Politecnico di Milano

Software Engineering 2 Project

MeteoCal

Project Reporting Document

Authors:
Andrea Celli
Stefano Cereda

February 10, 2015

Contents

I Introduction **2**

1 Size of the project **2**

2 Overall project time **2**

II Function point **3**

III COCOMO II **6**

3 Exponent **6**

4 Effort Adjustment Factor **8**

5 Effort computation **10**

6 Schedule estimation **11**

7 Schedule comparison **11**

Part I
Introduction

In the following document we analyze our project using two different approaches. In the first part we apply the Function Point approach in order to check whether the result is similar to the actual size of the project. In the second part we apply COCOMO formulas and compare the results with the the "real" effort spent to develop the project.

1 Size of the project

To perform the mentioned analysis we need to know the actual size of the project. In order to compute the number of lines of code (LOC) we used a plugin for NetBeans called Simple Code Metrics. Using this tool we computed an actual size of 4372 LOC. This number does not take into account the xhtml pages. Obviously we didn't take into consideration the lines of comments and testdrivers.

2 Overall project time

COCOMO II provides an estimation of the overall effort needed to develop the project. We want to make a comparison between the estimation and the actual time needed by the team to accomplish the task. In order to do so we kept track of the time each member of the team spent on the project.

1. $T_{RASD} = 30h + 30h = 60h$
2. $T_{DD} = 30h + 35h = 65h$
3. $T_{DEV} = 150h + 150h = 300h$

The overall time spent by the two members of the team on the project turns out to be:
$T_{TOT} = T_{RASD} + T_{DD} + T_{DEV} = 60h + 65h + 300h = 425h$

Part II
Function point

In order to determine the number of LOC (lines of code) of the project we first have to find out the number of Functions Points (FP) that have to be addressed.

1. Internal Logical Files (ILF)

   (a) The system has to store information about the user and his/her participations.

   Weight: *medium*

   The complexity is medium for the following reasons:

   - The "users" table contains several fields
   - The system has to manage security settings for the account (i.e. passwords have to be encrypted and stored securely)
   - A user calendar may contain a high number of events

   (b) The system has to store events

   Weight: *medium*

   The medium weight has been chosen because the "event" table has to contain several different fields.

   (c) The system has to store notification

   Weight: *simple*

   The simple weight has been chosen because notifications are quite simple entities (with a small number of parameters).

2. External Interface Files (EIF)

   (a) The system has to retrieve and store forecast's data

   Weight: *complex*

   The weight is complex because the management of external forecasts' data requires parsing information coming from the external service and the handling of two different kind of forecasts: daily and 3 hours.

   (b) The system has to store a list of available places and their information

   Weight: *simple*

   The system has just to download the list of places at the startup of the server.

3. External Inputs (EI)

   (a) Login and Logout

   Weight: *simple*

   These operations are very straightforward. They only involve checking the correspondence between username and password and eventually shutting down the session.

(b) Registration

Weight: *simple*

These operation consists only in inserting a new tuple in the user table while performing some basic checks on the input values.

(c) Change personal settings

Weight: *simple*

These operation consists only in updating a tuple in the user table while performing some basic checks on the input values.

(d) Event creation, modification

Weight: *complex*

These operations involves the insert/update of a tuple in the event table. Quite complex issues arise because the system has to take care of different roles of users (i.e. When the creator of an event modifies it all the participants has to be informed of that and their calendar have to be updated)

(e) Search users

Weight: *simple*

The system has just to perform some queries (addressing the user table) to retrieve the list of users corresponding to the specified search key.

(f) Answer to notifications

Weight: *complex*

After the user answers a notification the system has to take care of the update of several different tables.

4. External Inquiries (EI)

(a) Show "about page"

Weight: *simple*

The system has only to display a description of MeteoCal in order to inform a new user of its functionalities.

(b) Browse personal calendar

Weight: *simple*

The system retrieves events in which the user will take part and displays them.

(c) Browse external calendars Weight: *medium*

The system has to take care of privacy settings for both users' calendar and their events. It has to display the "external" calendar and its events according to the user's settings.

5. External Outputs (EO)

(a) Display weather forecasts

Weight: *complex*

The system has to manage the interaction with the external service providing the forecasts. Moreover it has to periodically update them.

(b) Display notifications

Weight: *complex*

The system has to detect various types of changes in the events details and generate notifications for the interested users accordingly. Thus several tables of the database are involved.

To calculate the LOC estimate we will use the following table of weights:

Function types	**Weights**		
	Simple	Medium	Complex
N.Inputs	3	4	6
N.Outputs	4	5	7
N.Inquiry	3	4	6
N.ILF	7	10	15
N.EIF	5	7	10

The Unadjust Function Point (UFP) is the result of the following contributes:

- **ILF** $= 1 * simple + 2 * medium = 7 + 10 + 10 = 27$
- **EIF** $= 1 * simple + 1 * complex = 5 + 10 = 15$
- **EI** $= 4 * simple + 2 * complex = 4 * 3 + 2 * 6 = 24$
- **EIQ** $= 2 * simple + 1 * medium = 2 * 3 + 1 * 4 = 10$
- **EO** $= 2 * complex = 2 * 7 = 14$

UFP $= 27 + 15 + 24 + 0 + 14 = 90 FPs$

In order to obtain the estimation of the lines of code we have to use the following simple formula:

$$LOC = AVG * UFP$$

Where AVG is a language dependent factor. We use the AVG specific for J2EE, which is the programming language mainly used to develop the project. The coefficient was taken from: `http://www.qsm.com/resources/function-point-languages-table` (we used the value in the Avg column)

The final estimation turns out to be:

$$LOC = 46 * 90 = 4140$$

Given an actual size of 4372 LOC the estimation turns out to be really precise. There is a difference of just 232 lines, which is the 5,3% of the total real size.

Part III
COCOMO II

(For this analysis we referred to the "COCOMO II Model Definition Manual" (version 2.1) available at: `http://csse.usc.edu/csse/research/COCOMOII/cocomo_downloads.html`)

In COCOMO II effort is expressed as Person-Months (PM). The effort is computed as follows:

$$PM = A * Size^E * \prod_{i=1}^{n} EM_i$$

where $A = 2.94$ (for COCOMO II.2000

The Size is expressed in KSLOC. This is derived from estimating the size of software modules that will constitute the application program.

The way in which the exponent E and the effort multipliers (EM) are calculated and their meaning are explained in the following sections.

3 Exponent

The exponent E 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:

- If $E < 1.0$, the project exhibits economies of scale.
- If $E = 1.0$, the economies and diseconomies of scale are in balance.
- If $E > 1.0$, the project exhibits diseconomies of scale.

The exponent E is calculated as follows:

$$E = B + 0.01 * \sum_{j=1}^{5} SF_j$$

where $B = 0.91$ (for COCOMO II.2000)

Each scale factors has a range of rating levels, from Very Low to Extra High. Each rating level has a weight. The specific value of the weight is called a scale factor (SF). The project's scale factors, the selected scale factors ratings, are summed and used to determine a scale exponent.

We used the following table to retrieve weights:

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

This is the list of appropriate scale factors rating levels for our project:

1. Precedentedness (PREC)

   Rating level: *nominal*

   The team never developed a web-based project and did not have any experience with the J2EE environment, but had some previous experience on Java developing.

2. Development Flexibility (FLEX)

   rating Level: *high*

   The only constraints we had were on the language to be used and on functional requirement that the system had to fulfill.

3. Architecture / Risk Resolution (RESL)

   Rating level: *nominal*

   We devoted quite a lot of time to establish the system architecture given the general product objective. Nevertheless it was difficult to predict risks since the environment was new and we couldn't rely on previous experience.

4. Team Cohesion (TEAM)

   Rating level: *very high*

   Stakeholders had shared objectives and were committed to reach them. The team had already worked on other projects and so it was rather easy to organize the work. We only had some minor "communication issues", for this reason the lever is *very high* and not *extra high*.

5. Process Maturity (PMAT)

   Rating level: *nominal*

   We chose level 2 because processes have been planned and documented at the project level.

Having chosen the rating levels for each scale factor we can now calculate the exponent:

$$E = 0.91 + 0.01 * (3.72 + 2.03 + 4.24 + 1.1 + 4.68) = 1.0677$$

The exponent shows that economies and dis-economies of scales are almost in balance. This is mainly due to the fact that, despite the lack of previous experience, the team was small and easy to organize.

4 Effort Adjustment Factor

The seventeen effort multipliers (EM) are used in the COCOMO II model to adjust the nominal effort, Person-Months, to reflect the software product under development.

Each multiplicative cost driver is defined below by a rating level and a corresponding effort multiplier (the values are taken from the document referenced at the beginning of this part).

1. Required Software Reliability (RELY)

   Rating level: *low*

   Effort multiplier: 0.92

   Software failures give rise to easily recoverable losses. There are neither financial losses nor risk for human life.

2. Data Base Size (DATA) Rating level: *low*

   Effort multiplier: 0,9

   We tested the system without the need of huge testing data sets.

3. Product Complexity (CPLX) Rating level: *low*

   Effort multiplier: 0,87

   Considering all the different parts of the system (including the gui and the data management system) the overall complexity was modest if compared with industry standards.

4. Developed for Reusability (RUSE)

   Rating level: *nominal*

   Effort multiplier: 1,00

   The reusability focus was across project. We designed project's components to be easily reusable.

5. Documentation match to lifecycle needs (DOCU)

   Rating level: *nominal*

   Effort multiplier: 1,00

   The documentation covers the basic needs of the project. The requirements analysis and the design document are complete and detailed.

6. Execution Time Constraint (TIME)

   Rating level: *nominal*

   Effort multiplier: 1,00

   The system didn't require high execution time to perform complex tasks.

7. Main Storage Constraint (STOR)

   Rating level: *nominal*

   Effort multiplier: 1,00

   The system used less (way less) of the 50% of available storage space.

8. Platform Volatility (PVOL)

   Rating level: *low*

   Effort multiplier: 0,87

   The development environment and the used tools didn't change significantly during our work.

9. Analyst Capability (ACAP)

   Rating level: *high*

   Effort multiplier: 0,85

   The team had good design ability and was able to efficiently cooperate in the analysis of the project.

10. Programmer Capability (PCAP)

    Rating level: *high*

    Effort multiplier: 0,88

    The team was able to efficiently make use of members' programming skills with a good organization of the work.

11. Personnel Continuity (PCON)

    Rating level: *very high*

    Effort multiplier: 0,81

    The team didn't undergo any personnel turnover.

12. Applications Experience (APEX)

    Rating level: *very low*

    Effort multiplier: 1,22

    The team experience in developing applications of this kind was less than 2 month (this type of application was completely new to us).

13. Platform Experience (PLEX)

    Rating level: *very low*

    Effort multiplier: 1,19

    The considered platform was completely new to the team.

14. Language and Tool Experience (LTEX)

    Rating level: *nominal*

    Effort multiplier: 1,00

    The programming language that has been mainly used is Java. Both team members have an experience with this programming language that is between 1 and 2 years.

15. Use of Software Tools (TOOL)

    Rating level: *nominal*

    Effort multiplier: 1,00

    The team utilized basic lifecycle tools with which was already experienced (we used similar tools in the Software Engineering 1 course the last year).

16. Multisite Development (SITE)

    Rating level: *low*

    Effort multiplier: 1,09

    The team members live in different cities and, during the project, had to communicate using individual phones or Social Networks.

17. Required Development Schedule (SCED)

    Rating level: *nominal*

    Effort multiplier: 1,00

    The time scheduled for planning activities has been fully used in order to develop the RASD and the Design Document.

We define the Effort Adjustment Factor (EAF) as the product of the effort multipliers corresponding to each of the cost drivers for the project. Given the previous cost drivers analysis we can calculate it:

$$EAF = 0.92*0.9*0.87*1*1*1*1*0.87*0.85*0.88*0.81*1.22*1.19*1*1*1.09*1 = 0.601$$

5 Effort computation

In order to compute the effort we need to know the KSLOC. This is a rather trivial computation since we already know the actual size of the project (see section 2).

$$KSLOC = \frac{SLOC}{1000} = \frac{4372}{1000} = 4.372$$

From the previous sections we know:

$$A = 2.94; E = 1.0677; EAF = 0.601$$

We can now compute the effort as:

$$PM = A * EAF * (KSLOC)E = 2.94 * 0.601 * 4.372^{1.0677} \approx 8.5(personMonths)$$

6 Schedule estimation

COCOMOII compute the time to develop the project as:

$$TDEV = [C * (PM_{NS})^{D+0.2*(E-B)}] * \frac{SCED\%}{100}$$

where $C = 3.67; D = 0.28; B = 0.91; PM_{NS}$ is the estimated PM excluding the SCED effort multipliers

With our values it turns out to be:

$$TDEV = [3.67 * (8.5)^{0.28+0.2*(1.0677-0.91)}] * 1 \approx 7.15(Months)$$

Therefore, the number of people that should have worked to the project is $\frac{8.5(personMonths)}{7.15(Months)} = 1.19(Persons) \approx 2$

7 Schedule comparison

COCOMO II treats the number of person-hours per person-month, PH/PM, as an adjustable factor with a nominal value of 152 hours per Person-Month. This number excludes time typically devoted to holidays, vacations, and weekend time off.

If we compute the number of *person-hours* corresponding to the computed effort we obtain:

$$PH = 152 * 8.5 = 1292(person - hours)$$

The time the team actually devoted to the project is 425 hours. The estimation is three times the real time. This may be due to the fact that COCOMO II has been developed to deal with big projects and teams. In this case, given that the team was made of just two members, we could avoid all the time required to cooperate with many other people. The avoidance of this overhead made the process quicker and can partially explain the overestimation.