Resource Estimation for Objectory Projects

Gustav Karner

Objective Systems SF AB
Torshamnsgatan 39, Box 1128
164 22 Kista
email: gustav@os.se
September 17, 1993

Abstract

In order to estimate the resources needed to develop a software system with the Objectory process, one would like to have a model which predict the total amount of resources early in the developing process. The model described in this article will help you to do a prediction like that.

1. Introduction

Objectory, see Jacobson I., Christerson M., Jonsson P. and Övergaard G. (1992), is a well defined process for developing industrial object oriented applications. The process is built of four different main processes which are:

- Requirements Analysis
- Robustness Analysis
- Construction
- Testing

It would be of great value if a prediction of the resources needed for these processes for a specific project could be done early in the developing process e.g. after the requirements analysis. With such an early estimation, one could more easily plan and predict for the rest of the project.

When a project is set up it is very important to know, in advance, how much resources is needed for the project to be completed. This kind of knowledge will help us to estimate the cost and the lead time for the project. It will also help to plan the use of the resources. These estimations should of course be available as soon as possible in the project.

This paper will initially survey a model for making early estimations called Function Points. It will then present a model for making early estimations when developing software with the Objectory process. The model is based on the Function Points.

2. Function Points

Function Points (FP) is a common model for estimations, proposed by Albrecht (1979). It is useful when you want to estimate man hours in an early phase, when you only have the specifications.

The model counts the number and complexity of (in order to estimate the size of the system):

1. Inputs, the number of different commands the software will accept.
2. Outputs, how many types of information it can generate.
3. Inquiries, how many different sorts of question a user can ask the system.
4. Files, how many it can cope with simultaneously.
5. Interfaces, the number of links it can have with other software.

Every one of these items are given a value as simple, average or complex with different weights (W_i). The Unadjusted Function Count (UFC) is:

$$UFC = \sum_{i=1}^{5} n_i * W_i$$

where n_i is the number of items of variety i where i stands for the number of items 1, items 2 etc.. and W_i is the weight of i.

These are later adjusted with a technical factor which describes the size of the technical complexity involved in the development and implementation of the system. The TCF is computed as:

$$TCF = C_1 + C_2 \sum_{i=1}^{n} F_i$$

where

$C_1 = 0.65$

$C_2 = 0.01$

F_i is the factors valued from 0 to 5. 0 if it is irrelevant and 5 if it is essential.

The Function Points are finally:

$$FP = UFC * TCF$$

When analysing the result we must have a statistical value from previous measures of how many Lines Of Code (LOC) a function point will need to be constructed. This value is multiplied with the number of function points of the system and we get the total amount of LOC needed. Typically one Function Point is 110 lines of COBOL code. This resulting value maybe used together with the COCOMO model, see Boehm (1981), to estimate the resources needed.

This model has been very popular recently. One of this model's greatest advantages is that it does not require a specific way of describing the system, for example a specific design method. The model has been criticised and some weakness has been noted, see Symons (1988). Some disadvantages is that function points cannot be computed automatically, it is not objective since you have to make many subjective decisions. This is because every input, output, inquire, file and interfaces must be valued as simple, average or complex and the technical factors must be valued by a human too. Furthermore it does not take into account any personnel factors.

3. Use Case Points

Our model *Use Case Points* is inspired by Function Points but with the benefit of the requirements analysis in the Objectory process. It starts with measuring the functionality of the system based on the use case model in a count called *Unadjusted Use Case Point* (UUCP). Technical factors involved in developing this functionality are assessed, similar to the Function Points. The last step in the estimation is however not from the Function Points and it is a new factor called *Environmental Factor* proposed by the author. This factor seems to be very important according to experienced Objectory users.

The Use Case Points (UCP) are the product of these three factors. The UCPs gives an estimation of the size of the effort to develop the system which can be mapped to man hours[1] to complete various phases of Objectory or complete the whole project.

The Use Case Points can also be mapped to for example the number of classes or LOC and from that estimate man hours needed with help from a model like the COCOMO model. The reason to do so is that the COCOMO model does not approximate the mapping as linear.

The weights in this article are a first approximation by people at Objective Systems.

3.1 UUCP - Unadjusted Use Case Point

To compute the UUCPs judge every actor if it is a simple, average or complex with help from Table 1 and every use case with help from Table 2. Only concrete actors and use cases are counted.

Complexity	Definition	Weight
SIMPLE	An actor is simple if it represents another system with a defined application programming interface.	1
AVERAGE	An actor is average if it is: 1. An interaction with another system through a protocol 2. A human interaction with a line terminal.	2
COMPLEX	An actor is complex if it interacts through a graphical user interface.	3

Table 1 Weighted actors.

Complexity	Definition	Weight
SIMPLE	A use case is simple if it has 3 or less transactions including alternative courses. You should be able to realise the use case with less than 5 analysis objects.	5

[1] The first approach is an approximation that within a given interval from 2 000 to 20 000 man hours the map can be done linear.

AVERAGE	A use case is average if it has 3 to 7 transactions including alternative courses. You should be able to realise the use case with 5 to 10 analysis objects.	10
COMPLEX	A use case is complex if it has more than 7 transactions including alternative courses. The use case should at least need 10 analysis objects to be realised.	15

Table 2 Weighted use cases.

We sum the weights of the actors and the use cases together to get the UUCP.

$$UUCP = \sum_{i=1}^{6} n_i * W_i$$

where n_i is the number of items of variety i.

If we do not have more information about the implementation project environment and the environment we can use the UUCP for our estimation. Otherwise we will adjust the UUCP to get a better estimation.

3.2 TCF - Technical Complexity Factor

The UUCP is weighted with the technical complexity factor (TCF), which vary depending on how difficult the system will be to construct. The TCF is nearly the same as for Function Points, the differences are that we have added some and removed some of the factors and we have weighted the factors differently based on experience in Objectory projects.

Symons (1988) define a criterion for a technical factor:

"A system requirement other than those concerned with information content intrinsic to and affecting the size of the task, but not arising from the project environment"

The constants and weights are proposed by Albrecht (1979) but C_1 is decreased from 0.65 to 0.6 to fit with the numbers of factors. The TCF is computed like this:

$$TCF = C_1 + C_2 \sum_{i=1}^{13} F_i * W_i$$

where

$C_1 = 0.6$

$C_2 = 0.01$

and

F_i	Factors Contributing to Complexity	W_i
F_1	Distributed systems.	2
F_2	Application performance objectives, in either response or throughput.	1
F_3	End user efficiency (on-line).	1
F_4	Complex internal processing.	1
F_5	Reusability, the code must be able to reuse in other applications.	1
F_6	Installation ease.	0.5
F_7	Operational ease, usability.	0.5
F_8	Portability.	2
F_9	Changeability.	1
F_{10}	Concurrency.	1
F_{11}	Special security features.	1
F_{12}	Provide direct access for third parties	1
F_{13}	Special user training facilities	1

Table 3 Factors contributing to complexity.

F_i is a factor which is rated on a scale 0, 1, 2, 3, 4 and 5. 0 means that it is irrelevant and 5 means it is essential. If the factor is not important nor irrelevant it will have the value 3. If all factors have the value 3 the TCF ≈ 1.

3.3 EF - Environmental Factor

We weight the UCP with the Environmental Factor (EF) which help us to estimate how efficient our project is. This factor is on the same form as the technical factor.

The constants are early estimations. They seem to be reasonable, based on interviews with experienced Objectory users at Objective Systems.

$$EF = C_1 + C_2 \sum_{i=1}^{8} F_i * W_i$$

where

$C_1 = 1.4$

$C_2 = -0.03$

and

F_i	Factors contributing to efficiency	W_i

F_1	Familiar with Objectory	1.5
F_2	Part time workers	-1
F_3	Analyst capability	0.5
F_4	Application experience	0.5
F_5	Object oriented experience	1
F_6	Motivation	1
F_7	Difficult programming language	-1
F_8	Stable requirements	2

Table 4 Factors contributing to efficiency.

F_i is a factor which is rated on a scale 0, 1, 2, 3, 4 and 5. 0 means that it is irrelevant and 5 means it is essential. If the factor is not important nor irrelevant it will have the value 3. If all factors have the value 3 the EF ≈ 1.

3.4 The Result and Analysis

Finally the Use Case Points is calculated as:

$$UCP = UUCP * TCF * EF .$$

Based on the calculated UCPs we look at statistics from earlier projects to see how much resources is needed per UCP. After that we multiply the number of UCP for our project with the *Mean Resources needed per UCP* (MR). We also use the *Standard Deviation of the MR* (SDMR) to see how good the estimations are.

Different MR and SDMR can be used for different Objectory processes (for example specific MR and SDMR for the robustness analysis) instead of the whole development. We also can have different statistics for different application domains, for example one for technical applications and one for MIS applications, to get a better estimation.

The analysis of the result is approximated as linear. This seems to be a good enough approximation for most of the projects. The institution tells us that the curve should be exponential since large projects typically have a lower productivity in general.

// More tomorrow!!

4. Projects Estimated

The model has been applied on a few projects of various size. The result is presented in this chapter.

4.1 The Measurements

Data from 3 projects is used to validate the approach above. Let us call these projects A, B and C.

Project A

The project developed an information system for operation support of performance management in telecommunication networks. A quite well defined project with only a few people who was newcomers to Objectory, but they were very motivated.

Unadjusted Use Case Points:
Number of Actors: 5 average actors
Number of Use Cases: 10 average use cases
UUCP = 110

Technical Complexity Factor:
All the factors have the default value 3.
TCF = 1

Environmental Factor:
Familiar with the method = 1.
Motivation = 5.
Stable requirements = 4.
Rest of the factors have the default value 3.
EF = 0.975

UCP = UUCP * TCF * EF = 107.25

Resources Used:
Man Hours to complete the project: 2150 h.

MR project A = 2150 / 107.25 ≈ 20.0

Project B

The project developed a LAN management system. The requirements were unstable. The developers had no previous experience of Objectory.

Unadjusted Use Case Points:
Number of Actors: 5 average actors
Number of Use Cases: 50 average use cases
UUCP = 510

Technical Complexity Factor:
All the factors have the default value 3.
TCF = 1

Environmental Factor:
Familiar with the method = 1.
Stable requirements = 1.
Rest of the factors have the default value 3.
EF = 1.175

UCP = UUCP * TCF * EF ≈ 599.25

Resources Used:
Man Hours to complete the project: 12 500 h.

MR project B = 12 500 / 599.25 ≈ 20.9

Project C

The project developed a telecommunication system. The team did not know much about Objectory.

Unadjusted Use Case Points:
Number of Actors: 5 average actors
Number of Use Cases: 15 average use cases
UUCP = 160

Technical Complexity Factor:
All the factors have the default value 3.
TCF = 1

Environmental Factor:
Familiar with the method = 1.
Application experience = 5.
Stable requirements = 2.
Difficult programming language = 5
Object oriented experience = 2.
Rest of the factors have the default value 3.
EF = 1.175

UCP = UUCP * TCF * EF ≈ 188.0

Resources Used:
Man Hours to complete the project: 5400 h.

MR project C = 5400 / 188 ≈ 28.7

4.2 The Result

The result is that there seems to be a correlation between UCPs and resources needed to finish a project. In figure 1 you can see the UCP plotted against the number of man hours for the projects mentioned here.

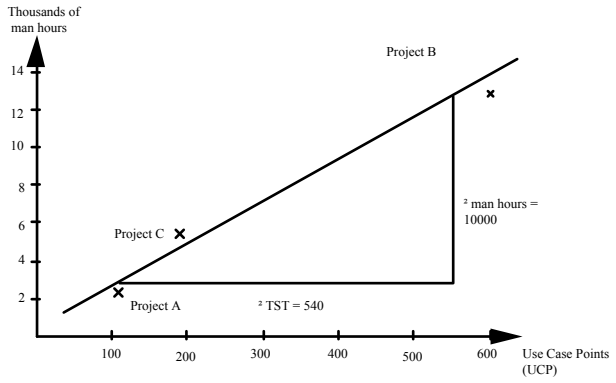

Figure 1 Man hours as a function of UCPs.

We can see from the plot that one UCP need 10000/540 ≈ 20 man hours to be completed.

The linear approximation gives us:

$$y = \alpha + \beta(x - \bar{x}),\ \bar{x} = \frac{1}{n}\sum x_i \approx 298.2$$

$$\alpha^* = \bar{y} \approx 6.683$$

$$\beta^* = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2} \approx 0.01981$$

$$y = 6.683 + 0.01981(x - 298.2) = 0.01981x + 0.7769$$

$$Q_0 = \sum [y_i - \alpha^* - \beta^*(x_i - \bar{x})]^2 \approx 54.64$$

$$s = \sqrt{\frac{Q_0}{n-2}} \approx 7.392$$

t-distribution with the probability that the values will be within the interval of 90 % gives us

$$t_{\alpha/2}(n-2) = t_{0.05}(1) = 6.31$$

the interval for α:

$$d = \frac{s}{\sqrt{n}} = 4.268$$

$$I_\alpha = (\alpha^* \pm t_{0.05}(1) * d) \approx 6.683 \pm 7.348 \Rightarrow$$

$$[-0.665, 14.03]$$

the interval for β:

$$d = \frac{s}{\sqrt{\sum (x_i - \bar{x})^2}} \approx 0.01981$$

$$I_\alpha = (\beta^* \pm t_{0.05}(1) * d) \approx 0.01981 \pm 0.1250 \Rightarrow$$

$$[-0.1052, 0.1448]$$

The intervals are much too wide to tell us anything. That is because the number of measured projects were too few.

5. Conclusions

We don not know if the model for estimating Objectory projects based on the use case model and some adjustments for technical and environmental factors works. That is because we have a too few projects to measure from so far. The work will continue with metrics from many more projects, because we still believe that this model could work a an estimation method.

6. Further Work

More data from projects need to be gathered, to adjust the model, weights and the constants. Objective Systems will have a database where everybody who is using Objectory in a project can send their data based on experience.

Anonymously will be guaranteed if requested. Please, send an email to magnus@os.se with the form in appendix A filled in.

References

Albrecht A. J. (1979). Measuring application development productivity. Proc. of IBM Applic. Dev. Joint SHARE/GUIDE Symposium, Monterey, CA, 1979, pp. 83-92.

Boehm B. W. (1981). Software engineering economics: Prentice-Hall, New York, 1981.

Jacobson I., Christerson M., Jonsson P. and Övergaard G. (1992). Object-Oriented Software Engineering: Addison-Wesley.

Symons C. R. (1988). Function Point Analysis : Difficulties and improvements. *IEEE Transactions on Software Engineering,* Vol. SE-14, No. 1. Jan. 1988.

Appendix A: Metrics Used to Adjust the Estimation Model

Actors

Complexity	Definition	Number of
SIMPLE	An actor is simple if it represents another system with a defined application programming interface.	
AVERAGE	An actor is average if it is: 1. An interaction with another system through a protocol 2. A human interaction with a line terminal.	
COMPLEX	An actor is complex if it interacts through a graphical user interface.	

Uses Cases

Complexity	Definition	Number of
SIMPLE	A use case is simple if it has 3 or less transactions including alternative courses. You should be able to realise the use case with less than 5 analysis objects.	
AVERAGE	A use case is average if it has 3 to 7 transactions including alternative courses. You should be able to realise the use case with 5 to 10 analysis objects.	
COMPLEX	A use case is complex if it has more than 7 transactions including alternative courses. The use case should at least need 10 analysis objects to be realised.	

Technical Complexity Factor

(If you can not fill in this factors for any reason use the default value 3 for every factors)

F_i	Factors Contributing to Complexity	0..5
F_1	Distributed systems.	
F_2	Application performance objectives, in either response or throughput.	
F_3	End user efficiency (on-line).	
F_4	Complex internal processing.	
F_5	Reusability, the code must be able to reuse in other applications.	
F_6	Installation ease.	
F_7	Operational ease, usability.	
F_8	Portability.	
F_9	Changeability.	
F_{10}	Concurrency.	
F_{11}	Special security features.	
F_{12}	Provide direct access for third parties	
F_{13}	Special user training facilities	

Environmental Factor

(If you can not fill in this factors for any reason use the default value 3 for every factors)

F_i	Factors contributing to efficiency	0..5
F_1	Familiar with Objectory	
F_2	Part time workers	
F_3	Analyst capability	
F_4	Application experience	
F_5	Object oriented experience	
F_6	Motivation	
F_7	Difficult programming language	
F_8	Stable requirements	

Resources used

Process	Man Hours