Politecnico di Milano

Software Engineering 2

MyTaxiService

# Project Plan Document

*Author:*
Andrea Tirinzoni

*Author:*
Matteo Papini

January 25, 2016

# Contents

# 1 Introduction

## 1.1 Purpose

This is the Project Plan Document for the myTaxiService system. It is meant to provide a detailed schedule of the project, based on an estimation of its costs and on a prediction of its possible risks, together with the allocation of human resources to each step of the development process.

## 1.2 Scope

The aim of this project is to develop a system to improve the taxi service of Milan, accessible both via web and mobile applications. For further details see RASD, section 1.2.

## 1.3 Definitions, Acronyms and Abbreviations

### 1.3.1 Definitions

For definitions regarding the system, see RASD, section 1.3.1;

### 1.3.2 Acronyms

- RASD: Requirements Analysis and Specification Document;
- DD: Design Document;
- ITPD: Integration Test Plan Document;
- UFP: Unajusted Function Points;
- SLOC: Source Lines of Code;
- KSLOC: Kilo Source Lines of Code (SLOC/1000);
- ILF: Internal Logical File;
- EIF: External Interface File;

### 1.3.3 Abbreviations

## 1.4 Reference Documents

- http://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_modelman2000.0.pdf : COCOMOII Model Definition Manual;
- http://www.qsm.com/resources/function-point-languages-table : Conversion from UFP to SLOC;
- RASD.pdf

- DD.pdf
- ITPD.pdf

## 2 Project Cost Estimation

In this section we provide an estimation of the overall project cost (in term of Person/Month effort) and of its duration.
The section is organized as follows:

- in section 2.1 we estimate the SLOC by using the Function Points approach;
- in section 2.2 we estimate the project effort and duration by using the COCOMO II model;
- in section 2.3 we provide a comment on the results obtained;

### 2.1 Function Points

In this section we first compute the Function Points of the system. We will refer to the following table for their weights:

| **Function Types** | **Simple** | **Medium** | **Complex** |
|---|---|---|---|
| External Input | 3 | 4 | 6 |
| External Output | 4 | 5 | 7 |
| External Inquiry | 3 | 4 | 6 |
| ILF | 7 | 10 | 15 |
| EIF | 5 | 7 | 10 |

We then estimate a possible value for SLOC.

#### 2.1.1 Internal Logical File

The ILFs, data created, stored and handled by the system, together with their complexity, are the following:

- passenger data: a set of primitive data types, thus considered as simple;
- taxi driver data: a set of primitive data types, thus considered as simple;
- taxi data: a set of primitive data types, thus considered as simple;
- taxi call data: a set of primitive data types, thus considered as simple;
- taxi reservation data: a set of primitive data types, thus considered as simple;
- queue data: this is a complex data structure including data from several objects; we treat it as complex;
- zone data: a set of primitive data types, thus considered as simple;

- request data: although this includes data from other objects, it has a really simple structure; thus, we treat it as simple;
- notification data: this is just text to be sent to the user, thus we treat it as simple;
- city topology data: this is the graph data structure mentioned in the DD; it is a big data structure retrieved from another source but still handled by the system; it is considered as complex;

By summing each weight for each ILF we find a total of 86 function points.

### 2.1.2 External Interface File

The EIFs, data used by the system but obtained from external sources, together with their complexity, are the following:

- GPS data from taxis: this is of course simple data;
- traffic data: this is used to update the city graph and it may contain a lot of information; thus we consider it as medium;
- Google maps data (used by clients): this could be quite complex information but, considering our application does not manage it directly (it uses Google APIs), we consider it as simple;

By summing each weight for each EIF we find a total of 17 function points.

### 2.1.3 External Input

The External Inputs, elementary operations to process incoming data without producing any substantial output, together with their complexity are the following:

- login (for passenger): this operation needs to perform only a few checks, thus we consider it simple;
- login (for drivers): there are just a few differences from the case above, thus we consider this as simple;
- registration: this must perform several operations (involving for example the validation via SMS), thus we consider it medium;
- taxi call: this checks only a few things more than login, thus we still consider it simple;
- taxi reservation: it is very similar to a taxi call, thus we consider it simple;
- set busy: this is a trivial operation (removal of a driver from a queue), thus we consider it simple;

By summing each weight for each External Input we find a total of 19 function points.

### 2.1.4 External Output

The External Outputs, elementary operations that generate data for the environment, together with their complexity, are the following:

- waiting time computation: this is a complex operation to be performed frequently (see DD, section 3.2);
- queue position computation: this is a simple operation to compute the position of the driver in a queue;
- request computation: this is a simple operation to compute a request to be forwarded to a driver (it just puts together simple data);
- validation code computation: this is a simple operation to generate a random code;

By summing each weight for each External Output we find a total of 19 function points.

### 2.1.5 External Inquiry

The External Inquiries, elementary operations involving input and output, together with their complexity, are the following:

- receive notifications: this is a simple operation to retrieve the notifications of a certain user;
- set free: this is a simple operation to set a driver in free state (it also returns the queue position, that is why it is an external inquiry);

By summing each weight for each External Inquiry we find a total of 6 function points.

### 2.1.6 Total Function Points

We summarize the function points in the following table:

| Function Types | Complexity | Weight |
|---|---|---|
| **ILF** | Total | 86 |
| passenger data | simple | 7 |
| taxi driver data | simple | 7 |
| taxi data | simple | 7 |
| taxi call data | simple | 7 |
| taxi reservation data | simple | 7 |
| queue data | complex | 15 |
| zone data | simple | 7 |
| request data | simple | 7 |
| notification data | simple | 7 |
| city topology data | complex | 15 |
| **EIF** | Total | 17 |
| GPS data | simple | 5 |
| traffic data | medium | 7 |
| Google maps data | simple | 5 |
| **External Input** | Total | 19 |
| login (passenger) | simple | 3 |
| login (driver) | simple | 3 |
| registration | medium | 4 |
| taxi call | simple | 3 |
| taxi reservation | simple | 3 |
| set busy | simple | 3 |
| **External Output** | Total | 19 |
| waiting time computation | complex | 7 |
| queue position computation | simple | 4 |
| request computation | simple | 4 |
| validation code computation | simple | 4 |
| **External Inquiry** | Total | 6 |
| receive notifications | simple | 3 |
| set free | simple | 3 |

We can sum all of them to find a total of 147 UFPs.

### 2.1.7 SLOC Estimation

We can now estimate the SLOC of the project. In order to do this, we refer to the table of "UFPs to SLOC conversion" (see the document linked in section 1.4). Considering we are using Java EE, we have a conversion coefficient of 46. Thus, we can estimate SLOC by using the following formula:

$SLOC = UFPs * 46$

We obtain an estimation of 6762 SLOC (thus 6.762 KSLOC).

## 2.2 COCOMO

In this section we estimate the project effort and duration by using the COCOMO II model. In order to find suitable values for all the parameters, we will refer to the COCOMO II Model Definition Manual (see the references in section 1.4). In particular, the equations that we will consider are introduced in section 1.2 of the manual.
We proceed as follows:

- in section 2.2.1 we compute the scale drivers;
- in section 2.2.2 we compute the cost drivers;
- in section 2.2.3 we use the computed values to estimate the above mentioned quantities;

### 2.2.1 Scale Drivers

In order to compute the scale drivers, we follow the rules and the weights introduced in section 3.1 of the COCOMO manual.

The Scale Drivers are:

- Precedentedness (PREC): myTaxiService is the first project of this type we have ever done; furthermore, we lack of experience in many technologies (e.g., Java EE and mobile development) and platforms (e.g., Glassfish); having some experience in a few related fields (e.g., Java programming and testing), we set the value for this driver to Low.
- Development Flexibility (FLEX): in this case, the client has only set general goals. Only a few constraints on the architecture were imposed (e.g., web and mobile interfaces). The project is overall very flexible, and we can give this driver a Very High value;
- Architecture / Risk Resolution (RESL): a general risk study (with possible countermeasures) is provided in section 5. Considering the study is not very deep, we assign a High value to this driver;
- Team Cohesion (TEAM): all the team members know each other and physically work together. There is no communication issue and there is some past experience on other projects. Thus, we consider for this driver a Extra High value;
- Process Maturity (PMAT): in order to estimate a value for this driver we use the procedure based on KPAs (Key Process Areas) described in section 3.1.5 of the COCOMO manual. We set an average compliance for all the 18 KPAs of 60% (this is reasonable since we performed many of the 18 activities, but with different efforts). By using the formula Eq. 13 (section 3.1.5 of the manual) we find a value of 3 for EPML. This gives us a value High for the driver;

We summarize the Scale Drivers, together with the corresponding Scale Factors (obtained from the table in section 3.1 of the COCOMO manual), in the following table:

| **Scale Driver** | **Value** | **Scale Factor** |
|---|---|---|
| PREC | Low | 4.96 |
| FLEX | Very High | 1.01 |
| RESL | High | 2.83 |
| TEAM | Extra High | 0 |
| PMAT | High | 3.12 |
| **Total** | | 6.96 |

We get a total sum of the Scale Factors of 6.96.

### 2.2.2 Cost Drivers

In order to compute the Cost Drivers, we follow the rules and weights introduced in section 3.2 of the COCOMO manual.

The Cost Drivers are:

- Required Software Reliability (RELY): we don't have very constraining reliability requirements (just availability, see RASD). We consider this driver as Low.
- Data Base Size (DATA): if we consider test data of 500KB and remembering that we have estimated 6762 SLOC, we have a ratio of 74, which, according to the table at page 26 of the manual, gives us a Nominal value for this driver;
- Product Complexity (CPLX): the project is of average complexity, thus we consider a Nominal value for this driver;
- Developed for Reusability (RUSE): we have no reusability requirement, thus we have a Low value for this driver;
- Documentation Match to Life-Cycle Needs (DOCU): the project needs a standard documentation, thus we have a Nominal value for this driver;
- Execution Time Constraint (TIME): considering we are using two clustered application servers (to be sure to achieve the required availability and performance), the average use of execution time can fall below 50%. Thus, we have a Nominal value for this driver;
- Main Storage Constraint (STOR): we have no such constraint, therefore we can forget about this driver (we still use a Nominal value, so that the contribute of 1 in the product will cancel out);

- Platform Volatility (PVOL): our system is composed by many different platforms, which are subject to frequent updates. We consider a High value for this driver, which corresponds to an average of one major update of at least one platform every two months;
- Analyst Capability (ACAP): we don't have much experience in software design and analysis, so we consider a Low value for this driver;
- Programmer Capability (PCAP): we have good team programming experience from past projects, so we can consider a High value for this driver;
- Personnel Continuity (PCON): the overall continuity of our team is very good, thus we consider a Very High value for this driver;
- Applications Experience (APEX): the team has a very poor experience on this type of project, so we consider a Very Low value for this driver;
- Platform Experience (PLEX): the team has a poor experience on this type of platforms, so we consider a Low value for this driver;
- Language and Tool Experience (LTEX): the team has a fairly good experience on the adopted language and tools, thus we consider a Nominal value for this driver;
- Use of Software Tools (TOOL): the team has a poor experience on life-cycle management tools, so we consider a Low value for this driver;
- Multisite Development (SITE): the team is physically working together and there are no communication issues. Thus, we consider a value of Extra High for this driver;
- Required Development Schedule (SCED): we don't have any acceleration or stretch-out of the schedule, so we consider a Nominal value for this driver;

We summarize the Cost Drivers, together with the associated Effort Multipliers, in the following table:

| **Cost Driver** | **Value** | **Effort Multiplier** |
|---|---|---|
| RELY | Low | 0.92 |
| DATA | Nominal | 1 |
| CPLX | Nominal | 1 |
| RUSE | Low | 0.95 |
| DOCU | Nominal | 1 |
| TIME | Nominal | 1 |
| STOR | Nominal | 1 |
| PVOL | High | 1.15 |
| ACAP | Low | 1.19 |
| PCAP | High | 0.88 |
| PCON | Very High | 0.81 |
| APEX | Very Low | 1.22 |
| PLEX | Low | 1.09 |
| LTEX | Nominal | 1 |
| TOOL | Low | 1.09 |
| SITE | Extra High | 0.8 |
| SCED | Nominal | 1 |
| **Total** | | 0.98861 |

By taking the product of all Effort Multipliers we obtain an overall contribution of 0.98861.

### 2.2.3 Project Effort and Duration

We can now compute the final estimation for the project effort and the project duration. As already mentioned, all the formulae are taken from the COCOMO manual (see references, section 1.4).
The project effort (PM) is given by:

$$PM = A * KSLOC^E * \prod EM(i)$$

where:

- A=2.94 is a constant;
- KSLOC has been computed in section 2.1.7;
- EM(i) is the i-th Effort Multiplier, and the product ranges over all Cost Drivers;

The parameter E is given by:

$$E = B + 0.01 * \sum SF(i)$$

where:

- B=0.91 is a constant;

- SF(i) is the i-th Scale Factor, and the sum ranges over all Scale Drivers;

Considering we have already computed:

$\sum SF(i) = 6.96$
$\prod EM(i) = 0.98861$

we can substitute the values and we get:

$E = 0.9796$
$PM = 18.9$

Thus we have a project effort of 18.9 Person/Month.

We can estimate the project duration as:

$TDEV = C * PM^F$

where C=3.67 is a constant. The parameter F is given by:

$F = D + 0.2 * (E - B)$

where D=0.28 is a constant.

By substituting the values we get:

$F = 0.29392$
$TDEV = 8.7$

Thus the project will take about 9 months.
We can easily compute the number of persons involved as:

$NP = PM/TDEV = 2.17$

which is really close to the actual number of people in the team (2).

## 2.3 Final Comments

We provide here a final comment regarding the results obtained in the previous sections.
We obtained an ideal number of people for our team of 2.17. Considering we are actually two people, the overall project duration should be slightly longer (it increases after the approximation). Nevertheless, the computed project duration is probably overestimated. This is due to the huge importance the COCOMO model gives to certain parameters, which in our application do not play a fundamental role.

In order to ease the scheduling process, we will consider an overall duration of 9 months, which is slightly less than the one estimated for a team of exactly two people (about 9.5 months). This approximation does not affect in any way the system development, and it should be still overestimated.

# 3 Project Tasks and Schedule

In this section the project tasks are identified and organized in a schedule:

- in section 3.1 some assumption are made in order to precisely define the problem;
- in section 3.2 tasks are identified and given an estimated duration;
- in section 3.3 the schedule is computed, taking into account dependencies between tasks;

## 3.1 Assumptions

We consider 15/10/2015 to be the starting date of the project. As for the documentation already completed at the time of writing, we behave as follows:

- RASD: considered as already completed before the starting date;
- DD: considered still to be produced (retrospectively);
- ITPD: considered still to be produced (retrospectively);

Two people collaborate to the project, who are also the authors of this document. From now on they will be referred to simply as Andrea and Matteo.
We assume a duration of 9 months for the project, as estimated in Section 2, with the resulting overall deadline being 15/07/2016.

## 3.2 Tasks and Their Duration

In order to identify the tasks and their duration, we first divide the project into phases, and assign to each phase a percentage of the available time, approximated in 270 days. To do so, we start from the estimations provided in Section 6.3 of the COCOMO manual and make some minor adjustments:

- Product Design: 15% $\rightarrow$ 40 days;
- Programming: 50% $\rightarrow$ 135 days;
- Integration and Test: 35% $\rightarrow$ 95 days;

Then, for each phase, we identify the tasks, and intuitively assign to each task a portion of the days previously assigned to the phase. At this point, it is assumed that the two team members work together on each task. The possibility of overlapping tasks by assigning them to one or the other of the team members is discussed in the next section.

Product Design

- Compilation of the DD: 35 days;

- Compilation of the ITPD: 5 days;

Programming

- Detailed design: 35 days; this task includes UML modeling of single classes/beans, and the making of program skeletons;
- Data Tier implementation and UT: 15 days; unit testing is performed along with coding;
- Business tier implementation and UT: 45 days; unit testing is performed along with coding;
- Web tier implementation and UT: 20 days; unit testing is performed along with coding;
- Mobile app development and testing: 20 days; unit testing is performed along with coding;

Integration and Test:

- Integration testing: 15 days;
- Complete system testing: 10 days;
- Performance testing: 10 days;
- Alpha testing: 15 days;
- Beta testing: 45 days;

## 3.3 Schedule

### 3.3.1 Dependencies

In order to compute the project schedule, we must consider the dependencies among tasks. We provide here a sufficient set of dependencies:

- “Compilation of the ITPD” requires that “Compilation of the DD” has been completed;
- “Detailed design” requires that all tasks in the product design phase have been completed;
- “Data Tier implementation and UT” requires that “Detailed design” has been completed;
- “Business tier implementation and UT”, “Web tier implementation and UT” and “Mobile app development and testing” require that “Data Tier implementation and UT” has been completed;
- “Integration testing” requires that “Compilation of the ITPD” and all tasks in the programming phase have been completed ;

- “Complete system testing” requires that “Integration testing” has been completed;
- “Performance testing” requires that “Complete system testing” has been completed;
- “Alpha testing” requires that “Performance testing” has been completed;
- “Beta testing” requires that “Alpha testing” has been completed;

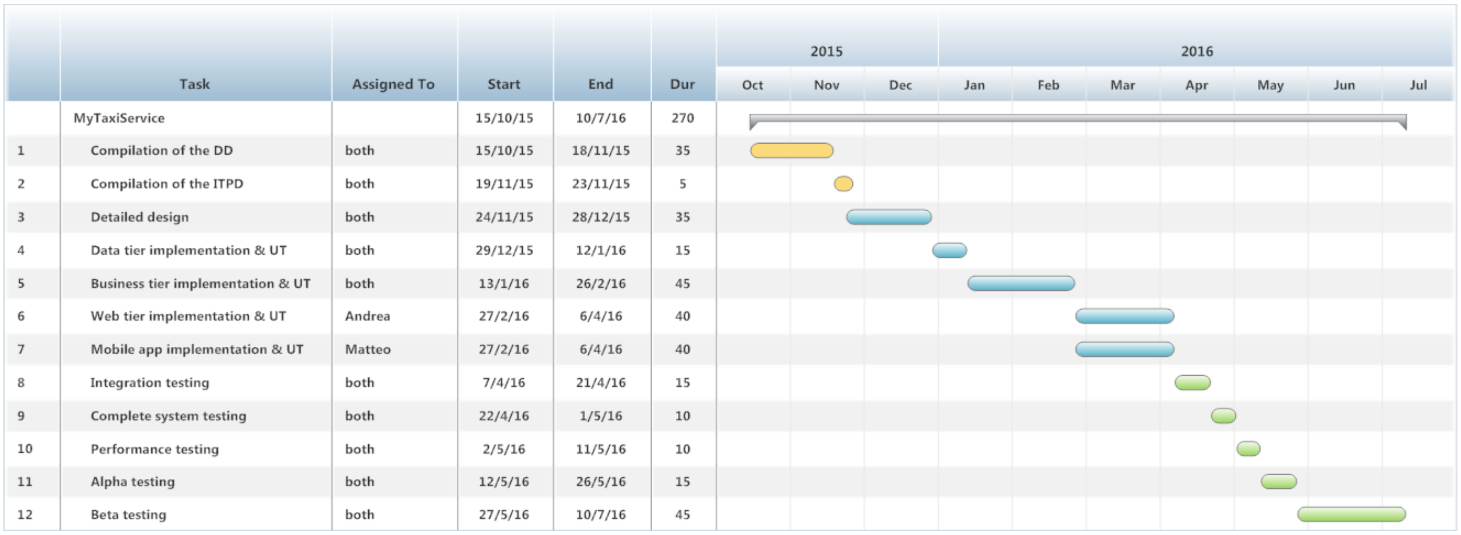
#### 3.3.2 Overlaps

The only possible overlaps are between “Business tier implementation and UT”, “Web tier implementation and UT” and “Mobile app development and testing”.

Considering that the number of developers is fixed and that they have similar experience and availability, we estimate that allocating a task to a single group member doubles its duration. Hence, no time can be saved by letting tasks overlap. We decide to allocate “Web tier implementation and UT” and “Mobile app development and testing” to single team members, being these relatively simple tasks requiring similar effort. Conversely, we keep the development of the business tier a shared task to exploit the continuous exchange of ideas and solutions between team members.

#### 3.3.3 Gantt Diagram

The schedule resulting from all previous consideration is provided here in form of a Gantt diagram:



| | Task | Assigned To | Start | End | Dur |
|---|---|---|---|---|---|
| | MyTaxiService | | 15/10/15 | 10/7/16 | 270 |
| 1 | Compilation of the DD | both | 15/10/15 | 18/11/15 | 35 |
| 2 | Compilation of the ITPD | both | 19/11/15 | 23/11/15 | 5 |
| 3 | Detailed design | both | 24/11/15 | 28/12/15 | 35 |
| 4 | Data tier implementation & UT | both | 29/12/15 | 12/1/16 | 15 |
| 5 | Business tier implementation & UT | both | 13/1/16 | 26/2/16 | 45 |
| 6 | Web tier implementation & UT | Andrea | 27/2/16 | 6/4/16 | 40 |
| 7 | Mobile app implementation & UT | Matteo | 27/2/16 | 6/4/16 | 40 |
| 8 | Integration testing | both | 7/4/16 | 21/4/16 | 15 |
| 9 | Complete system testing | both | 22/4/16 | 1/5/16 | 10 |
| 10 | Performance testing | both | 2/5/16 | 11/5/16 | 10 |
| 11 | Alpha testing | both | 12/5/16 | 26/5/16 | 15 |
| 12 | Beta testing | both | 27/5/16 | 10/7/16 | 45 |

# 4 Task Allocation

The development of the mobile app is assigned to Matteo, while the development of the web tier is assigned to Andrea. This allocation has been done at random, considering that both team members has no previous experience with the specific technologies to be used, and no preferences of sort. All other tasks are shared, in particular:

- Compilation of DD and ITPD are done together, since critical decisions has to be taken;
- Detailed design is done together, in order to exploit the better ideas of both members and to prevent integration problems;
- In implementing the Data Tier, each team member can focus on a different set of Entity Beans. The division of the effort must be as fair as possible, to avoid delays and lowering of motivation, so the exact allocation can be done only after the detailed design;
- Similarly, the components of the Business Tier can be divided between the two members. Considering the complexity of this task, we decide to allocate the components in advance, estimating the required effort on the base of the Design Document (see Section 4.1);
- Integration tests, as described in the ITPD, are divided between the two members in order to exploit the possible overlaps (see Section 4.2);
- All other tasks in the testing phase are performed together in order to identify and fix problems as soon as possible;

## 4.1 Business Tier Component allocation

To allocate the components of the Business Tier to the team members, we first give a raw estimation of the required effort by analyzing the DD, taking into account the defined interfaces, algorithms and interactions:

- Low effort components: Web Services, Login Manager, Registration Manager, Taxi Call Manager, Taxi Reservation Manager, GPS Manager, Logout Manager;
- Medium effort components: Communication Manager;
- High effort components: Taxi Allocation Manager, Queue Manager;

Then, on the base of this estimation, we allocate the components as fairly as possible, following the order imposed by the dependencies highlighted in the ITPD (Section 2.4.1), just for coherence:

Components allocated to Andrea: Login Manager, Registration Manager, Communication Manager, Taxi Allocation Manager, Web Services;

Components allocated to Matteo: GPS Manager, Taxi Reservation Manager, Queue Manager, Logout Manager, Taxi Call Manager;

## 4.2 Integration test allocation

The allocation is done so to respect the dependencies and exploit the possibility of overlapping between some tests. Here we make explicit reference to the ITPD, section 3.1:

Tests allocated to Andrea: 1, 4, 5, 6, 10, 12, 15, 17, 18;

Tests allocated to Matteo: 2, 3, 8, 7, 9 ,11, 13, 14, 16, 19;

## 5 Project Risks

In this section we provide a detailed analysis of the risks that can possibly occur during the project development. First, we identify the most important risks. For each of them, we estimate the probability that it will occur (expressed as Low,Medium,High) and its possible impact on the project (Negligible,Marginal,Critical,Catastrophic). Then, we define some countermeasures for each identified risk, specifying how to reduce its probability and/or how to remedy to its occurrence.

The risks we consider are specified in the following table:

| **Risk** | **Probability** | **Impact** |
|---|---|---|
| One team member can't work (e.g., illness) at critical phases | Medium | Critical |
| One team member leaves the project | Low | Catastrophic |
| Project cost estimates are inaccurate | Medium | Negligible |
| Dependencies between tasks are inaccurate | Low | Critical |
| Stakeholders become disengaged | Low | Marginal |
| Requirement misunderstanding | Low | Critical |
| Resources are inexperienced | High | Critical |
| Low team motivation | Medium | Marginal |
| Design is infeasible or not fit for the purpose | Low | Catastrophic |
| Failure to integrate components | Medium | Critical |
| The final system doesn't meet the performance requirements | Medium | Catastrophic |
| The final system doesn't pass alpha/beta test | Low | Critical |
| Requirements are incomplete | High | Marginal |
| Schedule is infeasible | Low | Critical |
| Difficulties to integrate external services | Medium | Critical |
| Requirements have big changes | Low | Critical |
| Product is complex to implement | Low | Critical |

We now analyze each of the above mentioned risks, providing, if possible, some countermeasures:

- One team member can't work (e.g., illness) at critical phases: considering the team is composed of two people, the project completion time could increase dramatically if this happened. A possible solution is to prepare the other member to substitute the unavailable colleague in case the latter is performing a high priority activity (such as one involved in many dependencies). In order to do that, both team members must be equally involved in the early phases of the project (requirements,design,planning, etc.), so that both have a complete knowledge of the whole system;

- One team member leaves the project: this of course would be catastrophic. Both team members must be motivating each other during the whole process in order to minimize to probability of this happening;

- Project cost estimates are inaccurate: as already mentioned in section 2.3, the cost obtained using COCOMO could be overestimated. Thus, even though the probability of this happening is pretty high, the impact on the project is not relevant in our case;

- Dependencies between tasks are inaccurate: this could have a big impact on the timeline, and must be prevented by carrying out a deep analysis of the tasks our project involves;

- Stakeholders become disengaged: even though this should not have a big impact on our project, we can remedy by putting a lot of effort in the requirement elicitation phase. If communication with the stakeholders in this phase is such that all the required information are obtained, we will need fewer interactions during the next phases;

- Requirement misunderstanding: our requirements were pretty straightforward, so the probability of any misunderstanding is low. Anyway, we can prevent this by putting a lot of effort in the requirement elicitation phase (as explained above);

- Resources are inexperienced: as already mentioned in section 2.2.2, the team lacks of experience in many of the adopted technologies. The impact on the project is reduced by considering a higher duration (see the Cost Drivers, section 2.2.2);

- Low team motivation: this is pretty likely considering the project has an overall duration of 9 months. As already said in the second risk, both team members must be motivating each other during the whole process;

- Design is infeasible or not fit for the purpose: this would be catastrophic for the project. In fact, it could lead to the restart of the project or even to its cancellation. To minimize the probability of this happening, a high percentage of the total duration is devoted to the system design. During this phase, a lot of possible cases must be taken into account, while keeping a continuous interaction with the stakeholders;

- Failure to integrate components: considering some components will be developed by one team member, and others by the other member, this is likely to happen. To minimize its probability, cooperation must be continuous during the whole project. Furthermore, component design must be performed by both members together;

- The final system doesn't meet the performance requirements: in case this happens, the consequences will be catastrophic for the system. In fact, it could imply the purchase of new hardware, the re-design of some components or even of the whole system. To avoid this, a deep analysis must be performed during the design phase, maybe asking advice to an expert of the field;

- The final system doesn't pass alpha/beta test: this could mean that the client is not satisfied with the final product, e.g. because some functionalities are missing. This is considered pretty unlikely, because the requirements were few and clear. Anyway, a constant communication with the stakeholders can help us avoid this;
- Requirements are incomplete: this is considered very likely (almost every time the customer forgets something). Nevertheless, we can avoid this by keeping a constant communication with the stakeholders (especially in the requirement elicitation phase);
- Schedule is infeasible: as already mentioned, this is very unlikely (the duration has probably been overestimated). Anyway, in case this happens we can easily postpone the final deadline, considering we have no constraint on this;
- Difficulties to integrate external services: considering the application uses many external services, this is pretty likely. During the design phase, it is possible to analyze several providers of the same service, so that, if problems arise with one of them, it will be possible to switch to another;
- Requirements have big changes: this is considered very unlikely in our case. Nevertheless, we can foresee big requirement changes by keeping a good interaction with the stakeholders in the early phases of the project, trying to understand deeply its scope. Furthermore, the system can be designed to be extensible (in particular, in terms of performance).
- Product is complex to implement: considering we don't have much experience in this type of project, it could happen that our complexity estimations are wrong. Nevertheless, our analysis was detailed and we consider this happening very unlikely. If this occurs, the implementation time could increase dramatically, and we would have to postpone the final deadline;

## 6 Appendix

### 6.1 Spent Hours

- Andrea Tirinzoni: ~15h
- Matteo Papini: ~15h

### 6.2 Total Hours

The total hours spent during the whole project are:

- RASD: 30h each;
- DD: 22h each;
- ID: 10h each;
- ITPD: 15h each;
- PPD: 15h each;

Overall, we spent 92h each for the whole project.