International Journal of Trend in Scientific Research and Development (IJTSRD)
Volume 3 Issue 5, August 2019 Available Online: www.ijtsrd.com e-ISSN: 2456 – 6470

A Study of Software Size Estimation with use Case Points

Aye Aye Seint

University of Computer Studies, Hinthada, Myanmar

How to cite this paper: Aye Aye Seint "A Study of Software Size Estimation with use Case Points" Published in International Journal of Trend in Scientific Research and Development (ijtsrd), ISSN: 2456-6470, Volume-3 | Issue-5, August 2019, pp.933-936, **https://doi.org/10.31142/ijtsrd26531**

IJTSRD26531

ABSTRACT

Estimates for cost and schedule in software projects are based on a prediction of the size of the system. Software size estimation is the most important role in software cost estimation. Use Case Point method can provide software size estimation at the early stage of the development process. Software size estimation is based on the high-level speciation of Use Case. This paper describes a simple approach to software size estimation base on use case models; the "Use Case Points Method. This model is imported into an estimating tool. To get software size with Use Case Point, the needed factors are the number of use cases and their complexity, the number of actors and their complexity, technical complexity factors (TCF), and environmental complexity factors (ECF). The system computes unadjusted use case points (UUCP), adjusted use case points (UPC), and the total effort in staff hours.

1. INTRODUCTION

The use case is a notional description of a system, frequently used at the earliest stages in a project. The use case is one type of graphically oriented notation in the Unified Modeling Language (UML), a family of notational methods used to describe various aspects of software and its underlying structures. Use cases have three descriptive characteristics, which can be exploited to provide sizing information [10].

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 for software cost estimation based on use case models: the 'Use Case Points Method'.

Use cases provided a high-level description of the intended function of the system. A small application might have only one use case, while very large applications may have hundreds. The numbers of scenarios are the potential outcomes of the software. There is no limit on the number of scenarios that a particular use case may have.

Actors are the "agents" that interact with the software and so, use cases must have at least one actor. A commonly accepted standard is one actor per use case.

In size estimating with use case point, various non-functional requirements are an important role. They are portability, performance, maintainability, security, easy to change and so on, those are not written as a use case.

Cost and effort estimation is an important aspect of the management of software development projects. Experience shows that accurate estimation is difficult. Most methods for estimating effort require an estimate of the size of the software.

The basic formula for converting all of this into a single measure, use case points, is that we will "weight" the complexity of the use cases and actors and adjust their combined weight to reflect the influence of the nonfunctional and environmental factors. Then use case point can be used to calculate the effort of the project[5].

2. SOFTWARE SIZE ESTIMATION

Software measurement is the process whereby numbers or symbols are assigned to entities in order to describe the entities in a meaningful way. Software size must be measured and translated into a number that represents the effort and duration of the project.

Software size can be defined as a set of internal attributes: length, functionality and complexity, and can be measured statically without executing the system. Reuse measures how much of a product was copied or modified, and can also be identified as an aspect of size. Length is the physical size of the product and can be measured for the specification, the design, and the code. Functionality measures the functions are seen by the user. Complexity refers to both efficiency and problem complexity [4].

Estimation may be needed for project estimation and to assess whether processor technology improvements are effective. These productivity estimates are usually based on measuring some attributes of the software and dividing this by the total effort required for the development. There are two types of measure Size-related measures and Function-related measures.

These are related to the size of some output from activity. The most common size-related measures are lines of delivered source code. Other measures which may be used are the number of delivered object code instructions or the number of pages of system documentation [1].

Function-related measures are related to the overall functionality of the delivered software. Productivity is expressed in terms of the amount of useful functionality produced in some given time [1].

3. USE CASE AND SCENARIO

Use cases and scenarios are concerned with the behavior of the system that is visible externally; they aim to establish what the system does from the user's point of view, in terms that the user can readily understand. Although they have become associated with object-orientation, both use case and scenarios are functional and can be used in conjunction with any approach to system development [2],[3].

The relationship between use cases and scenarios is that of the generic to the specific, see Figure 1. A scenario represents one instance of a use case, describing a particular sequence of events that may occur in trying to reach the use case goal. The representations may list what usually happens as follows :

- ➢ user on second-floor press lift button to descend
- ➢ lift button lights up
- ➢ lift button alerts lift to go to the second floor
- ➢ lift goes to the second floor
- ➢ lift button light goes out

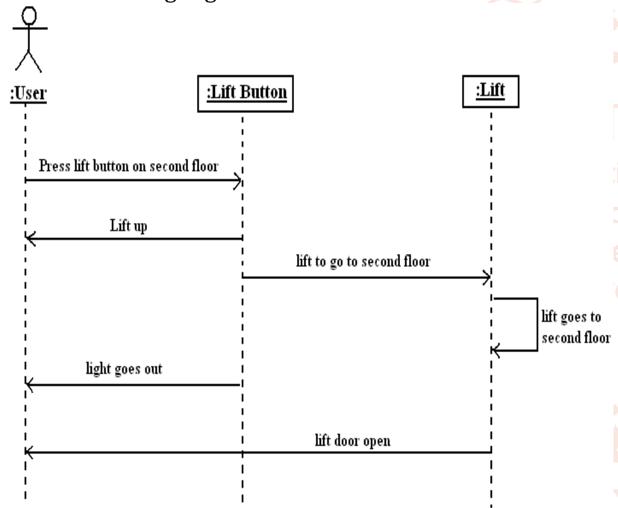
:User
:Lift Button
:Lift
Press lift button on second floor
Lift up
lift to go to second floor
lift goes to second floor
light goes out
lift door open

Figure1. Sequence diagram for a scenario

4. HIGH-LEVEL SPECIFICATION OF USE CASE

The use cases are fundamentally a text form, although they can be written using flow charts, sequence charts, Petri nets, or programming language. Under normal circumstances, they serve to communicate from one person to another, often to people with no special training. The simple text is, therefore, usually the best choice [6].

Many people think that the ellipses are the use cases, even though the ellipses convey very little information. The use case description, as a form of writing, can be put into service to stimulate discussion within a team about an upcoming system. They might later use the use case from the document the actual requirements. Another team might later document the final design with the same use case form. They might do this for a system as large as an entire company, or as small as a piece of a software application program [6].

The example of the high level specification of use case is;-

Use Case Name: Buy stocks over the web

Primary Actor: Personal Advisors

Level: User goal

Precondition: User already has PAF open.

Stakeholders and interests: Purchaser- wants to buy stocks, get them added to the PAF portfolio automatically.

Stock agency- wants full purchase information.

Precondition: User already has PAF open.

Minimal guarantee: sufficient logging information that PAF can detect that something went wrong and can ask the user to provide details.

Success guarantee: remote web site has acknowledged the purchase, the logs and the user's portfolio are updated.

Main success scenario:

The user selects to buy stocks over the web PAF gets the name of the web site to use from the user.

PAF open web connection to the site, retaining control.

User browses and buys stock from the web site.

PAF intercepts responses from the web site and updates the user's portfolio.

PAF shows the user the new portfolio standing.

Extensions: User wants a web site PAF does not support. The system gets new suggestion from the user, with the option to cancel use case.

Web failure of any sort during step: 3a1. System reports failure to the user with advice backs up to the previous step.

The user either backs out of this use case or tries again.

Computer crashes or gets switched off during a purchase transaction;

When the use cases document an organization's business process, the system under discussion is the organization itself. The stakeholders are the company shareholders, customers, vendors, and government regulatory agencies. The primary actors will include the company's customers and perhaps their suppliers.

5. THE USE CASE POINT METHOD

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 [4].

5.1. Classifying Actors and Use Case

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 [8],[9].

Actor type: Simple, weighting factor 1
Actor type: Average, weighting factor 2
Actor type: Complex, weighting factor 3

The total unadjusted actor weights (UAW) is calculated by counting how many actors there are of each kind (by the 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 a 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. Use case complexity is then defined and weighted in the following manner:

Simple: 3 or fewer transactions, weighting factor 5
Average: 4 to7 transactions, weighting factor 10
Complex: More than 7 transactions, weighting factor 15.

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 areas 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)[5].

The UAW is added to the UUCW to get the unadjusted use case points (UUPC):

UAW+UUCW=UUCP

5.2. Technical and Environmental Factors

The application of use case modeling for analyzing the functional requirements of a system. Because it focuses on how the system delivers or should deliver those functions, a use case model is developed in the analysis phase of the object-oriented system development life cycle

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 [7],[8]. 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 Table 1 and Table 2.

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

Table1. Technical complexity factor

Factor	Description	Weight
T1	Distributed system	2
T2	Performance objectives	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 use	1
T11	Security	1
T12	Access for third parties	1
T13	Training needs	1

Table2. Environmental factor

Factor	Description	Weight
E1	Familiar with the development process	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 staff	-1
E8	Difficult programming language	-1

6. STUDY OF PROJECT

Cost and effort estimation is an important aspect of the management of software development projects. Experience shows that accurate estimation is difficult. Most methods for estimating effort require an estimate of the size of the software.

Once a size estimate is available, models can be used that relate size to effort. Most cost estimation models attempt to generate an effort estimate, which can then be converted into the project duration and cost. Although effort and cost are closely related, they are not necessarily related by a simple transformation function.

Cost and effort estimation is an important aspect of the management of software development projects [9]. Experience shows that accurate estimation is difficult. Accurate software cost estimates are critical to both developers and customers.

This paper describes the use case point method applied to the project. Estimation results are computed for ten small projects with the use case point method and compared to estimates with actual effort. The main goal of the investigations is to determine the appropriate level of detail in use case descriptions written for estimation purposes and how different actual and estimate effort result.

As seen from the results present in Table 3, use cases must be written at a user goal level of detail if they are to be used effectively for estimation purposes. If descriptions are lacking in detail, and there is a doubt as to if all the functionality has been correctly described.

The results show that as opposed to the technical complexity factors, the environmental factors play a significant part in the estimates. Setting the scores too high leads to underestimation.

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 is 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.

Table3. The comparison result of projects

Project No	Estimate Effort(Hour)	Actual Effort (Hour)
1	418	420
2	282	298
3	197	212
4	335	341
5	394	420
6	519	500
7	132	143
8	605	595
9	610	598
10	520	499

When the total exceeds 4, it is recommended that changes should be made to the project so that the value can be adjusted.

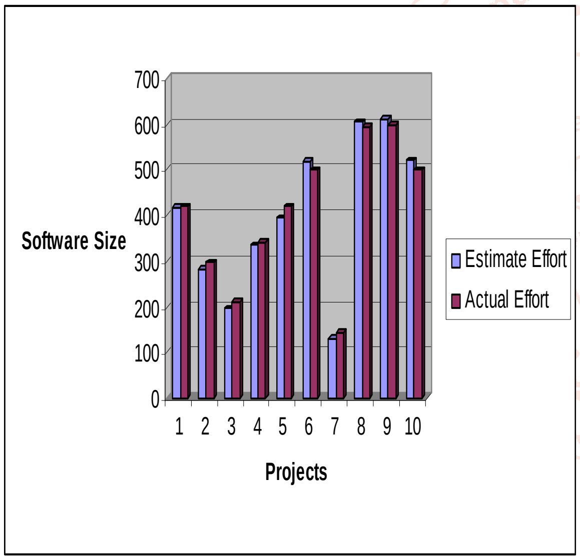

Figure2. Compare the result of estimate effort and actual effort (staff hour)

7. CONCLUSION

The system is calculated on applying a method for estimating software development effort based on use cases, the use case points method. The results indicate that this method can be used successfully since the use case estimates were close to the expert estimates. It is, therefore, our impression that the method may support expert knowledge. This system intends to further study the precision of the use case point method compared with expert estimates. The system will conduct a study where the estimators have different levels of experience.

The trouble with sizing use cases that are lacking in textual detail became evident. Sizing becomes time-consuming, and there that the functionality is not expressed well enough, and that the system will be underestimated.

This paper also would be useful to investigate how the use case points method, which provides top-down estimates based on a measure of size, can be combined with other methods that provide bottom-up estimates. The purpose of using the estimation method investigated in this paper is to provide a complete estimate for all the activities in the project.

REFERENCES

[1] Ian Sommerville, Software Engineering, eighth Editions, Addison Wesley, 2009

[2] Grady Booch, James Rumbaugh, Ivan Jacobson, the Unified Modeling Language User Guide, May 2003.

[3] Carol Britton (University of Hertfordshire) Jill Docke (Angli Polytechnic University), Object-Oriented System Development, McGraw-HILL INTERNATIONAL EDITION, 2001.

[4] Kirsten Ribu, Estimating Object-Oriented Software Projects With Use Case, The University of Oslo, 7th November 2001.

[5] Schneider and Winters, "Applying use Cases". Addison-Wesley, 1998

[6] Alistair Cockburn, Writing Effective Use Cases, 21st February 2000

[7] Bente Anda, "Effort Estimation of Use Case for Incremental Large-Scale software Development", Norwegian University of Science and Technology (NTNU)

[8] Bente Anda, HegenDreiem, Dag I.K Sjoberg and Magne Jorgensen, "Estimating Software Development Effort based on Use Cases Experiences from Industry", Department of Informatics, University of Oslo. NORWAY.

[9] Gautam Banerjee, "Use Case Points An Estimation Approach", August 2001.

[10] J. Kammelar, "A Sizing Approach for OO-environments". In forth International ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering, Cannes, 2000.