POLITECNICO DI MILANO

Master Degree course in Computer Science Engineering

Fifth Assignment: Project Plan Document

Instructor: Prof. Elisabetta Di Nitto

Authors:

Brunato Andrea	851413
Costantini Lorenzo	852599
Dell’Orto Alessandro	853050

TABLE OF CONTENT

1. Introduction

The purpose of this document is to make a rough estimation of the time and the effort for the realization of our system. We used two algorithmic methods: the functional point approach and the Cocomo model.
Other subjects treated are the analysis of all the type of risks concerned to the software realization and the allocation of work to the development team consisting of the three authors of this document.
This document should have been written before starting the elaboration of the RASD, imagining the general requirements and risks related to the software development, but actually this part of the project has been written at last, after the integration testing document and the DD. So, this document is totally coherent with the RASD and DD even if in a real situation this could be not true, because the detailed requirements and design choices aren't yet clearly defined. The RASD in particular is the main reference document, in which it can be found a detailed description of the features the system has to accomplish.

2. Function Point Approach

The Functional Point approach is a technique allowing the evaluation of the effort needed for the design and implementation of a project.
This technique evaluates the application dimension considering the amount and the complexity of the functionalities it has to implement and of the chosen data structures.
The division in function types and the parameters used for counting the function points are the following:

Function Types	Weight		
	simple	medium	complex
N.External Inputs	3	4	6
N.External Outputs	4	5	7
N. External Inquiry	3	4	6
N.Internal Logical File	7	10	15
N.External Interfaces File	5	7	10

2.1 Internal Logic File (ILF): It represents a set of homogeneous data handled by the system typically regarding the database table.
The application includes a number of ILFs that will be used to store the information about users,taxi drivers, requests and reservations.
In detail, the ILFs considered are:

- **Users**: The informations stored related to users are: name, surname, username, password, phone number, email, date of birth and gender. We can still consider this file as simple even if it contains 8 different fields, but each of them is simply stored as a string.
- **Taxi Drivers:** Taxi drivers informations will be stored in a different database table even if the fields will be very similar. There is only an additional information related to taxi drivers, which is the access code given to taxi drivers when they have been registered by employees of the city government. Again, we have a simple ILF for the same reason stated before for users.
- **Taxis:** The informations about taxis, their availability and their current drivers are considered as simple ILF.
- **City Zones:** The zones of the city will be identified by an integer number and a couple of coordinates that identifies univocally each zones. The number of instances stored will be immutable during the future activity of the system, saving for sporadic changes of city zones division. Also the number of zones won’t be high, and therefore this is surely a simple ILF.

- **Reservations:** This ILF can be considered of medium complexity: it will be necessary to store date and hour of the ride, the destination and the starting point, the fee per person and the number of passengers, the way to be contacted for the confirmation of the starting ride, the desire to share the route or not.
- **Requests:** The Request entity is a bit more complex given the fact that it's necessary also to memorize the users who does a shared ride together and additional fields like the information of route completed or yet to be completed and the relations to taxi driver and reservation tables. This is not however a sufficient reason to add 5 FP's making the requests data as a complex ILF.

This means that we will come out with 4 x 7 + 2 x 10 = 48 FPs concerning ILFs.

Internal Logic File	Complexity	FP Assigned
User	Simple	7
Taxi Driver	Simple	7
Taxi	Simple	7
City Zones	Simple	7
Reservation	Medium	10
Request	Medium	10
Total		48

2.2 External Interface File (EIF): It represents a set of homogeneous data used by the application but handled by external application.The system will have to deal with the Google Maps API in order to compute routes. For this interaction we can estimate a complex weight because of the non trivial tasks regarding route planning it has to deal with.

External Interface File	Complexity	FP Assigned
Position	High	10
Total		10

2.3 External Input: these are the functionalities dealing with the input of data external from the system

- **Login/logout**: these are simple operations for both taxi drivers and clients, so we can adopt the simple weight for them. 2 x 3 = 6 FPs
- **Client registration**: this is a simple operation that doesn't require complex controls for the final validation of the operation. We adopt the simple weight. 1 x 3 = 3 FPs
- **Request Creation**: this is an operation of medium complexity because it involves a number of informations relatively high to store in the database plus the add of the identifier into the queue of pending requests. Moreover an operation to control the existence of the address declared as a starting point of a ride should not be trivial. We can adopt a medium weight. 1 x 4 = 4 FPs.
- **Reservation Creation**: this is an operation of medium complexity for the same reason of the request creation. 1x4 = 4 FPs
- **Reservation Deletion**: this is a simple operation because it involves only the reservation entity and a control on the starting time of the future ride to see if it's possible to delete or not the reservation. 1 x 3 = 3 FPs
- **Taxi state change from busy to available**: this is an operation of medium complexity that consists only in the input of a boolean value. The main part of computation is related to the add of the taxi into the right queue, which involves the use of the GPS system and the algorithm for searching the zone corresponding to the coordinate of the taxi position received. 1x4 = 4 FPs
- **Taxi state change from available to busy**: this operation instead is really simple and consists only in the elimination of a taxi instance from a queue. We thought to separate both ways of taxi state changing as different inputs to be coherent with the separation of login and logout inputs. 1x3= 3 FPs.

- **Taxi Driver Request Acceptance**: this operation consists only in a choice between two alternatives but can be seen as a medium complexity operation. It involves the interaction with the Request Entity to update the previously null “Taxi Driver” field and the deletion from one of the queues of active taxis and the add of the taxi into the set of logged in but temporarily inactive taxis. Even if it’s an operation which is immediately followed by an output operation (notification to the clients involved), it’s not considered as an inquiry because there is an interaction with the database that doesn’t consist only in a simple reading of values, and some data structures are modified and not only visualized. 1x4 = 4 FPs.
- **Taxi Driver Request Decline**: this operation is simple and consists only in the moving of a taxi instance from the top of the queue to the bottom of the same queue. 1x3= 3 FPs.

External Input	Complexity	FP Assigned
Login/Logout	Simple	2 x 3
Registration	Simple	3
Request Creation	Medium	4
Reservation Creation	Medium	4
Reservation Deletion	Simple	3
Busy To Available	Medium	4
Available To Busy	Simple	3
Request Acceptance	Medium	4
Request Decline	Simple	3
Total		34

2.4 External Output: These are the functionalities dealing with the output of our system including also the logic allowing the system to compute the expected output.
The system will have to implement the following features for an interaction with users:

- **Request Acceptance Notification:** the confirmation of the request accepted by a taxi driver to a user through an SMS or an Email can be considered as an operation of medium complexity. This FP includes also the notification of the add of a new passenger when a request becomes shared. 1x5 = 5 FPs.
- **Taxi Driver Request/Reservation Proposal**: this is a crucial and complex task to implement in our system, especially for what regards the shared ride computation. The process that starts from the pick of a new request from the Request Collector component queue to the proposal of a request to a taxi driver deals with many application entities and four different components, and for this reason an high complexity will be assigned. 1x7 = 7 FPs.
- **Taxi Driver Shared Ride Proposal**: this is similar to the request proposal, but we have also to keep into account the interaction with the Shared Ride Manager component. It also seems to us that duplicating these last two function points can also adjust the scores given that only 7 points assigned to the main functionality of the system wasn't fair comparing the weight to the complete result of other more simple things presented in this document. 1x7 = 7 FPs.
- **Registration Confirmation:** this is a simple operation that starts right after the registration informations are saved in the database. 1x4 =4 FPs
- **Reservation/Request Confirmation**: this is the confirmation of the successful add of the just compiled requests by clients. This FP includes also the notification of the impossible add of the sixth reservation of the same client. 1x4 = 4 FPs

External Output	Complexity	FP Assigned
Request Acceptance by Taxi	Medium	5
Request/Reservation proposal	High	7
Shared Ride Proposal	High	7
Registration Confirmation	Simple	4
Reservation/Request Confirmation	Simple	4
Total		27

2.5 External Inquiries:
The application allows users to visualize informations about:

- **Profile informations and past rides:** We can adopt the medium weight given the different types of informations to retrieve from the database. 1 x 4 = 4 FPs
- **Future Reservations**: this operation is simple as only a simple query on the Reservation table is needed. Moreover the number of instances cannot be greater than a limited number (in the RASD we have limited to 5 the number of possible reservations per user) 1 x 3 = 3 FPs
- **Map Informations** : this operation, using the Google Maps API, allows the user to perform interactive operations with the map of the city: medium complexity is assigned.

External Inquiries	Complexity	FP Assigned
Profile Informations	Medium	4
Future Reservations	Simple	3
Map	Medium	4
Total		11

2.6 Resuming

Function Type	Functional Point
External Inputs	34
External Outputs	27
External Inquiries	11
Internal Logical File	48
External Interfaces Files	10
Total	130

3. Cocomo Approach

This algorithmic method uses the result of the function points calculated before for the estimation of the lines of code of the future software. This methods gives us important results like the estimated time of development and the number of people needed to complete the project in the estimated period.

3.1 Effort estimation

The estimated number of lines of code is:
LOC = 130 * 49 = 6370
We have used relative data for J2EE language and median complexity of the table of [1]. (see External links).

3.2 Scale drivers

This table is taken from link [2], at page 18 of that file, and presents the parameters of the COCOMO II scale factors.

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

Scale factor SF(i)	very low	low	nominal	high	very high	extra high
PREC	thoroughly unprecedented 6.20	largely unprecedented 4.96	somewhat unprecedented 3.72	generally familiar 2.48	largely familiar 1.24	thoroughly familiar 0.0
FLEX	rigorous 5.07	occasional relaxation 4.05	some relaxation 3.04	general conformity 2.03	some conformity 1.01	general goals 0.0
RESL	little (20%) 7.07	some (40%) 5.65	often (60%) 4.24	generally (75%) 2.83	mostly (90%) 1.41	full (100%) 0.0
TEAM	very difficult interactions 5.48	some difficult interactions 4.38	basically cooperative interactions 3.29	largely cooperative 2.19	highly cooperative 1.10	seamless interactions 0.0
PMAT	SW-CMM Level 1 Lower 7.80	SW-CMM Level 1 Upper 6.24	SW-CMM Level 2 4.68	SW-CMM Level 3 3.12	SW-CMM Level 4 1.56	SW-CMM Level 5 0.0

We have chosen the following values for each parameter:

• **PREC = 4.96, low**

Precedentedness: it reflects the previous experience that we had with this kind of projects.

While we have experience in the analysis and design of software components, we lack experience in J2EE platform because we have never programmed in that environment before, so the precedentedness value should be low.

• **FLEX = 2.03, high**

Development flexibility: it reflects the degree of flexibility in the development process. We have freedom in choosing the way to implement the shared ride functionality, in the choice of architecture to use and in the choice of domain assumptions. The value is not very high because we have some time constraints for the development of each part of the project.

• **RESL = 2.83, high**

Risk resolution: it reflects the extent of risk analysis carried out. In this document we present some possible risks, but it’s probable that we have forgotten something considering also the impossibility to prevent every possible unexpected problem.

• **TEAM = 1.10, very high**

Team cohesion: it reflects how well the development team know each other and work together.

There aren’t personal problems among the three people who works at this project. We also have more or less the same free working hours on our university course timetable, therefore we can choose the very high value. The value it’s not extra high because we cannot work together at home in the same place, even if we are able to contact each other with technological devices.

• **PMAT = 4.68, nominal**

Process maturity: it reflects the process maturity of the organisation. This was evaluated around the 18 Key Process Area (KPAs). We tried to answer to the questionnaire in the SEI Capability Maturity Model (page 21 of link [2]) and the result was 2.5. We have chosen to underestimate the value rounding it to 2, which corresponds to nominal.

• **Sum of all the Scale Factors = 15,6**

This value will be used later in the effort equation (section 3.4).

3.3 Cost Drivers

These data are taken again from link [2], starting from page 25 of that file, which contains the detailed explanation for every value of cost drivers. It would occupy too much space to copy all the tables into this document and it would detract the reader from its main content.

Cost Driver	Factor	Value
Required Software Reliability	low	0.92
Database Size	nominal	1.00
Product Complexity	high	1.17
Required Reusability	high	1.07
Documentation match to life-cycle needs	nominal	1.00
Execution Time Constraint	nominal	1.00
Main Storage Constraint	nominal	1.00
Platform Volatility	low	0.87
Analyst Capability	nominal	1.00
Programmer Capability	nominal	1.00
Application Experiences	low	1.10
Platform Experiences	very low	1.19
Language and Tool Experiences	low	1.09
Personnel continuity	very high	0.81
Usage of Software Tools	nominal	1.00
Multisite Development	very high	0.86
Required Development Schedule	high	1.00
Product		0.996

Some explanations about the values chosen above:

- We have chosen the low value for **Required Reliability** because if the software contains bugs, the damages to people are easily recoverable. Human lives aren't in trouble if a taxi request isn't correctly processed or other bugs occurs as it would happen for example in software for airplanes. Moreover the application won't deal with money transactions. However too many failures can lead to complaints by the citizens, an acceptable level of software quality is certainly needed because the investment of the government otherwise would fail soon. A nominal value could also fit in this case.
- **Personnel continuity** is very high because the people who will work at this project won't change until the end.

- **Platform Experience** is very low because we have never used Java EE platform, while the **Application Experience** is only low because we have at least designed a similar problem last year, though never programmed it.
- **Language Experience** is low because we have programmed in Java at least one software of medium complexity.

3.4 Effort equation

This final equation gives us the effort estimation measured in Person-Months (PM)
Effort := A * EAF * $KSLOC^E$
Where:

- A → 2.94 is a fixed parameter of COCOMO II
- EAF → Effort Adjustment Factor, product of all the cost drivers, equal to 0.996 in our case;
- E → exponent derived from Scale Drivers, calculated as:
  B + 0.01 * sum{i} SF[i] := B + 0.01 * 15.6 = 0.91 + 0.156 = 1.066,
  where the parameter B is equal to 0.91 for COCOMO II.
- KSLOC → estimated lines of code using the FP analysis, already calculated in section 3.1 of this document = 6.370

With this parameters we can compute the Effort value, that is equal to:
Effort := 2.94 *0.996 * $(6.370)^{1.066}$ ≈ 21 PM (person-months)

Now we can use the Effort result for the estimation of the application development duration:
Duration := 3.67 * $(Effort)^F$, where:

- F := 0.28 + 0.2 * (E – B) = 0.3112
- 0.28, 0.2 and 3.67 are constants used for the calculation of the duration.

It follows:
Duration := 3.67 * $21^{0.3112}$ = 9.46 ≈ 10 Months

We can now calculate the required number of people which is simply the division between effort and duration:
P = effort / duration = 21 / 10 = 2.1 ≈ 3 People.

3.5 Final Considerations

We prefer to overestimate the number of people required, at least the project will finish before the deadlines and it would be a positive thing.
It seems that the results of Cocomo are realistic, even if we haven't got a sufficient experience in this kind of projects, or many examples that would let us be able to compare the duration and the estimated efforts of similar projects.

4. Risks for the project

The type of risks which people involved in the project have to think to, detect and find a way to resolve them in case they will become real, fall under the following three categories: business risks, technical risks and project risks.

4.1 Business risks

They are the most relevant risks and the ones which are hard or impossible to prevent even if they are known risks.

- **Budget risk**: during the development time who is supposed to take 8 months according to the COCOMO model previously done, the government can change idea about the realisation of the project because of unexpected lack of money.

  This problem is critical. If it happens only part of the money will be received by the development team (the one already paid by the government) and part of the effort put in the realisation of the project could not be rewarded.

  A plan to react to this situation could be the initial acceptance by project manager to lower his income to a still acceptable quantity, or break the partnership to that city government and propose the same service to other cities now that the software will be partially realised.
- **Strategic risk:** during the development time suddenly the innovation of the city's taxi service can become no more worth the money to maintain it in the future.

The problem is critical and could be again resolved by negotiating with the government city.

- **Management risk**: in case of retards in the first tasks deadlines of the project and in general bad management of the project, the government can decide to assign the same task to another team accepting to lose an amount of money already invested or propose only a person change in the development team.

  Like every business risk, it's again a critical issue. The project manager has to continuously supervise the team work, behave loyally with the government and notify them about important inconveniences or retards in the fixed schedule.

For what concerns out project the **Market risk** is absent: in our case the work has been already committed.

4.2 Technical risks

- **Requirement changes:** It may happen that, due to a miscommunication, several functionalities reported in the RASD document don't reflect what is really asked by our customer. To avoid the risk of implementing wrong functionalities it's important to ask further informations and details about what the system has to deal with. If miscommunication happens the developers will write useless code: this situation has to be avoided because it's really expensive in effort terms.
- **Bad Design:** Another technical risk to take into account is bounded to design choices: if the project manager proposes a solution with a bad coupling between components and ignoring useful design patterns, the code developers will write a more fragile, bug oriented application. This could also lead into some further difficulties in terms of system performances. It's important that the project manager, when writing the Design Document with all the other team members, keeps in mind all the useful details in order to prevent this kind of risk.

- **Development environment:** It may happen that some difficulties are met with regard to the tools used by the developers. This kind of risk may lead into a project delay. We already had taken into account this problem in the COCOMO section but maybe the difficulties to adapt to a new development platform could postpone the project deadlines.

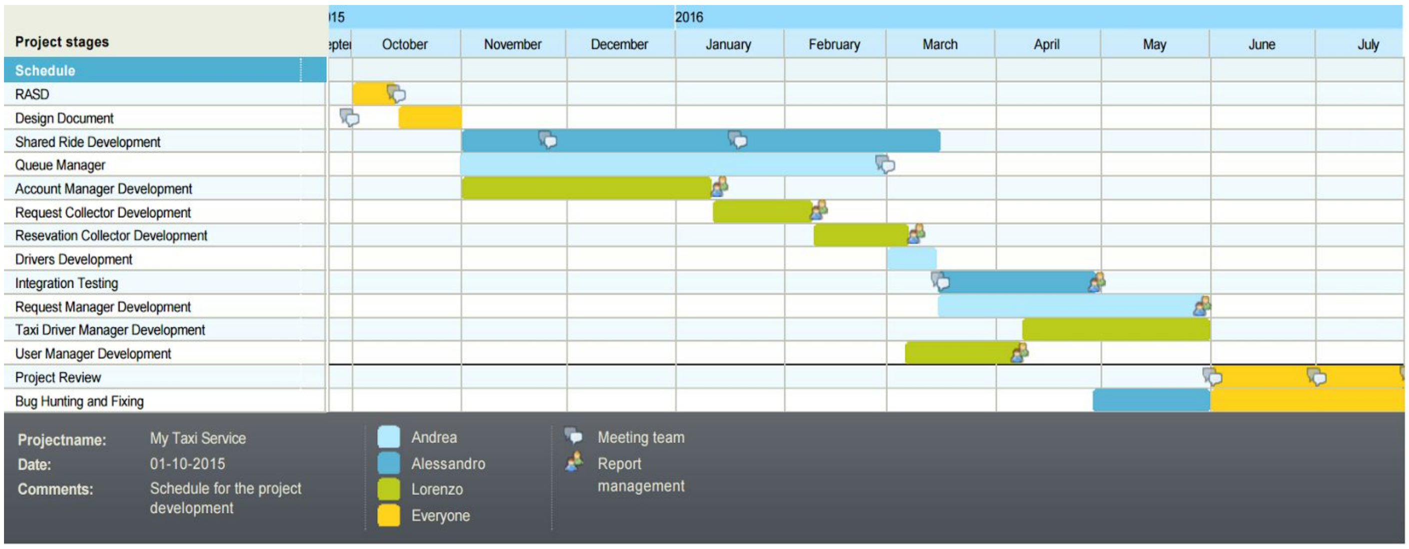
4.3 Project Risks

- **Personnel shortfall**: Our team members could be too few for the system we are planning to develop. We have already overestimated the people required to the project with the COCOMO approach but it is not to exclude that some more people are needed to the project.

  Recovery action: before starting the implementation we could add one or more member to our team. This shouldn't be done after the end of design decisions and the start of development, because adding more people to an already started project let the overall completion time grow. The design document forces the team members to reason on software details and can show that the software effort estimation done in the first project phase is an underestimation.
- **Unrealistic schedule**: The function point and Cocomo approaches could be misleading and underestimate the development effort. The recovery action consists into the notification to the government of the problem and into the addition of another member to the project, but only if the problem is detected at the very first stages of the project.
- **Personnel abandoning**: it's possible that some people can abandon the project because of various problems (health, lack of motivation, important argues between members, etc..). This problem can be mitigated assigning tasks to resources thinking also at this inconvenience.

5 Tasks Definition and Schedule

The following tasks are defined coherently with the Integration Testing Strategy section of the Test Plan Document; most of the project schedule was already defined in that part of the document.
Here we present a more detailed scheduling that takes into account not only the division of activities and their dependencies, but also the exact person who is going to do the activity, the time allocated to every task in the span of 10 months (estimated duration of the project according to COCOMO) starting from October 2015, the daily hours in which a person can work to the project and the allocated time for all the project tasks, not only considering the development. We have taken into account for the project scheduling the components complexity and their relation.



6. External links

[1]: http://www.qsm.com/resources/function-point-languages-table#MoreInfo
[2]: http://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_modelman2000.0.pdf