Project Plan

for

Taxi Management System

Version 1.0 - 01/02/2016

Prepared by Giuseppe Di Francesco & Domenico Iezzi

Sommario

1. Function Points

A function point is a "unit of measurement" to express the amount of business functionality an information system (as a product) provides to a user. Function points measure software size, based on some program characteristics such as external I/O, user interaction, external interfaces. For the FP weight count, we used the following tables extracted from the COCOMO II Model Definition Manual:

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

1.1 External Interface File

On the client side, application requires loading map data from OpenStreetMap service in order to show map for current zone to user. Once the show map page is open, application retrieves the position for the current user and send it to the server as a JSON variable; the server will respond to this request with the zone of the current position, so the page will start loading the OpenStreetMaps frame showing the current zone of the city.
This can be considered as a low-weighted operation.

EIF	**Complexity**	**FP**
OpenStreetMaps	Low	5
		Total: 5

1.2 Internal Logic Files

After receiving reservation from users, application will store into Database the time, date and information about user, so it will be able to forward a request when the time for reservation comes. It is able to store received requests, both for logging and recovering purpose. On the driver side, once the system receives a change status request from a driver, it will modify the current status value on the database.

System administrator, in his control panel is able to add or delete new drivers, and modify existing information.

ILF	**Complexity**	**FP**
Reservation	Low	7
Request	Low	7
Driver management	Low	7
		Total: 21

1.3 External Input

All of the server requests are sent by clients, which can be divided into two categories: normal users and taxi drivers. Every request in this category is sent to the server and then handled asynchronously. The server will elaborate the information and notify back the user or driver. In particular we have:

- **Reservation:** from the user which will be saved into the DB, and then forwarded as a normal request when the time of the reservation is about to come. The request can be considered as low-weighted because most of the work is done asynchronously (work that is considered as an external output), and the data sent is a JSON object containing *source position, destination position* as GPS Coordinates, *time* and user's *mail*.

- **Request:** same as before, request is saved into the DB and then handled async. Data sent is a JSON object containing *position* as Gps coordinates. We consider this kind of request low-weighted.

- **Change Status:** from the driver it is handled by the driver manager component, saving the status to the db and managing the queue where the driver is listed in order to add or remove him. We consider this kind of request low-weighted.

- **Login:** simple authentication request for the driver. Data sent to the server will be *username* and hashed *password.* We consider this kind of request low-weighted.

- **Driver Accept:** when the system is handling a request, it will call each available taxi to forward this request. This is a simple socket emit event which will get the response from the taxi Driver, so it can be considered low-weighted.

External Input	Complexity	FP
Reservation	Low	3
Request	Low	3
Change status	Low	3
Login	Low	3
Driver Accept	Low	3
		Total: 15

1.4 External Inquiries

After the user request the map, client application will send a request to the system for taxi positions in the current zone. To do so it will request through the driver manager component the taxi positions and return them to the user as a JSON object containing taxi ID and GPS position, which will be drawn into the loaded map by the client application.
We considered this operation as medium-weighted because it involves lots of entities.

External Inquiries	Complexity	FP
Show Taxi Positions	Avg.	4
		Total: 4

1.5 External Output

Client request are handled asynchronously by the server. These are the most complex functions of the system, because the data involved and the computations are multiple. We will consider them in more details.

- **Request:** after a user sends a request with the position, the request manager gets the queue object for the zone relative to the specified position, and it will send a notification to each taxi in the queue until one of them accepts. As soon as a taxi accepts request, the system will notify user about it. Data involved are the *queue* which contains available taxi informations as well as *socket* instance, *request data* from previous request. Since entities involved are multiple, we consider this as a high-weighted function.

- **Reservation handling:** after receiving a reservation, the reservation manager will schedule a request for ten minutes to the reservation time. Data involved are as the Reservation case in the external input, but entities involved are reservation manager and the request manager. We consider for this the medium weight.

- **Notification:** system is able to notify user's client about pending request, and driver's client about request. In this case the data involved is very small, because they are simple messages between client and server, and entities involved are clients and request manager. We will use the low weight.

External Output	**Complexity**	**FP**
Request	High	7
Reservation handling	Avg.	5
Driver management	Low	4
		Total: 16

1.6 Final considerations

By summing up previously obtained FPs we have a total of 61 FP. If we consider the table provided by QSM (link in the final chapter) with an average conversion ratio for JavaScript of 47, we have an estimated number of SLOC of **2867**.

Function type	**Value**
ILF	21
EIF	5
External Input	15
External Output	16
External Inquiries	4
Total UFP	**61**
SLOC	61 x 47 = **2867**

In order to calculate the adjusted function points, we need to find the TDI (Total Degree of Influence of the 14 General system characteristics), considering these general characteristics (each one with a degree of 0-5):

- Data Communication -> 4
- Distributed Data Processing -> 0
- Performance -> 5
- Heavily Used Configuration -> 1
- Transaction Role -> 0
- Online Data Entry -> 3
- End-User Efficiency -> 3
- Online Update -> 3
- Complex Processing -> 4
- Reusability -> 1
- Installation Ease -> 5
- Operational Ease -> 5
- Multiple Sites -> 1
- Facilitate Change -> 3

Using the following formula, we can retrieve the Adjusted FP for our project.

$$FP = UFP \times \left(0.65 + 0.01 \times \sum_{i=1}^{14} F_i \right)$$

FP = 58 x [0.65 + 0.01 x (38)] = **60**
LOC = 60 x 47 = 2820

The variation from UFP to FP is small, probably due to some functionalities and additional requirements not described by the used classification.

2. Cocomo Estimation

The Constructive Cost Model (COCOMO) is an algorithmic software cost estimation model developed by Barry W. Boehm. The model is based on the statistical analysis with parameters that are derived from historical project data and current as well as future project characteristics. COCOMO can be used to:

- Making investment or other financial decisions involving a software development effort
- Setting project budgets and schedules as a basis for planning and control
- Deciding on or negotiating tradeoffs among software cost, schedule, functionality, performance or quality factors
- Making software cost and schedule risk management decisions
- Deciding which parts of a software system to develop, reuse, lease, or purchase
- Making legacy software inventory decisions: what parts to modify, phase out, outsource, etc
- Setting mixed investment strategies to improve organization's software capability, via reuse, tools, process maturity, outsourcing, etc
- Deciding how to implement a process improvement strategy, such as that provided in the SEI CMM

2.1 Scale factor

The exponent in the equation used by the COCOMO model 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. The factor are:

Precedentedness	Reflects the previous experience of the organisation with this type of project. Very low means no previous experience, Extra high means that the organisation is completely familiar with this application domain.
Development flexibility	Reflects the degree of flexibility in the development process. Very low means a prescribed process is used; Extra high means that the client only sets general goals.
Architecture/risk resolution	Reflects the extent of risk analysis carried out. Very low means little analysis, Extra high means a complete a thorough risk analysis.
Team cohesion	Reflects how well the development team know each other and work together. Very low means very difficult interactions, Extra high means an integrated and effective team with no communication problems.
Process maturity	Reflects the process maturity of the organisation. The computation of this value depends on the CMM Maturity Questionnaire but an estimate can be achieved by subtracting the CMM process maturity level from 5.

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

Using the above table, we defined our scale factor as follows:

Scale Driver	Factor	Value
Precedentedness	Low	4.96
Development flexibility	High	2.03
Risk resolution	High	2.83
Team cohesion	Very high	1.10
Process maturity	Nominal	4.68
		Total: 15.6

2.2 Cost Drivers: Product Factors

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. Table 17 provides the COCOMOII.2000 rating scheme for RELY.

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 (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 in the program. The reason the size of the database is important to consider is because of the effort required to generate the test data that will be used to exercise the program. In other words, DATA is capturing the effort needed to assemble and maintain the data required to complete test of the program through IOC, see Table 18.

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

* DATA is rated as Low if D/P is less than 10 and it is very high if it is greater than 1000. P is measured in equivalent source lines of code (SLOC), which may involve function point or reuse conversions.

Product Complexity

Complexity is divided into five areas: control operations, computational operations, device-dependent operations, data management operations, and user interface management operations. Using Table 19, select the area or combination of areas that characterize the product or the component of the product you are rating. The complexity rating is the subjective weighted average of the selected area ratings. Table 20 provides the COCOMO II.2000 effort multipliers for CPLX.

Table 19. Component Complexity Ratings Levels

	Control Operations	Computational Operations	Device-dependent Operations	Data Management Operations	User Interface Management Operations
Very Low	Straight-line code with a few non-nested structured programming operators: DOs, CASEs, IF-THEN-ELSEs. Simple module composition via procedure calls or simple scripts.	Evaluation of simple expressions: e.g., A=B+C*(D-E)	Simple read, write statements with simple formats.	Simple arrays in main memory. Simple COTS-DB queries, updates.	Simple input forms, report generators.
Low	Straightforward nesting of structured programming operators. Mostly simple predicates	Evaluation of moderate-level expressions: e.g., D=SQRT(B**2-4.*A*C)	No cognizance needed of particular processor or I/O device characteristics. I/O done at GET/PUT level.	Single file subsetting with no data structure changes, no edits, no intermediate files. Moderately complex COTS-DB queries, updates.	Use of simple graphic user interface (GUI) builders.
Nominal	Mostly simple nesting. Some intermodule control. Decision tables. Simple callbacks or message passing, including middleware-supported distributed processing	Use of standard math and statistical routines. Basic matrix/vector operations.	I/O processing includes device selection, status checking and error processing.	Multi-file input and single file output. Simple structural changes, simple edits. Complex COTS-DB queries, updates.	Simple use of widget set.

Table 19. Component Complexity Ratings Levels

	Control Operations	Computational Operations	Device-dependent Operations	Data Management Operations	User Interface Management Operations
High	Highly nested structured programming operators with many compound predicates. Queue and stack control. Homogeneous, distributed processing. Single processor soft real-time control.	Basic numerical analysis: multivariate interpolation, ordinary differential equations. Basic truncation, round-off concerns.	Operations at physical I/O level (physical storage address translations; seeks, reads, etc.). Optimized I/O overlap.	Simple triggers activated by data stream contents. Complex data restructuring.	Widget set development and extension. Simple voice I/O, multimedia.
Very High	Reentrant and recursive coding. Fixed-priority interrupt handling. Task synchronization, complex callbacks, heterogeneous distributed processing. Single-processor hard real-time control.	Difficult but structured numerical analysis: near-singular matrix equations, partial differential equations. Simple parallelization.	Routines for interrupt diagnosis, servicing, masking. Communication line handling. Performance-intensive embedded systems.	Distributed database coordination. Complex triggers. Search optimization.	Moderately complex 2D/3D, dynamic graphics, multimedia.
Extra High	Multiple resource scheduling with dynamically changing priorities. Microcode-level control. Distributed hard real-time control.	Difficult and unstructured numerical analysis: highly accurate analysis of noisy, stochastic data. Complex parallelization.	Device timing-dependent coding, micro-programmed operations. Performance-critical embedded systems.	Highly coupled, dynamic relational and object structures. Natural language data management.	Complex multimedia, virtual reality, natural language interface.

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 (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

use in other applications. “Across project” could apply to reuse across the modules in a single financial applications project. “Across program” could apply to reuse across multiple financial applications projects for a single organization. “Across product line” could apply if the reuse is extended across multiple organizations. “Across multiple product lines” could apply to reuse across financial, sales, and marketing product lines, see Table 21. Development for reusability imposes constraints on the project's RELY and DOCU ratings. The RELY rating should be at most one level below the RUSE rating. The DOCU rating should be at least Nominal for Nominal and High RUSE ratings, and at least High for Very High and Extra High RUSE ratings.

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 (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. The rating scale goes from Very Low (many life-cycle needs uncovered) to Very High (very excessive for life-cycle needs), see Table 22. Attempting to save costs via Very Low or Low documentation levels will generally incur extra costs during the maintenance portion of the life-cycle. Poor or missing documentation will increase the Software Understanding (SU) increment discussed in Section 2.4.2.

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

2.3 Cost Drivers: Platform Factors

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 consuming 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, see Table 23.

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 (STOR)

This rating represents the degree of main storage constraint imposed on a software
system or subsystem. Given the remarkable increase in available processor execution time and main storage, one can question whether these constraint variables are still relevant. However, many applications continue to expand to consume whatever resources are available---particularly with large and growing COTS products---making these cost drivers still relevant. The rating ranges from nominal (less than 50%), to extra high (95%) see Table 24.

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 (PVOL)

"Platform" is used here to mean the complex of hardware and software (OS, DBMS, etc.)
the software product calls on to perform its tasks. If the software to be developed is an operating system then the platform is the computer hardware. If a database management system is to be developed then the platform is the hardware and the operating system. If a network text browser is to be developed then the platform is the network, computer hardware, the operating system, and the distributed information repositories. The platform includes any compilers or assemblers supporting the development of the software system. 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, see Table 25.

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

2.4 Cost Drivers: Personnel factors

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. The rating should not consider the level of experience of the analyst; that is rated with APEX, LTEX, and PLEX. Analyst teams that fall in the fifteenth percentile are rated very low and those that fall in the ninetieth percentile are rated as very high, see Table 26.

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 (PCAP)

Current trends continue to emphasize the importance of highly capable analysts. However the increasing role of complex COTS packages, and the significant productivity leverage associated with programmers' ability to deal with these COTS packages, indicates a trend toward higher importance of programmer capability as well.

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. The experience of the programmer should not be considered here; it is rated with APEX, LTEX, and PLEX. A very low rated programmer team is in the fifteenth percentile and a very high rated programmer team is in the ninetieth percentile, see Table 27.

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 (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, see Table 28.

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 (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. A very low rating is for application experience of less than 2 months. A very high rating is for experience of 6 years or more, see Table 29.

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 (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, see Table 30.

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 (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. A low rating is given for experience of less than 2 months. A very high rating is given for experience of 6 or more years.

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	

2.5 Cost Drivers: Project Factors

Use of Software Tools (TOOL)

Software tools have improved significantly since the 1970s' projects used to calibrate the
1981 version of COCOMO. The tool rating ranges from simple edit and code, very low, to
integrated life-cycle management tools, very high. A Nominal TOOL rating in COCOMO 81 is equivalent to a Very Low TOOL rating in COCOMO II. An emerging extension of COCOMO II is in the process of elaborating the TOOL rating scale and breaking out the effects of TOOL capability, maturity, and integration, see Table 32.

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 (SITE)

Given the increasing frequency of multisite developments, and indications that multisite
development effects are significant, the SITE cost driver has been added in COCOMO II.
Determining its cost driver rating involves the assessment and judgement-based averaging of two factors: site collocation (from fully collocated to international distribution) and communication support (from surface mail and some phone access to full interactive multimedia). For example, if a team is fully collocated, it doesn't need interactive multimedia to achieve an Extra High rating. Narrowband e-mail would usually be sufficient, see Table 33.

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 communication.	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 (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. Accelerated schedules tend to produce more effort in the earlier phases to eliminate risks and refine the architecture, more effort in the later phases to accomplish more testing and documentation in parallel. In Table 34, schedule compression of 75% is rated very low. A schedule stretch-out of 160% is rated very high. Stretch-outs do not add or decrease effort.

Their savings because of smaller team size are generally balanced by the need to carry project administrative functions over a longer period of time. The nature of this balance is undergoing further research in concert with our emerging CORADMO extension to address rapid application development. SCED is the only cost driver that is used to describe the effect of schedule compression/expansion for the whole project. The scale factors are also used to describe the whole project. All of the other cost drivers are used to describe each module in a multiple module project. Using the COCOMO II Post-Architecture model for multiple module estimation is explained in Section 3.3.

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

2.6 Cost Estimation Results

For our project, we considered those values of the previous factors:

Driver	**Factor**	**Value**
RELY	Low	0.92
DATA	Nominal	1.00
CPLX	Nominal	1.00
RUSE	High	1.07
DOCU	Nominal	1.00
TIME	High	1.11
STOR	Nominal	1.00
PVOL	Low	0.87
ACAP	Nominal	1.00
PCAP	High	0.88
PCON	Low	1.12
APEX	Very Low	1.22
PLEX	Nominal	1.00
LTEX	High	0.91
TOOL	Low	1.09
SITE	High	0.93
SCED	Nominal	1.00
Product:		**1.05**

Given these values, we can now estimate the amount of effort and calendar time it will take to develop this software project. The amount of effort in person-months, PM, is estimated by the formula:

$$PM_{NS} = A \times Size^{E} \times \prod_{i=1}^{n} EM_i$$

$$\text{where } E = B + 0.01 \times \sum_{j=1}^{5} SF_j$$

The amount of calendar time, TDEV, is estimated by the formula:

$$TDEV_{NS} = C \times (PM_{NS})^{F}$$

$$\text{where } F = D + 0.2 \times 0.01 \times \sum_{j=1}^{5} SF_j$$

$$= D + 0.2 \times (E - B)$$

Values from A, B, C and D for the COCOMOII.2000 calibration are:

A = 2.94	B = 0.91
C = 3.67	D = 0.28

So in this case given:

$$EM = 1.05$$
$$E = 0.91 + (0.01 \cdot 15.6)$$
$$KSLOC = 2.867$$
$$Effort = 2.94 \cdot 1.05 \cdot 2.91 = 9.48$$

And for the calendar time:

$$F = 0.28 + (0.2 \cdot 0.01 \cdot 15.6) = 0.31$$

$$TDEV = 3.67 \cdot 2 = 7.25$$

If we subtract months to PM we obtain number of person, that is 1.3 ~= 1 person.

In order to have a more view on our estimation, we also use the COCOMOII calculator offered by the USC Center for Systems and Software Engineering using the same values it returned slightly different values:

Software Size Sizing Method: Function Points

Unadjusted Function Points: 61 Language: 3rd Generation Language

Software Scale Drivers

Precedentedness	Low	Architecture / Risk Resolution	High	Process Maturity	Nominal
Development Flexibility	High	Team Cohesion	Very High		

Software Cost Drivers

Product		Personnel		Platform	
Required Software Reliability	Low	Analyst Capability	Nominal	Time Constraint	High
Data Base Size	Nominal	Programmer Capability	High	Storage Constraint	Nominal
Product Complexity	Nominal	Personnel Continuity	Low	Platform Volatility	Low
Developed for Reusability	High	Application Experience	Very Low	**Project**	
Documentation Match to Lifecycle Needs	Nominal	Platform Experience	Nominal	Use of Software Tools	Low
		Language and Toolset Experience	High	Multisite Development	High
				Required Development Schedule	Nominal

Maintenance Off

Software Labor Rates

Cost per Person-Month (Dollars): 2000

Calculate

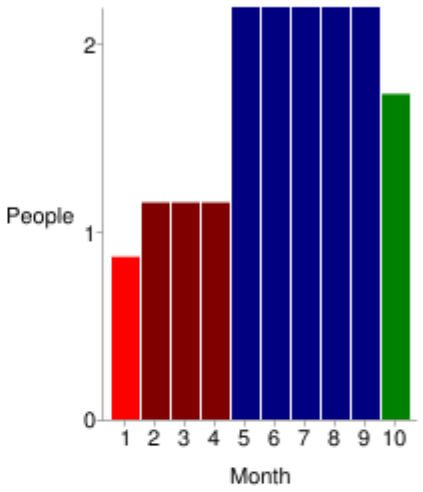
Results

Software Development (Elaboration and Construction)

Effort = 16.8 Person-months
Schedule = 9.3 Months
Cost = $33593

Total Equivalent Size = 4880 SLOC

Staffing Profile



Acquisition Phase Distribution

Phase	Effort (Person-months)	Schedule (Months)	Average Staff	Cost (Dollars)
Inception	1.0	1.2	0.9	$2016
Elaboration	4.0	3.5	1.2	$8062
Construction	12.8	5.8	2.2	$25531
Transition	2.0	1.2	1.7	$4031

Software Effort Distribution for RUP/MBASE (Person-Months)

Phase/Activity	Inception	Elaboration	Construction	Transition
Management	0.1	0.5	1.3	0.3
Environment/CM	0.1	0.3	0.6	0.1
Requirements	0.4	0.7	1.0	0.1
Design	0.2	1.5	2.0	0.1
Implementation	0.1	0.5	4.3	0.4
Assessment	0.1	0.4	3.1	0.5
Deployment	0.0	0.1	0.4	0.6

From this calculations, we obtained an effort of 16.8 person/month, a schedule of 9.3 months and an equivalent size of 4880 SLOC, which are higher compared to our values. By doing 16.8 / 9.3 we get 1.8 person ~= 2 people.

There could be some reasons for the difference between values. First of all it is possible that there are some errors in our calculations, or badly approximated values, or wrong complexity on some factor due to inexperience in project planning. Moreover, some Cocomo constants may be not addressed to our particular case, resulting in different results.

3. Task Allocation

Here will be defined the main task of this project, along with the time needed for each one. We will provide also a scheduling chart which will explain how we organized for the development of this software. These activities started on October 2015 and ended on the beginning of February 2016. Each task was planned considering the team is formed by two members.

Task	**Length**
Research	1 week
Project Plan	1 week
Requirement Analysis	4 weeks
Design	2 weeks
Implementation	5 weeks
Code Inspection	1 week
Integration Test	1 week

Task	Week														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Research															
Project Plan															
Requirement Analysis															
Design															
Implementation															
Code Inspection															
Integration Test															

While allocating resources, we considered that the amount of work for each group member would be almost the same for every task, due to the fact that this project is a small group work. We considered an estimation of the work hours for each task for each member.

Planning & Research

- ❖ Research
  - ➢ Domenico Iezzi: ~10h
  - ➢ Giuseppe Di Francesco: ~10h

Project Plan

- ❖ Project Plan Document
  - ➢ Domenico Iezzi: ~14h
  - ➢ Giuseppe Di Francesco: ~14h

Requirement Analysis

- ❖ Requirement Analysis and Specification Document
  - ➢ Domenico Iezzi: ~50h
  - ➢ Giuseppe Di Francesco: ~50h

Design

- ❖ Design Document
  - ➢ Domenico Iezzi: ~30h
  - ➢ Giuseppe Di Francesco: ~30h

Implementation

- ❖ Implementation
  - ➢ Domenico Iezzi: ~100h
  - ➢ Giuseppe Di Francesco: ~100h

Code Inspection

- ❖ Code Inspection Document
  - ➢ Domenico Iezzi: ~15h
  - ➢ Giuseppe Di Francesco: ~15h

Testing

- ❖ Integration Test Plan Document
  - ➢ Domenico Iezzi: ~12h
  - ➢ Giuseppe Di Francesco: ~12h

4. Risk management

In this section we will describe main risks we could encounter during the development of this project, analyzing also their impact and probability in order to define possible solutions or contingency plan.

Project schedule delays. Considering that the team is composed by two people, it could be possible that some tasks in the schedule will be delayed due to the fact that each group member has other commitments, or due to having found some problems in the specification or the implementation that needed a rewrite of some key parts of it.
To deal with this kind of risks, we decided to pay more attention to the requirement analysis and design phases, increasing number of days for each of these tasks in the schedule. This increment would give us more time to work on the specification of the project, and have, before the beginning of the development phase, an accurate documentation and a robust design.

Specification breakdown. It could be possible, during some later phases, that we find some requirements that we didn't considered in the requirement analysis, and is needed for a correct design of the software, or even some wrong requirements that needs to be fixed. In order to prevent this risk, during the planning-requirement phase we tried to involve stakeholders and ask them information. We also plan to do some research for similar projects, and also for taxi companies and their rules, laws.

Poor Productivity. Given that each team member has other commitments during the academic year, there is the possibility that the productivity will not be at its highest during the project time. In order to have the best productivity, we decided to make weekly deadlines for our work and keep track of the work already done. If the deadlines are not met, we would organize the following work in order to recover the lost hours, and try to optimize the work hours.

5. Links & References

Cocomo:

http://csse.usc.edu/tools/COCOMOII.php

http://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_modelman2000.0.pdf

http://csse.usc.edu/csse/research/COCOMOII/cocomo_main.html

Function Points:

http://www.codeproject.com/Articles/18024/Calculating-Function-Points

http://www.qsm.com/resources/function-point-languages-table

Course Slides and Project Plan Examples