Research Journal of Applied Sciences 2 (9): 1012-1015, 2007
ISSN: 1815-932X

Development in Source Line of Code, Function Point Analysis and User Case Point Analysis-A Review

[1]D. Lakshmanan, [2]V.Gunaraj, [3]M. Karnan, [3]R. Sivakumar
[1]Kumaraguru College of Technology, Coimbatore, Tamil Nadu, India
[2]RVS Engineering College, Coimbatore, Tamil Nadu, India
[3]Tamilnadu College of Engineering, Coimbatore, Tamil Nadu, India

Abstract: During the past three decades there had been some significant developments in effort estimation, size of the software, cost estimation methodology. This study describes development in Source Line of Code (SLOC), Function Point Analysis (FPA) and User Case Point Analysis (UCPA). Some of these developments are refinements and experience of earlier methodology. The review identifies SLOC as the basic effort estimation method and Function Point Analysis (FPA) as the technology and platform independence and it's available from early requirement phase. In the function point analysis users can readily understand about their software and its applications, from the customers and not the technician's perspective. In use case point had a development with the object oriented method and is a software sizing and estimation based on use case counts called use case points. Other new developments include, bottom up approach, top down approach, algorithmic method, putnam's model, graphical user interface metrics, cocomo and object metrics to assist the effort estimation in selecting the appropriate model for a specific case. Each of these developments is discussed in detail.

Key words: Source line of code, function point, use case point, software effort

INTRODUCTION

Source line of code: At the early stage SLOC was an effort measuring tool for function and assembly language because these languages are line oriented languages and developers used punch card data entry system.

Source Line of Code is used to measure the amount of code in a software program and is used to estimate the amount of effort that will be required to develop a program. It is also used to estimate the productivity and effort once the software is produced. SLOC measurement categorizes into two types, one is physical SLOC and the other is logical SLOC. Physical SLOC is a count of lines in the text of the program's source code including command lines. In this, blank line is also counted and more than 25% is not included. Logical SLOC is used to measure the number of statements and it is specific to computer languages. The larger value of SLOC needs more effort to develop a program (Albrecbt and Gaffney, 1983).

FUNCTION POINT ANALYSIS

Function Point Analysis (FPA) is a method to break systems into smaller components, so they can be better understood and analyzed. It is used to produce an estimate of software size from software measurement, comparison and analysis and also to find software size for input into software cost estimation models and tools that output effort (staff hours) which is based on empirical cost estimation relationships between function points and efforts (David longstreet).

The workflow diagram for function point analysis is given in Fig. 1.

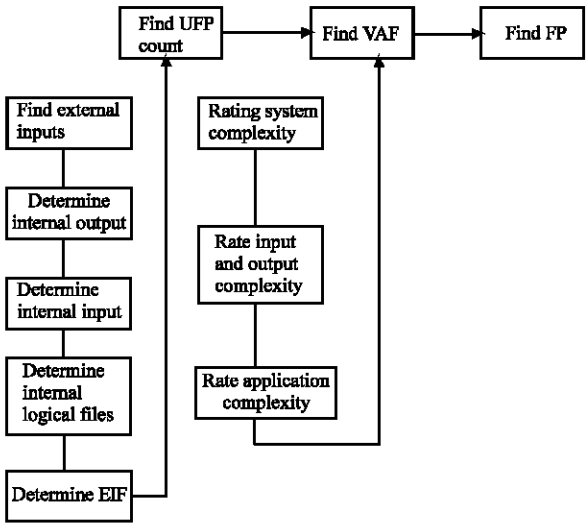

Fig. 1: Work flow diagram for FPA

Corresponding Author: D. Lakshmanan, Kumaraguru College of Technology, Coimbatore, Tamil Nadu, India

In function point analysis first function point is measured for external inputs, output external inquires, Internal logical files and external interface files with weighing factors and value adjustment factor are given for transactions, data processing, data communications, conversion and installation based on experience and procedure given by International Function Point User Group (IFPUG). Finally fatal function point is calculated by

$$FP = UFP \times (0.65 + .01 \times VAF)$$

FEATURE POINT

It is the extension of function points to include algorithms as a new class (Jones, 1997). An algorithm is defined as the set of rules which must be completely expressed to solve significant computational problems completely. Each algorithm used is given a weighing factor from 1 (simple) to 10 (highly complex) and function point is the weighted sum of the algorithms and function points. It is useful with few input/output and to solve high algorithm complexity problems.

TOP-DOWN AND BOTTOM-UP EXPERT EFFORT ESTIMATION

The total effort estimate may be based on properties of the project as a whole and distributed over project activities (top down), or calculated as the sum of the project activity estimates (bottom up). Expert estimation can be described as estimation conducted by persons recognized as experts where important parts of the estimation process is based on non explicitly non recoverable reasoning processes. It includes estimation strategies in the interval from completely un-aided judgment supported by historical data, process guidelines and check lists. It is more flexible regarding required estimation input and time spent on producing the estimate.

When applying top down estimation the total effort of a software project is estimated with a decomposition of the project into activities. In this estimation strategy is to compare the current project as a whole with previously completed similar projects. The estimated total effort is then distributed over activities applying for excess guidelines about the activities constituting typical proportion of the total effort (Molokken, 2002).

In bottom up estimation the project is typically divided into activities and effort of each activities are estimated. The project's estimate of the total effort is the sum of the expert estimates of each project activity possibly with the addition of the effort budget to cover unexpected activities and events (Connolly and Dean, 1997).

ESTIMATION BY ANALOGY

It is the simplified process of finding one or more projects that are similar to the one to be estimated and then deriving the estimate from the values of these projects. If the selected projects have an unusual high or low productivity then we should adjust the estimates toward productivity values of more average projects. It will compare the actual costs of previous projects by analogy. This can be done either at the total project level or at subsystems. The strength of the method is that the effort estimate is based on actual project experience (Walkerden and Jeffery, 1999).

ALGORITHMIC METHOD

Algorithmic method is a mathematical model and it involves cost factors

$$Effort = f(x_1, x_2, \ldots\ldots x_n)$$

$x_1, x_2 \ldots\ldots x_n$ denote the cost factors such as product, computer, personnel and project factors in linear models,

$$Effort = a_0 + \sum_{i=1}^{n} a_i x_i$$

Where, $a_i \ldots\ldots a_n$ are co-efficients selected from completed project data

The research of Nelson belongs to this model (Nelson, 1966)

Multiplicative models have the following form

$$Effort = a_0 \prod_{i=1}^{n} a_i x_i$$

Walston-Felix used this model, taking three possible values -1, 0 and +1. In power function model

$$Effort = a \times s^b$$

Where s = size of the code

a, b are functions of cost factors.

CONSTRUCTIVE COST MODEL COCOMO

The COCOMO research team at the Centre for Software Engineering at University of Southern California, Boehm (1981) formed this model and the code size is given in SLOC and effort in person month. In basic COCOMO model there are three sets of factors a and b depending on the complexity of the software.

$$Effort = a \times s^b$$

- For simple problems, a = 2.4, b = 1.05
- For complex problems, a = 3, b = 1.15
- For embedded softwares, a = 3.6, b =1.2

In intermediate COCOMO nominal effort estimation is determined with three sets of a, b with a slightly different from basic model.

- For simple problems, a = 3.2 b = 1.05
- For complex problems, a = 3 b = 1.15
- For embedded softwares, a = 2.8 b = 1.2

In cocomo II the exponent changes according to the cost factor. It is based on 5 scale factor namely pre sedateness, development flexibility, risk resolution, team cohesion and process maturity. It is a non linear re usable formula breakage rating which is used to address requirement volatility and auto calibration facilities (Putnam, 1978).

PUTNAM'S MODEL

This model is based on Norten/Rayleigh man power distribution (Putnam, 1978)) and also is called Software Equation which is as follows

$$S = E \times (effort)1/3 \times td^{4/3}$$

Where;
E = Environment factor
Td = Software Delivery time
Effort is in person year

Putnam formed another relation based on manpower buildup

$$Effort = D_0 \times td^3$$

Do = Manpower buildup

SLIM, a software tool based on Putnam's equation for effort, is used as a model for estimation of effort.

USE CASE POINT ANALYSIS

The use case points method is inspired by the function points method. It was proposed by Gustav Karner of Objectory (Now Rational Software) in 1993. It is an object oriented method. In this method, use cases are used to measure the functionality size of the system to be developed. For estimation of effort the quantity of functionality is very important. The use case points method is a software sizing and estimation based on use case counts called Use Case Points. The use case point is three step (Smith, 1999) procedure.

Step I: The actors are classification of actors. A single actor is another system with a defined application programming interface. Average actor is another system infecting through a protocol such as TCP/IP and complex actor may be person interacting through a Graphical User Interface (GUI) environment. The weighting factor based on complexity is given in Table 1.

The total Unadjusted Actor Weights (UAW) is calculated by counting actors, multiplying each total by its weighting factor.

Step II: The use cases are classified into simple, average or complex based on number of transactions based on the use case description (Karner, 1993) and number of scenarios (Vinsen *et al*., 2004). And it is tabulated in Table 2.

Each type of use cases are multiplied by the weighing factor and the products are added up to get the Un-adjusted Use Case Weights (UAUCW).

Unadjusted Use Case Points = Unadjusted weight UAW + Unadjusted use case weight.

The productivity factor i.e. the number of hours necessary to realize one use case point is calculated based on technical factors.

Step-III: The Environmental Factors is EF is calculated by multiplying the value of each factor (F1-F8) by its weight and adding the products to get the sum of the E Factor EF=1.4+(-0.03 × E factor) as given in Table 3.

The adjusted use case points

$$UPC = UUCP \times TCF \times EF$$

Table 1: The weighting factor

Actor	Weighting factor
Simple	1
Average	2
Complex	3

Table 2: Un-adjusted use case weights

Use Case	No. of transaction	Weighing factor
Simple	< = 3	1
Average	4 to 7	2
Complex	> = 7	3

Table 3: Environmental factor

	Description	Weight
F_1	Familiar with RUP	1.5
F_2	Application experience	0.5
F_3	Object oriented experience	1
F_4	Lead analyst capacity	0.5
F_5	Motivation	1
F_6	Stable requirement	2
F_7	Patience working	-1
F_8	Difficult programming language	2

Table 4: Technical complexity factor

Factor	Description	Weight
T_1	Distributed system	2
T_2	Response adjectives	2
T_3	End user efficiency	1
T_4	Complex process	1
T_5	Recursive case	1
T_6	Easy to install	0.5
T_7	Easy to use	0.5
T_8	Portage	2
T_9	Easy to change	1
T_{10}	Concurrent	1
T_{11}	Security features	1
T_{12}	Seeses for find parties	1
T_{13}	Special training required	1

The TCF is multiplied by original productivity factor. In Table 4 technical complexity factor and its weightage is given.

CONCLUSION

Today effort estimation in software industry is a complex problem which attracts considerable research attention. Recently artificial intelligence and case based reasoning (Finnie and Witting, 1996) used data set from ASMA (Australian Software Metrics Association). Today the model can estimate the effort required for software development with high degree of accuracy. The present effort of International Software Benchmarking Standards Group (ISBSG) has established repository of 790 projects (ISBSG). The effort estimates using different methods based on the situation and time pound and the accuracy of estimation of the effort also reduced due to development environment, lack of complexity measurement and large number of inter related papers. For best estimation of efforts, the estimation should improve their knowledge in project attributes and its relationship.

REFERENCES

Albrecbt, J. and J.E. Gaffney, 1983. Software function, source lines of codes and development effort predication a software science validation. IEEE. Trans. Software Eng., pp: 639-648.

David longstreet-www.softwaremetrices.com.

Jones, C., 1997. Applied Software Measurement. Assuring Productivity and Quality. McGraw Hill.

Top down and Bottom up Expert estimation of software development effort Magne Jorgensen. Simula research laboratory, Norway.

Molokken, K., 2002. Expert estimation of web development effort. Individual biases and group processes (master thesis) in Department of Informatics University of Oslo.

Connolly, T. and D. Dean, 1997. Decomposed versus holistic estimates of effort required for software writing tasks management science, 43: 1029-1045.

Software effort estimation by analogy and regression towards the man Magne Jorgensen, UIF Indatil Daj Ejaberg Simula Research Lab, Oslo University.

Walkerden F. and R. Jeffery, 1999. An empirical study of analogy based software effort estimation Empirical Software Eng., 4: 135-158.

Nelson, R., 1966. Management handbook for the estimation of computer programming costs ADA 48750 Systems Development Corp.

Boehm, B.W., 1981. Software Engineering Economics Englewood Cliffs N.J. Prentice Hall.

Miyazaki, Y. and Mori Cocomo, 1985. Evaluation and tailoring 8th Int. Conf. Software Eng., pp: 292-299.

Putnam, L.H., 1978. A general empirical solution to the macro software sizing and estimating problems. IEEE. Trans. Soft Eng., pp: 345-361.

Smith, J., 1999. The estimation of effort based on use case rational software white paper, pp: 171-99.

Karner, G., 1993. Resource Estimation for objectory projects objective systems SF AB 7.

Vinsen, K., D. Jamieson and G. Callender 2004. Use Case Estimation. The Devil is in the details the 12th International requirements Engineering Conference (RE) Kyoto Japan, pp: 10-15.

Finnie, G.R. and G.E. Wittig 1996. AI tools for software development effort estimation. Software Engineering and Education and Practice Conference. IEEE. Computer Society Press, pp: 346-353.

ISBSG.org/au.