Use Case Points

-An Estimation Approach

Gautam Banerjee

August 2001

Table of Contents

Introduction

Estimates of cost and schedule in software projects are based on a prediction of the size of the future system. Unfortunately, the software profession is notoriously inaccurate when estimating cost and schedule. Preliminary estimates of effort always include many elements of insecurity. Reliable early estimates are difficult to obtain because of the lack of detailed information about the future system at an early stage. However, early estimates are required when bidding for a contract or determining whether a project is feasible in the terms of a cost-benefit analysis. Since Process prediction guides decision-making, a prediction is useful only if it is reasonably accurate.

Traditional cost models take software size as an input parameter, and then apply a set of adjustment factors or 'cost drivers' to compute an estimate of total effort. In object-oriented software production, use cases describe functional requirements. The use case model may therefore be used to predict the size of the future software system at an early development stage. This paper describes a simple approach to software cost estimation based on use case models: the 'Use Case Points Method'. The method is not new, but has not become popular although it is easy to learn. Reliable estimates can be calculated in a short time with the aid of a spreadsheet.

Background

Cost models like COCOMO and sizing methods like Function Point Analysis (FPA) are well known and in widespread use in software engineering. But these approaches have some serious limitations. Counting function points requires experts.

In 1993 the 'Use Case Points' method for sizing and estimating projects developed with the object-oriented method was developed by Gustav Karner of Objectory (now Rational Software). The method is an extension of Function Point Analysis and Mk II Function Point Analysis (an adaption of FPA mainly used in the UK), and is based on the same philosophy as these methods. A few cost estimation tools apply use case point count as an estimation of size, adapting Karner's method. Karner's work on Use Case Point metrics was written as a diploma thesis at the University of Linköping. It was based on just a few small projects, so more research is needed to establish the general usefulness of the method. The work is now copyright of Rational Software, and is hard to obtain.

Use Case Points Based Estimation

An early estimate of effort based on use cases can be made when there is some understanding of the problem domain, system size and architecture at the stage at which the estimate is made. The use case points method is a software sizing and estimation method based on use case counts called use case points.

Classifying Actors and Use Cases

Use case points can be counted from the use case analysis of the system. The first step is to classify the actors as simple, average or complex. A simple actor represents another system with a defined Application Programming Interface, API, an average actor is another system interacting through a protocol such as TCP/IP, and a complex actor may be a person interacting through a GUI or a Web page. A weighting factor is assigned to each actor type.

Actor Type	Weighting Factor
Simple	1
Average	2
Complex	3

The total unadjusted actor weights (UAW) is calculated by counting how many actors there are of each kind (by degree of complexity), multiplying each total by its weighting factor, and adding up the products.

Each use case is then defined as simple, average or complex, depending on number of transactions in the use case description, including secondary scenarios. A transaction is a set of activities, which is either performed entirely, or not at all. Counting number of transactions can be done by counting the use case steps. Karner proposed not counting included and extending use cases, but why he did is not clear. Use case complexity is then defined and weighted in the following manner:

Use Case Type	No of Transactions	Weighting Factor
Simple	<=3	1
Average	4 to 7	2
Complex	>=7	3

Another mechanism for measuring use case complexity is counting analysis classes, which can be used in place of transactions once it has been determined which classes implement a specific use case. A simple use case is implemented by 5 or fewer classes, an average use case by 5 to 10 classes, and a complex use case by more than ten classes. The weights are as before. Each type of use case is then multiplied by the weighting factor, and the products are added up to get the unadjusted use case weights (UUCW).

Yet another way of defining complexity of use cases are as follows:

- If the use case is considered a simple piece of work, uses a simple user interface and touches only a single database entity, the use case is marked as 'Easy'. Rating: 5.
- If the use case is more difficult, involves more interface design and touches 2 or more database entities, the use case is defined as 'Medium'. Rating 10.
- If the use case is very difficult, involves a complex user interface or processing and touches 3 or more database entities, the use case is 'Complex'. Rating: 15.

The UAW is added to the UUCW to get the unadjusted use case points
UAW+UUCW=UUCP

Technical and Environmental Factors

The method also employs a technical factors multiplier corresponding to the Technical Complexity Adjustment factor of the FPA method, and an environmental factors multiplier in order to quantify non-functional requirements such as ease of use and programmer motivation.
Various factors influencing productivity are associated with weights, and values are assigned to each factor, depending on the degree of influence.

0 means no influence, 3 is average, and 5 means strong influence throughout.

See Tables below

Factor	Description	Weight
T1	Distributed System	2
T2	Response adjectives	2
T3	End-user efficiency	1
T4	Complex processing	1
T5	Reusable code	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	Security features	1
T12	Access for third parties	1
T13	Special training required	1

Technical Factors

Factor	Description	Weight
F1	Familiar with RUP	1.5
F2	Application experience	0.5
F3	Object-oriented experience	1
F4	Lead analyst capability	0.5
F5	Motivation	1
F6	Stable requirements	2
F7	Part-time workers	-1
F8	Difficult programming language	2

Environment Factors

The adjustment factors are multiplied by the unadjusted use case points to produce the adjusted use case points, yielding an estimate of the size of the software.

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

TCF=0.6+(0.01*TFactor)

The *Environmental Factor (EF)* is calculated by multiplying the value of each factor (F1-F8) by its weight and adding the products to get the sum called the *EFactor.* The following formula is applied:

EF= 1.4+(-0.03*EFactor)

The *adjusted use case points (UPC)* are calculated as follows:
UPC= UUCP*TCF*EF

Producing Estimates

Karner proposed a factor of 20 staff hours per use case point for a project estimate. Field experience has shown that effort can range from 15 to 30 hours per use case point, therefore converting use case points directly to hours may be an uncertain measure. Steve Sparks therefore suggests it should be avoided. Schneider and Winters suggest a refinement of Karner's proposition based on experience level of staff and stability of the project. The number of environmental factors in F1 through F6 that are above 3 are counted and added to the number of factors in F7 through F8 that are below 3. If the total is 2 or less, they propose 20 staff hours per UCP; if the total is 3 or 4, the value is 28 staff hours per UCP. When the total exceeds 4, it is recommended that changes should be made to the project so that the value can be adjusted. Another possibility is to increase the number of staff hours to 36 per use case point. The reason for this approach is that the environmental factors measure the experience level of the staff and the stability of the project. Negative numbers mean extra effort spent on training team members or problems due to instability. However, using this method of calculation means that even small adjustments of an environmental factor, for instance by half a point, can make a great difference to the estimate.

Contact gautambanerjee@msn.com for further exchange of information.