Project Plan
Version 1.0

Losio Davide Francesco, Luchetti Mauro, Mosca Paolo

February 1, 2016

Contents

1 — Introduction

This document aims to clearly stating the Project plan for the developing of MyTaxiService application. The developers team is composed by

- Losio Davide Francesco
- Luchetti Mauro
- Mosca Paolo

The document will be split up in the section stated below

1. Estimate of effort and cost, this will be done by the applying Function Points method to estimate the project size and then COCOMO to evaluate the tasks for the project and their schedule.
2. Resource allocation where all the tasks and resources will be clearly assigned to all the members of the group. Actual availability and needs of any member will be taken into account, and merged with the project contingencies.
3. Risk analysis, analysis of their relevance and the associated recovery and monitoring actions.

2 Effort and cost estimation

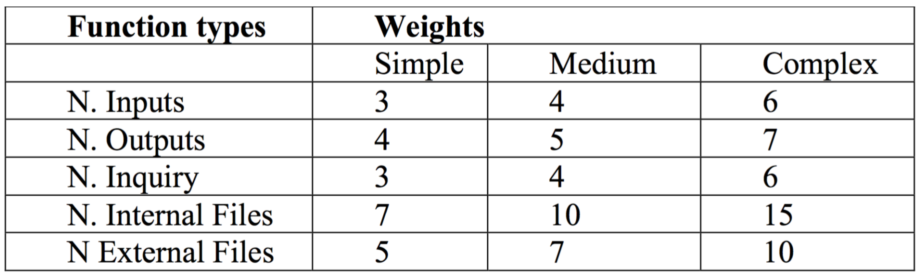
2.1 Functional points

2.1.1 Brief Explanation

We have chose Functional Point technique to evaluate the application dimensions basing on the functionalities, at the and of the overall analysis this will be implemented in SLOC estimation, that will be calculated with the so-called COCOMO method. The analyzed function are taken from DD, and RASD previously edited by the same team that is redacting this document. Functionalities are split in

- Internal Logic Files: These are homogeneous set of data used and managed by the application.
- External Interface Files: These are homogeneous sets of data used by the application but generated and maintained by other applications.
- External Inputs: These are elementary operations useful to elaborate data coming form the external environment.
- External Outputs: These are elementary operations that generate data for the external environment.
- External Inquiries: These are elementary operations that involve input and output without significant elaboration of data from logic files.

Functional points analysis will bind the right weights to all the individuated functions starting from the table stated below.



Function types	**Weights**		
	Simple	Medium	Complex
N. Inputs	3	4	6
N. Outputs	4	5	7
N. Inquiry	3	4	6
N. Internal Files	7	10	15
N External Files	5	7	10

2.1.2 Functional Points estimation

Internal Logical File

Driver On the driver side we need to store:

- Driver specific information: which can be considered of low complexity, because they only are personal and work related simple information. **7 FP**
- Queue: system need to store information and statistic about the queue situation, moreover this data has to be accessed, written and read several different times and retrieved from different sources. We can assume an average complexity. **10 FP**

Passenger On the passenger side we need to store:

- Passenger specific information: which can be considered of low complexity, because they only are personal simple information. **7 FP**

On both side

- Login data: which are a subsection of user information. We can consider them of low complexity because of their small size. Furthermore they will have a similar implementation with a huge code re-use, so we can consider them a single unit, not differentiated for passengers and drivers. **7 FP**
- Common ride info: we have to store information about the rides due to information retrieving facilities. Common data have to be split up between the two type of users, and each of them has to be capable to see only the data in which it is interested. so we can consider the structure of Average complexity. **10 FP**
- Zone: MTS need to archive some geographic and statistical information, to do is job. We can assume a low complexity for this kind of logical file, since despite of their big volume they have to remain basically unchanged for long time. **7 FP**

So in total we have about **48 FP** for this section.

External Interface File

Since MTS has to interface with the Google Maps API system, we keep in account about the navigation data received by this external system. We can consider them of High complexity, since they carry several different information both about positioning and routing. **10 FP**

So in total since this is the only external system providing data to MTS we have about **10 FP** for this section.

External Inputs

Driver On the driver side we have:

- Request accept/reject: here is requested a simple low complexity input. **3 FP**
- Feedback: Driver has to release a feedback on the passenger. This can be considered as a simple operation. **3 FP**
- Availability: we can consider this of average complexity, here gps information are needed. furthermore GPS involves both the Device and the application. **4 FP.**

Passenger On the passenger side we have:

- Feedback: Passenger has to release a feedback on the driver. This can be considered as a simple operation. **3 FP**
- Request insert/change input: here are requested different data, like: position, addresses, type of reservation so we can consider this of avg complexity. **2 x 4 FP**
- Request delete input: deleting a request is a simple operation that require a few input to be completed so we can consider this of low complexity. **3 FP**

On both Side common operation to the two user are:

- User login input: which can be considered of low complexity, because only few data are needed. **3 FP**
- User edit/create new user: both can be considered of average complexity, since we have to involve either user manager component, and dbms component. **2 X 4 FP**

So in total we have about **35 FP** for this section.

External Outputs

Driver On the driver side we have:

- Request forwarding: which can be considered of average complexity, because of the computation that have to be done for the driver selection. **5 FP**
- Queue status update: this is a periodical operation that requires to recalculate the driver position in the Taxi Queue. This has to be done each time a driver exit from the queue for several different reason. we can consider this operation of strictly high complexity. **7 FP**
- General operation confirmations: due to the pop-up acknowledgment. Low complexity output. **3 FP**
- Availability request: also here a low complexity output can do the job. **3 FP**

On both Side common operation to the two user are:

- Ride warn: Users has to be warn of the incoming rides appointment. This is a simple operation. **2 x 3 FP**

So in total we have about **24 FP** for this section.

External Inquiries

Driver On the driver side we have:

- Display queue position: position an information always available, his computation is transparent to his inquiry, given that we can consider this as a low complexity inquiry. **3 FP**

On both side common operation to the two user are:

- Display account: this can be considered as a simple inquiries, as it only have to display available data that hardly change during time. **3 FP**
- Display schedule: In this case, information are pretty static, not so many and easy to retrieve. We consider this operation of low complexity. **3 FP**
- Display Gps information: we can consider this operation of medium complexity, as it has to involve the device application/hardware component, predisposed for this feature. **4 FP**
- Display last feedback: this can be considered as a average complexity operation, since it could have to couple with data that have to be retrieved, either about the ride or the driver associated to the feedback. In order to achieve this goal, different file have to be accessed. **4 FP**

So in total we have about **21 FP** for this section.

2.1.3 Total

Categories	Sum
Internal Logical Files	48 +
External Interfaces Files	10 +
External Inputs	35 +
External Outputs	24 +
External Inquiries	17 =
Total:	135

The line of code estimation is done by using the usual formula, that is

$$LOC = AVC \cdot Number\ of\ FP$$

by using 46 as AVC parameter:

$$135 \cdot 46 = 6210 \text{ LOC.}$$

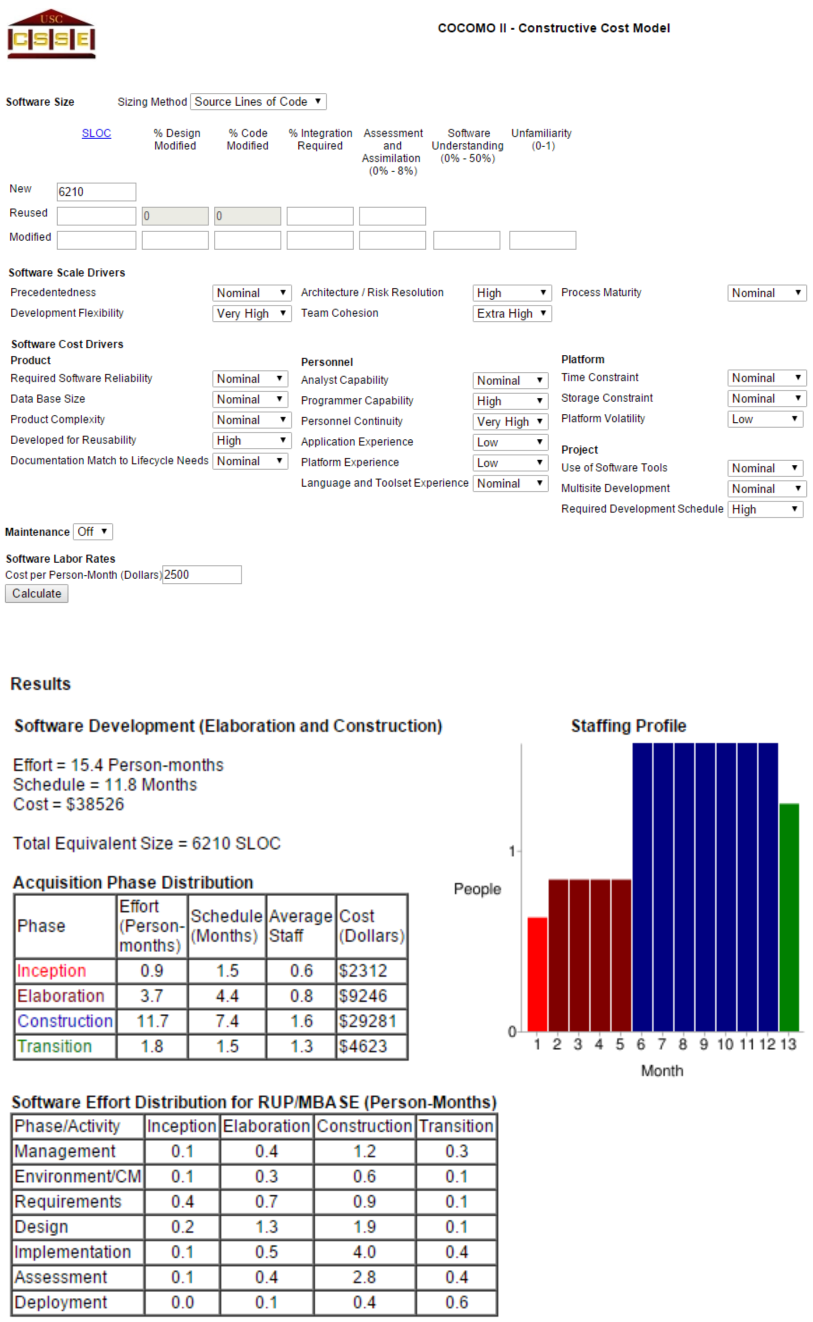
parameters for the calculus are taken from here:
http://www.qsm.com/resources/function-point-languages-table.

2.2 COCOMO

Mainly we have chosen the COCOMO effort evaluation method, because of his well known robustness. Furthermore since our company leak of a portfolio big enough to perform a calculus based on precedents it was the only available choice. Actually we have decided for the COCOMO II model, since this is the reviewed and corrected version. Here FP are used as a basis to estimate the size of the project in SLOC and then COCOMO method is applied to estimate the effort exploiting the SLOC estimation.

Below is reported the estimation done with the help of:

http://csse.usc.edu/tools/COCOMOII.php



COCOMO II - Constructive Cost Model

Software Size Sizing Method: Source Lines of Code

	SLOC	% Design Modified	% Code Modified	% Integration Required	Assessment and Assimilation (0% - 8%)	Software Understanding (0% - 50%)	Unfamiliarity (0-1)
New	6210						
Reused		0	0				
Modified							

Software Scale Drivers

Precedentedness	Nominal	Architecture / Risk Resolution	High	Process Maturity	Nominal
Development Flexibility	Very High	Team Cohesion	Extra High		

Software Cost Drivers

Product		Personnel		Platform	
Required Software Reliability	Nominal	Analyst Capability	Nominal	Time Constraint	Nominal
Data Base Size	Nominal	Programmer Capability	High	Storage Constraint	Nominal
Product Complexity	Nominal	Personnel Continuity	Very High	Platform Volatility	Low
Developed for Reusability	High	Application Experience	Low	**Project**	
Documentation Match to Lifecycle Needs	Nominal	Platform Experience	Low	Use of Software Tools	Nominal
		Language and Toolset Experience	Nominal	Multisite Development	Nominal
				Required Development Schedule	High

Maintenance Off

Software Labor Rates
Cost per Person-Month (Dollars) 2500

Calculate

Results

Software Development (Elaboration and Construction)

Effort = 15.4 Person-months
Schedule = 11.8 Months
Cost = $38526

Total Equivalent Size = 6210 SLOC

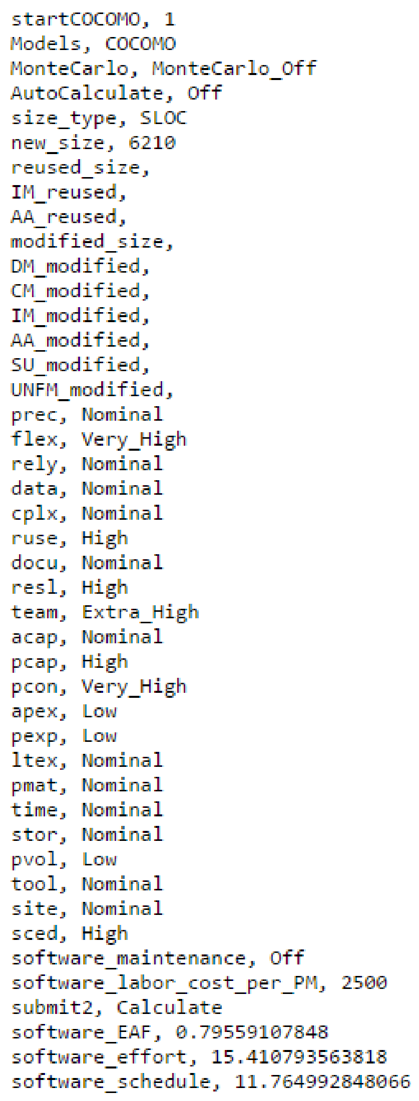
Acquisition Phase Distribution

Phase	Effort (Person-months)	Schedule (Months)	Average Staff	Cost (Dollars)
Inception	0.9	1.5	0.6	$2312
Elaboration	3.7	4.4	0.8	$9246
Construction	11.7	7.4	1.6	$29281
Transition	1.8	1.5	1.3	$4623

Software Effort Distribution for RUP/MBASE (Person-Months)

Phase/Activity	Inception	Elaboration	Construction	Transition
Management	0.1	0.4	1.2	0.3
Environment/CM	0.1	0.3	0.6	0.1
Requirements	0.4	0.7	0.9	0.1
Design	0.2	1.3	1.9	0.1
Implementation	0.1	0.5	4.0	0.4
Assessment	0.1	0.4	2.8	0.4
Deployment	0.0	0.1	0.4	0.6

It follows the output file generated by the application, carrying out all the settings:



```
startCOCOMO, 1
Models, COCOMO
MonteCarlo, MonteCarlo_Off
AutoCalculate, Off
size_type, SLOC
new_size, 6210
reused_size,
IM_reused,
AA_reused,
modified_size,
DM_modified,
CM_modified,
IM_modified,
AA_modified,
SU_modified,
UNFM_modified,
prec, Nominal
flex, Very_High
rely, Nominal
data, Nominal
cplx, Nominal
ruse, High
docu, Nominal
resl, High
team, Extra_High
acap, Nominal
pcap, High
pcon, Very_High
apex, Low
pexp, Low
ltex, Nominal
pmat, Nominal
time, Nominal
stor, Nominal
pvol, Low
tool, Nominal
site, Nominal
sced, High
software_maintenance, Off
software_labor_cost_per_PM, 2500
submit2, Calculate
software_EAF, 0.79559107848
software_effort, 15.410793563818
software_schedule, 11.764992848066
```

2.2.1 Comments on the chosen Driver

Below are stated the main motivation for the COCOMO parameters chosen.

Scale Drivers

- Precedentedness: We have set this value to low, since it is the measure of the development experience on the chosen development environment. Actually this is our first JavaEE application, so it seems quite reasonable to set this Scale Drivers to low.
- Development flexibility: Flexibility has to be high, as we have never couple with this kind of application, so we really don't know what to expect. Sure thing, is that we are going to do everything in the book, but we think it will be better to be conservative. As result we opted for a very high value.
- Risk resolution: As reported below, we have done a meticulous risk analysis so we opted to set this value to normal. Indeed we always prefer to be conservative.
- Team cohesion: As team, we have already works in several different project and we know each other very well. Only to report some proofs of this:
  - We have developed in team Escape From The Alien In Outer Space Online Game
  - *http://www.chrisportline.com/*
  - *http://www.pizzeriadasportogirasole-nerviano.it/home.html*
  - And many others works

  so it seems quite reasonable to set this driver to extra high.
- Process maturity: This was evaluated around the 18 Key Process Area (KPAs) in the SEI Capability Model. Because of the goals were consistently achieved these values will be set to normal

Cost Drivers

- Required Software Reliability: We have chosen to remain in an average setting. Indeed our application doesn't require any particular measure with respect to any other application.
- Data Base Size: Also here we have assumed to use the capacity of any other average application.
- Product Complexity: Set to high according to the nominal COCOMO II CPLEX rating scale.
- Required Reusability: MTS is designed with a SOA architecture, either for use or reuse. API of the main functionality will be available, so it is substantially designed to allow high reusability. High is the value we choose.

- Documentation match to life-cycle needs: This driver its suitability set to nominal since each aspect of our system has been described in the RASD or in DD.
- Execution Time Constraint: In our case this parameter is not relevant so it is reasonable to set it as nominal
- Main Storage Constraint: We don't have any particular storage constraints, we set this value to low.
- Platform Volatility: The application platform shouldn't change too often, or rather we have forecast to keep it as stable as possible. so this value is set to low.
- Analyst Capability: Actually this our first analysis study, so we think, not being presumptuous, to set our skill evaluation to low.
- Programmer Capability: Once again, we think that our CV talk for us, so we decide to set this value on high.
- Application Experience: Being our first javaEE project this value is equal to low.
- Platform Experience: As above, for the same motivations we chose low.
- Language and Tool Experience: Here we have decided to keep in account of our java SE knowledge, so we set the value to nominal.
- Personnel continuity: we set this value to high, since we are capable to work at least 8 our per day, almost every day.
- Usage of Software Tools: We actually planning to use JavaEE, gitHub and several different frameworks to speed-up development so we set this to nominal.
- Multisite development: we set this driver to nominal, as we actually prefer to avoid a massive multisite development, but we keep in account that we are capable to work on our own, also on high distance site and obviously in a more or less parallel way.
- Required development schedule: As this is our first approach to a project of this dimension we have preferred to produce a well documented development schedule. So we set this value to high.

For each clarification on drivers meaning, we refer, once again to the official manual:

http://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_modelman2000.0.pdf

3 — Task identification

This chapter provides a list of the main tasks of the whole project. All the necessary documents, all the deadlines associated with them, are taken into account in this section. Furthermore, it is shown the related schedule for each of the tasks individuated.

3.1 Project Tasks

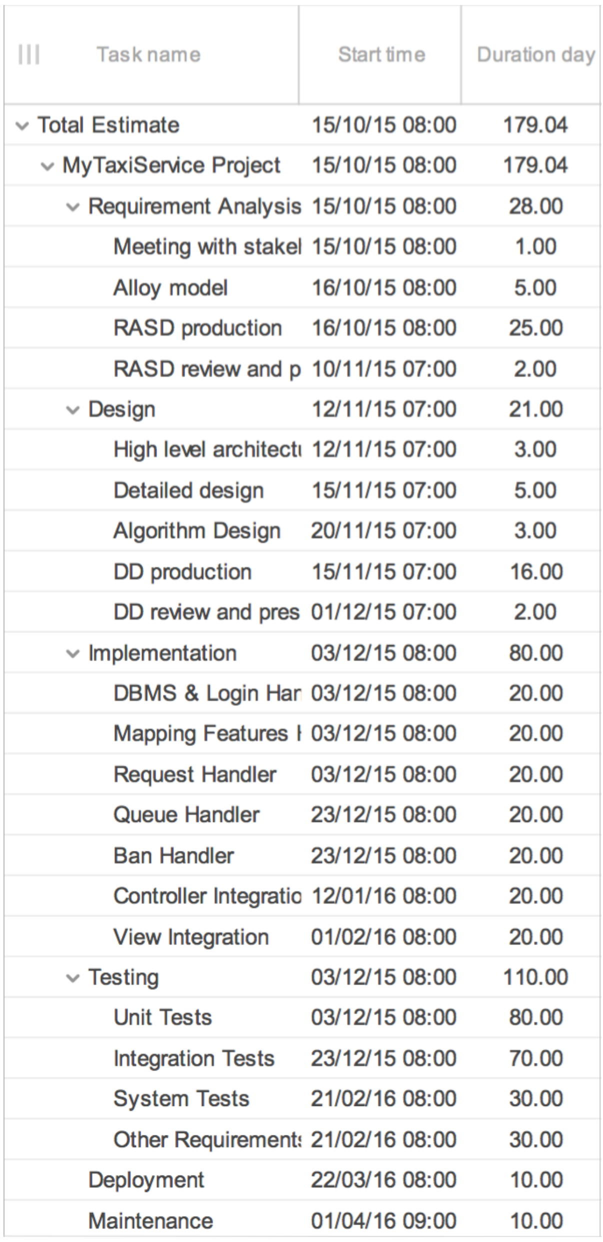
For each tasks, a possible fragmentation to subtasks can be considered:

1. **Requirement Analysis**
   (a) Meeting with stakeholders (Interview)
   (b) Model of the world (Alloy model)
   (c) RASD production
   (d) RASD review and presentation
2. **Design**
   (a) High level architecture
   (b) Detailed design structure
   (c) Algorithm design
   (d) DD production
   (e) DD review and presentation
3. **Implementation**
   (a) DBMS & Login Handler
   (b) Mapping Features Handler
   (c) Request Handler
   (d) Queue Handler
   (e) Ban Handler
   (f) Controller
   (g) View

4. **Testing**
    (a) JUnit tests over modules
    (b) Integration Tests
    (c) System Tests
    (d) Security Test
    (e) Usability Test
5. **Deployment**
6. **Maintenance**

3.2 Tasks schedule

The schedules for the various parts of the project are been introduced taking into account the project documentation deadlines provided:

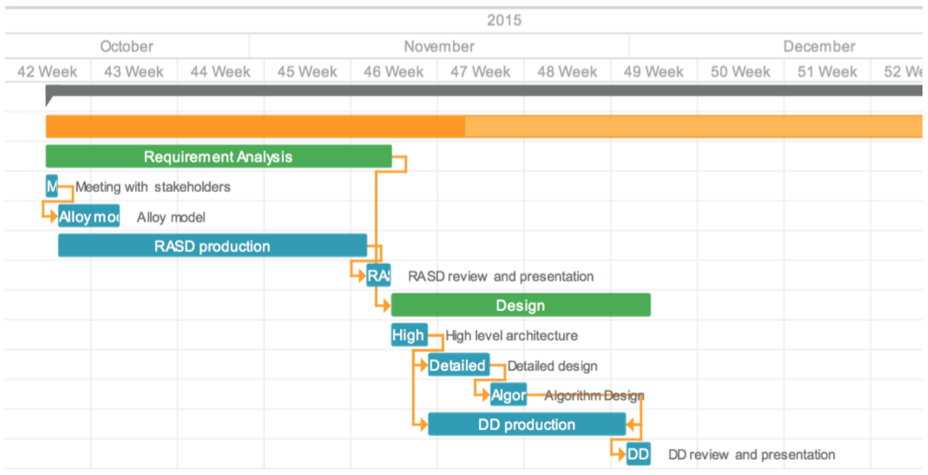


Task name	Start time	Duration day
Total Estimate	15/10/15 08:00	179.04
MyTaxiService Project	15/10/15 08:00	179.04
Requirement Analysis	15/10/15 08:00	28.00
Meeting with stakel	15/10/15 08:00	1.00
Alloy model	16/10/15 08:00	5.00
RASD production	16/10/15 08:00	25.00
RASD review and p	10/11/15 07:00	2.00
Design	12/11/15 07:00	21.00
High level architect	12/11/15 07:00	3.00
Detailed design	15/11/15 07:00	5.00
Algorithm Design	20/11/15 07:00	3.00
DD production	15/11/15 07:00	16.00
DD review and pres	01/12/15 07:00	2.00
Implementation	03/12/15 08:00	80.00
DBMS & Login Har	03/12/15 08:00	20.00
Mapping Features I	03/12/15 08:00	20.00
Request Handler	03/12/15 08:00	20.00
Queue Handler	23/12/15 08:00	20.00
Ban Handler	23/12/15 08:00	20.00
Controller Integratio	12/01/16 08:00	20.00
View Integration	01/02/16 08:00	20.00
Testing	03/12/15 08:00	110.00
Unit Tests	03/12/15 08:00	80.00
Integration Tests	23/12/15 08:00	70.00
System Tests	21/02/16 08:00	30.00
Other Requirement	21/02/16 08:00	30.00
Deployment	22/03/16 08:00	10.00
Maintenance	01/04/16 09:00	10.00

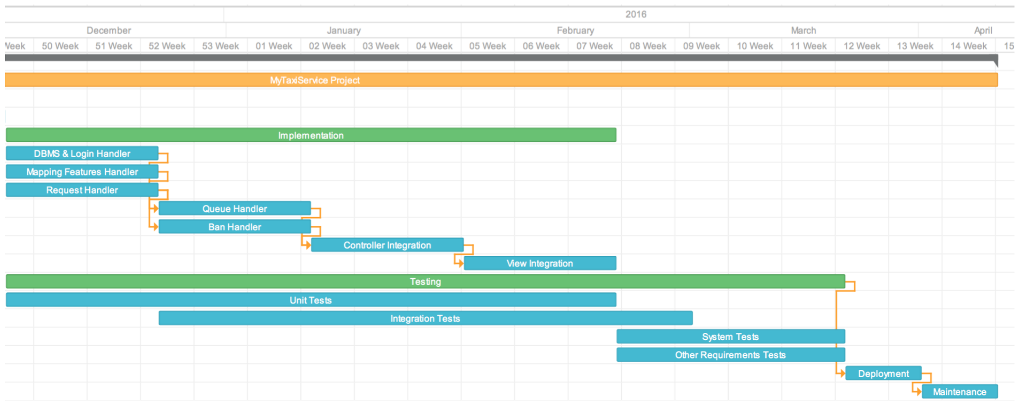
3.3 Gantt Chart

For having a better overview of the tasks and the related time associated to each of them, a Gantt chart has been provided (because of his dimensions it has been split in two parts):

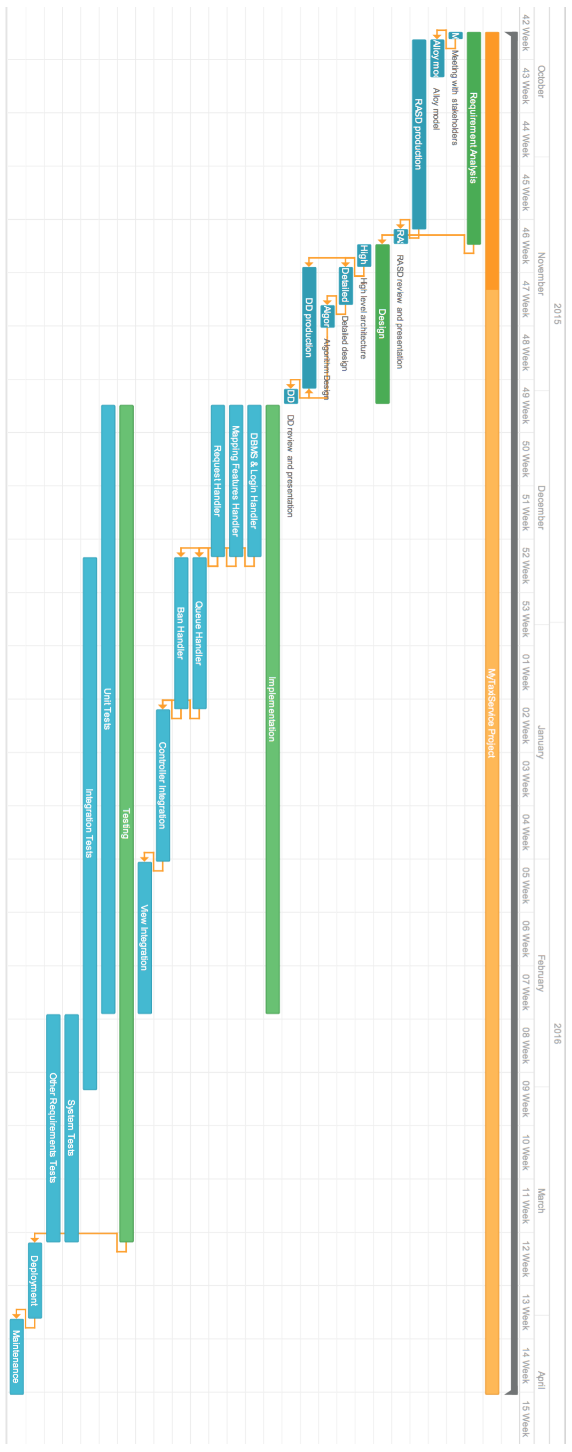
Study phase



Development phase



For having a complete overview of the chart:



4 — Resources allocation

As stated in the introduction of this document, the team working on the MyTaxiService system is composed by three members:

- Losio Davide Francesco
- Luchetti Mauro
- Mosca Paolo

Team members can be considered as resources of the project. In this chapter it is wanted to assign at each task individuated before a corresponding person. The tasks division is the one used above:

4.1 Requirement analysis

Task	**Resource**	**Motivation**
Stakeholders interview	All team members	In this phase there is the need to have the better understanding of the future requirements possible
Alloy model	Davide Losio	Fair job division
RASD production	All team members	Because of the document dimension, different parts can be assigned to the different team members
RASD review and presentation	All team members	Having all the members working on the revision of this document allows the better cohesion in the future phases

4.2 Design

Task	**Resource**	**Motivation**
High level architecture	Paolo Mosca	Fair job division
Detailed design structure	Davide Losio	Fair job division
Algorithm design	Mauro Luchetti	Fair job division
DD production	Sequentially: • Paolo Mosca • Davide Losio • Mauro Luchetti	Because of the sequentially nature of the three precedent tasks, after each completion, the related team member adds the proper part to the DD document
DD review and presentation	All team members	Having all the members working on the revision of this document allows the better cohesion in the future phases. Each of them can check and understand the parts done by the others

4.3 Implementation

Task	**Resource**	**Motivation**
DBMS & Login Handler	Davide Losio	Fair job division
Mapping Features Handler	Paolo Mosca	Fair job division
Request Handler	Mauro Luchetti	Fair job division
Queue Handler	Mauro Luchetti	This component is directly connected with the previous implemented one by the same member. This can ensure a better work-flow continuity
Ban Handler	Davide Losio	This component is directly connected with the previous implemented one by the same member. This can ensure a better work-flow continuity
Controller	Paolo Mosca, Davide Losio	The controller is composed by two similar component. Two team member can done the task in parallel
View	Paolo Mosca, Mauro Luchetti	The view is composed by two similar component. Two team member can done the task in parallel

4.4 Testing

Task	Resource	Motivation
JUnit tests over modules	All team members	Each team member should dedicate part of his time in doing tests over his specific component
Integration Tests	Paolo Mosca in the first phase	The other two members are employed to perform the implementation of other components. After completion, integration tests must proceed in parallel with implementation and they involve part of each member's time
System Tests	All team members	This tests are done when the implementation phase is over. All the members are available and can work together
Security Test	All team members	This tests are done when the implementation phase is over. All the members are available and can work together
Usability Test	All team members	This tests are done when the implementation phase is over. All the members are available and can work together

4.5 Deployment

Task	Resource	Motivation
Deployment	All team members	At this point each team member should have a clear idea of all the parts of the project and can partecipate at the distribution phase

4.6 Maintenance

Task	Resource	Motivation
Maintenance	All team members	Every possible maintenance action, because of the complete project knowledge of each member, can be handled by who is available

5 — Risks analysis

In this chapter Risks will be analyzed and will be provided some solutions.

Risk	**Probability**	**Impact**
Estimates are inaccurate	Moderate	Critical
Dependencies are inaccurate	low	Critical
Activities are missing from scope	Moderate	Serious
Stakeholders become disengaged	Low	Catastrophic
Architecture lacks flexibility	Medium	Serious
Technology components have security vulnerabilities	Moderate	Catastrophic
Delays to required infrastructure	Moderate	Serious
requirements have huge changes	Low	Critical
Architecture is infeasible	Low	Critical
Design is not fit for purpose	Low	Marginal
Components aren't interoperable	Moderate	Catastrophic
Components are over-engineered	Moderate	Serious

- **Estimates are inaccurate:** this might happen in case of wrong estimation of users. In this case probably hardware won't be able to support a huge number of users or in the otherwise powerful infrastructures might be an unnecessary cost.
  - **Solution:** an accurate estimation during the project planning, costs and prevision of the success, rewards. It would be a better hardware design expandable.
- **Dependencies are inaccurate:** a bad specification of requirements during the first phase might involve a wrong components development this will cause bad interaction among components and might cause bugs.
  - **Solution:** well done planning and specification understanding with team and stakeholders interaction.
- **Activities are missing from scope:** some main scopes might be misunderstood. This could cause a different usage of some functionalities.
  - **Solution:** starting by interacting with the stakeholders and among team components to make an informal draft of what the application must do which component do what.

- **Stakeholders become disengaged:** if stakeholders ignore project communications, the assignment quality probably get worse, a guide lines absence might cause a huge schedule delay as well as might cause inaccurate work or different functionalities from what expected.
  - **Solution:** close partnership with stakeholders even if it doesn't depend from the developing team. The second should make sure of the stakeholders seriousness and establish a safe relantionship with the first.
- **Architecture lacks flexibility:** if the architecture is incapable of supporting change requests and needs to be reworked.
  - **Solution:** a well done project at the begin and during the whole developing to make possible extensible components.
- **Technology components have security vulnerabilities:** if an accurate analysis or specific test aren't done in order to avoid bugs, exploits could be used to steal sensitive data or to cause software malfunction.
  - **Solution:** well security planning must be done during the whole time project.
- **Delays to required infrastructures:** this might cause a schedule delay as well as an inability to respect the requirements.
  - **Solution:** be sure to have the required infrastructures before starting the related activity.
- **Requirements have huge changes:** if during the project the organization or the stakeholders changes the requirements a huge part of the project must be redone.
  - **Solution:** be sure of the requires accuracy.
- **Architecture is infeasible:** the architecture is hard to implement, excessively costly or doesn't support the requirements.
  - **Solution:** during the design and development must ensure the reliability and that the components have a feaseabile cost.
- **Design is not fit for purpose:** the design is underestimated so it is bad projected.
  - **Solution:** during the software developing design must be implemented well during the entire project time, in particular way views must be done carefully.
- **Components aren't interoperable:** if the components aren't able to cooperate the entire application might not work appropriately.
  - **Solution:** interfaces between components must be created and tested during the whole project developing.

- **Components are over-engineered:** A component might bloated with unneeded functionality and design features.
  - **Solution:** components might be chosen and developed narrowly with clear and well specific tasks.

6 — Appendix

6.1 Used Tools

We used the following tools to make the ITP document:

- LYX 2.1: to redact and format the document
- GANTT PRO: to produce the Gantt chart. Available at: *https://ganttpro.com*

6.2 Time Spent

- For redact, correct and review this document we spent almost 10 hours per person.