter is Low.
- Development Flexibility (FLEX) : it refers to the coherence with the feature required by assignment. Since there are very strict requirements on the functionality but nothing specific is stated as for the technology to be used, the flexibility is Low too.
- Risk Resolution (RESL) : it refers to how much the system can be affected by risks (es. wrong client input and so on). We have considered in the development a lot of risks. This value is High.
- Team Cohesion (TEAM) : it refers to how well the members of the project work together. In our case the value is High.
- Process Maturity (PMAT) : all the goals have been successfully achieved on time, so this value is Very High.

Scale Driver	**Factor**	**Value**
PREC	Low	4,96
FLEX	Low	4,05
RESL	High	2,83
TEAM	High	2,19
PMAT	Very High	1,56
TOTAL		15,59

2.2.3 Cost Drivers

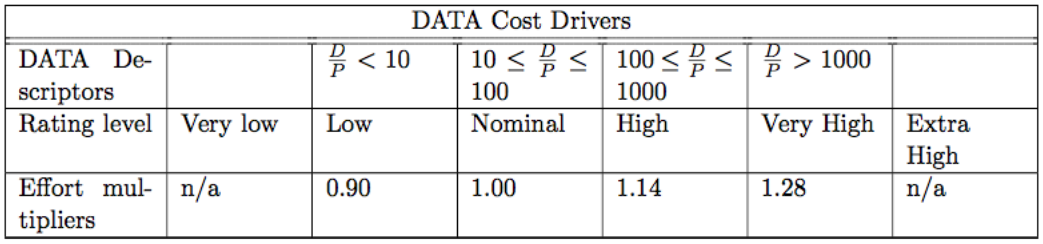
There are several cost drivers that characterize the variation of the effort. Now we are going to consider them one by one.

- Required Software Reliability : it refers to the grade of reliability of the System. For example, if a bug drives to a human loss, the RELY is very High. For our project, this value is set as Nominal, because in the worst case, the service becomes unavailable for some days, after that it should be recovered and the money loss regained.



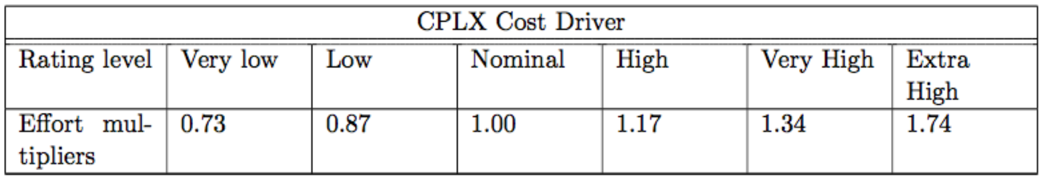
RELY Cost Drivers						
RELY Descriptors	slightly inconvenience	easily recoverable losses	moderate recoverable losses	high financial loss	risk to human life	
Rating level	Very low	Low	Nominal	High	Very High	Extra High
Effort multipliers	0.82	0.92	1.00	1.10	1.26	n/a

- Database Size : DATA is capturing the effort needed to assemble and maintain the data required to complete test of the program. In this case we obtained a value more than 1000 that is considered as very High because we have a 32 Gb Database.



DATA Cost Drivers						
DATA Descriptors		$\frac{D}{P} < 10$	$10 \le \frac{D}{P} \le 100$	$100 \le \frac{D}{P} \le 1000$	$\frac{D}{P} > 1000$	
Rating level	Very low	Low	Nominal	High	Very High	Extra High
Effort multipliers	n/a	0.90	1.00	1.14	1.28	n/a

- Product Complexity : it refers to the complexity of the whole project. We set this value as High.



CPLX Cost Driver						
Rating level	Very low	Low	Nominal	High	Very High	Extra High
Effort multipliers	0.73	0.87	1.00	1.17	1.34	1.74

- Required Reusability : this cost driver accounts for the additional effort needed to construct components intended for reuse on current or future projects. We developed our system to be very reusable.

So the Value is set as Very High.

RUSE Cost Driver						
RUSE Descriptors		None	Across project	Across program	Across product line	Across multiple product lines
Rating level	Very low	Low	Nominal	High	Very High	Extra High
Effort multipliers	n/a	0.95	1.00	1.07	1.15	1.24

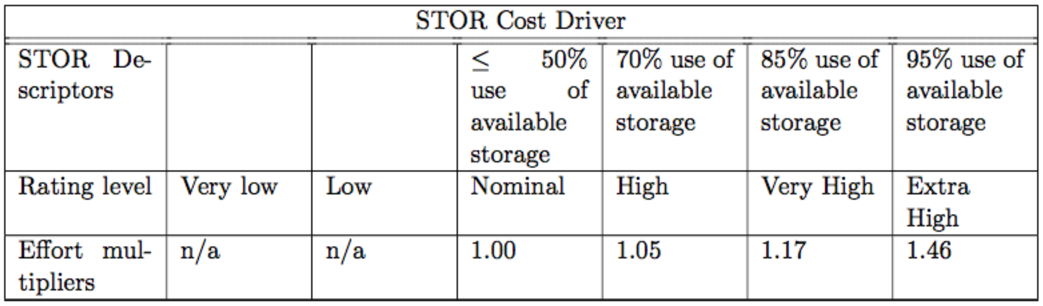
- Documentation match to life-cycle needs : it refers to how much the documentation respects the requirements. We set this value as Nominal because the documentation respects every specification.

DOCU Cost Driver						
DOCU Descriptors	Many life-cycle needs uncovered	Some life-cycle needs uncovered	Right-sized to life-cycle needs	Excessive for life-cycle needs	Very excessive for life-cycle needs	
Rating level	Very low	Low	Nominal	High	Very High	Extra High
Effort multipliers	0.81	0.91	1.00	1.11	1.23	n/a

- Execution Time Constraint : it refers to the amount of CPU usage with respect to the computational capabilities of the hardware. We set this parameter as High, because the logic part of the software requires a decent amount of computational resources (like the algorithms).

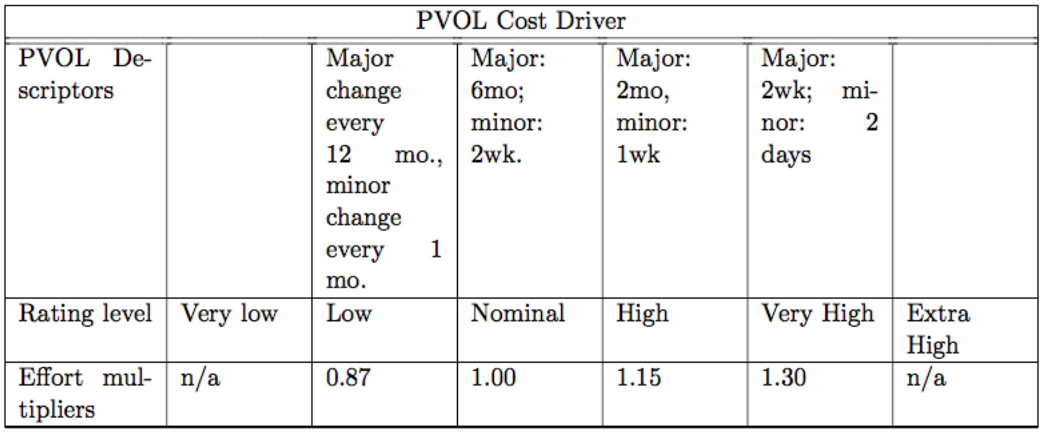
TIME Cost Driver						
TIME Descriptors			$\leq$ 50% use of available execution time	70% use of available execution time	85% use of available execution time	95% use of available execution time
Rating level	Very low	Low	Nominal	High	Very High	Extra High
Effort multipliers	n/a	n/a	1.00	1.11	1.29	1.63

- Storage Constraint : it refers to the expected amount of storage usage with respect to the availability of the hardware. Due to today level of technology, we set this parameter at nominal. There is no storage problem.



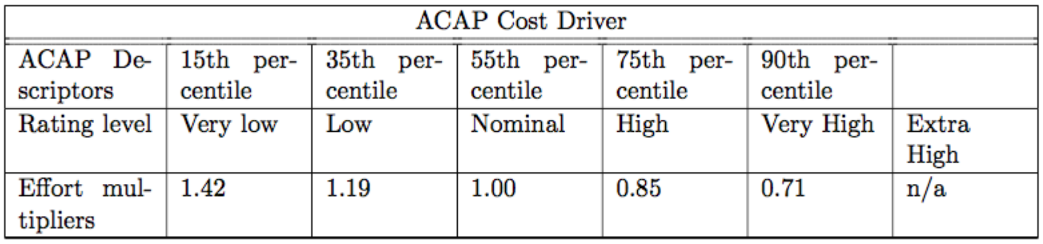
STOR Cost Driver						
STOR Descriptors			≤ 50% use of available storage	70% use of available storage	85% use of available storage	95% use of available storage
Rating level	Very low	Low	Nominal	High	Very High	Extra High
Effort multipliers	n/a	n/a	1.00	1.05	1.17	1.46

- Platform Volability : for what concerns the core system, we don't expect our fundamental platforms to change very often (except for sudden bugs). Obviously it will be periodically updated to stay on line with the newest O.S., after a fixed amount of time. We set this value as Low.



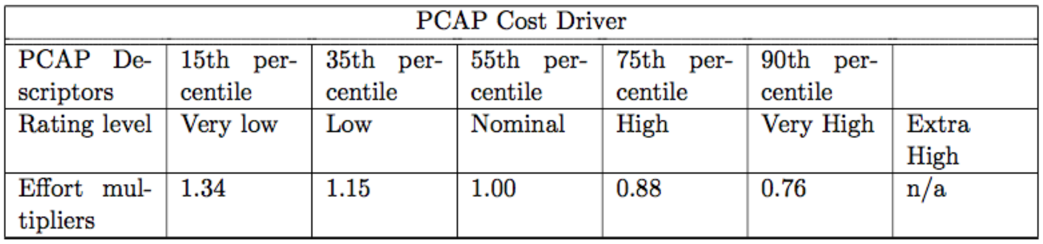
PVOL Cost Driver						
PVOL Descriptors		Major change every 12 mo., minor change every 1 mo.	Major: 6mo; minor: 2wk.	Major: 2mo, minor: 1wk	Major: 2wk; minor: 2 days	
Rating level	Very low	Low	Nominal	High	Very High	Extra High
Effort multipliers	n/a	0.87	1.00	1.15	1.30	n/a

- Analyst Capability : the analysis has been conducted with a particular attention to respect a possible real implementation. We set this value as High.



ACAP Cost Driver						
ACAP Descriptors	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile	
Rating level	Very low	Low	Nominal	High	Very High	Extra High
Effort multipliers	1.42	1.19	1.00	0.85	0.71	n/a

- Programmer Capability : since we have already done a Software Engineering project and a Software Project, we set this value as High, even if we didn't implemented this project for real.



PCAP Cost Driver						
PCAP Descriptors	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile	
Rating level	Very low	Low	Nominal	High	Very High	Extra High
Effort multipliers	1.34	1.15	1.00	0.88	0.76	n/a

- Application Experience : we have no experience with Java EE, even though we gained some experience in Java applications during the previous projects, so we set this Value as Low.

APEX Cost Driver						
APEX Descriptors	≤ 2 months	6 months	1 year	3 years	6 years	
Rating level	Very low	Low	Nominal	High	Very High	Extra High
Effort multipliers	1.22	1.10	1.00	0.88	0.81	n/a

- Platform Experience : since we have some experience in programming, interacting with databases, user interfaces and Java features in general, we set this parameter as Nominal.

PLEX Cost Driver						
PLEX Descriptors	≤ 2 months	6 months	1 year	3 years	6 years	
Rating level	Very low	Low	Nominal	High	Very High	Extra High
Effort multipliers	1.19	1.09	1.00	0.91	0.85	n/a

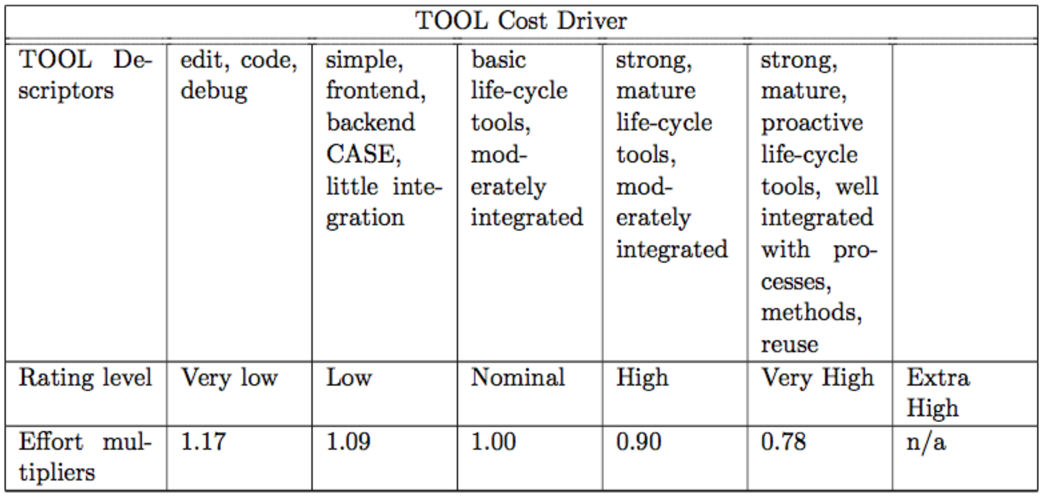
- Language and tool Experience : we have no experience with Java EE, but we have some experience with databases and interfaces, so we set this value as Nominal.

LTEX Cost Driver						
LTEX Descriptors	≤ 2 months	6 months	1 year	3 years	6 years	
Rating level	Very low	Low	Nominal	High	Very High	Extra High
Effort multipliers	1.20	1.09	1.00	0.91	0.84	n/a

- Personnel continuity : the team is composed of only two people and it is not going to be enlarged. So we set this value as very High.

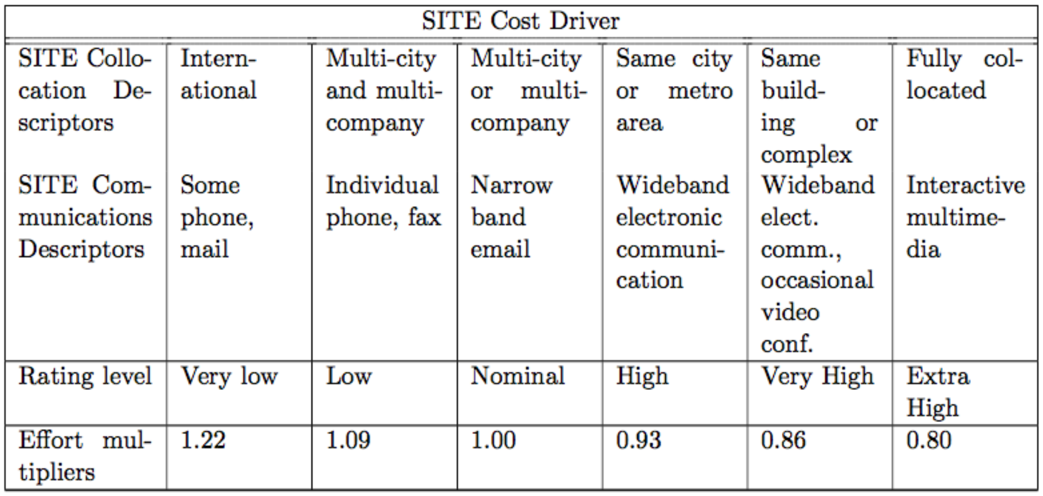
PCON Cost Driver						
PCON Descriptors	48% / year	24% / year	12% / year	6% / year	3% / year	
Rating level	Very low	Low	Nominal	High	Very High	Extra High
Effort multipliers	1.29	1.12	1.00	0.90	0.81	n/a

- Usage of software tool : it refers to the completeness of the tools we are using. Since our application environment (Java EE) is well integrated with all other environments, we set this value as High.



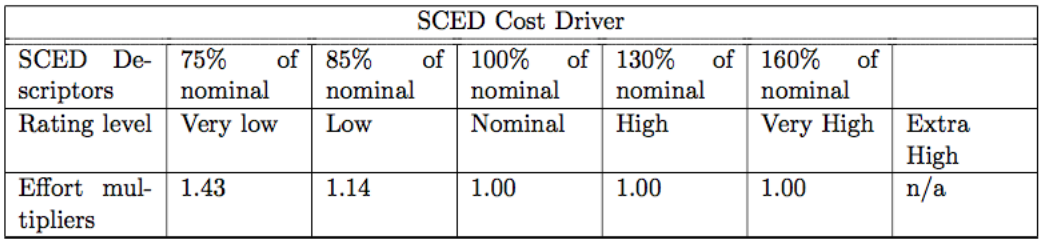
TOOL Cost Driver						
TOOL Descriptors	edit, code, debug	simple, frontend, backend CASE, little integration	basic life-cycle tools, moderately integrated	strong, mature life-cycle tools, moderately integrated	strong, mature, proactive life-cycle tools, well integrated with processes, methods, reuse	
Rating level	Very low	Low	Nominal	High	Very High	Extra High
Effort multipliers	1.17	1.09	1.00	0.90	0.78	n/a

- Multisite Development : it refers to the way of communication we used. We have used online sites to upload our project, social networks and conference calls. So we set this value as Very High.



SITE Cost Driver						
SITE Collocation Descriptors	International	Multi-city and multi-company	Multi-city or multi-company	Same city or metro area	Same building or complex	Fully collocated
SITE Communications Descriptors	Some phone, mail	Individual phone, fax	Narrow band email	Wideband electronic communication	Wideband elect. comm., occasional video conf.	Interactive multimedia
Rating level	Very low	Low	Nominal	High	Very High	Extra High
Effort multipliers	1.22	1.09	1.00	0.93	0.86	0.80

- Required development schedule : it refers to the schedule constraints imposed on the project team developing the software. We set this value as Nominal because we spent a lot of time thinking and writing the project, but we also finished them on time or before the end date.



SCED Cost Driver						
SCED Descriptors	75% of nominal	85% of nominal	100% of nominal	130% of nominal	160% of nominal	
Rating level	Very low	Low	Nominal	High	Very High	Extra High
Effort multipliers	1.43	1.14	1.00	1.00	1.00	n/a

Summary Table :

Cost Drivers	**Factor**	**Value**
Required Software Reliability (RELY)	Nominal	1.00
Database Size (DATA)	Very High	1.28
Product Complexity (CPLX)	High	1.17
Developed for Reusability (RUSE)	Very High	1.15
Documentation Match to Life-Cycle Needs (DOCU)	Nominal	1.00
Execution Time Constraint (TIME)	High	1.11
Main Storage Constraint (STOR)	Nominal	1.00
Platform Volatility (PVOL)	Low	0.87
Analyst Capability (ACAP)	High	0.85
Programmer Capability (PCAP)	High	0.88
Application Experience (APEX)	Low	1.10
Platform Experience (PLEX)	Nominal	1.00
Language and Tool Experience (LTEX)	Nominal	1.00
Personnel Continuity (PCON)	High	0.81
Use of Software Tools (TOOL)	High	0.90
Multisite Development (SITE)	Very High	0.86
Required Development Schedule (SCED)	Nominal	1.00
TOTAL (product of all the cost drivers)		0.858

2.2.4 Effort Equation

As previously stated, the effort related to the project is calculated with:

$$EFFORT = A * EAF * (KSLOC)^{E}$$

Where:

- A = 2.94 (for COCOMO II)
- EAF = 0.858 (product of cost drivers)
- KSLOC = kilo lines of code
- E = B + 0.01 $*\sum_i$SF[i] = 0.91 + 0.01 * 15.59 = 1.0659
  - B = 0.91 for COCOMO II
  - $\sum_i$SF[i] = sum of Scale Drivers

Having all the values, we can finally compute the effort of our project :

$$Average\ Effort = A * EAF_{average} * KSLOC^{E} = 2.94 * 0.858 * 9.292^{1.0659} = 27.15 = 28\ PM$$
$$Medium\ Effort = A * EAF_{medium} * KSLOC^{E} = 2.94 * 0.858 * 9.898^{1.0659} = 29.03 = 29\ PM$$
$$Lower\ Effort = A * EAF_{lower} * KSLOC^{E} = 2.94 * 0.858 * 3.030^{1.0659} = 8.22 = 9\ PM$$
$$Upper\ Effort = A * EAF_{upper} * KSLOC^{E} = 2.94 * 0.858 * 13.534^{1.0659} = 40.53 = 41\ PM$$

2.2.5 Schedule Estimation

As far as the duration of the project is concerned, to estimate it we use the following formula :

$$Duration = 3.67 * Effort^{F}$$

Where :

- F = 0.28 + 0.2 $*$ (E $-$ B) = 0.28 + 0.2 * (1.0659 - 0.91) = 0.31118

- E = 1.0659, the same coefficient calculated before
- B = 0.91
- Effort: is the same calculated in the previous sub subsection

The calculus of the duration is then:

$$\textit{Average Duration} = 3.67 * \textit{Average Effort}^{F} = 3.67 * 28^{0.31118} = 10.35 \textit{ months}$$
$$\textit{Medium Duration} = 3.67 * \textit{Medium Effort}^{F} = 3.67 * 29^{0.31118} = 10.46 \textit{ months}$$
$$\textit{Lower Duration} = 3.67 * \textit{Lower Effort}^{F} = 3.67 * 9^{0.31118} = 7.27 \textit{ months}$$
$$\textit{Upper Duration} = 3.67 * \textit{Upper Effort}^{F} = 3.67 * 41^{0.31118} = 11.66 \textit{ months}$$

The duration estimation seems fine, due to the complexity of the realization of all the part of the project.

3 Schedule

In this section we are going to estimate a possible schedule of the whole project. We used a Gantt graph to display our result.
The sequence of tasks involving this project are:

1. Preparation of the Requirement Analysis and Specification Document, containing the goals, the domain assumptions, and the functional and nonfunctional requirements of the project. There is a fixed deadline for the RASD delivery (blue in the graph).

2. Preparation of the Design Document, containing the architecture and the design of the project. There is a fixed deadline for the DD delivery (blue in the graph).

3. Preparation of the Integration Testing Plan Document, containing the strategy used to perform integration testing on the system. There is a fixed deadline for the ITPD (blue in the graph).

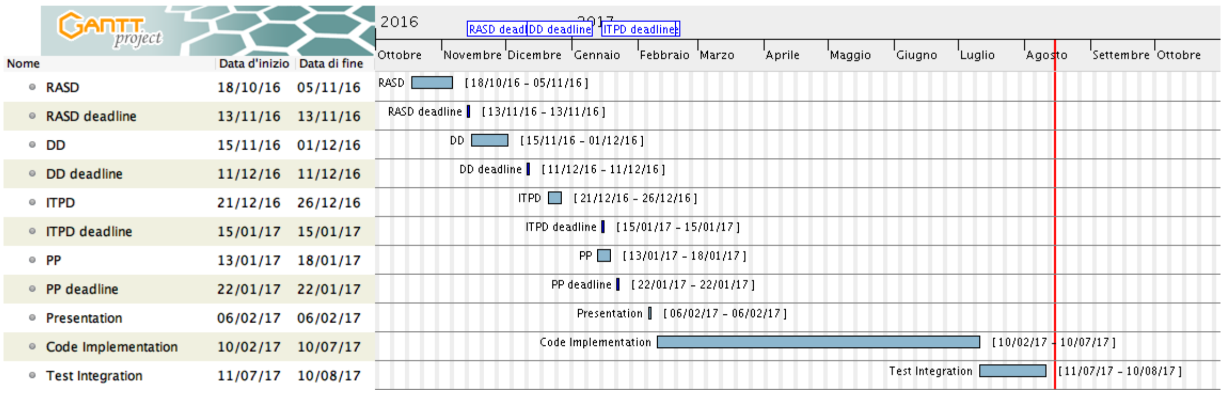
4. Preparation of the Project Plan, which is this document. There is a fixed deadline for the PP (blue in the graph).

5. Presentation of a brief explanation ($\sim$ 30 min) about the delivered documents, with slides, in a fixed day (6 February).

6. Implementation of the software system (coding) and write unit tests.

7. Perform integration testing on the software developed.

All the points are not strictly fixed, but can be iterated multiple times, as new requirements emerge, new choices are made. In particular, unit and integration testing will be continuously performed throughout the development process. Note also that some tasks need to be concluded before some other can begin, as the code implementation cannot happen before the design of the software, or the integration testing cannot happen before the code is implemented and the integration plan is made.
The last two parts of the schedule, that are the code implementation and the integration testing have not a real deadline. However, using the previously calculated data about the estimated duration of the project, we can use the average duration, so that the whole project will last for about 10 month, so from October 2016 to August 2017. We also fixed a "possible" deadline for the last step of the project, that is the 15th of August 2017.

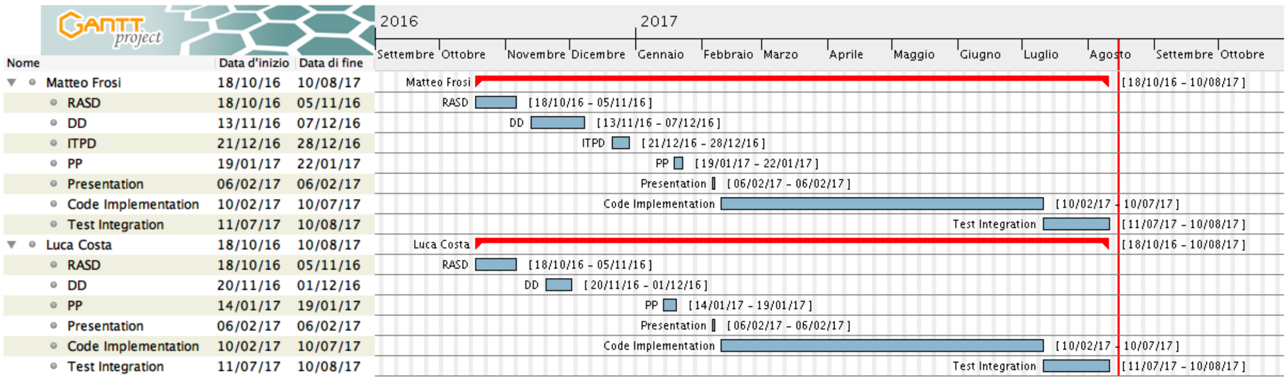


Nome	Data d'inizio	Data di fine
RASD	18/10/16	05/11/16
RASD deadline	13/11/16	13/11/16
DD	15/11/16	01/12/16
DD deadline	11/12/16	11/12/16
ITPD	21/12/16	26/12/16
ITPD deadline	15/01/17	15/01/17
PP	13/01/17	18/01/17
PP deadline	22/01/17	22/01/17
Presentation	06/02/17	06/02/17
Code Implementation	10/02/17	10/07/17
Test Integration	11/07/17	10/08/17

Figure 3: schedule of the project

4 Resource allocation

In this section we are going to map the different tasks on the team working at the project (us). Our team is composed of two persons, so we have theoretically split every part of the project in two.
However, during working process, it happens that the working time will not be the same because of common daily issues.
As we said previously, we considered a full project development to represent a more realistic estimation, that won't be really done (implementation and testing).
Almost every task involves the contribution of each member. This allows a more efficient and secure control over the work of the team members, due to the fact that there is more communication and possible incoherence or misunderstandings are avoided. Obviously each task can be further broken into smaller sub tasks, respectively representing the actual work done on the project. Since until now every deadline has been respected, the division of the work load coincides with what has been done for the delivery of the project documents, including this. For further information related to such division, all the documents can be consulted, checking the changelog and the hours of work section, at their end.
Figure 4 represents an high level allocation of the resources, considering small chunk of work, grouped by main task (that are the main documents and the real implementation).



Nome	Data d'inizio	Data di fine
Matteo Frosi	18/10/16	10/08/17
RASD	18/10/16	05/11/16
DD	13/11/16	07/12/16
ITPD	21/12/16	28/12/16
PP	19/01/17	22/01/17
Presentation	06/02/17	06/02/17
Code Implementation	10/02/17	10/07/17
Test Integration	11/07/17	10/08/17
Luca Costa	18/10/16	10/08/17
RASD	18/10/16	05/11/16
DD	20/11/16	01/12/16
PP	14/01/17	19/01/17
Presentation	06/02/17	06/02/17
Code Implementation	10/02/17	10/07/17
Test Integration	11/07/17	10/08/17

Figure 4: resource allocation of the project

5 Risk Management

In this section we are going to discuss some of the risks, that can be technical or economical, that the project could face during its development. Obviously not every single risk will be considered because it's impossible to predict what will really happen, hence what follows will represent a simple estimation.

5.1 Project risks

5.1.1 Delays

Let's consider the team of the project : a programmer is supposed to be skilled and working 8 hours a day. This estimation fails when the programmer gets ill or quits. A solution to this risk could be splitting the task between more programmers, so that when one of them gets ill, the others are able to work as well. This is not the best approach because it can lead to an increase of time of work, due to the preparation of new personnel. However, if a person quits, the project doesn't fail and can be finished by someone else. Lastly, even a bad estimation of the required time for each task can lead to a delay in the delivery of such task.

5.1.2 Lack of communication among the team

Those kind of difficulties can be overcome by explicitly defining the responsibilities and tasks to be done by each team member, and by writing clear and complete specification and design documents.

5.1.3 Lack of experience in programming

The team has no concrete experience in programming using Java EE. This will require more time during the coding phase, slowing the development phase.

5.1.4 Requirements change

The requirements may change during the development in unexpected way. This risk can't be prevented, but can be mitigated by writing reusable and extensible software.

Risk	**Probability**	**Effect**
Delays	Moderate	Moderate
Lack of communication	Low	Moderate
Lack of experience	Certain	Moderate
Requirements change	Low	Moderate

5.2 Technical risks

5.2.1 Data loss

Data can be lost due to various reasons, such as hardware failures or wrong usage of merging tools. This problem is easily avoided enforcing a periodic backup plan. Backups should be kept into a separate device from the main storage one.

5.2.2 Hardware failure

Cars should have a proper setup in order to work, especially hardware. The most critical factor is given by the VehicleInterface component, that has both software and hardware components. More attention is then required to avoid possible problems related to the car piece of project implementation, thus more work time.

5.2.3 Integration testing failure

Some components and subsystems may not pass the integration testing phase: this can require to rewrite large pieces of software. This risk can be reduced by defining precisely the interfaces between components and subsystems, and by doing integration tests early using stubs and drivers.

5.2.4 Downtime / unavailability of service

The system could go down for excessive load, software bugs, power outages or connection services faults. While the first two can be mitigated with a specific test on the hardware and software composing our system, the latter ones are unavoidable.

5.2.5 Incremental code

Being the basic structure monolithic, there is a good chance that the growth of the code may lead to an unreadable and messy agglomerate of modules and, in the worst case, to difficult understanding of the whole code. This can be avoided making the project extremely modularized, improving the quality of the Design Document before the code implementation phase. Moreover, code inspection can be done periodically. The adopted language helps us in giving a defined structure to our system.

Risk	**Probability**	**Effect**
Data loss	Very low	Catastrophic
Hardware failure	Low	High
Integration testing failure	Very low	High
Downtime	Low	Moderate
Incremental code	Low	Moderate

5.3 Economical risks

5.3.1 Bankruptcy

The income from the sales of the software may not be enough to sustain the development, maintenance and deployment of the software system. A good feasibility study helps to avoid this situation.

5.3.2 Competitors

Other companies can build equivalent or better products at a lower price and put PowerEnJoy out of use. This may not be very probable, because important components are self-made and so unique for our system, like the VehicleInterface. Car sharing is not a new business, so it is important to focus on improving some aspects of the service, to have a competitive advantage compared to the other sharing services.

Risk	**Probability**	**Effect**
Bankruptcy	Moderate	Catastrophic
Competitors	Low	Severe

5.4 User satisfaction risks

5.4.1 Requirements dissatisfaction

Some of the client requests may have not been accomplished or satisfied (like features not included or UIs not very friendly), leading to a loss of time to better improve the software, or even the future loss of the client. A possible way to avoid it is having a continuous dialogue with the client, to improve the quality and level of detail of the Requirement Analysis and Specification document.

5.4.2 Bad public opinion / unknown service

Apart from the initial period, gaining popularity is the key to back up the project. Once the project is presented, it is important to start collecting possible future users. Otherwise the risk of not being supported may lead to a failure. Hence it is important to properly advertize the app.

Risk	**Probability**	**Effect**
Requirements dissatisfaction	Low	Moderate
Bad public opinion / unknown service	Moderate	Severe

6 Document further information

6.1 References

- RASD_4.8, before version 4.8
- DD_3.3, before version 3.3
- ITPD_1.1, before version 1.1
- Assignments AA 2016-2017.pdf
- PP1.1.pdf
- ProjectPlan.pdf
- Project planning example document.pdf
- `http://www.qsm.com/resources/function-point-languages-table`, as a reference to select the gearing factors
- `http://sunset.usc.edu/research/COCOMOII/expert_cocomo/drivers.html`, as a reference to all the COCOMO II scale and cost drivers standards
- `http://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_modelman2000.0.pdf`, the COCOMO II manual

6.2 Used tools

In creating the PP document, the following tools have been used:

- Github, for version controller
- LyX, to write the document and converting in .pdf format
- Hunspell, for the spell check of the document
- Atom, for the project markdown
- GanttProject: to draw Gantt diagrams

6.3 ChangeLog

- [28/12/2016] [**Version 0.1**] :: Introduction of the document, with brief explanation of the estimation section.
- [15/01/2017] [**Version 0.2**] :: Function point part added, section 1.
- [17/01/2017] [**Version 0.3**] :: Estimation part concluded, section 2.
- [18/01/2017] [**Version 0.4**] :: Schedule and Resource allocation parts introduced, section 3 and 4.
- [19/01/2017] [**Version 1.0**] :: Risk management part introduced, section 5.
- [20/01/2017] [**Version 1.1**] :: Added images links.

- [21/01/2017] [**Version 1.2**] :: Detailed rewriting of section 5, about risk management, section 4, about resource allocation, and section 3, about the project schedule. Revision of the estimation part of the document, including the FP estimation and the calculus of the project duration and effort required.

6.4 Hours of work

Matteo Frosi [29/12/2016]: 1.00 hours (introduction of the document and writing of the document structure)
[20/01/2017]: 2.00 hours (first revision of the document and changes over the effort, cost and code lines estimation)
[21/01/2017]: 5.00 hours (completion and rewriting of some parts of the document, like scheduling, resource allocation. Almost complete rewriting of risk management chapter).
TOTAL: 8.00 hours

Luca Costa [14 / 01 / 17] : 2.30 hours (informing and writing first part about Function Points)
[15 / 01 / 17] : 3 hours (writing and completing the Function Points part)
[16 / 01 / 17] : 3 hours (informing and writing the effort and cost estimation part with COCOMO)
[17 / 01 / 17] : 1.45 hours (writing and completing the effort and cost estimation part with COCOMO)
[18 / 01 / 17] : 1.30 hours (thinking and writing schedule and resource allocation)
[19 / 01 / 17] : 2.15 hours (thinking and writing the risk management part)
TOTAL : 14 hours