Proceedings of the International MultiConference of Engineers and Computer Scientists 2013 Vol II,
IMECS 2013, March 13 - 15, 2013, Hong Kong

Early Stage Software Effort Estimation Using Function Point Analysis: Empirical Evidence

Tharwon Arnuphaptrairong

Abstract—Software effort and cost estimation are necessary at the early stage of the software development life cycle for the project manager to be able to successfully plan for the software project. Unfortunately, most of the estimation models depend on details that will be available at the later stage of the development process. This paper proposes to use Function Point Analysis in application with dataflow diagram to solve this timing critical problem. The proposed methodology was validated through the graduate student software projects at the Chulalongkorn University Business School. Although the results were disappointed but some interesting insights are worth looking into.

***Index Terms*—software effort estimations, early stage software effort estimation, early stage Function Point Analysis, software effort empirical evidence.**

I. Introduction

Software effort and cost estimation are necessary at the early stage of the software development life cycle for the project manager to be able to successfully plan for the software project. Unfortunately, most of the estimation models depend on details that will be available at the later stage of the development process. For example, the object oriented estimation models depend on the UML models – Use cases, Class diagrams and so on, which will not be available until the design stage. This situation –the need for information at the early stage but is available at the later, is referred to as software estimation paradox [1]. This paper proposes to use dataflow diagram to solve this timing critical problem. At the requirement stage, the DFD can be used to depict the functionality of the software system. The information available in the dataflow can be used to obtain Function Points and serve as the basis for software effort estimation.

This article is organized as follows. Section II overviews the software effort estimation methods related to our proposed methodologies i.e., Function Point Analysis, early Function Points, the Function Points estimation from data flow diagram method and COCOMO cost estimation model. Section III describes the proposed methodology. Section IV presents the empirical results. The discussions and the conclusions for this research are presented in section V and VI respectively.

T. Arnuphaptrairong is with the Department of Statistics, Chulalongkorn Business School, Chulalongkorn University, Bangkok 10250 Thailand (e-mail: Tharwon@acc.chula.ac.th).

II. Overview of Related Literature

This section reviews the software effort and cost estimation methods related to our proposed methodologies i.e., Function Point analysis, early Function Points, Function Points estimation from data flow diagram method and COCOMO cost estimation model.

A. Function Point Analysis

Function Points (FP) was originated in 1979 and widely accepted with a lot of variants, from both academics and practitioner [2]. The research in this area is also known as **Function Point Analysis (FPA) or Function Size Measurement (FSM)**. The FP measurement could be classified into FP counting and estimation [3].

Function Point was introduced by Albrecht [4], the concept is based on the idea that the functionality of the software delivered is the driver of the size of the software (Line of Codes). In other words, the more the functions delivered, the more the Line of Codes. The functionality size is measured in terms of Function Points (FP).

FPA assumes that a software program comprises of functions or processes. In turn, each function or process consists of five unique components or function types as shown in Figure 1. The five function types are External Input (EI), External Output (EO), External Query (EQ), Internal Logical File (ILF), and External Interface File (EIF).

Each of these five function types is individually assessed for complexity and given a Function Points value which varies from 3 (for simple external inputs) to 15 (for complex internal files). The Function Points values are based the complexity of the feature being counted.

The low, average and high complexity level of ILF and EIF are based on the number of Record Element Type (RET) and Data Element Type (DET). A Record Element Type (RET) is a subgroup of the data element (record) of an ILF or ELF. A data element type is a unique non-repeated data field.

The complexity level of EI and EO and EQ are based on the number of File Type Referenced (FTR) and Data Element Type (DET). A File Type Referenced (FTR) is an ILF or EIF.

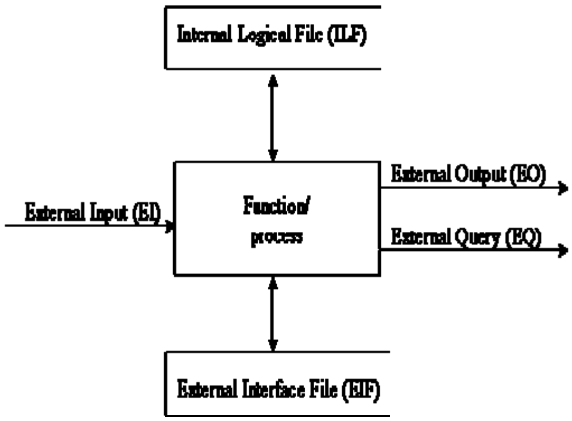

Fig. 1. The Albrecht five function types

The Unadjusted Function Points (UFP) or Unadjusted Function Points Counts (UFC) is calculated as follows [4]:

The sum of all the occurrences is computed by multiplying each function count (N) with a Function Point weighting (W) in Table I, and then the UFP is attained by adding up all the values as follows:

$$\text{UFP} = \sum_{i=1}^{5}\sum_{j=1}^{3} N_{ij}W_{ij}$$

Where Nij is the number of the occurrences of each function type i of the five types and Wij is the corresponding complexity function point weighting value j of the 3 levels –low, average and high.

TABLE I

THE FUNCTION POINT WEIGHTS

	Complexity		
Function Type	**Low**	**Average**	**High**
External Input	3	4	6
External Output	4	5	7
External Inquiry	3	4	6
Internal Logical File	7	10	15
External Interface File	5	7	10

The Function Point values obtained can be used directly for estimating the software project effort and cost. But in some cases, it may need further adjustments with the software development environment factors.

In order to find adjusted FP, UFP is multiplied by technical complexity factors (TCF) which can be calculated by the formula:

$$\text{TCF} = 0.65 + (\text{sum of factors}) / 100$$

There are 14 technical complexity factors --data communications, performance, heavily used configuration, transaction rate, online data entry, end user efficiency, online update, complex processing, reusability, installation ease, operations ease, multiple sites, facilitate change, distributed functions. Each complexity factor is rated on the basis of its degree of influence from no influence (0) to very influential (5). The adjusted Function Points (FP) or Function Point Counts (FC) is then derived as follows:

$$\text{FP} = \text{UFP x TCF}$$

The International Function Point User Group (IFPUG) is the organization establishes the standards for the Function Point Size Measurement to ensure that function points counting are the same and comparable across organizations**.** The counting manual can be found at http://www.ifpug.otg.

The International Standard Organization (ISO), in 1996, established the common standard, in order to support the consistency and promote the use of this Function Size Measurement (FSM). The updated versions are maintained. Besides the IFPUG FPA, three other FPA variants are also certified methods by ISO --Mk II, NESMA, and COSMIC FFP.

B. Early Function Points

Early Function Points (EFP) and Extended Function Points (XFP) were proposed by Meli [5], to anticipate for the need of software size estimate at the early stage of the development life cycle. The method requires the estimator to put in knowledge at different detail levels of a particular application. Functionalities are classified as: Macrofunction, Function, Microfunction, and Functional Primitive. Each type of functionality is assigned a set of FP value (minimum, average, and maximum). The **Early Function Points (EFP) and Extended Function Points (XFP)** are considered not very easy to used.

C. Function Points estimation from data flow diagram method

Functionality is the heart of FPA. One stream of research proposed that functionalities can be retrieved using Structured Analysis (SA) which expressed in the form of Dataflow Diagram (DFD) for process modeling and Entity Relationship Diagram (ERD) for data modeling.

DFD was proposed as the estimator for FPA by a number of papers using either DFD alone or together with ERD [6]-[12].

Rask [6, 7] introduced the algorithm for counting the Function Points using specification from DFD and ERD data model. The automated system was also built.

O'brien and Jones [8] proposed a set of counting rules to incorporate Structured Analysis and Design Method (SSADM) into Function Point Analysis. DFD, together with I/O structure diagram, Enquiring Access Path (EAP) and

Effect Correspondence Diagram (ECD) were applied to the counting rules for the Mark II FPA.

Shoval and Feldman [9] applied Mark II Function Points with Architectural Design of Information System Based on structural Analysis (ADISSA). The proposed method counts the attributes of all inputs and outputs from the Dataflow Diagram (DFD) of the system to be built and all of the relations in the database from the database design process, and then plugs in all the numbers in the Mark II model.

DFD was found also proposed to be used together with ERD in [10]. Lamma et al. [11] to solve the problem of counting error, a system for automating the counting is built and called FUN (FUNction points measurement). The system used the specification of a software system from Entity Relationship Diagram and Dataflow diagram to estimate software Function Points. Later, the system was automated by Grammantieri *et al* [12].

D. COCOMO Cost Estimation Model

COCOMO (Constructive Cost Model) was originated by Boehm in 1985 [13]. The model was based statistical analysis of data of 63 software development projects. By performing regression analysis on the of 63 software development projects, the following basic model software development effort estimation model was derived:

$$\text{Effort} = c\,(\text{size})^{k}$$

Where:

effort was measured in person month (pm) or the number of person months, a person month is of 152 person hours,

Size is the estimated size of the software, measured in kdsi (Kilo Source Instructions) and

c and k are constants.

III. The Proposed Methodology

One of the problems associated with FPA is the need for the early stage of the software development life cycle [5, 14]. According to IFPUG standard counting rules Functions specification should already be clear at least from 15 to 40% before FPs can be obtained. Otherwise it would not be possible to identify EI, EO, EQ, ILF and EIF [5, 15].

This research proposes to handle this problem by utilizing functional requirements available in the DFD at the requirement determination stage --early stage of the development life cycle. Using DFD is not new. At least two algorithms using DFD had been proposed for FP counts [6, 9]. To attain the FP counts using DFD, the proposed method adapted the method proposed by [6, 9].

The proposed method is to count for only EI, EO and ILF where EI is the data flow from external entity into the system, EO is data flow from the system to external entity, and the ILF is the file used inside the system. The DFD of the sales order system shown in Fig. 2 will be used to demonstrate how the proposed method works.

There are 3 functions, --Fill Order, Create Invoice and Apply Payment in the sale order system.

The Fill Order function consists of 1 EI (order from customer), 1 EO (packing list), and 3 ILF (customer file, product file, and order file)

The Create Invoice function consists of 1 EI (completed orders from warehouse), 1 EO (invoice), and 4 ILFs (customer file, product file, order file, and accounts receivable file)

The Apply Payment function consists of 1 EI (payment from customer), 3 EOs (cash receipt entry, bank deposit and commission report), and 1 ILF (accounts receivable file)

The numbers of Function Point counts of this software are then achieved by applying corresponding FP weightings to the EI, EO and ILF as follows:

$$FP = \sum_{i=1}^{3}\sum_{j=1}^{3} N_{ij}W_{ij}$$

$$\begin{aligned} FP &= (1*3 + 1*4 + 3*7) + (1*3 + 1*4 + 4*7) \\ &\quad + (1*3 + 3*4 + 1*7) \\ &= 28 + 35 + 22 \\ &= 85 \end{aligned}$$

Next, the size of the software is attained by multiplying the FP counts with the average number of line of codes per function point of the programming language used [16] to transform the FP counts to number of line of codes (LOC). Suppose that this software is to be implemented with C++, the average number of line of codes per function point for C++ is 53. The number of line of codes (LOC) for this software is calculated as follows:

$$\begin{aligned} LOC &= 85 * 53 \\ &= 4{,}505 \end{aligned}$$

And the required effort is then estimated using COCOMO cost estimation model as follows:

$$\begin{aligned} \text{Effort} &= c * (\text{size})^{k} \\ &= 2.4 * (4.505)^{1.05} \\ &= 11.66 \text{ person months} \end{aligned}$$

ISBN: 978-988-19252-6-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

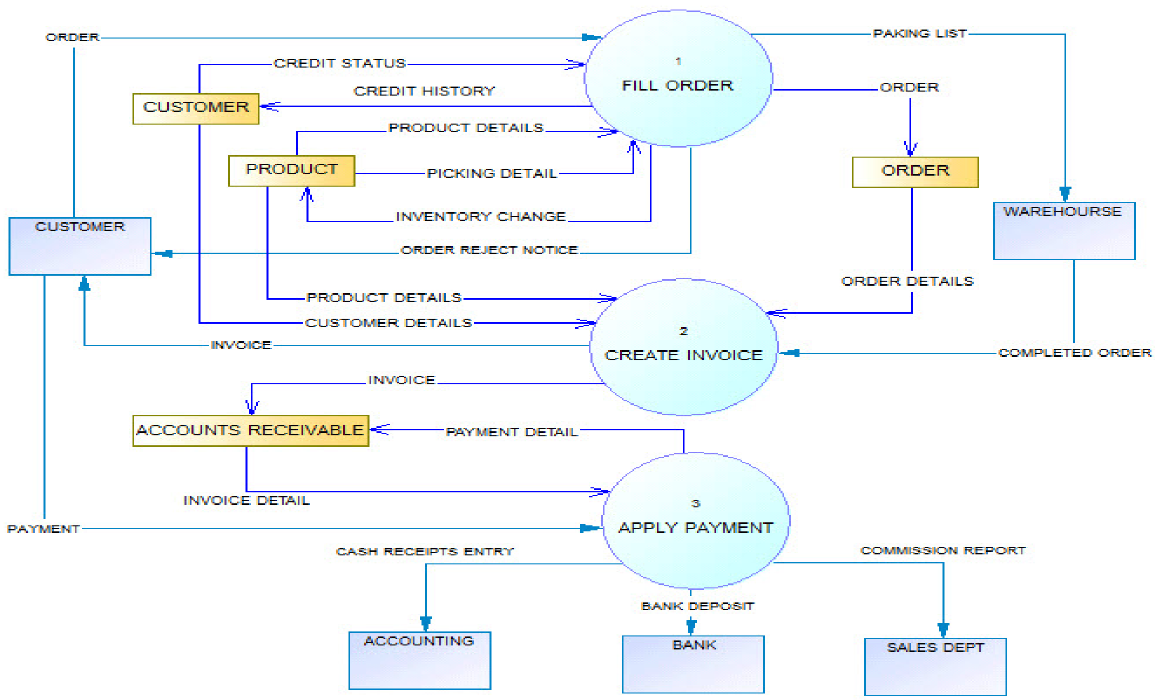

Fig. 2. First level DFD of a Sales Order System

IV. Empirical Validation

The proposed methodology was validated with graduate student projects in the master program in Business Software Development at the Department of Statistics, CBS Business School, Chulalongkorn University. The graduate students are required to have at least three year of experience in the software industry for the admission. After finishing 36 credits of course works, the students are required to have a 6 credits master project to develope a business software package.

A batch of the 25 graduating students of the master program was asked to participate in the experiment using the proposed methodology described in section III. When the students passed the project proposal, a questionnaire was distributed to each student to ask for the following information: student identification, name, programming experience, project name, start date, number of functions appeared in the DFD, languages and tools used, and the estimated function points. Finished date and actual effort in man hours used for the software projects were filled out in the same form by the students again when the projects were completed.

Fifteen students returned the final results. Table II shows the background data of the projects. The table shows the name of the software developed, languages and tools used, number of functions in the software, and the actual man hours spent in developing the software. FP-EST is the estimated unadjusted Function Points from DFD with the algorithm as explained in section III. On average, there are of 8 functions, 305.4 Function Points and 534 man hours employed per project.

Of the 15 questionnaires, two students did not return the final actual effort used for the projects. This resulted in only 13 usable project data sets. The analysis was carried on with the 13 projects. From the data gather in table II, the software size in source line of codes (LOC) was calculated by multiplying language factors (58 LOC for C#, 28 LOC for VB.net and 56 LOC for PHP). Then, the COCOMO model was used to find the effort needed to develop the software with c=2.4, and k=1.05 (Basic COCOMO Model).

The estimated effort obtained in person months was then multiplied by 152 to get the man hours. The accuracy of the estimation was measured using MRE (Magnitude of Relative Error) which is the absolute value of (estimated man hours – actual man hours / actual man hours). The results are shown in Table III. The results from table III show very high MRE with average of 1624.31%. The figures also show over estimates of the estimated effort for all projects. Two projects --3 and 13 are obviously outliers. With the two outliner projects removed, the MRE was improve to 517.84%

TABLE II

PROJECT BACKGROUND DATA

No.	Software	Language and Tool used	No of Functions	FP-EST	Man hours
1	Tap Water Production Maintenance and Service Using GPS System	C#.net, VS-Studio	5	207	400
2	Investment Support System	VB.net, ASP.NET, SQL server	12	564	1016
3	Software Inspection processing Compliance Support System	C#.net, VS-Studio	9	1321	560
4	Thai Language Data Mining for Marketing Research	PHP, SQL server	4	83	640
5	Vegetable Box Project Management Software	VB.net, SQL server	8	220	240
6	Personal Loan Follow up Management System	ASP.NET	9	139	N/A
7	Intelligent Room Assignment Dormitory System	PHP, SQL server	5	65	400
8	Software Supporting Buffet Business Via Web	C#.net, VS-Studio	9	254	458
9	Visual Challenge Library Support System	C#.net, ASP.NET,VS-Studio	7	186	384
10	Beauty Business Information System	VB.net, SQL server	12	112	550
11	Gold Retailing Business Information System Using RFID	VB.net, SQL server	11	110	600
12	Appraisal System	VB.net, C++,SQL server	5	268	520
13	Primary School Teaching Support System	C#.net, VS-Studio	7	154	1080
14	Electronic Menu Restaurant Support System	VB.net, Java Script	8	707	95
15	Software for Visual Challenge person travelling with public transportation	C#, Java,VB.net	9	191	N/A
		Average	8	305.4	534

TABLE III

THE RESULTS FROM THE ANALYSIS

No.	Language	FP-EST	LOC per FP	Estimated KLOC	Estimated Man months	Estimated Man hours	Actual man hours	MRE (%)
1	C#.net	207	58.00	12.01	32.63	4959.33	400.00	1139.83
2	VB.net	564	28.00	15.79	43.51	6613.23	1016.00	550.91
3	C#.net	1321	58.00	76.62	228.43	34721.80	560.00	6100.32
4	PHP	83	56.00	4.65	12.05	1830.98	640.00	186.09
5	VB.net	220	28.00	6.16	16.19	2461.02	240.00	925.42
6	PHP	65	56.00	3.64	9.32	1416.48	400.00	254.12
7	C#.net	254	58.00	14.73	40.45	6147.94	458.00	1242.34
8	C#.net	186	58.00	10.79	29.16	4432.44	384.00	1054.28
9	VB.net	112	28.00	3.14	7.97	1211.29	550.00	120.24
10	VB.net	110	28.00	3.08	7.82	1188.59	600.00	98.10
11	VB.net	268	28.00	7.50	19.92	3027.70	520.00	482.25
12	C#.net	154	58.00	8.93	23.92	3635.39	1080.00	236.61
13	VB.net	707	28.00	19.80	55.16	8384.19	95.00	8725.46
							Average	1624.31

ISBN: 978-988-19252-6-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

V. Discussion

It may not be surprised with the disappointed results of high MRE. However, the high MRE percentages are similar to the work of Kemerer "An Empirical Validation of Software Cost Estimation Model" [18].

The high MRE percentages may be attributed to many factors, including the followings:

1. COCOMO Model and its parameters
2. Programming language factors
3. One person student project
4. Small sample size
5. others

A. COCOMO Model

It may be hypothesized that the parameters of the COCOMO Model (i.e., the values of c and k) were not fit well with the experimental environment. This is probably because the parameters of the COCOMO model were discovered by performing the regression analysis on software project data gather in the USA. This indicates the need for localization.

B. Programming language factors

The programming language factors used to converse Function points to number of Source Line of Codes is another question. The programming language conversion table by Caper Jones [16] also produced using software project data gather in the States. This is consistent with the findings of Rollo in [17].

C. One person student project

The software projects used in this research were one person graduate student projects which are different from the real world project in many aspects, especially the number of team members.

D. Small sample size

Probably with small sample size of 13 or 11 (when the outliners were removed) is the bigger problem. Small sample limits the probing into the above speculations.

E. Others

Other factor that may contribute to the high MRE is the type of software application. It may need to be adjusted with the Technical Complexity Factors (TCF) [3].

V. Conclusions

The findings of this study show high MRE percentages. However, the results are similar to the prior work, for example, the work of Kemerer [18] and Miyasaki and Mori [19]. The results reveal the potential to explore into many issues, for example, the COCOMO Model and its parameters, and the programming language factors. The implication from this research is probably that one organization should maintain and calibrate its own software project data [20, 21]. And to reduce the variation due to the COCOMO Model parameters and the programming language factors, one organization may maintain and analyze its own Function Points productivity of a specific programming language and use its own FP productivity to estimate the effort and cost need instead.

References

[1] B.W. Boehm, "Software engineering economics," *IEEE Transaction of Software Engineering*, vol.10, no.1, pp. 4-21, 1984.

[2] C. Gencel and O. Demirors, "Functional size measurement revisited," *ACM Transaction on Software Engineering and methodology*, vol.17, no. 3, pp.15.1-15.36, June 2008.

[3] R. Meli and L. Santillo, "Function point estimation methods: a comparative overview," in *FESMA '99 Conference proceedings*, Amsterdam, 4-8 October, 1999.

[4] A. J. Albrecht, "Measuring application development productivity," in *Proceeding of the IBM Applications Development Symposium*, California, October 14-17, 1979, pp. 83-92.

[5] R. Meli, "Early and extended function point: a new method for function points estimation," IFPUG Fall Conference, September, 15-19, Arizona, 1997.

[6] R. Rask, "Algorithm for counting unadjusted function points from dataflow diagram" Technical report, University of Joensuu, 1991.

[7] R. Rask, "Counting function points from SA descriptions," *The Papers of the Third Annual Oregon Workshop on Software Metrics* (Ed. W. Harrision), Oregon, March 17-19, 1991.

[8] S. J. Obrien, and D. A. Jones, "Function points in SSADM," *Software Quality Journal*, vol. 2, no. 1, pp.1-11, 1993.

[9] P. Shoval, and O. Feldman, "Combining function points estimation model with ADISSA methodology for system analysis and design," in *Proceeding of ICCSSE"96*, 1996, pp.3-8.

[10] F. Gramantieri, E. Lamma, P. Mello, and F. Riguzzi, "A system for measuring function points from specification," Technical Report, Universitra di Bologna, 1997.

[11] E. Lamma, P. Mello and F. Riguzzi, "A system for measuring function points from an ER-DFD specification," *The Computer Journal*, vol. 47, no.3, pp.358-372, 2004.

[12] F. Gramantieri, E. Lamma, P. Mello, and F. Riguzzi, "A System for Measuring Function Points from Specifications," DEIS – Universita di Bologna, Bologna. and Dipartimento di Ingegneria, Ferrara, Tech. Rep DEIS-LIA-97-006, 1997.

[13] B.W. Boehm, *Software Estimation with COCOMO II*, Upper Saddle River, NJ, Prentice Hall, 2002.

[14] J. Wu, and X. Cai, "A Software Size Measurement Model for Large-Scale Business Applications," in Proceedings of 2008 International Conference on Computer Science and Software Engineering, 2008, pp.39-42.

[15] R. Meli and L. Santillo, "Function Point Estimation Methods: a Comparative Overview," in FESMA'99 Conference Proceedings, Amsterdam, 1999.

[16] C. Jones, *Applied Software Measurement, Assuring Productivity and Quality*, 2ed, McGraw-Hill, 1997.

[17] A. L. Rollo, "Functional Size Measurement and COCOMO –A Synergistic Approach," Proceeding of Software Measurement European Forum, 2006.

[18] C.F. Kremer, "An Empirical Validation of Software Cost Estimation Models," Communication of the ACM, vol. 30, no. 3, pp.416-429, 1987.

[19] Y. Miyazaki and K. Mori, "COCOMO Evaluation and Tailoring," in Proceeding of ICSE' 85 Proceeding of the 8th International Conference of Software Engineering, 1985, pp.292-299.

[20] M. Aguiar, COCOMO II Local Calibration Using Function Points, TI Metricas, Available at: http://csse.usc.edu/events/2005/COCOMO/presentations/CIILocalCalibrationPaper.pdf

[21] B. Clark, S. Davnani-Chulani, and B. Boehm, "Calibrating the COCOMO II Post-Architecture Model," in Proceeding of ICSE' 98 Proceeding of the 20th International Conference of Software Engineering, 1998, pp.477-480.