Project Plan Document

POLITECNICO
MILANO 1863

Version 1.0

Luca Santini	808710
Riccardo Remigio	874939

Index

1. Introduction

1.1 Purpose and scope

The purpose of this document is to estimate the overall size and the costs of the PowerEnJoy project, and to find how best to organize the work of all the personnel.
To do this we will use the constructive cost model (COCOMO), so we need to estimate first the Function points, then we will can estimate the Source Lines of Code. After the size estimation we will compute the scale factors, the cost drivers and the effort multipliers to apply the formula to calculate the effort estimation.
So we will distribute the effort through the development process activities, and we will assign the different tasks to our development team.
In the last part we are going to do the risk plan, so we are going to identify the possible risks by recognizing what can go wrong during the development process, and we plan a way to avoid them as much as possible.

1.2 Definitions, Acronyms and Abbreviations

RASD: Requirements and Specifications Document

DD: Design Document

COCOMO: Constructive Cost Model

FP: Function Points

UFP: Unadjusted Function Points

ILF: Internal Logical File

EIF: External Interface File

EO: External Output

EI: External Input

EQ: External Inquiries

(K)SLOC: (Kilo) Source Lines Of Code

EM: Effort Multiplier

PM: Person-Month

1.3 Reference Documents

- COCOMO II Model Definition Manual
- Assignments AA 2016-2017.pdf
- Project Management Basics + Advanced.pdf

2. Project size, cost and effort estimation

This chapter is devoted to the estimation of the overall size, cost and effort required to develop the project.
To apply the constructive cost model, we must initially determine whether we are in the case of post-architecture or early design. We have to design from scratch the architecture for our project by exploring different architectural alternatives, so we can affirm that we are in early design case.
Now we will estimate the size through the UFP computation, then we will compute Scale Factors, Cost Drivers and Effort Multipliers. In the end we can compute the effort with the COCOMO II Formula, and we can estimate a schedule.

2.1 Size estimation: Function points

The function point cost estimation approach is based on the amount of functionality in a software project and a set of individual project factors.
The following tables are taken from COCOMO II Model Definition Manual and explains what FPs are and how the FPs will be estimated.

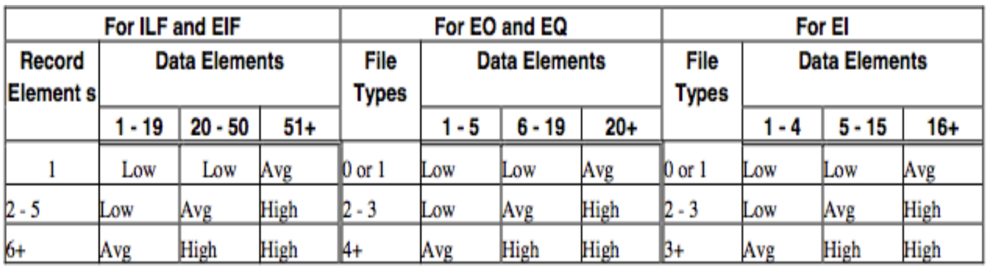
Table I: User Function Types

External Input (Inputs)	Count each unique user data or user control input type that (i) enters the external boundary of the software system being measured and (ii) adds or changes data in a logical internal file.
External Output (Outputs)	Count each unique user data or control output type that leaves the external boundary of the software system being measured.
Internal Logical File (Files)	Count each major logical group of user data or control information in the software system as a logical internal file type. Include each logical file (e.g., each logical group of data) that is generated, used, or maintained by the software system.
External Interface Files (Interfaces)	Files passed or shared between software systems should be counted as external interface file types within each system.
External Inquiry (Queries)	Count each unique input-output combination, where an input causes and generates an immediate output, as an external inquiry type.

Table II: Weight table

Function Type	Complexity-Weight		
	Low	Average	High
Internal Logical Files	7	10	15
External Interfaces Files	5	7	10
External Inputs	3	4	6
External Outputs	4	5	7
External Inquiries	3	4	6

Table III: FPs complexity



For ILF and EIF				For EO and EQ				For EI			
Record Element s	Data Elements			File Types	Data Elements			File Types	Data Elements		
	1 - 19	20 - 50	51+		1 - 5	6 - 19	20+		1 - 4	5 - 15	16+
1	Low	Low	Avg	0 or 1	Low	Low	Avg	0 or 1	Low	Low	Avg
2 - 5	Low	Avg	High	2 - 3	Low	Avg	High	2 - 3	Low	Avg	High
6+	Avg	High	High	4+	Avg	High	High	3+	Avg	High	High

2.1.1 Internal Logical Files (ILFs)

The PowerEnJoy project will require functionalities that include logical group of data that is generated and used from the software.
Therefore, we can identify these ILFs:

- User Data: is stored for each user registration, and will be used every time you perform a login and each time there will be a request from the user. User data that the system will have to keep in memory are many and will be used in different components. Therefore, we can set the weight as **average**.

- Technician Data: some of the technicians' data are stored, especially to recognize them when they log into the system, these data are few and we can set the weight as **low.**

- Payment Information Data: the payment information are simple data, consisting only of a few codes that allow us to retrieve the customer's account which will send to the external payment system. We can set its weight as **low**.

- Reservation Data: the system must keep track of the reservations that will be deleted as soon as terminate, also in this case the data to be stored will be few and we consider it as a **low** weight.

- **Ride Data:** the system must keep track of the rides, there aren't many attributes to keep stored, so we consider also this as a **low** weight internal logical file.

- **Vehicle Data:** the system has to keep stored all the data of all the vehicles of PowerEnJoy. These data are more complex because you have to track of the status and the location of each vehicle, besides to the license plate number and other identification data. So we consider this an **average** weight.

- **Safe Area Data:** this information consists of only one position and a radius representing the safe area. So it has a **low** weight.

- **Charging Station Data:** also the charging station data are simple, because for each charging station are stored only a position and a number of vehicles that it can contain. So we set the weight as **low**.

ILFs	Complexity	FPs
User Data	Average	10
Technician Data	Low	7
Payment Information Data	Low	7
Reservation Data	Low	7
Ride Data	Low	7
Vehicle Data	Average	10
Safe Area Data	Low	7
Charging Station Data	Low	7
TOTAL		62

2.1.2 External Interface Files (EIFs)

We have only two external data source that are Vehicle System Communication and the Map Service.

- **Payment Service:** for paying the services, PowerEnJoy clients have to communicate their payment method that will be stored in the PowerEnJoy database at the time of registration. The payments data are used by PowerEnJoy service, but are generated and maintained by the External Payment Service. This Function point can be considered as a **low** weight function point, because it treats simple and poorly structured information.

- **Map Service:** there is an interaction between our system and a mapping service system. We need to get the coordinates of a given address, and to get an address from the coordinates. The mapping service is also used to retrieve the graphical representation of the city map. The quantity of data exchanged is significantly high, so we can consider the complexity of this function points as **average**.

EIFs	Complexity	FPs
Payment Service	Low	5
Map Service	Average	7
TOTAL		**12**

2.1.3 External Inputs (EIs)

We have identified six External Inputs operations that elaborate data from the external environment.
These are:

- **Login operation:** consists only in reading from database the data inserted by the user in his browser page, it is a simple procedure, so we set his weight to **low**.

- **Registration operation:** it is a simple operation, that consists in an insertion of data by the user that are stored in the database. This is a simple operation so; the weight is set to **low**.

- **Unlock/lock car operation:** in this procedure a signal is sent to the system, that recognize the source and, if the source has the authorization, the system proceeds by sending the unlock/lock signal to the car. This procedure involves only few component, is considered a **low** weight operation.

- **Change vehicle state operation:** this operation can be performed only by the technicians; it consists in sending to the system a change state request for a vehicle. The system recognizes if the technician has the needed authorization, then changes the state of the vehicle saved in the database. This is a simple procedure, so his weight is set to **low**.

- **Reserve vehicle operation:** this operation is similar to precedent, it involves few data elements and therefore it is set to **low**.

- **Receive information from the vehicle:** this procedure is performed when a vehicle sends its updated data. First, the system edits the stored data of the vehicle with the new data, then it calculates the position by using the map service and the other factors that determine the payment charges and the state of the vehicle. This operation involves some ILFs, so we set it to **average** weight.

EIs	Complexity	FPs
Login operation	Low	3
Registration operation	Low	3
Unlock/lock car operation	Low	3
Change vehicle state operation	Low	3
Reserve vehicle operation	Low	3
Receive information from the vehicle	Average	4
TOTAL		**19**

2.1.4 External Inquiries (EQs)

We have estimated three External Inquiries operations that involve input and output without significant elaboration of data from logic files.

- **Visualization of Personal Information:** a user can visualize his personal information from the specific web page. The system retrieves the information of the user by reading it from the database. This is a simple operation, so we can classify it as a **low** weight procedure.

- **Location of the Vehicles Position:** this procedure can be performed by the users and by the technicians. When the system receives the request, it reads from the database the positions of the car and then it has to invoke the map service to retrieve the map with the position of all the cars. The system also must verify the permissions of who made the request and hide the vehicles that must not to be visualized. This procedure is quite complex, so we set its weight to **high.**

- **Visualization of the car’s information:** this procedure is invoked only by the technician. The system retrieves the information of the requested vehicle, by reading them in the database. The system must also verify the permissions of the user. This operation is set to **average.**

EQs	Complexity	FPs
Visualization of Personal Information	Low	3
Location of the Vehicles Position	High	6
Visualization of the car’s information	Average	4
TOTAL		**13**

2.1.5 External Outputs (EOs)

We have estimated two External Outputs operations that generates data for the external environment.

- **Sending Password for the Registration:** after the registration of the user, the system generates a password that sends to the user. This operation is very simple; therefore, the weight is set to **low.**

- **Charging for the User:** this is the operation that the system does to calculate the charges. It is a quite **complex** operation because the system has to retrieve all the information about the vehicle and the ride, and calculate the charges by using the algorithm. This operation is performed very often for ensuring the user that he will can visualize constantly the actual charges.

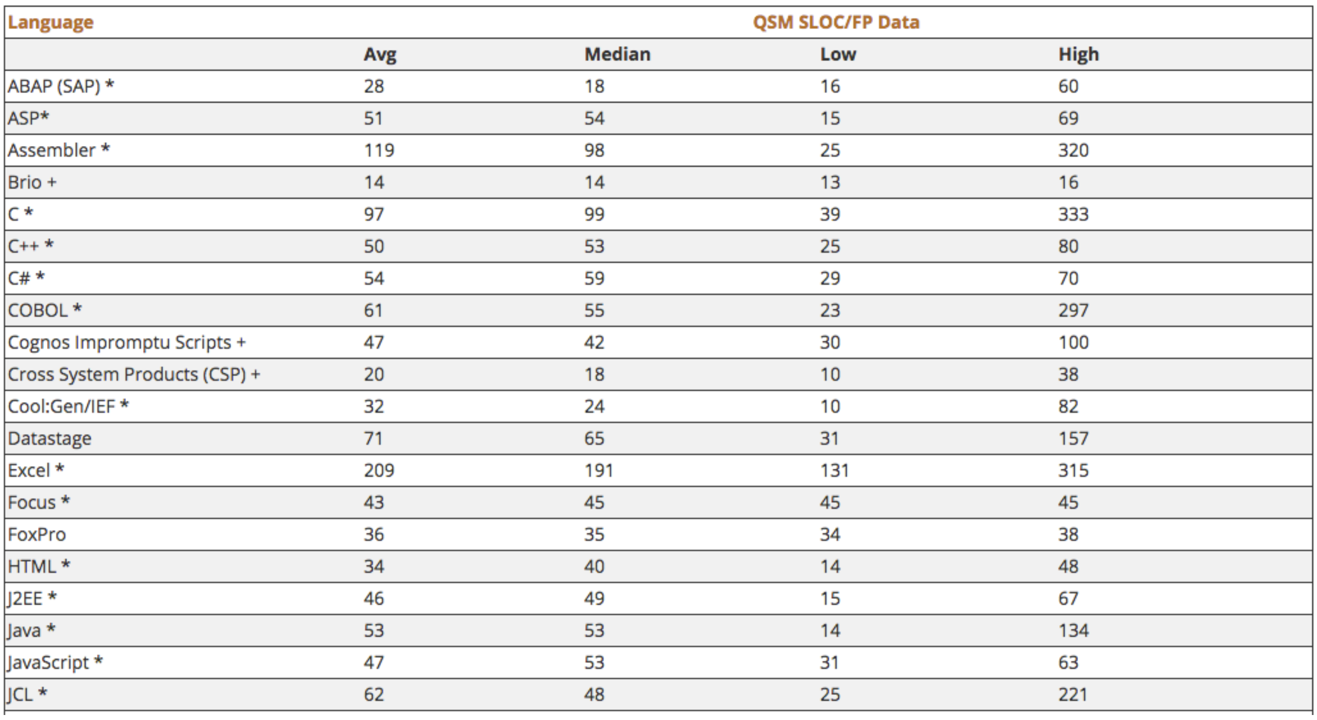
EQs	Complexity	FPs
Sending Password for the Registration	Low	4
Charging for the User	High	7
TOTAL		**11**

2.1.6 Overall Estimation

The following table summarizes the previous calculation of all FPs, and calculates the overall unadjusted function points (UFP).

Function Points Types	Value
Internal Logical Files (ILF)	62
External Interface Files (EIF)	12
External Inputs (EI)	19
External Inquiries (EQ)	13
External Outputs (EO)	11
Unadjusted Function Points (UDP)	117

We can use the UFP to estimate the lines of code by multiplying it for a factor representing the programming language. According with quantitative software management function point language table



Language	QSM SLOC/FP Data			
	Avg	Median	Low	High
ABAP (SAP) *	28	18	16	60
ASP*	51	54	15	69
Assembler *	119	98	25	320
Brio +	14	14	13	16
C *	97	99	39	333
C++ *	50	53	25	80
C# *	54	59	29	70
COBOL *	61	55	23	297
Cognos Impromptu Scripts +	47	42	30	100
Cross System Products (CSP) +	20	18	10	38
Cool:Gen/IEF *	32	24	10	82
Datastage	71	65	31	157
Excel *	209	191	131	315
Focus *	43	45	45	45
FoxPro	36	35	34	38
HTML *	34	40	14	48
J2EE *	46	49	15	67
Java *	53	53	14	134
JavaScript *	47	53	31	63
JCL *	62	48	25	221

We will use Java Enterprise Edition (J2EE) values, we will use the average value to estimate the lower bound, and the high value to estimate the upper bound. Here is the calculation of SLOC

Lower Bound:

$$SLOC = UDP * \ AVC_{Average} = 117 * 46$$
$$= 5382 \ lines \ of \ code$$

Upper Bound:

$$SLOC = UDP * \ AVC_{High} = 117 * 67$$

$$= 7839 \ lines \ of \ code$$

2.2 COCOMO II: Cost and Effort Estimation

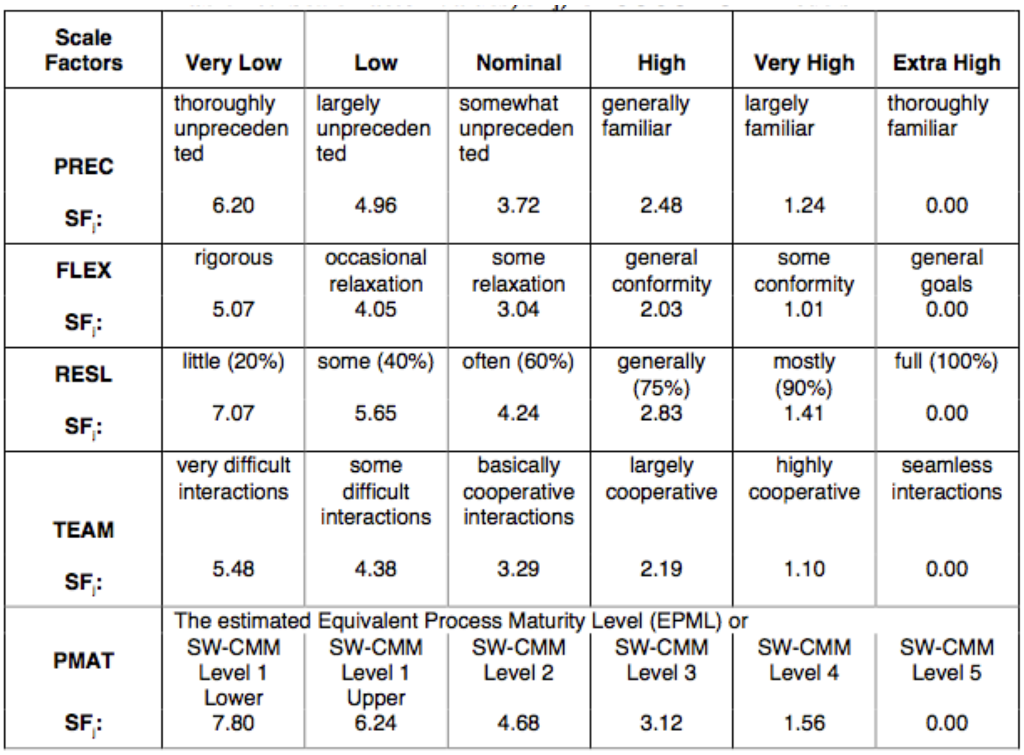
To estimate the effort, we are going to use the main COCOMO II formula, the effort equation:

$$PM = A * SIZE^{E} * \prod_{i=1}^{n} EM_i$$

where:
PM is Person-Months
A= 2,94 PM/KSLOC
SIZE is the estimated size of the project in KSLOC
E is an aggregation of five Scale Factors
EM is for Effort Multiplier; we can derive them from Cost Drivers.
Therefore, we are going to estimate Scale Factors and Cost Drivers.

2.2.1 Scale Factors

In order to evaluate the scale factors we are going to use the following COCOMO II official table



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

Precedentedness (PREC): If a product is similar to several previously developed projects, then the precedentedness is high. Because in our case we don't have any type of precedent experience in a project similar to this, we set PREC to **very low**.

Development Flexibility (FLEX): it reflects the degree of flexibility in the development process with respect to the external specification and requirements. In our case we have strictly functional requirements, but we are free to choose the design and the architecture, and we don't have specific non functional requirements. Accordingly, FLEX is set to **nominal**.

Architecture/Risk Resolution (RESL): reflects the level of awareness and reactiveness with respect to risks. In the chapter 4 we will discuss the risk management plan, so we can set the scale factor to **high**.

Team Cohesion (TEAM): it's an indicator of how well the team members know each other and work together in a cooperative way. In our case we are very cooperative and cohesively, as we have already collaborated on other projects. We set this scale factor to **very high.**

Process Maturity (PMAT): Refers to a well known method for assessing the maturity of a software organization, CMM, now evolved into CMMI. Since we are at maturity level 3 of CMMI we set this scale factor to **high** as the table suggests.

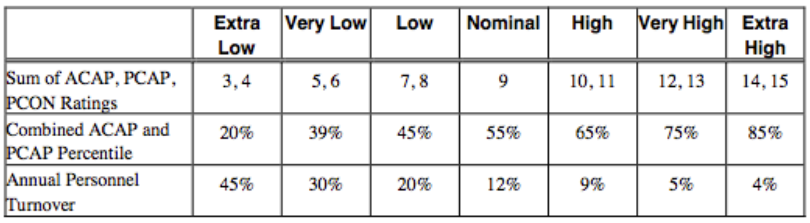
Scale Factor	Factor	Value
Precedentedness (PREC)	Very Low	6,20
Development Flexibility (FLEX)	Nominal	3,04
Architecture Risk Resolution (RESL)	High	2,83
Team Cohesion (TEAM)	Very High	1,10
Process Maturity (PMAT)	High	3,12
$\boldsymbol{E = 0,91 + 0,01 * \sum_{i=1}^{5} SF_i}$		1,0729

2.2.2 Cost Drivers

Personnel Capability: the early design PERS cost driver combines the post architecture cost drivers analyst capability (ACAP), programmer capability (PCAP), and personnel continuity (PCON).
ACAP is set to **nominal** because we don't have a high ability in design and analysis, but our level of communication and cooperation is very high.
PCAP is also set to **nominal** because our programmers ability is not very high, but as mentioned above our level of communication and cooperation compensates the lack of ability.
PCON is set to **very high** because there isn't any type of turnover.

Cost Drivers	Complexity	Value
ACAP	Nominal	3
PCAP	Nominal	3
PCON	Very High	5
	TOTAL	11

Table II-10: PERS Rating Levels



	Extra Low	Very Low	Low	Nominal	High	Very High	Extra High
Sum of ACAP, PCAP, PCON Ratings	3, 4	5, 6	7, 8	9	10, 11	12, 13	14, 15
Combined ACAP and PCAP Percentile	20%	39%	45%	55%	65%	75%	85%
Annual Personnel Turnover	45%	30%	20%	12%	9%	5%	4%

Therefore, **PERS** is set to **high** with a value of **0,83**.

Product Reliability and Complexity: the early design RCPX cost driver combines the post architecture cost drivers required software reliability (RELY), database size (DATA), product complexity (CPLX), and documentation match to life-cycle needs (DOCU).
RELY: if there is a failure in the system, there is only a moderate financial loss, so we can assume this cost driver set as **nominal**.
DATA: we have estimated the size of the database around 150 MB, so calculating the D/P ratio the result is greater than 1000 therefore we can set the DATA to **very high.**

	Control Operations	Computational Operations	Device-dependent Operations	Data Management Operations	User Interface Managemen t Operations
Very Low	Straight-line code with a few non-nested structured programming operators: DOs, CASEs, IFTHENELSEs. 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.
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, roundoff 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.

CPLX: in our case we set control operations at nominal, computational operations at very low, device-dependent operations at low, data management operations at low, user interface management at low.
Therefore, we estimated the overall cost driver CPLX at **low.**
DOCU: we set this driver at **high** because we believe that a good documentation can avoid extra costs during the maintenance portion of the life-cycle, and so we provide a detailed documentation.

Cost Drivers	Complexity	Value
RELY	Nominal	3
DATA	Very High	5
CPLX	Low	2
DOCU	High	4
	TOTAL	**14**

Therefore, **RCPX** is set to **high** with a value of **1,33**.

Required Reusability: this cost driver accounts for the additional effort needed to construct components for reuse on current or future projects.
We set this driver at **nominal** because we limited the reusability across this project.

RUSE is set to **nominal** with a value of **1,00**.

Platform Difficulty: this early design cost driver combines the three post-architecture cost drivers execution time constraint (TIME), main storage constraint (STOR) and platform volatility (PVOL).
TIME: we set this driver to **high** because the system will be used very frequently from the users to reserve or use vehicles.
STOR: we set this driver to **high** because, as said before, we have a high number of users and so also the respective information in the storage are many.
PVOL: PowerEnJoy platform doesn't need frequent major updates, but if there is a necessity, minor update will be released. Therefore, we set PVOL to **low.**

Cost Drivers	Complexity	Value
TIME	High	4
STOR	High	4
PVOL	Low	2
TOTAL		**10**

Therefore, **PDIF** is set to **high** with a value of **1,29**.

Personnel Experience: this early design cost driver combines the three post-architecture cost drivers application experience (APEX), language and tool experience (LTEX), and platform experience (PLEX).
We don't have any experience in JEE applications, we have never used a DBMS, and we have a very little experience in java and in networking. So we set APEX, LTEX and PLEX to **very low**.

Cost Drivers	Complexity	Value
APEX	Very Low	1
LTEX	Very Low	1
PLEX	Very Low	1
TOTAL		**3**

Therefore, **PREX** is set to **extra low** with a value of **1,59**.

Facilities: this early design cost driver combines two post-architecture cost drivers: use of software tools (TOOL) and multisite development (SITE).
TOOL: we set this driver to **nominal** because we have used tools during the development of the project along the life-cycle.
SITE: we set this driver to **very high** because we can work together most of the time due to the nearness of our houses.

Cost Drivers	Complexity	Value
TOOL	Nominal	3
SITE	Very High	5
TOTAL		**8**

Therefore, **FCIL** is set to **high** with a value of **0,87**.

Required Development Schedule: this rating measures the schedule constraint imposed on the project team developing the software. Therefore, we set it to **nominal**, because we have not flexible deadlines and we have to distribute the time equally.

Therefore, **SCED** is set to **nominal** with a value of **1**.

Cost Driver	Level	Value
PERS	High	0,83
RCPX	High	1,33
RUSE	Nominal	1
PDIF	High	1,29
PREX	Extra Low	1,59
FCIL	High	0,87
SCED	Nominal	1
Total (EAF)		1,96862

2.2.3 Effort Equation

Now that we have all the data, we can proceed to calculate the effort:

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

Summarizing we have:

$A = 2{,}94$

$EAF = 1{,}96862$

$E = 1{,}0729$

$KSLOC_{Average} = 5{,}382$

$KSLOC_{High} = 7{,}839$

$$
\begin{aligned}
Effort_{Average} &= A * EAF * \ KSLOC_{Average}^{E} \\
&= 2{,}94 * 1{,}96862 * \ 5{,}382^{1{,}0729} \\
&= 35{,}2159\ PM \sim 35\ PM
\end{aligned}
$$

$$
\begin{aligned}
Effort_{High} &= A * EAF * \ KSLOC_{High}^{E} \\
&= 2{,}94 * 1{,}96862 * \ 7{,}839^{1{,}0729} \\
&= 52{,}7183\ PM \sim 53\ PM
\end{aligned}
$$

2.2.4 Schedule Estimation

For the final schedule we will use this following formula

$$Duration = 3{,}67 * Effort^{F}$$

where:

$F = 0{,}28 + 0{,}2 * (E - B) = 0{,}28 + 0{,}2 * (1{,}0729 * 0{,}91)$
$= 0{,}28 + 0{,}2 * 0{,}9763 = 0{,}4753$

$B = 0{,}91\ for\ COCOMO\ II$

Using as lower bound the average KSLOC and as upper bound the high KSLOC previously calculated:

$$
\begin{aligned}
Duration_{Average} &= 3{,}67 * 35{,}2159^{0{,}4753} \\
&= 19{,}945\ months \sim 20\ months
\end{aligned}
$$

$$
\begin{aligned}
Duration_{High} &= 3{,}67 * 52{,}7183^{0{,}4753} \\
&= 24{,}161\ months \sim 24\ months
\end{aligned}
$$

3. Schedule

3.1 Tasks and Schedule

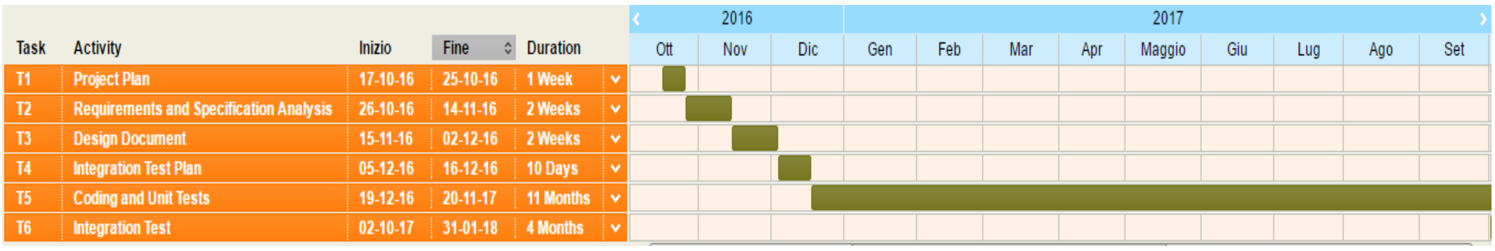
In this chapter we describe which are our tasks that have to be completed to reach the end of the project. Furthermore, we give a tentative schedule to distribute our tasks along the time, following the effort estimation given by the COCOMO II formulation in the chapters before.

Our project can be divided into 6 tasks that are:

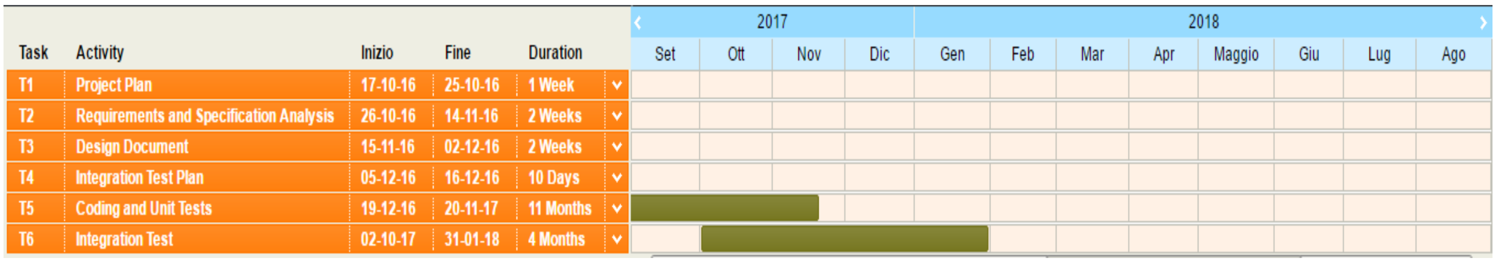
- T1 Project Plan: consists in the writing of a document in which there is an estimation of the effort to complete the project, a tentative schedule of the various tasks and possibly documentation about risk management.
- T2 RASD: consists in the writing of a document in which are expressed all the functional and non-functional requirements and the goals that the software must follow.
- T3 Design Document: consists in the writing of a document in which are explained the architectural and design choices of our software.
- T4 ITPD: consists in the writing of a document that describes the procedure to follow for the integration test phase.
- T5 Coding and Unit Tests: in this task the developers will implement the software and at the same time they will tests (only unit test) the written code.
- T6 Integration Test: in this task the software will be tested integrating all the components following the ITPD.

We haven't the intention to deploy a version of the software before the end of the project, like a preview, because it hasn't a lot of functionality, so a reduced version of the application would be useless. This could happen only if the development of the software will delay over the deadline.
However, during the various tasks, the documents will be released to the customer to show how the work is proceeding.

Here we show the tentative schedule of our tasks through a Gantt chart.



Task	Activity	Inizio	Fine	Duration
T1	Project Plan	17-10-16	25-10-16	1 Week
T2	Requirements and Specification Analysis	26-10-16	14-11-16	2 Weeks
T3	Design Document	15-11-16	02-12-16	2 Weeks
T4	Integration Test Plan	05-12-16	16-12-16	10 Days
T5	Coding and Unit Tests	19-12-16	20-11-17	11 Months
T6	Integration Test	02-10-17	31-01-18	4 Months



Task	Activity	Inizio	Fine	Duration
T1	Project Plan	17-10-16	25-10-16	1 Week
T2	Requirements and Specification Analysis	26-10-16	14-11-16	2 Weeks
T3	Design Document	15-11-16	02-12-16	2 Weeks
T4	Integration Test Plan	05-12-16	16-12-16	10 Days
T5	Coding and Unit Tests	19-12-16	20-11-17	11 Months
T6	Integration Test	02-10-17	31-01-18	4 Months

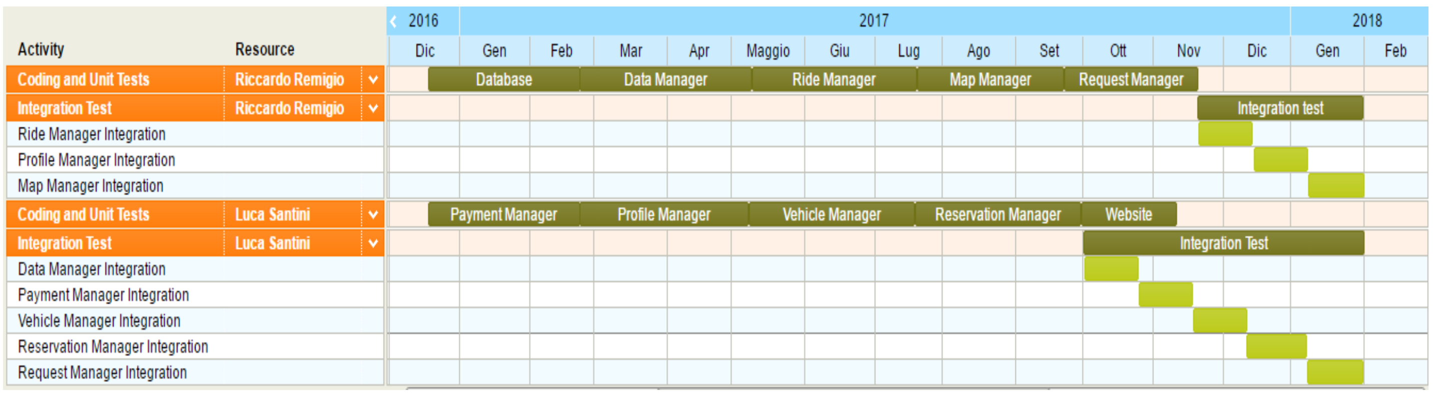
We can see in the schedule that almost all the activities have dependencies with the previous ones, except for the integration test phase that can be start during the last part of the development phase when the greater part of the components is completed.

3.2 Resource Allocation

In this chapter is described how the resources are allocated in a way to complete all the tasks. We don't need any special physical resource shared by the members of the group, so the allocation will regard only the human resources, therefore the members of the team.

For the documents, the team members can't work in parallel from the beginning because there will be an initial phase of group consultation to decide the contents of the document. Then they can elaborate and write, each on his own, the document so to speed up the task.
While for the phase of coding, unit tests and integration test, the team members divide the work trying to optimize the time and work in parallel. Based on the component view of the design document we decided an order of implementation of the components in a way to reduce to the minimum the dependencies.

The graph below shows a tentative schedule of resource allocation between the members of the team.
We have to say that the time allocated for each component is approximate, in fact the main goal of this diagram is to show a possible division of the work keeping in mind the constraints between the components during the development and integration phases.



4. Risk Management

In this chapter we describe the risks that can afflict our project along its life cycle, trying to predict them and searching for solutions that can reduce the consequences to minimum.
The risks can be of various nature: technical, business and project risks.
We will focus on the project and technical risks that in our case can give some troubles and only the risks with a moderate or high probability of happening.

Product size risks: the estimate of the size of the project can be wrong and can lead to a delay over the deadline. This risk can be prevented using a pessimistic estimate of effort in a way to give time for eventual problems. Obviously the estimate must not be too large otherwise the customer will not get us the work. If the delay arises anyway during the project, we can buy COTS so to speed up the development process and if we will go over the deadline we can deploy a preview of the software with some functionalities.

Bad external components: the external components used in our system may not work correctly. In this case we may keep possible alternatives so to not lose time in finding a solution when the problem will arise. Regarding the external component of the vehicles, it can't be substituted by others, so it has to be corrected in some way.

Staff illness: because of the presence of only two members in the team, the substitution of a person could be really complex and the unique solution may be to call an external person to proceed with the work.

Unrealistic schedule: this risk can arise for many factors. One can be the wrong estimate as said before, while another factor can be the low experience of the team members. This risk has a high probability of happening because the team members don't have worked with a lot of projects before this. Also in this case a solution can be the purchase of COTS to speed up the development phase. If we will go over the deadline, as said before, we can deploy a preview of the software to the customer with some functionalities.

5. Appendix

5.1 Used Tools

- GitHub
- Microsoft Word
- Tomsplanner

5.2 Hours of Work

These are approximatively the time we spent to write this document

Luca Santini: 20 hours

Riccardo Remigio: 20 hours