Journal of Advances in Engineering Science
Section C (3), July - December 2010, PP 41-48

COST ESTIMATION FOR DISTRIBUTED SYSTEMS USING USE CASE DIAGRAM

Prof. S. V. Pingale[1]*, Prof. R. S. Badodekar[2] and Prof. H. S. Badodekar[3]

1 - Department of Computer Engg, Sinhgad Institute of Technology, Lonavala, Dist-Pune, INDIA
2 - Department of Info-Technology, Sinhgad Institute of Technology, Lonavala, Dist-Pune, INDIA
3 - Department of E&TC, A. C. Patil College of Engg, Kharghar, New Mumbai, INDIA

ABSTRACT

Estimating the cost of development is based on a prediction of the size for future systems. A lot of cost estimation models were reported in the literature but many of these models became obsolete because of the rapid changes in technology. Reliable estimations are difficult to obtain because of the lack of detailed information about the future system at an early stage. Cost models like COCOMO (COnstructive COst MOdel) [5,6] and sizing methods like Function Point analysis are well known and in widespread use in Software Engineering. These models were applicable only to procedural paradigm, and are not directly applicable to software products developed using the object oriented methodology or real time systems. It is this idea that gave birth to the creation of Use Case Point (UCP) metrics, originally developed by Gustav Karner [3]. UCP uses use cases as the primary factor, use case model is the first model developed in an object-oriented design process using UML. In this paper we propose a novel approach to map the object oriented systems from their function points and converting use case point counts on the basis of actor (Customer's) interaction to the software and to estimate cost of development by using Synthesized UCP (s- UCP) model with additional information obtained from synthesized use case attributes.

Key words : *Use Case Point, COCOMO, Cost Estimation*

1. INTRODUCTION

This Software is an intangible product and hence probably the most crucial difference between the manufacturing industry and the software industry is that the former is able to stick to schedules and cost most of the time. Even when using a well-defined methodology, the development cost of a well-defined application is not easy to predict. Some key factors that contribute to this difficulty include the precise set of functionalities to be implemented, the various risks associated with the development process, the knowledge and experience of the development team and lack of detailed information about the system at an early stage. Among these, the set of functionalities to be implemented is the most crucial factor. Software cost estimation models developed in the 60's and 70's used the Line of Code (LOC) or Delivered Source Instruction (DSI) metrics. For example, COCOMO model [5, 6] used the DSI metrics while many others used LOC metrics. The major problem with these metrics was two-fold: (i) the lack of precise definition of LOC or DSI, and (ii) there is no reasonable methodology by which one can estimate the number of source code lines or instructions until the coding is over. Further, these metrics were defined for procedural, possibly line oriented, languages such as FORTRAN and COBOL. With the development of block structured languages such as Pascal, Algol, C etc., the notion of LOC or DSI was difficult to define precisely.

Albrecht came up with the Function Point (FP) metrics in 1979[1]. FP uses five parameters: number of inputs, number of outputs, number of inquiries, number of internal logical files and number of external logical files. Consequently, FP is based on the number of interactions and the size of data to be used in the end product. While FP eliminated the need for lines of code or delivered source instructions, and was

*sub.pingale83@gmail.com

widely used because of its independence on development platform and environment, it does not seem to be applicable to software products developed using the object-oriented methodology [8] for languages like C++ and JAVA. In particular, the notion of internal and external logical files is somewhat harder to identify in the object-oriented paradigm as well as real time systems.

Gustav Karner [3] came up with the notion of Use Case Point (UCP) which is somewhat similar to the notion of Function Point but based on use cases. Model based estimation will help in a greater extend to reverse engineering and maintenance of legacy software. The use case model is the front end model of the Unified Modeling Language (UML) [9]. With the emergence of the UML as the most commonly used notation to model and design object-oriented software products, the application of use cases for size and hence cost estimation seems to be a perfect fit. However, Karner's [3] method does not take into account some of the application domain details such as the number of interaction between actors, and relationships between use cases.

The objective of this research is to elaborate the UCP model by synthesizing the use cases with a focus on internal details of each use case as well as interaction between the use cases. It is common that every use case is supported by a use case synthesized attributes that explains the internal details of the use case. This paper describes a method for cost estimation based on the use case diagram at an early stage of software development. It focuses on the use case attributes, uses the interaction between the entities in a use case diagram, and hence closely estimates the size of the software product to be developed. The methodology was tested with the use cases developed for an automated indexing and searching system for a book publishing company. The estimated value for development efforts seems to match with the actual development efforts spent by the company.

The rest of the paper is organized as follows: Section 2 describes the step by step application of the synthesized UCP methodology. A case study with a step wise evaluation of UCP using s-UCP model is described in section 3. The paper concludes in Section 4 with the discussion on continuing work in this direction in Section 5. Due to space constraints, this paper does not include descriptions of a use case model such as notations and their semantics. Interested readers are referred to any UML book such as [10] or UML manual published by OMG [9].

2. PROPOSED METHOD : SYNTHESIZED UCP (S-UCP)

The s-UCP methodology uses every aspect of a use case model such as actors, use cases, interaction between actors and use cases, relationships between actors, relationships between use cases and finally the synthesized attributes of each use case. The last one is important because it describes the missing details of a use case diagram, while others can be directly extracted from a use case diagram itself. This makes the significant difference between UCP method given by Karner [3] and s-UCP. The attributed list also contains few use case narratives suggest by Periyasamy [4]. In order to concretize the methodology, the authors used the template for use case synthesized attributes in Table I.

Table 1 :Sample use case synthesized attributes

Use Case Name	A Descriptive Name of the Use Case
Purpose	A brief description of the tasks to be implemented by this use case
Input parameters	List of Input Parameters to the use case
Output parameter	List of Output parameters returned from the use case.
Primary actor	The list of actors who invoke this use case
Secondary Actor	The list of actors that are used by this use case
Precondition	Conditions that must test true to use the use case. Unlike assumptions, these conditions are tested by this use case before doing anything else. If the conditions are not true, the actor or other use case is refused entry.

Post condition	Conditions that must test true when the use case ends. You may never know what comes after the use case ends, so you must guarantee that the system is in a stable state when it does end.
Process	A step-by-step description of the dialog between the use case (the system) and the user (actor or other use case). Very often it is helpful to model this sequence of events using a flowchart or activity diagram just as you might model a protocol for communication between two business units.
Successful Scenario	A sequence of instruction that explain the successful scenario of invoking this use case.
Exceptions	A set of conditions that may make the use case fail when invoked.
Includes List	Use cases that can be included in this use case.
Extends List	Use cases that can be extended from this use case.

2.1 Use Case Estimation

The s-UCP method first calculates unadjusted UCP values, referred to as UUCP in this paper. This includes unadjusted actor weights, unadjusted use case weights. The unadjusted UCP values are then adjusted using three additional factors, namely technical complexity factor (TCF), environment factor (EF), and synthesized factor (SF). The final result is the adjusted UCP values, referred to as AUCP. The s-UCP method uses the same TCF and EF weights as given in the original UCP method [3], but it uses a different set of calculations for UCP values. The authors of this paper believe that the final UCP value (s-UCP) returned by AUCP is more precise compared to those returned by UCP.

2.1.1 Actor Weight Classification

Table II shows the assignment of weights to various actors based on the actor type and the number of transactions performed by an actor.

TABLE II : Actor Weight Classification

Actor type	Classification of Actors Weight	Weight
Very simple	Specialized Primary/ Secondary actor	1.0
Simple	Simple Primary actor with 1 < number of transactions <= 3	1.5
Less average	Primary actor with 3 < number of transactions <= 5	2.0
Average	Primary actor with number of transactions>5 Secondary actor with 1 transaction	2.5 2.5
Complex	Complex Secondary actor with 1 < number of transactions < = 3	3.0

This, in turn, is based on the number of transactions the actor has with the use cases. As the number of transactions increases, more effort is involved in coding and hence the complexity of the corresponding actor increases. The complexity of a secondary actor (such as a database) with 'n' transactions is higher than that of a primary actor (such as a user) with the same number of transactions because generally more effort is involved in coding transactions with the secondary actor compared to those with a primary actor. For example, if the secondary actor is a database, then coding efforts are required for checking the connection to the database, writing SQL statements for transactions, checking for database commit actions and so on.

2.1.2 Use Case Weight Classification

Similar to actors, each use case is assigned a different weight. The use case type is defined based on the number of transactions (the number of direct connections between actors and the use case). Table III lists the classification of use cases and their associated weights [7].

TABLE III : Use case weight classification

Use Case type	Classification of Use Cases	Weight
Simple	Number of transactions <= 3	5
Average	4 < number of transactions <= 7	10
Complex	7 < number of transactions <=	15

2.1.3 Weights for Synthesized use Case Attributes

A use case diagram must be supported by Synthesized Use case Attributes. Table I shows the structure of a synthesized use case attributes used in this methodology. Though there is no standard for the structure of a use case attributes, the authors found that the use case structure illustrated in Table I contains all information that many practitioners use. With this assumption, Table IV describes the weights associated with the different parameters of a use case attributes.

Notice that all the use case attributes shown in Table I are used in Table IV because all of them contribute to the coding efforts, and others such as actors are already taken into consideration. Moreover, it should also be noted that the weight associated with 'Precondition' in Table IV must be used for only one predicate in the precondition; the same applies to post condition as well.

Table IV : Weight for synthesized use case attributes

Use Case type	Classification of Use Cases	Weight
S1	Input parameter	0.1
S2	Output parameter	0.1
S3	A condition to execute each process	0.1
S4	A predicate in Precondition	0.1
S5	A predicate in Post-condition	0.1
S6	An action in Successful scenario	0.2
S7	An exception	0.1
S8	An Includes conditions	0.1
S9	An Extends conditions	0.1

Thus, a complex precondition such as *Bank account number must be valid* **^** *PIN code must be valid* will include two simple predicates. The justification of assigning separate weight for each individual predicate comes from the fact that each simple predicate is required to be implemented to validate the precondition.

3. EVALUATION OF s-UCP: A CASE STUDY

In the original method the total unadjusted actor weight (UAW) is calculated by counting how many actors there are of each kind (by degree of complexity), multiplying each total by its weighting factor & adding up the products. Each use case is then defined as simple, average complex depending on number of transactions in the use case descriptions, including secondary scenarios.

3.1 Proposed Estimation Method

The primary goal of proposed technique is to estimate the cost of software from number of different actors and its use case interactions between both of them during the time of implementing the application. In this proposed technique, all the actors are classified into very simple, simple, less average, average, complex and then the use case are classified into simple, average and complex based on the number of actor interacting with that use case along with use case interacting with another use case. Hence use case complexity can be defined easily. The s-UCP method also uses synthesized weight factor (SF) to improve the effectiveness of this technique. Further classification is made for all the use cases that interact with another use case. If the current use case is interacting with a simple use case then we will classify it as average and if the current use case is interacting with an average use case then we will classify it as complex. Then we will assign the weighting factor in the above said manner.

3.1.1 Estimating s-UCP

Finally, the synthesized use case point (s-UCP) is calculated by multiplying all the three values UUCP, TCF, EF & SF. That is,

s-UCP = UUCP * TCF * EF*SF (1)

CASE STUDY: - **A Book Indexing Project.**

Given bellow is the use case diagram of indexing project for which we will calculate the use case points.

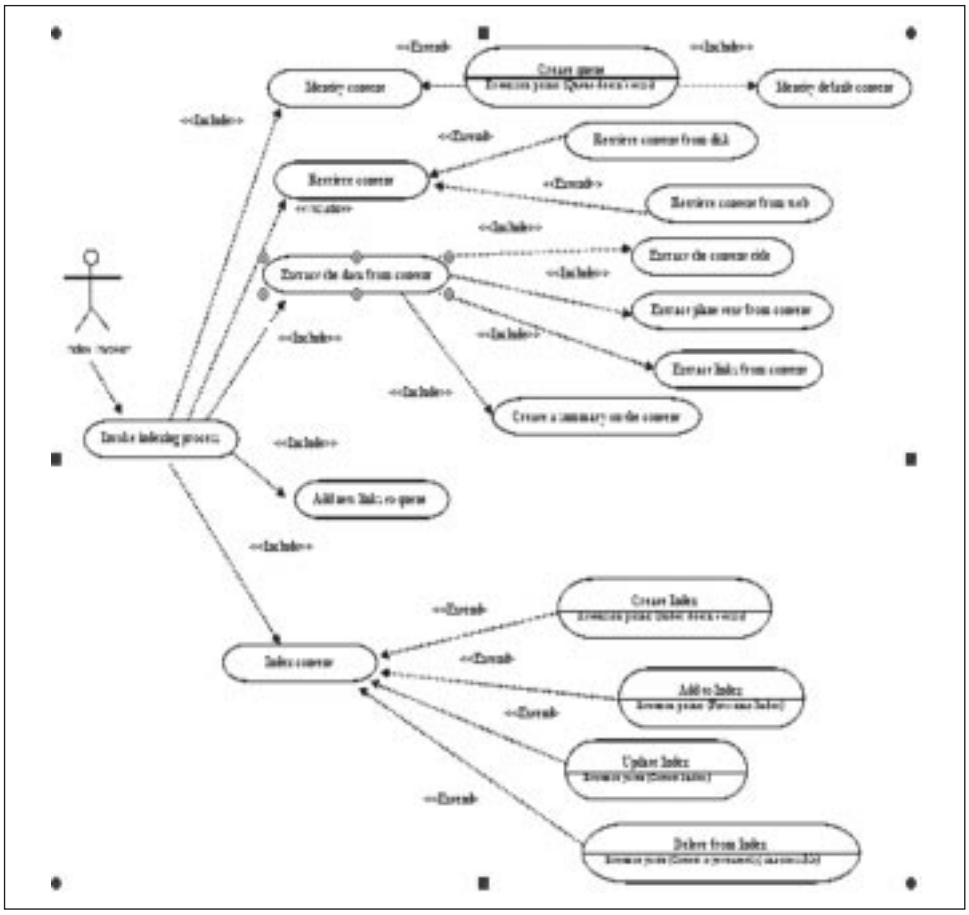

Figure 1 : Use case diagram for book indexing project

3.2 Weight Based Estimation

The method employs a technical factors multiplier corresponding to the technical complexity adjustment factor of the FPA method, environmental factors multiplier in order to quantify non-functional requirement & synthesized factor in order to quantify the use cases. Various factors influencing productivity are associated with weight and values are assigned to each factor, depending on the degree of influence.

The Technical Factor (TCF) is calculated multiplying the value of each factor (T1 – T13) by its weight and then adding all these numbers to get the sum called the T Factor. Finally, the following formula is applied:

TCF = 0.6 + (0.01 * T Factor) (2)

The Environmental Factor (EF) is calculated accordingly by multiplying the value of each factor (F1 – F8) by its weight and adding all the products to get the sum called the E Factor. The formula below is applied:

EF = 1.4+ (-0.03 * E Factor) (3)

The Synthesized Factor (SF) is calculated accordingly by multiplying the value of each factor (S1 – S9) by its weight and adding all the products to get the sum called the S Factor. The formula below is applied:

SF = 1.1+ (0.01 * S Factor) (4)

3.2.1 Weight based on Technical Factor

The technical factors and their weighted values are shown in the following table:-

Table I : Technical Factors in Project and their Weights

Factor Number	Description	Weight	Assessment	Impact
T1	Distributed System	2	0	0
T2	Response adjectives	1	4	4
T3	Response adjectives	1	5	5
T4	Complex processing	1	2	2
T5	Reusable code	1	3	3
T6	Easy to install	0.5	4	2
T7	Easy to use	0.5	4	2
T8	Portable	2	4	8
T9	Easy to change	1	3	3
T10	Concurrent	1	3	3
T11	Security features	1	3	3
T12	Access for third parties	1	1	1
T13	Special training required	1	4	4
T-FACTOR		**TOTAL WEIGHT**		**40**

The TCF valued is calculated using equation (2):

TCF = 0.6 + (0.01 * 40) = 1

3.2.2 Weight based on Environmental Factor

The environmental factors and their weighted values are shown in the following table:-

Table II : Environmental Factors in Project and Their Weights

Factor Number	Description	Weight	Assessment	Impact
F1	Familiar with RUP	1.5	2	3
F2	Application experience	0.5	1	0.5
F3	Object-Oriented experience	1	2	2
F4	Lead analyst capability	0.5	1	0.5
F5	Motivation	1	2	2
F6	Stable requirements	2	3	6
F7	Part-time workers	-1	0	0
F8	Part-time workers	-1	4	-4
E-FACTOR		**TOTAL WEIGHT**		**10**

The EF value is calculated using equation (3):

EF = 1.4 + (-0.03 * 10) = 1.1

3.2.3 Weight based on Synthesized Factor

TABLE III : Weight for use case on Synthesized Attribute

Attribute Number	Use case Synthesized Attributes	Weight	Assessment	Impact
S1	Input parameter	0.1	2	0.2
S2	Output parameter	0.1	2	0.2
S3	A condition to execute each process	0.1	1	0.1

S4	A predicate in Precondition	0.1	1	0.1
S5	A predicate in Post-condition	0.1	1	0.1
S6	An action in Successful scenario	0.2	1	0.2
S7	An exception	0.1	1	0.1
S8	An Includes conditions	0.1	1	0.1
S9	An Extends conditions	0.1	1	0.1
s-FACTOR		**TOTAL WEIGHT**		**1.2**

The SF value is calculated using equation (4):

SF = 1.1+ (0.01 * 1.2) = 1.112

The UUCP value is calculated using the following equation:

UUCP = UAW + UUCW (5)

UUCP = 1*2 + 15*5 + 3*10 + 0*15 = 107

Using the above values the use case points of this project is calculated using equation (1):

s-UCP = 107 * 1 * 1.1 * 1.112 = 130.882

3.3. Result Analysis

The UCP count using s-UCP method will give more accurate and effective estimation as compared to the UCP counts done by Bente Anda [2] and Periyasamy [4]. The comparison made is listed in Table VI.

Table IV : Comparative Study

Sr. No.	METHOD	UCP Count
1	Bente Anda [2]	107.23
2	Periyasamy [4] (e-UCP)	118.91
3	s-UCP (Proposed method)	130.882

4. CONCLUSION

This paper evaluates a use case diagram for an indexed project and calculated the synthesized use case points (s-UPC) according to proposed method which help us to calculate the cost of the project. Here the proposed method constructed a simple way to use minimal set of formula to calculate the cost of objects oriented software development. This motivates a relationship between some of the existing use case methods. With classifying the actors here, the approach was able to classify the software system with respect to the level of code quality and the better way to calculate the use case points.

This approach is very promising and more applicable to identifying low quality code than high and use case points due to specific theoretical concept employed to develop the approach. However, this is a mammoth step in the right direction in reducing the turnaround time. It takes to perform a code analysis on industrial software cost estimation.

5. REFERENCES

1. Albrecht, A.J., 1979. Measuring application development productivity. IBM Applications Development Symp, GUIDE Int. and SHARE Inc., IBM Corp., Monterey, CA, Oct. 14-17, 1979.
2. Anda B. et al. Estimating software development efforts based on use cases - Experience from industry. In M. Gogolla, C. Kobryn (Eds.): UML 2001 - The Unified Modeling Language. Springer-Verlag. 4th International Conference, Toronto, Canada, October 1-5, 2001, LNCS 218, 2001.
3. Karner, G. 1993. Metrics for Objectory. Diploma thesis,University of Linköping, Sweden. No. LiTHIDA- Ex- 9344:21. December 1993.
4. Periyaswamy,K., Ghode, A. “Effort Cost Estimation using extended Use Case Point (e-UCP) Model”, IEEE 2009
5. Boehm, B., 1981. Software Engineering Economics, Prentice Hall, 1981.

6. Boehm, B. et al., Software Cost Estimation with COCOMO II, Prentice Hall Englewood Cliffs, NJ, 2000.

7. Edward, C. R. “Estimating Software Based on Use Case Points.” Proceedings of the Object-Oriented, Programming, Systems, Languages, and Applications (OOPSLA) Conference, San Diego, CA, 2005.

8. Fetke, T., Abran, A., and Nguyen, T. 1997. Mapping the OO-Jacobsen approach into function point analysis. In proceedings of Technology of Object-Oriented Languages and Systems (TOOLS 97), 1997.

9. UML 2.0 Reference Manual, Object Management Group, www.omg.org, 2003.

10. Schneider, G., and winters, J.P., Applying Use Cases, Second edition, Addison Wesley, 2001.

■