Figure 1: Politecnico di Milano

Project Plan Document
Version 1.0

Emanuele Ricciardelli (mat. 875221)
Giorgio Tavecchia (mat. 874716)
Francesco Vetró (mat. 877593)

January 22, 2017

Contents

1 Introduction

1.1 Purpose and scope

The purpose of this document, the Project Plan, is to analyze the complexity of the PowerEnJoy system in order to make a more accurate prediction of the costs and efforts necessary for the development of all the components of the system. These data will then be useful to define budget, schedule and subdivision of resources and tasks within the team. This document will be structured in 3 macro parts:

- The first section will deal with the analysis necessary to define an estimate of the size of the project, in function of lines of code, as well as the cost / effort. This analysis will be conducted in terms of Function Points and COCOMO II approach.
- The second section will present a schedule for the project so as to ensure a proper distribution of tasks and times in order to achieve all of the system requirements, implementation and testing aspects.
- In the third section we will discuss about risk management, through an analysis of possible problems that may arise during the development and the applicable remedies.

1.2 Definitions and abbreviations

- RASD: Requirement Analysis and Specification Document.
- DD: Design Document
- ITPD: Integration Test Plan Document
- PM: person-months
- KSLOC: kilo-source lines of code
- ILF: Internal Logic Files
- ELF: External Interface Files
- EQ: External Inquiries
- EI: External Inputs
- EO: External Output

1.3 Reference documents

- Specication document: Assignments AA 2016-2017.pdf
- RASD Version 1.1
- DD Version 1.1

- ITPD Version 1.0
- Project planning example document.pdf
- Function Point Languages Table: *http://www.qsm.com/resources/function-point-languages-table*
- CII_modelman2000.0.pdf

2 Project size, cose and effort estimation

This section will focus on providing an estimate of the size of the system that will be, as well as costs and development efforts. Regarding the estimate of the size of the future system, we will make use of Function Point which will allow us to make an estimate based on the features that the system will have to offer. Concerning the estimated costs and effort related to the development, we will make use of COCOMO II and we will integrate the results previously obtained.

2.1 Size estimation: function points

In this chapter we will face an approach through Function Point in order to estimate the size of the system to be developed, using as a unit of measurement the KSLOC. In particular, we are going to make a calculation of all the Function Points of the system, by the following equation:

$$\text{UFP} = \sum (\#\text{ elements of a given type} \times \text{weight})$$

The possible types of elements affecting the expression concern:

- Data structure;
- Inputs and outputs;
- Inquiries;
- External Interface.

We provide a definition for each of them in the relevant section. While, with respect to the definition of the weights, we are going to use the following table, which allows us to define, for each element, the weight as a function of the inherent complexity of its implementation.

Table 1: Complexity weight

Function type	Low	Average	High
Internal Logic Files	7	10	15
External Interface Files	5	7	10
External Inputs	3	4	6
External Inquiries	3	4	6
External Outputs	4	5	7

2.1.1 Internal Logic Files (ILFs)

With Internal Logic Files we define internal data generated by the system, and used and maintained by it. This set contains homogenous data handled by the system, composed by all the data structure saved in the DBMS. In the list below we list the tables which compose our database. They fill the Internal Logic Files set.

- User(Password, email, name, surname,drivingLicenseNumber)
- Reservation(ID,userPassword,licensePlateCard, date,startingTime,endingTime)
- Car(licensePlate, model)
- MaintenanceRecord(ID,description,date, licensePlate)
- Sensor(ID, licensePlate)
- ExtraInRes(ID, IDextra, IDreservation)
- Extras(ID, type, value)
- Payment(prog_number, IDReservation, pending, date)
- Ride(ID, IDReservation, startingTime, endingTime, bill, lastBatteryLevel,date, startingSafeArea, endingArea)
- Ticket(IDTicket, IDRide, date)
- Area(ID, position)
- SafeArea(IDArea)
- UnsafeArea(IDArea)
- PowerGrid(ID, capacity, IDSafeArea)
- BonusMalus(ID, amount, description)
- AssignedBonusMalus(IDRide, IDBonusMalus)

With respect to the previous table of complexity weight, we decide to assign a *LOW* level to those tablet that are composed by a small number of attributes, while we define as *AVERAGE* level of complexity tables such as Reservation that is derived from the interaction of multiple tables. For the same reason, but with an higher level of complexity, we assign to the Ride table the *HIGH* level.

Table 2: ILFs

ILF	Complexity Level	FPs
User	Low	7
Reservation	Avg	10
Car	Low	7
Maintenance Record	Low	7
Sensor	Low	7
ExtraInRes	Low	7
Extras	Low	7
Payment	Low	7
Ride	High	15
Ticket	Low	7
Area	Low	7
Safe Area	Low	7
Unsafe Area	Low	7
Power Grid	Low	7
BonusMalus	Low	7
AssignedBonusMalus	Low	7
	TOT	123

2.1.2 External Interface Files (EIFs)

With External Interface Files we identify a set of data used by the PowerEnJoy system that are not stored in it but provided and handled by external companies. With respect to the PowerEnJoy service, the external data are provided by Google Services and in particular by Google Maps. The services above, are used by our system making use of the API provided by Google itself. Features requiring their use are as follows:

- Provide directions, given the current position of a car, to a charging station.

This functionality is implemented in the Money-Saving option, in which the user is able to enter their destination and the system returns the location of the nearest charging station to the destination, providing at the same time a uniform distribution of cars in the city; this position will later be used by the API to calculate the optimal route and return there. Given the complexity of the operations involved (especially those linked to the uniform distribution of cars in the city) we identify this feature as a *AVG* complexity.

Table 3: ELFs

ELF	Complexity Level	FPs
Money saving option	Avg	7

2.1.3 External Inputs (EIs)

With External Input we identify a set of elementary operation to elaborate data coming from the external environment. As defined in previous documents, our system interfaces with a number of clients aimed at different types of users, in particular we will have functionality addressed to users of car sharing service, the maintenance system and the payment system. In particular for each of them we will provide the following functionalities:

- Users of the car sharing service:

  - Login/Logout: the login and logout are key features to our system but there are translated into simple queries on the database, which is why the level of complexity is *LOW*.

  - Registration: the registration system is also based on a number of queries on the database, to which, however, are added a series of controls regarding the information entered, the validity of the license provided and the accuracy of data relating to the method of payment chosen. Despite this, it remains a *LOW* complexity.

  - Reserve a car: booking a car is a complex operation that requires a series of queries, followed by several controls both the user (Eg. check if it is suspended) and the selected car (Eg. make sure it is available) and finally a communication with the vehicle to inform the reservation. the multiplicity of the transactions underlying the achievement of this feature allows us to indicate it as a *AVG* complexity.

  - Delete a reservation: linear operation and *LOW* complexity since it corresponds to the simple closure of the reservation and the consequent reflection on the database.

  - Extend a reservation: the operations involved are the same as the *Delete a reservation*, the only addition corresponds to the operations necessary to register the payment due to the extension itself. For this reason, the level of complexity is *LOW*.

  - Start a ride: the operation to *start a ride* requires an exchange of information between the car and the system, in particular the operation can begin when the system validates the unlock code used on the car and the engine is ignited. The underlying operations of this feature affect several components that are going to instantiate different entity in the database. For these reason we define an *AVG* complexity.

  - End a ride: like the previous functionality, are involved different components and entities in the database. the complexity, also in this case, is *HIGH* because the system does not only apply queries on the database but compute all the necessary element to end a ride, like computing the payment and various discounts. This functionality is more complex than Start a Ride because in order to fulfill its task, it requires the product of more subprocesses like computing the total amount, performing the payment instantiation, unreserved the car and so on.

  Report a issue: reporting a problem to a car, it may seem a simple task since it requires only the creation of a new record maintenance, but underneath resides the management of the whole reservation or ride that interested the machine. for this reason the average complexity is *AVG*.

- Maintenance system:

  - Update a record about a car: the status update concerning a record of a car by the maintenance is a simple task that requires only to update tuples in the database related to the reported car. The complexity is *LOW*.

- Payment system:

  - Update payment informations: just as in the case of update by maintenance, the only operation involved is upgrading tuples in the database for a specified payment. The complexity is *LOW*.

Table 4: EIs

EI	Complexity Level	FPs
Login/Logout	Low	3×2=6
Registration	Low	3
Reserve a Car	Avg	4
Delete reservation	Low	3
Extend reservation	Low	3
Start ride	Avg	4
End ride	High	6
Report issue	Avg	4
Update record car	Low	3
Update payment information	Low	3
	TOT	39

2.1.4 External Inquiries (EQs)

With External Inquiries we define a set of elementary operation that involves input and output operations. With regard to this definition and using the previous subdivision related to different users of the system, we provide the following functionalities:

- Users of the car sharing service:

  - Retrieve information about his reservation

  - Retrieve unlock code: the previous two steps possess the same level of complexity as they are the result of a simple query to the database, for this reason, is associated to them a LOW complexity.

  - Retrieve payment history: since the payment-related information is obtained through analysis of all the costs relating to reservations and ride (e.g. the extension of a reservation or the application of discounts), the complexity of this operation is AVG.

  - View available cars: the identification of available cars is the result of an analysis of the non-reserved cars or currently in use /in maintenance. For this reason, the complexity is LOW.

- Maintenance system:

  - Retrieve a list of reported issue: as well as retrieving information associated with a reservation, cited above, retrieving reports is a simple task, result of an equally simple database query. Complexity LOW.

Table 5: EQs

EQ	Complexity Level	FPs
Retrieve reservation info	Low	3
Retrieve unlock code	Low	3
Retrieve payment history	Avg	4
View available cars	Low	3
Retrieve list of reported issues	Low	3
	TOT	16

2.1.5 External Outputs (EOs)

With External Output we define functionalities that our system perform and generate data to the output. With respect to this definition, we can define the following functionalities:

- Notify a user for the correct registration;
- Notify a user that a reservation has been performed;
- Notify a user if the car reserved is no longer available;
- Notify a user for the payment of a ride;
- Notify the maintenance system if a new issue is reported;
- Notify the payment system if a new debt is created. All the notifications listed above are considered simple and for this reason the LOW complexity.

Table 6: EOs

EO	Complexity Level	FPs
Notify correct registration	Low	4
Notify performed reservation	Low	4
Notify car no longer available	Low	4
Notify payment	Low	4
Notify new issue	Low	4
Notify new debt	Low	4
	TOT	24

2.1.6 Overall estimation

Given the following summary table:

Table 7: Summary

Function type	Value
Internal Logic Files	123
External Interface Files	7
External Inputs	39
External Inquiries	16
External Outputs	24
Total	209

$$\text{LOC}=\text{AVC}\times\text{number of function points}$$

Considering Java Enterprise Edition as the development platform, we can assign the value AVC following the Function Point Language Table linked in the document references. For the average number of SLOC AVC is equal to 46:

$$\text{SLOC}= 209\times46= 9614$$

For the computing an upperbound for the number of SLOC AVC is equal to 67:

$$\text{SLOC}=209\times67=14003$$

2.2 Cost and effort Estimation: COCOMO II

In this section, as previously expressed in the purpose of the document, we will use the COCOMO II approach in order to estimate cost and effort needed to develop the PowerEnJoy system. In particular we will focus on a Post-Architecture approach since we have already detailed informations about the different phases of development, and so we can achieve a more accurate cost estimate. We now provide an accurate analysis of all the elements needed to perform the following effort equation:

$$\text{PM}=\text{A}\times\text{Size}^{E} \times \prod_{1<=i<=n} EM_i$$

In particular, the meaning of the coefficients is the following:

- A: Its an approximation of the productivity constant in PM/KSLOC. This coefficient is standardly set to 2.94;
- Size: is the estimated size of the project in KSLOC, provided in the previous chapter;
- EM_i: effort multiplier, derived for each cost driver;
- E: is an aggregation of five Scale Factors.

2.2.1 Scale factors

The first parameter that we will estimate is E, obtained by the following equation, a function of scale factors that we will soon identify.

$$\text{E=B+0.01}\times \sum_{1<=j<=5} SF_j$$

Where B is set to 0.91, as an empirically-proven quantity for COCOMO II. Regarding SF_j, each member of the summation is equivalent to a coefficient of the following table, chosen in function of the range of verylow-extrahigh values, characteristic of each of them.

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_i:	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_i:	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_i:	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_i:	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_i:	7.80	6.24	4.68	3.12	1.56	0.00

Figure 2: Scale Factors

For each of these factors, we will now provide a definition and the reasoning behind the chosen value:

- **Precedentedness (PREC)**: The value of this coefficient decreases much higher is the degree of experience that the team presents in respect of a similar project. In particular, our experience level is *LOW* because we have not previously participated in the development of systems of this size.

- **Development Flexibility (FLEX)**: The degree of development flexibility reflects the flexibility that the project can presents regarding the requirements and objectives on which it was founded. The PowerEnJoy system has been defined in relation to requirements for the car sharing service that have stiffness from the point of view of the required functionality, with little choice margin, while from the architectural point of view is more flexible since there are no systems legacy to which it is necessary to integrate, but, at the same time, the type of service involves the development of an architecture able to ensure certain levels of operations. For this reason, we assign a *NOMINAL* value.

- **Risk Resolution (RSL)**: This factor reflects the ability of the development team to recognize and react promptly to the risks and problems that it could be encountered along the development. Also in relation to the same subject discussed in detail in the final chapter of this document, we assign the value *HIGH*.

- **Team Cohesion (TEAM)**: The Team Cohesion scale factor accounts for the sources of project turbulence and entropy because of difficulties in synchronizing the projects stakeholders: users, customers, developers etc. These difficulties may arise from differences in stakeholder objectives and cultures, difficulties in reconciling objectives and developers lack of experience and familiarity in operating as a team. With respect to this definition we can assign a *VERY HIGH* level, since the development team is collaborative and open to dialogue.

- **Process Maturity (PMAT)**: This factor is directly linked with the CMMI index, in particular, related to our system, the development project has been properly managed, planned and executed with stakeholders involved and resources rightly distributed. For this rationale, we assign a Level 2, so a *NOMINAL* factor.

We can now compute the previously presented equation:

Table 8: Scale Factor

Scale Factor	Factor	Value
Precedentedness (PREC)	Low	4.96
Development Flexibility (FLEX)	Nominal	3.04
Risk Resolution (RSL)	High	2.83
Team Cohesion(TEAM)	Very High	1.10
Process Maturity (PMAT)	Nominal	4.68
		16.61

Using the previously defined equation:

$$E= 0.91+0.01\times16.61=1.0761$$

2.2.2 Cost drivers

We now carry out an analysis of the Cost Drivers in order to estimate the value of EM in the equation at paragraph 2.2. As previously indicated, we will use a post-architecture approach, thus providing an accurate computation of the relative cost drivers.

- **Analyst Capability (ACAP)**: This driver measures the degree of reliability of the estimates and analyzes which have been carried out for the design of the project. In relation to PowerEnJoy system, we can claim to have performed a detailed analysis not only of the features to be implemented, but also about the environment in which the system have to operate, also investigating possible improper user in order to prevent the safeguard of the company. However, we have performed a series of assumptions, specified in the document RASD, which affect this analysis, allowing us to assign a *HIGH* level.

ACAP Descriptors:	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.42	1.19	1.00	0.85	0.71	n/a

Figure 3: ACAP

- **Programmer Capability(PCAP)**: The ability of our team is not only to be identified in the development from scratch but also in the use of APIs and services of third parties, which is why, despite the team's knowledge may not be high in any sphere, the high degree of dialogue and self-critical learning allow to better manage any eventuality. For this reason we define an *HIGH* level.

PCAP Descriptors	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.34	1.15	1.00	0.88	0.76	n/a

Figure 4: PCAP

- **Personnel Continuity (PCON)**: This rating is in terms of the projects annual personnel turnover. In particular, our team is composed by a limited number of people, and the possible turnover is very limited: so we assign a *VERY HIGH* value.

PCON Descriptors:	48% / year	24% / year	12% / year	6% / year	3% / year	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.29	1.12	1.00	0.90	0.81	

Figure 5: PCON

- **Required Software Reliability (RELY)**: This is the measure of the extent to which the software must perform its intended function over a period of time. With respect to the PowerEnjoy system, a software failure could block the functioning of the entire car sharing system, causing great losses in terms of money and image for the company. This driver cost is set to *HIGH*.

RELY Descriptors:	slight inconvenience	low, easily recoverable losses	moderate, easily recoverable losses	high financial loss	risk to human life	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	0.82	0.92	1.00	1.10	1.26	n/a

Figure 6: RELY

- **Database Size (DATA)**: This cost driver attempts to capture the effect large test data requirements have on product development. The rating is determined by calculating D/P, the ratio of bytes in the testing database to SLOC in the program. Since the car sharing service produces a high amount of information, divided between data relating to reservations, payments, users subscribed and other factors, given the amount of SLOC computed before, we can assign the value *VERY HIGH*.

DATA* Descriptors		Testing DB bytes/Pgm SLOC < 10	10 ≤ D/P < 100	100 ≤ D/P < 1000	D/P ≥ 1000	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	n/a	0.90	1.00	1.14	1.28	n/a

* DATA is rated as Low if D/P is less than 10 and it is very high if it is greater than 1000. P is measured in equivalent source lines of code (SLOC), which may involve function point or reuse conversions.

Figure 7: DATA

- **Product Complexity (CPLX)**: With respect to the COCOMO II definition of this parameter, defined by the usage of the following tables, we assign the *NOMINAL* level.

	Control Operations	Computational Operations	Device-dependent Operations	Data Management Operations	User Interface Management Operations
Very Low	Straight-line code with a few non-nested structured programming operators: DOs, CASEs, IF-THEN-ELSEs. Simple module composition via procedure calls or simple scripts.	Evaluation of simple expressions: e.g., A=B+C*(D-E)	Simple read, write statements with simple formats.	Simple arrays in main memory. Simple COTS-DB queries, updates.	Simple input forms, report generators.
Low	Straightforward nesting of structured programming operators. Mostly simple predicates	Evaluation of moderate-level expressions: e.g., D=SQRT(B**2-4.*A*C)	No cognizance needed of particular processor or I/O device characteristics. I/O done at GET/PUT level.	Single file subsetting with no data structure changes, no edits, no intermediate files. Moderately complex COTS-DB queries, updates.	Use of simple graphic user interface (GUI) builders.
Nominal	Mostly simple nesting. Some intermodule control. Decision tables. Simple callbacks or message passing, including middleware-supported distributed processing	Use of standard math and statistical routines. Basic matrix/vector operations.	I/O processing includes device selection, status checking and error processing.	Multi-file input and single file output. Simple structural changes, simple edits. Complex COTS-DB queries, updates.	Simple use of widget set.

Figure 8: CPLX1

	Control Operations	Computational Operations	Device-dependent Operations	Data Management Operations	User Interface Management Operations
High	Highly nested structured programming operators with many compound predicates. Queue and stack control. Homogeneous, distributed processing. Single processor soft real-time control.	Basic numerical analysis: multivariate interpolation, ordinary differential equations. Basic truncation, round-off concerns.	Operations at physical I/O level (physical storage address translations; seeks, reads, etc.). Optimized I/O overlap.	Simple triggers activated by data stream contents. Complex data restructuring.	Widget set development and extension. Simple voice I/O, multimedia.
Very High	Reentrant and recursive coding. Fixed-priority interrupt handling. Task synchronization, complex callbacks, heterogeneous distributed processing. Single-processor hard real-time control.	Difficult but structured numerical analysis: near-singular matrix equations, partial differential equations. Simple parallelization.	Routines for interrupt diagnosis, servicing, masking. Communication line handling. Performance-intensive embedded systems.	Distributed database coordination. Complex triggers. Search optimization.	Moderately complex 2D/3D, dynamic graphics, multimedia.
Extra High	Multiple resource scheduling with dynamically changing priorities. Microcode-level control. Distributed hard real-time control.	Difficult and unstructured numerical analysis: highly accurate analysis of noisy, stochastic data. Complex parallelization.	Device timing-dependent coding, micro-programmed operations. Performance-critical embedded systems.	Highly coupled, dynamic relational and object structures. Natural language data management.	Complex multimedia, virtual reality, natural language interface.

Figure 9: CPLX2

Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	0.73	0.87	1.00	1.17	1.34	1.74

Figure 10: CPLX3

- **Documentation match to life-cycle needs (DOCU)**: The rating scale for the DOCU cost driver is evaluated in terms of the suitability of the projects documentation to its life-cycle needs; with respect to our system the documentation provided is complete and cover all the necessary elements of the life-cycle, from the requirements analysis to the integration testing. For this reason we assign a *NOMINAL* level.

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 Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	0.81	0.91	1.00	1.11	1.23	n/a

Figure 11: DOCU

- **Developed for Reusability (RUSE)**: This cost driver reflects the additional effort needed to construct components intended for reuse on current or future projects. In particular, our components are strictly linked with the project itself but are designed in order to be expandable for future evolution of the service. Despite those consideration, however, we have decided to invest some effort in order to detect functionalities which are evidently (and so easily) generalizable for using across programs. We assign a *HIGH* value.

RUSE Descriptors:		none	across project	across program	across product line	across multiple product lines
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	n/a	0.95	1.00	1.07	1.15	1.24

Figure 12: RUSE

- **Execution Time Constraint (TIME)**: This cost driver reflect the time constraint imposed upon the software system, in particular it express the percentage of available execution time expected to be used by the system. Since the car sharing is a 24h/7days service, we suppose to have an average level of usage (average in terms of different time of the day, for example) set to *VERY HIGH*, since we expect to have a large number of users connected and using the system.

TIME Descriptors:			≤ 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 Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	n/a	n/a	1.00	1.11	1.29	1.63

Figure 13: TIME

- **Main storage constraint (STOR)**: This parameter reflects the expected amount of storage usage with respect to the availability of the hardware. Our system produces a huge amount of data every day but since the increasing dimension of disk storage availability in these years, we set this value to *NOMINAL*.

STOR Descriptors:			≤ 50% use of available storage	70% use of available storage	85% use of available storage	95% use of available storage
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	n/a	n/a	1.00	1.05	1.17	1.46

Figure 14: STOR

- **Platform Volatility (PVOL)**: This parameter points out the possible need to make updates to the system platforms, both hardware and software. In the short term there are no structural changes, however, you may need to update the user interface in case of the introduction of new features. We expect to implement new update on a *HIGH* level.

PVOL Descriptors:		Major change every 12 mo.; Minor change every 1 mo.	Major: 6 mo.; Minor: 2 wk.	Major: 2 mo.;Minor: 1 wk.	Major: 2 wk.;Minor: 2 days	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	n/a	0.87	1.00	1.15	1.30	n/a

Figure 15: PVOL

- **Application Experience (APEX)**: This driver is defined in terms of the project team level of experience with this type of application. Since is the first time, speaking for each member, to manage and develop an application with this dimension and amount of functionalities, the experience is limited to the 6 months of course lessons. We assign the *LOW* value.

APEX Descriptors:	≤ 2 months	6 months	1 year	3 years	6 years	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.22	1.10	1.00	0.88	0.81	n/a

Figure 16: APEX

- **Platform Experience (PLEX)**: The team has some previous experience in database management, user interfaces, client-server approach. The average knowledge about this kind of experience is comparable with a year so, with respect to the following table, we assign the value *NOMINAL*.

PLEX Descriptors:	≤ 2 months	6 months	1 year	3 years	6 year	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.19	1.09	1.00	0.91	0.85	n/a

Figure 17: PLEX

- **Language and Tool experience (LTEX)**: This parameter reflects the knowledge of the team about languages and tools that will be used in the system development. With respect to java language, J2EE, HTML and CSS the average knowledge of the team is a strong underground knowledge comparable with about 3 years. We set this value to *HIGH*.

LTEX Descriptors:	≤ 2 months	6 months	1 year	3 years	6 year	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.20	1.09	1.00	0.91	0.84	

Figure 18: LTEX

- **Use of software tools (TOOL)**: This parameter underlines the importance of the usage of tools well integrated in the environment in which the system is developed. Since our application make use of mature tools (as expressed in the architectural design in DD), we set this value to *HIGH*.

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 Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.17	1.09	1.00	0.90	0.78	n/a

Figure 19: TOOL

- **Multisite development (SITE)**: The development project has taken place mainly in remote, the team members has been kept informed each other regularly in case of autonomous work, however, the team has preferred to use conferencing tools to communicate and address the most important issues. For this reason we assign a *VERY HIGH* value.

SITE: Collocation Descriptors:	Inter-national	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 communicat ion.	Wideband elect. comm., occasional video conf.	Interactive multimedia
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.22	1.09	1.00	0.93	0.86	0.80

Figure 20: SITE

- **Requirement development schedule (SCED)**: This rating measures the schedule constraint imposed on the project team developing the software, in our particular case there were set strict deadline, so the schedule could not be relaxed, so we set the value to *NOMINAL*.

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 Multiplier	1.43	1.14	1.00	1.00	1.00	n/a

Figure 21: SCED

We now summarize all the selected cost drivers:

Table 9: Cost drivers

Cost driver	Factor	Value
Analyst Capability (ACAP)	High	0.85
Programmer Capability(PCAP)	High	0.88
Personnel Continuity (PCON)	Very High	0.81
Required Software Reliability (RELY)	High	1.10
Database size (DATA)	Very High	1.28
Product Complexity (CPLX)	Nominal	1.00
Documentation match to life-cycle needs (DOCU)	Nominal	1.00
Developed for Reusability (RUSE)	High	1.07
Execution Time Constraint (TIME)	Very High	1.29
Main storage constraint (STOR)	Nominal	1.00
Platform Volatility (PVOL)	High	1.15
Application Experience (APEX)	Low	1.10
Platform Experience (PLEX)	Nominal	1.00
Language and Tool experience (LTEX)	High	0.91
Use of software tools (TOOL)	High	0.9
Multisite development (SITE)	High	0.93
Requirement development schedule (SCED)	Nominal	1.00
	Product	1.135

2.2.3 Effort equation

We are now able to perform the effort equation expressed in paragraph 2.2 that we now rewrite with the computed coefficients:

$$\text{PM}=\text{A}\times\text{Size}^{E}\times\prod_{1<=i<=n}EM_i$$

- A = 2.94
- Average Size=9,614KSLOC; Upperbound Size= 14,003KSLOC;
- $\prod_{1<=i<=n}EM_i = 1.135$
- E = 1.0761

So:

- Average PM = 38.11 person-months
- UpperBound PM = 57.12 person-months

2.2.4 Schedule estimation

The duration of the project schedule is computed using the following equation:

$$\text{Duration}=3.67\times\text{PM}^{0.28+0.2\times(E-B)}$$

Where: B is equal to 0.91 for COCOMO II E is equal to 1.0761, computed as function of the scale factors, as above PM is the effort computed with the effort equation at paragraph 2.2.3. So we obtain:

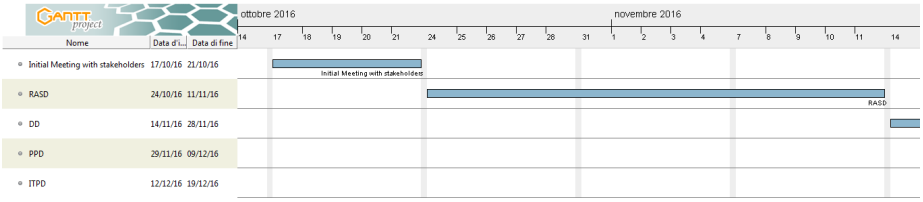
- Average duration = 11.48 months $\simeq$ 12 months
- Upperbound Duration= 13,03 months $\simeq$ 14 months

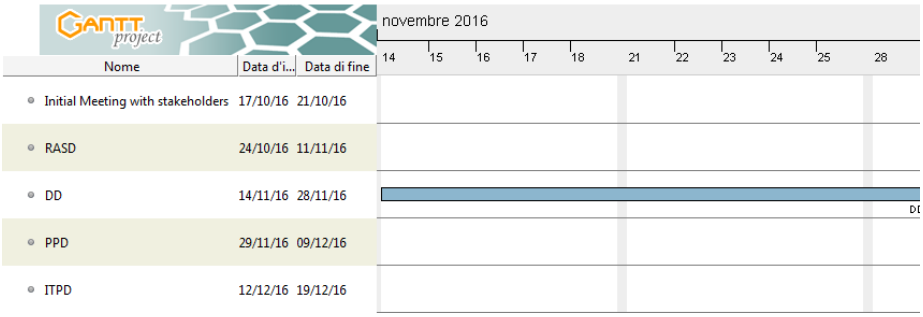
3 Schedule

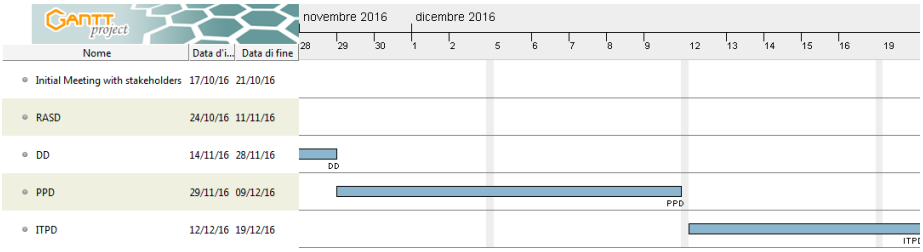
In this chapter were going to define a high-level schedule for the project. Its important to underline that the result of this schedule should not be considered as defined and unchangeable, primarily because of the unforeseen that could occur (see chapter related to Risk Management) and may not ensure the timely achievement of milestones. Another reason is that it may also be necessary to return to work on previous task in case of possible adjustments.

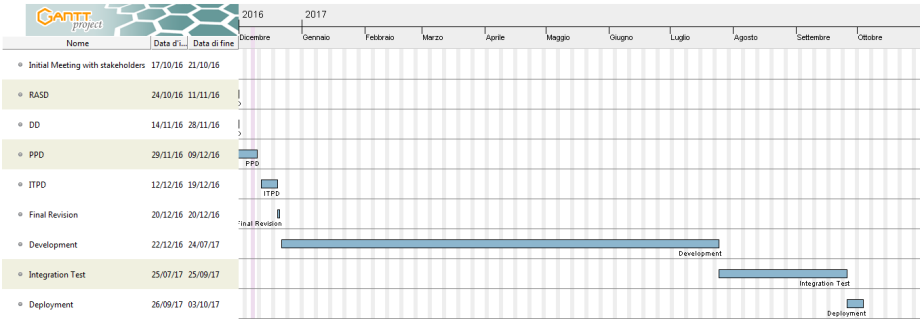
1. We now define the main tasks to be scheduled:
2. Meetings with stakeholders: in order to give a first definition of requirements and features that the system-to-be must implement.
3. Requirement analysis and specification document (rasd): definition of a first document containing a detailed description of all requirements (functional and non-funcional) and goals related to the project.
4. Design Document (DD): definito of a first document describing the architectural design chosen for the development of the system.
5. Integration test plan document (ITPD): definition of a first document containing the sequence of integration and procedure necessary to cover the integration testing of the components of the system.
6. Project Plan Document (PPD): definition of a first document containing an analys of the possible dimension of the project, an estimation of the effort and cost, the schedule, the resource allocation and the risk management for the project.
7. Development: Development of the software system
8. Integration Test: Integration testing of the software system
9. Deployment: Deliver and deployment of the software to the customers.

NB: The PP document is listed after the Design Document and not before because we are assuming that RASD and DD are already available and we could base this document also on what its written on them. To compute the following schedule, we used all data retrieved from the analysis in this document and the duration of the drafting of the previous documents; in particular, we assumed that the team will work 8 hours per day (so the deadlines of the real project do not coincide with the one related to the course) excluding 2 days a week.


GANTT project
ottobre 2016
novembre 2016
Nome
Data d'i...
Data di fine
Initial Meeting with stakeholders 17/10/16 21/10/16
RASD 24/10/16 11/11/16
DD 14/11/16 28/11/16
PPD 29/11/16 09/12/16
ITPD 12/12/16 19/12/16


GANTT project
novembre 2016
Nome
Data d'i...
Data di fine
Initial Meeting with stakeholders 17/10/16 21/10/16
RASD 24/10/16 11/11/16
DD 14/11/16 28/11/16
PPD 29/11/16 09/12/16
ITPD 12/12/16 19/12/16


GANTT project
novembre 2016
dicembre 2016
28 29 30 1 2 5 6 7 8 9 12 13 14 15 16 19
Nome
Data d'i...
Data di fine
Initial Meeting with stakeholders 17/10/16 21/10/16
RASD 24/10/16 11/11/16
DD 14/11/16 28/11/16
PPD 29/11/16 09/12/16
ITPD 12/12/16 19/12/16
DD
PPD
ITPD


GANTT project
2016
2017
Dicembre Gennaio Febbraio Marzo Aprile Maggio Giugno Luglio Agosto Settembre Ottobre
Nome
Data d'i...
Data di fine
Initial Meeting with stakeholders 17/10/16 21/10/16
RASD 24/10/16 11/11/16
DD 14/11/16 28/11/16
PPD 29/11/16 09/12/16
ITPD 12/12/16 19/12/16
Final Revision 20/12/16 20/12/16
Development 22/12/16 24/07/17
Integration Test 25/07/17 25/09/17
Deployment 26/09/17 03/10/17
PPD
ITPD
Final Revision
Development
Integration Test
Deployment

4 Resource allocation

In this paragraph we present the different steps that leads to the completion of all documents.

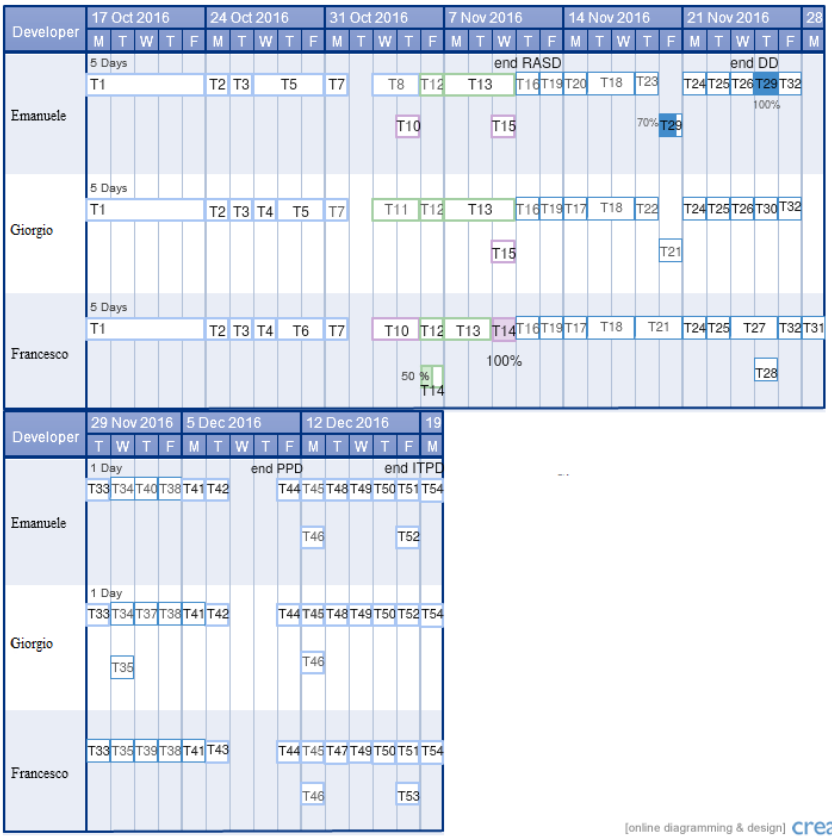


Table 10: Resource allocation

Document	Step
	1 - Meeting with stakeholders
RASD	2 -Discussion defining requirements
	3 - Discussion defining Goals and Text assumptions
	4 - Constraints Definitions
	5 - Functional Requirements
	6 - Non Functional Requirements
	7 - Meetings with stakeholders in order to check the requirements
	8 - Scenarios
	9 - Brief revision to check what has been done
	10 - Use Case and Class Diagram
	11 - Sequence, Activity and State Diagram
	12 - Brief revision to check what has been done
	13 - Alloy Modeling
	14 - Copy Word File into a LaTeXDocument
	15 - Last revision on RASD
DD	16 - Discussion defining Components and High-Level Components
	17 - High-Level Components Diagram
	18 - Component View Diagram
	19 - Discussion defining Database entities and database components
	20 - E-R Diagram
	21 - Deployment View Diagram and Style Architecture Patterns
	22 - Runtime Diagram
	23 - Component Interfaces
	24 - Brief revision to check what has been done
	25 - Discussion concerning algorithm
	26 - Algorithms Design
	27 - Mockups
	28 - Meetings with people in order to valuate the goodness of User Interface's Mockups
	29 - UX Diagrams
	30 - APIs and Requirements Traceability
	31 - Copy Word File into a LaTeXDocument
	32 - Last revision on DD

ITPD	33 - Discussion on what kind of integration strategy adopt
	34 - Discussion on Identifying individual steps of component integration
	35 - Integration Strategy
	36 - Individual Steps and Test Description
	37 - Brief revision to check what has been done
	38 - Discussion on drivers
	39 - Tools and Test Equipment
	40 - Driver description
	41 - Copy Word File into a LaTeXDocument
	42 - Last revision on ITPD
PPD	43 - Discussion on dispatching components into ILF, EIF, EI, EO, EQ
	44 - ILF and EIF FPs evaluation and description
	45 - EI, EO and EQ FPs evaluation and description
	46 - Brief revision to check what has been done
	47 - COCOMO II cost and effort estimation
	48 - Brief revision to check what has been done
	49 - Scheduling Plan
	50 - Resource Plan
	51 - Discussion on identifying what type of risks could affect the system
	52 - Risk Management
	53 - Copy Word File into a LaTeXDocument
	54 - Last revision on PPD

5 Risk management

This chapter will face the risk management related to the development of the PowerEnJoy system, in particular, we will try to identify the main risks that the development could come up against and, for each of them, well define some behaviours to apply before, to prevent the concrete risk, or after the actual hit to face it the best way. The first possible risk in which development might bump, is constituted by a change of the requirements and features to be implemented in the system, in progress of development. The first solution to solve this possibility is given by the construction of an architecture elastic, able to grant a large margin of changes and improvement (also in view of future updates); However, this should not be seen as the only solution to be taken as a continuous or sudden change in requirements can lead to problems of misunderstanding between teams and stakeholders, as well as within the team itself. For this reason, it is very important to prevent these possibilities maintaining a constant dialogue with stakeholders. Allowing them to verify the state of development of each functionality, thus also providing practical examples of immediate understanding in order to obtain the maximum possible agreement, in the event that changes were required, these may be implemented in the course of work. Another aspect related to the concept of collaboration and dialogue between stakeholders and team leads us to consider the problem of the response to the final user. In particular, we must take into account that nowadays the car sharing services are widespread, and that users prefer a service rather than another both for economic reasons and for simplicity in using the services user interface. In our case, this results in the development of a user friendly interface for the application, in order to make the best choices and user preferable, it is therefore advisable to carry out meetings with future users, randomly selected, and carefully study their feedback, focusing these meetings especially just before the release, with briefings or small interview asking what kind of environment the user is looking for. Earlier we mentioned the team and how certain factors can influence the development negatively. One of the problems that may be encountered is the lack of understanding. This may be due to multiple reasons, primarily due to the lack, or rare, presence of internal meetings: these in fact not only guarantee to maintain an updated flow of informations about the state of development through the developing teams, but also to promote constant motivation and teamwork. The misunderstanding may result from other factors, such as incohesive team, where the communication among its members is not proper, so a situation in which each member doesnt know how is the working progress, is very common. The choice of subdividing the work load through multiple teams (task subdivision) could be seen as a remedy in case of absence (illness or other reasons) of one of the members, so as not to slow down too much the development because they work on different tasks. Continuing on the team's considerations, we can say that the development environment consists on mature programs, which therefore should not show any problems or obstacles to development. Same thing can be done with regard to google APIs used, which at most may go and introduce / remove the feature; for this reason you need to stay informed about updates and then be able to implement them as soon as possible, both during and after the development. Although our team is equipped with the necessary knowledge for the development of this type of system, in order to avoid possible stalls or delays in the development of the

more complex features of the system, it has been carried out a careful analysis at the beginning of the process, so as to ensure the recruitment of specialized staff (for limited periods) to remedy at these eventualities. With regard to the maintenance company and the payment system, that are the outside companies with which we have made contracts for the management of particular fields, we must assume that these contracts may expire, and then the external companies may change. It is therefore necessary that our system is able to quickly interface with the new services, in order to minimize possible disruption of service: for this reason it is necessary to focus on the development of components as flexible as possible. Another risk in which the development and especially the release system may encounter is constituted by the hardware, that is necessary for our system. In order to avoid reliability problems (due to the nature of the service) and also the increasing cost due to the wrong choice of maintaining the system by our own (more expensive), we have decided to move the system onto a cloud architecture, that maintains lower costs and ensures a constant and efficient service. All these risks could slow down the achievement of one or more deadlines in the schedule, for this reason it is necessary to carry out an analysis of the same schedule in relation to risk management, so as to ensure safety margins. About this, we have counted few more hours in the Resource Planning to avoid that the slowing down of our work forces us to break the deadlines. We know that more working hours correspond to a more costly system for the customer, and taking care of that, the amount of surplus hours is the minimum necessary to allow the schedule to be more relaxed but however it does not too much burden the system cost. In that unlikely situation, we will release before the most important functionalities of our system and then well progressively release the remaining part with upgrades.

Used tools

- Github: for version control
- GoogleDoc: to write the document
- Draw.io: to create the diagrams
- LaTeX: to create the pdf
- Creately: for Resource Allocation diagram
- Gantt Project: to create Gantt Diagrams

Work hours

- Emanuele Ricciardelli: $\sim$ 30 hrs.
- Giorgio Tavecchia: $\sim$ 30 hrs.
- Francesco Vetró: $\sim$ 30 hrs.