

Politecnico di Milano

A.A. 2015-2016

Software Engineering 2: “myTaxiService”

Project Plan

Luiza Bentivoglio, Michele Cantarutti

2 February 2016

Summary

**1. INTRODUCTION ............................................................................................................4**

**2. PROJECT ESTIMATION..............................................................................................6**

2.1 SIZE ESTIMATION..............................................................................................................................6

2.2 EFFORT AND COST ESTIMATION………..............................................................................................6

**3. TASKS………………………..............................................................................................6**

**1. Introduction**

This document is meant to be a project plan for the myTaxiService project. The first part of the document will focus on the estimation of the project size (by applying Function Points) and of the effort and cost (by applying COCOMO). Next, we’ll identify the tasks of our project and their schedule, while allocating the resources to each one of them. Lastly, we’ll define the risks of the project.

**2. Project estimation**

**2.1 SIZE ESTIMATION**

To estimate the size of our software, we’re going to use the Function Points approach. Function points measure software size and, by definition, a function point is a unit of measurement to express the amount of business functionality a software provides to a user. We retrieved the functionalities our software must provide from our RASD and then we evaluated the complexity of each one of them. Moreover, we distinguished five categories of functionalities, sticking with the standard FP approach, as in the model laid by Allan Albrecht, as follows:

-**External Input**: elementary operation to elaborate data coming form

the external environment;

-**External Output**: elementary operation that generates data for the

external environment (it usually includes the elaboration of data from logic files);

-**External Inquiry**: elementary operation that involves input and output

(without significant elaboration of data from logic files).

-**Internal Logical File** (**ILF**): homogeneous set of data used and managed by the application;

-**External Interface File** (**EIF**): homogeneous set of data used by the application but generated and maintained by other applications;

As for the points given to each functionality, we’re once again sticking with the original model of the FP approach, as shown in the table below:

|  |  |  |  |
| --- | --- | --- | --- |
| Function types | Complexity | | |
|  | Simple | Medium | Complex |
| External Inputs | 3 | 4 | 6 |
| External Outputs | 4 | 5 | 7 |
| External Inquiry | 3 | 4 | 6 |
| ILF | 7 | 10 | 15 |
| EIF | 5 | 7 | 10 |

Now that we have defined the estimation model we intend to use and that we have specified what categories of functionalities our system provides, we can pass on to the proper estimation analysis. To do so, we went through each category, and for each category we analyzed the functionalities that belong to it, by rating the complexity of each one of them. Thus, we assigned the amount of points we felt appropriate for each functionality, based on its complexity.

**-Internal Logical File** (**ILF**): the application stores information about users, taxis, queues, zones and users’ requests. All of these entities have a simple structure composed of a small number of fields, apart from the users’ requests which have quite a few fields. Therefore we think it should be necessary to adopt medium complexity for the requests, whereas the other entities should have a simple complexity. We get 4 x 7 + 1 x 10 points, which add up to 38 FPs, concerning ILFs.

**-External Interface File** (**EIF**): there is only one operation that belongs to this category, and that is the payment, which will involve an interaction with an external system, that is a bank. The fact that interactions with an external system can be difficult to manage, coupled with the fact that transferring money requires high security, leads us to deem this operation a very complex one. Therefore we get 10 FPs.

-**External Input**: the application interacts with the users as follows:

• Login/logout: these are simple operations, so we can adopt the simple

weight for them. 2 x 3 = 6 FPs

• Sign up: this is a simple operation, so we can adopt the simple weight. 1 x 3 = 3 FPs

• Making a request/reservation: these operations involve quite a few entities and imply the creation of a new one, therefore, they’re highly complex: 2 x 6 = 12 FPs

• Cancelling a request/reservation: these operations are of medium complexity because the system also needs to notify other actors. We get 2 x 4 = 8 FPs

• Modifying a reservation: this operation is of medium complexity for the same reason as the operations above, because it involves notifying other actors. Therefore we get 1 x 4 = 4 FPs

• Taxi drivers can confirm their availability. This is a simple operation. 1 x 3 = 3 FPs

• Taxi drivers can accept calls. This operation is simple, so 1 x 3 = 3 FPs

-**External Output**:

• The system can notify the users about a modification or cancellation of a request. This is a simple operation, therefore we get 1 x 4 = 4 FPs

-**External Inquiries**: the application allows the users to request information

about their account, in particular:

• The passengers can view his/her profile and his/her history of requests and reservations. Retrieving this information is a simple information, so the cost is 2 x 3 = 6 FPs

**Total number of FPs:** the total number of Function Points adds up to 97 points. If we had historical data about previous projects, we could estimate the effort necessary for this project, based on the number of function points. Now we’re going to make another estimate of our project using another approach, that is COCOMO II.

**2.2 EFFORT AND COST ESTIMATION: COCOMO II**

**2.2.1 Brief Introduction**

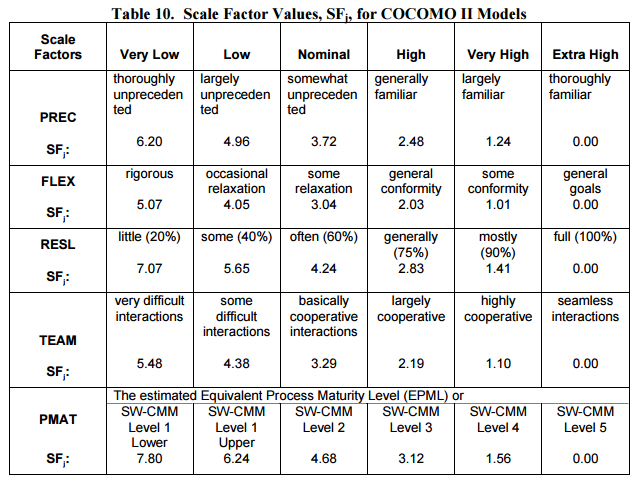
This estimate is achieved through a complex, nonlinear model that takes in account the characteristics of the product but also of people and process.

All the tables used in this analysis have been taken from COCOMO II, Model Definition.

Manual at:

<http://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_modelman2000.0.pdf>

**2.2.2 Scale Drivers**

****

These values are evaluated according to the following table:

• **Precedentedness:**

**r**eflects the previous experience of the organization with this type of project. We only had one previous experience in software engineering and it wasn’t very similar to this one. Therefore we think we didn’t have much experience from before and the value for this scale driver is low.

• **Development flexibility:**

It reflects the degree of flexibility in the development process. The customer set

the general specifications without going too much into detail. For this reason this value will be high.

• **Risk resolution:**

Reflects the extent of risk analysis carried out. We carried out a single risk analysis, thus we think the value is nominal.

• **Team cohesion:**

Reflects how well the development team know each other and work together.

Even though we worked as a team for the first time, we feel like we managed to work together well and we never had any major communication problems. Plus, we had similar schedules, which allowed us to be able to work together often. In conclusion, we believe the value of this scale driver is high.

• **Process maturity:**

This was evaluated around the 18 Key Process Area (KPAs) in the SEI Capability

Model. Because of the goals were consistently achieved these values will be set to high, level 3. (da mettere a posto)

The results are resumed in the following table:

|  |  |  |
| --- | --- | --- |
| **Scale Driver** | **Factor** | **Value** |
| Precedentedness Low  Development Flexibility  Risk Resolution  Team Cohesion  Process Maturity | Low  High  Nominal  High  High | 4.96  2.03  4.24  2.19  3.12 |

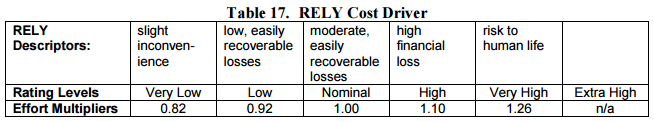
**Total:** 16.54

**2.2.2 Cost Drivers**

**• Required Software Reliability:**

This is the measure of the extent to which the software must perform its intended function over a period of time.

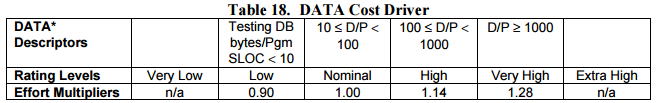
Software failures don't have critical consequences so this parameter is set to low, because this software simply manages taxis and not something critical like for example medical equipment.



**• Data Base Size:**

This measure attempts to capture the effect large data requirements have on product development. It translates the effects that large data have in our application. Our application only manages…nominal value perché application semplice

*Our test database size is equal to 323.0 KB and the program size dovrebbe venire all’incirca 6000 SLOC, the division D/P = 18.58 and then this parameter has a nominal value.*



**• Product Complexity:**

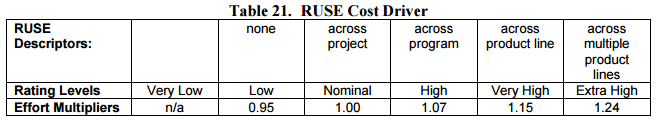
Set to nominal according to the new COCOMO II CPLEX rating scale.

****

• **Required Reusability:**

This cost driver accounts for the additional effort needed to construct components intended for reuse on the current or future projects. This effort is consumed with creating more generic design of software, more elaborate documentation, and more extensive testing to ensure components are ready for use in other applications. High perché applicazione semplice

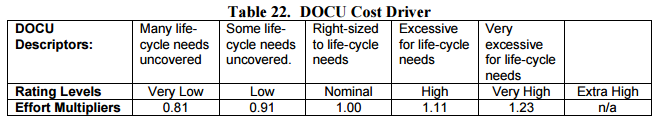
In our project there are different reusable components since our aim was to design the system as modular as possible. This parameter is therefore, set to high.



**• Documentation match to life-cycle needs:**

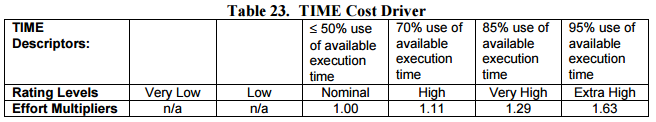
This parameter describes the relation between the provided documentation and the application requirements. Its suitability is setted to nominal since each aspect of our system to be described has been expressed in the RASD or in the DD. Nominal + non abbiamo fatto il codice quindi boh

On the other hand, there is no part of those document unrelated to the actual phase of the development the document is addressed to.

****

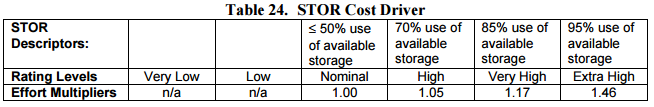
**• Execution Time Constraint:**

In our case this parameter is not relevant so is reasonable to set it as very low.



**• Main Storage Constraint:**

This rating represents the degree of main storage constraint imposed on a software system or subsystem. In our project this parameter is not relevant so is set as very low.



**• Platform Volatility:**

"Platform" is used here to mean the complex of hardware and software (OS, DBMS, etc.) the software product calls on to perform its tasks. If the software to be developed is an operating system then the platform is the computer hardware. If a database management system is to be developed then the platform is the hardware and the operating system. If a network text browser is to be developed then the platform is the network, computer hardware, the operating system, and the distributed information repositories. The platform includes any compilers or assemblers supporting the development of the software system. This rating ranges from low, where there is a major change every 12 months, to very high, where there is a major change every two weeks.

High e basta

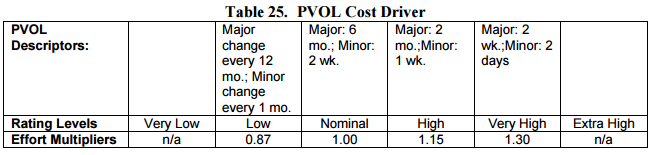
*In our application is reasonable to consider as platforms the DBMS, the operating*

*system, the browser that perform injections and the hardware as far as the*

*environment concerns. We have to consider also the compiler and the webServer*

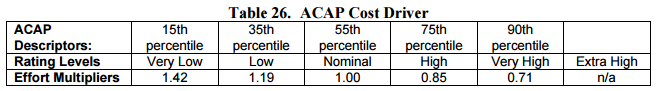
*that has taken an important role in the developing phase. (e l applicazione mobile come piattaforma…) The platform shouldn't*

*change too often so this value is set to low.*

**

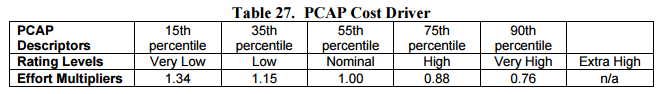
**• Analyst Capability:**

Analysts are personnel that 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. We set this value to high because we intend to focus on the analysis of the project a lot during the RASD and DD phases.

****

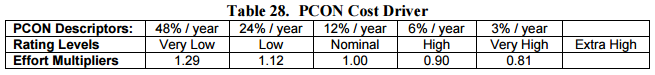
**• Programmer Capability:**

Evaluation should be based on the capability of the programmers as a team rather than as individuals, due to some small problems on it, this value it's set to nominal.



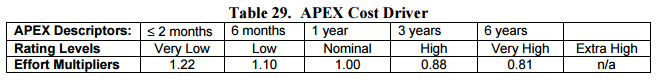
**• Personnel continuity:**

The rating scale for PCON is in terms of the project’s annual personnel turnover: we set it to very low.



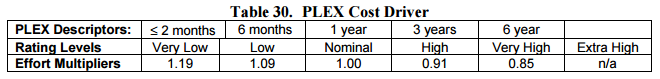
**• Application Experience:**

This rating is dependent on the level of applications experience of the project team developing the software system or subsystem. This is our first experience in this type of project, so this value is equal to low.



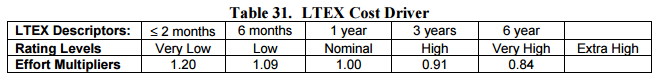
**• Platform Experience:**

Our average knowledge about platforms, such as databases, web and mobile platforms, server side development are around 6 months because we developed a project the past semester, so this parameter is set as low.



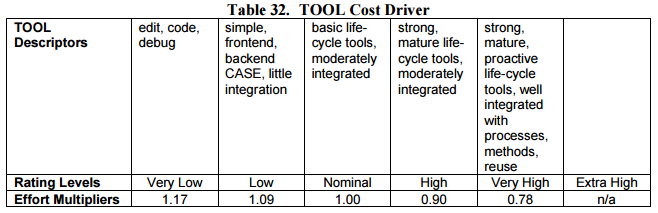
**• Language and Tool Experience:**

This is a measure of the level of programming language and software tool experience of the project team developing the software system or subsystem. We’ve only had one programming experience in one project, therefore this value is Low for us.



**• Usage of Software Tools:**

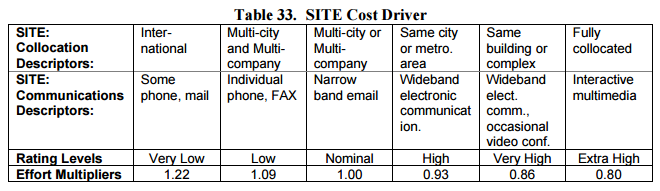
We will use NetBeans, Maven, AndroidStudio and NotePad to develop our project. We will use Git for the repository management. This value should be set to high.



**• Multisite development:**

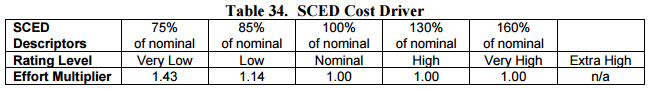
This parameter reflects how we handled the distribution of development over

distance and multiple platforms. We think the value should be set to high because the goal of our project is to make the application as versatile as possible, so that it can be used on any platform.



**• Required development schedule:**

This rating measures the schedule constraint imposed on the project team developing the software. The ratings are defined in terms of the percentage of schedule stretch-out or acceleration with respect to a nominal schedule for a project requiring a given amount of effort. This parameter should be set to high.



Our results are expressed in the following table:

|  |  |  |
| --- | --- | --- |
| **Scale Driver** | **Factor** | **Value** |
| Required Software Reliability  Data Base Size  Product Complexity  Required Reusability  Documentation match to life-cycle needs  Execution Time Constraint  Main Storage Constraint  Platform Volatility  Analyst Capability  Programmer Capability  Personnel Continuity  Application Experience  Platform Experience  Language and Tool Experience  Usage of Software Tools  Multisite development  Required development schedule | Low  Nominal  Nominal  High  Nominal  Very Low  Very Low  High  High  Nominal  Very Low  Low  Low  Low  High  High  High | 0.92  1.00  1.00  1.07  1.00  n/a  n/a  1.15  0.85  1.00  1.29  1.10  1.09  1.09  0.90  0.93  1.00 |

**Product:** 1.36

***2.4. Effort Equation***

To pass from FP to SLOC we use an average conversion factor of 46 as described

at http://www.qsm.com/resources/function-point-languages-table, an updated

version that adds J2EE of the table included in official manual

(<http://sunset.usc.edu/research/COCOMOII/Docs/modelman.pdf>).

97 FPs \* 53 = 5141 SLOC

This final equation gives us the effort estimation measured in Person-Months (PM)

Effort := A \* EAF \* KSLOCE

Where:

A → 2.94 (for COCOMO.2000)

EAF → product of all the cost drivers, equal to : 1.36 ;

E → exponent derived from Scale Drivers. Is calculated as:

B + 0.01 \* sum{i} SF[i] := B + 0.01 \*16.54 = 0.91 + 0.1654 = 1.0754;

in which B is equal to: 0.91 for COCOMO.2000 .

KSLOC → estimated lines of code using the FP analysis: 5.141

With this parameters we can compute the Effort value, that is equal to:

Effort := 2.94 \* 1.36 \* 5.1411.0754 = 23.3 PM

**2.5. Schedule Estimation**

As far as the schedule estimation we are going to use the following formula:

Duration := 3.67 \* EffortF

Where:

F := 0.28+0.2\*(E – B) = 0.28+0.2 \*(1.0754 – 0.91) = 0.28+0.2\*( 0.1654 ) = 0.31308

Follows then:

Duration := 3.67 \* 23.30.31308 = 9.8 Months

Now we can estimate the number of people needed to complete the project with

the following formula

Npeople = effort/Duration

Npeople = 23.3/9.8 = 2 people

**3. Tasks**

1.1 TASK

Documentazione:

RASD:

T1: introduction

T2: ACTORS IDENTIFYING

T3: REQUIREMENTS

T4: SCENARIOS IDENTIFYING

T5: UML MODELS

T6: ALLOY MODELLING

T7: USED TOOLS

T1+T2+T3 1 GIORNO

T4+T5 3 GIORNI

T6+T7 2 GIORNI **totale 5 giorni rasd**

DD:

T8: INTROCTION

T9: ARCHITECTURAL DESIGN

T10: ALGORITHM DESIGN

T11: USER INTERFACE DESIGN

T12: REQUIREMENTS TRACEABILITY

**TOTALE: 3 GIORNI**

TEST PLAN:

T13: INTRODUCTION

T14: INTEGRATION STRATEGY

T15: INDIVIDUAL STEPS AND TEST DESCRIPTION

T16: TOOLS AND TEST EQUIPMENT REQUIRED

T17: PROGRAM STUBS AND TEST DATA REQUIRED

**TOTALE: 1 GIORNO**

Programma:

T18: convertire la distribuzione dei component del Sistema nel diagramma delle classi (MVC -> model) **1 giorno**

T19: sviluppare il diagramma delle classi che si occuperà delle informazioni di comunicazione attraverso la rete tra i vari partecipanti del sistema (RMI, socket... ->controller) **3 giorni**

T20: sviluppare il modello + test **3 giorni**

T21:sviluppare il controller+test **4 giorni**

T23: eseguire test della comunicazione tra model+controller **1 giorno**

T24: sviluppo database (registrazione server + modello ) **2 giorni**

T25:svilluppare le varie view : piattaforma web e android (grafica e sviluppo) **2 settimane**

T26:eseguire test finali (view+model+controller) **3 giorni**

Finish

**Tempo totale -> 40 GIORNI.**