Politecnico di Milano

A.A. 2015-2016

Software Engineering 2: “myTaxiService”

Project **P**lan **D**ocument

Davide Azzalini (855185), Fabio Azzalini (855182)

1 Introduction

In this first chapter we give an introduction and a overall description of the whole document. In particular we present the purpose and the scope of the Project Plan Document. We also provide some other useful information such as a list of the definitions, abbreviations and acronym used.

1.1 Purpose

This document represent the Project Plan Document (PPD). The purpose of this document is to define two dimensions of crucial importance for a project: we will use Function Points to estimate the size and COCOMO to estimate the effort and the cost. Moreover, a detailed schedule for the development and an assessment about the major risks associated to the project is provided.

1.2 Scope

The government of a large city aims at optimizing it's taxi service. At this stake myTaxiService will be created, a brand-new application available in two variations: web application and mobile application.

In particular, the attention will be focused on making the access to the service easier from the passengers and on guaranteeing a fairer distribution of the work load among all taxi drivers.

The application will allow passengers to request a ride and taxi drivers to decide weather or not to give the ride. Once a taxi driver has been allocated the passenger will be informed about the waiting time and the code of the incoming taxi.

Furthermore, the application will ensure a more efficient management of the service since each taxi is provided with a GPS tracker and the city is divided in zones, as a result a reduction of waiting times it is expected.

In addition to the standard taxi service (in which the costumer contacts the taxi driver in the exact moment he/she needs the service), will also be possible for the costumers to make reservations for future rides.

Important is also for the system to be developed in such a way that a future addition of some new feature can be made easily and without too much effort.

Moreover, the possibility for payments to be electronic, hence recorded, should help decrease the amount of tax evasion as all transactions can be tracked and accounted for.

1.3 List of Definitions and Abbreviations

- PPD: Project Plan Document
- FP: Function Points
- UFP: Unadjusted Function Points
- SLOC: Source Lines Of Code
- LOC: Lines Of Code
- COCOMO: COnstructive COst MOdel
- ILF: Internal Logic Files
- ELF: External Logic Files
- EI: External Input
- EO: External Output
- EQ: External Inquiries
- PM: Person-Month
- EAF: Effort Adjustment Factor

1.4 List of Reference Documents

- Specification Document: myTaxiService Project AA 2015-2016.pdf
- RASD.pdf

- DD.pdf
- ITPD.pdf
- Assignment 5 - project plan.pdf
- Ian Sommerville - Software Engineering (Ninth Edition)
- COCOMO II, Model Defintion Manual
  http://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_modelman2000.0.pdf
- Lecture slides

1.5 Overall Description

The following is the structure of this project plan document:

- In **chapter 2** are outlined and performed the two algorithmic procedures: FP and COCOMO.
- In **chapter 3** is presented a development schedule and a possible task allocation.
- In **chapter 4** are pointed out those risks that to us are more dangerous and likely to happen.

2 Function Points and COCOMO

2.1 Function Points

The function point cost estimation approach is based on the amount of functionality in a software project. Function points are useful estimators since they are based on information that is available early in the project life-cycle.

Function points measure a software project by quantifying the information processing functionality associated with major external data or control input, output, or file types.

The following table identifies the five type of user Function Points.

Function Point	Description
External Input (EI)	Count each unique user data or user control input type that enters the external boundary of the software system being measured.
External Output (EO)	Count each unique user data or control output type that leaves the external boundary of the software system being measured.
Internal Logical File (ILF)	Count each major logical group of user data or control information in the software system as a logical internal file type. Include each logical file (e.g., each logical group of data) that is generated, used, or maintained by the software system.
External Interface Files (EIF)	Files passed or shared between software systems should be counted as external interface file types within each system.
External Inquiry (EQ)	Count each unique input-output combination, where input causes and generates an immediate output, as an external inquiry type.

The next step is to classify each of these function types by a complexity level.

For Internal Logical Files and External Interface Files

	Data Elements		
Record Elements	**1 - 19**	**20 - 50**	**51+**
1	Low	Low	Avg.
2 - 5	Low	Avg.	High
6+	Avg.	High	High

For External Output and External Inquiry

	Data Elements		
File Types	**1 - 5**	**6 - 19**	**20+**
0 or 1	Low	Low	Avg.
2 - 3	Low	Avg.	High
4+	Avg.	High	High

For External Input

	Data Elements		
File Types	**1 - 4**	**5 - 15**	**16+**
0 or 1	Low	Low	Avg.
2 - 3	Low	Avg.	High
3+	Avg.	High	High

Than each complexity level is mapped to a precise weight using the table below.

	Complexity-Weight		
Function Type	**Low**	**Average**	**High**
Internal Logical Files	7	10	15
External Interfaces Files	5	7	10
External Inputs	3	4	6
External Outputs	4	5	7
External Inquiries	3	4	6

The Unadjusted Function Points (UFP) is finally computed by adding together all the weighted function counts.

The conversion from UFP to Lines of Code is obtained by multiplying UFP by the conversion ratio of the chosen language. Conversion Ration can be found in the table below.

Language	Default SLOC / UFP	Language	Default SLOC / UFP
Access	38	Jovial	107
Ada 83	71	Lisp	64
Ada 95	49	Machine Code	640
AI Shell	49	Modula 2	80
APL	32	Pascal	91
Assembly - Basic	320	PERL	27
Assembly - Macro	213	PowerBuilder	16
Basic - ANSI	64	Prolog	64
Basic - Compiled	91	Query – Default	13
Basic - Visual	32	Report Generator	80
C	128	Second Generation Language	107
C++	55	Simulation – Default	46
Cobol (ANSI 85)	91	Spreadsheet	6
Database – Default	40	Third Generation Language	80
Fifth Generation Language	4	Unix Shell Scripts	107
First Generation Language	320	USR_1	1
Forth	64	USR_2	1
Fortran 77	107	USR_3	1
Fortran 95	71	USR_4	1
Fourth Generation Language	20	USR_5	1
High Level Language	64	Visual Basic 5.0	29
HTML 3.0	15	Visual C++	34
Java	53		

What follows is the implementation of this technique to our case.

2.1.1 Internal Logic Files

The application needs some ILFs in order to store the information about users, requests, reservations, payments and queues.

For users we memorize simple information, thus the complexity can be assumed to be low.

Requests, reservations, payments and queues store more complex data, GPS information and credit card information. We can consider these of average complexity.

In the table below it is shown the total amount:

ILF	Complexity	FP
User	Low	7
Request	Average	10
Reservation	Average	10
Payment	Average	10
Queue	Average	10
Total		**47**

2.1.2 External Logic Files

The system, in order to correctly work, needs to acquire data from three external sources: GPS, Google Maps and banking information.

A GPS position basically consist of two numbers, we can assume its complexity to be low.

When it comes to the other two sources data are more structured, therefore their complexity can be considered to be average.

ELF	Complexity	FP
GPS	Low	5
Google Maps	Average	7
Banking	Average	7
Total		**19**

2.1.3 External Inputs

Users can interact as follow:

- Login/Logout: simple operation.

- Registration/Modify Profile: these interactions involve quite big amount of data and some integrity checks have to be run in order to correctly complete them. The complexity is average.
- Set Availability: simple operation.
- Current Location: each taxi needs to constantly inform the system about its current location. Given the repetitiveness and thickness of this streaming of data we assume its complexity to be high.
- Make a Request/Reservation: average complexity operation.
- Accept a Request/Reservation: simple operation.
- Pay a Request/Reservation: since banking information are stored in the DataBase, the passenger doesn't need input them for each payment. The complexity is low.
- Inform the system that a costumer has paid (in case he/she decide to pay by cash), if a passenger showed up and when the destination is reached. These three operations are very simple, the driver, in fact, will probably just has to press a button.

EI	**Complexity**	**FP**
Login/Logout	Low	2x3
Registration/Modify Profile	Average	2x4
Set Availability	Low	3
Current Location	High	6
Make Request/Reservation	Average	2x4
Accept Request/Reservation	Low	2x3
Pay Request/Reservation	Low	2x3
Inform the System	Low	3x3
Total		**52**

2.1.4 External Outputs

The External Output provided by the system are essentially of three types: notifications, real time rendering and receipts. Notifications, for example, include

the taxi code and the waiting time when a driver accept, those to inform drivers of a new request, etc.

EO	Complexity	FP
System Notifications	Average	5
Real Time Rendering	High	7
Receipts	Average	5
Total		**17**

2.1.5 External Inquiries

Inquiries are those user-system interactions in which the system's reply is immediate.

We assume the all our inquiries to be of average complexity since they either involve many elements or need information whose collecting is not very simple.

EQ	Complexity	FP
Visualize Profile Info	Average	4
Future Reservation	Average	4
History	Average	4
Home Page	Average	4
Total		**16**

2.1.6 Final Results

The following table summarizes the results obtained up to now.

Function Type	Value
ILF	47
ELF	19
EI	52
EO	17
EQ	16
Total	**151**

Now, the last thing left to do is multiply our result by the conversion ratio of the language that will be chosen for the implementation.

Below, this computation is listed w.r.t. the languages in which the implementation is more likely to be:

- JAVA: 53 x 151 = 8003 lines of code.
- C++: 55 x 151 = 8305 lines of code.
- C#: 54 x 151 = 8154 lines of code.
- J2E: 46 x 151 = 6946 lines of code.

We remind that these results provide just an idea and the actual number of LOC could be slightly different.

2.2 COCOMO II

The effort estimation using COCOMO is achieved through a complex, non linear model that takes under consideration of course the product itself, but also the people and the process.

The most important element of the model is the effort-equation:

$$€ffort = 2.94 * EAF * (KSLOC)^E$$

The result is measured in Person-Month (PM). EAF is the Effort Adjustment Factor derived from the Cost Drivers. E is an exponent derived from the Scale Drivers. We will see what is the meaning of these drivers in the next two sections.

2.2.1 Scale Drivers

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

- Precedentedness (**PREC**): Reflects the previous experience of the organization with this type of project. In our case, we can assume this factor to be Low since this is the first time we work on a project of this kind. That said, we are not

completely new to the coding world thus there is no reason to consider it Very Low.

- Development Flexibility (**FLEX**): Reflects the degree of flexibility in the development process. The Specification Document: "myTaxiService Project AA 2015-2016.pdf" suffers from the typical drawbacks of natural language descriptions: it is informal, incomplete, uses different terms for the same concepts, etc. The value of this factor is hence Very High.
- Architecture/risk resolution (**RESL**): Reflects the extent of risk analysis carried out. A big emphasis has been put to the selection of best architectural style during the design, therefore this factor is High.
- Team cohesion (**TEAM**): Reflects how well the development team know each other and work together. Since we are twins we can consider this factor Extra High, with no need for further argumentation.
- Process maturity (**PMAT**): Reflects the process maturity of the organization. The procedure for determining PMAT is to decide the percentage of compliance for each of the 18 KPAs-Key Process Areas (we list only those which can be applied to our case):
  - Requirements Management: Almost Always.
  - Software Project Planning: Almost Always.
  - Software Project Tracking and Oversight: Frequently.
  - Software Quality Assurance (SQA): Frequently.
  - Software Configuration Management (SCM): Frequently.
  - Integrated Software Management: Frequently.
  - Software Product Engineering: Almost Always.
  - Intergroup Coordination: Almost Always.
  - Peer Reviews: Frequently.
  - Software Quality Management: Frequently.
  - Defect Prevention: Frequently.
  - Technology Change Management: About Half.
  - Process Change Management: About Half.

Once we have converted each KPA in percentage (100% for Almost Always, 75% for Frequently, 50% for About Half, 25% for Occasionally, 1% for Rarely if Ever) we can compute the following:

$$\text{EPML} = 5 \times \left(\sum_{i=1}^{n} \frac{\text{KPA\%}_i}{100} \right) \times \frac{1}{n}$$

And then look up the correspondent value in the table below. In our case EMPL is 3.94 thus the PMAT factor is very high.

PMAT Rating	Maturity Level	EPML
Very Low	CMM Level 1 (lower half)	0
Low	CMM Level 1 (upper half)	1
Nominal	CMM Level 2	2
High	CMM Level 3	3
Very High	CMM Level 4	4
Extra High	CMM Level 5	5

The result are resumed in the following table:

Scale Driver	Factor	Value
PREC	Low	4.96
FLEX	Very High	1.01
RESL	High	2.83
TEAM	Extra High	0.00
PMAT	Very High	1.56
Total		**10.36**

2.2.2 Cost Drivers

The seventeen cost drivers are used in the COCOMO II model to adjust the nominal effort, Person-Months, to reflect the software product under development. Each multiplicative cost driver is defined below:

- Required Software Reliability (**RELY**): This is the measure of the extent to which the software must perform its intended function over a period of time. In our case we assume it to be Low.
- Data Base Size (**DATA**): This cost driver attempts to capture the effect large test data requirements have on product development. The rating is determined by calculating D/P, the ratio of bytes in the testing database to SLOC in the

program. The reason the size of the database is important to consider is because of the effort required to generate the test data that will be used to exercise the program. In our case we assume it to be nominal.

- Product Complexity (**CPLX**): Complexity is divided into five areas: control operations, computational operations, device-dependent operations, data management operations, and user interface management operations. Given the complex nature of our system we assume it to be High
- Developed for Reusability (**RUSE**): This cost driver accounts for the additional effort needed to construct components intended for reuse on current or future projects. During the deign phase we give a lot of importance to reusability, in fact we have chosen the component based software engineering approach. The core of components is their reusability. We assume it to be Very High.
- Documentation Match to Life-Cycle Needs (**DOCU**): Several software cost models have a cost driver for the level of required documentation. In COCOMO II, the rating scale for the DOCU cost driver is evaluated in terms of the suitability of the project's documentation to its life-cycle needs. We have spent loads of hours writing the documentation of this project (RASD, DD, ITP, PPD) we assume it to be High.
- Execution Time Constraint (**TIME**): 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. In our case this factor has Nominal importance.
- Main Storage Constraint (**STOR**): This rating represents the degree of main storage constraint imposed on a software system or subsystem. We assume it to be Nominal.
- Platform Volatility (**PVOL**): "Platform" is used here to mean the complex of hardware and software (OS, DBMS, etc.) the software product calls on to perform its tasks. The only platform that could change in a reasonable magnitude is the Google Maps service. We assume that our system as well as banking services are not concerned volatility problems. We assume it to be Low.
- Analyst Capability (**ACAP**): 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. Since we have put a lot of effort in analyzing the requirements we can assume it to be High.

- Programmer Capability (**PCAP**): This factor reflects the capability of the programmers as a team rather than as individuals. We work well together, we can assume it to be High.
- Personnel Continuity (**PCON**): The rating scale for PCON is in terms of the project's annual personnel turnover: from 3%, very high continuity, to 48%, very low continuity. There is no turnover. We assume it to be Very Low.
- Applications Experience (**APEX**): The rating for this cost driver (formerly labeled AEXP) 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. Even though we have some coding background this is the first time we work on a project of this kind. We can assume it to be Low.
- Platform Experience (**PLEX**): The Post-Architecture model broadens the productivity influence of platform experience, PLEX (formerly labeled PEXP), by recognizing the importance of understanding the use of more powerful platforms, including more graphic user interface, database, networking, and distributed middleware capabilities. We have a one year experience with platforms such as Databases, user interfaces and serve-side development. We can assume it to be Nominal.
- Language and Tool Experience (**LTEX**):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. We can assume it to be Nominal.
- Use of Software Tools (**TOOL**): The use of software tools can play a big difference. The tool rating ranges from simple edit and code, very low, to integrated life-cycle management tools, very high. We used standard tools. We can assume it to be Nominal.
- Multisite Development (**SITE**): This parameter reflects how well we handled the distribution development over distance and multiple platforms. We are brother and we live together. We can assume it to be Extra High.
- Required Development Schedule (**SCED**): This rating measures the schedule constraint imposed on the project team developing the software. We can assume it to be Nominal.

The result is obtained multiplying all the cost drivers together.

Cost Driver	Factor	Value
RELY	Low	0.92
DATA	Nominal	1.00
CPLX	High	1.17
RUSE	Very High	1.15
DOCU	High	1.11
TIME	Nominal	1.00
STOR	Nominal	1.00
PVOL	Low	0.87
ACAP	High	0.85
PCAP	High	0.88
PCON	Very Low	1.29
APEX	Low	1.10
PLEX	Nominal	1.00
LTEX	Nominal	1.00
TOOL	Nominal	1.00
SITE	Extra High	0.80
SCED	Nominal	1.00
Total		**1.01**

2.2.3 Final Results

Now we have all the data and can go back to the computation of the effort equation.

*€ffort = 2.94 * EAF * (KSLOC)^E*

Where:

- EAF is the product of the cost drivers (1.01);

- E is derived multiplying the sum of the scale drivers by 0.01 and then adding 0.91;
- KSLOC is the lines of code (in thousands) obtained with the function points (8003);

We can now solve the equation:

*€ffort = 2.94 * 1.01 * (8.003)^1.0136 = 24.446 PM*

Since the number of persons needed is *N = €ffort / duration* and we are a team of two persons we can easily find the duration:

Duration = €ffort / 2 = 12.223 months

3 Task Identification

In the table below we show the schedule we have followed during the whole documentation phase (from task 1 to 4). The time allocated to each task is deliberately longer than necessary. The reason for this it is that we are students and we can't focus 100% of our day on the project. In an actual working place it would have probably taken less.

N°	Task	From	To	Total
1	Requirements Analysis and Specification Dcoument	21/10/2015	6/11/2015	2 weeks
2	Design Document	12/11/2015	4/12/2015	3 weeks
3	Integration Test Plan	13/01/2016	21/01/2016	1 week
4	Project Plan	27/01/2016	2/02/2016	1 week
5	Software Implementation + Unit Testing	to be defined	to be defined	-
6	Integration Testing	to be defined	to be defined	-
7	System Testing	to be defined	to be defined	-
8	User Acceptance (Alpha, Beta Testing)	to be defined	to be defined	-

4 Resources Allocation

Given the fact that we are brother and we live together there hasn't been a clear-cut distinction on which team member focused on which task. The same holds for the individual work loads.

N°	Task	Team Members	Individual Working Time	Total Working Time
1	Requirements Analysis and Specification Dcoument	Davide, Fabio	60 hrs	120 hrs
2	Design Document	Davide, Fabio	50 hrs	100 hrs
3	Integration Test Plan	Davide, Fabio	7 hrs	14 hrs
4	Project Plan	Davide, Fabio	15 hrs	30 hrs
5	Software Implementation + Unit Testing	Davide, Fabio	-	-
6	Integration Testing	Davide, Fabio	-	-
7	System Testing	Davide, Fabio	-	-
8	User Acceptance (Alpha, Beta Testing)	Davide, Fabio	-	-

5 Risks

There 3 main categories of risks that may affect a software project are:

- **Progect risks** are those which threaten the project plan and the project schedule.
- **Technical risks** are those which threaten the quality and the timeliness of the software to be produced and in particular its implementation.
- **Business risks** are those which threaten the viability of the software to be built and the whole project.

Let's start from the latter one. For what concern *market risk* in our opinion the risk does not subsist, our product will cover a service that already exist, the only difference will be how costumers access the service. There are no *strategic risk* nor *sales risk* since the product it is not meant to be sold the open market but to a single costumer. In case, in the future, the product will be sold to other costumers they will certainly be other cities with similar requirements. Since the costumer of our software is the government we think that there is more flexibility towards eventual *budget* variations, thus, if an eventual variation is not absurd the project would continue anyway. The personnel commitment is not under concern.

Given the not so high experience and knowledge of the developing team the risk of delays, and thus a raise in the budget, even if not so high is still present. This said, the development team is rather optimist that the product will be delivered in time.

During both planning and development the attention to risks prevention is of crucial importance and keep a proactive mindset is the key. It is better to be safe than to be sorry.

Even with a deep attention, the probability of some failure to occur is not negligible. All possible problems related to the actual usage will be discovered and fixed in the testing phase. An example is to check that the system does not go down even with a workload higher than the nominal one.

Another thing that could be very dangerous is the high dependence of the system on external services and platforms. Crucial are GPS data and in particular the internet connection, a prerogative that if unavailable makes the all system useless and unusable.

6 Appendix

6.1 Hours of work

- Azzalini Davide: 15 hours
- Azzalini Fabio: 15 hours