Effort Estimation Using Use Case Point for Software Development

Prof. Svapnil M Vakharia

Assistant Professor
Information Technology Department
Gandhinagar Institute of Technology, Gandhinagar, India

Abstract: This paper describes study of an effort estimation method based on use cases, the Use Case Points method. The original method was adapted to incremental development and evaluated on a large industrial system with modification of software from the previous release. I modified the following elements of the original method: a) complexity assessment of actors and use cases, and b) the handling of non-functional requirements and team factors that may affect effort. The study identified factors affecting effort on projects.

Index Terms **- Estimation, use cases, incremental development.**

I. INTRODUCTION

Effort estimation is a challenge in every software project. The estimates will impact costs and expectations on schedule, functionality and quality. While expert estimates are widely used, they are difficult to analyze and the estimation quality depends on the experience of experts from similar projects. Alternatively, more formal estimation models can be used. Traditionally, software size estimated in the number of Source Lines of Code (SLOC), Function Points (FP) and Object Points (OP) are used as input to these models, e.g. COCOMO and COCOMO II.

Because of difficulties in estimating SLOC, FP or OP, and because modern systems are often developed using the Unified Modeling Language (UML), UML-based software sizing approaches are proposed. Examples are effort estimation methods based on use cases and software size estimation in terms of FP from various UML diagrams.

The *Use Case Points* (UCP) estimation method introduced in 1993 by Karner estimates effort in person-hours based on use cases that mainly specify functional requirements of a system Use cases are assumed to be developed from scratch, be sufficiently detailed and typically have less than 10-12 transactions. The method has earlier been used in several industrial software development projects and in student projects. There have been promising results and the method was more accurate than expert estimates in industrial trials.

Recently, incremental or evolutionary development approaches have become dominant: requirements are covered (or discovered) in successive releases, and changing requirements (and software) is accepted as a core factor in software development. Each release of a software system may in turn be developed in iterations of fixed or variable duration, and may be maintained for a while, before being replaced with a new release. Project management in these projects needs an estimation method that can estimate the effort for each release based on changes in requirements. Furthermore, software is built on a previous release that should be modified or extended.

II. THE UNDERLYING ESTIMATION METHODS

2.1The Use Case Points Estimation Method

A use case model defines the functional scope of the system to be developed. Attributes of a use case model may therefore serve as measures of the size and complexity of the functionality of a system.

The Use Case Points (UCP) estimation method is an extension of the *Function Points Analysis* and *MK II Function Points Analysis.* Table 3 gives a brief introduction of the six-step UCP method. In step 4, there are 13 *technical factors* Table 1 (related to how difficult it is to build the system, e.g. distributed system, reusable code and changeability) and eight *environmental factors* Table 2 (related to the efficiency of the project e.g. object-oriented experience and stable requirements).

The weights and the formula for technical factors is borrowed form the Function Points method proposed by Albrecht. The formula for environmental factors is based on some estimation results. In step 6, the adjusted Use Case Points (UCP) is multiplied by person-hours needed to implement each use case point .

TABLE 1. TECHNICAL COMPLEXITY FACTORS

TECHNICAL FACTOR	DESCRIPTION	WEIGHT
T1	DISTRIBUTED SYSTEM	2
T2	PERFORMANCE	1
T3	END USER EFFICIENCY	1
T4	COMPLEX INTERNAL PROCESSING	1
T5	REUSABILITY	1
T6	EASY TO INSTALL	0.5
T7	EASY TO USE	0.5
T8	PORTABLE	2
T9	EASY TO CHANGE	1
T10	CONCURRENT	1
T11	SPECIAL SECURITY FEATURES	1
T12	PROVIDES DIRECT ACCESS FOR THIRD PARTIES	1
T13	SPECIAL USER TRAINING FACILITIES ARE REQUIRED	1

TABLE 2 ENVIRONMENTAL COMPLEXITY FACTORS

TECHNICAL FACTOR	DESCRIPTION	WEIGHT
E1	FAMILIARITY WITH UML	1.5
E2	APPLICATION EXPERIENCE	0.5
E3	OBJECT ORIENTED EXPERIENCE	1
E4	LEAD ANALYST CAPABILITY	0.5
E5	MOTIVATION	1
E6	STABLE REQUIREMENTS	2
E7	PART-TIME WORKERS	-1
E8	DIFFICULT PROGRAMMING LANGUAGE	2

TABLE 3. THE UCP ESTIMATION METHOD

STEP	RULE	OUTPUT
1	CLASSIFY ACTORS: A) SIMPLE, WF (WEIGHT FACTOR) = 1 B) AVERAGE, WF = 2 C) COMPLEX, WF = 3	UNADJUSTED ACTOR WEIGHTS (UAW) = Σ(#ACTORS * WF)
2	CLASSIFY USE CASES: A) SIMPLE- 3 OR FEWER TRANSACTIONS, WF = 5 B) AVERAGE- 4 TO 7 TRANSACTIONS, WF = 10 C) COMPLEX- MORE THAN 7 TRANSACTIONS, WF= 15	UNADJUSTED USE CASE WEIGHTS (UUCW) = Σ (#USE CASES * WF)
3	CALCULATE THE UNADJUSTED USE CASE POINT (UUCP).	UUCP = UAW + UUCW
4	ASSIGN VALUES TO THE TECHNICAL AND ENVIRONMENTAL FACTORS [0..5], MULTIPLY BY THEIR WEIGHTS [-1..2], AND CALCULATE THE WEIGHTED SUMS (TFACTOR AND EFACTOR). CALCULATE TCF AND EF AS SHOWN.	TECHNICAL COMPLEXITY FACTOR (TCF) = 0.6 + (0.01 * TFACTOR) ENVIRONMENTAL FACTOR (EF) = 1.4 + (-0.03 * EFACTOR)
5	CALCULATE THE ADJUSTED USE CASE POINTS (UCP).	UCP = UUCP * TCF * EF
6	ESTIMATE EFFORT (E) IN PERSON-HOURS.	E = UCP * PH PER UCP

III. THE ADAPTED UCP ESTIMATION METHOD

3.1 Overview of the Adapted Method

This section describes how the UCP method has been modified. The new rules are summarized in Table 4, with the same abbreviations as in Table 3 and as described below.

Step 1. Actors**.** An actor may be a human, another system or a protocol. Since the classification has little impact on the final estimation result, all actors are assumed to be average. Modified actors are also counted as the Modified Unadjusted Actor Weights (MUAW).

Step 2. Counting the UUCW and MUUCW**.** We started to count the Unadjusted Use Case Weights (UUCW) for Release 1 using the method described in Section 2.1. All use cases in this study would be classified as complex. Use cases should then be classified. Applying step 2 in Table 4 led to most new use cases being classified as simple (66%), and very few as complex. However, the complexity of transactions does not justify such distribution. Karner proposed not counting so-called included and extended use cases, but the reason is unclear. We have applied the same rules to all the use cases.

Step 3. Counting UUCP**.** The Unadjusted Use Case Points (UUCP) are calculated; once for all use cases (in Rule 3.1) and once for modifications (in Rule 3.2).

Step 4. TCF and EF**.** Assigning values to technical and environmental factors are usually done by experts or project leaders, based on their judgment. The impact of the Technical Complexity Factor (TCF) is small and it does not cover all the non-functional requirements either. We have therefore handled this otherwise, as described in step 6 below. The Environmental Factor (EF) is not relevant in this project as there are few changes to this factor from one release to another. However, this factor does have a large impact on the estimate and we have accounted for omitting it by using a high PHperUCP. Dropping these factors is also suggested in other cost models.

Step 5. The adjusted UCP and MUCP will be equal to the unadjusted ones since TCF and EF are set to 1.

Step 6. We assume that there are two mechanisms that consume effort in our model: *E_primary* estimates effort for realizing new and modified use cases, while *E_secondary* estimates effort for *secondary changes* as described below. The total estimated effort is the sum of these.

3.2 Effort Estimation for Secondary Changes of Software

In addition to changes related to functionality estimated in *E_primary*, there are several other reasons for why software is modified:

1. Perfective functional changes not specified in use cases: Functionality is also enhanced and improved in each release by initiating so-called change requests after requirement freeze. There may also be ripple effects of changes in use cases.
2. Perfective non-functional changes: Quality attributes are improved between releases (performance, security, reliability etc.), but these changes are not reflected in use cases Improving quality attributes is usually achieved by modifying software that is already implemented.
3. Corrective changes: Some effort is also spent on modifying software to correct detected defects.
4. Preventive changes to improve design and reduce software decay also consume effort. Also note that preventive changes to improve file structure or to reduce dependencies between software modules may later impact quality attributes such as maintainability.

Table 4 The adapted UCP estimation method

Step	Rule	Output
1	1.1. Classify all actors as average, WF = 2.	UAW = #Actors * 2
	1.2. Count the number of new/modified actors.	Modified UAW (MUAW) = #New or modified actors * 2
2	2.1. Since each transaction in the main flow contains one or several transactions, count each transaction as a single use case. 2.2. Count each alternative flow as a single use case. 2.3. Exceptional flows, parameters, and events are given weight 2. Maximum weighted sum is limited to 15 (a complex use case). 2.4. Included and extended use cases are handled as base use cases. 2.5. Classify use cases as: a) Simple- 2 or fewer transactions, WF = 5 b) Average- 3 to 4 transactions, WF = 10 c) Complex- more than 4 transactions, WF= 15	Unadjusted Use Case Weights (UUCW) = Σ (#Use Cases * WF) + Σ(#Use Case Points for exceptional flows and parameters from 2.3)
	2.6. Count points for modifications in use cases according to rules 2.1-2.5 to calculate the Modified Unadjusted Use Case Weights (MUUCW).	MUUCW = Σ(#New/modified Use Cases * WF) + Σ(Points for new/modified exceptional flows and parameters)
3	3.1. Calculate UUCP for all software.	UUCP = UAW + UUCW
	3.2. Calculate Modified UUCP (MUUCP).	MUUCP = MUAW + MUUCW
4	Assume average project.	TCF = EF = 1
5	5.1. Calculate adjusted Use Case Points (UCP).	UCP = UUCP
	5.2. Calculate adjusted Modified UCP (MUCP).	MUCP = MUUCP

6	6.1. ESTIMATE EFFORT FOR NEW/MODIFIED USE CASES.	*E_PRIMARY*= MUCP * PHPERUCP
	6.2. ESTIMATE EFFORT FOR SECONDARY CHANGES OF SOFTWARE.	*E_SECONDARY* = (UCP - MUCP) * EMF * PHPERUCP
	TOTAL	E = *E_PRIMARY* + *E_SECONDARY*

IV. CONCLUSION

From this paper , I classified two technique of Effort Estimation through Use Case point .Through this evaluate Technical Complexity Factor and Environmental Complexity Factor .It will make very clear and enhanced method for finding out Effort Estimation. One main assumption is that use cases may be used as a measure of the size of a system, and that changes in these may be used as a measure of changes in functionality between releases. Generally, predicting the size of a software system in SLOC or FP in early phases is as difficult as predicting the needed effort. For changes not reflected in use cases, an additional model is used.

The method does not depend on any tools (although there are tools for the original UCP method), paradigms or programming languages, and can promote high quality use cases. The method is cheap, transparent and easy to understand. The method is also suitable when development is being outsourced.

The proposed changes for breaking down use cases and new classification rules were necessary since use cases are written with different level of details in different projects.

Two mechanisms of effort consumption are identified:

a) Primary changes reflected in changes in use cases, and

b) Secondary changes or modification of software from a previous release as a ripple effect of primary changes, unplanned changes and improvements in quality attributes.

Use case diagrams are usually available before other UML diagrams, but have variable level of details. A challenge is to define some standards for these. Furthermore, use cases essentially express functional requirements. The influence of nonfunctional requirements should be included in the technical factors, the number of PH per UCP or as the estimated effort for secondary changes.

There are few empirical studies on estimation of incrementally developed projects. The UCP method showed flexibility in adapting to the context, but there are many project-specific factors in the original method and in our extensions of it. Future studies can help to understand the degree of modification in incremental development of a system (use cases, code and integration costs), and how the method works on other types of systems.

REFERENCES

[1]Boehm, B., Clark, B., Horowitz, E., Westland, C., Madachy, R., Selby, R. Cost models for future software life cycle processes:" *COCOMO 2.0. USC center for software engineering*," information retrieved on 01/03/2012 from, http://sunset.usc.edu/publications/TECHRPTS/1995/index.html
[2]Schneider, G., Winters, J.P. *Applying Use Cases, a Practical Guide*. Addison-Wesley, 1998.
[3] Smith, J. The estimation of effort based on use cases.*Rational Software*, White paper, 1999.
[4] Symons, P.R. *Software Sizing and Estimating MK II FPA(Function Point Analysis)*. John Wiley & Sons, 1991.
[5]Kemerer, C.F. An empirical validation of software cost estimation models. *CACM*, 30, 5 (May 1987), 416- 429