School of Industrial and Information Engineering

Department of Computer Science and Engineering

Software Engineering II

PowerEnJoy – Project Plan

Presented by:

Bachar Senno

Gianluigi Iobizzi

Onell Saleeby

Under the supervision of:

Prof. Luca Mottola

Prof. Elisabetta Di Nitto

V1.0, 22-01-2017

Table of Contents

1. Introduction

1.1 Revision History

- V1.0 Initial version

1.2 Purpose and Scope

This document includes the project plan for the PowerEnJoy car sharing service. The main purpose of this document is to provide an initial estimation of the effort needed, cost, and complexity of this project. To accomplish this purpose, we've divided this document into 4 main sections.

In the first section, we analyse the effort and time needed to implement this project, along with its size (expected number of lines of code). This is done by using the Function Points and COCOMO approaches in order to obtain educated estimates of these values.

In the second section, the previously found estimates are spread over a proposed schedule (Gantt chart), visualising the distribution of the tasks over time. These tasks include everything from the early requirements identification stage, to designing the architecture and components, as well as the actual implementation and testing phases.

Thirdly, these tasks are distributed over the members of the team to show a sample of how the work can be split.

And in the end, we included a risk management section, in which we discussed the various risks that might hinder the progress or cause any kind of issues, and discussed viable solutions to mitigate such risks.

1.3 Acronyms

- FP: Function Points.
- ILF: Internal logic file
- ELF: External logic file.
- EI: External Input.
- EO: External Output.
- EQ: External Inquiries.
- DBMS: Database Management System.
- OS: Operating System.
- API: Application Programming Interface.
- GUI: Graphical User Interface.

2. Project Size, Cost and Effort Estimation

In this chapter, we present some semi quantitative algorithmic techniques to give, as possible a realistic estimation of the project size and, consequentially, of the possible required amount time to develop it. In particular, we choose the COCOMO method (COnstructive COst Model v.II) originally presented by Barry Boehm in 1981 following a statistical approach to relate cost and size of a large number of previously developed software project. For the whole activity, the focus will be only on the application logic of the system, totally forgetting the user's interfaces of the mobile and in-car apps. At first, we will proceed with an estimated counting of the number of code lines (LOC) of our system using the Function Points approach that directly correlates the software's size to its various functionalities. Finally, the main COCOMO II algorithm will be applied after a reasoned determination of Cost Drivers and Scale Factors.

2.1 Size estimation: Function Points

As anticipated before the COCOMO technique expects as input an estimation of the project size expressed in LOC. Following the guidelines for our planning the Function Point approach will be used. Function points measure a software project by quantifying the information processing functionality associated with major external data or control input, output, or file types. In the following tables, we report an identification of the Function Points types to be used, together with the criteria to weight them basing on their technical complexity.

Our software will be entirely developed in the Java language.

Table 1. Function Point Types

Function Point	Description
External Input (EI)	Count each unique user data or user control input type that enters the external boundary of the software system being measured.
External Output (EO)	Count each unique user data or control output type that leaves the external boundary of the software system being measured.
Internal Logical File (ILF)	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 Logical Files (ELF)	Files passed or shared between software systems should be counted as external interface file types within each system.
External Inquiry (EQ)	Count each unique input-output combination, where input causes and generates an immediate output, as an external inquiry type.

Table 2. Complexity of ILF and ELF

	Data Elements		
Record Elements	*1-19*	*20-50*	*51+*
1	Lo	Low	Avg
2-5	Lo	Avg	Hig
6+	Av	High	Hig

Table 3. Complexity of EO and EQ

	Data Elements		
File Types	*1-5*	*6-19*	*20+*
0-1	Lo	Lo	Av
2-3	Lo	Av	Hig
4+	Av	Hi	Hig

Table 4. Complexity of EI

	Data Elements		
File Types	*1-4*	*5-15*	*16+*
0-1	Lo	Lo	Av
2-3	Lo	Av	Hig
4+	Av	Hi	Hig

Table 5. Function Points complexity weights

	Complexity Weight		
Function Type	*Low*	*Average*	*High*
Internal Logic	7	10	15
External Logic	5	7	10
External Inputs	3	4	6
External Outputs	4	5	7
External Inquiries	3	4	6

2.1.1 Internal Logical Files (ILF)

Our system uses several relational tables to store the information needed for its functioning. In the following we will analyze them to determine the related number of ILF and their complexity. Basing on the previous released class and BCE diagrams as data structure's reference (see RASD and DD), at first a single table is dedicated to store user's information which is composed by: username, email, chosen password, generated ID, address, complete name, credit card number, credit card expiration date and driving license number. As for users, also the cars need a DB table to reach a correct system functionality. In this case the table holds: car ID, plate number, number of, currently occupied seats, current power level, availability status and current location.

The reservation requests are naturally preserved into another dedicate table that contains: reserving user's ID number, reserved car's ID number, time of reservation's acceptance, a Boolean data type to register the money saving option's setting and, finally, the location coordinates. Similarly, for the registration request also the trip process requires to permanently store some data.

This is stored in a table that holds: trip ID number, user ID of the driver, ID number of the ridden car, starting point's geographical coordinates, destination's coordinates, cost of the ride, duration and the starting time of the ride. We need also a table that contains the ID, the number of available and total plugs and the location's coordination of all the system's charging stations.

Using the previously defined tables, this is the count we obtain:

ILF	Complexity	FPs
User data	Average	10
Car data	Average	10
Reservation requests	Low	7
Trip processes	Average	10
Charging station data	Low	7
Total		44

2.1.2 External Logical Files (ELF)

Analysing the system's logic, we can found four main data interfaces with external software services that are: the driving licences validation service, the credit card validation service, the payment service and, at last, the mapping service. The external service used for driving licences validation exchange with our software only a few data structures (essentially a tuple composed by a licence number, and holder name) and therefore can be certainly considerate a data interface of low complexity.

As to the credit card validation service, an analogue reasoning is applicable: the exchanged data is composed now only by the card number, the card expiration date and the holder's name and a simpler structure is returned in response, with a Boolean type as validity indicator. The same for the payment external service. The mapping service requires instead a greater data complexity since there are two different kind of interactions: given the coordinates of two locations get an estimate of the time that is necessary to drive from one to the other (medium complexity, small amount of data but continuously repeated during trip process), given an address get the correspondent pair of coordinates (low complexity).

The mapping service is also used to retrieve the graphical representation of the city map to be displayed on the smartphone of the user. Given the complexity of the interaction and the amount of data that is retrieved, it is reasonable to classify this last logical interface as a complex one.

ELF	Complexit	FP
Driving license validation	Low	5
Credit card validation	Low	5
Payment handling service	Low	5
ETA computation	Average	7
Address resolution	Low	5
Total		37

2.1.3 External Inputs (EI)

The PowerEnJoy application supports a wide range of user's input. In the following the different interaction with the users are analyzed and their complexity determined according to the previous guidelines.

The system user can:

- LOGIN to the system – this is a low complexity operation since it involves only user's email and password to be acquired and checked;
- REGISTER himself to the system – the creation of a new account requires a larger tuple to be acquired and so can be considered to have an average complexity;
- RETRIVE the forgotten password – many control inputs and steps are required so the operation is an input of average complexity;
- SET A STARTING ADDRESS to change the map screen of available cars – a low complexity input with only a string representing the starting address
- RESERVE an available car – a reservation request requires several data types like the car ID number, the userID, the destination address and the money saving option setting and so is an input of average complexity;
- UNLOCK a reserved car / START A TRIP by turning on the car engine – two simple inputs which consists on few bytes of data;
- CANCEL A TRIP before its natural ending thought the emergency call button – require both car's, user's and trip's data so it is of average complexity;

The whole analysis leads to this summary table:

EI	Complexit	FP
Login	Low	3
Password retrieval	Average	4
New account registration	Average	4
Starting address setting	Low	3
Car reservation request	Average	4
Car unlocking	Low	3
Trip starting (engine turned on)	Low	3
Emergency call during trip	Average	4
Total		28

2.1.4 External Inquiries (EQ)

An External Inquiry can be defined as a data retrieval performed by the system's users. In our case we have got only three types of these requests: a user can ask the system for a complete list his previous trips (several data fields involved so average complexity), get the details of a car on the map (such as the power level, the availability and the total number of seats – low complexity), and see the details of an already registered car reservation (another low complexity inquiry). Of course, it's predictable that the system's administrator would retrieve the full list of registered users, cars and see the single trip's details for statistical reasons (average complexity for the lists of cars and users because of the number of tuples and low complexity for trip details retrieval).

Therefore, the FP calculation is reported in the following table:

EQ	Complexi	FPs
Retrieve user's previous trips	Average	4
Retrieve user's current reservation details	Low	3
Retrieve car's details	Low	3
Retrieve list of registered users (admin)	Average	4
Retrieve list of active cars (admin)	Average	4
Retrieve details of a single trip (admin)	Low	3
Total		21

2.1.5 External Outputs (EO)

Apart from the inquires individuated before, the PowerEnJoy system has to use external communications in many functioning situations, especially for control reasons:

- Notify the user that a login request has been accepted / rejected;
- Notify the user that the registration procedure is correctly finished or not;
- Notify the user that an inserted starting point address is not valid;
- Notify the user that the reservation request has been accepted/ rejected;
- Notify the user near a reserved car that he can unlock it and start the trip;
- Notify the user about the total cost of the ride at the end of a trip

All the notifications are fairly simple and so the resulting table is the following one:

EO	Complexit	FP
Login request outcome	Low	4
Registration request outcome	Low	4
Starting address error notification	Low	4
Car reservation request outcome	Low	4
Reserved car unlocking notification	Low	4
Trip details summary	Low	4
Total		24

2.1.6 Overall estimation

In the following we summarize the results of our FP analysis :

Function Type	Valu
Internal Logic Files	44
External Logic Files	37
External Inputs	28
External Inquiries	21
External Outputs	24
Total	154

Since the code will be developed in Java language using the JEE platform we can take as valid a lower LOC/FP of 46 and 67 as the upper estimator. So, our final estimation of the project size, considering that the whole GUI complexity analysis was skipped, is:

$$LOC = 154 * 46 = 7084 \quad \text{(lower bound)}$$

$$LOC = 154 * 67 = 10318 \quad \text{(upper bound)}$$

2.2 Cost and Effort estimation: COCOMO II

In this section, we will use the already introduced COCOMO II method to get a good estimation of the effort, and consequentially, the cost of the whole project.

In particular, the analysis will lead to useful indicators about the duration required for software development in its various phase.

As soon as presented the various steps of the approach will be briefly explained. In order to obtain more details, please refer to the COCOMO II Model Definition Manual [1]

2.2.1 Scale drivers

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

Scale Factor Values, SF_j, for COCOMO II Models

Scale Factors	**Very Low**	**Low**	**Nominal**	**High**	**Very High**	**Extra High**
PREC SFj:	thoroughly unprecedented	largely unprecedented	somewhat unprecedented	Generally familiar	largely familiar	thoroughly familiar
FLEX **SFj:**	rigorous 5.07	occasional relaxation 4.05	some relaxation 3.04	general conformity 2.03	some conformity 1.01	general goals 0.00
RESL **SFj:**	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 **SFj:**	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 SFj:	The estimated Equivalent Process Maturity Level (EPML) or					
	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

Using the Model Definition Manual as guide, we compile a specific table for each indicator in order to evaluate its level:

PRECEDENTENESS EVALUATION

Feature	Very Low	Nominal / High	Extra High
Organizational understanding of product objectives	General	Considerable	Thorough
Experience in working with related software systems	Moderate	Considerable	Extensive
Concurrent development of associated new hardware and operational procedures	Extensive	Moderate	Some
Need for innovative data processing architectures, algorithms	Considerable	Some	Minimal

Indicator chosen level: low (not much experience with large scale projects)

DEVELOPMENT FLEXIBILITY EVALUATION

Feature	Very Low	Nominal / High	Extra High
Need for software conformance with pre-established requirements	Full	Considerable	Basic
Need for software conformance with external interface specifications	Full	Considerable	Basic
Combination of inflexibilities above with premium on early completion	High	Medium	Low

Indicator chosen level: low (strict requirements about functionalities but none on other project aspects)

RISK RESOLUTION EVALUATION

Characteristic	Very Low	Low	Nominal	High	Very High	Extra High
Risk Management Plan identifies all critical risk items, establishes milestones for resolving them by PDR or LCA.	None	Little	Some	Generally	Mostly	Fully
Schedule, budget, and internal milestones through PDR or LCA compatible with Risk Management Plan.	None	Little	Some	Generally	Mostly	Fully
Percent of development schedule devoted to establishing architecture, given general product objectives.	5	10	17	25	33	40
Percent of required top software architects available to project.	20	40	60	80	100	120
Tool support available for resolving risk items, developing and verifying architectural specs.	None	Little	Some	Good	Strong	Full
Level of uncertainty in key architecture drivers: mission, user interface, COTS, hardware, technology, performance.	Extreme	Significant	Considerable	Some	Little	Very Little
Number and criticality of risk items.	> 10 Critical	5-10 Critical	2-4 Critical	1 Critical	> 5 Non-Critical	< 5 Non-Critical

Indicator chosen level: high (extensive risk analysis activity performed)

TEAM COHESION EVALUATION

Characteristic	Very Low	Low	Nominal	High	Very High	Extra High
Consistency of stakeholder objectives and cultures	Little	Some	Basic	Considerable	Strong	Full
Ability, willingness of stakeholders to accommodate other stakeholders' objectives	Little	Some	Basic	Considerable	Strong	Full
Experience of stakeholders in operating as a team	None	Little	Little	Basic	Consider - able	Extensive
Stakeholder teambuilding to achieve shared vision and commitments	None	Little	Little	Basic	Consider - able	Extensive

Indicator chosen level: very high (both stakeholders and team members are well coordinated and strongly motivated to complete the project with success)

PROJECT MATURITY EVALUATION

Maturity Level	Process Characteristics	Behaviors
Level 5 - Optimizing	Focus is on continuous quantitative improvement	Focus on "fire prevention"; improvement anticipated and desired, and impacts assessed.
Level 4 - Quantitatively Managed	Process is measured and controlled	Greater sense of teamwork and interdependencies
Level 3 - Defined	Process is characterized for the organization and is proactive	Reliance on defined process. People understand, support and follow the process.
Level 2 - Managed	Process is characterized for projects and is often reactive	Over reliance on experience of good people – when they go, the process goes. "Heroics."
Level 1 - Initial	Process is unpredictable, poorly controlled, and reactive	Focus on "fire fighting"; effectiveness low – frustration high.

Indicator chosen level: level 3 (the project is well defined in its requirements, goals, structures and activities)

The Scale Drivers analysis leads to the following resulting table:

Scale Driver	Factor	Val
Precedentedness (PREC)	Low	4.96
Development flexibility (FLEX)	Low	4.05
Risk resolution (RESL)	High	2.83
Team cohesion (TEAM)	Very	1.10
Process maturity (PMAT)	Level 3	3.12
Total		**16.0**

2.2.2 Cost drivers

As done for the Scale Drivers estimation, we will follow step by step the COCOMO II Model Definition Manual.

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.

RELY Descriptors:	slight inconvenience	low, easily recoverable losses	moderate, easily recoverable	high financial loss	risk to human life	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort	0.82	0.92	1.00	1.10	1.26	n/a

Indicator chosen level: Nominal (the system doesn't consist in an essential service for citizenship.

In case of occasional break down of our car sharing the users can always use other means of transport such as metropolitan or taxies)

DATABASE SIZE (DATA)

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

DATA Descriptors		D/P < 10	10 <= D/P < 100	100 <= D/P < 1000	D/P >= 1000	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort	n/a	0.90	1.00	1.14	1.28	n/a

Indicator chosen level: High (with the kind of data to be stored a reasonable dimension of our DB can be 2.5 Gb, that with the previous estimated size leads to a D/L ratio between 242 and 353)

PRODUCT COMPLEXITY (CPLX)

Complexity is divided into five areas: control operations, computational operations, device-dependent operations, data management operations, and user interface management operations. The complexity rating is the subjective weighted average of the selected area ratings. After having performed the evaluations prescribed by table 19 of COCOMO II Model Definition Manual, our system can be classified to have an *high* complexity level.

Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort	0.73	0.87	1.00	1.17	1.34	1.74

REQUIRED REUSABILITY (RUSE)

This cost driver accounts for the additional effort needed to construct components intended for reuse on current or future projects.

RUSE Descriptors:		none	across project	across program	across product line	across multiple
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

Indicator chosen level: Nominal (components developed to be reused across the project itself)

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 project's documentation to its life-cycle needs

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	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort	0.81	0.91	1.00	1.11	1.23	n/a

Indicator chosen level: Nominal (complete but synthetic project documentation)

EXECUTION TIME CONSTRAINT (TIME)

This is a measure of the execution time constraint imposed upon a software system. The rating is expressed in terms of the percentage of available execution time expected to be used by the system or subsystem consuming the execution time resource.

TIME Descriptors:			<= 50% use of available execution	70% use of available execution	85% use of available execution	95% use of available execution
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort	n/a	n/a	1.00	1.11	1.29	1.63

Indicator chosen level: High (the software is moderately complex and, especially for as regards the mobile app, it occupies a lot of hardware subsystems)

MAIN STORAGE CONSTRAINT (STOR)

This rating represents the degree of main storage constraint imposed on a software system or subsystem.

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	n/a	n/a	1.00	1.05	1.17	1.46

Indicator chosen level: Nominal (storage devices of high capability for all the system's calculators)

PLATFORM VOLATILY (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.

PVOL Descriptors:		Major change every 12 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	n/a	0.87	1.00	1.15	1.30	n/a

Indicator chosen level: Nominal (two major releases per year to maintain a good functioning with new versions of mobile's operating systems and of DBMS. Frequent minor updates to fix bugs)

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 (not the experience).

ACAP Descriptors:	15th percentil	35th percentil	55th percentil	75th percentil	90th percentil	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort	1.42	1.19	1.00	0.85	0.71	n/a

Indicator chosen level: High (our analysts have demonstrated their capability through the fully achievement of project's design goals)

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 (not the experience).

PCAP Descriptors:	15th percentil	35th percentil	55th percentil	75th percentil	90th percentil	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort	1.34	1.15	1.00	0.88	0.76	n/a

Indicator chosen level: Nominal (we don't have to really implement the software but we have already studied the Java language)

PERSONELL CONTINUITY (PCON)

The rating scale for PCON is in terms of the project's annual personnel turnover.

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	

Indicator chosen level: High (is reasonably foreseeable that our personnel would be faithful to the project development till its end)

APPLICATIONS EXPERIENCE (APEX)

The rating for this cost driver 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.

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

Indicator chosen level: Low (our practical experience in developing software following JEE paradigm is rather limited)

PLATFORM EXPERIENCE (PLEX)

The COCOMO II model broadens the productivity influence of platform experience, PLEX, by recognizing the importance of understanding the use of more powerful platforms, including more graphic user interface, database, networking, and distributed middleware capabilities.

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

Indicator chosen level: Nominal (no much experience with JEE platform but familiarity with databases, GUI and networking)

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. 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.

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	

Indicator chosen level: Nominal (since we have almost never developed with JEE platform, we are familiar with some Java developing environments and testing tools. In addition, we have a good knowledge of Alloy modelling language for requirements analysis)

USE OF SOFTWARE TOOLS (TOOL)

The tool rating ranges from simple edit and code, very low, to integrated life-cycle management tools, very high

TOOL Descriptors	edit, code, debug	simple, frontend, backend CASE, little	basic life-cycle tools, moderately integrated	strong, mature life-cycle tools, moderately integrated	strong, mature, proactive lifecycle tools, well integrated with processes
Rating Levels	Very Low	Low	Nominal	High	Very High
Effort	1.17	1.09	1.00	0.90	0.78

Indicator chosen level: High (complete and integrated developing and testing environment)

MULTISITE DEVELOPMENT (SITE)

Determining this 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).

SITE: Collocation	International	Multi-city and Multi-	Multi-city or Multi-	Same city or metro area	Same building or	Fully collocated
SITE: Communications Descriptors:	Some phone, mail	Individual phone, FAX	Narrow band email	Wideband electronic communicatio	Wideband elect. comm.,	Interactiv e multimedi
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra
Effort	1.22	1.09	1.00	0.93	0.86	0.80

Indicator chosen level: High (development essentially performed in the same city. Wide use of electronic communication and social media)

REQUIRED DEVELOPMENT SCHEDULE (SCHED)

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.

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

Indicator chosen level: Low (strict and frequent deadlines have slightly accelerated the whole project)

Finally, the following table summarizes the obtained values for the cost drivers:

Cost Driver	Factor	Value
Required Software Reliability (RELY)	Nominal	1.00
Database size (DATA)	High	1.14
Product complexity (CPLX)	High	1.17
Required Reusability (RUSE)	Nominal	1.00
Documentation match to life-cycle needs (DOCU)	Nominal	1.00
Execution Time Constraint (TIME)	High	1.11
Main storage constraint (STOR)	Nominal	1.00
Platform volatility (PVOL)	Nominal	1.00
Analyst capability (ACAP)	High	0.85
Programmer capability (PCAP)	Nominal	1.00
Application Experience (APEX)	Low	1.10
Platform Experience (PLEX)	Nominal	1.00
Language and Tool Experience (LTEX)	Nominal	1.00
Personnel continuity (PCON)	High	0.90
Use of Software Tools (TOOL)	High	0.90
Multisite development (SITE)	High	0.93
Required development schedule (SCED)	Low	1.14
Total factor (multiplication)		**1.18877**

2.2.3 Effort equation

This final equation gives us the effort estimation measured in Person-Months (PM):

$$PM_{NS} = A \times Size^{E} \times \prod_{i=1}^{n} EM_i$$

$$\text{where } E = B + 0.01 \times \sum_{j=1}^{5} SF_j$$

With:

A = 2.94 B = 0.91 Size – Estimated KLOC

$\prod_{i=1}^{n} EM_i$ - Product of Cost Drivers = 1.18877

$\sum_{j=1}^{5} SF_j$ - Sum of Scale Drivers = 16.06

So:

E = 0.91 + (0.01 x 16.06) = 1.0706

With these parameters, we can compute the effort value, which has a lower bound of:

PM low = 2.94 x (7.084) ^ (1.0706) x 1.18877 = 28.428 => **28 PM**

and as upper bound:

PM high = 2.94 x (10.318) ^ (1.0706) x 1.18877 = 42.521 => **43 PM**

2.2.4 Schedule estimation

Regarding the final schedule, we are going to use the following formula:

$$\mathrm{TDEV}_{\mathrm{NS}} = \mathrm{C} \times (\mathrm{PM}_{\mathrm{NS}})^{\mathrm{F}}$$

$$\text{where } \mathrm{F} = \mathrm{D} + 0.2 \times 0.01 \times \sum_{\mathrm{j}=1}^{5} \mathrm{SF}_{\mathrm{j}}$$

$$= \mathrm{D} + 0.2 \times (\mathrm{E} - \mathrm{B})$$

With: C = 3.67; D = 0.28; F = 0.28 + 0.2 x (1.0706 – 0.91) = 0.31212

So, for the total duration of the project our lower esteem is:

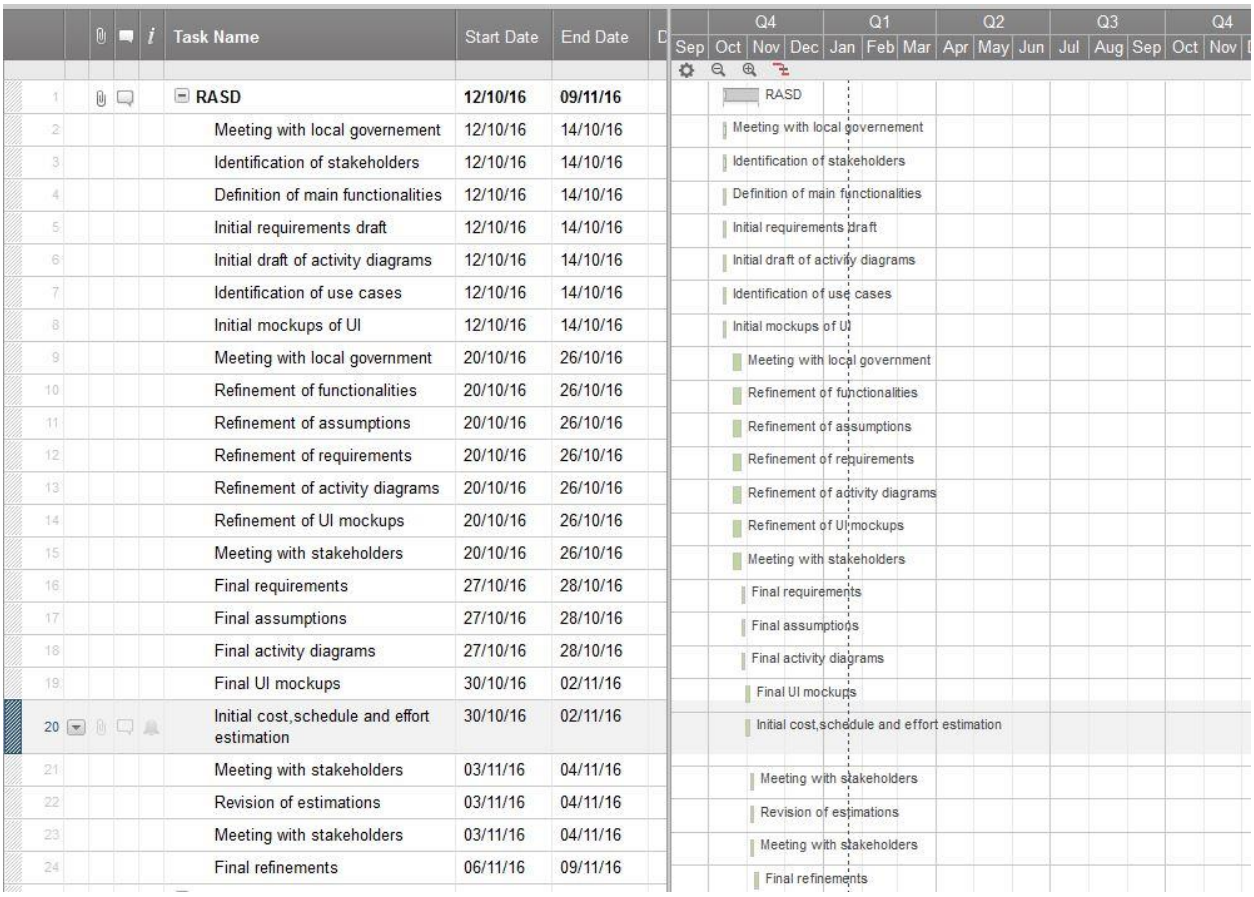
TDEVns low = 3.67 x (28.428) ^ (0.31212) = **10.43 months**

and as upper bound we have:

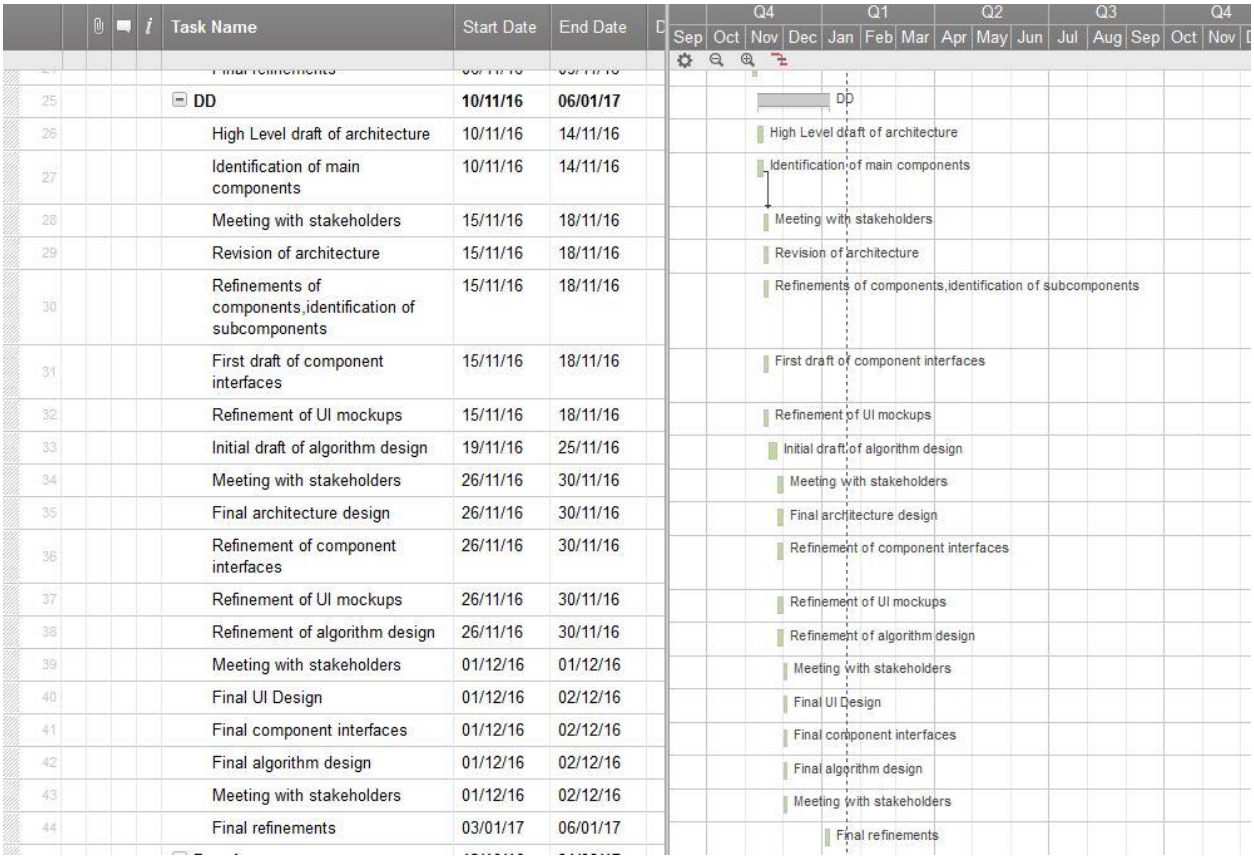
TDEVns high= 3.67 x (42.521) ^ (0.31212) = **11,83 months**

3. Schedule

In this chapter, we're going to provide a general, high-level project schedule. More refined schedules will be defined during the project to manage the internal organization of the single development phases. For being accurate, we have split the schedule into 4 quarters, Q1 from October to December, Q2 from January march, Q3 from April to June, Q4 from July to September

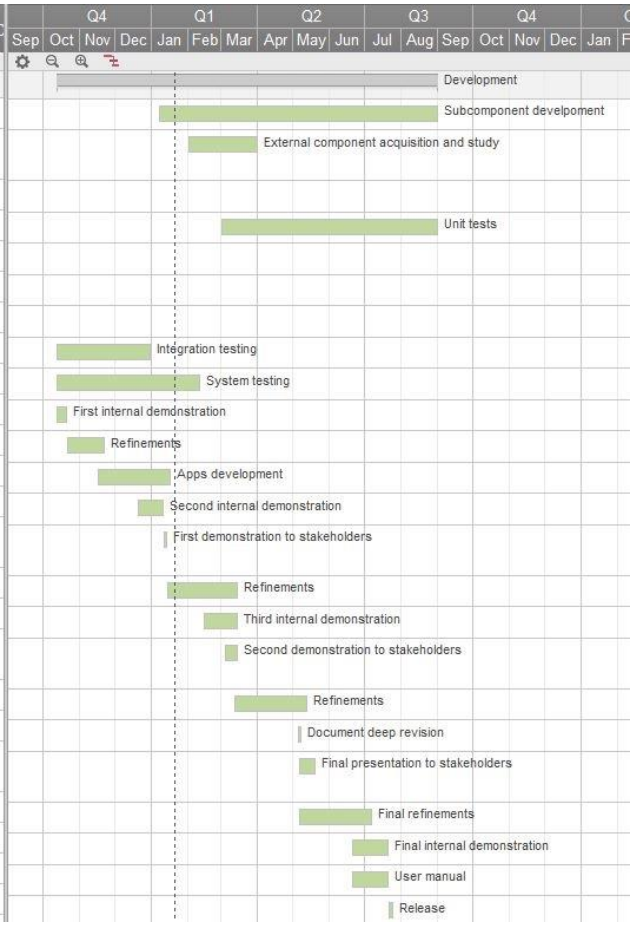


	Task Name	Start Date	End Date
1	**RASD**	**12/10/16**	**09/11/16**
2	Meeting with local governement	12/10/16	14/10/16
3	Identification of stakeholders	12/10/16	14/10/16
4	Definition of main functionalities	12/10/16	14/10/16
5	Initial requirements draft	12/10/16	14/10/16
6	Initial draft of activity diagrams	12/10/16	14/10/16
7	Identification of use cases	12/10/16	14/10/16
8	Initial mockups of UI	12/10/16	14/10/16
9	Meeting with local government	20/10/16	26/10/16
10	Refinement of functionalities	20/10/16	26/10/16
11	Refinement of assumptions	20/10/16	26/10/16
12	Refinement of requirements	20/10/16	26/10/16
13	Refinement of activity diagrams	20/10/16	26/10/16
14	Refinement of UI mockups	20/10/16	26/10/16
15	Meeting with stakeholders	20/10/16	26/10/16
16	Final requirements	27/10/16	28/10/16
17	Final assumptions	27/10/16	28/10/16
18	Final activity diagrams	27/10/16	28/10/16
19	Final UI mockups	30/10/16	02/11/16
20	Initial cost,schedule and effort estimation	30/10/16	02/11/16
21	Meeting with stakeholders	03/11/16	04/11/16
22	Revision of estimations	03/11/16	04/11/16
23	Meeting with stakeholders	03/11/16	04/11/16
24	Final refinements	06/11/16	09/11/16

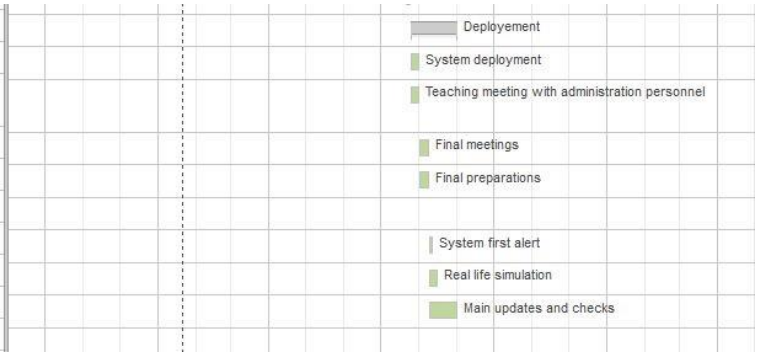


	Task Name	Start Date	End Date
25	**DD**	**10/11/16**	**06/01/17**
26	High Level draft of architecture	10/11/16	14/11/16
27	Identification of main components	10/11/16	14/11/16
28	Meeting with stakeholders	15/11/16	18/11/16
29	Revision of architecture	15/11/16	18/11/16
30	Refinements of components,identification of subcomponents	15/11/16	18/11/16
31	First draft of component interfaces	15/11/16	18/11/16
32	Refinement of UI mockups	15/11/16	18/11/16
33	Initial draft of algorithm design	19/11/16	25/11/16
34	Meeting with stakeholders	26/11/16	30/11/16
35	Final architecture design	26/11/16	30/11/16
36	Refinement of component interfaces	26/11/16	30/11/16
37	Refinement of UI mockups	26/11/16	30/11/16
38	Refinement of algorithm design	26/11/16	30/11/16
39	Meeting with stakeholders	01/12/16	01/12/16
40	Final UI Design	01/12/16	02/12/16
41	Final component interfaces	01/12/16	02/12/16
42	Final algorithm design	01/12/16	02/12/16
43	Meeting with stakeholders	01/12/16	02/12/16
44	Final refinements	03/01/17	06/01/17

	Task Name	Start Date	End Date
45	**Development**	**12/10/16**	**31/08/17**
46	Subcomponent develpoment	07/01/17	31/08/17
47	External component acquisition and study	01/02/17	31/03/17
48	Code inspection	15/02/17	31/08/17
49	Unit tests	01/03/17	31/08/17
50	Subcomponent integration		
51	Component development		
52	Component integration		
53	Integration testing	12/10/16	30/12/16
54	System testing	12/10/16	10/02/17
55	First internal demonstration	12/10/16	20/10/16
56	Refinements	21/10/16	21/11/16
57	Apps development	16/11/16	16/01/17
58	Second internal demonstration	20/12/16	10/01/17
59	First demonstration to stakeholders	11/01/17	13/01/17
60	Refinements	14/01/17	14/03/17
61	Third internal demonstration	14/02/17	14/03/17
62	Second demonstration to stakeholders	04/03/17	14/03/17
63	Refinements	12/03/17	12/05/17
64	Document deep revision	05/05/17	05/05/17
65	Final presentation to stakeholders	06/05/17	19/05/17
66	Final refinements	06/05/17	06/07/17
67	Final internal demonstration	20/06/17	20/07/17
68	User manual	20/06/17	20/07/17
69	Release	21/07/17	24/07/17

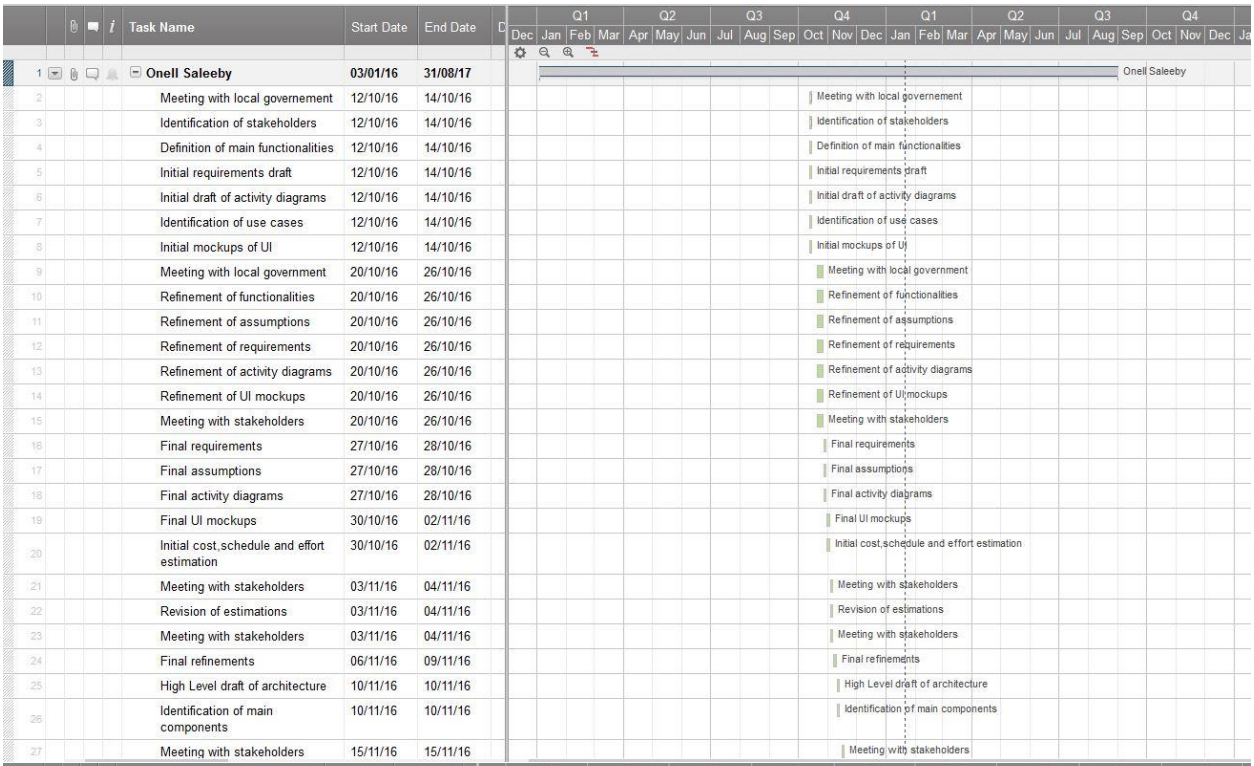


Deployement	**25/07/17**	**31/08/17**
System deployment	25/07/17	31/07/17
Teaching meeting with administration personnel	25/07/17	31/07/17
Final meetings	01/08/17	08/08/17
Final preparations	01/08/17	08/08/17
Start Up		
System first alert	09/08/17	10/08/17
Real life simulation	09/08/17	15/08/17
Main updates and checks	09/08/17	31/08/17

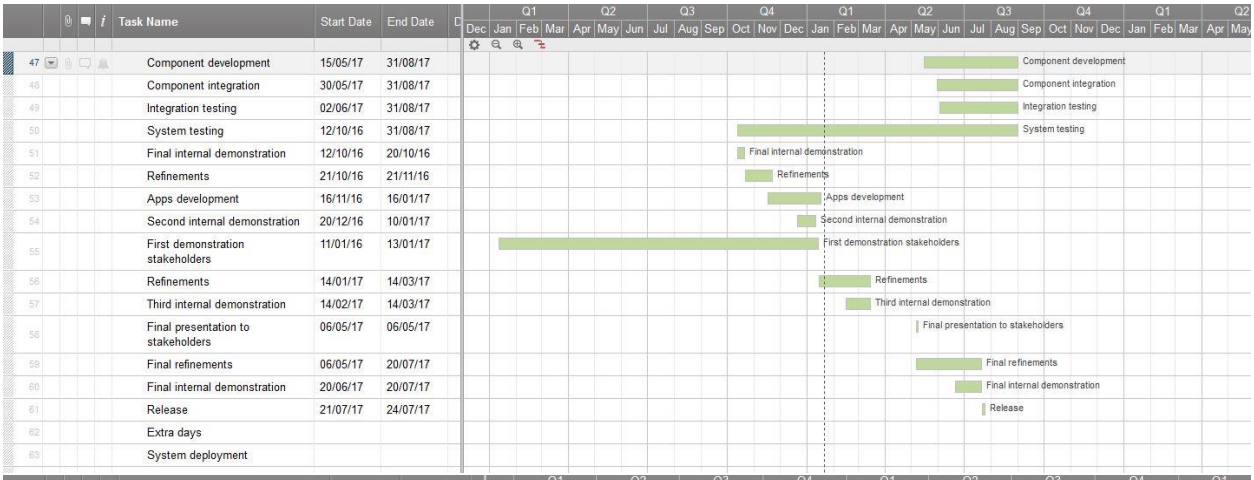


4. Resource Allocation

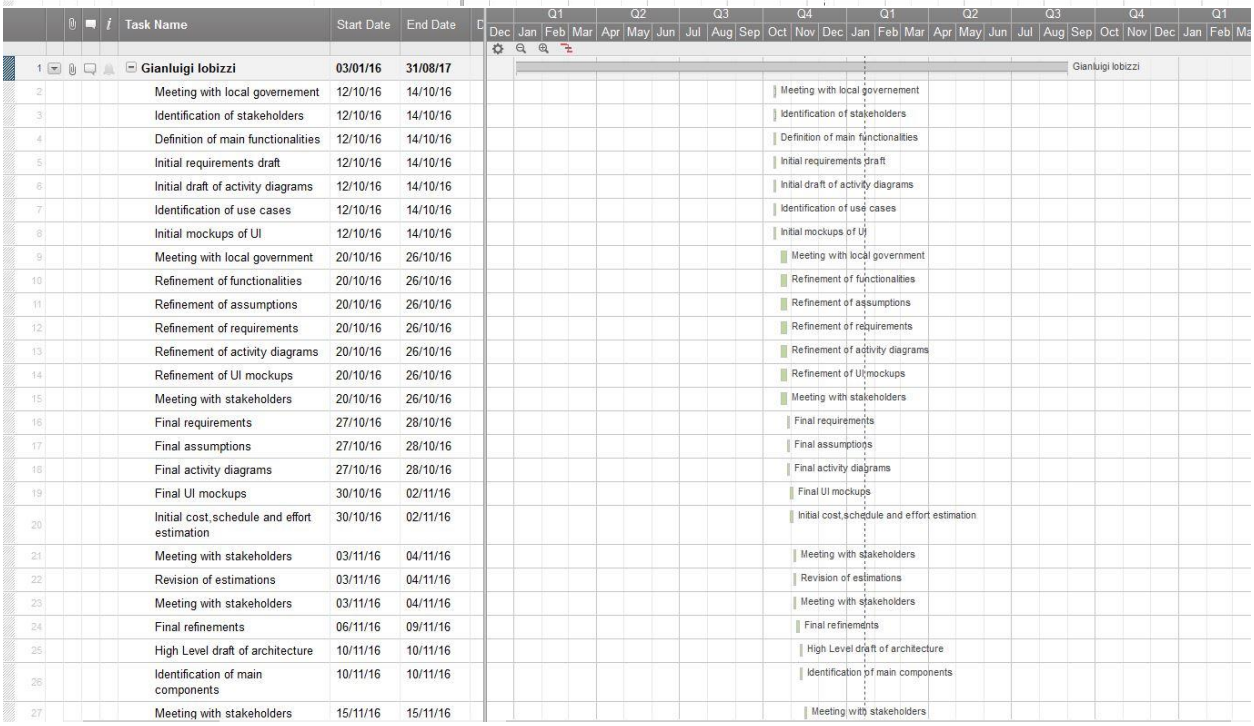
In this part, we're going to provide a general overview of how the tasks defined by the schedule in the previous section will be divided between the two members of the development team. For being accurate, we have split the schedule into 4 quarters, Q1 from October to December, Q2 from January march, Q3 from April to June, Q4 from July to September.



	Task Name	Start Date	End Date
1	**Onell Saleeby**	**03/01/16**	**31/08/17**
2	Meeting with local governement	12/10/16	14/10/16
3	Identification of stakeholders	12/10/16	14/10/16
4	Definition of main functionalities	12/10/16	14/10/16
5	Initial requirements draft	12/10/16	14/10/16
6	Initial draft of activity diagrams	12/10/16	14/10/16
7	Identification of use cases	12/10/16	14/10/16
8	Initial mockups of UI	12/10/16	14/10/16
9	Meeting with local government	20/10/16	26/10/16
10	Refinement of functionalities	20/10/16	26/10/16
11	Refinement of assumptions	20/10/16	26/10/16
12	Refinement of requirements	20/10/16	26/10/16
13	Refinement of activity diagrams	20/10/16	26/10/16
14	Refinement of UI mockups	20/10/16	26/10/16
15	Meeting with stakeholders	20/10/16	26/10/16
16	Final requirements	27/10/16	28/10/16
17	Final assumptions	27/10/16	28/10/16
18	Final activity diagrams	27/10/16	28/10/16
19	Final UI mockups	30/10/16	02/11/16
20	Initial cost,schedule and effort estimation	30/10/16	02/11/16
21	Meeting with stakeholders	03/11/16	04/11/16
22	Revision of estimations	03/11/16	04/11/16
23	Meeting with stakeholders	03/11/16	04/11/16
24	Final refinements	06/11/16	09/11/16
25	High Level draft of architecture	10/11/16	10/11/16
26	Identification of main components	10/11/16	10/11/16
27	Meeting with stakeholders	15/11/16	15/11/16
29	Refinemetns of components	15/11/16	15/11/16
30	First draft of algorithm design	15/11/16	15/11/16
31	Refinement of UI mockups	15/11/16	15/11/16
32	Initial draft if algorithm design	19/11/16	19/11/16
33	Meeting with stakeholders	26/11/16	26/11/16
34	Final architecture design	26/11/16	26/11/16
35	Refinement of component interfaces	26/11/16	26/11/16
36	Refinement of UI mockups	26/11/16	26/11/16
37	Final UI design	01/12/16	02/12/16
38	Final component Interfaces	01/12/16	02/12/16
39	Final algorithm design	01/12/16	01/12/16
40	Meeting with stakeholders	01/12/16	01/12/16
41	Final refinements	03/01/16	06/01/16
42	Subcomponent development	07/01/17	31/08/17
43	External component acquisition	01/02/17	31/03/17
44	Code inspection	15/02/17	31/08/17
45	Unit tests	01/03/17	31/08/17
46	Subcomponent integration	01/05/17	31/08/17
47	Component development	15/05/17	31/08/17
48	Component integration	30/05/17	31/08/17
49	Integration testing	02/06/17	31/08/17
50	System testing	12/10/16	31/08/17
51	Final internal demonstration	12/10/16	20/10/16
52	Refinements	21/10/16	21/11/16
53	Apps development	16/11/16	16/01/17
54	Second internal demonstration	20/12/16	10/01/17
55	First demonstration stakeholders	11/01/16	13/01/17

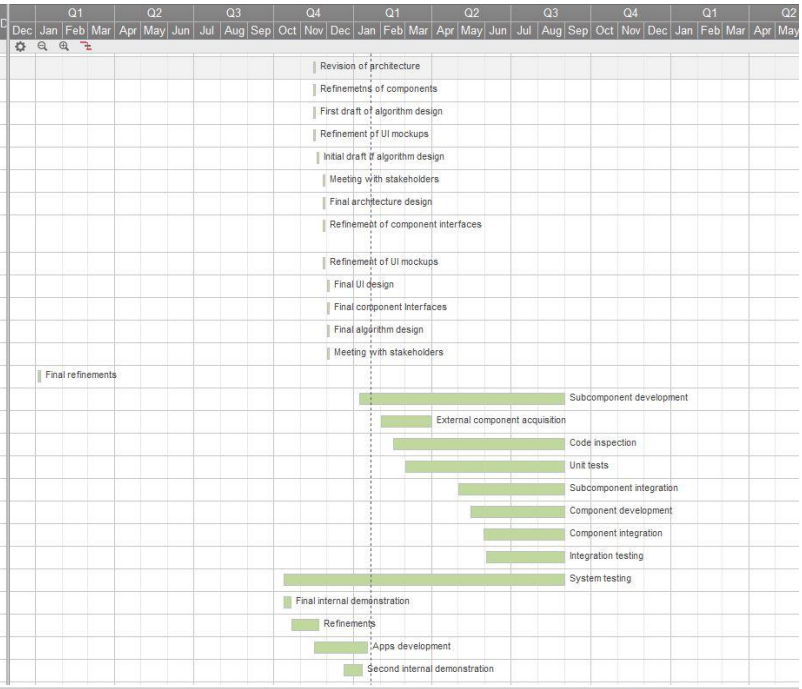


	Task Name	Start Date	End Date
47	Component development	15/05/17	31/08/17
48	Component integration	30/05/17	31/08/17
49	Integration testing	02/06/17	31/08/17
50	System testing	12/10/16	31/08/17
51	Final internal demonstration	12/10/16	20/10/16
52	Refinements	21/10/16	21/11/16
53	Apps development	16/11/16	16/01/17
54	Second internal demonstration	20/12/16	10/01/17
55	First demonstration stakeholders	11/01/16	13/01/17
56	Refinements	14/01/17	14/03/17
57	Third internal demonstration	14/02/17	14/03/17
58	Final presentation to stakeholders	06/05/17	06/05/17
59	Final refinements	06/05/17	20/07/17
60	Final internal demonstration	20/06/17	20/07/17
61	Release	21/07/17	24/07/17
62	Extra days		
63	System deployment		

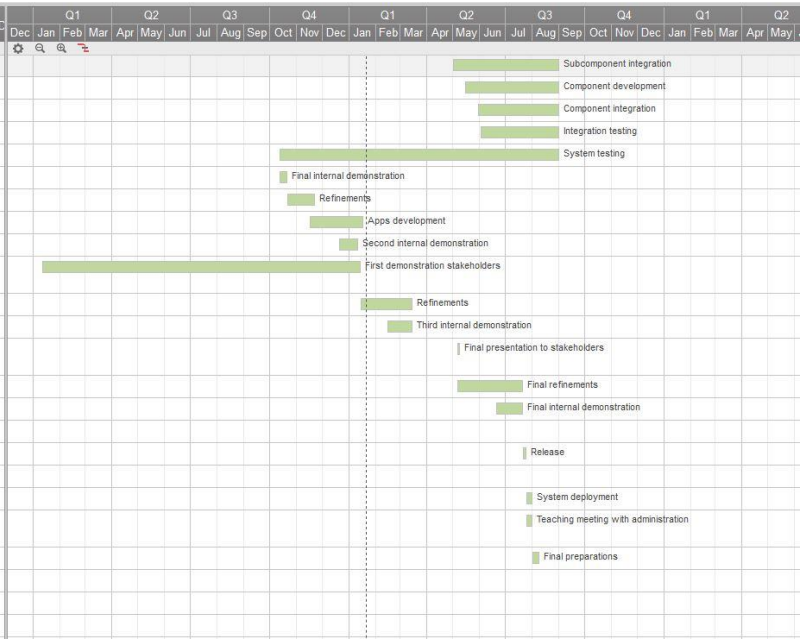


	Task Name	Start Date	End Date
1	**Gianluigi Iobizzi**	**03/01/16**	**31/08/17**
2	Meeting with local governement	12/10/16	14/10/16
3	Identification of stakeholders	12/10/16	14/10/16
4	Definition of main functionalities	12/10/16	14/10/16
5	Initial requirements draft	12/10/16	14/10/16
6	Initial draft of activity diagrams	12/10/16	14/10/16
7	Identification of use cases	12/10/16	14/10/16
8	Initial mockups of UI	12/10/16	14/10/16
9	Meeting with local government	20/10/16	26/10/16
10	Refinement of functionalities	20/10/16	26/10/16
11	Refinement of assumptions	20/10/16	26/10/16
12	Refinement of requirements	20/10/16	26/10/16
13	Refinement of activity diagrams	20/10/16	26/10/16
14	Refinement of UI mockups	20/10/16	26/10/16
15	Meeting with stakeholders	20/10/16	26/10/16
16	Final requirements	27/10/16	28/10/16
17	Final assumptions	27/10/16	28/10/16
18	Final activity diagrams	27/10/16	28/10/16
19	Final UI mockups	30/10/16	02/11/16
20	Initial cost,schedule and effort estimation	30/10/16	02/11/16
21	Meeting with stakeholders	03/11/16	04/11/16
22	Revision of estimations	03/11/16	04/11/16
23	Meeting with stakeholders	03/11/16	04/11/16
24	Final refinements	06/11/16	09/11/16
25	High Level draft of architecture	10/11/16	10/11/16
26	Identification of main components	10/11/16	10/11/16
27	Meeting with stakeholders	15/11/16	15/11/16

	Task Name	Start Date	End Date
28	Revision of architecture	15/11/16	15/11/16
29	Refinemetns of components	15/11/16	15/11/16
30	First draft of algorithm design	15/11/16	15/11/16
31	Refinement of UI mockups	15/11/16	15/11/16
32	Initial draft if algorithm design	19/11/16	19/11/16
33	Meeting with stakeholders	26/11/16	26/11/16
34	Final architecture design	26/11/16	26/11/16
35	Refinement of component interfaces	26/11/16	26/11/16
36	Refinement of UI mockups	26/11/16	26/11/16
37	Final UI design	01/12/16	02/12/16
38	Final component Interfaces	01/12/16	02/12/16
39	Final algorithm design	01/12/16	01/12/16
40	Meeting with stakeholders	01/12/16	01/12/16
41	Final refinements	03/01/16	06/01/16
42	Subcomponent development	07/01/17	31/08/17
43	External component acquisition	01/02/17	31/03/17
44	Code inspection	15/02/17	31/08/17
45	Unit tests	01/03/17	31/08/17
46	Subcomponent integration	01/05/17	31/08/17
47	Component development	15/05/17	31/08/17
48	Component integration	30/05/17	31/08/17
49	Integration testing	02/06/17	31/08/17
50	System testing	12/10/16	31/08/17
51	Final internal demonstration	12/10/16	20/10/16
52	Refinements	21/10/16	21/11/16
53	Apps development	16/11/16	16/01/17
54	Second internal demonstration	20/12/16	10/01/17



	Task Name	Start Date	End Date
46	Subcomponent integration	01/05/17	31/08/17
47	Component development	15/05/17	31/08/17
48	Component integration	30/05/17	31/08/17
49	Integration testing	02/06/17	31/08/17
50	System testing	12/10/16	31/08/17
51	Final internal demonstration	12/10/16	20/10/16
52	Refinements	21/10/16	21/11/16
53	Apps development	16/11/16	16/01/17
54	Second internal demonstration	20/12/16	10/01/17
55	First demonstration stakeholders	11/01/16	13/01/17
56	Refinements	14/01/17	14/03/17
57	Third internal demonstration	14/02/17	14/03/17
58	Final presentation to stakeholders	06/05/17	06/05/17
59	Final refinements	06/05/17	20/07/17
60	Final internal demonstration	20/06/17	20/07/17
61	User Manual		
62	Release	21/07/17	24/07/17
63	Extra days		
64	System deployment	25/07/17	31/07/17
65	Teaching meeting with administration	25/07/17	31/07/17
66	Final preparations	01/08/17	08/08/17
67			
68			
69			



5. Risk Management

In this section, we will discuss the various risks that might impede the progress of our project or cause (directly or indirectly) unwanted outcomes. We will be considering technical risks which are related to the design, architecture, and implementation of our system, as well as non-technical risks, be it political, financial or social risks.

Since our project aims to promote the use of electric cars, it's considered environment-friendly, and therefore in theory we should have the support of the people, as well as any organizations that promote environment cleanliness. However, many people today don't believe in the efficiency of electric cars, whether it's in terms of performance (compared to fuel-engine cars) or efficiency (distance to power ratio). Therefore, we must make sure that the cars use cutting-edge technology, guaranteeing the maximum efficiency possible while also providing the drivers with enough power to traverse long distances in a relatively short time.

Another risk that might arise resided in installing the charging stations. Since we want to provide our users with a great user experience, the number of charging stations should be adequate, and the distribution should be done in a way to guarantee the best results possible. The discount the users get when plugging the cars in charging stations after the trip is a good incentive, but we should also make sure that this is not causing inconveniences to the users. Therefore, as the system goes online and users start subscribing, we will have to study the locations where charging stations are most needed and increase their numbers accordingly if need be. Also, we must communicate with the local authorities and make sure that installing these stations won't cause them or the people any issues. So, studying the needs of the users and keeping the city government informed about our future plans should be enough to mitigate such risks.

Moreover, other electric car sharing services might be already operating/planning to invest in the area. However, we perceive this as more of a chance than a risk, because constructive competition should lead us, with careful planning, to earning the loyalty of our user base. Adopting good marketing strategies, making sure the application is user-friendly, and listening to the suggestions/demands of our users should take us a long way in this perspective.

Moving on to the technical side, one of the major issues that we should take care of is having clean, well-structured source code, especially at the early stages of the app development process. A big

problem faced by startup companies is having ill-structured code, which might produce the correct results for a while but cause problems on the long run. Such problems happen usually because of the tight schedules, which forces the development teams to rush, and produce less than perfect code: “Done on time is better than perfect”. While this is correct to some extent, this doesn’t mean that we should compromise when talking about the cleanliness and structure of the code, because the cost for detecting and fixing problems caused by bugs increases exponentially as a function of time. Employing various conde inspection techniques and holding regular code review meetings should help mitigate such risks. Also, while we encourage hiring fresh graduates as developers, we should always make sure that there are well-seasoned, trustworthy developers acting as team leaders, making sure that everything is on track.

This brings us to another point that we should consider which is flexibility. While it’s understandable that each developer can be specialized in a certain field (front-end, back-end, system architecture etc.), we should make sure that no one person is responsible, on his own, of a specific part. In other words, in case any person has to be absent for a few days, or even leave for another job, as the development field is quite flexible in terms of job offers, we should always be able to promptly fill the gap by another employee from our company, until we hire a suitable replacement.

Another important issue to consider is the use of outside resources/tools in our implementation. Naturally, we don’t want to re-invent the wheel. If there’s a trusted pre-existing tool/library that can help us achieve our goals, then by all means we should use it. The best of example of this is the maps API that we will be using. During the RASD and DD preparations, we decided to use the Google Maps API since it’s available for both Android and iOS, and it’s quite trustworthy in terms of completeness, features, and efficiency. The risk in this case would reside in the fact that the sources we’re getting these tools from might decide to cease the support of a specific library, or make it paid/increase the usage cost. So, to avoid such issues, only well-trusted sources that have an abundant number of users/clients should be employed in this sense. For example, it’s extremely unlikely that Google will introduce major changes to their API or make it unavailable for a certain platform, since the repercussions of such an action will be drastic.

Any database might crash or get corrupted for any number of reasons. Of course, our system should be able to recover from that. This can be done first and foremost by having redundant backups of the database for recovery purposes. Also, we should have a backup server that can be used in case of emergencies (primary server stopped working for some reason). Also, if needed, we can adopt a distributed system: for example, we can have a database used for user registration operations, another

database used for trips operations, and so on. This way, we can, to a certain degree, avoid major system failures that would prevent our users from using the application completely.

One final technical risk that we should be very aware of is scalability. As the number of our users increase, the number of requests to the server, the physical size of the database, the memory consumption, all of these will increase as well. Therefore, our systems should be able to handle expansions in order to adapt to such cases. This can be done by studying the variations of the number of users, and predicting how these numbers will change during a given period in the future. By doing this, we will be able to anticipate and act promptly, avoiding any problems.

As we go on with the implementation, the above-mentioned risks will be taken into account during each phase, and the list can be extended and edited as needed, and any efforts in this direction is welcomed because naturally, prevention is always better than detection and/or treatment.

6. Hours of work

Onell Saleeby

- 20/1: 2 hours
- 21/1: 3 hours
- 22/1: 1 hour

Bachar Senno

- 18/1: 2 hours
- 20/1: 3 hours
- 21/1: 2 hours
- 22/1: 3 hours

Gianluigi Iobizzi

- 17/1: 4 hours
- 18/1: 4 hours
- 19/1: 4 hours