Politecnico di Milano
A.A. 2015-2016
Software Engineering 2: “myTaxiService”

Project Plan

LuizaBentivoglio, Michele Cantarutti

2 February 2016

Summary

1. INTRODUCTION **4**

2. PROJECT ESTIMATION..**4**
2.1 SIZE ESTIMATION..4
2.2 EFFORT AND COST ESTIMATION…………..7

3. TASKS………………………..**17**

4. RESOURCE ALLOCATION..**18**

5. RISKS………………………..**19**
5.1 PROJECT RISKS...19
5.2TECHNICAL RISKS………..19
5.3BUSINESS RISKS………..19

1. Introduction

This document is meant to be a project plan for the myTaxiService project. The first part of the document will focus on the estimation of the project size (by applying Function Points) and of the effort and cost (by applying COCOMO). Next, we'll identify the tasks of our project and their schedule, while allocating the resources to each one of them. Lastly, we'll define the risks of the project.

2. Project estimation

2.1 SIZE ESTIMATION

To estimate the size of our software, we're going to use the Function Points approach. Function points measure software size and, by definition, a function point is a unit of measurement to express the amount of business functionality a software provides to a user. We retrieved the functionalities our software must provide from our RASD and then we evaluated the complexity of each one of them. Moreover, we distinguished five categories of functionalities, sticking with the standard FP approach, as in the model laid by Allan Albrecht, as follows:

-**External Input**: elementary operation to elaborate data coming form the external environment;

-**External Output**: elementary operation that generates data for the external environment (it usually includes the elaboration of data from logic files);

-**External Inquiry**: elementary operation that involves input and output (without significant elaboration of data from logic files).

-**Internal Logical File** (**ILF**): homogeneous set of data usedand managed by the application;

-**External Interface File** (**EIF**): homogeneous set of data used by the application but generated and maintained by other applications;

As for the points given to each functionality, we're once again sticking with the original model of the FP approach, as shown in the table below:

Function types	Complexity		
	Simple	Medium	Complex
External Inputs	3	4	6
External Outputs	4	5	7
External Inquiry	3	4	6
ILF	7	10	15
EIF	5	7	10

Now that we have defined the estimation model we intend to use and that we have specified what categories of functionalities our system provides, we can pass on to the proper estimation analysis. To do so, we went through each category, and for each category we analyzed the functionalities that belong to it, by rating the complexity of each one of them. Thus, we assigned the amount of points we felt appropriate for each functionality, based on its complexity.

-Internal Logical File (**ILF**): the application stores information about users, taxis, queues, zones and users' requests. All of these entities have a simple structure composed of a small number of fields, apart from the users' requests which have quite a few fields. Therefore we think it should be necessary to adopt medium complexity for the requests, whereas the other entities should have a simple complexity. We get 4x 7 +1x10 points, which add up to 38 FPs, concerning ILFs.

-External Interface File (**EIF**): there is only one operation that belongs to this category, and that is the payment, which will involve an interaction with an external system, that is a bank. The fact that interactions with an external system can be difficult to manage, coupled with the fact that transferring money requires high security, leads us to deem this operation a very complex one. Therefore we get 10 FPs.

-**External Input**: the application interacts with the users as follows:

• Login/logout: these are simple operations, so we can adopt the simple weight for them. 2 x 3 = 6 FPs

• Sign up:this is a simple operation, so we can adopt the simple weight. 1 x 3 = 3 FPs

• Making a request/reservation: these operations involve quite a few entities and imply the creation of a new one, therefore, they're highly complex: 2 x 6 = 12 FPs

• Cancelling a request/reservation: these operations are of medium complexity because the system also needs to notify other actors. We get 2 x 4 = 8 FPs

• Modifying a reservation: this operation is of medium complexity for the same reason as the operations above, because it involves notifying other actors. Therefore we get 1 x 4 = 4 FPs

•Taxi drivers can confirm their availability. This is a simple operation. 1 x 3 = 3 FPs

• Taxi drivers can accept calls. This operation is simple, so 1 x 3 = 3 FPs

-**External Output**:

• The system can notify the users about a modification or cancellation of a request. This is a simple operation, therefore we get 1 x 4 = 4 FPs

-**External Inquiries**: the application allows the users to request information about their account, in particular:

• The passengers can view his/her profile and his/her history of requests and reservations. Retrieving this information is a simple information, so the cost is 2 x 3 = 6 FPs

Total number of FPs: the total number of Function Points adds up to 97 points. If we had historical data about previous projects, we could estimate the effort necessary for this project, based on the number of function points. Now we're going to make another estimate of our project using another approach, that is COCOMO II.

2.2EFFORT AND COST ESTIMATION: COCOMO II

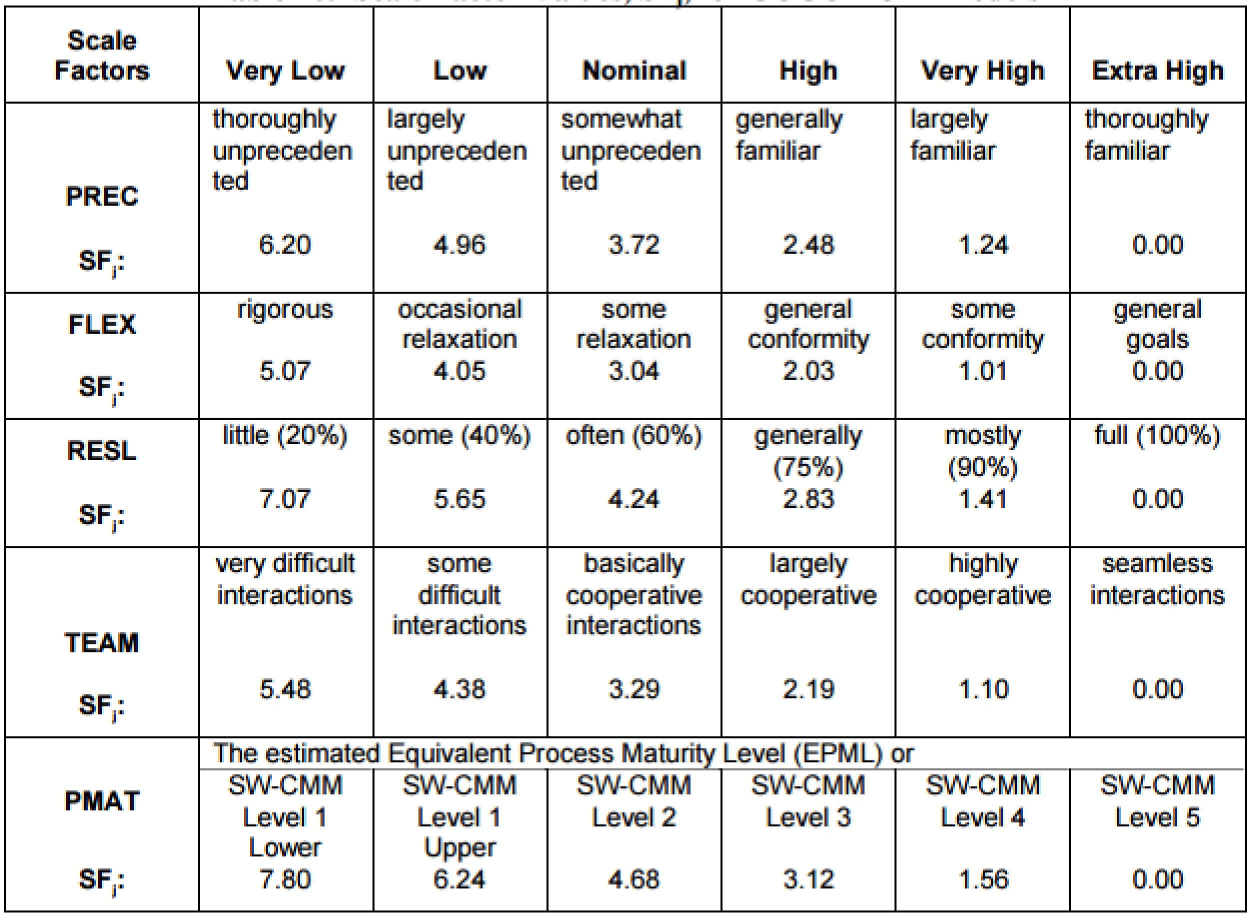
2.2.1 Brief Introduction

This estimate is achieved through a complex, nonlinear model that takes in account thecharacteristics of the product but also of people and process.
All the tables used in this analysis have been taken from COCOMO II, Model Definition.

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

2.2.2 Scale Drivers

Table 10. Scale Factor Values, SF_j, for COCOMO II Models



Scale Factors	Very Low	Low	Nominal	High	Very High	Extra High
PREC	thoroughly unprecedented	largely unprecedented	somewhat unprecedented	generally familiar	largely familiar	thoroughly familiar
SF_j:	6.20	4.96	3.72	2.48	1.24	0.00
FLEX	rigorous	occasional relaxation	some relaxation	general conformity	some conformity	general goals
SF_j:	5.07	4.05	3.04	2.03	1.01	0.00
RESL	little (20%)	some (40%)	often (60%)	generally (75%)	mostly (90%)	full (100%)
SF_j:	7.07	5.65	4.24	2.83	1.41	0.00
TEAM	very difficult interactions	some difficult interactions	basically cooperative interactions	largely cooperative	highly cooperative	seamless interactions
SF_j:	5.48	4.38	3.29	2.19	1.10	0.00
PMAT	The estimated Equivalent Process Maturity Level (EPML) or					
	SW-CMM Level 1 Lower	SW-CMM Level 1 Upper	SW-CMM Level 2	SW-CMM Level 3	SW-CMM Level 4	SW-CMM Level 5
SF_j:	7.80	6.24	4.68	3.12	1.56	0.00

These values are evaluated according to the following table:

• **Precedentedness:**
reflects the previous experience of the organization with this type ofproject. We only had one previous experience in software engineering and it wasn't very similar to this one. Therefore we think we didn't have much experience from before and the value for this scale driver is low.

• **Development flexibility:**
It reflects the degree of flexibility in the development process. The customer set the general specifications without going too much into detail.For this reason thisvalue will be high.

• **Risk resolution:**
Reflects the extent of risk analysis carried out. We carried out a single risk analysis, thus we think the value is nominal.

• **Team cohesion:**
Reflects how well the development team know each other and work together. Even though we worked as a team for the first time, we feel like we managed to work together well and we never had any major communication problems. Plus, we had similar schedules, which allowed us to be able to work together often. In conclusion, we believe the value of this scale driver is high.

• **Process maturity:**
Reflects the process maturity of the organization. We can't make an accurate computation for this value, but since we, as a team, always manage to achieve our goals within the deadlines, we will set this value to high

The results are resumed in the following table:

Scale Driver	**Factor**	**Value**
Precedentedness	Low	4.96
Development Flexibility	High	2.03
Risk Resolution	Nominal	4.24
Team Cohesion	High	2.19
Process Maturity	High	3.12

Total: 16.54

2.2.2 Cost Drivers

• Required Software Reliability:

This is the measure of the extent to which the software must perform its intended function over a period of time.
Software failures don't have critical consequences so this parameter is set to low, because this software simply manages taxis and not something critical like for example medical equipment.

Table 17. RELY Cost Driver

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

• Data Base Size:

This measure attempts to capture the effect large data requirements have on product development. It translates the effects that large data have in our application. Our application only manages simple data about users, taxis and queues, therefore, because this is not a very complex application, we think we should use the nominal value.

Table 18. DATA Cost Driver

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

• Product Complexity:

Set to nominal according to the new COCOMO II CPLEX rating scale.

Table 20. CPLX Cost Driver

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

• **Required Reusability:**
This cost driver accounts for the additional effort needed to construct components intended for reuse on the current or future projects. This effort is consumed with creating more generic design of software, more elaborate documentation, and more extensive testing to ensure components are ready for use in other applications. We'll set this to high because we intend to make a simple application, thus it won't be difficult to make it as modular as possible and add additional features in the future.

Table 21. RUSE Cost Driver

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

• **Documentation match to life-cycle needs:**
This parameter describes the relation between the provided documentation and the application requirements. We can have no accurate estimation regarding the code documentation, but we feel as though we've given a thorough description of the system in our RASD and DD, therefore we'll set this to high.

Table 22. DOCU Cost Driver

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

• **Execution Time Constraint:**
In our case this parameter is not relevant so is reasonable to set it as very low.

Table 23. TIME Cost Driver

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

• Main Storage Constraint:

This rating represents the degree of main storage constraint imposed on a software system or subsystem. In our project this parameter is not relevant so is set as very low.

Table 24. STOR Cost Driver

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

• Platform Volatility:

"Platform" is used here to mean the complex of hardware and software (OS, DBMS, etc.) the software product calls on to perform its tasks. If the software to be developed is an operating system then the platform is the computer hardware. If a database management system is to be developed then the platform is the hardware and the operating system. If a network text browser is to be developed then the platform is the network, computer hardware, the operating system, and the distributed information repositories. The platform includes any compilers or assemblers supporting the development of the software system. This rating ranges from low, where there is a major change every 12 months, to very high, where there is a major change every two weeks. We believe there aren't going to be any major changes in a very long time, thus we'll set this value to low.

Table 25. PVOL 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 Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	n/a	0.87	1.00	1.15	1.30	n/a

• **Analyst Capability:**
Analysts are personnel that 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. We set this value to high because we intend to focus on the analysis of the project a lot during the RASD and DD phases.

Table 26. ACAP Cost Driver

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

• **Programmer Capability:**
Evaluation should be based on the capability of the programmers as a team rather than as individuals. Our programming capabilities are average, thus we'll set this to nominal.

Table 27. PCAP Cost Driver

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

• **Personnel continuity:**
The rating scale for PCON is in terms of the project's annual personnel turnover: we set it to very low.

Table 28. PCON Cost Driver

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	

• **Application Experience:**
This rating is dependent on the level of applications experience of the project team developing the software system or subsystem. This is our first experience in thistype of project, so this value is equal to low.

Table 29. APEX Cost Driver

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

• **Platform Experience:**
Our average knowledge about platforms, such as databases, web and mobile platforms, serversidedevelopment are around 6 months because we developed a project the past semester, so this parameter is set as low.

Table 30. PLEX Cost Driver

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

• **Language and Tool Experience:**
This is a measure of the level of programming language and software tool experience of the project team developing the software system or subsystem. We've only had one programming experience in one project, therefore this value is Low for us.

Table 31. LTEX Cost Driver

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	

• **Usage of Software Tools:**

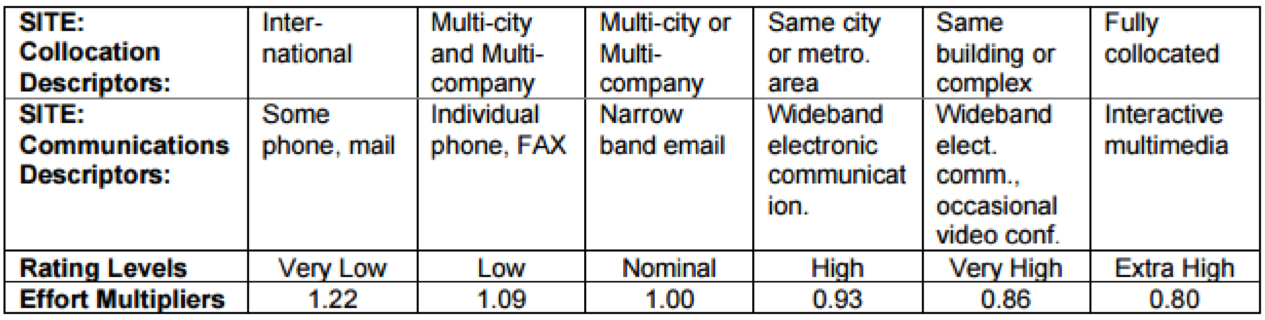
We will use NetBeans, Maven, AndroidStudio and NotePadto develop our project. We will useGit for the repository management. This value should be set to high.

Table 32. TOOL Cost Driver

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

• **Multisite development:**
This parameter reflects how we handled the distribution of development over distance and multiple platforms. We mostly worked together in the same building and room, and we occasionally discussed the project over the phone, thus we'll set this to very high.

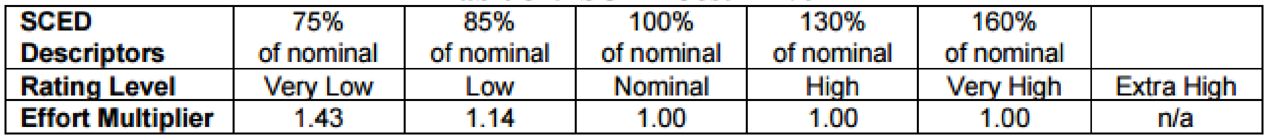
Table 33. SITE Cost Driver



SITE: Collocation Descriptors:	Inter-national	Multi-city and Multi-company	Multi-city or Multi-company	Same city or metro. area	Same building or complex	Fully collocated
SITE: Communications Descriptors:	Some phone, mail	Individual phone, FAX	Narrow band email	Wideband electronic communication.	Wideband elect. comm., occasional video conf.	Interactive multimedia
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.22	1.09	1.00	0.93	0.86	0.80

• **Required development schedule:**
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.We'll set this value to nominal because always managed to achieve our goals within the deadlines and we rarely ever needed to rush our work.

Table 34. SCED Cost Driver



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

Our results are expressed in the following table:

Scale Driver	Factor	Value
Required Software Reliability	Low	0.92
Data Base Size	Nominal	1.00
Product Complexity	Nominal	1.00
Required Reusability	High	1.07
Documentation match to life-cycle needs	High	1.11
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	Nominal	1.00

Personnel Continuity	Very Low	1.12
Application Experience	Low	1.10
Platform Experience	Low	1.09
Language and Tool Experience	Low	1.09
Usage of Software Tools	High	0.90
Multisite development	Very High	0.86
Required development schedule	Nominal	1.00

Product: 0.92

2.4. Effort Equation

To pass from FP to SLOC we use an average conversion factor of 46 as described at http://www.qsm.com/resources/function-point-languages-table, an updated version that adds J2EE of the table included in official manual (http://sunset.usc.edu/research/COCOMOII/Docs/modelman.pdf).

97FPs * 53 = 5141SLOC

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

Effort := A * EAF * $KSLOC^{E}$

Where:
A → 2.94 (for COCOMO.2000)
EAF → product of all the cost drivers, equal to : 0.92 ;
E → exponent derived from Scale Drivers. Is calculated as:
B + 0.01 * sum{i} SF[i] := B + 0.01 *16.54 = 0.91 + 0.1654 = 1.0754;
in which B is equal to: 0.91 for COCOMO.2000 .
KSLOC → estimated lines of code using the FP analysis: 5.141
With this parameters we can compute the Effort value, that is equal to:
Effort := 2.94 * 0.92 * $5.141^{1.0754}$= 15.7 PM

2.5. Schedule Estimation

As far as the schedule estimation we are going to use the following formula:

Duration := 3.67 * $Effort^{F}$

Where:
F := 0.28+0.2*(E – B) = 0.28+0.2 *(1.0754 – 0.91) = 0.28+0.2*(0.1654) = 0.31308

Follows then:
Duration := 3.67 * $15.7^{0.31308}$ = 8.6 Months

Now we can estimate the number of people needed to complete the project with the following formula

Npeople= effort/Duration

Npeople= 15.7/8.6 = 1.83 = 2 people

3. Tasks

The main tasks of this project are: creating the Requirement Analysis and Specification Document, creating the Design Document, creating the Integration Testing Plan Document, creating the Project Plan Document, then implementing the software itself and testing it. Each one of these tasks is composed of more sub-tasks which we will discuss later. As for deadlines, we will use the dates that have been assigned to us for delivering the aforementioned documents, whereas the actual implementation of the software doesn't have a fixed deadline, therefore we'll use the time estimation we gathered by using COCOMO. Starting in early October, these are the fixed deadlines of our project:

-Requirement Analysis and Specification Document, 6th November 2015
- Design Document, 4th December 2015
- Integration Testing Plan Document, 21st January 2016
- Project Plan Document, 2nd February 2016

As stated above, there's no fixed deadline for the implementation of the software, therefore we'll use the estimation we got by using COCOMO. Since our project should take approximately 8 months according to COCOMO and since we've already spent 4 months on creating documents, this means that the remaining 4 months should be dedicated to the implementation and the testing. Thus we believe it's a good idea to sort it this way:

- Software Implementation, 1st May 2016 (3 months)
- Integration Testing, 1st June (1 month)

By applying the waterfall model, we won't have any cyclic dependencies and therefore each phase will start when the previous one is completed. That being said, it might happen that our customers might change their minds about a specific feature of the system and this will require us to go back to the previous phases.

4. Resource Allocation

Now we're going to discuss the distribution of work amongst the team members. There are only two members which we will identify as M and L. We'll go through each task and try to sort the work equally. Each one of the subtasks that composes a task is contained in a chapter of the respective document. If a chapter is assigned to both the members of the team, it means that the two members worked on it together.

- Requirement Analysis and Specification Document

- **M**: chapter (1)Introduction, (2)Actors Identifying,(4)Scenarios Identifying, (6)Alloy Modeling.
- **L**: chapter (1)Introduction, (2)Actors Identifying, (3)Requirements,(5) UML Models.

- Design Document

- **M**: chapter (1)Introduction,(2) Architectural Design, (3) Algorithm Design, (4) User Interface Design, (5)Requirements Traceability.
- **L**: chapter (3) Algorithm Design, (4) User Interface Design.

- Integration Testing Plan Document

- **M**: chapter (1)Introduction, (2)Integration Strategy, (5) Program stubs and test data required.
- **L**: chapter (3)Individual steps and test description, (2)Integration Strategy.

- Project Plan Document

- **M**: chapter (1)Introduction, (2)Project Estimation-2.1FunctionPoints, (3)Tasks, (4)Resource Allocation, (5)Risks.
- **L**: chapter (2)Project Estimation-2.2COCOMO.

As for the implementation, we believe we could follow the strategy we followed in our previous project experience. That is, since our system is based on the MVC pattern, we could identify three main parts of the system, that are the model, the controller and the view. We can start developing the model together, because it's at the very base of the whole system and because we believe it could be a good starting point to define the structure of our system, mostly using the class diagram as a reference. After that, we should parallelize our work so that member M works on the controller and member L works on the view. From our previous experience, this is a good strategy, that turned out to be successful and

that’s why we intend to use it again.
After the development of their respective part, each member should carry out unit testing on his/her components most the time, whereas 10% of the time could be dedicated to test the other member’s components. Lastly, integration testing should be performed together to grant the best results.

5. Risks

5.1 Project Risks

We believe the main problem is the fact that the resources of the team are inexperienced and this might cause a delay in the project because the members won’t be able to keep up with the deadlines. That’s because the two members of the team have little experience from previous projects and they only have a theoretical knowledge of JEE. That being said, this problem can be overcome in different ways. First of all, there should be a good interaction between the two members so that they can help each other out if necessary. Secondly, an experienced supervisor could be assigned to them, not to develop the project, but at least to help them in the fields which mostly require experience. Shouldn’t this problem be overcome, this problem might turn into a technical one, that is the presence of bugs.

5.2 Technical Risks

As stated before, the inexperience of the members of the team might cause bugs in the code, or a low performance of the software in general. The best way to fix this problem is by scheduling a lot of testing on each component and making sure that the design is very robust from the start.

5.3 Business Risks

Preventing this sort of risks is difficult, even though their relevance is high. We can identify two kinds or risks, in different moments. If we consider the phase before the deployment, our company might go bankrupt and we could have a budget problem, but this problem doesn’t involve the developing team. On the other hand, if we consider the phase after the deployment, that is during the sales, we can overcome any budget problem by making a good analysis of what our potential customers want, so that our company can get the highest revenue possible. Thus, the critical phase is the development one, and the best we can do is avoiding making inaccurate cost estimates and forecasts.