Theoretical Aspects of Projects Estimation Using the Use Case Points Method

Michał Wolski, Marek Pilski
Institute of Computer Science
Siedlce University of Natural Sciences and Humanities
3 Maja Str. 54, 08-110 Siedlce, Poland

Abstract. This text is the first one of the series of articles concerning projects' estimation in Enterprise Architect. In this section, theoretical aspects of function points methods have been presented. Part two entitled: "Example of the Application of the Use of the Case Points Method" constitutes an example along with mathematical calculations and part three entitled: "Project Estimation with the Use of the Method of Use Cases in Enterprise Architect" is a practical application of the function points method in Enterprise Architect.

Key words: use case point method, project estimation.

1 Introduction

Estimation is the quantitative assessment of unknown parameters of the project on the basis of the defined scope of works and general knowledge about conditions of the project implementation. We estimate these figures which we cannot calculate or measure or such figures the measurement of which is very costly or complicated [1], [2]. However, before we proceed to estimation of the work effort, it is worth to consider what are the possible applications of the obtained estimation.

The results of the estimation are usually used in case of two approaches related to software development. The first of them is the approach to estimation of the whole project associated with preparation of the cost and time of the whole project estimate in the context of the preparation of the tender offer [3]. The Client usually prepares requirements for future IT system, while suppliers of the software estimate costs associated with their offers on the basis of defined requirements.

The second approach is associated with Agile software development in iterative approach [4]. Here, in the context of planning the nearest iterations, one should indicate which requirements will be the subject of works in the nearest development cycle. The role of supplier of the software focuses then on the examination at what time and with what work effort will he be able to perform a given functionality of the future system.

As it turns out, there are many methods associated with the estimation of the work effort and costs related to development of IT systems, these are, among others:

- COCOMO II (*COnstructive COst Model)* [5], [6],
- Delphic method [7],
- Source lines of code [8],
- Function Points Method [9], [10],
- Use case points method [11], [12].

In this article we will focus on one of these, namely one the use case points method, which constitutes the basis for mechanism of the project estimation applied in Enterprise Architect.

2 Function points method

The beginnings of the use case points method were based on other known method for the purposes of estimating the size of the software, namely the function points method. This method, proposed by Allan Albrecht (IBM) in 1979, was based on screens' designs as well as on the architecture of the system. It was an attempt to overcome problems related to the use of the number of code lines (which is not known at the stage of defining the requirements and which was the basis for the estimation) as software size measure and at the same time an attempt to develop a method for prediction of the work effort related to the development of the software. Five major classes of attributes of productivity were present in this method, which were characterized by the system:

- External Inputs,
- External Outputs,
- External Inquiries,
- Internal Logical Files,
- External Interface Files.

For each category three degrees of complexity were defined: easy, medium and difficult. On the other hand, determined weight values were associated with each degree of complexity.

3 Use case points method

Use case points method, UCP in abbreviated form, has mechanism similar to the one which have been used in case of function points method, however, the use of screens and architecture was abandoned. Why screens and architecture were abandoned? Well, imagine a situation when a client asks about the duration of project implementation at the stage of specifying requirements. As it turns out in this situation, function points method is useless, as at this point we do not possess defined architecture or proposal of screens. It is therefore necessary to consider whether the requirements of the system may be the basis for estimation of the work effort during the development of IT system. In 1993 Gustaw Karner - the creator of the use case points method - answered yes to that question. He replaced the architec-

ture and screens with specification of requirements prepared in the form of *specification of use cases* [13]. In addition, he suggested that attention should be paid to the so-called *environmental complexity factors* describing to a large extent the organization which develops software, *factors of technical complexity* describing properties of products and future *actors* of the system.

Classification of use cases constitutes in case of UCP the basis for estimation of the work effort when developing IT systems. Here we identify all use cases, and more precisely speaking specifications of use cases because the description of basic scenario and analytical classes model will be needed for the purposes of estimation (identification of classes involved in the implementation of a given use case). At this stage one should specify level of complexity for each use case. The method indicates three types of complexity.

Table 1. Classification of use cases complexity

The complexity of use case	Definition	Weight
Easy	• Easy user's interface. • Operates on a single database entity. • Basic scenario contains 3 or less steps. • Implementation includes less than 5 classes.	5
Medium	• Medium user's interface. • Operates on 2 (or more) database entities. • Basic scenario contains 4 – 7 steps. • Implementation includes 5 – 10 classes.	10
Difficult	• Difficult user's interface. • Operates on 3 (or more) database entities. • Basic scenario contains more than 7 steps. • Implementation includes more than 10 classes.	15

Then we calculate **unadjusted use cases weight**, UUCW in abbreviated form. In order to do that, we count the number of use cases classified to each of the designated category (easy, medium, difficult). Then we multiply the number of use cases in a given category by the weight of a particular category of use case complexity. The sum of predefined products of multiplication by each category of use case complexity will constitute the unadjusted use cases weight. To sum up, the formula for calculation of unadjusted use cases weight is as follows:

$$UUCW = \sum_{i=1}^{3} n_i w_i$$

where: n - number of use cases classified to n category of use case complexity,
w- weight for the i of this category of use case complexity.

Environmental complexity factors characterize the supplier of the software. The method indicates eight environmental complexity factors and to each of

them the weight is assigned which informs how the factor affects the final result of the estimation. The bigger weight of the factor, the greater effect the factor has on reduction of work effort when developing IT system. It is worth noting that most of the factors have positive weights. This means that the growth in their impact reduces work effort. On the other hand, the growth in the impact of factors with negative weights will result in increase in the estimation of the work effort when developing IT system.

Table 2. Environmental complexity factors

Symbol	Description	Weight
E1	Knowledge of methodology, UML language	1,5
E2	Experience of the team	0,5
E3	Knowledge of object-oriented techniques	1
E4	Skills of the main analyst	0,5
E5	Motivation of the team	1
E6	Stability of requirements	2
E7	Participation of part-time employees	-1
E8	Complicated programming languages	-1

Let's take a closer look at environmental complexity factors:

- E1 – informs if the team is familiar with problem domain and technical aspects of the solution to a customer's problem. Attention should be also paid to knowledge of methodology in which the project is implemented e.g. RUP (*Rational Unified Process*), as well as knowledge of system modelling languages e.g. UML (*Unified Modeling Language*).
- E2 – generally understood experience of the team in developing the software.
- E3 – experience in designing object-oriented applications related to the ability to design object-oriented applications and the ability to use support tools for designing IT systems.
- E4 – determines the ability of the analyst to properly acquire requirements from the client and the possession of knowledge related to the problem which is being solved.
- E5 – assesses the ability of the team to engage in the assigned task.
- E6 – defines if the requirements are not exposed to frequent changes.
- E7 – determines whether there is a big number of part-time staff in the team (e.g. interns, students)
- E8 – specifies how difficult it is to learn the programming language in which the future IT system will be implemented.

In the method of use cases points, the impact of each factor is assessed in the 0 to 5 scale (the range of natural numbers), whilst:

- 0 – means that the factor is insignificant for the project,
- 3 – means an average impact,
- 5 – means a strong impact.

Then we calculate the value of environmental complexity factor, ECF in the abbreviated form, on the basis of the following formula:

$$ECF = 1{,}4 + \left(-0{,}03 \sum_{i=1}^{8} w_i \cdot impact_i \right)$$

where: w - weight of i of this factor, $impact$ - assessment of the impact of i of this factor.

Technical complexity factors characterize the future IT system. The method defines and aims to precisely determine the special character of the product by indicating thirteen factors. Similarly to environmental factors, the weight reflecting the degree to which the factor affects the value of final estimation is assigned to each of them. Growth in the impact of each technical complexity factor always results in the growth in final value of estimated work effort.

Table 3. Technical complexity factors

Symbol	Description	Weight
T1	Distribution of the system	2
T2	Efficiency of the system	1
T3	Efficiency for the final user	1
T4	Complex internal processing	1
T5	Re-usability	1
T6	Ease of installation	0,5
T7	Ease of use	0,5
T8	Portability	2
T9	Ease of introducing changes	1
T10	Concurrency	1
T11	Special access security mechanisms	1
T12	Providing access for external users	1
T13	Additional trainings for users	1

Let's study also technical complexity factors:

- T1 - informs whether distributed data processing is required in the system.
- T2 - determines the system's efficiency with regard to response time to events, flow, etc.
- T3 - defines efficiency for the final user in the context of his or her perception.
- T4 - determines whether complicated operations related to data processing, use of advanced algorithms are required.
- T5 - informs whether elements or the code of the generated system will be used again.
- T6 - specifies the method of installation, ease of installation and whether it is user-friendly, indicates whether the participation of specialists will be required for the installation and initial setup on the part of the software supplier.

- T7 - determines the adjustment of user's interface to his or her needs, convenience in use and whether it is easy to learn how to use the system.
- T8 - informs whether application should operate in certain environments.
- T9 - determines whether the system is to be built in such a way that it will be easy to further develop it in the future.
- T10 - informs whether the concurrent processing will take place in the system.
- T11 - defines whether the system will require the use of mechanisms related to securing access to the system or data.
- T12 - determines to what degree other external systems or actors will use the system.
- T13 - determines the need to organize trainings for users.

By analogy, just as before, the impact of each factor is assessed in the 0 to 5 scale (the range of natural numbers), whilst:

- 0 – means that is factor insignificant for the project
- 3 – means an average impact
- 5 – means a strong impact

Then we calculate the value of technical complexity factor, TCF in the abbreviated form, on the basis of the following formula:

$$TCF = 0{,}6 + \left(0{,}01 \sum_{i=1}^{13} w_i \cdot impact_i \right)$$

where: w - weight of i of this factor, *impact* - assessment of the i of this factor.

Classification of actors is another stage related to the estimation of the project. Here we identify all actors who will interact with the future IT system. At this stage one should specify level of complexity for each type of an actor. The method indicates three types of complexity.

Table 4. Classification actors complexity

The complexity of an actor	Definition	Weight
Simple	External system which communicate with the target system via application programming interface (API).	1
Medium	External system communicating by means of more complex protocol, e.g. http. A person enters interacts with the system via text terminal.	2
Difficult	The final user (a person) who communicates with the system mostly via the graphical interface.	3

Then we calculate **the unadjusted actors weight,** UAW in abbreviated form. For this purpose we count the number of actors classified to each of the designated category (simple, medium, difficult). Then we multiply the number of actors

in a given category by the weight of a particular category. The sum of predefined products of multiplying by each actor complexity category will constitute the value of unadjusted actors weight. To sum up, the formula for calculation of unadjusted actors weight is as follows:

$$UAW = \sum_{i=1}^{3} n_i w_i$$

where: n - number of actors classified to i of this category, w – weight value for i of this category.

Activites associated with the use cases points method are slowly being finalized. We have already determined several factors, it is time to use their values. After calculation of unadjusted actors weights and unadjusted use cases weights UUCW, we can calculate unadjusted use case points, UUCP in abbreviated form, by normal summation of the aforementioned values:

$$UUCP = UAW + UUCW$$

where: *UAW* - the unadjusted actors weight, *UUCW* - the unadjusted use cases weights.

Whereas we will determine use case points UCP of particular interest for us with the use of a formula which uses previously designated values of technical complexity factor, TCF and environmental complexity factor, ECF:

$$UCP = UUCP \cdot TCF \cdot ECF$$

where: *UUCP* - unadjusted use case points, *TCF* - technical complexity factor, *ECF* - environmental complexity factor.

The last stage in the UCP method is to convert use case points into specific values of work effort calculated in the form of man-hours. It is done by multiplying UCP by the so called productivity factor. Productivity factor converts one point of use case into the number of man-hours. Historically, the PF value varied from 15 to 30 man-hours per one use case point. From researches carried out by Karner, the creator of the method, it results that implementation of one UCP requires approximately 20 man-hours.

Worth noting that the biggest problem in assessment of these factors (environmental complexity factors, Technical complexity factors and classification actors complexity) is lack of standardization, which leads to subjectivity in their assessment. You can use the proposal for standardization of factors presented in [14].

4 Conclusion

Use case points method, UCP in abbreviated form, developed in 1993 by Gustav Karner, is the method of the estimation of the work effort and the costs of the development of IT system already at the stage of collection of requirements. It uses

basic elements of object-oriented techniques of IT systems modeling such as an actor and use case, as well as characteristics of the future product and characteristics of the team working on the solution to the problem.

In subsequent parts

- Example of the application of the use case points method
- Project estimation with the use of the use case points method when using Enterprise Architect

practical example of the project estimation with the use of use case points method will be discussed.

5 References

1. Mohagheghi Parastoo, Bente Anda and Reidar Conradi, (2005) *Effort estimation of Use Cases for incremental large-scale software development*, International Conference on Software Engineering (ICSE), 303-31.
2. L. Laird, M. Brennan, (2006) *Software Measurement and Estimation: A Practical Approach,* Wiley-Interscience.
3. Issa Ayman, Mohammed Odeh, David Coward (2006) *Software Cost Estimation Using Use-Case Models: a Critical Evaluation*, Information and Communication Technologies, ICTTA '06. 2nd Volume 2, 2766-2771.
4. Cohn Mike (2005) *Agile Estimating and Planning*. Prentice Hall.
5. B.W. Boehm, C. Abts, A.W. Brown, S. Chulani, B.K. Clark, E. Horowitz, R. Madachy, D.J. Reifer, B. Steece, (2000). *Software Cost Estimation with Cocomo II*, Prentice Hall.
6. Software Project Cost Estimates Using COCOMO II Model (2005) http://www.codeproject.com/KB/architecture/cocomo2.aspx (last visited: December 2010).
7. Gene Rowe, George Wright, (1999) *The Delphi technique as a forecasting tool: issues and analysis*, International Journal of Forecasting, Volume 15, Issue 4, October, 353-375.
8. S.D. Conte, H.E. Dunsmore, V.Y. Shen, (1986). *Software engineering metrics and models*, Menlo Park, Kalifornia: The Benjamin/Cummings Publishing Company, Inc.
9. A.J. Albrecht, (1979). *Measuring Application Development Productivity,* Proceedings of the Joint SHARE, GUIDE, and IBM Application Development Symposium, Monterey, California, October 14–17, IBM Corporation, 83–92.
10. M.A. Parthasarathy (2007) *Practical Software Estimation: Function Point Methods for Insourced and Outsourced Projects*. Addison-Wesley Professional.
11. Sergey Diev, (2006). *Use cases modeling and software estimation: Applying Use Case Points*, ACM Software Engineering Notes, Volume 31, Number 6.
12. Vinsen Kevin, Diane Jamieson and Guy Callender (2004) *Use Case Estimation - The Devil is in the Detail*, 12th IEEE International Requirements Engineering Conference (RE'04), 10-15.
13. W. Dąbrowski, A. Stasiak, M. Wolski, (2007). *Modelowanie systemów informatycznych w języku UML 2.1 w praktyce*, MIKOM, Warszawa.
14. B. Anda, H. Dreiem, D. Sjoberg, M. Jorgensen (2001) *Estimating Software Development Effort Based on Use Cases-Experiences from Industry*, Proceeding, Fourth International Conference on the UML, Concepts, and Tools, Springer-Verlag London, 487-504.