POLITECNICO MILANO 1863

PROJECT PLAN DOCUMENT

Paolo Antonini 858242
Andrea Corneo 849793

version 1 – 2nd February 2016

myTaxiService

Contents

1 Introduction

1.1 Purpose and scope

After the long planning phase which myTaxiService underwent in the past months, it seems appropriate to define a precise planning for its development, as well as identify its supposed cost and the needed effort to implement it.

With this document we aim to reach this goal.

1.2 References

In the following chapters we will refer to the following documents:

- *COCOMO II.2000 model manual*: `http://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_modelman2000.0.pdf`.

- *Function Point Languages Table*: `http://www.qsm.com/resources/function-point-languages-table`.

Moreover, it is advisable to consider also the previous documents relating to myTaxiService:

- *Assignments 1 and 2*, section 2, parts I and II; 13th October 2015. This is to be regarded as the high level project description. Available at: `https://beep.metid.polimi.it/documents/3343933/d5865f65-6d37-484e-b0fa-04fcfe42216d`.

- *Requirement analysis and specification document*; 6th November 2015. Available at: `https://github.com/Cordaz/SE2_AntoniniCorneo/raw/master/Deliveries/1_RASD.pdf`.

- *Design document*; 4th December 2015. Available at: `https://github.com/Cordaz/SE2_AntoniniCorneo/raw/master/Deliveries/2_DD.pdf`.

- *Integration test plan document*; 21st January 2016. Available at: `https://github.com/Cordaz/SE2_AntoniniCorneo/raw/master/Deliveries/4_ITPD.pdf`.

1.3 *Overview of the document*

This document develops as follows.

In chapter 2 Function Points and COCOMO methods are applied in order to estimate the effort and cost to develop the project; the justification for some choices we made is left to appendix A, in order no to clutter chapter 2 itself. Chapter 3 deals with the scheduling of tasks and allocation of resources. Finally, in chapter 4 we report the main risks the development of the project may encounter.

2 Cost and effort estimation

In this chapter we are going to estimate the size of myTaxiService project by means of the Function points method, which is a language-independent way of quantifying program functionality.

Then, after converting Function points into number of lines of source code, we are able to derive a development effort estimation based on system requirements and design specification, in terms of time and number of people. This is possible thanks to the formulas elaborated in COCOMO II.2000 (*Constructive Cost Model*), which we are going to present in the next sections.

2.1 Function points

The Function points approach tries to measure the program functionalities, by quantifying the information processing functionality associated with major external data or control input, output, or file types. Five user function types are identified. At the end of estimation, the total number of function points (FP) is given by the following formula:

$$\text{FP} = \sum \#\text{functions\_by\_type} * \text{weight} \tag{2.1}$$

In table 2.1 the weights for each function type, according to their complexity, are presented, as a reference. Then, in the following paragraph, each type is analysed and the corresponding function points are calculated.

Table 2.1: Function points complexity weights table.

ID	Function type	Complexity weight		
		Simple	*Average*	*Complex*
`EI`	External input	3	4	6
`EO`	External output	4	5	7
`ILF`	Internal logical file	7	10	15
`EIF`	External interface files	5	7	10
`EQ`	External inquiry	3	4	6

External input

External input functions deal with unique user data or user control input type that enters the external boundary of the software system being measured.

Users interact with myTaxiService system in many ways:

- Login, logout and registration: these are simple operations managed by simple components.
- Memorise an address: it is a simple operation.
- Make a request, reservation: for each request or reservation the systems must interact with many entities. Moreover a taxi has to be allocated. These are complex operations, since many entities are involved.
- Cancel a reservation: a simple operation that requires only an object elimination.
- Report taxi availability: this operation requires looking for the pertinence area of the taxi, in order to correctly enqueue it; this can be estimated as an average operation.
- Taxi driver accepts or refuses a ride: the first case involves two simple operations, since the ride shall be initiated and a notification must be prepared. Instead, the second case is more complicated: whenever a ride is refused, the system makes a new allocation.

FUNCTION	COMPLEXITY	NUMBER	TOTAL WEIGHT
Login, logout, registration	*Simple*	3	$3 * 3 = 9$
Address memorisation	*Simple*	1	$1 * 3 = 3$
Request, reservation	*Complex*	3	$3 * 6 = 18$
Cancel reservation	*Simple*	1	$1 * 3 = 3$
Report availability	*Average*	1	$1 * 4 = 4$
Ride accepted	*Simple*	2	$2 * 3 = 6$
Ride refused	*Complex*	1	$1 * 6 = 6$
TOTAL EI			49

Table 2.2: External input summary table.

External output

This kind of functions counts each unique user data or control output type that leaves the external boundary of the software system.

myTaxiService system shall provide notification to all types of users, which can be of various type (SMS, push, ...). Each notification contains a small amount of data, however those must comply with protocols, which makes the operation a bit more complex: an average cost is to be considered.

Function	Complexity	Number	Total weight
Notification	*Average*	1	$1 * 5 = 5$
Total EO			5

Table 2.3: External output summary table.

Internal logical file

Internal logical files are the major logical group of user data or control information in the software system as a logical internal file type. These include each logical file that is generated, used, or maintained by the software system.

Our system includes many ILFs, as they are needed to store all information about customers, areas and addresses, cabs and drivers, requests, reservations and allocations. In detail:

- few information about customers are stored, in simple structures: first and last name, login credential and phone number. However, since registered customers' addresses are stored as well, it is appropriate to consider an average cost.
- to map the whole city in addresses and in their pertinence areas few data are needed: string and GPS data types may be sufficient to the purpose. However, every area contains a great amount of addresses, so an high complexity is to be accounted for.
- although many information about taxi drivers and taxi cabs are stored in the system, data needed for the operation of the system are more limited and very simple. Given that, we assign a low complexity level.
- the system needs to store many information for each request, reservation and many other information are needed to allocate the taxi. This requires a high complexity operation.

Function	Complexity	Number	Total weight
Customers	*Average*	1	$1 * 10 = 10$
Areas and addresses	*Complex*	2	$2 * 15 = 30$
Taxis, taxi drivers	*Simple*	1	$1 * 7 = 7$
Requests, reservations, allocations	*Complex*	3	$3 * 15 = 45$
Total ILF			92

Table 2.4: Internal logical files summary table.

External interface files

Files passed or shared between software systems should be counted as external interface file types within each system.

As a matter of fact, myTaxiService system does not require any external data, so this is set to 0.

External inquiry

Here we count each unique input-output combination, where input causes and generates an immediate output, as an external inquiry type.

Registered customers can access their profile and their memorised addresses. Both operations can be considered as simple.

Function	Complexity	Number	Total weight
Profile	*Simple*	1	$1 * 3 = 3$
Memorise address	*Simple*	1	$1 * 3 = 3$
Total `EQ`			6

Table 2.5: External inquiry summary table.

2.1.1 *Total function points*

The results from previous sections (tables 2.2 to 2.5) are summarised in table 2.6.

ID	Function type	FP
`EI`	External input	49
`EO`	External output	5
`ILF`	Internal logical file	92
`EIF`	External interface files	0
`EQ`	External inquiry	6

Table 2.6: Total function points.

By applying equation (2.1), we get the total number of Function points:

$$\text{FP} = 152 \tag{2.2}$$

Those we obtained are the Unadjusted function points (UFP). An attempt to adapt this result (which focuses on functionality) to the prediction of development costs was made, but as it was wittily noted, “it is like correcting the tallness of a person to relate it to a measure of his/her intelligence” (N. Fenton), so we are not going down this road.

2.2 *COCOMO*

In the following we are going to adopt the formulas defined in *COCOMO II.2000* standard. Only a brief introduction of the theoretic meaning of the formulas is given. For the sake of compactness, we are not going to present the whole tables from which values are extracted[1].

[1] Refer to the documents mentioned in section 1.2 for an insight on COCOMO model.

2.2.1 *Sizing the project*

A reasonable size estimate of a project is very important for a good model estimation. However, at this stage this is a challenging operation, since it is nearly impossible to correctly guess how much of myTaxiService code will be newly written, and how much, instead, will be reused or readapted.

That is why we are going to provide the (very unlikely) worst-case estimation: we assume that the whole code will be written from scratch. The quantity **s** is the estimated number of source lines of code (SLOC):

$$\mathbf{s} = p * \mathrm{FP} \tag{2.3}$$

Parameter $p = 46$ converts the function points to SLOC, in the specific case of J2EE[2] (Java Enterprise Edition).

[2] See section 1.2 for the source of this figure.

In our case the sizing of the project results:

$$\mathbf{s} = 46 * 152 = 6992 \tag{2.4}$$

2.2.2 *Effort estimation*

Effort is expressed in terms of person-months (PM). A person-month is the amount of time one person spends working on the software development project for one month. COCOMO defines the following formula to estimate it:

$$\mathbf{e} = 2.94 * \left(\frac{\mathbf{s}}{1000}\right)^{E} * \mathrm{EAF} \tag{2.5}$$

Exponent E and parameter EAF are defined as follows:

$$E = 0.91 + 0.01 * \sum_{i}^{5} \mathrm{SF}_i \tag{2.6}$$

$$\mathrm{EAF} = \prod_{i=1}^{n} \mathrm{EM}_i \tag{2.7}$$

In the following paragraphs we are going to present the factors from which exponent E and parameter EAF are derived. For a justification of the choices we made, refer to appendix A.

Scale factors

Exponent E is the aggregation of five scale factors that account for some possible overheads encountered for software projects. Each factor has a range of rating levels, from *Very low* to *Extra high*, and each rating level has a weight. The specific value of the weight is called a scale factor (SF). In table 2.7 we are presenting the values we adopted.

ID	Scale factor	Level	SF
PREC	Precedentedness	*Low*	4.96
FLEX	Development flexibility	*High*	2.03
RESL	Risk resolution	*Nominal*	4.24
TEAM	Team cohesion	*Very high*	1.10
PMAT	Process maturity	*High*	3.12

Table 2.7: Scale factors summary table.

As a result of the previous choices and on the basis of equation (2.6), the parameter E results:

$$E = 0.91 + 0.01 * 15.45 = 1.0645 \tag{2.8}$$

Cost drivers

The parameter EAF is derived from the effort multipliers (EM) of the Cost drivers. Cost drivers are used to capture characteristics of the product under development, of the personnel working on it, and of general practices that affect the effort to complete the project.

Again, in table 2.8 we present only the summary of our evaluation of the parameters.

ID	Cost factor	Level	EM
RELY	Required software reliability	*Low*	0.92
DATA	Data base size	*Nominal*	1.00
CPLX	Product complexity	*High*	1.17
RUSE	Developed for reusability	*High*	1.07
DOCU	Documentation for lifecycle needs	*High*	1.11
TIME	Execution time constraint	–	–
STOR	Main storage constraint	–	–
PVOL	Platform volatility	*Low*	0.87
ACAP	Analyst capability	*High*	0.85
PCAP	Programmer capability	*High*	0.88
PCON	Personnel continuity	*Very low*	1.29
APEX	Applications experience	*Nominal*	1.00
PLEX	Platform Experience	*Nominal*	1.00
LTEX	Language and tool experience	*Nominal*	1.00
TOOL	Use of software tools	*Nominal*	1.00
SITE	Multisite development	*Extra high*	0.80
SCED	Required development schedule	*High*	1.00

Table 2.8: Cost drivers summary table.

As a result of the previous choices, parameter EAF, defined in equation (2.7), results:

$$\text{EAF} = 0.8586 \tag{2.9}$$

Effort estimation

Thanks to the results in equations (2.8) and (2.9), we are now able to estimate the value of the effort, defined in equation (2.5):

$$\mathbf{e} = 2.94 * \left(\frac{6992}{1000}\right)^{1.0645} * 0.8586 = 20.0086 \tag{2.10}$$

So, according to COCOMO model and our estimation, almost exactly 20 person-months are needed to fully develop myTaxiService system.

2.2.3 *Schedule estimation*

Now, thanks to the previous results, it is possible to evaluate the time to develop (**t**). Time to develop is the calendar time in months that goes from the determination of the product's requirements to the completion of an acceptance activity certifying that the product satisfies its requirements.

The time to develop is given by this formula:

$$\mathbf{t} = 3.67 * (\mathbf{e})^{F} \tag{2.11}$$

Exponent F is defined as follows:

$$F = 0.28 + 0.2 * (E - 0.91) \tag{2.12}$$

By substituting the results from previous sections in equations (2.11) and (2.12), we obtain:

$$F = 0.28 + 0.2 * (1.0645 - 0.91) = 0.3109 \tag{2.13}$$

$$\mathbf{t} = 3.67 * (20.0086)^{0.3109} = 9.3157 \text{ m.} \simeq 40 \text{ w.} \tag{2.14}$$

The development of myTaxiService as a whole is expected to take more than 9 months.

By dividing the effort **e** by the time to develop **t**, we get the estimated number of people needed for the project:

$$\mathbf{n} = \frac{\mathbf{e}}{\mathbf{t}} = \frac{20.0086}{9.3157} = 2.1478 \tag{2.15}$$

Some considerations about these figures will be done in the next chapter.

3
Project schedule and resource allocation

Following the estimations about cost, effort and time to develop myTaxiService, it is appropriate to define the scheduling of the development and implementation of the system.

Le us start with the phases we already have gone though. For each, denoted by the abbreviation of the relating final document, we provide the total number of hours we dedicated to it (H), the actual available time (AT) and an estimate (ET) of how much time we would have needed, in case of full-time commitment (2 people, working 8 hours/day each)[1].

[1] The elements in this column are computed as follows:

$$ET = \left\lceil \frac{H}{2 \text{ people}} * \frac{1}{8 \text{ hours}} \right\rceil$$

PHASE	H	AT	ET
`RASD`	61	3 weeks	$\simeq$ 4 days
`DD`	68	4 weeks	$\simeq$ 5 days
`ITPD`	10	2 weeks	$\simeq$ 1 day
`PPD`	16	2 weeks	$\simeq$ 1 day

Notice that it is nearly impossible for us to define a role for the two of us in the team, because we always worked together on the whole documents.

Obviously, the time in ET is an underestimation. If we had the chance to dedicate our full time to the project, many more aspects would have been analysed (e.g., on the physical architecture of the system, the issue of code reusing, the system and performance testing), which would have required more time.

Now we want to relate these results with the time to develop $\mathbf{t} = 40$ weeks, computed in equation (2.14). Following COCOMO definition, this parameter considers the span of time between the beginning of the design phase and the end of the integration testing. In particular, we can identify the following phases which naturally occur in the considered span of time:

1. Product planning and detailed design phase (`DD`, `ITPD`, and `PPD`);
2. Code and unit test phase.

3. Integration and test phase.

As one can notice, the first phase is now completed with the delivery of this document. This means that we now have the following number of weeks to develop, integrate and test the system:

$$\mathbf{t} - 2 * \left\lceil \frac{\text{EM}(\texttt{DD}) + \text{EM}(\texttt{ITPD}) + \text{EM}(\texttt{PPD})}{7} \right\rceil = 38 \qquad (3.1)$$

With the doubled ceiling operation we are computing the "corrected" number of weeks (considering the issue of the underestimation, mentioned above) needed for the planning and design phase.

As of the development, we identify some critical components, which need to be developed as early as possible. In particular we notice that `AreaManagement`, `RequestManagement`, and `RequestCreation` are fundamental to allow the development the other components of the system, so they have to be developed first (and unit tested, of course).

On the other hand, `ReservationCreation` and `ReservationManagement` are far less critical, as the first release of the software system may not even include this functionality.

However, we find it pointless to prescribe the allocation of coding tasks to each developer, so we are not going to. We only suggest to dedicate about one third of the time to the integration testing phase, as we believe it to be crucial for the quality of the system.

4 Risk report

In order to perform risk identification and analysis, we are adopting Sommerville's checklist[1], as we believe it to be a good balance of compactness, comprehensiveness and intuitiveness.

[1] Ian Sommerville. *Software Engineering*. 10th edition. Pearson, 2015.

There are six types of risk that may be considered, and they are analysed in the following sections.

4.1 Technology risks

These risks derive from the software and hardware technologies that are used to develop the system.

We can identify a number of them:

- The database cannot process as many transactions per second as expected. To overcome this problem, the possibility of buying a higher-performance database should be considered.
- Reusable software components need severe refactoring. In-depth analysis should be performed before using already written code, and only valid components should be adopted.
- Data corruption in the communication between clients and servers occurs more frequently than expected. Protocols to improve the reliability of the communications should be introduced.
- The system may suffer from outages, due to power outages, hardware failures. System redundancy may be of help, in order to contrast this kind of risk.

4.2 People risks

People risks are associated with the people in the development team.

Trivially, we have the following:

- Key staff are ill and unavailable at critical times: since our team is currently composed of two people, any absence would almost certainly cause delays, unless our work is reorganised so that there is more overlap, and thus we get more insight into each other's jobs.

4.3 *Organisational risks*

Risks deriving from the organisational environment where the software is being developed are grouped in this category.

The following is the main risks in this category:

- Financial difficulties due to a reduced income from the municipality; however the city council's commitment is strong, so reductions in the project budget are very unlikely.

4.4 *Tools risks*

This kind of risks derive from the software tools and other support software used to develop the system.

In the previous documents, limited specification was made about the tools to be used. No relevant risk relating to this category can be pointed out.

4.5 *Requirements risks*

Risks that derive from changes to the customer requirements fall into this category.

The two main requirements risks we identify are:

- Changes to requirements that require major design rework may be proposed. This is totally beyond our control. The only countermeasure is to deliver a detailed requirement analysis document and thoroughly discuss it with the municipality. Also, deriving traceability information to assess requirements changes may be of help.
- Legislation concerning taxi service and privacy may change in unexpected ways; hardly we can do something to predict the probability and the impact on our project.

4.6 *Estimation risks*

Here we collect risks that derive from the management estimates of the resources required to build the system.

Only one risk can be categorised here:

- This risk depends mainly on the accuracy of the estimations in chapter 2, and can lead to an insufficient money or time allocation for the development of the project. Thorough investigations should be performed to check the correctness of the values.

A
Factors justification

We are going to motivate here the values we adopted for scale factors and cost drivers (section 2.2), which we used to compute some relevant COCOMO parameters. We chose not to clutter the main section and to dedicate some space here to this justification.

Each parameter is thoroughly described in COCOMO model (see section 1.2 for reference), where factor tables are provided in their entirety. We are not going to present them here, because a theoretical description of the model goes far beyond the scope of this document.

A.1 Scale factors

`PREC` It reflects the previous experience we had with this kind of projects. Since this was our first experience with this framework and methodologies, we set this value to *Low*.

`FLEX` It conveys the degree of flexibility in the development process. Since the high level description was pretty general, this value is set to *High*.

`RESL` It measures the degree of uncertainty and risk in the development process. It is set to *Nominal* because a great portion of time was dedicated to designing the system, but some aspects (e.g., user interface, COTS[1], hardware) are still partially undefined.

[1] A commercial-off-the-shelf (COTS) product is a software system, available in the commercial marketplace, that can be bought and adapted to the needs of different customers. COTS purchases are a form of software reuse, and constitute alternatives to custom developments.

`TEAM` It accounts for the sources of project turbulence and entropy because of difficulties in synchronising the stakeholders (users, customers, developers, maintainers, others). In our case, given the limited number of people involved in the project and the great mutual trust between the two of us, the parameter is set to *Very high*.

`PMAT` It evaluates the maturity of the project at a certain point. This was determined by means of the Key Process Area (KPA) Questionnaire, which gave a result of 2.87: this is why this parameter is set to *High* (level 3).

A.2 Cost drivers

`RELY` This is the measure of the extent to which the software must perform its intended function over a period of time. Since software failures don't have critical consequences, the parameter is set to *Low*.

`DATA` This cost driver attempts to capture the effect large test data requirements have on product development. The rating is determined by calculating the ratio of bytes in the testing database to SLOC in the program. At this stage there is no testing database, but considering the size of the project, it is possible to set this parameter to *Nominal*.

`CPLX` Complexity is divided into five areas: control operations, computational operations, device-dependent operations, data management operations, and user interface management operations. After considering the size and nature of myTaxiService system, we set this parameter to *High*.

`RUSE` This cost driver accounts for the additional effort needed to construct components intended for reuse on current or future projects. This effort is consumed with creating more generic design of software, more elaborate documentation, and more extensive testing to ensure components are ready for external use. Since we require extensive documentation and thorough testing, it is reasonable to set this parameter to *High*.

`DOCU` The rating scale for the DOCU cost driver is evaluated in terms of the suitability of the project's documentation to its lifecycle needs. We believe that documentation is crucial in any project, therefore we set this parameter to *High*.

`TIME` This is a measure of the execution time constraint imposed upon a software system. In our project this parameter is *not relevant*.

`STOR` This rating represents the degree of main storage constraint imposed on a software system or subsystem. In our project this parameter is *not relevant*.

`PVOL` The term *platform* is used here to mean the complex of hardware and software (OS, DBMS, etc.) the software product calls on to perform its tasks. Since the platform shouldn't change too often, this value is set to *Low*.

`ACAP` Analysts are personnel who work on requirements and 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. This parameter is set to *High*, since we dedicated a great effort in analysing the problem requirements and its potential integration in the real world, and also in designing the whole system.

`PCAP` Evaluation should be 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. Should development be upon us, we set this parameter to *High*.

`PCON` The rating scale for this factor is in terms of the project's annual personnel turnover. This parameter is set to *Very low*, since in our case the available time is less than half a year.

`APEX` The rating for this cost driver depends on the level of applications experience of the project team developing the software system. The ratings are defined in terms of the project team's equivalent level of experience with this type of application. Considering our education, we can set this parameter to *Nominal*.

`PLEX` This parameter recognises the importance of understanding the use of more powerful platforms, including more graphic user interface, database, networking, and distributed middleware capabilities. Since our knowledge about databases, user interfaces, and server-side development is around one year, this parameter is set to *Nominal*.

`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. In addition to experience in the project's programming language, experience on the project's supporting tool set also affects development effort. Considering our education, we can set this parameter to *Nominal*.

`TOOL` This parameter evaluates the use of tools to support the development and testing. Since we set no mandatory prescription to the developers, we set this parameter to *Nominal*.

`SITE` Given the increasing frequency of multisite developments, and indications that multisite development effects are significant, this parameter measures the impact on the development process of site collocation and communication support. Since we met daily and used phone calls and messaging applications to communicate, we set this parameter to *Extra high*.

`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. In spite of the well defined deadlines, which facilitated the distribution of our efforts over the time, some phases required much work. For this reason this parameter is set to *High*.

Appendix

Hours of work

The writing of this document took the following amount of time:

Paolo Antonini	10 hours.
Andrea Corneo	6 hours.