Project Plan

Figure 1: Politecnico di Milano

Version 1.0

Release date : 02/02/2016

- Claudio Cardinale (mat. 849760)
- Gilles Dejaegere (mat. 853950)
- Massimo Dragano (mat. 775392)

Contents

1. Introduction

1.1. Revision history

Date	Version	Description	Authors
02/02/2016	1	Original Version	C. Cardinale, G. Dejaegere and M. Dragano

1.2. Purpose and Scope

1.3. List of Definitions and abbreviations

- Function Points estimation : the function point estimation is an estimation of the size of an specific software in terms of line of codes.
- SLOC: **TODO**
- FP: **TODO**
- Gannt diagram: **TODO**
- Montecalro analysys: a statistical method that perform several iteration of some things to test the behavior in the feasible worst cases.
- standard deviation: **TODO**

1.4. List of Reference Documents

2. Estimate size, effort and cost

2.1. Function points

Function Points estimation : the function point estimation is an estimation of the size of an specific software to be based on its functional requirements. The count of function points of the software is language and technology independant. The size in terms of lines of codes of this software can be calculated using the following formula :

LOC = FP * AVC

where : * LOC = lines of code * FP = total estimated function points of the software * AVC = language specific constant varying from 2-40 for a fourth generation programming language.

Function points are classified under different types. For each type, a different weight is also assigned for each function point according if the concerned function is estimated as being simple, complex, or average. For the estimation of the size of our MyTaxiService application, we will uses the weights indicated in the table here under. These numbers are taken from the slides "Cost Estimation" provided by the professor Damian Andrew Tamburri.

function type	**Low**	**Average**	**High**
Internal Logic Files	7	10	15
External Interface Files	5	7	10
External Inputs	3	4	6
External Outputs	4	5	7
External Inquiries	3	4	6

2.1.1 Internal Logic Files

The different ILFs that will be used by our systems can be deduced from the class diagram representing the `model` of our application that is located in the RASD and is shown here under.

[Model taken from the RASD][model]

For each component, its complexity and the associated function points are listed in the table here under :

ILF	**Complexity = Function Points**
Zone	High = 15
Driver and Taxi	High = 15

ILF	Complexity = Function Points
Request	Average = 10
Ride	High = 15
StopPositions and Client	Low = 7
Position	Low = 7
Total	69

Since each Driver has exactly one taxi, and each stop position has exactly one client, these entities are managed and stored together for more simplicity.

2.1.2 External Interface Files

The application does not have to manage date furnished and maintained by external services.

Do we have to insert external services like push notification or SMS?

2.1.3 External Inputs

The application interacts both with drivers and with clients, each having their own set of possible interactions.

Concerning the driver, the system has to enable him to:

- login
- change availability : this operation consists in changing the availability of the driver object. If the driver becomes available, the driver object has to be put at the end of the appropriate waiting queue. In the other case, when the driver is not available anymore, he has to be withdrawed from the appropriate waiting queue. The appropriate waiting queue is found using the position of the driver. This operation is evaluated as complex.
- change position : this operation takes place when the driver changes of zone. If the driver concerned has his status "available", he has to be withdrawed from the queue of his previous zone and add to the queue of the new zone. This operation is complex.
- accept request : when the driver receive a request via the push notification service, his has to answer. Positive answer are handeled by the RideController (for the creation of the ride) and transmitted to the QueueManager (to prevent him asking the next driver). Negative answers are handeled by the RequestController that transmits it to the QueueManager so that he can ask the next taxi driver and put the former at the end of the queue.

Concerning the client, the system has to enable him to:

- require ride : this request requires to find a taxi driver avaible and ask him if he accepts the ride. The managing of the answer of the taxi driver and the mananging of the queue of taxi as already been taken into account for the taxi driver and this functionality is therefore judged as of an average complexity.
- reserve a ride : the reservations are preproccessed by a scheduler (to be triggered on the appropriate time) and are then managed as requests. The additionnal complexity of the reservation is estimated as low.
- share the ride : Shared request or reservations must be pre-processed, aiming to merge then in shared requests. This is an complex operation.

EI	**Complexity = Function Points**
login	Low = 3
change availability	High = 6
change position	High = 6
answer request	High = 6
require ride	Average = 4
reserve ride	Low = 3
share the ride	Low = 3
Total	31

2.1.4 External Outputs

Our application must send information to the clients and the drivers. Ride information : the system has to inform the client about the incoming taxi and the estimated arrival time. In addition, if the client has requested for a shared ride, the system has also to inform him about the estimated cost of his ride. The estimation of the cost of the ride is an complex operation since it dependes on the length of the client's itinary, but also on the itinaries of the other clients that share the ride (in order to know the proportion of the first clients itinary that is actually shared and with how many other clients). request information : the system has to notify the appropriate driver with the information concerning the appropriate request. These notification are send using an external gateway wich add some complexity to the sending of information. The sending of the request information to the taxi driver is therefore evaluated as having an average difficulty.

EO	**Complexity = Function Points**
ride information	High = 7
request information	Average = 6
Total	13

2.1.5 External Inquiries

There is only one type of external inquiries that must be managed by our system and it consist in showing to the taxi driver his position in the queue when the taxi driver requires it. This operation is estimated as complex since it has to retrieve this information by scanning the appropriate queue of drivers.

EI	**Complexity = Function Points**
driver position	High = 6
Total	6

2.2 Cumulated Function Points and code size estimation

Now that the amount of function points for each function type of our application has been estimated we can easily dress a table summarising the situation :

Function Type	**Function Points**
Internal Logic Files	69
External Interface Files	0
External Inputs	31
External Outputs	13
External Inquiries	6
Total	119

As we can see, there is an huge disparity between the values of each function type. Nearly 60% of the function points are associated to the internal logic, about 25% for the external input, and only slightly more than 15% is associated to the external output/inquiries. This can be explained by the purpose of the application. Indeed, MyTaxiService is an application aimed at managing taxi drivers waiting queue and helping user order a taxi-ride (which again requires accessing and modifying these waiting queues). On the other hand, the difference between the internal logic and the external inputs seems overestimated. This might be due to the fact that the weight in function points of the internal logic files is more than twice the one of the weight of the external inputs (cfr. section 2.1).

2.2. COCOMO

2.2.1. Introduction to COCOMO

To proceed with the calculation of SLOC we have to convert the FP obtained with the formula given in section 2.1. To do that we need a conversation factor

(AVC), we have found the following site that gives some converstion factors for some language: http://www.qsm.com/resources/function-point-languages-table but there is not php with laravel, so we considered the factor of J2EE that is analogous with php + laravel since they are booth two MVC web application framework and object languages. So the factor that we will use is **46**.

This leads to an ammount of line of codes equals to :

$$LOC = FP * AVC = 117 * 46 = 5382.$$

To perform the estimation of the effort needed to produce these line of codes, and therefore, our application, we will use the parameters of the offcial table http://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_modelman2000.0.pdf **TO IMPROVE**

2.2.2. Scale driver

In the following figure (found in the official documentation) we can see the values of the different scale drivers associated with different rating levels.

Copy table from officialfile

The different values chosen for our application can be seen in the table here under.

Scale Driver	rating level	value
Precedentedness	Nomimal	3.72
Development Flexibility	High	2.03
Risk Resolution	Nominal	4.24
Team Cohesion	Very High	2.19
Process maturity	High	3.12
Total		15.3

2.2.3. Cost driver

The different values chosen for our application can be seen in the table here under.

Cost Drivers	rating level	value
Required Software Reliability	Low	0.92
Data Base Size	High	1.14
Product Complexity	High	1.17
Developed for Reusability	Low	0.95
Documentation Match to Life-Cycle Needs	Nominal	1

Cost Drivers	rating level	value
Execution Time Constraint	Nominal	1
Main Storage Constraint	n/a	n/a
Platform Volatility	Nominal	1
Analyst Capability	Nominal	1
Programmer Capability	Nominal	1
Personnel Continuity	Very High	0.81
Applications Experience	High	0.88
Platform Volatility	Low	0.87
Platform Experience	Nominal	1
Language and Tool Experience	High	0.9
Use of Software Tools	Nominal	1
Multisite Development	Nominal	1
Required Development Schedule	Nominal	1
Total		0.65

2.2.4. Effort, duration and size of the team

The Cocomo II approches gives an equation for calculating the effort :

$$Effort = 2.94 \cdot EAF \cdot (KSLOC)^E$$

Where :

- Effort = estimated total number of Person-months needed
- EAF = product of the cost drivers, in our case EAF equals 0.65
- KSLOC = number of thousands of lines of codes estimated to be needed for our application. KSLOC has been approximated to 5.382
- E = Exponent derived from the scale drivers. E equals 0.91+(sum of the scale drivers)/100, in our case : 1.063

Using the parameters specific to the MyTaxiService application we arrive to an estimation of :

$$Effort = 2.94 \cdot 0.65 \cdot 5.382^{1.063} = 11.43PM$$

To estimate the duration of the development of the application, another formula is given by the Cocomo II approach :

$$Duration = 3.67 \cdot (Effort)^{0.28+0.2\cdot(E-0.91)}$$

Where the Effort and E are the same as the ones for the “Effort equation”.

We obtain :

$$Duration = 3.67 \cdot 11.43^(0.28 + 0.2 \cdot (1.063 - 0.91)) = 3.67 \cdot 11.43^{0.31} = 7.81$$

Finally, knowing the approximative effort and time needed to carry through the project, we can easily compute the size of the effort team needed :

$$\#people = Effort/Duration = 11.43/7.81 = 1.43$$

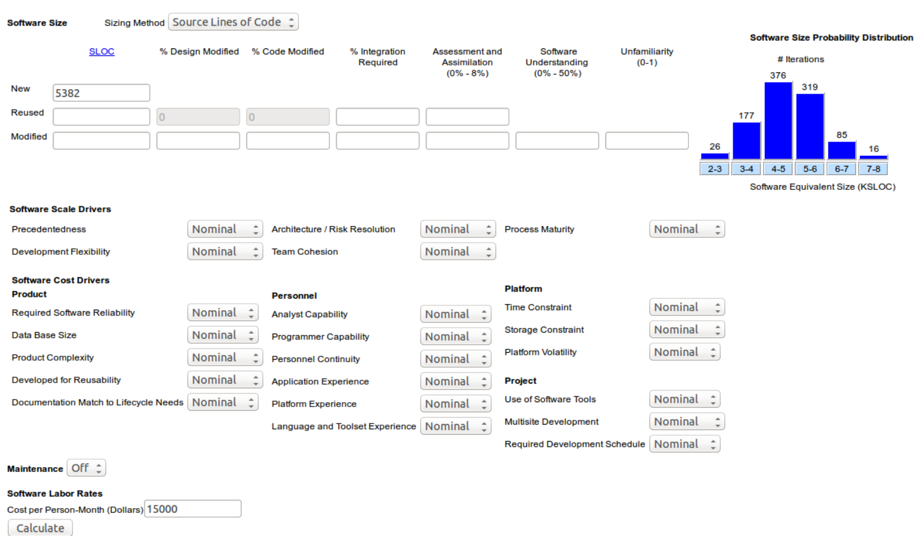
It is of course not possible to use 1.43 workers but we can easily imagine two people starting the project and one dropping out at about the half of the project. However, estimating the size of the team using the Effort and Duration obtained using the Cocomo II approach could seems paradoxal in the sens that an evaluation of the qualities of the developpong team needs to be done to compute the some drivers.

2.2.5. Detailed report

We decided to give a detailed report performed by the following external site http://csse.usc.edu/tools/COCOMOII.php.
This report gives us important information about effort like the effort estimated per month and per part.

We used nominal value to obtain a result analogous to the result obtained by our hands.

Improve introduction



Results

Software Development (Elaboration and Construction)

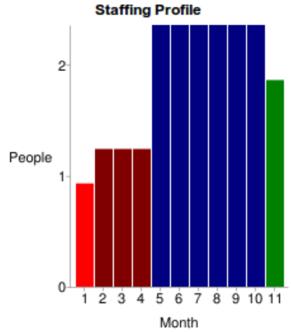
Effort = 18.7 Person-months
Schedule = 9.6 Months
Cost = $280709

Total Equivalent Size = 5382 SLOC

Acquisition Phase Distribution

Phase	Effort (Person-months)	Schedule (Months)	Average Staff	Cost (Dollars)
Inception	1.1	1.2	0.9	$16843
Elaboration	4.5	3.6	1.2	$67370
Construction	14.2	6.0	2.4	$213339
Transition	2.2	1.2	1.9	$33685

Staffing Profile



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

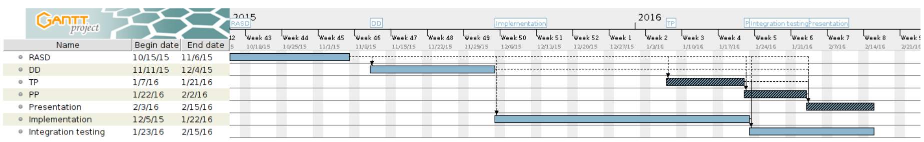
Phase/Activity	Inception	Elaboration	Construction	Transition
Management	0.2	0.5	1.4	0.3
Environment/CM	0.1	0.4	0.7	0.1
Requirements	0.4	0.8	1.1	0.1
Design	0.2	1.6	2.3	0.1
Implementation	0.1	0.6	4.8	0.4
Assessment	0.1	0.4	3.4	0.5
Deployment	0.0	0.1	0.4	0.7

3. Tasks

We considered all real assignments except the code inspection, since it is not related to this application, with the real deadlines.
We also considered other tasks like the implementation, that we decided to start immediately after the design, because we have everything needed to start it. We don't use the duration provided by the cocomo because we wouldn't have had the time to complete it in time (considering the course length). **Write in hypothetical form?**

We used a Gantt diagram to show the tasks with the deadlines.

TODO improve Introduction



FIX dates for development, presentation and so on FIX dashed bars

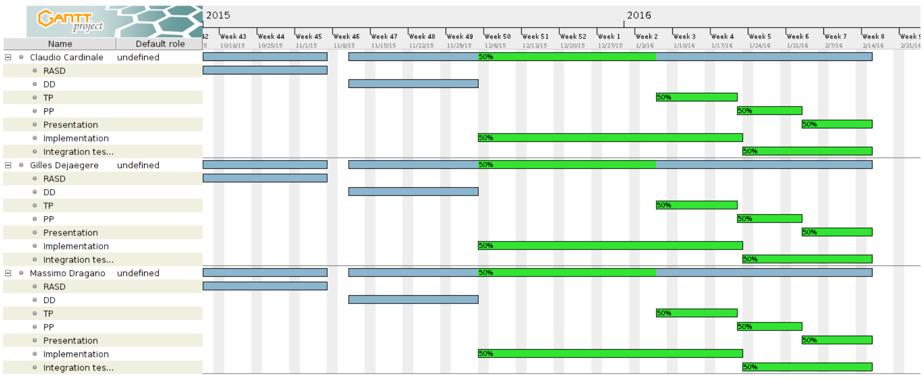
Write all words in glossary

4. Allocate resources

Since for the parts real done everyone worked with analogous working hours, we considered that for all parts everyone works on them with the same amount of hours.
We have decided to dedicate to development only the 50% of time because during it there are other parts to do, Christmas holiday and some in itinere exams.

We used a diagram to show the allocation of human resources linked with the tasks

TODO explain what we mean with 100% TODO improve Introduction



FIX dates, consider the 50% in the development

Write all words in glossary

5. Risks

5.1. Project risks

- requirements change: requirements are changed by the committer.
- bad schedule: tasks previsions may be unrealistic.
- regulatory changes: contry may change it's regulatory policy concerning our operation field.

5.2. Technical risks

- wrong UI: the developed User Interface does not satify the committer.
- bad external data source: external data sources may be badly exposed and organized.
- overload: the system cannot satify a large amount or requests.

5.3. Business risks

- too few clients: the sales team did not find enought customers for the product.
- competitors: another company can develop a similar product, reducing our market opportunities.

5.4. Risks probability and severity

Risk	Probability	Severity
requirements change	L	Critical
bad schedule	M	Critical
regulatory changes	L	Marginal
wrong UI	L	Marginal
bad ext. data source	H	Critical
overload	H	Critical
too few clients	M	Catastrophic
competitors	L	Critical

5.5. Contingency plan

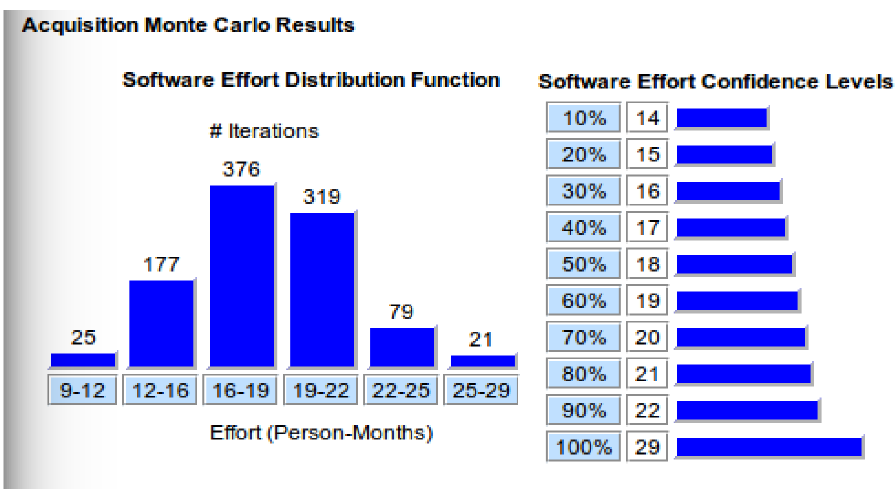
Risk	Strategy
bad ext. data source	Use the Adapter pattern to minimize changes to program code, developing an adapter for any new external source
overload	move the system to an highly scalable platform like AWS
too few clients	promote our product using some advertising

5.6. Montecarlo analysis

We decided also to provide the Montecarlo risk analysis, about effort.

In this case the Montecarlo analysis iterations are made by different (random) values for (k)slock with the real value as average and with a fixed standard deviation. So we have a normal distribution of some values for (k)slock.
In this case the system calculate the efforts for every different value of (k)slock and we obtain a normal distribution of different efforts.
The distribution used is inserted on the detailed report of section 2.2.. **insert link**

This is very useful to give us an idea about the feasible worst case and to test the robustness of the project plan.



6. Used tools

- Github: for version controller
- Gedit and ReText: to write MarkDown with spell check
- Ganttproject: to draw Gantt diagrams
- Pandoc: to create pdf

7. Hours of work

Claudio Cardinale

Gilles Dejaegere

Massimo Dragano