**P**roject **P**lan **D**ocument



Version 1.0

Luca Santini 808710

Riccardo Remigio 874939

Index

[1. Introduction 3](#_Toc472619862)

[1.1 Purpose and scope 3](#_Toc472619863)

[1.2 Definitions, Acronyms and Abbreviations 3](#_Toc472619864)

[1.3 Reference Documents 4](#_Toc472619865)

[2. Project size, cost and effort estimation 4](#_Toc472619866)

[2.1 Size estimation: Function points 5](#_Toc472619867)

[2.1.1 Internal Logical Files (ILFs) 6](#_Toc472619868)

[2.1.2 External Interface Files (EIFs) 8](#_Toc472619869)

[2.1.3 External Inputs (EIs) 8](#_Toc472619870)

[2.1.4 External Inquiries (EQs) 9](#_Toc472619871)

[2.1.5 External Outputs (EOs) 9](#_Toc472619872)

[2.1.6 Overall Estimation 9](#_Toc472619873)

[2.2 COCOMO II: Cost and Effort Estimation 9](#_Toc472619874)

[2.2.1 Scale Factors 9](#_Toc472619875)

[2.2.2 Cost Drivers 9](#_Toc472619876)

[2.2.3 Effort Equation 9](#_Toc472619877)

[2.2.4 Schedule Estimation 9](#_Toc472619878)

[3. Schedule 9](#_Toc472619879)

[3.1 Tasks and Schedule 9](#_Toc472619880)

[3.2 Resource Allocation 9](#_Toc472619881)

[4. Risk Management 9](#_Toc472619882)

[5. Appendix 9](#_Toc472619883)

[5.1 Used Tools 9](#_Toc472619884)

[5.2 Hours of Work 9](#_Toc472619885)

# Introduction

## 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.

## 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

## Reference Documents

COCOMO II Model Definition Manual

# 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.

## 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

**Table II:** Weight table

**Table III:** FPs complexity

### 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** |

### External Interface Files (EIFs)

We have on 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 and 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** |

### 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** |

### 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** |

### 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** |

### 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



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 x AVCAVERAGE = 117 x 46 = 5382 lines of code

Upper Bound:

SLOC = UDP x AVCHIGH = 117 x 67 = 7839 lines of code

## COCOMO II: Cost and Effort Estimation

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

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.

### Scale Factors

### 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.

**3+3+5=11 -> high -> 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.**





**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.

3+5+2+4=14 -> high -> 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.

Nominal->1

**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.**

**4+4+2=10 ->** **high -> 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**.

1+1+1=3 -> extra low -> 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.

3+5=8 -> high -> 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.

Nominal->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 |  | 1,96862 |

### Effort Equation

### Schedule Estimation

# Schedule

## Tasks and Schedule

## Resource Allocation

# Risk Management

# Appendix

## Used Tools

## 5.2 Hours of Work