Software Engineering 2

Project Plan Documentation

Authors:

Yuqi Liu

Xiuli Xu

Summary

1. Introduction

In this document a project plan for the myTaxiService application will be presented. The main chapters of the document will contain an analysis of the project size with the use of Function Points and an estimation of effort and cost using COCOMO. In the next chapters the project will be divided in tasks which will be scheduled and all the available resources will be assigned to the different tasks. Finally the risk for the project will be described and discussed.

2. Function Point Approach

The Function Point is a technique used "to assess the effort needed to design and develop custom software applications" (A. Albrecht). As such it uses different characteristics of the program, each one properly weighted, to evaluate myTaxiService' dimension as a project. The above mentioned weight are reported in the following table, with regards to the Function type and its complexity.

Function Type	**Complexity**		
	Simple	Medium	High
Internal Logical File	7	10	15
External Interface File	5	7	10
External Input	3	4	6
External Output	4	5	7
External Inquiry	3	4	6

Now an estimation of the dimension of the application can be done by summing all the number of Function Type multiplied by its weight, according to the following equation:

In order to calculate this sum, every single Function Type must be analysed in order to determine its number and the weight to be used.

1. Internal Logic Files (ILFs)

This type represents a homogeneous set of data used and managed by the application. As such it is all the data stored in the DB and to which the application has access at will. In myTaxiService the data stored in the DB will be: information about registered users and taxi drivers, the city zones, and the zones' queues. All these elements can be considered simple since they are made of a small number of fields (for example User name, password, Name, Surname and phone number for registered users) except for the zone queues, which have a higher complexity than all other ILFs. With these assumption the FPs for the internal logic files are:

2. External Logic Files (ELFs)

These are the files containing data that are used by the application but generated and maintained by other applications. In our case there is only one interaction of this type, the one with the GPS API used by the taxi drivers when they upload their location. According to this, we can consider a simple weight for this logic file, so the result will be: FPs.

3. External Inputs

These are all the elementary operations that elaborate on data coming from an external environment, so in the case of myTaxiService, these external inputs can be identified:

- *Log in/Log out and Registration*: These operations will have a simple weight since they only require a few operation on the DB. They will have a cost of: FPs.
- *Upload Location:* This operation done by the taxi driver should be considered of high complexity, since it has to interact with the GPS, DB, queues and zones. FPs.
- *Request a taxi:* This operation will interact with different entities of the application and trigger different functionalities of the application, which will interact with queues, zones, taxi drivers, registered users and will require the system to dedicate resources in handling the future communication between user and driver. Because of this, it will have a complex weight: FPs.
- *Accept/decline Taxi:* This information received by the system contains the user response to a proposed taxi. It only involves the registered user and the system, so it has a simple weight: FPs.
- *Accept/decline Customer:* This is the response of the driver to a proposed customer, and as the previous point, it has a simple weight: FPs.

4. External Output

These are the elementary operations that generate data for the external environment, such as registered user and taxi drivers. They usually include the elaboration of data from logic files. The external output of the application are:

- *After Log in, display user page and information:* This operation only involves the user entity in which all the needed information is stored. Because of this it is to be considered a simple weight: FPs.
- *After User request, notify taxi driver:* Once a user has requested a taxi, the system will look for the nearest driver and notify him. Since this operation will modify data from logic files by looking for a driver in the queues, this is considered an output. It involves n entities (registered user, taxi driver, zone, queue and notification) it has a complex weight: FPs.

5. External Inquiry

These are elementary operation that involve input and output, but they don't elaborate data from logic files.

- *Registered user notification:* Once the driver has accepted the registered user is notified with the driver decision. This operation involves few (Registered user, driver and notification) and as such has a medium weight: FPs.
- *User response notification to taxi driver:* Finally after the user has accepted or declined the taxi, the driver is notified. This has a complex weight since it implies a reordering of the queue and the elimination of the driver from it.   FPs.

6. Unadjusted Function Points

Once all the FPs from the previous chapters have been summed, we obtain a total of 90 unadjusted function points. This result can be used independently to estimate the size of the application. The function points will not be adjusted since they will later be used in the COCOMO approach.

3. COCOMO Approach

The Constructive Cost Model is an algorithmic technique that allows to estimate the required effort through a complex and non-linear model. The first step in this approach is to use the Effort Equation to estimate the number of Staff-Months required for our project. In order to utilize this equation we need the SLOC (source line of code) which can be calculated from the FP obtained in the previous chapter. The COCOMO II Model Manual does not include the conversion rate of UFP into SLOC for the JavaEE language, but from this updated link, http://www.qsm.com/resources/function-point-languages-table, we can find the conversion rate we need: 46. Now we can estimate the number of lines of code our application will have:

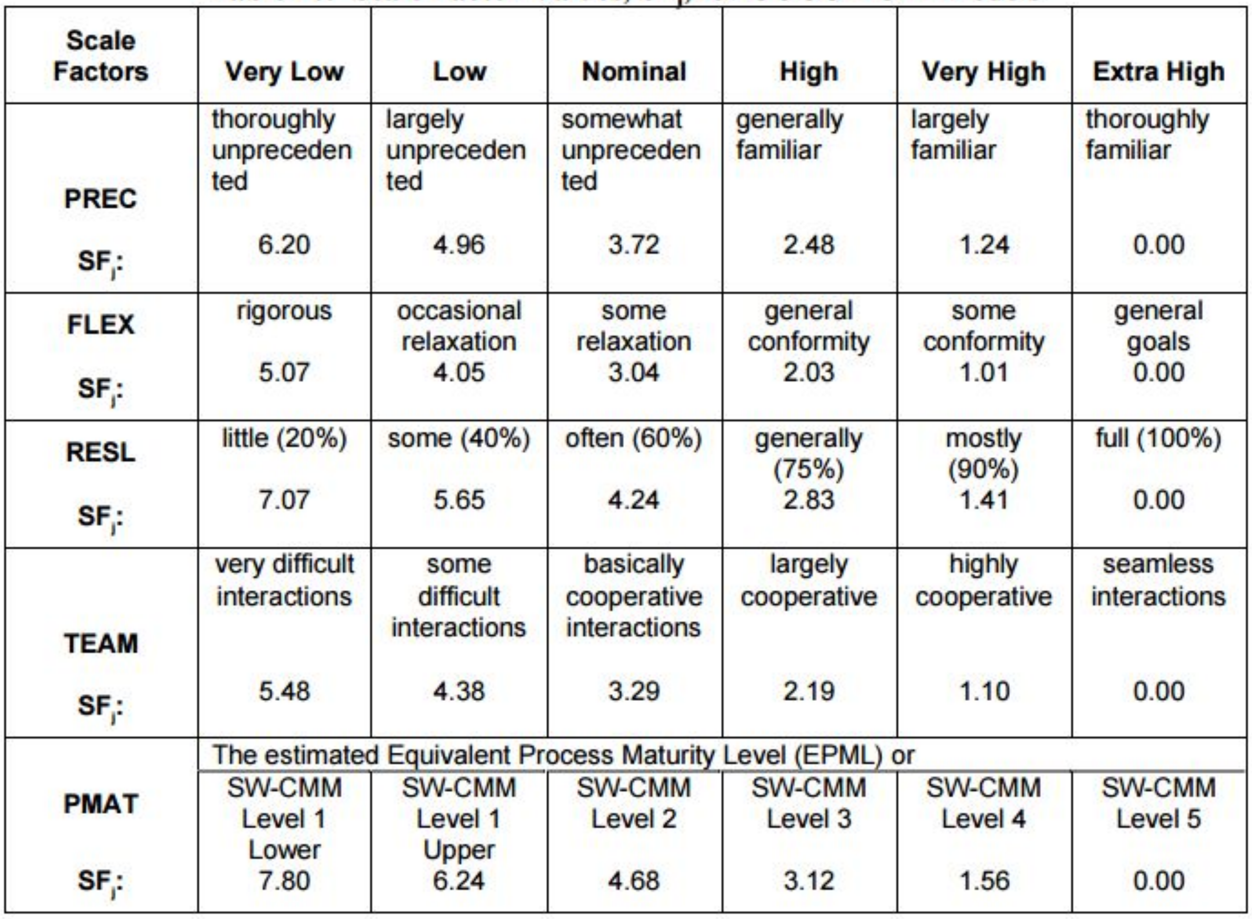
Now the Effort Equation formula is:

We still need the EAF and E to be able to calculate. EAF is the Effort Adjustment Factor, which can be derived from the Cost Drivers, while E in the Exponent derived from the Scale Drivers.

1. Scale Drivers

In order to calculate the value of E we need to analyse the Scale Drivers and see how they are to be considered inside the myTaxiService application. The five scale drivers that will be considered are: Precedentedness, Development Flexibility, Architecture/Risk Resolution, Team Cohesion and Process Maturity. The following table from the COCOMO II manual will be used to determine the values of the different scale drivers:

Table 10. Scale Factor Values, SF_j, 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_j:	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_j:	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_j:	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_j:	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_j:	7.80	6.24	4.68	3.12	1.56	0.00

- **Precedentedness:** Reflects the previous experience of the organisation with this type of project. For this project it is the first time using JavaEE and these methodologies, so the value will be low: 4.96.
- **Development Flexibility:** Reflects the degree of flexibility in the development process. The specification document did not go too much in detail on every aspect of the project, so a high degree of freedom was given in every phase of the project. This value will be high: 1.01.
- **Architecture/Risk Resolution:** Reflects the extent of risk analysis carried out. Since most of the project was theoretical with no coding, most risk were avoided. This value will be high: 2.83.
- **Team Cohesion:** Reflects how well the development team knew each other and work together. Since this is a one person project, this will be extra high: 0.00.
- **Process Maturity:** Reflect the process maturity of the organization. Since all set goals were reached, this value will be high: 3.12.

Summing all the values considered in the previous list, we obtain:

Now we can calculate the value of E with the following equation, where B is a constant of value 0.91 for COCOMO II:

2. Cost Drives

COCOMO defines 17 cost driver used to calculate the value of the Effort Adjustment Factor, which is obtained by multiplying all the 17 factors of the different cost drivers. In this chapter each one of the cost drivers will be briefly described and the corresponding value will be reported.

- **Required Software Reliability:** This is the measure of the extent to which the software must perform its intended function over a period of time. The factor used here is low: 0.92.
- **Data Base Size:** This cost driver attempts to capture the effect large test data requirements have on product development. Since only zones, queues and information about users are stored, the value would be low: 0.90.
- **Product Complexity:** Complexity is divided into five areas: control operations, computational operations, device-dependent operations, data management operations, and user interface management operations. This value is considered high: 1.17.
- **Developed for Reusability:** This cost driver accounts for the additional effort needed to construct components intended for reuse on current or future projects. The value is to be considered high: 1.07.
- **Documentation Match to Life-Cycle Needs:** The rating scale for the DOCU cost driver is evaluated in terms of the suitability of the project's documentation to its life-cycle needs. This value is set to nominal: 1.
- **Execution Time Constraint:** This is a measure of the execution time constraint imposed upon a software system. There are no such constraints in the project so the value will be n/a.
- **Main Storage Constraint:** This rating represents the degree of main storage constraint imposed on a software system or subsystem. Once again, no constraints of this type are present in the project: n/a.

- **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. 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. In our case, the platform is quite fixed and doesn’t change much, except for adding new taxi drivers, so the value is low: 0.87.
- **Analyst Capability:** 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. Since this project only revolved around analysing requirements and designing the project, this value should be set to very high: 0.71.
- **Programmer Capability:** Major factors which should be considered in the rating are ability, efficiency and thoroughness, and the ability to communicate and cooperate. The project will not be developed, so in order to not influence the final value with this factor, the value is set to nominal: 1.
- **Personnel Continuity** The rating scale for PCON is in terms of the project’s annual personnel turnover. This value will be very low: 1.29.
- **Applications Experience:** The rating for this cost driver is dependent on the level of applications experience of the project team developing the software system or subsystem. Since these are new environments, this value is set to low: 1.10.
- **Platform Experience:** The rating for this cost driver depends on the importance of understanding the use of more powerful platforms, including more graphic user interface, database, networking, and distributed middleware capabilities. These aspect are more known than the previous, so the value is set to nominal: 1.
- **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. The java language tools are known, but the experience with JavaEE is lower, so the value is low: 1.09.
- **Use of Software Tool:** The tool rating ranges from simple edit and code, very low, to integrated life-cycle management tools, very high. Since the experience in these environments is low, the best value here is nominal: 1.
- **Multisite Development:** Determining this cost driver rating involves the assessment and judgement-based averaging of two factors: site collocation and communication support. Since the project is developed by only one person in one city, this value is set to very low: 1.22.
- **Required Development Schedule:** This rating measures the schedule constraint imposed on the project team developing the software. The effort used for this project was distributed over the given time, even though not perfectly with high efforts near deadlines, for this reason the parameter is nominal : 1.

Once all the cost drivers have been analysed and defined, we can obtain the EAF by multiplying all the factor of the different drivers:

Cost Driver	Value
Required software Reliability	0.92
Data Base Size	0.90
Product Complexity	1.17
Required Reusability	1.07
Documentation match to life-cycle needs	1

Execution Time Constraints	n/a
Main Storage Constraints	n/a
Platform Volatility	0.87
Analyst Capability	0.71
Programmer Capability	1
Personnel Continuity	1.29
Application Experience	1.10
Platform Experience	1
Language and tool Experience	1.09
Use of Software Tools	1
Multisite Development	1.22
Required Development Schedule	1
EAF:	1,2082

3. Effort and Duration

At this point we have all necessary data to calculate the value of the Effort:

The COCOMO approach also allows us to calculate the number of months required to complete the software project, with the following equation:

Where SE in the schedule equation exponent obtained from the five Scale Drives. Its value can be calculated with the following equation, where E is the same used in the effort equation and B is a constant of value 0.91 for COCOMO II:

At this point we have all the values to calculate the duration:

This is the estimated time it would have taken to develop completely the application from nothing. Another important value is the number of people which would be ideal to develop this application. This number is obtained by dividing the effort for the duration:

In conclusion it would have taken two developers more than eight months to develop the application.

4. Task Identification and Schedule

The base on which the tasks will be defined are the different chapters in which the documents we produced are divided. A more in depth description on the division of the chapter in tasks will be given in the next chapter in which the resources will be assigned to the tasks. The hours I had available for working on the project depended on which day of the week it was and how much work I had to do from other courses. A general average of how many hours I invested every day of the week to the project is: Monday: 1-2 hours in the afternoon; Tuesday: 1-2 hours in the afternoon; Wednesday: 0-1 hour first half of semester, 1-2 hours the second half; Thursday: 0-30 min; Friday: 0-1 hour if I had lectures, 1-2 hours otherwise; Saturday and Sunday: 3-6 hours. Not every day of every week was spent working on the project: most of the work was done on Monday and Tuesday afternoon and in the weekends. The first document produced and first part of the project was the Requirement Analysis and Specification Document:

RASD: DEADLINE: 6/11/2015

TASK	DESCRIPTION(chapter name)	START DATE	END DATE
T1	Introduction	16/10/2015	18/10/2015
T2	Overall Description	19/10/2015	20/10/2015
T3	Specific Requirements	21/10/2015	25/10/2015
T4	Scenarios	26/10/2015	27/10/2015
T5	UML Models	28/10/2015	1/11/2015
T6	Alloy Models	2/11/2015	6/11/2015
T7	Appendix and Review	6/11/2015	6/11/2015

DESIGN DOCUMENT: DEADLINE: 4/12/2015

TASK	DESCRIPTION(chapter name)	START DATE	END DATE
T8	Introduction	8/11/2015	8/11/2015
T9	Architectural View	12/11/2015	22/11/2015
T10	Algorithmic Design	23/11/2015	26/11/2015
T11	User Interface Design	27/11/2015	29/11/2015
T12	Requirements Traceability	30/11/2015	3/12/2015
T13	Review	3/12/2015	4/12/2015

INSPECTION DOCUMENT: DEADLINE: 5/01/2016

TASK	DESCRIPTION(chapter name)	START DATE	END DATE
T14	Assigned Methods and Classes	10/12/2015	10/12/2015
T15	Functional role	10/12/2015	13/12/2015
T16	Code Inspection Checklist	13/12/2015	24/12/2015
T17	Other Problems and review	4/1/2016	5/01/2016

INTEGRATION TEST PLAN DOCUMENT: DEADLINE: 21/01/2016

TAKS	DESCRIPTION(chapter name)	START DATE	END DATE
T18	Introduction	10/01/2016	10/01/2016
T19	Integration Strategy	13/01/2016	15/01/2016
T20	Individual Steps and Test Description	18/01/2016	19/01/2016
T21	Tools and test Equipment	20/02/2016	21/01/2016
T22	Program Stubs, Drivers and Test Data	21/01/2016	21/01/2016

PROJECT PLAN DOCUMENT: DEADLINE: 2/02/2016

TASK	DESCRIPTION(chapter name)	START DATE	END DATE
T23	Introduction	27/01/2016	27/01/2016
T24	Function Points	27/01/2016	28/01/2016
T25	COCOMO	29/01/2016	30/01/2015
T26	Task Identification and Scheduling	31/01/2016	1/02/2015
T27	Resource Allocation	2/02/2016	2/02/2016
T28	Risk	2/02/2016	2/02/2016

5. Resource Allocation

For this chapter I supposed to have had a partner to work with in these moths and so I will divide the work in the way I would have if someone else worked in this project with me. This partner had the same available hours to work on the project as I had. Let’s suppose the two student’s name are S1 and S2. The working time for each task will be the same number of days described in the previous chapter, but since two people are working on the project, the document might be completed earlier. Sometimes a task will take more time to complete than stated in the previous chapter. In this case the document will contain more details. Normally the week scheduling goes from Monday to Friday, but in this case weekend are also considered working days.

Resource allocation for the RASD Document would be like this:

Student, Week	16/10-23/10	23/10-30/10	30/10-6/11
S1	T1 and T2	T4	T6
S2	T3	T5	T7

Resource allocation for Design Document:

Student, Week	8/11-15/11	15/11-22/11	22/11-29/11	29/11-4/12
S1	T8 and T9	T10	T12	T13
S2	T9	T11	T12	-

Resource Allocation for Inspection Document:

Student, Week	10/12-17/12	17/12-24/12	24/12-01/01
S1	T14 and T15	T16 and T17	-
S2	T16	T16	-

Resource Allocation for Integration Test Plan:

Student, Week	07/01-14/01	14/01-21/01
S1	T18 and T19	T21
S2	T20	T22

Resource Allocation for Project Plan Document:

Student, Week	22/01-29/01	29/01-02/02
S1	T23 and T24 and T25	-
S2	T26 and T27	T28

6. Risk Evaluation

In this chapter a few risks that might have happened during the production of these documents will be described, and a proactive solution will be given for each of them.

Risk	**Probability**	**Effects**
Illness	Moderate	Serious
Computer Crush/Power Cut	High	Serious
No internet connection for delivery	high	Serious
Changes to requirements	Moderate	Serious

- **Illness:** A period of illness close to the delivery of the project might cause the deadline to not be respected. For this reason the work was done as much as possible in advance in order to have a few days free before the delivery in case other days were lost because of illness.
- **Computer Crush/Power Cut:** Both these situation might result in the loss of an entire day worth of working. Microsoft Word does a good job in retrieving non saved version of a file in case of computer crush. Furthermore all the documents were saved on a cloud service, namely Dropbox, in order first of all to be accessible from anywhere on any machine, but also so that even in the case of a computer breaking, no work would be lost.

- **No internet connection for delivery:** Since the delivery of the project was done through the upload of the document online, an internet connection was required to do so. In order to make sure the documents were always uploaded, an earlier version, if the final wasn't yet ready, was uploaded some time before the delivery, to make sure that a copy of the document was delivered, even if not final.
- **Changes to Requirements:** Since it is a one person project, not many changes were made to the documents once they were delivered. This was made to avoid a cascade effect on all other documents which would have taken a lot of time to correct.