Journal of the Nigerian Association of Mathematical Physics
Volume **22 (November, 2012), pp 281 – 284**

The Usefulness of Function Point Analysis in Estimating Projects Cost, Managing change of Scope and defining functional requirements of a Software.

[1]P. O. Odion and [2]E. A. Onibere

[1]Department of Maths and Computer Science
Nigerian Defence Academy, Kaduna. Kaduna State, Nigeria.
[2]Department of Computer Science
University of Benin, Benin-city, Edo State, Nigeria.

Abstract

This paper presents an in-depth look at the usefulness of using Function Point analysis in estimating software cost in replacement of traditional Source Lines of Code (SLOC) -based model. Besides, it shows the applicability of the Function Point in defining the functional requirements of developing Software and determining its scope. The paper also comments on the advantages and disadvantages of both Function Point and SLOC.

Keywords: Usefulness, Function Point Analysis, Source Lines of Code, Estimating Projects Cost, Managing change of Scope, functional requirements.

1.0 Introduction

Estimating software cost during and after development is a difficult task. The more complex the software project, the more cumbersome the estimation process. Before the calibration of a software cost estimation model in 1981 by Barry Boehm [1], non-algorithmic models were used for estimating software cost. Such models are: Estimation-by-analogy, Price-to-win, Top-down and Bottom-up methods, Expert judgment, etc. Today, many small businesses still use these methods, while larger businesses tend to embrace a formal model for estimating software cost.

Algorithmic cost models use a mathematical formula to predict project costs based on estimates of the project size, the number of software engineers, and other process and product factors. An algorithmic cost model can be built by analyzing the costs and attributes of completed projects and finding the closest fit formula to actual experience. Algorithmic cost models are primarily used to make estimates of software development costs, but Boehm [2] discusses a range of other uses for algorithmic cost estimates, including estimates for investors in software companies, estimates of alternative strategies to help assess risks, and estimates to inform decisions about reuse, redevelopment or outsourcing.

In its most general form, an algorithmic cost estimate for software cost can be expressed as:

$$\text{Effort} = A * \text{Size}^{B} * M \qquad (1.0)$$

'A' is a constant factor that depends on local organizational practices and the type of software that is developed. 'Size' may be either an assessment of the code size of the software or a functionality estimate expressed in function or object points. The value of exponent 'B' usually lies between 1 and 1.5. M is a multiplier made by combining process, product and development attributes, such as the dependability requirements for the software and the experience of the development team. Most algorithmic estimation models have an exponential component (B in the above equation) that is associated with the size estimate. This reflects the fact that costs do not normally increase linearly with project size. As the size of the software increases, extra costs are incurred because of the communication overhead of larger teams, more complex configuration management, more difficult system integration, and so on. Therefore, the larger the size of the software to be developed, the larger the value of the exponent. It is worthy of note that Function Point is one of the methods of Algorithmic methods used to determine the size of a software.

[1]Corresponding author: ***P. O. Odion,*** E-mail: podion2012@gmail.com, Tel. +234 8028784023

The size of the task of designing and developing a business computerized information system is determined by the product of three factors according to Symons [3] (see Table I).

Table I: The three Components of system size.

Information Processing Size	**Technical Complexity Factors**	**Environmental Factor**
i. Inputs ii. Outputs iii. Procedures iv. Files	i. Batch vs on-line ii. Performance iii. Ease of use	i. Project management/ Risk ii. People Skills iii. Methods, Tools, Languages

The *information processing size,* that is some measure of the information processed and provided by the system.
A *technical complexity factor,* which is a factor that takes into account the size of the various technical and other factors involved in developing and implementing the information processing requirements.
Environmental factors, that is the group of factors arising from the project environment (typically assessed in project risk measures), from the skills, experience and motivation of the staff involved, and from the methods, languages, and tools used by the project team.
The first two of these factors are intrinsic to the size of the system in the sense that they result directly from the requirements for the system to be delivered to the user.
The original idea of Function Point (FP) Analysis was developed by Allan Albrecht in 1979 to help measure the size of a computerized business information system, and in 1984 the first formal function point guidelines was published [4]. Such sizes are needed as a component of the measurement of productivity in system development and maintenance activities, and as a component of estimating the effort needed for such activities.
The FP metric was originally developed as an alternative to SLOC to measure
productivity in the later stages of software development. However, Albrecht argued that the FP model could also be a powerful tool to estimate software cost in the early stages of the software development lifecycle [5]. A detailed description of the software requirements is all that is needed to conduct a complete FP analysis. This enables almost any member of a software project team to conduct the FP analysis and not necessarily a team member who is familiar with the details of software development [6].
Function points represent logical size, as opposed to physical size (like SLOC or objects size). It measures software size based on the functionality requested by and provided to the end users. Recently, Function Points have gained wider acceptance
for system size assessment as a component of productivity measurement, when system development or maintenance and enhancement activities are completed. Where historic productivity data are available, the method can also be used as an
aid in estimating man-hours, from the point where a functional requirements specification is reasonably complete [7].

2.0 Calculating Function Points

Albrecht provides five categories of functions to count in software development [6].

i. External inputs
ii. External outputs
iii. External inquiries
iv. External interfaces
v. Internal logical files.

External inputs consist of all the data entering the system from external sources
and triggering the processing of data. Fields of a form are not usually counted
individually but a data entry form would be counted as one external input.
External outputs consist of all the data processed by the system and sent outside
the system. Data that is printed on a screen or sent to a printer including a report, an error message, and a data file is counted as an external output.
External inquiries are input and output requests that require an immediate
response and that do not change the internal data of the system. The process of looking up a telephone number would be counted as one external inquiry.

External interfaces consist of all the data that is shared with other software systems outside the system. Examples include shared files, shared databases, and software libraries.

Internal files include the logical data and control files internal to the system. An internal file could be a data file containing addresses. A data file containing addresses and accounting information could be counted as two internal files.

The above five categories are divided into two functions. These are: Data functions and Transactional functions. Data functions represent logical groupings of the data end users need to do their jobs.

- Internal data files maintained by the application
- External interface files referenced by the application

While the Transactional functions are the processes and actions end users utilize to manipulate and manage that data in the course of doing their jobs.

- External Inputs (add, edit, delete, etc.)
- External Outputs (reports, etc.)
- External Inquiries (search, retrieve, etc.)

When a function is identified for a given category, the function's complexity is rated as *low, average, or high* as shown in Table II.

Table II: Function Count Weighting Factors

	Complexity			
Function	**Low**	**Average**	**High**	**Total (w_{ij})**
Internal Logical Files	_x7	_x10	_x15	
External Interface Files	_x5	_x7	_x10	
External Input	_x3	_x4	_x6	
External Output	_x4	_x5	_x7	
External Inquiry	_x3	_x4	_x6	
Total Unadjusted Function Points (UFP) (x_{ij})				$\sum w_{ij} x_{ij}$

Each function count is multiplied by the weight associated with its complexity and all of the function counts are summed to obtain the count for the entire system, known as the unadjusted function points (UFP). This calculation is summarized by the following equation:

$$UFP = \sum_{i=1}^{3} \sum_{j=1}^{5} w_{ij} x_{ij} \qquad (1.1)$$

where w_{ij} is the weight for row *i*, column *j*, and x_{ij} is the function count in cell *i, j* [6].

3.0 Function Points Methodology

Albrecht emphasized that UFP can give us a good idea of the number functions in a system; it doesn't take into account the environment variables for determining effort required to develop a software. He recognized this when developing the FP model and created a list of fourteen general system characteristics that are rated on a scale from 0 to 5 in terms of their likely effect for the system being counted.

These characteristics are as follows:

i. Data communications, ii. Distributed functions, iii. Performance, iv. Heavily used configuration, v. Transaction rate, vi. Online data entry, vii. End user efficiency, viii. Online update, ix. Complex processing, x. Reusability, xi. Installation ease xii. Operational ease, xiii. Multiple sites, and xiv. Facilitates change

The ratings given to each of the characteristics above c_i are then entered into the following formula to get the Value Adjustment Factor (VAF):

$$VAF = 0.65 + 0.01 * \sum_{C_i=1}^{14} (c_i) \qquad (1.2)$$

where c_i is the value of general system characteristic *i*, for $0 <= c_i <= 5$ [8]. Finally, the UFP and VAF values are multiplied to produce the Adjusted Function Point (AFP) count:

$$AFP = UFP * VAF \quad (1.3)$$

To have accurate Function Points count, the following steps have to be followed:

i. Determine type of function point count.
ii. Identify application boundary.
iii. Identify data functions and their complexity.
iv. Calculate Unadjusted Function Point count.
v. Identify transactional function and their complexity.
vi. Determine Value Adjustment Factor (VAF).
vii. Calculate final adjusted function point count.

4. THE USEFULNESS OF FUNCTION POINTS ANALYSIS

The uses of Function Points Analysis to estimate the size of software to be developed make it independent of the language and other implementation variables that are often difficult to take into consideration. The values of the counting method are technology dependent. This is implicit in the weights used for the UFP components.

FP analysis works for installed applications, not for tools, or languages, such as a general purpose retrieval language. However, the distinction between these two classes of systems is not always absolutely clear. Applications provide preprogrammed functions where the user is invited to enter data and receives output data. Tools provide commands with which the user can create his or her own functions. Business information systems are usually applications, but may sometimes incorporate tools, or features with tool-like characteristics as well [9].

Kemerer [8] believes that the FP model for software effort and cost estimates satisfies the need for a robust measurement metrics for software development.

The FP approach seems to present significant advantages over the traditional
SLOC approach for estimating software cost.

Touesnard [6] recommended that any organization that is beginning to adopt a formal cost estimation model should first take the time to carefully consider the FP model, before regressing to an older SLOC-based model. Simply choosing a SLOC-based model, because it is a familiar metric or it takes a little less effort to collect data is probably not good reasoning.

5. CONCLUSION

This work focuses on the analysis and usefulness of Function Points Estimating processes. For further studies, the application of FPA will be used as a substitute for SLOC using our local environment to estimate the cost, effort and duration of software development. This is where practical results will be showed and its discussions.

REFERENCES

[1] Boehm, B.W. (1981), *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, USA. pp 767-776.
[2] Boehm, B.W. (2000), Software Cost Estimate with COCOMO II, Prentice-Hall, Englewood Cliffs, USA.
[3] Symons C.R. (1988), *Function Point Analysis: Difficulties and Improvements*, IEEE Transactions on Software Engineering, Vol. 14, No. 1, pp 2-11
[4] Brown I (2005), *Certified Function Point Analysis Specialist*, Senior Associate, McLean, VA, (703) 902-4971, brown_ian@bah.com
[5] Albrecht A.J. (1979), *Measuring Application Development Productivity*, IBM Application Development Symposium, GUIDE Int and SHARE Inc, IBM Corp, Monterey, CA. pp 83-92.
[6] Touesnard B (2004), *Software Cost Estimation: SLOC-based Models and the Function Points Model* Version 1.1, http://www.ifpug.org
[7] Silvia A. and Oscar P. (2003), *Measuring the functional size of Web application*, International Journal of Web Engineering and Technology, Vol. 1, No. 1, pp 5-16.
[8] Kemerer C.F. (1993), *Reliability of Function Points Measurement. A Field Experiment,* Communications of the ACM, Vol.36, No.2, pp. 85-97.
[9] Hamilton B.A. (2005), *Software Cost Estimation Using Function Point Analysis*, http://www.ifpug.org/certification/CFPSbrochure.pdf