Volume 5, Issue 1, January 2015 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Metrics for Software Project Size Estimation

Harish Chandra Maurya
Assistant Professor (C.S.E)
Bhagwant University,
Ajmer, Rajasthan, India

Aasiya Khatoon
M.Tech Scholar (C.S.E)
Bhagwant University,
Ajmer, Rajasthan, India

Nalin Chaudhary
M.Tech Scholar (C.S.E)
Bhagwant University,
Ajmer, Rajasthan, India

Abstract- Two metrics are popularly being used widely to estimate size: lines of code (LOC) and function point (FP). The usage of each of these metrics in project size estimation has its own advantages and disadvantages. The focus of this paper is how to make software projects more successful by properly estimating the size. When there are a large number of undefined areas related to the project in the initial stages of project development then most of the estimates are already prepared. This paper proposes some size estimation processes that begin by addressing the need for project metrics and the fundamental software estimation concepts.

Keywords: Introduction, Lines of code (LOC), Function Point

I. INTRODUCTION

Software sizing is an activity in software engineering that is used to estimate the size of a software application or component in order to be able to implement other software project management activities (such as estimating or tracking). Size is an inherent characteristic of a piece of software just like weight is an inherent characteristic of a tangible material. Software sizing is different from software effort estimation. Sizing estimates the probable size of a piece of software while effort estimation predicts the effort needed to build it. The relationship between the size of software and the effort required to produce it is productivity. Size of software system is considered as the basic metrics in software metrics model. Size can be estimated in Lines of code or function point. Lines of code cannot be estimated correctly before software completion because it varies due to language complexities of different language. While Function points are technologically independent, consistent, repeatable, help normalize data, enable comparisons and set project scope and client expectation.

Software size contains important information for project planning. Costs and schedule estimates depend on its existence and its accuracy; indirectly the project's success also depends on it. A software project is successful if the requirements are fulfilled and no budget or deadline overflows occur [1]. With systematic size estimation, the risk of overflows is lower. Systematic software size estimation requires a method that defines a procedure for measurement, involving units and accuracy. In general, Functional Size Measurement (FSM) methods, as defined in ISO/IEC TR 14143 [2, 3, 4, 5] can be categorized into two groups. In the first group, there are technology-independent methods; an example would be the Function Point Analysis (FPA) method [6]. In the second group, there are technology-dependent methods; for example Lines of Code (LOC) or number of classes. The methods from the first group have obvious advantages over the methods from the second group. The ultimate goal is to use only technology-independent methods. The FPA method dates back to 1979 [7] and has been updated several times. However, the core concepts and the counting procedure remains the same. Every information system processes some data that can be stored in the application database or is taken from external applications. Four operations are performed on data records: create, read, update and delete. Besides that, information systems use several query functions for data retrieval and report construction. Each record consists of several fields of basic data types or another record that can be further deconstructed. The FPA method quantifies: the number of fields in each record, the distinct operations performed on these records, and the number of these operations that are necessary to perform a business function. The sum over all business functions, multiplied with some empirically determined weights, represents the unadjusted function point's value. The final calculation is made using a Value Adjustment Factor (VAF) that measures system complexity. Based on the FPA method, several methods like Feature points, Full Function Points, Function Weight, Function Bang, Mk II Function Points Analysis, COSMIC-FFP and NESMA evolved.

Two main ways you can estimate product size are:

1) Having done a similar kind of project in the past and knowing its size, one needs to estimate each major module of the new project as a percentage of the size of a similar module of the previous project. Estimate the total size of the new project by adding up the estimated sizes of each of the pieces. An experienced estimator can produce reasonably good size estimates by analogy if accurate size values are available for the previous project and if the new project is sufficiently similar to the previous one.

2) Counting product features and using an algorithmic approach such as Function Points in order to convert the count into an estimate of size. Macro-level "product features" may include the number of sub systems, classes/modules, methods/functions.

© 2015, IJARCSSE All Rights Reserved

II. LINES OF CODE (LOC)

LOC is the simplest among all metrics available to estimate project size. This metric is very popular because it is the simplest to use. Using this metric, the project size is estimated by counting the number of source instructions in the developed program. Obviously, while counting the number of source instructions, lines used for commenting the code and the header lines should be ignored.

Determining the LOC count at the end of a project is a very simple job. However, accurate estimation of the LOC count at the beginning of a project is very difficult. In order to estimate the LOC count at the beginning of a project, project managers usually divide the problem into modules and each module into sub modules and so on, until the sizes of the different leaf-level modules can be approximately predicted. To be able to do this, past experience in developing similar products is helpful. By using the estimation of the lowest level modules, project managers arrive at the total size estimation.

"*A line of code is any line of program text that is not a comment or blank line, regardless of the number of statements or fragments of statements on the line. This specifically includes all lines containing program header, declarations, and executable and non-executable statements.*"

For Example: Fig. 1 shows the C Program of calculating the Average of three numbers. This C program is written in 9 lines including comments that is begins with "/*****". Remember that C comments are not executed.

Line No.	
1	**Main()**
2	**{**
3	**int a, b, c, avg;**
4	**/*enter value of a,b,c*/**
5	**scanf("%d %d %d", &a, &b, &c**
6	**/*calculate average*/**
7	**avg = (a+b+c)/3;**
8	**printf("avg = %d", avg);**
9	**}**

Fig. 1: According to the definition of the LOC, above Program (shown in Fig. 1) has a 7 LOC.

Advantage of LOC:

- Counting LOC of any Program text is a very simplest method.

Disadvantage of LOC:

- LOC is language dependent. For example a line of assembler is not the same as a line of COBOL.
- They also reflect what the system is rather than what it does.
- Counting LOC is similar to counting bricks in a building. Measuring systems by the LOC is rather like measuring a building by the number of bricks which is used to create the building while Building are normally described in terms of number and size of the rooms, their total areas in square feet.
- Difficult to estimate LOC from problem description and therefore it is not very useful for project planning.

III. FUNCTION POINT

Function point metric was proposed by Albrecht [1983]. This metric overcomes many of the shortcomings of the LOC metric. Since its inception in late 1970s, function point metric has been slowly gaining popularity. One of the important advantages of using the function point metric is that it can be used to easily estimate the size of a software product directly from the problem specification. This is in contrast to the LOC metric, where the size can be accurately determined only after the product has fully been developed.

The conceptual idea behind the function point metric is that the size of a software product is directly dependent on the number of different functions or features it supports. A software product supporting many features would certainly be of larger size than a product with less number of features. Each function when invoked reads some input data and transforms it to the corresponding output data.

Function point measures functionality from the user's point of view, that is, on the basis of what the user requests and receives in return from the system. The principle of Albrecht's function point analysis (FPA) is that a system is decomposed into functional units**.**

Input: information entering the system.
Output: information leaving the system.
Enquiries: requests for instant access to information.
Internal Logic Files: information held within the file system.
External Interface Files: information held by other systems that is used by the system being analyzed.

© 2015, IJARCSSE All Rights Reserved

COUNTING FUNCTION POINTS:

The five functional units are raked according to their complexity i.e., Low, Average, or high using a set of prescriptive standards. Organizations that use FP methods develop criteria for determining whether a particular entry is Low, Average, or high.

After classifying each of the five function types, the Unadjusted Function Points (UFP) are calculated using predefined weights for each function type given in table.

Functional Units	**Low**	**Average**	**High**
External Input	**3**	**4**	**6**
External Output	**4**	**5**	**7**
External Inquires	**3**	**4**	**6**
Internal Logic files	**7**	**10**	**15**
External Interface files	**5**	**7**	**10**

The procedure for calculating Unadjusted Function Point in Mathematical form is:

$$UFP= \sum^{5}_{i=1} \sum^{3}_{j=1} (Z_{ij} W_{ij})$$

Where i and j are the row and column of the table.

W_{ij} : it is the entry of the i and j column

Z_{ij}: It has a count of the number of functional units of type i that have been classified as having a complexity corresponding to column j.

Organizations that use function point methods develop a criterion for determining whether a particular entry is Low, Average or High. The final number of function points arrives at by multiplying the UFP by an adjustment factor that is determined by considering 14 aspects of processing complexity which are given in table below. The adjustment factor allows the UFP count to be modified by at most (+-) 35%.

The final adjusted FP count is obtained by using the following relationship

FP=UFP*CAF

Where CAF is complexity adjustment factor and is equal to [0.65+0.01*∑fi]. The fi (i=1 to 14) are the degrees of influence and are based on responses to question noted below:

1. Does the system require reliable backup and recovery?
2. Is data communication required?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing heavily utilized operational environment?
6. Does the system require online data entry?
7. Does the online data require the input transition to be built over multiple screens or operations?
8. Are the master files updated online?
9. Is the input, outputs, files, or inquires complex?
10. Is the internal processing complex?
11. Is the code designed to be reusable?
12. Are conversion and installation included in the design?
13. Is the system designed for multiple installations in the different organizations?
14. Is the application designed to facilitate change and ease of use by the user?

The suggested score for degree of influence is as below in table 1.

Table 1. Degree of Influence

Score as	**Descriptions to determine Degree of influence**
0	None of the above
1	Incidental
2	Moderate
3	Average
4	Significant
5	Essential

© 2015, IJARCSSE All Rights Reserved

Example 1:
Consider a project with the following functional units:
Number of user inputs = 50
Number of user outputs = 40
Number of user enquiries = 35
Number of user files = 06
Number of external interfaces = 04
Assume all complexity adjustment factors and weighting factors are average. Compute the function points for the project.
Solution:
We know **UFP=** $\sum^{5}_{i=1} \sum^{3}_{j=1}$ **(Zij Wij)**
UFP= 50*4+40*5+35*4+6*10+4*7 = 628
CAF= (0.65+0.01(∑ fi))
= (0.65+0.01(14*3)) = 1.07
FP= UFP*CAF point
=628*1.07=672 **Ans.**

Advantages:
- Function Point helps in software development cost estimation.
- It's also helps in improving product quality.
- Function point work well with Use Cases.

Drawback:
Function point suffers from a major drawback that the size of a function is considered to be independent of its complexity.

IV. CONCLUSION

Today various estimation techniques are applicable in different types of projects. These techniques do not give hundred percent accuracy. Some organizations should use mix of methodologies and more than one methodology for the same estimation. Now days various types of tools are available in the market to automate size estimation.

REFRENCES

[1] Paulson, L.D. Adapting Methodologies for Doing Software Right. IT Pro, 7-8 (2001), pp. 13-15
[2] ISO/IEC TR 14143-1. Information technology - Software measurement - Functional size measurement, Part 1: Definition of concepts, First edition, ISO/IEC, 1998
[3] ISO/IEC TR 14143-2. Information technology - Software measurement - Functional size measurement, Part 2: Conformity evaluation of software size measurement methods to ISO/IEC 14143-1:1998, First edition, ISO/IEC, 2002
[4] ISO/IEC TR 14143-3. Information technology - Software measurement - Functional size measurement, Part 3: Verification of functional size measurement methods. First edition, ISO/IEC, 2003
[5] ISO/IEC TR 14143-4. Information technology - Software measurement - Functional size measurement, Part 4: Reference model, First edition, ISO/IEC, 2002
[6] IFPUG, Function Point Counting Practices Manual. International Function Point Users Group, Westerville, Ohio, 1999.
[7] Albrecht, A., Measuring Application Development Productivity, IBM Applications Development Symposium, 1979, pp. 83-92.
[8] De Marco, Tom, Controlling Software Projects, Prentice-Hall, 1982
[9] Humphrey, Watts, A Discipline for Software Engineering, Addison-Wesley, 1995

© 2015, IJARCSSE All Rights Reserved