Politecnico di Milano

A.A. 2015-2016

Software Engineering 2 Project: myTaxiService

Project Plan

Damiano Binaghi, Giovanni Zuretti

Contents

1 Introduction

In this document we are going to describe the project plan for the myTaxiService Application. This planning is done afterwards some activities and tasks we will identify, for example the requirement analysis and the design process. However we think it could still be useful for identifying the costs associated to each step of the development of the software, the tasks that still need to be accomplished, how to allocate resources to such tasks, and what are the main risks and critical issues associated to the project itself.

2 Function Points Approach

2.1 Introduction to function points analysis

Function Points approach is a technique for evaluating the effort needed for the development of software applications. The technique is based on assigning points to each functionality with respect of the complexity of the functionality itself. What is obtained in the end is an estimation of the effort needed to implement the application and the expected number of code lines. We will evaluate function points for all the functionalities we have identified in the RASD document. Such functionalities, according to the function points technique will be grouped in the following categories:

- Internal Logic Files (ILF), which represents the data managed inside the application, and for our application they will be the data stored in the Database.
- External Interface Files (EIF), which represent the data used by the application but managed by external application. In our case there are not such type of data.
- External Input, which will collect all those functionalities that allows external resources (such as users) to input data into the system.
- External Output, which will collect all those functionalities that provides an output from the system towards resources outside of the application.
- External Inquiry, which will collect all those functionalities involving both input and output operations.

For the calculation of the function point we will refer the following table, which assign a different cost based on the complexity of the functionality we are evaluating.

Function Types	Weight		
	Simple	Medium	Complex
N. Inputs	3	4	6
N. Outputs	4	5	7
N. Inquiry	3	4	6
N. ILF	7	10	15
N. EIF	5	7	10

2.2 Identification of Functionalities & Function Points Calculation

2.2.1 Internal Logic Files

The application include a number of Logic Files used to store information about the Customers, the Taxi Drivers, other type of Users (Taxi Data Managers and the system administrator), and the Booking Requests. For information about users we will adopt a simple weight, because all such entities will have a rather simple structure. While, for the booking requests, which involves the use of triggers, we will adopt a medium weight. The final estimation for the Internal Logic Files will be (3 x 7) + (1 x 10) = 31.

2.2.2 External Inputs

We have identified the following input (we have put between parenthesis the complexity of each input):

- Sign up (simple)
- Login (simple)
- Logout (simple)
- Edit Personal Data (simple)
- Insert new Taxi Driver (simple)
- Instant Booking Request (simple)
- Advance Booking Request (medium) We assigned medium weight to this operation because it is uses two entities one of which (the Advance Booking Entity) is of medium complexity
- Delete a request (simple)
- Send Position (simple)

- Confirm/Refuse Assignation (simple)
- Report Fake (simple)
- Report Accident (simple)
- Verification Code Insertion (simple)
- Start Ride (simple)
- End Ride (simple)
- API Authentication (simple)
- Instant Booking Request From API (simple)
- Advance Booking Request From API (medium) We assigned medium weight to this operation because it is uses two entities one of which (the Advance Booking Entity) is of medium complexity

The calculation resolves in (16x 3) + (2 x 4) = 56.

2.2.3 External Outputs

We have identified the following outputs:

- When a new request must be handle, the system should assign a driver to the request notifying him of the assignation. This is a simple operation, because the system must simply send a message to a taxi driver.
- When a driver is assigned successfully to a request, the system should notify the waiting customer that his request has been handled communicating the waiting time for the taxi to arrive. This must done both for the normal system and the API service offered by the system itself. This is a complex operation, because it requires the system to interact with the Google Maps API Service in order to retrieve the waiting time that must be sent to the customer, and it may also require the system to generate more than one assignation notification, if many drivers refuse the assignation.

The computation is (1 x 4) + (2 x 7) = 18.

2.2.4 External Inquiries

We have identified the following inquiries:

- Show booking history to customer (simple)
- Show fake reports to admin (simple)
- Show accidents report to admin (simple)

The calculation is: 3 x 3 = 9

2.2.5 Final Calculation and Lines Of Code Estimation

Here is the total sum of the Function Points:

ILF	31
External Inputs	56
External Outputs	18
External Inquiries	9
Total	**114**

We will now proceed in the estimation of the Lines Of Code (LOC) for our application based on the function points previously calculated. This is going to be a rough estimation of the real lines of code that will be needed. In particular it is evident how the core of the logic of our application, which is the management of the queue, it is not consider in the function points estimation because it does not interact with any entity. However it will surely require many lines of code to be correctly implemented, but these lines are not included in the estimation. Another thing that is left out in the calculation of the function point it is the effort needed to implement API Service. The functionalities concerning the API services are treated like other functionalities for complexity in respect to the entities needed to provide them, but they will surely require more thinking and effort at implementation time, because implementing an API service is not trivial.
The formula for calculating LOC makes use of a constant that depends on the language used for implementing the application. Our application is going to be implemented in JEE, thus, the constant value will be equal to 46, which we found to be used for other JEE applications. Therefore we have:

$$\mathbf{LOC = 114 \text{ x } 46 = 5244}$$

We will start from this estimation for calculating the effort and cost estimation with the COCOMO II algorithm.

3 Effort Estimation: COCOMO II

This estimation use a complex formula which take in accounts many aspects of the development process (Scale and Cost Drivers). For each of them we are going to give a brief description of what they are and their impact on our project.

3.1 Scale Drivers

According to the COCOMO II documentation, we have the following scale drivers:

- Precedentedness: This driver takes in account previous experience in developing similar projects. We did not do the implementation phase, however it would have been the first experience for us in implementing an application using JEE. Therefore this value will be low.
- Development Flexibility: This driver takes in account the flexibility of the development process. Our assignment contained general specification about the application, leaving to us the definition of the details, therefore the value for this parameter will be high.
- Risk Resolution: We will set this value to nominal. We have thought of some of the risks and critical aspects of the application, but we think more can be done.
- Team Cohesion: Though we did not implement the application, we are confident to set this parameter as very high, because of the other project assignment we carried out successfully. The work was done almost always together, and when we were not working together we accomplished to merge the work done in an efficient and fast way.
- Process Maturity: It is the first project of this kind we face and we did not carry out the implementation phase, thus this value will be set as low.

We represent with a table the over scale driver factor.

Scale Driver	***Factor***	***Value***
Precedentedness	Low	4.96
Development Flexibility	High	2.03
Risk Resolution	Nominal	4.24
Team Cohesion	Very High	2.19
Process Maturity	Low	6.24
Total		19.66

The figures were obtained from the following table, which is the standard for COCOMO II.

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

Scale Factors	Very Low	Low	Nominal	High	Very High	Extra High
PREC	thoroughly unprecedented	largely unprecedented	somewhat unprecedented	generally familiar	largely familiar	thoroughly familiar
SF_i:	6.20	4.96	3.72	2.48	1.24	0.00
FLEX	rigorous	occasional relaxation	some relaxation	general conformity	some conformity	general goals
SF_i:	5.07	4.05	3.04	2.03	1.01	0.00
RESL	little (20%)	some (40%)	often (60%)	generally (75%)	mostly (90%)	full (100%)
SF_i:	7.07	5.65	4.24	2.83	1.41	0.00
TEAM	very difficult interactions	some difficult interactions	basically cooperative interactions	largely cooperative	highly cooperative	seamless interactions
SF_i:	5.48	4.38	3.29	2.19	1.10	0.00
PMAT	The estimated Equivalent Process Maturity Level (EPML) or					
	SW-CMM Level 1 Lower	SW-CMM Level 1 Upper	SW-CMM Level 2	SW-CMM Level 3	SW-CMM Level 4	SW-CMM Level 5
SF_i:	7.80	6.24	4.68	3.12	1.56	0.00

3.2 Cost Drivers

The Cost Drivers are the following:

- Required Software Reliability: If we consider our software to become a standard for the booking of taxis for the city for which it was implemented, then a failure of the system would paralyze the taxi service for the whole city, i.e. customers could not book their taxis in any other ways and taxi drivers could not do their work. In such case we should consider this driver as high. However it could happen that the service of booking taxis by phone call is maintained in parallel with our application. In such case, a failure in our application would cause only the dissatisfaction of the customer, and the value of this driver would be low. Due to the fact, that this is a project for the Software Engeeniring 2 course and, thus, we do not know for which city this application will be implemented and which one of the previous described policies will be chosen, we decided for the worst case scenario and we will adopt a high value for this driver.

Table 17. RELY Cost Driver

RELY Descriptors:	slight inconven-ience	low, easily recoverable losses	moderate, easily recoverable losses	high financial loss	risk to human life	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	0.82	0.92	1.00	1.10	1.26	n/a

- Data Base Size: We did not implement the application, therefore we have no test database. However, based on the fact that we have few entities and, as stated in section 2.2.5, the fact that we will have many lines of code that will be dedicated to queue management and API service (which

are going to increase the actual LOC of the software), we decided to give a low weight to this driver.

Table 18. DATA Cost Driver

DATA* Descriptors		Testing DB bytes/Pgm SLOC < 10	10 ≤ D/P < 100	100 ≤ D/P < 1000	D/P ≥ 1000	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	n/a	0.90	1.00	1.14	1.28	n/a

- Product Complexity: We will set this driver to high, due to the following reasons.
  - Control operations have a high value do to the management of the queues.
  - Computational operations have a nominal value because the application we will use only standard mathematical routines (e.g for calculating in which zone the position of the taxi belongs)
  - Device-dependent operations have a high value because we require that the taxi driver device keep sending its position to the system.
  - Data-Management operations have a high value, because of the use of simple triggers.
  - User Interface Management Operations have a nominal value, because they are going to be standard interface operations.
  - The total average leans towards a high value, thus we decided to adopt a high value for the overall driver.

Table 20. CPLX Cost Driver

Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	0.73	0.87	1.00	1.17	1.34	1.74

- Developed for Reusability: Since we did not implemented the application, and however we think that future development of additional functionalities may require some reusability, we decided to set this value as nominal.

Table 21. RUSE Cost Driver

RUSE Descriptors:		none	across project	across program	across product line	across multiple product lines
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	n/a	0.95	1.00	1.07	1.15	1.24

- Documentation Match to Life-Cycle Needs: Since all the requirements and aspects of our system were described in the RASD and DD and we do not think that such documentation is excessive we will set this value

to nominal.

Table 22. DOCU Cost Driver

DOCU Descriptors:	Many life-cycle needs uncovered	Some life-cycle needs uncovered.	Right-sized to life-cycle needs	Excessive for life-cycle needs	Very excessive for life-cycle needs	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	0.81	0.91	1.00	1.11	1.23	n/a

- Execution Time Constraint: We do not have this particular constraint on our application, so its value will be set to low.

Table 23. TIME Cost Driver

TIME Descriptors:			≤ 50% use of available execution time	70% use of available execution time	85% use of available execution time	95% use of available execution time
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	n/a	n/a	1.00	1.11	1.29	1.63

- Main Storage Constraint: We do not have this particular constraint on our application, so its value will be set to low.

Table 24. STOR Cost Driver

STOR Descriptors:			≤ 50% use of available storage	70% use of available storage	85% use of available storage	95% use of available storage
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	n/a	n/a	1.00	1.05	1.17	1.46

- Platform Volatility: We can consider this driver as low, since we do not think about making often changes to our platform, such as the DBMS. However it might happen, that is why we think setting this driver to low would be the best choice.

Table 25. PVOL Cost Driver

PVOL Descriptors:		Major change every 12 mo.; Minor change every 1 mo.	Major: 6 mo.; Minor: 2 wk.	Major: 2 mo.;Minor: 1 wk.	Major: 2 wk.;Minor: 2 days	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	n/a	0.87	1.00	1.15	1.30	n/a

- Analyst Capability: We should set this driver to high, since we dedicate a lot of effort to the analysis of the requirements and the design, going through a deep thinking process before actually making decisions.

Table 26. ACAP Cost Driver

ACAP Descriptors:	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.42	1.19	1.00	0.85	0.71	n/a

- Programmer Capability: We did not implement the application, however we are positive in setting this parameter to high.

Table 27. PCAP Cost Driver

PCAP Descriptors	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.34	1.15	1.00	0.88	0.76	n/a

- Personnel Continuity: Since there was no turnover, this parameter will be set to extra high.

Table 28. PCON Cost Driver

PCON Descriptors:	48% / year	24% / year	12% / year	6% / year	3% / year	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.29	1.12	1.00	0.90	0.81	

- Applications Experience: We have already implemented applications in Java, but it would be the first time for us in interfacing with the JEE framework. For such reason this driver will be set to low.

Table 29. APEX Cost Driver

APEX Descriptors:	≤ 2 months	6 months	1 year	3 years	6 years	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.22	1.10	1.00	0.88	0.81	n/a

- Platform Experience: We will set this value to nominal, because we think we have a good-enough knowledge of the platforms that are going to be used (database, user interface, sever).

Table 30. PLEX Cost Driver

PLEX Descriptors:	≤ 2 months	6 months	1 year	3 years	6 year	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.19	1.09	1.00	0.91	0.85	n/a

- Language and Tool Experience: We will set this value to nominal, because it is the first time that we use some of the tools.

Table 31. LTEX Cost Driver

LTEX Descriptors:	≤ 2 months	6 months	1 year	3 years	6 year	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.20	1.09	1.00	0.91	0.84	

- Usage of Software Tools: We are planning to use either NetBeans or Eclipse for the implementation, which we consider basic tools and moderately integrated. Therefore this driver is going to be nominal.

Table 32. TOOL Cost Driver

TOOL Descriptors	edit, code, debug	simple, frontend, backend CASE, little integration	basic life-cycle tools, moderately integrated	strong, mature life-cycle tools, moderately integrated	strong, mature, proactive life-cycle tools, well integrated with processes, methods, reuse	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.17	1.09	1.00	0.90	0.78	n/a

- Multisite Development: This value is going to be extra high, since the work was always done in a fully collocated way.

Table 33. SITE Cost Driver

SITE: Collocation Descriptors:	Inter-national	Multi-city and Multi-company	Multi-city or Multi-company	Same city or metro. area	Same building or complex	Fully collocated
SITE: Communications Descriptors:	Some phone, mail	Individual phone, FAX	Narrow band email	Wideband electronic communicat ion.	Wideband elect. comm., occasional video conf.	Interactive multimedia
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.22	1.09	1.00	0.93	0.86	0.80

- Required Development Schedule: Since we did not carry out the implementation phase, and thus we have a partial vision of the actual effort, related to the scheduled time needed, to carry out the entire project, this value will be set to nominal.

Table 34. SCED Cost Driver

SCED Descriptors	75% of nominal	85% of nominal	100% of nominal	130% of nominal	160% of nominal	
Rating Level	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multiplier	1.43	1.14	1.00	1.00	1.00	n/a

We can now defining the following table to find the impact of the cost drivers.

Cost Drivers	***Factor***	***Value***
Required Software Reliability	High	1.10
Data Base Size	Low	0.9
Product Complexity	High	1.17
Developed for Reusability	Nominal	1.00
Documentation Match to Life-Cycle needs	Nominal	1.00
Execution Time Constraint	Low	n/a
Main Storage Constraint	Low	n/a
P(latform Volatility	Low	0.87
Analyst Capability	High	0.85
Programmer Capability	High	0.88
Personnel Continuity	Extra High	n/a
Application Experience	Low	1.10
Platform Experience	Nominal	1.00
Language and Tool Experience	Nominal	1.00
Usage of Software Tools	Very High	1.00
Multisite Development	Extra High	0.8
Required Development Schedule	Nominal	1.00
Product		0.66

3.3 Effort Estimation

The formula for calculating the effort estimation is

$Effort = A * EAF * KSLOC^E$

Where:

- A = 2.94 (Constant provided by COCOMO II)
- EAF = CostDriverFactor = 0.66
- KSLOC= 5.244
- E = 0.91 + ScaleDriverFactor * 0.01 = 0.91 + 19.66 * 0.01 = 1.1066 (again 0.91 is a constant defined in COCOMO II)

Therefore, we have

$Effort = 2.94 * 0.66 * 5.244^{1.1066} = 12.14\ Person/Month$

From the effort we can estimate the duration of the project using the following formula

$Duration = 3.67 * Effort^F$

Where

$F = 0.28 + 0.2 * (E - 0.91) = 0.31932$

What we have is

$Duration = 3.67 * 12.14^{0.31932} = 8,14 \sim 8\ Months$

This will allow us to obtain the number of people needed for the whole project:

$NumberOfPeople = Effort/Duration = 12.14/8.14 = 1.49 \sim 2\ People$

3.4 Conclusions

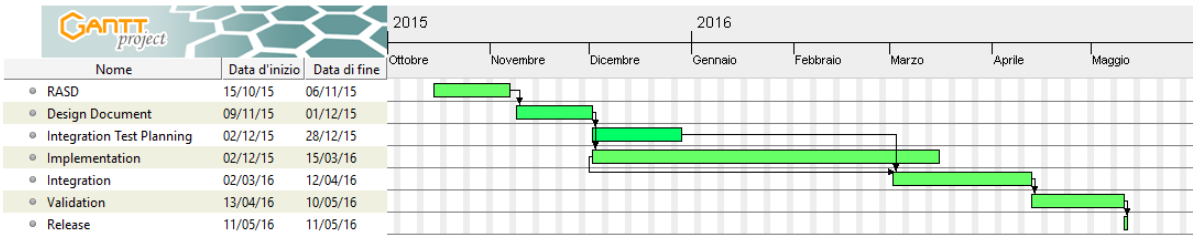
We think our estimations about effort and duration may be oversize, but we were surely biased by the fact that we did not implemented the application and that it is the first time we are involved in a project of this type. Therefore, we can only make estimates about the effort that implementation will require and this may have caused a non-appropriate estimation of some of the drivers.

4 Tasks Identification and Scheduling

We are now going to identify all the tasks needed for the development of the application and how much time each of them will require. We made an estimation, based on the deadlines we received during our project.
The tasks we have identified are:

- Requirement Analysis and Specification Document
- Design Document
- Integration Test Planning
- Implementation
- Integration
- Validation
- Release

The Gantt chart below shows how we intend to schedule these tasks, given the duration estimated in the previous section (i.e. 8 months).



Nome	Data d'inizio	Data di fine
RASD	15/10/15	06/11/15
Design Document	09/11/15	01/12/15
Integration Test Planning	02/12/15	28/12/15
Implementation	02/12/15	15/03/16
Integration	02/03/16	12/04/16
Validation	13/04/16	10/05/16
Release	11/05/16	11/05/16

We decided that Integration Test Planning could have been carried out in parallel with the initial Implementation phase and that the Integration phase could start before the implementation phase is actually fully completed, because for sure some of the component that must be integrated will be finished for that time. Therefore integration can start a little before implementation is totally completed.

5 Resource Allocation

For each of the previously defined tasks, we are now going to allocate each member of the group to what he needs to do in order to stick to the project deadlines. We want to underline that for the RASD, DD and Integration test planning document we are going to allocate resources based on our personal experience, because we actually have written those documents. For the other parts we are going to give a general division in terms of hours, which we noticed have been even for both members, for all the parts of the project. Therefore we will use this method (which is actually very rough) to allocate resources to tasks of which we have no experience.

5.1 Resource Allocation for RASD

We divided the internal tasks as follows:

- Giovanni Zuretti: Identification of the goals, identification of the domain properties, identification of the requirements, UML and Alloy modeling (the thinking part) and redirection of the document.
- Damiano Binaghi: Identification of the goals, identification of the domain properties, identification of the requirements, UML and Alloy modeling (both the thinking and the making parts)

5.2 Resource Allocation for the Design Document

We divided the internal tasks as follows:

- Giovanni Zuretti: Definition of the architecture of the system, high level and low level component view, requirement mapping, redirection of the document.
- Damiano Binaghi: Definition of the architecture of the system, runtime view, algorithm design, user interface design.

5.3 Resource Allocation for the Integration Test Plan Document

We divided the internal tasks as follows:

- Giovanni Zuretti: Definition of the subsystems and the integration strategy, definition of the sequence of integration, definition of the test cases, redirection of the document.
- Damiano Binaghi: Definition of the subsystems and the integration strategy, definition of the test cases, definition of the testing procedures.

5.4 Resource allocation for Implementation

As already said many times, we did not carry out the implementation. So, in this section, we proceed by estimating the number of hours that will be necessary for the implementation phase. This phase will start in parallel with the Integration Test Planning phase, therefore, for the initial part we will take in account half of the work hours for each day. We will do an estimation of 6 hours of work per day for each member, the Saturdays and the Sundays are not considered as work days, and the 24th and 25th of December are not considered work days. Furthermore the final phase of the implementation will be done only by one of the two members of the group, because the other will start the integration test phase. What we have, thus, is the following:

- Giovanni Zuretti (who will take car of starting the integration testing): (3 x 18) + (6 x 45) = 324 hours
- Damiano Binaghi: (3 x 18) + (6 x 56) = 390 hours

5.5 Resource allocation for Integration Testing

As said in the previous subsection, the member Giovanni Zuretti will start this phase before the other member, Damiano Binaghi. Therefore, what we have is:

- Giovanni Zuretti: 6 x 31 = 186 hours
- Damiano Binaghi: 6x20 = 120 hours

5.6 Resource allocation for Validation

Finally, for validation, the hours will be equally divided between the 2 member:

- Giovanni Zuretti: 6 x 20 = 120 hours
- Damiano Binaghi: 6 x 20 = 120 hours

6 Risks and Recovery Actions

A risk is a potential problem that we might face in developing the application. According to the risk categorization we have three types of risks:

- Project risks, which threaten the project plan.
- Technical risks, which threaten the quality and the timeliness of the software produce.
- Business risks, which threaten the viability of the software to be built.

For sure we have identified some possible business risk, for example what are called "the market risk" and "the strategic risk". For the former, the risk is that the service our application offers is not perceived as a quality-improvement by the customer and the taxi driver, making them not willing of changing their usual habits and start using our application. To try to mitigate such risk, a proactive approach should be used, for example, by doing a properly advertising of the application and by starting a process of "evangelization" towards the taxi drivers before the actual release of the application. In such way, taxi drivers will already know what is the application about, how it works and how it improves their working before the application is released. Once released, they would have no problems in using it. For the second, we think that the government which will ask us to do such application should be convinced from the very beginning to support the development until the end. If the premises to the development of the application are not that convincing, i.e. the government is not convinced in investing all the money needed to develop such application, or the government might change when development is started and the new government would cancel the budget promised for the application, all the project might fail. We think that to avoid such risks, those who make the budget plan should be convinced of their commitment from the very beginning, however this relies a lot on the honesty of the people involved in the project.
We think also that there might be some technical and project risk in the staff size and experience. Indeed, for us, is the first project of this type. Therefore this could threaten the quality of the software developed, as well the time needed to develop the project. To resolve such limits, we think that in a real business environment, we should probably be helped from someone who has already experience in developing this types of applications.