Politecnico di Milano

A.A. 2015-2016

Software Engineering 2: “myTaxiService”

Project Plan Document

version 1.0

Adrian Mihai Berbieru #854698

Attilio D’Onofrio #789614

02/02/2016

INDEX

1 Project Size and Effort

1.1 Function Points Analysis

The complexity values below are taken from the example showed during the lecture held by Professor Tamburri.

Component	Complexity		
	Low	**Medium**	**High**
External Input	3	4	6
External Outputs	4	5	7
External Inquiries	3	5	6
External Interface Files	5	7	10
Internal Logical Files	7	10	15

1.1.1 External Inputs

Operation	Estimated complexity	Function points
Customer/driver login	Low	6
Customer signup	Medium	4
Password recovery	Low	3
Instant booking	High	6
Long term booking	High	6
Driver status switch	Medium	4
Driver's call management	High	6

Total FP: 35

1.1.2 External Outputs

There are no external outputs provided by our system.

1.1.3 External Inquiries

Operation	Estimated complexity	Function points
List long term bookings	Medium	5

Total FP: 5

1.1.4 External Interface Files

Operation	Estimated complexity	Function points
Position by map service	Medium	7
Step by step navigation	High	10

Total FP: 17

1.1.5 Internal Logical Files

Operation	Estimated complexity	Function points
Customer	Low	7
Taxi driver	Low	7
Mapping address/city zone	Low	7
Queue	High	15
Long term bookings	Average	10

Total FP: 46

1.2 Unadjusted Function Value and Lines of Code Estimation

$$UFP = 35+5+17+46= 103$$

$$AVC_{Java\ 2EE} = 46$$

$$\mathbf{LOC} = AVC_{Java\ 2EE} * UFP = 4738$$

Therefore the number of lines of source code (excluding tests and comments) will be almost five thousands.

1.3 COCOMO Analysis

1.3.1 Scale drivers

Using the scale factors provided by the "COCOMO II, Model Definition Manual" at ***http://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_modelman2000.0.pdf*** we can define for each scale driver a value that best describes our project development.

Table 10. Scale Factor Values, SF_j, for COCOMO II Models

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

Reflects the previous experience of the organisation with this type of project. In our case this is the first time we've been assigned the development of a project of this caliber so we can set the value to "***Very Low***".

Development Flexibility

Reflects the degree of flexibility in the development process. In our case we can make the assumption that in a real world scenario this value would be "***High***" because as stated in the project description we have to model a new system that offers new services to people while at the same time we have to maintain conformity to the city's regulatory policies regarding taxi services.

Architecture / Risk Resolution

Reflects the extent of risk analysis carried out. Our project still being a prototype that hasn't been yet tested in a real world scenario we can assume that the value should be "***High***".

Team Cohesion

Reflects how well the development team know each other and work together. We've encountered some problems of synchronization and communication, mainly caused by different time schedules and thus different working hours. Nevertheless we've successfully overcome the difficulties and managed to work together, so the value should be "***Very High***".

Process Maturity

Reflects the process maturity of the organisation. The organization that has enquired us for the creation of the myTaxiService is the City Council that does not have a well defined processes standard at an Organizational level, if any at all, therefore we suppose that it should be "***Nominal***".

We can summarize all the results of the scale driver in the following table:

Scale drivers	***Factor***	***Value***
Precedentedness	Very Low	6.20
Development Flexibility	High	2.03
Architecture / Risk Resolution	High	2.83
Team Cohesion	Very High	1.10
Process Maturity	Nominal	4.68
Total		16.84

1.3.2 Cost Drivers

Required Software Reliability

In our case the measurement must be set to "***Nominal***" as the software or hardware failure of the system translates into big loss in money to the City Council, taxi drivers and problems created to customers.

DataBase Size

This parameter is set to "***Nominal***" as the application still hasn't been implemented.

Product Complexity

This parameter is set to "***Very High***".

Required Reusability

Designing the system in a modular way we can assume that this parameter can be set to "***High***".

Documentation match to life-cycle needs

Most of our system has been thoroughly described in the RASD and DD documents so therefore we can set this parameter to "***Nominal***".

Execution Time Constraint

In our case the parameter can be set to "***Nominal***" as most of the time the system will have to keep up to date queues of available taxis and dispatch cars to customers' locations.

Main Storage Constraint

In our case the parameter is set to "***Very Low***" as it is expected that the application won't consume large quantities of memory.

Platform Volatility

The parameter can be set to "***Low***" because any change regarding the system must be done after testing it on the field, therefore we can suppose a minimum waiting period of 12 months.

Analyst Capability

It can be set to "***High***" as the RASD and DD have been thoroughly created in order to deliver a stable and optimal product.

Programmer Capability

The parameter should be set to "***High***" this is because of the "bumps" we've encountered during the development.

Application Experience

Our project experience is evaluated according to our previous experience in web projects and also according to our abilities in programming in Java and most importantly in the Java EE framework. Since this is our first experience in this field the parameter is set to "***Low***".

Platform Experience

The parameter is set to "***Nominal***".

Language and Tool Experience

The parameter is set to "***Low***".

Personnel continuity

The parameter can be set to "***Very High***" considering the fact that all members of the group have worked on the project from the beginning until the end.

Usage of Software Tools

It can be set to "***Nominal***".

Multisite development

The parameter can be set to "***High***" because the application has been developed for a city that can be run on different platforms.

Required development schedule

Our efforts were well distributed over the available development time therefore we can set the parameter to "***Nominal***".

Cost Driver	***Factor***	***Value***
Required Software Reliability	Nominal	1
DataBase Size	Nominal	1
Product Complexity	Very High	1.34
Required Reusability	High	1.07
Documentation match to life-cycle needs	Nominal	1
Execution Time Constraint	Nominal	1
Main Storage Constraint	Very Low	n/a
Platform Volatility	Low	0.87
Analyst Capability	High	0.85
Programmer Capability	High	0.88
Application Experience	Low	1.10
Platform Experience	Nominal	1
Language and Tool Experience	Low	1.09
Personnel continuity	Very High	0.81
Usage of Software Tools	Nominal	1
Multisite development	High	0.93
Required development schedule	Nominal	1
Total		0.84

1.3.2 Effort and Duration

The COCOMO II model estimates the effort of development of the project as follows

$$Effort = 2.94 \cdot EAF \cdot KSLOC^{E} \ [Person/Months]$$

Where:

- EAF: Effort Adjustment Factor derived from Cost Drivers;
- E: Exponent derived from Scale Drivers and is calculated as follows $B + 0.01 \cdot \sum_{i} SF_i$;
- KSLOC: Thousands Source Lines of Code.

So we have

$$E = B + 0.01 \cdot 16.84 = 0.91 + 0.01 \cdot 16.84 = 1.0784$$

$$Effort = 2.94 \cdot 0.84 \cdot (5)^{1.0784} = 14 \ [P/M]$$

COCOMO II schedule equation predicts the number of months required to complete a software project. The duration of a project is based on the effort predicted by the effort equation:

$$Duration = 3.67 \cdot Effort^{F} \ [Months]$$

Where:

- Effort: Is the effort from the COCOMO II effort equation;
- F: Is the schedule equation exponent derived from the five Scale Drivers and is calculated as follows F := 0.28 + 0.2 * (E – B).

So we have

$$F = 0.28 + 0.2 \cdot (1.0784 - 0.91) = 0.31368$$

$$Duration = 3.67 \cdot (14)^{0.31368} = 8.4 \ [M]$$

Finally we can calculate the number of required people to work on this project following the COCOMO II estimation is

$$N = \frac{Effort}{Duration} = \frac{14}{8.4} = 1.7 \approx 2\ [Person]$$

Therefore to complete this project we need two people, this result is consistent with the composition of the group (as we're a group of two).

Let's recap:

Hours spent on RASD

- Adrian Mihai Berbieru: ~8 hours
- Attilio D'Onofrio: ~ 8 hours

Hours spent on DD

- Adrian Mihai Berbieru: ~20 hours
- Attilio D'Onofrio: ~20 hours

Hours spent on CD

- Adrian Mihai Berbieru: ~4 hours
- Attilio D'Onofrio: ~4 hours

Hours spent on ITPD

- Adrian Mihai Berbieru: ~8 hours
- Attilio D'Onofrio: ~8 hours

Hours spent on PPD

- Adrian Mihai Berbieru: ~10 hours
- Attilio D'Onofrio: ~10 hours

Hours spent on Developing a working prototype: ~240 hours

Therefore for the development of this project we've dedicated 340 hours.

$$\frac{340\ hours}{(40 \cdot 4)\ hours} = 2.2\ Person/Months$$

we suppose that a man can work 40 hours in a week so 40*4 is a number of hours that man works in a month.

	Estimated Value	Real Value
Effort	14 Person/Months	2.2 Person/Months
Duration	8.4 Months	3 Months
Number of people	2	2

2 Project Scheduling

The development of myTaxiService project has started in October 2015 and has lasted approximately 3 months. During this timespan 4 documents were created which reflect how our group handled the creation of the myTaxiService system and also the main parts that the development process can be divided into.

Document name	***Description***
Requirements Analysis and Specifications	Requirements analysis is the process of determining user expectations for a new or modified product. These features, called requirements, must be quantifiable, relevant and detailed. In software engineering, such requirements are often called functional specifications. Requirements analysis is an important aspect of project management. Requirements analysis involves frequent communication with system users to determine specific feature expectations, resolution of conflict or ambiguity in requirements as demanded by the various users or groups of users, avoidance of feature creep and documentation of all aspects of the project development process from start to finish. Energy should be directed towards ensuring that the final system or product conforms to client needs rather than attempting to mold user expectations to fit the requirements.
Design Document	A design description is a written description of a software product, that a software designer writes in order to give a software development team overall guidance to the architecture of the software project. An design description usually accompanies an architecture diagram with pointers to detailed feature specifications of smaller pieces of the design. Practically, the description is required to coordinate a large team under a single vision, needs to be a stable reference, and outline all parts of the software and how they will work.
Code Inspection	A code review can be done as a special kind of inspection in which the team examines a sample of code and fixes any defects in it. In a code review, a defect is a block of code which does not properly implement its requirements, which does not function as the programmer intended, or which is not incorrect but could be improved (for example, it could be made more readable or its performance could be improved).
Integration Test Plan	Integration testing is a software development process which program units are combined and tested as groups in multiple ways.

	In this context, a unit is defined as the smallest testable part of an application. Integration testing can expose problems with the interfaces among program components before trouble occurs in real-world program execution. Integration testing is a component of Extreme Programming (XP), a pragmatic method of software development that takes a meticulous approach to building a product by means of continual testing and revision.

2.1 Tasks

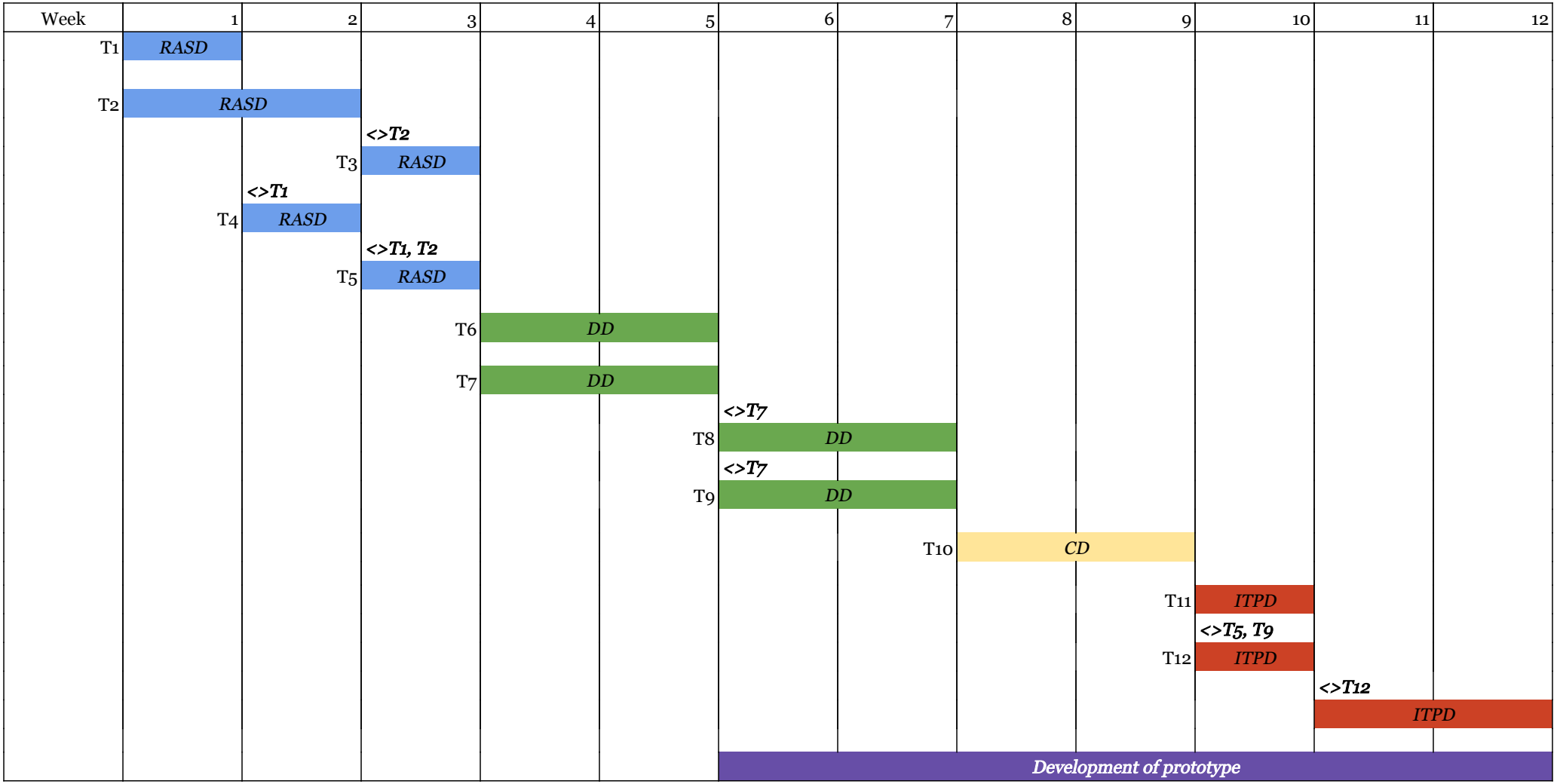
In this section we will illustrate how the 4 different sections concerning the development of the new system can be further smaller and non overlapping tasks that will be assigned to each member of the group accordingly.

Section	***Task number***	***Task description***
Requirements Analysis and Specification	T1	General presentation of the document including description of the scope of the new system, the actors involved, main functionalities the system has to offer and term definitions.
	T2	An initial draft of the system, including the assumptions regarding the context of the new system and some future expansions.
	T3	Design of the user interfaces, API interfaces and a general description of the hardware and software interfaces. And specifying for each goal the necessary functional requirements.
	T4	Elaboration of different scenarios users may interact with the system. Definition of the use cases and in depth analysis of the satisfiability of the goals through sequence diagrams, state diagrams. Description of the main non functional requirements.
	T5	Elaboration and testing of the abstract model created for MyTaxiService through a specification language, namely Alloy, in order to control the initials model integrity

		and consistency.
Design	T6	Document description, term definitions and scope of the design description.
	T7	Description of the chosen architectural design for the system and its various components. Presentation of the components of myTaxiService and their in depth analysis, also mentioning the subcomponents of the main parts.
	T8	Description of how the system is formed and how the different components interact with each other by describing their interfaces. Also giving reasons as to why the architectural designs have been chosen.
	T9	Explanation of the system works through the description of the algorithms used and through a detailed representation of the functions of the components.
Code Inspection	T10	Thorough inspection and testing of the methods assigned to our group.
Integration Test Plan	T11	General description of the document.
	T12	Presentation of the integration strategy used for testing the system as well as the tools used to accomplish this task.
	T13	Detailed explanations for each integration test presented at beginning of the document of how they are conducted and the expected result as well as the different stubs needed to accomplish this task.

3 Resource Allocation

Task number	*Dependencies*	*Team member assigned to*
T1		D'Onofrio
T2		Berbieru
T3	T2	D'Onofrio
T4	T1	Berbieru
T5	T1, T2	D'Onofrio
T6		D'Onofrio
T7		Berbieru
T8	T7	Berbieru
T9	T7	D'Onofrio
T10		Berbieru D'Onofrio
T11		D'Onofrio
T12	T5, T9	Berbieru
T13	T12	D'Onofrio


Week
1
2
3
4
5
6
7
8
9
10
11
12
T1
RASD
T2
RASD
<>T2
T3
RASD
<>T1
T4
RASD
<>T1, T2
T5
RASD
T6
DD
T7
DD
<>T7
T8
DD
<>T7
T9
DD
T10
CD
T11
ITPD
<>T5, T9
T12
ITPD
<>T12
ITPD
Development of prototype

4 Potential Risks

After a thorough analysis of the tasks the project is composed of, we present a list of potential problems we could stumble upon during the development and we prepare some countermeasures in order to avoid any impasse and stick to the time schedule.
This paragraph is divided into two subsections: the first one explains the problems we could face throughout the whole development, while the second one will consider the risks that may arise during the tasks presented in 2.1 section of this document.
The careful reader will notice that in the second paragraph there will not be issues related to the starting part of the project. The reason is that the first part of the project mainly involves skills taught during the Software Engineering I course, so that we had experience to some extent.

4.1 General Risks

One of the main obstacles we could face during the work (and in a special way in the second half) is the inexperience of the team members. In fact, this is the first time members took part in such kind of project. The online documentation will be helpful in providing a wider insights on what has to be done and to clarify our doubts. The professor's support via BeeP platform will be crucial to go on when we will be stuck in "deadlock".

Another obstacle we could face is consequence of splitting the workload and working in a parallel way for most of the time. In other words, sometimes two micro-tasks will have to be assigned to different team members and there will be a tight correlation between them. This could lead to some inconsistencies and to additional time spent to fix it.
In order to avoid such kind of problems we can split the tasks in a careful way and work together for the crucial parts of the documents.

There are also other risks related to the use of external services.
The first is Google APIs. Although it is very unlikely that Google may decide to discontinue the support, the event could worsen customer's experience while sending his position and driver's experience when picking the customer up. Given the unlikeliness of the event as well as the marginality of the global impact, we do not forecast special countermeasures (except recommending the drivers to keep a "old-fashioned" navigation system).
The second is the cloud hosting the system runs on. In case of extended downtime, the taxi service of the city would be completely unusable. We may subscribe a special contract with hosting provider in which there is a considerable penalty if the minimum uptime is not guaranteed.
The last dependency to take care of is the compliance with law and taxi policies. Since we have no clue about what could change in the future, we would put a significant effort in isolating the

modules of the software in which laws are encoded and apply software engineering best-practices to increase the maintainability of that portion of the code.

4.2 Task-related Risks

Design - algorithm description (T9)

The writing of the most critical algorithms requires a crystal-clear view of the entire application. Therefore during this task there may be some discrepancy regarding the work already done, so we take into account some extra time spent in checking what we have already written in order to be consistent.
Relevance: medium
Recovery actions: read the Design Document, focusing especially on the paragraph describing the architecture

Code Inspection (T10)

While the code inspection itself is not an hard task, getting a clear picture of what the class does requires a prior understanding about its structure and behaviour.
Relevance: low
Recovery actions: read the whole class and use the JavaDoc whenever required