Politecnico di Milano

Scuola di Ingegneria dell'Informazione

Corso di Laurea Magistrale di Computer Science and Engineering

Software Engineering 2 Project

myTaxiService

Part V : PP (Project Plan)

Principal Adviser:
Prof. Raffaela Mirandola

Authors:

Turconi Andrea ID n. 853589

Raimondi Lorenzo ID n. 859001

Accademic year 2015-2016

Contents

1 Introduction

1.1 Purpose

Project Plan Document describes a structured plan for myTaxiService project, that is task identification and allocation, time and effort estimation and risk analysis. Doing a sharp planning of project activities helps in their development and ensure a more coordinated work between team members, giving at the same time a useful tool for reporting project status to customers and stakeholders.

1.2 Document structure

This document is structured in three main sections: in the first one is performed the cost analysis using Function Points and Cocomo models; the second section presents the identification of project tasks and their allocation to team members over the project estimated time; in the third and last section is carried out the risk analysis, describing for each of them chances and, when possible, recovering solutions.

1.3 References

- Specification document: assignment 1 and 2.pdf
- Requirement Analysis and Specifications Document v1
- Design Document v1
- Integration Test Plan Document v1
- Project Plan assignment.pdf

2 Project estimations

2.1 Function Points

Function Points approach is a method to estimate the dimension of software by performing an abstraction on its functionalities and associating them with a weighed score, on the base of functionalities type. The five different functionalities groups are:

- **Internal Logic File**, homogeneous sets of data used and handled by the application, that is system database;
- **External Interface File**, homogeneous sets of data used by the application but generated from other applications, as external services;
- **External Input**, elementary operations requested from the external environment and requiring data elaboration;
- **External Output**, elementary operations that generates data for the external environment;
- **External Inquiry**, elementary operations involving input and output from/to external environment.

The scores, associated to a specific functionality basing on the above groups and on its estimated complexity, are reported in the following table.

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

Starting from the table and from RASD requirements and feature specification, we analyzed our system and computed step by step the Function Points number

Internal Logic File

Internal data stored in the database include information about users, requests, taxi queues and users' notification. Among these, users and notifications data

are the simpler, being composed by few simple fields, so we adopt the simple weight. Queues and requests have more detailed and composed fields, so we assign the medium weight for queues and the complex one for requests, the system core entity. Having said that the outcome is 2 x 7 + 1 x 10 + 1 x 15 = **39 FPs**.

External Interface File

For what regards this functionality group, the only external service that provide data to our system are Google Maps APIs. The data acquired by this source requires also some manipulation, so we consider it as medium complexity. So the simple total is of **10 FPs**.

External Input

Our system provides to the user several different services, and this reflect in the following input interactions:

- Login: as a simple operation, we assign 3 FPs.
- Logout: as a simple operation, we assign 3 FPs.
- Users registration: as simple operation, we assign 3 FPs.
- Users information editing: as simple operation, we assign 3 FPs.
- Request forwarding: requiring complex computation for its fulfilling, we assign 6 FPs.
- Request editing: requiring the verification on some editing constraints, we assign 4 FPs.
- Request deleting: as a simple operation, we assign 3 FPs.
- Taxi Driver reply: requiring some computation for operations on queues, we assign 4 FPs.
- Taxi Driver availability editing: as a simple operation, we assign 3 FPs.
- Taxi Driver unforeseen reporting: as an urgent and constraining input, its managing may be complex, and we assign 6 FPs.

Summing up all the external input functionalities, we get the result of **38 FPs**.

External Output

As external outputs we considered notifications that the system sends to users in differents use cases, and requires computing data from different and composed entities. In particular we considered:

- Notifications that update users on their waiting time;
- Notifications that signal to users unforeseen to their requests;

- Notifications that forward to drivers new ride requests.

Considering all of them of a medium complexity, the computed number is 3 x 5 = **15 FPs**.

External Inquiry
External inquiries that require input and output from and to the external environment are the ones related to viewing user requests and user profile. Assuming them as very simple inquiries we consider them of simple complexity, counting a total of 2 x 3 = **6 FPs**.

Summing all the Function Points computed for all the five groups we get the total of **108 UFPs**. This value appears meaningful comparing it to other comparable project estimations, and is mantained unadjusted with the assumption to proceed with the *COCOMO* method to estimate project effort.

2.2 COCOMO II

After Function Points calculation we proceeded with *COCOMO II* method using the official Model Definition Manual [1].
Following the manual we started the computation of *COCOMO* factors with the estimation of the *SLOC* value, representing Source Lines Of Code. As suggested in the manual, this value is the product between the previous *UFP* value and a constant value depending on the used programming language. The suggested value for J2EE is 46, so we found the estimated lines of code as

$$UFP * 46 = 108 * 46 = 4968 \ SLOC$$

After this estimation, the second step to take is to evaluate *Scale Factors*, a set of value that measures the presence or the absence of economies of scale according to different criteria, assigning them a level between Very Low and Extra High, and a corresponding numeric value. In particular we considered:

- **Precedenteness**, that represent the similarity of the project with previously developed ones, as **Low**, because for us this is the first project of this size and the first using J2EE.

$$PREC = 4.96$$

- **Development Flexibility**, that measures how and how much the project requirement can be bended, as **Low**, because we aim to meet as much as possible the initial assignment.

$$FLEX = 4.05$$

[1] COCOMO II Model Definition Manual version 2.1
http://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_modelman2000.0.pdf

- **Architecture/Risk Resolution**, meaning how risk management has been carried out, as **Nominal**, since risk management is performed in this document.

$$RESL = 4.24$$

- **Team Cohesion**, measuring team members cooperation attitudes, as **High**, having already accounted some project together.

$$TEAM = 2.19$$

- **Process Maturity**, that represents the overall maturity leves, has been estimated by means of the KPA Levels survey. Compiling it we obtained an 50% value that reflect in a process maturity value of **Nominal**.

$$PMAT = 4.68$$

The next step in *COCOMO* estimation is to assign the *Cost Drivers*, values used to detail more precisely nominal effort and person-month factor. Also these values have rating levels between Very Low and Extra High, with a corresponding effort multiplier. So we considered:

- **Required Software Reliability**, that measures how the system have to bear fault in its execution, as **Very Low**, considering acceptable to tolerate slight inconvenience, having myTaxiService an alternative system provided by the traditional taxi phone numbers.

$$RELY = 0.82$$

- **Database Size**, which reflect on costs of data testing, as **Nominal**, having a standard database without particular constraints.

$$DATA = 1.00$$

- **Product Complexity**, that measures operations, data, user interface and device managing complexity, as **Nominal**, having estimate the result with the given table.

$$CPLX = 1.00$$

- **Reusability**, which reflect the effort in building a reusable and flexible component architecture, as **High**, on the base of the scalable and service-oriented architecture designed for the system during design phase.

$$RUSE = 1.07$$

- **Documentation**, that is the production of the required level of documentation respect to the software life-cycle needs, as **Nominal**, having produced the requested documentation for the project.

$$DOCU = 1.00$$

- **Execution Time**, that represents execution time constraints imposed to the software system, as **High**, considering that the application would not require a persistent computation during all the day time.

$$TIME = 1.11$$

- **Storage**, which measure the degree of storage constraint, as **Low**, assuming that the available storage wouldn't represent high filling rates.

$$STOR = N/A$$

- **Platform Volability**, which means how often used platforms updates and cause intervention to conform the system, as **Low**, considering the used platform like J2EE or Glassfish Server as guarantors of long term support.

$$PVOL = 0.87$$

- **Analyst Capability**, representing the experience and efficiency of analysts, as **Low**, being this our first experience with a similar project and requirement identification.

$$ACAP = 1.19$$

- **Programmer Capability**, meaning the experience of programmers, as **Low**, being this the first time we use J2EE and develop a 3-tiered application.

$$PCAP = 1.15$$

- **Personnel Continuity**, which means how frequently the team members can work on the project, as **Nominal**, according to our actual availability.

$$PCON = 1.00$$

- **Application Experience**, that represents the experience of the team members regarding software projecting, as **Nominal**, having both about 1 year of experience with client-server applications.

$$APEX = 1.00$$

- **Platform Experience**, that measures the experience on similar system platforms, as **Low**, having about 6 months of experience on this specific developing platform.

$$PLEX = 1.09$$

- **Language and Tool Experience**, measuring team members experience on programming language and tools used in the project, as **Nominal**, having 1 year of experience on Java and Object-Oriented programming.

$$LTEX = 1.00$$

- **Use of Software Tools**, which reflect developing tools' grade of use, as **Nominal**, using a normal IDE to develop the system.

$$TOOL = 1.00$$

- **Multisite Development**, which represent distance between team members, as **Very High**, working side by side and living close one another.

$$SITE = 0.86$$

- **Required Develompent Schedule**, measuring the schedule constraint imposed on system development, as **Nominal**, assuming a standard developing, consistent with the planned one.

$$SCED = 1.00$$

Once assigned all the values to *Scale Factors* and *Cost Drivers* we proceeded with the calculation of effort and project duration. The effort is computed using the following formula, where A is a *COCOMO II* constant derived by statistical analysis, *Size* represents the number of line code expressed in thousands, and E depends on the values assigned to the seventeen *Cost Drivers*.

$$Effort = A * Size^E * \prod Cost\ Drivers$$

where
$A = 2.94$
$Size = \frac{SLOC}{1000} = 4.968$
$E = 0.91 + 0.01 * \sum Scale\ Factors = 1.11$

$$Effort = 2.94 * 4.968^{1.11} * 1.087 = 18.94\ \ Person\ Month$$

For what regards project duration, the following formula define the required months by means of C, a *COCOMO II* constant, the previously computed *Effort*, and F, depending on the five assigned *Scale Factors* values.

$$Duration = C * Effort^F$$

where
$C = 3.67$
$F = 0.28 + 0.2 * 0.01 * \sum Scale\ Factors = 0.32$

$$Duration = 3.67 * 18.94^{0.32} = 9.4\ Months$$

Finally, by means of *Duration* and needed *Effort*, we can get the average number of required team members as

$$Required\ Personnel = \frac{Effort}{Duration} = \frac{18.94}{9.4} = 2.014 \rightarrow 2\ people$$

We believe that these values based on statistical analysis are highly reliable and reflect very well our project size and our initial valutations, based on experience and common sense.

For more detailed information about COCOMO II method and factors identification we suggest to refer to the previously mentioned Model Definiton Manual.

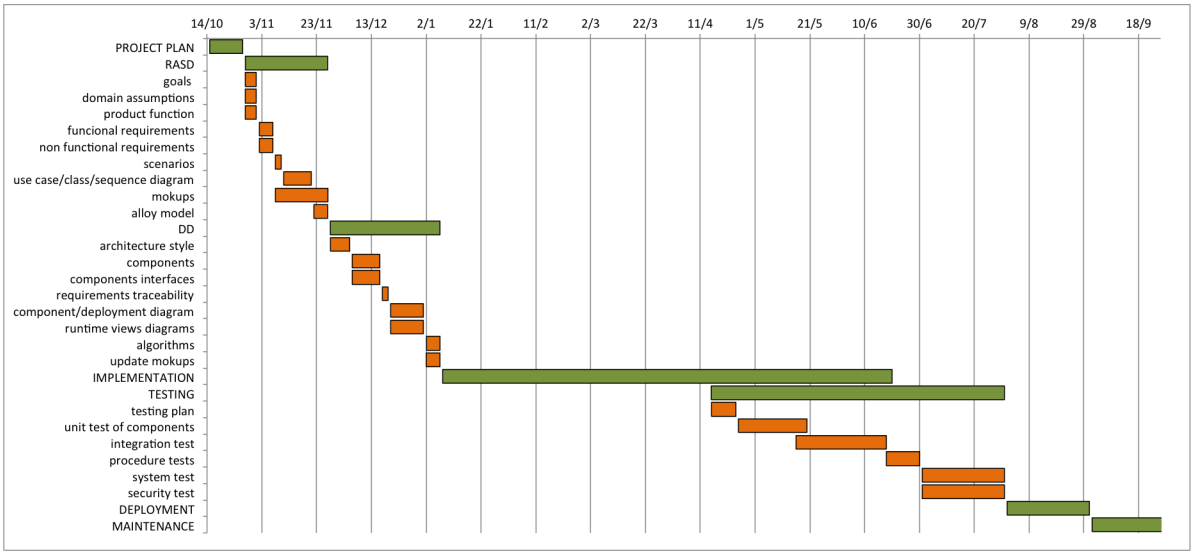
3 Task identification and allocation

The project is going to start on 15th October 2015 and in order to understand when to schedule the final date of the project, it is necessary to identify the tasks of the project. The following picture showed all the tasks with their duration.

CODE	TASK	START DATE	DURATION	END DATE	TEAM ALLOCATION
T1	PROJECT PLAN	15-ott-15	12	27-ott-15	
T2	RASD	28-ott-15	30	27-nov-15	
T2.1	goals	28-ott-15	4	1-nov-15	Andrea
T2.2	domain assumptions	28-ott-15	4	1-nov-15	Lorenzo
T2.3	product function	28-ott-15	4	1-nov-15	Lorenzo
T2.4	funcional requirements	2-nov-15	5	7-nov-15	Andrea
T2.5	non functional requirements	2-nov-15	5	7-nov-15	Lorenzo
T2.6	scenarios	8-nov-15	2	10-nov-15	Andrea
T2.7	use case/class/sequence diagram	11-nov-15	10	21-nov-15	teamwork
T2.8	mokups	8-nov-15	19	27-nov-15	teamwork
T2.9	alloy model	22-nov-15	5	27-nov-15	Lorenzo
T3	DD	28-nov-15	40	7-gen-16	
T3.1	architecture style	28-nov-15	7	5-dic-15	Lorenzo
T3.2	components	6-dic-15	10	16-dic-15	teamwork
T3.3	components interfaces	6-dic-15	10	16-dic-15	Lorenzo
T3.4	requirements traceability	17-dic-15	2	19-dic-15	Lorenzo
T3.5	component/deployment diagram	20-dic-15	12	1-gen-16	Andrea
T3.6	runtime views diagrams	20-dic-15	12	1-gen-16	teamwork
T3.7	algorithms	2-gen-16	5	7-gen-16	Andrea
T3.8	update mokups	2-gen-16	5	7-gen-16	teamwork
T4	IMPLEMENTATION	8-gen-16	164	20-giu-16	teamwork
T5	TESTING	15-apr-16	107	31-lug-16	
T5.1	testing plan	15-apr-16	9	24-apr-16	teamwork
T5.2	unit test of components	25-apr-16	25	20-mag-16	teamwork
T5.3	integration test	16-mag-16	33	18-giu-16	teamwork
T5.4	procedure tests	18-giu-16	12	30-giu-16	Andrea
T5.5	system test	1-lug-16	30	31-lug-16	teamwork
T5.6	security test	1-lug-16	30	31-lug-16	Lorenzo
T6	DEPLOYMENT	1-ago-16	30	31-ago-16	teamwork
T7	MAINTENANCE	1-set-16	1461	1-set-20	teamwork

Highlighted in green, we can see the main tasks that composed the project and in orange, there are the secondary tasks that specify a primary task. Obviously, a primary task ends when all its subtasks are completed. The duration of maintenance is an approximate number because it depends on the life-cycle of the project, so it is too difficult to evaluate a right time. In our case, we identify more or less 4 years of maintenance. The last column identifies which of the team members are are in charge to do that job. The word Teamwork identify the task that is assigned to all the members of the team.

Starting from the table it is possible to generate a Gantt diagram:



Also in this graph, the green bars are related to the main tasks and the orange ones are related to the subtasks that characterize primary tasks.

The project documents redacted are:

- Requirements Analysis and Specification Document
- Desing Document
- Inspection Document
- Integration Test Plan Document
- Project Plan

All these documents are composed of different subparts, that are assigned to each member of the team that develops this project, that are two: Andrea Turconi and Lorenzo Raimondi. The following tables define that division.

RASD:

In common	- preliminary discussion on goals, domain assumptions and constraints - mokups - use case diagram - sequence diagram
Andrea Turconi	- write goals - write functional requirements - write scenarios
Lorenzo Raimondi	- write domain assumption - write product function and product perspective - write non functional requirements - class diagram - alloy model

DD:

In common	- preliminary discussion on Architecture design and components - write components - mokups
Andrea Turconi	- component diagram - deployment diagram - write algorithms
Lorenzo Raimondi	- runtime views - write architecture and styles - write component interfaces - requirements traceability

Inspection Document:

In common	for all the functions analysed we perform - identify the functional role - find all line codes that don't respect the Code Inspection checklist
Andrea Turconi	- FindArgDepositer - minimizeStack
Lorenzo Raimondi	- MethodAnnoter - buildBasicAnnotation

ITPD:

In common	- discussion on which testing strategy is the best to be used - discussion on order of components to be tested
Andrea Turconi	- write integration test cases - write integration test procedures
Lorenzo Raimondi	- write integration strategy - write tools used - write program stubs

PPD:

In common	- Function points and COCOMO estimation
Andrea Turconi	- write task allocation - Gantt diagram
Lorenzo Raimondi	- write risk part

4 Risks analysis

Risks are events whose occurrence can produce negative (in some cases also positive) effects on project purpose. During a software project development, risk management is a very important task to accomplish, in order to reduce and prevent project failure. Since a complete and secure risk management is a quite impossible task, here we report a list of several different risks which our project may unfortunately encounter. Here we present a table reporting the analyzed risks with relative probability and impact over project. The risks have Low, Medium or High probability to happen, and may impact in a Marginal, Serious, Critical and Catastrophic way over projecting process. This risks are also categorized into dimensions, as suggested by Wallace and Keil in their analysis on risk management[2].

Risk	Dimension	Probability	Impact
Personnel shortfalls	Team	Medium	Serious
Changes in membership of project team	Team	Low	Critical
Bad coordination between team members	Team	Low	Serious
Unrealistic project time/cost estimation	Planning&Control	Low	Critical
Shortfalls of external supplied components	Planning&Control	Low	Serious
Software not fitting with purpose	Planning&Control	Low	Catastrophic
Design not flexible and scalable	Planning&Control	Moderate	Critical
Changes in Requirements	Requirements	Low	Catastrophic
Inadequate system security features	Complexity	Moderate	Critical
Detached stakeholders	Corporation	Low	Catastrophic

Personnel shortfalls

A team member may get into problems, like illness, unforeseen or many others. This would mean a high productivity decrease and a probable delay. Un-

[2]L. Wallace and M. Keil. "Software Project Risk and their Effect on Outcomes", Communication fo the ACM, vol 47 number 4, pp. 68-73, 2004.

fortunately these issues may happen without any expectation and there are no preventing action to avoid them.

Changes in membership of project team

For any reason can occure a substitution of a team member, that would reflect into training cost and project delays due to the drawback. Neverthless we think that our team cohesion will quite certainly avoid this type of problem.

Bad coordination between team members

Interaction and task division problems between team members can interfere with the regual project progress. Nevertheless this risk is reduced due to a good cohesion between team members and their near collocation.

Unrealistic project time/cost estimation

Project time and cost estimation are computed by using statistic methods, without a real guarantee about their accuracy. Errors in these extimations are probable, but their accuracy can be eventually improved by a constant revision of project status.

Shortfalls of external supplied components

Since the system uses external services (e.g. Google Maps API), its regular working condition depends on their availability. In case of a protracted downtime, system status can be recovered by using cached data or overriding the external services with approximated internal implementations.

Software not fitting with purpose

The project team may deviate the software development from the initial purpose individuated by stakeholders, for example bending or misunderstanding requirements or taking decision freely without any report. To prevent this possibility is needed a continued attention during all the development process, in particular during requirements identification and architecture design, keeping a frequent and profitable contact between team members and stakeholders.

Design not flexible and scalable

If system architecture lacks in cohesion and modulation, future new implementations or scaling will require bigger efforts. The only way to prevent this risk is to design from the beginning an uncoupled and clever architecture, oriented to scalability.

Changes in Requirements

Changes in Functional Requirements are probably the worst problem that could occur during system development. As much the changes occur faraway from the project starting, as much the effort for recovering becomes bigger. Basing on changes significance these events may cause different mutations, from reimplementing of some features to a project total revision (in the worst case).

Inadequate system security features

System security lacks may cause malicious attacks or sensitive data stealing, affecting users' privacy. To avoid these problems security requirements should be considered and developed during the whole projecting process.

Detached Stakeholders

If stakeholders focuses on project development they could generate motivation and effort in the team. Stakeholders not very interested in the project evolution process may create budget bounds, affect team productivity and the overall system quality. To avoid this chance the developing team should constantly get in touch with stakeholders, update them during projecting process and do its best to make them involved.