Politecnico di Milano
Scuola di Ingegneria Industriale e dell'Informazione
M.Sc. in Computer Science and Engineering
Dipartimento di Elettronica, Informazione e Bioingegneria

myTaxiService

Software Engineering 2 - Project

PP
Project plan

version 2.0

22nd February 2016

Authors:
Alberto Maria METELLI Matr. 850141
Riccardo MOLOGNI Matr. 852416

Academic Year 2015–2016

Contents

List of Figures

1 Introduction

1.1 Purpose

Project management represents a necessary condition for the success of any software project. Considered the intrinsic complexity of a software project, the difficulty in assessing the quality of the software and the organizational, economical, social and technical issues to be tackled in any enterprise environment, project management cannot be avoided. It comprises all the activities aimed to ensuring the delivering of the software on schedule and in accordance with context and requirements; in particular project planning, reporting, risk management and people management. The *PP* (*Project Planning*) is intended to be a trace of the myTaxiService project process. For academic reasons, this document is delivered as last assignment but it refers to both activities to be completed before the project initiation and activities to be performed during and at the end of the project. In particular this document is focused on *project planning* (estimation and scheduling of the process development, assignment of resources), *risk management* (definitions, strategies to tackle risk) and *cost estimation*. In an enterprise environment the first two are typically performed before the project starts while the latter is executed either after the requirement engineering or after the design phase.

This document is intended to be read by stakeholders in order to show the devised organization of resources and time and to have a general overview of the effort needed to carry out the project useful for the evaluation of the founding.

1.2 Scope

The *myTaxyService* is an application intended to optimize taxi service in a large city, making the access to service simpler for the passengers and ensuring a fair management of the taxi queues.

Passengers will be able to request a taxi either through a web application or a mobile app; of course the "traditional" ways to call for a taxi, like a phone call or stopping the taxi along the road, will be still available and integrated into the system to-be. The software will make the procedure of calling a taxi simpler (by using GPS information passenger doesn't need to know the address if the taxi is needed for the current position) and more usable (passenger will be provided with information about the waiting time). Moreover, by means of the application, the passenger can reserve a taxi for a certain date and time, specifying the origin and the destination of the ride.

Taxi drivers will use a mobile app to inform the system about their availability and to confirm that they are going to take care of a call (or to reject it for any reason). The software will make the taxi management more efficient: the system will be able to identify the position of each taxi by using GPS; the city will be divided in virtual zones and a suitable distribution of the taxi among the zones will automatically be computed.

1.3 Definitions, Acronyms, Abbreviations

In this paragraph all the terms, acronyms and abbreviations used in the following sections are listed.

1.3.1 Definitions

- *Request*: the action performed by the passenger of calling a taxi for the current position.
- *Confirmed request*: a request that has been accepted by a taxi driver.
- *Reservation*: the action performed by the passenger of booking a taxi for a specific address and specific date and time.

- *Waiting time*: an estimation of the time required to taxi driver to get to passenger's position.
- *Taxi code*: a unique alphanumerical identifier of the taxi.
- *Available taxi queues*: data structures used to store the references of the available taxis, also used to select the taxis to which forward a request.
- *Automatic geolocalization*: a system that provides the geographic coordinates of the user. For this document it can be either a GPS system or browser geolocalization.
- *Passengers' application*: the applications used by passengers to access to TS system. For this document it can be either PMA or PWA.
- *Login credentials*: username and password.
- *Notification*: communication from TS to taxi driver to move to a specific zone.

1.3.2 Acronyms

- TS: myTaxiService.
- PMA: Passenger mobile application.
- PWA: Passenger web application.
- TMA: Taxi driver mobile application.
- SLOC: Source Lines Of Code.
- FP(s): Function Point(s).

Other acronyms are explained in the corresponding sections.

1.4 Reference documents

[1] The assignment of the *myTaxiService*.

[2] RASD (Requirements Analysis and Specification Document) of the *myTaxiService*.

[3] DD (Design Document) of the *myTaxiService*.

[4] ITPD (Integration Testing Plan Document) of the *myTaxiService*.

[5] Software Engineering 2 course slides.

[6] COCOMO II Model Definition Manual `http://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_modelman2000.0.pdf`

[7] Function Point Languages Table Version 5.0 `http://www.qsm.com/resources/function-point-language`

1.5 Document Structure

This document is composed of six sections and an appendix.

- The first section, this one, is intended to define the goal of the document, a very high level description of the main functionalists of the *myTaxiService* system and the resources used to draw up this document.
- The second section describes some preliminary results for cost estimation. A brief theoretical introduction about Function Points method will be provided and then it will be applied to the myTaxiService system in order to estimate the complexity of the project and derive the expected number of SLOC.
- The third section is devoted to cost estimation. A general method, called COCOMO, will be presented and applied to the specific case of myTaxiService in order to estimate the effort of the project, the estimated time needed for the fulfillment and the number of people required.
- In the fourth section we discuss project planning and in particular, according to the activities identified we propose a possible schedule, according to both the real deadlines and some reasonable considerations. We also provide a graphical representation of the schedule by means of a Gantt chart.
- The fifth section is still devoted to project planning but the focus is on the resource allocation. Some general consideration will be given and the allocation strategy explained, we will also use a resource chart to clarify the results.
- The sixth section is devoted to risk management. The main risks affecting the myTaxiService project will be stated according to the traditional classification. Some consideration about the strategy adopted to tackle them will be given.
- The appendix contains a brief description of the tools used to produce this documents, the number of hours each group member has worked towards the fulfillment of this deadline and the revision history.

2 Function points

2.1 Introduction to cost estimation

Estimating the cost of a software project is a non trivial task. Many aspects contribute to determine the effort put in a software project and relationships among the characteristics of the project team (number of people, type of organization, ...), the features of the project itself (complexity) and the environmental influences are typically intricate and difficult to be estimated with an acceptable degree of approximation. Therefore often we rely on the *experience-based techniques* where the estimation is made by an experienced manager on the basis on the past projects and the application domain, sometimes this task is performed by a team of experts (community-based estimation). If no experts are available, or the domain of the application in too specific only *algorithmic cost modeling* is allowed. Those techniques are based on a formulated approach to compute the project effort based on estimates of product attributes. We will adopt the *Function Points* technique that was proposed by Allan Albrecht at IBM in 1965 and then COCOMOII that was developed around '80 based on the statistical analysis performed by Barry Boehn on the basis on many real projects coming from various domains.

2.2 Function points approach

There is an underlying hypothesis behind function points: the dimension of software can be characterized by *abstraction*. Therefore after the architectural design (early in the project life cycle), when the product model is almost clear, a rough evaluation of the size of the software can be performed. Albrecht's method identifies and counts the number of function types within the software (actually its model), those types constitutes the "external representation" of an application, that is, its functionality as defined from an abstract point of view. The types of functionalists are the following:

- *Internal Logical File* (ILF): homogeneous set of data used and managed by the application.
- *External Interface File* (EIF): homogeneous set of data used by the application but generated and maintained by other applications.
- *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.

The measure of the size of the software is given by the weighed average of the function points (number of each function type listed above), according to the following predefined table.

Function type	**Weight**		
	Simple	*Average*	*Complex*
EI (External Inputs)	3	4	6
EO (External Outputs)	4	5	7
EQ (External Inquiries)	3	4	6
ILF (Internal Logical Files)	7	10	15
ELF (External Logical Files)	5	7	10

The resulting sum is called UFP (Unadjusted Function Points). This value can be further manipulated with the "adjustment" formula to get an estimation of the effort, however the result is usually little significant, therefore it is suggested to use the UFP in combination with other effort estimation algorithms like COCOMO II.

2.3 Function point count

In this section we present the function point count; for each type of functionality we list the ones we have identified within the myTaxiService system with the corresponding complexity and a rational for our choice. According to the Albrecht definition, this process is performed by abstraction therefore the main resource is the RASD in particular the high level class diagram for the logical files and the use cases for the inputs, outputs and inquiries.

2.3.1 ILF (Internal logical files)

The application includes a number of ILFs that will be used to store information about:

- *passengers*, in particular username, password, lastname, firstname, email and address;
- *taxi drivers*, in particular username, password, firstname and lastname;
- *taxis*, in particular the plate number, the code, the number of seats and the current state;
- *requests*, in particular the date and time in which the request is sent, the number of passengers, the location (geographical coordinates and the address) of the passenger, possibly the waiting time and the taxis assigned to the request itself;
- *reservations*, in particular the date and time in which the request is sent, the number of passengers, the location (geographical coordinates and the address) of both origin and destination and the corresponding attached request;
- *zones*, in particular the name of the zone, the estimation of the requests per minute and the adjacency relation among zones;
- *queues*: in particular the proper size of the queue (and also the minimum and maximum number of taxis allowed) and the taxis belonging to the queue with the corresponding position.

Each of these entities has a simple structure with a limited number of fields except for the queue which requires a quite articulated structure to manage positions and sizes of queues. Thus, we select medium complexity for the latter and simple for the other ones.

$$ILF = 6 \cdot 7 + 1 \cdot 10 = 52FPs$$

2.3.2 ELF (External logical files)

myTaxiService has to interact with external systems, in particular with the GPS and the GoogleMaps API therefore we can identify the following ELFs:

- *Address validation* requires the interaction with the GoogleMaps API in order to check whether a string typed by the user corresponds to a valid address.
- *Coordinate/Address translation* requires the interaction with the GoogleMaps API in order to convert a coordinate retrieved by the GPS into an address and viceversa.

- *Travelling time* requires the interaction with the GoogleMaps API in order to compute the waiting time.
- *Passenger's geographic coordinates* requires the interaction with the GPS system installed on the passenger smartphone or with the browser geolocalization to retrieve the passenger's position.
- *Taxi geographic coordinates* requires the interaction with the GPS installed on each taxi in order to find the position of the taxi itself.

All those data items are very compact and they share the same simple structure, so a simple complexity should be appropriate.

$$ELF = 5 \cdot 5 = 25FPs$$

2.3.3 EI (External inputs)

Since myTaxiService is an application characterized of having a high degree of interaction with the final user, we can identify the following EI.

- *Registration*: this function requires the exchange of a relevant amount of information (username, password, lastname, firstname, email and address), in addition some checks has to be performed (like check that the username is not already used), so we can consider medium complexity with a contribution of 4 FPs.
- *Login/Logout*: this function is standard and requires exchange of basic structured information and simple operations, so it can be considered simple with a contribution of 3 FPs.
- *Request*: this function requires the insertion of some data (like address), the interaction with external systems (like GPS and GoogleMaps API) and with the DBMS, therefore it can be considered complex with a contribution of 6 FPs.
- *Taxi selection*: this function requires non trivial elaborations related to the algorithm used to select the taxis to fulfill a request/reservation; it requires interaction with the DBMS and external systems therefore it can be considered complex with a contribution of 6 FPs.
- *Taxi queue management*: this function requires the execution of the algorithm described in the design document for the taxi movement, in addition it requires to interact with external systems and the DBMS therefore it can be considered complex with a contribution of 6 FPs.
- *Reservation*: this function requires interaction with the DBMS and with external systems, it has also to instantiate a new request associated to the reservation and to perform validity checks on the inserted data, therefore it can be considered complex with a contribution of 6 FPs.
- *Modify reservation*: this function can be considered an extension of Reservation, adding a small new functionality, therefore it can be considered simple with a contribution of 3 FPs.
- *Cancel reservation*: this function can be considered an extension of Reservation, adding a small new functionality, therefore it can be considered simple with a contribution of 3 FPs.
- *Request evaluation*: this function requires to interact with the TMA and the analysis of the taxi queues, so it can be considered medium complexity with a contribution of 4 FPs.
- *Inform about availability*: this function requires to check some conditions to validate the state changing of the taxi driver, it can be considered medium complexity with a contribution of 4 FPs.

- *Insert phone request*: this function is just reduced to Request therefore it can be considered simple with a contribution of 3 FPs.

$$EI = 4 \cdot 3 + 3 \cdot 4 + 4 \cdot 6 = 48FPs$$

2.3.4 EO External output

The only external outputs generated by the myTaxiService are:

- *Movement notification:* that function is performed by the system to communicate to the taxi driver the notification of the movement, since it requires the evaluation of the taxi queues it can be considered medium complexity with a contribution of 5 FPs.
- *Request notification:* this function is performed in order to communicate to the taxi driver a request to be carried out, it requires only to send information about the address of the passenger therefore it can be considered simple with a contribution of 4 FPs.

$$EI = 1 \cdot 4 + 1 \cdot 5 = 9FPs$$

2.3.5 EQ External inquiries

We identified the following EQs.

- *Visualize request info*: this function allows passenger, both registered or not, to visualize the information about the last request including waiting time and the number of incoming taxi.
- *Visualize previous reservations*: this function allows registered passengers to visualize the previous sent reservations.
- *Visualize previous requests (taxi driver)*: this function allows the taxi driver to visualize the previous requests carried out.

All these functions requires the interaction with the DBMS therefore they can be considered medium complexity.

$$EI = 3 \cdot 4 = 12FPs$$

2.3.6 UFP Unadjusted Function Points count

Here we summarize the number of function points identified in every category and we provide the UFP (Unadjusted Function Points count) which could possibly be adjusted to take into account organizational aspects and get an estimation of the effort, however this approach is typically very imprecise therefore we will use UFP in combination with the COCOMO approach.

Function type	**FPs**
ILF (Internal Logical Files)	52
ELF (External Logical Files)	25
EI (External Inputs)	48
EO (External Outputs)	9
EQ (External Inquiries)	12

Thus, we get at the end the value of UFP

$$UFP = ILF + ELF + EI + EO + EQ = 146FPs$$

Here we provide an histogram representing the number of function points for each category.

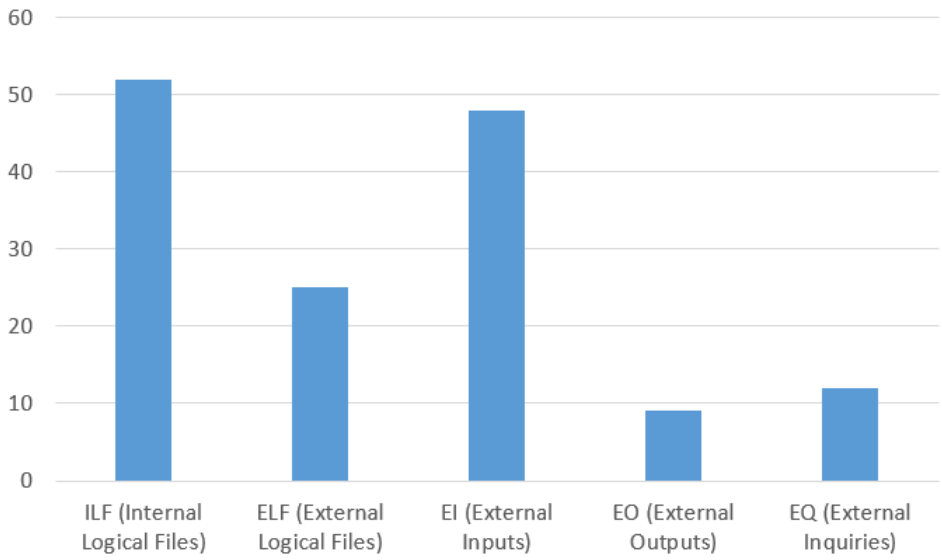


Figure 1: Function Points Histogram

2.4 Lines of code count

The Unadjusted Function Points (UFP) can be used to provide an estimation of the number of lines of code (SLOC) of the final project according to the specific implementation language chosen. Instead of referring to the "traditional" table proposed by Jones 1996 for the conversion factor, which by the way does not include a value for JEE, we adopt the more recent Function Points Language Table proposed in [7]. myTaxiService is intended to be implemented with JEE, the table provides a range for the conversion factor SLOC/FP

Language	*Average*	*Median*	*Low*	*High*
J2EE	46	49	15	67

Not having precise information about the complexity of the implementation we think an average value should be proper, so $SLOC/FP = 46$ and we get the value of SLOC.

$$SLOC = SLOC/FP \cdot UFP = 46 \cdot 146 = 6716$$

Notice that this value does not capture, for the most part, the client side of the application and the web server. Since both PMA, TMA and web server do not perform meaningful functions at requirement level but just auxiliary functions they are not highlighted in the function points count, however from an implementation point of view they might require a relevant effort.

3 COCOMO II

In this section, after a brief introduction to COCOMO II, we present the application of the method to myTaxiService system in order to get the effort estimation and the expected duration of the project.

3.1 COCOMOII approach

Mainly taken from [6].

COCOMO II is an updated version of COCOMO 81 which is based on a linear regression model of the function relating the size of the project and the effort. However the original version was affected by old time and non realistic assumptions on the *software lifecycle*, in particular it assumed that the development process followed the traditional waterfall scheme, that requirements were supposed to be stable during the project duration and that the documentation was written incrementally. COCOMO II is able to overcome those limitation and provide a reasonable estimation of both the effort and the duration of the project. In the following subsection we will describe the model.

3.1.1 Effort Equation

In COCOMO II *effort* is expressed as *Person-Months* (PM). [1]The *Effort equation* allows to compute the effort as a function of the *size* of software development expressed in KSLOC (thousand of SLOC)[2] that can be obtained by the Function Points method, a constant, A, an exponent, E, and a number of values called *effort multipliers* (EM). The number of effort multipliers depends on the model.

$$Effort = PM = A \cdot Size^{E} \cdot \prod_{i=1}^{n} EM_i$$

where $A = 2.94$ for COCOMO II. We now present how to compute E and each EM_i[3].

3.1.2 Software Scale Drivers

The exponent E in the effort equation is an aggregation of five *scale factors* (SF, also referred to *scale drivers* SD) is related to economic, organizational and technical aspects of the environment in which the project is going to be developed. In particular if account for the relative economies or diseconomies of scale encountered for software projects of different sizes [Banker et al. 1994]. [4]

The value of E is a contribution of the following scale factors.

PREC *Precedentedness*: reflects the previous experience of the organization with this type of project. Very low means no previous experience, Extra high means that the organization is completely familiar with this application domain.

[1]A person month is the amount of time one person spends working on the software development project for one month. COCOMO II treats the number of person-hours per person-month, PH/PM, as an adjustable factor with a nominal value of 152 hours per Person-Month.

[2]The SLOC in COCOMO represent only the line of codes that are going to be delivered and they have to be computed as *logical* SLOC.

[3]Sometimes the overall contribution of the EM_i is called EAF (Effort Adjustment Factor) which is trivially defined as $EAF = \prod_{i=i}^{n} EM_i$.

[4]If E<1.0, the project exhibits economies of scale. If the product's size is doubled, the project effort is less than doubled. The project's productivity increases as the product size is increased. Some project economies of scale can be achieved via project-specific tools (e.g., simulations, testbeds). For small projects, fixed start-up costs such as tool tailoring and setup of standards and administrative reports are often a source of economies of scale. If E=1.0, the economies and diseconomies of scale are in balance. This linear model is often used for cost estimation of small projects. If E>1.0, the project exhibits diseconomies of scale.

FLEX *Development Flexibility*: reflects the degree of flexibility in the development process. Very low means a prescribed process is used; Extra high means that the client only sets general goals.

RESL *Architecture / Risk Resolution*: reflects the extent of risk analysis carried out. Very low means little analysis, Extra high means a complete a thorough risk analysis.

TEAM *Team Cohesion*: reflects how well the development team know each other and work together. Very low means very difficult interactions, Extra high means an integrated and effective team with no communication problems.

PMAT *Process Maturity*: reflects the process maturity of the organization. The computation of this value depends on the CMM (Capability Maturity Model)[5] Questionnaire but an estimate can be achieved by subtracting the CMM process maturity level from 5.

The following table provides the rating levels for the COCOMO II scale factors. The selection of scale factors is based on the rationale that they are a significant source of exponential variation on a project's effort or productivity variation. Each scale factors has a range of rating levels, from Very Low to Extra High and each rating level has a weight.

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

To have a complete description of the meaning of each scale factor, please refer to [6] where also assessment methods are proposed.

Once the value of each scale factor SF_j is determined the value of the exponent E can be computed using the following formula.

$$E = B + 0.01 \cdot \sum_{j=i}^{5} SF_j$$

[5]CMM is a development model created to represent the "maturity" of software development process as the degree of formality and optimization of processes, from ad hoc practices, to formally defined steps, to managed result metrics, to active optimization of the processes. Nowadays the extension CMMI (Capability Maturity Model Integration) is used.

where $B = 0.91$ for COCOMO II. Notice that if all scale factors are judged as Nominal the resulting exponent value is $E = 1.0997$.

3.1.3 Software Cost Drivers

Cost drivers are used to capture characteristics of the software development that affect the effort to complete the project. All COCOMO II cost drivers have qualitative rating levels that express the impact of the driver on development effort. These ratings can range from Extra Low to Extra High. Each rating level of every multiplicative cost driver has a value, called an *effort multiplier* (EM), associated with it. The EM value assigned to a multiplicative cost driver's nominal rating is 1.00. All those are intended to be evaluated in post-architecture/early-design phase and they are organized in categories, for the complete description refer to [6].

- *Product* factors 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.

RELY	Required Software Reliability
DATA	Data Base Size
CPLEX	Product Complexity
RUSE	Developed for Reusability
DOCU	Documentation Match to Lifecycle Needs

- *Personnel* factors have the strongest influence in determining the amount of effort required to develop a software product. The Personnel Factors are for rating the development team's capability and experience – not the individual. These ratings are most likely to change during the course of a project reflecting the gaining of experience or the rotation of people onto and off the project.

ACAP	Analyst Capability
PCAP	Programmer Capability
PCON	Personnel Continuity
AEXP	Application Experience
PEXP	Platform Experience
LTEX	Language and Toolset Experience

- *Platform* factors refers to the target-machine complex of hardware and infrastructure software (previously called the virtual machine). The factors have been revised to reflect this as described in this section. Some additional platform factors were considered, such as distribution, parallelism, embeddedness, and real-time operations.

TIME	Time Constraint
STOR	Storage constraint
PVOL	Platform Volatility

- *Project* 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.

TOOL	Use of Software Tools
SITE	Multisite Development
SCED	Required Development Schedule

3.1.4 Schedule estimation

The *Schedule equation* allows to determine the *Time to Develop*, TDEV, expressed in number of months required to complete the software project.

$$Duration = TDEV = C \cdot (PM)^{(D+0.2(E-B))}$$

where $C = 3.67$, $D = 0.28$, $B = 0.91$.

C is a TDEV coefficient that can be calibrated, PM is the estimated effort, D is a TDEV scaling base exponent that can also be calibrated. E is the effort scaling exponent derived as the sum of project scale factors and B as the calibrated scale factor base-exponent. If we also know the average monthly salary S of the personnel we can estimate the personnel cost.[6]

$$Cost = PM \cdot S$$

3.1.5 Number of people estimation

Using PM and TDEV calculated before we can give an estimation of the number of required people.

$$N = \frac{PM}{TDEV}$$

3.2 COCOMO II calculation

In this subsection we perform the COCOMO II estimation starting from the SLOC determined in the previous stage thanks to Function Points approach. Given the academic context and the fact that implementation was not performed we will try to both perform the calculation with

- all SFs and CDs rated as nominal and
- SFs and CDs rated according our opinion with respect to a reasonable implementation phase.

Then we compere the results obtained by the two different approaches. We also assume that each person involved in the project is payed 2500$ per month. To easily compute the PM and TDEV we refer to the online tool `http://csse.usc.edu/tools/COCOMOII.php`.

3.2.1 COCOMO II with all SFs and CDs nominal

If we assume to set all SFs and CDs to nominal we get the parameters $E = 1.0997$ and $EAF = 1$ therefore knowing that $Size = KSLOC = 6.716$ we get the following results.

$$Effort = PM = 2.94 \cdot (6.716)^{1.0997} \cdot 1 = 23.9 \text{ person} - \text{month}$$

$$Duration = TDEV = 3.67 \cdot (23.9)^{0.31794} = 10.5 \text{ months}$$

$$Cost = 23.9 \cdot 2500\$ = 59684\ \$$$

[6] Notice that this is very different from the overall cost of the project since we do not take into account overheads (buildings, heating, lighting, ...).

$$N = \frac{23.9}{10.5} = 2.276 \text{ person}$$

We report some tables and graphs taken from the web site.

Acquisition Phase Distribution

Phase	*Effort (Person-months)*	*Schedule (Months)*	*Average Staff*	*Cost (Dollars)*
Inception	1.4	1.3	1.1	3581
Elaboration	5.7	3.9	1.5	14324
Construction	18.1	6.5	2.8	45360
Transition	2.9	1.3	2.2	7162

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

Phase/Activity	Inception	Elaboration	Construction	Transition
Management	0.2	0.7	1.8	0.4
Environment/CM	0.1	0.5	0.9	0.1
Requirements	0.5	1.0	1.5	0.1
Design	0.3	2.1	2.9	0.1
Implementation	0.1	0.7	6.2	0.5
Assessment	0.1	0.6	4.4	0.7
Deployment	0.0	0.2	0.5	0.6

Staffing profile

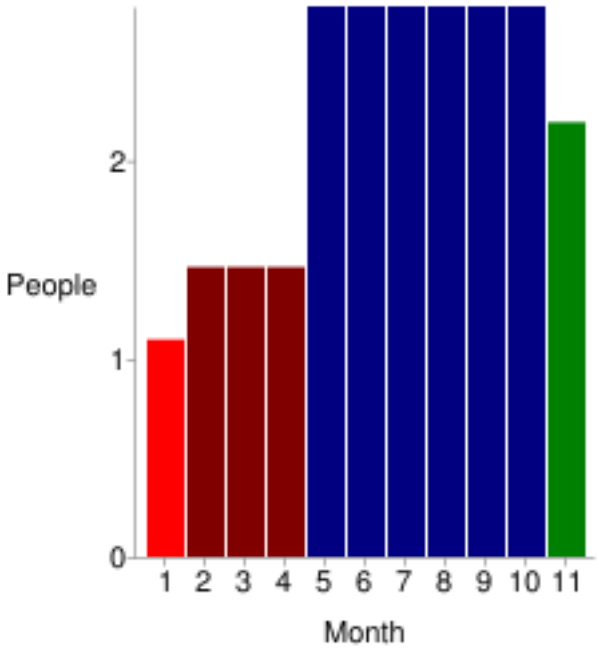


Figure 2: COCOMO II histogram - all nominal

From now on we will discuss how to rate Scale Drivers and Cost Drivers for myTaxiService.

3.2.2 Software Scale Drivers

In the following table we describe for each scale driver the motivations behind the rating level choice, we refer to the descriptors stated in [6].

Scale Factor	*Motivation*	*Rating level*	*Weight*
PREC	• Organizational understanding of product objectives: Thorough • Experience in working with related software systems: Moderate • Concurrent development of associated new hardware and operational procedures: Moderate • Need for innovative data processing architectures, algorithms: Some	Nominal	3.72
FLEX	• Need for software conformance with preestablished requirements: Basic (many aspects were not clearly specified) • Need for software conformance with external interface specifications: Considerable (integration with some external systems needed) • Combination of inflexibilities above with premium on early completion: High	High	2.03
RESL	• Percent of development schedule devoted to establishing architecture, given general product objectives: 25 • Percent of required top software architects available to project: 80 • Tool support available for resolving risk items, developing and verifying architectural specs: Some • Level of uncertainty in key architecture drivers: mission, user interface, COTS, hardware, technology, performance: Considerable • Number and criticality of risk items: Some	High	2.83

Scale Factor	*Motivation*	*Rating level*	*Weight*
TEAM	• Consistency of stakeholder objectives and cultures: Strong • Ability, willingness of stakeholders to accommodate other stakeholders' objectives: Considerable • Experience of stakeholders in operating as a team: Considerable • Stakeholder teambuilding to achieve shared vision and commitments: Considerable	Very High	1.10
PMAT	Answering to the KPAs (Key Process Area) we get a value of KPA thus the corresponding EPML (Equivalent Processing Maturity Level) is 1.5	Nominal	4.68

We obtain a total value for the exponent E of

$$E = B + 0.01 \cdot \sum_{j=i}^{5} SF_j = 1.0536$$

Notice that the value of the exponent is smaller with respect to the one obtained setting all SFs to nominal, this means that we are benefiting of the quality of the environment in which the software is developed.

3.2.3 Software Cost Drivers

In the following table we describe for each cost driver the motivations behind the rating level choice, we refer to the descriptors stated in [6]. Whenever the information for selecting the most appropriate rating level is not available we adopt nominal value.

Cost Driver	*Motivation*	*Rating level*	*Effort multiplier*
RELY	Moderate, easily recoverable losses.	Nominal	1.00
DATA	10 ≤Testing DB bytes/Pgm SLOC<100	Nominal	1.00
CPLEX	• *Control Operations*: Highly nested structured programming operators with many compound predicates. Queue and stack control. Homogeneous, distributed processing. Single processor soft real-time control. • *Computational Operations:* Use of standard math and statistical routines. Basic matrix/vector operations. • *Device dependent Operations*: Operations at physical I/O level (physical storage address translations; seeks, reads, etc.). Optimized I/O overlap. • *Data Management Operations*: Multi-file input and single file output. Simple structural changes, simple edits. Complex COTS-DB queries, updates. • *User Interface Management Operations*: Simple use of widget set.	High	1.17
RUSE	Across program[7]	High	1.07
DOCU	Right-sized to life-cycle needs	Nominal	1.00
ACAP	Analysts in the 75th percentile	High	0.85
PCAP	Programmes in the 75th percentile	High	0.88
PCON	Personell turnover 3% / year	Very High	0.81
AEXP	2 months	Very Low	1.22
PEXP	2 months	Very Low	1.19
LTEX	1 year	Nominal	1.00
TIME	85% use of available execution time	Very High	1.29
STOR	50% use of available storage	Nominal	1.00
PVOL	Major change every 12 mo.; Minor change every 1 mo.	Low	0.87
TOOL	basic lifecycle tools, moderately integrated	Nominal	1.00
SITE	• *Collocation Descriptors*: Same city or metro. area • *Communications Descriptors*: Wideband electronic communication.	High	0.93
SCED	Shedule compression 130% of nominal	High	1.00

We obtain a total value for the EAF of

$$EAF = \prod_{i=i}^{n} EM_i = 1.149$$

Note that the value of EAF is grater than the one obtained assuming all CDs as nominal, this mean that CDs turn out to affect negatively the overall effort.

3.2.4 Effort estimation and schedule estimation

Considering the values computed above, $E = 1.0536$ and $EAF = 1.149$, and knowing that $Size = KSLOC = 6.716$ we get the following results.

$$Effort = PM = 2.94 \cdot (6.716)^{1.0536} \cdot 1.149 = 25.1 \text{ person} - \text{month}$$

$$Duration = TDEV = 3.67 \cdot (25.1)^{0.30872} = 13.8 \text{ months}$$

$$Cost = 25.1 \cdot 2500\$ = 62832\ \$$$

$$N = \frac{25.1}{13.8} = 1.82 \text{ person}$$

Note that this second result is just slightly different with respect to the one obtained with all nominal SFs and CDs, in fact the effort here is grater of 1.2 person-month and also the duration is increased of 3.3 months; however the required number of people is still around 2.

We report some tables and graphs taken from the web site.

Acquisition Phase Distribution

Phase	*Effort (Person-months)*	*Schedule (Months)*	*Average Staff*	*Cost (Dollars)*
Inception	1.5	1.7	0.9	3770
Elaboration	6.0	52.	1.2	15080
Construction	19.1	8.6	2.2	47753
Transition	3.0	1.7	1.7	7540

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

Phase/Activity	Inception	Elaboration	Construction	Transition
Management	0.2	0.7	1.9	0.4
Environment/CM	0.2	0.5	1.0	0.2
Requirements	0.6	1.1	1.5	0.1
Design	0.3	2.2	3.1	0.1
Implementation	0.1	0.8	6.5	0.6
Assessment	0.1	0.6	4.6	0.7
Deployment	0.0	0.2	0.6	0.6

Staffing profile

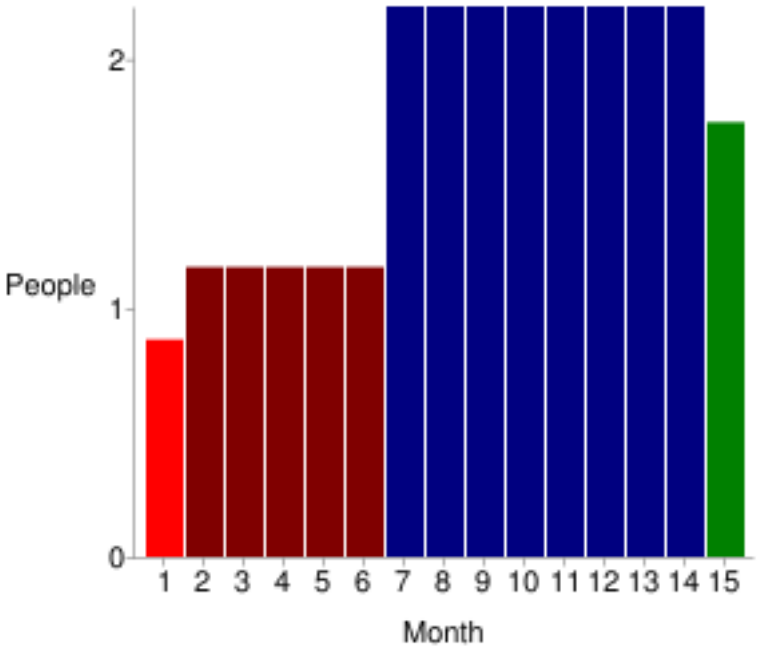


Figure 3: COCOMO II histogram

4 Tasks and schedule

Project planning is typically a in iterative process that starts at project initiation and continues evolving. In order to set up a suitable *project schedule*, and in particular decide the time and resource allocation, we need to identify:

- *tasks*: activities that must be completed to achieve the project goal and the corresponding dependencies;
- *milestones*: points in time in which you can assess the progress of the project;
- *deliverables*: work products delivered to the stakeholders.

This phase has to be driven by the *software development process* chosen and by the main features of the specific application we are going to design. In this section we provide a description of the tasks, their dependencies and a graphical representation of a schedule proposal.

4.1 Tasks

The following schedule tries to be a compromise between the need of producing a realistic program of the tasks and be compatible with the tasks we really performed during the semester. For what concerns the tasks related to RASD, DD and ITDP the dates are the real ones, for PP we assumed to having performed it as the first activity (as it is in a realistic context). Since we didn't perform the implementation of the software we couldn't exploit real data; to be realistic we allocate a duration of about three months. Also for code inspection and testing we choose reasonable duration. We can interpret this program as a schedule for an hypothetical annual project assigned to us at the beginning of the academic year. We didn't use the duration value provided by COCOMO II analysis since, also according to the applications made in the examples by the previous years, it tends to overestimate project duration and not to be coherent with the "breakneck speed" we are asked to keep at Politecnico.

The following table shows for each task the id, name, star and end date, the duration expressed in working days and the possible precedences. Task are grouped into macro tasks.

Task id	*Name*	*Start*	*End*	*Duration*	*Precendences*
T1	**Project planning**	01/10/2015	14/10/2015	10	-
T11	Project scheduling	01/10/2015	09/10/2015	7	-
T12	Resource allocation	12/10/2015	14/10/2015	3	T11
T13	Risk planning	05/10/2015	14/10/2015	8	-
M1	PP	15/10/2015	15/10/2015	-	
T2	**Requirement analysis and specification**	**15/10/2015**	**02/11/2015**	**13**	M1
T21	Requirement engineering	15/10/2015	21/10/2015	5	
T22	Use case design	22/10/2015	02/11/2015	8	T22
T23	High level data design	26/10/2015	28/10/2015	3	
T24	Alloy modeling	29/10/2015	02/11/2015	3	T23
M2	RASD	03/11/2015	03/11/2015	-	
T3	**Acceptance test plan design**	**03/11/2015**	**11/11/2015**	**7**	M2
T4	**Design**	**09/11/2015**	**04/12/2015**	**20**	M2
T41	Architectural design	09/11/2015	04/12/2015	20	
T42	Algorithm design	16/11/2015	04/12/2015	15	
T43	User interface design	30/11/2015	04/12/2015	5	
M3	DD	07/12/2015	07/12/2015	-	
T5	**Unit test plan design**	**07/12/2015**	**15/12/2015**	**7**	M2
T6	**Integration test plan design**	**07/01/2016**	**21/01/2016**	**11**	M2
M4	ITDP	22/01/2016	22/01/2016	-	
T7	**Implementation**	**07/12/2015**	**05/03/2016**	**65**	M2
T71	Components implementation	07/12/2015	05/03/2016	65	
T72	Subsystem implementation	08/02/2016	04/03/2016	20	
T8	**Code Inspection**	**07/03/2016**	**23/03/2016**	**13**	
T81	Manual inspection	07/03/2016	17/03/2016	9	T71
T82	Automated code inspection	18/03/2016	23/03/2016	4	T81
M5	CI	24/03/2016	24/03/2016	-	
T9	**Testing**	**24/03/2016**	**28/04/2016**	**26**	
T91	Unit testing	24/03/2016	08/04/2016	12	T5, T71
T92	Integration testing	11/04/2016	18/04/2016	6	T91, T6, T7
T93	System and performance testing	19/04/2016	20/04/2016	2	T92
T94	Acceptance testing	21/04/2016	28/04/2016	6	T93, T3
T10	**Deployment**	**29/04/2016**	**04/05/2016**	**4**	T9

The following is the same table in which just macro tasks and the delivarables are represented.

Task id	*Name*	*Start*	*End*	*Duration*
T1	**Project planning**	01/10/2015	14/10/2015	10
M1	PP	15/10/2015	15/10/2015	-
T2	**Requirement analysis and specification**	**15/10/2015**	**02/11/2015**	**13**
M2	RASD	03/11/2015	03/11/2015	-
T3	**Acceptance test plan design**	**03/11/2015**	**11/11/2015**	**7**
T4	**Design**	**09/11/2015**	**04/12/2015**	**20**
M3	DD	07/12/2015	07/12/2015	-
T5	**Unit test plan design**	**07/12/2015**	**15/12/2015**	**7**
T6	**Integration test plan design**	**07/01/2016**	**21/01/2016**	**11**
M4	ITDP	22/01/2016	22/01/2016	-
T7	**Implementation**	**07/12/2015**	**05/03/2016**	**65**
T8	**Code Inspection**	**07/03/2016**	**23/03/2016**	**13**
M5	CI	24/03/2016	24/03/2016	-
T9	**Testing**	**24/03/2016**	**28/04/2016**	**26**
T10	**Deployment**	**29/04/2016**	**04/05/2016**	**4**

4.2 Milestones

Here are the expected milestone of the project.

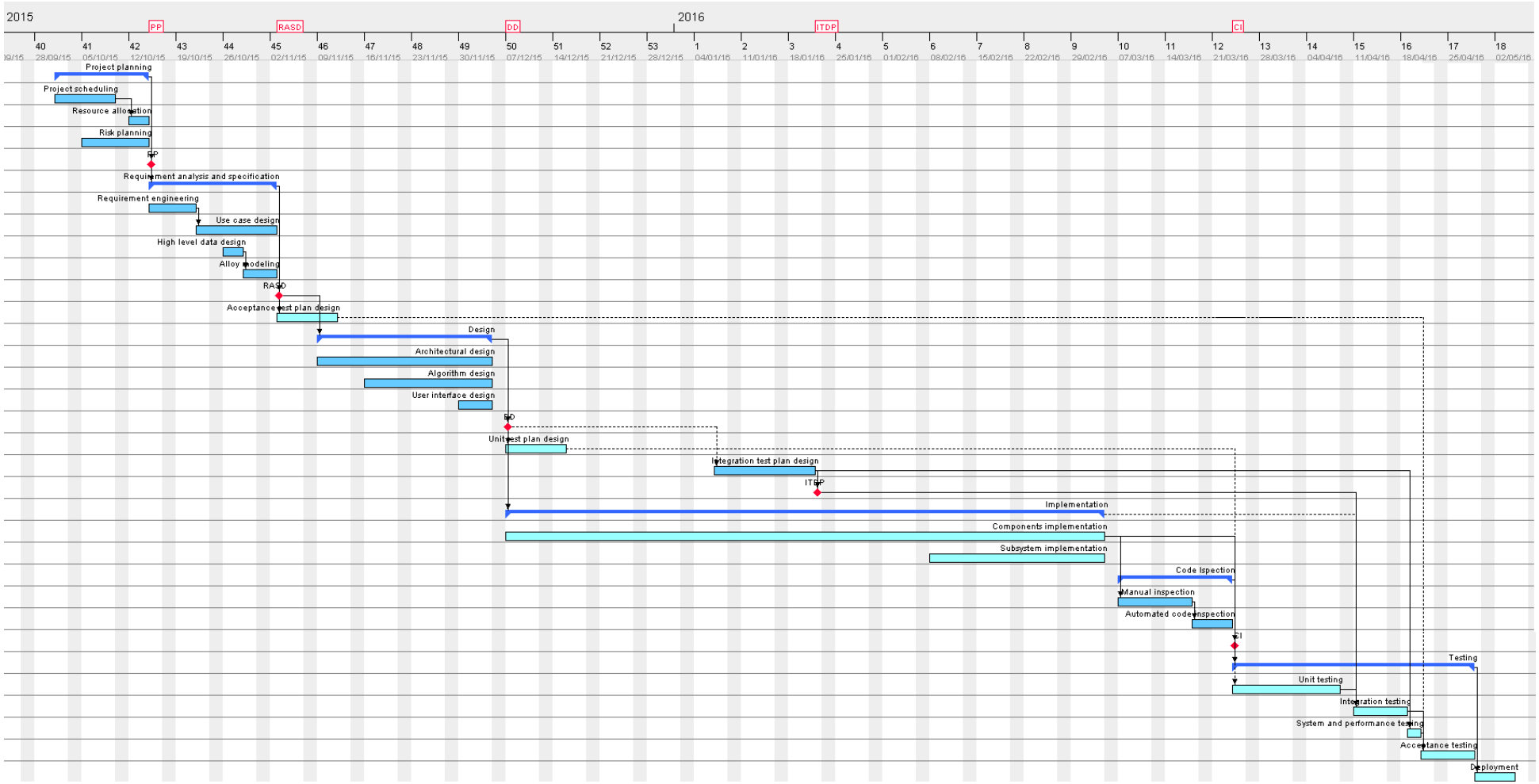
1. after RASD
2. after DD
3. after CI
4. after implementation
5. after testing
6. after deployment

4.3 Deliverables

Here are the expected deliverables of the project; the ones we really produced in the semester are highlighted in red in the previous table (actually we didn't perform code inspection on the myTaxiService but we included it as a deliverable anyway).

1. RASD (Requirement Analysis and Specification Document)
2. DD (Design Document)
3. CI (Code Inspection Report)
4. Code
5. UTPD (Unit Test Design Plan)
6. ITPD (Integration Test Design Plan)
7. User manual

4.4 Gantt chart



5 Resources

In this section we present the *resource allocation*. For each member of the team we will associate the tasks, as defined in the previous section, to be executed with reference to the available time of each team member. Since all team members contributed to the project playing different roles in the context of the software development cycle we will specify that role time by time.

5.1 Resource allocation

For the task we actually performed the hours specified are the real ones, for the others we estimated a reasonable allocation of time also according to our availability being university students. Moreover, according to our previous experiences, we qualified the implementation and the testing phases as the most time consuming.

The following table shows for each task the role played by the team member and the hours allocated; the attribute "Units", as commonly used in resource planning is the percentage ratio between the used hours and the allocated hours computed as the duration in days times 8 hours per day. As emerges from the table those values are by far lower than 50% because of our university activity.

Task id	*Name*	*Duration*	***Riccardo Mologni***			***Alberto Maria Metelli***		
			Role	*Hours*	*Units*	*Role*	*Hours*	*Units*
T1	**Project planning**	10	Project Manager	10	12,5%	Project Manager	10	12,5%
T11	Project scheduling	7	Project Manager	5	8,9%	Project Manager	3	5,4%
T12	Resource allocation	3	Project Manager	3	12,5%	Project Manager	3	12,5%
T13	Risk planning	8	Project Manager	2	3,1%	Project Manager	4	6,3%
T2	**Requirement analysis and specification**	13	Analyst	33	31,7%	Analyst	33	31,7%
T21	Requirement engineering	5	Requirements engineer	12	30,0%	Requirements engineer	12	30,0%
T22	Use case design	8	Analyst	15	23,4%	Analyst	11	17,2%
T23	High level data design	3	Analyst	3	12,5%	Analyst	3	12,5%
T24	Alloy modeling	3	Analyst	3	12,5%	Analyst	7	29,2%
T3	**Acceptance test plan design**	7	Analyst	8	14,3%	Analyst	8	14,3%
T4	**Design**	20	Software Architect	30	18,8%	Software Architect	35	21,9%
T41	Architectural design	20	Software Architect	15	9,4%	Software Architect	15	9,4%
T42	Algorithm design	15	Software Architect	7	5,8%	Software Architect	15	12,5%
T43	User interface design	5	Software Architect	8	20,0%	Software Architect	5	12,5%
T5	**Unit test plan design**	7	Analyst	13	23,2%	Analyst	13	23,2%
T6	**Integration test plan design**	11	Analyst	12	13,6%	Analyst	12	13,6%
T7	**Implementation**	65	Developer	260	50,0%	Developer	260	50,0%
T71	Components implementation	65	Developer	200	38,5%	Developer	200	38,5%
T72	Subsystem implementation	20	Developer	60	37,5%	Developer	60	37,5%
T8	**Code Ispection**	13	Ispector	15	14,4%	Ispector	18	17,3%
T81	Manual inspection	9	Ispector	13	18,1%	Ispector	13	18,1%
T82	Automated code inspection	4	Ispector	2	6,3%	Ispector	5	15,6%
T9	**Testing**	26	Tester	86	41,3%	Tester	86	41,3%
T91	Unit testing	12	Tester	45	46,9%	Tester	45	46,9%
T92	Integration testing	6	Tester	25	52,1%	Tester	25	52,1%
T93	System and performance testing	2	Tester	6	37,5%	Tester	6	37,5%
T94	Acceptance testing	6	Tester	10	20,8%	Tester	10	20,8%
T10	**Deployment**	4	Installer	15	46,9%	Installer	15	46,9%

The following is the same table in which just macro tasks are represented.

Task id	*Name*	*Duration*	***Riccardo Mologni***			***Alberto Maria Metelli***		
			Role	*Hours*	*Units*	*Role*	*Hours*	*Units*
T1	Project planning	10	Project Manager	10	12,5%	Project Manager	10	12,5%
T2	Requirement analysis and specification	13	Analyst	33	31,7%	Analyst	33	31,7%
T3	Acceptance test plan design	7	Analyst	8	14,3%	Analyst	8	14,3%
T4	Design	20	Software Architect	30	18,8%	Software Architect	35	21,9%
T5	Unit test plan design	7	Analyst	13	23,2%	Analyst	13	23,2%
T6	Integration test plan design	11	Analyst	12	13,6%	Analyst	12	13,6%
T7	Implementation	65	Developer	260	50,0%	Developer	260	50,0%
T8	Code Ispection	13	Ispector	15	14,4%	Ispector	18	17,3%
T9	Testing	26	Tester	86	41,3%	Tester	86	41,3%
T10	Deployment	4	Installer	15	46,9%	Installer	15	46,9%

5.2 Cumulative data

In this subsection we report some cumulative data derived from the previous table. We want to make a clear distinction between real deta referred to the task we actually performed and estimated data.

5.2.1 Estimated data

- *Total duration*: 176 days (1408 h)
- *Total working hours*
  - Riccardo Mologni: 482 h
  - Alberto Maria Metelli: 490 h
- *Working hours per day*
  - Riccardo Mologni: 2,72 h/day
  - Alberto Maria Metelli: 2,78 h/day
- *Units*
  - Riccardo Mologni: 34,2%
  - Alberto Maria Metelli: 34,8%

5.2.2 Real data

- *Total duration*: 67 days (536 h)
- *Total working hours*
  - Riccardo Mologni: 100 h
  - Alberto Maria Metelli: 108 h
- *Working hours per day*
  - Riccardo Mologni: 0,57 h/day
  - Alberto Maria Metelli: 0,61 h/day
- *Units*
  - Riccardo Mologni: 18,7%
  - Alberto Maria Metelli: 20,1%

5.3 Resource allocation chart

6 Risks and recovery actions

A *risk* is a potential condition that can lead to a the loss of some value, either resource (economic or not) or time. *Risk Mitigation, Monitoring, and Management* (*RMMM*) is a key step of any project plan, it is intended to help to pre-determine any possible major risks that may occur during development of this software. In this section we will identify the main risks concerning the development of myTaxiService and we will propose some strategy to tackle them.

6.1 Introduction

In this section we provide an analysis of the main risks harming the myTaxiService project. For each risk identified we will provide the the corresponding category *project risks*, *technical risks* and *business risks,* an estimation of the probability and the impact on the project [8].

6.1.1 Project risks

Project risks (also known as development risks) are risks that might threaten the project plan, if one of them becomes real it will harm the project schedule, making it slip and increase the overall cost. In myTaxiService we identified the following project risks.

Name	*Description*	*Probability*	*Impact*
Requirement problem - 1	Requirement engineers misunderstood the the requirements.	Possible	Catastrophic
Requirement problem - 2	Customers changes the requirement in the late phases of the software development.	Possible	Catastrophic
Requirement problem - 3	The customer is not available when needed	Likely	Marginal
Personnel problem - 1	Project manager is absent at critical times in the project.	Unlikely	Critical
Personnel problem - 2	Some team members are absent at the critical times in the project.	Unlikely	Marginal
Personnel problem - 3	Development team is not qulified to program a complex application with JEE.	Likely	Critical
Personnel problem - 4	Miscommunication in the project team.	Possible	Critical
Personnel problem - 5	Lack of documentation.	Rare	Critical

6.1.2 Business risks

Business risks can threaten the viability of the software to be produced; if one of them becomes real it can compromise the economic success of the project. In myTaxiService system we identified the following business risks.

[8] Probability and impact are expressed according to the standard qualitative scale. For probability: Certain, Likely, Possible, Unlikely, Rare. For impact/effect: Negligible, Marginal, Critical, Catastrophic.

Name	*Description*	*Probability*	*Impact*
Market risk	There is no demand for product, passengers prefer to use the traditional channels to call for a taxi.	Possible	Catastrophic
Budget risk	Due to organizational financial problems the budget is reduced.	Unlikely	Catastrophic

6.1.3 Technical risks

Technical risks can threaten the quality and the timeliness of the software to be developed; if a technical risk becomes real it can make the implementation more difficult or even impossible. In myTaxiService system we identified the following technical risks.

Name	*Description*	*Probability*	*Impact*
Design problem - 1	The architectural style and patter chosen in the architectural design phase are not suitable for the kind of application to be developed.	Unlikely	Catastrophic
Design problem - 2	The algorithm for taxi management proposed in the design phase are not correct.	Rare	Serious
Design problem - 3	Design does not reflect requirements	Unlikely	Catastrophic
Implementation problem - 1	Poor comments in the code.	Possible	Marginal
Implementation problems - 2	Code does not follow quality guidelines.	Unlikely	Critical
Implementation problem - 3	The code does not reflect the designed architecture.	Unlikely	Critical
Performance problems - 1	The DBMS cannot meet the performance requirements.	Unlikely	Catastrophic

6.2 Risk strategy

A *risk strategy* defines a set of rules to be applied in order to manage and tackle risk. Those strategies are typically divided into:

- *Reactive risk strategies*: there is no explicit risk management, nothing is done about risk until something goes wrong. This is the typical approach that is adopted by the majority of software teams and managers. This strategy is also known as *crisis management* since no preventive measure is designed.

- *Proactive risk strategies*: risk identification, analysis and ranking are performed in advance in order to develop a *contingency* plan to manage risk having high probability and high impact. This approach, even if more costly, is aimed to avoid risk in order to be able to deal with only unforeseen risks.

Since in the previous section we have identified the risks, we are actually following the proactive risk strategy. We think that identifying the risks, set up proper measures in order to tackle them in an appropriate way would make smaller the economic effort in the future stages.

Figure 4: Proactive risk strategy cycle

In the following tables we report for each risk identified in the previous section a possible *prevention plan* or a *correction plan*.

6.2.1 Project risks

Name	*Prevention*	*Correction*
Requirement problem - 1	• Adopt a formalized method for requirement engineering (like Jackson-Zave approach). • Perform validation of requirements right before design phase by meeting the stakeholders. • After requirement engineering phase provide customers with a small prototype.	Recycle on the requirement engineering phase fixing the RASD.
Requirement problem - 2	• It is explained to the customer, that after he has accepted a version of the RASD, it cannot be changed by the customer's wish only. • Trace the requirements in order to quantify the effort of changing.	Recycle on the requirement engineering phase fixing the RASD.
Requirement problem - 3	• Try to stay in touch with the customer in order to increase his interest on the product. • Meetings with the customer can be planned well in advance.	When the customer is not available, meetings may have to be rescheduled.
Personnel problem - 1	Nominate a vice project manager.	Vice project manager takes over the role of the project manager.
Personnel problem - 2	• Team members should warn the project manager timely before a planned period of absence. • Ensure that knowledge is shared between team members	An other team member takes care of the work.

Name	*Prevention*	*Correction*
Personnel problem - 3	Provide an introductory course to JEE	
Personnel problem - 4	• After a meeting, one group member creates an interview report. • Team members should not hesitate to ask and re-ask questions if things are unclear.	When it becomes clear that miscommunication is causing problems, the team members involved and the customer are gathered in a meeting to clear things up.
Personnel problem - 5	Impose that before starting a new phase of the software development the corresponding document is ready.	Ask the employee to write documentation.

6.2.2 Business risks

Name	*Prevention*	*Correction*
Market risk	Make sure there is the interest around the product.	Advertise the product.
Budget risk	Make sure at the beginning of the project that there is the availability of budget.	Communicate with the founders showing how the project makes a contribution to the goals of the business and argument about the fact that cuts to the budget would not be cost effective.

6.2.3 Technical risks

Name	*Prevention*	*Correction*
Design problem - 1	• Perform inspection of the DD. • Ask some advisor on his opinion about the feasibility and the correctness of certain design decisions.	• When errors in the design are noticed consulted some advisor to help correct the design errors as soon as possible. • Also all the work, that depends on the faulty design, should be halted until the error is corrected.
Design problem - 2	Use simulation techniques to test the correctness of the algorithm	Revise the algorithm in order to fix the flaws.
Design problem - 3	• Perform inspection of the DD. • Show at least some part of the DD to the customer in order to see his/her opinion.	Recycle on the design phase fixing the DD.
Implementation problem - 1	Provide the developer with a clear specification of what to comment.	Ask the developer to add the missing comments.
Implementation problems - 2	• Provide the developer with a document in which code conventions are stated. • Perform code inspection.	Ask the developer to fix the issues.
Implementation problem - 3	• Perform unit testing after each module is ready • Continuesly check the work of the developers	Ask the developer to fix the involved component.
Performance problems - 1	Estimate the load and acquire a suitable DBMS.	Investigate the possibility of acquiring a highier performance DBMS.

A Appendix

Used tools

1. LyX visual editor for LaTeX (`http://www.lyx.org/`) to write this document.
2. GanttProject for the Gantt diagram and the resource diagram `http://www.ganttproject.biz/`.

Hours of works

Time spent by each group member:

- Alberto Maria Metelli: 10 h
- Riccardo Mologni: 10 h

Revision history

Version	*Date*	*Revision description*	*Revision notes*
0.1	30-1-2016	Initial draft	-
1.0	2-2-2016	Final draft	-
2.0	22-2-2016	Final release	Fixed introduction and some terminology.