* 1. **Purpose and Scope**

The purpose of this document is to explain the Project Plan devised for the

system to be. All the people involved in the project could be considered as

possible readers of the document, but the document itself is more of a guide

for the Project Manager and the Management in general. The Project Plan

consists in tables, Gantt diagrams, charts and natural language descriptions of

the planning, scheduling and management of PowerEnJoy development.

* 1. **List of Definitions and Abbreviations**
  2. **List of Reference Documents**

**2.Function Points**

The Functional Point approach is a technique that allows to evaluate the effort needed for the design and implementation of a project. We have used this technique to evaluate the application dimension basing on the functionalities of the application itself. The functionalities list has been obtained from the RASD document and for each one of them the realization complexity has been evaluated. The functionalities has been groped in:

\_ Internal Logic File: it represents a set of homogeneous data handled by the system

\_ External Interface File: it represents a set of homogeneous data used by the application but handled by external application

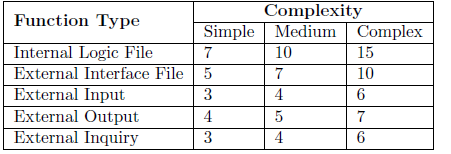
\_ External Input: elementary operation that allows input of data in the system

\_ External Output: elementary operation that creates a bit stream towards the outside of the application

\_ External Inquiry: elementary operation that involves input and output operations

The following table outline the number of Functional Point based on func-

tionality and relative complexity:



**2.2 Functional Point estimation**

**2.2.1 Internal Logic Files**

Ride (medium)

User (medium)

Problem Manager / Customer Service (simple)

Car (medium)

Payment (complex => compare position between two cars to obtain a discount )

Payment method (simple, Oggetto di un oggetto)

Park (simple)

Service Station (simple)

GPS Data (simple)

**2.2.2 External Logic Files**

Google Map (simple)

Pay Pal (simple)

**2.2.3 External Input**

Car Interaction (medium)

Login (simple)

Logout (simple)

Registration (simple)

Handle Personal Profile (simple)

Car Reservation (complex)

Problem (medium)

Rent (simple)

End of Rent (complex)

**2.2.4 External Output**

User Notification (simple)

Confirmation Email (simple)

Maintenance notification (simple)

Reservation (simple)

**2.2.5 External Inquiry**

Manage Car Status (simple)

Manager User Status (simple) (blocked without a valid payment)

**2.2.6 Summary**

149 total (ricontrollare)

**3 Cocomo**

This chapter describes the estimation achieved through COCOMO II: a com-

plex, non linear model that takes in account the characteristics of the product,

people and processes.

In order to generate the **Constructive Cost Model** we decided to use an online [COCOMO II calculator,](http://csse.usc.edu/tools/COCOMOII.php) using the **FP Sizing Method**.

We also used[COCOMO II - Model Definition Manual](http://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_modelman2000.0.pdf)to make better choices of the parameters we had to insert into the model.

### Software Size

**–Unadjusted Function Points:** The value *FP* has been taken as parameter, so **149** is the value chosen for this field.

**–Language:** The language of choice is **Java**, and not only Java EE, because the software is possibly a combination of different flavours of Java (Java SE, Java EE and Java for Android).

### Software Scale Drivers

**–Precedentedness:** Since we had a previous experience using *Java SE* for medium-size projects, but we have never used *Java EE* for developing such a big application we decided to set this parameter to **Nominal**.

**–Development Flexibility:** Given that we had not strict specifications so we set this parameter to **High**.

**–Architecture/Risk Resolution:** Since we have designed several documents before the actual development, including this *PPD*, the development of the system to be has little chances of failing, so we choose a **Nominal** value.

**–Team Cohesion:** After a few days dedicated to create a efficient and fast workspace, the cohesion reached a **Very High** level.

**–Process Maturity:** We understood, support and follow the process so we choose a **High** level for this parameter.

### Software Cost Drivers

**–Product**

* + - * **Required Software Reliability:** Given that a failure in the software system could lead to moderate problems we choose **Nominal** level.
      * **Data Base Size:** Since we have a distributed application, the

focus is on the lines of code instead of being on the size of the testing Database; so we choose a **Low** level parameter.

* + - * **Product Complexity:** We made an average of the various complexity areas and we choose a **High** level parameter.
      * **Developed for Reusability:** We decided to develop reusable

system components, so we came up with an **High** level parameter.

* + - * **Documentation Match to Lifecycle Needs:** The standard level of documentation is required, so the chosen level is **Nominal**.

### –Personnel

* + - * **Analyst Capability:** The personnel demonstrated a **Nominal**

level of analysis ability.

* + - * **Programmer Capability:** The personnel demonstrated efficiency working together as a team, so we chose **High** level for this parameter.
      * **Personnel Continuity:** The project will be developed always

by the initial programmers, so the *project’s annual personnel turnover* is very low (7 % for year). For this reason we have chosen **High** for this parameter.

* + - * **Application Experience:** Since the last time the development

team has worked on a so complicated project was a year ago, we

have chosen **Nominal** for this parameter.

* + - * **Platform Experience:** The same as *Application Experience* ; we have chosen **Nominal** level for this parameter too.
      * **Language and Toolset Experience:** The team is quite famil-

iar with the development, analysis and design representation, so we choose **High** level for this parameter.

### –Platform

* **Time Constraint:** We have no relevant time constraints, so we choose **Nominal** level for this parameter.
* **Storage Constraint:** We have no relevant storage constraint, so we choose **Nominal** level for this parameter.
* **Platform Volatility:** Our hardware and software platforms will not change often, so we will have no volatility and therefore we

choose a **Low** level for this constraint.

### –Project

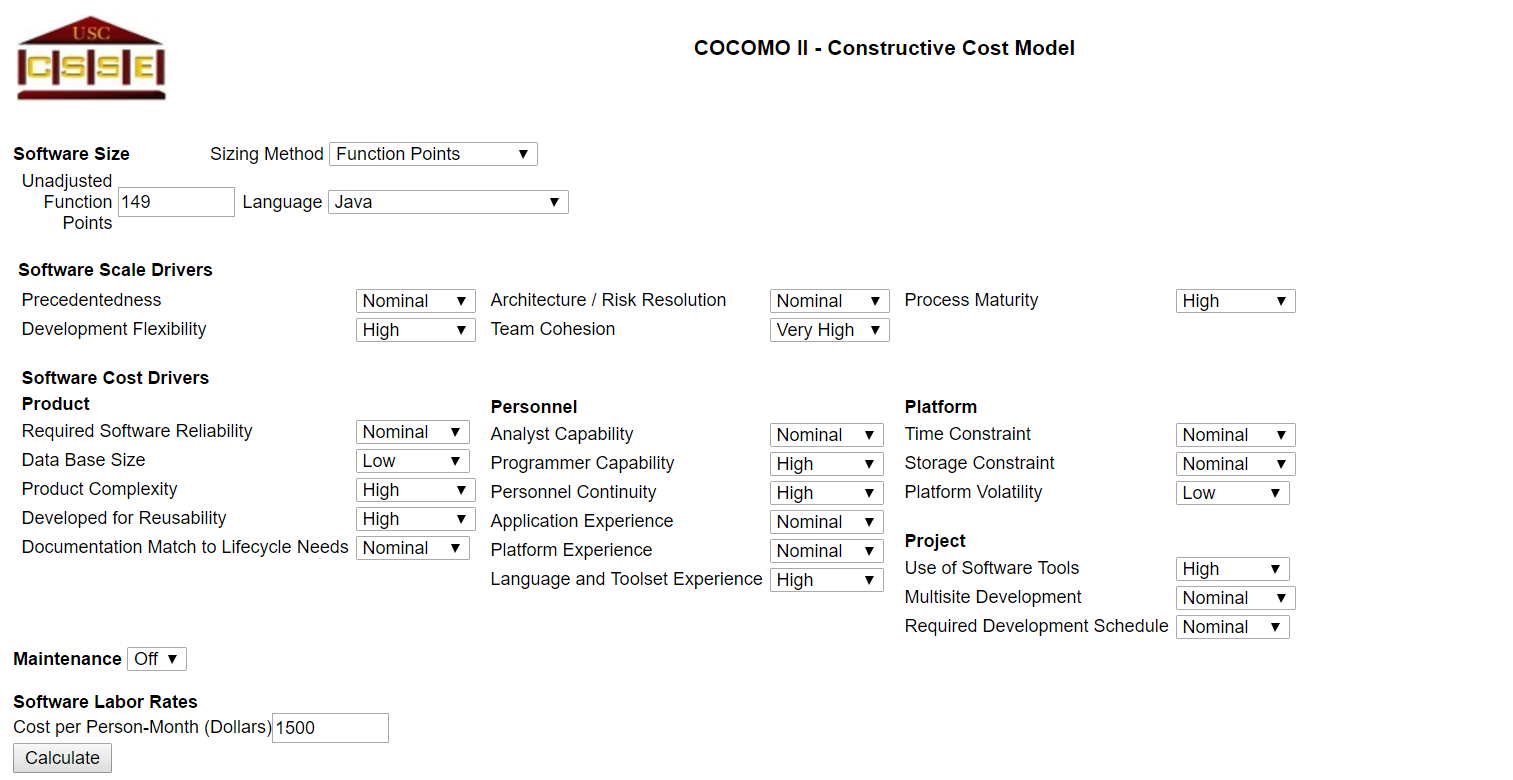
* **Use of Software Tools:** The team is provided of a set of strong and mature life-cycle tools, moderately integrated one into each other. So we choose an **High** level for this parameter.
* **Multisite Development:** The team is in average fully collo- cated, so the chosen level is **Nominal**.
* **Required Development Schedule:** The project is not sub- jected on a particular constraint oppression, so we have chosen

**Nominal** for this parameter.

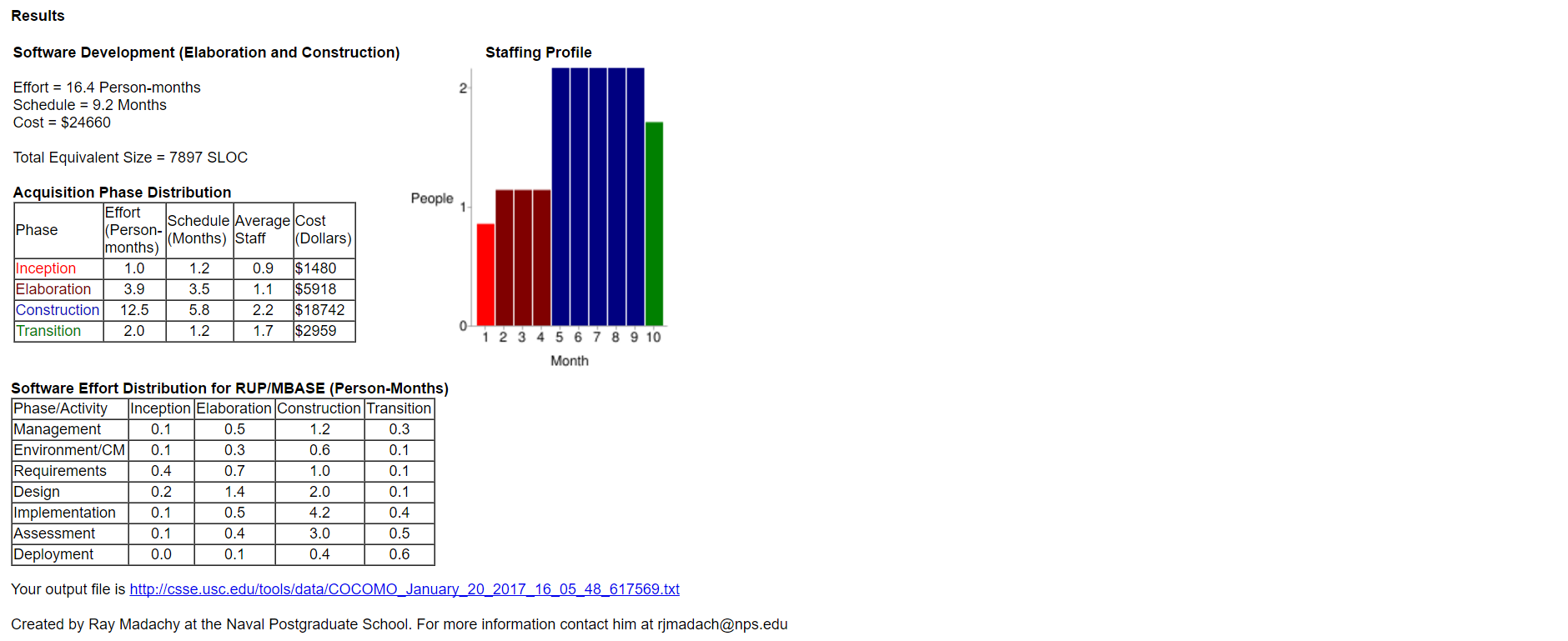
* + - **Maintenance** This value is set to **Off**.

### Software Labour Rates

**–Cost per Person-Month (Dollars):** We have chosen the average value of **1500$/month** for this parameter.

****

And here below we can see the result:



**4 Task and Schedule**

**4.1 Task**

Several tasks has been identified in our project. In the following table we can find three categories: one that labels the tasks, one for the description and the last one for the completion state to each task.

|  |  |  |
| --- | --- | --- |
| **Task** | **Description** | **Completed?** |
| T1a | RASD - Writing | Yes |
| T1b | RASD - Presentation | Yes |
| T2a | DD - Writing | Yes |
| T2b | DD - Presentation | Yes |
| T3a | ITPD - Writing | Yes |
| T4a | PPD - Writing | Yes |
| T4b | Final Presentation | No |
| T5 | Implementation | No |
| T6 | Unit Testing | No |
| T7 | Integration Testing | No |
| T8 | System Testing | No |
| T9 | User Acceptance - Alpha Testing | No |
| T10 | User Acceptance - Beta Testing | No |
| T11 | Release To Market | No |

**4.2 Schedule**

Below there are represented a table and a diagram that show how the time has been divided between the various phase of the software life cycle ; the result is then more clear in Gantt chart.

The table identifies:

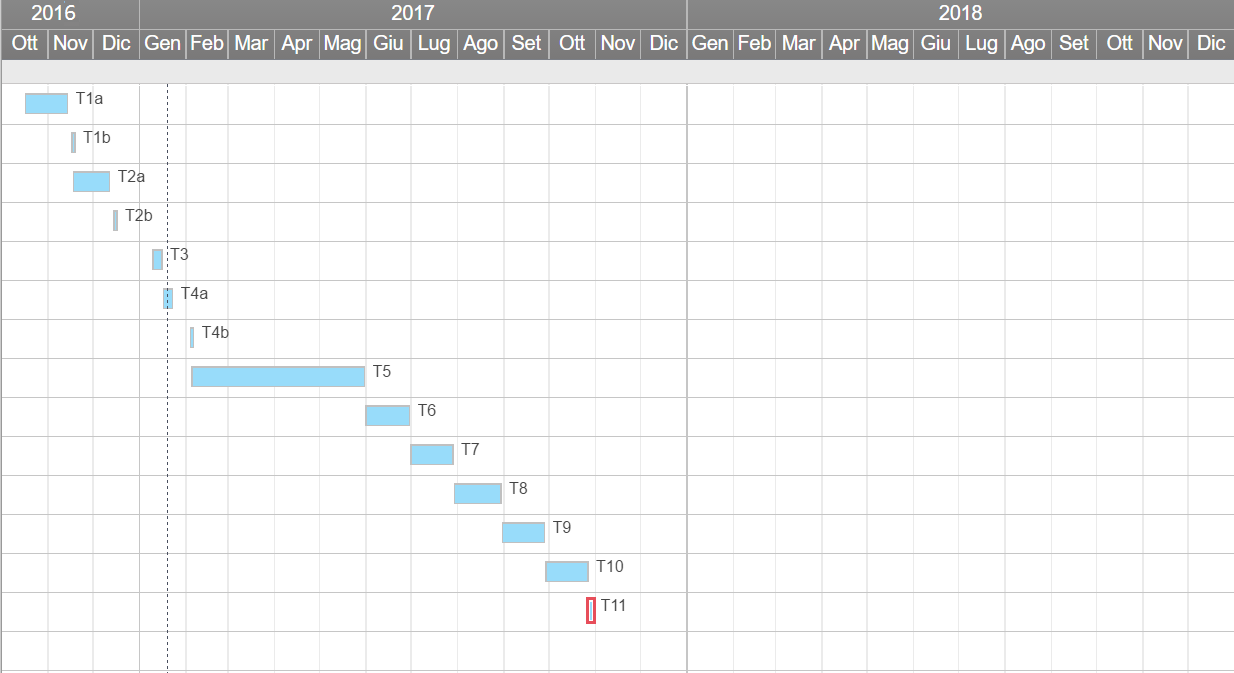
* + - * The date in which the given task starts,
      * The date in which the given task ends,
      * The interval in [*day*] that separates the starting date from the ending date.

|  |  |  |  |
| --- | --- | --- | --- |
| **Task** | **Start** | **End** | **Interval** |
| T1a | 16/10/2016 | 13/11/2016 | 28 |
| T1b | 16/11/2016 | 16/11/2016 | 1 |
| T2a | 17/11/2016 | 11/12/2016 | 24 |
| T2b | 14/12/2016 | 14/12/2016 | 1 |
| T3 | 09/01/2017 | 15/01/2017 | 6 |
| T4a | 16/01/2017 | 22/02/2017 | 6 |
| T4b | 03/02/2017 | 03/02/2017 | 1 |
| T5 | 04/02/2017 | 30/05/2017 | 117 |
| T6 | 31/05/2017 | 29/06/2017 | 29 |
| T7 | 30/06/2017 | 28/07/2017 | 28 |
| T8 | 29/07/2017 | 29/08/2017 | 31 |
| T9 | 30/08/2017 | 27/09/2017 | 28 |
| T10 | 28/09/2017 | 26/10/2017 | 28 |
| T11 | 27/10/2017 | 27/10/2017 | 1 |

**4.2.2 Gantt Diagram**

Here is built a Gantt Diagram showing the schedule chosen for *PowerEnJoy*

project tasks.



**5 Resources**

**6 Risk**

**Appendix**