Software Engineering 2: myTaxiService

Project Plan

Chitti Eleonora, De Nicolao Pietro, Delbono Alex
Politecnico di Milano

February 2, 2016

Contents

Chapter 1

Function Points estimation

1.1 Internal Logic Files

An Internal Logic File (ILF) is defined as a *homogeneous set of data used and managed by the application.*

The ILFs of our system match closely the database tables and are the following:

- User
- Passenger
- TaxiDriver
- TaxiLog
- Ride

1.2 External Interface Files

An External Interface File (EIF) is defined as a *homogeneous set of data used by the application but generated and maintained by other applications.*

The only EIF of the system consists in the interface with Google Maps services, described in the DD [4, p. 33]. We identify this EIF as *Maps* in the further tables.

1.3 External Input

An External Input (EI) is defined as an *elementary operation to elaborate data coming from the external environment.*

The EIs of the system are the following:

- User registration
- User profile management
- User login
- Standard taxi call
- Taxi availability
- Taxi reservation

1.4 External Output

An External Output (EO) is defined as an *elementary operation that generates data for the external environment.* It usually includes the elaboration of data from logic files.

The EOs of the system are the following:

- Notifications to users
- Confirmation mails
- Notifications to taxi drivers

1.5 External Inquiry

An External Inquiry (EQ) is defined as an *elementary operation that involves input and output, without significant elaboration of data from logic files.*

The EQs for the system are the following:

- Taxi driver ride request
- Ride sharing

1.6 Complexity levels of Function Points

Table 1.1 shows how to evaluate the weights of Function Points, based on the number of data elements (from [6]).

According to these criteria, Table 1.2 shows the weights for the functions of the system.

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

(a) Weight estimation for ILFs and EIFs.

	Data Elements		
Record Elements	1-5	6-19	20+
0-1	Low	Low	Avg.
2-3	Low	Avg.	High
4+	Avg.	High	High

(b) Weight estimation for EOs and EQs.

	Data Elements		
Record Elements	1-4	5-15	16+
1	Low	Low	Avg.
2-3	Low	Avg.	High
3+	Avg.	High	High

(c) Weight estimation for EIs.

Table 1.1: Estimation of weights for different types of Function Points.

1.7 Weights of Function Points

The scores associated to all types of function points, by type and weight, are shown in Table 1.3. The table is taken from [6].

The weights computed for our system are displayed in Table 1.4.

1.8 UFP to SLOC conversion

According to [6], the multiplicator to convert the Unadjusted Function Points to Source Lines of Code for Java is **53**.

1.9 Count of FPs, by type and weight

The total count of Function Points of our system is shown in Table 1.5.

Functions	Weights
User	High
Passenger	Avg
TaxiDriver	High
TaxiLog	High
Ride	High

(a) Computed weights for ILFs

Functions	Weights
Maps	High

(b) Computed weights for EIFs

Functions	Weights
User registration	Avg
User profile management	Low
User login	Low
Standard taxi call	High
Taxi availability	Low
Taxi reservation	High

(c) Computed weights for EIs

Functions	Weights
Notification to users	Low
Confirmation mails	Low
Notifications to taxi drivers	Low

(d) Computed weights for EOs

Functions	Weights
Taxi driver ride request	Low
Ride sharing	High

(e) Computed weights for EQs

Table 1.2: The evaluation of the weights of the functions according to Table 1.1.

Function Type	**Complexity-Weight**		
	Simple	Medium	Complex
External Input	3	4	6
External Output	4	5	7
External Inquiry	3	4	6
Internal Logic File	7	10	15
External Interface File	5	7	10

Table 1.3: Scores of function points by type and weight.

Function	**Type**	**Complexity**	**Weight**
User	ILF	High	15
Passenger	ILF	Avg	10
TaxiDriver	ILF	High	15
TaxiLog	ILF	High	15
Ride	ILF	High	15
Maps	EIF	High	6
User registration	EI	Avg	4
User profile management	EI	Low	3
User login	EI	Low	3
Standard taxi call	EI	High	6
Taxi availability	EI	Low	3
Taxi reservation	EI	High	6
Notification to users	EO	Low	4
Confirmation mails	EO	Low	4
Notifications to taxi drivers	EO	Low	4
Taxi driver ride request	EQ	Low	4
Ride sharing	EQ	High	6

Table 1.4: Weights computed for all the functions.

Function Type	**Complexity-Weight**			**Total Points**
	Simple	Medium	Complex	
External Input	$3 \cdot 3$	$1 \cdot 4$	$3 \cdot 6$	31
External Output	$3 \cdot 4$	$0 \cdot 5$	$0 \cdot 7$	12
External Inquiry	$1 \cdot 3$	$0 \cdot 4$	$1 \cdot 6$	9
Internal Logic File	$0 \cdot 7$	$1 \cdot 10$	$4 \cdot 15$	70
External Interface File	$0 \cdot 5$	$0 \cdot 7$	$1 \cdot 10$	10
Total Points	24	14	94	**132**
SLOC	$\times 53$			**6992**

Table 1.5: Count of Function Points of our system.

Chapter 2

COCOMO II estimation

2.1 Scale Drivers

Scale Drivers are the parameters that reflect the non-linearity of the effort with relation to the number of SLOC. They show up at the exponent of the Effort Equation (Equation 2.1).

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. The precedentedness is low because we have some experience of software design but most of the notions used in this project are new to us.

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. We set it to Nominal because we have to follow a prescribed process, but we had a certain degree of flexibility in the definition of the requirements and in the design process.

Risk resolution: reflects the extent of risk analysis carried out. Very low means little analysis, Extra high means a complete a thorough risk analysis. We set it to high, because a rather detailed risk analysis is carried out in chapter 5.

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

problems. We set it to very high, since the cohesion among the three of us is optimal.

Process maturity: reflects the process maturity of the organization. We set it at High, which corresponds to CMM Level 3[1]: "It is characteristic of processes at this level that there are sets of defined and documented standard processes established and subject to some degree of improvement over time. These standard processes are in place (i.e., they are the AS-IS processes) and used to establish consistency of process performance across the organization."

The estimated scale drivers for our project, together with the formula to compute the exponent E, are shown in Table 2.1.

2.2 Cost Drivers

Cost Drivers are the parameters of the Effort Equation that reflect some characteristics of the developing process and act as multiplicators on the effort needed to build the project. They appear as factors in the Effort Equation (Equation 2.1). Cost Drivers are described in detail in the COCOMO Manual [6, p. 25].

The estimated cost drivers for our project, together with the formula to compute the EAF (Effort Adjustment Factor), are shown in Table 2.2.

2.3 Effort

$$\text{Effort} = A \times EAF \times KSLOC^{E} \tag{2.1}$$

Where:

- $EAF = \prod_i C_i$: product of all cost drivers
- $E = 0.91 + 0.01 \times \prod_i SF_i$: depends on the scale drivers.
- $KSLOC$: Kilo-Source Lines of Code, as estimated in chapter 1.
- $A = 2.94$

The total effort results in **21.8 person-months**.

[1] `https://en.wikipedia.org/wiki/Capability_Maturity_Model#Levels`

Code	Name	Factor	Value
PREC	Precedentedness	Low	4.96
FLEX	Development flexibility	Nominal	3.04
RESL	Risk resolution	High	2.83
TEAM	Team cohesion	Very High	1.10
PMAT	Process maturity	High	3.12
Total	$E = 0.91 + 0.01 \times \sum_i SF_i$		1.0605

Table 2.1: Scale Drivers for our project.

Code	Name	Factor	Value
RELY	Required Software Reliability	Nominal	1.00
DATA	Data base size	Nominal	1.00
CPLX	Product Complexity	Nominal	1.00
RUSE	Required Reusability	High	1.07
DOCU	Documentation match to life-cycle needs	Nominal	1.00
TIME	Execution Time Constraint	Nominal	1.00
STOR	Main Storage Constraint	Nominal	1.00
PVOL	Platform Volatility	Low	0.87
ACAP	Analyst Capability	High	0.85
PCAP	Programmer Capability	Nominal	1.00
APEX	Application Experience	Very low	1.22
PLEX	Platform Experience	Very low	1.19
LTEX	Language and Tool Experience	Low	1.09
PCON	Personnel Continuity	Very high	0.81
TOOL	Usage of Software Tools	Nominal	1.00
SITE	Multisite Development	High	0.93
SCED	Required Development Schedule	High	1.00
Total	$EAF = \prod_i C_i$		0.943

Table 2.2: Cost Drivers for our project.

2.4 Duration

$$\text{Duration} = 3.67 \times (PM)^{0.28+0.2\times(E-0.91)} \tag{2.2}$$

Where:

- PM is the effort as calculated in Equation 2.1.
- E is the exponent depending on the scale drivers (the same one of the effort equation).

The duration calculated from Equation 2.2 (from [6]) results in a total of **9.5 months**. This would result in 2.3 developers needed for this project.

Since our group consists of 3 people, a reasonable development time would be:

$$\frac{21.8 \text{ pm}}{3 \text{ people}} = 7.3 \text{ months}$$

In order to be cautious with the task scheduling, we will approximate it to 8 months.

Chapter 3

Tasks and schedule

The main tasks involving this project are:

1. Deliver the *Requirement Analysis and Specification Document*, containing the goals, the domain assumptions, and the functional and nonfunctional requirements of the software system.
2. Deliver the *Design Document*, containing the architecture and the design of the software system.
3. Deliver the *Integration Testing Plan Document*, containing the strategy used to perform integration testing on the system.
4. Deliver the *Project Plan*, which is this document.
5. Prepare a brief presentation ($\sim$ 15 min) about the delivered documents, with slides.
6. Implement the software system and write unit tests.
7. Perform integration testing on the system.

Please note that, as new requirements can emerge, new choices are made and the development goes on, the process can be iterated multiple times. In particular, unit and integration testing will be continuously performed throughout the development process.

However, some tasks need to be concluded before some other can begin: the **dependency graph** for the tasks is shown in Figure 3.1.

The first five tasks for the project are already defined by the document about the project rules [1], together with the deadlines for the delivery of the RASD, the Design Document and the ITPD. The date for the presentation is also fixed. So, those activities are already scheduled.

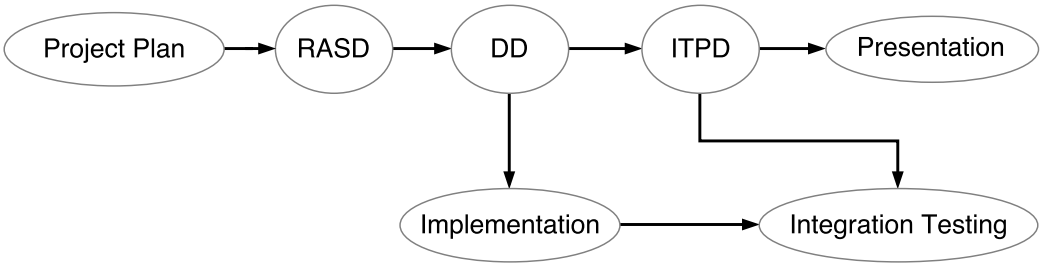

Figure 3.1: Directed Acyclic Graph showing the dependencies among tasks.

There are no fixed deadlines, instead, for the development of the software. From the COCOMO estimation performed in chapter 2, we expect the entire project to last 8 months, so it will be presumably finished by June 2016.

The schedule for our project is outlined in Table 3.1. The Gantt chart for the project is shown in Figure 3.2.

Activity	Start Date	Deadline
RASD	2015-10-15	2015-11-06
DD	2015-11-11	2015-12-04
ITPD	2016-01-07	2016-01-21
Project Plan	2016-01-21	2016-02-02
Presentation	2016-02-02	2016-02-17
Implementation	2016-02-18	2016-06-01
Integration Testing	2016-06-01	2016-06-15

Table 3.1: Schedule for project tasks.

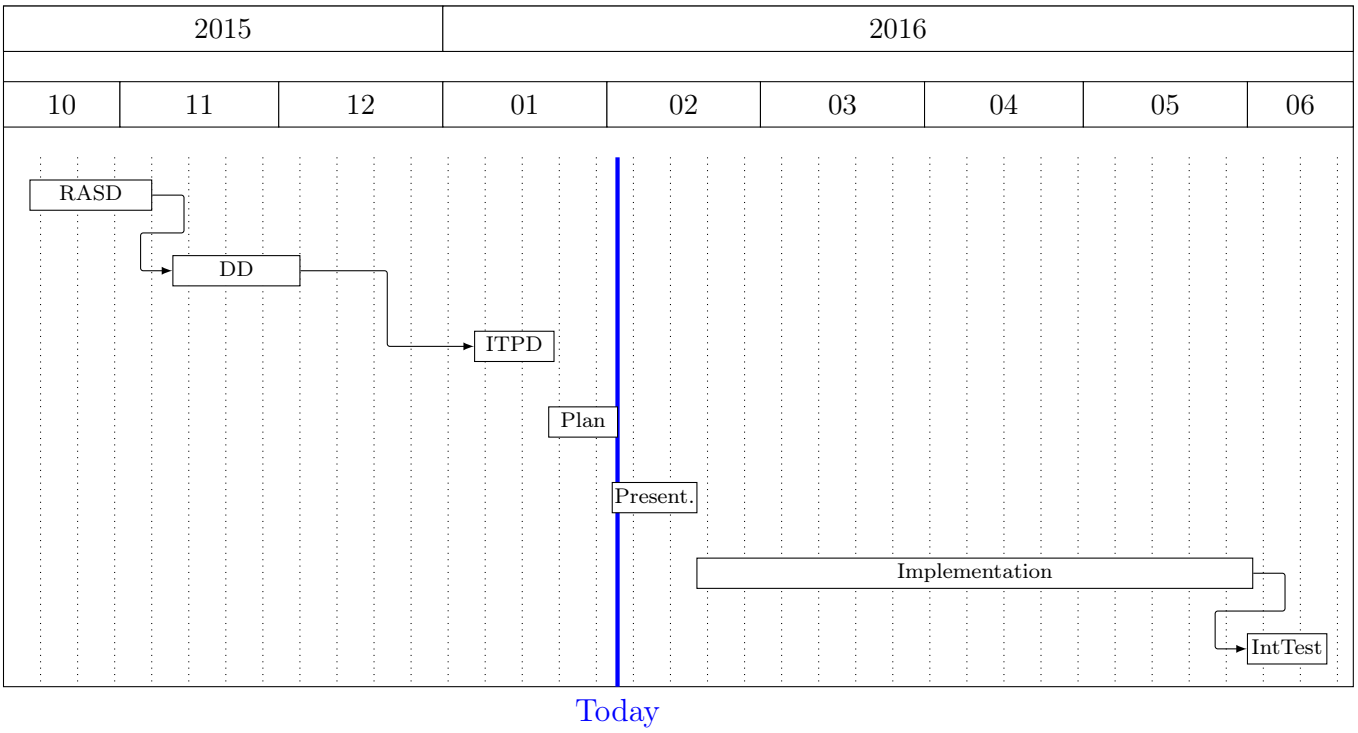

Figure 3.2: Gantt chart of the project.

Chapter 4

Resource allocation to tasks

This chapter has the purpose to show how the available resources are allocated to the project. It does not aim to be a very specific schedule, in fact the tasks are grouped with a high level of abstraction and resources are assigned to these high-level tasks. This is because micromanaging the work of people on a large project is of little use.

The resulting schedule gives to all the team members a general overview on the whole project: *every task involves the contribution of each member*. This enlarges slightly the time needed for the completion of the project, but also induces a greater awareness of all the members of the team. It also enables a more accurate control on the work of each other in order to limit the number of bugs and misunderstandings.

The documents to be written related to the project are divided into arguments which mainly reflect the division in chapters; each argument will be assigned to two people and the third will have the duty to revision it.

The code will be divided into tiers which reflect the system architecture. Every tier will be assigned to two people; the third will write and execute unit tests.

The division of the work is elaborated according to the schedule (Table 3.1, Figure 3.2).

Tables 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7 show the division of work among the team members.

Resource	**2015-10-15 to 2015-11-06**			
	1_{st} week	2_{nd} week	3_{rd} week	4_{th} week
Pietro	Intro	Overall desc	System features	Revision Spec req
Alex	Overall desc	Spec Req	System features	Revision Intro
Eleonora	Spec req	Intro	Revision Overall desc	Revision System features

Table 4.1: Resource allocation for RASD.

Resource	**2015-11-11 to 2015-12-04**			
	1_{st} week	2_{nd} week	3_{rd} week	4_{th} week
Pietro	Arch design	Algorithm design	UI design	Rev intro - req trac
Alex	Algorithms	Intro - Req trac	Revision Arch design	Rev UI design
Eleonora	Intro - Req trac	Arch design	UI design	Rev Algorithms

Table 4.2: Resource allocation for DD.

Resource	**2016-01-07 to 2016-01-21**	
	1_{st} week	2_{nd} week
Pietro	Integration strategy	Individual steps
Alex	Introduction - minor chapters	Individual steps
Eleonora	Individual steps	Rev individual steps

Table 4.3: Resource allocation for ITPD.

Resource	**2016-01-21 to 2016-02-02**	
	1_{st} week	2_{nd} week
Pietro	Cocomo II	Risk
Alex	Functions points	Resource allocation
Eleonora	Cocomo II	Task and schedule

Table 4.4: Resource allocation for Planning.

Resource	**2016-02-02 to 2016-02-17**	
	1_{st} week	2_{nd} week
Pietro		Slide
Alex		Slide
Eleonora		Slide

Table 4.5: Resource allocation for Presentation.

Resource	**2015-11-11 to 2015-12-04**			
	1_{st} month	2_{nd} month	3_{rd} month	4_{th} month
Pietro	Web tier	Database	Mobile client	Test Business
Alex	Business tier	Mobile client	Test Database	Test Web
Eleonora	Database	Web tier	Business tier	Test Mobile

Table 4.6: Resource allocation for Implementation.

Resource	**2016-01-21 to 2016-02-09**	
	1_{st} week	2_{nd} week
Pietro	Business - Data	Mobile - Business
Alex	Business - Data	Client - Web
Eleonora	Business - Web	Mobile - Business

Table 4.7: Resource allocation for Integration Testing.

Chapter 5

Risks associated with the project

The project is associated with many project, technical and economical risks.

5.1 Project risks

The project risks are listed below. An evaluation of those risks is provided in Table 5.1.

Delays over the expected deadlines: the project could require more time than expected. If this happens, we may release a first, incomplete but working version (e.g. without the web interface or the plugins for taxi reservation and taxi sharing) and build the less essential features later.

Lack of communication among team members: the team often works remotely and this can lead to misunderstandings in fundamental decisions, to conflicts over the division of the work among team members and to conflicts in code versioning. Those difficulties can be overcome by explicitly defining (e.g. in this document) the responsibilities of each team member, and by writing clear and complete specification and design documents.

Lack of experience in programming with the specific frameworks: the team has no actual experience in programming using Java EE. This will certainly slow down the development.

Requirements change: the requirements may change during the development in unexpected way. This risk can't be prevented, but can be mitigated by writing reusable and extensible software.

5.2 Technical risks

The technical risks are listed below. An evaluation of those risks is provided in Table 5.2.

Integration testing failure: after the implementation of some components, they may not pass the integration testing phase: this can require to rewrite large pieces of software. This risk can be mitigated by defining precisely the interfaces between components and subsystems, and by doing integration tests early using stubs and drivers.

Downtime: the system could go down for excessive load, software bugs, hardware failures or power outages. This risk can be mitigated by building redundant systems and placing them in geographically separate data centers and by performing testing at all levels.

Scalability issues: the system could not scale with a large number of users, requiring a major design rework. A possible plan is to use a cloud infrastructure from a third-party provider to host our system.

Spaghetti code: with the growth of the project, the code may become overloaded, badly structured and unreadable. This risk can be mitigated by writing a good Design Document before starting to code, and by performing code inspection periodically.

Deployment difficulties: if cities already have a taxi management system, it could be difficult to deploy our system by keeping and migrating the old data. This problem can be solved by hiring professional system integrators, but of course this would increase the costs.

Data loss: data can be lost because of hardware failures, misconfigured software or deliberate attacks. This problem can be prevented by enforcing a reliable backup plan. Backups should be kept in a separate place from the system, and offline.

Data leaks: misconfiguration, software bugs and deliberate attacks can expose the users' personal data. This risk must be prevented by adopting industry security standards, by encrypting communications and by doing regular penetration testing.

5.3 Economical risks

The economical risks are listed below. An evaluation of those risks is provided in Table 5.3.

Bankruptcy: the income from the sales of the software may not be enough to sustain the development, maintenance and deployment of the software system. A good feasibility study helps to avoid this situation.

Regulation change: local and State regulators can change the taxi regulation at any time, and this could have an unpredictable impact on the usage and the market penetration of our software. This risk can be partially avoided by a good feasibility study. This issue could be mitigated by doing lobbying on the political parties of the affected countries (wherever this is legal).

Competitors: other companies can build equivalent or better products at a lower price and put myTaxiService out of the market. The competitiveness of our product must be continuously enhanced by introducing innovative features.

Risk	**Probability**	**Effects**
Delays	High	Moderate
Lack of communication	Low	Moderate
Lack of experience	Certain	Moderate
Requirements change	Very low	Moderate

Table 5.1: Evaluation of project risks.

Risk	**Probability**	**Effects**
Integration testing failure	Low	High
Downtime	Moderate	High
Scalability issues	Low	Moderate
Spaghetti code	Low	Moderate
Deployment difficulties	Moderate	Moderate
Data loss	Low	Catastrophic
Data leaks	Moderate	Catastrophic

Table 5.2: Evaluation of technical risks.

Risk	**Probability**	**Effects**
Bankruptcy	Moderate	Catastrophic
Regulation change	Low	Severe
Competitors	Moderate	Severe

Table 5.3: Evaluation of economical risks.

Appendix A

Appendix

A.1 Software and tools used

- LaTeX for typesetting this document.
- GitHub[1] for version control and distributed work.

A.2 Hours of work

The statistics about commits and code contribution are available on GitHub [2]. Please keep in mind that many commits are actually group work (when this is the case, it is stated in the commit message).

- Eleonora Chitti: 2 hours
- Alex Delbono: 5 hours
- Pietro De Nicolao: 5 hours

[1] `https://github.com`
[2] `https://github.com/pietrodn/se2-mytaxiservice`

Bibliography

[1] Software Engineering 2 Project, AA 2015/2016, *Project goal, schedule and rules*

[2] Software Engineering 2 Project, AA 2015/2016, *Assignment 5: Project Plan*

[3] Chitti Eleonora, De Nicolao Pietro, Delbono Alex, *Software Engineering 2: myTaxiService – Requirements Analysis and Specification Document*

[4] Chitti Eleonora, De Nicolao Pietro, Delbono Alex, *Software Engineering 2: myTaxiService - Design Document*

[5] Chitti Eleonora, De Nicolao Pietro, Delbono Alex, *Software Engineering 2: myTaxiService - Integration Test Plan Document*

[6] COCOMO II - Model Definition Manual, Version 2.1, 1995 – 2000, Center for Software Engineering, USC. `http://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_modelman2000.0.pdf`