Project Plan Document

Authors:
Patrizia Porati, Tommaso Sardelli

POLITECNICO
MILANO 1863

Contents

1 Introduction

1.1 Revision History

This is version 1.0 of our Project Plan Document. In this paragraph we will list changes of each future version of the document.

Version	Date	Authors	Summary
1.0	22/01/217	Patrizia Porati, Tommaso Sardelli	First version

1.2 Purpose and Scope

This is the Project Plan Document for the "PowerEnJoy" web application. The aim of this document is to analyze the compleXity of the application and identify the main tasks needed to accomplish the all project in order to provide the cost and the effort estimation. In this way we define the required budjet, the resources allocation, the schedule of the activities and identify possible risks.
In this first chapter we are going to provide a brief introduction to our document.
In the second chapter we apply two of the main algorithmic estimation techniques (Function points and COCOMO II) used to estimate size in term of line of code, the cost and the effort required to develop the entire application.
In chapter three we schedule all the activities starting from the requirements identification and the fulfillment of the plan documents to the implementation and testing.
Then we allocate some tasks to each group member.
At the and we identify possible risks that can occur and draw some general conclusions.

1.3 List of Definitions and Abbreviations

The following are used throughout the document:

RASD Requirement Analysis and Specification Document

DD Design Document

ITPD Integration Test Plan Document

PPD Project Plan Document

PP Project Plan

SLOC Source Line Of Code

LOC Line Of Code

External Input (EI) elementary operation used to elaborate data coming from the external environment

External Output (EO) elementary operation that generates data for the external environment

External Inquiry (EQ) elementary operation that involves inputs and outputs of the system

External Logic File (ELF) homogeneous set of data used by the application but generated and maintained by other applications

Function Points (FP) a technique used to evaluate the effort needed to design and develop custom software applications

Internal Logic File (ILF) homogeneous set of data used and managed by the application itself

Project Planning phase of the Software Project Management that aims at identifying tasks, estimating and scheduling the project development and assigning people to tasks

Risk potential problem that can occur

Task an activity that must be completed in order to achieve the project goal

1.4 Reference Documents

- The project descritption document: Assignments AA 2016-2017.pdf
- PowerEnJoy Requirement Analysis and Specification Document: rasd.pdf
- PowerEnJoy Design Document: dd.pdf
- PowerEnJoy Integration Test Document: itpd.pdf
- COCOMO II - Model Definition Manual: http://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_modelman2000.0.pdf
- Function Point Languages Table: http://www.qsm.com/resources/function-point-languages-table

2 Project size, cost and effort estimation

In this chapter we provide the the expected size, the cost and effort estimation of the "PowerEnJoy" application.

2.1 Size estimation: Function Points

We apply the **Function Points** approach, based on the main functionalities of the application, to evaluate the size of the project and estimate the line of code to be written. Then we can proceed with the estimation of the effort and the duration applying the rules defined by the COCOMO II approach.

We use this table to evaluate the weight for every function:

Function types	**Simple**	**Medium**	**Complex**
N. Internal Files	7	10	15
N. External Files	5	7	10
N. Inputs	3	4	6
N. Output	4	5	7
N. Inquiries	3	4	6

2.1.1 Internal Logic Files (ILFs - *entities of the system*)

"PowerEnJoy" application relies on a number of ILFs to store the information it needs to offer the required functionalities.
The system has to store information about *Registered Users* data: id, name, surname, email, password, address, user status (active, insolvent, inactive), phone number, credit card number and driving license number.
It also stores *Administrators* information: name, email and password.
Reservations are stored in a dedicated table to store an id, the id of the user who reserved the car, the identifier of the reserved car and the date and the time of the reservation.
An other table containing information that the applications needs is the table of the *Cars*, where for each car are stored an id, the license plate number, the level of the battery, the position, the status (available, unavailable, reserved, in use), locked (boolean: true if the car is locked, false otherwise), charging status (boolean: true if the car is plugged to the power grid, false otherwise).
Then there is the *Ride* table, where data concerning the ride is stored: id, start position, end position, duration, number of passengers, total price and the identifier of the related reservation.
There is also a table for *Discounts* types containing an identifier, the name, the percentage and a description.
Each ride can be related to zero or more discounts and each discount can be

related to zero or more ride, so we need a table where store this information (*Discounts Applied*): this table only contains the ride identifier and the ride identifier.
Payments information is also stored in a dedicated table containing the identifier number, the price, the time and the date, the registered user's identifier, the identifier of the related ride.
The system needs to know where and which are the *safe areas*, so this information is stored in a dedicated table, that contains: id number, address, longitude, latitude, total places, free places and power grid (boolean: true if it's a PowerGridStation, false if it's a simple SafeArea).

ILFs	**Complexity**	**FPs**
Registered Users	high	15
Administrators	simple	7
Reservations	medium	10
Cars	medium	10
Rides	high	15
Discounts	simple	7
Discounts Applied	medium	10
Payments	high	15
Safe Areas	medium	10
Total		**99**

2.1.2 External Logic Files (ELFs - *communication between different software*)

"PowerEnJoy" application has to manage data from two external services: **MapService** and **PaymentService**. The data obtained through the services APIs, retuned in JSON or XML format, must be converted to be used by our system. Between the system and the **MapService** there are two main king of interactions :

- given the coordinates of two locations, get the distance between them;
- given the coordinates, get the corresponding address;
- given an address, get the corresponding pair of coordinates (reverse geocoding);

The mapping service is also used to provide the graphical representation of the city map to be displayed on the user's device and on the car screen.

Between the system and the **PaymentService** there is only one kind of interactions:

- given the total to be paid and the user payment credentials, get a response about the payment success or failure.

All the data coming from the considered interactions are classified as *simple* complexity External Logic Files.

ELFs	**Complexity**	**FPs**
Get distance	simple	5
Get address	simple	5
Get coordinates	simple	5
Graphical representation	simple	5
Pay	simple	5
Total		**25**

2.1.3 External Inputs (EIs)

"PowerEnJoy" web application supports many kind of interactions with different actors. Here we summarize the EIs grouping them by actor's category:

- The application interacts with Guests:
  - **registration**: this operation has a *medium* complexity because it requires some checks on the validity of the inputs (it contributes 4 FPs).
- The application interacts with RegisteredUsers:
  - **login**: it is a *simple* operation that requires only email and password (it contributes 3 FPs);
  - **logout**: this is a *simple* operation that involve only the UserHandler component, so we adopt the simple weight for it (it contributes 3 FPs);
  - **password recovery**: this operation requires some steps in order to create a new password and update the database, so we adopt the *medium* weight for it (it contributes 4 FPs);
  - **update personal information**: this is a *medium* complexity operation because it requires to update some fields and check their validity (it contributes 4 FPs);
  - **reserve a car**: this operation requires a some difficult steps (like 'search available cars', 'check that the user hasn't already an active reservation') to reach the goal, so it's an operation with *high* complexity (it contributes 6 FPs);
  - **unlock the reserved car**: this operation requires some steps (like 'check user distance from the car' and 'change car status'), so it is an *high* complexity operation (it contributes 6 FPs).
- The application interacts also with Administrators:

- **insert, delete and update PowerEnJoy areas (SafeAreas and PowerGridStations)**: these operations require to check the validity of the inputs and to update the database, so we adopt the *medium* weight for them (it contributes 4 FPs each);
- **create new administrators**: this operation has a *medium* complexity because it requires to check the inputs and to update the database (it contributes 4 FPs);
- **insert, delete and update electric cars**: these operations require to check the validity of the input, check and sometimes change the status of the car, provide some car information (like the actual position) and update the database, so we adopt the *high* weight for them (it contributes 6 FPs each);
- **insert, delete and update users**: this operation has a *medium* complexity because it requires to check the inputs and to update the database (it contributes 4 FPs each);

EIs	**Complexity**	**FPs**
Registration	medium	4
Login	simple	3
Logout	simple	3
Password recovery	medium	4
Update personal information	medium	4
Reserve a car	high	6
Unlock the reserved car	high	6
Insert, delete and update areas	medium	3 * 4
Add a new admin	medium	4
Insert, delete and update cars	high	3 * 6
Insert, delete and update users	medium	3 * 4
Total		**76**

2.1.4 External Inquiries (EQs - *actions that required data retrieval from the database*)

- The "PowerEnJoy" application allows RegisteredUsers to request:
  - to visualize their personal information;
  - the list of all rides and reservation;
  - details of a selected ride or reservation;
  - the list of all available cars.
- The application also allows the Administrators to request:
  - the list of all users

- information of a selected user (personal information, status);
- the list of all payments of a selected user;
- the list of all the rides and reservations of a selected user;
- the list of all cars;
- details of a selected car;
- the list of all unavailable cars;
- the list of all safe areas;
- the list of a selected safe area.

All of these external enquiries can be considered as operation with a *simple* complexity.

EQs	**Complexity**	**FPs**
Visualize personal information	simple	3
Get the list of rides and reservations	simple	3
Get ride and reservation details	simple	3
Get the list of available cars	simple	3
Get the list of users	simple	3
Get user personal information	simple	3
Get the list of payments of a selected user	simple	3
Get the list of rides and reservations of a selected user	simple	3
Get the list of cars	simple	3
Get car details	simple	3
Get the list of unavailable cars	simple	3
Get the list of safe areas	simple	3
Get safe area details	simple	3
Total		**39**

2.1.5 External Outputs (EOs - *things created from the system for the user*)

Sometimes the "PowerEnJoy" application needs to interact with the users outside the context of an inquiry.

- **Registration**: this is an operation with *simple* complexity because the system only sends a password to the new registered user through an email notification;
- **Password recovery**: this is an operation with *simple* complexity because the system only sends a new password to the registered user through an email notification;

- **Notification at the end of the ride**: the system sends an email to the user containing all the details of his/her last ride and successful payment (total cost, applied discounts and fees, confirmation of successful payment, ride duration, starting location, ending location...), then this operation has a *high* complexity because it requires four entities: User, Payment, Discounts, Ride;

- **Notification confirming status changes**: the system sends to the user a notification email when his/her status changes (automatically for insolvent payments or by an administrator's operation), then this is a *simple* complexity operation because requires only the status of the user.

- **Notification for payment failure**: the system sends a notification email to the user if the payment is not successful, then this is an operation with *medium* complexity because it requires two entities: User and Payment.

EOs	**Complexity**	**FPs**
Registration	simple	4
Password recovery	simple	4
Notification at the end of the ride	high	7
Notification confirming status changes	simple	4
Notification for payment failure	medium	5
Total		**24**

2.1.6 Overall estimation

This table is a summary of the number of function points that we have found in our project, each multiplied by its own weight:

Function types	**Simple**	**Medium**	**Complex**	**FPs**
N. Internal Files	2 * **7**	4 * **10**	3 * **15**	99
N. External Files	5 * **5**	0 * **7**	0 * **10**	25
N. Inputs	2 * **3**	10 * **4**	5 * **6**	76
N. Output	3 * **4**	1 * **5**	1 * **7**	24
N. Inquiries	13 * **3**	0 * **4**	0 * **6**	39
TOTAL				**264**

The value obtained, representing the total number of Function Points, can be used to estimate the size of the project in terms of Source Lines of Code.

$$\textbf{LOC} = \textbf{AVG} * \textbf{FPs}$$

AVG is a language-dependent factor.

Although in the previous documents (RASD, DD, ITPD) we decided not to select a specific programming language in order to leave more margin of choice to the developers, at this point of the project we have to choose a programming language because we need to calculate a size, effort and cost estimations of the application development. We suppose to develop out system using JavaScript, whose average lines of code per FPs is 47.
Then our application source will have approximately

$$47 * 264 = \mathbf{12\ 498} \text{ Lines Of Code.}$$

2.2 Cost and effort estimation: COCOMO II

In this section we are going to estimate the effort (expressed in person-months), the number of people and the total time (expressed in months) required to complete the "PowerEnJoy" project. We can do this using COCOMO II approach, which is a software cost estimation model used for estimating effort, cost and schedule of software projects.

COCOMO II is based on:

- **Source Line Of Code (SLOC)**
- **Scale Drivers (SD)**: their values determine the exponent used in the Effort Equation.
- **Cost Drivers (CD)**: their values determine the multiplicative factor used in the Effort Equation.
- **Effort Equation**: it estimates the effort measured in person-months

$$\text{Effort} = 2.94 * \text{EAF} * (\text{KSLOC})^{E}$$

  where:
  EAF: Effort Adjustment Factor derived from the Cost Drivers
  E: exponent derived from the values of the Scale Drivers

- **Schedule Equation**: it predicts the number of months that are required to complete the software project

$$\text{Duration} = 3.67 * (\text{Effort})^{SE}$$

  where:
  Effort: the result of the Effort Equation
  SE: schedule equation exponent derived from the values of the Scale Drivers

2.2.1 Scale Drivers

Description of Scale Drivers:

- **Precedentedness**: if a product is similar to several previously developed projects, then the precedentedness is high.
  Since we haven't any previous experience with this kind of projects, this value will be *low*.

- **Development Flexibility**: if there are no specific constraints to conform to pre-established requirements and external interface specifications, then the development flexibility is high.
  Since for this project there are requirements not too much detailed and no specification for the technology to be used, this value will be *nominal*.

- **Architecture / Risk Resolution**: if we have a good risk management plan, clear definition of budget and schedule, focus on architectural definition, then the risk resolution is high.
  Since the risk analysis we performed is quite extensive, then the value will be set to *nominal*.

- **Team Cohesion**: if all stakeholders are able to work in a team and share the same vision and commitment, then the team cohesion is high.
  Since in our team all the group members know each other and work together in a cooperative way, this value is *high*.

- **Process Maturity**: refers to a well known method for assessing the maturity of a software organization.
  We set this value to *nominal*.

In order to evaluate the values of the scale drivers, we refer to the following official COCOMO II table:

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

The results of our evaluation are:

Scale Driver	Factor	Value
Precedentedness (PREC)	low	4.96
Development Flexibility (FLEX)	nominal	3.04
Risk Resolution (RESL)	nominal	4.24
Team Cohesion (TEAM)	high	2.19
Process Maturity (PMAT)	nominal	2.68
Total		**17.11**

2.2.2 Cost Drivers

The selection of used cost drivers depend on whether we are in the case of post-architecture or early design. For **post-architecture** there are 16 different factors grouped in five categories: Product Factors, Platform factors, Personnel Factors, Project Factors and General Factor.

Product Factors:

- **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. If the effect of a software failure is only slight inconvenience then RELY is very low. If a failure would risk human life then RELY is very high.
  Since "PowerEnJoy" isn't the only car sharing service in the city, a software failure could lead to important financial losses, then this value 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

- **Data Base Size (DATA)**: this measure consider the effective size of our database. 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.
  We don't have the ultimate size of the database, but it might be about 1GB. Since the program size is equal to 12498 SLOC, the rate D/P is 80.012, then this value is set to *nominal*.

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

- **Product Complexity (CPLX)**: set to *high* according to the new COCOMO II CPLEX rating scale.

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

- **Developed for Reusability (RUSE)**: this cost driver accounts for the additional effort needed to construct components intended for reuse on current or future projects. This effort is consumed with creating more generic design of software, more elaborate documentation, and more extensive testing to ensure components are ready for use in other applications.
  In our project there are some reusable components since we tried to design the system as modular as possible. Then the RUSE cost driver is *nominal*.

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

- **Documentation Match to Life-Cycle Needs (DOCU)**: in COCOMO II, the rating scale for the DOCU cost driver is evaluated in terms of the suitability of the projects documentation to its life-cycle needs. The rating scale goes from Very Low (many life-cycle needs uncovered) to Very High (very excessive for life-cycle needs).
  In our case each aspect of the system os already described in the RASD, in the DD and in the ITPD, so the DOCU cost driver is set to *nominal*.

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

Platform factors:

- **Execution Time Constraint (TIME)**: this parameter describes the expected amount of CPU usage with respect to the computational capabilities of the hardware. The rating ranges from nominal, less than 50% of the execution time resource used, to extra high, 95% of the execution time resource is consumed.
  Since our application is a complex piece of software, then the TIME value is *high*.

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

- **Main Storage Constraint (STOR)**: this parameter describes the expected amount of storage usage with respect to the availability of the hardware. Since our storage availablity is of several terabytes, then the STOR value si *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

- **Platform Volatility (PVOL)**: "Platform" is used here to mean the complex of hardware and software (OS, DBMS, etc.) the software product calls on to perform its tasks. If the software to be developed is an operating system then the platform is the computer hardware. If a database management system is to be developed then the platform is the hardware and the operating system. If a network text browser is to be developed then the platform is the network, computer hardware, the operating system, and the distributed information repositories. The platform includes any compilers or assemblers supporting the development of the software system. This rating ranges from low, where there is a major change every 12 months, to very high, where there is a major change every two weeks.
  Since "PowerEnJoy" application shouldn't change too often: it could only require some updates once a year. Then the PVOL value is *low*.

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

Personnel Factors:

- **Analyst Capability (ACAP)**: analysts are personnel who work on requirements, high-level design and detailed design. The major attributes that should be considered in this rating are analysis and design ability, efficiency and thoroughness, and the ability to communicate and cooperate.
  Since we dedicated a lot of effort analysing the requirements problems

keepeing in mind the possible real implementation. In particular, not only we can grant that the requirements have been correctly studied and accomplished, but also that our design makes our application actually useful for the user, providing all the basic functionalities he/she may need. In particular we resolved any ambiguity present in the initial description in the RASD. Then the ACAP value is set to *high*.

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

- **Programmer Capability (PCAP)**: evaluation should be based on the capability of the programmers as a team rather than as individuals. Major factors which should be considered in the rating are ability, efficiency and thoroughness, and the ability to communicate and cooperate.
  We set the PCAP value to *high*.

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

- **Personnel Continuity (PCON)**: the rating scale for PCON is in terms of the projects annual personnel turnover: from 3%, very high continuity, to 48%, very low continuity.
  Since the time we can spend on this project is limited and also the available time is less than a year, then this value is *nominal*.

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	

- **Applications Experience (APEX)**: the rating for this cost driver (formerly labeled AEXP) is dependent on the level of applications experience of the project team developing the software system or subsystem. The ratings are defined in terms of the project team's equivalent level of experience with this type of application. A very low rating is for application experience of less than 2 months. A very high rating is for experience of 6 years or more.
  Since we haven't a lot of experience in this typology of project, then this value is equal to *low*.

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

- **Platform Experience (PLEX)**: the Post-Architecture model broadens the productivity influence of platform experience, PLEX (formerly labeled PEXP), by recognizing the importance of understanding the use of more powerful platforms, including more graphic user interface, database, networking, and distributed middleware capabilities.
  We have some experience with databases, user interfaces and server-side development, so this parameter value is *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

- **Language and Tool Experience (LTEX)**: this is a measure of the level of programming language and software tool experience of the project team developing the software system or subsystem. Software development includes the use of tools that perform requirements and design representation and analysis, configuration management, document extraction, library management, program style and formatting, consistency checking, planning and control, etc. In addition to experience in the projects programming language, experience on the projects supporting tool set also affects development effort. A low rating is given for experience of less than 2 months. A very high rating is given for experience of 6 or more years.
  Since we have some experiences with JavaScript, databases, user interfaces and server-side development, then this value is *nominal*.

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	

Project Factors:

- **Use of Software Tools (TOOL)**: the tool rating ranges from simple edit and code, very low, to integrated life-cycle management tools, very high.
  Since our application environment is complete and well integrated, so this parameter value is *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

- **Multisite Development (SITE)**: because of multisite development effects are significant, it's important to consider the SITE cost driver. Determining SITE cost driver rating involves the assessment and judgement-based averaging of two factors: site collocation (from fully collocated to international distribution) and communication support (from surface mail and some phone access to full interactive multimedia).
  We live in different cities, but we collaborate using phones, email and social networks. So we set this parameter to *high*.

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

General Factors:

- **Required Development Schedule (SCED)**: this rating measures the schedule constraint imposed on the project team developing the software. The ratings are defined in terms of the percentage of schedule stretch-out or acceleration with respect to a nominal schedule for a project requiring a given amount of effort. Accelerated schedules tend to produce more effort in the earlier phases to eliminate risks and refine the architecture, more effort in the later phases to accomplish more testing and documentation in parallel.
  Our effort were well distribuited over the available time, but the requirements and design architecture analysis required a lot of time. Then this parameter value is *high*.

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

Here are our results:

Cost Driver	**Factor**	**Value**
Product Factors		
Required Software Reliability (RELY)	high	1.10
Data Base Size (DATA)	nominal	1.00
Product Complexity (CPLX)	high	1.17
Developed for Reusability (RUSE)	nominal	1.00
Documentation Match to Life-Cycle Needs (DOCU)	nominal	1.00
Platform Factors		
Execution Time Constraint (TIME)	high	1.11
Main Storage Constraint (STOR)	nominal	1.00
Platform Volatility (PVOL)	low	0.87
Personnel Factors		
Analyst Capability (ACAP)	high	0.85
Programmer Capability (PCAP)	high	0.88
Personnel Continuity (PCON)	nominal	1.00
Applications Experience (APEX)	low	1.10
Platform Experience (PLEX)	nominal	1.00
Language and Tool Experience (LTEX)	nominal	1.00
Project Factors		
Use of Software Tools (TOOL)	high	0.90
Multisite Development (SITE)	high	0.93
General Factors		
Required Development Schedule (SCED)	high	1.00
Total		**0.85593**

2.2.3 Effort Equation

This equation gives us the effort estimation measured in Person-Months:

$$Effort := A * EAF * (KSLOC)^E$$

where:

A: 2.94 (for COCOMO.2000)

EAF: product of all the Cost Drivers, that in our case is: **0.85593**

E: exponent derived from Scale Drivers. It's calculated as:

$$E := B + 0.01 * \sum_j SF_j$$

that in our case is: 0.91 + 0.01 * **17.11** = **1.0811**
in which B is equal to 0.91 for COCOMO.2000.

KLOC: estimated lines of code using the FP analysis, in our case are: **12.498**KLOC

With thes parameters we can compute the Effort value, that is equal to:

$$Effort = 2.94 * 0.85593 * 12.498^{1.0811} = \mathbf{38.6}\ PM \sim \mathbf{39}\ PM$$

2.2.4 Schedule Estimation

This equation gives us the schedule estimation measured in Month:

$$Duration := 3.67 * (Effort)^{SE}$$

where:

Effort: the result of the Effort Equation, that in our case is **38.6**PM

SE: schedule equation exponent derived from the values of the Scale Driver. It's calculated as:

$$SE := 0.28 + 0.2 * (E - B)$$

that in our case is: 0.28 + 0.2 * 0.1711 = **0.31422**

With these parameters we can compute the Duration value, that is equal to:

$$Duration = 3.67 * (38.6)^{0.31422} = \mathbf{11.56}\ months$$

3 Schedule

In this chapter we are going to identify all the tasks that had to be done for complete the project.
These are the main tasks:

T1: Creation of **Requirement Analysis and Specification Document**, which explains in details functional and non-functional requirements, domain assumption and goals of the application

- **T1.1:** Read carefully the project specification and provide a general introduction concerning the requirements analysis phase
- **T1.2:** Identify requirements, goals and domain properties of the project
- **T1.3:** Identify the actors that interact with the application
- **T1.4:** Identify the main functions of the system
- **T1.5:** Identify possible scenarios
- **T1.6:** Design UML diagrams for the application functionalities
- **T1.7:** Model the system in Alloy
- **T1.8:** Revise all the document

T2: Creation of the **Design Document**, which explains in detail the architecture of the application

- **T2.1:** Provide a general introduction concerning the design phase
- **T2.2:** Describe the architecture of the system (components, deployment, runtime view)
- **T2.3:** Describe some of the main algorithms
- **T2.4:** Illustrate the user interface providing some mockups
- **T2.5:** Provide the mappings between architectural components and requirements
- **T2.6:** Revise all the document

T3: Creation of the **Integration Test Plan Document**, which explains in detail the integration testing strategy that we want to use for checking the system behaviour

- **T3.1:** Provide a general introduction concerning the integration testing phase
- **T3.2:** Define the integration strategy
- **T3.3:** Describe in detail each step of the integration testing
- **T3.4:** Provide a description of the tools and the test data required to perform the testing

T3.5: Revise all the document

T4: Creation of the **Project Plan Document**, this document.

T4.1: Provide a general introduction on the project planning and cost/effort estimation

T4.2: Describe and perform the Function Point algorithm to estimate the size of the project providing an approximate number of lines of code number

T4.3: Describe and apply the COCOMO II model in order to evaluate the cost and the effort in terms of working months

T4.4: Identify the main tasks, provide a possible schedule and allocate the needed resources

T4.5: Identify and describe possible risks concerning the project development

T4.6: Revise all the document

T5: Preparation of a **presentation** about the mentioned documents to our client

T5.1: Preparation of slides and speech to Requirements presentation

T5.2: Preparation of slides and speech for Design and Architecture presentation

T5.3: Preparation of slides and speech for Integration Testing presentation

T5.4: Preparation of slides and speech for Project Plan presentation

T5.5: Revise all the slides

T6: **Development** of the entire application and check each components with unit tests

T6.1: Develop the database

T6.2: Develop RESTfulAPI component

T6.3: Develop UserHandler component

T6.4: Develop CarHandler component

T6.5: Develop AreaHandler component

T6.6: Develop ReservationHandler component

T6.7: Develop RideHandler component

T6.8: Develop PaymentHandler component

T7: Run **integration testing** on the application following the Integration Test Plan.

T7.1: Run Integration 1 (refers to the TestCases in the ITPD)

T7.2: Run Integration 2 (refers to the TestCases in the ITPD)

T7.3: Run Integration 3 (refers to the TestCases in the ITPD)

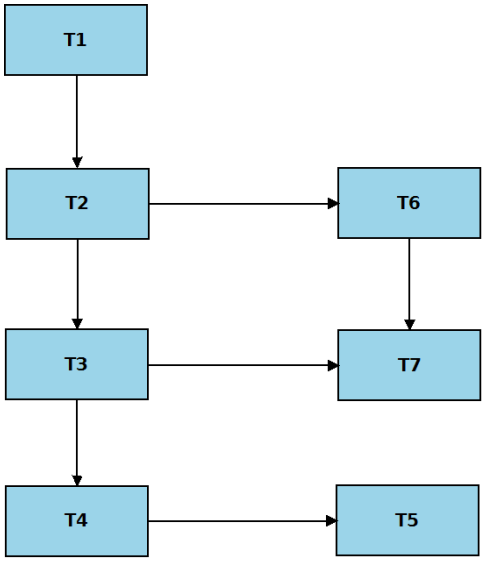
T7.4: Run Integration 4 (refers to the TestCases in the ITPD)

T7.5: Run Integration 5 (refers to the TestCases in the ITPD)

T7.6: Run Integration 6 (refers to the TestCases in the ITPD)

Tasks T1, T2 had T3 had been completed before the given submission deadlines, that were 13/11/2016 for RASD, 11/12/2016 for DD and 15/01/2017 for ITPD.
Task T4 and T5, that are the submission of this document and the presentation of the entire project documents, will be finished within the established dates; in particular T4 has to be completed before 22/01/2017 and T5 before mid-February.
Concerning tasks T6 and T7, no schedule was given so, according to the COCOMO II estimation, we expect to finish the entire project implementation within about 11.5 months, that means before the end of December 2017.
In particular the integration testing will take place during the last month of the project implementation, which means that starts at November 2017.
The implementation of the "PwerEnJoy" application started right after the DD delivery. Unit tests are made in parallel with the development in order to verify the all the functionalities once at a time.

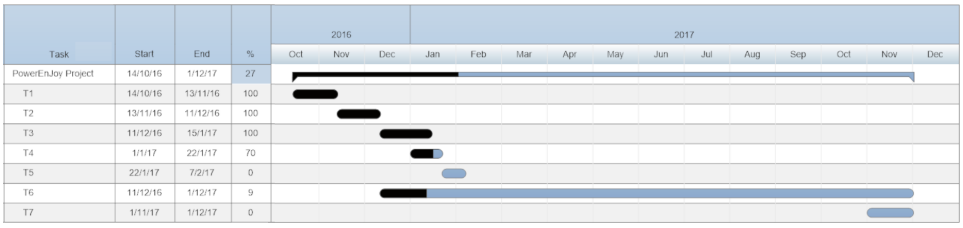
Here we provide a graph in order to better explain the dependencies between the tasks.



In the following table, for each task, we are going to show the Duration, expressed in days, the Effort, measured in person-days, and the Dependencies with other tasks.

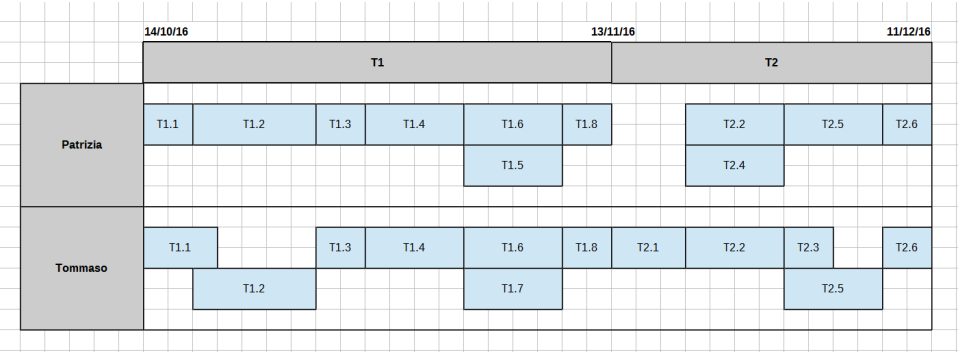
Task	**Effort(person-days)**	**Duration(days)**	**Dependencies**
T1	146	30	none
T2	96	28	T1
T3	43	35	T1, T2
T4	66	21	T1, T2
T5	50	66	T1, T2, T3, T4
T6	10000	365	T1, T2
T7	200	30	T1, T2, T3, T6

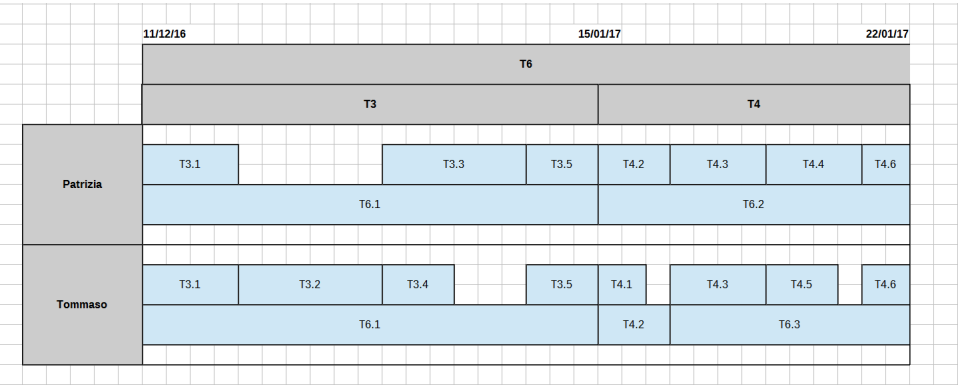
This is the **Activity bar chart**, in which we provide a schedule in order to show when each task has to be done. It's alto highlighted the percentage of working already done until this moment.

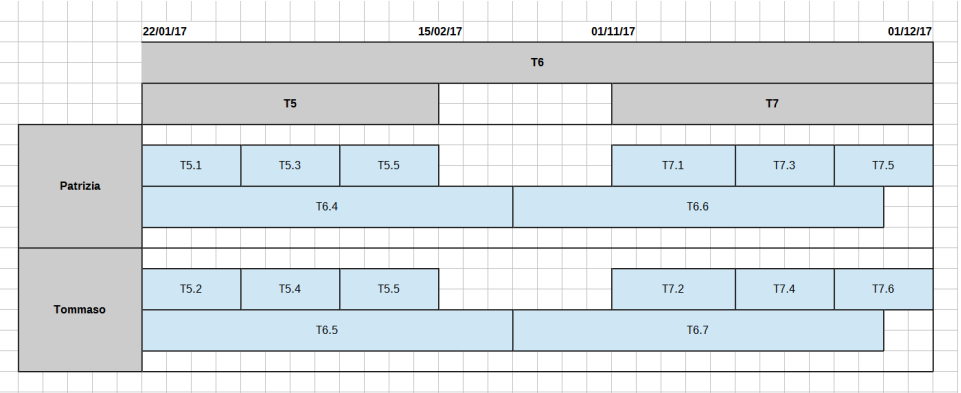


4 Resource Allocation

In this chapter we provide a **Staff Allocation Chart** (divided into three part for better readability), that explains how the resources have been allocated to each task. In particular it's indicated who took care, or will take care of which tasks and how long.







5 Risks

Table 1: Risks

#	Risk	Probability	Effects
01	Frequent changes in requirements that can slow down the project.	High	Serious
02	Impossible to recruit staff with the necessary skill level.	Moderate	Catastrophic
03	Low understanding of the architecture	Moderate	Serious
04	Insufficient communication between developers of separate subcomponents.	Moderate	Serious
05	Key staff are ill at critical times	Moderate	Serious
06	Difficulties in the integration of the system with cars and gps.	High	Catastrophic
07	High initial costs for server deployment risk to overestimate the needs.	High	Moderate

Table 2: Strategy

#	Strategy
01	Strive to have constant feedback from the customer and guide him with questions in initial analysis.
02	Alert the costumer pre-emptively and discuss possible delays.
03	Arrange reviews before starting the actual implementation and examine the architecture together
04	Arrange code reviews periodically and moments to discuss about components integration
05	Make sure that everyone in the project has at least an overall idea of others people work so that they can take their place if necessary
06	Start testing this components as early as possible and consider to hire field experts to help getting done with these tasks
07	Use IaaS for the infrastructure with the possibility to scale up upon needs

6 Effort spent

Section	Working hours	Author
Introduction	3	Tommaso Sardelli
Function Points	8	Patrizia Porati, Tommaso Sardelli
COCOMO II	12	Patrizia Porati, Tommaso Sardelli
Schedule	5	Patrizia Porati
Resource allocation	5	Patrizia Porati
Risks magagement	7	Tommaso Sardelli
PPD review	3	Patrizia Porati, Tommaso Sardelli