Chapter 1

Project Estimations

In this chapter we are going to estimate the main features of *myTaxiService* project, by using COCOMO II. Reading from the reference manual:

> *The COCOMO II model is part of a suite of Constructive Cost Models. This suite is an effort to update and extend the well-known COCOMO (Constructive Cost Model) software cost estimation model originally published in Software Engineering Economics by Barry Boehm in 1981.*

In the section 1.1 we focus on the project's size in term of lines of code, whereas in the section 1.2 other metrics, such the required time and the costs will be analysed.

1.1 Project Size (Function Points)

Reading from the reference manual:

> *The function point cost estimation approach is based on the amount of functionality in a software project and a set of individual project factors [Behrens 1983; Kunkler 1985; IFPUG 1994]. Function points are useful estimators since they are based on information that is available early in the project life-cycle. A brief summary of function points and their calculation in support of COCOMO II follows.*

The function types are five, described in the table[1].

Function Point	**Description**
External Input (EI)	Count each unique user data or user control input type that enters the external boundary of the software system being measured.
External Output (EO)	Count each unique user data or control output type that leaves the external boundary of the software system being measured.
Internal Logical File (ILF)	Count each major logical group of user data or control information in the software system as a logical internal file type. Include each logical file (e.g., each logical group of data) that is generated, used, or maintained by the software system.
External Interface Files (EIF)	Files passed or shared between software systems should be counted as external interface file types within each system.
External Inquiry (EQ)	Count each unique input-output combination, where input causes and generates an immediate output, as an external inquiry type.

Finally, to perform the analysis we have to present other two tables from the same reference manual of the other one. The first one will be used to classify each function on three level of complexity (high, medium and low).

The second one shows the weights to be used into the estimation formulas[2].

[1]The table is given by the COCOMO II reference manual.

[2]The UFP acronym means Unadjusted Function Points

Table 2. FP Counting Weights

For Internal Logical Files and External Interface Files

	Data Elements		
Record Elements	**1 - 19**	**20 - 50**	**51+**
1	Low	Low	Avg.
2 - 5	Low	Avg.	High
6+	Avg.	High	High

For External Output and External Inquiry

	Data Elements		
File Types	**1 - 5**	**6 - 19**	**20+**
0 or 1	Low	Low	Avg.
2 - 3	Low	Avg.	High
4+	Avg.	High	High

For External Input

	Data Elements		
File Types	**1 - 4**	**5 - 15**	**16+**
0 or 1	Low	Low	Avg.
2 - 3	Low	Avg.	High
3+	Avg.	High	High

Table 3. UFP Complexity Weights

	Complexity-Weight		
Function Type	**Low**	**Average**	**High**
Internal Logical Files	7	10	15
External Interfaces Files	5	7	10
External Inputs	3	4	6
External Outputs	4	5	7
External Inquiries	3	4	6

Up to now, we have presented the Function Points technique. Now, we are going to start our analysis, split by the function type.

1.1.1 Internal Logic Files

The system has to manage Internal Logic Files to store information related to the users (both *normal* and drivers), the *historical* rides, the areas and the driver work shifts[3]
The users have from 12 to 16 fields to be stored (the second number is referred to the drivers case) and only the alerts and the zerotime or future rides have to be stored, thus the complexity is low. The areas and the work shifts can also be considered as low complexity type. In fact they have a few fields and less than six extra records.
The rides have 10 fields, including two positions, the driver and the passenger, all saved in separate entities. They can be considered as an average complexity type, since we have about seven records per ride (in fact in addition to the five presented, the positions requires additional records).

In the table the analysis is summarize:

ILF	Complexity	FP
User	Low	7
Area	Low	7
Workshift	Low	7
Ride	Average	10
Total		31

1.1.2 External Logic Files

The system acquires data from te GPS interface. A GPS position is essentially a tuple of type Position, described in our database. Hence, we have a low complexity type and 5 FP.

[3]See the logic schema at the page 10 of the Design Document to have a detailed description of each part of the database.

1.1.3 External Inputs

The possible interactions between the users and the system, defined in the RASD, are now quickly described in terms of complexity:

- Login/Logout: these operations are simple due to one entity only is involved, so the complexity is low;
- Start Waiting Time/End of a ride: these operation requires to interact with three types of files (Position, Area and Driver Waiting) with one element per type, thus the complexity is low;
- Check the Reservations: this is a group of three related operations (shows the alerts and gives the possibilities to modify or to cancel a ride) that involve one type and potentially more than 16 elements, so the complexity is average;

In the following table we have summarized the results:

EI	Complexity	FP
Login/Logout	Low	2x3
Start Waiting Time / End of a Ride	Low	2x3
Check the Reservations	Average	3x4
Total		24

1.1.4 External Inquiries

The system allows the user to interact with it through the following operations:

- Registration: this operation is performed only by simple user (not a driver) and involves one data type, the one related to the new user. Its complexity is low;
- Profile Management: this operation allows the user to modify a little personal data, so the complexity is low;
- Work shift Management: this operation requires to interact to 2 entities (driver and work shift) and can involve more than 20 elements to perform the validity checks. Hence the complexity is high;
- Book a ride (both future or zerotime): these operations needs many inter-

actions between the system and the user, for instance to show the detected position or to analyse the inserted time/address;

- Ride Allocation: we also insert this function because the system has to interact in a simple way with the taxi driver. Since many and many taxi drivers or queues may be involved before one is available, the complexity is high. Note that the notifications of the operations are not reported here, but in the External Input section;
- Taxi Driver Ride Request: this is the operation used to ask a rider the availability for a ride, the complexity is low.

In the following table we have summarize the results:

EQ	Complexity	FP
Registration	Low	3
Profile Management	Low	3
Workshift Management	High	6
Book a ride	High	2x6
Ride Allocation	High	6
Taxi Driver Ride Request	Low	3
Total		33

1.1.5 External Outputs

The external outputs shown by the system are all related with the notifications. In fact the system administrators can notify all the users about service situations (for instance a strike, an incident that forbids the access in some city areas and so on). Other kind of notifications, are the ones about the requested ride's status.
Finally we have the operations of shown these notifications (Read the Alerts). All the described operations have to interact with about three types of record (always the user and the alert. If any, also other files are involved, as the ride or the position) and many files (both users or old alerts when we are showing the alerts), thus the complexity is high. Instead, the ride status notifications have to interact with one user, so the complexity is low.

The results are summarized in the following table:

EO	Complexity	FP
System Notifications	High	7
Ride Status Notifications	Low	4
Read The Alerts	High	7
Total		18

1.1.6 Final Results

The Function Point found in the previous sections are reported in the following table:

Function Type	Value
Internal Logic Files (ILF)	31
External Logic Files (ELF)	5
External Inputs (EI)	24
External Inquiries (EQ)	33
External Outputs (EO)	18
Total	111

Since our project has no a specific programming language to be used, in the following table we report the project size estimations both for the C++ and for the Java. In addition we report a few interesting measure with C, Assembler and Machine Code:

Programming Language	UFP to SLOC Conversion Ratios[4]	Lines of Code
Java	53	5833
C++	55	6105
C	128	14208
Assembly - Basic	320	35520
Machine Code	640	71040

[4]The values are still taken from the COCOMO II reference manual.

1.2 Effort Estimation (COCOMO II)

In this section we make an Effort Estimation using the model of COCOMO II. Following this model we can calculate the required effort measured in Person-Months with the effort equation, that is:

$$effort = 2.94 * EAF * (KSLOC)^E \tag{1.1}$$

where the first factor 2.94 is a constant calculated by COCOMO II 2000.0, **EAF** is the *Effort Adjustment Factor* calculated as the product of the effort multipliers corresponding to each of the *Cost Drivers*, **KSLOC** is the estimated number of lines of code calculated at the end of section 1.1 and measured in thousands, the exponent **E** is the exponent derived from the *Scale Drivers*.

First we calculate the *Scale Drivers* in the first subsection and then the *Cost Drivers*. Finally we will compute the result of the effort equation presented above and also the duration of the necessary work to complete the project and the number of required people.

1.2.1 Scale Drivers

The exponent **E** in equation 1.1 is an aggregation of five scale factors (SF) that account for the relative economies or diseconomies of scale encountered for software projects of different sizes. These five scale factors with their corresponding numerical values are represented in the Table 1.1

Table 1.1: 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	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

The scale factors and their values for our project are:

- **PREC - Precedentedness:**
  Reflects the previous experience of the organisation with this type of project. Since we haven't realised before this kind of commercial, distributed and multi-platform system (accessible both via website and smartphone), the value of this scale factor is *Low* (4.96).
- **FLEX - Development Flexibility:**
  Reflects the degree of flexibility in the development process. The instructor has set only some general requirements and goals without going in detail, hence the development of the project is very flexible. The value of this scale factor is *Very High* (1.01).

- **RESL - Architecture/Risk Resolution:**
  Reflects the extend of risk analysis carried out. We have studied the necessary architecture for the system and we have identified and prevented many critical security risks, hence this value is *High* (2.83).
- **TEAM - Team Cohesion:**
  Reflects how well the development team know each other and work together. In our case we know each other very well and we have also worked together before for another important project. For the development of myTaxiService we have worked together most of the time and other times we have split equally the work between us and worked independently at home, always communicating. Thus this value is *Extra High* (0.00).
- **PMAT - Process Maturity:**
  Reflects the process maturity of the organisation. We have calculated this value basing on the 18 Key Process Areas (KPAs) in the SEI Capability Maturity Model[5]. We report only the assigned value for each KPA:
  1. Requirements Management: Almost-Always
  2. Software Project Planning: Almost-Always
  3. Software Project Tracking and Oversight: Frequently
  4. Software Subcontract Management: Does not apply
  5. Software Quality Assurance (SQA): Frequently
  6. Software Configuration Management (SCM): Frequently
  7. Organization Process Focus: Does not apply
  8. Organization Process Definition: Does not apply
  9. Training Program: Does not apply
  10. Integrated Software Management: Almost-Always
  11. Software Product Engineering: Almost-Always
  12. Intergroup Coordination: Almost-Always
  13. Peer Reviews: Frequently
  14. Quantitative Process Management: About Half
  15. Software Quality Management: About Half
  16. Defect Prevention: Rarely

[5]Paulk et al. 1995

17. Technology Change Management: Rarely
18. Process Change Management: About Half

In the manual we can translate this classifications into a value between zero and five with a formula which we do not report. The given value is 3.4, thus the value is high.

The results are resumed below:

Scale Driver	**Factor**	**Value**
PREC - Precedentedness	Low	4.96
FLEX - Development Flexibility	Very High	1.01
RESL - Architecture/Risk Resolution	High	2.83
TEAM - Team Cohesion	Extra High	0.00
PMAT - Process Maturity	High	3.12
Total		**11.92**

1.2.2 Cost Drivers

The factor **EAF** in equation 1.1 is the product of seventeen Post-Architecture effort multipliers used in COCOMO II model to adjust the nominal effort, Person-Months, to reflect the software product under development. We will describe each of the multipliers and compute the corresponding value for our project. The values are obtained from the tables in the official COCOMO II Model 2000.0 and we do not report them here.

1. **Required Software Reliability (RELY)**:
   This is the measure of the extent to which the software must perform its intended function over a period of time. If the effect of a software failure is only slight inconvenience then RELY is very low. If a failure would risk human life then RELY is very high.
   This factor for myTaxiService has value *Nominal* (1.00).
2. **Data Base Size (DATA)**:
   This cost driver attempts to capture the effect large test data requirements

have on product development. The rating is determined by calculating D/P, the ratio of bytes in the testing database to SLOC (Source Lines Of Code) in the program.

Since we desire to have a *Nominal* value (1.00) in this section (the database is one of the most important parts of our system) and we have a dimension in the order of SLOC near 6000[6], we need a database dimension in the order of 6000*25 150 KB, where 25 is a *Nominal* rate for the division D/SLOC (D = Database dimension).

3. **Product Complexity (CPLX)**:
   Complexity is divided into five areas: control operations, computational operations, device-dependent operations, data management operations, and user interface management operations. The complexity rating is the subjective weighted average of the selected area ratings.
   The value of this factor for our project is High (1.07).
4. **Developed for Reusability (RUSE)**:
   This cost driver accounts for the additional effort needed to construct components intended for reuse on current or future projects. Our application can be used by cab companies (and consequently the customers) of many cities or countries. Thus this value of reusability is intended as "across product line " and set to *Very High* (1.15).
5. **Documentation Match to Life-Cycle Needs (DOCU)**:
   Several software cost models have a cost driver for the level of required documentation. In COCOMO II, 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. Our project is well documented during it's life-cycle, hence this value is *Nominal* (1.00).
6. **Execution Time Constraint (TIME)**:
   This is a measure of the execution time constraint imposed upon a software system. The rating is expressed in terms of the percentage of available execution time expected to be used by the system or subsystem con-

[6]See the section 1.1 to have two possible precise estimations for an OO programming language.

suming the execution time resource. The rating ranges from nominal, less than 50% of the execution time resource used, to extra high, 95% of the execution time resource is consumed.

The execution time of myTaxyService operations is supposed to be less than 50% of the total available time, in normal conditions, because for instance a ride should be rapidly assigned to a driver when a customer asks for it. Hence the value of this factor is *Nominal* (1.00).

7. **Main Storage Constraint (STOR)**:
   This rating represents the degree of main storage constraint imposed on a software system or subsystem. The rating ranges from nominal (less than 50%), to extra high (95%). The use of storage by myTaxiService should be less than 50%, so the value is *Nominal* (1.00).
8. **Platform Volatility (PVOL)**:
   In our application we can consider as platforms the DBMS, the operating system, the distributed information repositories, the browser, the web server, the Google Maps APIs and the hardware as far as the environment concerns. The platforms could change frequently (especially the Google Maps APIs), hence this value is *Nominal* (1.00).
9. **Analyst Capability (ACAP)**:
   Analysts are personnel who 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 have analysed any possible scenario of using of myTaxiService and any problem that may occur. The requirements have been correctly satisfied with the solutions described both in the RASD and DD. This value is set to *Very High* (0.71).
10. **Programmer Capability (PCAP)**:
    This value is based on the capability of the programmers as a team rather than as individuals. Major factors which should be considered in the rating are ability, efficiency and thoroughness, and the ability to communicate and cooperate.

We have worked together efficiently and we have equally split the work between us. Hence the value for this factor is *Very High* (0.76).

11. **Personnel Continuity (PCON)**:
    The rating scale for PCON is in terms of the project's annual personnel turnover: from 3%, very high continuity, to 48%, very low continuity. Since we are the only two people working on this project this value is set to *Very High* continuity (0.81).
12. **Applications Experience (APEX)**:
    The rating for this cost driver (formerly labeled AEXP) is dependent on the level of applications experience of the project team developing the software system or subsystem. The ratings are defined in terms of the project team's equivalent level of experience with this type of application. Since we haven't realised before this kind of commercial, distributed and multi-platform system (accessible both via website and smartphone), the value of this scale factor is *Very Low* (1.22).
13. **Platform Experience (PLEX)**:
    The Post-Architecture model broadens the productivity influence of platform experience, PLEX (formerly labeled PEXP), by recognizing the importance of understanding the use of more powerful platforms, including more graphic user interface, database, networking, and distributed middleware capabilities.
    Our knowledge about the aforementioned platforms is about one year and half, hence this value is set to *Nominal* (1.00).
14. **Language and Tool Experience (LTEX)**:
    This is a measure of the level of programming language and software tool experience of the project team developing the software system or subsystem. Software development includes the use of tools that perform requirements and design representation and analysis, configuration management, document extraction, library management, program style and formatting, consistency checking, planning and control, etc.
    This value is set to *Nominal* (1.00).

15. **Use of Software Tools (TOOL)**:
    The tool rating ranges from simple edit and code, to integrated life-cycle management tools. We used StarUML to develop the diagrams (class diagram, sequence diagrams, statecharts, UX and so on) of the application and other tools useful for the design and representation of the architecture, such as Microsoft Visio and MySQL Workbench, and finally Git for the repository management. Thus this value is set to *Nominal* (1.00).
16. **Multisite Development (SITE)**:
    This parameter reflects how we handled the distribution of development over distance and multiple platforms. We've used phones and chat, so this value is set to *Nominal* (1.00).
17. **Required Development Schedule (SCED)**:
    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.
    Our efforts were well distributed over the available development time, hence this value is set to *Nominal* (1.00).

The results are resumed below:

Cost Driver	**Factor**	**Value**
Required Software Reliability (RELY)	Nominal	1.00
Data Base Size (DATA)	Nominal	1.00
Product Complexity (CPLX)	High	1.07
Developed for Reusability (RUSE)	Very High	1.15
Documentation Match to Life-Cycle Needs (DOCU)	Nominal	1.00
Execution Time Constraint (TIME)	Nominal	1.00
Main Storage Constraint (STOR)	Nominal	1.00
Platform Volatility (PVOL)	Nominal	1.00
Analyst Capability (ACAP)	Very High	0.71
Programmer Capability (PCAP)	Very High	0.76
Personnel Continuity (PCON)	Very High	0.81
Applications Experience (APEX)	Very Low	1.22
Platform Experience (PLEX)	Nominal	1.00
Language and Tool Experience (LTEX)	Nominal	1.00
Use of Software Tools (TOOL)	Nominal	1.00
Multisite Development (SITE)	Nominal	1.00
Required Development Schedule (SCED)	Nominal	1.00
Total product		**0.656**

1.2.3 Effort Estimation Results

Now that we have calculated all the values for the the scale drivers and the cost drivers we can compute the result of the Effort Equation in terms of Person-Months required and also the duration of the project development by imposing that the number of people working on it is two.
For the effort equation we have to calculate the value of the exponent **E** of the

equation, with the following formula:

$$E = B + 0.01 * \sum SF_i = B + 0.01 * 11.92 = 0.91 + 0.1192 = 1.0292$$

where the value of **B** is 0.91 for COCOMO II 2000.0.

Now that we have the value of E and the value of **EAF** (that is the product of the cost drivers), if we program the software in Java (for example) the estimated number of source lines of code is 5833 [7] (which corresponds to a value of KSLOC equal to 5.833), thus the result of the effort equation is:

$$effort = 2.94 * EAF * (KSLOC)^E = 2.94 * 0.656 * (5.833)^{1.0292} = 11.84424\ PM$$

Since we are are developing the project only in two people, we impose the number that could be estimated as $N = effort/duration$ equal to 2, so that we can estimate the duration of the development as:

$$duration = effort/N = 11.84424/2 = 5.922 \approx 6\ months$$

[7] we have estimated this value in section 1.1