International Journal of Computing and Business Research (IJCBR)
ISSN (Online) : 2229-6166
Volume 6 Issue 1 January 2015

SELECTION OF DIRECT AND DERIVED FUNCTION POINT ESTIMATION METHODS

Edna Tarverdian, Michael Scott Brown, Michael Pelosi

University of Maryland University College
etarverdian@student.umuc.edu
Michael.brown@umuc.edu
Michael.pelosi@faculty.umuc.edu

Abstract: In this paper we discuss software maintenance tools. Maintenance tools may include everything from large-scale integrated CASE tools to simple one-function commands. The objective of this paper is to learn more about software maintainers' needs for software tools, and in particular methods for estimating software size, so the maintainers may become more productive. This paper discusses that the standard Function Point technique has allowed maintainers to increase significantly in software maintenance practices. However, there are situations where estimation methods may be more compatible to the standard rules of the Function Point (FP). First situation may be when enhancement or development project is in early stages, which is not possible to perform FP count. Another situation may be when the necessary documentation or the required time and resources are not available to perform a standard FP count. Thus, the FP estimation methods may be very decisive for these situations. In this paper, we present a review of several estimation methods with their characteristics.

Keywords: Function Points, FP Estimation Methods, Software Size Estimation, Software Measurement**.**

1. Introduction

Software maintenance tools are a significant help for software engineers and maintainers for performing maintenance. Software maintenance tools may include anything functional that help software maintainers solve maintenance problems. Generally, these tools are software programs or parts of computer programs (Lethbridge, 1996). Lower-level CASE tools such as Command Interpreter, Syntax Checker e.g. W3C CSS Validator, W3C HTML Validator are examples of these tools.

Software size measures are one of the most significant measures among software maintenance tools. These measures have direct relevance with maintenance planning, tracking and estimating software projects. In addition, they are used to compute productivities, to normalize quality indicators, and to derive measures for memory utilization and test coverage (Park, 1992).

In the effort to discuss the subject maintenance tools, the opportunity is presented in order to distinguish effects of these tools on maintenance activities. Since software maintenance activities take longer to perform than the development activities, measuring the size of the software to be maintained has a significant effect on the overall maintenance process.

In this effort, a review of two widely used measures for software size will be presented along with analysis on their strengths and weaknesses. Based on each measure's advantages and disadvantages, Function Points (FP) measurement will be explained as a more reliable measurement. A description of different methods for estimating FPs will be discussed before finally discussing the implications of the best estimation method and conclusion.

2. Software Size Measures

Essentially, there are two software size measures that are currently used widely in software maintenance. First is the number of Source Line of Code (SLOC) and second is the number of Function Points (FP). Research shows that in practice, it is difficult to estimate the number of line of code accurately early in a project (Low, 1990). Particularly, when comparing systems that are written in different languages, it has been proven that SLOC measures have weaknesses to estimate productivity (Jones, 1986). Moreover, modern development techniques, such as object-oriented programming, re-use of library components, and use of open source components make the relationship of the SLOC and software attributes less accurate (Fairley, 2009).

The other software size measure, FP, was developed by Allan Albrecht of IBM to measure the external size of data processing applications, and is the most broadly popular functional type metrics. It is suitable for evaluating a software application (Meli, 1998). Research has proven that FPs were found to be the best productivity measure (Perry, 1986), and have been widely used in cost estimations (Kemerer, 1987), software development productivity evaluation (Behrens, 1983), software maintenance productivity evaluation (Banker, 1991), software quality evaluation (Cooprider, 1989), and software project sizing (Banker, 1989).

3. Function Points Measurement

Function points (FPs) are calculated by counting the number of different kinds of inputs, outputs, internal files, queries, and interfaces in software to be estimated (Fairley, 2009).

Each of the function point factors is then weighted as low, average or high. The weighted values are summed in order to provide a total number of Unadjusted Function Points (UFPs). The UPF is then combined with the Value Adjustment Factor (VAF) to attain the final number of FP. The Value Adjustment Factor (VAF) may be computed by (Kemerer, 1993):

$$VAF = 0.65 + 0.01 \sum_{i=1}^{14} C_i \quad (1)$$

Where C_i is the value for general system characteristic i, for $0 <= C_i <= 5$. These characteristics are 1) data communications, 2) distributed functions, 3) performance, 4) heavily used configuration, 5) transaction rate, 6) on-line data entry, 7) end user efficiency, 8) on-line update, 9) complex processing, 10) reusability, 11) installation ease, 12) operational ease, 13) multiple sites, and 14) facilitates change. Table 1 illustrates an example of function point approach for measuring functional size of software (Fairley, 2009).

Table 1 A function point example

Complexity:	Simple	Average	Complex	Total
Inputs	3×3	2×4	0×6	17
Outputs	4×4	6×5	3×7	67
Files	5×7	2×10	0×15	55
Queries	0×3	9×5	4×7	73
Interfaces	0×5	0×7	3×10	30
			Total	242

As it is shown in Table 1, the number of Unadjusted Function Points (UFP) in the above example is *242*. Assuming the composite adjustment factor (VAF) as 1.13, the number of adjusted function points will be approximately 1.13 x 242 = 276. The adjusted function points can be used to compare against other projects. This can be used to determine the duration of the work.

4. Estimating Function Points

FP is considered the most efficient technique for estimating software size. However, in order to follow the standard IFPUG Counting Practices, the complete and detailed set of user functional requirements of the estimated software should be available (IFPUG, 1994). Thus, in some situations, estimation methods shall be an alternative to the standards rules of FP. In cases when the project is in early phases of software development or enhancement projects, it is not possible to identify the elements of FP in order to perform a standard FP count. In addition, there may be cases where the necessary documentation or the required time and resources to perform FP count are not available.

In order to evaluate the software size soon and/or with the smallest need for resources, several estimation methods have been proposed. Meli and Santillo (Meli, 1998) claim that "Estimating means using less time and effort in obtaining an appropriate value of FP". It should be considered that the accuracy of estimation methods might be less than the standard FP calculations.

Estimation models may be characterized as input-processing-output system. The input variables are the information on the software that should be sized, and the output is the functional size, which is the FP. The estimation methods are categorized as: 1) Direct Estimation, and 2) Derived Estimation (Meli, 1998).

Direct Estimation methods are often involved with consultation of one or more experts. These methods are also called Expert Opinion methods, in which the consulting experts would guess the FP estimation based on their past experience. Direct estimations may improve by use of Analogy or Delphi methods, which will be described in a future section of this paper.

Derived Estimation methods are often associated with decomposition of an application. By decomposing an application to its major functions, estimation can be performed in a stepwise fashion (Meli, 1998).

The main difference of direct estimation and derived estimation is that in direct estimation the estimations are made directly on FP values, whereas in derived estimation they are made on different software attributes that are associated with the FP values. Such attributes could be adaptability, robustness, invariance and compatibility.

5. FP Estimation Methods

The VAF may be determined with few available details for a standard count. Therefore, below we review most common methods for estimating the value of UFP.

5.1. Direct Estimation Methods

Direct Estimation methods are entirely influenced and dependent on expert(s)'s opinion(s). In some cases where more than one expert is involved, then the estimates may be influenced by personal relationships as well. Below are the most common direct FP estimation methods.

Delphi Techniques

In these techniques, each individual's prediction is collected anonymously and is combined together. The iteration cycle continues until the estimates meet an acceptable range. Generally, the group estimate is a better estimate than an individual estimate (Meli, 1998).

Simple Analogy Method

In order to estimate software size, this method looks into the historical database for systems that are similar to the estimated application. Using this data enables estimator to provide a quick estimate of the product size.

Structured Analogy Method

In this method the estimator compares the estimated application with one or more existing applications. The estimator first identifies the type of the application, makes an initial estimation, and then improves the initial estimation within the original range.

This method is basically the more formal approach of the Simple Analogy approach.

5.2. Derived Estimation Methods

Derived Estimation methods are mainly algorithmic models for estimating FP size. Below are the most widely used algorithmic models.

Extrapolative Counts

The models for Extrapolative Counts assume that the estimator can only count one FP component, which is usually the number of Internal Logical Files (ILF), and obtain the rest of the counts on a statistical or theoretical basis. Following are some of these models.

Tichenor ILF Model is one model that uses Extrapolative counts technique, which demonstrates a strong relation between the number of ILFs and UFP count as below (Meli, 1998):

$$Est.UFP = \#ILF \times 11.01$$

(If Batch System)

$$Est.UFP = \#ILF \times 14.93 \quad (2)$$

(If Transactional System, Create/Read/Update/Delete)

This model also provides the estimated Adjusted FP as follows (Meli, 1998):

$$Estimated\ FP = 1.0163 \times (UFP \times VAF)\ \hat{}\ 1.0024 \quad (3)$$

Tichenor IL model provides FP factors average ratios to ILF as below in Table 2 (Meli, 1998).

Table 2 Tichenor ILF Ratios

Application Characteristic	ILF Ratio
External Inputs	0.33
External Outputs	0.39
External Inquiry	0.01
External Interface Files	0.097

FP Prognosis by CNV AG is another approach that provides ratios for systems that are outlined in Table 3 (Bundschuh, 1998).

Table 3 Tichenor ILF Ratios

Application Characteristic	ILF Ratio
External Inputs	2.66
External Outputs	3.14
External Inquiry	1.20
External Interface Files	0.40

This model provides estimation for FP by summing the quantities of EI and EO (IO) (Bundschuh, 1998):

$$FP = 7.3 \times \#IO + 56 \tag{4}$$

One advantage of FP Prognosis model is that it can be used in early stages without having to investigate ILF and EIF. Meli and Santillo (Meli, 1998) claim that the error range for using this model is 20%.

Indicative FP, also known as "the Dutch method", was proposed by NESMA (Netherlands Software Metrics Association), which calculates the UFP from the number of data functions as follows (Tichenor, 1997):

$$Indicative\ Size\ (UFP) = 35 \times \#ILF + 15 \times \#EIF \tag{5}$$

The *Indicative FP* in Equation 6 is based on the assumption that "there will be about three EI (to add, change, and delete information in the ILF), two EO, and one EQ on average for every ILF, and about one EO and one EQ for every EIF" (Milne, 1998).

The release 5 of ISBSG has provided more generic set of ratios on average over 400 cases as follows (Hill, 1999):

$$ILF\ 22.3\%\quad EIF\ 3.8\%\quad EI\ 37.2\%\quad EO\ 23.5\%\quad EQ\ 13. \tag{6}$$

The *ISBSG Benchmark* also states that most ILF are rated with low complexity (Hill, 1999); therefore Meli and Santillo (Meli, 1998) stated that:

$$UFP\ (Only\ ILF) = 7.4 \times \#ILF$$

$$UFP\ (total) = \frac{UFP\ (Only\ ILF)}{22 \times 100} \tag{7}$$

ISBSG states that because implicit functionalities aren't visible in early stages but they are included in standard FP counts, then a 20-30% possibility should be added to the above estimates (Hill, 1999).

Sampled Counts

Using this method estimator may estimate the size of the product by counting a portion of the system with respect to some FP components (EI, EO, EQ, ILF or EIF), while the IFPUG count investigates the whole system (Meli, 1998).

Average Complexity Estimation

Table 4 illustrates the average function complexity for Development Projects by Release 5 of the ISBSG Benchmark (Hill, 1999):

Table 4 Average Function Complexity

	Average UFP	IFPUG
ILF	7.4	10
EIF	5.5	7
EI	4.3	4
EO	5.4	5
EQ	3.8	4

Average complexity method estimates the product size by identifying all the components of the IFPUG count (EI, EO, EQ, ILF, and EIF), and then assign a weighted average complexity rating to them by using the following equation (Meli, 1998):

$$Est.UFP = \#EI \times 4.3 + \#EO \times 5.4 + \#EQ \times 3.8 - \#ILF \times 7.4 - \#EIF \times 5.5 \quad (8)$$

Early Function Points

Early function points technique provides a better estimate for software size by using both analogical and analytical classification of functionalities. In addition, this technique lets the estimator use multilevel approach, which is to use different levels of details for different branches of the system (Meli, 1997). One benefit of using multilevel approach is that it lets the estimator to utilize his knowledge on one particular branch of the estimated system without having to ask questions that are difficult to answer in early stages.

Early FP estimation key factors are Macrofunctions, Functions, Microfunctions, Functional Primitives and Logical Data Groups (Meli, 1997). Functional Primitives are the standard FP estimation factors, such as, EI, EO, and EQ. Macrofunctions, Functions, and Microfunctions are different combination of more than one Functional Primitive at different detail level. Logical Data Groups are the standard Logical Files without the differentiation of external and internal.

Early FP estimation assigns each object a set of minimum, average, and maximum FP values based on analytical tables. Then the values are summed up, which provides the UFP (unadjusted FP) estimate (Milne, 1998). Based on the chosen detail level, the estimates that are provided by this estimation technique may be indicated as detailed, intermediate or summary (Meli, 1997).

The reliability of EFP depends on the estimator's ability to identify the system's components as part of one of the proposed classes. This ability may be improved through practice. Besides the estimator's ability, research has proven that EFP technique is quite effective and it provides a response within $\pm$ 10% of the real FP value in most cases. Moreover, it provides a time and cost savings between 50% and 90% with respect to equivalent standard counts (Meli, 1997).

6. Conclusion

In this work, several software size estimation methods were discussed, and functional software metrics Function Point IFPUG 4.0 was assumed as a standard. Table 5 illustrates a summary of the presented estimation methods.

Table 5 Software Size Estimation Methods

	Advantages	**Disadvantages**
Direct Estimation Methods		
Delphi Techniques	May result in accurate estimates	Difficult to justify the results
Simple Analogy Method	Based on estimator's experience	Highly depends on the availability of the historical data and expertise of the estimator
Structured Analogy Method	Based on estimators' experience	Highly depends on the availability of the historical data and expertise of the estimator
Derived Estimation Methods		
Extrapolative Counts	Counts only one FP component, which is mostly Internal Logical Files (ILF)	Derives the rest of the count on a theoretical basis
Sampled Counts	Counts a portion of the system with respect to some FP components (EI, EO, EQ, ILF or EIF)	Estimates the count of the rest of the system
Average Complexity Estimation	Generally results in more accurate estimates	Identifies all the components of the IFPUG count
Early Function Points	Provides a better estimate by using both analogical and analytical functionalities	Highly depends on the ability of the estimator to recognize the components of the system

None of the presented alternatives are better than the other as each method has their own strengths and weaknesses. We must note that the level of detail of information that is needed to estimate Function Points by most of the Derived Estimation Methods are very similar to the ones that is needed by a standard count, and therefore these estimation methods are not mostly dominant. However, the Early Function Point method is an exclusion from this.

As a conclusion, based on the strengths and weaknesses of different presented methods, the best estimation should be the one that uses a combination of techniques, and the comparison and iteration of the estimates that is obtained from each one.

References

Lethbridge, T. C., & Singer, J. (1996) "Understanding Software Maintenance Tools: Some Empirical Research", *1st Workshop on Empirical Studies of Software Maintenance.*

Park, E. R. (1992) "Software Size Measurement: A Framework for Counting Source Statements", Retrieved from http://www.sei.cmu.edu/reports/92tr020.pdf

Low, C. G., & Jeffery, D. R. (1990) "Function Points in the estimation and evaluation of the software process", *IEEE Trans. Softw. Eng., 16*(1), 64-71.

Jones, C. (1986) *Programming Productivity*. New York: McGraw-Hill.

Fairley, R. E. (2009) *Managing AND Leading Software Projects.*Hoboken, NJ: A John Wiley & Sons, Inc., Publication.

Meli, R., & Santillo, L. (1998) "Function Point Estimation Methods: A Comparative Overview".

Perry, W. E. (1986) "The best measures for measuring data processing quality and productivity", Tech. Rep. Quality Assurance Institute.

Kemerer, C. F. (1987) "An empirical validation of software cost estimation models", *Commun.ACM, 30*(5).

Behrens, C. A. (1983) "Measuring the productivity of computer systems development activities with Function Points", *IEEE Trans. Softw. Eng. SE-9*(6), 648-652.

Banker, R. D., Datar, S. M. & Kemerer, C. F. (1991) "A model to evaluate variables impacting productivity on software maintenance projects", *Manage. Sci, 37*(1), 1-18.

Cooprider, J., & Henderson, J. (1989) "A multi-dimensional approach to performance evaluation for I/S development", Working Paper 197, MIT Center for Information Systems Research. Cambridge, Mass.

Banker, R. D., & Kemerer, C. F. (1989) "Scale economies in new software development", *IEEE Trans. Softw. Eng., SE-15*(10), 416-429.

Kemerer, C. F. (1993) "Reliability of Function Points Measurement", *Communication Of The ACM, 36*(2).

IFPUG – Function Point Counting Practices Manual, Rel. 4.0. (1994)

Bundschuh, M. (1998). "Function Point Prognosis", FESMA 98 Procs.

Tichenor, C. (1997). "The IRS Development and Application of the Internal Logical File Model to Estimate Function Point Counts", *IFPUG Fall Conference.*

Milne, B. J., & Luxford, K. B. G. (1998). "ISBSG – Worldwide Software Development", *The Benchmack, Release 5*. 23-36.

Hill, P. R. (1999). ISBSG – Software Project Estimation, A Workbook for Macro-Estimation of Software Development Effort and Duration.

Meli, R. (1997). "Early and Extended FP: a New Estimation Method for Software Projects", *IFPUG Fall Conference.*