Test Plan Document

Michele Madaschi Lidia Moioli Luca Martinazzi

February 2, 2016

Contents

Introduction

1.1 Scope and purpose

In this document we want to give an overall description of the project planning of *My Taxi Service*. We will:

1. identify deliverables and deadlines
2. extimate the total effort required
3. analize possible risks and contengency plans

Function Points

In order to evaluate the cost of the project we have to identify the function points and estimate the complexity of each one. To each point we assign a weight referring to this table:

Function types	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

- Internal Logic File: users (guest, taxidriver and passenger), ride, sharedride, taxiqueue
- External Interface File: gps coordinates, map service
- External Input: login, logout, request, reserve, delete, reserve shared, accept call, refuse call, report, taxi available, taxi not available, change settings
- External Output: message (eta, no taxi message)
- External Inquiry: see profile, see active ride list

2.1 Complexity and cost evaluation

2.1.1 Internal Logic File

According to our previous specification (explained in the RASD and DD documents), **users** and **ride** have to store few informations, thus we can adopt the simple cost weight for those ones. On the other hand, **sharedride** and **taxiqueue** have to store a dynamic list, that require more attention, so we adopt a medium cost weight.

$$4 * 7 + 2 * 10 = 48 \text{ FPs} \tag{2.1}$$

2.1.2 External Interface File

The interactions with **gps** coordinates and the **map service** are very simple, because we need to gather few information from them, so we adopt a simple weight for both of External Internal Files.

$$2 * 5 = 10 \text{ FPs} \tag{2.2}$$

2.1.3 External Input

Most of the external inputs are simple actions involving a few number of entities, therefore we can adopt a simple weight cost for them. **request** and **change settings** however are more complex, and thus require a medium weight cost.

$$10 * 3 + 2 * 4 = 38 \text{ FPs} \tag{2.3}$$

2.1.4 External Output

Sending **eta** requires to access the map service that calculate, on its own, the appropriate value, so we adopt a simple cost weight for message.

$$2 * 4 = 8 \text{ FPs} \tag{2.4}$$

2.1.5 External Inquiry

see profile requires only to send some fields saved in the current user, while **see active ride** list requires to scan the ridehistory and check its status (active or not). Therefore, we adopt a simple cost weight for the former, and a medium cost weight for the latter.

$$1 * 3 + 1 * 4 = 7 \text{ FPs} \tag{2.5}$$

2.1.6 Overall

In summary we have $\text{FPs} = \sum_{i=1}^{5} \text{FP}i = 111$

COCOMO II

3.1 Approach

In order to apply the COCOMOII method, we apply first a simple *Funtion Points to Lines Of Code* conversion. The adjustment factor of 46 is provided by the updated table available at http://www.qsm.com/resources/function-point-languages-table

$$111 \text{ FPs} * 46 = 5106 \text{ SLOCs} \tag{3.6}$$

The effort estimation takes into account a "Nominal" value for all the Cost Drivers and Scale Drivers.

$$\begin{aligned}
&\text{effort} = 2.94 * EAF * (KLOC)^{E} \\
&EAF = 1.0 \\
&KLOC = 5.106 \\
&E = 1.0997 \\
&\text{Therefore, effort} = 2.94 * 1.0 * (5.106)^{1.0997} = 17.66 \text{ Person/Months}
\end{aligned} \tag{3.7}$$

Next, we estimate the project duration using the following equation.

$$\begin{aligned}
&\text{duration} = 3.67 * (\mathit{effort})^{Se} \\
&Se = 0.3179 \\
&\text{Therefore, duration} = 3.67 * (17.66)^{0.3179} = 9.14 \text{ Months}
\end{aligned} \tag{3.8}$$

Last, we determine the number of people required to complete the project within the previously computed parameters

$$\begin{aligned}
&N_{people} = \lceil \mathit{effort}/\mathit{duration} \rceil \\
&Se = 0.3179 \\
&\text{Therefore, } N_{people} = \lceil 17.66/9.14 \rceil = 2 \text{ People}
\end{aligned} \tag{3.9}$$

3.2 Results

We employed an online COCOMOII calculator in order to generate a more fine-grained estimate.
The following results were obtained using the tool available at http://csse.usc.edu/tools/COCOMOII.php

3.2.1 Input (software sizing)

	SLOC	% Design Modified	% Code Modified	% Integration Required	Assessment and Assimilation (0% - 8%)	Software Understanding (0% - 50%)	Unfamiliarity (0-1)
New	5106						
Reused		0	0				
Modified							

3.2.2 Input (drivers)

Precedentedness: Nominal

Development Flexibility: Nominal

Required Software Reliability: Nominal

Data Base Size: Nominal

Product Complexity: Nominal

Developed for Reusability: Nominal

Documentation Match to Lifecycle Needs: Nominal

Architecture / Risk Resolution: Nominal

Team Cohesion: Nominal

Analyst Capability: Nominal

Programmer Capability: Nominal

Personnel Continuity: Nominal

Application Experience: Nominal

Platform Experience: Nominal

Language and Toolset Experience: Nominal

Process Maturity: Nominal

Time Constraint: Nominal

Storage Constraint: Nominal

Platform Volatility: Nominal

Use of Software Tools: Nominal

Multisite Development: Nominal

Required Development Schedule: Nominal

We'll also assume a cost per Person-Month of 2000$

3.2.3 Output

Software Development (Elaboration and Construction)

Effort = 17.7 Person-months
Schedule = 9.5 Months
Cost = $35322

Total Equivalent Size = 5106 SLOC

Acquisition Phase Distribution

Phase	Effort (Person-months)	Schedule (Months)	Average Staff	Cost (Dollars)
Inception	1.1	1.2	0.9	$2119
Elaboration	4.2	3.5	1.2	$8477
Construction	13.4	5.9	2.3	$26845
Transition	2.1	1.2	1.8	$4239

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

Phase/Activity	Inception	Elaboration	Construction	Transition
Management	0.1	0.5	1.3	0.3
Environment/CM	0.1	0.3	0.7	0.1
Requirements	0.4	0.8	1.1	0.1
Design	0.2	1.5	2.1	0.1
Implementation	0.1	0.6	4.6	0.4
Assessment	0.1	0.4	3.2	0.5
Deployment	0.0	0.1	0.4	0.6

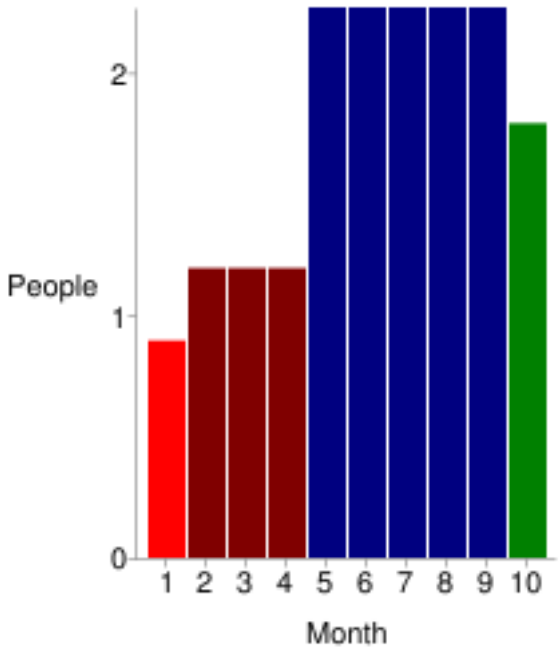

Figure 3.1: Staffing chart

Schedule and resources allocation

4.1 Design

For this project we had to arrange several deliverables, each one with a strict deadline. In particular:

1. RASD - 06/11/2015
2. DD - 04/12/2015
3. INSPECTION - 05/01/2016
4. INTEGRATION TESTING - 21/01/2016
5. PROJECT PLANNING - 02/02/2016

To accomplish the work we folllowed the instructions of each assignment, referring to course material and past years projects.
Our team strategy was definying all together the main guidelines of the document to be created, with one scribe. Then at home each of us expanded and clarified the content previously decided.
A special case was the inspection document, when we associated randomly the points in the checklist to each member.
The whole project lasted 4 months, with an individual work of 110 hours/person approximately.

4.2 Implementation

The following section describes the partitioning of the implementation tasks, in compliance to the project's design document and COCOMO II's data.
The job will be distributed to 2 developers, and the project will be split up into smaller releases. Every release is scheduled in the same way: an initial phase, during which development of code and unit tests are carried out in

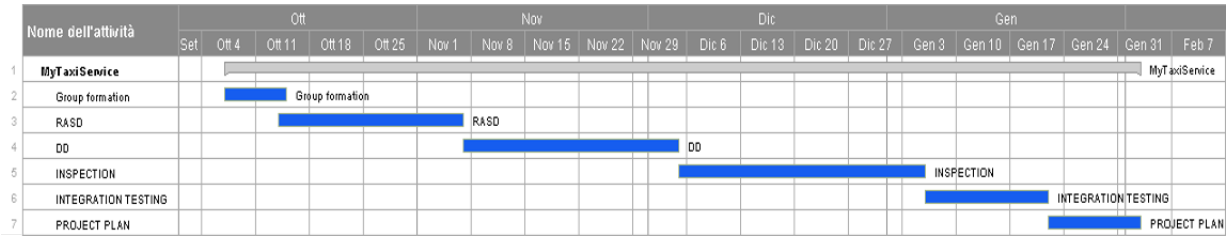

Figure 4.2: Gantt's diagram (design phase)

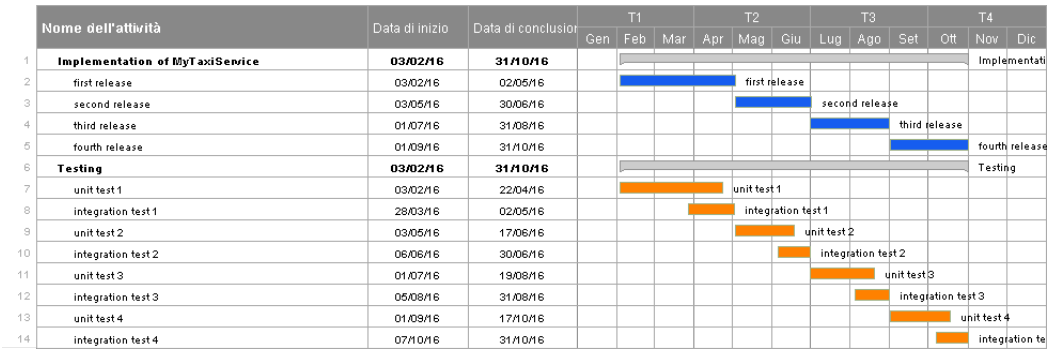

Figure 4.3: Gantt's diagram (implementation phase)

parallel; an intermediate phase, in which integration tests begin, and a final phase, when unit tests are completed and the code is fixed as needed.
More precisely, the plan includes 4 releases:

1. In the first release, we aim to provide a prototype of the application, capable of executing the basic operations; *developer 1* is tasked with writing the client-side "major classes", including the ones devoted to the graphic user interface; on top of that, (s)he will also add *actions* and *activities*, which allow to call a taxi, append it to a queue and responding to a call;
   developer 2 is tasked with the server-side logic, required for the correct working of the client functions, but this first release will feature only a single *universal user*. (s)he will also work on the map-service integration.
   Each developer will also test their own components.

2. In the second release, *developer 1* is tasked with adding support for the user-management activities (i.e. login, logout) and the *master terminal* user interface.
   developer 2 will work on the server-side user management.
   Each developer is still required to test their own components, but, given the significative job already assigned to *developer 2*, *developer 1*

will work on the integration tests.
This version is already deliverable to the end users, albeit with missing features.

3. In the third release, *reservations* and *shared rides* will be implemented, following the same work distribution as before (*developer 1* on the client, *developer 2* on the server, each one write their own tests)

4. Finally, *developer 1* will implement the *report* system and the *ride cost calculation* system, while *developer 2* will implement the *API*.

Risk evaluation

5.1 Risk evaluation and avoidance

In this section we describe the risk identification: in Table 1 for each risk we assign a probability and the degree of seriousness, in Table 2 we define a plan to follow in case the risk occurs.

Risk	Probability	Effects
1.Key staff are ill at critical times in the project	Moderate	Serious
2.Changes to requirements that require major design rework	Moderate	Serious
3. Loss of data	Low	Catastrophic
4. Poor collaboration among team members	Moderate	Serious

Risk	Strategy
1.	Each member is aware of the job done by other components so that he can review/finish the task if someone gets sick
2.	Pay attention and if needed ask for clarification
3.	Keep all material synchronized with Github
4.	Keep in good relationship and talk about the project issues