POLITECNICO
MILANO 1863

22 January 2017

PowerEnJoy
Project Plan

Blanco Federica 875487
Casasopra Fabiola 864412

Software Engineering 2 Project
2016/2017

Contents

List of Figures

List of Tables

1 Introduction

In the Project Plan document, we will show the following information:

- evaluation of the estimated size of our project;
- evaluation of the estamated effort and cost;
- identification of the tasks to be carried out and their schedule;
- allocation of the available resources to the various tasks previously defined;
- evaluation of the risks for the project.

1.1 List of Definitions and Abbreviations

Here there is the acronims and abbreviations list:

ACAP Analyst Capability

APEX Applications Experience

API Application Programming Interface

CMMI Capability Maturity Model Integration

COCOMO COnstructive COst MOdel

CPLX Product Complexit

DATA Data Base Size

DD Design Document

DBMS DataBase Memory System

DOCU Documentation Match to Life-Cycle Needs

EI External Input

EIF External Interface File

EO External Output

EQ External Inquiry

FLEX Development Flexibility

FP Function Point

ILF Internal Logic File

ITPD Integration Test Plan Document

JEE Java Platform, Enterprise Edition

LTEX Language and Tool Experience

OS Operating System

PCAP Programmer Capability

PCON Personnel Continuity

PLEX Platform Experience

PP Project Plan

PREC Precedentedness

PMAT Process Maturity

PVOL Platform Volatility

RASD Requirements Analysis and Specifications Document

RELY Required Software Reliability

RESL Risk Resolution

RUSE Developed for Reusability

SCED Required Development Schedule

SITE Multisite Development

SLOC Source Lines Of Code

STOR Main Storage Constraint

TEAM Team Cohesion

TIME Execution Time Constraint

TOOL Use of Software Tools

UFP Unadjusted Function Poin

1.2 List of Reference Documents

- Specification document: Assignments AA 2016-2017.pdf
- Requirements Analysis and Specifications Document: RASD.pdf (`https://github.com/fabiola-casasopra/sw-eng-2-project/tree/master/RASD/RASD.pdf`)
- Design Document: DD.pdf (`https://github.com/fabiola-casasopra/sw-eng-2-project/blob/master/DD/DD.pdf`)
- Integration Test Plan Document: ITPD.pdf (`https://github.com/fabiola-casasopra/sw-eng-2-project/blob/master/ITDP/ITDP.pdf`)

- http://www.qsm.com/resources/function-point-languages-table: to know the conversion coefficient between FPs and SLOC
- http://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_modelman2000.0.pdf: COCOMO II model definition manual

1.3 Document overview

Here we show the stucture of the document, with a brief overview of each section.

Section 1 There is an introduction with general information about this document.

Section 2 There is the application of Function Points to estimate the size of our project

Section 3 There is the application of COCOMO to estimate effort and cost of our project.

Section 4 Here, we are going to identify the tasks to be carried out for our project and their schedule. Moreover, we are going to allocate the available resources to the various tasks.

Section 5 Here, we are going to define the risks for the project, their relevance and the associated recovery actions.

Section 6 Here there are given additional information that may be useful to the reader, such as the tools used and the time spent to redact this document.

2 Function Points Approach

The Function Point approach has been defined in 1975 by Allan Albrecht at IBM. This technique allows to evaluate the total dimension of the program, making the assumption that the dimension of software can be characterized based on the functionalities that it has to offer. Once we have estimated the size of our project, we can therefore estimete the effort needed to complete it.

The Albrecht's method identifies and count the number of function types, that represent the different functionality of the application. In particular, there are five distinct function types:

- **Internal Logic File**: ILF is the homogeneous set of data used and managed by the application;
- **External Interface File**: EIF is the homogeneous set of data used by the application but generated and maintained by other applications;
- **External Input**: EI is the elementary operation to elaborate data coming from the external environment;
- **External Output**: EO is the elementary operation that generates data for the external environment and it usually includes the elaboration of data from logic files.
- **External Inquiry**: EQ is the elementary operation that involves input and output, without significant elaboration of data from logic files.

In order to identify the development effort basing on these elements, Albrecht considered 24 applications basing on which he defined the number of FP as the sum of Function Types weighted and he compiled the following table:

Function Types	**Simple**	**Medium**	**Complex**
EI	3	4	6
EO	4	5	7
EQ	3	4	6
ILF	7	10	15
EIF	5	7	10

Table 1: Correlation between FP weight and the development effort

In this way, we can calculate the FPs of the our system as the sum of the FPs obtained by evaluating each one of the five different types of Functional

Points. In particular, we are going to define a number of items for every type of FPs and assign them a weight representing its complexity. As far as the list of functionalities is concernerd, we obtain it from the previously written documents: the Requirements Analysis and Specifications Document and theDesign Document.

- **Internal Logic File**: our application includes a number of ILFs used to store the information needed to offer the required functionalities. In particular, this information are about *users*, *cars*, *reservation*, *safe areas*, *power grid station* and*transactions*.

  User entity has a simple structure as it is composed of a small number of field;*Car* and *Transaction* entities have a medium complex structure; *Reservation* entity instead has a complex structure. Thus, we decide to adopt the `simple weight` for *User*, the `medium weight` for *Car*, *Safe Areas*, *Power Grid Station* and *Transaction* and, finally, the `complex weight` for *Reservation*. In a mathematical way, this means that we have $1 * 7 + 4 * 10 + 1 * 15 = 62$ FPs concerning ILFs.
- **External Interface File**: our application features only one EIF, in order to manage the interaction with **Google Maps** APIs. Our application interacts with the Mapping Service in different ways:

  - given an address, get the correspondent pair of coordinates (reverse geocoding);
  - retrieve the graphical representation of the city map to be displayed on the users' smartphone.

  Since we may hypotize that these intercations are complex and there is a large amount of data to be retrieved, we decide to adopt a `complex weight`. In a mathematical way, this means that we have $2 * 10 = 20$ FPs concerning EIFs.
- **External Input**: our application interacts with the user to allow them to:

  - *Login*, *Logout*: these are simple operations, so we can adopt the `simple weight` for them: $2 * 3 = 6$ FPs.
  - *Sign Up*: this is not a simple operation, as it involves a number of checks on the validity of the fields. So, we think that it is suitable to adopt the `medium weight`: $1 * 4 = 4$ FPs.
  - *Update Account*: this is a simple operation, so we can adopt a `simple weight`: $1 * 3 = 3$ FPs.

- *Reserve a Car*: this is not a simple operation, since a large number of components are involved. So, we think that it is suitable to adopt the `complex weight`: $1 * 6 = 6$ FPs.
- *Look for Safe Areas and Power Grid Stations*: this is not a simple operation, as it involves interaction with other components. So, we think that it is suitable to adopt the `complex weight`: $2 * 6 = 12$ FPs.

  after a specific request, our application shows the user the *Safe Areas* and the *Power Grid Station* around a choosen location. For this operation, we think that it is suitable to adopt the `complex weight`: $2 * 6 = 12$ FPs.
- *Cancel a Reservation*: this is a simple operation, so we think that it is suitable to adopt the `simple weight`: $1 * 3 = 3$ FPs.
- *View Transaction History*: this is a simple operation, so we can adopt a `simple weight`: $1 * 3 = 3$ FPs.

As a result, we get $6 + 4 + 3 + 6 + 12 + 3 + 3 = 37$ FPs.

- **External Output**:
  - *Notify a User that his Reservation has expired*: for this operation, we think that it is suitable to adopt the `simple weight`: $1 * 4 = 4$ FPs.
  - *Notify a User that he cannot park the Car since it is not in a Safe Area*: for this operation, we think that it is suitable to adopt the `simple weight`: $1 * 4 = 4$ FPs.
  - *Notify a User that his fee will be charged and why*: for this operation, we think that it is suitable to adopt the `simple weight`: $1 * 4 = 4$ FPs.
  - *Notify a User that his fee will be discounted and why*: for this operation, we think that it is suitable to adopt the `simple weight`: $1 * 4 = 4$ FPs.
  - *Notify the User the amount to pay*: for this operation, we think that it is suitable to adopt the `simple weight`: $1 * 4 = 4$ FPs.

    As a result, we get $5 * 4 = 20$ FPs.
- **External Inquiry**: our application allows users to access some information.

– *Retrieve Transactions' Details*: this is a simple operation, so we can adopt a `simple weight`: $1 * 3 = 3$ FPs.

As a result, we get 3 FPs.

Now, we can compute the value for the Unadjusted Function Poin, with the following formula:

$$UFP = \sum(\#elements\_of\_a\_given\_type * weight)$$

Considering the previously part, we have that the total UFP is computed as following: $UFP = 62 + 20 + 37 + 20 + 3 = 142$

Thanks to this value, we can estimate the effort in two ways:

– *directly*: this approach is viable only in case we have some historical data, so that we can know how much time it usually take for developing a FP;

– *indirectly*: this approach consists in computing the size of the project in Source Lines Of Code, starting from the total UFP, and then use another approach, such as COCOMO II, in order to estimate the effort.

Since we don't have any historical data, we aren't able to make a direct estimation, so we take in consideration the undirect approach. In order to convert the UFP into SLOC, we decided to use the standard conversion coefficient. We find out that the coefficient related to the specific framework we are using, that is JEE, is equal to 46, as shown in the table at the following link: `http://www.qsm.com/resources/function-point-languages-table` Therefore, we can finally compute the SLOC for our project:

$$SLOC = 46 * FPs = 6532$$

3 COCOMO Approach

In this section of the document, we are going to apply the COCOMO approachin order to estimate effort and cost for our project. In order to do that, we are going to refer to the COCOMO II model definition manual at the following link: `http://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_modelman2000.0.pdf`.

Scale Driver	Factor	Value
PREC	`High`	2.48
FLEX	`Nominal`	3.04
RESL	`Low`	5.65
TEAM	`High`	2.19
PMAT	`High`	3.12

Table 2: Scale drivers and the corresponding Factor and Value

We considered the following scale drivers:

- *Precedentedness*: if a product is similar to several previously developed projects, then PREC is high. Since on the market there are already some project similar to our, we rate this scale driver as `High`, with a corresponding value of 2.48
- *Development Flexibility*: if there are no specific constraints to conform to pre-established requirements and external interface specifications, then FLEX is high. Since we had to remain compliance with the specification given, but with some relaxation for some of the specifications, we rate this scale driver as `Nominal`, with a corresponding value of 3.04
- *Risk Resolution*: if we have a good risk management plan, clear definition of budget and schedule, focus on architectural definition, then RESL is high. Since it is the first time we have to manage a project, we are not too confident about the precision of our Risk Management Plan, Scheduling and Resource Allocation. So, we rate this scale driver as `Low`, with a corresponding value of 5.65
- *Team Cohesion*: if all stakeholders are able to work in a team and share the same vision and commitment, then TEAM is high. In these past months our team worked very well together, but we think that the external stakeholders may have a vision different (even a little) from ours. For this reason, we rate this scale driver as `High`, with a corresponding value of 2.19

– *Process Maturity*: in order to rate this driver, we have to refers to a well known method for assessing the maturity of a software organization, that is CMMI. We can estimate that our project is at `Maturity Level 3 - Defined`, that correponds to a PMAT at level `High` and a value of 3.12.

Notice that while the PREC and FLEX scale factors are largely intrinsic to a project and uncontrollable, the following three scale factors identify management controllables by which projects can reduce diseconomies of scale by reducing sources of project turbulence, entropy, and rework.

Cost Driver	**Factor**	**Value**
RELY	`Nominal`	1.00
DATA	`Nominal`	1.00
CPLX	`Nominal`	1.00
RUSE	`High`	1.07
DOCU	`High`	1.11
TIME	`Nominal`	1.00
STOR	`Nominal`	1.00
PVOL	`Nominal`	1.00
ACAP	`Nominal`	1.00
PCAP	`Nominal`	1.00
PCON	`Very High`	0.81
APEX	`Low`	1.10
PLEX	`Very Low`	1.19
LTEX	`Low`	1.09
TOOL	`Nominal`	1.00
SITE	`Low`	1.09
SCED	`Nominal`	1.00

Table 3: Cost drivers and the corresponding Factor and Value

– **Product factors**: they account for variation in the effort required to develop software caused by characteristics of the product under development. A product that is complex, has high reliability requirements, or works with a large testing database will require more effort to complete. There are five product factors, and complexity has the strongest influence on estimated effort.

- *Required Software Reliability*: this is the measure of the extent to which the software must perform its intended function over a period of time. If the effect of a software failure is only slight inconvenience then RELY is very low. If a failure would risk human life then RELY is very high. In our case, we think that the appropriate rate for this driver is `Nominal`, with a corresponding value of 1.00.
- *Data Base Size*: this cost driver attempts to capture the effect large test data requirements have on product development. DATA is capturing the effort needed to assemble and maintain the data required to complete test of our program. In our case, we can estimate that the rate for this driver is `Nominal`, with a corresponding value of 1.00.
- *Product Complexit*: complexity is divided into five areas: control operations, computational operations, device-dependent operations, data management operations, and user interface management operations. Basing on this, we can estimate that the rate for this driver is `Nominal`, with a corresponding value of 1.00.
- *Developed for Reusability*: this cost driver accounts for the additional effort needed to construct components intended for reuse on 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. In our case, we can estimate that the rate for this driver is `High`, with a corresponding value of 1.07.
- *Documentation Match to Life-Cycle Needs*: this cost driver is about the level of required documentation. The rating scale for this cost driver is evaluated in terms of the suitability of the project's documentation to its life-cycle needs. The rating scale goes from `Very Low`, if many life-cycle needs are left uncovered, to `Very High`, if the documentation is very excessive for life-cycle needs). In our case, we can estimate that the rate for this driver is `High`, with a corresponding value of 1.11

- **Platform factors**: the platform refers to the target-machine complex of hardware and infrastructure software. The factors have been revised and some additional platform factors were considered, such as distribution, parallelism, embeddedness, and real-time operations.
  - *Execution Time Constraint*: this is a measure of the execution

time constraint imposed upon a software system. The rating is expressed in terms of the percentage of available execution time expected to be used by the system or subsystem consuming the execution time resource. The rating ranges from `Nominal`, less than 50% of the execution time resource used, to `Extra High`, when 95% of the execution time resource is consumed. In our case, we can estimate that the rate for this driver is `Nominal`, with a corresponding value of 1.00

  - *Main Storage Constraint*: this rating represents the degree of main storage constraint imposed on a software system or subsystem. The rating ranges from `Nominal`, less than 50% , to `Extra High`, whan 95%. In our case, we can estimate that the rate for this driver is `Nominal`, with a corresponding value of 1.00

  - *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. 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. In our case, we can estimate that the rate for this driver is `Nominal`, with a corresponding value of 1.00

- **Personnel Factors**: after product size, people factors have the strongest influence in determining the amount of effort required to develop a software product. These factors are for rating the development team's capability and experience, not the individual.

  - *Analyst Capability*: analysts are personnel who work on requirements, high-level design and detailed design. The major attributes that should be considered in this rating are analysis and design ability, efficiency and thoroughness, and the ability to communicate and cooperate. Analyst teams that fall in the fifteenth percentile are rated `Very Low` and those that fall in the ninetieth percentile are rated as `Very High`. In our case, we can estimate that the rate for this driver is `Nominal`, with a corresponding value of 1.00

  - *Programmer Capability*: Current trends continue to emphasize the importance of highly capable analysts. However, we have a trend toward higher importance of programmer capability as well. Evaluation should be based on the capability of the programmers as a team rather than as individuals. Major factors which should be considered in the rating are ability, efficiency and thoroughness,

and the ability to communicate and cooperate. A `Very Low` rated programmer team is in the fifteenth percentile and a `Very High` rated programmer team is in the ninetieth percentile. In our case, we can estimate that the rate for this driver is `Nominal`, with a corresponding value of 1.00

- *Personnel Continuity*: the rating for this cost driver is in terms of the project's annual personnel turnover: from 3%, very high continuity, to 48%, very low continuity. In our case, we can estimate that the rate for this driver is `Very High`, with a corresponding value of 0.81
- *Applications Experience*: the rating for this cost driver is dependent on the level of applications experience of the project team developing the software system or subsystem. The ratings are defined in terms of the project team's equivalent level of experience with this type of application. A `Very Low` rating is for application experience of less than 2 months. A very `Very High` is for experience of 6 years or more. In our case, we can estimate that the rate for this driver is `Low`, with a corresponding value of 1.10
- *Platform Experience*: the Post-Architecture model broadens the productivity influence of platform experience by recognizing the importance of understanding the use of more powerful platforms, including more graphic user interface, database, networking, and distributed middleware capabilities. In our case, we can estimate that the rate for this driver is `Very Low`, with a corresponding value of 1.19.
- *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. Software development includes the use of tools that perform requirements and design representation and analysis, configuration management, document extraction, library management, program style and formatting, consistency checking, planning and control, etc. In addition to experience in the project's programming language, experience on the project's supporting tool set also affects development effort. A `Very Low` rating is given for experience of less than 2 months. A `Very High` rating is given for experience of 6 or more years. In our case, we can estimate that the rate for this driver is `Low`, with a corresponding value of 1.09.

- **Project Factors**: these factors account for influences on the estimated

effort such as use of modern software tools, location of the development team, and compression of the project schedule.

  - *Use of Software Tools*: the tool rating ranges from simple edit and code, `Very Low`, to integrated life-cycle management tools, `Very High`. In our case, we can estimate that the rate for this driver is `Nominal`, with a corresponding value of 1.00.
  - *Multisite Development*: the rate of this cost driver involves the assessment and judgement-based averaging of two factors: site collocation (from fully collocated to international distribution) and communication support (from surface mail and some phone access to full interactive multimedia). In our case, we can estimate that the rate for this driver is `Low`, with a corresponding value of 1.09.

- **General Factor**

  - *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. Accelerated schedules tend to produce more effort in the earlier phases to eliminate risks and refine the architecture, more effort in the later phases to accomplish more testing and documentation in parallel. Schedule compression of 75% is rated `Very Low`. A schedule stretch-out of 160% is rated `Very High`. Stretch-outs do not add or decrease effort. Their savings because of smaller team size are generally balanced by the need to carry project administrative functions over a longer period of time. n our case, we can estimate that the rate for this driver is `Nominal`, with a corresponding value of 1.0.

In the second phase of the estimation analysis, we used an online calculator in order to compute the effort, the total cost, the duration and the number of person estimated for the completion of our project. You can find the online calculator that we used at the following link: `http://csse.usc.edu/tools/COCOMOII.php`.

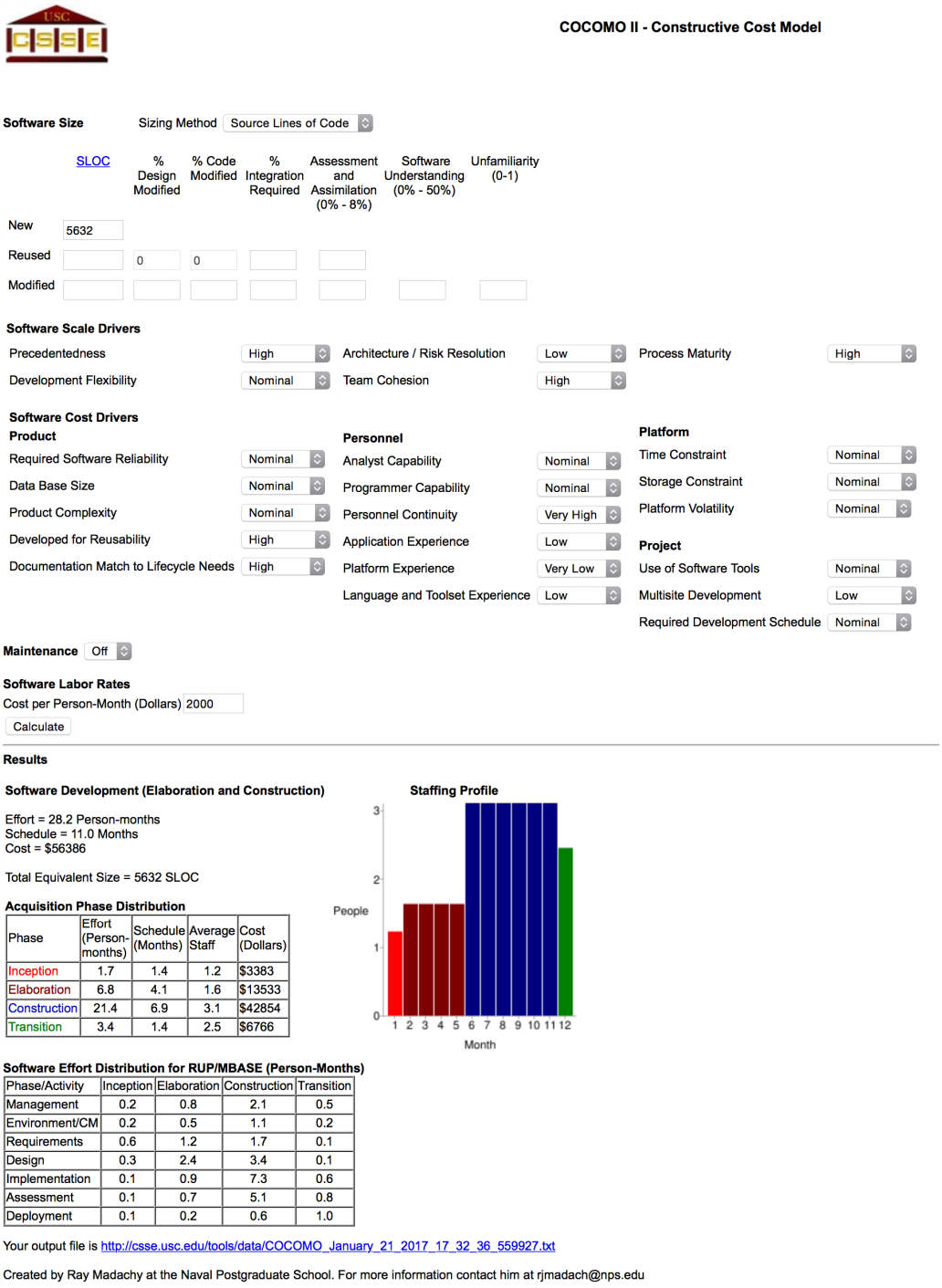

Figure 1: Grafical representation of effort, total cost, duration and number of person estimated for the completion of our project

In conclusion, we calculated this value for **effort**, **total cost**, **duration** and **team size**:

- **Effort**: 28.2 Person-months
- **Total Cost**: $56386, if we consider an average cost equals to $2000 per Person-months
- **Duration**: 11.0 Months
- **Team Size**:

$$\frac{\textbf{Effort}}{\textbf{Duration}} = \frac{28.2}{11.0} = 2.56\, members$$

4 Project Schedule and Resource Allocation

In this section, we are going to show a general, high-level project schedule and a general overview of the tasks division between the two members of the our team. We think it is more suitable to define more detailed schedules during the development phases of our project, in order to manage the internal organization in a better way, highlighting the difference of each phase.

During the scheduling definition, we decided to be as coherent as possible to the results of the analysis described in the previous sections (Section 2 and Section 3) but adapting them to our specific case, that is a team with two members.

The project scheduling consist in deciding how the work in a project will be organized as separate tasks, and when and how these tasks will be executed. So, we need to estimates the calendar time needed to complete each task, the effort required, who will work on the tasks that have been identified and the resources needed to complete each task.

In order to fully develop our project, we have identified 6 different tasks:

T1 : feasibility study;

T2 : analysis of requirement and specification;

T3 : design of the system;

T4 : coding and unit testing;

- **T4.1** : client side;
- **T4.2** : server side;
- **T4.3** : database;

T5 : integration test and system test;

T6 : deployment;

We decided to represent our schedule using a bar chart, since graphical notations are normally used to illustrate the project schedule. Moreover, we tried to grand an equal workload for each team member, so we have developed the following Gantt diagram, showing the distribution of every task during the months, forecasting a total duration of 14 months (since we want to balance the results obtained by the evaluation with COCOMO II: our team is composed by 2 people, instead of the 2.56 resulting from the estimation).

ACTIVITY		PERIOD OF TIME		
LEVEL 1	LEVEL 2	STARTING	DURATION (days)	ENDING
Task 1	Feasibility Study	16th October 2016	7	23rd October 2016
Task 2	Analysis of Requirement and Specification	16th October 2016	45	30th November 2016
Task 3	Design of the System	1st December 2016	62	31st January 2016
Task 4	Coding and Unit Testing Client	1st February 2017	181	31st July 2017
	Coding and Unit Testing Server	1st February 2017	242	30th September 2017
	Create and Populate Database	1st August 2017	61	30th September 2017
Task 5	Integration Test and System Test	1st October 2017	50	19th November 2017
Task 6	Deployment	20th November 2017	26	16 December 2017

Figure 2: Project Scheduling

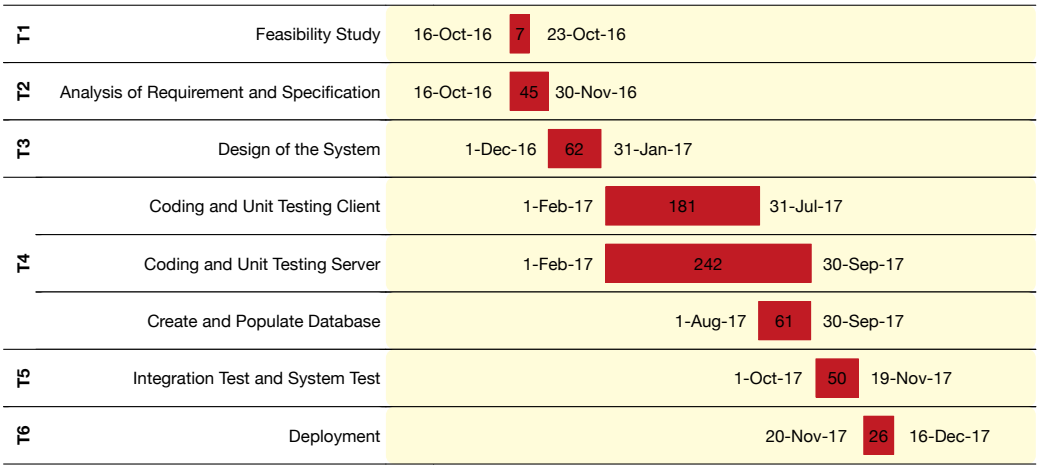

Figure 3: Graphical Represenattion of Project Scheduling

In Figure 4, we are showing the reader how the staff will be allocated in order to accomplish each task.

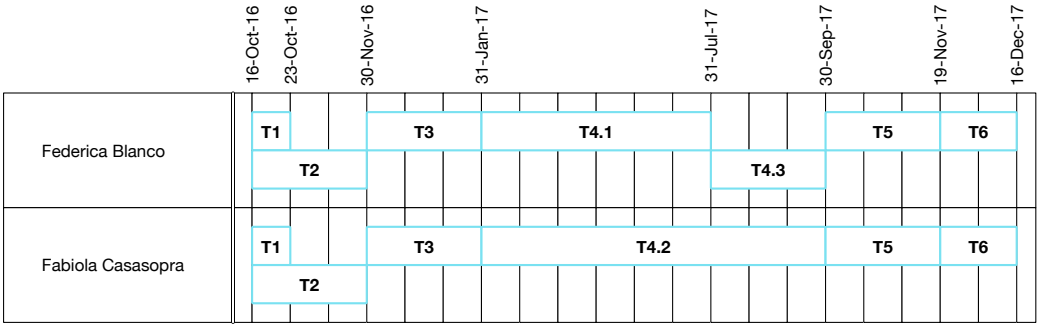

Figure 4: Staff Allocation

5 Project Risks

In this section, we are going to define which are the main risks for our project, how relevant they are and which could be an appropriate recovery action. A risk concerns future happenings, that may involve change in mind, opinion, actions, places, etc., that is everytime we make a choice, we have to deal with the uncertainty that our choice entails.

One of the main risks for our project is related to **staff size and experience**, that is associated with overall technical and project experience of the software engineers who will do the work. In our case, given the fact that our team is composed by only two people, we have risks related to the availability of all the team member and related to the time contraints of the schedule. Our team members have the same importance and, moreover, they have similar skills. A problem can be the **illness of a team member**, maybe even for a long period of time. A way to solve this problem can be that there is more overlap of work and everyone can therefore understand other's jobs. However, it remains the problem related to the schedule: a person alone can't carry out the work of two in the scheduled amount of time. Feasible solution mey be investigate buying-in components or the use of a program generator. Adding other people to the project should not be seen as the primary solution, since it is very difficult the integretion of a new tem member while the work is already in progress. In order to make it easier and faster, we can focus on the documentation, that should be as much complete and effective as possible.

Another risks that we have to take into considerations is related to our **development environment** that is associated with availability and quality of the tools to be used to build the project. In our case this could be a seriuos problem, since we have some dependency on external services and components. If there is a change in the terms and conditions of the Google Maps service, or even just a modification of the API itself, it could result in serious financial or technical problems. Also, faults in reusable software components have to be repaired before these components are reused and this could lead to significant issues on the financial and business side. A possible solution in order to mitigate this problem is to design the code so that it can be as portable as possible and with a great modularity and independence between components.

Moreover, we have to consider the risk related to the **process definition**, that is associated with the degree to which the software process has been defined and is followed. In our case, we could have problems related to the project scheduling. In this document, we provide an initial overall schedule; however, we didn't take into account all the possible issues that we can

encounter during the different phase of our project. So, a solution could be to allocate some extra time at the predicted end of each major activity, in order to not have strict deadlines and the possibility to make some adjustments.

Finally, we should not forget to consider the risk related to the **business impact** that is associated with constraints imposed by management or the marketplace. In our case, the problem coud be the fact that there is concurrency, since there are other systems that offer a solution similar to ours. Here, ther is also the risk related to **customer characteristics**, taht is associated with sophistication of the customer and the developer's ability to communicate with the customer in a timely manner. A way to solve this issue could be to dedicate some effort for marketing, maybe with the help of some experts of this field.

6 Appendix

In this section, we will give the information about the used tools, the hours of work done by the members of the group.

6.1 Used Tools

In this phase of the project, the following tools have been used:

- LaTeX and TeXMaker editor: to redact and to format this document
- Omnigraffle (`https://www.omnigroup.com/omnigraffle`): to make some graphs
- `http://csse.usc.edu/tools/COCOMOII.php`: the online calculator that we used for estimation

6.2 Working Hours

Last Name	First Name	Total Hours
Blanco	Federica	2 h
Casasopra	Fabiola	30 h