Figure 1: Politecnico di Milano

Project Plan Document
Version 1.0

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

January 21, 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

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*

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.

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

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. For each of these factors, we will now provide a

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: Scale Factors

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:

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