POLITECNICO
MILANO 1863

Project Plan Document

Software Engineering 2: “myTaxiService”

Fioratto Raffaele, Longoni Nicolò
Politecnico di Milano

A.Y. 2015/2016
Version 1.0

February 2, 2016

Contents

1 Introduction **3**
1.1 Revision History . . . 3
1.2 Purpose . . . 4
1.3 Scope . . . 4
1.4 Definitions and Abbreviations . . . 4
1.5 Reference Documents . . . 5

2 Cost Estimation **6**
2.1 Function Points . . . 6
2.1.1 Brief introduction . . . 6
2.1.2 Internal Logic Files . . . 7
2.1.3 External Logic Files . . . 7
2.1.4 External Inputs . . . 8
2.1.5 External Outputs . . . 9
2.1.6 External Inquiries . . . 9
2.1.7 Recap . . . 9
2.2 COCOMO . . . 10
2.2.1 Brief introduction . . . 10
2.2.2 Lines of Codes . . . 10
2.2.3 Scale Drivers . . . 10
2.2.4 Cost Drivers . . . 12
2.2.5 Cost Driver Recap . . . 19
2.2.6 Effort computation . . . 19

3 Project Tasks and Schedule **21**

4 Resource and Task Allocation **23**

5 Project Risks **25**

Appendices **28**

A Tools **28**

B Hours of Work **29**

1 Introduction

1.1 Revision History

Version	**Date**	**Author(s)**	**Summary**
0.1	01/21/2016	Fioratto Raffaele	Document Creation
0.2	01/22/2016	Fioratto Raffaele	Initial Function Points and COCOMO sections implementation
0.3	01/23/2016	Fioratto Raffaele	Cost Estimation chapter completed
0.4	01/25/2016	Longoni Nicolò	Project Task and Schedule chapter completed
0.5	01/26/2016	Longoni Nicolò	Resource and Task chapter completed
0.6	01/27/2016	Longoni Nicolò	Introduction chapter completed
0.7	01/28/2016	Fioratto Raffaele	Risks written
0.9	02/01/2016	Fioratto Raffaele and Longoni Nicolò	Minor corrections
1.0	02/02/2016	Fioratto Raffaele and Longoni Nicolò	Final Version

1.2 Purpose

Project Plan document collects an investigation about all costs and risks of the overall project and not only the development phase.
This document includes various topics: first of all cost estimation which is a study about spendings in the project in terms of money, staff and other external efforts, besides development costs.
We continue with a Gantt diagram which is a working timetable that shows days assigned to complete specific tasks. After there is a schedule that explains the division of work between teammates.
We conclude with an analysis of risks, inclusive of probability and relevance for every possible risk listed. A way to avoid or solve these problematic situations is also presented.

1.3 Scope

In this document we'll analyze all the project things related to cost estimation and efforts. These evaluations are done through Function Point, a unit of measure that expresses the complexity of several activities related to software that are internal and external inputs, files and inquiries. With this data and the weight of every complexity level, we can calculate the UFP value to estimate the the overall cost of the project in FP.
Later we'll use COCOMO, an estimation model that estimates the effort, the lines of code written, driver scale and cost, project duration and people necessary to complete the work of a software.
Then a Gantt diagram will be presented to show, from the date when specifics have been given to us to project this software, how much time it is necessary to complete requested work, the subdivision of every project phase and the assignment of each part to a member of working group.
This document ends with a list of risks that could appear during planning matched with their probability to emerge and relevance. It is also presented a solution to prevent these inconveniences or in case this wasn't possible, a way to solve them.

1.4 Definitions and Abbreviations

- PPD: **P**roject **P**lan **D**ocument
- ITPD: **I**ntegration **T**est **P**lan **D**ocument
- DD: **D**esign **D**ocument
- RASD: **R**equirement **A**nalysis and **S**pecification **D**ocument
- FP: **F**unction **P**oint
- UFP: **U**nadjusted **F**unction **P**oint
- COCOMO: **CO**nstructive **CO**st **MO**del
- Sys Admin: System Administrator

- SF: **S**cale **F**actor

1.5 Reference Documents

- myTaxiService Project Specification Document
- myTaxiService RASD
- myTaxiService DD
- myTaxiService ITP
- COCOMO II Model Definition Manual[1]

[1] Available at: `http://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_modelman2000.0.pdf`

2 Cost Estimation

2.1 Function Points

2.1.1 Brief introduction

The Function Point estimation approach is based on the principle of extracting functions from a software, to classify them using a well defined set of classes and estimating their complexity.
This kind of estimation is extremely useful since it can be done at a very early stage of a project life-cycle, ideally after the implementation of the RASD. Moreover it can be used like a basis for performing a cost estimation using well-known models such as COCOMO (explained later).
This estimation is a single number called UFP that can be computed using a simple formula.
A high-level procedure that explains how to calculate this number is the following:

1. Classify each function of the software to one of these five possible classes called Function Types (explained in detail later):
   - Internal Logic Files
   - External Interfaces Files
   - External Inputs
   - External Outputs
   - External Inquiries
2. For each function a complexity is defined, it can be:
   - Low
   - Average
   - High
3. Calculate the UFP by using this formula:

   $$\sum_{f \in F,\ c \in C} ((\# \text{ of functions of type } f \text{ and complexity } c) \cdot (\text{weight for type } f \text{ and complexity } c))$$

   where $F = \{\text{ILF, ELF, EI, EO, EIQ}\}$ and $C = \{\text{Low, Average, High}\}$.
   Refer to this table to determine the proper weight for each type and complexity:

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

Table 1: UFP Complexity Weights

Further manipulation of the UFP can be done in order to use it in Cost Estimation Models such as COCOMO, but this will be explained later.

2.1.2 Internal Logic Files

The application should manage an internal database which is used to store information about the three categories of registered users: Passengers, Taxi Drivers, Sys Admins. For each of them, basic data is collected: name, surname, password, e-mail and so on. Thus we think that the complexity for this entities should be set to **Low**. On the other hand the system also maintains a history of Rides carried out throughout its life-cycle, since this entity is a bit complex than the previous ones, we set its complexity to **Average**.

ILF	Complexity	FP
Passengers	Low	7
Taxi Drivers	Low	7
Sys Admins	Low	7
Rides	Average	10
Total		31

Table 2: ILFs Table Recap

2.1.3 External Logic Files

The application should interact with a Maps service which provides information regarding city map and ETA of Rides. This interaction requires to understand some APIs and the need of parsing data coming from the Internet, so we set this function complexity to **Average**.
Another interaction is with an external Database containing various information about Cars and Drivers. This interaction is simpler than the previous one because Java has already well-defined interfaces to dialog with a Database, so we set the complexity to **Low**.

ELF	**Complexity**	**FP**
Maps Service	Average	7
Cars and Drivers Database	Low	5
Total		12

Table 3: ELFs Table Recap

2.1.4 External Inputs

The application has to deal with these external interactions coming from the users:

- *Login/logout*, these are simple operations (they are two distinct functions) so we can set their complexity to **Low**.
- *Passenger/Taxi Driver registration*, as for the previous functions, these are simple too, so their complexity is again **Low**.
- *Perform Request/Reservation*, these are the most complex operations since several components are involved and some of them are also external, so we set their complexity to **High**.
- *Accept Request from Taxi Driver/Passenger*, these operations involve a medium number of components, also they interact with the internal database in order to add relevant information to Ride history, thus the complexity is set to **Average**.
- *Modify Passenger's profile*, this is a simple operation, its complexity is **Low**.
- *Change Taxi Driver's status*, this operation requires to remove/add the Taxi to the proper queue, but nevertheless this is a fairly simple operation, so the complexity is **Low**.

EI	**Complexity**	**FP**
Login	Low	3
Logout	Low	3
Passenger Registration	Low	3
Taxi Driver Registration	Low	3
Perform Request	High	6
Perform Reservation	High	6
Accept Request from Taxi Driver	Average	4
Accept Request from Passenger	Average	4
Modify Passenger's Profile	Low	3
Change Taxi Driver's status	Low	3
Total:		38

Table 4: EIs Table Recap

2.1.5 External Outputs

The applications generate the following complex outputs to the users:

- *Propose Ride to Taxi Driver/Passenger*, this is a fairly simple operation once the overall elaboration and computation has been done on the back-end, so we set the complexity to **Low**.
- *Calculate ETA*, for this function a bunch of input data has to be collected from the environment (e.g. GPS positions), and an external service, which computes the actual result, must be contacted, so we set the complexity of this to **Average**.
- *Select Best Taxi for Incoming Ride*, this depends on chosen policies to manage queues in terms of assignment to city map and algorithm for taxi selection from a specific queue. We set the complexity of this function to **Average**.

EO	**Complexity**	**FP**
Propose Ride to Taxi Driver	Low	4
Propose Ride to Passenger	Low	4
Calculate ETA	Average	5
Select Best Taxi for Incoming Ride	Average	5
Total		18

Table 5: EOs Table Recap

2.1.6 External Inquiries

The application allows users to request information about themselves and other similar things, but all of them are simple to implement, so their complexity is set to **Low**.

EIQ	**Complexity**	**FP**
Show Passenger's Profile	Low	3
Show Taxi Driver's Profile	Low	3
Show Sys Admin's Profile	Low	3
Show ETA	Low	3
Total		12

Table 6: EIQs Table Recap

2.1.7 Recap

The following table summarizes the amount of FP for each type and the overall sum that corresponds to the final UFP value.

Function Type	**Value**
Internal Logic Files	31
External Logic Files	12
External Inputs	38
External Outputs	18
External Inquiries	12
Total (UFP)	111

Table 7: Final FPs recap with UFP calculation

2.2 COCOMO

2.2.1 Brief introduction

This estimation is achieved using a complex, non linear model that takes into account not only characteristics of a product, but also of people and processes.
Basically, this model estimates the effort needed to complete a project. To compute this number we need to define some concepts which will be used later in the actual calculation.
The equation to compute the effort is the following:

$$\text{Effort} = A \cdot \text{SLOC}^E \cdot \prod_{i=1}^{n} EM_i$$

Where $A = 2.94$ for COCOMO II.2000, SLOC is the amount of lines of code estimated to be written for implementing the software, E is explained in detail in Section 2.2.3 and EM is called effort multiplier which is defined for each Cost Driver (explained in Section 2.2.4), in COCOMO II.2000 the number of Cost Driver are 17 thus $n = 17$.

2.2.2 Lines of Codes

This datum can be determined using a simple conversion equation of this kind

$$\text{SLOC} = k \cdot \text{UFP}$$

Where k is a coefficient which is assigned given the programming language chosen for implementing the software, in our case the language is Java thus $k = 53$ $^{\text{SLOC}}/_{\text{UFP}}$.
Given this information, we can estimate our SLOC for the sotware which is:

$$SLOC = 53 \cdot 111 = 5,883 = 5.88\ KSLOC$$

This value will be used later for computing the effort.

2.2.3 Scale Drivers

The exponent E in the Effort equation is an aggregation of five *scale factors* (SF) that accounts for the relative economies or diseconomies of scale encountered for software

projects of different sizes.
If $E < 1.0$, the project exhibits economies of scale. If the product size is doubled, the project effort is less than doubled. The project productivity increases as the product size is increased.
If $E = 1.0$, economies and diseconomies of scale are in balance. This linear model is often used for small project cost estimation.
If $E > 1.0$, the project exhibits diseconomies of scale. This is generally because of two main factors: growth of interpersonal communication overhead and growth of large-system integration overhead.
E can be computed using this equation:

$$E = B + 0.01 \cdot \sum_{j=1}^{5} \mathrm{SF}_j$$

where $B = 0.91$ (for COCOMO II.2000).
Scale Factors are defined in the following 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
$\mathrm{SF_i}$:	6.20	4.96	3.72	2.48	1.24	0.00
FLEX:	Rigorous	Occasional relaxation	Some relaxation	General conformity	Some conformity	General goals
$\mathrm{SF_i}$:	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%)
$\mathrm{SF_i}$:	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
$\mathrm{SF_i}$:	5.48	4.38	3.29	2.19	1.10	0.00
	The estimated Equivalent Process Maturity Level (EPML) or					
PMAT:	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
$\mathrm{SF_i}$:	7.80	6.24	4.68	3.12	1.56	0.00

Table 8: Scale Factor Values, $\mathrm{SF_i}$, for COCOMO II Models

According to the table we can evaluate scale factors for our project:

- **Precedentedness**, it measures previous experiences we had with this kind of technologies. Since this is our first project using JEE and these development methodologies, we set it to **Low**.

- **Development flexibility**, its value express the degree of elasticity that we have during the development of the project. In particular it depends on how the software needs to be accurate with respect to pre-established requirements, with external interface specifications and so on. Since the original project specification presents this problem without going too much in detail, this may induce us to set this value to Extra High, however during the writing of the RASD, some requirements expressed the need to interact with some external components which have a fixed and already established interface. Thus, we decided to adjust this SF to a more fair value of **High**.
- **Risk resolution**, it reflects the amount of effort and budget expected to be spent during the development to seek for possible risks and their resolution that can occur anytime during the life-cycle of a software. These two problems have not been debated in any previous document until now. The PPD has a dedicated section that explains these kind of issues, so we can set this SF to **Nominal**.
- **Team cohesion**, it measures the overall capabilities of all actors (users, customers, developers, maintainers, etc.) in the development process to interact each other to carry out main objectives. Since in our case, the team is composed only by two members, that already worked together to other projects in the past, we set this value to **Very High**.
- **Process maturity**, it is an indicator that shows how the overall process of development is sophisticated and well-defined with the respect to some general criteria. Since precise instructions have been given to us before each phase of the project, we set it to **High** (CMM Level 3).

The results are recapped in the following table:

Scale Driver	**Selected Factor**	**Value**
Precedentness	Low	4.96
Development Flexibility	High	2.03
Risk Resolution	Nominal	4.24
Team Cohesion	Very High	2.19
Process Maturity	High	3.12
Total		16.60

2.2.4 Cost Drivers

The Cost Drivers called also Effort Multipliers are multiplicative factors that estimate the effort to complete the whole project. As already explained in Section 2.2.1, the COCOMO II model has 17 Cost Drivers, this section briefly explains all of them and, for each Cost Driver, the corresponding value for our project.

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. A failure of the system might result in some financial losses, however since its complexity is not exaggerated, almost all losses can be recovered easily, so we set this value to **Nominal**

RELY Descriptors:	Slight in-covenience	Low, easily recoverable losses	Moderate, easily recoverable losses	High financial loss	Risk to human life	
Rating Level	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	0.82	0.92	1.00	1.10	1.26	n/a

Table 9: RELY Cost Driver

Database Size (DATA) This Cost Driver attempts to capture the effect of large test data requirements made on product in development. The rating is determined by calculating $^D/_P$, the ratio of bytes in the testing database to SLOC program. Since we cannot determine precisely the size of our testing database we can only give an estimate based on what we have written in the ITPD, so we set this value to **Nominal**

DATA Descriptors:		Testing DB bytes/Pgm SLOC < 10	$10 \leq$ D/P ≤ 100	$100 \leq$ D/P $\leq$ 1000	D/P ≥ 100	
Rating Level	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	n/a	0.90	1.00	1.14	1.28	n/a

Table 10: DATA Cost Driver

Product Complexity (CPLX) It is divided into five areas:

- control operations
- computational operations
- device-dependent operations
- data management operations
- user interface management

Using the COCOMO II CPLX rating scale this is set to **Nominal**.

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

Table 11: CPLX Cost Driver

Developed for Reusability (RUSE) This cost driver accounts for additional effort needed to construct components intended for reuse on current or future projects. This concept is used extensively especially in the DD so we set the value of this Cost Driver to **Very High**.

RUSE Descriptors:		None	Across project	Across program	Across product line	Across multiple product lines
Rating Level	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	n/a	0.95	1.00	1.07	1.15	1.24

Table 12: RUSE Cost Driver

Documentation Match to Life-Cycle Needs (DOCU) The rating scale for this Cost Driver is evaluated in terms of project document suitability to its life-cycle needs. We can assume that the documents written till now are sufficient to clarify all the issues that might arise during the next development phases, so we set this rating scale to **Nominal**.

DOCU Descriptors:	Many life-cycle needs unconvered	Some life-cycle needs uncovered	Right-sized to life-cycle needs	Excessive for life-cycle needs	Very excessive for life-cycle needs	
Rating Level	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	0.81	0.91	1.00	1.11	1.23	n/a

Table 13: DOCU Cost Driver

Execution Time Constrains (TIME) It values the execution time constraint imposed upon a software system. The rating is expressed in percentage of available execution time expected to be used by the system or subsystem consuming resources. Since the system will run on dedicated servers, there are no particular constraints on the execution time. Thus we can set this Cost Driver to **Very Low**.

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 Level	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	n/a	n/a	1.00	1.11	1.29	1.63

Table 14: TIME Cost Driver

Main Storage Constraint (STOR) This rating represents the degree of main storage constraint imposed on a software or subsystem. As before, the usage of a dedicated system doesn't impose to us any particular constraint, so we can also set this Effort Multiplier to **Very Low**.

STOR Descriptors:			≤ 50% use of available storage	70% use of available storage	85% use of available storage	95% use of available storage
Rating Level	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	n/a	n/a	1.00	1.05	1.17	1.46

Table 15: STOR Cost Driver

Platform Volatility (PVOL) "Platform" here means the complexity of hardware and software (OS, DMBS, etc.) called to perform their tasks. We think that the platform shouldn't change too much over time, thus **Low** is the appropriate value for this Cost Driver.

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 Level	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	n/a	0.87	1.00	1.15	1.30	n/a

Table 16: PVOL Cost Driver

Analyst Capability (ACAP) Analysts are personnel who works on requirements, high-level design and detailed design. Since a lot of effort was spent in writing the RASD and the DD, we believe they capture all the needs and they were correctly studied, so we set this estimator to **High**.

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

Table 17: ACAP Cost Driver

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 efficiency, thoroughness, the ability to communicate and to cooperate. In our situation we chose the value of **High**.

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

Table 18: PCAP Cost Driver

Applications Experience (APEX) This rating depends on previous experiences of team members with applications used to develop the software, e.g. JEE. Since this is our first experience we set this to **Low**.

APEX Descriptors:	$\leq$ 2 months	6 months	1 year	3 year	6 years	
Rating Level	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.22	1.10	1.00	0.88	0.81	n/a

Table 19: APEX Cost Driver

Platform Experience (PLEX) This index expresses the capacity of the team to work with the platform created (hardware and software) that allows the final product to operate accurately. This is set to **Nominal** since our knowledge about the overall platform is about 1 year long.

PLEX Descriptors:	$\leq$ 2 months	6 months	1 year	3 year	6 years	
Rating Level	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.19	1.09	1.00	0.91	0.85	n/a

Table 20: PLEX Cost Driver

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. As for the previous Cost Driver this is set to **Nominal** too.

LTEX Descriptors:	$\leq$ 2 months	6 months	1 year	3 year	6 years	
Rating Level	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.20	1.09	1.00	0.91	0.84	n/a

Table 21: LTEX Cost Driver

Personnel Continuity (PCON) The rating scale is in terms of project annual personnel turnovers. In our case it is set to **Very Low**.

PCON Descriptors:	48% / year	24% / year	12% / year	6% / year	3% / year	
Rating Level	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.29	1.12	1.00	0.90	0.81	n/a

Table 22: PCON Cost Driver

Use of Software Tools (TOOL) Since no actual implementation is done, we can assume the use of a Java IDE such as Eclipse and a VCS of choice such as Git which can be easily integrated. So we set this value to **Nominal**.

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 Level	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.17	1.09	1.00	0.90	0.78	n/a

Table 23: TOOL Cost Driver

Multisite Development (SITE) Since the service is established in a city and involves the use of wideband electronic communications by means of smartphones, we set this Cost Driver to **High**.

SITE: Collocation Descriptors:	Interna-tional	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 communica-tion	Wideband elect. comm. occasional video conf.	Interactive multimedia
Rating Level	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.22	1.09	1.00	0.93	0.86	0.80

Table 24: SITE Cost Driver

Required Development Schedule (SCED) This rating measures the schedule constraints imposed on the project team developing the software. The ratings are defined in percentage of schedule stretch-outs or accelerations with respect to a nominal schedule for a project requiring a given amount of effort. There are no particular reasons to set this value different from **Nominal**.

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 Multipliers	1.43	1.14	1.00	1.00	1.00	n/a

Table 25: SCED Cost Driver

2.2.5 Cost Driver Recap

Our chosen values are recapped in the following table:

Cost Driver	**Selected Factor**	**Value**
Required Software Reliability	Nominal	1.00
Database Size	Nominal	1.00
Product Complexity	Nominal	1.00
Required Reusability	Very High	1.15
Documentation match to life-cycle needs	Nominal	1.00
Execution Time Constraint	Very Low	n/a
Main Storage Constraint	Very Low	n/a
Platform Volatility	Low	0.87
Analyst Capability	High	0.85
Programmer Capability	High	0.88
Application Experience	Low	1.10
Platform Experience	Nominal	1.00
Language and Tool Experience	Nominal	1.00
Personnel Continuity	Very Low	1.29
Usage of Software Tools	Nominal	1.00
Multi-site Development	High	0.93
Required Development Schedule	Nominal	1.00
Product		0.99

2.2.6 Effort computation

Using the values computed in the previous sections we can now calculate the effort. Given that $A = 2.94$, $EAF = 0.99$ and $E = 0.91 + 0.01 \cdot 16.60 = 1.08$ the effort is

$$\text{Effort} = 2.94 \cdot 0.85 \cdot 5.88^{1.08} = 16.93 \text{ PM}$$

We can also calculate the duration of a project in terms of months required to complete it with this equation:

$$\text{Duration} = 3.67 \cdot (\text{Effort})^{SE} = 3.67 \cdot 16.93^{0.31} = 8.82 \text{ Months}$$

Where $SE = D + 0.2 \cdot (E - B)$ and $D = 0.28$. Given the values of Effort and Duration for the project we can compute the number of required people “N” is:

$$\mathrm{N}_{\text{people}} = \left\lceil \frac{\text{Effort}}{\text{Duration}} \right\rceil = \lceil 1.92 \rceil = 2 \text{ People}$$

3 Project Tasks and Schedule

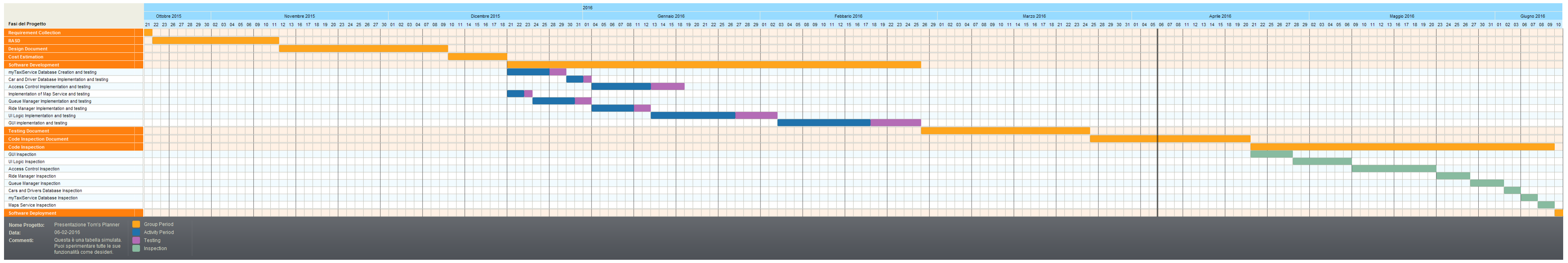

Figure 1: Gantt diagram of the project

In this section is presented the Gantt diagram of our project.
On 21st October it has been set the meeting with the company that required us to project myTaxiService application. During this day we talked about features and goals of the software to develop. We estimated the collection of this information requires only one day to be completed.
From the 22nd of October to the 11th of November we worked on RASD: using information gather during the meeting, we prepared the document to present it before the deadline.
Following Gantt diagram, it is possible to see that we continued our project writing the DD and like the previous document, we guaranteed the applicants to complete it in more or less a month.
After these two documents, we thought it was the time to write the Project Plan document, keeping one month the time necessary to conclude and to present it.
Software development is the next task, we estimate this phase will last approximately two months taking into consideration code writing part and the testing one.
Composition of Testing document is the next step and it will last four weeks. It will contains all mistakes present in developed code.
We continue with Code Inspection document that will show how various components have to test each other. Two months are necessary to complete inspection phase.
At the end, we will deploy the software to the company on 10th June.
The overall time requested to complete the project is eight months as we estimated.
As you can see from Gantt diagram, weekends are not considered working day (due to working laws).

4 Resource and Task Allocation

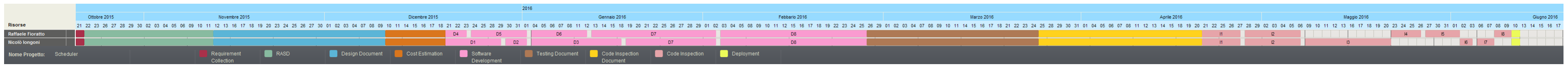

Figure 2: Scheduler

The diagram above shows work division between two teammates and it is a direct consequence of previous Gantt diagram.
Most of the work has been done at the same time by the two. This because, like it is shown in Gantt diagram, initial meeting, deployment phase and document writings are impossible to divide into sub tasks, in fact first two have to be done by both teammates due to their relevant importance, whereas for the last one it is unworkable to give assignements from the start.
Because of the complexity of development, inclusive of testing, inspection parts and previous reasons exposed in precedent paragraph, code writing and inspections are the tasks fragmented and assigned to the two teammates. A letter and a number are allocated to every sub task: a `D` is given to software development assignments, whereas a `I` to inspection tasks.
Some sub assignments are carried out simultaneously due to their complexity and need to be perfectly completed, others can be done separately by two teammates.
Now two lists follow, explaining the meaning of every letter - number couple.

Deployment Phase and Testing

- D1: myTaxiService Database Creation and testing
- D2: Car and Driver Database implementation and testing
- D3: Access Control Implementation and testing
- D4: Implementation of Map Service and testing
- D5: Queue Manager Implementation and testing
- D6: Ride Manager Implementation and testing
- D7: UI Logic Implementation and testing
- D8: GUI implementation and testing

Inspection Phase

- I1: GUI Inspection
- I2: UI Logic Inspection
- I3: Access Control Inspection
- I4: Ride Manager Inspection
- I5: Queue Manager Inspection
- I6: Cars and Drivers Database Inspection
- I7: myTaxiService Database Inspection
- I8: Maps Service Inspection

5 Project Risks

In this section risks for the project, their relevance in terms of reliability, availability of the system and associated recovery actions are defined. For each risk, a probability of occurrence and relevance are given. Concerning probability possible values are:

- Low
- Moderate
- High

Whereas for relevance, it can be one of the following possible values in order:

- Negligible
- Marginal
- Serious
- Critical
- Catastrophic

Risk	**Probability**	**Relevance**
Requirement volatility over time	Low	Serious
Problem with team communication or subversive team members	Low	Serious
Performance lack of external components (Databases, etc.)	Low	Marginal
Delays over expected deadline	High	Critical
Change of Regulations (e.g. Taxi policy, Car and Driver licenses, etc.)	Low	Serious
Competitors	Moderate	Serious
User acceptance (e.g. cumbersome and difficult to use interface, lack of responsiveness, etc.)	Moderate	Catastrophic
Bankruptcy	Moderate	Catastrophic
Integration Testing Failure	Low	Critical
Downtime	Moderate	Critical
Scalability issues	Moderate	Serious
Security issues (e.g. loss of data caused by system hacking)	Moderate	Critical

Table 26: Summary of Project's risks

Next table illustrates possible actions in order to avoid or recover risks defined in the previous table, whenever possible, actions are designed to avoid risks proactively rather than reactively when they happen.

Risk	**Strategy**
Requirement volatility	This can't be proactively avoided in a definitive and fixed way, but can be mitigated designing reusable and extensible software.
Team issues	Assigning tasks and responsibilities clearly to each project member. Try to maintain a fair and relaxed environment among members, in order to keep them highly motivated.
Performance lack of external components	Consider either to investigate the reasons for poor performance or to buy another software component.
Delays over expected deadline	Release a version of the software with missing features and then release an updated version with (hopefully) all the functionalities implemented.
Change of Regulations	Perform a thoroughly study on current regulations in order to take them into account accordingly, however this may not be sufficient to avoid this risk completely.
Competitors	Develop marketing plans (electronic advertising such as banners, publicity campaigns, etc.) in order to increase attractiveness of our product with respect to the others.
User acceptance	Design carefully graphic interface to have a cutting-edge user experience, which encourages the acceptance.
Bankruptcy	Carefully analyze costs and avoid delays, as well design a reliable system, so to reduce as much as possible needs to maintain the software (bug fixing, testing, patch development, etc.).
Integration Testing Failure	Write carefully the DD and implement the component accordingly trying to follow as close as possible to the specification. Perform tests as early as possible.
Downtime	This risk can be avoided by using some sort of redundancy at hardware or software level
Scalability issues	Carefully design and implement the software to avoid this risk, accordingly test under heavy load conditions.

Risk	**Strategy**
Security issues	Use defensive programming as much as possible, never “trust” users, plan proper penetration tests to put under pressure the system, analyze results and if necessary, make changes.

Table 27: Risks avoidance strategies

Appendices

A Tools

- Document written in LaTeX
- Basic MiKTeX 2.9.5721 64-bit – `http://miktex.org/`
- Texmaker 4.5 – `http://www.xm1math.net/texmaker/index.html`
- draw.io – `https://www.draw.io/`
- tomsplanner – `http://www.tomsplanner.it/`

B Hours of Work

- Raffaele Fioratto: 12 hours
- Nicolò Longoni: 12 hours